Banc GSM émulé — dossier de run

/opt/GSM/c54x_exe

Banc GSM émulé — banc-max

2026-10-25

Table des matières

1 Synthèse **2**

2 Resultats des tests **3**

2.1 /opt/GSM/c54x_exe/LAUNCH.md . . . 3
2.1.1 Lancer le côté mobile, processus par processus . . . 3
2.2 /opt/GSM/c54x_exe/MAILBOX.md . . . 6
2.2.1 Boîte aux lettres ARM <-> DSP (Calypso API RAM) . . . 6
2.3 /opt/GSM/c54x_exe/README.md . . . 52
2.3.1 c54x_exe — the Calypso DSP, run as a native process . . . 52
2.4 /opt/GSM/c54x_exe/balayage_full.tsv . . . 55
2.5 /opt/GSM/c54x_exe/balayage_v2.tsv . . . 58
2.6 /opt/GSM/c54x_exe/proof_toa_23.md . . . 69
2.6.1 OsmocomBB's "magic 23", proven from the ROM, the layer1 code and hardware logs . . . 69
2.7 /opt/GSM/c54x_exe/docs/CLAIMS-2026-10-04.md . . . 82
2.7.1 Emulating the baseband — claims and evidence (2026-10-04) . . . 82
2.8 /opt/GSM/c54x_exe/docs/README.fr.md . . . 86
2.8.1 c54x_exe — le DSP Calypso sans QEMU . . . 86
2.9 /opt/GSM/c54x_exe/docs/carte-memoire.md . . . 92
2.9.1 Carte mémoire du DSP Calypso — API RAM, NDB, PARAM, mémoire interne, ports . . . 92
2.10 /opt/GSM/c54x_exe/docs/emulating-the-baseband.md . . . 102
2.10.1 Emulating the baseband . . . 102
2.11 /opt/GSM/c54x_exe/tools/isa_tests.txt . . . 104
2.12 /opt/GSM/c54x_exe/tools/cch_ref/cch_deinterleave_inverse_ref.csv . . . 137
2.13 /opt/GSM/c54x_exe/tools/cch_ref/cch_interleave_ref.csv . . . 139
2.14 /opt/GSM/c54x_exe/tools/cch_ref/cch_testvectors.txt . . . 141

3 Fichiers **141**

3.1 /opt/GSM/c54x_exe/.gitignore . . . 141
3.2 /opt/GSM/c54x_exe/Makefile . . . 142
3.3 /opt/GSM/c54x_exe/balayage.sh . . . 144
3.4 /opt/GSM/c54x_exe/harvard.sh . . . 145
3.5 /opt/GSM/c54x_exe/install.sh . . . 160
3.6 /opt/GSM/c54x_exe/mobile_pont.cfg . . . 168
3.7 /opt/GSM/c54x_exe/repro_toa.sh . . . 170
3.8 /opt/GSM/c54x_exe/run.sh . . . 172
3.9 /opt/GSM/c54x_exe/run_real.sh . . . 188
3.10 /opt/GSM/c54x_exe/run_si.sh . . . 189
3.11 /opt/GSM/c54x_exe/sign.sh . . . 190
3.12 /opt/GSM/c54x_exe/tout-en-un.py . . . 191
3.13 /opt/GSM/c54x_exe/src/carte_tx.h . . . 197
3.14 /opt/GSM/c54x_exe/src/cellule.c . . . 198
3.15 /opt/GSM/c54x_exe/src/cellule.h . . . 209
3.16 /opt/GSM/c54x_exe/src/main.c . . . 209
3.17 /opt/GSM/c54x_exe/src/montant.c . . . 214

3.18 /opt/GSM/c54x_exe/src/montant.h . . . 239
3.19 /opt/GSM/c54x_exe/src/pcb-minimal.c . . . 240
3.20 /opt/GSM/c54x_exe/src/pont.c . . . 240
3.21 /opt/GSM/c54x_exe/src/pont.h . . . 287
3.22 /opt/GSM/c54x_exe/src/rejouer.c . . . 288
3.23 /opt/GSM/c54x_exe/src/rejouer.h . . . 328
3.24 /opt/GSM/c54x_exe/src/tsp_tx.c . . . 328
3.25 /opt/GSM/c54x_exe/src/tsp_tx.h . . . 329
3.26 /opt/GSM/c54x_exe/src/verbosite.c . . . 330
3.27 /opt/GSM/c54x_exe/src/verbosite.h . . . 332
3.28 /opt/GSM/c54x_exe/tools/analyse_sb.py . . . 332
3.29 /opt/GSM/c54x_exe/tools/cmp_codeur.py . . . 333
3.30 /opt/GSM/c54x_exe/tools/comparer_parole.py . . . 334
3.31 /opt/GSM/c54x_exe/tools/decoder_add.py . . . 337
3.32 /opt/GSM/c54x_exe/tools/dsp_banc_commun.c . . . 338
3.33 /opt/GSM/c54x_exe/tools/dsp_banc_commun.h . . . 350
3.34 /opt/GSM/c54x_exe/tools/dsp_tester.c . . . 353
3.35 /opt/GSM/c54x_exe/tools/dsp_txt2bin.py . . . 366
3.36 /opt/GSM/c54x_exe/tools/injecteur_bruit.py . . . 369
3.37 /opt/GSM/c54x_exe/tools/isa_examples.py . . . 380
3.38 /opt/GSM/c54x_exe/tools/isa_test.c . . . 387
3.39 /opt/GSM/c54x_exe/tools/layer1_tester.c . . . 390
3.40 /opt/GSM/c54x_exe/tools/layer1_tester.py . . . 400
3.41 /opt/GSM/c54x_exe/tools/rejeu_banc.c . . . 406
3.42 /opt/GSM/c54x_exe/tools/sacch_tf_decode.c . . . 418
3.43 /opt/GSM/c54x_exe/tools/cch_ref/cch_interleave_ref.py . . . 420

4 Logs **423**
4.1 /opt/GSM/c54x_exe/toa.log . . . 423

1 Synthèse

run	élément	valeur
(pas de verdict.txt)		

dossier de run	contenu
Verdict et couverture	échelle des barreaux et tableau de couverture couche 1
Résultats des tests	captures VTY, diagnostics, rapports pytest, grafcets (.txt .md .mmd)
Fichiers	sources, scripts, configs et docs du banc (tout fichier texte)
Logs	journaux (.log), en dernier

i Méthode

Dossiers : /opt/GSM/c54x_exe. Extensions : tous les fichiers texte (hors .git, caches, binaires, sauvegardes, LICENSE et sorties précédentes). Entrée 1417 Ko, sortie 1412 Ko.
Compactage sans perte : fichiers identiques cités une fois ; lignes consécutives identiques « ×N » ; lignes ne différant que par des nombres : gabarit avec ⟨k⟩ puis la liste ordonnée des valeurs (k-uplets), chaque ligne se reconstruit en remettant les valeurs dans l'ordre. Seuls les codes couleur, la négociation telnet des captures VTY et les caractères de commande sont retirés ; les lignes de plus de 1000 caractères sont coupées en morceaux terminés par ↩. Les .md sont inclus tels quels (titres rétrogradés) et les .mmd rendus en diagrammes Mermaid, sans compactage.

2 Resultats des tests

2.1 /opt/GSM/c54x_exe/LAUNCH.md

11864 octets, 196 lignes

2.1.1 Lancer le côté mobile, processus par processus

Deux montages, cinq processus au plus, tous côté **mobile**. Le réseau (osmo-bts-trx, BSC, MSC, HLR…) n'est jamais lancé ici ; le pont l'attend s'il existe.

montage	nom à taper	ce qui tourne
dsp	`qosmo-dsp`	`c54x_exe --arm` → `qosmo` (QEMU, `CALYPSO_DSP_EXTERN=1`) → `osmocon` → `mobile` → `pont_dsp.py --dsp-port 6702`
grgsm	`qosmo-grgsm`	`qosmo` (QEMU, couche 1 gr-gsm intégrée) → `osmocon` → `mobile` → `pont.py`

`qosmo-dsp` et `qosmo-grgsm` sans argument lancent tout dans l'ordre, chaque étape attendant la précédente sur un critère observable. `--status`, `--logs`, `--stop`, `--step N`. Journaux et pid dans `/tmp/c54x-pont/`. Les anciens lanceurs C du même nom sont conservés en `qosmo-dsp-launch` / `qosmo-grgsm-launch` (ils visaient un rootfs ISO disparu) et reçoivent les appels avec options QEMU (`-k`, `-dsp`, …).

> [**2026-09-22**] Le tri des deux enveloppes (`/usr/local/bin/qosmo-dsp`, `/usr/local/bin/qosmo-grgsm`, hors dépôt) oubliait `--qemu`, `--cpu` et `--monitor`, et ne regardait que le **premier** argument. Or `qosmo-grgsm/run_modules/40-qemu.sh` appelle `qosmo-grgsm --qemu <bin> -k <elf> --bin <bin> --cpu arm946 --gdb N --rundir <dir> --monitor <dir>/qemu-monitor.sock` — `--qemu` en tête. Le motif ne matchait pas, l'appel partait donc sur `c54x_exe/run.sh`, qui sortait aussitôt sur `option inconnue : --qemu`. QEMU ne démarrait jamais et le module échouait trente secondes plus tard sur « socket du moniteur QEMU : toujours pas prêt » — un message qui désigne le moniteur alors que rien n'avait été lancé. Les deux enveloppes balaient maintenant **tous** les arguments et connaissent les six options du lanceur C. Elles ne sont dans aucun dépôt : une réinstallation les écrasera.

Chaque processus se lance aussi seul, par son nom, dans cet ordre.

2.1.1.1 1. `c54x_exe` — le DSP (montage dsp seulement)

```
c54x_exe                    # = /opt/GSM/c54x_exe/c54x_exe --arm -v
c54x_exe --arm --iq cell    # + une cellule GMSK synthétique (FCCH/SCH/factice)
c54x_exe --trames 50        # mode autonome, sans ARM : la mask-ROM seule
```

- **Rôle** : la mask-ROM TI du TMS320C54x, exécutée hors QEMU (`/opt/GSM/qosmo/hw/arm/calypso/l1-dsp/`).
- **Publie** : `/dev/shm/calypso_api_ram` (la fenêtre API, alias de `data[0x0800..]` du C54x) et `/tmp/calypso_dsp.sock` (verrou trame par trame, protocole `calypso_dsp_pont.h`).
- **Écoute** : UDP **6702**, les bursts descendants pour le BSP (`calypso_bsp.c`).
- **Attendu** : `pont : en attente de l'ARM sur /tmp/calypso_dsp.sock`.
- **Vérifier** : `ls -la /dev/shm/calypso_api_ram /tmp/calypso_dsp.sock`.
- **Options** : `--insns N` budget par trame (200000 par défaut avec `--arm`, 80000 via `run.sh`), `--iq fcch|cell|tone:x|noise`, `--amp N`, `-v` à `-vvvvvv` pour les traces du cœur (sondes pures coupées par défaut : `CALYPSO_SONDES=1` ou `-vvvv` les rallume).

- **Environnement** (2026-09-23) : `CALYPSO_BSP_ATTENTE_MS=40` (posé par `run.sh`), `PONT_TCH_DEPOT_IDLE=0` (ancien dépôt du burst TCH, avant l'IDLE), `PONT_DONE_TOT=0` (DONE rendu en fin de trame, ancien ordre). `dsp.log` imprime toutes les 1000 trames `[chrono] ... qemu | A | go | B | apres DONE`. Sondes : `/dev/shm/calypso_bsp_dedie` (magasin dédié du BSP : joués/manqués/perdues) ; sur canal dédié, enregistrement `/dev/shm/calypso_rejeu_tch.bin` pour `tools/rejeu_banc` (`CALYPSO_REJEU_ENREG=0` coupe).
- **Lien montant** : publie `/dev/shm/calypso_rach`, `calypso_sdcch_ul`, `calypso_tch_facch_ul`, `calypso_tch_sacch_ul`, `calypso_tch_ul` en scrutant l'API RAM (`src/montant.c`) — c'est par là que le RACH du mobile atteint pont.py puis la BTS. `MONTANT=0` coupe, `MONTANT_DEBUG=N` règle les traces, `MONTANT_CONSOMME_RACH=1` rend le déclenchement exact. Depuis le 2026-09-23 : `MONTANT_KC=0` (Kc dans `calypso_kc_l1`), `MONTANT_PAROLE_TI=0` (parole montante brute, sans conversion TI -> FR), `MONTANT_TCH_TACHE=0` (bascule TCH à l'annonce du pont au lieu de la tâche du firmware), `MONTANT_AFD=0` / `MONTANT_ADD=0` (sondes `[a_fd]` / `[a_dd]`). **Vérifier** : `./c54x_exe` imprime `[montant] RACH ra=0x.. bsic=..` et `pont.log` passe de `rach=0` à `rach=N`. En appel : `[montant] TCH : le firmware poste la tache 13 a fn=..., BSP bascule sur TS2`, puis `le firmware est revenu sur le SDCCH (tache ALLC)` à la libération.

2.1.1.2 2. `qosmo` — l'ARM et la layer1 osmocom-bb

```
# montage dsp
CALYPSO_DSP_EXTERN=1 /opt/GSM/qosmo/build/qemu-system-arm -M calypso -cpu arm946 \
  -display none -parallel none -serial pty -serial pty \
  -monitor unix:/tmp/qemu-monitor-pont.sock,server,nowait \
  -kernel /opt/GSM/firmware/board/compal_e88/layer1.highram.elf
# montage grgsm : la même ligne sans CALYPSO_DSP_EXTERN
```

- **Rôle** : le Calypso (ARM946) qui exécute `layer1.highram.elf`. `-kernel` est obligatoire : sans lui le CPU part à 0 et plante en `0x840000`, le romload d'osmocon ne charge rien.
- **Avec `CALYPSO_DSP_EXTERN=1`** : la fenêtre API `0xFFD00000` est le segment partagé, la couche 1 gr-gsm est désactivée avant d'ouvrir ses ports, le registre `CNTL_RST` relaie `RESET_DSP` au DSP, un timer de boot cadence le DSP avant l'activation du TPU.
- **Sans** : la couche 1 gr-gsm (le shunt) écoute UDP **4730** (GSMTAP) et **4731** (SCH), nourris par le pont.
- **Attendu** (stderr) :

  ```
  char device redirected to /dev/pts/N (label serial0)       <- le pty modem
  [trx] pont DSP : API RAM partagee ... + socket /tmp/calypso_dsp.sock      (dsp)
  [trx] pont DSP : RESET_DSP relache par le firmware -> PONT_RESET          (dsp)
  [trx] pont DSP : le TDMA du firmware prend le relais du timer de boot     (dsp)
  [l1] backend gr-gsm : GSMTAP udp/4730, SCH udp/4731                       (grgsm)
  ```

  et côté `c54x_exe` : `RESET #1 ... pc=0xff80` puis `DSP boote (premier IDLE)`.
- **Vérifier** : `printf 'xp /96bx 0x008305f0\n' | nc -U /tmp/qemu-monitor-pont.sock` montre le dernier printf du firmware (`DSP API Version: 0x4e2a 0x491a`). `run.sh` écrit le pty dans `/tmp/c54x-pont/modem.pty`.
- **Ce que `run.sh` ajoute** (2026-09-23) : `-gdb tcp:127.0.0.1:1234` et la console `telnet 0 44444` (`qosmo-dsp/tools/gdb-telnet.py` ; Ctrl-C arrête l'ARM, « continue & » le relance ; `GDB=0` coupe) ; `CALYPSO_PONT_RETRY_DIV=64` et, en `LOCKSTEP=1` (défaut), `CALYPSO_PONT_LOCKSTEP=1` sur QEMU ; `ASSEMBLY_LOGS=1` donne `qemu-asm.log`. `qemu.log` s'écrit dans `/run/user/0/osmo-nitb/logs/qemu.log`, avec un lien dans `/tmp/c54x-pont/`.

2.1.1.3 3. `osmocon` — le chargeur et le relais L1CTL

```
osmocon                    # = osmocon -m romload -i 100 -p $(cat /tmp/c54x-pont/modem.pty) \
                           #      -s /tmp/osmocom_l2 layer1.highram.bin
```

- **Rôle** : joue le protocole romload avec le stub UART de QEMU, puis relaie le L1CTL entre le firmware (sercomm sur le pty) et la socket `/tmp/osmocom_l2`. Affiche la console du firmware (`FB0 (fn:att): TOA=… Power=… Angle=…`).
- **Attendu** : `Received ident ack`, `Progress: 100%`, **`Received branch ack, your code  is running now!`**.
- **Piège** : un osmocon tué en plein téléchargement laisse le stub romload de l'UART à mi-bloc ; relancer QEMU (étape 2) puis osmocon.

2.1.1.4 4. `mobile` — les couches 2/3

```
mobile -c /opt/GSM/c54x_exe/mobile_pont.cfg     # layer2-socket /tmp/osmocom_l2, VTY 4347
```

- **Config** (2026-09-23) : `layer2-socket /tmp/osmocom_l2`, `sap-socket /tmp/osmocom_sap` (les mêmes que sans `--dsp`, plus de variantes `_pont`), VTY 4347 ; `tch-voice` gapk `io-tch-format ti`, ALSA `gsm_out`/`gsm_in` ; `tch-data` unix-sock `/tmp/ms_data` (CSD 9600). `run.sh` pose `L23_SYNC_RETRIES_SELECTION=8` en `MODE=dsp` (binaire mobile partagé, défaut 1 ailleurs).
- **Attendu** dans les 5 s : côté osmocon `L1CTL_PM_REQ`, `L1CTL_RESET_REQ: FULL!`, `L1CTL_FBSB_REQ (arfcn=514 …)` ; côté DSP le passage de 428 à ~570 insn/trame et `a_sch=0100 …` (la L1 ARM a posé `d_task_md=5`, recherche FB).
- **Sans burst** (pas de pont, ou pas de BTS) : `FBSB RESP: result=255` en boucle, attendu.
- **Vérifier** : `telnet 127.0.0.1 4347` puis `show ms`. Le port 4347 est hors de la plage 42xx des composants réseau ; `run.sh` refuse de lancer si le port est pris.

2.1.1.5 5. `grgsm_exe` — le pont TRX (gr-gsm)

```
grgsm_exe                    # = python3 pont/pont_dsp.py --no-record --dsp-port 6702
PONT_DSP_PORT=0 grgsm_exe    # montage grgsm : pont/pont.py, vers la L1 de QEMU seulement
```

- **Rôle** : reçoit les bursts du BTS en TRXD (UDP **5700-5702** depuis osmo-bts-trx), les décode avec gr-gsm et alimente la couche 1 du mobile ; renvoie l'uplink au BTS.
- **Vers la L1 gr-gsm** (montage grgsm) : blocs L2 en GSMTAP sur 4730, FN de synchro sur 4731.
- **Vers le DSP** (montage dsp, `--dsp-port 6702`, ajouté le 17/09) : chaque burst DL est ré-emballé au format du BSP, 8 octets `[tn, fn BE32, att, 0, 0]` + 148 bits 0/1 ; le BSP les convertit en I/Q et les dépose en DARAM `0x2a00`, d'où le DSP les lit.
- **Par `run.sh`** (`PONT=1`, défaut 0) : le pont est lancé **sans** `--no-record` (`PONT_AIRREC=1`, pour la FFT du panneau), à la différence de l'enveloppe `grgsm_exe`.
- **Attendu** : `pont TRX : ports 5700/5701/5702, ARFCN 514, BSIC 7` puis des lignes `STATS fn=… | DL bursts=N`. `DL bursts=0` tant qu'aucun BTS n'émet : normal sans réseau.

2.1.1.6 Ordre d'arrêt et nettoyage

`qosmo-dsp --stop` (ou `run.sh --stop`) arrête dans l'ordre inverse (pont, mobile, osmocon, gdb, QEMU, DSP) et efface `/dev/shm/calypso_api_ram`, `/tmp/calypso_dsp.sock`, `/tmp/osmocom_l2`, `/tmp/qemu-monitor-pont.sock`, `/tmp/osmocom_sap`, `/tmp/ms_data`, `/dev/shm/calypso_horloge`, les side-bands montants (`calypso_rach`, `calypso_sdcch_ul`, `calypso_tch_*_ul`) et, en montage dsp, `calypso_tch_cfg`. La session est rangée dans `/tmp/c54x-pont/archives/<date>/` (`JOURNAUX_GARDES=10`).

2.1.1.7 Ce que ça donne aujourd'hui (23/09, runs du banc DSP de 20:22 et 20:32)

- Montage dsp avec la BTS (`PONT=1`) : SCH et BCCH décodés, LU ACCEPT, SMS MO et MT dans les deux sens, appel MO vers l'écho 600 et appel MT depuis 100102 complets (ACTIVE, DISCONNECT, TCH fermé), A5/1 confirmé par la BTS sur les cinq établissements, parole audible dans les deux sens (run de 20:22). Aucune LOS, aucune ligne `[garde-3d89]` : le correctif MVKD/MVDK du cœur `qosmo` tient sur ce run. Plus de SABM répétés depuis `pont/dsp/clock.py` (aucune ligne SABM dans les journaux osmocom).
- Ouvert, par ordre d'importance : B_BFI sur toute la parole (run de 20:32, sonde `[a_dd]` étendue, non commitée : `bfi=2200` sur `vues=2200`, `err` 0 sur 19 des 20 premières trames, puis 15 à 93) ; une LOS en TCH au run de 20:32 (premier appel, 20:32:45, SACCH/TF FIRE KO à chaque bloc, garde muette ; l'appel suivant reste en T3230, le troisième est sain) ; le SDCCH/8 (27 trames jetées à 20:22, dont 15 SACCH) ; la synchro SB qui ne passe qu'une fois sur trois à cinq ; la marge temps réel en TCH (4.3 à 4.6 ms de travail DSP pour 4.62 ms, `[chrono]`). Le détail : `MAILBOX.md`.

Historique (17/09) :

- Montage dsp, cellule synthétique (`IQ=cell qosmo-dsp`) : le DSP **détecte la FCCH** (`d_fb_det=1`, `FB0`/`FB1` avec TOA/puissance/angle réels), grâce à deux bugs du décodeur corrigés ce jour (`CALYPSO_FIX_NORM_SD`, `CALYPSO_FIX_F7_DELAYED`). Le décodage du SCH reste ouvert (`a_sch[3]=0xf8d8` constant, `DSP Error Status: 8`), cf. `qosmo-dsp/hw/arm/calypso/doc/RAPPORT_DFBDET.md` §8.
- Montage grgsm : la chaîne complète du banc, qui campe et fait des appels dès que le réseau et un BTS sont là.

2.2 /opt/GSM/c54x_exe/MAILBOX.md

183099 octets, 3215 lignes

2.2.1 Boîte aux lettres ARM <-> DSP (Calypso API RAM)

Deux vues de la MÊME mémoire : l'ARM l'adresse en octets depuis 0xFFD00000, le DSP en mots depuis 0x0800. Conversion : `mot_dsp = 0x0800 + octet_arm/2`.

Sources : `qosmo/include/hw/arm/calypso/calypso_api.h` et `osmocom-bb/src/target/firmware/include/calypso/dsp_api.h` — vérifiés concordants (`#define DSP 36`, branche a_serv_demod/a_pm/a_sch).

2.2.1.1 Plan général

```
zone          ARM (octets)   DSP (mots)   taille
W page 0      0xFFD00000     0x0800       20 mots   ARM -> DSP
W page 1      0xFFD00028     0x0814       20 mots
R page 0      0xFFD00050     0x0828       20 mots   DSP -> ARM
R page 1      0xFFD00078     0x083C       20 mots
NDB           0xFFD001A8     0x08D4      268 mots   partagé, non paginé
```

2.2.1.2 Page W (ARM -> DSP), offsets en mots

```
+0  d_task_d      0x0800 / 0x0814
+1  d_burst_d
+2  d_task_u
+3  d_burst_u
+4  d_task_md     0x0804 / 0x0818   <- la mission : 5=FB 6=SB 8/9=TCH
+8  d_fn          0x0808 / 0x081C
+15 d_afc         0x080F / 0x0823   <- DAC AFC relayé au TWL3025
+16 d_ctrl_system 0x0810 / 0x0824
```

2.2.1.3 Page R (DSP -> ARM), offsets en mots

```
+0   d_task_d
+8   a_serv_demod[4]  0x0830 / 0x0844   TOA, PM, ANGLE, SNR
+12  a_pm[3]          0x0834 / 0x0848
+15  a_sch[5]         0x0837 / 0x084B   <- en-tête + charge utile SB
     a_sch[0] statut : bit15 B_BLUD (bloc présent), bit8 B_SCH_CRC (1=ERREUR)
     a_sch[3..4] : le mot SB, sb = a_sch[3] | a_sch[4]<<16
                   BSIC = (sb>>2) & 0x3f
```

2.2.1.4 NDB (partagé)

```
+0    d_dsp_page      0x08D4   0=reset, 2=armé page0, 3=armé page1
+14   d_dsp_state     0x08E2
+36   d_fb_det        0x08F8   non nul = FCCH trouvée
+37   d_fb_mode       0x08F9   0=recherche large, 1=étroite
+38   a_sync_demod[4] 0x08FA   TOA / PM / ANGLE / SNR
                      0x08FA TOA, 0x08FB PM, 0x08FC ANGLE, 0x08FD SNR
```

2.2.1.5 Qui écrit quoi — côté ARM (osmocom-bb, vérifié en source)

```
dsp_end_scenario()   calypso/dsp.c:471
    d_dsp_page = B_GSM_TASK | w_page ; puis w_page ^= 1
l1s_reset_hw()       layer1/sync.c:158
    d_dsp_page = 0 ; r_page = 0 ; db_r -> R page 0
    appelé par prim_fbsb.c:254 et :429, donc à CHAQUE cycle FBSB raté
l1s_dsp_post()       layer1/sync.c:263
    memset de la page R, puis a_sch[0] = (1<<B_SCH_CRC) = 0x0100,
    puis r_page ^= 1
-> l'ARM n'écrit a_sch[0] qu'avec 0x0100. Rien d'autre.
```

2.2.1.6 Qui écrit quoi — côté DSP (ROM masque, PC relevés à l'adresse exacte)

```
0xb446   a_sch[0..4] <- 0 sur LES DEUX pages        initialisation
0xaba2   a_sch[0] <- 0x1111                         marqueur
         précédé de 0xaba0 : LD #0x1111, A
         encodage 80e2 000f = store *(AR2 + 0x0f), AR2 = base page R
0xb214   a_sch[0..4] <- statut + charge utile       RÉSULTAT
         statut observé : 0x8100 (BLUD + CRC faux) uniquement
0xb2cc   d_fb_det <- 0        entrée de tâche FB : remise à zéro
0xb2cf   a_sync[TOA] <- 0
0xb2d2   a_sync[PM] <- 0
0xb2d5   a_sync[ANGLE] <- 0
0xb2d8   a_sync[SNR] <- 0
0xb2c4   *(0x3fb5) <- 0x0cce   pointeur du tampon d'entrée du corrélateur FB
0xb2c9   *(0x3fb5) <- 0x0d2e   idem, second tampon
0x795a   a_sync[TOA] <- valeur
0x798b   a_sync[ANGLE] <- valeur
0x798d   a_sync[SNR] <- 0x4000
0x79d4   a_sync[SNR] <- valeur
0x79de   a_sync[PM] <- valeur
```

2.2.1.7 Séquence d'une tâche SB

```
ARM : d_task_md = 6 sur la page W courante
ARM : d_dsp_page = 2|w_page       (dsp_end_scenario)
DSP : 0xb446 met a_sch à zéro
DSP : 0xaba2 pose le marqueur 0x1111
DSP : 0xb214 écrit le résultat, a_sch[0] = 0x8100 (CRC faux)
ARM : lit db_r->a_sch, voit B_SCH_CRC=1, conclut « SB not found »
ARM : memset page R, a_sch[0] = 0x0100, r_page ^= 1
ARM : après 2 tentatives, L1CTL_RESET_REQ -> l1s_reset_hw -> d_dsp_page = 0
```

2.2.1.8 Pièges rencontrés

- Deux chemins d'écriture côté émulateur : `data_write()` et `data_write_locked()` (c54x_mem.c). Une sonde posée sur un seul en rate la moitié — a_sch passe par le second, a_sync_demod par le premier.
- L'ARM et le DSP sont dans DEUX PROCESSUS qui partagent ce mapping. Une sonde qui compare avant/après une instruction DSP peut imputer au DSP une écriture de l'ARM. Toujours relever l'adresse passée en argument, jamais un delta.
- d_dsp_page annonce la page d'ÉCRITURE de la tâche. La page de LECTURE est `dsp_api.r_page`, un compteur distinct côté ARM. Les deux ne sont pas liés.

2.2.1.9 Pourquoi le TOA ne vaut jamais 23 [2026-09-19]

Le firmware vise un ToA de 23 (`prim_fbsb.c:207 last_fb->toa -= 23`) et ne le voit jamais. Mesure sur 49 stores de `a_sync[TOA]` par la ROM en 0x795a, le mot stocké valant exactement `B` et `T` valant 48 sur tous :

```
AR2 = 0x0cce  (tampon d'entree du correlateur FB)   26 stores
   valeurs : 48 x10, 96 x8, 144 x4, 192 x2, 240, 384
   multiples exacts de 48 : 26/26 = 100 %

AR2 = autre pointeur                                23 stores
   valeurs : 27, 31, 43, 47, 51, 55, 71, 79, 87, 91
   multiples de 48 : 0/23 = 0 %
```

Separation parfaite, aucun recouvrement. Et 48 echantillons complexes = 96 mots = UNE PAGE DMA (ALGTH=192 octets, mesure “en 4 page(s)” de 96 mots pour un burst de 296).

Quand le correlateur lit le tampon de burst, son pic tombe donc sur une frontiere de page DMA, a tous les coups. 23 est a l'interieur d'une page : il n'appartient pas a l'ensemble des valeurs atteignables. Ce n'est pas un probleme de seuil ni de rapport signal/bruit.

```
0x7953 :  770e 0030    store de la constante 48 -> T
0x795a :  81f8 08fa    STH A, *(0x08fa)  -> a_sync[TOA], vaut B
```

A rapprocher de calypso_rhea_dma.c:490, qui decrit le mode defaillant du chainage de pages comme “corrupted FCCH, correlator peaking at the edge (TOA=39)”. Le correctif “pages contigues” a fait tomber la proportion de multiples de 48 de 86,6 % (CALYPSO_RHEA_DMA_PINGPONG=1) a 46,4 %, sans l'eliminer : il reste une couture a la granularite de la page.

2.2.1.10 Chaine complete : l'angle mort commande le doublement d'horloge [2026-09-19]

```
a_sync[ANGLE] rend ~0 quel que soit l'offset injecte (pente nulle sur +-2000 Hz)
  -> prim_fbsb.c:312  freq_diff = ANGLE_TO_FREQ(angle) ~ 0
  -> prim_fbsb.c:488  if (abs(freq_diff) < freq_err_thresh2 && snr > FB1_SNR_THRESH)
                      mesure a chaud : thresh2 = 800 Hz (0x008320b2),
                      FB1_SNR_THRESH = 0 (les seuils 2000/3000 sont sous #if 0),
                      SNR = 0x4000 ecrit par la ROM en 0x798d
                      -> 175/180 detections passent le garde, soit 97 %
  -> prim_fbsb.c:492  synchronize_tdma() sur le chemin FB, rien ne repose
                      l'horloge derriere
  -> prim_fbsb.c:345  fnr_delta = fnr_report - attempt, fnr_report etant une
                      COPIE de current_time.fn et attempt valant 1 ou 2
  -> prim_fbsb.c:348  cinfo->fn_offset = fnr_delta   (un ABSOLU)
  -> sync.c:149       l1s_time_inc(current_time, fn_offset)  -> horloge doublee
  -> bursts normaux EMPTY -> L1CTL_RESET_REQ -> le cycle FBSB repart
```

L'autre appel, prim_fbsb.c:231 (chemin SB), est inoffensif : gsm_fn2gsmtime() repose l'horloge en absolu trois lignes plus loin. D'ou l'echec du SB force : il repare une fois par cycle ce que le chemin FB casse a chaque detection.

Mesure a chaud confirmant le doublement, par le moniteur QEMU :

```
current_time.fn   fn_offset    rapport
   64797           129070       1.992     fn_offset gele entre deux FB
   65499           129070       1.971
   65565           131012       1.998     rafraichi a ~2x l'horloge
```

Adresses de surveillance (xp/1wx sur /tmp/qemu-monitor-pont.sock) :

```
l1s.current_time.fn         0x0083762c
l1s.serving_cell.arfcn      0x00837644   = 514
l1s.serving_cell.bsic       0x00837646   = 42 avec PONT_CAN_SB
l1s.serving_cell.fn_offset  0x00837648   doit rester petit ; vaut ~2x l'horloge
fbs.req.band_arfcn          0x008320ac   = 514
fbs.req.freq_err_thresh2    0x008320b2   = 800
```

2.2.1.11 Le chemin SB lit d'AUTRES cellules que le chemin FB [2026-09-19]

```
read_fb_result()  prim_fbsb.c:306   dsp_api.ndb->a_sync_demod[]   NDB    0x08FA..0x08FD
read_sb_result()  prim_fbsb.c:148   dsp_api.db_r->a_serv_demod[]  page R 0x0830 / 0x0844
```

Deux groupes distincts. Toute mesure faite sur l'un ne dit rien de l'autre.

Les quatre resultats du SB sont publies par la ROM en 0xb1e7..0xb1f5 depuis quatre cellules de travail, avec ANGLE et SNR CROISES par rapport a l'ordre des adresses :

```
0x3fa4 -> a_serv_demod[TOA]     store 0xb1e9
0x3fa5 -> a_serv_demod[PM]      store 0xb1ed
0x3fa7 -> a_serv_demod[ANGLE]   store 0xb1f1
0x3fa6 -> a_serv_demod[SNR]     store 0xb1f5
la cellule 0x3fa6 est elle-meme ecrite en PC=0x7e88
```

2.2.1.12 L'AFC est sous le seuil, pas morte

```
AFC_SNR_THRESHOLD = 2560   (afc.h:4)
AFC_PERIOD        = 40     AFC_MIN_MUN_VALID = 8   (afc.c)

mesure sur 111 publications de a_serv_demod[SNR] :
   SNR > 2560 : 17 = 15,3 %   -> 6,1 mesures valides par fenetre de 40
   il en faut 8. Distribution BINAIRE : 16384 (0x4000) ou un ou deux chiffres.

runavg_check_output (avg.c:34) ne remet PAS les compteurs a zero quand le
minimum n'est pas atteint, donc l'AFC finit par emettre -- vers 52 trames.
Or un cycle FBSB dure 53,6 trames en moyenne et afc_reset() remet le DAC a
-700 a chaque l1s_reset_hw(). Les deux echeances sont quasi identiques :
la correction arrive au moment ou elle est effacee.
```

2.2.1.13 Ce qui est DISCULPE par la mesure

- Le tampon du correlateur 0x0cce : quand une vraie FCCH y est, le DSP la voit parfaitement – coherence 0.999, dphi +1.565 pour +1.571 theorique (mesure fn=2060, p51=20). Le signal arrive et l'estimateur le reconnait.
- Le chemin de resultat SB de bout en bout : avec PONT_CAN_SB, le firmware extrait BSIC=42 sur 14 acceptations sur 14, et le stocke en l1s.serving_cell.bsic (0x00837646). Ecriture des deux pages, handshake r_page, read_sb_result, desassemblage T1/T2/T3' : tout fonctionne.
- L'AFC comme cause de l'echec SB : CALYPSO_TWL3025_AFC=0 ne change ni le verdict CRC ni la quantification du TOA.

2.2.1.14 Retractations de la session

- "le DSP n'ecrit jamais a_sch" : faux, sonde posee sur un seul des deux chemins d'ecriture (data_write vs data_write_locked).
- "le maximum du SNR est 864, 100 % des mesures AFC invalides" : faux, echantillon tronque. Le maximum est 16384 et 15,3 % passent le seuil.
- "l'estimateur d'angle est mort" : vrai pour le NDB (chemin FB), faux pour a_serv_demod (chemin SB) ou il prend des valeurs variees.
- "le doublement d'horloge est benin car ecrase par la pose absolue" : faux, l'ordre reel est inverse sur le chemin FB (prim_fbsb.c:492).

2.2.1.15 Pourquoi le TOA vaut 1251 en vivant et pas en rejeu [2026-09-20]

Rejeu (`--rejouer`) : le TOA du FB1 croit avec la distance de la FCCH dans la fenetre (6288, 7584, 8832, 9991, 11243 = 5 a 9 trames), ntdma est juste, la SB est decodee : 56 CRC OK sur 3000 trames, 56/56 avec le BSIC injecte, T3 valide et FN = trame du burst. Balayage BSIC 0/7/13/21/42/63 : `BSIC=(sb>>2)&0x3f` colle a chaque fois. La sortie depend de l'entree.

Vivant (`run.sh`) : TOA = 1251 (ou 1247, 1296) a CHAQUE detection, quelle que soit la distance de la FCCH. Cause, dans qemu.log :

```
[trx] pont DSP : DSP en retard, tick saute (fn=5548, 4340 sauts, 1202 trames jouees)
```

Le C54x emule coute 6,7 ms par trame (mesure : 500 trames de rejeu en 3,35 s) contre 4,615 ms de temps reel GSM. QEMU cadence le TDMA a l'horloge murale et saute la trame quand le DONE du DSP n'est pas arrive : 3 trames sur 4 perdues, la ROM ne recoit qu'une trame sur 4 a 6 (transferts DMA en paquets de 13 paires aux fn 252, 258, 264, 270, 276…). Son compteur de blocs FB n'avance donc que d'une trame environ entre la commande et la FCCH : TOA ~ 1250 + quelques echantillons, ntdma = 0, fn_offset faux, la SB visee tombe 1 a 2 trames apres la trame SCH, CRC KO a tous les coups.

Correctif : `CALYPSO_PONT_LOCKSTEP=1` (calypso_trx.c, existait deja) - QEMU n'avance la trame que quand le DSP a fini la precedente. `run.sh` l'exporte par defaut en montage dsp (`LOCKSTEP=0` pour revenir a l'horloge murale). Mesure en pas-a-pas : plus aucun saut (trames=1953 a fn=1951), TOA = 11239 a 11243 (ntdma=8, delay=10, comme en rejeu), et dans osmocon.log :

```
SB1 (1089343:1): TOA=   24, Power= -52dBm, Angle= -152Hz
=> SB 0x001001a8: BSIC=42 fn=3826(2/ 4/ 1) qbits=4
```

BSIC 42 = celui de la cellule synthetique, TOA 24 pour un attendu de 23 : la ROM a decode une vraie SB sur le chemin vivant, ARM reel + DSP reel.

2.2.1.16 Les SB « delirantes » : une page R lue a zero [2026-09-20]

Symptome (osmocon.log) :

```
SB1 (1649571:1): TOA=    0, Power=-138dBm, Angle=    0Hz
=> SB 0x00000000: BSIC=0 fn=52(0/ 0/ 1) qbits=4908
```

Le mot SB vaut 0, TOA 0, puissance -138 dBm (a_serv_demod a zero aussi) : l'ARM a lu une page R que la ROM venait de remettre a zero en 0xb446 (init de tache, LES DEUX pages) et sur laquelle aucun resultat n'etait encore ecrit. `l1s_sbdet_resp` ne teste que B_SCH_CRC (bit 8) : 0x0000 passe pour un CRC OK et BSIC=0 / FN=52 sont pris pour argent comptant. Le firmware se cale alors sur un FN faux, lit les bursts normaux n'importe ou et le mobile jette tout (`Dropping frame with 210 bit errors` : ~46 % d'erreurs sur 456 bits, du hasard). Vu dans qemu.log comme `page0 0100->0000` suivi d'une lecture ARM de la page 0.

Deux populations de CRC OK, a distinguer par le mot lui-meme :

```
sb = 0, TOA = 0, PM = -138 dBm    -> page vide, faux positif
sb != 0, TOA ~ 23, PM ~ -52 dBm   -> vraie SB (BSIC 42 ici)
```

En rejeu le faux positif n'existe pas : l'ARM rejoue lit a la fin de la trame, apres l'ecriture du DSP. Il n'apparait qu'avec l'ARM QEMU, dont la lecture peut tomber entre la remise a zero et le resultat, ou sur l'autre page R quand les bascules r_page (ARM) et page du DSP se sont desynchronisees par des trames sautees. Le pas-a-pas reduit le second cas ; le premier reste a mesurer.

Le mobile apres une SB acceptee : avec `PONT=0` la cellule synthetique n'a que FCCH, SCH et bursts factices ; les trames BCCH decodees sont donc du bruit (`Dropping frame with N bit errors`, N ~ 190-224) meme quand la SB est vraie. Pour aller au-dela il faut le BTS via pont.py (`PONT=1`) ou des bursts BCCH (SI1-4) dans cellule.c.

2.2.1.17 Etat du banc ISA [2026-09-20]

`make isa_test && ./isa_test tools/isa_tests.txt` : 208 exemples SPRU172C, 139 ok, 69 FAIL, 22 non assembles. Une partie des FAIL vient d'attendus mal extraits du PDF (ex. `LD *AR4+, A` attend un AR5 qui n'intervient pas), le reste sont de vrais ecarts (RETF, RPTB, SUBC, MVDP/MVPD, NEG/RND/SFTA sur les drapeaux). Aucun n'empeche le decodage SB observe ci-dessus.

2.2.1.18 Romload fige a 38-55 % en pas-a-pas [2026-09-20]

Symptome : osmocon reste sur `Progress: 38%` (ou 55 %), QEMU dit pourtant que le firmware charge par `-kernel` a deja lance son TDMA. Cause : c'est `tdma_tick` (calypso_trx.c) qui pompe le pty serie vers l'UART emulee (`calypso_uart_poll_backend`). Sous `CALYPSO_PONT_LOCKSTEP=1`, quand le DSP n'a pas fini la trame precedente, le tick sortait AVANT ce pompage et se rearmait : la serie n'avancait qu'au rythme du DSP (~7 ms par trame) et le romload, qui a un delai par bloc, decrochait. Les runs precedents passaient de justesse. Correctif (qosmo 77f61dd) : l'UART est pompee aussi sur le chemin d'attente du DSP. Mesure : 71 blocs, « your code is running now », puis FBSB_REQ dans la foulee.

2.2.1.19 L'interruption trame du DSP est un bit a usage unique [2026-09-20]

Etat de depart, en pas-a-pas : timing FB juste (TOA 11239/11243, delay=10), tache SB postee sur la trame SCH avec le burst S livre, et pourtant 1 SB en 130 cycles. Sondes :

```
D_TASK_MD-RD : la ROM lit d_task_md a CHAQUE trame (0xb011 -> 0xb554 ->
               0xb0b4 -> 0xab7a pour FB, -> 0xaba4 pour SB) et relit la
               MEME page avec la meme valeur 3 a 4 trames de suite.
PONT_PC_COUNT : aba4 (dispatch SB) 27 passages / 217 trames, b219 : 0.
DMA2 (PC ajoute aux journaux) : armements par 0xa5ef/0xa5f6, desarmement
               0xa646 ; en vivant, flux continu FB rearme a CHAQUE FCCH
               (trames 204, 214, 234, 244...), jamais de fenetre 764.
Rejeu : fenetre one-shot de 764 octets (382 mots) armee sur la trame de
               la commande SB, et la FB1 (mode etroit) en une fenetre 764
               a la trame predite, pas en flux continu.
```

Pourquoi la ROM relit la page : sync.c l1_sync() efface la page W COURANTE a chaque trame (ligne 244) mais ne bascule w_page que si la trame porte un item DSP (dsp_end_scenario, ligne 276). La page remise au DSP garde donc sa tache tant qu'aucun nouveau scenario ne rebascule ; la ROM n'ecrit jamais d_task_md ni d_dsp_page (WATCH-WR : 0 ecriture). Le rejeu, lui, bascule w_page a chaque trame (rejouer.c l1_sync) : les pages y sont propres, d'ou le decodage.

Ce qui l'empeche sur silicium : dsp_end_scenario() appelle tpu_dsp_frameirq_enable() (TPU_CTRL_DSP_EN) a CHAQUE scenario et personne ne l'eteint jamais (tpu_frame_irq_en(1,1) seulement). Un bit qu'on rearme a chaque fois est un bit a usage unique : le TPU ne donne l'interruption trame au DSP que sur les trames ou l'ARM lui a remis une page. Le pont la levait a chaque tick. qosmo ne modelisait pas ce bit (le commentaire de calypso_c54x.c qui le pretend est perime : grep TPU_CTRL_DSP_EN ne donne que le #define).

Correctif : qosmo (calypso_trx.c) met dans TICK.b bit 16 « l'ARM a arme DSP_EN depuis le tick precedent » et consomme le bit ; pont.c ne leve vec 28 que sur ce bit (PONT_IRQ_TRAME=1 pour l'ancien comportement). Verifie d'abord en rejeu avec REJEU_IRQ_SCENARIO=1 : memes TOA, meme taux SB (27/76).

Mesure en vivant, 10 cycles FB1 :

```
SB acceptees            : 5, toutes BSIC=42, TOA=24, -51 dBm   (avant : 1/130)
fenetres one-shot 764   : 7 (380 mots), burst S (n_iq=380) sur p51 = 41/11/31
flux FB continu         : plus rearme a chaque FCCH
```

Restes : - FB1 lue vide : `FB1 (5294:8): TOA=0 Power=-138dBm` deux fois de suite, meme mecanisme que la SB delirante (a_sync pas encore ecrit), l'ARM retente. - FB0 a l'essai 1 avec TOA=1296 : la premiere paire de pages porte encore une FCCH du flux precedent (src=GRILLE), detection immediate et fausse distance. - Apres une SB vraie le mobile lit le BCCH et jette tout (`Dropping frame with  208 bit errors`, `MM_EVENT_NO_CELL_FOUND`) : la cellule synthetique n'a ni SI1-4 ni bursts normaux. Etape suivante : bursts BCCH dans cellule.c, ou `PONT=1` avec le BTS.

2.2.1.20 Banc BTS reel (pont.py –dsp-port 6702) : ce qui manquait [2026-09-20]

1. pont.py sans `--dsp-port 6702` : rien ne part vers le DSP (defaut 0 = coupe).
2. Le BSP n'appariait les bursts que par FN (fenetre +-64) : le FN du firmware est arbitraire avant la SB et saute de centaines de milliers a chaque detection FB (prim_fbsb.c, l1s_time_inc absolu). Rien n'etait livre. `CALYPSO_BSP_STREAM=1` livre dans l'ordre d'arrivee, c'est le chemin a utiliser des qu'une source temps reel alimente le DSP.
3. Le BSP ajoutait ses 7 timeslots de remplissage derriere chaque TS0 alors que pont.py envoie les 8 TS : 15 TS par trame (qosmo 6ad7003).
4. Le DSP consomme ~100 trames/s contre 217 emises : la file de 128 bursts par TS debordait chaque seconde et jetait les plus anciens ; espacements FCCH mesures 4, 26, 18 trames au lieu de 10, 88 SB tentees / 0 decodee. File portee a 8192 (qosmo 54d8320) : flux coherent, seulement en retard.
5. La livraison STREAM chargeait les bursts bruts (148 symboles) et rien pour les TS idle que le BTS n'envoie pas : trames de 1184 symboles ou moins, le compteur de la ROM derivait (offsets intra-trame du TOA FB : 264, 434, 632, 399). Assembleur de trame (qosmo 28e9989) : 8 TS a 156/157 symboles, burst factice pour les absents, fenetre SB = TS0 + 21 de marge.

Vitesse du coeur (qosmo d68baaf, 54d8320) : getenv memoise, chemin rapide sans sondes ni mutex par acces memoire. Rejeu 6,7 -> 2,5-3,0 ms/trame. Le vivant reste vers 10 ms (24-38 k instructions de ROM par trame, ~200 ns chacune, le corps de c54x_run porte encore des dizaines de comparaisons de sonde par instruction) : lockstep toujours necessaire, et la file profonde compense le retard sur le BTS.

2.2.1.21 Bursts BCCH dans la cellule, ordre ARM/DSP dans la trame [2026-09-21]

cellule.c porte maintenant les bursts normaux du TN0 : SI1-4 sur le bloc BCCH (p51 2..5, TC = (fn/51)%8 : SI1 0/4, SI2 1/5, SI3 2/6, SI4 3/7), paging vide sur les CCCH (6..9, 12..15, 16..19), factice ailleurs. Codage libosmocoding (gsm0503_xcch_encode) + assemblage sched_lchan_xcch.c d'osmo-bts, TSC = BCC, identite MCC 001 MNC 01 LAC 1 CI 6001 ARFCN 514 (CELLULE_MCC/MNC/LAC/CI/ARFCN). Verifie hors DSP : les 4 bursts reassembles redonnent les 23 octets exacts (xcch_decode). Fenetre NB de la ROM : 151 echantillons (ALGTH 604), burst a 3 de marge (tpu_window.c L1_NB_MARGIN_Q), trame remplie a la longueur exacte.

Premier symptome cote firmware : `EMPTY`, `BURST ID 2!=1`, `3!=2`, `2!=0` avec un decalage qui derive. Cause : dans le tick QEMU, le TICK partait au DSP et l'IRQ trame a l'ARM en meme temps ; le DSP (un autre processus) ecrivait la page R du burst N pendant que l1_sync(N) lisait encore le burst N-2, avec un retard variable selon la charge de l'ARM. Ordre silicium mesure par sonde (PONT_NB_DEBUG, [pgA]/[pgG]/[pgB]) : la ROM copie d_task_d/d_burst_d dans la page R et arme la fenetre dans son ISR de trame, AVANT que l'ARM ne change d_dsp_page ; le resultat est ecrit apres le burst, dans la meme trame. Correctif (qosmo calypso_trx.c + calypso_inth.c, c54x_exe pont.c) : TICK en deux phases. QEMU envoie TICK(N) (bit 17 CALYPSO_PONT_TICK_DEUX_PHASES), le DSP joue l'ISR jusqu'a l'armement de la fenetre et repond DONE|PHASE_A ; QEMU leve alors l'IRQ trame, attend la fin de l1_sync(N) (ecriture IRQ_CTRL bit 0 par irq() apres le handler, comptee par l'INTH quand IRQ_NUM valait 4), puis envoie PONT_GO ; le DSP livre le burst et finit la trame. Cible EOI cumulative, resynchronisee sur timeout (256 x 290 us). Mesure : 0 EMPTY, 0 BURST ID sur des dizaines de blocs, aucun timeout. CALYPSO_PONT_ARM_FIRST=0 revient a l'ancien ordre.

Ce qui reste, et ce qui a ete mesure sur la demodulation NB de la ROM :

- Les bits demodules sont dans data[0x2be2..+148] (sonde [scan] : signe positif = 1, 146-147/148 sur un bon burst, le dernier bit toujours faux : la ROM lit 150 echantillons). Par bloc, 1 a 3 bursts sortent parfaits, les autres a ~45 % d'erreurs avec un motif CONSTANT sur la sequence d'apprentissage ; TOA=3 SNR>0 quand c'est bon, TOA=5 SNR=0 quand c'est mauvais. Deterministe : meme burst, meme resultat d'un run a l'autre.
- Le meme burst repete aux 4 positions donne 4 resultats differents : ce n'est pas le contenu seul, l'etat interne de la ROM entre en jeu.
- Sans effet sur le partage bon/mauvais : amplitude (30000 -> 6000), moyenne nulle forcee, phase porteuse (0/22.5/45), TSC 0..7 dans le burst (2 = BCC attendu), marge 0..7 (seule 3 donne quelque chose), fenetre exacte.
- Instant d'echantillonnage : SEUL 0.5 symbole marche (balayage 2.3 a 4.7 par pas de 0.1 : 3.4 et 3.6 echouent sur tous les bursts) ; MSK pur (diagonales exactes) echoue partout. Un egaliseur se degraderait en pente douce ; ici c'est un fil du rasoir.
- SNR rapporte par la ROM sur un burst PARFAIT : 91 a 600 (la SB rend 16384). L'estimateur voit un gros residu meme quand les decisions sont justes.

- Le tampon DARAM (AAD 0x0cce, 302 mots) est identique aux echantillons livres juste apres le DMA ; la ROM y recrit ensuite les mots 1..29.
- La capture reelle (IQ=reelle, BSIC 48) : TOA stable 1-2, SNR 500-660, ~200 erreurs sur 456 a chaque bloc, le motif 0x9999 dans a_cd aussi.
- Coeur C54x : histogramme d'opcodes de la phase B (PONT_NB_HIST) : 12-14 k instructions par burst, 39 k sur le 4e (decodage). La ROM bascule OVM 20 fois par burst, SAT 214 fois, NEG 720, SFTA 193, NORM 217. Le coeur n'implementait ni OVA/OVB ni la saturation OVM (le mot n'apparaissait qu'en commentaire) : ajoute en post-instruction (calypso_c54x.c, CALYPSO_C54X_OVM=0 pour revenir), plus RND src/dst, MIN/MAX (C=1 si egaux), SFTA (C = bit 32-SHIFT), SUBB (retenue inversee), retenue de ADD/SUB src,SHIFT,dst, OV efface une fois teste. Banc ISA : 139 -> 147 ok. Aucun effet sur le partage bon/mauvais des bursts.

Le decodage SB en rejeu (56 CRC OK / ~300 SCH) a le meme profil : une demodulation qui reussit sur une fraction des bursts selon le contenu. Cause commune probable, dans le coeur emule ou dans la forme du signal 1 ech/symbole que la ROM attend de la chaine analogique ; a chercher sur le chemin SB, mieux instrumente (rejouer.c), plutot que sur le NB.

Rejouer : `IQ=cell PONT_NB_DEBUG=1 ./run.sh` puis `grep -a '\[scan\]\|\[nb\]\|\[pg' /tmp/c54x-pont/dsp.log` ; balayages CELLULE_NB_DEC=auto|x, CELLULE_NB_PHASE, CELLULE_TSC=auto, PONT_NB_MARGE=auto, CELLULE_NB_FINE=1, CELLULE_NB_REPEAT=k, CELLULE_NB_AMP, CELLULE_NB_ZERO_DC, CELLULE_NB_MSK ; PONT_NB_HIST=

.

2.2.1.22 Phase porteuse = temps, et le TOA 24 de la SB [2026-09-21, suite]

- Kill-switch OVM (CALYPSO_C54X_OVM=0) : partage bon/mauvais identique, drapeaux OVA/OVB/C/TC tous a zero a l'entree de chaque burst (sonde [zones]). L'ajout OVM n'est ni la cause ni un remede. Pas de fuite de drapeaux entre bursts.
- Pas d'inversion I/Q demandee par le firmware en reception (trf6151_iq_swapped rend 0 hors TX 850) : d_task_d = 24, sans le bit 0x8000.
- Balayage de la phase porteuse par quadrants (CELLULE_NB_PHASE=quad) : 0 deg = le partage habituel ; 90 et 270 = tout mauvais ; 180 = le burst bon ressort INVERSE (143/148). Marge 2 + 270 deg reproduit exactement marge 3 + 0 deg, et marge 2 + 90 deg rend le bloc SI2 parfait mais inverse. Pour la ROM un echantillon de decalage vaut 90 deg (rotation j^n du MSK) : sa demodulation est COHERENTE sur une reference de phase fixe, l'estimation de canal sur le TSC ne resout ni le quadrant ni le signe.
- Pourquoi la SB converge a 24 et non 23 : notre burst S est place a 21 echantillons dans la fenetre, un de trop pour la geometrie que la ROM attend. IQ=cell:42:0.5:20 avec CELLULE_SB_PHASE=270 : TOA=23, qbits=0, 8/8 SB. Le NB n'en profite pas (marge 2 + 270 = marge 3 + 0).
- Traces d'execution (PONT_NB_HIST : trace_.txt, watch_.txt) : deux bursts 0 (bon 359, mauvais 410) suivent le MEME chemin jusqu'au pas 1698, une boucle argmax en 0x8551-0x8557 (MAX B, XC 1,NC, valeurs A ~0x3dc685d contre 0x3a0e498 : un profil PLAT a 6 % pres, pas un pic de correlation) ; c'est la que la position est choisie et qu'elle part a 5 au lieu de 3. Les « taps » ecrits en 0x2cd1.. (7 groupes) sont ensuite decales d'un mot dans le groupe pour les mauvais bursts. La reference 0x2b28+48 est lue depuis le tampon de burst a AR2 = 0x0d9f.. (mot 209, echantillon 104, un echantillon sur deux) — a comprendre en desassemblant 0x7ef4-0x8557 (PROM0).

Prochaine etape : desassembler la routine NB de la ROM autour de 0x8551 (argmax) et 0x81cd (lecture du tampon a un echantillon sur deux) pour savoir quel profil elle attend a cet endroit ; les traces de 8 bursts sont dans le repertoire donne a PONT_NB_HIST.

2.2.1.23 Le BCCH ne decode pas : le quantifieur des soft bits sort +1 partout [2026-09-21, soir]

Etat : les 4 bursts d'un bloc BCCH sont demodules sans erreur (147/148 en 0x2be2, le dernier bit hors fenetre) une fois le burst elargi (CELLULE_NB_SYM=0.3, cf. supra). Le bloc reste faux (Fire KO, a_cd = bruit). La chaine apres l'egaliseur, lue dans les traces (PONT_NB_HIST) :

```
0x75c9-0x75e2  division SUBC : ratio = (count << 15) / somme, count et
               somme = statistiques du residu sur le TSC (0x7638-0x7697),
               ex. 6 / 1523 -> 0x81 (129)
0x8154-0x815e  echelle T = ((ratio << 10) >> 16) * 0x4eb8 << 2 >> 16 = 2
               (ou 0 quand count = 0)
```

```
0x8166-0x8177  soft_scaled = (rnd(T * soft) << 4 >> 1) >> 16, soft = +-3000..6800
               -> 0..3, signe perdu
0x82bd-0x82cd  table de 129 mots lue en PROM (reada 0x7b9e..) en 0x2a8e
0x82d0-0x82dd  index = soft_scaled >> 8, dest = table[0x2ace + index]
               -> index 0 partout -> +1 partout
0x9a07/0x9a0a  stockage 4 bits par soft bit, 29 mots par burst en
               0x4200 + 29 x burst : mesure 0x1111 sur tous les mots
0x9a6a-0x9a76  desentrelacement par table + LUT (0x2c08) vers 0x2a00
0x9a78-0x9aad  treillis 16 etats (dadst/dsadt/cmps), st TRN
0x9ab8-0x9ad1  traceback (bitt/roltc), puis compaction 0x9ac5.., a_cd
```

Le treillis recoit donc des metriques (c0+c1, c0-c1) sans information et sort un chemin d'egalites, juste sur les suites de zeros et faux ailleurs (36 a 78 des 228 bits) : ce n'est PAS un defaut du desentrelaceur (test A du dossier tools/cch_ref impossible tant que le bloc ne porte pas de signe : aucun des trois candidats ne matche, 65-70 %).

Corriges au passage (isa_test 148 ok) : MPYR / MACR / MASR effacent les 16 bits bas apres l'arrondi (isa_test 132). Sans effet sur le bloc.

Bequille de diagnostic CALYPSO_HACK_SOFT_SCALE=k (c54x_exec.c, MPYR aux trois sites de l'echelle) : x256 fait apparaitre quelques -4 dans le bloc et change a_cd, sans decoder. Le facteur manquant est de l'ordre de 2^12 a 2^20 d'apres l'arithmetique ci-dessus, ce qui n'est pas plausible pour une seule instruction : une des semantiques de la chaine (sfta/sth ASM/norm/exp sur ces valeurs, ou le residu 0x7638 qui fixe count et somme) est lue de travers par le coeur ou par moi. Prochaine etape : rejouer cette chaine (entree 0x2be2 -> sortie 0x2a00) sur les valeurs des traces avec les semantiques du manuel, instruction par instruction, jusqu'au premier ecart.

2.2.1.24 Le BCCH decode : SI1-4, lai=001-01-1, le mobile campe [2026-09-21, apres-midi]

Le quantifieur n'etait pas en cause : quatre semantiques du coeur, toutes en aval de l'egaliseur, lues de travers. Trouvees en rejouant chaque etage en Python sur les traces (PONT_NB_HIST) jusqu'au premier ecart :

1. `add Xmem,Ymem,B` (0xA1xx) etait execute comme SQDST (qui est 0xE2xx) : A <- Ymem«16, B += (AH-Xmem)^2. Site 0x8251 (fin du filtre 3 coefficients de l'egaliseur, 206 fois par burst) : la somme construite dans A etait remplacee par l'echantillon Q derotate. D'ou les "soft bits" propres mais inverses et decales d'un symbole (out[n] = -bit[n+2] au lieu de +bit[n+3]), la correlation TSC du residu nulle (count 6), l'echelle T=2 et les +1 partout. Aussi 0x8565, 0x81fa, 0x7f03, 0x7fab-0x80ec. (c54x_exec.c, bloc `hi8 == 0xA1` desactive.)
2. `*+ARx(lk)%` (mode 14) utilisait encore la grille "base = AR - AR % BK" : au 4e passage du desentrelaceur (0x9a15/0x9a34/0x9a59, BK=456, `mar *+AR4(57)%`), AR4 = 0x2aab+57 donnait 0x291c au lieu de 0x2ae4 et la moitie impaire du bloc partait sous le tampon. (c54x_decode.c -> c54x_circ_ref, comme les modes 8-11.)
3. `sfta A,1` posait C = bit 31 (regle SFTL) ; la LFSR du code de Fire (0xa168-0xa175, registre 40 bits dans A, generateur 0x04820009 via `xc C -> xor #0x0482,16,A ; xor @60(=9),A`) veut le bit 39 : syndrome nul sur un bloc juste seulement avec C = src(40-SHIFT). Le manuel dit src(39-SHIFT) et son exemple (80AA001234«5 -> C=1) ne colle a aucune des deux ; isa_test 188 echoue desormais, la ROM a raison. Avant : FIRE1 (a_cd[0]=0x8040) sur chaque bloc, "Dropping frame with N bit errors".
4. `and/or/xor src,SHIFT,dst` avec src=B (0xF2xx/0xF3xx) : operandes inverses (dst = src OP (dst«SHIFT)) ; corrige en dst OP (src«SHIFT) (audit qosmo-dsp vs l1-dsp). Sites 0x8ec7-0x8eca (repliement du CRC).

Verifications intermediaires (scratchpad repro.py / vit.py) : egaliseur Python avec les coefficients de la ROM = +bit[n+3] 142/142 et = la sortie ROM apres (1) ; nibbles 0x4200 = 456/456 via tools/cch_ref ; bloc 0x2a00 = 456/456 apres (2) ; mots TRN du treillis identiques au modele (228/228), traceback -> 0 erreur sur u224 ; mots 0x2c3c.. exacts.

Attention, la reference u228 de cellule_u228_attendu() etait fausse : les octets L2 se deplient LSB en premier (osmo_pbit2ubit_ext lsb_mode=1) avant le Fire et le convolutif ; en MSB d'abord la sonde [u228] annoncait "78 faux" sur un bloc juste. Corrige ; la sonde [u228]/[bloc] reste a revoir (elle lit 0x2d66, qui n'est pas la sortie).

Resultat (IQ=cell CELLULE_NB_SYM=0.3) : a_cd = 8000 06f9 0026 : 0631 001c 10f1 0100 4040 00e5 2b00 … = SI4 LAI 001-01-1 ; mobile.log : New SYSTEM INFORMATION 1/2/3/4, lai=001-01-1, 0 "Dropping frame", "We are camping normally". Reste : a_cd[2] (0x26/0x35 "erreurs de bits") non nul sur un bloc parfait, sans effet (fire_crc=0) ; isa_test 147 ok / 61 FAIL.

2.2.1.25 Pas de LU ACCEPT : en montage DSP, le lien montant n'existait pas [2026-09-21, soir]

Symptome : le mobile campe (SI 1-4, `lai=001-01-1`, `CGI=001-01-1-6001`, cote pont `DL bursts=2268 blocs=567 crc=0`), demande sa mise a jour de position, puis tourne en rond :

```
mobile.log  : CHANNEL REQUEST: 00 (Location Update with NECI)
              RANDOM ACCESS (requests left 8..4), T3211 qui refire
osmocon.log : L1CTL_RACH_REQ (ra=0x01, offset=9, combined=1, uic=0xff) x5
pont.log    : UL bursts=0 tard=0 rach=0            <- rien ne remonte
dsp.log     : aucune occurrence de RACH ni de UL
/dev/shm/   : calypso_api_ram seul, pas de calypso_rach
```

Sans RACH, pas d'IMM ASS, donc pas de SDCCH, donc jamais de LU ACCEPT. La descente n'etait pas en cause.

Cause, en trois ruptures sur le meme chemin :

1. `calypso_trx.c:325` voit bien l'ecriture de `d_rach` et appelle `calypso_l1_do_rach_written()`, qui relaie a `l1->rach_written` (`calypso_l1_dispatch.c:102`). Mais sous `CALYPSO_DSP_EXTERN=1`, `calypso_l1_do_init()` appelle `calypso_l1_disable()` (qemu.log : « couche 1 « grgsm » desactivee (DSP externe) ») : `l1 == NULL`, le hook est un no-op. Idem pour `_page_written` (d_task_u).
2. Cote `c54x_exe`, le BSP contient tout le necessaire — `calypso_bsp_tx_rach_burst()` (calypso_bsp.c:2487), `send_rach_ra()` (:2541), `send_ul()` (:2344), `tx_burst()` (:2397) — mais **personne ne les appelle** : code mort. Dans les arbres precedents les appels etaient cote QEMU (`qosmo-dsp/hw/arm/calypso/calypso_trx.c:1200` sur ecriture de d_rach, `:1945-1959` poll de d_task_ra/d_task_u par trame) ; le refactor « couche 1 enregistree » les a perdus et ils n'ont pas ete reportes dans le processus DSP.
3. `pont.py` n'a qu'une entree montante : les side-bands `/dev/shm` (`pont/uplink.py:13,134`), ecrits uniquement par la couche 1 gr-gsm (`qosmo-grgsm/.../calypso_l1_grgsm.c:738`) — justement celle qui est desactivee. `--dsp-port 6702` est unidirectionnel (`pont/trx.py:83-89`).

Correctif : `src/montant.c`, appele une fois par trame depuis le PONT_TICK de `src/pont.c`. Il scrute l'API RAM partagee et alimente les memes side-bands qu'en montage grgsm (RACH, SDCCH UL, FACCH, SACCH, parole), avec les captures reprises telles quelles de la couche 1 gr-gsm (fenetre a_cu + 6 et son heuristique d'en-tete LAPDm, `take_ul` sur B_BLUD, anneau TCH). La page W est choisie sur le `d_dsp_page` frais du NDB (dsp_end_scenario ecrit `B_GSM_TASK | w_page` avant de basculer), pas sur celui echantillonne au TICK : `l1_sync()` tourne entre le TICK et le GO en mode deux phases.

Seul point ou la scrutation n'est pas equivalente au callback : le RACH. Le firmware ecrit `d_rach` (prim_rach.c:72) puis `d_task_ra`, et rien ne les efface (sync.c:307 ne le fait que sur ABORT). On declenche donc sur front de `d_rach`, avec une garde de 4 trames. Angle mort : deux tentatives de suite avec la meme RA (tiree au hasard par gsm48_rr, ~1/256) ; `MONTANT_CONSOMME_RACH=1` remet `d_rach` a zero apres publication et rend le declenchement exact. `MONTANT=0` coupe tout, `MONTANT_DEBUG=N` regle le nombre d'evenements imprimes (20 par defaut).

Pourquoi les side-bands et pas TRXD : `calypso_bsp_send_ul()` emet vers 127.0.0.1:5702, c'est-a-dire la socket **descendante** de pont.py, dont `run_data()` fait `self.bts_data = addr` sur tout paquet recu — le burst montant serait relu comme une descente et l'adresse de la BTS ecrasee. Une voie TRXD native demanderait d'abord un port montant dedie cote pont.

A cote, meme session : une SB annoncant `BSIC=7` alors que le BSC est a 21 signifie que le QEMU lance n'a **pas** `CALYPSO_DSP_EXTERN=1` (il ecoute alors sur udp/4730-4731). La SB est alors fabriquee par le shunt gr-gsm a partir du paquet `SCH2` de `pont/downlink.py:30`, avec `cfg.bsic` = 7 par defaut (`pont/config.py:45`) : `PONT_BSIC=21`, ou le montage DSP.

2.2.1.26 Le RACH passe, l'IMM ASS revient, le mobile la jette : la reference de requete [2026-09-21, nuit]

Avec `src/montant.c` en place, la boucle complete se mesure sur le banc reel (sonde a 3 ms sur `/dev/shm/calypso_api_ram` et `/dev/shm/calypso_rach`) :

```
22:07:26 RACH publie seq=2 ra=0x0d bsic=21
22:07:29 >>> IMM ASS ra=0x0d  ref T1'=3 T2=9 T3=35
         brut = 2d 06 3f 03 41 a2 02 0d 1c 69 00 00 2b...
22:07:30 >>> IMM ASS ra=0x04  ref T1'=3 T2=17 T3=23
22:07:32 >>> IMM ASS ra=0x06  ref T1'=3 T2=2 T3=14
```

Donc : le RACH part, la BTS l'entend, le BSC ouvre un SDCCH, l'IMMEDIATE ASSIGNMENT redescend sur l'AGCH, le DSP la decode et le bloc arrive dans `a_cd` avec la BONNE RA. Et pourtant le mobile reste en `connection pending`, `/dev/shm/calypso_sdcch_ul` n'est jamais cree (aucune tache `d_task_u`), et le BSC compte douze `lchan allocation failed ... WAIT_RLL_RTP_ESTABLISH Timeout` en quatre minutes.

Cause : `gsm48_match_ra()` (osmocom-bb `gsm48_rr.c:3359`) n'accepte une assignation que si la RA **et** T1'/T2/T3 correspondent a ce que sa propre couche 1 lui a confirme, et journalise sinon « request %02x matches but not frame number ». Or le banc n'emet pas l'access-burst a la trame ou le firmware a cru l'emettre : `pont.py` le programme sur SON horloge (`pont/uplink.py:_poll_rach` -> `_next_fn(4, ...)`), plusieurs trames plus tard, et la BTS horodate la reference avec cette trame-la.

Ce n'est pas une decouverte : la couche 1 gr-gsm contient deja le contournement (`qosmo-grgsm/.../calypso_l1_grgsm.c:836-845` `feed_agch()` reecrit les octets 8-9 de tout IMM ASS avec le `last_rach` du firmware, lu par symbole ELF). Sous `CALYPSO_DSP_EXTERN=1` cette couche 1 est desactivee, donc plus personne ne le faisait.

Correctif, cote QEMU cette fois (`hw/arm/calypso/calypso_trx.c`) :

- `calypso_l1_dispatch.c` retient le chemin de l'ELF passe a `calypso_l1_do_init()` et expose `calypso_firmware_symbol()` (table des symboles ELF32, reprise de la couche 1 gr-gsm). `last_rach` est GLOBAL a 0x00837624 dans `layer1.highram.elf`.
- `api_write` retient la RA a chaque ecriture de `d_rach`, et `pont_rach_suivi()` releve `last_rach.fn` une fois par trame (sur l'ecriture de `d_dsp_page`, que `dsp_end_scenario()` fait exactement une fois par trame) pour tenir un historique **par RA**. C'est necessaire : les assignations reviennent dans le desordre (mesure ci-dessus : 0x0d, puis 0x04, puis 0x06 alors que la derniere RA ecrite etait 0x0e).
- `api_read` intercepte la lecture ARM du seul mot concerne, `API_NDB + NDB_A_CD + 14` (octets 8-9 du bloc L2, la reference de requete), quand le bloc est bien `06 3f`, et rend la reference recalculee depuis la trame memorisee pour CETTE RA. Si aucune trame n'est connue pour elle, on ne corrige pas : fabriquer une correspondance serait pire que l'echec. `MONTANT_REQREF=0` coupe la correction.

Au passage, deux corrections sur le montant lui-meme :

- Le declencheur du RACH etait la valeur de `d_rach`. Faux : ce mot du NDB est aussi de la memoire du C54x (`data[0x0A3A]`) et la ROM y laisse du residu. Echantillonnage a 4 ms pendant 90 s : `d_rach = 0xfe00` avec `d_task_ra = 0` sur les deux pages W dans 3219 relevés, contre trois vraies tentatives (0x0d54, 0x0954, 0x0c54) portant toutes `d_task_ra = 0x000a`. Or `RACH_DSP_TASK = 10` (firmware `include/calypso/l1_environment.h:49`). Le declencheur est donc `d_task_ra`, et un access-burst bidon (ra=0xfe, bsic=0) partait vers la BTS a chaque demarrage. `MONTANT_RACH_SUR_DRACH=1` retablit l'ancien comportement.
- `calypso_bsp.c` imprimait une ligne `[ts0] tick=... NB fenetre=151` par trame livree : la condition `nwin > 0` est vraie pour tout burst normal depuis la DMA one-shot. Repliee derriere `CALYPSO_BSP_TS0_DEBUG=1`.

2.2.1.27 Le mobile passe en mode dedie, sur un intervalle que le DSP ne recoit pas [2026-09-21, nuit, suite]

Avec la reference de requete corrigee, la suite se deroule : le mobile accepte l'assignation, passe en mode dedie et emet sa demande. Le bloc montant capture dans `/dev/shm/calypso_sdcch_ul` est exactement celui qu'on cherchait :

```
01 3f 49 | 05 08 70 00 f1 10 ff fe 30 08 09 10 10 00 10 00 00 10
|  |  └── L = 18
|  └───── SABM, P=1
└──────── SAPI 0
          05 08 = MM / LOCATION UPDATING REQUEST, LAI 001-01
          LAC=0xfffe (efface), classmark 30, IMSI 001010001000001
```

Deux choses l'empechaient d'arriver a la BTS, toutes deux du meme genre que le reste : un point de branchement que `calypso_l1_disable()` avait neutralise.

1. Le canal dedie n'etait annonce a personne. `pont.py` ne lit pas les IMM ASS : sa classe `Dedicated` (`pont/state.py`) attend le canal dans `/dev/shm/calypso_dcch_cfg`, et tant qu'il manque, `uplink.py:_poll_sdcch()` jette tout le montant. Ce fichier etait ecrit par le tap L1CTL de la couche 1 gr-gsm (`l1-grgsm/calypso_l1ctl_tap.c`), branche par la vtable. Nouveau `hw/arm/calypso/calypso_dcch_tap.c`, qui ne depend d'aucune couche 1 : il renifle le flux sercomm du firmware, retient le `chan_nr` des `L1CTL_DATA_CONF`/`DATA_IND`

et publie. `calypso_l1_do_uart_tx_byte()` l'appelle quand aucune L1 n'est enregistree ; `d_dsp_page = 0` libere. Verifie sur le banc : `[dcch] canal dedie arme : chan_nr=0x51 SDCCH/8 SS=2 TN=1`.

2. Le DSP ne recevait que TS0. Le BSC alloue le SDCCH/8 sur **TS1** ; sous `CALYPSO_BSP_STREAM=1`, `calypso_bsp.c` arretait TS1..TS7 a la reception et `bsp_ts0_livrer()` completait la trame avec du bourrage a zero. Le mobile n'entendait donc ni le UA ni le LU ACCEPT, d'ou le `[dcch] canal dedie libere` immediat et le retour en `C1 normal cell selection`. `g_bsp_tpu_offset`, la position de la fenetre RX relayee par QEMU, etait stockee et jamais lue - et de toute facon TPU_OFFSET est le decalage de synchro global, pas l'intervalle.

Correctif : QEMU sait desormais quel canal le mobile utilise (point 1), donc il l'annonce au DSP par un nouveau message du pont, `PONT_DCCH` (a = TN, b = genre, c = sous-voie), emis juste avant un TICK - jamais pendant l'attente d'un `PONT_GO`, que le DSP ignorerait. Le BSP garde alors les bursts de CET intervalle par numero de trame BTS (`bsp_dedie_stocker`, anneau de 2^16 trames parce que le DSP en pas-a-pas derive de plusieurs secondes) et `bsp_ts0_livrer()` les joue **a la place** de TS0. Un seul burst par tick, donc le cadencement en 1250 symboles par trame n'est pas touche.

Au passage, l'appariement RA -> trame. Premiere version : relever `last_rach.fn` une fois par trame et l'attribuer a la derniere RA ecrite. Mesure : trois `IMM ASS ra=0x0c : aucune trame memorisee` de suite. En rafale, le firmware ecrit le `d_rach` suivant avant que `last_rach` n'ait bouge pour le precedent. Deuxieme version, exacte : les RA sont mises en file a l'ecriture de `d_rach` et chaque `L1CTL_RACH_CONF` (lu par le meme tap sercomm) en depile une - c'est l'appariement que fait le mobile lui-meme dans `cr_hist`.

Et une mesure qui commande le reste. L'ecart entre la reference de la BTS et celle du mobile n'est pas constant, il grandit :

```
ra=0x08  BTS 7/14/45 = fn 9582   mobile fn 8074   1508 trames  ~7,0 s
ra=0x0e  BTS  8/5/33 = fn 10743  mobile fn 9464   1279 trames  ~5,9 s
ra=0x0e  BTS  8/0/14 = fn 11336  mobile fn 9464   1872 trames  ~8,6 s
```

C'est la derive du pas-a-pas : le C54x emule coute ~6,7 ms par trame contre 4,615 ms de temps reel, donc l'horloge du mobile prend du retard sur celle du reseau en continu. Toute comparaison de numero de trame entre les deux cotes doit donc passer par la valeur du mobile, jamais par celle du reseau.

2.2.1.28 Le LU va jusqu'a l'authentification, et meurt d'un SABM de trop [2026-09-21, nuit, fin]

Avec l'intervalle dedie livre au DSP, la procedure se deroule enfin :

```
IMMEDIATE ASSIGNMENT: (ta 0/0m ra 0x0b chan_nr 0x41 ARFCN 514 TS 1 SS 0 TSC 5)
request 0b matches (fn=4,6,23)            <- la reference de requete colle
new state connection pending -> dedicated
New SYSTEM INFORMATION 6 / 5 (SACCH descendante decodee)
RR_EST_CNF -> location updating initiated
MT_MM_ID_REQ  -> IDENTITY RESPONSE
MT_MM_AUTH_REQ -> AUTHENTICATION RESPONSE
MT_MM_LOC_UPD_REJECT                      <- et la, non
```

Le rejet n'est pas une affaire d'authentification (la Ki du `test-sim` et celle de `auc_2g` sont la meme, comp128v1) : c'est une consequence. Le BSC dit pourquoi, a la seconde pres :

```
22:30:54 lchan(0-0-1-SDCCH8-0){ESTABLISHED}: ERROR INDICATION
         cause=SABM frame with information not allowed in this state
```

Un deuxieme SABM sur un lien deja etabli. La LAPDm du BTS casse le canal, le MSC se retrouve en `MSC_A_ST_RELEASING` et repond LOCATION UPDATING REJECT au milieu de la procedure (`/var/log/osmocom/osmo-msc.log`, gsm_04_08.c:112).

Coupable : l'anti-doublon repris de la couche 1 gr-gsm, qui republie un bloc identique passe 60 trames (`SDCCH_UL_DEDUP_TICKS`). Le firmware laisse son bloc dans `a_cu` ; nous le republiions, le pont le reemettait. Desormais un bloc n'est publie que si son CONTENU change, et la memoire de l'anti-doublon repart a zero a la liberation du canal (`montant_canal_libere()`, appele sur PONT_DCCH genre 0xFF) – sinon le SABM de la connexion suivante, octet pour octet identique, serait pris pour un doublon et ne partirait

jamais. `MONTANT_SDCCH_REPETE=N` retablit une republication au bout de N trames. Compromis assume : une retransmission LAPDm du mobile porte les memes octets et sera avalee ; l'inverse casse le lien a coup sur.

A surveiller au prochain run : les « Dropping frame with 110 bit errors » de la descente dediee. 110 erreurs sur 456 bits, c'est la signature d'UN burst sur quatre manquant ou faux dans le bloc. Elles sont nombreuses au moment de l'etablissement (le canal n'est arme qu'au premier DATA_CONF/IND, donc les premiers blocs partent sans intervalle dedie), puis rares pendant la transaction (SI5, SI6, ID REQUEST, AUTH REQUEST passent tous), puis permanentes apres le rejet (le BTS n'emet plus rien sur TS1).

2.2.1.29 Le SABM ne partait plus du tout : la liberation lue au mauvais endroit [2026-09-21, nuit, suite]

Apres avoir mis « un seul SABM par connexion », le BSC ne dit plus `ERROR INDICATION` mais `WAIT_RLL_RTP_ESTABLISH: Timeout` : plus de doublon, et plus de SABM du tout. La trace QEMU montre le defaut en deux lignes :

```
[dcch] canal dedie arme   : chan_nr=0x41 SDCCH/8 SS=0 TN=1
[dcch] canal dedie libere : chan_nr=0x00 SDCCH/4 SS=0 TN=0
```

Armé puis libéré dans la seconde. Le tap publiait « libere » sur `d_dsp_page == 0`, condition reprise du `l1_reset()` de la couche 1 gr-gsm. Faux ici : le firmware fait justement un `l1s_dsp_abort()` (sync.c:308) au moment ou il bascule VERS le canal dedie – c'est le « resetting scheduler » du journal du mobile, juste apres l'IMMEDIATE ASSIGNMENT. `pont.py` voyait donc un canal libere et jetait le SABM.

Tant que le SABM etait republie toutes les secondes, le defaut etait masque : une republication finissait toujours par tomber dans une fenetre ou le canal etait arme. C'est ce qui explique que la session de 22:30 ait pu aller jusqu'a l'AUTHENTICATION RESPONSE malgre les doublons.

Deux corrections :

- **La liberation se lit sur le retour aux voies communes**, pas sur `d_dsp_page`. Un `L1CTL_DATA_IND/DATA_CONF` portant un `chan_nr` non dedie (BCCH, CCCH) veut dire que le mobile est revenu sur les voies communes ; en mode dedie il n'en lit aucun, donc ca ne peut pas arriver au milieu d'une connexion. `calypso_l1_do_page_written()` ne libere plus rien.
- **Le bloc montant attend son canal** (`pont/uplink.py:_poll_sdcch`). La couche 1 publie le SABM a l'instant ou elle l'emet, et `Dedicated.read()` ne relit `/dev/shm/calypso_dcch_cfg` qu'une fois par `DCCH_TTL` (100 ms) : le premier bloc d'une connexion tombait regulierement dans cette fenetre et disparaissait. On force desormais une relecture des qu'un bloc arrive, et on garde le bloc jusqu'a une seconde si le canal n'est pas encore connu, au lieu de le jeter (`SDCCH_ATTENTE`).

2.2.1.30 Le canal dedie battait : arme/libere des dizaines de fois par seconde [2026-09-21, nuit, suite]

Deuxieme version de la liberation (« un bloc sur une voie commune veut dire que le mobile est revenu ») : pire que la premiere. Trace QEMU pendant une connexion :

```
[dcch] canal dedie arme   : chan_nr=0x41 SDCCH/8 SS=0 TN=1
[dcch] canal dedie libere : chan_nr=0x00 SDCCH/4 SS=0 TN=0
[dcch] canal dedie arme   : chan_nr=0x41 SDCCH/8 SS=0 TN=1
[dcch] canal dedie libere : chan_nr=0x00 ...        (x N, en boucle)
```

Le BSP basculait donc entre TS0 et TS1 a chaque bloc. Cote mobile : tous les blocs descendants a 110/98/87 erreurs, aucun UA, `MDL-ERROR-IND cause 1` (T200/N200) et liberation – alors que l'IMMEDIATE ASSIGNMENT avait ete acceptee (`request 07 matches (fn=2,13,26)`).

Deux criteres ajoutes, tous deux necessaires pour liberer :

- **Un vrai canal commun.** 44.004 8.3 : 0x80 BCCH, 0x88 RACH, 0x90 PCH/AGCH, donc `(chan_nr & 0xE0) == 0x80`. Ce qui declenchait la liberation portait `chan_nr = 0x00`, qui n'est pas un canal.
- **Un ecart de trames.** Le bloc commun doit etre au moins DCCH_LIBERE_APRES_TRAMES (100) trames apres le dernier bloc du canal dedie, d'apres le numero de trame que porte l'en-tete L1CTL. Un bloc CCCH en retard, delivre juste apres la bascule, porte un numero proche et ne libere donc rien.

Rappel de l'historique de ce seul point, parce qu'il resume la difficulte : la liberation a d'abord ete lue sur `d_dsp_page == 0` (tire a l'entree en mode dedie, pas a la sortie), puis sur le premier bloc commun venu (tire en boucle pendant la connexion). Le bon signal est le retour durable sur les voies communes.

2.2.1.31 a_cu porte un drapeau : il n'y avait rien a deviner [2026-09-21, nuit, fin de l'histoire]

Trois versions successives de l'anti-doublon SDCCH montant, trois echecs :

1. Fenetre heuristique + republication apres 60 trames (repris de la couche 1 gr-gsm) : le meme SABM repartait sur un lien etabli, le BTS repondait « SABM frame with information not allowed in this state » et cassait le canal en pleine procedure.
2. Comparaison du contenu sur 23 octets : la friture apres la charge utile bouge d'une lecture a l'autre, 32 blocs « neufs » publies pour un seul SABM.
3. Verrou « un seul SABM par connexion » : plus aucun SABM des que le verrou restait arme, et il ne retombait que sur un RACH publie ou une liberation annoncee – deux evenements qui peuvent ne jamais venir. Un verrou qui coince le banc definitivement.

Le firmware annonce pourtant chaque bloc, explicitement (`layer1/prim_tx_nb.c:80-101`) :

```
uint16_t *info_ptr = dsp_api.ndb->a_cu;
info_ptr[0] = (1 << B_BLUD);                    /* bloc present */
info_ptr[1] = 0; info_ptr[2] = 0;
dsp_memcpy_to_api(&info_ptr[3], data, 23, 0);  /* les 23 octets L2 */
```

C'est exactement la disposition que `prendre_ul()` lit deja pour le TCH, et le drapeau est a usage unique. Donc : on le teste, on prend les 23 octets du mot 3, on l'efface. Un bloc pose = une publication, sans fenetre, sans comparaison, sans temporisation, sans verrou.

`MONTANT_SDCCH_FENETRE=1` force l'ancienne voie, et elle prend le relais toute seule si B_BLUD ne se leve jamais alors que le firmware pose des taches montantes (cas ou la ROM consommerait le drapeau avant la scrutation) : 400 taches sans drapeau, un message, et bascule.

Lecon : chercher le signal que le firmware pose deja, avant d'inventer une heuristique pour le reconstituer.

2.2.1.32 L'horloge du banc etait celle du mur, pas celle du DSP [2026-09-22]

Le run de 10:01 campait, lisait SI1-4 et faisait sa mise a jour de localisation… pendant dix secondes. Apres, plus rien : a 10:04 l'appel partait en `T3126`, a 10:06 le SMS n'obtenait meme plus d'IMMEDIATE ASSIGNMENT, et le canal dedie sortait des « Dropping frame with 110 bit errors » en continu.

La mesure qui tranche, deux compteurs lus au meme instant a 10:09 :

```
pont.py   STATS fn=99674
[ts0]     tick=81089 fn=80410        (offset ARM-tick fige a -679)
```

19 000 trames, **87 secondes** d'ecart, et l'ecart grandissait. Le C54x emule coute ~5,8 ms par trame contre 4,615 ms de temps reel ; `bsp_ts0_service()` joue la trame BTS `tick + g_ts0_offset`, l'offset est fixe une fois pour toutes sur la premiere SB, et rien ne rattrapait le reste. La BTS remplissait l'anneau 1,25 fois plus vite que le DSP ne le vidait : le mobile vivait une minute et demie dans le passe. Tout le reste en decoule –

- `T3126`, `T3101`, `T200` sont des temporisations en secondes de MUR : une reponse qui met 87 s a revenir les a toutes epuisees ;
- le canal dedie encore plus vite : `g_dedie` ne commence a se remplir qu'a l'armement du canal, or le DSP lisait des trames d'AVANT cet instant. Releve dans `/dev/shm/calypso_bsp_dedie` : `stockes=3202 joues=244 manques=150`. 38 % des trames du canal partaient sans burst – un burst sur quatre absent d'un bloc, c'est exactement 110 erreurs sur 456.

Le DSP ne peut pas rattraper, il tourne deja a fond. C'est donc la BTS qui ralentit : `calypso_bsp.c` publie la trame que le BSP reclame (`/dev/shm/calypso_horloge`, 16 octets : seq, cale, fn_bts, tick) et `pont/trx.py` y asservit l'horloge qu'il envoie a osmo-bts-trx en IND CLOCK.

Asservir en FREQUENCE, pas par recalage. Premiere version : repousser `t0` des qu'on devance le DSP de plus de 12 trames. Mesure immediate, `UL bursts=11 tard=232` : `Transmitter.run()` jette tout burst dont la trame s'ecarte de plus de `window_tol` (1 trame) de l'horloge au moment de l'envoi, et une horloge qui avance par a-coups en sort a chaque fois. Plus un SABM n'arrivait a la BTS -> pas de UA -> `MDL-ERROR-IND cause 1`, plus aucune mise a jour. La version qui tient ne change que la VITESSE (`self.dur`, la duree effective d'une

trame) : cadence du DSP mesuree toutes les 250 ms, correction proportionnelle de la phase sur ~400 trames, et `t0` rebase a chaque changement pour que `fn()` reste continue.

Apres (run de 10:20, MSC) :

```
10:20:04  VLR: update ... TMSInew-0x46FB2C10
10:20:06  VLR: update ... TMSI-0x46FB2C10
```

`TMSInew-` devenu `TMSI-`, donc le TMSI REALLOCATION COMPLETE est revenu et le VLR l'a confirme : le LU va au bout, sans `LOCATION UPDATING REJECT`, sans `ERROR INDICATION cause=SABM frame with information not allowed in this state` (le double SABM de 10:01 etait une retransmission T200 du mobile, pas un doublon du pont : le UA mettait plus de 700 ms a revenir). Cote pont, `fn=26609` contre `fn_bts=27045` : verrouille.

`PONT_HORLOGE=0` revient a l'horloge murale, `PONT_HORLOGE_AVANCE` regle l'avance visee (12 trames), `CALYPSO_BSP_HORLOGE=0` coupe la publication.

Reste ouvert : l'etablissement DESCENDANT. Le MSC tente un MT SMS, reste 10 s en `MM_CONN_PENDING` puis `MMSMS-REL-IND` – le paging ou la reponse du mobile ne passe pas. A regarder avec le filtre des pagings vides de `pont/downlink.py` (`is_empty_paging`, qui ne sert qu'au montage gr-gsm) et le groupe de paging que le firmware ecoute.

2.2.1.33 Le magasin du canal dedie commencait trop tard [2026-09-22, suite]

Le paging n'est PAS en cause : le mobile recoit bien le sien et repond (`CHANNEL REQUEST: 80 (PAGING Any channel)`, 10:22:33). Ce qui le tue est la ligne d'apres, `LOS during RACH request` – la couche 3 avait deja declare la perte de couverture. Meme cause que tout le reste : le canal dedie.

Les compteurs de `/dev/shm/calypso_bsp_dedie` a la liberation :

```
tn=-1 ss=0 stockes=4311 joues=404 manques=138 libere
```

138 trames du canal sur 542 jouees **sans burst**, une sur quatre. C'est exactement la signature observee cote mobile :

- `Dropping frame with 110 bit errors` (110 sur 456 = un burst sur quatre) ;
- `MON: f=514 lev=<=-110 snr=0 ... TS=1/0` – rien du tout sur l'intervalle dedie pendant trois secondes ;
- `Received frame for unsupported SAPI 2!` et `MDL-ERROR-IND cause 3`, ce que LAPDm sort d'un bloc reconstitue a partir de trois bursts sur quatre.

Au meme instant, cote TS0 : 11 manques en tout. Ce n'est donc pas la BTS qui est en retard sur le DSP (l'horloge asservie tient), c'est ce magasin-ci qui ne couvre pas assez loin. `g_dedie` n'est alloue et rempli qu'a l'ARMEMENT du canal, or le BSP joue la trame BTS `tick + g_ts0_offset`, en retard sur celle qui arrive : toutes les trames du canal anterieures a l'armement sortaient vides.

Il n'y avait rien a stocker de plus. `g_autres` garde DEJA les sept intervalles de chaque trame, sans condition et des le premier burst (il sert a completer la trame continue) : `g_dedie` en est un doublon partiel. `bsp_dedie_bits()` retombe donc dessus quand son propre anneau n'a pas la trame, et compte ses replis. `MONTANT_DEDIE_STRICT=1` retablit l'ancien comportement pour remesurer l'ecart.

A lire au prochain run, dans `/dev/shm/calypso_bsp_dedie` : `manques` doit tomber pres de zero et `replis` dire combien de trames le repli a rattrapees. S'il reste des manques, c'est que la BTS n'a vraiment rien emis sur ces trames-la, et il faudra le chercher cote osmo-bts-trx.

Lecon, la meme que pour a_cu : la donnee etait deja la, dans un autre magasin du meme fichier. Avant d'en remplir un nouveau, regarder qui garde deja ce qu'on cherche.

2.2.1.33.1 Correction : le compteur `manques` ne voulait pas dire ca [2026-09-22]

Le « une trame du canal sur quatre sans burst » ci-dessus s'appuyait sur `manques=138` de `/dev/shm/calypso_bsp_dedie`. Ce compteur etait faux, dans les deux sens :

- `a_nous` etait calcule par `bsp_dedie_trame(fn)` SANS verifier qu'un canal soit arme. Hors connexion, `g_dedie_ss` vaut 0, donc une trame sur huit etait declaree « du canal », `bsp_dedie_bits()` rendait NULL sur sa garde `g_dedie_tn <= 0`, et `manques` montait – pendant tout le campement, ou jouer TS0 est justement la bonne chose ;
- `manques` (et le nouveau `replis`) n'etaient pas remis a zero a l'armement, contrairement a `stockes` et `joues` : le fichier melangeait toutes les sessions depuis le demarrage.

Ce qui donnait des lectures impossibles, `joues=72 manques=162` – plus de trames manquees que jouees sur un canal ouvert quelques secondes. C'est aussi pourquoi le repli sur `g_autres` affichait `replis=0` : il est place APRES la garde `g_dedie_tn <= 0`, donc jamais atteint dans le cas qui gonflait le compteur.

Corrige : `a_nous` exige `g_dedie_tn > 0`, et les quatre compteurs repartent de zero a chaque armement. Le repli sur `g_autres` reste : il couvre le vrai trou, les trames du canal anterieures a l'armement. La mesure est donc A REFAIRE avant de conclure quoi que ce soit sur le canal dedie.

Lecon : un compteur qu'on n'a pas verifie n'est pas une mesure. Celui-ci a servi de preuve a un diagnostic chiffre, et le chiffre etait du bruit.

2.2.1.33.2 start-direct –dsp : appeler l'amorce qui marche, pas la reecrire

Rapport du banc : « ca marche avec run_real.sh et pas avec start-direct ». Les environnements des trois processus, releves dans `/proc/<pid>/environ`, sont pourtant identiques a deux variables pres, toutes deux sans effet (`CALYPSO_BSP_DIRECT_FEED` n'est plus dans le binaire – `strings` ne donne que `CALYPSO_BSP_DIRECT_BRINT0` – et `PONT_NB_DEBUG` n'est qu'une trace). La difference etait donc dans l'AMORCE – et plus precisement dans ce qui n'etait PAS monte : le coeur reseau, la BTS, le side-car et tmux sont des run_modules du fork, et `--dsp` avait remplace le run.sh du fork par celui de c54x_exe, qui ne les connait pas. Deuxieme essai, delegation a `run_real.sh --secondes 0` : meme trou, plus un « ECHEC : aucun SI lu » imprime apres zero seconde d'observation.

Version qui tient : `--dsp` ne change ni de fork ni de profil (l'hybride reste le defaut). Le plan du fork se joue en entier, avec `--skip qemu,pty,osmocon,l2` – les quatre modules de la chaine Calypso – et `--no-attach` ; le banc DSP prend le relais juste apres avec ses cinq etapes. `MOD_REQUIRED[bts]=0` et sa barriere porte sur la VTY, pas sur le transceiver : osmo-bts-trx peut donc demarrer avant pont.py, comme dans run_real.sh.

Deux details d'usage corriges au passage : `--stop` arrete le banc DSP meme sans `--dsp` sur la ligne de commande (sinon cinq processus survivaient en tenant 5700-5702, et la pile paraissait arretee), et les aides phonesim/RIL sont `disownees` – `setsid` detache la session mais pas le job, et bash annoncait leur mort par un « line NNNN: Killed setsid … » par-dessus le prompt.

2.2.1.33.3 Deux scripts, deux sens pour `MODE` [2026-09-22]

```
[ .. ] Arret de la pile via run.sh[run] ECHEC : MODE=faketrx-qemu inconnu (dsp|grgsm)
```

`start-direct.sh` exporte son propre `MODE` (le profil : `faketrx-qemu`, ligne `export CALYPSO_PROFILE MODE`), et `c54x_exe/run.sh` lit la MEME variable pour choisir `dsp|grgsm`. Le banc heritait donc du profil et sortait avant d'avoir rien arrete – silencieusement, l'appelant ignorant son code de retour. Verifie des deux cotes :

```
$ MODE=faketrx-qemu bash run.sh --status
[run] ECHEC : MODE=faketrx-qemu inconnu (dsp|grgsm)
$ MODE=faketrx-qemu bash -c 'MODE=dsp bash run.sh --status'
  dsp      pid 147597   /tmp/c54x-pont/dsp.log
```

Ce que ca laissait derriere, releve juste apres un `--stop` : `dsp` encore vivant, `qemu/osmocon/mobile/pont` arretes – le teardown du fork connait ces quatre-la (ils sont dans ses patterns) mais pas `c54x_exe`, qui restait seul a tenir `/tmp/calypso_dsp.sock` et `/dev/shm/calypso_api_ram`.

On ne renomme pas le `MODE` du banc (run_real.sh et les habitudes s'en servent) : `start-direct.sh` passe desormais par un `banc_dsp()` qui pose `MODE=dsp` a chaque appel.

2.2.1.33.4 Ou en est le canal dedie [2026-09-22, 11:08]

Etabli, compteurs corriges a l'appui :

- **les bursts sont tous la.** `/dev/shm/calypso_bsp_dedie` : `stockes=4288 joues=416 manques=0 replis=5`. Le repli sur `g_autres` ne rattrape que 5 trames (le retard d'armement) : la theorie du « un burst sur quatre manquant » est morte, c'etait le compteur qui mentait ;
- **TS0 n'est pas affame** : 11 a 12 `[ts0] pas de burst BTS` par run, tous au demarrage. L'horloge asservie tient ;
- **la signature a change** : plus de `unsupported SAPI 2`, plus de `MDL-ERROR cause 3`. Reste 89 blocs a **exactement 96 erreurs sur 456**, la meme valeur a chaque fois. Constant = corruption systematique, pas un trou. Pour comparaison, un bloc BCCH du meme run est a ber 46-50 et passe.

Cote reseau, la consequence se lit maintenant en une ligne : le VLR alloue le TMSI (`TMSInew-0x1BCA53DE`), donc le LOCATION UPDATING ACCEPT est parti, et cinq secondes plus tard `MSC_A_ST_RELEASING: LOCATION UPDATING REJECT` – le TMSI REALLOCATION COMPLETE n'est jamais revenu. Le side-car MS#2, lui, boucle son LU (`TMSInew-` puis `TMSI-`) : le coeur reseau est hors de cause.

Question ouverte : pourquoi 96, toujours 96 ? La geometrie de fenetre est la seule difference plausible entre un bloc SDCCH et un bloc BCCH dans ce chemin de livraison, et elle n'etait pas mesurable – la trace `[ts0]` est plafonnee a 20 lignes puis une sur 5000, donc on n'avait que le demarrage. Elle trace desormais TOUTE trame du canal dedie (300 au plus, sans `CALYPSO_BSP_TS0_DEBUG`) et dit si son burst vient bien de cet intervalle :

```
[ts0] tick=... fn=... p51=0 NB fenetre=151 marge=3 rif_avant=0  <- canal dedie
```

A comparer avec un bloc BCCH (p51 = 2-5) du meme run. Avec `PONT_NB_DEBUG=1` en plus, `[nb]` donne le TOA et l'en-tete `a_cd` (mot de Fire, erreurs) par burst : si le TOA des bursts dedies differe de celui des bursts BCCH, c'est le calage de la fenetre, et les 96 erreurs s'expliquent.

2.2.1.33.5 « Trop tot » etait compte comme « trop tard » [2026-09-22, 11:12]

```
UL bursts=116 tard=18
```

13 % des bursts montants JETES. Un bloc en demande quatre : ~40 % des blocs montants n'arrivaient pas entiers a la BTS. C'est exactement ce qu'on lisait en bout de chaine – TMSI REALLOCATION COMPLETE absent (LU REJECT a 11:06), CP-ACK absent (MT SMS a 11:10, `WAIT_CP_ACK` puis abandon), SABM retransmis par T200.

`Transmitter.schedule()` calcule l'instant d'emission `post` a la MISE EN FILE, d'apres la cadence de l'horloge a ce moment-la. Depuis qu'elle suit le DSP, cette horloge n'avance plus au rythme du mur : quand le DSP marque le pas, elle aussi, et le reveil tombe AVANT que la trame visee ne soit arrivee. `run()` comparait alors `abs(off) > window_tol` et jetait – sans distinguer le burst en avance (qu'il suffit d'attendre) du burst en retard (perdu).

Corrige : en avance, on remet en file avec un `post` recalcule sur l'horloge courante, jusqu'a `PONT_WINDOW_ESSAIS` fois (12) ; seul `off < -window_tol` compte comme un retard. C'est une consequence directe de l'asservissement ([[horloge-banc-suit-dsp]]) : une file d'emission datee en temps mur ne peut pas servir une horloge qui ne l'est plus.

2.2.1.33.6 Deux pistes ecartees, une mesuree

- **geometrie de fenetre** : ecartee. La trace dit la meme chose des deux cotes, `fenetre=151 marge=3` pour une trame du canal dedie comme pour un bloc BCCH.
- **bursts manquants** : ecartee, `manques=0`.
- **chiffrement** : `ENCRYPTION="a5 0"` arrive enfin jusqu'a `/etc/osmocom/osmo-bsc.cfg` (`encryption a5 0`) depuis que l'environnement gagne sur globals.conf, et c'est a partir de la que le LU passe en entier (`TMSInew-0xA7CFD9FF` puis `TMSI-0xA7CFD9FF`, 11:09:45).

Reste, par ordre :

1. `Received frame for unsupported SAPI 5!` + `MDL-ERROR-IND cause 3` en rafale pendant la connexion dediee, avec `MON: lev=<=-110 snr=0 ... TS=1/0`. Des blocs qui passent le code de Fire mais dont l'adresse LAPDm est fausse : ce ne sont pas des blocs abimes, ce sont d'AUTRES blocs. Piste : un bloc SACCH rendu sur la liaison SDCCH. L'octet 0 d'un en-tete L1 SACCH est le niveau de puissance ordonne, et `(0x08 >> 2) & 7 = 2` comme `(0x15 >> 2) & 7 = 5` : les deux SAPI vus, 2 et 5, sont exactement ce que donne un en-tete L1 lu comme une adresse. A verifier sur le tap GSMTAP (udp/4729) en comparant ce que la BTS emet en TS1/p51=0-3 et en TS1/p51=32-35.
2. `FBSB RESP: result=255` en rafale sur la cellule SERVANTE (281 demandes sur l'ARFCN 514 contre 5 sur 614, 228 echecs) -> `MM_EVENT_LOST_COVERAGE` -> « no cell available ». Toute transaction lancee dans cette fenetre meurt sur-le-champ : c'est ce qui tue l'appel de 11:12:17 (`MMCC_EST_REQ` recu en « no cell available », `MMCC_REL_IND` dans la seconde) et ce qui produisait les `LOS during RACH request`.

2.2.1.33.7 Le correctif montant, mesure sur le banc [2026-09-22, 11:16]

Avant / apres, meme banc, meme configuration :

```
UL bursts=116 tard=18    (13,4 % jetes)
UL bursts=185 tard=4     ( 2,1 % jetes)
```

Six fois moins, et on retrouve le niveau d'avant l'asservissement de l'horloge (310/4, soit 1,3 %). Le reste tient a la gigue residuelle du DSP : un burst vraiment en retard reste perdu, c'est le comportement voulu.

Ce n'etait pas toute l'histoire pour autant : le LU de 11:15:46 echoue encore (`TMSInew-0x2BC8C863` puis REJECT), et le canal dedie sort toujours ses blocs a 96 erreurs (87 dans ce run). Le montant n'etait qu'un des deux chemins.

2.2.1.33.8 FBSB mid-campement : le DSP ne detecte rien [2026-09-22, 11:16]

Sonde en lecture seule sur `/dev/shm/calypso_api_ram` (cf. [[sonde-api-ram-vivante]]), 3850 echantillons a 3 ms pendant que le mobile campait et que la couche 3 enchainait ses `FBSB RESP: result=255` :

```
fb_det   fb_mode    TOA     PM       angle    SNR     vu
0        0          0       5424     -1       2539    3850
```

`d_fb_det` reste a 0 sur TOUTE la fenetre : le DSP ne pose aucun resultat de detection FB. Ce n'est donc pas la porte FB0->FB1 de prim_fbsb.c qui rejette, c'est la tache FB qui ne rend rien du tout – alors que le meme DSP demodule sans peine les blocs BCCH de la meme cellule (SI1-4 en continu, ber 46-50). Sur ce run : 19 `L1CTL_FBSB_REQ` sur l'ARFCN 514, 16 `result=255`.

La consequence se lit trois lignes plus loin dans le journal du mobile : `MM_EVENT_LOST_COVERAGE` -> « no cell available », et toute transaction lancee dans cette fenetre meurt sur-le-champ (`MMCC_EST_REQ` recu en « no cell available », `MMCC_REL_IND` dans la seconde, appel de 11:12:17).

Piste a suivre : au demarrage la meme tache FB reussit. La difference est que `g_ts0_offset` vaut alors INT64_MIN et que `bsp_ts0_service()` joue les bursts DANS L'ORDRE D'ARRIVEE, un par tick – un flux continu. Une fois cale, il joue `tick + offset` et saute les ticks dont la trame manque. A verifier : ce que voit la tache FB quand elle est relancee en cours de campement, et si un retour au mode « ordre d'arrivee » pendant une recherche FB la debloque.

2.2.1.33.9 Ce n'est pas un bloc abime, c'est le meme bloc [2026-09-22, 11:18]

Le LU passe desormais en entier – `LOCATION UPDATING ACCEPT (lai=001-01-1)`, `got TMSI 0x0e67ce01`, `TMSI REALLOCATION COMPLETE` emis. C'est APRES, en attente de la liberation, que le canal part en vrille, et la signature est enfin lisible :

```
N(S) sequence error: N(S)=1, V(R)=2     (x25, TOUJOURS la meme paire)
Received frame for unsupported SAPI 6!  (2 le 11:11, 5 le 11:15, 6 ici)
```

Deux faits qui tranchent :

- `N(S)=1` repete alors que le mobile attend `N(S)=2` : LAPDm recoit encore et encore LE MEME I-frame. Un bloc abime ne repasse pas le code de Fire vingt fois de suite avec le meme N(S) ;
- le SAPI illegal CHANGE d'un run a l'autre (2, 5, 6) mais reste CONSTANT dans un run. C'est le contenu fige d'un tampon, pas du bruit.

Ce qui disqualifie l'hypothese « bloc SACCH rendu sur la liaison SDCCH » avancee plus haut, et renvoie a deux choses deja ecrites ici : [[tpu-dsp-frame-irq-oneshot]] (la ROM re-dispatche une page perimee quand elle recoit une IRQ trame qu'elle n'attendait pas) et [[sb-delirant-page-zero]] (l'ARM relit une page R qui ne porte pas de nouveau resultat). Les deux ont ete diagnostiquees en mode FB/SB ; ici c'est le mode DEDIE.

Cote BSP le magasin est hors de cause : `g_ts0`, `g_dedie` et `g_autres` n'apparient que sur `fn` EXACT, aucun ne peut rejouer une trame. Et `manques=0`.

Prochaine mesure : `PONT_NB_DEBUG=1` imprime l'en-tete `a_cd` (mot de Fire, erreurs) et les premiers octets decodes a chaque burst 3. Si le meme `a_cd` ressort trame apres trame pendant la connexion dediee, la boucle est cote ROM/page et non cote radio – et les 96 erreurs ne sont qu'un effet de bord du bloc fige.

2.2.1.33.10 Le chiffrement : personne ne dechiffrait le descendant [2026-09-22, 11:21]

Run avec `ENCRYPTION="a5 1"`. La transaction va plus loin que jamais – SABM/UA, IDENTITY REQUEST/RESPONSE, puis AUTHENTICATION REQUEST/RESPONSE, tout en clair – et la tempete commence a la ligne EXACTE ou le chiffrement s'arme :

```
11:21:02  CIPHERING MODE COMMAND (sc=1, algo=A5/1 cr=1)
11:21:02  CIPHERING MODE COMPLETE (cr 1)
11:21:02  Dropping frame with 96 bit errors     <- et sans interruption ensuite
```

En « a5 0 » la meme transaction va au bout (LU complet du 11:18). La correlation est nette dans les deux sens.

Deux faits qui l'expliquent :

- **il n'y a aucun A5 dans le modele Calypso.** `d_a5mode` n'apparait que dans `hw/arm/calypso/l1-grgsm/calypso_l1_grgs` ; rien dans `l1-dsp/`. En montage DSP, le mobile n'a donc rien pour dechiffrer ;
- **le pont est asymetrique.** `Trx.send_ul()` CHIFFRE le montant (`self.cipher.apply(burst, fn, True)`) : il tient la place de la voie d'emission du DSP. Mais `run_data()` relayait le descendant BRUT vers le DSP. Le `cipher.apply(..., False)` de `Downlink._decode()` ne sert qu'au decodage L2 du pont lui-meme et ne touche pas ce que recoit le DSP.

Corrige : `run_data()` dechiffre le burst avant de l'envoyer au DSP, A5 etant symetrique. **Seulement sur l'intervalle dedie** (`_tn_dedie()`, lu via `Dedicated` deja mis en cache) : la BCCH et la CCCH ne sont jamais chiffrees, les toucher detruirait le campement. La condition est `cipher.dl_active`, que `Downlink._decode()` leve deja quand un bloc ne decode qu'une fois dechiffre – le meme signal que la BTS (osmo-bts l1sap.c check_for_first_ciphrd).

Effet de bord utile : `stats.a5_dl` cesse d'etre a zero, ce qui rend le dechiffrement visible dans la ligne STATS.

A noter pour la suite : dans ce meme run le MSC a lache a 11:21:00, soit DEUX SECONDES AVANT que le mobile ne recoive la commande de chiffrement. La fenetre morte de 11:20:58-11:21:01 (`MON lev=<=-110 snr=0`, `MDL-ERROR cause 12`) reste donc un defaut a part entiere, independant du chiffrement.

2.2.1.33.11 `./start-direct.sh` sans –dsp : QEMU ne demarrait pas [2026-09-22, 11:23]

```
[FAIL] Calypso emulator (QEMU) (started but never ready:
       socket du moniteur QEMU : toujours pas pret apres 30s)
```

Le message designe le moniteur, mais rien n'avait ete lance. Les deux enveloppes `/usr/local/bin/qosmo-dsp` et `/usr/local/bin/qosmo-grgsm` (hors depot) trient ainsi :

```
case "${1:-}" in -k|-kernel|-r|--rundir|-M|-s|-p|--bin|--gdb|--bind|-o)
    exec /usr/local/bin/qosmo-grgsm-launch "$@";; esac
exec env MODE=grgsm PONT=1 /opt/GSM/c54x_exe/run.sh "$@"
```

`--qemu`, `--cpu` et `--monitor` manquent a la liste, et le test ne porte que sur le PREMIER argument. Or `run_modules/40-qemu.sh` appelle

```
qosmo-grgsm --qemu <bin> -k <elf> --bin <bin> --cpu arm946 \
            --gdb N --rundir <dir> --monitor <dir>/qemu-monitor.sock
```

soit `--qemu` en tete. Le motif ne matche pas, l'appel part sur `c54x_exe/run.sh`, qui sort immediatement :

```
$ qosmo-grgsm --qemu ... -k ... --monitor ...
[run] ECHEC : option inconnue : --qemu (voir --help)
```

Le vrai lanceur C, lui, comprend les six (`strings` : `--bin --cpu --gdb --monitor --qemu --rundir`). Corrige dans les deux enveloppes : balayage de TOUS les arguments, et les trois options ajoutees. Sauvegardes dans /tmp/qosmo-{grgsm,dsp}.bak. Elles ne sont dans aucun depot – une reinstallation les ecrasera ; c'est note dans LAUNCH.md.

Defaut PRE-EXISTANT, sans rapport avec le montage DSP : `--dsp` ne passe pas par ce module (il est dans `--skip qemu,pty,osmocon,l2`), ce qui explique qu'on ne l'ait vu qu'en lancant start-direct SANS `--dsp`.

2.2.1.33.12 Le Kc n'etait publie par personne [2026-09-22, 11:29]

Le dechiffrement ajoute dans `pont/trx.py` n'a rien change au run de 11:26, et la raison est en amont :

```
$ ls /dev/shm/calypso_kc_l1
ls: cannot access '/dev/shm/calypso_kc_l1': No such file or directory
$ grep -rn calypso_kc_l1 qosmo/hw/arm/calypso/
l1-grgsm/calypso_l1_grgsm.c:36:#define SHM_KC "/dev/shm/calypso_kc_l1"
```

Un seul ecrivain, et c'est la couche 1 gr-gsm – celle que `calypso_l1_disable("DSP externe")` desactive justement en montage DSP. Sans ce fichier, `Cipher.current()` rend None et `cipher.apply()` rend le burst INCHANGE dans les deux sens : ni dechiffrement de la descente, ni chiffrement de la montee. Le compteur le disait depuis le debut, on ne l'avait pas lu : `A5 dl=0 ul=0` a chaque ligne STATS.

C'est exactement la cause qui a fait naitre ce fichier (montant.c) pour le RACH et le SDCCH : sous DSP_EXTERN, tout ce que publiait la couche 1 gr-gsm disparait, et il faut le republier par scrutation. Le Kc avait ete oublie.

`montant.c` publie donc maintenant `/dev/shm/calypso_kc_l1` depuis `d_a5mode` et `a_kc[4]` du NDB, disposition reprise TELLE QUELLE de `publish_kc()` (seq(4) algo(1) longueur(1) Kc[8] 0xFF, mots de a_kc en gros-boutiste et a l'envers) pour que `pont/cipher.py` la lise sans changement. Meme grace de 5 scrutations avant d'annoncer un retour en clair, pour la meme raison : le firmware efface `d_a5mode` a chaque DM_REL_REQ, y compris quand le Kc revient juste apres, alors que la BTS ne cesse jamais de chiffrer. `MONTANT_KC=0` coupe la publication.

A verifier au prochain run en A5/1 : `[montant] chiffrement A5/1 : Kc publie` dans dsp.log, puis `A5 dl=... ul=...` non nuls dans la ligne STATS du pont, et la transaction qui passe le CIPHERING MODE COMPLETE sans tomber a 96 erreurs.

2.2.1.33.13 CORRECTION : le DSP detecte bien la FB [2026-09-22, 11:30]

L'entree « FBSB mid-campement : le DSP ne detecte rien » ci-dessus est FAUSSE. Elle s'appuyait sur un echantillonnage de `/dev/shm/calypso_api_ram` a 3 ms pendant 12 s qui ne voyait jamais `d_fb_det=1`. Les jalons du meme run disent le contraire :

```
total jalons d_fb_det=1 : 196
[jalon] fn=17980 SB PLAUSIBLE page=1 BSIC=7 a_sch=8000 0707 061c 0191
SB decodees (osmocon) : 25   pour   147 L1CTL_FBSB_REQ
```

`d_fb_det` est TRANSITOIRE – leve puis efface dans la trame. L'echantillonner a 3 ms sur des trames de 5,8 ms le rate la plupart du temps, et ne pas le voir ne prouve rien. La sonde de [[sonde-api-ram-vivante]] vaut pour les valeurs qui DURENT (TOA, PM, SNR, d_fb_mode) ; pour un drapeau fugace, il faut les jalons de dsp.log, qui sont poses par le code au moment ou il le lit.

Ce que disent les vrais chiffres : 25 succes pour 147 demandes, soit 17 % – le meme taux que les 58/281 releves a 10:09. Le DSP detecte la FB et decode la SB ; c'est la PROCEDURE FBSB qui expire avant que la SB n'arrive. Le banc en pas-a-pas est plus lent que le budget de tentatives du firmware. C'est la qu'il faut chercher, pas dans la detection.

2.2.1.33.14 Le Kc est publie ; le blocage est maintenant AVANT le chiffrement [2026-09-22, 11:33]

Le side-band manquant est comble : `/dev/shm/calypso_kc_l1` existe (32 octets, ecrit une seconde apres le demarrage de c54x_exe) et `[montant] retour en clair (seq=1)` est trace. Mais `d_a5mode` n'est jamais devenu non nul dans ce run : le CIPHERING MODE COMMAND n'est jamais arrive. **La voie A5 n'est donc toujours pas exercee** – ni le publieur de Kc de montant.c, ni le dechiffrement de pont/trx.py. Ne pas les compter comme valides.

Ce qui bloque avant, et c'est net dans le journal du mobile :

```
11:32:50  AUTHENTICATION RESPONSE           <- la transaction va jusque-la
11:32:50  Unnumbered frame not allowed      <- puis 4 s de
11:32:50  MDL-ERROR-IND cause 12               « fenetre morte »
11:32:52  MON: lev=<=-110 snr=0 ... TS=1/0
11:32:54  Dropping frame with 96 bit errors  <- et la tempete s'installe
11:33:09  T3210 expire
```

Le MSC, lui, avait lache a 11:32:53.

Et les compteurs du meme run disent que ce n'est PAS un probleme de livraison :

```
tn=-1 ss=0 stockes=4392 joues=428 manques=0 replis=6
trames du canal tracees : 300     dont « BURST MANQUANT » : 0
UL bursts=218 tard=3              (1,4 %)
```

Chaque trame du canal a recu son burst, aucune n'a ete jouee vide, et le montant ne perd plus rien. Pendant que le mobile mesure `lev=<=-110 snr=0` sur son intervalle dedie, le BSP lui a bel et bien remis un burst pour chacune de ses trames.

La perte est donc APRES la remise, dans la demodulation du burst dedie par le DSP – ni dans le pont, ni dans le magasin, ni dans l'horloge. Les trois premiers sont maintenant mesures et hors de cause. C'est la que doit porter la suite : `PONT_NB_DEBUG=1` donne le TOA, le PM, le SNR et l'en-tete `a_cd` par burst ; comparer ceux d'un burst du canal dedie a ceux d'un burst BCCH de la meme seconde dira si le probleme est un calage (TOA) ou une amplitude (PM).

Et l'`Unnumbered frame not allowed` arrive sur un AUTRE datalink que les I-frames (`dl=0x...b9e8` contre `0x...bda8` ailleurs) : c'est la liaison SAPI 3, restee IDLE. Une trame U qui atterrit sur SAPI 3 pendant une transaction SAPI 0, c'est encore une adresse LAPDm lue de travers – meme famille que les SAPI 2/5/6 deja vus.

2.2.1.33.15 A5/1 : la chaine fonctionne [2026-09-22, 11:49]

Premier run qui franchit le chiffrement. Les deux correctifs de 11:22 et 11:29 sont valides par la mesure :

```
[montant] chiffrement A5/1 : Kc publie vers /dev/shm/calypso_kc_l1 (seq=2)
[pont]    chiffrement descendant confirme par la BTS (fn=1836)
[pont]    A5 dl=904 ul=68

11:49:02  LOCATION UPDATING ACCEPT (lai=001-01-1)
11:49:02  got TMSI 0x9e61e668
11:49:02  TMSI REALLOCATION COMPLETE
```

Le mobile lit un LOCATION UPDATING ACCEPT **chiffre** : impossible une heure plus tot, ou la tempete de 96 erreurs commencait a la ligne du CIPHERING MODE COMPLETE. 904 bursts descendants dechiffres par le pont, et le drapeau `dl_active` leve par la BTS comme prevu.

Le blocage a donc encore avance d'un cran – il est maintenant sur le DERNIER bloc MONTANT. Cote MSC :

```
11:49:01  TMSInew-0x9E61E668
11:49:03  LOCATION UPDATING REJECT     (reste « TMSInew », jamais « TMSI- »)
```

Le mobile a emis son TMSI REALLOCATION COMPLETE, le MSC ne l'a pas recu.

Hypothese a verifier, encore une asymetrie de chiffrement : le pont chiffre le montant des que `cipher.current()` rend une cle, donc des que `montant.c` a publie le Kc. Or `publier_kc()` scrute toutes les 22 trames (~100 ms) et peut publier AVANT que le mobile n'ait bascule lui-meme – il ne chiffre qu'apres avoir emis son CIPHERING MODE COMPLETE. Dans cette fenetre le pont chiffrerait un bloc que le mobile a emis en clair, et la BTS ne le lirait pas. Ca corromprait exactement les blocs autour de la bascule.

Mesure a faire : comparer l'horodatage du `seq=2` (publication du Kc) a celui du CIPHERING MODE COMPLETE. Si le premier precede le second, la fenetre existe. Le remede serait le symetrique de ce que fait deja la descente : ne chiffrer le montant qu'apres une preuve, pas des que la cle est connue.

2.2.1.33.16 Le Kc perime chiffrait la connexion suivante [2026-09-22, 11:57]

Bug introduit par le publieur de Kc, trouve et corrige dans la foulee. La grace `KC_GRACE_CLAIR` (5 scrutations avant d'annoncer un retour en clair), recopiee de publish_kc(), sert au cas INTRA-connexion : le firmware efface d_a5mode a chaque DM_REL_REQ, y compris pendant un Assignment Command ou le Kc revient juste apres. Mais entre DEUX connexions elle est nuisible : l'enregistrement algo=1 restait lisible cinq scrutations de plus, et pont.py – qui relache pourtant sa cle a chaque IMMEDIATE ASSIGNMENT (downlink.py, `cipher.release`) – la relisait aussitot dans le fichier et la restaurait.

Il chiffrait alors le montant de la connexion SUIVANTE des son premier bloc, pendant que le mobile emettait encore en clair. Mesure du run de 11:53 :

```
A5 dl=0 ul=200          <- tout le montant chiffre, rien de dechiffre
fn=2525  01 52 19 05 14 ...   AUTHENTICATION RESPONSE
fn=2729  01 52 19 05 14 ...   LE MEME, faute d'acquittement
```

Corrige : `montant_canal_libere()` (deja appele sur PONT_DCCH genre 0xFF) leve un drapeau qui fait publier le retour en clair IMMEDIATEMENT, sans la grace. Au passage, `d_a5mode` est desormais lu a CHAQUE trame (deux acces memoire) et non plus une sur 22 : un changement de mode ne peut plus attendre 128 ms, ce qui laissait partir en clair le premier bloc chiffre du montant.

Effet mesure au run de 11:58, retour au bon profil :

```
A5 dl=905 ul=68     chiffrement descendant confirme par la BTS (fn=1836)
11:58:13  LOCATION UPDATING ACCEPT + got TMSI 0x9846D415
```

2.2.1.33.17 Ce qui reste : l'acquittement LAPDm descendant

```
fn=2430  01 64 35 06 32 ...   CIPHERING MODE COMPLETE
fn=2583  01 74 35 06 32 ...   LE MEME, retransmis (bit P)
(aucun TMSI REALLOCATION COMPLETE n'est jamais publie)
```

Le reseau a pourtant RECU le CIPHERING MODE COMPLETE – sans lui le MSC n'aurait pas envoye le LOCATION UPDATING ACCEPT. Ce qui manque est donc l'acquittement LAPDm DESCENDANT : le mobile ne le voit pas, reste en retransmission, et sa fenetre de 1 l'empeche d'emettre le bloc suivant.

Le montant est hors de cause (`tard=0`), le chiffrement aussi (correct dans les deux sens, mesure). Il reste la perte residuelle de blocs DESCENDANTS sur le canal dedie – la famille des « 96 erreurs », deja isolee comme etant apres la remise des bursts par le pont (`manques=0`, `BURST MANQUANT=0`) et donc dans la demodulation du burst dedie par le DSP.

CORRECTION : l'hypothese « le pont chiffre le montant trop TOT parce que le Kc est publie en avance » ecrite plus haut est fausse dans ce sens-la. `publier_kc` scrutait en RETARD (22 trames), pas en avance. Le vrai defaut etait le Kc PERIME d'une connexion precedente, ci-dessus.

2.2.1.33.18 Le burst dedie arrive au DSP a moitie puissance [2026-09-22, 12:12]

Mesure par la sonde `[nb]` (plafond porte de 400 a 20000 : les 400 etaient consommees par le campement avant toute connexion), bornee a la fenetre ou le canal etait REELLEMENT arme (ticks 2047..3529 du run de 12:04) :

```
bursts DEDIES        n=176    TOA moy 1,9   PM 2310   SNR 232
bursts BCCH / CCCH   n=2800   TOA moy 4,2   PM ~4500  SNR ~500
```

Le burst du canal dedie arrive **deux qbits trop tot** dans la fenetre et a **la moitie de la puissance**, avec un SNR deux fois moindre. Ce n'est pas du bruit : 176 bursts, ecart systematique.

⚠ Piege de lecture : le `p51` imprime par `[nb]` est celui du TICK de l'ARM, pas celui de la trame BTS. Les deux different de `g_ts0_offset` (-670 sur ce run). Un premier regroupement fait sans cette conversion donnait des moyennes mixtes et ne montrait rien.

Trois causes possibles, toutes ECARTEES par la mesure :

- **la geometrie de fenetre** : identique, `fenetre=151 marge=3` sur les 75 trames dediees jouees, comme pour un bloc BCCH ;
- **le dechiffrement A5 que j'ai ajoute** : `gsm.a5_xor()` n'XORe que les bits 3-59 et 88-144, les deux moities de donnees. La sequence d'apprentissage reste intacte, le correlateur du DSP n'est pas touche ;
- **la qualite des bursts eux-memes** : le pont decode les MEMES bursts avec zero echec – `TS1/0:28/0 TS1/32:10/0`, `crc=0`. Les bits sont bons.

Donc : le pont livre des bursts CORRECTS, TOUS (`manques=0`, `BURST MANQUANT=0`), dans la BONNE fenetre – et le DSP les demodule a moitie puissance et deux qbits trop tot. Le defaut est dans l'injection/demodulation cote DSP de l'intervalle dedie substitue, pas dans le pont.

C'est la cause directe des « 96 bit errors », de la perte de ~3 blocs descendants sur 4 (chaque aller-retour LAPDm coute 4 multitrames au lieu d'une, cf. la cadence du montant plus haut) et donc du LOCATION UPDATING REJECT : le MSC lache avant que le TMSI REALLOCATION COMPLETE ne puisse partir.

Ou chercher : `bsp_ts0_livrer()` module le burst dedie exactement comme un burst TS0 (`gmsk_moduler(bits,148,30000,0,0.5,1`
puis `gmsk_elargir(..., CALYPSO_BSP_NB_SYM=0.3)`), et pourtant le resultat differe. La difference doit etre en aval : ce que le RIF/DMA fait de ces echantillons quand la fenetre est armee sur TS1 et non sur TS0. `CALYPSO_BSP_TS0_DEBUG=1` donne `rif_avant=` par trame ; le comparer entre trames dediees et trames TS0 est la prochaine mesure.

2.2.1.33.19 Deux pistes de plus ecartees, et une a creuser [2026-09-22, 12:20]

- **`rif_avant`** : identique. 432 trames dediees a `f=151 m=3 rif=0`, et 1297 trames NON dediees exactement pareil. Le niveau du RIF ne distingue pas les deux. Piste morte.
- **`CALYPSO_BSP_VERIF=1`** (compare la DARAM au burst remis au BSP, apres que le DSP a tourne) : 2849 « partiel » contre 8 « VALIDE ». Reparti par type de trame, avec la conversion tick -> trame BTS (offset -690) :

  ```
  TS0    n=3360  echantillons identiques : 14 % en moyenne, 2775 a 0 %
  DEDIE  n=640   echantillons identiques :  1 % en moyenne,  626 a 0 %
  ```

  L'ecart va dans le bon sens mais la sonde est dominee par un effet attendu : elle compare APRES `jouer_trame`, donc apres que le DSP a consomme et reecrit le tampon. Un « 0 % » ne prouve pas que la livraison a rate. A reprendre en comparant AVANT que le DSP ne tourne, ou en marquant le tampon.

Etat du diagnostic, tout ce qui est mesure :

```
le pont livre        des bursts corrects   (TS1/0:28/0, crc=0)
                     tous                  (manques=0, BURST MANQUANT=0)
                     dans la bonne fenetre (f=151 m=3, identique a TS0)
                     au bon niveau RIF     (rif_avant=0, identique a TS0)
le DSP en tire       TOA 1,9  PM 2310  SNR 232
alors que sur TS0    TOA 4,2  PM ~4500 SNR ~500
```

Tout ce qui precede la demodulation est desormais mesure et identique entre les deux cas. Le defaut est dans ce que le DSP fait de ces echantillons quand la fenetre est armee sur l'intervalle dedie.

2.2.1.33.20 CALYPSO_BSP_PAGE_FOLLOW=1 : essai negatif [2026-09-22, 12:23]

L'experience laissee ouverte le 2026-09-19 a maintenant une reponse. Sa premisse est confirmee par une mesure independante d'aujourd'hui :

```
w_page=0 -> 0x0cce 1811x, 0x0e4e 41x
w_page=1 -> 0x0cce 2115x, 0x0e4e 33x
```

98 % des bursts atterrissent a la MEME adresse quelle que soit la page que la tache DSP va lire, alors que d_dsp_page alterne bien (0x0002/0x0003). Le taux de correspondance DARAM suit : 12 % a 0x0cce contre 39-43 % a 0x0e4e.

Mais deposer a `base + w_page*stride` (PAGE_FOLLOW=1, pas 0x180) n'ameliore pas : la transaction meurt juste apres l'IDENTITY RESPONSE, deux LOCATION UPDATING REJECT sans meme un `TMSInew` cote VLR – alors que la reference atteignait le CIPHERING MODE COMPLETE et obtenait un TMSI. Un seul run, et la variance de ce banc est grande, mais le sens est clair : ce n'est pas la bonne correction. Defaut remis a 0.

Ce que ca apprend quand meme : l'adresse n'est pas le probleme, ou pas seule. La ROM programme son AAD, le BSP la suit (AAD_FOLLOW=1), et forcer une autre adresse casse. Le ping-pong manquant se joue ailleurs – peut-etre que la ROM ne reprogramme l'AAD qu'une fois sur N parce que son ISR de fin de DMA ne s'execute pas a chaque trame (cf. DSP Error Status 24 = DMA_PROG|DMA_TASK, permanent depuis le boot, files de requetes qui debordent).

C'est le fil a tirer : pourquoi la ROM ne reprogramme-t-elle pas son AAD a chaque trame, et pourquoi ses files DMA debordent-elles en permanence.

2.2.1.33.21 La ROM n'acquitte pas ses fins de DMA [2026-09-22, 12:47]

Chaine complete, chaque maillon mesure aujourd'hui :

```
la ROM ne lit jamais DMA2_CTRL avec IRQ_STATE
    (0 trace « effaces a la lecture » sur un run de 2334 erreurs 24 ;
     20 sur le run precedent, contre des milliers de transferts)
  -> IRQ_STATE reste pose, ses files circulaires de 14 entrees debordent
    (`DSP Error Status: 24` = DMA_PROG|DMA_TASK, permanent depuis le boot)
  -> l'AAD n'est pas reprogrammee d'une trame a l'autre
    (98 % des bursts a 0x0cce quelle que soit la page : w_page=0 -> 1811x,
     w_page=1 -> 2115x, alors que d_dsp_page alterne bien)
  -> le burst N ecrase le N-1 avant lecture
  -> demodulation degradee sur l'intervalle dedie
    (TOA 1,9 / PM 2310 / SNR 232 contre 4,2 / ~4500 / ~500 sur TS0)
  -> ~3 blocs descendants perdus sur 4, chaque aller-retour LAPDm coute
     4 multitrames au lieu d'une
  -> le MSC lache avant le TMSI REALLOCATION COMPLETE : LU REJECT.
```

C'est le premier enchainement qui relie TOUT ce qu'on observe depuis ce matin, et chaque maillon est chiffre.

L'interruption de fin de DMA est pourtant censee partir : `calypso_rhea_dma.c` la leve si `CTRL_IRQ_MODE` est pose, et la valeur observee (0x05ab) l'a bien (bit 7). Mais la trace « end-DMA -> INT10n » n'apparait PAS non plus.

⚠ RESERVE SUR L'INSTRUMENT. Sur le dernier run, AUCUNE trace `[rhea-dma]` ne sort, y compris celles qui existaient avant mes modifications et qui sortaient la veille. Et le compteur que j'ai ajoute (« bilan : N transferts finis… ») n'imprime jamais alors que `strings` le trouve dans le binaire EN COURS et que la trace situee deux lignes plus bas, dans le meme bloc `if`, imprime. Cette contradiction n'est pas resolue : tant qu'elle ne l'est pas, « 0 acquittement » peut vouloir dire « la ROM n'acquitte pas » OU « la sortie de ce module est perdue ». A trancher AVANT d'en tirer un correctif – par exemple en verifiant que stderr de calypso_rhea_dma.c arrive bien dans dsp.log (un test avec un fprintf inconditionnel au premier appel suffit).

2.2.1.34 2026-09-22 16:00 — Le workflow refute deux de mes conclusions

Vingt agents relus contradictoirement. Deux de mes affirmations, que j'avais presentees comme etablies, ne tiennent pas.

1. « La ROM n'acquitte pas ses fins de DMA » : FAUX. Le code prouve le contraire sans meme lancer le banc. La trace d'acquittement n'avait pas disparu du run : elle avait disparu de la SORTIE. `verbosite.c` classe les lignes de stderr par mots-cles, dans l'ordre. En reformulant le message le 2026-09-22 j'en avais retire le mot qui le placait au niveau 0 ; ne restait que « DMA », donc niveau 3, donc invisible au `-v` par defaut. J'ai lu une absence d'instrument comme une absence de comportement.

2. « Le pont livre tout (manques=0), le defaut est dans la demodulation du DSP » : NON ETAYE, et la perte majoritaire est ailleurs. `bsp_ts0_service()` repart par un `return` des qu'il n'a pas de burst du BTS pour la trame reclamee — AVANT d'appeler `bsp_ts0_livrer()`. Or `joues` et `manques` ne sont touches que DANS `bsp_ts0_livrer()`. Une trame dediee sautee en entier n'est donc vue par aucun des deux. Mesure du workflow : **79 trames dediees sur 172 jamais livrees, les deux compteurs a 0.** Mon « manques=0 » ne disait pas « rien ne se perd », il disait « je ne regarde pas la ou ca se perd ».

3. Et la cause amont, c'est ma propre horloge. La boucle d'asservissement posee ce matin (periode 0,25 s, rattrapage en 400 trames) tourne a ~0,07 Hz — SOUS la cadence a laquelle la vitesse du DSP varie. Elle ne suit plus : la phase part en cycle limite de +/- 100 trames, et a chaque demi-tour negatif le pont se retrouve derriere la trame reclamee. **12000 trames entierement sautees sur 120201 ticks, 10 %.** J'avais releve « 11 a 12 par run » a 11:08 — je comptais les trames tardives, pas les trames sautees.

Enchainement reel : mon horloge saute des trames -> le burst dedie n'est pas la -> `bsp_ts0_service()` repart sans rien livrer -> aucun compteur ne bouge -> je conclus que le DSP demodule mal. Trois fausses pistes de la journee partaient de ce zero.

2.2.1.34.1 Correctifs poses (16:04, binaire reconstruit)

#	Fichier	Correctif
1	`pont/trx.py` 41-43	`HORLOGE_PERIODE` 0,25 -> 0,05 ; `HORLOGE_PHASE_N` 400 -> 100. Boucle remontee a ~1,3 Hz, au-dessus de la perturbation. Ne PAS masquer ca avec `PONT_HORLOGE_AVANCE`.
2	`pont/trx.py`	`_tn_dedie()` -> `_burst_dedie(tn, fn)`. Le test portait sur l'INTERVALLE : en CCCH+SDCCH/4 le canal dedie vit sur TS0, celui qui porte aussi FCCH/SCH/BCCH/CCCH — jamais chiffrees. Des que A5 s'activait, tout TS0 partait XORe vers le DSP et le campement se defaisait. Meme tri que `Downlink._signalling`. TCH ouvert couvert aussi (necessaire pour un appel chiffre).
3	`calypso_bsp.c`	Nouveau compteur `g_dedie_perdues` : les trames dediees sautees en entier. Publie dans `/dev/shm/calypso_bsp_dedie`. **Ne plus jamais lire `manques=0` sans lire `perdues` en meme temps.**
4	`montant.c` 550	Le retour anticipe consomme `g_kc_liberer`. Sans ca la liberation restait armee, `seq` montait a chaque scrutation, et le pont rechargeait sans fin une cle inchangee.
5	`calypso_rhea_dma.c`	Bilan finis/acquittes promu en WARN (niveau 1) **quand il est mauvais** seulement. Pas de « ERR » force sur une ligne saine.
6	`calypso_rhea_dma.c` 554	`CTRL_IDLE` rendu sur la sortie anticipee. Toutes les autres sorties le reposaient ; celle-la laissait le canal annoncer un transfert qui n'aurait pas lieu.
7	`start-direct.sh`	`OSMO_MOB_VTY_PORT` passe a phonesim : il retombait sur 4247 alors qu'en `--dsp` le mobile lie 4347. Le modem oFono parlait a un port que personne n'ecoute — sans erreur.

Onze constats forts du workflow n'ont pas ete verifies (plafond a 12) : un second passage les vaut.

2.2.1.35 2026-09-22 16:15 — La mecanique reelle, mesuree cette fois

Le correctif d'horloge pose a 16:04 **n'a pas corrige la perte** : premiere mesure apres coup, `manques=2000` sur 10490 ticks, 19 % — soit pire que les 10 % que le workflow reprochait a l'ancien reglage. J'ai donc arrete de regler a l'aveugle : `PONT_HORLOGE_PERIODE`, `_KP` et `_PHASE_N` sont desormais lisibles dans l'environnement (valeurs par defaut inchangees), et j'ai instrumente le SIGNE de l'ecart au lieu de le supposer.

`bsp_ts0_stocker()` retient maintenant `g_ts0_fn_max`, la trame la plus recente recue du BTS. Sur un manque, la trace dit de quel cote vient l'ecart.

Resultat, sans ambiguite :

```
[ts0] tick=19316 : pas de burst BTS pour fn=18650 (manques=800/19316) ;
      derniere trame recue fn=18635, soit +15 : le DSP COURT DEVANT le BTS
[ts0] tick=20323 : pas de burst BTS pour fn=19657 (manques=1200/20323) ;
      derniere trame recue fn=19641, soit +16 : le DSP COURT DEVANT le BTS
```

L'ecart est **systematiquement positif**, de +1 a +16, jamais negatif. Le DSP reclame des trames que le BTS n'a pas encore produites. Ce n'est pas un cycle limite symetrique autour de zero — c'est un BIAIS constant.

Ce que ca change. Le workflow concluait « cycle limite de +/- 100 trames » et recommandait explicitement de NE PAS augmenter `PONT_HORLOGE_AVANCE`, au motif que la marge masquerait l'oscillation sans la supprimer.

Ce raisonnement vaut contre une oscillation ; il ne vaut pas contre un biais. Un biais systematique se corrige justement par un terme d'avance. La consigne du workflow reposait sur une hypothese que la mesure ne soutient pas.

Le taux en regime etabli reste ~20-25 % avec l'avance a 12 : de quoi expliquer que le canal dedie ne tienne pas quatre bursts de suite, donc pas de bloc LAPDm complet, donc pas de UA, donc pas de LU.

Balayage en cours de `PONT_HORLOGE_AVANCE` sur 12 / 30 / 60 / 120, 90 s de periode calibree par point, taux de manques + `tard/UL` + ecart median. Le reglage sera choisi sur la courbe, pas sur une intuition — la quatrieme de la journee aurait ete une de trop.

2.2.1.36 2026-09-22 16:20 — Asservir le pont au DSP est une erreur d'architecture

Trois mesures successives, chacune refutant la precedente, finissent par donner la mecanique. Dans l'ordre :

1. Le signe de l'ecart. Systematiquement positif, +1 a +16 : le DSP reclame des trames que le BTS n'a pas produites. Pas un cycle limite.

2. La perte n'est pas un taux, elle est bimodale. Entre checkpoints : 27 %, 25 %, puis **100 % sur 200 trames consecutives**, en alternance. Des coupures totales d'environ une seconde, pas une gigue.

3. Le DSP ne sprinte pas — il traine. Sonde sur `/dev/shm/calypso_horloge`, 4591 echantillons a 5 ms :

```
temps reel GSM     : 216.7 trames/s
cadence DSP mediane: 195.7 trames/s      deficit 9.7 %
9e decile          : 197.3
plus gros bond     : 2 trames
part a plus de 2x le temps reel : 0.0 %
```

Jamais d'emballement. Un deficit **constant de 9,7 %**. Et les « 12000 trames sautees sur 120201 ticks » du workflow, c'est 10,0 % : le meme nombre.

4. Le journal du BTS nomme le coupable.

```
DL1C NOTICE FN timer expire_count=7: We missed 6 timers (scheduler_trx.c:427)
DL1C ERROR  No more clock from transceiver (scheduler_trx.c:435)
```

5. Et les sources du BTS expliquent pourquoi c'est structurel. `osmo-bts/src/osmo-bts-trx/scheduler_trx.c` : les trames sont battues par un **timerfd cale en dur sur `GSM_TDMA_FN_DURATION_uS`**, 4615 us, temps reel. `IND CLOCK` ne sert qu'a CORRIGER ce timer :

- `elapsed_fn < 0` -> « We were N FN faster than TRX, compensating », il retarde ;
- `|elapsed_fn| > MAX_FN_SKEW` (50) -> resynchronisation brutale ;
- `fn_without_clock_ind == TRX_LOSS_FRAMES` (400) -> il s'arrete.

Et surtout, ligne 571, un `TODO` du projet :

> *put this computed error_us_since_clk into some filter function and use that to adjust our regular timer interval to compensate for clock drift*

Le filtre de derive n'existe pas. osmo-bts-trx ne SAIT PAS tourner a une cadence autre que le temps reel. Lui donner une horloge 9,7 % lente le laisse en permanence « plus rapide que le TRX » : il compense, il resynchronise, et entre deux son propre timer continue a battre au temps reel — produisant des trames que le DSP n'a pas encore atteintes, et en sautant d'autres.

2.2.1.36.1 Ce que ca dit de mon correctif du matin

Asservir l'horloge du pont au DSP prend le probleme a l'envers. Le BTS est le maitre temps reel par construction ; on ne peut pas le ralentir. C'est le DSP qui doit tenir la cadence. L'asservissement que j'ai pose ce matin a supprime la derive non bornee — ca, c'etait un vrai gain — mais il l'a remplacee par un desaccord permanent de 9,7 % que le BTS ne sait pas absorber.

Balayage en cours de `INSNS` (60000 / 54000 / 46000 / 38000) : cadence mediane obtenue, deficit, taux de manques, nombre de resynchronisations du BTS. Si la cadence suit bien 1/INSNS, le point qui annule le deficit annule la perte. Reserve a verifier au point retenu : `INSNS` est aussi un budget de FIDELITE — trop bas, la ROM n'a plus assez d'instructions pour finir son travail de trame, et c'est la detection FB/SB qui tombe.

2.2.1.37 2026-09-22 16:35 — La cause racine, trouvee et chiffree

Deux mesures decisives, chacune reproductible.

1. `INSNS` ne cadence rien. Balayage sur un facteur 2 :

```
INSNS=60000 -> 195.6 trames/s      INSNS=40000 -> 195.8
INSNS=54000 -> 196.0               INSNS=30000 -> 195.7
INSNS=48000 -> 195.8
```

Plat. Le budget d'instructions du DSP n'a aucun effet : le DSP n'est pas limite par son propre calcul.

2. Sans pas-a-pas, la machine vole. `LOCKSTEP=0` : **588,9 trames/s** median, jusqu'a 797 — 2,7 fois le temps reel. L'hote n'est pas sature (32 coeurs, charge 1,97).

Donc c'est l'echange de pas-a-pas lui-meme qui coute. Le TICK/GO en deux phases fait deux allers-retours de socket par trame entre QEMU et `c54x_exe`. Le gestionnaire de trame met ~5,11 ms la ou une trame GSM en fait 4,615 : il depasse le budget d'environ 0,5 ms.

Et `tdma_pacer()` transforme ce depassement en perte silencieuse. `calypso_trx.c:1069` :

```
target += GSM_TDMA_NS;
while (target <= now) {
    target += GSM_TDMA_NS;     /* en retard : on saute une trame */
}
```

Le stimulateur est juste — cible absolue, pas de derive accumulee — mais quand le gestionnaire depasse systematiquement le budget, `target` est toujours derriere `now`, la boucle avance d'une trame a chaque fois, et la cadence effective devient 1/(duree du gestionnaire) = 195,7 trames/s. `s->fn`, lui, n'avance que de 1 par tick : **l'interface air emulee prend 9,7 % de retard sur le temps reel, en permanence.**

2.2.1.37.1 Enchainement complet, du symptome a la cause

1. Le pas-a-pas coute ~0,5 ms/trame -> le DSP tourne a 195,7 au lieu de 216,7.
2. Mon horloge asservie repercute fidelement cette cadence lente au BTS.
3. `osmo-bts-trx` ne sait pas suivre une horloge lente : son timerfd est cale en dur sur 4615 us et le filtre de derive est un `TODO` vide (`scheduler_trx.c:571`). Il se croit en permanence « plus rapide que le TRX », compense, puis resynchronise.
4. Pendant une resynchronisation il ne produit rien : **coupures de ~200 trames consecutives**, soit une seconde de silence.
5. Le canal dedie n'obtient plus 4 bursts de suite -> pas de bloc LAPDm -> pas de UA -> pas de mise a jour de localisation.

Les 10 % de trames sautees que le workflow avait comptes, c'est exactement le deficit de cadence. Ce n'etait ni un cycle limite, ni la demodulation du DSP, ni l'acquittement DMA.

2.2.1.37.2 Trois voies possibles, aucune choisie sans arbitrage

Voie	Ou	Portee
A. Reduire le cout du pas-a-pas	`calypso_trx.c` / `pont.c`	2 allers-retours de socket par trame pour ~0,5 ms : c'est beaucoup trop pour un socket UNIX. Probablement une attente a timeout quelque part. Si on descend sous 4,615 ms, tout le reste tombe. **La plus propre.**

Voie	Ou	Portee
B. Implementer le filtre de derive du BTS	`osmo-bts/src/osmo-bts-trx/scheduler_trx.c`	Le TODO du projet lui-meme. Rend le BTS capable de suivre n'importe quelle cadence. Mais touche un 3e depot et le binaire `/usr/local/bin/osmo-bts-trx` sert AUSSI au mode sans `--dsp`.
C. Rendre la perte uniforme au lieu de groupee	`calypso_bsp.c` + `pont/trx.py`	Horloge du pont au temps reel (BTS content), DSP qui lit la trame la plus recente. On perd toujours 10 %, mais 1 sur 10 eparpillee au lieu de 200 d'affilee — un bloc LAPDm a 4 bursts et FEC survit a la premiere, pas a la seconde. Casse la continuite de FN vue par le firmware.

2.2.1.38 2026-09-22 16:40 — Trois hypotheses de plus, trois refutations

J'ai teste, chacune par balayage sur le banc, les trois causes candidates du deficit de 9,7 %. **Les trois sont fausses.** Je les consigne pour qu'on ne les re-essaie pas.

Hypothese	Balayage	Resultat
Le budget d'instructions du DSP (`INSNS`)	60000 / 54000 / 48000 / 40000 / 30000	195,6 / 196,0 / 195,8 / 195,8 / 195,7 — **plat sur un facteur 2**
Le quantum de re-essai du pas-a-pas	`GSM_TDMA_NS`/16 (288 us) puis /64 (72 us)	195,7 / 195,5 — **aucun effet**
La periode du coup de pouce au CPU (`CPU_KICK_NS`)	5,00 / 1,15 / 0,58 / 0,29 ms	195,8 / 195,9 / 195,8 / 195,9 — **plat sur un facteur 17**

Sur la troisieme j'ai failli me faire avoir : dans ce banc les variables d'environnement ne vont pas toutes au meme processus (cf. la note `env-calypso-quel-processus`), donc un balayage sans effet peut simplement vouloir dire que la variable n'arrivait pas. Verification faite dans `/proc/<qemu>/environ` : `CALYPSO_CPU_KICK_NS=288461` y etait bien. La refutation tient.

Les deux valeurs par defaut ont ete **remises a l'original** — je ne livre pas un changement de comportement que la mesure ne justifie pas. Les molettes (`CALYPSO_PONT_RETRY_DIV`, `CALYPSO_CPU_KICK_NS`) restent, elles servent a mesurer.

2.2.1.38.1 Ce qu'il reste, et pourquoi INSNS ne pouvait pas marcher

Les 5,11 ms par trame sont du **travail**, pas de l'attente :

- sans pas-a-pas, QEMU seul fait 588 trames/s, soit 1,70 ms par trame ;
- le pas-a-pas **serialise** l'ARM et le DSP au lieu de les laisser se recouvrir ; le reste, ~3,4 ms, c'est l'interpreteur C54x qui execute la trame.

Et `INSNS` ne pouvait rien y faire : c'est un **plafond**, pas une quantite de travail. La ROM finit son travail de trame et passe en idle bien avant de l'atteindre — baisser le plafond ne retire donc aucune instruction. C'est pour ca que le balayage est plat, et j'aurais du le prevoir avant de le lancer.

La piste suivante est la VITESSE de l'interpreteur, pas son budget : combien d'instructions la ROM execute reellement par trame, et a quel debit l'interpreteur les rend (~7 MIPS d'apres le calcul inverse). Les sondes de `c54x_probes.c` et les copies memoire par trame de `PONT_NB_DEBUG` sont les premieres choses a chiffrer.

2.2.1.39 2026-09-22 18:05 — SB : le TOA n'est pas la cause. La fenetre, peut-etre.

Refutation de ma propre these du jour. J'ai cru, et ecrit, que la SB echouait parce qu'elle atterrit a TOA 7 au lieu de 23. La mesure dit non :

```
[sb] fn=302  toa=11243  a_sch=0100  crc_ko
[sb] fn=333  toa=8743   a_sch=8000  CRC_OK     <- 8743 % 156 = 7
[sb] fn=371  toa=8743   a_sch=0100  crc_ko     <- meme 7
[sb] fn=402  toa=8743   a_sch=0100  crc_ko     <- meme 7
```

Le meme TOA donne CRC bon et CRC faux. Le TOA ne discrimine pas. Le correctif pose sur `g_toa_bias` perd donc sa justification (il est de toute facon inerte, voir plus bas).

Au passage, deux autres choses que j'avais dites et qui sont fausses : * « TOA=5 est une anomalie, le nominal est 23 » – non : `calypso_bsp.c:1505` dit que 5 est la valeur attendue d'un burst NORMAL sur ce banc (marge de 3 echantillons). Les 23 ne concernent que la SB (marge de 21). * « La DMA est en mode continu quand le burst arrive » – non : `one_shot=1` apparait 760 fois sur 3389.

2.2.1.39.1 Ce que la mesure etablit, en revanche

Longueurs de fenetre RIF reellement demandees sur un run de 2 min :

fenetre	occurrences	`nwin = len/2`
302 mots	3316	151
128 mots	73	64

Aucune fenetre >= 190. Or le cadrage de `calypso_bsp.c:1527` choisit sa marge ainsi : `nwin >= 190 ? 21 : nwin >= 150 ? 3 : 0`. La fenetre SB attendue – ALGTH 764, donc `nwin` 382 – **n'existe jamais**. La branche a 21 echantillons est inatteignable par construction, et la SB est demodulee dans une fenetre de burst normal.

C'est aussi pourquoi la sonde `[cadre]` compte **0** alors que `one_shot=1` arrive 760 fois : le bloc est bien garde par `one_shot`, mais aucune fenetre n'atteint jamais le seuil SB.

2.2.1.39.2 Etat des deux correctifs du jour sur ce chemin

Correctif	Verdict
`calypso_bsp.c` : appliquer `g_toa_bias` a la marge SB	**INERTE** – le bloc n'est jamais atteint (`[cadre]=0`), et sa justification est refutee (meme TOA, issues opposees). A retirer ou a garder en dormance documentee.
`gsm322.c` : `SYNC_RETRIES_CONN 8`	**tient jusqu'ici** – 0 `LOS during RACH` sur 5 tentatives reparties sur deux runs, contre 4 sur 8 avant. Pas encore etabli, mais rien ne le contredit.

2.2.1.39.3 La boucle TOA, defaut reel mais secondaire

`calypso_bsp_toa_feedback()` est appelee (`pont.c:1357`, `en=1` verifie), elle integre dans `g_toa_bias`… que **personne ne lisait** avant aujourd'hui, malgre le commentaire « samples, applied to the DARAM placement ». Et son entree est instable : `within = toa % 156` donne 60, puis 11, puis 48 d'un appel a l'autre – le signe de l'erreur alterne, l'integrateur fait +/-1 et revient a zero. Meme branchee, elle ne pourrait pas rattraper 16 echantillons. Deux defauts distincts, a traiter ensemble ou pas du tout.

2.2.1.39.4 La question suivante, et elle est nette

Pourquoi la ROM n'arme-t-elle jamais une fenetre de 764 mots pour la SB ? C'est elle qui programme ALGTH. Soit elle ne le fait pas, soit le modele RHEA ne le lui rend pas. C'est mesurable : tracer les ecritures de ALGTH par la ROM dans `calypso_rhea_dma.c`, et comparer a ce que `calypso_rhea_dma_get_len_words()` rend au moment de la SB.

2.2.1.40 2026-09-22 18:35 — Le recalage TS0 : mesure A/B, et refutation de mon predicteur

Hypothese : le DSP court devant le BTS (ecart toujours positif, +1 a +40) parce que l'offset tick->trame BTS est pose une seule fois et jamais revu ; quand le BTS s'arrete pour resynchroniser, le DSP reclame des trames inexistantes et le flux se troue. Or un bloc LAPDm, ce sont QUATRE bursts consecutifs. Correctif pose : reculer l'offset pour repartir de la trame la plus recente, en s'appuyant sur le contrat du mode STREAM (« only the ORDER matters »).

A/B, 2 x 6 min, meme banc, meme protocole :

	temoin	recalage
perdues / joues	20/132	0/161
recalages (recul)	0	433 (433)
MDL-ERROR	18	4
LU ACCEPT / REQUEST	1/3	1/2
trames jetees	686	**1409**
bits faux (mediane)	95	96

Ca ne marche pas. `perdues` tombe a zero et MDL-ERROR est divise par quatre, mais les trames jetees DOUBLENT et la mediane de bits faux ne bouge pas. Le LU reste a 1 dans les deux bras.

Explication qui colle : en remplacant une trame absente par la plus recente disponible, on ne livre pas un trou mais **le mauvais burst a la bonne place**. Pour le desentrelaceur, une donnee fausse mais plausible est pire qu'une absence – il ne peut plus la traiter comme un effacement. J'ai converti des effacements en erreurs.

Corollaire, et c'est le point important : « perdues » n'est PAS un predicteur du succes. La correlation que j'avais tiree de trois runs (7 % -> LU accepte, 0 % -> accepte, 42 % -> rejete) ne survit pas au test controle. Trois points suffisaient a la suggerer, pas a l'etablir.

Defaut remis a OFF (`CALYPSO_BSP_RECALE=1` pour le reessayer). Comme pour le quantum de re-essai et `CPU_KICK_NS` : on ne livre pas un changement de comportement que la mesure ne justifie pas.

Deux pistes si on y revient : ne recaler que HORS du canal dedie, ou marquer le burst rejoue comme peu fiable pour que le desentrelaceur l'efface au lieu de le croire.

Defaut de mesure a signaler : le chemin de recalage sort avant `manques++`, donc il aveugle ce compteur. Les deux bras n'etaient pas comparables sur cette metrique. C'est moi qui ai casse l'instrument en posant le correctif – exactement le genre de piege que `g_dedie_perdues` avait ete ajoute pour eviter ce matin.

2.2.1.40.1 Ce que la journee laisse debout

- LU : aboutit, chiffre A5/1, TMSI committe cote VLR. Repete de nombreuses fois. Quand le canal est propre, la transaction complete prend 4 secondes (assignation -> RR_EST_CNF -> IDENTITY -> LOC_UPD_ACCEPT -> TMSI REALLOC).
- `SYNC_RETRIES_CONN 8` (gsm322.c) : 0 `LOS during RACH` sur toutes les tentatives depuis qu'il est pose, contre 4 sur 8 avant. Le seul correctif du jour qui tienne.
- Le SMS atteint desormais `MMSMS-EST-CNF` puis `WAIT_CP_ACK`, et `SAPI 3 established` a ete vu. Avant il mourait en `MM_CONN_PENDING`.
- Une transaction Call Control a ete allouee pour la premiere fois (callref 0x138c), finie en `Timeout of T308`.
- Defaut restant : le descendant dedie se corrompt (69 a 99 bits faux sur 184). Observation non expliquee, relevee a 18:23 : le BER monte MONOTONEMENT de 43 a 95 en 17 s a `lev >= -47` constant. Une rampe, pas des creneaux – ce qui ne ressemble pas a une perte de bursts par paquets et suggere un desalignement cumulatif. A creuser.

2.2.1.41 2026-09-22 19:00 — Deux correctifs de plus, et le SMS montant passe

2.2.1.41.1 1. Effacement au lieu de la page perimee (calypso_bsp.c)

Trouve par l'arbitrage du workflow `wahnj19z5`, qui a au passage REFUTE 3/3 ses propres trois voies (cout du pas-a-pas, filtre de derive du BTS, perte uniforme). Mecanisme verifie a la main, quatre points :

- `calypso_bsp_rx_burst()` est le seul ecrivain de la DARAM des bursts, et n'est appelee que depuis `bsp_ts0_livrer()` – que le chemin de manque saute ;
- `calypso_rif_drain()` rend 0 sur FIFO vide ;
- le transfert sort alors par `if (got <= 0) break` SANS rien ecrire : **la page API garde le burst du tick precedent** ;
- sur le chemin DRR, le source dit lui-meme : « On an empty FIFO, DRR keeps its last value [...] returning 0 would fabricate a sample ».

Donc une trame manquante n'est pas un trou : **c'est la trame precedente rejouee**, et le decodeur tourne dessus. Signature mesuree : les blocs rejetes ont un nombre d'erreurs IDENTIQUE, 17 rejets = 96 neuf fois, 105 cinq fois. Un canal bruite ne rend pas neuf fois le meme compte.

C'est aussi pourquoi mon A/B du recalage etait aveugle : ses deux bras substituaient un burst FAUX (le plus recent d'un cote, le precedent de l'autre), **jamais un effacement**.

Resultat, 2 min : MDL-ERROR 18 -> **0**, LU accepte au PREMIER essai, trames jetees 114/min -> 66/min, perdues/joues 13 % -> 0,6 %. Reserve : l'histogramme des comptes d'erreurs reste concentre, donc la signature n'a pas disparu. Non explique.

2.2.1.41.2 2. Ne plus jeter la premiere FACCH montante (pont/uplink.py)

`_poll_facch()` faisait, au changement d'epoque TCH : `skip_pending()` + `return`. Or `tch.seq` est incremente par `tch.arm()`, appele quand le pont decode l'ASSIGNMENT COMMAND descendante – et l'ASSIGNMENT COMPLETE est LA PREMIERE chose que le mobile emet sur le nouveau TCH. Elle tombait dans cette fenetre et etait marquee « deja vue ».

Mesure, appel vers 600 a 18:55 : le mobile emet bien « ASSIGNMENT COMPLETE (cause #0) » ; le pont journalise « FACCH montante » SANS le suffixe « , ASSIGNMENT COMPLETE » ; le BSC conclut « Assignment failed in state WAIT_RR_ASS_COMPLETE, cause EQUIPMENT FAILURE: Timeout ».

La SACCH garde le saut (un rapport de mesure perime ne sert a rien), la FACCH est desormais traitee. `PONT_FACCH_SKIP=1` retablit l'ancien comportement.

2.2.1.41.3 Ce que le banc fait maintenant

- LU accepte au premier essai, chiffre, TMSI committe cote VLR.
- **SMS MONTANT ARRIVE AU RESEAU** : `db.c:695 Stored SMS id=33 in DB`.
- **SETUP d'appel recu par le MSC** : `gsm_04_08_cc.c:704 SETUP to 600`.
- Restent : la livraison MT du SMS (pas de CP-ACK) et l'aboutissement de l'appel.

2.2.1.41.4 Deux erreurs de lecture a noter, meme cause qu'au matin

J'ai affirme (a) qu'aucun `L1CTL_RACH_REQ` n'existait, (b) que le pont ne suivait pas le mobile sur le TCH. **Les deux etaient faux**, demolis par le journal complet deux commandes plus tard : dans les deux cas un `head -10` ou un `tail -6` avait tronque la sortie. C'est exactement la faute du matin avec `manques=0` : conclure d'une absence sans verifier que l'instrument regardait au bon endroit. A surveiller.

2.2.1.42 2026-09-22 19:47 — L'APPEL PASSE L'ASSIGNATION : le chainon manquant du TCH

2.2.1.42.1 Le defaut

`pont.py` publiait DEJA l'intervalle du canal de trafic dans `/dev/shm/calypso_tch_cfg` (`pont/state.py`, `Tch._write_cfg` : seq, tn, tsc, arfcn) des qu'il decodait l'ASSIGNMENT COMMAND descendante – verifie, le fichier contenait bien `seq=1 tn=2 tsc=7 arfcn=514`. Mais **personne ne le lisait cote DSP** : `calypso_bsp_set_dedie()` n'etait appelee que depuis la bande laterale SDCCH (`montant.c:425`). Le BSP continuait donc de substituer l'intervalle SDCCH (TS1) pendant que le mobile ecoutait le TCH (TS2).

2.2.1.42.2 La preuve, en croisant les deux journaux

Cote mobile :

```
19:42:05  MON: ... TS=2   ber=161        <- il EST sur le TCH
19:42:06  MON: no cell info  rxlev-full=-110   <- le plancher : du SILENCE
19:42:07  MON: no cell info  rxlev-full=-110
19:42:08  Channel type 64, subch 0, ts 1       <- il revient sur TS1
19:42:08  ASSIGNMENT FAILURE (cause #1)
```

Cote DSP, sur toute la session : `arme TS1 SDCCH/8`, **jamais TS2**. Cote BSC : `Assignment failed in state WAIT_RR_ASS_COMPLETE, cause EQUIPMENT FAILURE: Timeout`.

Ce n'etait donc pas un defaut de qualite du lien – c'etait un chainon absent. Et ca invalide au passage ma lecture precedente : j'avais accuse la premiere FACCH montante jetee (correctif `uplink.py`), qui etait un vrai bug mais PAS celui-la.

2.2.1.42.3 Le correctif

- `calypso_bsp.c` : genre `BSP_DEDIE_TCH` (=2). `bsp_dedie_trame()` retourne vrai pour TOUTES les trames de l'intervalle – sur un TCH/F la multitrame de 26 est entierement a la connexion (24 trafic + 1 SACCH + 1 libre), contrairement a un sous-canal SDCCH ou seuls 4 blocs sur 51 comptent.
- `montant.c` : `scruter_tch()`, pose a cote de `scruter_dcch()`, lit `/dev/shm/calypso_tch_cfg` et arme le BSP sur le bon intervalle. `MONTANT_TCH=0` coupe.

2.2.1.42.4 Le resultat

```
19:47:16  Sending 'SETUP'
19:47:17  ASSIGNMENT COMMAND
19:47:17  ASSIGNMENT COMPLETE (cause #0)      <- recue par le BSC cette fois
19:47:20  received CALL PROCEEDING
19:47:20  INITIATED -> MO_CALL_PROC
```

```
[montant] TCH (seq=1) : arme TS2 TSC=7 - toutes ses trames
[ts0] canal dedie arme : TCH TS2 (toutes les trames)
/dev/shm/calypso_bsp_dedie : tn=2 joues=1486 manques=0 perdues=0
```

Plus aucun `Assignment failed` cote BSC. L'appel atteint `MO_CALL_PROC`.

2.2.1.42.5 Bilan des quatre correctifs de la journee

#	Fichier	Defaut
1	`gsm322.c`	sortie d'idle : `ccch_state` gele + un seul re-essai FBSB -> aucun burst RACH emis
2	`calypso_bsp.c`	trame manquante = trame precedente REJOUEE (page API jamais reecrite)
3	`pont/uplink.py`	premiere FACCH montante apres armement TCH jetee par `skip_pending()`
4	`calypso_bsp.c` + `montant.c`	l'intervalle du TCH publie par le pont n'etait lu par personne

Mesures : LU accepte (contre 1 sur 3), `LOS during RACH` 4/8 -> 0, MDL-ERROR 18 -> 0 sur 31 min, SMS montant arrive au reseau, appel jusqu'a CALL PROCEEDING.

2.2.1.43 2026-09-22 19:55 — LE VERROU RESTANT, mesure : la ROM n'arme presque jamais sa fenetre SB

Question posee par un contraste : le pont livre une BCCH **parfaite** (741 blocs, 0 echec de CRC) et le DSP, nourri par le meme chemin au meme instant, ne decode la SB qu'une fois sur cinq. Ce n'est donc ni le lien, ni le pont, ni la qualite des echantillons.

Sonde posee dans `bsp_ts0_livrer()`, sur les bursts que `bsp_ts0_est_sb()` reconnait comme des SCH :

```
[sbwin] SB #500 : one_shot=0 nwin=0 marge=0 -> cadree comme un burst normal
        (dans une vraie fenetre SB : 12/500)
```

12 bursts SCH sur 500 tombent dans une vraie fenetre SB : 2,4 %. Les 97,6 % restants arrivent avec `one_shot=0`, `nwin=0`, donc `marge=0` : le burst est pose a l'offset ZERO, sans les 21 echantillons silencieux que le correlateur SB de la ROM attend (c'est eux qui donnent le TOA de 23).

Dans la meme fenetre de mesure : **1 CRC bon sur 16 jobs SB**. Les deux taux sont du meme ordre. C'est la premiere explication de la journee qui a le BON ORDRE DE GRANDEUR.

2.2.1.43.1 Ce que ca corrige dans mes conclusions precedentes

J'avais accuse le TOA (7 au lieu de 23) puis conclu, mesure a l'appui, que le TOA ne discriminait pas – meme `toa=8743` donnant `CRC_OK` et `crc_ko`. Les deux etaient vrais et je n'en tirais rien : **le TOA n'est qu'une consequence du cadrage**. Comme le cadrage est presque toujours le meme (mauvais), le TOA est presque toujours le meme aussi. Je cherchais la cause dans la consequence.

2.2.1.43.2 La question suivante, nette

Pourquoi la ROM n'arme-t-elle presque jamais une DMA one-shot pour son job SB ? Elle reste en mode continu (trame de 1250 symboles, TS0 sans marge). C'est elle qui programme ALGTH ; soit elle ne le fait pas, soit le modele RHEA ne le lui rend pas. A tracer dans `calypso_rhea_dma.c` (ecritures de ALGTH et de CTRL_ONE_SHOT par la ROM) en regard des jobs SB.

2.2.1.43.3 Pourquoi je n'ai PAS pose le correctif a chaud

Forcer `marge = 21` quand le burst est une SCH decalerait de 21 echantillons une trame dont la longueur est contrainte a 1250 en mode continu, et casserait le compteur de symboles de la ROM. C'est exactement le genre de correctif applique avant d'etre compris qui a coute quatre fausses pistes aujourd'hui.

Ce verrou commande tout le reste : le taux de synchronisation, donc la re-acquisition apres chaque liberation de canal, donc la fiabilite du SMS et de l'appel.

2.2.1.44 2026-09-23 10:55 — LU ACCEPT puis LU REJECT : le temporisateur X1 du MSC, pas la radio

Symptome, reproduit deux fois (10:43, 10:45) : le mobile recoit LU ACCEPT, emet TMSI REALLOCATION COMPLETE, et deux secondes plus tard recoit LU REJECT (il repond MM STATUS #98). Le MSC le journalise en `{MSC_A_ST_RELEASING}`. Consequence : le VLR oublie l'abonne, le SMS MT est jete (« Freeing transaction that still contains an SMS »), le CM SERVICE suivant est rejete (cause 4) et tout repart par un nouveau LU.

Cause : `timer geran X1` de osmo-msc (« Complete Layer 3, Authentication and Ciphering timeout »), **5 s** par defaut. Il court de la COMPL_L3 jusqu'a la fin du LU, TMSI REALLOC COMPLETE compris. Chronologie du 10:45 : COMPL_L3 10:45:47, chiffrement 10:45:50, X1 echu 10:45:52, avant l'arrivee du TMSI REALLOC COMPLETE. Le banc, plus lent que le temps reel, ne tient pas 5 s.

Correctif : `timer geran X1 30` sous `msc` dans /etc/osmocom/osmo-msc.cfg et dans le gabarit osmo-operator/configs/osmo-msc.cfg (pose aussi a chaud par la VTY 4254).

Ce qui reste, mesure juste apres : un LU a 10:50:33 meurt sur T3260 (12 s), le DTAP descendant (demande d'identite ou d'authentification) n'arrive jamais au mobile. Sur le SDCCH on voit « Dropping frame with 96 bit errors » (x3, toujours la meme signature 96) et des trames a « 0 bit errors » jetees quand meme (fire_crc >= 2). Le descendant dedie reste le verrou, avec la fenetre SB.

2.2.1.44.1 Les journaux du banc DSP dans le panneau

Le panneau (tmux calypso, osmo-fft-snap) suit /run/user/0/osmo-nitb/logs/{qemu,osmocon,mobile}.log et /dev/shm/pont.log. run.sh n'ecrivait que dans /tmp/c54x-pont : tous les volets restaient vides. Desormais le vrai fichier est pose au chemin du panneau, et $RUNDIR/.log est un lien vers lui. Ce sens-la est obligatoire, car un `tail -F` ouvert refuse un fichier remplace par un lien. pont.py n'est plus lance avec `--no-record` : c'est ce drapeau qui laissait /tmp/iq_fft_ms.fifo, donc la FFT, sans producteur. PONT_AIRREC=0 et PANNEAU_LOGS=none retablissent l'ancien comportement. Au passage : la trace `[trx] A_SCH` de QEMU s'arrete apres 40 lignes (calypso_trx.c:1076), ce qui est voulu ; ce n'est pas un gel.

2.2.1.45 2026-09-23 10:58 — Run de 10:53 : LU accepte, SMS MT perdu sur le descendant dedie

- X1 a 30 s : le LU passe, TMSI 0x26861AD4 valide par le VLR.
- Le MSC livre aussitot un SMS MT en attente. La SAPI 3 s'etablit (10:53:42, « new SAPI 3 link state idle -> established »), mais le CP-DATA n'arrive jamais au mobile. Son T3240 expire a 10:53:48, il libere le canal, et le MSC abandonne le SMS (« dropping pending message »).
- Erreurs du descendant dedie sur tout le run : 80 a 114 bits faux sur 456, plus 16 trames a « 0 bit errors » jetees quand meme. 114 = un burst entier, donc c'est typiquement UN burst sur les quatre du bloc qui est faux.
- Cote BSP : tn=1 joues=464 manques=0 perdues=0. Chaque trame dediee recoit un burst du bon intervalle, et pourtant un burst sur quatre est faux.
- Piste suivante, deja instrumentee mais muette par defaut : « dedie : DESACCORD table=.. tpu=.. » (calypso_bsp.c, bsp_ts0_livrer). Il faut CALYPSO_DEBUG=BSP sur c54x_exe. Un decalage d'une trame entre la table 45.002 et la fenetre TPU programmee par l'ARM donnerait exactement un burst faux sur quatre.
- Lien montant : « UL bursts=66 tard=225 » dans les 40 premieres secondes, puis tard fige a 247. Les pertes se concentrent pendant le calage de l'horloge, qui tombe au moment du premier LU.
- Avertissements du build corriges : prototypes de calypso_trx_get_fn et de calypso_inth_arm_ack (appels implicites en int), -Waddress, et les if enchaines sur une ligne. Zero avertissement.

2.2.1.46 2026-09-23 11:06 — PREMIER SMS MT LIVRE DE BOUT EN BOUT ; le montant jetait 69 % de ses bursts

- 11:05:56 : le mobile recoit « test » (OA 777), repond CP-ACK puis RP-ACK. Cote MSC : CP-ACK 11:06:00, RP-ACK 11:06:03, transaction fermee proprement. Cote mobile : « % SMS from 777: 'test' ». Ce qui l'a debloque : le correctif SAPI 3 de pont/uplink.py, pose par une autre session le 2026-09-23. L'UA du SAPI 3 partait sur la SACCH montante, la BTS n'etablissait jamais le SAPI 3.
- Traces LAPDm du mobile (llapd debug, pose a chaud par la VTY 4347) : toutes les trames I montantes sont retransmises 2 a 5 fois (TIMER_RECOV) avant d'etre acquittees. Le descendant SDCCH, lui, decode une trame par multitrame sans trou.
- Cause : pont/trx.py Transmitter.run JETAIT tout burst montant reveille apres sa trame (off < -window_tol). Run de 11:04 : 141 emis, 314 jetes (69 %). Les runs sans enregistrement I/Q en jetaient 16 %. Le reveil du thread souffre du GIL partage avec record.py, que la FFT a reactive.
- Or osmo-bts-trx range un burst montant par son fn, compare au dernier fn traite du canal logique (common/scheduler.c, trx_sched_route_burst_ind). L'heure d'arrivee n'y entre pas : un vrai TRX livre toujours le montant apres coup. Correctif : envoyer le burst en retard tant qu'il reste dans PONT_UL_RETARD_MAX = 26 trames. Le compteur « tard » compte desormais les bursts en retard, envoyes ou pas. A mesurer au prochain run : les retransmissions T200 doivent disparaitre.

2.2.1.47 2026-09-23 11:20 — Appel : CONNECT jamais recu ; pont coupe en deux points d'entree

2.2.1.47.1 Appel vers 600, run de 11:10

SETUP (11:11:34), assignation TCH/F TS2 FR, ASSIGNMENT COMPLETE, puis CALL PROCEEDING recu par FACCH (11:11:38). Un SMS MT passe meme en pleine communication (SAPI 3 sur la SACCH). Le 600 decroche : le MSC passe en CONNECT_IND et envoie le CONNECT vers 11:11:37. Le mobile ne le recoit jamais, et T313 expire a 11:12:07. * Le firmware ne remonte un FACCH de TCH que si le DSP a pose B_BLUD dans a_fd[0] en fin de bloc (prim_tch.c:246). Sans detection des bits de vol, rien ne remonte : c'est le silence observe. * Le masque A5 du pont (gsm.a5_xor) laisse bien les bits de vol (60, 87) en clair. Ce n'est donc pas lui. * Le pont decode lui-meme le TCH descendant : facch 5 -> 39 entre 11:11:39 et 11:11:54 (les retransmissions du

CONNECT), tch_dl fige a 219, puis tch_crc qui grimpe de 47 a 877. A partir d'environ 11:11:49, le pont n'arrive plus a decoder une seule trame TCH descendante, ni parole ni FACCH. A creuser en premier : le dechiffrement du TCH par le pont (cle, fn, dl_active), puis la detection FACCH du DSP une fois la parole lancee.

2.2.1.47.2 Decoupage du pont (demande operateur)

- pont/pont.py = montage DSP (c54x_exe/run.sh MODE=dsp, grgsm_exe sans argument).
- pont/pont_uncipher.py = montage grgsm (start-direct.sh, run.sh MODE=grgsm, PONT_DSP_PORT=0 grgsm_exe). Il refuse –dsp-port.
- Meme paquet et A5 dans le pont des deux cotes. Ce qui differe passe par des defauts poses avant l'import (os.environ.setdefault) : PONT_UL_RETARD_MAX vaut 0 pour grgsm (jeter le burst en retard, comme avant) et 26 pour le DSP. Le defaut du paquet est 0 : le chemin grgsm garde exactement son comportement d'avant.
- Motifs de processus mis a jour : 09-teardown.sh, rapport-run.sh, conky-osmo-status.sh, build-debs.sh. /usr/local/bin/grgsm_exe (hors depot) aussi, avec une copie de l'original dans le scratchpad de la session.

2.2.1.48 2026-09-23 11:27 — La boucle « no service » toutes les 5 a 6 s : la seconde synchro de la selection

Trace DCS (cs debug, pose par la VTY 4347), une boucle complete : 1. Balayage : « Sync to ARFCN=514 (No sysinfo yet) », « Channel synched », SI lues, « Cell found », C1 = 0. 2. « Cell ARFCN 514 selected », puis « Tune to frequency 514 » : une SECONDE acquisition FB+SB (gsm322_sync_to_cell, mode CCCH COMB). 3. Elle echoue : « Channel sync error, try again », puis a nouveau « Channel sync error ». SYNC_RETRIES vaut 1 : la cellule est desselectionnee (« Unselect cell due to sync error », « Loss of CCCH »), MM_EVENT_LOST_COVERAGE, et la boucle repart en 1. Sur ce run, 93 FBSB_REQ sur 514 et 4 sur le voisin 614. Il faut deux synchros reussies de suite, alors que le DSP n'en reussit qu'une sur trois a cinq : c'est le verrou de la fenetre SB (voir 2026-09-22 19:55).

Correctif cote mobile : gsm322.c, sync_retries_selection(). Le nombre d'essais du « Tune to frequency » se lit dans L23_SYNC_RETRIES_SELECTION. Le defaut reste SYNC_RETRIES (1) : le binaire /usr/local/bin/mobile est partage avec le montage grgsm et le mobile fake_trx. c54x_exe/run.sh pose 8 en MODE=dsp. L'ancien binaire est garde dans le scratchpad de la session.

A noter aussi : la mesure de puissance rend souvent « rxlev <=-110 (0) », le plancher, soit C1 = 0 tout juste. Avec un RXLEV_ACCESS_MIN plus haut, la cellule serait jugee inutilisable. A regarder dans calypso_bsp_rssi_apm.

2.2.1.49 2026-09-23 12:47 — Le pont DSP (pont/dsp/) : la bascule TCH suit le firmware

2.2.1.49.1 Ce qui cassait les appels du run de 12:22 (confirme dans le code)

- Appels 2 et 3 : le pont decode l'ASSIGNMENT COMMAND et ecrit aussitot /dev/shm/calypso_tch_cfg (Tch.arm). montant.c scruter_tch armait alors le BSP en TCH des la trame suivante. Pour un TCH, bsp_dedie_trame() est vrai sur TOUTES les trames : le BSP jouait TS2 a la place de TS0 et du SDCCH TS1 pour toutes les trames pas encore jouees. Or le DSP a 12 a ~200 trames de retard sur la BTS, et l'ASSIGNMENT COMMAND elle-meme en faisait partie. Dans dsp.log : « TCH (seq=2) : arme TS2 », un seul a_cd FIRE KO, puis plus aucun bloc SDCCH. Cote mobile : 80-100 bit errors, jamais d'ASSIGNMENT COMMAND, puis LOS.
- Appel 1, retour sur le SDCCH : rien ne rendait TS1 au BSP apres l'ASSIGNMENT FAILURE, parce que le tap QEMU ne republie dcch_cfg que si chan_nr change. D'ou 90-110 bit errors jusqu'a la liberation. En plus, l'ASSIGNMENT FAILURE, publiee sur calypso_sdcch_ul, partait en FACCH sur TS2 (uplink.py : is_open() suffisait).
- Le TS du TCH n'etait dechiffre vers le DSP qu'apres Tch.prove(), alors qu'osmo-bts le chiffre des l'activation. Les premiers blocs, dont l'UA, etaient perdus.
- Le Kc etait lache a la liberation lue dans dcch_cfg, donc a l'heure du DSP, alors que la BTS chiffrait encore. Blocs TS1 en echec cote pont.

2.2.1.49.2 Ce qui change

- pont/pont_dsp.py lance pont.dsp.main (nouveau sous-paquet pont/dsp/). C'est deja le PONT_PY de run.sh en MODE=dsp, donc aussi celui de start-direct.sh –dsp. Il refuse de demarrer sans –dsp-port. pont/**init**.py et pont.py (grgsm) ne l'importent pas. La seule retouche du code partage est l'extraction de methode Uplink._route_sdcch, sans effet : un harnais qui rejoue SABM, MEAS, UA SAPI3, ASSIGNMENT COMPLETE, arm et prove donne un resultat identique octet pour octet avant et apres.
- TchDsp : l'ecriture dans tch_cfg n'est plus qu'une ANNONCE (meme format, meme moment). abandon() ecrit seq+1 avec tn=0 : retour au SDCCH, Kc garde.
- montant.c : scruter_tch ne fait plus que memoriser l'annonce. La nouvelle fonction suivre_tache_tch lit d_task_d dans la page W fraiche. Sur TCHT, TCHA ou TCHD avec une annonce, le BSP bascule sur le TS du TCH. Sur ALLC (24, la tache de lecture de bloc SDCCH/SACCH, prim_rx_nb.c:200 ; pas DDL/ADL comme je le croyais), il revient au SDCCH memorise par scruter_dcch. Log : « [montant] TCH : le firmware poste la tache 13 a fn=…, BSP bascule sur TS2 » et « le firmware est revenu sur le SDCCH (tache ALLC) ». MONTANT_TCH_TACHE=0 retablit l'armement a l'annonce.
- Nouvelle sonde a_fd (NDB+0x21A), active seulement sur TCH : « [a_fd] fn= fn%13= etat= BLUD=1 FIRE= d_tch_mode= L2=… ». MONTANT_AFD=0 la coupe.
- TrxDsp : le TS du TCH est dechiffre des l'annonce, toujours sous cipher.dl_active.
- UplinkDsp : un bloc sdcch_ul n'est jamais une FACCH. S'il arrive pendant un TCH annonce, il declenche abandon() quand c'est une ASSIGNMENT FAILURE ou quand le mobile etait deja passe sur le TCH. _poll_release ne lache plus le Kc : il tombe a l'IMMEDIATE ASSIGNMENT suivante. PONT_KC_RETENTION=1 est pose par pont_dsp.py.
- FeederDsp et DownlinkDsp : plus de GSMTAP 4730/4731 ni de /dev/shm/calypso_tch_dl (lus seulement par la L1 gr-gsm). Le tap 4729 reste.
- run.sh : calypso_tch_cfg est efface avant l'etape 1 et, en MODE=dsp seulement, a l'arret. c54x_exe a ete reconstruit.
- build-debs.sh embarque pont/dsp/.

2.2.1.49.3 A mesurer au prochain run (banc non relance par cette etape)

1. dsp.log : « TCH … annonce par le pont », puis « le firmware poste la tache 13|14 » plus tard, jamais avant. Entre les deux, les a_cd SDCCH doivent rester ok.
2. mobile.log : ASSIGNMENT COMMAND recu a chaque appel, plus de rafale de 80-110 bit errors apres l'annonce. /dev/shm/calypso_bsp_dedie doit passer de tn=1 a tn=2 seulement a la tache TCH.
3. FACCH DL sur TCH (T200 de l'appel 1) : si les lignes [a_fd] BLUD=1 arrivent, la plomberie est bonne. Si a_fd reste muet alors que le pont decode des FACCH, il faut chercher dans la ROM ou le coeur (SP-CORRUPT pc=0x0000 au tick 10253, watchpoint calypso_c54x.c:6226-6240).
4. ASSIGNMENT FAILURE provoquee : « revenu sur le SDCCH », « TCH TN=2 abandonne » cote pont, SDCCH de nouveau decode par le mobile.
5. Fin d'appel : pas de « Kc lache » a la liberation, plus d'echecs TS1/8 cote pont apres. Au RACH suivant : « Kc lache (IMMEDIATE ASSIGNMENT) ».
6. Non-regression grgsm : MODE=grgsm, un appel MO, memes messages qu'avant.

2.2.1.50 2026-09-23 18:45 — SACCH en TCH : depot apres l'IDLE, enregistreur, et le coupable MVKD/MVDK

Journal reconstitue des commits de l'apres-midi (bbc66c4 16:24, d474b76 17:40, 05388d8 18:42, et qosmo f5c8110 16:24, 7f27ab7 17:40, 9e61950 18:42). Banc non relance par cette mise a jour de la doc.

2.2.1.50.1 1. Le burst TCH depose trop tot (bbc66c4)

- Pour une tache TCHA (SACCH/TF), la ROM arme la fenetre de N+1 PUIS demodule, au debut de N+1, le burst SACCH de N laisse en 0x0cce. Le pont deposait le burst des l'armement et l'ecrasait : a_cd FIRE KO a chaque bloc, 113-118 bits faux, LOS au 32e bloc.
- `src/pont.c` : sur le TCH, la phase A va jusqu'a l'IDLE avant le depot (budget/2 au plus). Reproducteur tch_rejeu : SACCH FIRE=0 « 07 00 03 », FACCH 10/10 bonnes contre 6/12. `PONT_TCH_DEPOT_IDLE=0` = ancien depot ; trace `[depot_tch]` (30 premieres trames). Hors TCH, arret a l'armement inchange.
- `montant.c` : le tap QEMU (calypso_dcch_tap.c, qosmo f5c8110) annonce maintenant le TCH (genre 2 = TCH/F, 3 = TCH/H). C'est un simple constat : le SDCCH memorise n'est PAS ecrase, c'est lui que l'ASSIGNMENT FAILURE retrouve.
- `INSNS` 60000 -> 80000 dans run.sh : 60000 debordait en TCH (jusqu'a 87000 insn/trame).

2.2.1.50.2 2. Enregistrer le banc, le rejouer ailleurs (d474b76)

- `c54x_exe` ecrit, sur tout canal dedie, les ecritures ARM dans l'API RAM (par difference, avant le TICK et entre phase A et GO), les TICK, et les livraisons d'I/Q du BSP dans `/dev/shm/calypso_rejeu_tch.bin` (60000 livraisons au plus ; `CALYPSO_REJEU_ENREG=0` coupe).
- `tools/rejeu_banc [fichier] [ticks]` rejoue le tout dans l'ordre de pont.c et imprime chaque a_cd / a_fd. `REJEU_SANS_D=1` garde l'etat du boot local.
- `tools/sacch_tf_decode [fichier] [TN] [Kc]` decode hors DSP la SACCH/TF enregistree par le BSP (`/dev/shm/calypso_sacch_tf.bin`) avec le Kc de `calypso_kc_l1`. Mesure de la session : 63/63 blocs, 0 erreur. Les bursts que recoit le BSP sont bons ; le defaut est apres lui. (Chiffre de la note de session, non verifiable dans le code.)
- Le rejeu reproduit l'echec du banc. Cause : `data[0x3d89]`, pointeur du tampon SACCH (normalement 0x4dxx), vaut deja 0x08xx au debut du TCH ; la copie de 28 mots depuis 0x0cce ecrase alors les pages W/R et le NDB, dont `d_debug_ptr` (DSP perdu).
- L'ecrivain : la routine de mesure de puissance (autour de 0x76c0), a cause du coeur. MVKD (0x70) / MVDK (0x71) en adressage long lisaient dmad en pc+1 et lk en pc+2 ; le bon ordre est lk puis dmad (comme PORTW, ST et binutils). `70f8 0012 0014` faisait AR4 <- AR2 au lieu de AR2 <- AR4, et la PM ecrivait en 0x3d0b..0x3d89. Corrige dans qosmo `c54x_exec.c` par 9e61950 (18:42, livre avec 05388d8 ; `CALYPSO_MVKD_DMAD_AVANT=1` = ancien ordre, A/B). Garde permanente `[garde-3d89]` dans `c54x_mem.c` (7f27ab7) : les 12 premieres ecritures de 0x3d89 hors 0x4d00..0x4dff, avec PC, DP, ST0/ST1, SP, AR2, BK.
- A5 : `calypso_a5.c` (qosmo 7f27ab7), le coprocesseur XIO sur les ports 0x2800..0x2818, compile dans c54x_exe (Makefile, d474b76). `CALYPSO_A5=0` le coupe.

2.2.1.50.3 3. Le temps d'une trame (05388d8)

- `PONT_DONE_TOT` (defaut 1) : DONE rendu des le burst depose, le DSP finit la trame pendant que QEMU repart. `0` = ancien ordre.
- `[chrono]` dans dsp.log toutes les 1000 trames : `qemu | A | go | B | apres  DONE | trame` en ms, contre 4,62 ms de temps reel.
- run.sh pose `CALYPSO_PONT_RETRY_DIV=64` sur QEMU : relance vers le DSP toutes les trame/64 (0,07 ms) au lieu de trame/16 (0,29 ms), latence payee deux fois par trame (DONE de la phase A, GO).

2.2.1.50.4 4. La parole montante TI -> FR (05388d8)

mobile_pont.cfg dit `io-tch-format ti` : gapk rend la voix au format du DSP TI, le firmware la copie dans a_du, et le pont la passait a `gsm0503_tch_fr_encode(..., net_order=1)`, qui attend du FR TS 101 318. La BTS recevait une parole melangee, l'echo la renvoyait en bruit sature. `montant.c` convertit (repris de fmt_ti.c puis fmt_gsm.c) ; `MONTANT_PAROLE_TI=0` = passage brut.

2.2.1.50.5 5. Sondes et outillage (05388d8, sauf mention)

- `[a5-arm]` : chaque changement de d_a5mode ou du Kc pose par l'ARM.
- `[d_fn]` (bbc66c4) : la position que le firmware donne au DSP sur TCHA.
- run.sh : gdbstub QEMU `tcp:127.0.0.1:1234` et console `telnet 0 44444` (`GDB=0` coupe) ; `ASSEMBLY_LOGS=1` = trace asm ARM dans qemu-asm.log.
- Makefile : cible `.PHONY` et dependance aux en-tetes (d474b76), `-O3 -march=native` (05388d8). `make` reconstruit c54x_exe a chaque appel ; plus besoin de `make clean`.

2.2.1.50.6 A mesurer

1. Correctif MVKD/MVDK sur le banc : plus aucune ligne `[garde-3d89]`, a_cd FIRE=0 sur la SACCH/TF, plus de LOS en appel, et l'appel suivant ne tombe plus sur une connexion SCCP restee ouverte.
2. SABM repetes / UA perdu (« SABM frame with information not allowed » au BSC, 19:06) : cote osmo-operator, `pont/dsp/clock.py` mesurait l'avance de la BTS contre l'horloge reelle ; corrige a 20:14 (boucle fermee sur DSP + avance, `PONT_AVANCE_MIN` 4 -> 10), apres le dernier commit de c54x_exe. Lire ensemble « marge DL reelle … moy » dans pont.log (doit rester ~>= 10) et la colonne B de `[chrono]` (B » 1 ms = le DSP attend la BTS).
3. Chiffre de vitesse du coeur avec les sondes coupees, a remesurer avant de l'ecrire dans le README.

2.2.1.51 2026-09-23 20:40 — Runs du banc DSP de 20:22 et 20:32 : tout passe a 20:22, mais chaque trame de parole est BFI

Sources : archives `/tmp/c54x-pont/archives/20260923-202448` (run de 20:22) et `20260923-203413` (run de 20:32), journaux osmocom du run de 20:22 (20:22:09 -> 20:24:56 ; pas de journaux reseau pour 20:32). Mobile DSP = MS 1, MSISDN 100101 ; l'autre mobile = 100102. Banc relance par l'utilisateur, pas par cette mise a jour.

2.2.1.51.1 Ce qui marche (constate, run de 20:22)

- LU : IMMEDIATE ASSIGNMENT 20:22:41, U1_UPDATED 20:22:45, liberation 20:22:47.
- Appel MO 100101 -> 600 (echo Asterisk) : ASSIGNMENT COMPLETE 20:22:54 (TCH/F TN=2, bascule du BSP par la tache firmware), ACTIVE 20:22:55, DISCONNECT 20:23:27, TCH ferme 20:23:28. STATS du pont a 20:23:30 (cumuls de la session) : TCH dl=1607 ul=1601 perdus=11, FACCH ul=14, SACCH ul=87. Parole audible dans les deux sens ; GAPK monte la chaine au decrochage (FR, ti-fr).
- SMS : 100102 -> 100101 (MT 20:23:53-55) et 100101 -> 100102 (MO 20:24:05, SAPI 3 sur DCCH), CP-ACK et RP-ACK recus.
- Appel MT 100102 -> 100101 (osmo-sip-connector) : ASSIGNMENT COMPLETE 20:24:25, ACTIVE 20:24:28, release normal, TCH ferme 20:24:34.
- A5/1 : « chiffrement descendant confirme par la BTS » a 20:22:45, 20:22:53, 20:23:53, 20:24:04, 20:24:24 (LU, appel 600, SMS MT, SMS MO, appel MT).
- Correctif MVKD/MVDK (A mesurer 1 de l'entree de 18:45) : sur ce run, SACCH/TF `a_cd` KO a fn=4746 et 4954 (bascule), puis ok jusqu'au dernier bloc journalise, fn=9426 (ok 9 -> 51, ko fige a 9) ; cote mobile, aucun bloc SACCH jete entre la bascule (20:22:55) et la liberation (20:23:28). Aucune LOS, aucune ligne `[garde-3d89]`, et l'appel MT qui suit l'appel 600 aboutit. Confirme pour 20:22 (voir la LOS de 20:32 plus bas).
- clock.py (A mesurer 2) : aucune ligne SABM dans les journaux osmocom. Marge DL reelle min +0 (20:22:47), +9 (20:22:57, 20:23:07), puis +13 a +15, moyenne 22.3 a 38.9. L'avance visee monte a 40 a 20:22:37 et 20:22:56 (38 a 20:23:02). BSC : aucune ERROR INDICATION « SABM frame with information not allowed ».
- BSP dedie : `stockes=1751 joues=1752 manques=0 perdues=0 recales=0 silences=0`.
- Temps reel : 29 513 trames, un seul tick saute, au boot (fn=0).

2.2.1.51.2 Anomalies ouvertes, par ordre d'importance

1. **B_BFI sur toute la parole descendante.** Run de 20:22 : les 40 etats `[a_dd]` (21 x c214, 18 x c204, 1 x 8084) ont le bit 2 = B_BFI (l1_environment.h:272) ; le `ko=0` d'alors comptait 0x0040 = B_FIRE1, sans objet sur la parole. Sonde etendue (`src/montant.c sonde_add` : `BFI=`, `err=` a_dd_0[2], compteur `bfi=` ; non commitee), passee au run de 20:32 : `vues=2200 ko=376 bfi=2200`. Sur les 42 lignes : 19 x c214 err=0 (19 des 20 premieres, fn 5839-5921, juste apres la bascule ; la 18e, fn=5912, est le 8084), 17 x c204 err=15..80, 5 x 80c4 err=81..93, 1 x 8084 err=58. On entend quand meme : prim_tch.c:327 ne teste que B_BLUD et ne remonte pas le BFI, GAPK decode les 33 octets tels quels. Ce n'est donc pas « bits bons, BFI faux » : la ROM compte 15 a 93 erreurs par trame. Reste a separer signal (BSP, IQ, egalisation) et coeur C54x (Viterbi, recomptage) : comparer bit a bit les 33 octets livres par la ROM aux trames de la BTS (RTP du MGW ou pont). Le `ko` (B_FIRE1, 376/2200) n'a pas de sens defini sur la parole.
2. **LOS en TCH au run de 20:32.** Appel MO ACTIVE 20:32:30 ; SACCH/TF `a_cd` ok a fn=5991 puis FIRE KO a chaque bloc des fn=6095 (ko 8 -> 40) ; « LOSS counter for ACCH » descend de 31 a 0 et « LOS during dedicated mode » a 20:32:45. Aucune ligne `[garde-3d89]` : ce n'est pas l'ecrasement du pointeur 0x3d89 corrige. Appel suivant (20:32:57) : IMMEDIATE ASSIGNMENT puis T3230 a 20:33:13. Troisieme (20:33:35) : ASSIGNMENT COMPLETE 20:33:38, ACTIVE 20:33:39, DISCONNECT 20:34:06, SACCH/TF ok (ok 14 -> 54, ko fige a 46). Correlation : `err` de la parole 63 a 93 sur l'appel en LOS a partir de fn=6273 (jusqu'a 9306 ; ses 20 premieres trames, fn 5839-5921, a 0 sauf une a 58), 15 a 38 sur le troisieme (fn 21248-26881). Non localise.
3. **SDCCH/8 descendant** (run de 20:22) : 30 « Dropping frame with N bit errors » (l1ctl.c:278, toute trame a fire_crc >= 2), dont 18 precedees de « LOSS counter for ACCH » (l1ctl.c:268, SACCH seulement). Sur SDCCH/8 : 27 trames jetees (84 a 114 bit errors), 4 a 7 par session dediee, 7 sur le LU entre 20:22:42 et 20:22:47 ; 15 SACCH, 12 du canal principal. Sur TCH/F, une perte SACCH a la bascule (20:22:55, 54 bit errors) et une a chaque liberation (20:23:28, 20:24:33). Suspect : la table 45.002 du BSP pour la SACCH/8 (alternance sur 102 trames), mais le canal principal est touche aussi. Mesure a faire : fn % 102 des blocs jetes contre ceux acceptes.

4. **Marge temps reel en TCH** : `[chrono]` fn 5997-11997 (20:22) = A 0.33 + go 0.40 + B 0.16 + apres DONE 3.37-3.68 ms ; qemu 0.04-0.32 ms. Meme profil a 20:32 (fn 6997-8997, 21997-26997). Hors TCH, apres DONE 0.07-0.95 ms. Tenu, sans reserve.
5. UI SAPI 0 sur le TCH : BSC 20:22:54 lchan(0-0-2-TCH_F-0){WAIT_RLL_RTP_ESTABLISH} « SAPI=0 UNIT DATA INDICATION: unimplemented Abis RLL message type », juste apres l'ASSIGNMENT COMPLETE. Pas la trame de bourrage (UI de longueur 0, jetee par libosmocore). Hypothese : un MEASUREMENT REPORT sur le lien principal. Non bloquant.
6. Mineures : MM_EVENT_NO_CELL_FOUND transitoire a 20:24:08 apres le SMS sortant ; MSC 20:24:26 « Duplicate DTAP » sur la reponse au paging, sans consequence.

2.2.1.51.3 Ce qui n'est PAS une anomalie

- Les echecs CRC du moniteur TCH descendant du pont : meme profil sur les deux appels, la BTS n'a rien a mettre sur le TCH tant que le RTP ne coule pas. Decodage du pont, independant du DSP.
- Le `ko` de `[a_dd]` sur la parole (B_FIRE1), voir 1.
- Le tick saute au boot (fn=0), et l'arret de 20:24:52 (SIGINT volontaire).

2.2.1.51.4 A mesurer

1. B_BFI : comparaison bit a bit ROM / BTS sur un appel sain, puis rejeu hors banc (`tools/rejeu_banc`) pour isoler le coeur.
2. LOS de 20:32 : rejouer l'enregistrement TCH de cet appel si `/dev/shm/calypso_rejeu_tch.bin` le contient encore ; regarder ce qui change a fn=6095.
3. SACCH/8 : fn % 102 des blocs jetes.

2.2.1.52 2026-09-29 23:30 — Le verrou SB, mesure et correctif : le SCH n'etait jamais cadre pour la tache SB

Run dsp de 22:38 (banc-max), 600 tentatives SB dans dsp.log, 540 FBSB_REQ dans osmocon.log : 577 FB trouves, 130 SB. Cote mobile, chaque SB manquee est un « FBSB RESP: result=255 », une perte de cellule, 10 a 30 s de recherche PLMN ; l'appel de 23:06:35 est tombe dedans (« LOS during RACH request »), les SMS et l'USSD passent quand ils tombent dans une fenetre « normal service ».

2.2.1.52.1 Ce que les traces etablissent

1. Le TOA du FB est IDENTIQUE a chaque tentative (8743, ou 9995 = une trame plus loin) : le cadencement est deterministe.
2. La tache SB est postee a la bonne phase de multitrame dans 94 % des cas (`[cmd] tache SB postee`, fn mod 51 = 6/17/27/37/47, toujours la meme famille).
3. `[a_sch]` : a CHAQUE paire de tentatives, la page 0 recoit le meme resultat `8100 0016 f85d 01fb` (CRC faux, contenu constant = tampon sans burst) ; la page 1 recoit un contenu variable et decode 131 fois sur 600 (`8000 xxxx  001c yyyy`, BSIC 7).
4. `[sbwin]` : 100 % des SCH sont livres avec `one_shot=0 nwin=0 marge=0`, donc a l'offset zero d'une trame de 1250 echantillons suivie de sept intervalles de bourrage – le cadrage de la recherche FB, pas celui d'une fenetre SB (burst a 23 symboles, tpu_window.c L1_SB_MARGIN_Q).

Donc : le temps est juste, c'est le CONTENU que la ROM lit qui varie – residu du RIF (4 mots par trame de recherche FB) et instant d'armement de la DMA par rapport au depot. Quand la DMA n'est pas armee au depot, calypso_rif_rx_burst() jette le burst (« n_muets ») et la tache SB lit du perime : la page 0.

2.2.1.52.2 Correctif (calypso_bsp.c, bsp_ts0_livrer ; pont.c)

Derriere CALYPSO_BSP_SB_FENETRE (1 par defaut) : * un SCH livre alors que la ROM a programme une page de tache (ALGTH/2 >= 150 echantillons) sans ONE_SHOT est cadre comme une fenetre SB : residu RIF vide, CALYPSO_BSP_SB_MARGE (21) echantillons de silence en tete, bloc de DEUX pages exactement (la pompe ne transfere qu'a deux pages pleines), sans bourrage ; * DMA non armee au depot : le bloc est garde et relivre des l'armement (calypso_bsp_sb_retenter(), appelee dans la boucle de pompe de pont.c et au tick suivant) ; * `[sbwin]` dit armee/one_shot/page_prog/rif_avant et l'action ; `[a_sch]` porte le TOA, PM, SNR de la page R (plafond 600). Un SB bien pose lit TOA=23. La recherche FB (page de 48 echantillons) et les

fenetres ONE_SHOT ne changent pas. Non mesure : a lire sur le prochain run dans `[sbwin]` (page_prog reel de la tache SB) et `[a_sch]` (TOA, taux de CRC_OK sur la page 0 comme sur la 1).

A cote, banc-max : 87-ussd.sh est optionnel (un echec USSD ne saute plus appel et voix) ; 99-couverture.sh lit ses champs sur « ; » (les motifs « | » cassaient SMS, SDCCH descendant, SACCH en TCH, retour BSP), calcule son bilan hors du sous-shell (le verdict disait 0/0/0/0) et cherche des motifs que le mobile ecrit vraiment (MMSMS_EST_REQ, « new state dedicated -> release pending »).

2.2.1.53 2026-09-29 23:45 — Mesure du correctif SB (run de 23:33) : la fenetre SB existe, c'est l'ordre qui manque ; v2

`[sbwin]` avec les nouveaux champs :

```
[sbwin] SB #7 fn=327 p51=21 : armee=1 one_shot=1 page_prog=191 ... -> FENETRE SB (one-shot)
[a_sch] fn=994 page=1 : ... -> 8000 0720 001c 003d  TOA=24 PM=5516 SNR=16384 CRC_OK
[sbwin] SB #8 fn=337 p51=31 : armee=0 one_shot=0 page_prog=191 ... -> DMA NON ARMEE
[a_sch] page 0 : 8100 0016 f85d 01fb  TOA=0 PM=5521 SNR=12   (fenetre VIDE)
[a_sch] page 1 : 8100 xxxx xxxx xxxx  TOA=24 PM=5516 SNR=16384 crc_ko  (burst de la trame SUIVANTE)
```

1. **La ROM arme bien une fenetre SB one-shot de 191 echantillons (ALGTH 764).** Le MAILBOX du 22/09 (« la fenetre 382 mots n'existe jamais ») mesurait la longueur au moment du depot, pas celle de la tache SB. Quand elle est armee AVANT le depot, tout est natif : burst a 21, TOA 24, CRC OK.
2. Le defaut est un defaut d'ORDRE dans le tick : la fenetre est le plus souvent armee APRES le depot du SCH ; calypso_rif_rx_burst() jette le burst (« n_muets »), la page 0 lit une fenetre vide (SNR=12, resultat constant), la page 1 lit le burst de la trame suivante a TOA 24 (SNR sature, CRC faux). Ce n'est ni le residu du RIF ni une « double page » (hypotheses du premier jet).
3. **Le premier jet regressait** : la relivraison se faisait au premier armement venu, y compris les fenetres NB de la lecture BCCH (`RELIVRE ... one_shot=0 page=151 marge=3`) ; le SCH remplacait le burst BCCH, plus aucune SI, mobile en « C6 any cell selection », pytest mobile/reseau en echec.

v2 (calypso_bsp.c) : le SCH depose sans DMA armee est garde UN tick et livre uniquement dans une fenetre SB one-shot (>= 190), cadre a 21, avec recalage de l'offset ARM-tick sur le tick de livraison (bsp_ts0_service saute alors sa trame, rejouee au tick suivant : flux contigu). Le chemin « deux pages » est retire. A lire au prochain run : `RELIVRE au tick`, `CRC_OK` sur la page 0 aussi, et le ratio `=> SB / L1CTL_FBSB_REQ` d'osmocon.log (24/90 a 23:37).

2.2.1.54 2026-09-29 23:50 — v2 mesuree (run de 23:41) : le SCH part dans la fenetre PM ; v3 memorise toujours le dernier SCH

- Mobile campe (C3, normal service) : la regression de la v1 est levee.
- Mais SB 29/118 (osmocon), `[a_sch]` page 0 : 0 CRC_OK sur 215 (fenetre VIDE, TOA=0 SNR=12), page 1 : 29 CRC_OK sur 214 (burst a TOA 24, le SCH une fois sur sept, sinon la trame suivante). Une seule relivraison sur tout le run.
- Cause : au depot du SCH la DMA est le plus souvent ARMEE, mais pour autre chose – fenetre PM de 64 echantillons (`armee=1 one_shot=1 page_prog=64`, SB #2010-2090 a p51=31) : le SCH y est consomme, puis la fenetre SB s'arme et ne trouve rien. La condition « DMA non armee » de la v2 ne voyait pas ce cas.
- Ma trace `[sbwin]` n'echantillonnait qu'un SCH sur dix apres le 200e : le depot exact autour des tentatives etait invisible.

v3 : le dernier SCH depose est TOUJOURS memorise (fn, tick, bits, livre-en- fenetre-SB ou non) ; si une fenetre SB one-shot (>= 190) s'arme dans le tick ou le suivant sans qu'il y soit alle, il y est livre (residu RIF vide, marge 21, offset ARM-tick recale). Trace `[sbwin]` complete (avec le tick) pendant 60 ticks apres chaque detection FB et 8 ticks apres chaque tache SB postee (calypso_bsp_sb_trace(), appelee par pont.c). A lire au prochain run : `RELIVRE au tick`, `CRC_OK` sur la page 0, ratio `=> SB/FBSB_REQ`.

2.2.1.55 2026-09-30 00:15 — La fenetre SB etait bonne ; c'est la demodulation du SCH qui basculait. Mesure en rejeu, reglages portes au banc

Trace complete (v3, `[sbwin]` a chaque SCH pendant 60 ticks apres un FB) : * tentative 1 (page 0) tombe TOUJOURS sur la trame FCCH (p51 = SCH - 1, burst nul) : fenetre vide normale, pas un defaut ; * tentative 2 tombe sur le SCH, livre NATIVEMENT en fenetre SB one-shot 191 au bon TOA (23-24) ; aucune relivraison n'est necessaire (v2/v3 inutiles ici, gardees, inertes) ; * et pourtant : 129 fenetres SB natives comparees bit a bit au mot attendu (fn BTS connu) -> CRC OK sur la moitie ; sur la plupart des echecs seuls les bits 0-2 du mot sont faux (…1b/1f/1d au lieu de …1c). Les memes positions du burst basculent selon le contenu : le « fil du rasoir » deja vu sur les NB.

Mesure hors banc, `c54x_exe --rejouer` avec `REJEU_CONTINUER=1` (8000 trames, deterministe, 785 FB / 12000 trames), SCH decodes / SCH presentes :

reglage du SCH	decodes
tel quel (instant 0.5, sans elargir, sans bruit)	34 %
elargissement 0.15 / 0.3 / 0.5 / 1.0 / 1.5	50 / 60 / 63 / 66 / 66 %
+ instant 0.35 (0.3 : 71 %, 0.4 : 72 %, 0.5 : 66 %)	76 %
+ bruit sigma 3000 (300 : 77 %, 1000 : 79 %, 6000 : 73 %)	**82 %**
phase porteuse 10/22/45, marge 19-23	sans effet
amplitude 8000 / 15000 / 22000	0 FB / 2 SB / 82 %

Zero faux positif dans tous les cas. Porte au banc : `sb_moduler()` dans calypso_bsp.c (trois sites : bsp_ts0_livrer, sb_cadrer, chemin deliver), defauts CALYPSO_BSP_SB_SYM=1.0, CALYPSO_BSP_SB_DEC=0.35, CALYPSO_BSP_SB_NOISE=3000 (0 / 0.5 / 0 = avant). Cote rejeu : CELLULE_SB_SYM, CELLULE_SB_NOISE (0 par defaut, le rejeu reste la reference). Attendu au prochain run : `=> SB` / `L1CTL_FBSB_REQ` de ~25 % a ~80 %, donc des pertes de cellule rares, et un RACH d'appel qui ne tombe plus en LOS. Le reste (18 %) est encore dans la demod SB de la ROM emulee ; a chercher avec REJEU_DUMP_SOUPLES sur les SCH qui echouent a ces reglages.

2.2.1.56 2026-09-30 00:20 — Run banc-max de 00:07 (dsp, –restart) : l'appel s'etablit, 31 fonctions GERE sur 37

Avec sb_moduler() (instant 0.35, elargissement 1.0, bruit 3000) : SB 33/69 FBSB_REQ (etait 1/4), camp en 1 s, LU, SMS MO/MT, **appel CC ACTIVE en 1 s** (SETUP, CALL PROCEEDING, ASSIGNMENT COMMAND/COMPLETE, CONNECT), TCH/F decode (a_dd vues=3400 ko=0), FACCH/SACCH/parole montants vus par montant.c, liberation radio. Couverture : 31 gere, 1 degrade, 2 non gere, 3 non observe. Echelle 23/23, 2 echecs : * **voix** : le ton 1 kHz ne revient pas (raie 380 Hz, +0.3 dB) – coherent avec B_BFI=1 sur TOUTES les trames de parole descendantes (a_dd bfi=3400/3400, Viterbi ko=0) : le point ouvert n°1 du README, inchange. C'est le prochain verrou. * **ussd** : intermittent. Cause lue dans mobile.log : apres chaque liberation le mobile resynchronise (FB+SB), la synchro echoue encore une fois sur deux, le mobile passe 3-10 s « no cell », et le RACH lance pendant ce trou meurt en « LOS during RACH request » ; 87-ussd.sh enchainait *#101# 2 s apres* #100#. Correctif banc : attendre_service() (_lib.sh) avant chaque transaction (ussd, sms-mo, appel, voix). Le fond (18 % de SCH non decodes) reste cote ROM emulee. * garde MVKD/MVDK non observee = la garde n'a pas eu a jouer ; retour BSP sur le SDCCH non observe = la liberation s'est faite sur le TCH (FACCH), normal.

2.2.1.57 2026-09-30 00:45 — Voix : la boucle est coupee en deux hors banc, c'est le MONTANT qui casse ; le BFI n'y est pour rien

1. `tools/comparer_parole.py` sur l'appel de 00:14 : **1116 trames de parole descendantes sur 1117 identiques au bit pres** entre a_dd (ROM) et libosmocoding sur les bursts livres ; 12 FACCH identiques. La ROM rapporte pourtant ~27 erreurs canal par trame (a_dd[2]) la ou le Viterbi de reference en voit 0, et B_BFI=1 partout, y compris sur les trames a err=0. Le BFI ne vient donc pas du decodage (compteur ou metrique de qualite, a voir en rejeu avec bruit/amplitude) ; et le firmware l'ignore (prim_tch.c ne teste que B_BLUD) : il ne casse pas la voix.

2. Les memes trames a_dd, decodees par libgsm (RFC 3551, independant de la ROM et du mobile) : RMS constant ~3300, raie 200-600 Hz mouvante, 0 % d'energie a 1 kHz sur 23 s. Or a_dd == ce que la BTS a emis == l'echo d'Asterisk. **Ce qui est revenu du reseau etait deja du bruit : le montant envoie du bruit.** Le descendant est sain jusqu'a a_dd ; pas besoin du test Playback().
3. Verifie hors banc : parole_ti_vers_fr() est l'inverse exact de gapk (ti_fr_from_canon o gsm_to_canon, aller-retour identique) ; le firmware ecrit a_du avec dsp_memcpy_to_api(…, 1) = octet fort d'abord, comme prendre_ul. Restent : la capture GAPK (gsm_in -> gsm_mic.monitor), le firmware, le pont (codage, A5 montant).
4. Pour trancher au prochain appel : montant.c note chaque trame montante dans /dev/shm/calypso_add_ul.bin (type 1 brute TI, type 0 convertie FR) ; `tools/decoder_add.py ul` (et `dl`) decode avec libgsm et cherche le ton. Si le ton est dans add_ul : GAPK, firmware et montant.c sont bons, le defaut est au pont ou au-dela (capturer le RTP au MGW). S'il n'y est pas : capture GAPK ou firmware (a_du).

2.2.1.58 2026-09-30 00:50 — Pourquoi l'USSD (et tout RACH) passe une fois sur trois : la resynchro de gsm322 avant chaque RACH

Run de 00:36 : 10 CHANNEL REQUEST, 3 « LOS during RACH request » ; 7 liberations, 3 LOST_COVERAGE. Dans chaque cas rate, le RACH n'est pas emis : pas de ligne « RANDOM ACCESS », mais deux « FBSB RESP: result=255 » dans la seconde qui suit le CHANNEL REQUEST, puis le LOS.

Mecanisme, lu dans osmocom-bb gsm322.c : a chaque sortie de veille (GSM322_EVENT_LEAVE_IDLE -> gsm322_c_conn_mode_1/2), la selection de cellule refait une synchro FB+SB complete sur la cellule serveuse (gsm322_sync_to_cell(cs, NULL, 1)) AVANT le RACH, avec SYNC_RETRIES = 1 essai. Idem au retour en veille apres une liberation. Sur silicium c'est instantane et sur ; ici la ROM emulee decode le SB ~80 % du temps (apres sb_moduler), donc ~1 RACH sur 3 tombe en FBSB_ERR -> gsm48_rr_los -> « LOS during RACH », et ~1 liberation sur 2 laisse le mobile 3-10 s sans cellule. Le `ber= 47-49` des lignes MON et le `err~27`/BFI de la parole sont le meme artefact de compteur d'erreurs de la ROM sur nos echantillons, sans lien avec ces pertes.

Decision : NE PAS toucher a osmocom-bb (un patch de SYNC_RETRIES par variable d'environnement a ete essaye puis retire, aucun binaire installe). Le levier est du cote emulation : les 18 % de SCH que la ROM ne decode toujours pas (REJEU_DUMP_SOUPLES sur les echecs a SYM=1.0 DEC=0.35 NOISE=3000). Cote banc, attendre_service() + le rejeu du barreau (–essais 2) absorbent l'intermittence tant qu'elle dure ; le run de 00:36 l'a montre (USSD OK au 2e essai).

2.2.1.59 2026-09-30 01:30 — Les 20 % de SCH restants : ce qui a ete mesure, ce qui ne marche pas, ou chercher

Outils ajoutes au rejeu (tous a 0 par defaut, le rejeu reste la reference) : `SBresp` imprime le mot decode meme en CRC faux + TOA/PM/ANGLE/SNR ; CELLULE_SB_GARDE (garde continue au lieu du silence), CELLULE_SB_ISI (traine causale), CELLULE_SB_AMP (amplitude du seul SCH), CELLULE_SB_FC (GMSK sur-echantillonnee + Butterworth ordre 3 + decimation, gmsk_moduler_filtre()). Analyse : scratchpad analyse_sb.py (a recopier dans tools/ si utile).

Ce que les traces etablissent (rejeu SYM 1.0, DEC 0.35, bruit 3000, 12000 trames) : * les tentatives « fenetre vide » (TOA 43, SNR ~10) sont les tentatives 1, sur la trame FCCH : normal ; * les vrais rates (20 %) ont TOA 23, SNR sature ; le mot lu est faux sur ses ~14 premiers bits d'information et juste ensuite : c'est la MOITIE AVANT la sequence d'apprentissage qui est demodulee en garbage, la seconde tient ; * l'ensemble des rates est fixe par le CONTENU (31/33 communs entre deux instants d'echantillonnage, 24/33 avec un autre elargissement) ; * ce qui discrimine : les bits codes qui BORDENT la sequence (c38 juste avant : P(KO)=10 % si 0, 32 % si 1 ; c40 juste apres : 11 % / 31 %). L'ISI aux bords contamine l'estimation de canal de la ROM, qui equalise la premiere moitie a reculons depuis la sequence ; * l'angle (frequence estimee par la ROM, +70 Hz sur un signal a 0 Hz) bouge avec les reglages (+82 a +4 Hz) sans que le taux suive : pas la cause.

Ce qui ne change RIEN (a +-3 %) : amplitude du SCH (3000 a 30000), inversion d'un bit code quelconque, garde continue (zeros / aleatoire / uns), instant 0.2-0.8, elargissement 0.3-2.0, traine causale. Le filtre « realiste » (Butterworth 60-220 kHz) fait MOINS bien (44-65 %) : la ROM prefere une ISI forte et nette a trois coefficients. Meilleurs jeux sur 12000 trames : SYM 0.3 + ISI 0.3,0.1 -> 81 % ; SYM 1.0 + ISI 0.8,0.4 -> 80 % ; SYM 1.0 seul -> 77 %. Au banc (run de 00:36), 77 CRC_OK / 104 fenetres SB natives = 74 %.

Ou chercher ensuite : (a) la RE de l'estimation de canal SB de la ROM (quelle portion de la sequence elle correlle, sur quels lags, comment elle choisit sa fenetre : meme mecanique que le NB, 0x8551) — c'est la ou le contenu decide ; (b) cote emulation, une seconde chance : la tentative 1 du firmware tombe sur la trame FCCH et ne sert a rien ; le BSP a deja le SCH de la trame suivante dans son anneau et pourrait le livrer dans

cette fenetre-la aussi (l'offset ARM-tick se recale sur la trame ou le SB est reellement livre, comme le fait deja calypso_bsp_sb_retenter()) — deux essais par cycle au lieu d'un, si possible avec deux formes d'onde dont les ensembles de rates se recouvrent peu. REJEU_FN_DEBUT (demarrer a un fn realiste) a ete essaye et retire : la recherche FB du rejeu suppose fn depuis 0 (0 FB accepte a fn=100000).

2.2.1.60 2026-09-30 10:45 — « (MO) SMS rejected » : meme cause que l'USSD

Run de 10:33, essai 1 du SMS MO a 10:37:03 : CM SERVICE REQUEST, CHANNEL REQUEST, puis « LOS during RACH request », MM avorte (cause 47), le mobile affiche « SMS to 100102 failed: (MO) SMS rejected » ; rien n'atteint le MSC. Essai 2 a 10:38:30 : RACH, IMMEDIATE ASSIGNMENT, SMS remis. C'est la resynchro FB+SB de gsm322 avant chaque RACH (1 essai) qui echoue une fois sur cinq avec un SB a ~80 % : le SMS, l'USSD et l'appel y passent tous, le rejeu du barreau (–essais 2) l'absorbe. Le fond reste le taux SB (voir 01:30). 80-sms-mo.sh garde desormais la sortie VTY de chaque essai (»).

2.2.1.61 2026-09-30 10:55 — Voix : le ton EST dans le montant, 10 s trop tard ; le test regardait 12 s

Run de 10:33, calypso_add_ul.bin decode par tools/decoder_add.py ul : silence (RMS 13-90) pendant 13 s, puis le ton 1 kHz propre (RMS ~13000, 100 % de l'energie a 1000 Hz) de t=13 a t=17 s de l'appel. Recale sur l'heure murale (pont STATS fn=103984 <-> 10:42:23), le ton monte vers 10:42:21-10:42:28 alors qu'il a ete joue de 10:42:11 a 10:42:15 et que l'enregistrement du retour s'arretait a 10:42:20 : l'echo ne pouvait pas y etre. Le descendant vu dans la fenetre etait l'echo du silence tamponne, reencode. Donc : capture GAPK, firmware, a_du, conversion TI -> FR, pont, BTS, echo : tous BONS. Le defaut est une LATENCE d'environ 10 s sur le montant : les trames de parole montantes ont des trous (ecarts fn de 9 et 13 au lieu de 4-5, ~20 % des trames non produites), le codeur consomme l'audio moins vite que le temps reel et le tampon de capture grossit. La MGW le voit aussi (« input timestamp alignment error » a 10:42:27). 92-voix.sh : enregistrement 30 s (VOIX_REC_S), recherche du ton sur toute la duree, latence dans le verdict. A chercher ensuite : pourquoi ~20 % des trames TCH montantes manquent (prendre_ul/a_du vs cadence du firmware, B_PLAY_UL).

2.2.1.62 2026-09-30 11:20 — Voix : la boucle marche, echo compris ; le test injectait le ton pendant l'annonce d'Asterisk

Appel de 10:54 (run 10:45), trois chronologies recalees sur l'heure murale (pont STATS fn=107288 <-> 10:55:00) : * descendant a_dd : 10:54:47 -> 10:55:09, RMS ~3300, raies 200-400 Hz = la VOIX de l'annonce demo-echotest (22 s, /usr/share/asterisk/sounds/en/demo-echotest.gsm), jouee par l'extension 600 (extensions.conf:291-294 : Answer, Playback, Echo(), Playback) ; puis silence quand le montant est silencieux ; puis ce que le micro a capte a 10:55:13-14 revient a 10:55:11-14 + 1-2 s (DL 24 s, 27 s) ; * montant a_du : ton 1 kHz propre 10:55:01 -> 10:55:05 pour un paplay 10:54:59 -> 10:55:03 : latence montant ~2 s sur ce run (2000 trames UL pour 1880 attendues, aucun trou). Les 10 s de 10:42 venaient des ~20 % de trames UL manquantes de ce run-la (file de capture qui grossit) : intermittent, a garder a l'oeil, pas bloquant ; * le « ton retrouve a t=8 s, 19 % » du verdict etait la voix de l'annonce. Donc micro, GAPK, firmware, montant.c, pont, BTS, MGW, Asterisk, echo, ROM, mobile, sortie audio : TOUS bons. Le banc tirait 19 s trop tot. 92-voix.sh : attend le silence descendant (fin d'annonce, VOIX_ATTENTE_ANNONCE_S, 30 s max) avant la reference et le ton ; enregistrement 15 s (VOIX_REC_S). Le BFI a 100 % reste un artefact de compteur (a_dd bit-exact, firmware ne le lit pas).

2.2.1.63 2026-09-30 11:05 — SB : deux formes d'onde, deux tentatives ; la « seconde chance » est codee

Rejeu, 8000 trames, 209 SCH, neuf variantes de forme d'onde comparees par l'ensemble de leurs SCH rates (SBresp, snr > 1000) : * A (SYM 1.0, DEC 0.35, bruit 3000) : 42 rates (20 %) ; D (A + traine causale 0.8, 0.4) : 35 (17 %) ; E (SYM 0.3 + traine 0.3, 0.1) : 35 ; * la meme forme A avec une autre graine de bruit : 61 et 57 rates, dont seulement 19-22 communs avec A -> les rates sont en bonne partie MARGINAUX ; * rates communs des meilleures paires : A+D 15 (7 %), C+D 16, A3+C 17 ; triplets : 8-10 (4-5 %). Le firmware poste deja deux taches SB sur deux trames consecutives, mais la premiere tombe sur la trame FCCH. calypso_bsp.c (bsp_ts0_service) : quand la trame a jouer est un FCCH et que la DMA est armee en one-shot >= 190 (fenetre SB de la tentative 1), on livre le SCH de la trame suivante (deja dans l'anneau) en forme D ; au tick suivant on lit a_sch dans l'API RAM (B_BLUD sans CRC, T2 <= 25, T3' <= 4) : decode -> offset ARM-tick recale d'une trame (fn_sch - tick), flux contigu ; rate -> le SCH natif part en tentative 2 en forme A. Attendu : ~93 % de cycles FBSB reussis au lieu de ~80 %, donc des RACH qui ne tombent presque plus en LOS. CALYPSO_BSP_SB_DOUBLE=0 coupe. A lire au prochain run : lignes « [sbwin] SECONDE CHANCE … DECODE / rate », ratio « => SB » /

FBSB_REQ dans osmocon.log, « LOS during RACH » dans mobile.log. Non teste hors banc : le rejeu ne passe pas par bsp_ts0_service.

2.2.1.64 2026-09-30 11:40 — Seconde chance, run de 11:34 : elle se declenche (43 fois, 30 decodes en tentative 1) mais ne gagnait rien

osmocon : 31 SB / 55 FBSB_REQ, comme avant ; SB1 27, SB2 4. Cause lue dans dsp.log : bsp_ts0_livrer() recale l'offset ARM-tick des qu'un SCH part dans une fenetre SB ; la livraison de seconde chance (SCH fn+1 au tick de la trame FCCH fn) le faisait donc avancer d'une trame AVANT le verdict de la ROM. Quand la forme D ratait (13 fois), le tick suivant livrait fn+2 et la tentative 2 voyait un burst normal (a_sch 8100 0172 86be 0060, TOA 23 SNR 427 = « pas de burst ») : D remplacait A au lieu de s'y ajouter. Correctif : offset sauve et restaure autour de la livraison de seconde chance ; le recalage n'a lieu qu'au tick suivant sur CRC OK lu dans l'API RAM (deja code). Attendu : ~30 + 13 x 0.8 = ~40 SB sur 43 cycles avec seconde chance.

2.2.1.65 2026-09-30 11:55 — Run de 11:37 : plus aucun LOS pendant le RACH ; un appel sur trois perd la FACCH descendante ; garde-fou de temps dans le banc

- Seconde chance corrigee (offset restaure) : 34 declenchements, 28 decodes en tentative 1, 6 replis ; osmocon SB1 25 / SB2 6 ; TOUS les CHANNEL REQUEST du run (SMS, paging, USSD x2, appels) ont eu leur IMMEDIATE ASSIGNMENT, zero « LOS during RACH request ». Le trou de resynchro avant RACH est ferme en pratique.
- Appel de 11:42:46 rate : SETUP recu par le MSC, ASSIGNMENT COMMAND vers TCH TS2, le mobile bascule et envoie SABM sur la FACCH (ASSIGNMENT COMPLETE en attente de l'etablissement) ; SABM repete 5 fois, jamais de UA : MDL-Error T200 a 11:42:55, ASSIGNMENT FAILURE, retour SDCCH, puis LOS dedie a 11:43:10. La BTS a bien repondu (le moniteur du pont decode 4 FACCH descendantes = les UA) ; la ROM n'a leve aucun a_fd pendant l'appel, alors que la SACCH du meme TCH passait a chaque bloc (donc fn et COUNT A5 justes : l'horloge n'est pas en cause). Les deux appels suivants (11:48, 11:49) passent : intermittent, c'est la FACCH descendante sur TCH du 23/09 20:40. A chercher dans le decodage TCH de la ROM (a_fd / stealing flags), pas dans la synchro.
- banc-max.sh : chaque barreau tourne sous surveillance (MOD_TIMEOUT, defaut 240 s, verdict ECHEC « timeout » au-dela) ; 90-appel.sh borne sa boucle en secondes (CALL_MAX) et non en tours de VTY.

2.2.1.66 2026-09-30 12:05 — Seconde chance : desactivee par defaut ; les appels de plus de 30 s tombaient

Run de 11:37 (binaire 11:36, seconde chance active) : zero « LOS during RACH » sur tout le run (SMS, paging, USSD x2, appels : tous les CHANNEL REQUEST servis), mais les deux appels de la voix tombent a ~30 s en « LOS during dedicated mode » (compteur SACCH 31 -> 0 en 15 s) avec un TCH descendant qui se degrade progressivement (a_dd err 19 -> 65 -> 91, FIRE=1, ko 0 -> 370). Le pont, lui, decode SACCH et TCH descendants et garde une marge DL >= 11 trames. Tous les appels des runs SANS seconde chance (00:14, 10:54) tenaient 40 s et plus avec le compteur a 31. Mecanisme suspecte : chaque decode en tentative 1 avance d'une trame l'index que le DSP reclame a la BTS (offset ARM-tick -673 -> -638 sur le run, 35 fois) ; la marge du pont ne se refait pas (la BTS ne va pas plus vite que le temps reel) et la boucle d'horloge pousse la BTS (avance visee 14 -> 27). Non prouve : a confirmer par A/B (CALYPSO_BSP_SB_DOUBLE=1). En attendant : defaut 0, et jamais en canal dedie (g_dedie_tn > 0). Le taux SB revient a ~74-80 % par SCH et le RACH retombe a ~1 echec sur 5, absorbe par –essais 2 et attendre_service(). L'echec de l'appel de 11:42 (FACCH descendante non decodee par la ROM, SABM repete, UA emis par la BTS) est independant : intermittent, point du 23/09.

2.2.1.67 2026-09-30 12:45 — Le ton casse le son : IT trame perdue, compteur TDMA du firmware decale d'une trame ; correctif dans qosmo (INTH + pont QEMU)

Symptome (appels de 12:09, run 11:57, et de 12:26, manuel) : des que le ton est injecte dans gsm_mic, la parole descendante devient du bruit (a_dd err 20-30 -> 70-95, FIRE), la ROM leve des FACCH a faux (a_fd BLUD=1 FIRE=1 a chaque bloc), la SACCH/TF ne passe plus (LOSS 31 -> 0), LOS ~15 s plus tard. Le montant s'effondre en meme temps (a_du une trame sur trois). Le pont, lui, garde sa marge DL (min +13) et le DSP tient 4,62 ms/trame. Preuve : la trace [a5] passe de « ecart +1 » (COUNT = dernier depot + 1, regime de tous les appels reussis) a « ecart 0 » exactement au moment de la bascule (#20500 fn=53467 le 12:26 ; #30500 fn=153420 le 12:09, avec alternance 0/+1 pendant ~1000 trames puis 0 pour de bon). Ecart 0 = la ROM lit une page W dont a_a5fn vaut fn-1 : le firmware est UNE TRAME EN RETARD sur l'interface radio. Il n'y a pas de correction possible de son cote : osmocom-bb ne detecte pas une IT trame manquee, l1s.next_time avance

d'une unite par l1_sync(). Une IT trame perdue = glissement permanent jusqu'a la fin du canal. Comment on la perdait : calypso_trx.c leve l'IT trame TPU (IRQ 4) en impulsion de 1 ms (FRAME_IRQ_PULSE_NS) ; le firmware la configure sur FRONT (irq_config(IRQ_TPU_FRAME, 1, 1, 0)), mais le modele INTH (hw/intc/calypso_inth.c) effacait le bit IT_REG a la retombee de la ligne, comme une source de niveau. Si l'ARM etait dans un traitement de plus d'1 ms sous IRQ masquees – la reception UART/L1CTL des TRAFFIC_REQ, en rafales quand paplay alimente le puits nul –, l'IT etait perdue ; au bout de 256 attentes (74 ms, la creux de -12 trames que le pont voit a 12:10:07) le GO etait envoye « quand meme » et la cible d'EOI recalee : le glissement s'installait sans une ligne de plus (le message n'etait imprime qu'une fois par run). Correctif (qosmo, l'ARM et la ROM ne bougent pas) : * calypso_inth.c : une source configuree sur front (ILR bit 1) parmi 4/5/15 reste memorisee jusqu'a la lecture d'IRQ_NUM. Plus d'IT trame perdue. Accesseur calypso_inth_irq_pending(). * calypso_trx.c : au-dela de 256 attentes, on n'envoie plus le GO avec une page perimee ; on attend l1_sync jusqu'a 32 x 256 essais (~2,4 s) en servant les UART, avec trace « on attend au lieu de forcer le GO » puis « finie apres N attentes » (20 premieres). Le temps mural perdu se paye en trames BTS trop tardives chez le BSP (perte radio, que le firmware sait absorber), pas en glissement TDMA. Le forcage ne reste que pour un ARM mort (message aux 20 premieres puis toutes les 2170). A verifier au prochain run (redemarrage de la pile : barreau 1) : [a5] ecart +1 sur tout l'appel voix, qemu.log sans « GO envoye quand meme », et le ton qui revient (92-voix). Si des « on attend » apparaissent dans qemu.log, leur duree dit combien l'ARM est en retard a l'injection du ton. Note : le modele INTH decode ILR a l'envers du firmware (FIQ lu au bit 8, prio aux bits 0-4, alors que le firmware ecrit prio«2 | edge«1 | fiq) : l'IT trame est servie comme IRQ et non comme FIQ, donc elle ne preempte pas le traitement UART – c'est ce qui rend le retard possible. Pas touche : la comptabilite EOI (frame_eoi sur IRQ_NUM) en depend. A reprendre si le retard reste visible.

Reponses aux points de couverture demandes : * « retour BSP sur le SDCCH : NON OBSERVE » : normal. montant.c ne rend le SDCCH au BSP que si le firmware reposte une tache ALLC pendant un TCH, ce qui n'arrive que sur ASSIGNMENT FAILURE (retour a l'ancien canal, 04.08 3.4.3.3, vu une fois le 11:42). Un appel normal finit par CHANNEL RELEASE. Ligne de 99-couverture.sh annotee « attendu absent ». * « TCH/F descendant : qualite (B_BFI) NON GERE » : la sonde [a_dd] montre BFI=1 sur 100 % des trames, y compris err=0, et err ~28 en regime normal (~6 % de 456, soit un burst sur huit a moitie faux ou 3-4 bits en bord de chaque burst) alors que les 260 bits sont exacts. Le BFI est donc une metrique de qualite de la ROM sur des bursts synthetiques trop propres ou mal bordes, pas une erreur de donnees. Outillage pose pour l'etudier hors banc : tools/rejeu_banc (recompile, sans cible Makefile : commande dans README) imprime desormais a_dd (BFI, erreurs ; REJEU_PAROLE=1 par trame) ; l'enregistrement /dev/shm/calypso_rejeu_tch.bin peut se limiter au TCH (CALYPSO_REJEU_ENREG=tch) et son plafond se regler (CALYPSO_REJEU_ENREG_MAX) – le run de 11:57 avait rempli ses 60000 livraisons avec les SDCCH du debut (fn 1376..34942), aucun TCH dedans. Le rejeu de cet enregistrement donne 23 SACCH/8 toutes Fire KO avec err 79-100 la ou le banc les decodait : le rejeu n'est pas encore fidele sur le SDCCH (Kc / etat 'D' ?), a regarder avant de s'en servir pour le BFI.

2.2.1.68 2026-10-03 — Montant : la ROM code tout le SDCCH ; 4e bug du cœur ; bursts 0x3f8a branchés sur le pont ; chemin ROM/hôte

- Les 56 bits « non matérialisés » de la parité FIRE montante étaient faux, pas absents : la sortie ROM est un code convolutif auto-cohérent, la parité avait ses 24 bits hauts à 0. La routine FIRE (PROM0 0x9013) rend la bonne parité dans A ; le rangement `0x8bc6 f1b0 = OR A,-16,B` était volé par un faux décodage « F0Bx/F1Bx = RSBX/SSBX » (vrais : F4B0/F5B0). Corrigé dans qosmo (1e47cd72, gate CALYPSO_F0BX_SBIT=1 = ancien). Banc : camp→voix 25/25.
- Les 2 bits « faux » restants (burst 0, bits 112-113) : la sérialisation TX de la ROM (0x890e-0x891b) copie chaque trame le burst courant vers data[0x3f8a..0x3f91] puis met à 0 son mot 7 dans 0x4280. 0x3f8a = burst émis ; trace d'un bloc ff×23 : 4 bursts = gsm0503_xcch_encode bit à bit.
- Branché : c54x_exe 44b1ec8 (tx_rom_publier, /dev/shm/calypso_xcch_ul_rom, avec le flux A5 montant que la ROM XORe sur 0x3f8a en chiffré), pont 78c6de6 (UplinkDsp._bursts_xcch : redécode, défait le XOR, rechiffre au fn de l'air, repli hôte après 120 ms). Banc 17:11 : xCCH ul rom=134 hote=1, 25/25. Le premier essai sans flux A5 : rom=33 hote=79 (tout bloc chiffré refusé).
- Tests 77b0baf : section « chemin ROM / hôte » dans 99-couverture.sh. Run 17:11 : 6 ROM (FB, SCH, a_cd, a_fd, a_dd, A5 DL), 7 hôte (IMM ASS retouché, RACH, A5 UL, FACCH UL, SACCH TCH UL, parole UL, vocodage), 1 mixte (SDCCH UL, 1 bloc sur 135 codé hôte).
- Descendant vérifié (lecture des sources + journaux) : tout décodé par la ROM ; le BFI de la parole est posé à tort (err=0 partout, BFI=1 dès le 1er bloc incomplet) ; SACCH/8 DL ~80 % FIRE KO. Prochaine cible : l'instruction ROM qui pose B_BFI dans a_dd_0.

2.2.1.69 2026-10-04 — Le burst montant FINAL de la ROM : deux bugs du cœur (CMPR, LD src,ASM), le script TSP, RACH bit-exact, tuyau unique vers le pont

- **Le « magic 23 »** : `proof_toa_23.md` (anglais, signé par `sign.sh`) et son dossier de preuves. Le DSP rend la position du burst dans sa fenêtre (indice du retard retenu par le corrélateur, bord d'attaque de l'estimation de canal, PROM0 0x84a1-0x7cbf) ; les fenêtres 191 (SB) et 151 (NB) sont des immédiats de la ROM (`stm #191,ar3` 0xb21d, `stm #151,ar3` 0xb2b1) = 148 + 2 · marge − 3 avec les marges 23 et 3 de `tpu_window.c` ; le chemin FB soustrait 23 deux fois depuis `cb71b972` ; journaux silicium (liste, 2011 et 2024) : qbits = ((TOA−46) mod 1250)×4 (12/12), 1re SB après synchro FB à 27 ± 2 (n = 6). Désassembleur : binutils 2.21.1 `--target=tic54x-coff` (sources dans la couche containerd), dump public FreeCalypso 3606 = PROM0 local mot pour mot.
- **Deux bugs du cœur** (qosmo `c54x_exec.c`, gates `CALYPSO_CMPR_ANCIEN` / `CALYPSO_LDASM_ANCIEN`) : `CMPR CC,ARx` (0xF4A8 masque 0xFCF8, 78 sites) n'existait pas — l'émetteur de bursts `0x8608 cmpr eq,ar2 ; rc tc` rendait la main sur un TC périmé ; `LD src,ASM` (0xF482) ignorait son décalage — les 16 mots BULDATA sortaient à 0xff06. ISA inchangée (150 ok / 58 FAIL / 22 S sur 230) : ces 58 sont la file de travail avant tout nouveau tuyau.
- **Le script TSP** (`src/tsp_tx.c`) : chaque trame la ROM écrit en data[0x3cbb..] le script pour l'ABB ; marqueur 0x1c0a (TOGBR2), puis 16 mots BULDATA = (10 bits « 6) | 0x06 (émetteur 0x8605, tables DROM 0xa0c9 TSC / 0xa0d9 sync RACH / 0xa0dc queue ; `xorm 0xaa80` à 0xb773 = inversion I/Q). 160 bits = 8 de garde + le burst de 148 + garde. `dsp_tester rach` : la ROM exécute la tâche RACH, 36 bits codés = `gsm0503_rach_ext_encode` et access-burst 05.02 exact (5/5) ; `tx-sdcch` 24/24 (capture hl/hu du banc alignée sur pont.c) ; `tx-sdcch-a5` 24/24 : la ROM chiffre avec osmo_a5(kc, fn de la commande) = le fn du burst. `tools/rejeu_banc` (cible make) capture ces bursts sur l'enregistrement TCH de 15:47 : 180 NB, TSC 7, décalage 8.
- **Tuyau unique** : pont.c `tx_tsp_publier` publie chaque burst final (fn, type, TSC, 148 bits) dans `/dev/shm/calypso_tx_rom` (`MONTANT_ROM_TX=0` coupe) ; montant.c laisse `d_task_ra` à la ROM (`MONTANT_EFFACE_TASK_RA=1` = ancien) ; pont/dsp/uplink.py `RomTxRing` + `PONT_UL_RAW=1` (run.sh : `UL_RAW`, 1 par défaut) : la BTS reçoit les bursts de la ROM tels quels à son fn, RACH sur TS0, NB sur le TN du canal dédié, sans codage ni A5 hôte ; les chemins hôte tournent encore pour les journaux mais leurs bursts vont au puits. Non testé en bout en bout : à valider au prochain run (`UL_RAW=0` pour revenir). osmo-bts `trx_sched_ul_burst` ne rejette pas un burst montant sur son fn.
- **Firmware** : retour à `layer1.highram` d'origine (variante patchée supprimée ; B_BFI à surveiller).
- **Temps réel** (son coupé sous charge) : l'émulateur coûte ~23 ns/instruction, attentes en `poll()` ; run.sh lance la chaîne en `nice -10` (`RT_NICE`), `RT_FIFO`, `RT_CPUS`, `RT_RENICE_AUTRES`, `./run.sh --charge`.
- Banc : `DSP_TESTER_NOFORK=1` (gdb/gprof), captures génériques des anneaux TX et du script TSP ; `./dsp_tester --all` : 9 PASS (dont rach, tx-sdcch, tx-sdcch-a5), 4 FAIL (fb0, fb1, fbsb, nb).

2.3 /opt/GSM/c54x_exe/README.md

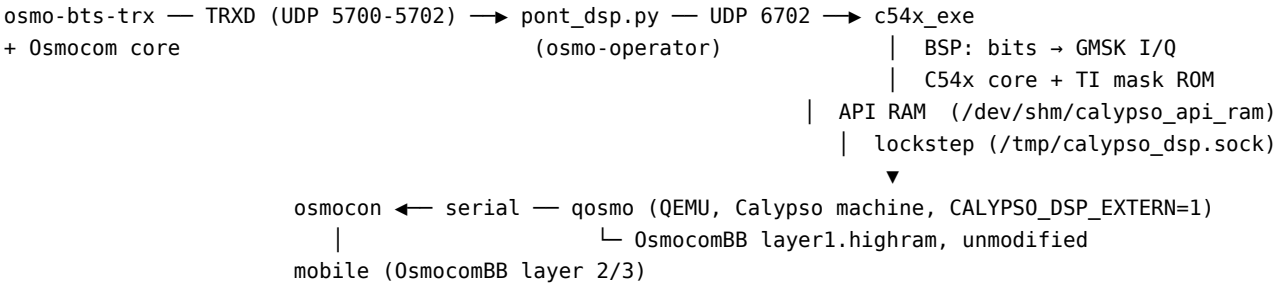
10031 octets, 174 lignes

2.3.1 c54x_exe — the Calypso DSP, run as a native process

`c54x_exe` runs the **original TI mask ROM of the TMS320C54x DSP** found in Calypso GSM basebands (the chips behind OsmocomBB's Motorola C1xx targets), on a C54x core emulator, as a plain Linux process. It can run alone, replaying recorded bursts in milliseconds, or serve as the DSP of a full phone: the unmodified OsmocomBB layer 1 firmware runs under QEMU and talks to this process through the real shared API RAM, frame by frame, exactly as the ARM talks to the DSP on silicon. No radio, no SDR, no license anywhere in the chain.

State on 2026-10-03 (runs recorded in `docs/README.fr.md` and `MAILBOX.md`): cell selection, location update, SMS in both directions, mobile-originated and mobile-terminated voice calls with A5/1 confirmed by the network, speech audible both ways. All downlink channel decoding is done by the ROM. Since 2026-10-03 the **uplink SDCCH/SACCH bursts sent to the BTS are the ROM's own coded bursts** (bit-exact with GSM 05.03 once four C54x core bugs were fixed). Open: the ROM flags every downlink speech frame as bad (BFI) although speech is intelligible, SACCH/8 downlink often fails its FIRE check, and the SB window is armed rarely enough that sync takes a few tries. The bench's coverage step prints, per signal-processing step, whether the run did it in the ROM or on the host (`tests/modules/99-couverture.sh`).

2.3.1.1 Architecture



- **Downlink**: bursts leave the BTS as bits, the bridge forwards them, the BSP turns them into GMSK samples, and the ROM does what it does on a phone: FCCH/SCH detection, BCCH, TCH/F decoding, A5.
- **ARM ↔ DSP**: the ROM's API RAM window is memory-mapped and shared with QEMU. Both sides advance one TDMA frame at a time over a Unix socket; the per-frame sequence mirrors `calypso_tdma_tick()` of the in-QEMU DSP, so any divergence is localised (`src/pont.c`).
- **Uplink**: the ARM posts its blocks in the API RAM; `src/montant.c` scans it once per frame and publishes RACH, SDCCH, SACCH, FACCH and speech (TI format → FR) to `/dev/shm`. **SDCCH and SACCH-on-SDCCH are channel-coded by the ROM**: `src/pont.c` (`tx_rom_publier`) picks up each burst the ROM emits in `data[0x3f8a]` and publishes the 4 bursts of a block to `/dev/shm/calypso_xcch_ul_rom`; the bridge sends them (after checking they decode to the block, undoing the ROM's A5 XOR and re-ciphering at the air frame number). RACH, FACCH, SACCH on TCH, uplink speech and uplink A5 are still done by the bridge. `MONTANT_ROM_UL=0` (here) or `PONT_UL_ROM=0` (bridge) go back to host coding.
- Every deliberate departure from silicon behaviour is an environment variable, listed at startup (`hacks_actifs()` in `src/pont.c`).

2.3.1.2 Building

The C54x core, the Calypso peripherals and the DSP glue live in qosmo (`hw/arm/calypso/l1-dsp/`) and are compiled from there — nothing is copied into this tree, on purpose.

```
# needs: qosmo checked out (default /opt/GSM/qosmo), libosmocore + libosmocoding, gcc
make                      # QOSMO=/path/to/qosmo make
./c54x_exe --trames 200   # the ROM alone, no ARM: does it boot, what does it write?
./c54x_exe --help
```

The ROM images (`rom/calypso_dsp.*.bin`) are dumps of the TI mask ROM at their silicon addresses (`src/main.c`, `ROMS[]`).

2.3.1.3 Running a phone

```
./run.sh              # DSP + QEMU + osmocon + mobile, in order; logs in /tmp/c54x-pont/
PONT=1 ./run.sh       # + the TRX bridge, against a running osmo-bts-trx / core network
./run.sh --status | --logs | --stop
```

Process-by-process launch, expected log lines and checks: `LAUNCH.md`. The bridge and the network side are in osmo-operator.

2.3.1.4 Benches and tools

what	where
Replay the ROM alone on a recorded FB/SB acquisition, deterministic, no QEMU	`./c54x_exe --rejouer`, `src/rejouer.c`
Replay a recorded dedicated-channel session (TCH) through the ROM	`tools/rejeu_banc.c`
C54x instruction-level tests (the bugs they caught are listed in the source)	`tools/isa_test.c`, `tools/isa_tests.txt`

what	where
Bit-by-bit comparison: what the ROM decodes vs. libosmocoding on the same bursts	`tools/comparer_parole.py`
CCH interleaving reference vectors	`tools/cch_ref/`
Trace levels of the core, `-v` … `-vvvvvv`	`src/verbosite.c`

2.3.1.5 Repository layout

```
src/        main.c (ROM loading, CLI), pont.c (ARM/DSP bridge, per-frame sequence),
            montant.c (uplink from API RAM), cellule.c (synthetic FCCH/SCH/BCCH),
            rejouer.c (replay), verbosite.c (trace levels)
rom/        TI mask ROM dumps
tools/      benches, test vectors, analysis scripts
docs/       README.fr.md — the detailed, dated engineering notes (French)
LAUNCH.md   process-by-process launch guide
MAILBOX.md  dated log of hypotheses, refutations and measurements
```

2.3.1.6 License

GPL-2.0-or-later, see `LICENSE`.

2.3.1.7 ROM and uplink probes

The DSP ROM is **not** in this repository. `rom/fetch-rom.sh` downloads the FreeCalypso dump (version 3606), converts it with `tools/dsp_txt2bin.py` into `calypso_dsp.{DROM,PDROM,PROM0..3}.bin` and checks `rom/SHA256SUMS.3606` (`--dest DIR` repeatable, `--version 3311` for the D-Sample dump). `calypso_dsp.Registers.bin` is a register snapshot, not ROM: it stays in the repository.

Two inert-by-default probes in `src/pont.c` (environment variables, `--arm` mode):

- `PONT_TX_SONDE=N` — for N frames where the ARM has posted `d_task_u`/`d_task_ra`, log which DSP data words the ROM wrote and dump the changed ranges to `/tmp/tx-sonde/`.
- `PONT_TX_INJECT=FILE` — after `montant.c` has published a real uplink block (so the link is untouched), overwrite `a_cu` with the next 23-byte block of FILE (one hex line each); the ROM output for each block lands in `/tmp/tx-sonde/inj_<n>.txt`. `PONT_TX_TRACE_BLOC=N` single-steps the ROM on injected block N and logs every write to the TX buffers with its PC (`/tmp/c54x-pont/trace-tx.txt`).
- `PONT_TX_OMBRE=1` — compares each real SDCCH block coded by the ROM with `gsm0503_xcch_encode` (`/tmp/tx-sonde/ombre.txt`, `reel.txt`).

Uplink coding, as measured on 2026-10-03 (`tests/rom-tx/README.md`): the ROM writes the interleaved block in `0x4280` (4 bursts × 8 words, circular, BK=32); FIRE parity is computed at PROM0 `0x9013`; at each transmit frame `0x890e-0x891b` copies the current burst to `data[0x3f8a..0x3f91]` (114 bits + hl/hu, MSB first), advances the pointer `data[0x3d9b]` and clears word 7 of that burst in `0x4280`; in ciphered mode the A5 keystream (`0x3f9b..`, `calypso_a5.c`) is XORed onto `0x3f8a` itself. Four C54x core bugs had to be fixed in qosmo `c54x_exec.c` for the output to match GSM 05.03 on all 456 bits: logical (not arithmetic) 40-bit shift in `AND/OR/XOR src,SHIFT,dst`, SXM-respecting `LD/ADD/SUB Smem,16` and `LDM`, and `F0Bx/F1Bx` decoded as `OR A,-n,dst` instead of RSBX/SSBX (gates `CALYPSO_LOGIC40`, `CALYPSO_LD16_SEXT`, `CALYPSO_F0BX_SBIT` = old behaviour).

`C54X_BIN=/path/to/c54x_exe ./run.sh` runs another build of the executable.

2.3.1.8 Noise injection (downlink)

Off by default. Two independent ways to degrade what the DSP receives, both driven from the environment of `run.sh` (and therefore of `osmo-operator/start-native.sh --dsp`):

- `BRUIT_SNR_DB=<dB>` alone: AWGN added by the C54x core itself to every downlink I/Q sample (`c54x_bsp_load()` in qosmo `calypso_c54x.c`); `run.sh` passes it to `c54x_exe` as `CALYPSO_BSP_SNR_DB` (and `BRUIT_GRAINE` as `CALYPSO_BSP_BRUIT_GRAINE`).

- `BRUIT_MODE=ber|souple|iq|relais`: `run.sh` inserts the UDP proxy `/opt/GSM/qosmo/tools/injecteur_bruit.py` (fallback: `tools/injecteur_bruit.py`, a link to it) as step **b**: `c54x_exe` listens on `BRUIT_PORT_DSP` (16702, via `CALYPSO_BSP_PORT`), the proxy takes `127.0.0.1:6702`, degrades the bridge's 8-byte-header + 148 hard-bit bursts and forwards them; anything the BSP sends back is relayed unchanged to the bridge. `ber` flips bits (`BRUIT_BER`, bursty with `BRUIT_RAFALES=<mean length in bits>`), `souple` adds AWGN at `BRUIT_SNR_DB` then hard-decides (the stream carries no soft values), `iq` (experimental, numpy, refused under `CALYPSO_BSP_STREAM=1`) sends GMSK I/Q + AWGN through the BSP's IQ_PASSTHROUGH path, `relais` changes nothing. `BRUIT_PERTE` erases bursts, `BRUIT_TN` restricts timeslots, `BRUIT_GRAINE` makes errors reproducible per (fn, tn). In `MODE=grgsm` the proxy sits between `pont.py` (`PONT_GSMTAP_PORT=14730`) and QEMU's GSMTAP port 4730 (`ber`/`relais` only).

```
BRUIT_MODE=ber BRUIT_BER=0.01 PONT=1 ./run.sh       # 1 % bit errors on every downlink burst
BRUIT_SNR_DB=12 PONT=1 ./run.sh                     # core AWGN at 12 dB, no proxy
./run.sh --step b                                   # the proxy alone; --status shows its last STATS line
```

Log: `$RUNDIR/bruit.log` (periodic `STATS`, final `BILAN`); `./run.sh --stop` stops it. A rejected configuration stops `run.sh` before anything is launched. Details: `/opt/GSM/osmo-operator/wiki/Injecteur-bruit.md`.

2.4 /opt/GSM/c54x_exe/balayage_full.tsv

23719 octets, 851 lignes → 215 lignes (1 groupes compactés)

```
decalage    unique  phase   marge   rx_avant    daram_len   vraies  crc_ok  sb_tentees  fb_acceptees
-⟨1⟩    ⟨2⟩ 0.⟨3⟩   ⟨4⟩ ⟨5⟩ ⟨6⟩ 0   ⟨7⟩ ⟨8⟩ ⟨9⟩  ×850
    ⟨⟩ = (3,0,0,0,0,296,0,64,66) (3,0,0,0,0,380,0,64,66) (3,0,0,0,1,296,0,58,60) (3,0,0,0,1,380,0,58,60)
        (3,0,0,10,0,296,2,62,67) (3,0,0,10,0,380,2,62,67) (3,0,0,10,1,296,2,55,60) (3,0,0,10,1,380,2,55,60)
        (3,0,0,21,0,296,1,64,67) (3,0,0,21,0,380,1,64,67) (3,0,0,21,1,296,0,58,60) (3,0,0,21,1,380,0,58,60)
        (3,0,0,30,0,296,0,64,66) (3,0,0,30,0,380,0,64,66) (3,0,0,30,1,296,0,58,59) (3,0,0,30,1,380,0,58,59)
        (3,0,0,41,0,296,1,67,71) (3,0,0,41,0,380,1,67,71) (3,0,0,41,1,296,3,57,59) (3,0,0,41,1,380,3,57,59)
        (3,0,1,0,0,296,2,67,70) (3,0,1,0,0,380,2,67,70) (3,0,1,0,1,296,3,57,60) (3,0,1,0,1,380,3,57,60)
        (3,0,1,10,0,296,2,61,66) (3,0,1,10,0,380,2,61,66) (3,0,1,10,1,296,0,58,62) (3,0,1,10,1,380,0,58,62)
        (3,0,1,21,0,296,0,64,67) (3,0,1,21,0,380,0,64,67) (3,0,1,21,1,296,2,57,59) (3,0,1,21,1,380,2,57,59)
        (3,0,1,30,0,296,4,65,70) (3,0,1,30,0,380,4,65,70) (3,0,1,30,1,296,1,57,59) (3,0,1,30,1,380,1,57,59)
        (3,0,1,41,0,296,1,65,70) (3,0,1,41,0,380,1,65,70) (3,0,1,41,1,296,4,57,60) (3,0,1,41,1,380,4,57,60)
        (3,0,2,0,0,296,2,62,64) (3,0,2,0,0,380,2,62,64) (3,0,2,0,1,296,0,58,61) (3,0,2,0,1,380,0,58,61)
        (3,0,2,10,0,296,2,62,63) (3,0,2,10,0,380,2,62,63) (3,0,2,10,1,296,2,56,63) (3,0,2,10,1,380,2,56,63)
        (3,0,2,21,0,296,0,64,65) (3,0,2,21,0,380,0,64,65) (3,0,2,21,1,296,5,57,60) (3,0,2,21,1,380,5,57,60)
        (3,0,2,30,0,296,4,59,64) (3,0,2,30,0,380,4,59,64) (3,0,2,30,1,296,3,56,60) (3,0,2,30,1,380,3,56,60)
        (3,0,2,41,0,296,8,62,70) (3,0,2,41,0,380,8,62,70) (3,0,2,41,1,296,0,58,62) (3,0,2,41,1,380,0,58,62)
        (3,0,3,0,0,296,1,62,65) (3,0,3,0,0,380,1,62,65) (3,0,3,0,1,296,2,55,59) (3,0,3,0,1,380,2,55,59)
        (3,0,3,10,0,296,1,67,71) (3,0,3,10,0,380,1,67,71) (3,0,3,10,1,296,2,57,59) (3,0,3,10,1,380,2,57,59)
        (3,0,3,21,0,296,1,61,65) (3,0,3,21,0,380,1,61,65) (3,0,3,21,1,296,2,54,60) (3,0,3,21,1,380,2,54,60)
        (3,0,3,30,0,296,2,65,69) (3,0,3,30,0,380,2,65,69) (3,0,3,30,1,296,0,58,61) (3,0,3,30,1,380,0,58,61)
        (3,0,3,41,0,296,4,63,68) (3,0,3,41,0,380,4,63,68) (3,0,3,41,1,296,3,56,59) (3,0,3,41,1,380,3,56,59)
        (3,0,4,0,0,296,3,66,68) (3,0,4,0,0,380,3,66,68) (3,0,4,0,1,296,2,58,59) (3,0,4,0,1,380,2,58,59)
        (3,0,4,10,0,296,0,66,67) (3,0,4,10,0,380,0,66,67) (3,0,4,10,1,296,2,57,61) (3,0,4,10,1,380,2,57,61)
        (3,0,4,21,0,296,3,65,67) (3,0,4,21,0,380,3,65,67) (3,0,4,21,1,296,1,57,59) (3,0,4,21,1,380,1,57,59)
        (3,0,4,30,0,296,3,62,67) (3,0,4,30,0,380,3,62,67) (3,0,4,30,1,296,6,54,61) (3,0,4,30,1,380,6,54,61)
        (3,0,4,41,0,296,4,61,67) (3,0,4,41,0,380,4,61,67) (3,0,4,41,1,296,1,57,59) (3,0,4,41,1,380,1,57,59)
        (3,0,5,0,0,296,4,64,72) (3,0,5,0,0,380,4,64,72) (3,0,5,0,1,296,1,58,61) (3,0,5,0,1,380,1,58,61)
        (3,0,5,10,0,296,2,64,67) (3,0,5,10,0,380,2,64,67) (3,0,5,10,1,296,3,56,60) (3,0,5,10,1,380,3,56,60)
        (3,0,5,21,0,296,1,66,69) (3,0,5,21,0,380,1,66,69) (3,0,5,21,1,296,1,58,59) (3,0,5,21,1,380,1,58,59)
        (3,0,5,30,0,296,0,64,65) (3,0,5,30,0,380,0,64,65) (3,0,5,30,1,296,1,56,60) (3,0,5,30,1,380,1,56,60)
        (3,0,5,41,0,296,5,61,68) (3,0,5,41,0,380,5,61,68) (3,0,5,41,1,296,0,58,58) (3,0,5,41,1,380,0,58,58)
        (3,0,6,0,0,296,3,65,69) (3,0,6,0,0,380,3,65,69) (3,0,6,0,1,296,0,58,59) (3,0,6,0,1,380,0,58,59)
        (3,0,6,10,0,296,4,63,67) (3,0,6,10,0,380,4,63,67) (3,0,6,10,1,296,3,56,62) (3,0,6,10,1,380,3,56,62)
        (3,0,6,21,0,296,1,64,68) (3,0,6,21,0,380,1,64,68) (3,0,6,21,1,296,1,57,59) (3,0,6,21,1,380,1,57,59)
        (3,0,6,30,0,296,4,65,70) (3,0,6,30,0,380,4,65,70) (3,0,6,30,1,296,0,58,59) (3,0,6,30,1,380,0,58,59)
        (3,0,6,41,0,296,2,63,66) (3,0,6,41,0,380,2,63,66) (3,0,6,41,1,296,1,57,60) (3,0,6,41,1,380,1,57,60)
        (3,0,7,0,0,296,3,66,70) (3,0,7,0,0,380,3,66,70) (3,0,7,0,1,296,1,58,61) (3,0,7,0,1,380,1,58,61)
        (3,0,7,10,0,296,6,63,70) (3,0,7,10,0,380,6,63,70) (3,0,7,10,1,296,1,57,61) (3,0,7,10,1,380,1,57,61)
        (3,0,7,21,0,296,2,68,70) (3,0,7,21,0,380,2,68,70) (3,0,7,21,1,296,1,57,61) (3,0,7,21,1,380,1,57,61)
        (3,0,7,30,0,296,0,66,69) (3,0,7,30,0,380,0,66,69) (3,0,7,30,1,296,0,58,61) (3,0,7,30,1,380,0,58,61)
```

(3,0,7,41,0,296,3,67,70) (3,0,7,41,0,380,3,67,70) (3,0,7,41,1,296,1,57,60) (3,0,7,41,1,380,1,57,60)
(3,0,8,0,0,296,6,65,69) (3,0,8,0,0,380,6,65,69) (3,0,8,0,1,296,1,58,58) (3,0,8,0,1,380,1,58,58)
(3,0,8,10,0,296,1,60,62) (3,0,8,10,0,380,1,60,62) (3,0,8,10,1,296,2,56,60) (3,0,8,10,1,380,2,56,60)
(3,0,8,21,0,296,3,62,67) (3,0,8,21,0,380,3,62,67) (3,0,8,21,1,296,2,58,59) (3,0,8,21,1,380,2,58,59)
(3,0,8,30,0,296,2,61,64) (3,0,8,30,0,380,2,61,64) (3,0,8,30,1,296,1,58,59) (3,0,8,30,1,380,1,58,59)
(3,0,8,41,0,296,2,61,66) (3,0,8,41,0,380,2,61,66) (3,0,8,41,1,296,2,57,61) (3,0,8,41,1,380,2,57,61)
(3,0,9,0,0,296,1,62,63) (3,0,9,0,0,380,1,62,63) (3,0,9,0,1,296,2,58,59) (3,0,9,0,1,380,2,58,59)
(3,0,9,10,0,296,1,63,66) (3,0,9,10,0,380,1,63,66) (3,0,9,10,1,296,1,57,59) (3,0,9,10,1,380,1,57,59)
(3,0,9,21,0,296,6,60,67) (3,0,9,21,0,380,6,60,67) (3,0,9,21,1,296,2,58,59) (3,0,9,21,1,380,2,58,59)
(3,0,9,30,0,296,3,60,65) (3,0,9,30,0,380,3,60,65) (3,0,9,30,1,296,0,58,60) (3,0,9,30,1,380,0,58,60)
(3,0,9,41,0,296,4,64,68) (3,0,9,41,0,380,4,64,68) (3,0,9,41,1,296,1,57,59) (3,0,9,41,1,380,1,57,59)
(3,1,0,0,0,296,1,37,78) (3,1,0,0,0,380,1,37,78) (3,1,0,0,1,296,1,29,60) (3,1,0,0,1,380,1,29,60)
(3,1,0,10,0,296,2,37,75) (3,1,0,10,0,380,2,37,75) (3,1,0,10,1,296,1,29,61) (3,1,0,10,1,380,1,29,61)
(3,1,0,21,0,296,2,37,77) (3,1,0,21,0,380,2,37,77) (3,1,0,21,1,296,1,29,59) (3,1,0,21,1,380,1,29,59)
(3,1,0,30,0,296,1,36,74) (3,1,0,30,0,380,1,36,74) (3,1,0,30,1,296,1,29,60) (3,1,0,30,1,380,1,29,60)
(3,1,0,41,0,296,2,36,74) (3,1,0,41,0,380,2,36,74) (3,1,0,41,1,296,1,29,61) (3,1,0,41,1,380,1,29,61)
(3,1,1,0,0,296,1,36,75) (3,1,1,0,0,380,1,36,75) (3,1,1,0,1,296,1,29,61) (3,1,1,0,1,380,1,29,61)
(3,1,1,10,0,296,1,37,76) (3,1,1,10,0,380,1,37,76) (3,1,1,10,1,296,2,29,60) (3,1,1,10,1,380,2,29,60)
(3,1,1,21,0,296,4,37,75) (3,1,1,21,0,380,4,37,75) (3,1,1,21,1,296,1,29,61) (3,1,1,21,1,380,1,29,61)
(3,1,1,30,0,296,1,38,79) (3,1,1,30,0,380,1,38,79) (3,1,1,30,1,296,1,29,59) (3,1,1,30,1,380,1,29,59)
(3,1,1,41,0,296,3,37,76) (3,1,1,41,0,380,3,37,76) (3,1,1,41,1,296,1,29,60) (3,1,1,41,1,380,1,29,60)
(3,1,2,0,0,296,1,37,77) (3,1,2,0,0,380,1,37,77) (3,1,2,0,1,296,1,30,62) (3,1,2,0,1,380,1,30,62)
(3,1,2,10,0,296,1,36,73) (3,1,2,10,0,380,1,36,73) (3,1,2,10,1,296,2,30,62) (3,1,2,10,1,380,2,30,62)
(3,1,2,21,0,296,2,36,73) (3,1,2,21,0,380,2,36,73) (3,1,2,21,1,296,1,29,62) (3,1,2,21,1,380,1,29,62)
(3,1,2,30,0,296,3,36,75) (3,1,2,30,0,380,3,36,75) (3,1,2,30,1,296,2,30,64) (3,1,2,30,1,380,2,30,64)
(3,1,2,41,0,296,1,36,74) (3,1,2,41,0,380,1,36,74) (3,1,2,41,1,296,1,29,60) (3,1,2,41,1,380,1,29,60)
(3,1,3,0,0,296,1,37,76) (3,1,3,0,0,380,1,37,76) (3,1,3,0,1,296,2,29,61) (3,1,3,0,1,380,2,29,61)
(3,1,3,10,0,296,2,36,77) (3,1,3,10,0,380,2,36,77) (3,1,3,10,1,296,1,29,59) (3,1,3,10,1,380,1,29,59)
(3,1,3,21,0,296,1,37,77) (3,1,3,21,0,380,1,37,77) (3,1,3,21,1,296,3,29,60) (3,1,3,21,1,380,3,29,60)
(3,1,3,30,0,296,2,38,77) (3,1,3,30,0,380,2,38,77) (3,1,3,30,1,296,2,29,60) (3,1,3,30,1,380,2,29,60)
(3,1,3,41,0,296,2,38,79) (3,1,3,41,0,380,2,38,79) (3,1,3,41,1,296,2,29,60) (3,1,3,41,1,380,2,29,60)
(3,1,4,0,0,296,1,37,76) (3,1,4,0,0,380,1,37,76) (3,1,4,0,1,296,1,29,61) (3,1,4,0,1,380,1,29,61)
(3,1,4,10,0,296,2,35,72) (3,1,4,10,0,380,2,35,72) (3,1,4,10,1,296,1,29,60) (3,1,4,10,1,380,1,29,60)
(3,1,4,21,0,296,2,37,76) (3,1,4,21,0,380,2,37,76) (3,1,4,21,1,296,3,30,63) (3,1,4,21,1,380,3,30,63)
(3,1,4,30,0,296,1,37,78) (3,1,4,30,0,380,1,37,78) (3,1,4,30,1,296,1,29,59) (3,1,4,30,1,380,1,29,59)
(3,1,4,41,0,296,2,37,75) (3,1,4,41,0,380,2,37,75) (3,1,4,41,1,296,2,29,58) (3,1,4,41,1,380,2,29,58)
(3,1,5,0,0,296,3,37,76) (3,1,5,0,0,380,3,37,76) (3,1,5,0,1,296,2,29,60) (3,1,5,0,1,380,2,29,60)
(3,1,5,10,0,296,2,36,74) (3,1,5,10,0,380,2,36,74) (3,1,5,10,1,296,1,28,60) (3,1,5,10,1,380,1,28,60)
(3,1,5,21,0,296,3,38,78) (3,1,5,21,0,380,3,38,78) (3,1,5,21,1,296,1,29,60) (3,1,5,21,1,380,1,29,60)
(3,1,5,30,0,296,1,37,77) (3,1,5,30,0,380,1,37,77) (3,1,5,30,1,296,1,29,60) (3,1,5,30,1,380,1,29,60)
(3,1,5,41,0,296,1,38,78) (3,1,5,41,0,380,1,38,78) (3,1,5,41,1,296,1,29,59) (3,1,5,41,1,380,1,29,59)
(3,1,6,0,0,296,1,38,77) (3,1,6,0,0,380,1,38,77) (3,1,6,0,1,296,1,29,61) (3,1,6,0,1,380,1,29,61)
(3,1,6,10,0,296,2,35,73) (3,1,6,10,0,380,2,35,73) (3,1,6,10,1,296,1,29,59) (3,1,6,10,1,380,1,29,59)
(3,1,6,21,0,296,1,37,76) (3,1,6,21,0,380,1,37,76) (3,1,6,21,1,296,1,28,59) (3,1,6,21,1,380,1,28,59)
(3,1,6,30,0,296,1,37,76) (3,1,6,30,0,380,1,37,76) (3,1,6,30,1,296,1,29,59) (3,1,6,30,1,380,1,29,59)
(3,1,6,41,0,296,1,36,76) (3,1,6,41,0,380,1,36,76) (3,1,6,41,1,296,1,29,59) (3,1,6,41,1,380,1,29,59)
(3,1,7,0,0,296,1,37,75) (3,1,7,0,0,380,1,37,75) (3,1,7,0,1,296,2,30,62) (3,1,7,0,1,380,2,30,62)
(3,1,7,10,0,296,1,38,78) (3,1,7,10,0,380,1,38,78) (3,1,7,10,1,296,2,30,61) (3,1,7,10,1,380,2,30,61)
(3,1,7,21,0,296,1,38,78) (3,1,7,21,0,380,1,38,78) (3,1,7,21,1,296,2,29,61) (3,1,7,21,1,380,2,29,61)
(3,1,7,30,0,296,1,41,83) (3,1,7,30,0,380,1,41,83) (3,1,7,30,1,296,1,30,61) (3,1,7,30,1,380,1,30,61)
(3,1,7,41,0,296,1,41,83) (3,1,7,41,0,380,1,41,83) (3,1,7,41,1,296,2,30,61) (3,1,7,41,1,380,2,30,61)
(3,1,8,0,0,296,2,37,77) (3,1,8,0,0,380,2,37,77) (3,1,8,0,1,296,1,29,60) (3,1,8,0,1,380,1,29,60)
(3,1,8,10,0,296,1,37,76) (3,1,8,10,0,380,1,37,76) (3,1,8,10,1,296,2,29,59) (3,1,8,10,1,380,2,29,59)
(3,1,8,21,0,296,1,38,78) (3,1,8,21,0,380,1,38,78) (3,1,8,21,1,296,2,29,60) (3,1,8,21,1,380,2,29,60)
(3,1,8,30,0,296,2,38,78) (3,1,8,30,0,380,2,38,78) (3,1,8,30,1,296,1,29,60) (3,1,8,30,1,380,1,29,60)
(3,1,8,41,0,296,5,37,76) (3,1,8,41,0,380,5,37,76) (3,1,8,41,1,296,3,29,60) (3,1,8,41,1,380,3,29,60)
(3,1,9,0,0,296,2,35,71) (3,1,9,0,0,380,2,35,71) (3,1,9,0,1,296,1,29,59) (3,1,9,0,1,380,1,29,59)
(3,1,9,10,0,296,2,37,77) (3,1,9,10,0,380,2,37,77) (3,1,9,10,1,296,1,29,60) (3,1,9,10,1,380,1,29,60)
(3,1,9,21,0,296,2,36,75) (3,1,9,21,0,380,2,36,75) (3,1,9,21,1,296,1,30,65) (3,1,9,21,1,380,1,30,65)
(3,1,9,30,0,296,1,37,76) (3,1,9,30,0,380,1,37,76) (3,1,9,30,1,296,1,29,59) (3,1,9,30,1,380,1,29,59)
(3,1,9,41,0,296,2,37,76) (3,1,9,41,0,380,2,37,76) (3,1,9,41,1,296,3,29,59) (3,1,9,41,1,380,3,29,59)
(2,0,0,0,0,296,1,58,62) (2,0,0,0,0,380,1,58,62) (2,0,0,0,1,296,2,58,60) (2,0,0,0,1,380,2,58,60)
(2,0,0,10,0,296,4,58,61) (2,0,0,10,0,380,4,58,61) (2,0,0,10,1,296,7,54,60) (2,0,0,10,1,380,7,54,60)
(2,0,0,21,0,296,1,59,63) (2,0,0,21,0,380,1,59,63) (2,0,0,21,1,296,2,57,60) (2,0,0,21,1,380,2,57,60)
(2,0,0,30,0,296,2,56,61) (2,0,0,30,0,380,2,56,61) (2,0,0,30,1,296,2,57,59) (2,0,0,30,1,380,2,57,59)
(2,0,0,41,0,296,0,58,63) (2,0,0,41,0,380,0,58,63) (2,0,0,41,1,296,3,57,61) (2,0,0,41,1,380,3,57,61)

(2,0,1,0,0,296,2,58,61) (2,0,1,0,0,380,2,58,61) (2,0,1,0,1,296,1,56,59) (2,0,1,0,1,380,1,56,59)
(2,0,1,10,0,296,2,57,63) (2,0,1,10,0,380,2,57,63) (2,0,1,10,1,296,4,55,58) (2,0,1,10,1,380,4,55,58)
(2,0,1,21,0,296,0,60,63) (2,0,1,21,0,380,0,60,63) (2,0,1,21,1,296,12,50,59) (2,0,1,21,1,380,12,50,59)
(2,0,1,30,0,296,2,58,62) (2,0,1,30,0,380,2,58,62) (2,0,1,30,1,296,6,52,59) (2,0,1,30,1,380,6,52,59)
(2,0,1,41,0,296,0,58,61) (2,0,1,41,0,380,0,58,61) (2,0,1,41,1,296,2,58,59) (2,0,1,41,1,380,2,58,59)
(2,0,2,0,0,296,5,56,60) (2,0,2,0,0,380,5,56,60) (2,0,2,0,1,296,8,54,61) (2,0,2,0,1,380,8,54,61)
(2,0,2,10,0,296,2,57,60) (2,0,2,10,0,380,2,57,60) (2,0,2,10,1,296,1,57,60) (2,0,2,10,1,380,1,57,60)
(2,0,2,21,0,296,3,59,63) (2,0,2,21,0,380,3,59,63) (2,0,2,21,1,296,0,58,61) (2,0,2,21,1,380,0,58,61)
(2,0,2,30,0,296,3,56,61) (2,0,2,30,0,380,3,56,61) (2,0,2,30,1,296,6,53,60) (2,0,2,30,1,380,6,53,60)
(2,0,2,41,0,296,2,58,62) (2,0,2,41,0,380,2,58,62) (2,0,2,41,1,296,1,57,60) (2,0,2,41,1,380,1,57,60)
(2,0,3,0,0,296,1,58,62) (2,0,3,0,0,380,1,58,62) (2,0,3,0,1,296,5,54,59) (2,0,3,0,1,380,5,54,59)
(2,0,3,10,0,296,1,58,61) (2,0,3,10,0,380,1,58,61) (2,0,3,10,1,296,3,54,60) (2,0,3,10,1,380,3,54,60)
(2,0,3,21,0,296,0,62,63) (2,0,3,21,0,380,0,62,63) (2,0,3,21,1,296,4,56,61) (2,0,3,21,1,380,4,56,61)
(2,0,3,30,0,296,4,59,64) (2,0,3,30,0,380,4,59,64) (2,0,3,30,1,296,2,54,59) (2,0,3,30,1,380,2,54,59)
(2,0,3,41,0,296,6,58,63) (2,0,3,41,0,380,6,58,63) (2,0,3,41,1,296,4,53,60) (2,0,3,41,1,380,4,53,60)
(2,0,4,0,0,296,0,60,61) (2,0,4,0,0,380,0,60,61) (2,0,4,0,1,296,2,57,60) (2,0,4,0,1,380,2,57,60)
(2,0,4,10,0,296,1,57,61) (2,0,4,10,0,380,1,57,61) (2,0,4,10,1,296,3,53,59) (2,0,4,10,1,380,3,53,59)
(2,0,4,21,0,296,1,59,64) (2,0,4,21,0,380,1,59,64) (2,0,4,21,1,296,1,56,61) (2,0,4,21,1,380,1,56,61)
(2,0,4,30,0,296,1,58,60) (2,0,4,30,0,380,1,58,60) (2,0,4,30,1,296,2,57,58) (2,0,4,30,1,380,2,57,58)
(2,0,4,41,0,296,1,58,60) (2,0,4,41,0,380,1,58,60) (2,0,4,41,1,296,0,58,59) (2,0,4,41,1,380,0,58,59)
(2,0,5,0,0,296,2,58,63) (2,0,5,0,0,380,2,58,63) (2,0,5,0,1,296,1,58,58) (2,0,5,0,1,380,1,58,58)
(2,0,5,10,0,296,1,58,61) (2,0,5,10,0,380,1,58,61) (2,0,5,10,1,296,4,53,58) (2,0,5,10,1,380,4,53,58)
(2,0,5,21,0,296,6,57,62) (2,0,5,21,0,380,6,57,62) (2,0,5,21,1,296,1,57,61) (2,0,5,21,1,380,1,57,61)
(2,0,5,30,0,296,0,60,65) (2,0,5,30,0,380,0,60,65) (2,0,5,30,1,296,1,58,61) (2,0,5,30,1,380,1,58,61)
(2,0,5,41,0,296,1,57,61) (2,0,5,41,0,380,1,57,61) (2,0,5,41,1,296,1,58,59) (2,0,5,41,1,380,1,58,59)
(2,0,6,0,0,296,1,60,63) (2,0,6,0,0,380,1,60,63) (2,0,6,0,1,296,5,56,58) (2,0,6,0,1,380,5,56,58)
(2,0,6,10,0,296,2,57,60) (2,0,6,10,0,380,2,57,60) (2,0,6,10,1,296,1,57,60) (2,0,6,10,1,380,1,57,60)
(2,0,6,21,0,296,4,59,62) (2,0,6,21,0,380,4,59,62) (2,0,6,21,1,296,2,57,61) (2,0,6,21,1,380,2,57,61)
(2,0,6,30,0,296,2,58,65) (2,0,6,30,0,380,2,58,65) (2,0,6,30,1,296,2,57,60) (2,0,6,30,1,380,2,57,60)
(2,0,6,41,0,296,0,60,62) (2,0,6,41,0,380,0,60,62) (2,0,6,41,1,296,3,56,59) (2,0,6,41,1,380,3,56,59)
(2,0,7,0,0,296,0,58,61) (2,0,7,0,0,380,0,58,61) (2,0,7,0,1,296,7,52,60) (2,0,7,0,1,380,7,52,60)
(2,0,7,10,0,296,2,58,60) (2,0,7,10,0,380,2,58,60) (2,0,7,10,1,296,7,52,60) (2,0,7,10,1,380,7,52,60)
(2,0,7,21,0,296,0,58,60) (2,0,7,21,0,380,0,58,60) (2,0,7,21,1,296,10,52,61) (2,0,7,21,1,380,10,52,61)
(2,0,7,30,0,296,0,58,60) (2,0,7,30,0,380,0,58,60) (2,0,7,30,1,296,3,56,60) (2,0,7,30,1,380,3,56,60)
(2,0,7,41,0,296,0,58,60) (2,0,7,41,0,380,0,58,60) (2,0,7,41,1,296,8,52,61) (2,0,7,41,1,380,8,52,61)
(2,0,8,0,0,296,2,56,60) (2,0,8,0,0,380,2,56,60) (2,0,8,0,1,296,0,58,59) (2,0,8,0,1,380,0,58,59)
(2,0,8,10,0,296,1,57,61) (2,0,8,10,0,380,1,57,61) (2,0,8,10,1,296,5,56,59) (2,0,8,10,1,380,5,56,59)
(2,0,8,21,0,296,4,59,61) (2,0,8,21,0,380,4,59,61) (2,0,8,21,1,296,7,54,59) (2,0,8,21,1,380,7,54,59)
(2,0,8,30,0,296,1,60,61) (2,0,8,30,0,380,1,60,61) (2,0,8,30,1,296,6,55,59) (2,0,8,30,1,380,6,55,59)
(2,0,8,41,0,296,1,59,62) (2,0,8,41,0,380,1,59,62) (2,0,8,41,1,296,4,54,59) (2,0,8,41,1,380,4,54,59)
(2,0,9,0,0,296,5,57,63) (2,0,9,0,0,380,5,57,63) (2,0,9,0,1,296,6,54,60) (2,0,9,0,1,380,6,54,60)
(2,0,9,10,0,296,2,56,60) (2,0,9,10,0,380,2,56,60) (2,0,9,10,1,296,8,54,61) (2,0,9,10,1,380,8,54,61)
(2,0,9,21,0,296,1,60,63) (2,0,9,21,0,380,1,60,63) (2,0,9,21,1,296,4,56,60) (2,0,9,21,1,380,4,56,60)
(2,0,9,30,0,296,3,57,62) (2,0,9,30,0,380,3,57,62) (2,0,9,30,1,296,3,57,59) (2,0,9,30,1,380,3,57,59)
(2,0,9,41,0,296,3,56,61) (2,0,9,41,0,380,3,56,61) (2,0,9,41,1,296,10,50,61) (2,0,9,41,1,380,10,50,61)
(2,1,0,0,0,296,2,33,68) (2,1,0,0,0,380,2,33,68) (2,1,0,0,1,296,1,29,60) (2,1,0,0,1,380,1,29,60)
(2,1,0,10,0,296,1,31,63) (2,1,0,10,0,380,1,31,63) (2,1,0,10,1,296,2,28,60) (2,1,0,10,1,380,2,28,60)
(2,1,0,21,0,296,1,32,67) (2,1,0,21,0,380,1,32,67) (2,1,0,21,1,296,1,29,60) (2,1,0,21,1,380,1,29,60)
(2,1,0,30,0,296,1,31,64) (2,1,0,30,0,380,1,31,64) (2,1,0,30,1,296,1,29,59) (2,1,0,30,1,380,1,29,59)
(2,1,0,41,0,296,2,31,65) (2,1,0,41,0,380,2,31,65) (2,1,0,41,1,296,2,29,59) (2,1,0,41,1,380,2,29,59)
(2,1,1,0,0,296,0,31,64) (2,1,1,0,0,380,0,31,64) (2,1,1,0,1,296,3,29,60) (2,1,1,0,1,380,3,29,60)
(2,1,1,10,0,296,0,32,67) (2,1,1,10,0,380,0,32,67) (2,1,1,10,1,296,1,29,61) (2,1,1,10,1,380,1,29,61)
(2,1,1,21,0,296,1,31,66) (2,1,1,21,0,380,1,31,66) (2,1,1,21,1,296,2,29,60) (2,1,1,21,1,380,2,29,60)
(2,1,1,30,0,296,2,30,66) (2,1,1,30,0,380,2,30,66) (2,1,1,30,1,296,1,29,61) (2,1,1,30,1,380,1,29,61)
(2,1,1,41,0,296,1,33,68) (2,1,1,41,0,380,1,33,68) (2,1,1,41,1,296,1,29,60) (2,1,1,41,1,380,1,29,60)
(2,1,2,0,0,296,1,33,68) (2,1,2,0,0,380,1,33,68) (2,1,2,0,1,296,2,29,62) (2,1,2,0,1,380,2,29,62)
(2,1,2,10,0,296,1,32,66) (2,1,2,10,0,380,1,32,66) (2,1,2,10,1,296,2,29,63) (2,1,2,10,1,380,2,29,63)
(2,1,2,21,0,296,1,33,71) (2,1,2,21,0,380,1,33,71) (2,1,2,21,1,296,0,29,61) (2,1,2,21,1,380,0,29,61)
(2,1,2,30,0,296,1,33,68) (2,1,2,30,0,380,1,33,68) (2,1,2,30,1,296,1,29,62) (2,1,2,30,1,380,1,29,62)
(2,1,2,41,0,296,2,31,66) (2,1,2,41,0,380,2,31,66) (2,1,2,41,1,296,1,29,60) (2,1,2,41,1,380,1,29,60)
(2,1,3,0,0,296,2,33,68) (2,1,3,0,0,380,2,33,68) (2,1,3,0,1,296,3,28,59) (2,1,3,0,1,380,3,28,59)
(2,1,3,10,0,296,1,30,63) (2,1,3,10,0,380,1,30,63) (2,1,3,10,1,296,2,29,59) (2,1,3,10,1,380,2,29,59)
(2,1,3,21,0,296,1,32,68) (2,1,3,21,0,380,1,32,68) (2,1,3,21,1,296,1,29,59) (2,1,3,21,1,380,1,29,59)
(2,1,3,30,0,296,1,32,68) (2,1,3,30,0,380,1,32,68) (2,1,3,30,1,296,2,29,61) (2,1,3,30,1,380,2,29,61)
(2,1,3,41,0,296,1,31,65) (2,1,3,41,0,380,1,31,65) (2,1,3,41,1,296,2,29,59) (2,1,3,41,1,380,2,29,59)
(2,1,4,0,0,296,2,34,71) (2,1,4,0,0,380,2,34,71) (2,1,4,0,1,296,2,29,59) (2,1,4,0,1,380,2,29,59)

(2,1,4,10,0,296,2,31,64) (2,1,4,10,0,380,2,31,64) (2,1,4,10,1,296,2,29,60) (2,1,4,10,1,380,2,29,60)
(2,1,4,21,0,296,1,31,64) (2,1,4,21,0,380,1,31,64) (2,1,4,21,1,296,1,29,59) (2,1,4,21,1,380,1,29,59)
(2,1,4,30,0,296,3,32,66) (2,1,4,30,0,380,3,32,66) (2,1,4,30,1,296,2,28,61) (2,1,4,30,1,380,2,28,61)
(2,1,4,41,0,296,0,33,69) (2,1,4,41,0,380,0,33,69) (2,1,4,41,1,296,3,29,60) (2,1,4,41,1,380,3,29,60)
(2,1,5,0,0,296,1,32,66) (2,1,5,0,0,380,1,32,66) (2,1,5,0,1,296,1,29,61) (2,1,5,0,1,380,1,29,61)
(2,1,5,10,0,296,1,31,63) (2,1,5,10,0,380,1,31,63) (2,1,5,10,1,296,2,29,60) (2,1,5,10,1,380,2,29,60)
(2,1,5,21,0,296,1,31,64) (2,1,5,21,0,380,1,31,64) (2,1,5,21,1,296,2,29,59) (2,1,5,21,1,380,2,29,59)
(2,1,5,30,0,296,1,31,65) (2,1,5,30,0,380,1,31,65) (2,1,5,30,1,296,2,28,60) (2,1,5,30,1,380,2,28,60)
(2,1,5,41,0,296,1,31,65) (2,1,5,41,0,380,1,31,65) (2,1,5,41,1,296,1,29,58) (2,1,5,41,1,380,1,29,58)
(2,1,6,0,0,296,1,32,68) (2,1,6,0,0,380,1,32,68) (2,1,6,0,1,296,1,29,59) (2,1,6,0,1,380,1,29,59)
(2,1,6,10,0,296,3,32,67) (2,1,6,10,0,380,3,32,67) (2,1,6,10,1,296,2,29,60) (2,1,6,10,1,380,2,29,60)
(2,1,6,21,0,296,2,33,68) (2,1,6,21,0,380,2,33,68) (2,1,6,21,1,296,1,29,59) (2,1,6,21,1,380,1,29,59)
(2,1,6,30,0,296,2,32,65) (2,1,6,30,0,380,2,32,65) (2,1,6,30,1,296,1,29,59) (2,1,6,30,1,380,1,29,59)
(2,1,6,41,0,296,5,32,65) (2,1,6,41,0,380,5,32,65) (2,1,6,41,1,296,1,29,60) (2,1,6,41,1,380,1,29,60)
(2,1,7,0,0,296,3,33,69) (2,1,7,0,0,380,3,33,69) (2,1,7,0,1,296,0,29,61) (2,1,7,0,1,380,0,29,61)
(2,1,7,10,0,296,3,33,67) (2,1,7,10,0,380,3,33,67) (2,1,7,10,1,296,0,29,60) (2,1,7,10,1,380,0,29,60)
(2,1,7,21,0,296,3,33,68) (2,1,7,21,0,380,3,33,68) (2,1,7,21,1,296,2,29,61) (2,1,7,21,1,380,2,29,61)
(2,1,7,30,0,296,1,34,70) (2,1,7,30,0,380,1,34,70) (2,1,7,30,1,296,0,29,60) (2,1,7,30,1,380,0,29,60)
(2,1,7,41,0,296,1,34,70) (2,1,7,41,0,380,1,34,70) (2,1,7,41,1,296,1,29,60) (2,1,7,41,1,380,1,29,60)
(2,1,8,0,0,296,2,32,68) (2,1,8,0,0,380,2,32,68) (2,1,8,0,1,296,2,29,58) (2,1,8,0,1,380,2,29,58)
(2,1,8,10,0,296,1,31,64) (2,1,8,10,0,380,1,31,64) (2,1,8,10,1,296,1,29,60) (2,1,8,10,1,380,1,29,60)
(2,1,8,21,0,296,0,31,65) (2,1,8,21,0,380,0,31,65) (2,1,8,21,1,296,2,29,59) (2,1,8,21,1,380,2,29,59)
(2,1,8,30,0,296,3,33,68) (2,1,8,30,0,380,3,33,68) (2,1,8,30,1,296,1,29,59) (2,1,8,30,1,380,1,29,59)
(2,1,8,41,0,296,2,31,65) (2,1,8,41,0,380,2,31,65) (2,1,8,41,1,296,1,29,59) (2,1,8,41,1,380,1,29,59)
(2,1,9,0,0,296,3,32,67) (2,1,9,0,0,380,3,32,67) (2,1,9,0,1,296,2,29,59) (2,1,9,0,1,380,2,29,59)
(2,1,9,10,0,296,1,32,67) (2,1,9,10,0,380,1,32,67) (2,1,9,10,1,296,1,29,61) (2,1,9,10,1,380,1,29,61)
(2,1,9,21,0,296,1,31,66) (2,1,9,21,0,380,1,31,66) (2,1,9,21,1,296,1,29,59) (2,1,9,21,1,380,1,29,59)
(2,1,9,30,0,296,0,31,64) (2,1,9,30,0,380,0,31,64) (2,1,9,30,1,296,1,29,60) (2,1,9,30,1,380,1,29,60)
(2,1,9,41,0,296,1,31,65) (2,1,9,41,0,380,1,31,65) (2,1,9,41,1,296,1,29,59) (2,1,9,41,1,380,1,29,59)
(1,0,0,0,0,296,2,58,63) (1,0,0,0,0,380,2,58,63) (1,0,0,0,1,296,3,55,59) (1,0,0,0,1,380,3,55,59)
(1,0,0,10,0,296,0,58,60) (1,0,0,10,0,380,0,58,60) (1,0,0,10,1,296,1,56,59) (1,0,0,10,1,380,1,56,59)
(1,0,0,21,0,296,1,57,60) (1,0,0,21,0,380,1,57,60) (1,0,0,21,1,296,6,55,59) (1,0,0,21,1,380,6,55,59)
(1,0,0,30,0,296,4,55,62) (1,0,0,30,0,380,4,55,62) (1,0,0,30,1,296,1,56,59) (1,0,0,30,1,380,1,56,59)
(1,0,0,41,0,296,0,58,61) (1,0,0,41,0,380,0,58,61) (1,0,0,41,1,296,7,53,59) (1,0,0,41,1,380,7,53,59)
(1,0,1,0,0,296,1,58,61) (1,0,1,0,0,380,1,58,61) (1,0,1,0,1,296,9,55,58) (1,0,1,0,1,380,9,55,58)
(1,0,1,10,0,296,2,56,60) (1,0,1,10,0,380,2,56,60) (1,0,1,10,1,296,2,54,60) (1,0,1,10,1,380,2,54,60)
(1,0,1,21,0,296,3,56,60) (1,0,1,21,0,380,3,56,60) (1,0,1,21,1,296,7,54,59) (1,0,1,21,1,380,7,54,59)
(1,0,1,30,0,296,0,58,62) (1,0,1,30,0,380,0,58,62) (1,0,1,30,1,296,2,55,59) (1,0,1,30,1,380,2,55,59)
(1,0,1,41,0,296,5,55,60) (1,0,1,41,0,380,5,55,60) (1,0,1,41,1,296,2,56,59) (1,0,1,41,1,380,2,56,59)
(1,0,2,0,0,296,2,57,60) (1,0,2,0,0,380,2,57,60) (1,0,2,0,1,296,7,55,60) (1,0,2,0,1,380,7,55,60)
(1,0,2,10,0,296,6,57,60) (1,0,2,10,0,380,6,57,60) (1,0,2,10,1,296,2,54,60) (1,0,2,10,1,380,2,54,60)
(1,0,2,21,0,296,1,58,60) (1,0,2,21,0,380,1,58,60)

2.5 /opt/GSM/c54x_exe/balayage_v2.tsv

74343 octets, 2801 lignes → 703 lignes (2 groupes compactés)

```
decalage    unique  phase   marge   rx_avant    daram_len   vraies  mots_distincts  crc_ok  sb_tentees
-⟨1⟩    ⟨2⟩ 0.⟨3⟩   ⟨4⟩ ⟨5⟩ ⟨6⟩ 0   ⟨7⟩ ⟨8⟩ ⟨9⟩  ×2000
    ⟨⟩ = (5,0,0,0,0,296,1,1,75) (5,0,0,0,0,380,1,1,75) (5,0,0,0,1,296,0,0,60) (5,0,0,0,1,380,0,0,60)
        (5,0,0,10,0,296,0,0,78) (5,0,0,10,0,380,0,0,78) (5,0,0,10,1,296,0,0,60) (5,0,0,10,1,380,0,0,60)
        (5,0,0,21,0,296,0,0,74) (5,0,0,21,0,380,0,0,74) (5,0,0,21,1,296,0,0,64) (5,0,0,21,1,380,0,0,64)
        (5,0,0,30,0,296,1,2,75) (5,0,0,30,0,380,1,2,75) (5,0,0,30,1,296,0,0,62) (5,0,0,30,1,380,0,0,62)
        (5,0,0,41,0,296,1,1,73) (5,0,0,41,0,380,1,1,73) (5,0,0,41,1,296,1,1,61) (5,0,0,41,1,380,1,1,61)
        (5,0,1,0,0,296,4,4,75) (5,0,1,0,0,380,4,4,75) (5,0,1,0,1,296,1,1,63) (5,0,1,0,1,380,1,1,63)
        (5,0,1,10,0,296,1,1,76) (5,0,1,10,0,380,1,1,76) (5,0,1,10,1,296,0,0,66) (5,0,1,10,1,380,0,0,66)
        (5,0,1,21,0,296,0,0,74) (5,0,1,21,0,380,0,0,74) (5,0,1,21,1,296,0,0,64) (5,0,1,21,1,380,0,0,64)
        (5,0,1,30,0,296,2,2,74) (5,0,1,30,0,380,2,2,74) (5,0,1,30,1,296,3,3,66) (5,0,1,30,1,380,3,3,66)
        (5,0,1,41,0,296,0,0,74) (5,0,1,41,0,380,0,0,74) (5,0,1,41,1,296,2,2,63) (5,0,1,41,1,380,2,2,63)
        (5,0,2,0,0,296,1,4,73) (5,0,2,0,0,380,1,4,73) (5,0,2,0,1,296,5,5,62) (5,0,2,0,1,380,5,5,62)
        (5,0,2,10,0,296,1,1,77) (5,0,2,10,0,380,1,1,77) (5,0,2,10,1,296,1,3,63) (5,0,2,10,1,380,1,3,63)
        (5,0,2,21,0,296,1,1,80) (5,0,2,21,0,380,1,1,80) (5,0,2,21,1,296,2,4,59) (5,0,2,21,1,380,2,4,59)
        (5,0,2,30,0,296,2,2,74) (5,0,2,30,0,380,2,2,74) (5,0,2,30,1,296,4,9,61) (5,0,2,30,1,380,4,9,61)
        (5,0,2,41,0,296,2,2,74) (5,0,2,41,0,380,2,2,74) (5,0,2,41,1,296,3,3,62) (5,0,2,41,1,380,3,3,62)
```

(5,0,3,0,0,296,4,4,75) (5,0,3,0,0,380,4,4,75) (5,0,3,0,1,296,3,3,62) (5,0,3,0,1,380,3,3,62)
(5,0,3,10,0,296,3,5,72) (5,0,3,10,0,380,3,5,72) (5,0,3,10,1,296,2,2,62) (5,0,3,10,1,380,2,2,62)
(5,0,3,21,0,296,1,1,74) (5,0,3,21,0,380,1,1,74) (5,0,3,21,1,296,3,3,62) (5,0,3,21,1,380,3,3,62)
(5,0,3,30,0,296,2,2,75) (5,0,3,30,0,380,2,2,75) (5,0,3,30,1,296,2,2,63) (5,0,3,30,1,380,2,2,63)
(5,0,3,41,0,296,5,6,72) (5,0,3,41,0,380,5,6,72) (5,0,3,41,1,296,3,3,60) (5,0,3,41,1,380,3,3,60)
(5,0,4,0,0,296,1,1,77) (5,0,4,0,0,380,1,1,77) (5,0,4,0,1,296,1,2,64) (5,0,4,0,1,380,1,2,64)
(5,0,4,10,0,296,6,6,78) (5,0,4,10,0,380,6,6,78) (5,0,4,10,1,296,0,0,60) (5,0,4,10,1,380,0,0,60)
(5,0,4,21,0,296,3,6,76) (5,0,4,21,0,380,3,6,76) (5,0,4,21,1,296,2,2,66) (5,0,4,21,1,380,2,2,66)
(5,0,4,30,0,296,2,4,75) (5,0,4,30,0,380,2,4,75) (5,0,4,30,1,296,1,1,68) (5,0,4,30,1,380,1,1,68)
(5,0,4,41,0,296,1,1,73) (5,0,4,41,0,380,1,1,73) (5,0,4,41,1,296,2,2,62) (5,0,4,41,1,380,2,2,62)
(5,0,5,0,0,296,2,2,75) (5,0,5,0,0,380,2,2,75) (5,0,5,0,1,296,1,1,65) (5,0,5,0,1,380,1,1,65)
(5,0,5,10,0,296,1,1,75) (5,0,5,10,0,380,1,1,75) (5,0,5,10,1,296,1,1,66) (5,0,5,10,1,380,1,1,66)
(5,0,5,21,0,296,1,3,76) (5,0,5,21,0,380,1,3,76) (5,0,5,21,1,296,1,1,66) (5,0,5,21,1,380,1,1,66)
(5,0,5,30,0,296,2,2,77) (5,0,5,30,0,380,2,2,77) (5,0,5,30,1,296,2,2,65) (5,0,5,30,1,380,2,2,65)
(5,0,5,41,0,296,2,2,75) (5,0,5,41,0,380,2,2,75) (5,0,5,41,1,296,1,1,65) (5,0,5,41,1,380,1,1,65)
(5,0,6,0,0,296,4,4,76) (5,0,6,0,0,380,4,4,76) (5,0,6,0,1,296,1,1,62) (5,0,6,0,1,380,1,1,62)
(5,0,6,10,0,296,2,2,76) (5,0,6,10,0,380,2,2,76) (5,0,6,10,1,296,1,1,62) (5,0,6,10,1,380,1,1,62)
(5,0,6,21,0,296,2,2,76) (5,0,6,21,0,380,2,2,76) (5,0,6,21,1,296,1,1,60) (5,0,6,21,1,380,1,1,60)
(5,0,6,30,0,296,1,1,75) (5,0,6,30,0,380,1,1,75) (5,0,6,30,1,296,2,2,61) (5,0,6,30,1,380,2,2,61)
(5,0,6,41,0,296,1,1,76) (5,0,6,41,0,380,1,1,76) (5,0,6,41,1,296,1,1,60) (5,0,6,41,1,380,1,1,60)
(5,0,7,0,0,296,2,2,79) (5,0,7,0,0,380,2,2,79) (5,0,7,0,1,296,0,0,72) (5,0,7,0,1,380,0,0,72)
(5,0,7,10,0,296,2,3,76) (5,0,7,10,0,380,2,3,76) (5,0,7,10,1,296,1,3,68) (5,0,7,10,1,380,1,3,68)
(5,0,7,21,0,296,1,2,77) (5,0,7,21,0,380,1,2,77) (5,0,7,21,1,296,1,3,64) (5,0,7,21,1,380,1,3,64)
(5,0,7,30,0,296,1,1,79) (5,0,7,30,0,380,1,1,79) (5,0,7,30,1,296,0,0,70) (5,0,7,30,1,380,0,0,70)
(5,0,7,41,0,296,1,1,77) (5,0,7,41,0,380,1,1,77) (5,0,7,41,1,296,1,4,66) (5,0,7,41,1,380,1,4,66)
(5,0,8,0,0,296,5,5,72) (5,0,8,0,0,380,5,5,72) (5,0,8,0,1,296,0,0,70) (5,0,8,0,1,380,0,0,70)
(5,0,8,10,0,296,3,4,74) (5,0,8,10,0,380,3,4,74) (5,0,8,10,1,296,5,5,59) (5,0,8,10,1,380,5,5,59)
(5,0,8,21,0,296,1,1,78) (5,0,8,21,0,380,1,1,78) (5,0,8,21,1,296,2,2,65) (5,0,8,21,1,380,2,2,65)
(5,0,8,30,0,296,3,3,76) (5,0,8,30,0,380,3,3,76) (5,0,8,30,1,296,2,2,60) (5,0,8,30,1,380,2,2,60)
(5,0,8,41,0,296,1,1,76) (5,0,8,41,0,380,1,1,76) (5,0,8,41,1,296,1,2,68) (5,0,8,41,1,380,1,2,68)
(5,0,9,0,0,296,3,3,77) (5,0,9,0,0,380,3,3,77) (5,0,9,0,1,296,0,0,60) (5,0,9,0,1,380,0,0,60)
(5,0,9,10,0,296,5,6,74) (5,0,9,10,0,380,5,6,74) (5,0,9,10,1,296,0,0,60) (5,0,9,10,1,380,0,0,60)
(5,0,9,21,0,296,1,1,75) (5,0,9,21,0,380,1,1,75) (5,0,9,21,1,296,0,0,64) (5,0,9,21,1,380,0,0,64)
(5,0,9,30,0,296,3,3,77) (5,0,9,30,0,380,3,3,77) (5,0,9,30,1,296,0,0,64) (5,0,9,30,1,380,0,0,64)
(5,0,9,41,0,296,3,3,73) (5,0,9,41,0,380,3,3,73) (5,0,9,41,1,296,1,1,59) (5,0,9,41,1,380,1,1,59)
(5,1,0,0,0,296,1,1,41) (5,1,0,0,0,380,1,1,41) (5,1,0,0,1,296,1,1,34) (5,1,0,0,1,380,1,1,34)
(5,1,0,10,0,296,1,1,42) (5,1,0,10,0,380,1,1,42) (5,1,0,10,1,296,2,2,36) (5,1,0,10,1,380,2,2,36)
(5,1,0,21,0,296,1,1,42) (5,1,0,21,0,380,1,1,42) (5,1,0,21,1,296,2,3,33) (5,1,0,21,1,380,2,3,33)
(5,1,0,30,0,296,1,1,42) (5,1,0,30,0,380,1,1,42) (5,1,0,30,1,296,1,1,33) (5,1,0,30,1,380,1,1,33)
(5,1,0,41,0,296,1,1,42) (5,1,0,41,0,380,1,1,42) (5,1,0,41,1,296,2,2,33) (5,1,0,41,1,380,2,2,33)
(5,1,1,0,0,296,2,2,43) (5,1,1,0,0,380,2,2,43) (5,1,1,0,1,296,2,2,35) (5,1,1,0,1,380,2,2,35)
(5,1,1,10,0,296,1,1,41) (5,1,1,10,0,380,1,1,41) (5,1,1,10,1,296,1,1,36) (5,1,1,10,1,380,1,1,36)
(5,1,1,21,0,296,1,1,42) (5,1,1,21,0,380,1,1,42) (5,1,1,21,1,296,1,1,36) (5,1,1,21,1,380,1,1,36)
(5,1,1,30,0,296,2,2,42) (5,1,1,30,0,380,2,2,42) (5,1,1,30,1,296,2,3,36) (5,1,1,30,1,380,2,3,36)
(5,1,1,41,0,296,3,3,42) (5,1,1,41,0,380,3,3,42) (5,1,1,41,1,296,1,1,36) (5,1,1,41,1,380,1,1,36)
(5,1,2,0,0,296,2,2,41) (5,1,2,0,0,380,2,2,41) (5,1,2,0,1,296,1,1,36) (5,1,2,0,1,380,1,1,36)
(5,1,2,10,0,296,1,1,42) (5,1,2,10,0,380,1,1,42) (5,1,2,10,1,296,2,3,37) (5,1,2,10,1,380,2,3,37)
(5,1,2,21,0,296,2,2,43) (5,1,2,21,0,380,2,2,43) (5,1,2,21,1,296,3,3,37) (5,1,2,21,1,380,3,3,37)
(5,1,2,30,0,296,2,2,43) (5,1,2,30,0,380,2,2,43) (5,1,2,30,1,296,4,4,35) (5,1,2,30,1,380,4,4,35)
(5,1,2,41,0,296,3,3,42) (5,1,2,41,0,380,3,3,42) (5,1,2,41,1,296,2,2,35) (5,1,2,41,1,380,2,2,35)
(5,1,3,0,0,296,2,2,41) (5,1,3,0,0,380,2,2,41) (5,1,3,0,1,296,2,2,37) (5,1,3,0,1,380,2,2,37)
(5,1,3,10,0,296,4,4,41) (5,1,3,10,0,380,4,4,41) (5,1,3,10,1,296,3,3,38) (5,1,3,10,1,380,3,3,38)
(5,1,3,21,0,296,2,2,42) (5,1,3,21,0,380,2,2,42) (5,1,3,21,1,296,1,1,35) (5,1,3,21,1,380,1,1,35)
(5,1,3,30,0,296,2,2,42) (5,1,3,30,0,380,2,2,42) (5,1,3,30,1,296,1,1,36) (5,1,3,30,1,380,1,1,36)
(5,1,3,41,0,296,2,2,40) (5,1,3,41,0,380,2,2,40) (5,1,3,41,1,296,2,2,36) (5,1,3,41,1,380,2,2,36)
(5,1,4,0,0,296,2,2,42) (5,1,4,0,0,380,2,2,42) (5,1,4,0,1,296,4,5,37) (5,1,4,0,1,380,4,5,37)
(5,1,4,10,0,296,2,2,43) (5,1,4,10,0,380,2,2,43) (5,1,4,10,1,296,1,1,37) (5,1,4,10,1,380,1,1,37)
(5,1,4,21,0,296,1,1,42) (5,1,4,21,0,380,1,1,42) (5,1,4,21,1,296,1,1,38) (5,1,4,21,1,380,1,1,38)
(5,1,4,30,0,296,2,2,40) (5,1,4,30,0,380,2,2,40) (5,1,4,30,1,296,1,1,36) (5,1,4,30,1,380,1,1,36)
(5,1,4,41,0,296,2,2,42) (5,1,4,41,0,380,2,2,42) (5,1,4,41,1,296,2,2,35) (5,1,4,41,1,380,2,2,35)
(5,1,5,0,0,296,3,3,42) (5,1,5,0,0,380,3,3,42) (5,1,5,0,1,296,3,3,36) (5,1,5,0,1,380,3,3,36)
(5,1,5,10,0,296,2,2,43) (5,1,5,10,0,380,2,2,43) (5,1,5,10,1,296,1,1,37) (5,1,5,10,1,380,1,1,37)
(5,1,5,21,0,296,5,5,42) (5,1,5,21,0,380,5,5,42) (5,1,5,21,1,296,5,8,37) (5,1,5,21,1,380,5,8,37)
(5,1,5,30,0,296,1,1,43) (5,1,5,30,0,380,1,1,43) (5,1,5,30,1,296,2,2,36) (5,1,5,30,1,380,2,2,36)
(5,1,5,41,0,296,2,2,43) (5,1,5,41,0,380,2,2,43) (5,1,5,41,1,296,1,1,36) (5,1,5,41,1,380,1,1,36)
(5,1,6,0,0,296,2,2,42) (5,1,6,0,0,380,2,2,42) (5,1,6,0,1,296,3,3,37) (5,1,6,0,1,380,3,3,37)

(5,1,6,10,0,296,3,3,43) (5,1,6,10,0,380,3,3,43) (5,1,6,10,1,296,3,4,37) (5,1,6,10,1,380,3,4,37)
(5,1,6,21,0,296,1,1,41) (5,1,6,21,0,380,1,1,41) (5,1,6,21,1,296,2,4,37) (5,1,6,21,1,380,2,4,37)
(5,1,6,30,0,296,1,1,43) (5,1,6,30,0,380,1,1,43) (5,1,6,30,1,296,1,1,37) (5,1,6,30,1,380,1,1,37)
(5,1,6,41,0,296,2,2,42) (5,1,6,41,0,380,2,2,42) (5,1,6,41,1,296,3,3,37) (5,1,6,41,1,380,3,3,37)
(5,1,7,0,0,296,1,1,44) (5,1,7,0,0,380,1,1,44) (5,1,7,0,1,296,1,1,37) (5,1,7,0,1,380,1,1,37)
(5,1,7,10,0,296,1,1,44) (5,1,7,10,0,380,1,1,44) (5,1,7,10,1,296,2,2,37) (5,1,7,10,1,380,2,2,37)
(5,1,7,21,0,296,4,4,42) (5,1,7,21,0,380,4,4,42) (5,1,7,21,1,296,2,2,37) (5,1,7,21,1,380,2,2,37)
(5,1,7,30,0,296,3,3,44) (5,1,7,30,0,380,3,3,44) (5,1,7,30,1,296,4,6,37) (5,1,7,30,1,380,4,6,37)
(5,1,7,41,0,296,1,1,43) (5,1,7,41,0,380,1,1,43) (5,1,7,41,1,296,3,4,36) (5,1,7,41,1,380,3,4,36)
(5,1,8,0,0,296,2,2,42) (5,1,8,0,0,380,2,2,42) (5,1,8,0,1,296,1,3,36) (5,1,8,0,1,380,1,3,36)
(5,1,8,10,0,296,1,1,43) (5,1,8,10,0,380,1,1,43) (5,1,8,10,1,296,2,2,37) (5,1,8,10,1,380,2,2,37)
(5,1,8,21,0,296,2,2,43) (5,1,8,21,0,380,2,2,43) (5,1,8,21,1,296,2,4,36) (5,1,8,21,1,380,2,4,36)
(5,1,8,30,0,296,1,1,42) (5,1,8,30,0,380,1,1,42) (5,1,8,30,1,296,1,1,38) (5,1,8,30,1,380,1,1,38)
(5,1,8,41,0,296,1,1,42) (5,1,8,41,0,380,1,1,42) (5,1,8,41,1,296,2,3,38) (5,1,8,41,1,380,2,3,38)
(5,1,9,0,0,296,2,2,42) (5,1,9,0,0,380,2,2,42) (5,1,9,0,1,296,3,5,35) (5,1,9,0,1,380,3,5,35)
(5,1,9,10,0,296,2,2,42) (5,1,9,10,0,380,2,2,42) (5,1,9,10,1,296,2,3,37) (5,1,9,10,1,380,2,3,37)
(5,1,9,21,0,296,2,2,43) (5,1,9,21,0,380,2,2,43) (5,1,9,21,1,296,1,1,36) (5,1,9,21,1,380,1,1,36)
(5,1,9,30,0,296,1,1,40) (5,1,9,30,0,380,1,1,40) (5,1,9,30,1,296,1,1,37) (5,1,9,30,1,380,1,1,37)
(5,1,9,41,0,296,1,1,42) (5,1,9,41,0,380,1,1,42) (5,1,9,41,1,296,3,4,37) (5,1,9,41,1,380,3,4,37)
(4,0,0,0,0,296,0,0,70) (4,0,0,0,0,380,0,0,70) (4,0,0,0,1,296,1,1,58) (4,0,0,0,1,380,1,1,58)
(4,0,0,10,0,296,2,2,73) (4,0,0,10,0,380,2,2,73) (4,0,0,10,1,296,1,1,58) (4,0,0,10,1,380,1,1,58)
(4,0,0,21,0,296,1,1,72) (4,0,0,21,0,380,1,1,72) (4,0,0,21,1,296,0,0,58) (4,0,0,21,1,380,0,0,58)
(4,0,0,30,0,296,1,1,69) (4,0,0,30,0,380,1,1,69) (4,0,0,30,1,296,0,0,58) (4,0,0,30,1,380,0,0,58)
(4,0,0,41,0,296,1,1,71) (4,0,0,41,0,380,1,1,71) (4,0,0,41,1,296,0,0,58) (4,0,0,41,1,380,0,0,58)
(4,0,1,0,0,296,3,3,72) (4,0,1,0,0,380,3,3,72) (4,0,1,0,1,296,0,0,58) (4,0,1,0,1,380,0,0,58)
(4,0,1,10,0,296,3,4,69) (4,0,1,10,0,380,3,4,69) (4,0,1,10,1,296,1,1,57) (4,0,1,10,1,380,1,1,57)
(4,0,1,21,0,296,2,2,71) (4,0,1,21,0,380,2,2,71) (4,0,1,21,1,296,1,1,58) (4,0,1,21,1,380,1,1,58)
(4,0,1,30,0,296,1,1,72) (4,0,1,30,0,380,1,1,72) (4,0,1,30,1,296,0,0,58) (4,0,1,30,1,380,0,0,58)
(4,0,1,41,0,296,0,0,72) (4,0,1,41,0,380,0,0,72) (4,0,1,41,1,296,1,1,58) (4,0,1,41,1,380,1,1,58)
(4,0,2,0,0,296,1,1,72) (4,0,2,0,0,380,1,1,72) (4,0,2,0,1,296,1,1,57) (4,0,2,0,1,380,1,1,57)
(4,0,2,10,0,296,3,4,70) (4,0,2,10,0,380,3,4,70) (4,0,2,10,1,296,2,2,59) (4,0,2,10,1,380,2,2,59)
(4,0,2,21,0,296,5,6,72) (4,0,2,21,0,380,5,6,72) (4,0,2,21,1,296,1,1,57) (4,0,2,21,1,380,1,1,57)
(4,0,2,30,0,296,1,4,71) (4,0,2,30,0,380,1,4,71) (4,0,2,30,1,296,1,1,59) (4,0,2,30,1,380,1,1,59)
(4,0,2,41,0,296,1,1,71) (4,0,2,41,0,380,1,1,71) (4,0,2,41,1,296,1,1,60) (4,0,2,41,1,380,1,1,60)
(4,0,3,0,0,296,2,2,71) (4,0,3,0,0,380,2,2,71) (4,0,3,0,1,296,2,4,56) (4,0,3,0,1,380,2,4,56)
(4,0,3,10,0,296,4,4,73) (4,0,3,10,0,380,4,4,73) (4,0,3,10,1,296,2,2,57) (4,0,3,10,1,380,2,2,57)
(4,0,3,21,0,296,2,2,72) (4,0,3,21,0,380,2,2,72) (4,0,3,21,1,296,6,6,55) (4,0,3,21,1,380,6,6,55)
(4,0,3,30,0,296,1,2,73) (4,0,3,30,0,380,1,2,73) (4,0,3,30,1,296,2,2,58) (4,0,3,30,1,380,2,2,58)
(4,0,3,41,0,296,4,5,74) (4,0,3,41,0,380,4,5,74) (4,0,3,41,1,296,4,4,57) (4,0,3,41,1,380,4,4,57)
(4,0,4,0,0,296,4,8,69) (4,0,4,0,0,380,4,8,69) (4,0,4,0,1,296,0,0,58) (4,0,4,0,1,380,0,0,58)
(4,0,4,10,0,296,0,0,74) (4,0,4,10,0,380,0,0,74) (4,0,4,10,1,296,3,3,58) (4,0,4,10,1,380,3,3,58)
(4,0,4,21,0,296,1,4,73) (4,0,4,21,0,380,1,4,73) (4,0,4,21,1,296,1,1,58) (4,0,4,21,1,380,1,1,58)
(4,0,4,30,0,296,1,1,69) (4,0,4,30,0,380,1,1,69) (4,0,4,30,1,296,1,1,57) (4,0,4,30,1,380,1,1,57)
(4,0,4,41,0,296,1,1,74) (4,0,4,41,0,380,1,1,74) (4,0,4,41,1,296,3,3,57) (4,0,4,41,1,380,3,3,57)
(4,0,5,0,0,296,3,4,72) (4,0,5,0,0,380,3,4,72) (4,0,5,0,1,296,1,1,59) (4,0,5,0,1,380,1,1,59)
(4,0,5,10,0,296,1,1,70) (4,0,5,10,0,380,1,1,70) (4,0,5,10,1,296,0,0,56) (4,0,5,10,1,380,0,0,56)
(4,0,5,21,0,296,2,3,72) (4,0,5,21,0,380,2,3,72) (4,0,5,21,1,296,1,1,58) (4,0,5,21,1,380,1,1,58)
(4,0,5,30,0,296,1,1,72) (4,0,5,30,0,380,1,1,72) (4,0,5,30,1,296,0,0,58) (4,0,5,30,1,380,0,0,58)
(4,0,5,41,0,296,1,1,72) (4,0,5,41,0,380,1,1,72) (4,0,5,41,1,296,0,0,58) (4,0,5,41,1,380,0,0,58)
(4,0,6,0,0,296,2,5,71) (4,0,6,0,0,380,2,5,71) (4,0,6,0,1,296,0,0,60) (4,0,6,0,1,380,0,0,60)
(4,0,6,10,0,296,4,4,70) (4,0,6,10,0,380,4,4,70) (4,0,6,10,1,296,3,3,56) (4,0,6,10,1,380,3,3,56)
(4,0,6,21,0,296,3,3,70) (4,0,6,21,0,380,3,3,70) (4,0,6,21,1,296,1,1,55) (4,0,6,21,1,380,1,1,55)
(4,0,6,30,0,296,3,3,68) (4,0,6,30,0,380,3,3,68) (4,0,6,30,1,296,2,2,57) (4,0,6,30,1,380,2,2,57)
(4,0,6,41,0,296,3,7,70) (4,0,6,41,0,380,3,7,70) (4,0,6,41,1,296,0,0,58) (4,0,6,41,1,380,0,0,58)
(4,0,7,0,0,296,1,1,73) (4,0,7,0,0,380,1,1,73) (4,0,7,0,1,296,2,2,58) (4,0,7,0,1,380,2,2,58)
(4,0,7,10,0,296,1,1,74) (4,0,7,10,0,380,1,1,74) (4,0,7,10,1,296,2,2,57) (4,0,7,10,1,380,2,2,57)
(4,0,7,21,0,296,2,2,71) (4,0,7,21,0,380,2,2,71) (4,0,7,21,1,296,2,2,59) (4,0,7,21,1,380,2,2,59)
(4,0,7,30,0,296,1,1,75) (4,0,7,30,0,380,1,1,75) (4,0,7,30,1,296,1,1,57) (4,0,7,30,1,380,1,1,57)
(4,0,7,41,0,296,3,4,71) (4,0,7,41,0,380,3,4,71) (4,0,7,41,1,296,1,1,59) (4,0,7,41,1,380,1,1,59)
(4,0,8,0,0,296,0,0,72) (4,0,8,0,0,380,0,0,72) (4,0,8,0,1,296,2,2,57) (4,0,8,0,1,380,2,2,57)
(4,0,8,10,0,296,5,5,69) (4,0,8,10,0,380,5,5,69) (4,0,8,10,1,296,1,1,57) (4,0,8,10,1,380,1,1,57)
(4,0,8,21,0,296,1,3,75) (4,0,8,21,0,380,1,3,75) (4,0,8,21,1,296,2,2,58) (4,0,8,21,1,380,2,2,58)
(4,0,8,30,0,296,3,4,72) (4,0,8,30,0,380,3,4,72) (4,0,8,30,1,296,0,0,58) (4,0,8,30,1,380,0,0,58)
(4,0,8,41,0,296,1,1,73) (4,0,8,41,0,380,1,1,73) (4,0,8,41,1,296,1,1,57) (4,0,8,41,1,380,1,1,57)
(4,0,9,0,0,296,1,1,72) (4,0,9,0,0,380,1,1,72) (4,0,9,0,1,296,0,0,58) (4,0,9,0,1,380,0,0,58)
(4,0,9,10,0,296,3,6,68) (4,0,9,10,0,380,3,6,68) (4,0,9,10,1,296,0,0,58) (4,0,9,10,1,380,0,0,58)

(4,0,9,21,0,296,3,4,71) (4,0,9,21,0,380,3,4,71) (4,0,9,21,1,296,2,2,59) (4,0,9,21,1,380,2,2,59)
(4,0,9,30,0,296,2,4,73) (4,0,9,30,0,380,2,4,73) (4,0,9,30,1,296,0,0,58) (4,0,9,30,1,380,0,0,58)
(4,0,9,41,0,296,2,7,71) (4,0,9,41,0,380,2,7,71) (4,0,9,41,1,296,2,2,57) (4,0,9,41,1,380,2,2,57)
(4,1,0,0,0,296,1,1,39) (4,1,0,0,0,380,1,1,39) (4,1,0,0,1,296,2,2,31) (4,1,0,0,1,380,2,2,31)
(4,1,0,10,0,296,1,1,40) (4,1,0,10,0,380,1,1,40) (4,1,0,10,1,296,1,1,33) (4,1,0,10,1,380,1,1,33)
(4,1,0,21,0,296,1,1,37) (4,1,0,21,0,380,1,1,37) (4,1,0,21,1,296,1,1,31) (4,1,0,21,1,380,1,1,31)
(4,1,0,30,0,296,1,1,41) (4,1,0,30,0,380,1,1,41) (4,1,0,30,1,296,1,1,30) (4,1,0,30,1,380,1,1,30)
(4,1,0,41,0,296,1,1,38) (4,1,0,41,0,380,1,1,38) (4,1,0,41,1,296,1,1,32) (4,1,0,41,1,380,1,1,32)
(4,1,1,0,0,296,1,1,40) (4,1,1,0,0,380,1,1,40) (4,1,1,0,1,296,2,2,32) (4,1,1,0,1,380,2,2,32)
(4,1,1,10,0,296,1,1,39) (4,1,1,10,0,380,1,1,39) (4,1,1,10,1,296,1,1,32) (4,1,1,10,1,380,1,1,32)
(4,1,1,21,0,296,2,2,39) (4,1,1,21,0,380,2,2,39) (4,1,1,21,1,296,2,2,32) (4,1,1,21,1,380,2,2,32)
(4,1,1,30,0,296,1,1,40) (4,1,1,30,0,380,1,1,40) (4,1,1,30,1,296,3,3,33) (4,1,1,30,1,380,3,3,33)
(4,1,1,41,0,296,2,2,40) (4,1,1,41,0,380,2,2,40) (4,1,1,41,1,296,4,4,33) (4,1,1,41,1,380,4,4,33)
(4,1,2,0,0,296,1,1,40) (4,1,2,0,0,380,1,1,40) (4,1,2,0,1,296,0,0,32) (4,1,2,0,1,380,0,0,32)
(4,1,2,10,0,296,2,2,39) (4,1,2,10,0,380,2,2,39) (4,1,2,10,1,296,1,2,31) (4,1,2,10,1,380,1,2,31)
(4,1,2,21,0,296,2,2,40) (4,1,2,21,0,380,2,2,40) (4,1,2,21,1,296,0,0,31) (4,1,2,21,1,380,0,0,31)
(4,1,2,30,0,296,2,2,40) (4,1,2,30,0,380,2,2,40) (4,1,2,30,1,296,1,2,31) (4,1,2,30,1,380,1,2,31)
(4,1,2,41,0,296,2,2,39) (4,1,2,41,0,380,2,2,39) (4,1,2,41,1,296,1,1,32) (4,1,2,41,1,380,1,1,32)
(4,1,3,0,0,296,2,2,39) (4,1,3,0,0,380,2,2,39) (4,1,3,0,1,296,1,1,31) (4,1,3,0,1,380,1,1,31)
(4,1,3,10,0,296,2,2,40) (4,1,3,10,0,380,2,2,40) (4,1,3,10,1,296,1,1,32) (4,1,3,10,1,380,1,1,32)
(4,1,3,21,0,296,1,1,38) (4,1,3,21,0,380,1,1,38) (4,1,3,21,1,296,3,4,32) (4,1,3,21,1,380,3,4,32)
(4,1,3,30,0,296,2,2,40) (4,1,3,30,0,380,2,2,40) (4,1,3,30,1,296,4,4,32) (4,1,3,30,1,380,4,4,32)
(4,1,3,41,0,296,2,2,38) (4,1,3,41,0,380,2,2,38) (4,1,3,41,1,296,2,2,30) (4,1,3,41,1,380,2,2,30)
(4,1,4,0,0,296,1,1,41) (4,1,4,0,0,380,1,1,41) (4,1,4,0,1,296,2,2,29) (4,1,4,0,1,380,2,2,29)
(4,1,4,10,0,296,1,1,40) (4,1,4,10,0,380,1,1,40) (4,1,4,10,1,296,2,2,32) (4,1,4,10,1,380,2,2,32)
(4,1,4,21,0,296,0,0,39) (4,1,4,21,0,380,0,0,39) (4,1,4,21,1,296,2,3,32) (4,1,4,21,1,380,2,3,32)
(4,1,4,30,0,296,2,2,38) (4,1,4,30,0,380,2,2,38) (4,1,4,30,1,296,2,2,31) (4,1,4,30,1,380,2,2,31)
(4,1,4,41,0,296,1,1,38) (4,1,4,41,0,380,1,1,38) (4,1,4,41,1,296,1,1,31) (4,1,4,41,1,380,1,1,31)
(4,1,5,0,0,296,1,1,42) (4,1,5,0,0,380,1,1,42) (4,1,5,0,1,296,2,2,32) (4,1,5,0,1,380,2,2,32)
(4,1,5,10,0,296,2,2,40) (4,1,5,10,0,380,2,2,40) (4,1,5,10,1,296,1,1,33) (4,1,5,10,1,380,1,1,33)
(4,1,5,21,0,296,1,1,40) (4,1,5,21,0,380,1,1,40) (4,1,5,21,1,296,2,2,33) (4,1,5,21,1,380,2,2,33)
(4,1,5,30,0,296,1,1,40) (4,1,5,30,0,380,1,1,40) (4,1,5,30,1,296,3,3,31) (4,1,5,30,1,380,3,3,31)
(4,1,5,41,0,296,1,1,40) (4,1,5,41,0,380,1,1,40) (4,1,5,41,1,296,2,2,32) (4,1,5,41,1,380,2,2,32)
(4,1,6,0,0,296,2,2,41) (4,1,6,0,0,380,2,2,41) (4,1,6,0,1,296,2,2,30) (4,1,6,0,1,380,2,2,30)
(4,1,6,10,0,296,2,2,40) (4,1,6,10,0,380,2,2,40) (4,1,6,10,1,296,2,2,31) (4,1,6,10,1,380,2,2,31)
(4,1,6,21,0,296,1,1,40) (4,1,6,21,0,380,1,1,40) (4,1,6,21,1,296,2,2,30) (4,1,6,21,1,380,2,2,30)
(4,1,6,30,0,296,0,0,40) (4,1,6,30,0,380,0,0,40) (4,1,6,30,1,296,4,4,32) (4,1,6,30,1,380,4,4,32)
(4,1,6,41,0,296,1,1,40) (4,1,6,41,0,380,1,1,40) (4,1,6,41,1,296,3,4,32) (4,1,6,41,1,380,3,4,32)
(4,1,7,0,0,296,3,3,41) (4,1,7,0,0,380,3,3,41) (4,1,7,0,1,296,0,0,33) (4,1,7,0,1,380,0,0,33)
(4,1,7,10,0,296,1,1,40) (4,1,7,10,0,380,1,1,40) (4,1,7,10,1,296,1,1,33) (4,1,7,10,1,380,1,1,33)
(4,1,7,21,0,296,1,1,42) (4,1,7,21,0,380,1,1,42) (4,1,7,21,1,296,2,3,32) (4,1,7,21,1,380,2,3,32)
(4,1,7,30,0,296,1,1,41) (4,1,7,30,0,380,1,1,41) (4,1,7,30,1,296,0,0,34) (4,1,7,30,1,380,0,0,34)
(4,1,7,41,0,296,1,1,43) (4,1,7,41,0,380,1,1,43) (4,1,7,41,1,296,1,3,34) (4,1,7,41,1,380,1,3,34)
(4,1,8,0,0,296,2,2,39) (4,1,8,0,0,380,2,2,39) (4,1,8,0,1,296,1,1,35) (4,1,8,0,1,380,1,1,35)
(4,1,8,10,0,296,1,1,40) (4,1,8,10,0,380,1,1,40) (4,1,8,10,1,296,5,6,32) (4,1,8,10,1,380,5,6,32)
(4,1,8,21,0,296,0,0,41) (4,1,8,21,0,380,0,0,41) (4,1,8,21,1,296,2,5,35) (4,1,8,21,1,380,2,5,35)
(4,1,8,30,0,296,1,1,39) (4,1,8,30,0,380,1,1,39) (4,1,8,30,1,296,2,3,35) (4,1,8,30,1,380,2,3,35)
(4,1,8,41,0,296,1,1,40) (4,1,8,41,0,380,1,1,40) (4,1,8,41,1,296,1,1,35) (4,1,8,41,1,380,1,1,35)
(4,1,9,0,0,296,1,1,41) (4,1,9,0,0,380,1,1,41) (4,1,9,0,1,296,2,3,31) (4,1,9,0,1,380,2,3,31)
(4,1,9,10,0,296,2,2,41) (4,1,9,10,0,380,2,2,41) (4,1,9,10,1,296,1,1,32) (4,1,9,10,1,380,1,1,32)
(4,1,9,21,0,296,1,1,41) (4,1,9,21,0,380,1,1,41) (4,1,9,21,1,296,1,1,33) (4,1,9,21,1,380,1,1,33)
(4,1,9,30,0,296,0,0,40) (4,1,9,30,0,380,0,0,40) (4,1,9,30,1,296,1,1,33) (4,1,9,30,1,380,1,1,33)
(4,1,9,41,0,296,0,0,40) (4,1,9,41,0,380,0,0,40) (4,1,9,41,1,296,1,1,32) (4,1,9,41,1,380,1,1,32)
(3,0,0,0,0,296,0,0,64) (3,0,0,0,0,380,0,0,64) (3,0,0,0,1,296,0,0,58) (3,0,0,0,1,380,0,0,58)
(3,0,0,10,0,296,1,2,62) (3,0,0,10,0,380,1,2,62) (3,0,0,10,1,296,2,2,55) (3,0,0,10,1,380,2,2,55)
(3,0,0,21,0,296,1,1,64) (3,0,0,21,0,380,1,1,64) (3,0,0,21,1,296,0,0,58) (3,0,0,21,1,380,0,0,58)
(3,0,0,30,0,296,0,0,64) (3,0,0,30,0,380,0,0,64) (3,0,0,30,1,296,0,0,58) (3,0,0,30,1,380,0,0,58)
(3,0,0,41,0,296,1,1,67) (3,0,0,41,0,380,1,1,67) (3,0,0,41,1,296,3,3,57) (3,0,0,41,1,380,3,3,57)
(3,0,1,0,0,296,2,2,67) (3,0,1,0,0,380,2,2,67) (3,0,1,0,1,296,3,3,57) (3,0,1,0,1,380,3,3,57)
(3,0,1,10,0,296,1,2,61) (3,0,1,10,0,380,1,2,61) (3,0,1,10,1,296,0,0,58) (3,0,1,10,1,380,0,0,58)
(3,0,1,21,0,296,0,0,64) (3,0,1,21,0,380,0,0,64) (3,0,1,21,1,296,2,2,57) (3,0,1,21,1,380,2,2,57)
(3,0,1,30,0,296,2,4,65) (3,0,1,30,0,380,2,4,65) (3,0,1,30,1,296,1,1,57) (3,0,1,30,1,380,1,1,57)
(3,0,1,41,0,296,1,1,65) (3,0,1,41,0,380,1,1,65) (3,0,1,41,1,296,4,4,57) (3,0,1,41,1,380,4,4,57)
(3,0,2,0,0,296,1,2,62) (3,0,2,0,0,380,1,2,62) (3,0,2,0,1,296,0,0,58) (3,0,2,0,1,380,0,0,58)
(3,0,2,10,0,296,1,2,62) (3,0,2,10,0,380,1,2,62) (3,0,2,10,1,296,2,2,56) (3,0,2,10,1,380,2,2,56)
(3,0,2,21,0,296,0,0,64) (3,0,2,21,0,380,0,0,64) (3,0,2,21,1,296,3,5,57) (3,0,2,21,1,380,3,5,57)

(3,0,2,30,0,296,3,4,59) (3,0,2,30,0,380,3,4,59) (3,0,2,30,1,296,3,3,56) (3,0,2,30,1,380,3,3,56)
(3,0,2,41,0,296,4,8,62) (3,0,2,41,0,380,4,8,62) (3,0,2,41,1,296,0,0,58) (3,0,2,41,1,380,0,0,58)
(3,0,3,0,0,296,1,1,62) (3,0,3,0,0,380,1,1,62) (3,0,3,0,1,296,2,2,55) (3,0,3,0,1,380,2,2,55)
(3,0,3,10,0,296,1,1,67) (3,0,3,10,0,380,1,1,67) (3,0,3,10,1,296,2,2,57) (3,0,3,10,1,380,2,2,57)
(3,0,3,21,0,296,1,1,61) (3,0,3,21,0,380,1,1,61) (3,0,3,21,1,296,2,2,54) (3,0,3,21,1,380,2,2,54)
(3,0,3,30,0,296,2,2,65) (3,0,3,30,0,380,2,2,65) (3,0,3,30,1,296,0,0,58) (3,0,3,30,1,380,0,0,58)
(3,0,3,41,0,296,4,4,63) (3,0,3,41,0,380,4,4,63) (3,0,3,41,1,296,3,3,56) (3,0,3,41,1,380,3,3,56)
(3,0,4,0,0,296,3,3,66) (3,0,4,0,0,380,3,3,66) (3,0,4,0,1,296,1,2,58) (3,0,4,0,1,380,1,2,58)
(3,0,4,10,0,296,0,0,66) (3,0,4,10,0,380,0,0,66) (3,0,4,10,1,296,2,2,57) (3,0,4,10,1,380,2,2,57)
(3,0,4,21,0,296,3,3,65) (3,0,4,21,0,380,3,3,65) (3,0,4,21,1,296,1,1,57) (3,0,4,21,1,380,1,1,57)
(3,0,4,30,0,296,3,3,62) (3,0,4,30,0,380,3,3,62) (3,0,4,30,1,296,2,6,54) (3,0,4,30,1,380,2,6,54)
(3,0,4,41,0,296,3,4,61) (3,0,4,41,0,380,3,4,61) (3,0,4,41,1,296,1,1,57) (3,0,4,41,1,380,1,1,57)
(3,0,5,0,0,296,4,4,64) (3,0,5,0,0,380,4,4,64) (3,0,5,0,1,296,1,1,58) (3,0,5,0,1,380,1,1,58)
(3,0,5,10,0,296,2,2,64) (3,0,5,10,0,380,2,2,64) (3,0,5,10,1,296,3,3,56) (3,0,5,10,1,380,3,3,56)
(3,0,5,21,0,296,1,1,66) (3,0,5,21,0,380,1,1,66) (3,0,5,21,1,296,1,1,58) (3,0,5,21,1,380,1,1,58)
(3,0,5,30,0,296,0,0,64) (3,0,5,30,0,380,0,0,64) (3,0,5,30,1,296,1,1,56) (3,0,5,30,1,380,1,1,56)
(3,0,5,41,0,296,4,5,61) (3,0,5,41,0,380,4,5,61) (3,0,5,41,1,296,0,0,58) (3,0,5,41,1,380,0,0,58)
(3,0,6,0,0,296,3,3,65) (3,0,6,0,0,380,3,3,65) (3,0,6,0,1,296,0,0,58) (3,0,6,0,1,380,0,0,58)
(3,0,6,10,0,296,3,4,63) (3,0,6,10,0,380,3,4,63) (3,0,6,10,1,296,3,3,56) (3,0,6,10,1,380,3,3,56)
(3,0,6,21,0,296,1,1,64) (3,0,6,21,0,380,1,1,64) (3,0,6,21,1,296,1,1,57) (3,0,6,21,1,380,1,1,57)
(3,0,6,30,0,296,4,4,65) (3,0,6,30,0,380,4,4,65) (3,0,6,30,1,296,0,0,58) (3,0,6,30,1,380,0,0,58)
(3,0,6,41,0,296,2,2,63) (3,0,6,41,0,380,2,2,63) (3,0,6,41,1,296,1,1,57) (3,0,6,41,1,380,1,1,57)
(3,0,7,0,0,296,3,3,66) (3,0,7,0,0,380,3,3,66) (3,0,7,0,1,296,1,1,58) (3,0,7,0,1,380,1,1,58)
(3,0,7,10,0,296,6,6,63) (3,0,7,10,0,380,6,6,63) (3,0,7,10,1,296,1,1,57) (3,0,7,10,1,380,1,1,57)
(3,0,7,21,0,296,2,2,68) (3,0,7,21,0,380,2,2,68) (3,0,7,21,1,296,1,1,57) (3,0,7,21,1,380,1,1,57)
(3,0,7,30,0,296,0,0,66) (3,0,7,30,0,380,0,0,66) (3,0,7,30,1,296,0,0,58) (3,0,7,30,1,380,0,0,58)
(3,0,7,41,0,296,1,3,67) (3,0,7,41,0,380,1,3,67) (3,0,7,41,1,296,1,1,57) (3,0,7,41,1,380,1,1,57)
(3,0,8,0,0,296,4,6,65) (3,0,8,0,0,380,4,6,65) (3,0,8,0,1,296,1,1,58) (3,0,8,0,1,380,1,1,58)
(3,0,8,10,0,296,1,1,60) (3,0,8,10,0,380,1,1,60) (3,0,8,10,1,296,2,2,56) (3,0,8,10,1,380,2,2,56)
(3,0,8,21,0,296,1,3,62) (3,0,8,21,0,380,1,3,62) (3,0,8,21,1,296,2,2,58) (3,0,8,21,1,380,2,2,58)
(3,0,8,30,0,296,2,2,61) (3,0,8,30,0,380,2,2,61) (3,0,8,30,1,296,1,1,58) (3,0,8,30,1,380,1,1,58)
(3,0,8,41,0,296,2,2,61) (3,0,8,41,0,380,2,2,61) (3,0,8,41,1,296,2,2,57) (3,0,8,41,1,380,2,2,57)
(3,0,9,0,0,296,1,1,62) (3,0,9,0,0,380,1,1,62) (3,0,9,0,1,296,2,2,58) (3,0,9,0,1,380,2,2,58)
(3,0,9,10,0,296,1,1,63) (3,0,9,10,0,380,1,1,63) (3,0,9,10,1,296,1,1,57) (3,0,9,10,1,380,1,1,57)
(3,0,9,21,0,296,4,6,60) (3,0,9,21,0,380,4,6,60) (3,0,9,21,1,296,1,2,58) (3,0,9,21,1,380,1,2,58)
(3,0,9,30,0,296,1,3,60) (3,0,9,30,0,380,1,3,60) (3,0,9,30,1,296,0,0,58) (3,0,9,30,1,380,0,0,58)
(3,0,9,41,0,296,1,4,64) (3,0,9,41,0,380,1,4,64) (3,0,9,41,1,296,1,1,57) (3,0,9,41,1,380,1,1,57)
(3,1,0,0,0,296,1,1,37) (3,1,0,0,0,380,1,1,37) (3,1,0,0,1,296,1,1,29) (3,1,0,0,1,380,1,1,29)
(3,1,0,10,0,296,2,2,37) (3,1,0,10,0,380,2,2,37) (3,1,0,10,1,296,1,1,29) (3,1,0,10,1,380,1,1,29)
(3,1,0,21,0,296,2,2,37) (3,1,0,21,0,380,2,2,37) (3,1,0,21,1,296,1,1,29) (3,1,0,21,1,380,1,1,29)
(3,1,0,30,0,296,1,1,36) (3,1,0,30,0,380,1,1,36) (3,1,0,30,1,296,1,1,29) (3,1,0,30,1,380,1,1,29)
(3,1,0,41,0,296,2,2,36) (3,1,0,41,0,380,2,2,36) (3,1,0,41,1,296,1,1,29) (3,1,0,41,1,380,1,1,29)
(3,1,1,0,0,296,1,1,36) (3,1,1,0,0,380,1,1,36) (3,1,1,0,1,296,1,1,29) (3,1,1,0,1,380,1,1,29)
(3,1,1,10,0,296,1,1,37) (3,1,1,10,0,380,1,1,37) (3,1,1,10,1,296,2,2,29) (3,1,1,10,1,380,2,2,29)
(3,1,1,21,0,296,3,4,37) (3,1,1,21,0,380,3,4,37) (3,1,1,21,1,296,1,1,29) (3,1,1,21,1,380,1,1,29)
(3,1,1,30,0,296,1,1,38) (3,1,1,30,0,380,1,1,38) (3,1,1,30,1,296,1,1,29) (3,1,1,30,1,380,1,1,29)
(3,1,1,41,0,296,3,3,37) (3,1,1,41,0,380,3,3,37) (3,1,1,41,1,296,1,1,29) (3,1,1,41,1,380,1,1,29)
(3,1,2,0,0,296,1,1,37) (3,1,2,0,0,380,1,1,37) (3,1,2,0,1,296,1,1,30) (3,1,2,0,1,380,1,1,30)
(3,1,2,10,0,296,1,1,36) (3,1,2,10,0,380,1,1,36) (3,1,2,10,1,296,2,2,30) (3,1,2,10,1,380,2,2,30)
(3,1,2,21,0,296,2,2,36) (3,1,2,21,0,380,2,2,36) (3,1,2,21,1,296,1,1,29) (3,1,2,21,1,380,1,1,29)
(3,1,2,30,0,296,2,3,36) (3,1,2,30,0,380,2,3,36) (3,1,2,30,1,296,2,2,30) (3,1,2,30,1,380,2,2,30)
(3,1,2,41,0,296,1,1,36) (3,1,2,41,0,380,1,1,36) (3,1,2,41,1,296,1,1,29) (3,1,2,41,1,380,1,1,29)
(3,1,3,0,0,296,1,1,37) (3,1,3,0,0,380,1,1,37) (3,1,3,0,1,296,2,2,29) (3,1,3,0,1,380,2,2,29)
(3,1,3,10,0,296,2,2,36) (3,1,3,10,0,380,2,2,36) (3,1,3,10,1,296,1,1,29) (3,1,3,10,1,380,1,1,29)
(3,1,3,21,0,296,1,1,37) (3,1,3,21,0,380,1,1,37) (3,1,3,21,1,296,3,3,29) (3,1,3,21,1,380,3,3,29)
(3,1,3,30,0,296,2,2,38) (3,1,3,30,0,380,2,2,38) (3,1,3,30,1,296,2,2,29) (3,1,3,30,1,380,2,2,29)
(3,1,3,41,0,296,2,2,38) (3,1,3,41,0,380,2,2,38) (3,1,3,41,1,296,2,2,29) (3,1,3,41,1,380,2,2,29)
(3,1,4,0,0,296,1,1,37) (3,1,4,0,0,380,1,1,37) (3,1,4,0,1,296,1,1,29) (3,1,4,0,1,380,1,1,29)
(3,1,4,10,0,296,2,2,35) (3,1,4,10,0,380,2,2,35) (3,1,4,10,1,296,1,1,29) (3,1,4,10,1,380,1,1,29)
(3,1,4,21,0,296,2,2,37) (3,1,4,21,0,380,2,2,37) (3,1,4,21,1,296,2,3,30) (3,1,4,21,1,380,2,3,30)
(3,1,4,30,0,296,1,1,37) (3,1,4,30,0,380,1,1,37) (3,1,4,30,1,296,1,1,29) (3,1,4,30,1,380,1,1,29)
(3,1,4,41,0,296,2,2,37) (3,1,4,41,0,380,2,2,37) (3,1,4,41,1,296,2,2,29) (3,1,4,41,1,380,2,2,29)
(3,1,5,0,0,296,3,3,37) (3,1,5,0,0,380,3,3,37) (3,1,5,0,1,296,2,2,29) (3,1,5,0,1,380,2,2,29)
(3,1,5,10,0,296,2,2,36) (3,1,5,10,0,380,2,2,36) (3,1,5,10,1,296,1,1,28) (3,1,5,10,1,380,1,1,28)
(3,1,5,21,0,296,3,3,38) (3,1,5,21,0,380,3,3,38) (3,1,5,21,1,296,1,1,29) (3,1,5,21,1,380,1,1,29)
(3,1,5,30,0,296,1,1,37) (3,1,5,30,0,380,1,1,37) (3,1,5,30,1,296,1,1,29) (3,1,5,30,1,380,1,1,29)

(3,1,5,41,0,296,1,1,38) (3,1,5,41,0,380,1,1,38) (3,1,5,41,1,296,1,1,29) (3,1,5,41,1,380,1,1,29)
(3,1,6,0,0,296,1,1,38) (3,1,6,0,0,380,1,1,38) (3,1,6,0,1,296,1,1,29) (3,1,6,0,1,380,1,1,29)
(3,1,6,10,0,296,2,2,35) (3,1,6,10,0,380,2,2,35) (3,1,6,10,1,296,1,1,29) (3,1,6,10,1,380,1,1,29)
(3,1,6,21,0,296,1,1,37) (3,1,6,21,0,380,1,1,37) (3,1,6,21,1,296,1,1,28) (3,1,6,21,1,380,1,1,28)
(3,1,6,30,0,296,1,1,37) (3,1,6,30,0,380,1,1,37) (3,1,6,30,1,296,1,1,29) (3,1,6,30,1,380,1,1,29)
(3,1,6,41,0,296,1,1,36) (3,1,6,41,0,380,1,1,36) (3,1,6,41,1,296,1,1,29) (3,1,6,41,1,380,1,1,29)
(3,1,7,0,0,296,1,1,37) (3,1,7,0,0,380,1,1,37) (3,1,7,0,1,296,2,2,30) (3,1,7,0,1,380,2,2,30)
(3,1,7,10,0,296,1,1,38) (3,1,7,10,0,380,1,1,38) (3,1,7,10,1,296,2,2,30) (3,1,7,10,1,380,2,2,30)
(3,1,7,21,0,296,1,1,38) (3,1,7,21,0,380,1,1,38) (3,1,7,21,1,296,2,2,29) (3,1,7,21,1,380,2,2,29)
(3,1,7,30,0,296,1,1,41) (3,1,7,30,0,380,1,1,41) (3,1,7,30,1,296,1,1,30) (3,1,7,30,1,380,1,1,30)
(3,1,7,41,0,296,1,1,41) (3,1,7,41,0,380,1,1,41) (3,1,7,41,1,296,2,2,30) (3,1,7,41,1,380,2,2,30)
(3,1,8,0,0,296,2,2,37) (3,1,8,0,0,380,2,2,37) (3,1,8,0,1,296,1,1,29) (3,1,8,0,1,380,1,1,29)
(3,1,8,10,0,296,1,1,37) (3,1,8,10,0,380,1,1,37) (3,1,8,10,1,296,2,2,29) (3,1,8,10,1,380,2,2,29)
(3,1,8,21,0,296,1,1,38) (3,1,8,21,0,380,1,1,38) (3,1,8,21,1,296,2,2,29) (3,1,8,21,1,380,2,2,29)
(3,1,8,30,0,296,2,2,38) (3,1,8,30,0,380,2,2,38) (3,1,8,30,1,296,1,1,29) (3,1,8,30,1,380,1,1,29)
(3,1,8,41,0,296,5,5,37) (3,1,8,41,0,380,5,5,37) (3,1,8,41,1,296,3,3,29) (3,1,8,41,1,380,3,3,29)
(3,1,9,0,0,296,2,2,35) (3,1,9,0,0,380,2,2,35) (3,1,9,0,1,296,1,1,29) (3,1,9,0,1,380,1,1,29)
(3,1,9,10,0,296,2,2,37) (3,1,9,10,0,380,2,2,37) (3,1,9,10,1,296,1,1,29) (3,1,9,10,1,380,1,1,29)
(3,1,9,21,0,296,2,2,36) (3,1,9,21,0,380,2,2,36) (3,1,9,21,1,296,1,1,30) (3,1,9,21,1,380,1,1,30)
(3,1,9,30,0,296,1,1,37) (3,1,9,30,0,380,1,1,37) (3,1,9,30,1,296,1,1,29) (3,1,9,30,1,380,1,1,29)
(3,1,9,41,0,296,2,2,37) (3,1,9,41,0,380,2,2,37) (3,1,9,41,1,296,3,3,29) (3,1,9,41,1,380,3,3,29)
(2,0,0,0,0,296,1,1,58) (2,0,0,0,0,380,1,1,58) (2,0,0,0,1,296,2,2,58) (2,0,0,0,1,380,2,2,58)
(2,0,0,10,0,296,4,4,58) (2,0,0,10,0,380,4,4,58) (2,0,0,10,1,296,4,7,54) (2,0,0,10,1,380,4,7,54)
(2,0,0,21,0,296,1,1,59) (2,0,0,21,0,380,1,1,59) (2,0,0,21,1,296,1,2,57) (2,0,0,21,1,380,1,2,57)
(2,0,0,30,0,296,1,2,56) (2,0,0,30,0,380,1,2,56) (2,0,0,30,1,296,2,2,57) (2,0,0,30,1,380,2,2,57)
(2,0,0,41,0,296,0,0,58) (2,0,0,41,0,380,0,0,58) (2,0,0,41,1,296,3,3,57) (2,0,0,41,1,380,3,3,57)
(2,0,1,0,0,296,2,2,58) (2,0,1,0,0,380,2,2,58) (2,0,1,0,1,296,1,1,56) (2,0,1,0,1,380,1,1,56)
(2,0,1,10,0,296,2,2,57) (2,0,1,10,0,380,2,2,57) (2,0,1,10,1,296,1,4,55) (2,0,1,10,1,380,1,4,55)
(2,0,1,21,0,296,0,0,60) (2,0,1,21,0,380,0,0,60) (2,0,1,21,1,296,1,12,50) (2,0,1,21,1,380,1,12,50)
(2,0,1,30,0,296,2,2,58) (2,0,1,30,0,380,2,2,58) (2,0,1,30,1,296,1,6,52) (2,0,1,30,1,380,1,6,52)
(2,0,1,41,0,296,0,0,58) (2,0,1,41,0,380,0,0,58) (2,0,1,41,1,296,2,2,58) (2,0,1,41,1,380,2,2,58)
(2,0,2,0,0,296,3,5,56) (2,0,2,0,0,380,3,5,56) (2,0,2,0,1,296,2,8,54) (2,0,2,0,1,380,2,8,54)
(2,0,2,10,0,296,2,2,57) (2,0,2,10,0,380,2,2,57) (2,0,2,10,1,296,1,1,57) (2,0,2,10,1,380,1,1,57)
(2,0,2,21,0,296,3,3,59) (2,0,2,21,0,380,3,3,59) (2,0,2,21,1,296,0,0,58) (2,0,2,21,1,380,0,0,58)
(2,0,2,30,0,296,3,3,56) (2,0,2,30,0,380,3,3,56) (2,0,2,30,1,296,2,6,53) (2,0,2,30,1,380,2,6,53)
(2,0,2,41,0,296,2,2,58) (2,0,2,41,0,380,2,2,58) (2,0,2,41,1,296,1,1,57) (2,0,2,41,1,380,1,1,57)
(2,0,3,0,0,296,1,1,58) (2,0,3,0,0,380,1,1,58) (2,0,3,0,1,296,3,5,54) (2,0,3,0,1,380,3,5,54)
(2,0,3,10,0,296,1,1,58) (2,0,3,10,0,380,1,1,58) (2,0,3,10,1,296,1,3,54) (2,0,3,10,1,380,1,3,54)
(2,0,3,21,0,296,0,0,62) (2,0,3,21,0,380,0,0,62) (2,0,3,21,1,296,4,4,56) (2,0,3,21,1,380,4,4,56)
(2,0,3,30,0,296,4,4,59) (2,0,3,30,0,380,4,4,59) (2,0,3,30,1,296,1,2,54) (2,0,3,30,1,380,1,2,54)
(2,0,3,41,0,296,5,6,58) (2,0,3,41,0,380,5,6,58) (2,0,3,41,1,296,3,4,53) (2,0,3,41,1,380,3,4,53)
(2,0,4,0,0,296,0,0,60) (2,0,4,0,0,380,0,0,60) (2,0,4,0,1,296,2,2,57) (2,0,4,0,1,380,2,2,57)
(2,0,4,10,0,296,1,1,57) (2,0,4,10,0,380,1,1,57) (2,0,4,10,1,296,2,3,53) (2,0,4,10,1,380,2,3,53)
(2,0,4,21,0,296,1,1,59) (2,0,4,21,0,380,1,1,59) (2,0,4,21,1,296,1,1,56) (2,0,4,21,1,380,1,1,56)
(2,0,4,30,0,296,1,1,58) (2,0,4,30,0,380,1,1,58) (2,0,4,30,1,296,2,2,57) (2,0,4,30,1,380,2,2,57)
(2,0,4,41,0,296,1,1,58) (2,0,4,41,0,380,1,1,58) (2,0,4,41,1,296,0,0,58) (2,0,4,41,1,380,0,0,58)
(2,0,5,0,0,296,2,2,58) (2,0,5,0,0,380,2,2,58) (2,0,5,0,1,296,1,1,58) (2,0,5,0,1,380,1,1,58)
(2,0,5,10,0,296,1,1,58) (2,0,5,10,0,380,1,1,58) (2,0,5,10,1,296,4,4,53) (2,0,5,10,1,380,4,4,53)
(2,0,5,21,0,296,6,6,57) (2,0,5,21,0,380,6,6,57) (2,0,5,21,1,296,1,1,57) (2,0,5,21,1,380,1,1,57)
(2,0,5,30,0,296,0,0,60) (2,0,5,30,0,380,0,0,60) (2,0,5,30,1,296,1,1,58) (2,0,5,30,1,380,1,1,58)
(2,0,5,41,0,296,1,1,57) (2,0,5,41,0,380,1,1,57) (2,0,5,41,1,296,1,1,58) (2,0,5,41,1,380,1,1,58)
(2,0,6,0,0,296,1,1,60) (2,0,6,0,0,380,1,1,60) (2,0,6,0,1,296,4,5,56) (2,0,6,0,1,380,4,5,56)
(2,0,6,10,0,296,2,2,57) (2,0,6,10,0,380,2,2,57) (2,0,6,10,1,296,1,1,57) (2,0,6,10,1,380,1,1,57)
(2,0,6,21,0,296,4,4,59) (2,0,6,21,0,380,4,4,59) (2,0,6,21,1,296,2,2,57) (2,0,6,21,1,380,2,2,57)
(2,0,6,30,0,296,2,2,58) (2,0,6,30,0,380,2,2,58) (2,0,6,30,1,296,2,2,57) (2,0,6,30,1,380,2,2,57)
(2,0,6,41,0,296,0,0,60) (2,0,6,41,0,380,0,0,60) (2,0,6,41,1,296,2,3,56) (2,0,6,41,1,380,2,3,56)
(2,0,7,0,0,296,0,0,58) (2,0,7,0,0,380,0,0,58) (2,0,7,0,1,296,1,7,52) (2,0,7,0,1,380,1,7,52)
(2,0,7,10,0,296,2,2,58) (2,0,7,10,0,380,2,2,58) (2,0,7,10,1,296,1,7,52) (2,0,7,10,1,380,1,7,52)
(2,0,7,21,0,296,0,0,58) (2,0,7,21,0,380,0,0,58) (2,0,7,21,1,296,2,10,52) (2,0,7,21,1,380,2,10,52)
(2,0,7,30,0,296,0,0,58) (2,0,7,30,0,380,0,0,58) (2,0,7,30,1,296,2,3,56) (2,0,7,30,1,380,2,3,56)
(2,0,7,41,0,296,0,0,58) (2,0,7,41,0,380,0,0,58) (2,0,7,41,1,296,2,8,52) (2,0,7,41,1,380,2,8,52)
(2,0,8,0,0,296,2,2,56) (2,0,8,0,0,380,2,2,56) (2,0,8,0,1,296,0,0,58) (2,0,8,0,1,380,0,0,58)
(2,0,8,10,0,296,1,1,57) (2,0,8,10,0,380,1,1,57) (2,0,8,10,1,296,1,5,56) (2,0,8,10,1,380,1,5,56)
(2,0,8,21,0,296,4,4,59) (2,0,8,21,0,380,4,4,59) (2,0,8,21,1,296,2,7,54) (2,0,8,21,1,380,2,7,54)
(2,0,8,30,0,296,1,1,60) (2,0,8,30,0,380,1,1,60) (2,0,8,30,1,296,2,6,55) (2,0,8,30,1,380,2,6,55)
(2,0,8,41,0,296,1,1,59) (2,0,8,41,0,380,1,1,59) (2,0,8,41,1,296,1,4,54) (2,0,8,41,1,380,1,4,54)

(2,0,9,0,0,296,4,5,57) (2,0,9,0,0,380,4,5,57) (2,0,9,0,1,296,4,6,54) (2,0,9,0,1,380,4,6,54)
(2,0,9,10,0,296,2,2,56) (2,0,9,10,0,380,2,2,56) (2,0,9,10,1,296,1,8,54) (2,0,9,10,1,380,1,8,54)
(2,0,9,21,0,296,1,1,60) (2,0,9,21,0,380,1,1,60) (2,0,9,21,1,296,1,4,56) (2,0,9,21,1,380,1,4,56)
(2,0,9,30,0,296,3,3,57) (2,0,9,30,0,380,3,3,57) (2,0,9,30,1,296,1,3,57) (2,0,9,30,1,380,1,3,57)
(2,0,9,41,0,296,3,3,56) (2,0,9,41,0,380,3,3,56) (2,0,9,41,1,296,2,10,50) (2,0,9,41,1,380,2,10,50)
(2,1,0,0,0,296,2,2,33) (2,1,0,0,0,380,2,2,33) (2,1,0,0,1,296,1,1,29) (2,1,0,0,1,380,1,1,29)
(2,1,0,10,0,296,1,1,31) (2,1,0,10,0,380,1,1,31) (2,1,0,10,1,296,2,2,28) (2,1,0,10,1,380,2,2,28)
(2,1,0,21,0,296,1,1,32) (2,1,0,21,0,380,1,1,32) (2,1,0,21,1,296,1,1,29) (2,1,0,21,1,380,1,1,29)
(2,1,0,30,0,296,1,1,31) (2,1,0,30,0,380,1,1,31) (2,1,0,30,1,296,1,1,29) (2,1,0,30,1,380,1,1,29)
(2,1,0,41,0,296,2,2,31) (2,1,0,41,0,380,2,2,31) (2,1,0,41,1,296,2,2,29) (2,1,0,41,1,380,2,2,29)
(2,1,1,0,0,296,0,0,31) (2,1,1,0,0,380,0,0,31) (2,1,1,0,1,296,3,3,29) (2,1,1,0,1,380,3,3,29)
(2,1,1,10,0,296,0,0,32) (2,1,1,10,0,380,0,0,32) (2,1,1,10,1,296,1,1,29) (2,1,1,10,1,380,1,1,29)
(2,1,1,21,0,296,1,1,31) (2,1,1,21,0,380,1,1,31) (2,1,1,21,1,296,2,2,29) (2,1,1,21,1,380,2,2,29)
(2,1,1,30,0,296,2,2,30) (2,1,1,30,0,380,2,2,30) (2,1,1,30,1,296,1,1,29) (2,1,1,30,1,380,1,1,29)
(2,1,1,41,0,296,1,1,33) (2,1,1,41,0,380,1,1,33) (2,1,1,41,1,296,1,1,29) (2,1,1,41,1,380,1,1,29)
(2,1,2,0,0,296,1,1,33) (2,1,2,0,0,380,1,1,33) (2,1,2,0,1,296,2,2,29) (2,1,2,0,1,380,2,2,29)
(2,1,2,10,0,296,1,1,32) (2,1,2,10,0,380,1,1,32) (2,1,2,10,1,296,2,2,29) (2,1,2,10,1,380,2,2,29)
(2,1,2,21,0,296,1,1,33) (2,1,2,21,0,380,1,1,33) (2,1,2,21,1,296,0,0,29) (2,1,2,21,1,380,0,0,29)
(2,1,2,30,0,296,1,1,33) (2,1,2,30,0,380,1,1,33) (2,1,2,30,1,296,1,1,29) (2,1,2,30,1,380,1,1,29)
(2,1,2,41,0,296,2,2,31) (2,1,2,41,0,380,2,2,31) (2,1,2,41,1,296,1,1,29) (2,1,2,41,1,380,1,1,29)
(2,1,3,0,0,296,2,2,33) (2,1,3,0,0,380,2,2,33) (2,1,3,0,1,296,3,3,28) (2,1,3,0,1,380,3,3,28)
(2,1,3,10,0,296,1,1,30) (2,1,3,10,0,380,1,1,30) (2,1,3,10,1,296,2,2,29) (2,1,3,10,1,380,2,2,29)
(2,1,3,21,0,296,1,1,32) (2,1,3,21,0,380,1,1,32) (2,1,3,21,1,296,1,1,29) (2,1,3,21,1,380,1,1,29)
(2,1,3,30,0,296,1,1,32) (2,1,3,30,0,380,1,1,32) (2,1,3,30,1,296,2,2,29) (2,1,3,30,1,380,2,2,29)
(2,1,3,41,0,296,1,1,31) (2,1,3,41,0,380,1,1,31) (2,1,3,41,1,296,2,2,29) (2,1,3,41,1,380,2,2,29)
(2,1,4,0,0,296,2,2,34) (2,1,4,0,0,380,2,2,34) (2,1,4,0,1,296,2,2,29) (2,1,4,0,1,380,2,2,29)
(2,1,4,10,0,296,2,2,31) (2,1,4,10,0,380,2,2,31) (2,1,4,10,1,296,2,2,29) (2,1,4,10,1,380,2,2,29)
(2,1,4,21,0,296,1,1,31) (2,1,4,21,0,380,1,1,31) (2,1,4,21,1,296,1,1,29) (2,1,4,21,1,380,1,1,29)
(2,1,4,30,0,296,3,3,32) (2,1,4,30,0,380,3,3,32) (2,1,4,30,1,296,2,2,28) (2,1,4,30,1,380,2,2,28)
(2,1,4,41,0,296,0,0,33) (2,1,4,41,0,380,0,0,33) (2,1,4,41,1,296,3,3,29) (2,1,4,41,1,380,3,3,29)
(2,1,5,0,0,296,1,1,32) (2,1,5,0,0,380,1,1,32) (2,1,5,0,1,296,1,1,29) (2,1,5,0,1,380,1,1,29)
(2,1,5,10,0,296,1,1,31) (2,1,5,10,0,380,1,1,31) (2,1,5,10,1,296,2,2,29) (2,1,5,10,1,380,2,2,29)
(2,1,5,21,0,296,1,1,31) (2,1,5,21,0,380,1,1,31) (2,1,5,21,1,296,2,2,29) (2,1,5,21,1,380,2,2,29)
(2,1,5,30,0,296,1,1,31) (2,1,5,30,0,380,1,1,31) (2,1,5,30,1,296,2,2,28) (2,1,5,30,1,380,2,2,28)
(2,1,5,41,0,296,1,1,31) (2,1,5,41,0,380,1,1,31) (2,1,5,41,1,296,1,1,29) (2,1,5,41,1,380,1,1,29)
(2,1,6,0,0,296,1,1,32) (2,1,6,0,0,380,1,1,32) (2,1,6,0,1,296,1,1,29) (2,1,6,0,1,380,1,1,29)
(2,1,6,10,0,296,3,3,32) (2,1,6,10,0,380,3,3,32) (2,1,6,10,1,296,2,2,29) (2,1,6,10,1,380,2,2,29)
(2,1,6,21,0,296,2,2,33) (2,1,6,21,0,380,2,2,33) (2,1,6,21,1,296,1,1,29) (2,1,6,21,1,380,1,1,29)
(2,1,6,30,0,296,1,2,32) (2,1,6,30,0,380,1,2,32) (2,1,6,30,1,296,1,1,29) (2,1,6,30,1,380,1,1,29)
(2,1,6,41,0,296,5,5,32) (2,1,6,41,0,380,5,5,32) (2,1,6,41,1,296,1,1,29) (2,1,6,41,1,380,1,1,29)
(2,1,7,0,0,296,2,3,33) (2,1,7,0,0,380,2,3,33) (2,1,7,0,1,296,0,0,29) (2,1,7,0,1,380,0,0,29)
(2,1,7,10,0,296,3,3,33) (2,1,7,10,0,380,3,3,33) (2,1,7,10,1,296,0,0,29) (2,1,7,10,1,380,0,0,29)
(2,1,7,21,0,296,3,3,33) (2,1,7,21,0,380,3,3,33) (2,1,7,21,1,296,2,2,29) (2,1,7,21,1,380,2,2,29)
(2,1,7,30,0,296,1,1,34) (2,1,7,30,0,380,1,1,34) (2,1,7,30,1,296,0,0,29) (2,1,7,30,1,380,0,0,29)
(2,1,7,41,0,296,1,1,34) (2,1,7,41,0,380,1,1,34) (2,1,7,41,1,296,1,1,29) (2,1,7,41,1,380,1,1,29)
(2,1,8,0,0,296,2,2,32) (2,1,8,0,0,380,2,2,32) (2,1,8,0,1,296,2,2,29) (2,1,8,0,1,380,2,2,29)
(2,1,8,10,0,296,1,1,31) (2,1,8,10,0,380,1,1,31) (2,1,8,10,1,296,1,1,29) (2,1,8,10,1,380,1,1,29)
(2,1,8,21,0,296,0,0,31) (2,1,8,21,0,380,0,0,31) (2,1,8,21,1,296,2,2,29) (2,1,8,21,1,380,2,2,29)
(2,1,8,30,0,296,3,3,33) (2,1,8,30,0,380,3,3,33) (2,1,8,30,1,296,1,1,29) (2,1,8,30,1,380,1,1,29)
(2,1,8,41,0,296,2,2,31) (2,1,8,41,0,380,2,2,31) (2,1,8,41,1,296,1,1,29) (2,1,8,41,1,380,1,1,29)
(2,1,9,0,0,296,3,3,32) (2,1,9,0,0,380,3,3,32) (2,1,9,0,1,296,2,2,29) (2,1,9,0,1,380,2,2,29)
(2,1,9,10,0,296,1,1,32) (2,1,9,10,0,380,1,1,32) (2,1,9,10,1,296,1,1,29) (2,1,9,10,1,380,1,1,29)
(2,1,9,21,0,296,1,1,31) (2,1,9,21,0,380,1,1,31) (2,1,9,21,1,296,1,1,29) (2,1,9,21,1,380,1,1,29)
(2,1,9,30,0,296,0,0,31) (2,1,9,30,0,380,0,0,31) (2,1,9,30,1,296,1,1,29) (2,1,9,30,1,380,1,1,29)
(2,1,9,41,0,296,1,1,31) (2,1,9,41,0,380,1,1,31) (2,1,9,41,1,296,1,1,29) (2,1,9,41,1,380,1,1,29)
(1,0,0,0,0,296,2,2,58) (1,0,0,0,0,380,2,2,58) (1,0,0,0,1,296,1,3,55) (1,0,0,0,1,380,1,3,55)
(1,0,0,10,0,296,0,0,58) (1,0,0,10,0,380,0,0,58) (1,0,0,10,1,296,1,1,56) (1,0,0,10,1,380,1,1,56)
(1,0,0,21,0,296,1,1,57) (1,0,0,21,0,380,1,1,57) (1,0,0,21,1,296,3,6,55) (1,0,0,21,1,380,3,6,55)
(1,0,0,30,0,296,3,4,55) (1,0,0,30,0,380,3,4,55) (1,0,0,30,1,296,1,1,56) (1,0,0,30,1,380,1,1,56)
(1,0,0,41,0,296,0,0,58) (1,0,0,41,0,380,0,0,58) (1,0,0,41,1,296,3,7,53) (1,0,0,41,1,380,3,7,53)
(1,0,1,0,0,296,1,1,58) (1,0,1,0,0,380,1,1,58) (1,0,1,0,1,296,1,9,55) (1,0,1,0,1,380,1,9,55)
(1,0,1,10,0,296,2,2,56) (1,0,1,10,0,380,2,2,56) (1,0,1,10,1,296,2,2,54) (1,0,1,10,1,380,2,2,54)
(1,0,1,21,0,296,2,3,56) (1,0,1,21,0,380,2,3,56) (1,0,1,21,1,296,2,7,54) (1,0,1,21,1,380,2,7,54)
(1,0,1,30,0,296,0,0,58) (1,0,1,30,0,380,0,0,58) (1,0,1,30,1,296,1,2,55) (1,0,1,30,1,380,1,2,55)
(1,0,1,41,0,296,2,5,55) (1,0,1,41,0,380,2,5,55) (1,0,1,41,1,296,2,2,56) (1,0,1,41,1,380,2,2,56)
(1,0,2,0,0,296,1,2,57) (1,0,2,0,0,380,1,2,57) (1,0,2,0,1,296,2,7,55) (1,0,2,0,1,380,2,7,55)

(1,0,2,10,0,296,2,6,57) (1,0,2,10,0,380,2,6,57) (1,0,2,10,1,296,1,2,54) (1,0,2,10,1,380,1,2,54)
(1,0,2,21,0,296,1,1,58) (1,0,2,21,0,380,1,1,58) (1,0,2,21,1,296,2,4,55) (1,0,2,21,1,380,2,4,55)
(1,0,2,30,0,296,1,1,57) (1,0,2,30,0,380,1,1,57) (1,0,2,30,1,296,2,6,53) (1,0,2,30,1,380,2,6,53)
(1,0,2,41,0,296,1,2,56) (1,0,2,41,0,380,1,2,56) (1,0,2,41,1,296,2,3,55) (1,0,2,41,1,380,2,3,55)
(1,0,3,0,0,296,2,3,56) (1,0,3,0,0,380,2,3,56) (1,0,3,0,1,296,2,2,54) (1,0,3,0,1,380,2,2,54)
(1,0,3,10,0,296,1,2,57) (1,0,3,10,0,380,1,2,57) (1,0,3,10,1,296,1,2,56) (1,0,3,10,1,380,1,2,56)
(1,0,3,21,0,296,3,7,56) (1,0,3,21,0,380,3,7,56) (1,0,3,21,1,296,3,4,54) (1,0,3,21,1,380,3,4,54)
(1,0,3,30,0,296,1,1,57) (1,0,3,30,0,380,1,1,57) (1,0,3,30,1,296,1,2,57) (1,0,3,30,1,380,1,2,57)
(1,0,3,41,0,296,3,4,56) (1,0,3,41,0,380,3,4,56) (1,0,3,41,1,296,2,3,54) (1,0,3,41,1,380,2,3,54)
(1,0,4,0,0,296,2,4,56) (1,0,4,0,0,380,2,4,56) (1,0,4,0,1,296,2,2,54) (1,0,4,0,1,380,2,2,54)
(1,0,4,10,0,296,3,4,56) (1,0,4,10,0,380,3,4,56) (1,0,4,10,1,296,1,1,56) (1,0,4,10,1,380,1,1,56)
(1,0,4,21,0,296,1,1,57) (1,0,4,21,0,380,1,1,57) (1,0,4,21,1,296,1,1,55) (1,0,4,21,1,380,1,1,55)
(1,0,4,30,0,296,1,1,58) (1,0,4,30,0,380,1,1,58) (1,0,4,30,1,296,0,0,56) (1,0,4,30,1,380,0,0,56)
(1,0,4,41,0,296,1,2,57) (1,0,4,41,0,380,1,2,57) (1,0,4,41,1,296,3,3,58) (1,0,4,41,1,380,3,3,58)
(1,0,5,0,0,296,5,7,54) (1,0,5,0,0,380,5,7,54) (1,0,5,0,1,296,0,0,56) (1,0,5,0,1,380,0,0,56)
(1,0,5,10,0,296,6,6,56) (1,0,5,10,0,380,6,6,56) (1,0,5,10,1,296,1,1,57) (1,0,5,10,1,380,1,1,57)
(1,0,5,21,0,296,5,7,57) (1,0,5,21,0,380,5,7,57) (1,0,5,21,1,296,2,2,56) (1,0,5,21,1,380,2,2,56)
(1,0,5,30,0,296,0,0,58) (1,0,5,30,0,380,0,0,58) (1,0,5,30,1,296,2,2,56) (1,0,5,30,1,380,2,2,56)
(1,0,5,41,0,296,2,3,57) (1,0,5,41,0,380,2,3,57) (1,0,5,41,1,296,0,0,56) (1,0,5,41,1,380,0,0,56)
(1,0,6,0,0,296,3,6,55) (1,0,6,0,0,380,3,6,55) (1,0,6,0,1,296,2,4,56) (1,0,6,0,1,380,2,4,56)
(1,0,6,10,0,296,0,0,58) (1,0,6,10,0,380,0,0,58) (1,0,6,10,1,296,1,6,52) (1,0,6,10,1,380,1,6,52)
(1,0,6,21,0,296,2,2,56) (1,0,6,21,0,380,2,2,56) (1,0,6,21,1,296,2,4,57) (1,0,6,21,1,380,2,4,57)
(1,0,6,30,0,296,1,1,57) (1,0,6,30,0,380,1,1,57) (1,0,6,30,1,296,2,2,56) (1,0,6,30,1,380,2,2,56)
(1,0,6,41,0,296,1,6,53) (1,0,6,41,0,380,1,6,53) (1,0,6,41,1,296,2,6,58) (1,0,6,41,1,380,2,6,58)
(1,0,7,0,0,296,2,9,51) (1,0,7,0,0,380,2,9,51) (1,0,7,0,1,296,1,2,57) (1,0,7,0,1,380,1,2,57)
(1,0,7,10,0,296,3,6,54) (1,0,7,10,0,380,3,6,54) (1,0,7,10,1,296,2,6,55) (1,0,7,10,1,380,2,6,55)
(1,0,7,21,0,296,1,1,58) (1,0,7,21,0,380,1,1,58) (1,0,7,21,1,296,1,6,56) (1,0,7,21,1,380,1,6,56)
(1,0,7,30,0,296,1,1,57) (1,0,7,30,0,380,1,1,57) (1,0,7,30,1,296,0,0,58) (1,0,7,30,1,380,0,0,58)
(1,0,7,41,0,296,2,6,55) (1,0,7,41,0,380,2,6,55) (1,0,7,41,1,296,1,5,54) (1,0,7,41,1,380,1,5,54)
(1,0,8,0,0,296,1,15,49) (1,0,8,0,0,380,1,15,49) (1,0,8,0,1,296,2,4,57) (1,0,8,0,1,380,2,4,57)
(1,0,8,10,0,296,2,3,57) (1,0,8,10,0,380,2,3,57) (1,0,8,10,1,296,1,12,46) (1,0,8,10,1,380,1,12,46)
(1,0,8,21,0,296,2,9,53) (1,0,8,21,0,380,2,9,53) (1,0,8,21,1,296,1,5,55) (1,0,8,21,1,380,1,5,55)
(1,0,8,30,0,296,0,0,58) (1,0,8,30,0,380,0,0,58) (1,0,8,30,1,296,3,3,56) (1,0,8,30,1,380,3,3,56)
(1,0,8,41,0,296,3,7,55) (1,0,8,41,0,380,3,7,55) (1,0,8,41,1,296,2,8,55) (1,0,8,41,1,380,2,8,55)
(1,0,9,0,0,296,2,9,53) (1,0,9,0,0,380,2,9,53) (1,0,9,0,1,296,1,2,55) (1,0,9,0,1,380,1,2,55)
(1,0,9,10,0,296,2,9,53) (1,0,9,10,0,380,2,9,53) (1,0,9,10,1,296,3,4,53) (1,0,9,10,1,380,3,4,53)
(1,0,9,21,0,296,1,2,57) (1,0,9,21,0,380,1,2,57) (1,0,9,21,1,296,1,5,55) (1,0,9,21,1,380,1,5,55)
(1,0,9,30,0,296,2,9,53) (1,0,9,30,0,380,2,9,53) (1,0,9,30,1,296,4,7,52) (1,0,9,30,1,380,4,7,52)
(1,0,9,41,0,296,2,15,51) (1,0,9,41,0,380,2,15,51) (1,0,9,41,1,296,1,9,50) (1,0,9,41,1,380,1,9,50)
(1,1,0,0,0,296,2,2,29) (1,1,0,0,0,380,2,2,29) (1,1,0,0,1,296,1,1,29) (1,1,0,0,1,380,1,1,29)
(1,1,0,10,0,296,1,1,29) (1,1,0,10,0,380,1,1,29) (1,1,0,10,1,296,2,3,28) (1,1,0,10,1,380,2,3,28)
(1,1,0,21,0,296,2,2,29) (1,1,0,21,0,380,2,2,29) (1,1,0,21,1,296,2,2,29) (1,1,0,21,1,380,2,2,29)
(1,1,0,30,0,296,2,2,29) (1,1,0,30,0,380,2,2,29) (1,1,0,30,1,296,2,2,29) (1,1,0,30,1,380,2,2,29)
(1,1,0,41,0,296,1,1,29) (1,1,0,41,0,380,1,1,29) (1,1,0,41,1,296,1,1,29) (1,1,0,41,1,380,1,1,29)
(1,1,1,0,0,296,1,1,29) (1,1,1,0,0,380,1,1,29) (1,1,1,0,1,296,1,1,28) (1,1,1,0,1,380,1,1,28)
(1,1,1,10,0,296,2,2,29) (1,1,1,10,0,380,2,2,29) (1,1,1,10,1,296,2,5,29) (1,1,1,10,1,380,2,5,29)
(1,1,1,21,0,296,1,1,30) (1,1,1,21,0,380,1,1,30) (1,1,1,21,1,296,2,9,29) (1,1,1,21,1,380,2,9,29)
(1,1,1,30,0,296,3,3,29) (1,1,1,30,0,380,3,3,29) (1,1,1,30,1,296,2,3,29) (1,1,1,30,1,380,2,3,29)
(1,1,1,41,0,296,1,1,29) (1,1,1,41,0,380,1,1,29) (1,1,1,41,1,296,1,1,29) (1,1,1,41,1,380,1,1,29)
(1,1,2,0,0,296,2,2,29) (1,1,2,0,0,380,2,2,29) (1,1,2,0,1,296,2,2,29) (1,1,2,0,1,380,2,2,29)
(1,1,2,10,0,296,2,2,29) (1,1,2,10,0,380,2,2,29) (1,1,2,10,1,296,2,2,29) (1,1,2,10,1,380,2,2,29)
(1,1,2,21,0,296,2,2,29) (1,1,2,21,0,380,2,2,29) (1,1,2,21,1,296,2,2,29) (1,1,2,21,1,380,2,2,29)
(1,1,2,30,0,296,2,2,29) (1,1,2,30,0,380,2,2,29) (1,1,2,30,1,296,2,4,29) (1,1,2,30,1,380,2,4,29)
(1,1,2,41,0,296,3,3,29) (1,1,2,41,0,380,3,3,29) (1,1,2,41,1,296,1,1,28) (1,1,2,41,1,380,1,1,28)
(1,1,3,0,0,296,2,2,29) (1,1,3,0,0,380,2,2,29) (1,1,3,0,1,296,2,5,29) (1,1,3,0,1,380,2,5,29)
(1,1,3,10,0,296,1,1,29) (1,1,3,10,0,380,1,1,29) (1,1,3,10,1,296,2,3,28) (1,1,3,10,1,380,2,3,28)
(1,1,3,21,0,296,1,1,30) (1,1,3,21,0,380,1,1,30) (1,1,3,21,1,296,1,1,28) (1,1,3,21,1,380,1,1,28)
(1,1,3,30,0,296,1,1,29) (1,1,3,30,0,380,1,1,29) (1,1,3,30,1,296,2,4,29) (1,1,3,30,1,380,2,4,29)
(1,1,3,41,0,296,2,2,30) (1,1,3,41,0,380,2,2,30) (1,1,3,41,1,296,2,4,29) (1,1,3,41,1,380,2,4,29)
(1,1,4,0,0,296,1,1,30) (1,1,4,0,0,380,1,1,30) (1,1,4,0,1,296,3,3,28) (1,1,4,0,1,380,3,3,28)
(1,1,4,10,0,296,2,2,30) (1,1,4,10,0,380,2,2,30) (1,1,4,10,1,296,2,2,29) (1,1,4,10,1,380,2,2,29)
(1,1,4,21,0,296,1,1,29) (1,1,4,21,0,380,1,1,29) (1,1,4,21,1,296,1,1,28) (1,1,4,21,1,380,1,1,28)
(1,1,4,30,0,296,2,2,29) (1,1,4,30,0,380,2,2,29) (1,1,4,30,1,296,2,2,29) (1,1,4,30,1,380,2,2,29)
(1,1,4,41,0,296,3,3,29) (1,1,4,41,0,380,3,3,29) (1,1,4,41,1,296,1,1,29) (1,1,4,41,1,380,1,1,29)
(1,1,5,0,0,296,2,2,30) (1,1,5,0,0,380,2,2,30) (1,1,5,0,1,296,1,1,29) (1,1,5,0,1,380,1,1,29)
(1,1,5,10,0,296,2,2,29) (1,1,5,10,0,380,2,2,29) (1,1,5,10,1,296,1,1,29) (1,1,5,10,1,380,1,1,29)

(1,1,5,21,0,296,2,2,30) (1,1,5,21,0,380,2,2,30) (1,1,5,21,1,296,1,1,28) (1,1,5,21,1,380,1,1,28)
(1,1,5,30,0,296,1,1,30) (1,1,5,30,0,380,1,1,30) (1,1,5,30,1,296,1,1,29) (1,1,5,30,1,380,1,1,29)
(1,1,5,41,0,296,1,1,29) (1,1,5,41,0,380,1,1,29) (1,1,5,41,1,296,1,1,29) (1,1,5,41,1,380,1,1,29)
(1,1,6,0,0,296,2,2,30) (1,1,6,0,0,380,2,2,30) (1,1,6,0,1,296,3,3,29) (1,1,6,0,1,380,3,3,29)
(1,1,6,10,0,296,1,1,30) (1,1,6,10,0,380,1,1,30) (1,1,6,10,1,296,3,4,28) (1,1,6,10,1,380,3,4,28)
(1,1,6,21,0,296,2,2,30) (1,1,6,21,0,380,2,2,30) (1,1,6,21,1,296,2,6,29) (1,1,6,21,1,380,2,6,29)
(1,1,6,30,0,296,1,1,30) (1,1,6,30,0,380,1,1,30) (1,1,6,30,1,296,2,2,29) (1,1,6,30,1,380,2,2,29)
(1,1,6,41,0,296,1,1,30) (1,1,6,41,0,380,1,1,30) (1,1,6,41,1,296,2,2,29) (1,1,6,41,1,380,2,2,29)
(1,1,7,0,0,296,1,1,29) (1,1,7,0,0,380,1,1,29) (1,1,7,0,1,296,2,6,29) (1,1,7,0,1,380,2,6,29)
(1,1,7,10,0,296,1,1,29) (1,1,7,10,0,380,1,1,29) (1,1,7,10,1,296,2,2,29) (1,1,7,10,1,380,2,2,29)
(1,1,7,21,0,296,1,1,29) (1,1,7,21,0,380,1,1,29) (1,1,7,21,1,296,2,5,29) (1,1,7,21,1,380,2,5,29)
(1,1,7,30,0,296,1,1,29) (1,1,7,30,0,380,1,1,29) (1,1,7,30,1,296,2,4,29) (1,1,7,30,1,380,2,4,29)
(1,1,7,41,0,296,1,1,29) (1,1,7,41,0,380,1,1,29) (1,1,7,41,1,296,2,3,29) (1,1,7,41,1,380,2,3,29)
(1,1,8,0,0,296,2,2,29) (1,1,8,0,0,380,2,2,29) (1,1,8,0,1,296,1,1,29) (1,1,8,0,1,380,1,1,29)
(1,1,8,10,0,296,2,2,29) (1,1,8,10,0,380,2,2,29) (1,1,8,10,1,296,2,5,28) (1,1,8,10,1,380,2,5,28)
(1,1,8,21,0,296,2,2,30) (1,1,8,21,0,380,2,2,30) (1,1,8,21,1,296,2,4,29) (1,1,8,21,1,380,2,4,29)
(1,1,8,30,0,296,1,1,29) (1,1,8,30,0,380,1,1,29) (1,1,8,30,1,296,2,6,28) (1,1,8,30,1,380,2,6,28)
(1,1,8,41,0,296,2,2,30) (1,1,8,41,0,380,2,2,30) (1,1,8,41,1,296,2,4,29) (1,1,8,41,1,380,2,4,29)
(1,1,9,0,0,296,2,2,29) (1,1,9,0,0,380,2,2,29) (1,1,9,0,1,296,2,8,29) (1,1,9,0,1,380,2,8,29)
(1,1,9,10,0,296,2,2,29) (1,1,9,10,0,380,2,2,29) (1,1,9,10,1,296,2,5,29) (1,1,9,10,1,380,2,5,29)
(1,1,9,21,0,296,1,1,30) (1,1,9,21,0,380,1,1,30) (1,1,9,21,1,296,2,2,28) (1,1,9,21,1,380,2,2,28)
(1,1,9,30,0,296,2,2,30) (1,1,9,30,0,380,2,2,30) (1,1,9,30,1,296,2,3,29) (1,1,9,30,1,380,2,3,29)
(1,1,9,41,0,296,1,1,29) (1,1,9,41,0,380,1,1,29) (1,1,9,41,1,296,2,7,28) (1,1,9,41,1,380,2,7,28)
⟨1⟩ ⟨2⟩ 0.⟨3⟩ ⟨4⟩ ⟨5⟩ ⟨6⟩ 0 ⟨7⟩ ⟨8⟩ ⟨9⟩ ×800
⟨⟩ = (0,0,0,0,0,296,2,5,55) (0,0,0,0,0,380,2,5,55) (0,0,0,0,1,296,2,4,54) (0,0,0,0,1,380,2,4,54)
(0,0,0,10,0,296,0,0,58) (0,0,0,10,0,380,0,0,58) (0,0,0,10,1,296,1,1,56) (0,0,0,10,1,380,1,1,56)
(0,0,0,21,0,296,1,1,57) (0,0,0,21,0,380,1,1,57) (0,0,0,21,1,296,0,0,56) (0,0,0,21,1,380,0,0,56)
(0,0,0,30,0,296,3,4,56) (0,0,0,30,0,380,3,4,56) (0,0,0,30,1,296,1,1,58) (0,0,0,30,1,380,1,1,58)
(0,0,0,41,0,296,1,1,57) (0,0,0,41,0,380,1,1,57) (0,0,0,41,1,296,1,1,58) (0,0,0,41,1,380,1,1,58)
(0,0,1,0,0,296,1,5,56) (0,0,1,0,0,380,1,5,56) (0,0,1,0,1,296,1,1,57) (0,0,1,0,1,380,1,1,57)
(0,0,1,10,0,296,2,7,53) (0,0,1,10,0,380,2,7,53) (0,0,1,10,1,296,1,2,58) (0,0,1,10,1,380,1,2,58)
(0,0,1,21,0,296,0,0,58) (0,0,1,21,0,380,0,0,58) (0,0,1,21,1,296,1,1,57) (0,0,1,21,1,380,1,1,57)
(0,0,1,30,0,296,3,3,57) (0,0,1,30,0,380,3,3,57) (0,0,1,30,1,296,1,1,55) (0,0,1,30,1,380,1,1,55)
(0,0,1,41,0,296,1,1,58) (0,0,1,41,0,380,1,1,58) (0,0,1,41,1,296,1,1,56) (0,0,1,41,1,380,1,1,56)
(0,0,2,0,0,296,1,2,56) (0,0,2,0,0,380,1,2,56) (0,0,2,0,1,296,0,0,58) (0,0,2,0,1,380,0,0,58)
(0,0,2,10,0,296,2,3,57) (0,0,2,10,0,380,2,3,57) (0,0,2,10,1,296,2,6,54) (0,0,2,10,1,380,2,6,54)
(0,0,2,21,0,296,4,8,53) (0,0,2,21,0,380,4,8,53) (0,0,2,21,1,296,2,2,57) (0,0,2,21,1,380,2,2,57)
(0,0,2,30,0,296,1,4,55) (0,0,2,30,0,380,1,4,55) (0,0,2,30,1,296,1,2,56) (0,0,2,30,1,380,1,2,56)
(0,0,2,41,0,296,0,0,58) (0,0,2,41,0,380,0,0,58) (0,0,2,41,1,296,3,3,56) (0,0,2,41,1,380,3,3,56)
(0,0,3,0,0,296,1,4,57) (0,0,3,0,0,380,1,4,57) (0,0,3,0,1,296,4,4,57) (0,0,3,0,1,380,4,4,57)
(0,0,3,10,0,296,0,0,56) (0,0,3,10,0,380,0,0,56) (0,0,3,10,1,296,1,1,56) (0,0,3,10,1,380,1,1,56)
(0,0,3,21,0,296,3,4,56) (0,0,3,21,0,380,3,4,56) (0,0,3,21,1,296,1,1,58) (0,0,3,21,1,380,1,1,58)
(0,0,3,30,0,296,2,2,56) (0,0,3,30,0,380,2,2,56) (0,0,3,30,1,296,2,3,55) (0,0,3,30,1,380,2,3,55)
(0,0,3,41,0,296,2,3,55) (0,0,3,41,0,380,2,3,55) (0,0,3,41,1,296,4,4,56) (0,0,3,41,1,380,4,4,56)
(0,0,4,0,0,296,1,3,57) (0,0,4,0,0,380,1,3,57) (0,0,4,0,1,296,1,1,55) (0,0,4,0,1,380,1,1,55)
(0,0,4,10,0,296,1,1,57) (0,0,4,10,0,380,1,1,57) (0,0,4,10,1,296,2,2,55) (0,0,4,10,1,380,2,2,55)
(0,0,4,21,0,296,1,1,57) (0,0,4,21,0,380,1,1,57) (0,0,4,21,1,296,0,0,56) (0,0,4,21,1,380,0,0,56)
(0,0,4,30,0,296,2,2,58) (0,0,4,30,0,380,2,2,58) (0,0,4,30,1,296,0,0,58) (0,0,4,30,1,380,0,0,58)
(0,0,4,41,0,296,3,4,54) (0,0,4,41,0,380,3,4,54) (0,0,4,41,1,296,0,0,58) (0,0,4,41,1,380,0,0,58)
(0,0,5,0,0,296,1,1,57) (0,0,5,0,0,380,1,1,57) (0,0,5,0,1,296,2,2,57) (0,0,5,0,1,380,2,2,57)
(0,0,5,10,0,296,2,2,57) (0,0,5,10,0,380,2,2,57) (0,0,5,10,1,296,4,5,56) (0,0,5,10,1,380,4,5,56)
(0,0,5,21,0,296,2,2,57) (0,0,5,21,0,380,2,2,57) (0,0,5,21,1,296,4,4,55) (0,0,5,21,1,380,4,4,55)
(0,0,5,30,0,296,2,3,58) (0,0,5,30,0,380,2,3,58) (0,0,5,30,1,296,4,4,55) (0,0,5,30,1,380,4,4,55)
(0,0,5,41,0,296,2,2,56) (0,0,5,41,0,380,2,2,56) (0,0,5,41,1,296,4,4,57) (0,0,5,41,1,380,4,4,57)
(0,0,6,0,0,296,2,7,57) (0,0,6,0,0,380,2,7,57) (0,0,6,0,1,296,3,3,57) (0,0,6,0,1,380,3,3,57)
(0,0,6,10,0,296,1,1,57) (0,0,6,10,0,380,1,1,57) (0,0,6,10,1,296,2,2,55) (0,0,6,10,1,380,2,2,55)
(0,0,6,21,0,296,0,0,58) (0,0,6,21,0,380,0,0,58) (0,0,6,21,1,296,3,4,58) (0,0,6,21,1,380,3,4,58)
(0,0,6,30,0,296,1,2,57) (0,0,6,30,0,380,1,2,57) (0,0,6,30,1,296,4,4,56) (0,0,6,30,1,380,4,4,56)
(0,0,6,41,0,296,0,0,58) (0,0,6,41,0,380,0,0,58) (0,0,6,41,1,296,2,2,55) (0,0,6,41,1,380,2,2,55)
(0,0,7,0,0,296,1,10,53) (0,0,7,0,0,380,1,10,53) (0,0,7,0,1,296,0,0,58) (0,0,7,0,1,380,0,0,58)
(0,0,7,10,0,296,1,2,57) (0,0,7,10,0,380,1,2,57) (0,0,7,10,1,296,0,0,58) (0,0,7,10,1,380,0,0,58)
(0,0,7,21,0,296,2,7,54) (0,0,7,21,0,380,2,7,54) (0,0,7,21,1,296,0,0,58) (0,0,7,21,1,380,0,0,58)
(0,0,7,30,0,296,0,0,58) (0,0,7,30,0,380,0,0,58) (0,0,7,30,1,296,2,2,57) (0,0,7,30,1,380,2,2,57)
(0,0,7,41,0,296,1,5,55) (0,0,7,41,0,380,1,5,55) (0,0,7,41,1,296,0,0,58) (0,0,7,41,1,380,0,0,58)
(0,0,8,0,0,296,2,9,53) (0,0,8,0,0,380,2,9,53) (0,0,8,0,1,296,0,0,56) (0,0,8,0,1,380,0,0,56)
(0,0,8,10,0,296,1,3,57) (0,0,8,10,0,380,1,3,57) (0,0,8,10,1,296,2,4,55) (0,0,8,10,1,380,2,4,55)

(0,0,8,21,0,296,1,7,56) (0,0,8,21,0,380,1,7,56) (0,0,8,21,1,296,0,0,56) (0,0,8,21,1,380,0,0,56)
(0,0,8,30,0,296,0,0,58) (0,0,8,30,0,380,0,0,58) (0,0,8,30,1,296,0,0,58) (0,0,8,30,1,380,0,0,58)
(0,0,8,41,0,296,2,6,55) (0,0,8,41,0,380,2,6,55) (0,0,8,41,1,296,0,0,58) (0,0,8,41,1,380,0,0,58)
(0,0,9,0,0,296,1,5,55) (0,0,9,0,0,380,1,5,55) (0,0,9,0,1,296,1,1,55) (0,0,9,0,1,380,1,1,55)
(0,0,9,10,0,296,3,3,57) (0,0,9,10,0,380,3,3,57) (0,0,9,10,1,296,1,1,55) (0,0,9,10,1,380,1,1,55)
(0,0,9,21,0,296,1,1,58) (0,0,9,21,0,380,1,1,58) (0,0,9,21,1,296,1,1,55) (0,0,9,21,1,380,1,1,55)
(0,0,9,30,0,296,1,1,58) (0,0,9,30,0,380,1,1,58) (0,0,9,30,1,296,1,1,55) (0,0,9,30,1,380,1,1,55)
(0,0,9,41,0,296,1,10,51) (0,0,9,41,0,380,1,10,51) (0,0,9,41,1,296,2,2,57) (0,0,9,41,1,380,2,2,57)
(0,1,0,0,0,296,1,1,29) (0,1,0,0,0,380,1,1,29) (0,1,0,0,1,296,1,1,29) (0,1,0,0,1,380,1,1,29)
(0,1,0,10,0,296,1,1,29) (0,1,0,10,0,380,1,1,29) (0,1,0,10,1,296,1,1,28) (0,1,0,10,1,380,1,1,28)
(0,1,0,21,0,296,1,1,29) (0,1,0,21,0,380,1,1,29) (0,1,0,21,1,296,1,1,28) (0,1,0,21,1,380,1,1,28)
(0,1,0,30,0,296,3,3,29) (0,1,0,30,0,380,3,3,29) (0,1,0,30,1,296,1,1,28) (0,1,0,30,1,380,1,1,28)
(0,1,0,41,0,296,1,1,29) (0,1,0,41,0,380,1,1,29) (0,1,0,41,1,296,1,1,29) (0,1,0,41,1,380,1,1,29)
(0,1,1,0,0,296,1,1,29) (0,1,1,0,0,380,1,1,29) (0,1,1,0,1,296,1,1,29) (0,1,1,0,1,380,1,1,29)
(0,1,1,10,0,296,3,3,29) (0,1,1,10,0,380,3,3,29) (0,1,1,10,1,296,2,2,28) (0,1,1,10,1,380,2,2,28)
(0,1,1,21,0,296,2,5,29) (0,1,1,21,0,380,2,5,29) (0,1,1,21,1,296,2,2,29) (0,1,1,21,1,380,2,2,29)
(0,1,1,30,0,296,1,1,29) (0,1,1,30,0,380,1,1,29) (0,1,1,30,1,296,1,1,29) (0,1,1,30,1,380,1,1,29)
(0,1,1,41,0,296,2,5,29) (0,1,1,41,0,380,2,5,29) (0,1,1,41,1,296,1,1,28) (0,1,1,41,1,380,1,1,28)
(0,1,2,0,0,296,2,4,29) (0,1,2,0,0,380,2,4,29) (0,1,2,0,1,296,2,2,29) (0,1,2,0,1,380,2,2,29)
(0,1,2,10,0,296,1,1,29) (0,1,2,10,0,380,1,1,29) (0,1,2,10,1,296,2,2,29) (0,1,2,10,1,380,2,2,29)
(0,1,2,21,0,296,1,1,29) (0,1,2,21,0,380,1,1,29) (0,1,2,21,1,296,3,4,28) (0,1,2,21,1,380,3,4,28)
(0,1,2,30,0,296,1,1,29) (0,1,2,30,0,380,1,1,29) (0,1,2,30,1,296,2,2,29) (0,1,2,30,1,380,2,2,29)
(0,1,2,41,0,296,2,3,29) (0,1,2,41,0,380,2,3,29) (0,1,2,41,1,296,3,3,29) (0,1,2,41,1,380,3,3,29)
(0,1,3,0,0,296,2,3,29) (0,1,3,0,0,380,2,3,29) (0,1,3,0,1,296,2,2,29) (0,1,3,0,1,380,2,2,29)
(0,1,3,10,0,296,2,3,29) (0,1,3,10,0,380,2,3,29) (0,1,3,10,1,296,1,1,28) (0,1,3,10,1,380,1,1,28)
(0,1,3,21,0,296,2,4,29) (0,1,3,21,0,380,2,4,29) (0,1,3,21,1,296,3,3,29) (0,1,3,21,1,380,3,3,29)
(0,1,3,30,0,296,1,1,29) (0,1,3,30,0,380,1,1,29) (0,1,3,30,1,296,2,2,29) (0,1,3,30,1,380,2,2,29)
(0,1,3,41,0,296,1,1,29) (0,1,3,41,0,380,1,1,29) (0,1,3,41,1,296,1,1,29) (0,1,3,41,1,380,1,1,29)
(0,1,4,0,0,296,2,2,29) (0,1,4,0,0,380,2,2,29) (0,1,4,0,1,296,1,1,29) (0,1,4,0,1,380,1,1,29)
(0,1,4,10,0,296,2,2,29) (0,1,4,10,0,380,2,2,29) (0,1,4,10,1,296,1,1,28) (0,1,4,10,1,380,1,1,28)
(0,1,4,21,0,296,1,1,29) (0,1,4,21,0,380,1,1,29) (0,1,4,21,1,296,2,2,28) (0,1,4,21,1,380,2,2,28)
(0,1,4,30,0,296,1,1,29) (0,1,4,30,0,380,1,1,29) (0,1,4,30,1,296,1,1,28) (0,1,4,30,1,380,1,1,28)
(0,1,4,41,0,296,2,3,29) (0,1,4,41,0,380,2,3,29) (0,1,4,41,1,296,1,1,29) (0,1,4,41,1,380,1,1,29)
(0,1,5,0,0,296,2,2,29) (0,1,5,0,0,380,2,2,29) (0,1,5,0,1,296,1,1,28) (0,1,5,0,1,380,1,1,28)
(0,1,5,10,0,296,2,2,29) (0,1,5,10,0,380,2,2,29) (0,1,5,10,1,296,1,1,28) (0,1,5,10,1,380,1,1,28)
(0,1,5,21,0,296,2,2,29) (0,1,5,21,0,380,2,2,29) (0,1,5,21,1,296,1,1,28) (0,1,5,21,1,380,1,1,28)
(0,1,5,30,0,296,1,1,29) (0,1,5,30,0,380,1,1,29) (0,1,5,30,1,296,2,2,29) (0,1,5,30,1,380,2,2,29)
(0,1,5,41,0,296,2,2,29) (0,1,5,41,0,380,2,2,29) (0,1,5,41,1,296,1,1,28) (0,1,5,41,1,380,1,1,28)
(0,1,6,0,0,296,2,3,29) (0,1,6,0,0,380,2,3,29) (0,1,6,0,1,296,1,1,28) (0,1,6,0,1,380,1,1,28)
(0,1,6,10,0,296,1,1,29) (0,1,6,10,0,380,1,1,29) (0,1,6,10,1,296,1,1,29) (0,1,6,10,1,380,1,1,29)
(0,1,6,21,0,296,2,2,29) (0,1,6,21,0,380,2,2,29) (0,1,6,21,1,296,1,1,29) (0,1,6,21,1,380,1,1,29)
(0,1,6,30,0,296,2,2,29) (0,1,6,30,0,380,2,2,29) (0,1,6,30,1,296,2,2,28) (0,1,6,30,1,380,2,2,28)
(0,1,6,41,0,296,2,9,29) (0,1,6,41,0,380,2,9,29) (0,1,6,41,1,296,1,1,29) (0,1,6,41,1,380,1,1,29)
(0,1,7,0,0,296,2,4,29) (0,1,7,0,0,380,2,4,29) (0,1,7,0,1,296,1,1,29) (0,1,7,0,1,380,1,1,29)
(0,1,7,10,0,296,2,2,29) (0,1,7,10,0,380,2,2,29) (0,1,7,10,1,296,1,1,29) (0,1,7,10,1,380,1,1,29)
(0,1,7,21,0,296,2,2,29) (0,1,7,21,0,380,2,2,29) (0,1,7,21,1,296,1,1,29) (0,1,7,21,1,380,1,1,29)
(0,1,7,30,0,296,1,1,29) (0,1,7,30,0,380,1,1,29) (0,1,7,30,1,296,1,1,29) (0,1,7,30,1,380,1,1,29)
(0,1,7,41,0,296,2,4,29) (0,1,7,41,0,380,2,4,29) (0,1,7,41,1,296,1,1,29) (0,1,7,41,1,380,1,1,29)
(0,1,8,0,0,296,2,5,29) (0,1,8,0,0,380,2,5,29) (0,1,8,0,1,296,2,2,28) (0,1,8,0,1,380,2,2,28)
(0,1,8,10,0,296,2,3,29) (0,1,8,10,0,380,2,3,29) (0,1,8,10,1,296,3,3,28) (0,1,8,10,1,380,3,3,28)
(0,1,8,21,0,296,2,9,29) (0,1,8,21,0,380,2,9,29) (0,1,8,21,1,296,2,4,29) (0,1,8,21,1,380,2,4,29)
(0,1,8,30,0,296,1,1,29) (0,1,8,30,0,380,1,1,29) (0,1,8,30,1,296,1,1,29) (0,1,8,30,1,380,1,1,29)
(0,1,8,41,0,296,2,4,29) (0,1,8,41,0,380,2,4,29) (0,1,8,41,1,296,2,2,29) (0,1,8,41,1,380,2,2,29)
(0,1,9,0,0,296,2,5,29) (0,1,9,0,0,380,2,5,29) (0,1,9,0,1,296,2,2,29) (0,1,9,0,1,380,2,2,29)
(0,1,9,10,0,296,2,2,29) (0,1,9,10,0,380,2,2,29) (0,1,9,10,1,296,1,1,29) (0,1,9,10,1,380,1,1,29)
(0,1,9,21,0,296,2,2,29) (0,1,9,21,0,380,2,2,29) (0,1,9,21,1,296,2,2,29) (0,1,9,21,1,380,2,2,29)
(0,1,9,30,0,296,2,4,29) (0,1,9,30,0,380,2,4,29) (0,1,9,30,1,296,1,1,29) (0,1,9,30,1,380,1,1,29)
(0,1,9,41,0,296,2,7,29) (0,1,9,41,0,380,2,7,29) (0,1,9,41,1,296,2,2,29) (0,1,9,41,1,380,2,2,29)
(1,0,0,0,0,296,1,3,58) (1,0,0,0,0,380,1,3,58) (1,0,0,0,1,296,1,2,57) (1,0,0,0,1,380,1,2,57)
(1,0,0,10,0,296,1,1,57) (1,0,0,10,0,380,1,1,57) (1,0,0,10,1,296,2,6,57) (1,0,0,10,1,380,2,6,57)
(1,0,0,21,0,296,1,1,57) (1,0,0,21,0,380,1,1,57) (1,0,0,21,1,296,2,2,57) (1,0,0,21,1,380,2,2,57)
(1,0,0,30,0,296,1,1,58) (1,0,0,30,0,380,1,1,58) (1,0,0,30,1,296,2,2,56) (1,0,0,30,1,380,2,2,56)
(1,0,0,41,0,296,0,0,58) (1,0,0,41,0,380,0,0,58) (1,0,0,41,1,296,3,3,54) (1,0,0,41,1,380,3,3,54)
(1,0,1,0,0,296,1,1,57) (1,0,1,0,0,380,1,1,57) (1,0,1,0,1,296,1,1,58) (1,0,1,0,1,380,1,1,58)
(1,0,1,10,0,296,1,1,58) (1,0,1,10,0,380,1,1,58) (1,0,1,10,1,296,1,3,56) (1,0,1,10,1,380,1,3,56)
(1,0,1,21,0,296,1,1,57) (1,0,1,21,0,380,1,1,57) (1,0,1,21,1,296,0,0,58) (1,0,1,21,1,380,0,0,58)

(1,0,1,30,0,296,2,2,58) (1,0,1,30,0,380,2,2,58) (1,0,1,30,1,296,2,5,56) (1,0,1,30,1,380,2,5,56)
(1,0,1,41,0,296,3,3,57) (1,0,1,41,0,380,3,3,57) (1,0,1,41,1,296,1,2,57) (1,0,1,41,1,380,1,2,57)
(1,0,2,0,0,296,0,0,58) (1,0,2,0,0,380,0,0,58) (1,0,2,0,1,296,3,3,54) (1,0,2,0,1,380,3,3,54)
(1,0,2,10,0,296,2,2,57) (1,0,2,10,0,380,2,2,57) (1,0,2,10,1,296,0,0,58) (1,0,2,10,1,380,0,0,58)
(1,0,2,21,0,296,1,1,57) (1,0,2,21,0,380,1,1,57) (1,0,2,21,1,296,1,1,58) (1,0,2,21,1,380,1,1,58)
(1,0,2,30,0,296,3,4,57) (1,0,2,30,0,380,3,4,57) (1,0,2,30,1,296,3,3,57) (1,0,2,30,1,380,3,3,57)
(1,0,2,41,0,296,2,2,57) (1,0,2,41,0,380,2,2,57) (1,0,2,41,1,296,3,3,57) (1,0,2,41,1,380,3,3,57)
(1,0,3,0,0,296,4,4,55) (1,0,3,0,0,380,4,4,55) (1,0,3,0,1,296,0,0,58) (1,0,3,0,1,380,0,0,58)
(1,0,3,10,0,296,2,2,58) (1,0,3,10,0,380,2,2,58) (1,0,3,10,1,296,0,0,58) (1,0,3,10,1,380,0,0,58)
(1,0,3,21,0,296,1,1,58) (1,0,3,21,0,380,1,1,58) (1,0,3,21,1,296,0,0,56) (1,0,3,21,1,380,0,0,56)
(1,0,3,30,0,296,3,3,55) (1,0,3,30,0,380,3,3,55) (1,0,3,30,1,296,0,0,58) (1,0,3,30,1,380,0,0,58)
(1,0,3,41,0,296,1,1,58) (1,0,3,41,0,380,1,1,58) (1,0,3,41,1,296,1,1,55) (1,0,3,41,1,380,1,1,55)
(1,0,4,0,0,296,3,4,55) (1,0,4,0,0,380,3,4,55) (1,0,4,0,1,296,0,0,58) (1,0,4,0,1,380,0,0,58)
(1,0,4,10,0,296,0,0,58) (1,0,4,10,0,380,0,0,58) (1,0,4,10,1,296,1,1,58) (1,0,4,10,1,380,1,1,58)
(1,0,4,21,0,296,4,4,56) (1,0,4,21,0,380,4,4,56) (1,0,4,21,1,296,0,0,56) (1,0,4,21,1,380,0,0,56)
(1,0,4,30,0,296,4,4,57) (1,0,4,30,0,380,4,4,57) (1,0,4,30,1,296,2,2,57) (1,0,4,30,1,380,2,2,57)
(1,0,4,41,0,296,0,0,58) (1,0,4,41,0,380,0,0,58) (1,0,4,41,1,296,1,1,55) (1,0,4,41,1,380,1,1,55)
(1,0,5,0,0,296,1,1,58) (1,0,5,0,0,380,1,1,58) (1,0,5,0,1,296,3,3,56) (1,0,5,0,1,380,3,3,56)
(1,0,5,10,0,296,0,0,58) (1,0,5,10,0,380,0,0,58) (1,0,5,10,1,296,1,1,57) (1,0,5,10,1,380,1,1,57)
(1,0,5,21,0,296,0,0,58) (1,0,5,21,0,380,0,0,58) (1,0,5,21,1,296,1,1,57) (1,0,5,21,1,380,1,1,57)
(1,0,5,30,0,296,3,3,55) (1,0,5,30,0,380,3,3,55) (1,0,5,30,1,296,3,3,54) (1,0,5,30,1,380,3,3,54)
(1,0,5,41,0,296,1,1,57) (1,0,5,41,0,380,1,1,57) (1,0,5,41,1,296,1,1,57) (1,0,5,41,1,380,1,1,57)
(1,0,6,0,0,296,4,4,57) (1,0,6,0,0,380,4,4,57) (1,0,6,0,1,296,3,3,58) (1,0,6,0,1,380,3,3,58)
(1,0,6,10,0,296,0,0,58) (1,0,6,10,0,380,0,0,58) (1,0,6,10,1,296,0,0,56) (1,0,6,10,1,380,0,0,56)
(1,0,6,21,0,296,0,0,58) (1,0,6,21,0,380,0,0,58) (1,0,6,21,1,296,0,0,58) (1,0,6,21,1,380,0,0,58)
(1,0,6,30,0,296,1,1,58) (1,0,6,30,0,380,1,1,58) (1,0,6,30,1,296,0,0,56) (1,0,6,30,1,380,0,0,56)
(1,0,6,41,0,296,2,2,57) (1,0,6,41,0,380,2,2,57) (1,0,6,41,1,296,1,1,58) (1,0,6,41,1,380,1,1,58)
(1,0,7,0,0,296,1,1,58) (1,0,7,0,0,380,1,1,58) (1,0,7,0,1,296,0,0,58) (1,0,7,0,1,380,0,0,58)
(1,0,7,10,0,296,0,0,58) (1,0,7,10,0,380,0,0,58) (1,0,7,10,1,296,0,0,58) (1,0,7,10,1,380,0,0,58)
(1,0,7,21,0,296,2,2,58) (1,0,7,21,0,380,2,2,58) (1,0,7,21,1,296,0,0,58) (1,0,7,21,1,380,0,0,58)
(1,0,7,30,0,296,0,0,58) (1,0,7,30,0,380,0,0,58) (1,0,7,30,1,296,0,0,58) (1,0,7,30,1,380,0,0,58)
(1,0,7,41,0,296,1,1,58) (1,0,7,41,0,380,1,1,58) (1,0,7,41,1,296,0,0,58) (1,0,7,41,1,380,0,0,58)
(1,0,8,0,0,296,3,4,55) (1,0,8,0,0,380,3,4,55) (1,0,8,0,1,296,2,2,56) (1,0,8,0,1,380,2,2,56)
(1,0,8,10,0,296,3,3,55) (1,0,8,10,0,380,3,3,55) (1,0,8,10,1,296,1,1,58) (1,0,8,10,1,380,1,1,58)
(1,0,8,21,0,296,0,0,58) (1,0,8,21,0,380,0,0,58) (1,0,8,21,1,296,0,0,58) (1,0,8,21,1,380,0,0,58)
(1,0,8,30,0,296,1,2,56) (1,0,8,30,0,380,1,2,56) (1,0,8,30,1,296,1,1,58) (1,0,8,30,1,380,1,1,58)
(1,0,8,41,0,296,1,3,55) (1,0,8,41,0,380,1,3,55) (1,0,8,41,1,296,1,1,58) (1,0,8,41,1,380,1,1,58)
(1,0,9,0,0,296,1,1,58) (1,0,9,0,0,380,1,1,58) (1,0,9,0,1,296,3,3,54) (1,0,9,0,1,380,3,3,54)
(1,0,9,10,0,296,2,2,57) (1,0,9,10,0,380,2,2,57) (1,0,9,10,1,296,3,3,56) (1,0,9,10,1,380,3,3,56)
(1,0,9,21,0,296,3,3,57) (1,0,9,21,0,380,3,3,57) (1,0,9,21,1,296,1,1,57) (1,0,9,21,1,380,1,1,57)
(1,0,9,30,0,296,1,1,58) (1,0,9,30,0,380,1,1,58) (1,0,9,30,1,296,2,2,54) (1,0,9,30,1,380,2,2,54)
(1,0,9,41,0,296,1,1,58) (1,0,9,41,0,380,1,1,58) (1,0,9,41,1,296,1,1,57) (1,0,9,41,1,380,1,1,57)
(1,1,0,0,0,296,1,1,29) (1,1,0,0,0,380,1,1,29) (1,1,0,0,1,296,3,3,29) (1,1,0,0,1,380,3,3,29)
(1,1,0,10,0,296,1,1,29) (1,1,0,10,0,380,1,1,29) (1,1,0,10,1,296,1,1,28) (1,1,0,10,1,380,1,1,28)
(1,1,0,21,0,296,1,1,29) (1,1,0,21,0,380,1,1,29) (1,1,0,21,1,296,1,1,28) (1,1,0,21,1,380,1,1,28)
(1,1,0,30,0,296,1,1,29) (1,1,0,30,0,380,1,1,29) (1,1,0,30,1,296,2,2,29) (1,1,0,30,1,380,2,2,29)
(1,1,0,41,0,296,1,1,29) (1,1,0,41,0,380,1,1,29) (1,1,0,41,1,296,1,1,29) (1,1,0,41,1,380,1,1,29)
(1,1,1,0,0,296,2,2,29) (1,1,1,0,0,380,2,2,29) (1,1,1,0,1,296,1,1,29) (1,1,1,0,1,380,1,1,29)
(1,1,1,10,0,296,2,2,29) (1,1,1,10,0,380,2,2,29) (1,1,1,10,1,296,1,1,29) (1,1,1,10,1,380,1,1,29)
(1,1,1,21,0,296,2,2,29) (1,1,1,21,0,380,2,2,29) (1,1,1,21,1,296,1,1,29) (1,1,1,21,1,380,1,1,29)
(1,1,1,30,0,296,2,2,29) (1,1,1,30,0,380,2,2,29) (1,1,1,30,1,296,1,1,28) (1,1,1,30,1,380,1,1,28)
(1,1,1,41,0,296,1,1,29) (1,1,1,41,0,380,1,1,29) (1,1,1,41,1,296,2,2,28) (1,1,1,41,1,380,2,2,28)
(1,1,2,0,0,296,2,2,29) (1,1,2,0,0,380,2,2,29) (1,1,2,0,1,296,1,1,29) (1,1,2,0,1,380,1,1,29)
(1,1,2,10,0,296,2,2,29) (1,1,2,10,0,380,2,2,29) (1,1,2,10,1,296,1,1,29) (1,1,2,10,1,380,1,1,29)
(1,1,2,21,0,296,1,1,29) (1,1,2,21,0,380,1,1,29) (1,1,2,21,1,296,2,2,29) (1,1,2,21,1,380,2,2,29)
(1,1,2,30,0,296,2,2,29) (1,1,2,30,0,380,2,2,29) (1,1,2,30,1,296,1,1,28) (1,1,2,30,1,380,1,1,28)
(1,1,2,41,0,296,1,1,29) (1,1,2,41,0,380,1,1,29) (1,1,2,41,1,296,3,3,29) (1,1,2,41,1,380,3,3,29)
(1,1,3,0,0,296,2,2,29) (1,1,3,0,0,380,2,2,29) (1,1,3,0,1,296,2,2,29) (1,1,3,0,1,380,2,2,29)
(1,1,3,10,0,296,2,2,28) (1,1,3,10,0,380,2,2,28) (1,1,3,10,1,296,1,1,28) (1,1,3,10,1,380,1,1,28)
(1,1,3,21,0,296,2,2,29) (1,1,3,21,0,380,2,2,29) (1,1,3,21,1,296,1,1,29) (1,1,3,21,1,380,1,1,29)
(1,1,3,30,0,296,1,1,29) (1,1,3,30,0,380,1,1,29) (1,1,3,30,1,296,1,1,28) (1,1,3,30,1,380,1,1,28)
(1,1,3,41,0,296,1,1,29) (1,1,3,41,0,380,1,1,29) (1,1,3,41,1,296,2,2,28) (1,1,3,41,1,380,2,2,28)
(1,1,4,0,0,296,2,2,29) (1,1,4,0,0,380,2,2,29) (1,1,4,0,1,296,3,3,28) (1,1,4,0,1,380,3,3,28)
(1,1,4,10,0,296,1,1,29) (1,1,4,10,0,380,1,1,29) (1,1,4,10,1,296,1,1,28) (1,1,4,10,1,380,1,1,28)
(1,1,4,21,0,296,1,1,29) (1,1,4,21,0,380,1,1,29) (1,1,4,21,1,296,1,1,28) (1,1,4,21,1,380,1,1,28)
(1,1,4,30,0,296,1,1,29) (1,1,4,30,0,380,1,1,29) (1,1,4,30,1,296,1,1,29) (1,1,4,30,1,380,1,1,29)

```
(1,1,4,41,0,296,1,1,29) (1,1,4,41,0,380,1,1,29) (1,1,4,41,1,296,1,1,29) (1,1,4,41,1,380,1,1,29)
(1,1,5,0,0,296,2,2,29) (1,1,5,0,0,380,2,2,29) (1,1,5,0,1,296,2,2,28) (1,1,5,0,1,380,2,2,28)
(1,1,5,10,0,296,1,1,29) (1,1,5,10,0,380,1,1,29) (1,1,5,10,1,296,2,2,28) (1,1,5,10,1,380,2,2,28)
(1,1,5,21,0,296,2,2,29) (1,1,5,21,0,380,2,2,29) (1,1,5,21,1,296,2,2,28) (1,1,5,21,1,380,2,2,28)
(1,1,5,30,0,296,1,1,29) (1,1,5,30,0,380,1,1,29) (1,1,5,30,1,296,2,2,28) (1,1,5,30,1,380,2,2,28)
(1,1,5,41,0,296,2,2,29) (1,1,5,41,0,380,2,2,29) (1,1,5,41,1,296,2,2,29) (1,1,5,41,1,380,2,2,29)
(1,1,6,0,0,296,2,2,29) (1,1,6,0,0,380,2,2,29) (1,1,6,0,1,296,3,3,29) (1,1,6,0,1,380,3,3,29)
(1,1,6,10,0,296,1,1,29) (1,1,6,10,0,380,1,1,29) (1,1,6,10,1,296,2,2,29) (1,1,6,10,1,380,2,2,29)
(1,1,6,21,0,296,1,1,29) (1,1,6,21,0,380,1,1,29) (1,1,6,21,1,296,3,3,29) (1,1,6,21,1,380,3,3,29)
(1,1,6,30,0,296,1,1,29) (1,1,6,30,0,380,1,1,29) (1,1,6,30,1,296,3,3,29) (1,1,6,30,1,380,3,3,29)
(1,1,6,41,0,296,2,2,29) (1,1,6,41,0,380,2,2,29) (1,1,6,41,1,296,1,1,28) (1,1,6,41,1,380,1,1,28)
(1,1,7,0,0,296,1,1,29) (1,1,7,0,0,380,1,1,29) (1,1,7,0,1,296,1,1,29) (1,1,7,0,1,380,1,1,29)
(1,1,7,10,0,296,1,1,29) (1,1,7,10,0,380,1,1,29) (1,1,7,10,1,296,1,1,29) (1,1,7,10,1,380,1,1,29)
(1,1,7,21,0,296,1,1,29) (1,1,7,21,0,380,1,1,29) (1,1,7,21,1,296,1,1,29) (1,1,7,21,1,380,1,1,29)
(1,1,7,30,0,296,1,1,29) (1,1,7,30,0,380,1,1,29) (1,1,7,30,1,296,1,1,29) (1,1,7,30,1,380,1,1,29)
(1,1,7,41,0,296,1,1,29) (1,1,7,41,0,380,1,1,29) (1,1,7,41,1,296,1,1,29) (1,1,7,41,1,380,1,1,29)
(1,1,8,0,0,296,1,1,29) (1,1,8,0,0,380,1,1,29) (1,1,8,0,1,296,1,1,28) (1,1,8,0,1,380,1,1,28)
(1,1,8,10,0,296,1,1,29) (1,1,8,10,0,380,1,1,29) (1,1,8,10,1,296,2,8,29) (1,1,8,10,1,380,2,8,29)
(1,1,8,21,0,296,1,1,29) (1,1,8,21,0,380,1,1,29) (1,1,8,21,1,296,1,1,28) (1,1,8,21,1,380,1,1,28)
(1,1,8,30,0,296,1,1,29) (1,1,8,30,0,380,1,1,29) (1,1,8,30,1,296,1,1,29) (1,1,8,30,1,380,1,1,29)
(1,1,8,41,0,296,2,3,29) (1,1,8,41,0,380,2,3,29) (1,1,8,41,1,296,1,1,29) (1,1,8,41,1,380,1,1,29)
(1,1,9,0,0,296,1,1,29) (1,1,9,0,0,380,1,1,29) (1,1,9,0,1,296,2,2,28) (1,1,9,0,1,380,2,2,28)
(1,1,9,10,0,296,1,1,29) (1,1,9,10,0,380,1,1,29) (1,1,9,10,1,296,2,2,28) (1,1,9,10,1,380,2,2,28)
(1,1,9,21,0,296,3,3,29) (1,1,9,21,0,380,3,3,29) (1,1,9,21,1,296,2,2,28) (1,1,9,21,1,380,2,2,28)
(1,1,9,30,0,296,1,1,29) (1,1,9,30,0,380,1,1,29) (1,1,9,30,1,296,2,2,28) (1,1,9,30,1,380,2,2,28)
(1,1,9,41,0,296,1,1,29) (1,1,9,41,0,380,1,1,29) (1,1,9,41,1,296,2,2,29) (1,1,9,41,1,380,2,2,29)
```

2.6 /opt/GSM/c54x_exe/proof_toa_23.md

40376 octets, 717 lignes

2.6.1 OsmocomBB's "magic 23", proven from the ROM, the layer1 code and hardware logs

The FIXME: `prim_fbsb.c:333` and `:469`, `/* FIXME: where did this magic 23 come from? */`, followed by `last_fb->toa -= 23;`.

Claim. The DSP returns a position inside a 191-sample window hard-wired by TI; 23 is the only margin that, in the `tpu_window.c` formula, produces that window; `toa - 23` turns the position into an offset from an SB sitting at its nominal place, and the `+ 75` qbits of `synchronize_tdma` then move the burst to the centre of the NB window, where the silicon does return the 16-qbit residual that `toa.c` expects.

2.6.1.1 Rules of evidence and sources

The evidence rests on three sources:

source	what it is	where
ROM	TI Calypso DSP mask-ROM, version 3606, disassembled (binutils `tic54x`)	public dump: `ftp://ftp.freecalypso.org/pub/GSM/Calypso/dsp` (Pirelli DP-L10, Calypso PD751992AZHH, OsmocomBB dump tool). The PROM0 (0x7000-0xdfff, 28,672 words) of that dump is **identical word for word** to the PROM0 used here (0 differences out of 28,672).
layer1	OsmocomBB firmware sources, `src/target/firmware/layer1/{prim_fbsb.c,tpu_window.c,sync.c,toa.c}` and their git history	`osmocom-bb` repository

source	what it is	where
hardware logs	firmware output from real Calypso phones, posted to the baseband-devel list	[1] Aegean Chou, 2011-09-26, *Re: About sniff multi bursts in a frame, CCCH_CONF* (DSP API 0x3606, firmware without TX): https://lists.osmocom.org/pipermail/baseband-devel/2011-September/002526.html — [2] nish079858, 2024-10-02, *Osmocom bb mobile station cannot register to YateBTS basestation* (Motorola C115, `layer1.highram.bin` e88): https://www.mail-archive.com/baseband-devel@lists.osmocom.org/msg01046.html

The emulation bench (`c54x_exe`, `dsp_tester`) is **not** a source of evidence: it served to find out where to look (§7), and every time it is cited this is stated.

Levels: **PROVEN** (static reading of the ROM or of the code, or a hardware log landing on a numerical prediction), **VERY LIKELY**, **HYPOTHESIS**.

2.6.1.1.1 The result on one page

1. The DSP does not return a timing error. Its TOA is the **index of the lag it retained** in the correlation with the training sequence, the leading edge of its channel estimate (ROM, §2). A burst whose first sample is at position k of the window yields TOA = k + 3 (correlation starting at sample 39, training sequence at bit 42), or less if an earlier lag of the 7-lag window reaches the threshold (§2.4).
2. The receive windows are **immediate constants in the ROM**: 191 samples for the SB, 151 for normal bursts (§1). `tpu_window.c` computes its own as `burst + 2·margin − 3` with `L1_SB_MARGIN_Q = 23×4` and `L1_NB_MARGIN_Q = 3×4`: $\mathbf{148 + 2 \cdot 23 - 3 = 191}$ and $\mathbf{148 + 2 \cdot 3 - 3 = 151}$. Conversely (191 − 148 + 3)/2 = 23: 23 is the only margin compatible with TI's window (§4.1).
3. SB path: `TOA − 23`, then `+ 75` qbits, a net shift of `TOA_sb − 4.25` bits: the burst is placed at a TOA of **4.25**, the centre of the NB window (3 samples of play, centre 1.5, plus the ROM's offset of 3), which `toa.c` then holds at 16 qbits (§4.3). FB path: 23 subtracted **twice** since `cb71b972` (2010-05-20), net shift `TOA_fb − 27.25`, which makes the SB arrive at a TOA of 27.25 (§4.4).
4. The hardware logs land on these numbers with no free parameter (§5): printed qbits = ((TOA − 46) mod 1250)×4 (12 lines out of 12: 6 after an FB1, 6 after an SB); every first SB after an FB synchronisation, in both logs (n = 6): 29, 29, 27, 27, 24, 26 for a prediction of 27.25 (mean deviation −0.25, within the FB's granularity of 4); after SB synchronisation, every NB TOA deviation that `toa.c` prints is within 4 qbits of 16; FB TOA mode 1 3 (mod 4) (10/10) and mode 0 0 (mod 48) (7/7), as the ROM's immediates say.
5. **23 is the margin of the SB window**, the firmware's and TI's. The FIXME persists because 23 (`prim_fbsb.c`), 75 (`sync.c`) and 16 qbits (`toa.c`) only make sense together, and because the 191-sample window is written nowhere in the firmware.

2.6.1.2 1. ROM: the receive windows are immediate constants — PROVEN

Common DMA programming routine (PROM0, binutils disassembly):

```
a5cd  ld     #134,dp
a5e3  ldm    ar2,a               ; A = API address of the page (0x0cce)
a5e4  sub    #2048,a
a5e6  stl    a,1,DP+0x56         ; (A - 0x800) << 1
a5e8  portw  DP+0x56,pa64548     ; port 0xfc24: address (DMA2_AAD)
a5ea  ldm    ar3,a               ; A = AR3 = length in samples
a5ed  stl    a,2,DP+0x56         ; A << 2 = length in bytes (4 per I/Q sample)
a5ef  portw  DP+0x56,pa64550     ; port 0xfc26: length (DMA2_ALGTH)
```

```
a5f1  st     #1197,DP+0x56
a5f3  xc     2,beq
a5f4  st     #1193,DP+0x56
a5f6  portw  DP+0x56,pa64552      ; port 0xfc28: control (DMA2_CTRL)
```

The three task routines load AR3 with an immediate and then call this code through the ROM's dispatcher (`ld #41,a ; call 0xa9ea`: code 41 designates routine 0xa5cd; in emulation, AR3 does hold 191 on entry to 0xa5cd during an SB task):

task	PC	instruction	window
SB	0xb21d	`stm #191,ar3`	191 samples (764 bytes)
NB	0xb2b1	`stm #151,ar3`	151 samples (604 bytes)
FB	0xb32d	`stm #48,ar3`	48 samples per page (192 bytes), continuous stream

No word written by the ARM is involved: the window lengths are fixed by TI in the ROM. (The identification of ports 0xfc24/26/28 with the AAD/ALGTH/CTRL registers of the RHEA controller comes from the Calypso register map; the next point depends only on the immediates 191 and 151.)

2.6.1.3 2. ROM: the SB TOA is the index of the retained lag (leading edge of the channel estimate, §2.4) — PROVEN

Complete chain, static reading of PROM0.

2.1 The DMA page is copied without offset. `0x7c3d-0x7c47`: `stm #3278,ar2` (API page 0x0cce), `stm #10752,ar3` (0x2a00), `stm #10944,ar4` (0x2ac0), `stm #3660,ar5` (0x0e4c: I/Q DC component computed at `0x81a5-0x81c7` over the 191 samples), `calld 0x81c8` with `stm #47,brc` in the delay slot (48 iterations of 4 samples = 192 191). The loop `0x81cc-0x81db` reads the page in order (`*ar2+`), subtracts the DC component and stores sample n at **index n** of both buffers, applying the MSK derotation $(-j)^{\wedge}(n+1)$ through the signs and the two destinations: buffer 0x2a00 ← Q0, −I1, −Q2, I3, …; buffer 0x2ac0 ← −I0, −Q1, I2, Q3, …

2.2 The correlator looks for the extended training sequence. `0x7c49-0x7c58`: 64 words read from program memory starting at `0x7a1c` (`ld #31260,a ; rpt #25 ; reada *ar2+` ×2, `rpt #11 ; reada`) into 0x2cea. These 64 words are ±512 and their sign is, bit for bit, the **SB extended training sequence** (GSM 05.02 §5.2.5, 64 bits), with 1 → −512 and 0 → +512:

```
ETS            1011100101100010000001000000111100101101010001010111011000011011
sign(coeff)    0100011010011101111110111111100001101001010111010100010011110010   (= bitwise complement)
```

2.3 The correlation starts at sample 39. `0x84a1-0x84c6`: `stm #10791,ar3` (0x2a27 = 0x2a00 + **39**), `stm #10983,ar4` (0x2ae7 = 0x2ac0 + 39), `stm #49,brc` (50 lags), `stm #-63,ar0`; for each lag i, 64 MACs of coefficients × buffer[39 + i + j] on each component (`sth a,*ar1+` to 0x2c56 + i, `sth a,*ar5+` to 0x2c88 + i), then `add *ar3+,*ar4+,a` which advances by one sample. `0x84ca-0x84d8`: energy $I^2 + Q^2$ of lag i at 0x2be4 + 2i (`dst a,*ar4+`).

2.4 Choice of lag: leading edge of the channel estimate. Two blocks, static reading.

7-lag window. `0x84dd-0x84ef`: A = 0, B = 0, AR0 = AR1 = 0; for i = 0..42: `dadd *ar3+,a` (A += E[i+7]), `dsub *ar2+,a` (A −= E[i]), `max b`, `mar *ar0+`, `xc 1,nc ; mvmm ar0,ar1`. After iteration i, A = W[i+1] − W[0] where W[s] = E[s] + … + E[s+6]; AR1 receives i+1 whenever A exceeds the current maximum. Hence **s = argmax W[s]** (s = 0 by default if W[0] dominates), stored at 0x2f06.

Threshold. `0x7c5f-0x7c80`: energy of the 64 samples starting at 0x2a2a + s (`squra` ×64, `sfta a,-6`: average per sample), then × 1638/65536 (`stm #1638,t ; mpyu`) → DP+0x00 (0x2f00): **2.5 % of the mean received energy per sample**.

First significant lag. `0x7c8f-0x7c95`: the 7 + 7 channel coefficients (0x2cce..) are zeroed. `0x7ca6`: `rsbx tc`. `0x7ca7-0x7cae`: AR6 = AR7 = 0x2be4 + 2s. `0x7ca9-0x7cbb`, 7 iterations:

```
7caf  dld    DP+0x00,a          ; A = threshold
7cb0  dsub   *ar6,a             ; A = threshold − E[lag]
7cb1  bc     0x7cb7,agt         ; E < threshold: lag ignored
7cb3  ssbx   st0,tc             ; E ≥ threshold: TC = 1 (and what follows)
7cb4  dadd   *ar6,b
7cb5  mvdd   *ar2,*ar4          ; channel coefficient I ← correlation I of this lag
7cb6  mvdd   *ar3,*ar5          ;                      Q
7cb7  mvdd   *ar4+,*ar2+       ; copy of the coefficient (zero if ignored) into the correlation buffer
7cb8  mvdd   *ar5+,*ar3+
7cb9  dadd   *ar6+,a            ; A = threshold; next lag
7cba  xc     1,ntc              ; as long as no lag has reached the threshold:
7cbb  mvmm   ar6,ar7            ;     AR7 = pointer to the next lag
```

TC is never cleared inside the loop: AR7 stops on the **first lag of the window whose energy reaches the threshold**, and the lags before it are erased from the channel estimate. The TOA is therefore the leading edge of the estimated channel: equal to the peak k + 3 for a clean pulse, **k + 3 or less** otherwise (k + 2 as soon as the energy at k + 2 exceeds 2.5 % of the mean energy, widened pulse; k + 1 with a strong precursor, the rule taking the first lag threshold from s onwards). Then:

```
7cbc  ldm    ar7,a
7cbd  sub    #11236,a           ; 11236 = 0x2be4
7cbf  stl    a,-1,DP+0x1f       ; TOA = (AR7 − 0x2be4) / 2  →  0x2f1f
7e8a  mvdk   DP+0x1f,0x3fa4
b1e7  ld     *(0x3fa4),a
b1e9  stl    a,*ar0(0x8)        ; word 8 of the DB page = a_serv_demod[0] = D_TOA (dsp_api.h, DSP 36)
```

No constant is added: the TOA is the index of the retained lag. (Checked in emulation on three energy profiles read from table 0x2be4: the rule yields 23, 24 and 23 where the ROM published 23, 24 and 23, §7.)

2.5 Geometry. In the SB, the extended training sequence occupies bits 42 to 105 (3 tail bits, 39 data, 64 training, 39, 3). If the first sample of the burst is at index k of the DMA window, hence of the buffer (§2.1), the sequence starts at index k + 42, and the lag i that aligns it with the coefficients satisfies 39 + i = k + 42:

```
TOA_SB = k + 3   (peak), or less if an earlier lag reaches the threshold (leading edge, §2.4)
```

2.6 A false 23 not to be mistaken for the real one. If the energy of the retained channel coefficients is zero (`0x7ccd`, `bcd 0x7ced,aneq` not taken), the fallback `0x7ce5 mvmd ar0,0x2f1f` publishes the coarse index AR0 instead. In that case there is no channel estimate and the SB does not decode (bad CRC). Every TOA cited here comes with a correct CRC decode, hence from the main path (audit in §7).

2.6.1.4 3. ROM: the FB TOA — PROVEN for the form, partial for the detail

`0x7931-0x795a` (mode 1):

```
7931  rptb   0x793d              ; scan of the 32-bit energy buffer (*ar2, dld), tracking the maximum
793e  ld     *ar4+,b            ; B = c, index of the maximum
793f  sfta   b,2                ; B = 4·c
7940  ld     *(0x3fb4),16,a     ; A = N (page counter, 0x3fb4)
7942  sub    #3,16,a
7944  add    *ar4,16,a          ; + p (word following the index)
7945  sub    #2,16,a
7947  mpya   a                  ; × T = 48 (one page)
7948  add    a,b
7949  sub    #5,b               ; TOA = 48·(N + p − 5) + 4·c − 5
794b  sub    *(0xc3d),b,a ; rc aleq       ; lower and upper bounds …
795a  stl    b,*(0x8fa)         ; NDB a_sync_demod[D_TOA]
```

Immediates: 48 (DMA page), −5, step of 4 on c. In mode 1, **TOA 3 (mod 4)**, granularity 4 samples; 23 is reachable ($48 \cdot m + 23$: the logs give $10007 = 48 \cdot 208 + 23$). In mode 0 the logs give exact multiples of 48 (§5.2): the position is only known to the page, and 23 is not reachable there. The origin of N, p and the −5 has not been worked out.

2.6.1.5 4. layer1: what the firmware does with this TOA — PROVEN (reading of the code and of git)

2.6.1.5.1 4.1 The firmware's margins are the ROM's

`tpu_window.c:38-44`:

```
#define L1_NB_MARGIN_Q      (3 * 4)
#define L1_SB_MARGIN_Q      (23 * 4)
#define L1_TAIL_DURATION_Q  (3 * 4)
#define L1_NB_DURATION_Q    (L1_BURST_LENGTH_Q + 2 * L1_NB_MARGIN_Q - L1_TAIL_DURATION_Q)
#define L1_SB_DURATION_Q    (L1_BURST_LENGTH_Q + 2 * L1_SB_MARGIN_Q - L1_TAIL_DURATION_Q)
```

Same formula, with a 148-bit burst, as the immediates of §1:

```
SB: 148 + 2·23 − 3 = 191        NB: 148 + 2·3 − 3 = 151        and  (191 − 148 + 3) / 2 = 23
```

Two windows, two margins, one formula: the 151 constrains the formula as much as the 191 does; this is not a decomposition chosen after the fact. With `L1_BURST_LENGTH_Q = 625` qbits (one slot of 156.25 bits), the firmware opens its TPU windows 8.25 bits longer than the DSP's DMA, on both sides: the guard period.

2.6.1.5.2 4.2 The windows open at the same instant

On the ARM side, `l1s_rx_win_ctrl` (`tpu_window.c:100-110`): `start = DSP_SETUP_TIME` for every window type, only the duration changes. On the DSP side (§1), the SB and NB tasks arm the DMA through the same routine 0xa5cd, with the same page address (`stm #3278,ar2` at 0xb21b and 0xb2af), the same control word 0x04ad (ONE_SHOT; the FB gets 0x04a9, without ONE_SHOT), only the length differs: the ROM programs no task-specific offset. The start is therefore the same for both windows (the hardware trigger of the DMA has not been traced; §5.4 confirms it on silicon), and a burst has the same position k in the SB window (191) and in the NB window (151); only the room left around it changes.

2.6.1.5.3 4.3 SB path: TOA − 23, then + 75 qbits — target 4.25

`prim_fbsb.c:207-231` (`l1s_sbdet_resp`):

```
last_fb->toa -= 23;
qbits = last_fb->toa * 4;
…
cinfo->time_alignment = qbits;
synchronize_tdma(&l1s.serving_cell);
```

`sync.c:113-121` (`synchronize_tdma`):

```
uint32_t tpu_shift = cinfo->time_alignment;
/* NB detection only works if the TOA of the SB
 * is within 0...8. We have to add 75 to get an SB TOA of 4. */
tpu_shift += 75;
```

Time-base shift: (TOA_sb − 23)×4 + 75 qbits = **TOA_sb − 4.25 bits**. After this shift the burst sits at a TOA of 4.25 in any window opened at the same instant (§4.2), hence in the NB window. And `toa.c` (2011, `937023be`) then holds the average TOA of normal bursts at **16 qbits = 4 bits**, accepting only 0 TOA 31 qbits. In the 151-sample NB window a 148-sample burst is centred at k = 1.5, i.e. TOA = 4.5 (peak, §2.5) or 3.5 (leading edge): 4.25 is the centre of the NB window to within 0.25, which is also what the comment says ("within 0…8", "an SB TOA of 4"). And 75 qbits = 23 − 4.25 bits: the +75 is the difference between the firmware's nominal SB position and the centre of the NB window.

2.6.1.5.4 4.4 FB path: 23 is subtracted twice — net shift TOA_fb − 27.25

`prim_fbsb.c:465-493` (`l1s_fbdet_resp`, mode 1):

```
last_fb->toa -= 23;                     /* 1st subtraction, for ntdma and delay */
…
fbinfo2cellinfo(&l1s.serving_cell, last_fb);   /* 2nd subtraction, prim_fbsb.c:333 */
synchronize_tdma(&l1s.serving_cell);           /* + 75 */
tdma_schedule_set(delay, sb_sched_set, 0);
```

`time_alignment` is ((TOA_fb − 46) mod 1250)×4, and the net shift is **TOA_fb − 46 + 18.75 = TOA_fb − 27.25 bits**. The SCH follows the FCCH by exactly one frame, at the same position within the frame: in the SB window opened on the new time base it sits at 27.25 bits, plus any difference between the ROM's FB and SB measurement conventions, written Δ:

```
TOA_sb (first SB after FB synchronisation) = 27.25 + Δ
```

Git: the double subtraction dates from `cb71b972` (Welte, 2010-05-20, "Make new L1CTL_FBSB_REQ work reliably"), which introduces `fbinfo2cellinfo()` without removing the `toa -= 23` of the mode-1 block. In the initial import (`fbe7b94c`, 2010-02-18) the FB path subtracted 23 only once. `L1_SB_MARGIN_Q`, `L1_NB_MARGIN_Q`, both `toa -= 23` and `tpu_shift += 75` with its comment are all in the initial import.

2.6.1.6 5. Hardware logs: the predictions land — PROVEN

Relevant lines, verbatim, from log [1] (2011, four complete synchronisations) and log [2] (2024, one synchronisation):

```
[1] FB0 (11:6): TOA= 7200          FB1 (21:8): TOA= 9651         =>FB @ FNR 20 fn_offset=20 qbits=3420
    SB1 (46:1): TOA=   29          => SB 0x00c12184: BSIC=33 fn=89118(67/16/21) qbits=24
    =>FB @ FNR 45 fn_offset=89118 qbits=4932
    TOA AVG is not 16 qbits, correcting (got 20)
   FB0 (91861:3): TOA= 2544      FB1 (91871:8): TOA= 8751     =>FB @ FNR 91869 fn_offset=91869 qbits=4820
    SB1 (183745:1): TOA=   29      => SB 0x01e12284: BSIC=33 fn=91882(69/24/31) qbits=24
    =>FB @ FNR 183744 fn_offset=91882 qbits=4932
   FB0 (92514:8): TOA= 8784      FB1 (92524:8): TOA= 8755     =>FB @ FNR 92522 fn_offset=92522 qbits=4836
    SB1 (185051:1): TOA=   27      => SB 0x00852284: BSIC=33 fn=92535(69/ 1/21) qbits=16
    =>FB @ FNR 185050 fn_offset=92535 qbits=4924
    TOA AVG is not 16 qbits, correcting (got 19)
   FB0 (93574:9): TOA=10032      FB1 (93585:9): TOA=10003     =>FB @ FNR 93583 fn_offset=93583 qbits=4828
    SB1 (187172:1): TOA=   27      => SB 0x01582384: BSIC=33 fn=93596(70/22/11) qbits=16
    =>FB @ FNR 187171 fn_offset=93596 qbits=4924
    FB0 (94186:1): TOA=   48       FB1 (94197:9): TOA=10007
    FB1 (94217:11): TOA=12507      FB1 (94237:11): TOA=12507     FB1 (94248:2): TOA= 1259
[2] FB0 (192454:2): TOA= 1296      FB0 (192464:8): TOA= 8784     FB1 (192475:9): TOA=10007
    =>FB @ FNR 192473 fn_offset=192473 qbits=4844
    SB1 (384952:1): TOA=   24      => SB 0x019c4808: BSIC=2 fn=192485(145/ 7/11) qbits=4
    =>FB @ FNR 384951 fn_offset=192485 qbits=4912
    TOA AVG is not 16 qbits, correcting (got 17)
    FB0 (194158:8): TOA= 8784      FB0 (194168:8): TOA= 8784     FB1 (194178:8): TOA= 8755
    =>FB @ FNR 194176 fn_offset=194176 qbits=4836
    SB1 (388359:1): TOA=   26      => SB 0x01514908: BSIC=2 fn=194188(146/20/31) qbits=12
    =>FB @ FNR 388358 fn_offset=194188 qbits=4920
    (the "got 13" quoted earlier in this log precedes these lines)
```

5.1 The double subtraction (§4.4) is in the binary that runs. The qbits printed by `fbinfo2cellinfo` after the FB1 are ((TOA − 46) mod 1250)×4, never ((TOA − 23) mod 1250)×4:

FB1 TOA	printed qbits	((TOA − 46) mod 1250)×4	((TOA − 23) mod 1250)×4
9651	3420	**3420**	3512
8751	4820	**4820**	4912
8755	4836	**4836**	4928
10003	4828	**4828**	4920
10007 [2]	4844	**4844**	4936
8755 [2]	4836	**4836**	4928

And the `=>FB @ …` line printed after each SB comes from the task-completion handler `l1a_fb_compl` (`prim_fbsb.c:523-535`, registered at `:572`, triggered by `l1s_compl_sched(L1_COMPL_FB)` at `:266`), which calls `fbinfo2cellinfo` again (`:532`) on `last_fb`, where the SB result has already been reduced by 23 by `l1s_sbdet_resp` (`:207`): it reads ((TOA_sb − 46) mod 1250)×4, i.e. 29 → 4932 (×2), 27 → 4924 (×2), 24 → 4912, 26 → 4920. In total **12 lines out of 12** (6 after an FB1, 6 after an SB) follow the rule "−23 on every pass through `fbinfo2cellinfo`".

5.2 The ROM's immediates (§3). FB mode 1: 9651, 8751, 8755, 10003, 10007, 12507, 12507, 1259, 10007, 8755 are **all 3 (mod 4)** (10/10; 10007 = 48 · 208 + 23). FB mode 0: 7200, 2544, 8784, 10032, 48, 1296, 8784 are **all multiples of 48** (7/7: pages 150, 53, 183, 209, 1, 27, 183).

5.3 The hand-over from FB to SB (§4.4). Criterion: *every* first SB following an FB synchronisation, in both logs, with no exclusion (n = 6; synchronisations 2 and later start from an already aligned time base, which does not change the prediction, which depends only on TOA_fb):

log	FB1 TOA	shift applied (bits, mod 1250)	predicted SB position	measured SB TOA	Δ
[1]	9651	873.75	27.25	29	+1.75
[1]	8751	−26.25	27.25	29	+1.75
[1]	8755	−22.25	27.25	27	−0.25
[1]	10003	−24.25	27.25	27	−0.25
[2]	10007	−20.25	27.25	24	−3.25
[2]	8755	−22.25	27.25	26	−1.25

Mean Δ: **−0.25 bit**; spread ± 2.5, which is what the FB's granularity of 4 (§3) and the SB's integer rounding impose. The ROM's FB and SB measurement conventions coincide on silicon (Δ = 0 ± 2), and the SB arrives where the code puts it: 27 ± 2, **never 23**.

5.4 The 4.25 target (§4.3). After SB synchronisation, `toa.c` prints the average NB TOA over 250 frames **only when it differs from 16**: the lines "got 20", "got 19" ([1]), "got 13", "got 17" ([2]) are therefore the deviations by construction, and the periods at 16 are invisible. What can be said: every printed deviation is within 4 qbits (1 bit) of 16, for a prediction of 17 qbits = 4.25 bits (+ Δ', the NB/SB difference); none is at 23 or 27. And the FB1s of the subsequent synchronisations, taken from an already aligned time base, have TOA mod 1250 = 1, 5, 3, 7, 7, 9 (mean 5.3): the FCCH shows up at 4 ± 4 where the NB loop holds the bursts at 4. The three DSP tasks measure the TOA in the same convention, up to granularity.

2.6.1.7 6. What the 23 is — summary

statement	level	support
The DSP's TOA is a position within the window (index of the retained lag, leading edge of the channel estimate, §2.4), not an offset from nominal time	PROVEN	§2
The SB (191) and NB (151) windows are hard-wired in the ROM and equal 148 + 2 · 23 − 3 and 148 + 2 · 3 − 3	PROVEN	§1, §4.1

statement	level	support
The 23 of `prim_fbsb.c` is `L1_SB_MARGIN_Q`: same originating commit, same value, and it is the only margin that reproduces TI's window	PROVEN for the equality and the common origin; the author's intent is not documented	§4.1, §4.4
"TOA = 23" is, in the firmware's model, an SB at its nominal position, 23 bits after the window opens; in the ROM an SB centred in its 191 samples yields TOA 24.5 (21.5 + 3)	PROVEN (numbers); the 1-to-2-bit gap is absorbed by the +75 and by `toa.c`	§2.5, §4.3
The +75 qbits (= 23 − 4.25 bits) move the burst from the nominal SB position to the centre of the NB window, 40 samples shorter; `toa.c` then holds 16 qbits	PROVEN (code) and verified (logs: 13 to 20 qbits)	§4.3, §5.4
FB path: −23 twice, +75; the SB arrives at 27.25	PROVEN (code, git) and verified (logs: 24 to 29, mean 27.2)	§4.4, §5.1, §5.3
The second subtraction of the FB path was not removed when `fbinfo2cellinfo` was introduced (`cb71b972`); it places the SB near the centre of its window (27 instead of 4); intended or not: a question for the author	FACT (git); intent HYPOTHESIS	§4.4
The number 23 was chosen from the DSP's 191-sample window (TI documentation or experiment) rather than the other way round	VERY LIKELY (the ROM is earlier and frozen; the NB confirms the formula); not documented	§1, §4.1

Phrased for the list: *the DSP returns the position of the burst within its window (the leading edge of its channel estimate), not an offset; the DSP's SB window is 148 + 2 · 23 − 3 = 191 samples (a ROM immediate), and 23 is the margin with which `tpu_window.c` opens its own; `toa - 23` turns the position into an offset from an SB at its nominal place, and the `+ 75` of `synchronize_tdma` (= 23 − 4.25 bits) then shifts to the centre of the NB window, which `toa.c` holds at 16 qbits; on silicon, the SB arrives at 27 ± 2 after FB synchronisation (n = 6) and the NB TOA deviations that `toa.c` reports stay within 4 qbits of 16, as the code predicts.*

2.6.1.8 7. What the bench showed (orientation, not evidence)

- SB TOA = k + 2 (widened waveform) or k + 3 (clean waveform, at sampling instant d = 0), slope 1 with no spread over k = 0 to 40; NB TOA = k + 2 over k = −2 to 6. The "36/40 at k + 3" of an earlier version was measured at d = 0.5, a rounding switch point (TOA = round(k + 3 − d)); at d = 0 it is 40/40. Energy profiles read from the ROM: peak at k + 3 in all three waveforms, with 4 sliding 7-lag sums within 1 % of one another. With the rule of §2.4 (first lag of the window above threshold): bench waveform, E[k+2]/E[k+3] = 41 % → 23; bridge without widening, 1.4 % → 24; widened bridge, 45 % → 23; the three values the ROM published. This is what led to the static readings of §2. Whether the spread of the silicon SBs (24 to 29) comes from the same mechanism (a precursor path, or the receive filter putting energy at lag k + 2) remains a hypothesis.
- SB window: SCH decoded for k from −3 to 46 (centre 21.4); NB window: valid TOAs [0; 8], centre 4: this is what led to §4.3.
- **Fallback audit (§2.6)**: on a complete SB test (20 SCHs targeted, 40 attempts), the `A == 0` test at 0x7ccd was taken 20 times out of 40, always on the 2nd attempt, the one following the SCH frame (firmware and bench post two SB tasks on two consecutive frames, "as it is done by the TSM30"; the 2nd receives the next burst, which is not an SCH, and its channel coefficients are zero). All 20 retained TOAs (correct CRC) come from the main path. The 21/09 note in `MAILBOX.md` (TOA 23, 8/8 SBs decoded) meets the same criterion. The 23 published by the fallback is 21 + 2, the bench's default margin plus the ROM's offset: a coincidence of the bench, unrelated to the firmware's 23.

- **Withdrawn**: the "FCCH in the 1st search frame" test (TOA k − 7 to k − 11 in steps of 4, non-monotonic detection from k 34). Unexplained, hence outside the evidence; the logs (§5.2, §5.3) are sufficient for the FB.
- The `BANC_*` variables added to `tools/dsp_banc_commun.c` and the `fbpos` test remain in the repository for anyone who wants to redo these measurements.

2.6.1.9 8. What remains open, and whose it is

1. **The author's intent.** Can only be closed by Harald Welte or Dieter Spaar. Three questions for the list: was 23 chosen from the DSP's 191-sample window (TI documentation, experiment, TSM30); is the second subtraction of 23 in the FB path (`cb71b972`) intended; did the comment "an SB TOA of 4" refer to the residual after −23.
2. **Δ and Δ' to a tenth of a bit.** No bearing on the claim ($\Delta = 0 \pm 2$ is enough); out of scope without hardware. Verifiable by anyone with a C1xx: first SB after FB synchronisation at **27 ± 2** (firmware later than 2010-05-20), **4 ± 2** on earlier firmware. The list archives ("SB1 (", "TOA AVG is not") may still enlarge n.
3. **The ROM's choice between lags**: closed by static reading (§2.4, leading edge above 2.5 % of the mean energy). The behaviour on a real RF chain (how much energy at lag k + 2) is out of scope.
4. **N, p and the −5 of the FB formula** (§3): not worked out, no bearing on the conclusion.
5. **FreeCalypso / TI**: the TI sources (TCS211) may contain the SB window constant and its comment; not consulted here, status to be assessed.

2.6.1.10 Reproducing

```
#!/bin/bash
# Reproduces the "magic 23" evidence and writes everything to toa.log.
# Idempotent: skips downloads and builds already present.
# Needs: curl, python3, gcc, make; osmocom-bb is cloned if absent.
set -u
LOG=toa.log
: > "$LOG"
say() { printf '%s\n' "$*" | tee -a "$LOG"; }
run() { say "\$ $*"; "$@" 2>&1 | tee -a "$LOG"; say ""; }

say "# magic 23 reproduction — $(date -u +%Y-%m-%dT%H:%MZ) — $(uname -srm)"
say ""

# ------------------------------------------------------------------ 1. ROM
say "## 1. ROM dump -> PROM0.bin"
[ -f dsp-rom-3606-dump.txt ] || run curl -sO ftp://ftp.freecalypso.org/pub/GSM/Calypso/dsp-rom-3606-dump.txt
run sha256sum dsp-rom-3606-dump.txt
python3 - <<'E' 2>&1 | tee -a "$LOG"
import re,struct
sec=None;w={}
for l in open("dsp-rom-3606-dump.txt",errors="replace"):
    m=re.match(r"DSP dump: (\w+) \[",l)
    if m: sec=m.group(1); continue
    m=re.match(r"([0-9a-f]{5}) : ((?:[0-9a-f]{4} ?)+)",l)
    if m and sec=="PROM0":
        a=int(m.group(1),16)
        for i,x in enumerate(m.group(2).split()): w[a+i]=int(x,16)
open("PROM0.bin","wb").write(b"".join(struct.pack("<H",w[a]) for a in range(0x7000,0xe000)))
print("PROM0.bin: %d words (0x7000-0xdfff)" % len(range(0x7000,0xe000)))
for a in (0x7cbd,0x84a4,0xb21d,0xb2b1,0xb32d):
    print("  %04x: %04x %04x" % (a,w[a],w[a+1]))
E
run sha256sum PROM0.bin
say ""
```

```
# ------------------------------------------------------------ 2. disassembler
say "## 2. objdump tic54x (binutils 2.21.1)"
OBJDUMP=build-tic54x/binutils/objdump
if [ ! -x "$OBJDUMP" ]; then
   [ -f binutils-2.21.1.tar.bz2 ] || run curl -sO https://ftp.gnu.org/gnu/binutils/binutils-2.21.1.tar.bz2
    [ -d binutils-2.21.1 ] || tar xjf binutils-2.21.1.tar.bz2
    mkdir -p build-tic54x && ( cd build-tic54x && \
      ../binutils-2.21.1/configure --target=tic54x-coff --disable-werror --disable-nls --disable-gdb \
        --disable-gas --disable-ld --disable-gold --disable-gprof --disable-sim MAKEINFO=true \
        CFLAGS='-O1 -w -std=gnu89' >/dev/null && make -j8 all-binutils MAKEINFO=true >/dev/null ) \
      && say "built $OBJDUMP" || say "BUILD FAILED"
fi
run "$OBJDUMP" --version
"$OBJDUMP" -D -b binary -m tms320c54x --adjust-vma=0x7000 PROM0.bin > PROM0.dis 2>&1
say "full disassembly: PROM0.dis ($(wc -l < PROM0.dis) lines)"
say ""
say "### expected: 7cbd sub #11236,a | 84a4 stm #10791,ar3 | b21d stm #191,ar3 | b2b1 stm #151,ar3 | b32d stm #48,ar3"
grep -E '^ *(7cbd|84a4|b21d|b2b1|b32d):' PROM0.dis | tee -a "$LOG"
say ""
say "### SB TOA chain"
for a in 7c3d 81c8 7c49 7a1c 84a1 84ca 84dd 7c5f 7ca6 7cbc 7e8a b1e7; do
    grep -E "^ *$a:" PROM0.dis | tee -a "$LOG"
done
say ""
say "### FB TOA: 0x793e-0x795a"
awk '/^ *793e:/,/^ *795a:/' PROM0.dis | tee -a "$LOG"
say ""
say "### DMA programming routine: 0xa5cd-0xa5f6"
awk '/^ *a5cd:/,/^ *a5f6:/' PROM0.dis | tee -a "$LOG"
say ""

# ------------------------------------------------------------------ 3. git
say "## 3. layer1 git history"
[ -d osmocom-bb ] || run git clone -q https://gitea.osmocom.org/phone-side/osmocom-bb
run git -C osmocom-bb log --format='%h %ad %an %s' --date=short -S"toa -= 23" -- src/target/firmware/layer1
run git -C osmocom-bb log --format='%h %ad %an %s' --date=short -S"magic 23" -- src/target/firmware/layer1
say "### cb71b972: toa -= 23 / fbinfo2cellinfo lines (expect added fbinfo2cellinfo, no removed toa -= 23)"
git -C osmocom-bb show cb71b972 -- src/target/firmware/layer1/prim_fbsb.c | grep -nE '^[-+].*(toa -= 23|fbinfo2cellinf
say ""
say "### current constants"
grep -nE 'L1_(SB|NB)_MARGIN_Q|L1_TAIL_DURATION_Q|L1_(SB|NB)_DURATION_Q' osmocom-bb/src/target/firmware/layer1/tpu_w
grep -nE 'toa -= 23|magic 23' osmocom-bb/src/target/firmware/layer1/prim_fbsb.c | tee -a "$LOG"
grep -nE 'tpu_shift \+= 75|SB TOA of 4' osmocom-bb/src/target/firmware/layer1/sync.c | tee -a "$LOG"
say ""

# ------------------------------------------------------------------ 4. logs
say "## 4. hardware logs: qbits check ((TOA-46) mod 1250)*4"
say "sources: http://lists.osmocom.org/pipermail/baseband-devel/2011-September/002526.html"
say "         https://www.mail-archive.com/baseband-devel@lists.osmocom.org/msg01046.html"
python3 - <<'E' 2>&1 | tee -a "$LOG"
fb1 = [(9651,3420),(8751,4820),(8755,4836),(10003,4828),(10007,4844),(8755,4836)]
sb  = [(29,4932),(29,4932),(27,4924),(27,4924),(24,4912),(26,4920)]
ok = 0
for toa,q in fb1+sb:
    p46 = ((toa-46)%1250)*4; p23 = ((toa-23)%1250)*4
    ok += (p46==q)
    print("  TOA=%5d printed=%4d  -46:%4d %s  -23:%4d" % (toa,q,p46,"OK " if p46==q else "NO ",p23))
print("  %d/%d lines match the -46 rule" % (ok,len(fb1)+len(sb)))
fb0=[7200,2544,8784,10032,48,1296,8784]; m1=[9651,8751,8755,10003,10007,12507,12507,1259,10007,8755]
print("  FB mode 0 multiples of 48: %d/%d" % (sum(t%48==0 for t in fb0),len(fb0)))
print("  FB mode 1 == 3 mod 4:      %d/%d" % (sum(t%4==3 for t in m1),len(m1)))
```

```
d=[t-27.25 for t,_ in sb]
print("  first SB after FB sync: %s  mean delta %.2f" % ([t for t,_ in sb], sum(d)/len(d)))
print("  148 + 2*23 - 3 = %d ; 148 + 2*3 - 3 = %d ; (191-148+3)/2 = %g" % (148+2*23-3,148+2*3-3,(191-148+3)/2))
E
say ""
say "# done — $(wc -l < "$LOG") lines in $LOG"

# magic 23 reproduction — 2026-10-04T16:50Z — Linux 6.8.0-146-generic x86_64

## 1. ROM dump -> PROM0.bin
$ sha256sum dsp-rom-3606-dump.txt
e35798f50fd0a29aa61cd53e417c010d7f172582c7bc055211daab8af1e05470  dsp-rom-3606-dump.txt

PROM0.bin: 28672 words (0x7000-0xdfff)
  7cbd: f010 2be4
  84a4: 7713 2a27
  b21d: 7713 00bf
  b2b1: 7713 0097
  b32d: 7713 0030
$ sha256sum PROM0.bin
75ce5375a07eac85ee42ba00af7ca82089d15b5df517710a756e6c26497021d7  PROM0.bin


## 2. objdump tic54x (binutils 2.21.1)
$ build-tic54x/binutils/objdump --version
GNU objdump (GNU Binutils) 2.21.1
Copyright 2011 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.

full disassembly: PROM0.dis (28665 lines)

### expected: 7cbd sub #11236,a | 84a4 stm #10791,ar3 | b21d stm #191,ar3 | b2b1 stm #151,ar3 | b32d stm #48,ar3
    7cbd:   f010    sub    #11236,a
    84a4:   7713    stm    #10791,ar3
    b21d:   7713    stm    #191,ar3
    b2b1:   7713    stm    #151,ar3
    b32d:   7713    stm    #48,ar3

### SB TOA chain
    7c3d:   7712    stm    #3278,ar2
    81c8:   f272    rptbd  0x81db
    7c49:   7712    stm    #11498,ar2
    7a1c:   fe00    retd
    84a1:   f7b6    ssbx   st1,frct
    84ca:   f6b6    rsbx   st1,frct
    84dd:   e900    ld     #0,b
    7c5f:   7210    mvdm   0x2f06,ar0
    7ca6:   f4bc    rsbx   st0,tc
    7cbc:   4817    ldm    ar7,a
    7e8a:   711f    mvdk   DP+0x1f,0x3fa4
    b1e7:   10f8    ld     *(0x3fa4),a

### FB TOA: 0x793e-0x795a
    793e:   1194    ld     *ar4+,b
    793f:   f762    sfta   b,2
    7940:   44f8    ld     *(0x3fb4),16,a
    7941:   3fb4
    7942:   f061    sub    #3,16,a
    7943:   0003
```

```
    7944:   3c84    add    *ar4,16,a
    7945:   f061    sub    #2,16,a
    7946:   0002
    7947:   f48c    mpya   a
    7948:   f500    add    a,b
    7949:   f310    sub    #5,b
    794a:   0005
    794b:   6ff8    sub    *(0xc3d),b,a
    794c:   0c3d
    794d:   0e20
    794e:   fc47    rc     aleq
    794f:   44f8    ld     *(0x3fb3),16,a
    7950:   3fb3
    7951:   f061    sub    #3,16,a
    7952:   0003
    7953:   770e    stm    #48,t
    7954:   0030
    7955:   f48c    mpya   a
    7956:   08f8    sub    *(0xc3e),a
    7957:   0c3e
    7958:   f620    sub    b,a
    7959:   fc47    rc     aleq
    795a:   81f8    stl    b,*(0x8fa)

### DMA programming routine: 0xa5cd-0xa5f6
    a5cd:   ea86    ld     #134,dp
    a5ce:   76f8    st     #-3980,*(0x158)
    a5cf:   0158
    a5d0:   f074
    a5d1:   76f8    st     #29250,*(0x159)
    a5d2:   0159
    a5d3:   7242
    a5d4:   76f8    st     #-3981,*(0x15a)
    a5d5:   015a
    a5d6:   f073
    a5d7:   4811    ldm    ar1,a
    a5d8:   80f8    stl    a,*(0x15b)
    a5d9:   015b
    a5da:   69f8    orm    #4,*(0x3fdc)
    a5db:   3fdc
    a5dc:   0004
    a5dd:   75f8    portw  *(0x3fdc),pa63744
    a5de:   3fdc
    a5df:   f900
    a5e0:   47f8    rpt    *(0xc32)
    a5e1:   0c32
    a5e2:   f495    nop
    a5e3:   4812    ldm    ar2,a
    a5e4:   f010    sub    #2048,a
    a5e5:   0800
    a5e6:   6f56    stl    a,1,DP+0x56
    a5e7:   0c81
    a5e8:   7556    portw  DP+0x56,pa64548
    a5e9:   fc24
    a5ea:   4813    ldm    ar3,a
    a5eb:   f110    sub    #48,a,b
    a5ec:   0030
    a5ed:   6f56    stl    a,2,DP+0x56
    a5ee:   0c82
    a5ef:   7556    portw  DP+0x56,pa64550
    a5f0:   fc26
```

```
    a5f1:   7656    st     #1197,DP+0x56
    a5f2:   04ad
    a5f3:   ff4d    xc     2,beq
    a5f4:   7656    st     #1193,DP+0x56
    a5f5:   04a9
    a5f6:   7556    portw  DP+0x56,pa64552

## 3. layer1 git history
$ git clone -q https://gitea.osmocom.org/phone-side/osmocom-bb

$ git -C osmocom-bb log --format=%h %ad %an %s --date=short -Stoa -= 23 -- src/target/firmware/layer1
8978ec57 2010-05-19 Harald Welte layer1/l1ctl: Split L1CTL_NEW_CCCH_REQ in FBSB_REQ nad SYNC_REQ
5f751d3d 2010-04-09 Harald Welte layer1: move FB and SB detection into prim_fbsb.c
fbe7b94c 2010-02-18 Harald Welte Initial import of OsmocomBB into git repository

$ git -C osmocom-bb log --format=%h %ad %an %s --date=short -Smagic 23 -- src/target/firmware/layer1
8978ec57 2010-05-19 Harald Welte layer1/l1ctl: Split L1CTL_NEW_CCCH_REQ in FBSB_REQ nad SYNC_REQ
5f751d3d 2010-04-09 Harald Welte layer1: move FB and SB detection into prim_fbsb.c
48dfd41e 2010-04-07 Harald Welte layer1: add 'struct mon_state' argument to read_{fb,sb}_result()

### cb71b972: toa -= 23 / fbinfo2cellinfo lines (expect added fbinfo2cellinfo, no removed toa -= 23)
20:@@ -87,7 +87,7 @@ static void dump_mon_state(struct mon_state *fb)
29:@@ -98,14 +98,16 @@ static int l1ctl_fbsb_resp(uint8_t res)
48:@@ -179,25 +181,24 @@ static int l1s_sbdet_resp(__unused uint8_t p1, uint8_t attempt,
80:@@ -225,25 +226,14 @@ static int l1s_sbdet_resp(__unused uint8_t p1, uint8_t attempt,
108:@@ -259,18 +249,13 @@ static int l1s_sbdet_resp(__unused uint8_t p1, uint8_t attempt,
134:@@ -279,6 +264,9 @@ static int l1s_sbdet_cmd(__unused uint8_t p1, __unused uint8_t p2,
144:@@ -330,6 +318,41 @@ static int read_fb_result(struct mon_state *st, int attempt)
148:+static void fbinfo2cellinfo(struct l1_cell_info *cinfo,
154:+   last_fb->toa -= 23;
186:@@ -354,6 +377,8 @@ static int l1s_fbdet_cmd(__unused uint8_t p1, __unused uint8_t p2,
195:@@ -363,16 +388,7 @@ static int l1s_fbdet_resp(__unused uint8_t p1, uint8_t attempt,
213:@@ -385,8 +401,22 @@ static int l1s_fbdet_resp(__unused uint8_t p1, uint8_t attempt,
236:@@ -398,7 +428,8 @@ static int l1s_fbdet_resp(__unused uint8_t p1, uint8_t attempt,
246:@@ -408,6 +439,7 @@ static int l1s_fbdet_resp(__unused uint8_t p1, uint8_t attempt,
254:@@ -423,8 +455,15 @@ static int l1s_fbdet_resp(__unused uint8_t p1, uint8_t attempt,
267:+               fbinfo2cellinfo(&l1s.serving_cell, last_fb);
271:@@ -458,8 +497,6 @@ static const struct tdma_sched_item fb_sched_set[] = {
280:@@ -467,35 +504,10 @@ static void l1a_fb_compl(__unused enum l1_compl c)
285:-   last_fb->toa -= 23;
313:+   fbinfo2cellinfo(&l1s.serving_cell, last_fb);
319:@@ -513,6 +525,9 @@ void l1s_fbsb_req(uint8_t base_fn, struct l1ctl_fbsb_req *req)

### current constants
38:#define L1_NB_MARGIN_Q       (3 * 4)
39:#define L1_SB_MARGIN_Q       (23 * 4)
40:#define L1_TAIL_DURATION_Q   (3 * 4)
43:#define L1_NB_DURATION_Q (L1_BURST_LENGTH_Q + 2 * L1_NB_MARGIN_Q - L1_TAIL_DURATION_Q)
44:#define L1_SB_DURATION_Q (L1_BURST_LENGTH_Q + 2 * L1_SB_MARGIN_Q - L1_TAIL_DURATION_Q)
54: [L1_RXWIN_SB]   = L1_SB_DURATION_Q,
55: [L1_RXWIN_NB]   = L1_NB_DURATION_Q,
206:    last_fb->toa -= 23;
333:    /* FIXME: where did this magic 23 come from? */
334:    last_fb->toa -= 23;
469:    /* FIXME: where did this magic 23 come from? */
470:    last_fb->toa -= 23;
119:     * is within 0...8. We have to add 75 to get an SB TOA of 4. */
120:    tpu_shift += 75;

## 4. hardware logs: qbits check ((TOA-46) mod 1250)*4
sources: http://lists.osmocom.org/pipermail/baseband-devel/2011-September/002526.html
```

```
        https://www.mail-archive.com/baseband-devel@lists.osmocom.org/msg01046.html
  TOA= 9651 printed=3420  -46:3420 OK   -23:3512
  TOA= 8751 printed=4820  -46:4820 OK   -23:4912
  TOA= 8755 printed=4836  -46:4836 OK   -23:4928
  TOA=10003 printed=4828  -46:4828 OK   -23:4920
  TOA=10007 printed=4844  -46:4844 OK   -23:4936
  TOA= 8755 printed=4836  -46:4836 OK   -23:4928
  TOA=   29 printed=4932  -46:4932 OK   -23:  24
  TOA=   29 printed=4932  -46:4932 OK   -23:  24
  TOA=   27 printed=4924  -46:4924 OK   -23:  16
  TOA=   27 printed=4924  -46:4924 OK   -23:  16
  TOA=   24 printed=4912  -46:4912 OK   -23:   4
  TOA=   26 printed=4920  -46:4920 OK   -23:  12
  12/12 lines match the -46 rule
  FB mode 0 multiples of 48: 7/7
  FB mode 1 == 3 mod 4:      10/10
  first SB after FB sync: [29, 29, 27, 27, 24, 26]  mean delta -0.25
  148 + 2*23 - 3 = 191 ; 148 + 2*3 - 3 = 151 ; (191-148+3)/2 = 23

# done — 195 lines in toa.log
```

2.7 /opt/GSM/c54x_exe/docs/CLAIMS-2026-10-04.md

19354 octets, 272 lignes

2.7.1 Emulating the baseband — claims and evidence (2026-10-04)

This is the evidence file for the article, not the article. Each quoted sentence is followed by what the code, the ROM and the bench show. Read « VERIFIED » as « verified in the full system (the phone registers, calls, speaks) » unless the bench says otherwise, in which case both verdicts are given.

The text below is the article as drafted (quoted blocks), followed, claim by claim, by what the repositories, the ROM and the bench actually show on 2026-10-04. Each check is marked **VERIFIED** (reproducible from code, ROM disassembly or a run), **CORRECTED** (the claim as written is not what the code does; the corrected statement follows), or **OPEN** (not checked). Sources: `c54x_exe` (this repository), `/opt/GSM/qosmo` (QEMU fork + C54x core, `hw/arm/calypso/`), `/opt/GSM/osmocom-bb` (layer 1), `/opt/GSM/osmo-operator/pont` (bridge), `/opt/GSM/tests` (bench), the public ROM dump `ftp://ftp.freecalypso.org/pub/GSM/Calypso/dsp-rom-3606-dump.txt` (PROM0 identical, word for word, to the one used here), and the binutils `tic54x` disassembler built from the 2.21.1 sources.

> A complete GSM call on an ordinary PC, where the phone's real layer 1 firmware and real baseband DSP code take part, down to the lowest layers of 2G. I forked QEMU to emulate the TI Calypso platform so that OsmocomBB's layer 1 runs on it, and I wrote an emulator for the TI C54x DSP that layer 1 talks to. Nothing goes on air: no SDR, no radio hardware, no spectrum licence.

VERIFIED (call, SMS, A5, speech both ways) — `MAILBOX.md`, entry « 2026-09-23 20:40 », run of 20:22: location update, MO call to the Asterisk echo (ASSIGNMENT COMPLETE 20:22:54, ACTIVE, DISCONNECT), SMS MT and MO with CP-ACK/RP-ACK, MT call via osmo-sip-connector, « chiffrement descendant confirmé par la BTS » five times, speech audible both ways. Nothing on air by construction: the bridge speaks TRXD over UDP to `osmo-bts-trx` (`README.md`, « Architecture »).

VERIFIED (QEMU fork, Calypso machine) — `/opt/GSM/qosmo/hw/arm/calypso/`, machine `-M calypso -cpu arm946` (`run.sh`, QEMU command line); `calypso_l1_dispatch.c`, `l1-dsp/`, `l1-grgsm/`.

VERIFIED (C54x emulator) — `/opt/GSM/qosmo/hw/arm/calypso/l1-dsp/{c54x_exec.c,c54x_decode.c,c54x_mem.c,c54x_irq.c,calypso_c54x.c}`, 1.1 MB of C; `c54x_exe` links them (`Makefile`, `SRC`).

> On the phone side, the QEMU Calypso machine loads the OsmocomBB layer 1 firmware as the real ARM would, and a native C54x emulator runs the DSP with its ROM downloaded from FreeCalypso rather than redistributed.

VERIFIED since 2026-10-04 (was CORRECTED) — until this audit, `run.sh` (be10de1) defaulted to `layer1.highram_patched.elf`, a build that passed downlink speech frames flagged `B_BFI` at zero gain (« trames B_BFI, vols FACCH compris, passées à charge nulle, masquées par l'ECU de GAPK »). On 2026-10-04 the default was put back to the unmodified `layer1.highram.elf` and the patched `.elf/.bin/.map` were removed from `/opt/GSM/firmware/board/compal_e88/` (the `layer1.highram_patched.patch` file stays as a record of what was tried). The consequence to watch: whatever the ROM flags `B_BFI` is now handled by the firmware as on a phone.

CORRECTED (ROM download) — `install.sh` contains no FreeCalypso URL (`grep -n freecalypso install.sh` is empty); the ROM images are read from `rom/calypso_dsp.*.bin` (`src/main.c`, `ROMS[]`), and `rom/calypso_dsp.Registers.bin` is tracked in git. The PROM0 used is identical to the public FreeCalypso dump (28,672 words, 0 differences), so the sentence is true of the content, not of the mechanism. Either make `install.sh` fetch the dump, or write « identical to the public FreeCalypso dump ».

Layer 1 drives the DSP through its shared API, as on real hardware.

VERIFIED — `README.md` (« the ROM's API RAM window is memory-mapped and shared with QEMU », `/dev/shm/calypso_api_ram`); frame-by-frame lockstep over `/tmp/calypso_dsp.sock` (`CALYPSO_PONT_LOCKSTEP=1`, `run.sh:57-63`).

In the downlink, the DSP ROM itself does the work: frequency burst detection, synchronisation, control channel and speech decoding, A5 deciphering.

VERIFIED in the full system, FAIL on the bench for FB and NB — `/opt/GSM/tests/modules/99-couverture.sh`, section « CHEMIN ROM / HOTE », mode `dsp`, lines 128-134: FB (`d_fb_det`), SB/SCH (`a_sch`), BCCH/CCCH/SDCCH/SACCH (`a_cd`), FACCH (`a_fd`), TCH/F speech (`a_dd`, read as is by the firmware), A5 downlink (`[a5]`, modelled coprocessor `calypso_a5.c`). Run of 2026-10-03 17:11 (`MAILBOX.md`, last entry): « 6 ROM ». Known limits in the same table: SACCH/8 downlink « souvent FIRE KO » (the 2026-10-03 note in `MAILBOX.md` says « ~80 % », without a count), B_BFI frames masked by the patched firmware. On the bench (`./dsp_tester --all`, 2026-10-04): `sb` PASS (20/20), but `fb0`/`fb1` FAIL (5/6 detections, one FCCH frame mis-deduced), `fbsb` FAIL, `nb` FAIL (8/8 blocks with FIRE OK but a content different from what the bench expects — the bench's expectation has not been checked). The system works end to end; the bench says that on FB and NB either the ROM, the core or the bench computes something wrong, and that is not settled.

In the uplink, SDCCH is now produced by the ROM too, after four bugs in the emulated core were fixed;

VERIFIED — `src/pont.c`, `tx_rom_publier()` (bursts the ROM serialises into `data[0x3f8a]`, published to `/dev/shm/calypso_xcch_ul_rom`); bridge `pont/dsp/uplink.py`, `_bursts_xcch()` (`PONT_UL_ROM`); run 17:11: « xCCH ul rom=134 hote=1 ». The four core fixes are in the qosmo history: `4771f91d` (0xA1xx = ADD Xmem,Ymem,B; circular `*+ARx(lk)%`; SFTA carry; AND/OR/XOR src,SHIFT,dst), `1e47cd72` (F0Bx/F1Bx are OR A,-16..-1,dst, not RSBX/SSBX), `e8f477ad` (SQURS, POLY, MAC*, SUBC carry), `a19dde59` (MVKD/MVDK under RPT increment DAR). Bench: `./dsp_tester tx-sdcch` → PASS, 24/24 bursts identical to `gsm0503_xcch_encode` (after aligning the bench's hl/hu capture with `pont.c`, 2026-10-04).

RACH, FACCH/SACCH on traffic channels, uplink speech and uplink A5 are still encoded on the host side, and moving them into the DSP is the current work.

VERIFIED (state on 2026-10-03) — `99-couverture.sh:135-141`: RACH (`montant.c` reads `d_rach`, the bridge codes), A5 uplink (the bridge undoes the ROM's XOR and re-ciphers at the air fn), FACCH/F (`a_cu` → bridge `TchScheduler`), SACCH on TCH (bridge), TCH/F speech (`montant.c` TI→FR → bridge `tch_fr_encode`), vocoder (GAPK). `src/montant.c:1236-1245`: `d_task_ra` is erased from the W page by default (`MONTANT_EFFACE_TASK_RA=1`), so the ROM never executes the RACH task in the full system.

NEW on 2026-10-04 — RACH is bit-exact in the ROM *on the bench*; in the full system the host still codes it (`d_task_ra` erased, bridge `gsm.rach_burst`) — see « What the RACH trace shows » below. The diagram and the article must say « RACH: ROM on the bench, host in the full system ».

A bridge connects the phone to osmo-bts-trx, reusing code from Vadim Yanitskiy (fixeria),

OPEN (attribution not found) — `grep -rin "fixeria\|yanitskiy" /opt/GSM/osmo-operator/pont/` finds nothing in the bridge sources or its README. If code was reused, the attribution should be in the files; if the sentence refers to libosmocoding/trxcon more generally, say so.

and an Osmocom core network handles the call on the other side. An end-to-end test bench checks the whole chain, from boot to attach, SMS, USSD, a voice call and a test tone recovered through it.

VERIFIED — `/opt/GSM/tests/modules/`: `00-demarrage`, `10-coeur`, `20-bts`, `30-services`, `60-camp`, `70-attache`, `80-sms-mo`, `85-sms-mt`, `87-ussd`, `90-appel`, `92-voix`, `99-couverture`.

> An experimental mode already runs layer 1 without any TI ROM, a first step toward a fully free baseband stack.

CORRECTED (« fully free baseband stack ») — `MODE=grgsm ./run.sh` (`run.sh:16,42,95-98`: « montage grgsm : pas de c54x_exe, la couche 1 est dans QEMU »), `/opt/GSM/qosmo/hw/arm/calypso/l1-grgsm/`. In this mode the DSP functions are reimplemented on the host (gr-gsm, libosmocoding, the bridge for A5): « layer 1 without the TI ROM, with host-side signal processing », not a free baseband.

2.7.1.1 Three stages, as they appear in the code

1. Layer 1 in QEMU. The QEMU fork runs an unmodified `layer1.highram` on the Calypso machine.

VERIFIED since 2026-10-04 — see above: the patched default is gone.

2. The gr-gsm shunt. […] The DSP was bypassed; the host rebuilt the signal processing.

VERIFIED — `99-couverture.sh:117-124`, mode `grgsm`: « FB/SB : hôte : gr-gsm dans QEMU », « descendant : hôte : pont (libosmocoding) », « A5 : hôte : pont », « montant : hôte ».

3. The DSP that works. […] State on 2026-10-03: the whole downlink is ROM; uplink SDCCH is ROM; still on the host: RACH, FACCH/SACCH on TCH, uplink speech, uplink A5, the vocoder, plus a small patch of the IMMEDIATE ASSIGNMENT.

VERIFIED — `99-couverture.sh:131`: « IMMEDIATE ASSIGNMENT : ROM + retouche hôte de la référence (qosmo calypso_trx.c) ; l'hôte réécrit la référence de requête lue dans a_cd ». The line has no ROM pattern, so the table's verdict for it is HOTE by construction (it is one of the 7 host steps of the 17:11 run); the single MIXTE line is the SDCCH uplink (`:136`, rom=134 hote=1). The retouch exists because the RACH is sent by the bridge at a frame number the firmware did not choose; a ROM RACH sent at the ROM's frame removes the need for it (not yet done in the full system).

2.7.1.2 What "processor fidelity" means now

> Capturing at `0x3f8a` (what `tx_rom_publier` does) takes the coded bits *before* the modulator. The host still has to add the training sequence, synchronisation and tail bits, and redo A5 at the right frame number. That is ROM at about 80 %.

VERIFIED (mechanism), CORRECTED (the number) — `src/pont.c:555-600`: 114 data bits + hl/hu read from `data[0x3f8a..0x3f91]`; the A5 stream the ROM XORed is read back from `data[0x3f9b..]` and published so that the bridge can undo it and re-cipher (`pont/dsp/uplink.py:24-36`). « About 80 % » is not a measurement; the only count available is the coverage table (per treatment: 6 ROM, 7 host, 1 mixed on 2026-10-03). Replace the percentage by that count or by a bit count.

> Capturing the TSP script (`0x3cbb..`, marker `0x1c0a` = TOGBR2, then 16 BULDATA words at 10 bits) takes the stream *the ABB would receive on silicon*, with training sequence, tails, I/Q inversion and A5 already applied. The host then only converts ABB words into 148 bits and places them at the right fn/tn. This capture point is valid for every burst type (NB, RACH, TCH), and it is the one that matches the goal: a single pipe, the ROM decides everything.

VERIFIED in the ROM, and now reproduced in emulation (2026-10-04) — static reading of PROM0:

- `0xb5b8 stm #15547,ar2` (0x3cbb): the ROM builds a TSP script for the ABB there each frame (register writes requested through NDB words 0x8ed..0x8f7, `0xb5ba-0xb5e5`).
- `0xb6fd` (RACH) / `0xb70f` (NB): `st #7178,*ar2+` → 0x1c0a = TOGBR2 (page 0, reg 5) with data 0x70, then the burst through dispatcher code 39 (`data[0x43ae]` = 0xb77f → `fcall 0x859f` → emitter `0x85a2`) or code 40 (NB emitter `0x85c1`), then the power word `*(0x3fc9)`, then `st #18` (AUXAPC).
- Emitter `0x8605-0x868f`: a segment table driven by DP+0x49 (loads from the sync/TSC copy at 0x3dca, the data at 0x3f8a, the tail/guard words at 0x3dcd), a bit accumulator in B, and `sth a,*ar2+` at `0x8632` with the register bits added (`add #6,16,a` = BULDATA1, reg 3; TSP encoding `(data & 0x3ff) << 6 | reg << 1`, `abb/twl3025.c:156`). The emitted word is therefore `(10 data bits) << 6 | 0x06`.

- `0x85c1` (NB): `xor *ar4+` over the 8 words 0x3f8a..0x3f91 with 0x3f9b.. (A5 stream), then the 26-bit TSC copied from DROM 0xa0c9 + 2 · tsc (the 8 TSCs are at DROM 0xa0c9..0xa0d8). `0x85a2` (RACH): the 41-bit sync copied from DROM 0xa0d9..0xa0db (`4b7f 99aa 3c00` = 01001011011111111001100110101010001111000). Tail/guard words DROM 0xa0dc..0xa0dd (`1fff ffff`).
- `0xb773`: `xorm #0xAA80` over the 16 words when bit 14 of `0x3f92` is set — the I/Q inversion flips every other data bit (mask 1010101010 in the 10 data bits).

Bench, `./dsp_tester rach` (new test, 2026-10-04): for five random RA values with BSIC 7, the 16 words after 0x1c0a decode to 160 bits = `11111111` (ramp fill) + `00111010` (extended tail, 05.02 §5.2.7) + the 41-bit synchronisation sequence + the 36 coded bits + `000` + guard ones; the 36 coded bits are identical to `gsm0503_rach_ext_encode(ra, bsic)` (5/5), and the ROM's own copy of them in `data[0x3f8a..0x3f8c]` as well (5/5). Example, ra=0xd4: `1111111100111010 01001011011111111001100110101010001111000 000011011110000101011001111010110000 1111…`.

CORRECTED (« reachable without touching the core ») — it was not. In emulation the emitter never produced the 16 words before 2026-10-04, for two reasons in the C54x core (`qosmo`, `l1-dsp/c54x_exec.c`):

1. `CMPR CC, ARx` (binutils `0xF4A8` mask `0xFCF8`, 78 sites in PROM0) was not implemented: `cmpr eq` fell into `unimpl` (a no-op, TC unchanged) and `cmpr lt/gt/neq` into `RPT #k`, the F6xx no-op and the F7xx block. The emitter's loop `0x8608 cmpr eq,ar2 ; 0x8609 rc tc` (AR0 = end of the 16 words) returned at once on a stale TC = 1 (probe: ST0 = 0x107b before and after the CMPR). Fixed; `CALYPSO_CMPR_ANCIEN=1` restores the old behaviour.
2. `LD src, ASM [, dst]` (`0xF482` mask `0xFCFF`) ignored the ASM shift (ST1[4:0], two's complement): the accumulator was never shifted and all 16 words came out as `0xff06` (probe: B unchanged across `0x8625 ld b,asm` with ASM = 10). Fixed; `CALYPSO_LDASM_ANCIEN=1` restores the old behaviour.

So the TSP script is the right capture point, but it only exists in emulation since these two fixes. With them, the bench's `tx-sdcch` (normal bursts, ROM-coded SDCCH) still passes 24/24.

> What the RACH trace shows points the same way: the ROM does execute the RACH task (`0xb6fd`, then `0x85a2` on each attempt), writes its 36 coded bits to `0x3f8a`, and the transmitter fills in the script. The only obstacle is on the system side: `montant.c` clears `d_task_ra` by default (`MONTANT_EFFACE_TASK_RA=1`), so the ROM has never been allowed to do a real RACH. The bit differences seen against the host-side encoder settle nothing until that 05.03 implementation has been checked against `gsm0503_rach_ext_encode` on the bench, which is the next measurement.

VERIFIED, and the measurement is done — probe on the bench: `0xb6fd` and `0x85a2` on each of the 5 attempts; `data[0x3f8a..0x3f8c]` = the 36 coded bits (`0de1 59eb 3000` for ra=0xd4). The « bit differences » were an error in a hand-written 05.03 encoder (the CRC-6 remainder is inverted, `remainder=0x3f` in `gsm0503_rach_crc6`); against libosmocoding the ROM is exact, 5/5. The system-side obstacle is as stated (`montant.c:1236-1245`); the second obstacle, the two core bugs above, is now removed on the bench. What remains for the full system: publish the TSP burst from `pont.c`, let the bridge send it at the ROM's frame, stop erasing `d_task_ra`.

2.7.1.3 The hierarchy for what comes next

> Maximum fidelity, reachable without touching the core: make the TSP script the single source of all uplink bursts (RACH, SDCCH, TCH, SACCH, FACCH), and transmit at the ROM's frame number so that its A5 is the right one.

CORRECTED (core) — see above. **OPEN (A5 at the ROM's frame)** — the bridge's comment says the ROM « chiffre à l'heure du DSP, pas au fn émis » (`pont/dsp/uplink.py:24`, `99-couverture.sh:137`); which frame number the ROM uses for the uplink stream (`a_a5fn`, `d_fn`, or its own counter) has not been read in the ROM, and whether « the ROM's frame » equals the air frame once the bridge sends immediately has not been measured. To check before building on it.

> Fidelity that requires work on the core: the cases where the ROM computes wrongly or not at all (downlink SACCH/8 at 80 % FIRE failures, `B_BFI`, the bench's `nb` and `fb0/fb1` tests).

VERIFIED (the list), and one number the article must state — ISA conformance of the emulated core (`./isa_test tools/isa_tests.txt`, the worked examples of SPRU172C): **of 230 examples, 150 pass, 58 fail and 22 cannot be assembled by the harness** (208 `T` records and 22 `S` records in `tools/isa_tests.txt`; unchanged by the two fixes above; CMPR is among the « not assembled »). A quarter of the manufacturer's examples fail. Every « the ROM does X » is conditional on the core executing the instructions of that path correctly, and the two bugs of 2026-10-04 were found exactly that way. The list of failing examples

(`./isa_test tools/isa_tests.txt | grep ^FAIL`) is the most profitable work queue of the project before any new pipe. Bench on 2026-10-04: `fb0`, `fb1`, `fbsb`, `nb` FAIL; `boot`, `idle`, `checksum`, `fb-afc`, `sb`, `tx-sdcch`, `rach` PASS. `99-couverture.sh:130,133`: SACCH/8 FIRE KO, B_BFI.

> The most distant fidelity: the vocoder in the ROM, which means modelling the audio path from the DSP to the ABB.

OPEN — plausible (the ROM delivers TI-format speech frames to `a_dd`, GAPK decodes them on the host, `99-couverture.sh:141`); nothing measured.

> In other words, the DSP has stopped being an obstacle to work around and has become the reference: whenever the host still does something, it is either because it has not yet been given the ROM's pipe, or because the core still has a bug. Both are measurable, and that is what is being measured.

VERIFIED as a description of the method — the RACH case of 2026-10-04 is exactly that: a ROM pipe not yet used (`d_task_ra` erased), and two core bugs found by following the ROM's own code to the point where it writes the burst.

2.7.1.4 Reproducing the new measurements

```
# ROM : disassembly of the emitter and of the TSP script builder
# the tic54x target is gone from current binutils (2.42): build objdump from the GNU 2.21.1 sources
# (tested 2026-10-04, gcc 13 -- the old C needs -std=gnu89)
curl -O https://ftp.gnu.org/gnu/binutils/binutils-2.21.1.tar.bz2
tar xjf binutils-2.21.1.tar.bz2 && mkdir build-tic54x && cd build-tic54x
../binutils-2.21.1/configure --target=tic54x-coff --disable-werror --disable-nls --disable-gdb --disable-gas \
    --disable-ld --disable-gold --disable-gprof --disable-sim MAKEINFO=true CFLAGS='-O1 -w -std=gnu89'
make -j8 all-binutils MAKEINFO=true && cd ..
build-tic54x/binutils/objdump -D -b binary -m tms320c54x --adjust-vma=0x7000 PROM0.bin | less    # 0xb5b8, 0xb6fd, 0xb7
                                                          # 0x859f/0x85a2, 0x859c/0x85c1, 0x8605-0x868f
# DROM tables : 0xa0c9..0xa0d8 (8 TSC), 0xa0d9..0xa0db (RACH sync), 0xa0dc..0xa0dd (tail/guard)
# bench
make dsp_tester && ./dsp_tester rach tx-sdcch -v
CALYPSO_CMPR_ANCIEN=1 ./dsp_tester rach          # before the CMPR fix : no burst in the script
CALYPSO_LDASM_ANCIEN=1 ./dsp_tester rach         # before the LD src,ASM fix : 16 x 0xff06
```

2.8 /opt/GSM/c54x_exe/docs/README.fr.md

22514 octets, 410 lignes

2.8.1 c54x_exe — le DSP Calypso sans QEMU

Fait tourner la **mask-ROM TI du TMS320C54x** seule : pas de QEMU, pas d'ARM, pas de firmware osmocom-bb.

```
make
./c54x_exe --trames 200
./c54x_exe --trames 50 --verbeux          # une ligne par trame
./c54x_exe -vv                            # traces du coeur : -v .. -vvvvvv, voir --help
```

Par defaut seules les erreurs du coeur C54x passent sur stderr ; le bilan dit combien de lignes ont ete masquees et avec quel `-v` les voir. `-vvvvvv` rend stderr brut. Le classement est par mots-clefs sur le nom de la sonde (`src/verbosite.c`), pas par liste : une sonde nouvelle tombe dans un niveau raisonnable sans declaration.

2.8.1.1 Lancer exe par exe : le cote mobile, sans le reseau

[2026-09-17] Quatre executables, dans cet ordre, chacun attendant le precedent. `run.sh` fait exactement ca ; ce qui suit est la version a la main, pour voir ce que chaque etape produit. Aucun element reseau (BTS, BSC, MSC, pont gr-gsm) : le mobile cherche une cellule et n'en trouve pas, c'est attendu tant qu'aucun burst n'est injecte dans le DSP. Avec la BTS et `PONT=1`, voir l'etat du 2026-09-23 plus bas.

```
./run.sh              # tout, dans l'ordre        ./run.sh --status   qui tourne
./run.sh --logs       # suivre les 4 journaux      ./run.sh --stop     tout arreter, nettoyer
./run.sh --step 3     # une seule etape (les precedentes doivent tourner)
INSNS=8000 VERB=-vv ./run.sh           # budget DSP par trame (80000 par defaut), niveau de traces
MODE=grgsm ./run.sh                    # l'autre montage : couche 1 gr-gsm dans QEMU, sans c54x_exe
PONT=1 ./run.sh                        # avec l'etape 5 (le pont TRX) ; defaut PONT=0 : le mobile seul
IQ=cell ./run.sh                       # c54x_exe fabrique une cellule GMSK (FCCH/SCH) a chaque trame
LOCKSTEP=0 ./run.sh                   # horloge murale : QEMU n'attend pas le DSP et saute des trames (defaut 1)
```

`run.sh` connait deux montages (`MODE=dsp`, le defaut, et `MODE=grgsm`) et une cinquieme etape, le pont TRX, qui relie le BTS osmo-bts-trx (TRXD 5700-5702) a la couche 1 du mobile : GSMTAP 4730/4731 pour gr-gsm, `--dsp-port 6702` vers le BSP de `c54x_exe` pour le DSP. [2026-09-23] Le defaut est `PONT=0` (depuis le 17/09) : `PONT=1 ./run.sh` ajoute l'etape 5. En `MODE=dsp` elle lance `pont/pont_dsp.py` (sous-paquet `pont/dsp/`, bascule TCH suivie par le firmware), en `MODE=grgsm` `pont/pont.py` ; `PONT_PY` force l'un ou l'autre. Le pont n'est plus lance avec `--no-record` (`PONT_AIRREC=1` par defaut, pour la FFT du panneau ; `PONT_AIRREC=0` le remet). Les memes chaines se lancent par leur nom : `qosmo-dsp`, `qosmo-grgsm`, et exe par exe `c54x_exe`, `osmocon`, `grgsm_exe` (`/usr/local/bin`). Le detail processus par processus, avec les lignes de journal attendues, est dans `LAUNCH.md`.

Journaux et pid dans `/tmp/c54x-pont/`. [2026-09-23] Le mobile de ce montage utilise maintenant les memes sockets que le run sans `--dsp` : `/tmp/osmocom_l2` et `/tmp/osmocom_sap` (plus de variantes `_pont`) ; VTY mobile 4347 et moniteur QEMU `/tmp/qemu-monitor-pont.sock` inchanges. Lancer un banc grgsm et celui-ci en meme temps les fait se disputer : compromis assume, cf. l'en-tete de `mobile_pont.cfg`. `--stop` efface aussi `/tmp/osmocom_sap`, `/tmp/ms_data`, `/dev/shm/calypso_horloge` et, en `MODE=dsp`, `/dev/shm/calypso_tch_cfg`.

Autres variables de `run.sh` (2026-09-23) :

- `INSNS=80000` : 60000 debordait en TCH (jusqu'a 87000 insn/trame).
- `GDB=1` : gdbstub QEMU en `tcp:127.0.0.1:1234` et console `telnet 0 44444` (`qosmo-dsp/tools/gdb-telnet.py`, journal `gdb.log`) ; `GDB=0` coupe les deux.
- `ASSEMBLY_LOGS=1` : trace asm de l'ARM dans `qemu-asm.log` (`ASSEMBLY_LOGS_FLAGS`, defaut `in_asm,exec,nochain` ; `ASSEMBLY_LOGS_FILTRE` pour `-dfilter`).
- poses d'office : `CALYPSO_PONT_RETRY_DIV=64` sur QEMU (relance vers le DSP toutes les trame/64 au lieu de trame/16), `CALYPSO_BSP_ATTENTE_MS=40` sur `c54x_exe` (attendre une trame livree en retard par la BTS plutot que la jouer en effacement), `L23_SYNC_RETRIES_SELECTION=8` sur le mobile en `MODE=dsp` (defaut du binaire, 1, ailleurs).
- `PANNEAU_LOGS` : `qemu.log`, `osmocon.log` et `mobile.log` s'ecrivent dans `/run/user/0/osmo-nitb/logs/`, `pont.log` dans `/dev/shm/pont.log`, ceux que suit le panneau ; `$RUNDIR` n'en a que des liens (`dsp.log` y reste un vrai fichier). `PANNEAU_LOGS=none` pour ne rien toucher.
- `JOURNAUX_GARDES=10` : au lancement et a `--stop`, la session precedente est rangee dans `$RUNDIR/archives/<date>/`, avec `bsp_dedie.txt`.

2.8.1.1.1 Le lien montant (RACH, SDCCH, SACCH, parole)

En montage `dsp`, la couche 1 gr-gsm de QEMU est desactivee (`CALYPSO_DSP_EXTERN=1`), donc les hooks qui publiaient le montant cote QEMU ne tirent plus. C'est `src/montant.c` qui s'en charge : une scrutation de l'API RAM partagee a chaque trame, qui alimente les memes side-bands que consomme `pont.py` (`pont/uplink.py`) :

```
/dev/shm/calypso_rach          RACH (ra, bsic) lu dans NDB d_rach
/dev/shm/calypso_sdcch_ul      bloc L2 montant (a_cu)
/dev/shm/calypso_tch_facch_ul  FACCH montante
/dev/shm/calypso_tch_sacch_ul  SACCH montante
/dev/shm/calypso_tch_ul        anneau de trames de parole
```

Sans lui : `pont.log` affiche `UL bursts=0 rach=0`, la BTS ne voit aucun acces aleatoire, le mobile epuise ses huit tentatives et il n'y a jamais de LOCATION UPDATING ACCEPT. Reglages : `MONTANT=0` coupe la publication, `MONTANT_DEBUG=N` regle le nombre d'evenements imprimes (20 par defaut), `MONTANT_RACH_SUR_DRACH=1` revient a l'ancien declencheur (transition de `d_rach` au lieu de `d_task_ra`).

[2026-10-03] **Le codage canal SDCCH/SACCH montant est celui de la ROM.** La ROM code le bloc posé dans `a_cu` (parité FIRE en PROM0 `0x9013`, convolutif, entrelacement dans `0x4280`) et, à chaque trame d'émission, copie le burst courant dans `data[0x3f8a..0x3f91]` (114 bits + hl/hu ; en mode chiffré, le flux A5 de `0x3f9b..` y est XORé). `src/pont.c` (`tx_rom_publier`) suit le pointeur `data[0x3d9b]` et publie les 4 bursts d'un bloc, avec le flux de clé, dans `/dev/shm/calypso_xcch_ul_rom`. Le pont (`pont/dsp/uplink.py`) les redécode, défait le XOR de la ROM (elle chiffre à l'heure du DSP) et les émet rechiffrés au fn de l'air ; sans bursts ROM au bout de 120 ms, il code le bloc lui-même. Banc du 2026-10-03 17:11 : `xCCH ul rom=134 hote=1`, échelle 25/25. Il a fallu corriger quatre bugs du cœur C54x (qosmo `c54x_exec.c`, voir README anglais). Restent sur l'hôte : RACH, FACCH, SACCH en TCH, parole et A5 montants. `MONTANT_ROM_UL=0` / `PONT_UL_ROM=0` reviennent au codage hôte. Le barreau couverture du banc imprime, traitement par traitement, le chemin ROM ou hôte du run.

[2026-09-23] Aussi dans `montant.c` : - le Kc, publie dans `/dev/shm/calypso_kc_l1` (`MONTANT_KC=0` coupe) ; - la parole montante, au format TI (`io-tch-format ti`), est convertie en FR TS 101 318 avant publication (`MONTANT_PAROLE_TI=0` = passage brut ; c'etait l'ancien comportement, qui donnait un echo sature) ; - la bascule SDCCH <-> TCH du BSP suit la tache posee par le firmware dans la page W (`d_task_d` = TCHT 13, TCHA 14 ou TCHD 28 ; retour sur ALLC 24) ; `calypso_tch_cfg` ecrit par le pont n'est plus qu'une annonce (`MONTANT_TCH_TACHE=0` = armement a l'annonce) ; - sondes `[a_fd]`, `[a_dd]` (TCH seulement ; `MONTANT_AFD=0`, `MONTANT_ADD=0`), `[a5-arm]` et `[d_fn]`.

Cote QEMU, `MONTANT_REQREF=0` coupe la correction de la reference de requete des IMMEDIATE ASSIGNMENT (`calypso_trx.c`) : sans elle le mobile jette l'assignation, parce que pont.py emet l'access-burst sur sa propre horloge et que `gsm48_match_ra()` exige une correspondance exacte du numero de trame. Causes et mesures : `MAILBOX.md`, sections « Pas de LU ACCEPT » et « Le RACH passe, l'IMM ASS revient ».

2.8.1.1.2 1. Le DSP : `c54x_exe --arm`

```
cd /opt/GSM/c54x_exe && ./c54x_exe --arm -v
```

Cree `/dev/shm/calypso_api_ram` (la fenetre API, 16 Ko, qui EST `data[0x0800..]` du C54x) et `/tmp/calypso_dsp.sock`, charge les 7 sections de ROM, `c54x_reset()`, puis attend l'ARM. Attendu :

```
pont : en attente de l'ARM sur /tmp/calypso_dsp.sock (API RAM : /dev/shm/calypso_api_ram)
```

`--insns` : budget d'instructions par trame TDMA. [2026-09-23] Defaut 200000 avec `--arm` (2300 sans), plancher 32000 en `--rejouer` ; `run.sh` passe 80000. Depuis le 2026-09-23 les sondes pures du coeur sont coupees par defaut (`CALYPSO_SONDES=1`, `CALYPSO_DEBUG` ou `-vvvv` pour les rallumer), pour accelerer le coeur (gain a remesurer). En pas-a-pas (`LOCKSTEP=1`, defaut de `run.sh`) QEMU attend le DSP au lieu de lui sauter des ticks ; le DSP rend `PONT_DONE` des le burst depose et finit la trame en parallele de l'ARM (`PONT_DONE_TOT=1` par defaut, `0` = ancien ordre). La ligne `[chrono]` de `dsp.log`, toutes les 1000 trames, dit ou passe le temps (`qemu | A | go | B | apres DONE`, en ms par trame).

2.8.1.1.3 2. L'ARM : QEMU `qosmo` avec `CALYPSO_DSP_EXTERN=1`

```
cd /opt/GSM/qosmo && CALYPSO_DSP_EXTERN=1 build/qemu-system-arm -M calypso -cpu arm946 \
  -display none -parallel none -serial pty -serial pty \
  -monitor unix:/tmp/qemu-monitor-pont.sock,server,nowait \
  -kernel /opt/GSM/firmware/board/compal_e88/layer1.highram.elf
```

Sans `-kernel`, le CPU part a 0 et plante a `0x840000` : l'ELF est obligatoire, le romload d'osmocon ne charge rien (le stub UART ne fait qu'acquitter). Attendu sur stderr :

```
char device redirected to /dev/pts/N (label serial0)        <- le pty modem, pour osmocon
[trx] pont DSP : API RAM partagee (/calypso_api_ram) + socket /tmp/calypso_dsp.sock - ...
calypso: couche 1 « grgsm » desactivee (DSP externe, CALYPSO_DSP_EXTERN)
[trx] pont DSP : timer de boot lance (echange DSP seul, sans IRQ TPU, ...)
[trx] pont DSP : RESET_DSP relache par le firmware -> PONT_RESET (fn=0)
[trx] pont DSP : le TDMA du firmware prend le relais du timer de boot (N trames de boot)
```

Et cote `c54x_exe` :

```
pont : ARM connecte, API RAM 32768 mots, 200000 insn/trame
pont : RESET #1 (DL_STATUS=0x0000) fn=0 pc=0xff80
pont : DSP boote (premier IDLE) fn=0 insn=5701
  fn=1251  page=0 insn=428  IDLE  d_fb_det=0  a_sch=0000 ...
```

Le firmware a asserte puis relache `RESET_DSP` (registre CNTL_RST), la ROM est repartie de `0xff80`, a pose IDLE, s'est parquee ; l'ARM a envoye `COPY_BLOCK` vers `0x7000` ; la ROM a saute dans sa L1 et ecrit sa version. Verifier depuis le moniteur QEMU :

```
printf 'xp /96bx 0x008305f0\n' | nc -U /tmp/qemu-monitor-pont.sock    # printf_buffer du firmware :
                                                                      # "DSP API Version: 0x4e2a 0x491a"
od -An -tx2 -j 0x01B4 -N 2 /dev/shm/calypso_api_ram                   # 3606 = version ecrite par la L1 DSP
```

2.8.1.1.4 3. osmocon : le relais L1CTL

```
/opt/GSM/osmocom-bb/src/host/osmocon/osmocon -m romload -i 100 -p /dev/pts/N \
  -s /tmp/osmocom_l2 /opt/GSM/firmware/board/compal_e88/layer1.highram.bin
```

`/dev/pts/N` est le `serial0` de l'etape 2. Attendu : `Received ident ack`, `Progress: 100%`, puis **`Received branch ack, your code is running now!`**. A partir de la, osmocon relaie le L1CTL entre le firmware et `/tmp/osmocom_l2`. Si le telechargement reste a `starting download` sans progres : un osmocon precedent a ete tue a mi-bloc et le stub romload de l'UART attend la fin de ce bloc - relancer QEMU (etape 2) puis osmocon.

2.8.1.1.5 4. Le mobile

```
mobile -c /opt/GSM/c54x_exe/mobile_pont.cfg        # copie de ~/.osmocom/bb/mobile.cfg :
                                                   # layer2-socket /tmp/osmocom_l2, vty 4347
```

Attendu dans les 5 s : `L1CTL_PM_REQ`, `L1CTL_RESET_REQ: FULL!`, `L1CTL_FBSB_REQ (arfcn=514 ...)` cote osmocon ; cote `c54x_exe`, le passage de 428 a ~570 insn par trame et `a_sch=0100 ...` : la L1 ARM a programme `d_task_md=5` (recherche FB) et le DSP l'execute. Cote mobile, `FBSB RESP: result=255` en boucle **sans pont** : `calypso_bsp.c` attend les bursts descendants en UDP sur le port 6702 et personne ne les envoie. Avec la BTS et `PONT=1`, c'est `pont_dsp.py` qui les envoie.

```
telnet 127.0.0.1 4347            # VTY du mobile ; « show ms »
od -An -tx2 -N 16 /dev/shm/calypso_api_ram     # page d'ecriture : d_task_md au 5e mot
```

2.8.1.1.6 Le pont, en une page

Protocole dans `qosmo/include/hw/arm/calypso/calypso_dsp_pont.h`, cote QEMU dans `qosmo/hw/arm/calypso/calypso_trx.c` (`pont_*`), `calypso_soc.c` (CNTL_RST -> RESET_DSP) et `calypso_l1_dispatch.c` (`calypso_l1_disable`) ; cote DSP dans `src/pont.c`. Il recopie section par section le `tdma_tick` de qosmo-dsp : DMA tick, boot jusqu'au premier IDLE, IRQ TPU-frame si l'IMR l'arme, un budget de `c54x_run`, front occupe -> IDLE = IRQ API cote ARM.

Quatre pieges rencontres, tous dans le pont et non dans le DSP : - **la boucle principale de QEMU ne doit pas attendre le DSP** : une trame de 64000 insn dure ~26 ms, l'ARM (qui a besoin du verrou global a chaque acces MMIO) etait affame et restait dans `hwtimer_config`. Le pont est en pipeline : DONE(N) releve au tick N+1, tick saute si le DSP est en retard. - **le DSP doit tourner avant que le firmware n'active le TPU** : un timer de boot cadence l'echange seul, sans IRQ TPU-frame (l'ARM n'a pas encore ses vecteurs), jusqu'a ce que le vrai tick prenne le relais. - **le segment partage doit ETRE `data[0x0800..]` du C54x**, pas une copie : le coeur ecrit sa fenetre API dans `data[]` et ne recopie `api_ram` que sur certains chemins. `pont_allouer_dsp()` aligne `data[]` sur une page et y pose le segment en `MAP_FIXED` ; `api_ram` en est l'alias. - [2026-09-23] **sur le TCH, jouer la phase A jusqu'a l'IDLE (budget/2 au plus) avant de deposer le burst** : pour une tache TCHA (SACCH/TF), la ROM demodule au debut de N+1 le burst SACCH de N qu'elle a laisse en `0x0cce`. Deposer des l'armement de la fenetre l'ecrasait (a_cd FIRE KO a chaque bloc, LOS). `PONT_TCH_DEPOT_IDLE=0` retablit l'ancien depot ; trace `[depot_tch]`.

Et une chose a savoir sur `qosmo` : `fw_console.c` lit `printf_buffer` a une adresse codee en dur (`0x831018`) qui n'est pas celle de cet ELF (`0x8305f0`, `nm layer1.highram.elf | grep printf_buffer`), donc pas de `[fw-console]`.

2.8.1.1.7 Outils de rejeu hors banc

[2026-09-23] Sur tout canal dedie, `c54x_exe` enregistre les ecritures de l'ARM dans l'API RAM, les TICK et les livraisons d'I/Q du BSP dans `/dev/shm/calypso_rejeu_tch.bin` (`CALYPSO_REJEU_ENREG=0` coupe, 60000 livraisons au plus).

```
tools/rejeu_banc [fichier] [ticks]              # rejoue hors banc, imprime chaque a_cd / a_fd
REJEU_SANS_D=1 tools/rejeu_banc                 # garde l'etat du boot local au lieu de 'D'
tools/sacch_tf_decode [fichier] [TN=2] [Kc]     # decode hors DSP la SACCH/TF jouee par le BSP
make isa_test && ./isa_test tools/isa_tests.txt  # conformite ISA (exemples SPRU172C)
```

`sacch_tf_decode` lit `/dev/shm/calypso_sacch_tf.bin` (sonde `[sacch_tf]` du BSP) avec le Kc de `calypso_kc_l1` : si la SACCH decode la et pas dans la ROM, le defaut est apres le BSP ; sinon, dans ce que le BSP recoit. Aucune cible Makefile pour ces deux outils : les binaires sont commites a cote des sources.

2.8.1.2 Pourquoi c'est possible

Mesure faite sur les objets compilés **avant** d'écrire une ligne : sur les 26 631 lignes de couche 1 C54x, l'accroche à QEMU tient en **14 symboles**.

fichier	symboles QEMU
`calypso_c54x.c` (21 296 l.)	**2** — des mutex
`calypso_{arm2dsp,dma,fbsb,mailbox,rhea_dma,rif,twl3025}.c`	**0**
`calypso_full_pcb.c`, `calypso_tint0.c`	8 — le câblage, pas le DSP

Les cales sont dans `qosmo/contrib/hors-qemu/` : ~120 lignes de doublures d'en-têtes (`qemu/osdep.h`, `thread.h`, `timer.h`…) et 73 lignes d'équivalents POSIX. Rien n'y modélise QEMU — le jour où une cale doit devenir autre chose qu'un `pthread`, c'est que le DSP s'est mis à dépendre de QEMU, et il faut le savoir.

Les sources ne sont pas recopiées. Ce binaire compile celles de `/opt/GSM/qosmo` (`QOSMO=... make` pour pointer ailleurs). [2026-09-23] `make` reconstruit `c54x_exe` a chaque appel (cible `.PHONY`, qui depend aussi des entetes), en `-O3 -march=native` : plus besoin de `make clean`. `calypso_a5.c` (le coprocesseur A5 sur les ports XIO `0x2800..0x2818`, `CALYPSO_A5=0` le coupe) est compile dedans. Recopier, c'était refaire la divergence que `qosmo` vient de supprimer. La seule copie est `src/pcb-minimal.c`, quatre helpers DARAM repris mot pour mot, et elle est signalée comme telle dans le fichier.

2.8.1.3 Ce que ça mesure, et ce que ça ne mesure pas encore

État au 2026-10-03 (bancs DSP de 16:49, 17:04 et 17:11, échelle camp → voix 25/25). Chemin vérifié dans le code et les journaux : **tout le décodage descendant est fait par la ROM** (`a_cd`, `a_fd`, `a_dd` lus tels quels par le firmware ; en dsp le décodage du pont ne sert qu'à ses STATS, `pont/dsp/downlink.py`), la seule retouche hôte est la référence de requête des IMMEDIATE ASSIGNMENT (qosmo `calypso_trx.c`). Le montant SDCCH/SACCH est codé par la ROM (ci-dessus). Mises à jour des points ouverts ci-dessous : - **B_BFI** : toujours posé sur 100 % de la parole (`bfi=2600/2600`), mais `err` vaut désormais **0 partout** (46 lignes du run, 7 archives du jour) — les « 15 à 93 » du 2026-09-23 sont périmés. Dès le premier bloc après la bascule (incomplet, le TCH/F s'entrelace sur 8 bursts) on lit déjà `BFI=1 err=0` : le BFI ne suit pas la qualité réelle. Parole exacte (+59 dB à 1 kHz au test voix ; bit-exactitude mesurée le 2026-09-30, à re-mesurer avec le cœur corrigé). États `c214`/`c204`, bit 9 (B_ECRC) posé en FR V1 : à suivre ; - SACCH en TCH : aucune LOS, aucune `[garde-3d89]` sur les runs du jour ; - SDCCH/8 descendant : la SACCH/8 échoue au FIRE ~80 % du temps (`[a_cd] ok=113 ko=37`, 95 trames jetées par le mobile) ; - fenêtre SB : `44/3630` fenêtres SB réellement armées ; - marge temps réel : en TCH, A 0.19 + go 0.30 + B 0.07 + après DONE 2.09 ms.

État au 2026-09-23 (runs du banc DSP de 20:22 et 20:32). En montage dsp avec la BTS (`PONT=1`), le DSP détecte FB et SB, décode les BCCH (SI1-4), le mobile obtient le LU ACCEPT, le premier SMS MT est livré de bout en bout (2026-09-23 11:06), l'appel passe l'ASSIGNMENT (2026-09-22 19:47) et la bascule TCH suit la tâche du firmware ; l'A5 est modélisé dans le DSP (`calypso_a5.c`), la parole montante est convertie TI -> FR. Run de 20:22, constaté dans les journaux : LU, appel MO vers l'écho 600 (ACTIVE 20:22:55, DISCONNECT 20:23:27), SMS MO et MT dans les deux sens, appel MT depuis 100102 (ACTIVE 20:24:28, release normal), A5/1 confirmé par la BTS sur les cinq établissements, parole audible dans les deux sens (décodage canal TCH/F descendant par la ROM TI, codec GAPK FR et codage montant sur l'hôte), 29 513 trames avec un seul tick sauté (au boot, fn=0). Restent ouverts, par ordre d'importance : - **B_BFI sur toute la parole** : la ROM marque chaque trame TCH/F comme mauvaise. Run de 20:32, sonde `[a_dd]` étendue (`src/montant.c sonde_add`, non commitée) : `vues=2200 bfi=2200` ; `err` (a_dd_0[2], erreurs rapportées par la ROM) vaut 0 sur 19 des 20 premières trames après la bascule (`c214` ; la 18e, fn=5912, est `8084` à 58), puis 15 à 93. Les trames sont réellement dégradées ; le FR reste intelligible parce que le firmware ne remonte pas le BFI (`prim_tch.c:327` ne teste que B_BLUD). Signal (BSP, IQ, égalisation) ou cœur C54x (Viterbi, recomptage) : à trancher par comparaison bit à bit avec les trames de la BTS. Le `ko` de la sonde (B_FIRE1) ne dit rien sur la parole ; - la SACCH en TCH : le correctif MVKD/MVDK (`qosmo c54x_exec.c`, `CALYPSO_MVKD_DMAD_AVANT=1` = ancien ordre ; garde `[garde-3d89]` dans `c54x_mem.c`) tient au run de 20:22 (deux appels complets, aucun bloc SACCH/TF jeté par le mobile hors bascule et libérations, aucune LOS, aucune ligne `[garde-3d89]`). Mais au run de 20:32 le premier appel tombe en LOS (20:32:45) : SACCH/TF FIRE KO à chaque bloc dès fn=6095, `err` de la parole 63 à 93 de fn=6273 à 9306 contre 15 à 38 sur le troisième appel, sain ; le deuxième reste bloqué en attente de la connexion MM (T3230). Garde muette : autre cause, non localisée ; - le SDCCH/8 descendant : au run de 20:22 le mobile jette 27 trames sur SDCCH/8 (4 à 7 par session dédiée), dont 15 SACCH (ligne « LOSS counter for ACCH ») et 12 du canal principal ; suspect, le BSP sur le SDCCH/8 (table 45.002) ; - la fenêtre SB, rarement armée par la ROM (d'où `L23_SYNC_RETRIES_SELECTION=8`) : une synchro sur trois à cinq ; - la marge temps réel : en TCH, `[chrono]` donne A 0.33 + go 0.40 + B 0.16 + après DONE 3.37-3.68 ms, soit 4.3 à 4.6 ms de travail DSP pour 4.62 ms.

Ne sont pas des anomalies : les échecs CRC du moniteur TCH du pont tant que le RTP ne coule pas (décodage du pont, indépendant du DSP), et l'UA / SABM répétés, disparus au run de 20:22 avec `osmo-operator pont/dsp/clock.py` (`PONT_AVANCE_MIN=10`) : aucune ligne SABM dans les journaux osmocom (ni ERROR INDICATION au BSC), marge DL réelle min +0 au premier relevé (20:22:47), +9 ensuite, +13 à +15 à partir de 20:23:17.

Le détail, jour par jour : `MAILBOX.md`.

Le cas particulier du mode autonome, sans ARM (`./c54x_exe --trames N`) : les 7 sections de ROM se chargent, `c54x_reset()` passe, et la mask-ROM **exécute** — les `BRANCH-TRACE` à `PC=0xb41f` sont du vrai code TI. **Aucun burst n'est injecté** : le DSP tourne sur une API RAM vierge, sans ARM ni TPU. Le binaire le dit lui-même dans son bilan. `a_sch[3]` y sort `0x771a` et non le `0xf8d8` du README de qemu-calypso — les deux valeurs sont constantes, mais elles diffèrent parce que l'entrée diffère, ce qui est déjà une information.

L'intérêt visé est la question du README : *« un décodeur dont la sortie ne dépend pas de l'entrée ne décode pas »*. Y répondre demande d'injecter des bursts et de faire varier l'entrée — le montage dsp avec la BTS le fait depuis, et le rejeu hors banc (`tools/rejeu_banc`) le rend tenable, parce qu'un essai coûte des millisecondes au lieu d'un boot complet.

2.8.1.4 Dépendances

`libosmocoding` / `libosmocore` (pour `calypso_bsp.c`), `pthread`, `libm`. Les ROM : `calypso_dsp.{PROM0..3,DROM,PDROM,Registers}.` dans `--rom-dir` (défaut `/opt/GSM`).

2.9 /opt/GSM/c54x_exe/docs/carte-memoire.md

32466 octets, 430 lignes

2.9.1 Carte mémoire du DSP Calypso — API RAM, NDB, PARAM, mémoire interne, ports

> **État : PREMIÈRE VERSION, interrompue (limite d'usage, 2026-10-03 ~18:15).** Les tables de la section 2 sont générées (DWARF de l'ELF + mesure du rejeu) ; le reste est écrit à la main à partir de la mesure et des sources citées. Le générateur `tools/layer1_tester.py` est inachevé (voir « Reste à faire ») : cette page n'est pas encore régénérable d'une commande.

2.9.1.1 Comment ce fichier a été produit

- Offsets des champs : `gdb-multiarch -batch -ex 'ptype /o T_NDB_MCU_DSP' /opt/GSM/firmware/board/compal_e88/layer1.` (et `T_DB_MCU_TO_DSP`, `T_DB_DSP_TO_MCU`, `T_PARAM_MCU_DSP`) : c'est la disposition **compilée** (DSP=36, CHIPSET=12, ANLG_FAM=2, `include/calypso/l1_environment.h`). Bases : `print/x dsp_api` dans le même ELF (ndb=0xffd001a8, db_r=0xffd00050, db_w=0xffd00000, param=0xffd00862).
- Mesure : `make layer1_tester && tools/layer1_tester -o mesure.tsv -j journal.tsv -p pcs.tsv rejeu.bin` — rejeu de l'enregistrement de canal dédié (copie de `/dev/shm/calypso_rejeu_tch.bin`, ticks 1712..35633, 18391 ticks, 393 M instructions, 46 s). Fidélité : SACCH 132 bonnes / 33 Fire KO, FACCH 35 / 0, parole 2654 trames toutes B_BFI = mêmes bilans que `tools/rejeu_banc`.
- Conversion : mot DSP = 0x0800 + (adresse ARM − 0xFFD00000)/2 ; « octet API » = adresse ARM − 0xFFD00000.
- « ROM lit / écrit » = accès faits par une instruction de la ROM (crochet `calypso_mbx_evt`) ; « ARM modifie » = mots changés côté QEMU entre deux trames (enregistrements `A`), donc des modifications, pas toutes les écritures ; valeurs = jusqu'à 4 plus fréquentes (valeur:compte). 1er PC en hexadécimal (`x:pppp` si page XPC ≠ 0). Champs `*hole*` non touchés omis.

2.9.1.2 1. Plan de l'espace de données (vu du DSP)

plage	contenu	source
0x0000..0x001f	MMR du C54x (IMR, IFR, ST0/1, AR0..7, SP, BK, BRC, PMST, XPC…)	qosmo `l1-dsp/calypso_c54x.h:28-54`
0x0020..0x005f	périphériques (timer 0x24..0x26, DMA 0x54..0x57)	`calypso_c54x.h`, `calypso_dma.h:45-48`
0x0800..0x0813	page W0 (`T_DB_MCU_TO_DSP`, ARM 0xFFD00000)	ELF
0x0814..0x0827	page W1 (ARM 0xFFD00028)	ELF
0x0828..0x083b	page R0 (`T_DB_DSP_TO_MCU`, ARM 0xFFD00050)	ELF
0x083c..0x084f	page R1 (ARM 0xFFD00078)	ELF
0x08d4..0x153d	NDB (`T_NDB_MCU_DSP`, ARM 0xFFD001A8, 6356 octets)	ELF
0x0c31..0x0cc1	PARAM (`T_PARAM_MCU_DSP`, ARM 0xFFD00862) — **dans** NDB.a_sr_holes1 (290 octets = les 145 mots utiles de PARAM)	ELF
0x0cce..0x0df5	tampon DMA des I/Q reçus (AAD 0x99c) — dans NDB.a_cport_holes	`c54x_mem.c` (FEED-DST) ; mesure : 3219 écritures hors instruction (DMA/BSP)

plage	contenu	source
0x0ffc..0x0fff	mots du chargeur BL_ADDR_HI / BL_SIZE / BL_ADDR_LO / BL_CMD_STATUS (ARM 0xFFD00FF8..FFE) — dans NDB.a_cport_holes	`firmware/calypso/dsp.c:54-57`
0x153e..0x27ff	API RAM hors structures layer1	—
0x2800	mémoire interne du DSP (section 4)	—

2.9.1.3 2. API RAM — tous les champs (générée)

zone	champ	octet API	ARM	mot DSP	ROM lit (n, 1er PC)	ROM écrit (n, 1er PC)	ARM modifie (n, valeurs)
W0	d_task_d	0x000	0xFFD00000	0x0800	25920, b001	0	1296, 000d:445,000e:361,0000:2
W0	d_burst_d	0x002	0xFFD00002	0x0801	14950, b005	361, ba17	777, 0000:475,0002:102,0001:1
W0	d_task_u	0x004	0xFFD00004	0x0802	13229, b009	0	1298, 000d:445,000e:361,0000:2
W0	d_burst_u	0x006	0xFFD00006	0x0803	8345, b00d	361, ba19	780, 0000:475,0002:103,0001:1
W0	d_task_md	0x008	0xFFD00008	0x0804	19843, b011	0	300, 0001:150,0000:150
W0	d_background	0x00a	0xFFD0000A	0x0805	6614, b0db	0	0
W0	d_debug	0x00c	0xFFD0000C	0x0806	13229, b015	0	0
W0	d_task_ra	0x00e	0xFFD0000E	0x0807	6614, b097	0	0
W0	d_fn	0x010	0xFFD00010	0x0808	13613, b548	0	5760, ,+
W0	d_ctrl_tch	0x012	0xFFD00012	0x0809	52823, b54b	0	518, 0000:255,0030:169,0011:8
W0	hole	0x014	0xFFD00014	0x080a	0	0	0
W0	d_ctrl_abb	0x016	0xFFD00016	0x080b	6615, b5a5	0	576, 0019:288,0010:288
W0	a_a5fn[2]	0x018	0xFFD00018	0x080c..0x080d	13228, b044	0	7247, ,+
W0	d_power_ctl	0x01c	0xFFD0001C	0x080e	6615, b5a9	0	587, 0000:276,2e92:111,3212:7
W0	d_afc	0x01e	0xFFD0001E	0x080f	6615, b5ad	0	1, fd44:1
W0	d_ctrl_system	0x020	0xFFD00020	0x0810	19856, a53c	13, a549	522, 0007:255,0000:254,8000:1
W1	d_task_d	0x028	0xFFD00028	0x0814	25607, b001	0	1293, 0000:567,000d:443,0018:2
W1	d_burst_d	0x02a	0xFFD0002A	0x0815	14297, b005	81, ba17	571, 0000:267,0001:102,0003:1
W1	d_task_u	0x02c	0xFFD0002C	0x0816	13206, b009	0	1294, 0000:566,000d:443,000c:2
W1	d_burst_u	0x02e	0xFFD0002E	0x0817	7703, b00d	81, ba19	571, 0000:264,0003:103,0001:1
W1	d_task_md	0x030	0xFFD00030	0x0818	19845, b011	0	866, 0001:432,0000:427,0005:7
W1	d_background	0x032	0xFFD00032	0x0819	6603, b0db	0	0
W1	d_debug	0x034	0xFFD00034	0x081a	13206, b015	0	0
W1	d_task_ra	0x036	0xFFD00036	0x081b	6603, b097	0	0

zone	champ	octet API	ARM	mot DSP	ROM lit (n, 1er PC)	ROM écrit (n, 1er PC)	ARM modifie (n, valeurs)
W1	d_fn	0x038	0xFFD00038	0x081c	13325, b548	0	5756, 0000:1,+
W1	d_ctrl_tch	0x03a	0xFFD0003A	0x081d	51969, b54b	0	1061, 0000:531,0011:363,0030:1
W1	hole	0x03c	0xFFD0003C	0x081e	0	0	0
W1	d_ctrl_abb	0x03e	0xFFD0003E	0x081f	6603, b5a5	0	1132, 0019:566,0010:566
W1	a_a5fn[2]	0x040	0xFFD00040	0x0820..0x0821	13206, b044	0	7791, 0095:1,0000:1,+
W1	d_power_ctl	0x044	0xFFD00044	0x0822	6603, b5a9	0	1141, 0000:501,2392:255,2e92:4
W1	d_afc	0x046	0xFFD00046	0x0823	6603, b5ad	0	1, fd44:1
W1	d_ctrl_system	0x048	0xFFD00048	0x0824	19809, a53c	0	1057, 0000:531,0007:526
R0	d_task_d	0x050	0xFFD00050	0x0828	0	6615, b003	6076, 0000:6076
R0	d_burst_d	0x052	0xFFD00052	0x0829	0	6615, b007	302, 0000:302
R0	d_task_u	0x054	0xFFD00054	0x082a	0	11838, b00b	6080, 0000:6080
R0	d_burst_u	0x056	0xFFD00056	0x082b	0	6615, b00f	305, 0000:305
R0	d_task_md	0x058	0xFFD00058	0x082c	0	6615, b013	150, 0000:150
R0	d_background	0x05a	0xFFD0005A	0x082d	0	0	0
R0	d_debug	0x05c	0xFFD0005C	0x082e	0	6615, b017	0
R0	d_task_ra	0x05e	0xFFD0005E	0x082f	0	0	0
R0	a_serv_demod[4]	0x060	0xFFD00060	0x0830..0x0833	0	24288, b267	24225, 0000:6076
R0	a_pm[3]	0x068	0xFFD00068	0x0834..0x0836	0	150, b34b	150, 0000:150
R0	a_sch[5]	0x06e	0xFFD0006E	0x0837..0x083b	0	0	0
R1	d_task_d	0x078	0xFFD00078	0x083c	0	6603, b003	5796, 0000:5796
R1	d_burst_d	0x07a	0xFFD0007A	0x083d	0	6603, b007	304, 0000:304
R1	d_task_u	0x07c	0xFFD0007C	0x083e	0	11823, b00b	5800, 0000:5800
R1	d_burst_u	0x07e	0xFFD0007E	0x083f	0	6603, b00f	307, 0000:307
R1	d_task_md	0x080	0xFFD00080	0x0840	0	6603, b013	427, 0000:427
R1	d_background	0x082	0xFFD00082	0x0841	0	0	0
R1	d_debug	0x084	0xFFD00084	0x0842	0	6603, b017	0
R1	d_task_ra	0x086	0xFFD00086	0x0843	0	0	0
R1	a_serv_demod[4]	0x088	0xFFD00088	0x0844..0x0847	0	23212, b267	23120, 0000:5796
R1	a_pm[3]	0x090	0xFFD00090	0x0848..0x084a	0	432, b34b	427, 0000:427
R1	a_sch[5]	0x096	0xFFD00096	0x084b..0x084f	0	0	0
NDB	d_dsp_page	0x1a8	0xFFD001A8	0x08d4	13218, a51c	0	13234, 0002:6614,0003:6610,000
NDB	d_error_status	0x1aa	0xFFD001AA	0x08d5	0	13218, b10a	0
NDB	d_spcx_rif	0x1ac	0xFFD001AC	0x08d6	13218, b56c	0	0

zone	champ	octet API	ARM	mot DSP	ROM lit (n, 1er PC)	ROM écrit (n, 1er PC)	ARM modifie (n, valeurs)
NDB	d_tch_mode	0x1ae	0xFFD001AE	0x08d7	14711, 85e4	0	13, ca00:7,ca08:6
NDB	d_debug1	0x1b0	0xFFD001B0	0x08d8	11838, b585	0	0
NDB	d_dsp_test	0x1b2	0xFFD001B2	0x08d9	10616, a90a	0	0
NDB	d_version_number1	0x1b4	0xFFD001B4	0x08da	0	0	0
NDB	d_version_number2	0x1b6	0xFFD001B6	0x08db	0	0	0
NDB	d_debug_ptr	0x1b8	0xFFD001B8	0x08dc	81581, b524	81581, b530	0
NDB	d_debug_bk	0x1ba	0xFFD001BA	0x08dd	81581, b528	0	0
NDB	d_pll_config	0x1bc	0xFFD001BC	0x08de	34471, a687	0	0
NDB	p_debug_buffer	0x1be	0xFFD001BE	0x08df	0	0	0
NDB	d_debug_buffer_size	0x1c0	0xFFD001C0	0x08e0	0	0	0
NDB	d_debug_trace_type	0x1c2	0xFFD001C2	0x08e1	0	0	0
NDB	d_dsp_state	0x1c4	0xFFD001C4	0x08e2	0	0	0
NDB	d_mcsi_select	0x1ce	0xFFD001CE	0x08e7	0	0	0
NDB	d_apcdel1_bis	0x1d0	0xFFD001D0	0x08e8	0	0	0
NDB	d_apcdel2_bis	0x1d2	0xFFD001D2	0x08e9	0	0	0
NDB	d_apcdel2	0x1d4	0xFFD001D4	0x08ea	0	0	0
NDB	d_vbctrl2	0x1d6	0xFFD001D6	0x08eb	0	0	0
NDB	d_bulgcal	0x1d8	0xFFD001D8	0x08ec	0	0	0
NDB	d_afcctladd	0x1da	0xFFD001DA	0x08ed	13218, b5ba	0	0
NDB	d_vbuctrl	0x1dc	0xFFD001DC	0x08ee	13226, b5c4	0	0
NDB	d_vbdctrl	0x1de	0xFFD001DE	0x08ef	13218, b5c9	0	0
NDB	d_apcdel1	0x1e0	0xFFD001E0	0x08f0	11885, b660	0	0
NDB	d_apcoff	0x1e2	0xFFD001E2	0x08f1	36990, b5bf	11886, b67e	11886, 1817:11886
NDB	d_bulioff	0x1e4	0xFFD001E4	0x08f2	13218, b5d3	0	0
NDB	d_bulqoff	0x1e6	0xFFD001E6	0x08f3	13218, b5d8	0	0
NDB	d_dai_onoff	0x1e8	0xFFD001E8	0x08f4	13218, b5ed	0	0
NDB	d_auxdac	0x1ea	0xFFD001EA	0x08f5	13218, b5dd	0	0
NDB	d_vbctrl1	0x1ec	0xFFD001EC	0x08f6	13230, b5e2	4, b6a8	0
NDB	d_bbctrl	0x1ee	0xFFD001EE	0x08f7	13218, b5ce	0	0
NDB	d_fb_det	0x1f0	0xFFD001F0	0x08f8	6397, 778a	6397, b2cc	0
NDB	d_fb_mode	0x1f2	0xFFD001F2	0x08f9	18874, 781f	0	0
NDB	a_sync_demod[4]	0x1f4	0xFFD001F4	0x08fa..0x08fd	0	25468, b2cf	0
NDB	a_sch26[5]	0x1fc	0xFFD001FC	0x08fe..0x0902	0	0	0
NDB	d_audio_gain2	0x206	0xFFD00206	0x0903	0	0	0
NDB	d_audio_gain1	0x208	0xFFD00208	0x0904	0	0	0
NDB	d_audio_compressor_ctrl	0x20a	0xFFD0020A	0x0905	0	0	0
NDB	d_audio_init	0x20c	0xFFD0020C	0x0906	13218, c1fe	0	0
NDB	d_audio_status	0x20e	0xFFD0020E	0x0907	13218, c1ff	0	0
NDB	d_toneskb_int	0x210	0xFFD00210	0x0908	5306, 7382	0	0

zone	champ	octet API	ARM	mot DSP	ROM lit (n, 1er PC)	ROM écrit (n, 1er PC)	ARM modifie (n, valeurs)
NDB	d_toneskb_state	0x212	0xFFD00212	0x0909	23882, 7365	5308, 73af	0
NDB	d_k_x1_t0	0x214	0xFFD00214	0x090a	0	0	0
NDB	d_k_x1_t1	0x216	0xFFD00216	0x090b	0	0	0
NDB	d_k_x1_t2	0x218	0xFFD00218	0x090c	0	0	0
NDB	d_pe_rep	0x21a	0xFFD0021A	0x090d	0	0	0
NDB	d_pe_off	0x21c	0xFFD0021C	0x090e	0	0	0
NDB	d_se_off	0x21e	0xFFD0021E	0x090f	0	0	0
NDB	d_bu_off	0x220	0xFFD00220	0x0910	0	0	0
NDB	d_t0_on	0x222	0xFFD00222	0x0911	0	0	0
NDB	d_t0_off	0x224	0xFFD00224	0x0912	0	0	0
NDB	d_t1_on	0x226	0xFFD00226	0x0913	0	0	0
NDB	d_t1_off	0x228	0xFFD00228	0x0914	0	0	0
NDB	d_t2_on	0x22a	0xFFD0022A	0x0915	0	0	0
NDB	d_t2_off	0x22c	0xFFD0022C	0x0916	0	0	0
NDB	d_k_x1_kt0	0x22e	0xFFD0022E	0x0917	0	0	0
NDB	d_k_x1_kt1	0x230	0xFFD00230	0x0918	0	0	0
NDB	d_dur_kb	0x232	0xFFD00232	0x0919	0	0	0
NDB	d_shiftdl	0x234	0xFFD00234	0x091a	0	0	0
NDB	d_shiftul	0x236	0xFFD00236	0x091b	0	0	0
NDB	d_aec_ctrl	0x238	0xFFD00238	0x091c	10625, bf28	0	0
NDB	d_es_level_api	0x23a	0xFFD0023A	0x091d	0	0	0
NDB	d_mu_api	0x23c	0xFFD0023C	0x091e	0	0	0
NDB	d_melo_osc_use	0x23e	0xFFD0023E	0x091f	0	0	0
NDB	d_melo_osc_addr	0x240	0xFFD00240	0x0920	0	0	0
NDB	a_melo_note0[4]	0x242	0xFFD00242	0x0921..0x0924	0	0	0
NDB	a_melo_note1[4]	0x24a	0xFFD0024A	0x0925..0x0928	0	0	0
NDB	a_melo_note2[4]	0x252	0xFFD00252	0x0929..0x092c	0	0	0
NDB	a_melo_note3[4]	0x25a	0xFFD0025A	0x092d..0x0930	0	0	0
NDB	a_melo_note4[4]	0x262	0xFFD00262	0x0931..0x0934	0	0	0
NDB	a_melo_note5[4]	0x26a	0xFFD0026A	0x0935..0x0938	0	0	0
NDB	a_melo_note6[4]	0x272	0xFFD00272	0x0939..0x093c	0	0	0
NDB	a_melo_note7[4]	0x27a	0xFFD0027A	0x093d..0x0940	0	0	0
NDB	d_melody_select	0x282	0xFFD00282	0x0941	0	0	0
NDB	d_sr_status	0x28a	0xFFD0028A	0x0945	0	0	0
NDB	d_sr_param	0x28c	0xFFD0028C	0x0946	0	0	0
NDB	d_sr_bit_exact_test	0x28e	0xFFD0028E	0x0947	0	0	0
NDB	d_sr_nb_words	0x290	0xFFD00290	0x0948	0	0	0
NDB	d_sr_db_level	0x292	0xFFD00292	0x0949	0	0	0
NDB	d_sr_db_noise	0x294	0xFFD00294	0x094a	0	0	0
NDB	d_sr_mod_size	0x296	0xFFD00296	0x094b	0	0	0
NDB	a_n_best_words[4]	0x298	0xFFD00298	0x094c..0x094f	0	0	0
NDB	a_n_best_score[8]	0x2a0	0xFFD002A0	0x0950..0x0957	0	0	0
NDB	a_dd_1[22]	0x2b0	0xFFD002B0	0x0958..0x096d	0	0	0
NDB	a_du_1[22]	0x2dc	0xFFD002DC	0x096e..0x0983	32416, a925	2656, a925	3815, 8000:2488
NDB	d_v42b_negot	0x308	0xFFD00308	0x0984	0	0	0
NDB	d_v42b_negot	0x30a	0xFFD0030A	0x0985	0	0	0
NDB	d_v42b_contro	0x30c	0xFFD0030C	0x0986	0	0	0
NDB	d_v42b_ratio_	0x30e	0xFFD0030E	0x0987	0	0	0
NDB	d_mcu_contro	0x310	0xFFD00310	0x0988	0	0	0
NDB	d_mcu_contro	0x312	0xFFD00312	0x0989	0	0	0
NDB	d_background_	0x314	0xFFD00314	0x098a	94035, dde8	0	0
NDB	d_background_	0x316	0xFFD00316	0x098b	0	0	0
NDB	d_background_	0x318	0xFFD00318	0x098c	0	59564, dde8	0

zone	champ	octet API	ARM	mot DSP	ROM lit (n, 1er PC)	ROM écrit (n, 1er PC)	ARM modifie (n, valeurs)
NDB	d_max_backgrou	0x31a	0xFFD0031A	0x098d	0	0	0
NDB	a_background_ta	0x31c	0xFFD0031C	0x098e..0x099d	0	0	0
NDB	a_back_task_0[3	0x336	0xFFD0033C	0x099e..0x09ad	0	0	0
NDB	d_gea_mode_0	0x35c	0xFFD0035C	0x09ae	0	0	0
NDB	a_gea_kc_ovly[4	0x35e	0xFFD0035E	0x09af..0x09b2	0	0	0
NDB	d_thr_usf_dem	0x374	0xFFD00374	0x09ba	0	0	0
NDB	d_a5mode	0x376	0xFFD00376	0x09bb	37225, b138	0	15, 0001:8,0000:7
NDB	d_sched_mode_	0x378	0xFFD00378	0x09bc	13, a541	0	0
NDB	a_ramp[16]	0x384	0xFFD00384	0x09c2..0x09d1	190160, b642	0	166, 4fd4:6,8fd4:1,+
NDB	a_cd[15]	0x3a4	0xFFD003A4	0x09d2..0x09e0	4056, 9718	8736, 96dd	220, 0040:110
NDB	a_fd[15]	0x3c2	0xFFD003C2	0x09e1..0x09ef	11180, b10d	3634, a72c	70, 0040:35
NDB	a_dd_0[22]	0x3e0	0xFFD003E0	0x09f0..0x0a05	97541, a38e	65580, 928a	5308, 0000:2654
NDB	a_cu[15]	0x40c	0xFFD0040C	0x0a06..0x0a14	4095, aa5a	315, aa5a	2239, 8000:315
NDB	a_fu[15]	0x42a	0xFFD0042A	0x0a15..0x0a23	8292, a8bd	2656, a928	109, 8000:26
NDB	a_du_0[22]	0x448	0xFFD00448	0x0a24..0x0a39	65781, ad0d	105565, ad90	0
NDB	d_rach	0x474	0xFFD00474	0x0a3a	0	0	0
NDB	a_kc[4]	0x476	0xFFD00476	0x0a3b..0x0a3e	147204, b145	0	8, 8c00:1,f000:1
NDB	d_ra_conf	0x47e	0xFFD0047E	0x0a3f	2629, b124	0	0
NDB	d_ra_act	0x480	0xFFD00480	0x0a40	0	0	0
NDB	d_ra_test	0x482	0xFFD00482	0x0a41	0	0	0
NDB	d_ra_statu	0x484	0xFFD00484	0x0a42	0	0	0
NDB	d_ra_statd	0x486	0xFFD00486	0x0a43	0	0	0
NDB	d_fax	0x488	0xFFD00488	0x0a44	0	0	0
NDB	a_data_buf_ul[2	0x48a	0xFFD0048A	0x0a45..0x0a59	0	0	0
NDB	a_data_buf_dl[2	0x4b4	0xFFD004B4	0x0a5a..0x0a70	0	0	0
NDB	a_sr_holes1[105	0x862	0xFFD00862	0x0c31..0x0cc1	133372, a5e0	0	0
NDB	d_cport_init	0x984	0xFFD00984	0x0cc2	0	0	0
NDB	d_cport_ctrl	0x986	0xFFD00986	0x0cc3	0	0	0
NDB	a_cport_cfr[2]	0x988	0xFFD00988	0x0cc4..0x0cc5	0	0	0
NDB	d_cport_tcl_0	0x98c	0xFFD0098C	0x0cc6	0	0	0
NDB	d_cport_tdat	0x98e	0xFFD0098E	0x0cc7	0	0	0
NDB	d_cport_tvs	0x990	0xFFD00990	0x0cc8	0	0	0
NDB	d_cport_status	0x992	0xFFD00992	0x0cc9	0	0	0
NDB	d_cport_reg_0	0x994	0xFFD00994	0x0cca	0	0	0
NDB	a_cport_holes[1	0x996	0xFFD00996	0x0ccb..0x10bd	14510066, 81ad	1096940, 819b	0
NDB	a_model[1041]	0x117c	0xFFD0117C	0x10be..0x14ce	0	0	0
NDB	a_amr_config[4]	0x19ca	0xFFD019CA	0x14e5..0x14e8	0	0	0
NDB	a_ratscch_ul[0]	0x19d2	0xFFD019D2	0x14e9..0x14ee	0	0	0
NDB	a_ratscch_dl[0]	0x19de	0xFFD019DE	0x14ef..0x14f4	0	0	0
NDB	d_amr_snr_est	0x19ea	0xFFD019EA	0x14f5	0	0	0
NDB	d_thr_onset_ul	0x19ee	0xFFD019EE	0x14f7	0	0	0
NDB	d_thr_sid_first_	0x19f0	0xFFD019F0	0x14f8	0	0	0
NDB	d_thr_ratscch_	0x19f2	0xFFD019F2	0x14f9	0	0	0
NDB	d_thr_update_ul	0x19f4	0xFFD019F4	0x14fa	0	0	0
NDB	d_thr_onset_dl	0x19f6	0xFFD019F6	0x14fb	0	0	0
NDB	d_thr_sid_alt	0x19f8	0xFFD019F8	0x14fc	0	0	0

zone	champ	octet API	ARM	mot DSP	ROM lit (n, 1er PC)	ROM écrit (n, 1er PC)	ARM modifie (n, valeurs)
NDB	d_thr_ratscch_hi	0x19fa	0xFFD019FA	0x14fd	0	0	0
NDB	d_thr_sp_dgf	0x19fc	0xFFD019FC	0x14fe	0	0	0
NDB	d_thr_soft_bts	0x19fe	0xFFD019FE	0x14ff	0	0	0
PARAM	d_transfer_rate	0x862	0xFFD00862	0x0c31	0	0	0
PARAM	d_lat_mcu_hscsd	0x864	0xFFD00864	0x0c32	43808, a5e0	0	0
PARAM	d_lat_mcu_hscsdam	0x866	0xFFD00866	0x0c33	0	0	0
PARAM	d_lat_mcu_burst_access	0x868	0xFFD00868	0x0c34	0	0	0
PARAM	d_lat_dsp_afs_am	0x86a	0xFFD0086A	0x0c35	0	0	0
PARAM	d_gprs_install_address	0x86c	0xFFD0086C	0x0c36	0	0	0
PARAM	d_misc_config	0x86e	0xFFD0086E	0x0c37	34471, a684	0	0
PARAM	d_cn_sw_workaround	0x870	0xFFD00870	0x0c38	0	0	0
PARAM	d_hole2_param	0x872	0xFFD00872	0x0c39..0x0c3c	0	0	0
PARAM	d_fb_margin_beg	0x87a	0xFFD0087A	0x0c3d	29, 794b	0	0
PARAM	d_fb_margin_end	0x87c	0xFFD0087C	0x0c3e	29, 7956	0	0
PARAM	d_nsubb_idle	0x87e	0xFFD0087E	0x0c3f	36, b316	0	0
PARAM	d_nsubb_ded	0x880	0xFFD00880	0x0c40	0	0	0
PARAM	d_fb_thr_det_sacch	0x882	0xFFD00882	0x0c41	6224, 78ef	0	0
PARAM	d_fb_thr_det_stack	0x884	0xFFD00884	0x0c42	0	0	0
PARAM	d_dc_off_thres	0x886	0xFFD00886	0x0c43	11875, 7ea1	0	0
PARAM	d_dummy_thres	0x888	0xFFD00888	0x0c44	0	0	0
PARAM	d_dem_pond_gsm	0x88a	0xFFD0088A	0x0c45	11875, 8154	0	0
PARAM	d_dem_pond_edge	0x88c	0xFFD0088C	0x0c46	11875, 815a	0	0
PARAM	d_maccthresh	0x88e	0xFFD0088E	0x0c47	0	0	0
PARAM	d_mldt	0x890	0xFFD00890	0x0c48	4, e59f	0	0
PARAM	d_maccthresh	0x892	0xFFD00892	0x0c49	4, e5ad	0	0
PARAM	d_gu	0x894	0xFFD00894	0x0c4a	4, e5b8	0	0
PARAM	d_go	0x896	0xFFD00896	0x0c4b	4, e5bc	0	0
PARAM	d_attmax	0x898	0xFFD00898	0x0c4c	0	0	0
PARAM	d_sm	0x89a	0xFFD0089A	0x0c4d	4, e5c4	0	0
PARAM	d_b	0x89c	0xFFD0089C	0x0c4e	0	0	0
PARAM	d_v42b_switch_hyst	0x89e	0xFFD0089E	0x0c4f	0	0	0
PARAM	d_v42b_switch_min	0x8a0	0xFFD008A0	0x0c50	0	0	0
PARAM	d_v42b_switch_max	0x8a2	0xFFD008A2	0x0c51	0	0	0
PARAM	d_v42b_reset_delay	0x8a4	0xFFD008A4	0x0c52	0	0	0
PARAM	d_ldT_hr	0x8a6	0xFFD008A6	0x0c53	0	0	0
PARAM	d_maccthresh_hr	0x8a8	0xFFD008A8	0x0c54	0	0	0
PARAM	d_maccthresh_hr	0x8aa	0xFFD008AA	0x0c55	0	0	0
PARAM	d_gu_hr	0x8ac	0xFFD008AC	0x0c56	0	0	0
PARAM	d_go_hr	0x8ae	0xFFD008AE	0x0c57	0	0	0
PARAM	d_b_hr	0x8b0	0xFFD008B0	0x0c58	0	0	0
PARAM	d_sm_hr	0x8b2	0xFFD008B2	0x0c59	0	0	0
PARAM	d_attmax_hr	0x8b4	0xFFD008B4	0x0c5a	0	0	0
PARAM	c_mldt_efr	0x8b6	0xFFD008B6	0x0c5b	0	0	0
PARAM	c_maccthresh_efr	0x8b8	0xFFD008B8	0x0c5c	0	0	0
PARAM	c_maccthresh_efr	0x8ba	0xFFD008BA	0x0c5d	0	0	0
PARAM	c_gu_efr	0x8bc	0xFFD008BC	0x0c5e	0	0	0
PARAM	c_go_efr	0x8be	0xFFD008BE	0x0c5f	0	0	0
PARAM	c_b_efr	0x8c0	0xFFD008C0	0x0c60	0	0	0
PARAM	c_sm_efr	0x8c2	0xFFD008C2	0x0c61	0	0	0
PARAM	c_attmax_efr	0x8c4	0xFFD008C4	0x0c62	0	0	0
PARAM	d_sd_min_thres_hfs	0x8c6	0xFFD008C6	0x0c63	2619, a31d	0	0
PARAM	d_ma_min_thres_hfs	0x8c8	0xFFD008C8	0x0c64	5238, a33b	0	0

zone	champ	octet API	ARM	mot DSP	ROM lit (n, 1er PC)	ROM écrit (n, 1er PC)	ARM modifie (n, valeurs)
PARAM	d_md_max_thr_tchfs	0x8ca	0xFFD008CA	0x0c65	0	0	0
PARAM	d_md1_max_thr_tchfs	0x8cc	0xFFD008CC	0x0c66	2619, a356	0	0
PARAM	d_sd_min_thr_tchhs	0x8ce	0xFFD008CE	0x0c67	0	0	0
PARAM	d_ma_min_thr_tchhs	0x8d0	0xFFD008D0	0x0c68	0	0	0
PARAM	d_sd_av_thr_tchhs	0x8d2	0xFFD008D2	0x0c69	0	0	0
PARAM	d_md_max_thr_tchhs	0x8d4	0xFFD008D4	0x0c6a	0	0	0
PARAM	d_md1_max_thr_tchhs	0x8d6	0xFFD008D6	0x0c6b	0	0	0
PARAM	d_sd_min_thr_tchefs	0x8d8	0xFFD008D8	0x0c6c	0	0	0
PARAM	d_ma_min_thr_tchefs	0x8da	0xFFD008DA	0x0c6d	0	0	0
PARAM	d_md_max_thr_tchefs	0x8dc	0xFFD008DC	0x0c6e	0	0	0
PARAM	d_md1_max_thr_tchefs	0x8de	0xFFD008DE	0x0c6f	0	0	0
PARAM	d_wed_fil_in	0x8e0	0xFFD008E0	0x0c70	0	0	0
PARAM	d_wed_fil_tc	0x8e2	0xFFD008E2	0x0c71	0	0	0
PARAM	d_x_min	0x8e4	0xFFD008E4	0x0c72	0	0	0
PARAM	d_x_max	0x8e6	0xFFD008E6	0x0c73	0	0	0
PARAM	d_slope	0x8e8	0xFFD008E8	0x0c74	0	0	0
PARAM	d_y_min	0x8ea	0xFFD008EA	0x0c75	0	0	0
PARAM	d_y_max	0x8ec	0xFFD008EC	0x0c76	0	0	0
PARAM	d_wed_diff_threshold	0x8ee	0xFFD008EE	0x0c77	0	0	0
PARAM	d_mabfi_min_thr_tchhs	0x8f0	0xFFD008F0	0x0c78	0	0	0
PARAM	d_facch_thr	0x8f2	0xFFD008F2	0x0c79	2654, 91e3	0	0
PARAM	d_max_ovsp_thr	0x8f4	0xFFD008F4	0x0c7a	0	0	0
PARAM	d_sync_thres	0x8f6	0xFFD008F6	0x0c7b	0	0	0
PARAM	d_idle_thres	0x8f8	0xFFD008F8	0x0c7c	0	0	0
PARAM	d_m1_thres	0x8fa	0xFFD008FA	0x0c7d	0	0	0
PARAM	d_max_ovsp_thr2	0x8fc	0xFFD008FC	0x0c7e	0	0	0
PARAM	d_gsm_bgd_thr	0x8fe	0xFFD008FE	0x0c7f	0	0	0
PARAM	a_fir_holes[4]	0x900	0xFFD00900	0x0c80..0x0c83	0	0	0
PARAM	a_fir31_uplink[30]	0x908	0xFFD00908	0x0c84..0x0ca1	0	0	0
PARAM	a_fir31_downlink[31]	0x946	0xFFD00946	0x0ca3..0x0cc1	0	0	0

Bits utiles (`include/calypso/l1_environment.h`, **positions** de bit) : B_BLUD 15 (bloc présent), B_AF 14, B_EMPTY_BLOCK 10, B_ECRC 9, B_SCH_CRC 8, B_FIRE1 6, B_FIRE0 5, B_BFI 2, B_UFI 0 ; d_dsp_page : B_GSM_PAGE 1«0, B_GSM_TASK 1«1 (masques) ; d_ctrl_system : B_TSQ 0..2, B_BCCH_FREQ_IND 3, B_TASK_ABORT 15 ; d_ctrl_tch : B_CHAN_MODE 0, B_CHAN_TYPE 4, B_RESET_SACCH 6, B_VOCODER_ON 7, B_SYNC_TCH_UL 8, B_SYNC_TCH_DL 9, B_STOP_TCH_UL 10, B_STOP_TCH_DL 11, B_TCH_LOOP 12, B_SUBCHANNEL 15. Tâches : NO 0, FB 5, SB 6, TCH_FB 8, TCH_SB 9, RACH 10, AUL 11, DUL 12, TCHT 13, TCHA 14, NBN 17, EBN 18, NBS 19, EBS 20, NP 21, EP 22, ALLC 24, CB 25, DDL 26, ADL 27, TCHD 28, CHECKSUM 33 (bit 15 = inversion I/Q, `dsp_task_iq_swap`).

PARAM (valeurs de `dsp_params`, lues dans l'ELF : `print/x dsp_params`) : d_transfer_rate 0x6666, d_lat_mcu_bridge 15, d_lat_mcu_hom2sam 12, d_lat_mcu_bef_fast_access 5, d_lat_dsp_after_sam 4, d_gprs_install_address 0x7002, d_misc_config 1, d_cn_sw_workaround 0xe, d_fb_margin_beg 24, d_fb_margin_end 22, d_nsubb_idle 296, d_nsubb_dedic 30, d_fb_thr_det_iacq 0x3333, d_fb_thr_det_track 0x28f6, d_dc_off_thres 0x7fff, d_dummy_thres 0x4400, d_dem_pond_gewl 0x6800, d_dem_pond_red 0x4eb8, d_maccthresh1 0x1ec0, d_mldt −4, … (suite : `firmware/calypso/dsp_params.c`).

2.9.1.4 3. Faits mesurés sur l'interface (rejeu)

- **d_dsp_page (0x08d4)** : l'ARM l'alterne 0x0002/0x0003 (13234 modifications) ; la ROM le lit 13218 fois (1er PC 0xa51c) ; chaque modification est lue au tick suivant (latence 1,00).
- **a_cu[0] (0x0a06)** : l'ARM pose B_BLUD (0x8000) 315 fois ; la ROM le lit 315 fois et le réécrit 315 fois (PC 0xaa5a) : c'est la ROM qui consomme B_BLUD (le `w[0] &= ~B_BLUD` de `montant.c:262` imite donc le bon comportement).

- **a_du_1[0] (0x096e)** : l'ARM pose B_BLUD 2488 fois (parole montante TCH/F) ; la ROM le lit 5144 fois et l'écrit 2656 fois (PC 0xa925). **a_du_0 (0x0a24..)** : jamais modifié par l'ARM, mais lu 5314 / écrit 7829 fois par la ROM. Voir incohérence n° 6.
- **a_cd[0] (0x09d2)** : la ROM l'écrit 624 fois ; l'ARM y remet 0x0040 (B_FIRE1, `prim_tch.c:700`).
- **a_dd_0[0] (0x09f0)** : ROM lit 7892 / écrit 7892 ; l'ARM y remet 0 après lecture (2654 fois).
- **d_version_number1/2 (0x08da/0x08db)** : jamais touchés pendant le rejeu (écrits au boot).

2.9.1.5 4. Mémoire interne du DSP (établie le 2026-10-03 + mesure du rejeu)

adresse	signification	ROM lit (n, 1er PC)	ROM écrit (n, 1er PC)	source
0x4280..0x429f	tampon montant entrelacé xCCH, 4 bursts × 8 mots, circulaire BK=32	1578, 8dbd	1579, 8869	mémoire rom-full-reste ; MAILBOX.md 2026-10-03
0x3f8a..0x3f91	burst montant émis (114 bits + hl/hu dans les 2 bits bas du mot 7), copié par PROM0 0x8900..0x891b	1441, 85d3	13324, 8916	idem
0x3f9b..0x3fa2	flux A5 montant XORé sur 0x3f8a (PROM0 0x85d1)	1441, 85d4	37225, b185	`src/pont.c:570`
0x3f93..0x3f9a	flux A5 descendant	11875, 8575	37225, b185	brief 2026-10-03
0x3f92	bit 0x1000 = flux A5 prêt	168751, b04c	99947, b04c	brief
0x3d9b	pointeur de lecture du circulaire 0x4280	1573, 8bd5	1577, 8820	`src/pont.c:571`
0x2bfa..	codé natural-order / tampon de travail	305283, a2c4	61427, 8ba8	mémoire rom-montant-codage
0x2bf8	drapeau parité classe 1a	338033, 8df0	84547, 8daf	brief
0x3d90	drapeaux qualité CHED (0x01 parité, 0x02 MA, 0x08 SD-min, 0x10 …)	37455, 8758	29527, 8758	brief
0x2d42.. (50 mots)	valeurs SD	16195, 9abe	16164, 9aad	brief
0x2c00..0x2c1f	métriques Viterbi	2679012, a2c4	398814, 8ba8	brief
0x3fad / 0x3fae	bits de mode (0x3fad bit 15 = verrou du répartiteur FB, PC 0x8753)	343904 / 92658	27812 / 22517	brief ; `c54x_mem.c` PROBE-3FAD-GATE
0x3d89	pointeur d'écriture du circulaire 232 mots en 0x4d00 (init 0x90c0)	13281, 9169	10634, 90c0	`c54x_mem.c` garde-3d89
0x43d8	slot du gestionnaire courant (`ld *(0x43d8),A ; cala A` en 0xb01c)	13218, b01c	0	`c54x_mem.c` HANDLER-WATCH

Récolte plus large (MAILBOX.md, README.fr.md, sources) : `recolte_c54x.tsv` (brouillon, non encore fusionné ici).

2.9.1.6 5. Ports I/O (mesurés : PORTR/PORTW 0x74xx/0x75xx du rejeu)

port	sens	n	1er PC	rôle	source
0x0002	W	26436	b572	RIF SPCX (contrôle émission)	`calypso_rif.h:19`
0x0003	R/W	81949 / 88202	a675 / a67b	RIF SPCR (contrôle réception)	`calypso_rif.h:20`
0x2800	W	147204	b142	A5 : commande (0 arrêt, 0x16/0x17)	`calypso_a5.c:39`
0x2801, 0x2802	R	36801	b1a0, b16c	A5 : trace, état	`calypso_a5.c:40-41`
0x2803..0x2806	W	36801	b145..	A5 : Kc (= a_kc[0..3])	`calypso_a5.c:42`
0x2807, 0x2808	W	36801	b151, b154	A5 : a_a5fn[0] (T3«5|T2), a_a5fn[1] (T1)	`calypso_a5.c:13-15`
0x2809..0x2818	R	36801	b1ab..	A5 : flux de clé (descendant puis montant)	`calypso_a5.c:45`
0xf900	W	232373	a6c3	API Control (CAL207 7.2.2)	`calypso_xio.c:38`
0xfa01	W	18809	a658	INTH du DSP	`calypso_xio.c`
0xfc24..0xfc28	R/W	~12500..31315	a5e8..	fenêtre DMA RHEA (AAD 0x099c = tampon 0x0cce)	`calypso_rhea_dma.c`

Vecteurs d'interruption (`calypso_c54x.h:162-191`) : TPU frame vec 28 / IMR bit 12, API vec 25 / bit 9, DMA vec 30 / bit 14, RIF RX vec 16 / bit 0, A5 (CRYPT) vec 27 / bit 11.

2.9.1.7 6. Incohérences relevées (layer1 ↔ qosmo ↔ c54x_exe)

1. **`calypso_bsp_tx_burst()` lit `data[0x0900 + i]`** (`qosmo/.../calypso_bsp.c:3572`) : 0x0900 est dans le NDB (a_sch26[2..]), pas un tampon d'émission. Le burst émis par la ROM est en 0x3f8a (déjà connu ; c54x_exe le prend là via `pont.c tx_rom_publier`).
2. **Même nom, unités différentes** : `NDB_D_DSP_PAGE`/`NDB_D_FB_DET` = offsets d'**octets** dans le NDB (0x000/0x048, `include/hw/arm/calypso/calypso_api.h`) et adresses **mot DSP absolues** (0x08D4/0x08F8, `l1-dsp/calypso_fbsb.h:56-63`). Valeurs justes chacune dans sa convention ; une unité qui inclurait les deux en-têtes aurait une redéfinition silencieuse (à vérifier).
3. **B_BLUD** : position de bit (15) côté layer1, masque (1u«15) côté qosmo — même nom.
4. **CALYPSO_API_SIZE = 64 Kio** (32768 mots, `calypso_api.h`) contre 16 Kio réels (API_SIZE 0x2000 mots, `firmware/calypso/dsp.c:45`) ; l'enregistrement `S` fait 32768 mots, le DSP n'en voit que 0x2000. Aucune écriture ARM au-delà de 0x2000 dans le rejeu (0 mot).
5. **d_ctrl_tch** : `dsp.c:503` documente b_chan_type sur les bits 4..7, `dsp_api.h` sur 4..5 avec bit 6 = reset SACCH et bit 7 = vocodeur ON ; `dsp_load_tch_param(chan_type=SDCCH_8=4)` pose donc le bit 6. Valeurs réellement écrites dans le rejeu : à analyser.
6. **Parole montante TCH/F dans a_du_1** : `prim_tch.c:485 tch_sub ? a_du_0 : a_du_1` (alors que la descente prend `tch_sub ? a_dd_1 : a_dd_0`, `prim_tch.c:322`) ; l'ARM écrit 0x096e (a_du_1), la ROM le lit, mais elle lit et écrit aussi a_du_0 (0x0a24) que l'ARM ne touche jamais. `montant.c:314` lit NDB_A_DU_1 : cohérent avec la layer1. Rôle exact de a_du_0 pour la ROM : à confirmer.
7. Tailles : sizeof ARM ≠ hôte (T_DB_MCU_TO_DSP 36/34, T_NDB 6356, T_PARAM 292 : bourrage de fin) ; offsets identiques, sans effet.
8. Recoupements **vérifiés justes** (à la main, contre l'ELF) : tous les WP_*/RP*_/NDB_* de `calypso_api.h`, API_VERSION/API_VERSION2 (= d_version_number1/2), API_BL_STATUS 0x0FFE, `calypso_fbsb.h` (d_dsp_page, d_dsp_state 0x08E2, d_fb_det, d_fb_mode, a_sync_demod[0..3]), `calypso_arm2dsp.c` A2D_DSP_PAGE_OFF 0x01A8, `calypso_bsp.c` D_RACH_DEFAULT_OFFSET 0x023A (= d_rach), `src/rejouer.c` W_PAGE/R_PAGE/NDB/PARAM/BL_*, `tools/rejeu_banc.c`

(a_cd 0x1FC, a_fd 0x21A, a_dd_0 0x238, a_kc 0x2CE, d_a5mode 0x1CE), `src/montant.c` NDB_D_TCH_MODE 0x006, `dsp.c` SC_CHKSUM_VER (0x08DB = d_version_number2).

2.9.1.8 7. Reste à faire

- Finir `tools/layer1_tester.py` (seule la partie extraction est écrite) : recoupement automatique, balayage statique des fonctions layer1, jugements PASS/FAIL par champ et par fonction, génération de cette page.
- Analyser les valeurs ARM (d_ctrl_tch, d_fn, a_a5fn, page W vs d_dsp_page, écho W→R) — données prêtes dans la mesure et le journal.
- Fusionner la récolte (MAILBOX.md, README.fr.md, sources qosmo) dans la section 4.

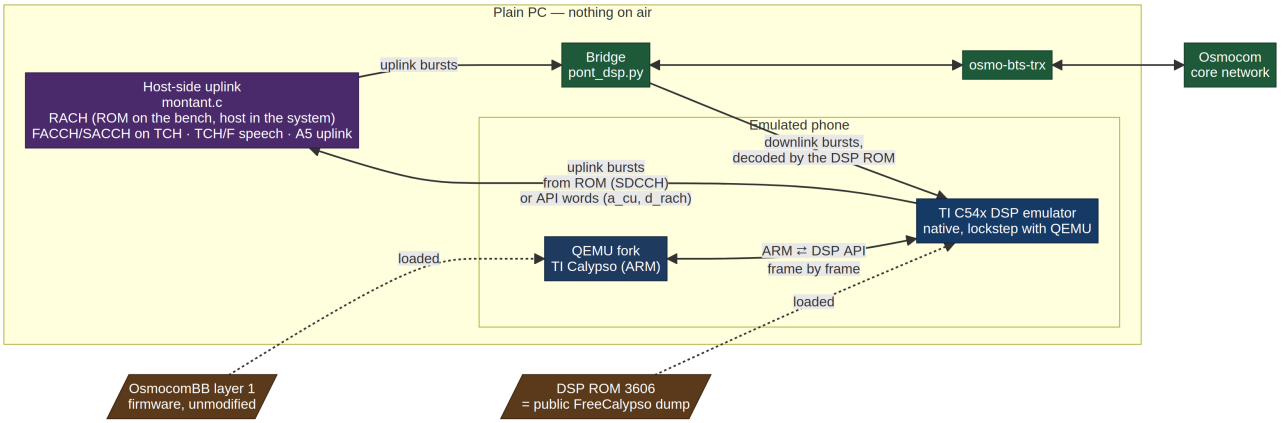
2.10 /opt/GSM/c54x_exe/docs/emulating-the-baseband.md

8368 octets, 124 lignes

2.10.1 Emulating the baseband

A complete GSM call on an ordinary PC, where the phone's real layer 1 firmware and the real baseband DSP code take part, down to the lowest layers of 2G. I forked QEMU to emulate the TI Calypso platform so that OsmocomBB's layer 1 runs on it, and I wrote an emulator for the TI C54x DSP that layer 1 talks to. Nothing goes on air: no SDR, no radio hardware, no spectrum licence. The baseband is the most closed part of a phone. Ever wanted to watch what a 2G DSP actually does, step by step, without any hardware? Now you can.

On the phone side, the QEMU Calypso machine loads the unmodified OsmocomBB layer 1 firmware as the real ARM would, and a native C54x emulator runs the DSP from its mask ROM — identical, word for word, to the public FreeCalypso dump of ROM version 3606, which is where to get it; nothing is redistributed here.



Layer 1 drives the DSP through its shared API, as on real hardware. In the downlink, the DSP ROM itself does the work: frequency burst detection, synchronisation, control channel and speech decoding, A5 deciphering. In the uplink, SDCCH is produced by the ROM too, after four bugs in the emulated core were fixed; RACH, FACCH/SACCH on traffic channels, uplink speech and uplink A5 are still encoded on the host side in the full system, and moving them into the DSP is the current work (the RACH already is, on the bench: see below). A bridge connects the phone to osmo-bts-trx, following the design of fixeria's trxcon, and an Osmocom core network handles the call on the other side. An end-to-end test bench checks the whole chain, from boot to attach, SMS, USSD, a voice call and a test tone recovered through it. An experimental mode runs layer 1 without the TI ROM, with host-side signal processing (gr-gsm, libosmocoding): not a free baseband yet, but the layer above it running without the vendor's code.

One number to keep in mind while reading the rest: of the 230 worked examples of TI's instruction set manual (SPRU172C) that the bench extracts, the emulated C54x core passes 150, fails 58, and 22 cannot be assembled by the harness. Every « the ROM does X » below is conditional on the core executing the instructions of that path correctly; the two bugs found on 2026-10-04 were found exactly that way.

2.10.1.1 Three stages, as they appear in the code

1. **Layer 1 in QEMU.** The QEMU fork runs an unmodified `layer1.highram` on the Calypso machine (a patched build that passed bad speech frames at zero gain was used from 2026-10-03 to 2026-10-04 and has been removed). The API RAM is the real one (`/dev/shm/calypso_api_ram`), and the ARM DSP dialogue happens frame by frame.
2. **The gr-gsm shunt.** The `grgsm` mode of `run.sh`, the `shunt_*` traces and `l1-grgsm/`: a « host path » in which gr-gsm did FB/SB detection inside QEMU, the libosmocoding bridge did channel decoding and the bridge did A5. The DSP was bypassed; the host rebuilt the signal processing.
3. **The DSP that works.** `c54x_exe`: the TI mask ROM executed on the emulated C54x core, in lockstep with QEMU. State on 2026-10-03, from the bench's coverage table (one line per signal-processing step): 6 steps done by the ROM (FB, SCH, control channels, FACCH, speech decoding, A5 downlink), 7 by the host (IMMEDIATE ASSIGNMENT reference retouched — the table counts it as a host step, the ROM pattern being absent by construction —, RACH, A5 uplink, FACCH uplink, SACCH on TCH uplink, uplink speech, vocoder), 1 mixed (SDCCH uplink: 134 blocks by the ROM, 1 by the host).

2.10.1.2 What "processor fidelity" means now

Fidelity is no longer measured as « does the phone register » but as « how many API RAM words and how many bits at the antenna are produced by the ROM itself ». The capture point sets the fidelity:

- **Capturing at `0x3f8a`** (what `tx_rom_publier` does today) takes the coded bits *before* the modulator. The host still has to add the training sequence, synchronisation and tail bits, and redo A5 at the right frame number.
- **Capturing the TSP script** (`0x3cbb..`, marker `0x1c0a` = TOGBR2, then 16 BULDATA words of 10 bits) takes the stream *the ABB would receive on silicon*, with training sequence, tails, I/Q inversion and A5 already applied. The host then only converts ABB words into 148 bits and places them at the right fn/tn. In the ROM this capture point serves every burst type (NB, RACH, TCH): one pipe, the ROM decides everything.

That second capture point did not exist in emulation until 2026-10-04, and not because of the host: the ROM wrote the burst, the core did not know how to execute two of the instructions on the way. The emitter loop at PROM0 `0x8608` begins with `cmpr eq,ar2 ; rc tc` and the core had no `CMPR` at all (a no-op, so the loop returned at once on a stale flag); once that was fixed, the 16 words came out as `0xff06` sixteen times, because `LD src, ASM` ignored its shift and the bit accumulator never moved. With both fixed, the bench's new `rach` test shows the ROM executing the RACH task (`0xb6fd`, then `0x85a2`), writing its 36 coded bits to `0x3f8a`, and the emitter producing the exact 05.02 access burst in the script — extended tail `00111010`, the 41-bit synchronisation sequence copied from the data ROM at `0xa0d9`, the 36 bits, three tail bits, guard — bit for bit what libosmocoding's `gsm0503_rach_ext_encode` gives, 5 attempts out of 5. On the bench only: in the full system `montant.c` still erases `d_task_ra` before the ROM can see it, and the bridge still codes the RACH. Connecting the pipe is the next step, and it is now a plumbing step.

2.10.1.3 The hierarchy for what comes next

- **Before any new pipe: the 58 failing SPRU172C examples.** They are the manufacturer's own test vectors; each is a candidate for the next « the ROM computes wrongly ». Two were found by symptom this week (CMPR, LD src,ASM); the list finds them by construction.
- **Then the single pipe:** make the TSP script the source of all uplink bursts (RACH, SDCCH, TCH, SACCH, FACCH), and transmit at the ROM's frame number so that its A5 is the right one — after reading in the ROM which frame counter it uses for the uplink stream, which has not been done.
- **Cases where the ROM computes wrongly or not at all in the system:** downlink SACCH/8 often failing its FIRE check (the 2026-10-03 note says « ~80 % », without a count), `B_BFI` on speech, and on the bench `nb` and `fb0`/`fb1`. Instruction-level debugging, like the four bugs behind the SDCCH uplink.
- **The most distant fidelity:** the vocoder in the ROM, which means modelling the audio path from the DSP to the ABB.

In other words, the DSP has stopped being an obstacle to work around and has become the reference: whenever the host still does something, it is either because it has not yet been given the ROM's pipe, or because the core still has a bug. Both are measurable, and that is what is being measured — the evidence for every sentence above is in `docs/CLAIMS-2026-10-04.md`.

2.11 /opt/GSM/c54x_exe/tools/isa_tests.txt

32706 octets, 2208 lignes → 2206 lignes (1 groupes compactés)

```
T 1 ABDST Xmem, Ymem | ABDST *AR3+, *AR4+
W e39a
B A ffabcd0000
B AR3 100
B AR4 200
B B 0
B FRCT 0
B M 0100 0055
B M 0200 00aa
A A ffffab0000
A AR3 101
A AR4 201
A B 5433
A FRCT 0
A M 0100 0055
A M 0200 00aa
E
T 2 ABS src [, dst ] | ABS A, B
W f585
B A fffffffcb
B B fffffffc18
A A fffffffcb
A B 35
E
T 3 ABS src [, dst ] | ABS A
W f485
B A 312345678
B OVM 1
A A 7fffffff
A OVM 1
E
T 4 ABS src [, dst ] | ABS A
W f485
B A 312345678
B OVM 0
A A 312345678
A OVM 0
E
T 5 6: Class 7 (see page 3-12) | ADD *AR3+, 14, A
W 909e
B A 1200
B AR3 100
B C 1
B SXM 1
B M 0100 1500
A A 5401200
A AR3 101
A C 0
A SXM 1
A M 0100 1500
E
T 6 6: Class 7 (see page 3-12) | ADD A, –8, B
W f518
B A 1200
B B 1800
B C 1
A A 1200
A B 1812
A C 0
E
T 7 6: Class 7 (see page 3-12) | ADD #4568h, 8, A, B
W f108 4568
```

```
B A 1200
B B 1800
B C 1
A A 1200
A B 457a00
A C 0
E
T 9 ADDC Smem, src | ADDC *+AR2(5), A
W 06ea 0005
B A 13
B AR2 100
B C 1
B M 0105 0004
A A 18
A AR2 105
A C 0
A M 0105 0004
E
T 10 ADDM #lk, Smem | ADDM 0123Bh, *AR4+
W 6b94 123b
B AR4 100
B M 0100 0004
A AR4 101
A M 0100 123f
E
T 11 ADDM #lk, Smem | ADDM 0FFF8h, *AR4+
W 6b94 fff8
B AR4 100
B OVM 1
B SXM 1
B M 0100 8007
A AR4 101
A OVM 1
A SXM 1
A M 0100 8000
E
T 12 ADDS Smem, src | ADDS *AR2–, B
W 038a
B AR2 104
B B 3
B M 0104 f006
A AR2 103
A B f009
A C 0
A M 0104 f006
E
T 13 4: Class 1 (see page 3-3) | AND *AR3+, A
W 1893
B A ff1200
B AR3 100
B M 0100 1500
A A 1000
A AR3 101
A M 0100 1500
E
T 14 4: Class 1 (see page 3-3) | AND A, 3, B
W f183
B A 1200
B B 1800
A A 1200
A B 1000
E
T 15 ANDM #lk, Smem | ANDM #00FFh, *AR4+
W 6894 00ff
B AR4 100
B M 0100 0444
```

```
A AR4 101
A M 0100 0044
E
T 17 B[D] pmad | B 2000h
W f073 2000
B PC 1f45
A PC 2000
E
T 18 B[D] pmad | BD 1000h
W f273 1000
B PC 1f45
A PC 1000
E
T 19 BACC[D] src | BACC A
W f4e2
B A 3000
B PC 1f45
A A 3000
A PC 3000
E
T 20 BACC[D] src | BACCD B
W f7e2
B B 2000
B PC 1f45
A B 2000
A PC 2000
E
T 21 BANZ[D] pmad, Sind | BANZ 2000h, *AR3–
W 6c8b 2000
B AR3 5
B PC 1000
A AR3 4
A PC 2000
E
T 22 BANZ[D] pmad, Sind | BANZ 2000h, *AR3–
W 6c8b 2000
B AR3 0
B PC 1000
A AR3 ffff
A PC 1002
E
T 23 BANZ[D] pmad, Sind | BANZ 2000h, *AR3(–1)
W 6ce3 ffff 2000
B AR3 1
B PC 1000
A AR3 1
A PC 1003
E
T 24 BANZ[D] pmad, Sind | BANZD 2000h, *AR3–
W 6e8b 2000
B AR3 4
B PC 1000
A AR3 3
A PC 2000
E
T 25 BC[D] pmad, cond [, cond [, condĂ ]ā] | BC 2000h, AGT
W f846 2000
B A 53
B PC 1000
A A 53
A PC 2000
E
T 26 BC[D] pmad, cond [, cond [, condĂ ]ā] | BC 2000h, AGT
W f846 2000
B A fffffffff
B PC 1000
```

```
A A ffffffffff
A PC 1002
E
T 27 BC[D] pmad, cond [, cond [, condĂ ]ā] | BCD 1000h, BOV
W fa78 1000
B OVB 1
B PC 3000
A OVB 1
A PC 1000
E
T 28 BC[D] pmad, cond [, cond [, condĂ ]ā] | BC 1000h, TC, NC, BIO
W f83b 1000
B C 1
B PC 3000
A C 1
A PC 3002
E
T 29 BIT Xmem, BITC | BIT *AR5+, 15-12; test bit 12
W 96b3
B AR5 100
B TC 0
B M 0100 7688
A AR5 101
A TC 1
A M 0100 7688
E
T 30 BITF Smem, #lk | BITF 5, 00FFh
W 6105 00ff
B DP 4
B M 0205 5400
A DP 4
A TC 0
A M 0205 5400
E
T 31 BITF Smem, #lk | BITF 5, 0800h
W 6105 0800
B DP 4
B M 0205 0f7f
A DP 4
A TC 1
A M 0205 0f7f
E
T 32 BITT Smem | BITT *AR7+0
W 34b7
B AR0 8
B AR7 100
B T c
B TC 0
B M 0100 0008
A AR0 8
A AR7 108
A T c
A TC 1
A M 0100 0008
E
T 33 CALA[D] src | CALA A
W f4e3
B A 3000
B PC 25
B SP 1111
B M 1110 4567
A A 3000
A PC 3000
A SP 1110
A M 1110 0026
E
```

```
T 34 CALA[D] src | CALAD B
W f7e3
B B 2000
B PC 25
B SP 1111
B M 1110 4567
A B 2000
A PC 2000
A SP 1110
A M 1110 0028
E
T 35 CALL[D] pmad | CALL 3333h
W f074 3333
B PC 25
B SP 1111
B M 1110 4567
A PC 3333
A SP 1110
A M 1110 0027
E
T 36 CALL[D] pmad | CALLD 1000h
W f274 1000
B PC 25
B SP 1111
B M 1110 4567
A PC 1000
A SP 1110
A M 1110 0029
E
T 37 CC[D] pmad, cond [, cond [, cond ] ] | CC 2222h, AGT
W f946 2222
B A 3000
B PC 25
B SP 1111
B M 1110 4567
A A 3000
A PC 2222
A SP 1110
A M 1110 0027
E
T 38 CC[D] pmad, cond [, cond [, cond ] ] | CCD 1000h, BOV
W fb78 1000
B OVB 1
B PC 25
B SP 1111
B M 1110 4567
A OVB 0
A PC 1000
A SP 1110
A M 1110 0029
E
T 39 CMPL src [, dst ] | CMPL A, B
W f593
B A fcdffaaeaa
B B 7899
A A fcdffaaeaa
A B 320055155
E
T 40 CMPM Smem, #lk | CMPM *AR4+, 0404h
W 6094 0404
B AR4 100
B TC 1
B M 0100 4444
A AR4 101
A TC 0
A M 0100 4444
```

```
E
S 41 CMPR CC, ARx | CMPR 2, AR4 | cmpr['OP_CC3', 'OP_ARX']: cond: '2'
T 42 CMPS src, Smem | CMPS A, *AR4+
W 8e94
B A 23457899
B AR4 100
B TC 0
B TRN 4444
B M 0100 0000
A A 23457899
A AR4 101
A TC 1
A TRN 8889
A M 0100 7899
E
T 43 DADD Lmem, src [, dst ] | DADD *AR3+, A, B
W 5193
B A 56788933
B AR3 100
B B 0
B C16 0
B M 0100 1534
B M 0101 3456
A A 56788933
A AR3 102
A B 6bacbd89
A C16 0
A M 0100 1534
A M 0101 3456
E
T 44 DADD Lmem, src [, dst ] | DADD *AR3–, A, B
W 518b
B A 56783933
B AR3 100
B B 0
B C16 1
B M 0100 1534
B M 0101 3456
A A 56783933
A AR3 fe
A B 6bac6d89
A C16 1
A M 0100 1534
A M 0101 3456
E
T 45 DADD Lmem, src [, dst ] | DADD *AR3–, A, B
W 518b
B A 56783933
B AR3 101
B B 0
B C16 0
B M 0100 1534
B M 0101 3456
A A 56783933
A AR3 ff
A B 8ace4e67
A C16 0
A M 0100 1534
A M 0101 3456
E
T 46 DADST Lmem, dst | DADST *AR3–, A
W 5a8b
B A 0
B AR3 100
B C16 1
B T 2345
```

```
B M 0100 1534
B M 0101 3456
A A 38791111
A AR3 fe
A C16 1
A T 2345
A M 0100 1534
A M 0101 3456
E
T 47 DADST Lmem, dst | DADST *AR3+, A
W 5a93
B A 0
B AR3 100
B C16 0
B T 2345
B M 0100 1534
B M 0101 3456
A A 3879579b
A AR3 102
A C16 0
A T 2345
A M 0100 1534
A M 0101 3456
E
T 48 DELAY Smem | DELAY *AR3
W 4d83
B AR3 100
B M 0100 6cac
B M 0101 0000
A AR3 100
A M 0100 6cac
A M 0101 6cac
E
T 49 DLD Lmem, dst | DLD *AR3+, B
W 5793
B AR3 100
B B 0
B M 0100 6cac
B M 0101 bd90
A AR3 102
A B 6cacbd90
A M 0100 6cac
A M 0101 bd90
E
T 50 DRSUB Lmem, src | DRSUB *AR3+, A
W 5893
B A 56788933
B AR3 100
B C16 0
B M 0100 1534
B M 0101 3456
A A ffbebbab23
A AR3 102
A C 0
A C16 0
A M 0100 1534
A M 0101 3456
E
T 51 DRSUB Lmem, src | DRSUB *AR3–, A
W 588b
B A 56783933
B AR3 100
B C 1
B C16 1
B M 0100 1534
B M 0101 3456
```

```
A A ffbebcfb23
A AR3 fe
A C 0
A C16 1
A M 0100 1534
A M 0101 3456
E
T 52 DSADT Lmem, dst | DSADT *AR3+, A
W 5e93
B A 0
B AR3 100
B C 0
B C16 0
B T 2345
B M 0100 1534
B M 0101 3456
A A fff1ef1111
A AR3 102
A C 0
A C16 0
A T 2345
A M 0100 1534
A M 0101 3456
E
T 53 DSADT Lmem, dst | DSADT *AR3–, A
W 5e8b
B A 0
B AR3 100
B C 0
B C16 1
B T 2345
B M 0100 1534
B M 0101 3456
A A fff1ef579b
A AR3 fe
A C 1
A C16 1
A T 2345
A M 0100 1534
A M 0101 3456
E
T 54 DST src, Lmem | DST B, *AR3+
W 4f93
B AR3 100
B B 6cacbd90
B M 0100 0000
B M 0101 0000
A AR3 102
A B 6cacbd90
A M 0100 6cac
A M 0101 bd90
E
T 55 DST src, Lmem | DST B, *AR3–
W 4f8b
B AR3 101
B B 6cacbd90
B M 0100 0000
B M 0101 0000
A AR3 ff
A B 6cacbd90
A M 0100 bd90
A M 0101 6cac
E
T 56 DSUB Lmem, src | DSUB *AR3+, A
W 5493
B A 56788933
```

```
B AR3 100
B C16 0
B M 0100 1534
B M 0101 3456
A A 414454dd
A AR3 102
A C16 0
A M 0100 1534
A M 0101 3456
E
T 57 DSUB Lmem, src | DSUB *AR3–, A
W 548b
B A 56783933
B AR3 100
B C 1
B C16 1
B M 0100 1534
B M 0101 3456
A A 414404dd
A AR3 fe
A C 1
A C16 1
A M 0100 1534
A M 0101 3456
E
T 58 DSUBT Lmem, dst | DSUBT *AR3+, A
W 5c93
B A 0
B AR3 100
B C16 0
B T 2345
B M 0100 1534
B M 0101 3456
A A fff1ef1111
A AR3 102
A C16 0
A T 2345
A M 0100 1534
A M 0101 3456
E
T 59 DSUBT Lmem, dst | DSUBT *AR3–, A
W 5c8b
B A 0
B AR3 100
B C16 1
B T 2345
B M 0100 1534
B M 0101 3456
A A fff1ef1111
A AR3 fe
A C16 1
A T 2345
A M 0100 1534
A M 0101 3456
E
T 60 EXP src | EXP A
W f48e
B A fffffffcb
B T 0
A A fffffffcb
A T 19
E
T 61 EXP src | EXP B
W f58e
B B 785432105
B T fffc
```

```
A B 785432105
A T fffc
E
T 62 FB[D] extpmad | FB 012000h
W f881 2000
B PC 1000
B XPC 0
A PC 2000
A XPC 1
E
T 63 FB[D] extpmad | FBD 7F1000h
W faff 1000
B PC 2000
B XPC 0
A PC 1000
A XPC 7f
E
T 64 FBACC[D] src | FBACC A
W f4e6
B A 13000
B PC 1000
B XPC 0
A A 13000
A PC 3000
A XPC 1
E
T 65 FBACC[D] src | FBACCD B
W f7e6
B B 7f2000
B XPC 1
A B 7f2000
A XPC 7f
E
T 66 FCALA[D] src | FCALA A
W f4e7
B A 7f3000
B PC 25
B SP 1111
B XPC 0
B M 110f 4567
B M 1110 4567
A A 7f3000
A PC 3000
A SP 110f
A XPC 7f
A M 110f 0000
A M 1110 0026
E
T 67 FCALA[D] src | FCALAD B
W f7e7
B B 202000
B PC 25
B SP 1111
B XPC 7f
B M 110f 4567
B M 1110 4567
A B 202000
A PC 2000
A SP 110f
A XPC 20
A M 110f 007f
A M 1110 0028
E
T 68 FCALL[D] extpmad | FCALL 013333h
W f981 3333
B PC 25
```

```
B SP 1111
B XPC 0
B M 110f 4567
B M 1110 4567
A PC 3333
A SP 110f
A XPC 1
A M 110f 0000
A M 1110 0027
E
T 69 FCALL[D] extpmad | FCALLD 301000h
W fbb0 1000
B PC 3001
B SP 1111
B XPC 7f
B M 110f 4567
B M 1110 4567
A PC 1000
A SP 110f
A XPC 30
A M 110f 007f
A M 1110 3005
E
S 70 FIRS Xmem, Ymem, pmad | FIRS *AR3+, *AR4+, COEFFS | firs['OP_Xmem', 'OP_Ymem', 'OP_pmad']: nombre: 'COEFFS'
T 71 FRAME K | FRAME 10h
W ee10
B SP 1000
A SP 1010
E
T 72 FRET[D] | FRET
W f4e4
B PC 2112
B SP 300
B XPC 1
B M 0300 0005
B M 0301 1000
A PC 1000
A SP 302
A XPC 5
A M 0300 0005
A M 0301 1000
E
T 73 FRETE[D] | FRETE
W f4e5
B PC 2112
B SP 300
B XPC 5
B M 0300 006e
B M 0301 0110
A PC 110
A SP 302
A XPC 6e
A M 0300 006e
A M 0301 0110
# valeur: ['xCxx']
E
T 77 INTR K | INTR 3
W f7c3
B INTM 0
B PC 25
B SP 1000
B M 0fff 9653
A INTM 1
A PC ff8c
A SP fff
A M 0fff 0026
```

```
E
T 78 4: Class 4B (see page 3-8) | LD *AR1, A
W 1081
B A 0
B AR1 200
B SXM 0
B M 0200 fedc
A A fedc
A AR1 200
A SXM 0
A M 0200 fedc
E
T 79 4: Class 4B (see page 3-8) | LD *AR1, A
W 1081
B A 0
B AR1 200
B SXM 1
B M 0200 fedc
A A ffffffedc
A AR1 200
A SXM 1
A M 0200 fedc
E
T 80 4: Class 4B (see page 3-8) | LD *AR1, TS, B
W 1581
B AR1 200
B B 0
B SXM 1
B T 8
B M 0200 fedc
A AR1 200
A B fffffedc00
A SXM 1
A T 8
A M 0200 fedc
E
T 81 4: Class 4B (see page 3-8) | LD *AR3+, 16, A
W 4493
B A 0
B AR3 300
B SXM 1
B M 0300 fedc
A A fffedc0000
A AR3 301
A SXM 1
A M 0300 fedc
# ligne registre incomplete: AR3                  0300                    AR1                  0301
E
T 82 4: Class 4B (see page 3-8) | LD #248, B
W e9f8
B B 0
B SXM 1
A B f8
A SXM 1
E
T 83 4: Class 4B (see page 3-8) | LD A, 8, B
W f548
B A 7ffd0040
B B ffff
B OVB 0
B SXM 1
B M 0200 fedc
A A 7ffd0040
A B 7ffd004000
A OVB 1
A SXM 1
```

```
A M 0200 fedc
E
T 84 2: Class 5B (see page 3-9) | LD *AR3+, T
W 3093
B AR3 300
B T 0
B M 0300 fedc
A AR3 301
A T fedc
A M 0300 fedc
E
T 85 2: Class 5B (see page 3-9) | LD *AR4, DP
W 4684
B AR4 200
B DP 1ff
B M 0200 fedc
A AR4 200
A DP dc
A M 0200 fedc
E
T 86 2: Class 5B (see page 3-9) | LD #23, DP
W ea17
B DP 1ff
A DP 17
E
T 87 2: Class 5B (see page 3-9) | LD 15, ASM
W ed0f
B ASM 0
A ASM f
E
T 88 2: Class 5B (see page 3-9) | LD 3, ARP
W f4a3
B ARP 0
A ARP 3
E
T 89 2: Class 5B (see page 3-9) | LD 0, ASM
W 3200
B ASM 0
B DP 4
B M 0200 fedc
A ASM 1c
A DP 4
A M 0200 fedc
E
T 90 LDM MMR, dst | LDM AR4, A
W 4814
B A 1111
B AR4 ffff
A A ffff
A AR4 ffff
E
T 91 LDM MMR, dst | LDM 060h, B
W 4960
B B 0
B M 0060 1234
A B 1234
A M 0060 1234
E
T 92 LD Xmem, dst | LD *AR4+, A
W a8ab
B A 1000
B AR4 100
B AR5 200
B B 1111
B FRCT 0
B T 400
```

```
B M 0100 1234
B M 0200 4321
A A 12340000
A AR4 101
A AR5 201
A B 10c9511
A FRCT 0
A T 400
A M 0100 1234
A M 0200 4321
E
T 93 LD Xmem, dst | LD *AR4+, A
W aaab
B A 1000
B AR4 100
B AR5 200
B B 1111
B FRCT 0
B T 400
B M 0100 1234
B M 0200 4321
A A 12340000
A AR4 101
A AR5 201
A B 10d0000
A FRCT 0
A T 400
A M 0100 1234
A M 0200 4321
E
T 94 LD Xmem, dst | LD *AR4+, A
W acab
B A 1000
B AR4 100
B AR5 200
B B 1111
B FRCT 0
B T 400
B M 0100 1234
B M 0200 4321
A A 12340000
A AR4 101
A AR5 201
A B fffef38d11
A FRCT 0
A T 400
A M 0100 1234
A M 0200 4321
E
T 95 LD Xmem, dst | LD *AR4+, A
W aeab
B A 1000
B AR4 100
B AR5 200
B B 1111
B FRCT 0
B T 400
B M 0100 1234
B M 0200 4321
A A 12340000
A AR4 101
A AR5 201
A B fffef40000
A FRCT 0
A T 400
A M 0100 1234
```

```
A M 0200 4321
E
T 96 LDR Smem, dst | LDR *AR1, A
W 1681
B A 0
B AR1 200
B SXM 0
B M 0200 fedc
A A fedc8000
A AR1 200
A SXM 0
A M 0200 fedc
E
T 97 LDU Smem, dst | LDU *AR1, A
W 1281
B A 0
B AR1 200
B M 0200 fedc
A A fedc
A AR1 200
A M 0200 fedc
E
T 98 LMS Xmem, Ymem | LMS *AR3+, *AR4+
W e19a
B A 77778888
B AR3 100
B AR4 200
B B 100
B FRCT 0
B M 0100 0055
B M 0200 00aa
A A 77cd0888
A AR3 101
A AR4 201
A B 3972
A FRCT 0
A M 0100 0055
A M 0200 00aa
E
T 99 LTD Smem | LTD *AR3
W 4c83
B AR3 100
B T 0
B M 0100 6cac
B M 0101 6cac
A AR3 100
A T 6cac
A M 0100 6cac
E
T 100 4: Class 6B (see page 3-11) | MAC *AR5+, A
W 2895
B A 1000
B AR5 100
B FRCT 0
B T 400
B M 0100 1234
A A 48e000
A AR5 101
A FRCT 0
A T 400
A M 0100 1234
E
T 101 4: Class 6B (see page 3-11) | MAC #345h, A, B
W f167 0345
B A 1000
B B 0
```

```
B FRCT 1
B T 400
A A 1000
A B 1a3800
A FRCT 1
A T 400
E
T 102 4: Class 6B (see page 3-11) | MAC *AR5+, #1234h, A
W 6495 1234
B A 1000
B AR5 100
B FRCT 0
B T 0
B M 0100 5678
A A 6261060
A AR5 101
A FRCT 0
A T 5678
A M 0100 5678
E
T 103 4: Class 6B (see page 3-11) | MAC *AR5+, *AR4+,A, B ;(AR6->AR4)
W b1ba
B A 1000
B AR4 200
B AR5 100
B B 4
B FRCT 1
B T 8
B M 0100 5678
B M 0200 1234
A A 1000
A AR4 201
A AR5 101
A B c4c10c0
A FRCT 1
A T 5678
A M 0100 5678
A M 0200 1234
E
T 104 4: Class 6B (see page 3-11) | MACR *AR5+, A
W 2a95
B A 1000
B AR5 100
B FRCT 0
B T 400
B M 0100 1234
A A 490000
A AR5 101
A FRCT 0
A T 400
A M 0100 1234
E
T 105 4: Class 6B (see page 3-11) | MACR *AR5+, *AR4+,A, B ;(AR6->AR4)
W b5ba
B A 1000
B AR4 200
B AR5 100
B B 4
B FRCT 1
B T 8
B M 0100 5678
B M 0200 1234
A A 1000
A AR4 201
A AR5 101
A B c4c0000
```

```
A FRCT 1
A T 5678
A M 0100 5678
A M 0200 1234
E
T 106 MACA[R] Smem [, B ] | MACA *AR5+
W 3595
B A 12340000
B AR5 100
B B 0
B FRCT 0
B T 400
B M 0100 5678
A A 12340000
A AR5 101
A B 6260060
A FRCT 0
A T 5678
A M 0100 5678
E
T 107 MACA[R] Smem [, B ] | MACA T, B, B
W f788
B A 12340000
B B 20000
B FRCT 1
B T 444
A A 12340000
A B 9d4ba0
A FRCT 1
A T 444
E
T 108 MACA[R] Smem [, B ] | MACAR *AR5+, B
W 3795
B A 12340000
B AR5 100
B B 0
B FRCT 0
B T 400
B M 0100 5678
A A 12340000
A AR5 101
A B 6260000
A FRCT 0
A T 5678
A M 0100 5678
E
T 109 MACA[R] Smem [, B ] | MACAR T, B, B
W f789
B A 12340000
B B 20000
B FRCT 1
B T 444
A A 12340000
A B 9d0000
A FRCT 1
A T 444
E
S 110 MACD Smem, pmad, src | MACD *AR3–, COEFFS, A | macd['OP_Smem', 'OP_pmad', 'OP_SRC1']: nombre: 'COEFFS'
S 111 MACP Smem, pmad, src | MACP *AR3–, COEFFS, A | macp['OP_Smem', 'OP_pmad', 'OP_SRC1']: nombre: 'COEFFS'
T 112 MACSU Xmem, Ymem, src | MACSU *AR4+, *AR5+, A
W a6ab
B A 1000
B AR4 100
B AR5 200
B FRCT 0
B T 8
```

```
B M 0100 8765
B M 0200 1234
A A 9a0aa84
A AR4 101
A AR5 201
A FRCT 0
A T 8765
A M 0100 8765
A M 0200 1234
E
T 113 MAR Smem | MAR *AR3+
W 6d93
B AR3 100
B ARP 0
B CMPT 0
A AR3 101
A ARP 0
A CMPT 0
E
T 114 MAR Smem | MAR *AR0–
W 6d88
B AR4 100
B ARP 4
B CMPT 1
A AR4 ff
A ARP 4
A CMPT 1
E
T 115 MAR Smem | MAR *AR3
W 6d83
B AR0 8
B AR3 100
B ARP 0
B CMPT 1
A AR0 8
A AR3 100
A ARP 3
A CMPT 1
E
T 116 MAR Smem | MAR *+AR3
W 6d9b
B AR3 100
B ARP 0
B CMPT 1
A AR3 101
A ARP 3
A CMPT 1
E
T 117 MAR Smem | MAR *AR3–
W 6d8b
B AR3 100
B ARP 0
B CMPT 1
A AR3 ff
A ARP 3
A CMPT 1
E
T 118 2: Class 7 (see page 3-12) | MAS *AR5+, A
W 2c95
B A 1000
B AR5 100
B FRCT 0
B T 400
B M 0100 1234
A A ffffb74000
A AR5 101
```

```
A FRCT 0
A T 400
A M 0100 1234
E
T 119 2: Class 7 (see page 3-12) | MAS *AR5+, *AR4+, A, B ;(AR6->AR4)
W b9ba
B A 1000
B AR4 200
B AR5 100
B B 4
B FRCT 1
B T 8
B M 0100 5678
B M 0200 1234
A A 1000
A AR4 201
A AR5 101
A B fff3b40f40
# errata SPRU172C: exemple MAS Xmem,Ymem calcule sans FRCT=1 (cf. MAC Xmem,Ymem test 103 qui l'applique)
A FRCT 1
A T 5678
A M 0100 5678
A M 0200 1234
E
T 120 2: Class 7 (see page 3-12) | MASR *AR5+, A
W 2e95
B A 1000
B AR5 100
B FRCT 0
B T 400
B M 0100 1234
A A ffffb70000
A AR5 101
A FRCT 0
A T 400
A M 0100 1234
E
T 121 2: Class 7 (see page 3-12) | MASR *AR5+, *AR4+, A, B ;(AR6->AR4)
W bdba
B A 1000
B AR4 200
B AR5 100
B B 4
B FRCT 1
B T 8
B M 0100 5678
B M 0200 1234
A A 1000
A AR4 201
A AR5 101
A B fff3b40000
# errata SPRU172C: idem MAS, rnd(A - 2*X*Y)
A FRCT 1
A T 5678
A M 0100 5678
A M 0200 1234
E
T 122 2: Class 1 (see page 3-3) | MASA *AR5+
W 3395
B A 12340000
B AR5 100
B B 20000
B FRCT 0
B T 400
B M 0100 5678
A A 12340000
```

```
A AR5 101
A B fff9dbffa0
A FRCT 0
A T 5678
A M 0100 5678
E
T 123 2: Class 1 (see page 3-3) | MASA T, B
W f78a
B A 12340000
B B 20000
B FRCT 1
B T 444
A A 12340000
A B ffff66b460
A FRCT 1
A T 444
E
T 124 2: Class 1 (see page 3-3) | MASAR T, B
W f78b
B A 12340000
B B 20000
B FRCT 1
B T 444
A A 12340000
A B ffff670000
A FRCT 1
A T 444
E
T 125 MAX dst | MAX A
W f486
B A fff6
B B ffcb
B C 1
A A fff6
A B ffcb
A C 0
E
T 126 MAX dst | MAX A
W f486
B A 55
B B 1234
B C 0
A A 1234
A B 1234
A C 1
E
T 127 MIN dst | MIN A
W f487
B A ffcb
B B fff6
B C 1
A A ffcb
A B fff6
A C 0
E
T 128 MIN dst | MIN A
W f487
B A 1234
B B 1234
B C 0
A A 1234
A B 1234
A C 1
E
T 129 4: Class 2 (see page 3-4) | MPY 13, A
W 200d
```

```
B A 36
B DP 8
B FRCT 1
B T 6
B M 040d 0007
A A 54
A DP 8
A FRCT 1
A T 6
A M 040d 0007
E
T 130 4: Class 2 (see page 3-4) | MPY *AR2–, *AR4+0%, B;
W a54e
B AR0 1
B AR2 1ff
B AR4 300
B B fffffffe0
B FRCT 0
B M 01ff 0010
B M 0300 0002
A AR0 1
A AR2 1fe
A AR4 301
A B 20
A FRCT 0
A M 01ff 0010
A M 0300 0002
E
T 131 4: Class 2 (see page 3-4) | MPY #0FFFEh, A
W f066 fffe
B A 0
B FRCT 0
B T 2000
A A ffffffc000
A FRCT 0
A T 2000
E
T 132 4: Class 2 (see page 3-4) | MPYR 0, B
W 2300
B B fffe000001
B DP 4
B FRCT 0
B T 1234
B M 0200 5678
A B 6260000
A DP 4
A FRCT 0
A T 1234
A M 0200 5678
E
T 133 2: Class 1 (see page 3-3) | MPYA *AR2
W 3182
B A ff87651111
B AR2 200
B B 320
B FRCT 0
B T 1234
B M 0200 5678
A A ff87651111
A AR2 200
A B ffd7436558
A FRCT 0
A T 5678
A M 0200 5678
E
T 134 2: Class 1 (see page 3-3) | MPYA B
```

```
W f58c
B A ff87651111
B B 320
B FRCT 0
B T 4567
A A ff87651111
A B ffdf4db2a3
A FRCT 0
A T 4567
E
T 135 MPYU Smem, dst | MPYU *AR0–, A
W 2488
B A ff80000000
B AR0 1000
B FRCT 0
B T 4000
B M 1000 fe00
A A 3f800000
A AR0 fff
A FRCT 0
A T 4000
A M 1000 fe00
E
T 136 MVDD Xmem, Ymem | MVDD *AR3+, *AR5+
W e59b
B AR3 8000
B AR5 200
B M 0200 abcd
B M 8000 1234
A AR3 8001
A AR5 201
A M 0200 1234
A M 8000 1234
E
T 137 MVDK Smem, dmad | MVDK 10, 8000h
W 710a 8000
B DP 4
B M 020a 1234
B M 8000 abcd
A DP 4
A M 020a 1234
A M 8000 1234
E
T 138 MVDK Smem, dmad | MVDK *AR3–, 1000h
W 718b 1000
B AR3 1ff
B M 01ff 1234
B M 1000 abcd
A AR3 1fe
A M 01ff 1234
A M 1000 1234
E
T 139 MVDM dmad, MMR | MVDM 300h, BK
W 7219 0300
B BK abcd
B M 0300 1234
A BK 1234
A M 0300 1234
E
T 140 MVDP Smem, pmad | MVDP 0, 0FE00h
W 7d00 fe00
B DP 4
B M 0200 0123
B P fe00 ffff
A DP 4
A M 0200 0123
```

```
E
T 141 MVKD dmad, Smem | MVKD 300h, 0
W 7000 0300
B DP 4
B M 0200 abcd
B M 0300 1234
A DP 4
A M 0200 1234
A M 0300 1234
E
T 142 MVKD dmad, Smem | MVKD 1000h, *+AR5
W 709d 1000
B AR5 1ff
B M 0200 abcd
B M 1000 1234
A AR5 200
A M 0200 1234
A M 1000 1234
E
T 143 MVMD MMR, dmad | MVMD AR7, 8000h
W 7317 8000
B AR7 1234
B M 8000 abcd
A AR7 1234
A M 8000 1234
E
T 144 MVMM MMRx, MMRy | MVMM SP, AR1
W e781
B AR1 3eff
B SP 200
A AR1 200
A SP 200
E
T 145 MVPD pmad, Smem | MVPD 0FE00h, 5
W 7c05 fe00
B DP 6
B M 0305 ffff
B P fe00 8a55
A DP 6
A M 0305 8a55
E
T 146 MVPD pmad, Smem | MVPD 2000h, *AR7–0
W 7caf 2000
B AR0 2
B AR7 ffe
B M 0ffe abcd
B P 2000 1234
A AR0 2
A AR7 ffc
A M 0ffe 1234
E
T 147 NEG src [, dst ] | NEG A, B
W f584
B A fffffff228
B B 1234
B OVA 0
A A fffffff228
A B dd8
A OVA 0
E
T 148 NEG src [, dst ] | NEG B, A
W f684
B A 1234
B B 80000000
B OVB 0
A A ff80000000
```

```
A B 80000000
A OVB 0
E
T 149 NEG src [, dst ] | NEG A
W f484
B A 8000000000
B OVA 0
B OVM 0
A A 8000000000
A OVA 1
A OVM 0
E
T 150 NEG src [, dst ] | NEG A
W f484
B A 8000000000
B OVA 0
B OVM 1
A A 7fffffff
A OVA 1
A OVM 1
E
T 152 NORM src [, dst ] | NORM A
W f48f
B A fffffff001
B T 13
A A ff80080000
A T 13
E
T 153 NORM src [, dst ] | NORM B, A
W f68f
B A fffffff001
B B 210a0a0a0a
B T ff9
A A 42141414
A B 210a0a0a0a
A T ff9
E
T 154 4: Class 1 (see page 3-3) | OR *AR3+, A
W 1a93
B A ff1200
B AR3 100
B M 0100 1500
A A ff1700
A AR3 101
A M 0100 1500
E
T 155 4: Class 1 (see page 3-3) | OR A, +3, B
W f1a3
B A 1200
B B 1800
A A 1200
A B 9800
E
T 156 ORM #lk, Smem | ORM 0404h, *AR4+
W 6994 0404
B AR4 100
B M 0100 4444
A AR4 101
A M 0100 4444
E
T 157 POLY Smem | POLY *AR3+%
W 36d3
B A 12340000
B AR3 200
B B 10000
B T 5678
```

```
B M 0200 2000
A A 6270000
A AR3 201
A B 20000000
A T 5678
A M 0200 2000
E
T 158 POPD Smem | POPD 10
W 8b0a
B DP 8
B SP 300
B M 0300 0092
B M 040a 0055
A DP 8
A SP 301
A M 0300 0092
A M 040a 0092
E
T 159 POPM MMR | POPM AR5
W 8a15
B AR5 55
B SP 3f0
B M 03f0 0060
A AR5 60
A SP 3f1
A M 03f0 0060
E
S 160 PORTR PA, Smem | PORTR 05, INDAT ; INDAT .equ 60h | portr['OP_PA', 'OP_Smem']: nombre: 'INDAT'
S 161 PORTW Smem, PA | PORTW OUTDAT, 5h ; OUTDAT .equ 07h | portw['OP_Smem', 'OP_PA']: nombre: 'OUTDAT'
T 162 PSHD Smem | PSHD *AR3+
W 4b93
B AR3 200
B SP 8000
B M 0200 07ff
B M 7fff 0092
A AR3 201
A SP 7fff
A M 0200 07ff
A M 7fff 07ff
E
T 163 PSHM MMR | PSHM BRC
W 4a1a
B BRC 1234
B SP 2000
B M 1fff 07ff
A BRC 1234
A SP 1fff
A M 1fff 1234
E
T 164 RC[D] cond [, cond [, condĂ] ] | RC AGEQ, ANOV              ; return is executed if the accumulator A
W fc62
B OVA 0
B PC 807
B SP 308
B M 0308 2002
A OVA 0
A PC 2002
A SP 309
A M 0308 2002
E
T 165 READA Smem | READA 6
W 7e06
B A 23
B DP 4
B M 0206 0075
B P 0023 0306
```

```
A A 23
A DP 4
A M 0206 0306
E
T 166 RESET | RESET
W f7e0
B INTM 0
B IPTR 1
B PC 25
A INTM 1
A IPTR 1
A PC 80
E
T 167 RET[D] | RET
W fc00
B PC 2112
B SP 300
B M 0300 1000
A PC 1000
A SP 301
A M 0300 1000
E
T 168 RETE[D] | RETE
W f4eb
B PC 1c3
B SP 2001
B M 2001 0110
A PC 110
A SP 2002
A M 2001 0110
# valeur: ['xCxx']
E
T 169 RETF[D] | RETF
W f49b
B INTM 1
B PC 1c3
B SP 2001
B M 2001 0110
A INTM 0
A PC 110
A SP 2002
A M 2001 0110
E
T 170 RND src [, dst ] | RND A, B
W f59f
B A fffffffff
B B 1
B OVM 0
A A ffffffffff
A B 7fff
A OVM 0
E
T 171 RND src [, dst ] | RND A
W f49f
B A 7fffffff
B OVM 1
A A 7fffffff
A OVM 1
E
T 172 ROL src | ROL A
W f491
B A 5fb0001234
B C 0
A A 60002468
A C 1
E
```

```
T 173 ROLTC src | ROLTC A
W f492
B A 81c0005555
B TC 1
A A 8000aaab
A C 1
A TC 1
E
T 174 ROR src | ROR A
W f490
B A 7fb0001235
B C 0
A A 5800091a
A C 1
E
S 175 3: Class 2 (see page 3-4) | RPT DAT127 ; DAT127 .EQU 0FFF | rpt['OP_Smem']: nombre: 'DAT127'; rpt['OP_k8u']:
 ↪  nombre: 'DAT127'; rpt['OP_lku']: nombre: 'DAT127'
T 178 RPTB[D] pmad | ST #99, BRC ; RPTB end_block-1 (end_block-1 = 100Fh)
W 761a 0063 f072 100f
N 2
B BRC 1234
B PC ffe
B RSA 5678
A BRC 63
A PC 1002
A RSA 1002
A REA 100f
E
T 179 RPTB[D] pmad | ST #99, BRC ; RPTBD end_block-1 ; 2 delay-slot NOPs
W 761a 0063 f272 100f
N 4
B BRC 1234
B PC ffe
B RSA 5678
A BRC 63
A PC 1004
A RSA 1004
A REA 100f
E
T 180 RPTZ dst, #lk | RPTZ A, 1023 ; Repeat the next instruction 1024 times
W f071 03ff
B A ffe008000
A A 0
E
T 181 RSBX N, SBIT | RSBX SXM ; SXM means: n=1 and SBIT=8
W f6b8
B ST1 35cd
A ST1 34cd
E
T 182 RSBX N, SBIT | RSBX 1,8
W f6b8
B ST1 35cd
A ST1 34cd
E
T 183 SACCD src, Xmem, cond | SACCD A, *AR3+0%, ALT
W 9ed3
B A fffe004321
B AR0 2
B AR3 202
B ASM 1
B M 0202 0101
A A fffe004321
A AR0 2
A AR3 204
A ASM 1
A M 0202 fc00
```

```
E
T 184 SAT src | SAT B
W f583
B B 7123456789
A B 7fffffff
A OVB 1
E
T 185 SAT src | SAT A
W f483
B A f812345678
A A ff80000000
A OVA 1
E
T 186 SAT src | SAT B
W f583
B B 123456
A B 123456
A OVB 0
E
T 187 SFTA src, SHIFT [, dst ] | SFTA A, -5, B
W f57b
B A ff87650055
B B 43211234
B SXM 1
A A ff87650055
A B fffc3b2802
A C 1
A SXM 1
E
T 188 SFTA src, SHIFT [, dst ] | SFTA B, +5
W f765
B B 80aa001234
B C 0
B OVM 0
B SXM 0
A B 1540024680
A C 1
A OVM 0
A SXM 0
E
T 189 SFTC src | SFTC A
W f494
B A fffffff001
A A ffffffe002
A TC 0
E
T 190 SFTL src, SHIFT [, dst ] | SFTL A, -5, B
W f1fb
B A ff87650055
B B ff80000000
B C 0
A A ff87650055
A B 43b2802
A C 1
E
T 191 SFTL src, SHIFT [, dst ] | SFTL B, +5
W f3e5
B B 80aa001234
B C 0
A B 40024680
A C 1
E
S 192 SQDST Xmem, Ymem | SQDST *AR3+, AR4+ | sqdst['OP_Xmem', 'OP_Ymem']: Xmem: 'AR4+'
T 193 2: Class 1 (see page 3-3) | SQUR 30, B
W 271e
B B 1f4
```

```
B DP 6
B FRCT 0
B T 3
B M 031e 000f
A B e1
A DP 6
A FRCT 0
A T f
A M 031e 000f
E
T 194 2: Class 1 (see page 3-3) | SQUR A, B
W f58d
B A f0000
B B 1010101
B FRCT 1
A A f0000
A B 1c2
A FRCT 1
E
T 195 SQURA Smem, src | SQURA 30, B
W 391e
B B 3200000
B DP 6
B FRCT 0
B T 3
B M 031e 000f
A B 32000e1
A DP 6
A FRCT 0
A T f
A M 031e 000f
E
T 196 SQURA Smem, src | SQURA *AR3+, A
W 3893
B A 1f4
B AR3 31e
B FRCT 0
B T 3
B M 031e 000f
A A 2d5
A AR3 31f
A FRCT 0
A T f
A M 031e 000f
E
T 197 SQURS Smem, src | SQURS 9, A
W 3a09
B A 14b5db0
B DP 6
B FRCT 0
B T 8765
B M 0309 1234
A A 320
A DP 6
A FRCT 0
A T 1234
A M 0309 1234
E
T 198 SQURS Smem, src | SQURS *AR3, B
W 3b83
B AR3 309
B B 14b5db0
B FRCT 0
B T 8765
B M 0309 1234
A AR3 309
```

```
A B 320
A FRCT 0
A T 1234
A M 0309 1234
E
T 199 SRCCD Xmem, cond | SRCCD *AR5–, AGT
W 9d76
B A 70ffffff
B AR5 202
B BRC 4321
B M 0202 1234
A A 70ffffff
A AR5 201
A BRC 4321
A M 0202 4321
E
T 200 SSBX N, SBIT | SSBX SXM     ; SXM means: N=1, SBIT=8
W f7b8
B ST1 34cd
A ST1 35cd
E
T 201 SSBX N, SBIT | SSBX 1,8
W f7b8
B ST1 34cd
A ST1 35cd
E
T 202 3: Class 12B (see page 3-27) | ST FFFFh, 0
W 7600 ffff
B DP 4
B M 0200 0101
A DP 4
A M 0200 ffff
E
T 203 3: Class 12B (see page 3-27) | ST TRN, 5
W 8d05
B DP 4
B TRN 1234
B M 0205 0030
A DP 4
A TRN 1234
A M 0205 1234
E
T 204 3: Class 12B (see page 3-27) | ST T, *AR7–
W 8c8f
B AR7 321
B T 4210
B M 0321 1200
A AR7 320
A T 4210
A M 0321 4210
E
T 205 4: Class 11B (see page 3-25) | STH A, 10
W 820a
B A ff87654321
B DP 4
B M 020a 1234
A A ff87654321
A DP 4
A M 020a 8765
E
S 206 4: Class 11B (see page 3-25) | STH B, –8, *AR7– | sth['OP_SRC1', 'OP_Smem']: Smem direct hors 0..7F: '-8';
 ↪  sth['OP_SRC1', 'OP_ASM', 'OP_Smem']: ASM attendu; sth['OP_SRC1'
S 207 4: Class 11B (see page 3-25) | STH A, –4, 10 | sth['OP_SRC1', 'OP_Smem']: Smem direct hors 0..7F: '-4';
 ↪  sth['OP_SRC1', 'OP_ASM', 'OP_Smem']: ASM attendu; sth['OP_SRC1'
T 208 4: Class 11B (see page 3-25) | STL A, 11
W 800b
```

```
B A ff87654321
B DP 4
B M 020b 1234
A A ff87654321
A DP 4
A M 020b 4321
E
S 209 4: Class 11B (see page 3-25) | STL B, -8, *AR7- | stl['OP_SRC1', 'OP_Smem']: Smem direct hors 0..7F: '-8';
 ↪  stl['OP_SRC1', 'OP_ASM', 'OP_Smem']: ASM attendu; stl['OP_SRC1'
S 210 4: Class 11B (see page 3-25) | STL A, 7, 11 | stl['OP_SRC1', 'OP_Smem']: operandes en trop: ['11'];
 ↪  stl['OP_SRC1', 'OP_ASM', 'OP_Smem']: ASM attendu; stl['OP_SRC1', '
T 211 STLM src, MMR | STLM A, BRC
W 881a
B A ff87654321
A A ff87654321
E
T 212 STLM src, MMR | STLM B, *AR1-
W 8989
B AR1 3f17
B B ff84211234
A AR1 16
A B ff84211234
E
T 213 STM #lk, MMR | STM 0FFFFh, IMR
W 7700 ffff
B IMR ff01
A IMR ffff
E
T 214 STM #lk, MMR | STM 8765h, *AR7+
W 7797 8765
B AR0 0
B AR7 8010
A AR0 8765
A AR7 11
E
S 215 ST src, Ymem | ST A, *AR3 | st['OP_T', 'OP_Smem']: T attendu; st['OP_TRN', 'OP_Smem']: TRN attendu;
 ↪  st['OP_lk', 'OP_Smem']: nombre: 'A'
S 216 ST src, Ymem | ST B, *AR2- | st['OP_T', 'OP_Smem']: T attendu; st['OP_TRN', 'OP_Smem']: TRN attendu;
 ↪  st['OP_lk', 'OP_Smem']: nombre: 'B'
S 217 ST src, Ymem | ST A, *AR3 | st['OP_T', 'OP_Smem']: T attendu; st['OP_TRN', 'OP_Smem']: TRN attendu;
 ↪  st['OP_lk', 'OP_Smem']: nombre: 'A'
S 219 ST src, Ymem | ST A, *AR4- | st['OP_T', 'OP_Smem']: T attendu; st['OP_TRN', 'OP_Smem']: TRN attendu;
 ↪  st['OP_lk', 'OP_Smem']: nombre: 'A'
S ⟨1⟩ ST src, Ymem | ST A, *AR⟨2⟩+ | st['OP_T', 'OP_Smem']: T attendu; st['OP_TRN', 'OP_Smem']: TRN attendu;
 ↪  st['OP_lk', 'OP_Smem']: nombre: 'A'  ×4
    ⟨⟩ = (220,4) (221,4) (222,4) (223,3)
S 224 ST src, Ymem | ST A, *AR3- | st['OP_T', 'OP_Smem']: T attendu; st['OP_TRN', 'OP_Smem']: TRN attendu;
 ↪  st['OP_lk', 'OP_Smem']: nombre: 'A'
T 225 STRCD Xmem, cond | STRCD *AR5-, AGT
W 9c76
B A 70ffffff
B AR5 202
B T 4321
B M 0202 1234
A A 70ffffff
A AR5 201
A T 4321
A M 0202 4321
E
S 226 6: Class 7 (see page 3-12) | SUB *AR1+, 14, A | sub['OP_SRC', 'OPT|OP_SHIFT', 'OPT|OP_DST']: src attendu,
 ↪  '*AR1+'; sub['OP_SRC', 'OP_ASM', 'OPT|OP_DST']: src attendu, '
T 227 6: Class 7 (see page 3-12) | SUB A, -8, B
W f538
B A 1200
B B 1800
B SXM 1
```

```
A A 1200
A B 17ee
A C 1
A SXM 1
E
T 228 6: Class 7 (see page 3-12) | SUB #12345, 8, A, B
W f118 3039
B A 1200
B B 1800
B SXM 1
A A 1200
A B ffffcfd900
A C 0
A SXM 1
E
T 229 SUBB Smem, src | SUBB 5, A
W 0e05
B A 6
B C 0
B DP 8
B M 0405 0006
A A fffffffff
A C 0
A DP 8
A M 0405 0006
E
T 230 SUBB Smem, src | SUBB *AR1+, B
W 0f91
B AR1 405
B B ff80000006
B C 1
B OVM 1
B M 0405 0006
A AR1 406
A B ff80000000
A C 1
A OVM 1
A M 0405 0006
E
T 231 SUBC Smem, src | SUBC 2, A
W 1e02
B A 4
B DP 6
B M 0302 0001
A A 8
A C 0
A DP 6
A M 0302 0001
E
T 232 SUBC Smem, src | RPT #15 ; SUBC *AR1, B
W ec0f 1f81
B AR1 1000
B B 41
B M 1000 0007
A AR1 1000
A B 20009
A C 1
A M 1000 0007
E
T 233 SUBS Smem, src | SUBS *AR2–, B
W 0b8a
B AR2 100
B B 2
B M 0100 f006
A AR2 ff
A B ffffff0ffc
```

```
A C 0
A M 0100 f006
E
T 234 TRAP K | TRAP 10h
W f4d0
B PC 1233
B SP 3ff
B M 03fe 9653
A PC ffc0
A SP 3fe
A M 03fe 1234
E
T 235 WRITA Smem | WRITA 5
W 7f05
B A 257
B DP 32
B M 1005 4339
B P 0257 0306
A A 257
A DP 32
A M 1005 4339
E
T 236 n       Opcode N | XC 1, ALEQ ; MAR *AR1+
W fd47 6d91
N 2
B A fffffffff
B AR1 32
A A fffffffff
A AR1 33
E
T 237 4: Class 1 (see page 3-3) | XOR *AR3+, A
W 1c93
B A ff1200
B AR3 100
B M 0100 1500
A A ff0700
A AR3 101
A M 0100 1500
E
T 238 4: Class 1 (see page 3-3) | XOR A, +3, B
W f1c3
B A 1200
B B 1800
A A 1200
A B 8800
E
T 239 XORM #lk, Smem | XORM 0404h, *AR4–
W 6a8c 0404
B AR4 100
B M 0100 4444
A AR4 ff
A M 0100 4040
# ligne registre incomplete: C†
# ligne registre incomplete: TC†
# ligne registre incomplete: TC = 1
# ligne registre incomplete: TC = 0
# ligne registre incomplete: ST0 and ST1 contain the status of various conditions and modes; PMST con-
# ligne registre incomplete: ARP                          Auxiliary register pointer
# ligne registre incomplete: ASM                          Accumulator shift mode
# ligne registre incomplete: BRAF                         Block repeat active flag
# ligne registre incomplete: C                            Carry
# ligne registre incomplete: CMPT                         Compatibility mode
# ligne registre incomplete: CPL                          Compiler mode
# ligne registre incomplete: C16                          Dual 16-bit/double-precision arithmetic mode
# ligne registre incomplete: DP                           Data page pointer
# ligne registre incomplete: FRCT                         Fractional mode
```

```
# ligne registre incomplete: HM                                    Hold mode
# ligne registre incomplete: INTM                                  Interrupt mode
# ligne registre incomplete: OVA                                   Overflow flag A
# ligne registre incomplete: OVB                                   Overflow flag B
# ligne registre incomplete: OVM                                   Overflow mode
# ligne registre incomplete: SXM                                   Sign-extension mode
# ligne registre incomplete: TC                                    Test/control flag
# ligne registre incomplete: XF                                    External flag status
# ligne registre incomplete: ARP               TC       C       OVA     OVB                                DP
# ligne registre incomplete: BRAF      CPL        XF       HM      INTM        0      OVM        SXM       C16
 ↪  FRCT     CMPT          ASM
# ligne registre incomplete: A
# ligne registre incomplete: B
# ligne registre incomplete: C
# ligne registre incomplete: a single instruction is executed.
# ligne registre incomplete: T
# ligne registre incomplete: XF pin.
# ligne registre incomplete: a 0.
# ligne registre incomplete: a 0.
# ligne registre incomplete: A                                                 subtract instructions 2-3 to 2-4
# ligne registre incomplete: B instruction 4-14
# ligne registre incomplete: C                                               DELAY instruction 4-41
# ligne registre incomplete: C address bus (CAB) C-3                          direct memory address, definition C-4
# ligne registre incomplete: C bus (CB), definition C-3                        DLD instruction 4-42
# ligne registre incomplete: C16 C-3                                         double (32-bit operand) instructions 2-6
# ligne registre incomplete: T                                             zero detect bit A (ZA), definition C-10
E
T 301 [banc 2026-10-16] ADD src, ASM, dst | ADD B, ASM, A ; codeur FR ROM PDROM 0xeac1 (RPE, xM << 6-exp)
W f680
B A 0
B B d990000
B ASM 3
A A 6cc80000
A B d990000
A ASM 3
E
T 302 [banc 2026-10-16] SUB src, ASM, dst | SUB B, ASM, A
W f681
B A 0
B B d990000
B ASM 3
A A ff93380000
A B d990000
A ASM 3
E
```

2.12 /opt/GSM/c54x_exe/tools/cch_ref/cch_deinterleave_inverse_ref.csv

7138 octets, 457 lignes → 80 lignes (1 groupes compactés)

```
iB_index,burst_B,pos_j,source_coded_k,lane_G
⟨1⟩,⟨2⟩,⟨3⟩,⟨4⟩,⟨5⟩  ×456
    ⟨⟩ = (0,0,0,0,0) (1,0,1,228,0) (2,0,2,64,0) (3,0,3,292,0) (4,0,4,128,0) (5,0,5,356,0) (6,0,6,192,0)
         (7,0,7,420,0) (8,0,8,256,0) (9,0,9,28,0) (10,0,10,320,0) (11,0,11,92,0) (12,0,12,384,0)
         (13,0,13,156,0) (14,0,14,448,0) (15,0,15,220,0) (16,0,16,56,0) (17,0,17,284,0) (18,0,18,120,0)
         (19,0,19,348,0) (20,0,20,184,0) (21,0,21,412,0) (22,0,22,248,0) (23,0,23,20,0) (24,0,24,312,0)
         (25,0,25,84,0) (26,0,26,376,0) (27,0,27,148,0) (28,0,28,440,0) (29,0,29,212,0) (30,0,30,48,0)
         (31,0,31,276,0) (32,0,32,112,0) (33,0,33,340,0) (34,0,34,176,0) (35,0,35,404,0) (36,0,36,240,0)
         (37,0,37,12,0) (38,0,38,304,0) (39,0,39,76,0) (40,0,40,368,0) (41,0,41,140,0) (42,0,42,432,0)
         (43,0,43,204,0) (44,0,44,40,0) (45,0,45,268,0) (46,0,46,104,0) (47,0,47,332,0) (48,0,48,168,0)
         (49,0,49,396,0) (50,0,50,232,0) (51,0,51,4,0) (52,0,52,296,0) (53,0,53,68,0) (54,0,54,360,0)
         (55,0,55,132,0) (56,0,56,424,0) (57,0,57,196,0) (58,0,58,32,0) (59,0,59,260,0) (60,0,60,96,0)
         (61,0,61,324,0) (62,0,62,160,0) (63,0,63,388,0) (64,0,64,224,0) (65,0,65,452,0) (66,0,66,288,0)
         (67,0,67,60,0) (68,0,68,352,0) (69,0,69,124,0) (70,0,70,416,0) (71,0,71,188,0) (72,0,72,24,0)
```

(73,0,73,252,0) (74,0,74,88,0) (75,0,75,316,0) (76,0,76,152,0) (77,0,77,380,0) (78,0,78,216,0)
(79,0,79,444,0) (80,0,80,280,0) (81,0,81,52,0) (82,0,82,344,0) (83,0,83,116,0) (84,0,84,408,0)
(85,0,85,180,0) (86,0,86,16,0) (87,0,87,244,0) (88,0,88,80,0) (89,0,89,308,0) (90,0,90,144,0)
(91,0,91,372,0) (92,0,92,208,0) (93,0,93,436,0) (94,0,94,272,0) (95,0,95,44,0) (96,0,96,336,0)
(97,0,97,108,0) (98,0,98,400,0) (99,0,99,172,0) (100,0,100,8,0) (101,0,101,236,0) (102,0,102,72,0)
(103,0,103,300,0) (104,0,104,136,0) (105,0,105,364,0) (106,0,106,200,0) (107,0,107,428,0)
(108,0,108,264,0) (109,0,109,36,0) (110,0,110,328,0) (111,0,111,100,0) (112,0,112,392,0)
(113,0,113,164,0) (114,1,0,57,1) (115,1,1,285,1) (116,1,2,121,1) (117,1,3,349,1) (118,1,4,185,1)
(119,1,5,413,1) (120,1,6,249,1) (121,1,7,21,1) (122,1,8,313,1) (123,1,9,85,1) (124,1,10,377,1)
(125,1,11,149,1) (126,1,12,441,1) (127,1,13,213,1) (128,1,14,49,1) (129,1,15,277,1) (130,1,16,113,1)
(131,1,17,341,1) (132,1,18,177,1) (133,1,19,405,1) (134,1,20,241,1) (135,1,21,13,1) (136,1,22,305,1)
(137,1,23,77,1) (138,1,24,369,1) (139,1,25,141,1) (140,1,26,433,1) (141,1,27,205,1) (142,1,28,41,1)
(143,1,29,269,1) (144,1,30,105,1) (145,1,31,333,1) (146,1,32,169,1) (147,1,33,397,1) (148,1,34,233,1)
(149,1,35,5,1) (150,1,36,297,1) (151,1,37,69,1) (152,1,38,361,1) (153,1,39,133,1) (154,1,40,425,1)
(155,1,41,197,1) (156,1,42,33,1) (157,1,43,261,1) (158,1,44,97,1) (159,1,45,325,1) (160,1,46,161,1)
(161,1,47,389,1) (162,1,48,225,1) (163,1,49,453,1) (164,1,50,289,1) (165,1,51,61,1) (166,1,52,353,1)
(167,1,53,125,1) (168,1,54,417,1) (169,1,55,189,1) (170,1,56,25,1) (171,1,57,253,1) (172,1,58,89,1)
(173,1,59,317,1) (174,1,60,153,1) (175,1,61,381,1) (176,1,62,217,1) (177,1,63,445,1) (178,1,64,281,1)
(179,1,65,53,1) (180,1,66,345,1) (181,1,67,117,1) (182,1,68,409,1) (183,1,69,181,1) (184,1,70,17,1)
(185,1,71,245,1) (186,1,72,81,1) (187,1,73,309,1) (188,1,74,145,1) (189,1,75,373,1) (190,1,76,209,1)
(191,1,77,437,1) (192,1,78,273,1) (193,1,79,45,1) (194,1,80,337,1) (195,1,81,109,1) (196,1,82,401,1)
(197,1,83,173,1) (198,1,84,9,1) (199,1,85,237,1) (200,1,86,73,1) (201,1,87,301,1) (202,1,88,137,1)
(203,1,89,365,1) (204,1,90,201,1) (205,1,91,429,1) (206,1,92,265,1) (207,1,93,37,1) (208,1,94,329,1)
(209,1,95,101,1) (210,1,96,393,1) (211,1,97,165,1) (212,1,98,1,1) (213,1,99,229,1) (214,1,100,65,1)
(215,1,101,293,1) (216,1,102,129,1) (217,1,103,357,1) (218,1,104,193,1) (219,1,105,421,1)
(220,1,106,257,1) (221,1,107,29,1) (222,1,108,321,1) (223,1,109,93,1) (224,1,110,385,1)
(225,1,111,157,1) (226,1,112,449,1) (227,1,113,221,1) (228,2,0,114,0) (229,2,1,342,0) (230,2,2,178,0)
(231,2,3,406,0) (232,2,4,242,0) (233,2,5,14,0) (234,2,6,306,0) (235,2,7,78,0) (236,2,8,370,0)
(237,2,9,142,0) (238,2,10,434,0) (239,2,11,206,0) (240,2,12,42,0) (241,2,13,270,0) (242,2,14,106,0)
(243,2,15,334,0) (244,2,16,170,0) (245,2,17,398,0) (246,2,18,234,0) (247,2,19,6,0) (248,2,20,298,0)
(249,2,21,70,0) (250,2,22,362,0) (251,2,23,134,0) (252,2,24,426,0) (253,2,25,198,0) (254,2,26,34,0)
(255,2,27,262,0) (256,2,28,98,0) (257,2,29,326,0) (258,2,30,162,0) (259,2,31,390,0) (260,2,32,226,0)
(261,2,33,454,0) (262,2,34,290,0) (263,2,35,62,0) (264,2,36,354,0) (265,2,37,126,0) (266,2,38,418,0)
(267,2,39,190,0) (268,2,40,26,0) (269,2,41,254,0) (270,2,42,90,0) (271,2,43,318,0) (272,2,44,154,0)
(273,2,45,382,0) (274,2,46,218,0) (275,2,47,446,0) (276,2,48,282,0) (277,2,49,54,0) (278,2,50,346,0)
(279,2,51,118,0) (280,2,52,410,0) (281,2,53,182,0) (282,2,54,18,0) (283,2,55,246,0) (284,2,56,82,0)
(285,2,57,310,0) (286,2,58,146,0) (287,2,59,374,0) (288,2,60,210,0) (289,2,61,438,0) (290,2,62,274,0)
(291,2,63,46,0) (292,2,64,338,0) (293,2,65,110,0) (294,2,66,402,0) (295,2,67,174,0) (296,2,68,10,0)
(297,2,69,238,0) (298,2,70,74,0) (299,2,71,302,0) (300,2,72,138,0) (301,2,73,366,0) (302,2,74,202,0)
(303,2,75,430,0) (304,2,76,266,0) (305,2,77,38,0) (306,2,78,330,0) (307,2,79,102,0) (308,2,80,394,0)
(309,2,81,166,0) (310,2,82,2,0) (311,2,83,230,0) (312,2,84,66,0) (313,2,85,294,0) (314,2,86,130,0)
(315,2,87,358,0) (316,2,88,194,0) (317,2,89,422,0) (318,2,90,258,0) (319,2,91,30,0) (320,2,92,322,0)
(321,2,93,94,0) (322,2,94,386,0) (323,2,95,158,0) (324,2,96,450,0) (325,2,97,222,0) (326,2,98,58,0)
(327,2,99,286,0) (328,2,100,122,0) (329,2,101,350,0) (330,2,102,186,0) (331,2,103,414,0)
(332,2,104,250,0) (333,2,105,22,0) (334,2,106,314,0) (335,2,107,86,0) (336,2,108,378,0)
(337,2,109,150,0) (338,2,110,442,0) (339,2,111,214,0) (340,2,112,50,0) (341,2,113,278,0)
(342,3,0,171,1) (343,3,1,399,1) (344,3,2,235,1) (345,3,3,7,1) (346,3,4,299,1) (347,3,5,71,1)
(348,3,6,363,1) (349,3,7,135,1) (350,3,8,427,1) (351,3,9,199,1) (352,3,10,35,1) (353,3,11,263,1)
(354,3,12,99,1) (355,3,13,327,1) (356,3,14,163,1) (357,3,15,391,1) (358,3,16,227,1) (359,3,17,455,1)
(360,3,18,291,1) (361,3,19,63,1) (362,3,20,355,1) (363,3,21,127,1) (364,3,22,419,1) (365,3,23,191,1)
(366,3,24,27,1) (367,3,25,255,1) (368,3,26,91,1) (369,3,27,319,1) (370,3,28,155,1) (371,3,29,383,1)
(372,3,30,219,1) (373,3,31,447,1) (374,3,32,283,1) (375,3,33,55,1) (376,3,34,347,1) (377,3,35,119,1)
(378,3,36,411,1) (379,3,37,183,1) (380,3,38,19,1) (381,3,39,247,1) (382,3,40,83,1) (383,3,41,311,1)
(384,3,42,147,1) (385,3,43,375,1) (386,3,44,211,1) (387,3,45,439,1) (388,3,46,275,1) (389,3,47,47,1)
(390,3,48,339,1) (391,3,49,111,1) (392,3,50,403,1) (393,3,51,175,1) (394,3,52,11,1) (395,3,53,239,1)
(396,3,54,75,1) (397,3,55,303,1) (398,3,56,139,1) (399,3,57,367,1) (400,3,58,203,1) (401,3,59,431,1)
(402,3,60,267,1) (403,3,61,39,1) (404,3,62,331,1) (405,3,63,103,1) (406,3,64,395,1) (407,3,65,167,1)
(408,3,66,3,1) (409,3,67,231,1) (410,3,68,67,1) (411,3,69,295,1) (412,3,70,131,1) (413,3,71,359,1)
(414,3,72,195,1) (415,3,73,423,1) (416,3,74,259,1) (417,3,75,31,1) (418,3,76,323,1) (419,3,77,95,1)
(420,3,78,387,1) (421,3,79,159,1) (422,3,80,451,1) (423,3,81,223,1) (424,3,82,59,1) (425,3,83,287,1)
(426,3,84,123,1) (427,3,85,351,1) (428,3,86,187,1) (429,3,87,415,1) (430,3,88,251,1) (431,3,89,23,1)
(432,3,90,315,1) (433,3,91,87,1) (434,3,92,379,1) (435,3,93,151,1) (436,3,94,443,1) (437,3,95,215,1)
(438,3,96,51,1) (439,3,97,279,1) (440,3,98,115,1) (441,3,99,343,1) (442,3,100,179,1) (443,3,101,407,1)
(444,3,102,243,1) (445,3,103,15,1) (446,3,104,307,1) (447,3,105,79,1) (448,3,106,371,1)
(449,3,107,143,1) (450,3,108,435,1) (451,3,109,207,1) (452,3,110,43,1) (453,3,111,271,1)
(454,3,112,107,1) (455,3,113,335,1)

2.13 /opt/GSM/c54x_exe/tools/cch_ref/cch_interleave_ref.csv

11686 octets, 457 lignes → 143 lignes (1 groupes compactés)

```
k,"lane_G(0=G0/c0,1=G1/c1)",burst_B,pos_j,iB_index,j_if_LSB_dropped,j_if_LSB_inverted,iB_index_if_LSB_inverted
⟨1⟩,⟨2⟩,⟨3⟩,⟨4⟩,⟨5⟩,⟨6⟩,⟨7⟩,⟨8⟩  ×456
    ⟨⟩ = (0,0,0,0,0,0,1,1) (1,1,1,98,212,98,99,213) (2,0,2,82,310,82,83,311) (3,1,3,66,408,66,67,409)
         (4,0,0,51,51,50,50,50) (5,1,1,35,149,34,34,148) (6,0,2,19,247,18,18,246) (7,1,3,3,345,2,2,344)
         (8,0,0,100,100,100,101,101) (9,1,1,84,198,84,85,199) (10,0,2,68,296,68,69,297)
         (11,1,3,52,394,52,53,395) (12,0,0,37,37,36,36,36) (13,1,1,21,135,20,20,134) (14,0,2,5,233,4,4,232)
         (15,1,3,103,445,102,102,444) (16,0,0,86,86,86,87,87) (17,1,1,70,184,70,71,185)
         (18,0,2,54,282,54,55,283) (19,1,3,38,380,38,39,381) (20,0,0,23,23,22,22,22) (21,1,1,7,121,6,6,120)
         (22,0,2,105,333,104,104,332) (23,1,3,89,431,88,88,430) (24,0,0,72,72,72,73,73)
         (25,1,1,56,170,56,57,171) (26,0,2,40,268,40,41,269) (27,1,3,24,366,24,25,367) (28,0,0,9,9,8,8,8)
         (29,1,1,107,221,106,106,220) (30,0,2,91,319,90,90,318) (31,1,3,75,417,74,74,416)
         (32,0,0,58,58,58,59,59) (33,1,1,42,156,42,43,157) (34,0,2,26,254,26,27,255) (35,1,3,10,352,10,11,353)
         (36,0,0,109,109,108,108,108) (37,1,1,93,207,92,92,206) (38,0,2,77,305,76,76,304)
         (39,1,3,61,403,60,60,402) (40,0,0,44,44,44,45,45) (41,1,1,28,142,28,29,143) (42,0,2,12,240,12,13,241)
         (43,1,3,110,452,110,111,453) (44,0,0,95,95,94,94,94) (45,1,1,79,193,78,78,192)
         (46,0,2,63,291,62,62,290) (47,1,3,47,389,46,46,388) (48,0,0,30,30,30,31,31) (49,1,1,14,128,14,15,129)
         (50,0,2,112,340,112,113,341) (51,1,3,96,438,96,97,439) (52,0,0,81,81,80,80,80)
         (53,1,1,65,179,64,64,178) (54,0,2,49,277,48,48,276) (55,1,3,33,375,32,32,374) (56,0,0,16,16,16,17,17)
         (57,1,1,0,114,0,1,115) (58,0,2,98,326,98,99,327) (59,1,3,82,424,82,83,425) (60,0,0,67,67,66,66,66)
         (61,1,1,51,165,50,50,164) (62,0,2,35,263,34,34,262) (63,1,3,19,361,18,18,360) (64,0,0,2,2,2,3,3)
         (65,1,1,100,214,100,101,215) (66,0,2,84,312,84,85,313) (67,1,3,68,410,68,69,411)
         (68,0,0,53,53,52,52,52) (69,1,1,37,151,36,36,150) (70,0,2,21,249,20,20,248) (71,1,3,5,347,4,4,346)
         (72,0,0,102,102,102,103,103) (73,1,1,86,200,86,87,201) (74,0,2,70,298,70,71,299)
         (75,1,3,54,396,54,55,397) (76,0,0,39,39,38,38,38) (77,1,1,23,137,22,22,136) (78,0,2,7,235,6,6,234)
         (79,1,3,105,447,104,104,446) (80,0,0,88,88,88,89,89) (81,1,1,72,186,72,73,187)
         (82,0,2,56,284,56,57,285) (83,1,3,40,382,40,41,383) (84,0,0,25,25,24,24,24) (85,1,1,9,123,8,8,122)
         (86,0,2,107,335,106,106,334) (87,1,3,91,433,90,90,432) (88,0,0,74,74,74,75,75)
         (89,1,1,58,172,58,59,173) (90,0,2,42,270,42,43,271) (91,1,3,26,368,26,27,369) (92,0,0,11,11,10,10,10)
         (93,1,1,109,223,108,108,222) (94,0,2,93,321,92,92,320) (95,1,3,77,419,76,76,418)
         (96,0,0,60,60,60,61,61) (97,1,1,44,158,44,45,159) (98,0,2,28,256,28,29,257) (99,1,3,12,354,12,13,355)
         (100,0,0,111,111,110,110,110) (101,1,1,95,209,94,94,208) (102,0,2,79,307,78,78,306)
         (103,1,3,63,405,62,62,404) (104,0,0,46,46,46,47,47) (105,1,1,30,144,30,31,145)
         (106,0,2,14,242,14,15,243) (107,1,3,112,454,112,113,455) (108,0,0,97,97,96,96,96)
         (109,1,1,81,195,80,80,194) (110,0,2,65,293,64,64,292) (111,1,3,49,391,48,48,390)
         (112,0,0,32,32,32,33,33) (113,1,1,16,130,16,17,131) (114,0,2,0,228,0,1,229) (115,1,3,98,440,98,99,441)
         (116,0,0,83,83,82,82,82) (117,1,1,67,181,66,66,180) (118,0,2,51,279,50,50,278)
         (119,1,3,35,377,34,34,376) (120,0,0,18,18,18,19,19) (121,1,1,2,116,2,3,117)
         (122,0,2,100,328,100,101,329) (123,1,3,84,426,84,85,427) (124,0,0,69,69,68,68,68)
         (125,1,1,53,167,52,52,166) (126,0,2,37,265,36,36,264) (127,1,3,21,363,20,20,362) (128,0,0,4,4,4,5,5)
         (129,1,1,102,216,102,103,217) (130,0,2,86,314,86,87,315) (131,1,3,70,412,70,71,413)
         (132,0,0,55,55,54,54,54) (133,1,1,39,153,38,38,152) (134,0,2,23,251,22,22,250) (135,1,3,7,349,6,6,348)
         (136,0,0,104,104,104,105,105) (137,1,1,88,202,88,89,203) (138,0,2,72,300,72,73,301)
         (139,1,3,56,398,56,57,399) (140,0,0,41,41,40,40,40) (141,1,1,25,139,24,24,138) (142,0,2,9,237,8,8,236)
         (143,1,3,107,449,106,106,448) (144,0,0,90,90,90,91,91) (145,1,1,74,188,74,75,189)
         (146,0,2,58,286,58,59,287) (147,1,3,42,384,42,43,385) (148,0,0,27,27,26,26,26)
         (149,1,1,11,125,10,10,124) (150,0,2,109,337,108,108,336) (151,1,3,93,435,92,92,434)
         (152,0,0,76,76,76,77,77) (153,1,1,60,174,60,61,175) (154,0,2,44,272,44,45,273)
         (155,1,3,28,370,28,29,371) (156,0,0,13,13,12,12,12) (157,1,1,111,225,110,110,224)
         (158,0,2,95,323,94,94,322) (159,1,3,79,421,78,78,420) (160,0,0,62,62,62,63,63)
         (161,1,1,46,160,46,47,161) (162,0,2,30,258,30,31,259) (163,1,3,14,356,14,15,357)
         (164,0,0,113,113,112,112,112) (165,1,1,97,211,96,96,210) (166,0,2,81,309,80,80,308)
         (167,1,3,65,407,64,64,406) (168,0,0,48,48,48,49,49) (169,1,1,32,146,32,33,147)
         (170,0,2,16,244,16,17,245) (171,1,3,0,342,0,1,343) (172,0,0,99,99,98,98,98) (173,1,1,83,197,82,82,196)
         (174,0,2,67,295,66,66,294) (175,1,3,51,393,50,50,392) (176,0,0,34,34,34,35,35)
         (177,1,1,18,132,18,19,133) (178,0,2,2,230,2,3,231) (179,1,3,100,442,100,101,443)
         (180,0,0,85,85,84,84,84) (181,1,1,69,183,68,68,182) (182,0,2,53,281,52,52,280)
         (183,1,3,37,379,36,36,378) (184,0,0,20,20,20,21,21) (185,1,1,4,118,4,5,119)
         (186,0,2,102,330,102,103,331) (187,1,3,86,428,86,87,429) (188,0,0,71,71,70,70,70)
         (189,1,1,55,169,54,54,168) (190,0,2,39,267,38,38,266) (191,1,3,23,365,22,22,364) (192,0,0,6,6,6,7,7)
         (193,1,1,104,218,104,105,219) (194,0,2,88,316,88,89,317) (195,1,3,72,414,72,73,415)
         (196,0,0,57,57,56,56,56) (197,1,1,41,155,40,40,154) (198,0,2,25,253,24,24,252) (199,1,3,9,351,8,8,350)
         (200,0,0,106,106,106,107,107) (201,1,1,90,204,90,91,205) (202,0,2,74,302,74,75,303)
```

(203,1,3,58,400,58,59,401) (204,0,0,43,43,42,42,42) (205,1,1,27,141,26,26,140)
(206,0,2,11,239,10,10,238) (207,1,3,109,451,108,108,450) (208,0,0,92,92,92,93,93)
(209,1,1,76,190,76,77,191) (210,0,2,60,288,60,61,289) (211,1,3,44,386,44,45,387)
(212,0,0,29,29,28,28,28) (213,1,1,13,127,12,12,126) (214,0,2,111,339,110,110,338)
(215,1,3,95,437,94,94,436) (216,0,0,78,78,78,79,79) (217,1,1,62,176,62,63,177)
(218,0,2,46,274,46,47,275) (219,1,3,30,372,30,31,373) (220,0,0,15,15,14,14,14)
(221,1,1,113,227,112,112,226) (222,0,2,97,325,96,96,324) (223,1,3,81,423,80,80,422)
(224,0,0,64,64,64,65,65) (225,1,1,48,162,48,49,163) (226,0,2,32,260,32,33,261)
(227,1,3,16,358,16,17,359) (228,0,0,1,1,0,0,0) (229,1,1,99,213,98,98,212) (230,0,2,83,311,82,82,310)
(231,1,3,67,409,66,66,408) (232,0,0,50,50,50,51,51) (233,1,1,34,148,34,35,149)
(234,0,2,18,246,18,19,247) (235,1,3,2,344,2,3,345) (236,0,0,101,101,100,100,100)
(237,1,1,85,199,84,84,198) (238,0,2,69,297,68,68,296) (239,1,3,53,395,52,52,394)
(240,0,0,36,36,36,37,37) (241,1,1,20,134,20,21,135) (242,0,2,4,232,4,5,233)
(243,1,3,102,444,102,103,445) (244,0,0,87,87,86,86,86) (245,1,1,71,185,70,70,184)
(246,0,2,55,283,54,54,282) (247,1,3,39,381,38,38,380) (248,0,0,22,22,22,23,23) (249,1,1,6,120,6,7,121)
(250,0,2,104,332,104,105,333) (251,1,3,88,430,88,89,431) (252,0,0,73,73,72,72,72)
(253,1,1,57,171,56,56,170) (254,0,2,41,269,40,40,268) (255,1,3,25,367,24,24,366) (256,0,0,8,8,8,9,9)
(257,1,1,106,220,106,107,221) (258,0,2,90,318,90,91,319) (259,1,3,74,416,74,75,417)
(260,0,0,59,59,58,58,58) (261,1,1,43,157,42,42,156) (262,0,2,27,255,26,26,254)
(263,1,3,11,353,10,10,352) (264,0,0,108,108,108,109,109) (265,1,1,92,206,92,93,207)
(266,0,2,76,304,76,77,305) (267,1,3,60,402,60,61,403) (268,0,0,45,45,44,44,44)
(269,1,1,29,143,28,28,142) (270,0,2,13,241,12,12,240) (271,1,3,111,453,110,110,452)
(272,0,0,94,94,94,95,95) (273,1,1,78,192,78,79,193) (274,0,2,62,290,62,63,291)
(275,1,3,46,388,46,47,389) (276,0,0,31,31,30,30,30) (277,1,1,15,129,14,14,128)
(278,0,2,113,341,112,112,340) (279,1,3,97,439,96,96,438) (280,0,0,80,80,80,81,81)
(281,1,1,64,178,64,65,179) (282,0,2,48,276,48,49,277) (283,1,3,32,374,32,33,375)
(284,0,0,17,17,16,16,16) (285,1,1,1,115,0,0,114) (286,0,2,99,327,98,98,326) (287,1,3,83,425,82,82,424)
(288,0,0,66,66,66,67,67) (289,1,1,50,164,50,51,165) (290,0,2,34,262,34,35,263)
(291,1,3,18,360,18,19,361) (292,0,0,3,3,2,2,2) (293,1,1,101,215,100,100,214)
(294,0,2,85,313,84,84,312) (295,1,3,69,411,68,68,410) (296,0,0,52,52,52,53,53)
(297,1,1,36,150,36,37,151) (298,0,2,20,248,20,21,249) (299,1,3,4,346,4,5,347)
(300,0,0,103,103,102,102,102) (301,1,1,87,201,86,86,200) (302,0,2,71,299,70,70,298)
(303,1,3,55,397,54,54,396) (304,0,0,38,38,38,39,39) (305,1,1,22,136,22,23,137) (306,0,2,6,234,6,7,235)
(307,1,3,104,446,104,105,447) (308,0,0,89,89,88,88,88) (309,1,1,73,187,72,72,186)
(310,0,2,57,285,56,56,284) (311,1,3,41,383,40,40,382) (312,0,0,24,24,24,25,25) (313,1,1,8,122,8,9,123)
(314,0,2,106,334,106,107,335) (315,1,3,90,432,90,91,433) (316,0,0,75,75,74,74,74)
(317,1,1,59,173,58,58,172) (318,0,2,43,271,42,42,270) (319,1,3,27,369,26,26,368)
(320,0,0,10,10,10,11,11) (321,1,1,108,222,108,109,223) (322,0,2,92,320,92,93,321)
(323,1,3,76,418,76,77,419) (324,0,0,61,61,60,60,60) (325,1,1,45,159,44,44,158)
(326,0,2,29,257,28,28,256) (327,1,3,13,355,12,12,354) (328,0,0,110,110,110,111,111)
(329,1,1,94,208,94,95,209) (330,0,2,78,306,78,79,307) (331,1,3,62,404,62,63,405)
(332,0,0,47,47,46,46,46) (333,1,1,31,145,30,30,144) (334,0,2,15,243,14,14,242)
(335,1,3,113,455,112,112,454) (336,0,0,96,96,96,97,97) (337,1,1,80,194,80,81,195)
(338,0,2,64,292,64,65,293) (339,1,3,48,390,48,49,391) (340,0,0,33,33,32,32,32)
(341,1,1,17,131,16,16,130) (342,0,2,1,229,0,0,228) (343,1,3,99,441,98,98,440) (344,0,0,82,82,82,83,83)
(345,1,1,66,180,66,67,181) (346,0,2,50,278,50,51,279) (347,1,3,34,376,34,35,377)
(348,0,0,19,19,18,18,18) (349,1,1,3,117,2,2,116) (350,0,2,101,329,100,100,328)
(351,1,3,85,427,84,84,426) (352,0,0,68,68,68,69,69) (353,1,1,52,166,52,53,167)
(354,0,2,36,264,36,37,265) (355,1,3,20,362,20,21,363) (356,0,0,5,5,4,4,4)
(357,1,1,103,217,102,102,216) (358,0,2,87,315,86,86,314) (359,1,3,71,413,70,70,412)
(360,0,0,54,54,54,55,55) (361,1,1,38,152,38,39,153) (362,0,2,22,250,22,23,251) (363,1,3,6,348,6,7,349)
(364,0,0,105,105,104,104,104) (365,1,1,89,203,88,88,202) (366,0,2,73,301,72,72,300)
(367,1,3,57,399,56,56,398) (368,0,0,40,40,40,41,41) (369,1,1,24,138,24,25,139) (370,0,2,8,236,8,9,237)
(371,1,3,106,448,106,107,449) (372,0,0,91,91,90,90,90) (373,1,1,75,189,74,74,188)
(374,0,2,59,287,58,58,286) (375,1,3,43,385,42,42,384) (376,0,0,26,26,26,27,27)
(377,1,1,10,124,10,11,125) (378,0,2,108,336,108,109,337) (379,1,3,92,434,92,93,435)
(380,0,0,77,77,76,76,76) (381,1,1,61,175,60,60,174) (382,0,2,45,273,44,44,272)
(383,1,3,29,371,28,28,370) (384,0,0,12,12,12,13,13) (385,1,1,110,224,110,111,225)
(386,0,2,94,322,94,95,323) (387,1,3,78,420,78,79,421) (388,0,0,63,63,62,62,62)
(389,1,1,47,161,46,46,160) (390,0,2,31,259,30,30,258) (391,1,3,15,357,14,14,356)
(392,0,0,112,112,112,113,113) (393,1,1,96,210,96,97,211) (394,0,2,80,308,80,81,309)
(395,1,3,64,406,64,65,407) (396,0,0,49,49,48,48,48) (397,1,1,33,147,32,32,146)
(398,0,2,17,245,16,16,244) (399,1,3,1,343,0,0,342) (400,0,0,98,98,98,99,99) (401,1,1,82,196,82,83,197)
(402,0,2,66,294,66,67,295) (403,1,3,50,392,50,51,393) (404,0,0,35,35,34,34,34)
(405,1,1,19,133,18,18,132) (406,0,2,3,231,2,2,230) (407,1,3,101,443,100,100,442)
(408,0,0,84,84,84,85,85) (409,1,1,68,182,68,69,183) (410,0,2,52,280,52,53,281)

```
(411,1,3,36,378,36,37,379) (412,0,0,21,21,20,20,20) (413,1,1,5,119,4,4,118)
(414,0,2,103,331,102,102,330) (415,1,3,87,429,86,86,428) (416,0,0,70,70,70,71,71)
(417,1,1,54,168,54,55,169) (418,0,2,38,266,38,39,267) (419,1,3,22,364,22,23,365) (420,0,0,7,7,6,6,6)
(421,1,1,105,219,104,104,218) (422,0,2,89,317,88,88,316) (423,1,3,73,415,72,72,414)
(424,0,0,56,56,56,57,57) (425,1,1,40,154,40,41,155) (426,0,2,24,252,24,25,253) (427,1,3,8,350,8,9,351)
(428,0,0,107,107,106,106,106) (429,1,1,91,205,90,90,204) (430,0,2,75,303,74,74,302)
(431,1,3,59,401,58,58,400) (432,0,0,42,42,42,43,43) (433,1,1,26,140,26,27,141)
(434,0,2,10,238,10,11,239) (435,1,3,108,450,108,109,451) (436,0,0,93,93,92,92,92)
(437,1,1,77,191,76,76,190) (438,0,2,61,289,60,60,288) (439,1,3,45,387,44,44,386)
(440,0,0,28,28,28,29,29) (441,1,1,12,126,12,13,127) (442,0,2,110,338,110,111,339)
(443,1,3,94,436,94,95,437) (444,0,0,79,79,78,78,78) (445,1,1,63,177,62,62,176)
(446,0,2,47,275,46,46,274) (447,1,3,31,373,30,30,372) (448,0,0,14,14,14,15,15)
(449,1,1,112,226,112,113,227) (450,0,2,96,324,96,97,325) (451,1,3,80,422,80,81,423)
(452,0,0,65,65,64,64,64) (453,1,1,49,163,48,48,162) (454,0,2,33,261,32,32,260)
(455,1,3,17,359,16,16,358)
```

2.14 /opt/GSM/c54x_exe/tools/cch_ref/cch_testvectors.txt

2882 octets, 24 lignes → 26 lignes (1 groupes compactés)

```
# GSM 05.03 xCCH stage-isolation test vector
# Fixed pseudo-random 228-bit u (u[224:228]=tail=0).
# TEST A (interleaver): feed burst0..3 into YOUR deinterleaver,
#   compare the 456-bit result against the three candidates below:
#     == cB_correct       -> interleaver OK, bug is downstream
#     == cB_lsb_dropped   -> you never apply the ((k%8)>>2) term
#     == cB_lsb_inverted  -> that term is inverted / off-by-one
#     == none of them     -> different bug (burst order? half swap?)
# TEST B (Viterbi/lane): feed cB_correct into YOUR Viterbi,
#   compare the 228-bit result against u. Mismatch with cB correct
#   => c0/c1 swap, wrong polynomials, or tail handling.

u_228           =
 ↪  00101111001011011001000010100111010011010010111011110101101101001110101100000011111010010110111111011000001000010101001100010

burst(1)_114      = ⟨2⟩  ×4
    ⟨⟩ =

 ↪  (0,01011110011011111100000111101111010000101100100010011111101110010010000111100111011110001100001011001110001011100001)

 ↪  (1,11111011000011100110011000100111111110111000111000111001010111010011010111101101101001110000001011101010110101000011)

 ↪  (2,10100000011101000100000011100000100010000111100111101100011000011101000001110101100000111000010011001000011110010 0)

 ↪  (3,00111001110110100001010110010110101000010011010100111011111011111100100000000001100010101011101110000111010101010 0)

cB_correct      =
 ↪  00001101110101010100111011011000101011111000111111011110110001010001110001010001110010110110001001010111010001100010100001

cB_lsb_dropped  =
 ↪  00000101110100010100010111010100101010011000010011011011110001000001100101011100110010110110110001010001010010100010111101

cB_lsb_inverted =
 ↪  11000101110000011101010110100100111011001011101001000101111000100110010010001110001101011011111000100000101001010110011110 0
```

3 Fichiers

3.1 /opt/GSM/c54x_exe/.gitignore

373 octets, 18 lignes → 18 lignes

```
# build products (make) — never commit binaries
/c54x_exe
/isa_test
/dsp_tester
/tch_now
/tools/rejeu_banc
/tools/sacch_tf_decode
*.o
__pycache__/
*.pyc
# run artefacts
/tmp/
# ROM du DSP TI : jamais dans le depot, telechargee par rom/fetch-rom.sh (FreeCalypso)
/rom/calypso_dsp.*.bin
!/rom/calypso_dsp.Registers.bin
/rom/calypso_dsp.txt
/c54x_exe.*
/tools/layer1_tester
```

3.2 /opt/GSM/c54x_exe/Makefile

4973 octets, 102 lignes → 102 lignes

```
# c54x_exe - le DSP Calypso hors QEMU.
#
# Les sources viennent de /opt/GSM/qosmo et ne sont PAS recopiees ici.
QOSMO   ?= /opt/GSM/qosmo
CAL     := $(QOSMO)/hw/arm/calypso
L1DSP   := $(CAL)/l1-dsp
HORS    := $(QOSMO)/contrib/hors-qemu

CC      ?= gcc
CFLAGS  ?= -O3 -march=native -g -Wall -Werror=format -Werror=format-extra-args -Wno-unused-function
 ↪  -Wno-unused-variable \
          -Wno-unused-but-set-variable -Wno-sign-compare
CPPFLAGS := -D_GNU_SOURCE -I$(HORS)/doublures -I$(L1DSP) -I$(CAL) -I$(QOSMO)/include -I$(QOSMO)
# calypso_bsp.c encode les bursts RACH/NB : gsm0503_rach_ext_encode
OSMO    := $(shell pkg-config --cflags --libs libosmocoding libosmocore 2>/dev/null)
LDLIBS  := -lpthread -lm $(OSMO)

SRC := src/main.c src/rejouer.c src/pcb-minimal.c src/verbosite.c src/pont.c src/cellule.c src/montant.c
 ↪  src/tsp_tx.c $(HORS)/cales-qemu.c \
       $(L1DSP)/calypso_gmsk.c \
       $(L1DSP)/calypso_c54x.c \
       $(L1DSP)/c54x_exec.c \
       $(L1DSP)/c54x_decode.c \
       $(L1DSP)/c54x_mem.c \
       $(L1DSP)/c54x_irq.c \
       $(L1DSP)/c54x_probes.c \
       $(L1DSP)/calypso_bsp.c \
       $(L1DSP)/calypso_arm2dsp.c \
       $(L1DSP)/calypso_mailbox.c \
       $(L1DSP)/calypso_fbsb.c \
       $(L1DSP)/calypso_dma.c \
       $(L1DSP)/calypso_rhea_dma.c \
       $(L1DSP)/calypso_rif.c \
       $(L1DSP)/calypso_a5.c \
       $(L1DSP)/calypso_twl3025.c \
       $(L1DSP)/calypso_vsp.c \
       $(CAL)/calypso_xio.c \
       $(CAL)/calypso_iota.c \
       $(CAL)/calypso_trf6151.c \
       $(CAL)/calypso_debug.c \
       $(CAL)/calypso_invariants.c
```

```
all: c54x_exe

# [2026-09-23] Les en-tetes aussi : sans eux, une modification d un .h seul
# (calypso_c54x.h, calypso_bsp.h...) laissait un binaire perime.
HDR := $(wildcard src/*.h $(L1DSP)/*.h $(CAL)/*.h $(QOSMO)/include/hw/arm/calypso/*.h)

# [2026-09-23] RECOMPILATION A CHAQUE make. Les sources viennent d un autre
# depot (qosmo) et « make: Nothing to be done » laissait douter du binaire
# lance : c54x_exe est reconstruit a chaque appel, comme apres un make clean.
# Une seule commande cc, pas de .o : rien d autre a nettoyer.
.PHONY: c54x_exe
c54x_exe: $(SRC) $(HDR)
    $(CC) $(CFLAGS) $(CPPFLAGS) -o $@ $(SRC) $(LDLIBS)

# ISA conformance: the SPRU172C worked examples replayed on the core.
#   make isa_test && ./isa_test tools/isa_tests.txt 2>/dev/null
COEUR := $(filter-out src/%,$(SRC))
tools/isa_tests.txt: tools/isa_examples.py
    python3 tools/isa_examples.py > $@

# Rejeu hors banc d'un TCH enregistre (tools/rejeu_banc.c) : /dev/shm/calypso_rejeu_tch.bin
#   make tools/rejeu_banc && ./tools/rejeu_banc [fichier] [ticks max]
tools/rejeu_banc: tools/rejeu_banc.c src/tsp_tx.c src/pcb-minimal.c src/verbosite.c $(COEUR) $(HDR)
    $(CC) $(CFLAGS) $(CPPFLAGS) -Isrc -o $@ tools/rejeu_banc.c src/tsp_tx.c src/pcb-minimal.c src/verbosite.c
 ↪  $(COEUR) $(LDLIBS)

isa_test: tools/isa_test.c src/pcb-minimal.c $(COEUR) tools/isa_tests.txt
    $(CC) $(CFLAGS) $(CPPFLAGS) -o $@ tools/isa_test.c src/pcb-minimal.c $(COEUR) $(LDLIBS)

# layer1_tester : adresses et fonctions de l'interface layer1 <-> ROM, carte
# docs/carte-memoire.md. Ne reconstruit PAS c54x_exe. Le coeur est lie SANS
# calypso_mailbox.c : tools/layer1_tester.c fournit calypso_mbx_evt, le crochet
# de chaque acces DSP ; fopen/open/socket/sendto sont enveloppes (rien n'est
# ecrit hors du fichier de sortie, aucune socket).
#   make layer1_tester && python3 tools/layer1_tester.py [--rejeu FICHIER] [--md docs/carte-memoire.md]
#   make api_carte [REJEU=FICHIER]      (alias : regenere docs/carte-memoire.md)
L1T_COEUR := $(filter-out $(L1DSP)/calypso_mailbox.c,$(COEUR))
L1T_WRAP  :=
 ↪  -Wl,--wrap=fopen,--wrap=open,--wrap=socket,--wrap=sendto,--wrap=calypso_a5_portw,--wrap=calypso_a5_portr
REJEU     ?=
tools/layer1_tester: tools/layer1_tester.c src/pcb-minimal.c $(L1T_COEUR) $(HDR)
    $(CC) $(CFLAGS) $(CPPFLAGS) -o $@ tools/layer1_tester.c src/pcb-minimal.c $(L1T_COEUR) $(LDLIBS) $(L1T_WRAP)

layer1_tester: tools/layer1_tester

api_carte: tools/layer1_tester
    python3 tools/layer1_tester.py $(if $(REJEU),--rejeu $(REJEU)) --md docs/carte-memoire.md

# Testeur de fonctions du DSP (tools/dsp_tester.c) : l'ARM osmocom-bb et la BTS
# sont simules (tools/dsp_banc_commun.c), une ligne PASS/FAIL/NON-IMPL par
# tache DSP. Ne reconstruit PAS c54x_exe.
#   make dsp_tester && ./dsp_tester --all
# Les structures de l'API RAM et la table de parametres sont celles du firmware
# osmocom-bb (dsp_api.h, dsp_params.c), lues dans son arbre.
OSMOBB  ?= /opt/GSM/osmocom-bb/src/target/firmware
BANC_CPPFLAGS := -idirafter $(OSMOBB)/include/calypso -idirafter $(OSMOBB)/calypso
BANC    := tools/dsp_banc_commun.c src/pcb-minimal.c src/verbosite.c
dsp_tester: tools/dsp_tester.c $(BANC) tools/dsp_banc_commun.h $(COEUR) $(HDR)
    $(CC) $(CFLAGS) $(CPPFLAGS) -Isrc $(BANC_CPPFLAGS) -o $@ tools/dsp_tester.c $(BANC) $(COEUR) $(LDLIBS)

clean:
    rm -f c54x_exe isa_test tools/layer1_tester dsp_tester

.PHONY: all clean layer1_tester api_carte
```

3.3 /opt/GSM/c54x_exe/balayage.sh

3968 octets, 76 lignes → 76 lignes

```
#!/usr/bin/env bash
# balayage.sh - explore les parametres du rejeu dont on n'est PAS sur, et classe
# les combinaisons par le seul critere qui compte : le nombre de SB VRAIES
# (CRC OK ET BSIC = celui injecte ET T3 valide ET FN = la trame).
#
# [2026-09-18] NE PAS CLASSER SUR crc_ok. Le CRC de la SCH fait 10 bits : une
# entree aleatoire passerait ~0,1 % du temps, or on mesurait 20 %. Un tel taux
# ne vient pas du hasard mais d'une entree DEGENERee (quasi constante) sur
# laquelle le Viterbi converge toujours vers le meme mot. Classer sur crc_ok
# recompense donc le figement : c'est ainsi qu'une phase de 0,75 est sortie en
# tete alors qu'elle produisait 12 CRC OK pour UN SEUL mot distinct. Le critere
# secondaire est le nombre de mots DISTINCTS decodes : lui mesure si la sortie
# depend de l'entree.
#
# Pourquoi : plusieurs boutons du banc sont des hypotheses non verifiees --
# la phase d'echantillonnage (0,5 = centre du symbole), la marge de tete, le
# calage de la commande SB (le DSP demodule le burst de la DERNIERE trame de
# commande, mesure du 18/09), le nombre de commandes, l'ordre RX, la longueur
# de fenetre DMA. Les regler un par un fait rater les interactions.
#
#   ./balayage.sh              balayage complet, resultats dans balayage.tsv
#   ./balayage.sh --rapide     grille reduite
#   TRAMES=1200 ./balayage.sh  plus de trames par essai (defaut 600)
#   sort -t$'\t' -k6,6nr -k5,5nr balayage.tsv | head   relire le classement
set -uo pipefail
HERE="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
EXE="$HERE/c54x_exe"
OUT="${OUT:-$HERE/balayage.tsv}"
TRAMES="${TRAMES:-600}"
[ -x "$EXE" ] || { echo "binaire absent : $EXE" >&2; exit 1; }

if [ "${1:-}" = "--rapide" ]; then
    DECALAGES="-1 0";           UNIQUES="0 1"
    PHASES="0.0 0.25 0.5 0.75"; MARGES="0 21 41"
    RXAVANT="0";                LENS="380"
else
    DECALAGES="-5 -4 -3 -2 -1 0 1";   UNIQUES="0 1"
    PHASES="0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9"
    MARGES="0 10 21 30 41";     RXAVANT="0 1"
    LENS="296 380"
fi

total=0
for a in $DECALAGES; do for b in $UNIQUES; do for c in $PHASES; do
for d in $MARGES; do for e in $RXAVANT; do for f in $LENS; do
    total=$((total+1)); done; done; done; done; done; done

printf 'decalage\tunique\tphase\tmarge\trx_avant\tdaram_len\tvraies\tmots_distincts\tcrc_ok\tsb_tentees\n' >
 ↪  "$OUT"
echo "[balayage] $total combinaisons, $TRAMES trames chacune -> $OUT"
n=0; t0=$(date +%s)
for a in $DECALAGES; do
 for b in $UNIQUES; do
  for c in $PHASES; do
   for d in $MARGES; do
    for e in $RXAVANT; do
     for f in $LENS; do
      n=$((n+1))
      res=$(REJEU_SB_FORCE=1 \
            REJEU_SB_DECALAGE="$a" REJEU_SB_UNIQUE="$b" \
            REJEU_DECALAGE_SYMB="$c" REJEU_MARGE="$d" \
            REJEU_RX_AVANT="$e" CALYPSO_BSP_DARAM_LEN="$f" \
```

```
            timeout 180 "$EXE" --rejouer --verbeux --trames "$TRAMES" 2>/dev/null)
      vraies=$(printf '%s' "$res" | sed -n 's/.*VRAIES ([^)]*): \([0-9]*\).*/\1/p' | head -1)
      crc=$(   printf '%s' "$res" | sed -n 's/.*CRC OK: \([0-9]*\).*/\1/p' | head -1)
      tent=$(  printf '%s' "$res" | sed -n 's#.*SB tentees / CRC KO: \([0-9]*\) /.*#\1#p' | head -1)
      mots=$(  printf '%s' "$res" | grep -oE '^  SB[0-9] fn=[0-9]+ : sb=0x[0-9a-f]+' | grep -oE '0x[0-9a-f]+' |
       ↪  sort -u | wc -l)
      printf '%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\t%s\n' \
             "$a" "$b" "$c" "$d" "$e" "$f" "${vraies:-0}" "${mots:-0}" "${crc:-0}" "${tent:-0}" >> "$OUT"
      if [ $((n % 25)) -eq 0 ]; then
          dt=$(( $(date +%s) - t0 ))
          echo "[balayage] $n/$total  ${dt}s  meilleur vraies=$(tail -n +2 "$OUT" | cut -f7 | sort -nr | head -1)
           ↪  mots_distincts=$(tail -n +2 "$OUT" | cut -f8 | sort -nr | head -1)"
      fi
    done; done; done; done; done; done
echo "[balayage] termine, $n essais."
echo "--- 15 meilleures combinaisons (vraies, puis mots distincts) ---"
{ head -1 "$OUT"; tail -n +2 "$OUT" | sort -t$'\t' -k7,7nr -k8,8nr | head -15; } | column -t -s$'\t'
```

3.4 /opt/GSM/c54x_exe/harvard.sh

62879 octets, 839 lignes → 839 lignes

```
#!/bin/sh
# harvard.sh - shell « Harvard » pour le DSP TMS320C54x du Calypso (sh POSIX, dash suffit).
#
# Le C54x est une machine Harvard : l'espace P (programme, les opcodes) et l'espace D (donnees,
# registres, API RAM) sont separes. Ce shell garde les deux de front :
#   - cote P : on tape une instruction (mnemonique assemble par gas tic54x, ou des mots hex),
#              on desassemble la ROM TI (PROM0 a 0x7000, PDROM a 0xE000, pages 1..3 via XPC) ;
#   - cote D : un etat (registres + mots de memoire donnees) qu'on lit, qu'on pose, et que
#              chaque instruction executee fait avancer ; la DROM (0x9000) et la PDROM se lisent.
# L'execution d'une instruction passe par ./isa_test (le vrai coeur C54x de qosmo, pcb-minimal) :
# on lui donne l'etat « avant » et les mots, il rend l'etat « apres ». Rien n'est simule ici.
# Les commandes « run » lancent les outils du banc tels quels (c54x_exe, dsp_tester, isa_test, run.sh).
#
#   ./harvard.sh                menu interactif
#   ./harvard.sh repl           invite de commandes directe
#   ./harvard.sh pas 'LD #0x1234, A'    une commande puis sortie (toutes celles du REPL)
#   ./harvard.sh aide
# Variables : HARVARD_DIR (etat, defaut ~/.local/state/harvard-c54x), ROM_DIR (defaut ./rom),
#             ISA_TEST, OBJDUMP, GAS (chemins des outils).

set -u
HERE=$(cd "$(dirname "$0")" && pwd)
DIR=${HARVARD_DIR:-${XDG_STATE_HOME:-$HOME/.local/state}/harvard-c54x}
ROM_DIR=${ROM_DIR:-$HERE/rom}
ISA_TEST=${ISA_TEST:-$HERE/isa_test}
OBJDUMP=${OBJDUMP:-$HERE/build-tic54x/binutils/objdump}
GAS=${GAS:-$HERE/build-tic54x/gas/as-new}
ETAT=$DIR/etat.txt          # une ligne par element : "REG hex" | "M addr hex" | "P addr hex"
JOURNAL=$DIR/journal.txt
LOG=$DIR/coeur.log          # stderr du coeur (avertissements [c54x] ...)
REGS="A B T SP PC ST0 ST1 PMST AR0 AR1 AR2 AR3 AR4 AR5 AR6 AR7 BK BRC RSA REA TRN IMR IFR XPC"
SENT_REG=deadbeef1          # sentinelle 36 bits : jamais egale a un registre -> isa_test affiche « obtenu »
SENT_MEM=1ffff              # sentinelle 17 bits : jamais egale a un mot de 16 bits
mkdir -p "$DIR"; : >> "$ETAT"; : >> "$JOURNAL"

# ---------------------------------------------------------------- utilitaires
rouge() { if [ -t 1 ]; then printf '\033[31m%s\033[0m\n' "$*"; else printf '%s\n' "$*"; fi; }
vert()  { if [ -t 1 ]; then printf '\033[32m%s\033[0m\n' "$*"; else printf '%s\n' "$*"; fi; }
gras()  { if [ -t 1 ]; then printf '\033[1m%s\033[0m\n' "$*"; else printf '%s\n' "$*"; fi; }
hex() { # hex <texte> : accepte 0x1234, 1234, 1:8000 (page:adresse) ; rend l'entier
    case $1 in
        *:*) printf '%d' $(( ( ${1%%:*} << 16 ) | 0x${1#*:} )) ;;
```

```
        0x*|0X*) printf '%d' $(( $1 )) ;;
        *) printf '%d' $(( 0x$1 )) ;;
    esac 2>/dev/null
}
est_hex_mots() { # vrai si tous les arguments sont des mots hex de 1 a 4 chiffres
    for m in "$@"; do case $m in ''|*[!0-9a-fA-F]*|?????*) return 1;; esac; done; return 0
}
verifier_outils() {
    [ -x "$ISA_TEST" ] || rouge "isa_test absent ($ISA_TEST) : make isa_test"
    [ -x "$OBJDUMP" ]  || rouge "objdump tic54x absent ($OBJDUMP) : pas de desassemblage"
    [ -x "$GAS" ]      || rouge "gas tic54x absent ($GAS) : seuls les mots hex sont acceptes"
}

# ------------------------------------------------------------------ etat (cote D)
etat_reg() { awk -v r="$1" '$1==r {v=$2} END {print v}' "$ETAT"; }
etat_fixe() { # etat_fixe <cle...> <val> : remplace ou ajoute la ligne "<cles> <val>"
    cle=$1; shift
    [ $# -gt 1 ] && { cle="$cle $1"; shift; }
    { grep -v "^$cle " "$ETAT"; echo "$cle $1"; } > "$ETAT.new" && mv "$ETAT.new" "$ETAT"
}
etat_mem() { awk -v a="$1" '$1=="M" && $2==a {v=$3} END {print v}' "$ETAT"; }
drapeaux() { # drapeaux <st0> <st1> : noms des bits a 1
    st0=$(( 0x${1:-0} )); st1=$(( 0x${2:-0} )); d=""
    [ $(( st0 >> 12 & 1 )) = 1 ] && d="$d TC";  [ $(( st0 >> 11 & 1 )) = 1 ] && d="$d C"
    [ $(( st0 >> 10 & 1 )) = 1 ] && d="$d OVA"; [ $(( st0 >> 9 & 1 )) = 1 ]  && d="$d OVB"
    [ $(( st1 >> 15 & 1 )) = 1 ] && d="$d BRAF"; [ $(( st1 >> 14 & 1 )) = 1 ] && d="$d CPL"
    [ $(( st1 >> 13 & 1 )) = 1 ] && d="$d XF";  [ $(( st1 >> 12 & 1 )) = 1 ] && d="$d HM"
    [ $(( st1 >> 11 & 1 )) = 1 ] && d="$d INTM"; [ $(( st1 >> 9 & 1 )) = 1 ] && d="$d OVM"
    [ $(( st1 >> 8 & 1 )) = 1 ]  && d="$d SXM"; [ $(( st1 >> 7 & 1 )) = 1 ]  && d="$d C16"
    [ $(( st1 >> 6 & 1 )) = 1 ]  && d="$d FRCT"; [ $(( st1 >> 5 & 1 )) = 1 ] && d="$d CMPT"
    printf 'ARP=%d DP=%03x ASM=%02x flags:%s' $(( st0 >> 13 )) $(( st0 & 0x1ff )) $(( st1 & 0x1f )) "${d:- (aucun)}"
}
montrer_regs() {
    [ -s "$ETAT" ] || { echo "(etat vide : les registres ont les valeurs d'init du coeur ; faites un pas)";
     ↪  return; }
    printf '  A  =%12s   B  =%12s   T  =%6s   PC =%6s   SP =%6s   XPC=%4s\n' \
        "$(etat_reg A)" "$(etat_reg B)" "$(etat_reg T)" "$(etat_reg PC)" "$(etat_reg SP)" "$(etat_reg XPC)"
    printf '  AR0=%6s AR1=%6s AR2=%6s AR3=%6s AR4=%6s AR5=%6s AR6=%6s AR7=%6s\n' \
        "$(etat_reg AR0)" "$(etat_reg AR1)" "$(etat_reg AR2)" "$(etat_reg AR3)" "$(etat_reg AR4)" "$(etat_reg
         ↪  AR5)" "$(etat_reg AR6)" "$(etat_reg AR7)"
    printf '  ST0=%6s ST1=%6s PMST=%5s BK=%6s BRC=%6s RSA=%6s REA=%6s TRN=%6s IMR=%5s IFR=%5s\n' \
        "$(etat_reg ST0)" "$(etat_reg ST1)" "$(etat_reg PMST)" "$(etat_reg BK)" "$(etat_reg BRC)" "$(etat_reg
         ↪  RSA)" "$(etat_reg REA)" "$(etat_reg TRN)" "$(etat_reg IMR)" "$(etat_reg IFR)"
    printf '  %s\n' "$(drapeaux "$(etat_reg ST0)" "$(etat_reg ST1)")"
}
montrer_mem() {
    n=$(grep -c '^M ' "$ETAT"); [ "$n" = 0 ] && { echo "  D : aucun mot pose"; return; }
    echo "  D (mots poses / lus, $n) :"; awk '$1=="M" {printf "   [%04x]=%04x", strtonum("0x"$2),
     ↪  strtonum("0x"$3); if (++k%6==0) print ""} END {if (k%6) print ""}' "$ETAT" 2>/dev/null \
      || awk '$1=="M" {printf "   [%s]=%s", $2, $3; if (++k%6==0) print ""} END {if (k%6) print ""}' "$ETAT"
    n=$(grep -c '^P ' "$ETAT"); [ "$n" != 0 ] && { echo "  P (mots poses, $n) :"; awk '$1=="P" {printf "
     ↪  [%s]=%s", $2, $3; if (++k%6==0) print ""} END {if (k%6) print ""}' "$ETAT"; }
}
cmd_etat() { gras "--- registres (cote D)"; montrer_regs; montrer_mem; }
cmd_raz() { : > "$ETAT"; echo "etat remis a zero (valeurs d'init du coeur au prochain pas)"; }
cmd_reg() {
    [ $# -lt 2 ] && { montrer_regs; return; }
    r=$(printf '%s' "$1" | tr a-z A-Z); v=$(hex "$2") || { rouge "valeur invalide"; return 1; }
    etat_fixe "$r" "$(printf '%x' "$v")"; echo "$r <- $(printf '%x' "$v")"
}

# ------------------------------------------------------------------ assemblage / desassemblage (cote P)
asm() { # asm <instruction> -> mots hex sur stdout, 1 si echec
    if est_hex_mots $1; then printf '%s\n' "$*" | tr A-F a-f; return 0; fi
    [ -x "$GAS" ] || { rouge "pas d'assembleur : donnez les mots hex"; return 1; }
    printf '\t.text\n\t%s\n' "$*" > "$DIR/a.s"
```

```
    if ! "$GAS" -o "$DIR/a.o" "$DIR/a.s" 2> "$DIR/a.err"; then
        rouge "assemblage refuse :"; sed 's/^/    /' "$DIR/a.err"; return 1
    fi
    "$OBJDUMP" -d "$DIR/a.o" | sed -n 's/^ *[0-9a-f]*:[[:space:]]*\([0-9a-f]\{4\}\).*/\1/p' | tr '\n' ' ' | sed
     ↪  's/ $/\n/'
}
dis_mots() { # dis_mots <adresse> <mots hex...> : desassemble des mots
    a=$1; shift; : > "$DIR/d.bin"
    for m in "$@"; do v=$(( 0x$m )); printf "\\$(printf '%03o' $(( v & 255 )))\\$(printf '%03o' $(( v >> 8 )))" >>
     ↪  "$DIR/d.bin"; done
    "$OBJDUMP" -D -b binary -m tms320c54x --adjust-vma="0x${a:-9000}" "$DIR/d.bin" | sed -n 's/^
     ↪  *\([0-9a-f]*:.*\)/    \1/p'
}
rom_fichier() { # rom_fichier <adr lineaire P ou D> <P|D> -> "fichier base_mots"
    a=$1
    if [ "$2" = P ]; then
        page=$(( a >> 16 )); off=$(( a & 0xffff ))
        case $page in
            0) if [ $off -ge $(( 0xe000 )) ]; then echo "$ROM_DIR/calypso_dsp.PDROM.bin $(( 0xe000 ))"
               elif [ $off -ge $(( 0x7000 )) ]; then echo "$ROM_DIR/calypso_dsp.PROM0.bin $(( 0x7000 ))"; else
                ↪  return 1; fi ;;
            1|2|3) [ $off -ge $(( 0x8000 )) ] || return 1; echo "$ROM_DIR/calypso_dsp.PROM$page.bin $(( (page <<
             ↪  16) | 0x8000 ))" ;;
            *) return 1 ;;
        esac
    else
        if [ $a -ge $(( 0xe000 )) ]; then echo "$ROM_DIR/calypso_dsp.PDROM.bin $(( 0xe000 ))"
        elif [ $a -ge $(( 0x9000 )) ]; then echo "$ROM_DIR/calypso_dsp.DROM.bin $(( 0x9000 ))"; else return 1; fi
    fi
}
cmd_p() { # p <adresse> [n mots] : desassembler la ROM programme
    [ $# -ge 1 ] || { echo "usage : p <adresse> [n]   (0x7000.., 0xe000.., 1:8000 = page 1)"; return 1; }
    a=$(hex "$1") || { rouge "adresse invalide"; return 1; }; n=${2:-16}
    set -- $(rom_fichier "$a" P) || { rouge "$(printf '%x' "$a") : hors ROM programme (PROM0 7000-dfff, PDROM
     ↪  e000-ffff, pages 1-3 : 8000-ffff)"; return 1; }
    f=$1; base=$2; [ -r "$f" ] || { rouge "ROM absente : $f (rom/fetch-rom.sh)"; return 1; }
    page=$(( a >> 16 )); off=$(( a & 0xffff )); vma=$(( base & 0xffff ))
    [ $page -gt 0 ] && gras "  page $page (XPC=$page), fichier $(basename "$f")"
    "$OBJDUMP" -D -b binary -m tms320c54x --adjust-vma=$vma --start-address=$off --stop-address=$(( off + n ))
     ↪  "$f" \
        | sed -n 's/^ *\([0-9a-f]*:.*\)/    \1/p'
}
cmd_drom() { # drom <adresse> [n] : lire la ROM donnees
    [ $# -ge 1 ] || { echo "usage : drom <adresse> [n]   (DROM 9000-dfff, PDROM e000-ffff)"; return 1; }
    a=$(hex "$1") || { rouge "adresse invalide"; return 1; }; n=${2:-16}
    set -- $(rom_fichier "$a" D) || { rouge "$(printf '%x' "$a") : hors ROM donnees"; return 1; }
    f=$1; base=$2; [ -r "$f" ] || { rouge "ROM absente : $f"; return 1; }
    od -An -v -tx2 -w16 -j $(( (a - base) * 2 )) -N $(( n * 2 )) "$f" | awk -v a=$a '{printf "    [%04x]", a;
     ↪  print; a+=8}'
}
cmd_asm() { m=$(asm "$@") || return 1; printf '    mots : %s\n' "$m"; dis_mots "$(etat_reg PC)" $m; }
cmd_dis() { est_hex_mots "$@" || { echo "usage : dis <mots hex>"; return 1; }; dis_mots "$(etat_reg PC)" "$@"; }

# ------------------------------------------------------------------ execution d'une instruction (P -> D)
surveiller() { # surveiller <mot1> [mot2] : adresses D a lire a chaque pas (celles qu'une instruction isolee peut
 ↪  toucher)
    w1=$(( 0x$1 )); st0=$(etat_reg ST0); sp=$(etat_reg SP)
    {
        for rg in AR0 AR1 AR2 AR3 AR4 AR5 AR6 AR7; do v=$(etat_reg $rg); [ -n "$v" ] || continue
            v=$(( 0x$v )); printf '%04x\n%04x\n%04x\n' $(( (v - 1) & 0xffff )) $v $(( (v + 1) & 0xffff )); done
        [ -n "$sp" ] && { v=$(( 0x$sp )); printf '%04x\n%04x\n%04x\n' $(( (v - 1) & 0xffff )) $v $(( (v + 1) &
         ↪  0xffff )); }
        printf '%04x\n' $(( ((0x${st0:-0} & 0x1ff) << 7) | (w1 & 0x7f) ))      # adressage direct DP:dma
        [ $# -ge 2 ] && printf '%04x\n' $(( 0x$2 & 0xffff ))                    # adresse longue lk
    } | sort -u > "$DIR/cand.txt"; awk '$1=="M" {print $2}' "$ETAT" > "$DIR/connu.txt"; grep -v -F -x -f
     ↪  "$DIR/connu.txt" "$DIR/cand.txt" 2>/dev/null || cat "$DIR/cand.txt"
```

```
}
executer() { # executer <garder 0|1> <titre> <mots hex...> [adresses D a lire...]  ->  $DIR/pas.apres : lignes
↪  "NOM valeur"
    garder=$1; titre=$2; mots=$3; shift 3
    t=$DIR/pas.txt; nm=$(grep -c '^M ' "$ETAT"); place=$(( 28 - nm )); [ $place -lt 0 ] && place=0
    {
        echo "T 0 $titre"; echo "W $mots"
        sed -n 's/^\([A-Z][A-Z0-9]*\) \([0-9a-fA-F]*\)$/B \1 \2/p; s/^\([MP]\) \([0-9a-fA-F]*\)
        ↪  \([0-9a-fA-F]*\)$/B \1 \2 \3/p' "$ETAT"
        for rg in $REGS; do echo "A $rg $SENT_REG"; done
        awk '$1=="M" {print "A M " $2 " '"$SENT_MEM"'"}' "$ETAT"
        { for adr in "$@"; do echo "$adr"; done; [ $# -eq 0 ] && surveiller $mots; } | head -n $place | sed "s/^/A
        ↪  M /; s/\$/ $SENT_MEM/"
        echo "E"
    } > "$t"
    "$ISA_TEST" "$t" 2>> "$LOG" > "$DIR/pas.out"
    grep -q '^FAIL' "$DIR/pas.out" || { rouge "isa_test n'a rien rendu :"; cat "$DIR/pas.out"; return 1; }
    grep -q '\[[0-9a-f ]*?\]' "$DIR/pas.out" && rouge "  (un registre de l'etat est inconnu d'isa_test, ignore)"
    # "  A: attendu deadbeef1, obtenu 12  data[0100]: attendu 1ffff, obtenu 0000" -> "A 12" / "M 0100 0000"
    sed -n 's/^FAIL[^]]*\]//p' "$DIR/pas.out" | sed 's/  /\n/g' | sed -n \
        's/^data\[\([0-9a-f]*\)\]: attendu [0-9a-f]*, obtenu \([0-9a-f]*\)$/M \1 \2/p;
         s/^prog\[\([0-9a-f]*\)\]: attendu [0-9a-f]*, obtenu \([0-9a-f]*\)$/P \1 \2/p;
         s/^\([A-Z][A-Z0-9]*\): attendu [0-9a-f]*, obtenu \([0-9a-f]*\)$/\1 \2/p' > "$DIR/pas.apres"
    if [ "$garder" = 1 ]; then
        # on garde : les registres, les mots P poses, les mots D deja dans l'etat, et tout mot D devenu non nul
        { grep '^P ' "$ETAT"
          awk 'FILENAME==ARGV[1] { if ($1=="M") connu[$2]=1; next }
               $1=="P" { next } $1!="M" { print; next } ($2 in connu) || ($3 != "0000" && $2 >= "0060") { print }'
↪  "$ETAT" "$DIR/pas.apres"
        } > "$ETAT.new" && mv "$ETAT.new" "$ETAT"
        printf '%s  %s  | %s\n' "$(date +%H:%M:%S)" "$mots" "$titre" >> "$JOURNAL"
    fi
}
diff_etat() { # diff_etat <avant> <apres> : ce qui a change
    awk 'FILENAME==ARGV[1] { if ($1=="M"||$1=="P") av[$1" "$2]=$3; else av[$1]=$2; next }
         { if ($1=="M"||$1=="P") { k=$1" "$2; v=$3 } else { k=$1; v=$2 }
           if (!(k in av)) { if (k !~ /^M |^P /) nouveaux=nouveaux" "k"="v; else if (v != "0000") printf "    %-10s
↪  %12s -> %s\n", k, "(nouveau)", v }
           else if (av[k] != v) printf "    %-10s %12s -> %s\n", k, av[k], v }
         END { if (nouveaux != "") print "    (premiere lecture :" nouveaux ")" }' "$1" "$2"
}
cmd_pas() { # pas <instruction | mots hex>
    [ $# -ge 1 ] || { echo "usage : pas <instruction>   ex : pas LD #0x1234, A   |   pas f020 1234"; return 1; }
    mots=$(asm "$@") || return 1
    pc=$(etat_reg PC); printf '  P[%s] %s   <- %s\n' "${pc:-9000}" "$mots" "$*"
    cp "$ETAT" "$DIR/avant.txt"
    executer 1 "$*" "$mots" || return 1
    gras "  changements :"; diff_etat "$DIR/avant.txt" "$ETAT" | grep . || echo "    (aucun registre ni mot pose
    ↪  n'a change)"
    printf '  PC=%s  A=%s  B=%s  T=%s  %s\n' "$(etat_reg PC)" "$(etat_reg A)" "$(etat_reg B)" "$(etat_reg T)"
    ↪  "$(drapeaux "$(etat_reg ST0)" "$(etat_reg ST1)")"
}
cmd_essai() { # essai <instruction> : executer sans garder l'etat
    [ $# -ge 1 ] || { echo "usage : essai <instruction>"; return 1; }
    mots=$(asm "$@") || return 1
    executer 0 "$*" "$mots" || return 1
   gras "  resultat (etat NON garde) :"; diff_etat "$ETAT" "$DIR/pas.apres" | grep . || echo "    (rien ne change)"
}
cmd_d() { # d <adresse> [n] : lire la memoire donnees du coeur (via un NOP non garde)
    [ $# -ge 1 ] || { echo "usage : d <adresse> [n<=24]"; return 1; }
    a=$(hex "$1") || { rouge "adresse invalide"; return 1; }; n=${2:-8}; [ $n -gt 24 ] && n=24
    liste=""; i=0; while [ $i -lt $n ]; do liste="$liste $(printf '%04x' $(( a + i )))"; i=$(( i + 1 )); done
    executer 0 "lecture D" f495 $liste || return 1
    awk -v a=$a 'BEGIN{k=0} $1=="M" { lu[$2]=$3 }
         END { for (i=0;i<'"$n"';i++) { k=sprintf("%04x", a+i); printf "    [%s]=%s", k, (k in lu)?lu[k]:"????";
↪  if ((i+1)%8==0) print "" } if (i%8) print "" }' "$DIR/pas.apres"
```

```
    st0=$(etat_reg ST0); echo "    (lecture par le coeur avec l'etat courant ; DP=$(printf '%03x' $(( 0x${st0:-0}
     ↪  & 0x1ff ))) ; les mots non poses valent 0 hors registres mappes 0000-005f)"
}
cmd_d_pose() { # d! <adresse> <mots...> : poser des mots en D
    [ $# -ge 2 ] || { echo "usage : d! <adresse> <mot> [mot...]"; return 1; }
    a=$(hex "$1") || { rouge "adresse invalide"; return 1; }; shift
    for m in "$@"; do v=$(hex "$m") || { rouge "mot invalide : $m"; return 1; }
        etat_fixe M "$(printf '%04x' $a)" "$(printf '%04x' $(( v & 0xffff )))"; printf '    D[%04x] <- %04x\n' $a
         ↪  $(( v & 0xffff )); a=$(( a + 1 )); done
}
cmd_p_pose() { # p! <adresse> <instruction | mots> : poser des mots en P (precharges a chaque pas)
    [ $# -ge 2 ] || { echo "usage : p! <adresse> <instruction|mots hex>   (ex. une sous-routine appelee par
     ↪  CALL)"; return 1; }
    a=$(hex "$1") || { rouge "adresse invalide"; return 1; }; shift
    mots=$(asm "$@") || return 1
    for m in $mots; do etat_fixe P "$(printf '%04x' $a)" "$m"; printf '    P[%04x] <- %s\n' $a "$m"; a=$(( a + 1
     ↪  )); done
}
cmd_journal() { gras "--- journal des pas"; tail -n ${1:-30} "$JOURNAL"; }
cmd_log() { gras "--- avertissements du coeur (dernieres lignes)"; tail -n ${1:-20} "$LOG"; }
cmd_sauve() { [ $# -ge 1 ] || { echo "usage : sauve <fichier>"; return 1; }; cp "$ETAT" "$1" && echo "etat -> $1"; }
cmd_charge() { [ $# -ge 1 ] && [ -r "$1" ] || { echo "usage : charge <fichier>"; return 1; }; cp "$1" "$ETAT" &&
 ↪  echo "etat <- $1" && montrer_regs; }

# ------------------------------------------------------------- cours / arret : courir dans la ROM (etat du
 ↪  shell)
# « arret » pose des points d'arret (adresses P lineaires, 1:8000 = page 1) ; « cours » part du PC courant
# (ff80 = vecteur de reset si l'etat est vide), lit chaque instruction dans la ROM (ou dans les mots P
# poses, qui ont priorite), l'execute par isa_test comme « pas », et s'arrete sur : un point d'arret,
# IDLE, une adresse hors ROM (RAM sans mot pose), un PC qui ne bouge pas, ou le budget de pas.
# Tout ceci agit sur l'ETAT DU SHELL, jamais sur le c54x_exe en marche (pour lui : stop / go / dsp).
# Limites (le coeur repart d'un etat « registres + mots » a chaque pas) : le creux d'un branchement
# retarde (BD, CALLD, RETD...) et le compteur d'un RPT ne survivent pas d'un pas a l'autre.
ARRETS=$DIR/arrets.txt; : >> "$ARRETS"
cmd_arret() { # arret [adr...] | arret - : poser / lister / effacer les points d'arret
    if [ $# -eq 0 ]; then
        [ -s "$ARRETS" ] && { gras "--- points d'arret (adresses P)"; sed 's/^/    /' "$ARRETS"; } || echo "  aucun
         ↪  point d'arret (arret <adr> [adr...])"; return
    fi
    [ "$1" = - ] && { : > "$ARRETS"; echo "  points d'arret effaces"; return; }
    for s in "$@"; do a=$(hex "$s") || { rouge "adresse invalide : $s"; return 1; }
        a=$(printf '%x:%04x' $(( a >> 16 )) $(( a & 0xffff ))); grep -q -x -F "$a" "$ARRETS" || echo "$a" >>
         ↪  "$ARRETS"; echo "    arret a $a"; done
}
rom_insn() { # rom_insn <lineaire P> -> "mots hex|mnemonique" ; priorite aux mots P poses ; 1 si hors ROM
    a=$1; page=$(( a >> 16 )); off=$(( a & 0xffff )); : > "$DIR/i.bin"
    i=0; while [ $i -lt 3 ]; do
        lin=$(( (page << 16) | ((off + i) & 0xffff) )); ad=$(printf '%04x' $(( lin & 0xffff )))
        m=$(awk -v a="$ad" '$1=="P" && $2==a {v=$3} END {print v}' "$ETAT")
        if [ -z "$m" ]; then
            if fb=$(rom_fichier $lin P); then set -- $fb; m=$(od -An -v -tx2 -j $(( (lin - $2) * 2 )) -N 2 "$1" |
             ↪  tr -d ' ')
            elif [ $i -eq 0 ]; then return 1; else m=f495; fi
        fi
        v=$(( 0x$m )); printf "\\$(printf '%03o' $(( v & 255 )))\\$(printf '%03o' $(( v >> 8 )))" >> "$DIR/i.bin";
         ↪  i=$(( i + 1 ))
    done
    "$OBJDUMP" -D -b binary -m tms320c54x --adjust-vma="0x$(printf '%04x' $off)" "$DIR/i.bin" | sed -n 's/^
     ↪  *[0-9a-f]*:[[:space:]]*//p' \
      | awk 'NR==1 { w=$1; $1=""; sub(/^ +/,""); m=$0; next } NF>1 { exit } { w=w" "$1 } END { print w "|" m }'
}
cmd_cours() { # cours [N pas] : courir depuis le PC courant jusqu'a un arret
    n=${1:-200}; case $n in ''|*[!0-9]*) echo "usage : cours [nombre de pas max]"; return 1;; esac
    pc=$(etat_reg PC); xpc=$(etat_reg XPC)
    [ -n "$pc" ] || { etat_fixe PC ff80; etat_fixe XPC 0; pc=ff80; xpc=0; echo "  (etat vide : depart au vecteur
     ↪  de reset ff80)"; }
```

```
    gras "  cours depuis $(printf '%x:%04x' $(( 0x${xpc:-0} )) $(( 0x$pc )))  (budget $n pas ; arrets : $(tr '\n'
     ↪  ' ' < "$ARRETS"))"
    : > "$DIR/cours_adr.txt"; k=0; motif=""
    while :; do
        pc=$(etat_reg PC); xpc=$(etat_reg XPC); lin=$(( (0x${xpc:-0} << 16) | 0x$pc )); cle=$(printf '%x:%04x' $((
         ↪  lin >> 16 )) $(( lin & 0xffff )))
        if [ $k -gt 0 ] && grep -q -x -F "$cle" "$ARRETS"; then motif="point d'arret $cle"; break; fi
        [ $k -lt $n ] || { motif="budget de $n pas epuise"; break; }
        r=$(rom_insn $lin) || { motif="$cle hors ROM programme (RAM sans mot pose)"; break; }
        mots=${r%%|*}; mn=${r#*|}
        printf '%s\n' "$cle" >> "$DIR/cours_adr.txt"
        case $mn in idle*) printf '    %-7s %-14s %s\n' "$cle" "$mots" "$mn"; motif="IDLE a $cle"; k=$(( k + 1 ));
         ↪  break ;; esac
        executer 1 "cours $cle $mn" "$mots" || { motif="isa_test en echec a $cle"; break; }
        k=$(( k + 1 )); pc2=$(etat_reg PC); xpc2=$(etat_reg XPC)
        printf '    %-7s %-14s %-28s -> %x:%04x  A=%s B=%s\n' "$cle" "$mots" "$mn" $(( 0x${xpc2:-0} )) $((
         ↪  0x${pc2:-0} )) "$(etat_reg A)" "$(etat_reg B)"
        [ "$pc2" = "$pc" ] && [ "${xpc2:-0}" = "${xpc:-0}" ] && { motif="PC immobile a $cle (boucle sur place /
         ↪  instruction non executee)"; break; }
    done
    gras "  arret : $motif  ($k pas)"
    gras "  adresses parcourues ($(sort -u "$DIR/cours_adr.txt" | wc -l) distinctes, dans l'ordre) :"
    awk '!(seen[$0]++)' "$DIR/cours_adr.txt" | tr '\n' ' ' | fold -s -w 96 | sed 's/^/    /'; echo
    printf '  PC=%s  A=%s  B=%s  T=%s  %s\n' "$(etat_reg PC)" "$(etat_reg A)" "$(etat_reg B)" "$(etat_reg T)"
     ↪  "$(drapeaux "$(etat_reg ST0)" "$(etat_reg ST1)")"
}

# ------------------------------------------------------------------ DSP EN MARCHE : stop / go / dsp sur le c54x_exe
 ↪  --arm
# Ici on agit sur le PROCESSUS VIVANT (celui que run.sh a lance), jamais sur l'etat du shell :
#   stop   pause du DSP. Binaire recent (src/pont.c SIGUSR1) : pause cooperative a la frontiere de trame
#          suivante (DONE rendu, DSP en IDLE, l'ARM attend son DONE suivant) ; binaire ancien : SIGSTOP.
#          Dans les deux cas rien n'est ecrit dans l'API RAM ni dans le coeur : le firmware ne voit qu'une
#          trame plus longue. Les registres sont lus dans la memoire du processus (/proc/PID/mem, decalages
#          de C54xState donnes par gdb sur l'executable), et on affiche LES ADRESSES : PC (ou le DSP s'est
#          arrete, IDLE) et les vecteurs armes (ou il repartira : IPTR de PMST, IT trame vec 28 = bit 12).
#   go     reprise (SIGUSR2 ou SIGCONT), puis un instantane 0,3 s plus tard pour voir qu'il a repris.
#   dsp    instantane sans arreter (lecture pendant qu'il tourne : coherente a une instruction pres).
PAUSE_FICHIER=${PAUSE_FICHIER:-/dev/shm/calypso_dsp_pause}
VIVANT=$DIR/vivant.txt
IRQ_NOMS="12=IT_trame_TPU(vec28) 14=DMA_INT10n(vec30)"   # c54x_irq.c:44-60 ; les autres bits : non nommes ici
vivant_pid() { # le c54x_exe --arm : comm == c54x_exe (lisible par tous, meme si le process est a root) + cmdline
 ↪  --arm
    for p in $(pgrep -x c54x_exe 2>/dev/null); do
        [ "$(cat /proc/$p/comm 2>/dev/null)" = c54x_exe ] || continue
        grep -qa -- --arm /proc/$p/cmdline 2>/dev/null && { echo $p; return 0; }
    done; return 1
}
vivant_accessible() { # vivant_accessible <pid> : ce compte peut-il piloter ce process (signal + /proc/mem) ?
    kill -0 "$1" 2>/dev/null && [ -r "/proc/$1/mem" ]
}
vivant_diag() { # message clair quand le banc tourne sous un autre compte
    pid=$1; u=$(stat -c %U /proc/$pid 2>/dev/null); moi=$(id -un)
    rouge "le banc c54x_exe (pid $pid) tourne sous « ${u:-?} », toi tu es « $moi »."
    rouge "  -> stop/go/dsp exigent le meme compte. Lance harvard ainsi :  sudo ./harvard.sh"
    rouge "     (ou relance le banc sous ton compte). Lecture seule (d@) possible, ecriture (d@!) non."
}
vivant_coop() { grep -q -a calypso_dsp_pause /proc/$1/exe 2>/dev/null; }
vivant_decalages() { # vivant_decalages <pid> -> fichier "champ decalage" (gdb sur l'executable, une fois par pid)
    f=$DIR/decalages.$1.txt
    if ! [ -s "$f" ]; then
        command -v gdb > /dev/null || { rouge "gdb absent : impossible de situer C54xState dans le processus" >&2;
         ↪  return 1; }
        { for c in data prog pc xpc idle running a b ar t sp st0 st1 pmst imr ifr bk brc rsa rea trn insn_count
         ↪  last_exec_pc last_exec_op; do
              printf 'printf "%s %%ld\\n", (long)&((C54xState*)0)->%s\n' "$c" "$c"; done
```

```
          printf 'printf "FN %%ld\\n", (long)&g_c54x_exe_fn\n'; } > "$DIR/decal.gdb"
        gdb -batch -q -nx -ex 'set debuginfod enabled off' -x "$DIR/decal.gdb" /proc/$1/exe 2>/dev/null | grep -E
         ↪  '^[a-zA-Z_0-9]+ [0-9]+$' > "$f"
        [ "$(wc -l < "$f")" -ge 25 ] || { rm -f "$f"; rouge "gdb n'a pas rendu les decalages (executable sans -g
         ↪  ?)" >&2; return 1; }
    fi
    echo "$f"
}
vivant_lire() { # vivant_lire <pid> <etiquette> : registres du coeur du processus -> $VIVANT (lignes "CLE val")
    pid=$1; d=$(vivant_decalages $pid) || return 1
    shm=$(awk '/calypso_api_ram/ {print $1; exit}' /proc/$pid/maps); [ -n "$shm" ] || { rouge "pas de fenetre API
     ↪  RAM dans le processus $pid"; return 1; }
    base=$(( 0x${shm%%-*} - 0x1000 - $(awk '$1=="data" {print $2}' "$d") ))      # data[0x0800] est mappe sur
     ↪  /dev/shm
    exe=$(awk '$6 ~ /c54x_exe/ && $3=="00000000" {print $1; exit}' /proc/$pid/maps); exe=$(( 0x${exe%%-*} ))
    dec() { awk -v k="$1" '$1==k {print $2}' "$d"; }
    lire() { dd if=/proc/$pid/mem bs=1 skip=$1 count=$2 2>/dev/null | od -An -v -tx$2 | tr -d ' \n'; }   #
     ↪  petit-boutiste -> hex
    echo "$pid $base $(dec prog) $(dec data) $(( 0x$(lire $(( base + $(dec pmst) )) 2) ))" >
     ↪  "$DIR/vivant_prog.txt"   # pour vivant_prog
    pc=$(lire $(( base + $(dec pc) )) 4) || return 1
    [ -n "$pc" ] || { rouge "lecture de /proc/$pid/mem refusee (ptrace)"; return 1; }
    {
        echo "ETAT $2"; echo "FN $(( 0x$(lire $(( exe + $(dec FN) )) 4) ))"
        printf 'PC %04x\nXPC %x\nIDLE %d\nRUNNING %d\nINSN %d\nLAST_PC %04x\nLAST_OP %04x\n' $(( 0x$pc & 0xffff ))
         ↪  0x$(lire $(( base + $(dec xpc) )) 2) \
            0x$(lire $(( base + $(dec idle) )) 1) 0x$(lire $(( base + $(dec running) )) 1) 0x$(lire $(( base +
↪  $(dec insn_count) )) 4) \
            0x$(lire $(( base + $(dec last_exec_pc) )) 2) 0x$(lire $(( base + $(dec last_exec_op) )) 2)
        printf 'A %x\nB %x\n' $(( 0x$(lire $(( base + $(dec a) )) 8) & 0xFFFFFFFFFF )) $(( 0x$(lire $(( base +
         ↪  $(dec b) )) 8) & 0xFFFFFFFFFF ))
        for r in t sp st0 st1 pmst imr ifr bk brc rsa rea trn; do printf '%s %x\n' "$(printf '%s' $r | tr a-z A-Z)"
         ↪  0x$(lire $(( base + $(dec $r) )) 2); done
        i=0; for v in $(dd if=/proc/$pid/mem bs=1 skip=$(( base + $(dec ar) )) count=16 2>/dev/null | od -An -v
         ↪  -tx2); do printf 'AR%d %x\n' $i 0x$v; i=$(( i + 1 )); done
    } > "$VIVANT"
}
vivant_prog() { # vivant_prog <adr> <n> : n mots de la memoire programme du processus vivant, comme prog_fetch
    set -- $(cat "$DIR/vivant_prog.txt") "$1" "$2"      # pid base prog data pmst adr n
    # c54x_mem.c prog_fetch : PMST.OVLY (bit 5) et 0x60 <= adr < 0x2800 -> data[adr] (DARAM recouverte), sinon
     ↪  prog[adr]
    if [ $(( $5 & 0x20 )) != 0 ] && [ $6 -ge 96 ] && [ $6 -lt 10240 ]; then o=$4; else o=$3; fi
    dd if=/proc/$1/mem bs=1 skip=$(( $2 + o + $6 * 2 )) count=$(( $7 * 2 )) 2>/dev/null | od -An -v -tx2 | tr -s '
     ↪  \n' ' '
}
montrer_vivant() { # montrer_vivant <pid> : $VIVANT a l'ecran, avec les adresses du stop / go
    v() { awk -v k="$1" '$1==k {print substr($0, length($1) + 2)}' "$VIVANT"; }
    pc=$(v PC); xpc=$(v XPC); idle=$(v IDLE); pmst=$(v PMST); imr=$(v IMR); ifr=$(v IFR)
    gras "  DSP vivant (pid $1, $(vivant_coop $1 && echo 'pause cooperative SIGUSR1 disponible' || echo 'binaire
     ↪  ancien : SIGSTOP/SIGCONT')) : $(v ETAT)"
    printf '  fn=%s  PC=%x:%s  idle=%s running=%s  insn=%s  derniere instruction executee : P[%s]=%s\n' "$(v FN)"
     ↪  $(( 0x$xpc )) "$pc" "$idle" "$(v RUNNING)" "$(v INSN)" "$(v LAST_PC)" "$(v LAST_OP)"
    ( ETAT=$VIVANT; montrer_regs )
    iptr=$(( 0x$pmst >> 7 )); vbase=$(( iptr << 7 ))
    gras "  adresses de ce stop / go :"
    if [ "$idle" = 1 ]; then printf '    stop : PC=%x:%s  IDLE, le DSP attend une interruption (INTM=%d)\n' $((
     ↪  0x$xpc )) "$pc" $(( 0x${v_st1:-$(v ST1)} >> 11 & 1 ))
    else printf '    stop : PC=%x:%s  (en cours d execution, pas en IDLE)\n' $(( 0x$xpc )) "$pc"; fi
    printf '    go   : vecteurs armes (IMR=%04x IFR=%04x, IPTR=%03x -> table a %04x) :\n' $(( 0x$imr )) $(( 0x$ifr
     ↪  )) $iptr $vbase
    b=0; while [ $b -lt 16 ]; do
        if [ $(( 0x$imr >> b & 1 )) = 1 ]; then
            vec=$(( b + 16 )); adr=$(( vbase + vec * 4 )); nom=$(printf '%s\n' $IRQ_NOMS | awk -F= -v k=$b '$1==k
             ↪  {print $2}')
            cible=$(dis_mots "$(printf '%04x' $adr)" $(vivant_prog $adr 2) | head -1 | sed 's/^
             ↪  *[0-9a-f]*:[[:space:]]*//; s/[[:space:]]\+/ /g')
```

```
            printf '          bit %2d vec %2d -> %04x %-22s %s%s\n' $b $vec $adr "${nom:-?}" "${cible:+: $cible}"
             ↪  "$([ $(( 0x$ifr >> b & 1 )) = 1 ] && echo '   <- EN ATTENTE (IFR)')"
        fi; b=$(( b + 1 )); done
    lin=$(( (0x$xpc << 16) | 0x$pc ))
    if rom_fichier $lin P > /dev/null 2>&1; then gras "  ROM a PC :"; cmd_p "$(printf '%x:%04x' $(( lin >> 16 ))
     ↪  $(( lin & 0xffff )))" 4; fi
}
cmd_dsp() { # dsp : instantane du DSP en marche, sans l'arreter
    pid=$(vivant_pid) || { rouge "aucun c54x_exe --arm en marche (./run.sh)"; return 1; }
    vivant_accessible "$pid" || { vivant_diag "$pid"; return 1; }
    vivant_lire $pid "instantane sans arret$([ -f "$DIR/pause.$pid" ] && echo ' (en pause : go pour reprendre)')"
     ↪  || return 1; montrer_vivant $pid
}
cmd_stop() { # stop : pause du DSP en marche
    pid=$(vivant_pid) || { rouge "aucun c54x_exe --arm en marche (./run.sh)"; return 1; }
    vivant_accessible "$pid" || { vivant_diag "$pid"; return 1; }
    [ -f "$DIR/pause.$pid" ] && { echo "  deja en pause ($(cat "$DIR/pause.$pid")) : go pour reprendre"; cmd_dsp;
     ↪  return; }
    if vivant_coop $pid; then
        rm -f "$PAUSE_FICHIER"; kill -USR1 $pid || return 1; i=0
        while [ $i -lt 50 ] && ! grep -q '^ETAT pause' "$PAUSE_FICHIER" 2>/dev/null; do sleep 0.1; i=$(( i + 1 ));
         ↪  done
        grep -q '^ETAT pause' "$PAUSE_FICHIER" 2>/dev/null || { rouge "pas de pause apres 5 s (le DSP attend un GO
         ↪  de l'ARM ?) ; kill -USR2 $pid pour annuler"; return 1; }
        mode="SIGUSR1 (frontiere de trame, fn=$(awk '$1=="FN"{print $2}' "$PAUSE_FICHIER"))"
    else
        kill -STOP $pid || return 1; mode="SIGSTOP"
    fi
    echo "$mode" > "$DIR/pause.$pid"
    vivant_lire $pid "EN PAUSE par $mode" || return 1; montrer_vivant $pid
    echo "  (l'ARM/QEMU attend le DONE du DSP : toute la chaine est suspendue ; « go » pour reprendre)"
}
cmd_go() { # go : reprise du DSP en marche
    pid=$(vivant_pid) || { rouge "aucun c54x_exe --arm en marche"; rm -f "$DIR"/pause.*; return 1; }
    [ -f "$DIR/pause.$pid" ] || { echo "  pas de pause en cours (stop d'abord)"; return 1; }
    mode=$(cat "$DIR/pause.$pid"); rm -f "$DIR/pause.$pid"
    case $mode in SIGUSR1*) kill -USR2 $pid; i=0; while [ $i -lt 30 ] && ! grep -q '^ETAT reprise'
     ↪  "$PAUSE_FICHIER" 2>/dev/null; do sleep 0.1; i=$(( i + 1 )); done ;;
                *) kill -CONT $pid ;; esac
    avant_insn=$(awk '$1=="INSN"{print $2}' "$VIVANT" 2>/dev/null); avant_fn=$(awk '$1=="FN"{print $2}' "$VIVANT"
     ↪  2>/dev/null)
    echo "  go : reprise ($mode)"; sleep 0.3
    vivant_lire $pid "0,3 s apres la reprise" || return 1
    v() { awk -v k="$1" '$1==k {print $2}' "$VIVANT"; }
    printf '  0,3 s apres : fn=%s (+%d trames)  PC=%x:%s idle=%s  insn=%s (+%d)\n' "$(v FN)" $(( $(v FN) -
     ↪  ${avant_fn:-0} )) $(( 0x$(v XPC) )) "$(v PC)" "$(v IDLE)" "$(v INSN)" $(( $(v INSN) - ${avant_insn:-0} ))
}

# ------------------------------------------------------------------ ARM : le firmware layer1 sous gdb (telnet
 ↪  44444)
# run.sh lance QEMU avec -gdb tcp::1234 et la console tools/gdb-telnet.py sur le port 44444 : un
# gdb-multiarch attache au firmware (symboles layer1.highram.elf), panneau cmd.gdb + « demo ».
# Une seule session a la fois (le gdbstub n'accepte qu'un client) : si un telnet est deja ouvert,
# on ne force rien, on dit quoi y taper.
GDB_TELNET=${GDB_TELNET:-44444}
telnet_occupe() { ss -Htn state established "( sport = :$GDB_TELNET )" 2>/dev/null | grep -q . ; }
cmd_gdb() { # gdb <commande...> : envoyer une commande au gdb de l'ARM via le telnet, montrer la sortie
    [ $# -ge 1 ] || { echo "usage : gdb <commande du panneau osmocom>   ex : gdb fn | gdb l1 | gdb help_osmo | gdb
     ↪  demo etat"; return 1; }
    ss -Hltn "( sport = :$GDB_TELNET )" 2>/dev/null | grep -q . || { rouge "rien n'ecoute sur le port $GDB_TELNET
     ↪  (run.sh avec GDB=1)"; return 1; }
    if telnet_occupe; then rouge "une session telnet est deja ouverte sur le port $GDB_TELNET (une seule a la
     ↪  fois) : tape dans ce telnet :   $*"; return 1; fi
    gras "  telnet 127.0.0.1:$GDB_TELNET  (gdb) $*"
    python3 - "$GDB_TELNET" "${GDB_ATTENTE:-90}" "$*" <<'PY'
import re, socket, sys, time
```

```
port, attente, cmd = int(sys.argv[1]), float(sys.argv[2]), sys.argv[3]
s = socket.create_connection(("127.0.0.1", port), timeout=5)
def lire(jusqua, t):
    buf = b""; fin = time.time() + t
    while time.time() < fin:
        s.settimeout(max(0.1, fin - time.time()))
        try: d = s.recv(4096)
        except socket.timeout: break
        if not d: break
        buf += d
        if jusqua in buf: break
    return buf
lire(b"(gdb) ", 20)                                  # gdb attache, cible relancee (continue &)
s.sendall(b"\x03"); lire(b"(gdb) ", 5); time.sleep(0.5)   # Ctrl-C : la cible s'arrete, le prompt revient
s.sendall(cmd.encode() + b"\necho __FIN__\\n\n")
out = lire(b"__FIN__", attente)
s.sendall(b"quit\n"); time.sleep(0.3); s.close()      # a la fermeture, la console detache gdb : l'ARM repart
txt = re.sub(rb"\xff[\xfb-\xfe].|\xff\xfa.*?\xff\xf0", b"", out).decode("utf-8", "replace").replace("\r", "")
txt = txt.split("__FIN__")[0]
print("\n".join("    " + l for l in txt.split("\n") if not l.startswith("echo __FIN__") and l.strip() != "(gdb)"))
PY
    echo "  (session fermee : gdb detache, l'ARM tourne)"
}
# ------------------------------------------------------------- demo : la main sur le DSP vivant, puis sur
 ↪  l'ARM
DEMO_PAUSE=${DEMO_PAUSE:-4}    # secondes entre deux etapes de la demo, le temps de lire (0 = enchainer)
DEMO_STOP=${DEMO_STOP:-1.5}    # duree de la pause reelle du DSP : assez pour que la voix se coupe a l'oreille,
 ↪  pas plus
demo_pause() { [ "$DEMO_PAUSE" != 0 ] && { echo "    ... (suite dans $DEMO_PAUSE s)"; sleep "$DEMO_PAUSE"; }; }
A_CD=09d2       # a_cd[0] (flag FIRE) ; a_cd[3]=0x09d5 debut du bloc L2 de 23 octets (dsp_memcpy_from_api be=0)
# Dans le bloc descendant CCCH/BCCH : octet2 (type) = 0x09d6 bas ; cell_identity = 0x09d6 haut (MSB) | 0x09d7 bas
 ↪  (LSB).
shm_lire() { od -An -v -tx2 -j $(( (0x$1 - API_BASE) * 2 )) -N 2 "$API_SHM" 2>/dev/null | tr -d ' \n'; }  # 1 mot
 ↪  hex
shm_ecrire() { local v=$(( 0x$2 & 0xffff )); printf "\\$(printf '%03o' $(( v & 255 )))\\$(printf '%03o' $(( v >> 8
 ↪  )))" \
    | dd of="$API_SHM" bs=1 seek=$(( (0x$1 - API_BASE) * 2 )) conv=notrunc status=none 2>/dev/null; }
api_mot() { cmd_d_chaud "$1" 1 2>/dev/null | sed -n 's/.*\]=\([0-9a-fA-F]\{4\}\).*/\1/p' | head -1; }  #
 ↪  EXACTEMENT 4 hex, sinon vide
demo_si3() { # cell_id -> CELL_ID (defaut 777) dans SI3 : boucle vive, on reecrit a chaque bloc SI3 qui passe
    cible=${1:-777}
    [ -r "$API_SHM" ] || { rouge "    SI3 : pas de segment API RAM ; ./run.sh"; return 1; }
    [ -w "$API_SHM" ] || { rouge "    SI3 : segment en lecture seule ; chmod 0666 $API_SHM (ou sudo)"; return 1; }
    gras "    SI3 : relabellise le cell_id en $cible (0x$(printf %04x $cible)) a chaque bloc SI3 (type L2 0x1b)
     ↪  qui passe dans a_cd"
    fin=$(( $(date +%s%3N) + ${DEMO_HOLD_MS:-1500} )); n=0; av=""
    while [ $(date +%s%3N) -lt $fin ]; do
        w4=$(shm_lire 09d6); case $w4 in ''|*[!0-9a-fA-F]*) continue ;; esac
        [ $(( 0x$w4 & 0xff )) = $(( 0x1b )) ] || continue            # pas un SI3 a cet instant
        w5=$(shm_lire 09d7); case $w5 in ''|*[!0-9a-fA-F]*) continue ;; esac
        [ -z "$av" ] && { av=$(( (0x$w4 >> 8 & 0xff) << 8 | (0x$w5 & 0xff) )); echo "       1er SI3 vu : cell_id =
         ↪  $av (0x$(printf %04x $av))"; }
       shm_ecrire 09d6 $(printf '%04x' $(( (0x$w4 & 0x00ff) | ((cible >> 8 & 0xff) << 8) )))   # garde le type 0x1b
        shm_ecrire 09d7 $(printf '%04x' $(( (0x$w5 & 0xff00) | (cible & 0xff) )))              # garde l'octet LAI
        n=$(( n + 1 ))
    done
    if [ "$n" = 0 ]; then echo "       (aucun SI3 vu en ${DEMO_HOLD_MS:-1500} ms ; le mobile doit lire la BCCH/CCCH
     ↪  -- reessaie quand il campe)"; return 1; fi
    vert "       $n blocs SI3 relabellises en $cible pendant ${DEMO_HOLD_MS:-1500} ms (le flag FIRE reste valide,
     ↪  pas de CRC L3)"
    echo "       -> a chaque SI3 que le firmware lit maintenant, le cell_id vaut $cible"
}
LAC_W=09d9      # a_cd[7] : les 2 octets du LAC du SI3 (octet-echanges : a_cd[7] = LAC_LSB<<8 | LAC_MSB)
demo_lac() { # lac [valeur] : forcer le LAC du SI3 a valeur (defaut 777) EN VIF via le shm, sans relance ni rebuild
    cible=${1:-777}
    [ -r "$API_SHM" ] || { rouge "    LAC : pas de segment API RAM ; ./run.sh"; return 1; }
```

```
    [ -w "$API_SHM" ] || { rouge "    LAC : segment en lecture seule ; chmod 0666 $API_SHM (ou sudo)"; return 1; }
    sw=$(printf '%04x' $(( ((cible & 0xff) << 8) | ((cible >> 8) & 0xff) )))   # octet-echange pour a_cd[7]
    gras "    LAC : force a $cible (0x$(printf %04x $cible)) dans a_cd[7]=0x$sw, EN VIF, a chaque bloc SI3 (sans
     ↪  relance)"
    fin=$(( $(date +%s%3N) + ${DEMO_HOLD_MS:-3000} )); n=0; av=""
    while [ $(date +%s%3N) -lt $fin ]; do
        w4=$(shm_lire 09d6); case $w4 in ''|*[!0-9a-fA-F]*) continue ;; esac
        [ $(( 0x$w4 & 0xff )) = $(( 0x1b )) ] || continue             # SI3 seulement
        w7=$(shm_lire $LAC_W); case $w7 in ''|*[!0-9a-fA-F]*) continue ;; esac
        [ -z "$av" ] && { av=$(( ((0x$w7 & 0xff) << 8) | ((0x$w7 >> 8) & 0xff) )); echo "       1er SI3 vu : LAC =
         ↪  $av (0x$(printf %04x $av))"; }
        shm_ecrire $LAC_W $sw; n=$(( n + 1 ))
    done
    [ "$n" = 0 ] && { echo "       (aucun SI3 vu en ${DEMO_HOLD_MS:-3000} ms ; le mobile doit lire la BCCH)";
     ↪  return 1; }
    vert "       $n ecritures LAC=$cible pendant ${DEMO_HOLD_MS:-3000} ms, en vif (course avec le DSP : le mobile
     ↪  le voit si on gagne)"
    echo "       -> verifie : grep -ao 'CGI=[0-9-]*' /tmp/c54x-pont/mobile.log | tail  (champ LAC = 3e groupe)"
}
demo_sb() { # DEAD BEEF dans a_sch (le SB decode que l'ARM lit), en vif : on reecrit en boucle courte, on relit
    [ -r "$API_SHM" ] || { rouge "    SB : pas de segment API RAM ($API_SHM) ; ./run.sh"; return 1; }
    [ -w "$API_SHM" ] || { rouge "    SB : segment en lecture seule pour ce compte ; chmod 0666 $API_SHM (ou
     ↪  sudo)"; return 1; }
    a=0837   # a_sch[0], page R0 (voir d@ noms a_sch)
    gras "    SB : a_sch a $a (le SCH decode que le firmware lit en 0xffd00000), ecriture vive"
    echo "       avant : a_sch[0..1] = $(shm_lire $a) $(shm_lire 0838)"
    fin=$(( $(date +%s%3N) + ${DEMO_HOLD_MS:-1200} ))
    while [ $(date +%s%3N) -lt $fin ]; do shm_ecrire $a 40c3; shm_ecrire 0838 2026; done
    echo "       pendant : a_sch[0..1] = $(shm_lire $a) $(shm_lire 0838)   <- 40C3 2026 (CCC 2026) impose au SB
     ↪  que lit le firmware"
    vert "       (reecrit en boucle ${DEMO_HOLD_MS:-1200} ms ; le DSP reprend la main a la prochaine tache SB)"
}
demo_si3_preuve() { # attend le camping et montre le CGI vu par le mobile (doit finir par <cible>)
    cible=$1; log=${RUNDIR_OMBRE:-/tmp/c54x-pont}/mobile.log; dlog=${RUNDIR_OMBRE:-/tmp/c54x-pont}/dsp.log
    gras "    SI3 : preuve cote mobile -- cell_id force a $cible a la source (ombre descendante dans c54x_exe)"
    echo "       injections SI3 (DSP) : $(grep -ac 'ombre-dl.*SI3 cell_id' "$dlog" 2>/dev/null)   (0 si le mobile
     ↪  ne lit pas encore la BCCH)"
    i=0; cgi=""
    while [ $i -lt ${DEMO_CGI_ESSAIS:-40} ]; do
        cgi=$(grep -ao 'CGI=[0-9-]*' "$log" 2>/dev/null | tail -1)
        case $cgi in *-$cible) break ;; esac
        sleep 1; i=$(( i + 1 ))
    done
    case $cgi in
        *-$cible) vert "       mobile.log : $cgi   <- le telephone a decode le cell_id $cible (ombre reussie)" ;;
        '') echo "       (mobile pas encore campe apres ${DEMO_CGI_ESSAIS:-40} s ; grep CGI $log plus tard)" ;;
        *) echo "       mobile.log : $cgi (pas encore $cible ; injections=$(grep -ac 'ombre-dl.*SI3 cell_id'
         ↪  "$dlog" 2>/dev/null), laisse camper puis grep CGI $log)" ;;
    esac
}
demo_dsp() { # DSP vivant, SANS relance : force le LAC du SI3 en vif (shm), preuve CGI, stop/go
    pid=$(vivant_pid) || { rouge "demo dsp : aucun c54x_exe --arm en marche (./run.sh)"; return 1; }
    gras "===== DEMO DSP (c54x_exe --arm pid $pid, le vrai coeur C54x qui sert l'ARM) ====="
    log=${RUNDIR_OMBRE:-/tmp/c54x-pont}/mobile.log
    cible=${DEMO_CELL_ID:-777}
    gras "--- 1. LAC du SI3 force a $cible EN VIF (shm, SANS relance ; le banc reste campe)"
    av=$(grep -ao 'CGI=[0-9-]*' "$log" 2>/dev/null | tail -1); echo "       CGI avant : ${av:-<pas encore de camp>}"
    DEMO_HOLD_MS=${DEMO_LAC_MS:-5000} demo_lac "$cible"
    demo_pause
    gras "--- 2. preuve cote mobile : le LAC (3e groupe du CGI) doit passer a $cible"
    i=0; cgi=""
    while [ $i -lt ${DEMO_CGI_ESSAIS:-20} ]; do
        cgi=$(grep -ao 'CGI=[0-9-]*' "$log" 2>/dev/null | tail -1)
        case $cgi in CGI=*-*-$cible-*) break ;; esac
        DEMO_HOLD_MS=1500 demo_lac "$cible" >/dev/null 2>&1   # re-hammer pendant que le mobile relit la BCCH
        sleep 1; i=$(( i + 1 ))
```

```
    done
    case $cgi in
        CGI=*-*-$cible-*) vert "       mobile.log : $cgi   <- LAC = $cible, le telephone l'a decode (en vif, sans
         ↪  relance)" ;;
        '') echo "       (pas de CGI : le mobile n'est pas campe ; il faut qu'il lise la BCCH)" ;;
        *) echo "       mobile.log : $cgi (pas encore $cible au 3e groupe ; course avec le DSP, relance 'lac
         ↪  $cible' ou laisse camper)" ;;
    esac
    demo_pause
    if vivant_accessible "$pid"; then
        gras "--- 3. stop : pause du DSP, registres et LES ADRESSES"; cmd_stop || return 1; demo_pause
        gras "--- 4. go : reprise (fn, instructions)"; cmd_go
        gras "===== FIN DEMO DSP : LAC du SI3 force a $cible en vif, DSP relance, rien de casse, pas de relance du
         ↪  banc ====="
    else
        echo; vivant_diag "$pid"
        gras "===== FIN DEMO DSP : LAC force en vif. stop/go : harvard sous le compte du banc ====="
    fi
}
cmd_demo() { # demo [tout|arm [etat|trame|hello|go]|dsp] : d'abord l'ARM sous gdb (hello world from ram), puis le
 ↪  DSP vivant
    quoi=${1:-tout}; [ $# -gt 0 ] && shift
    case $quoi in
        dsp) demo_dsp ;;
        arm) gras "===== DEMO ARM (firmware layer1 dans QEMU, gdb par le telnet $GDB_TELNET) ====="; cmd_gdb demo
         ↪  "$@" ;;
        tout) gras "===== DEMO ARM D'ABORD (firmware layer1 dans QEMU, gdb par le telnet $GDB_TELNET : hello world
         ↪  from ram) ====="
              cmd_gdb demo "$@"; demo_pause; echo
              demo_dsp
              echo; gras "===== FIN DE LA DEMO : ARM en marche, DSP relance, rien de casse ====="; demo_pause ;;
        *) echo "usage : demo [tout | dsp | arm [etat|trame|hello|go]]"; return 1 ;;
    esac
}

# ------------------------------------------------------------------ API RAM a chaud (le DSP en marche, --arm)
API_SHM=${API_SHM:-/dev/shm/calypso_api_ram}   # = data[0x0800..0x27ff] du c54x_exe --arm, partage avec l'ARM
 ↪  (MAP_SHARED)
API_BASE=$(( 0x0800 )); API_MOTS=8192
CARTE=${CARTE:-$HERE/docs/carte-memoire.md}    # table « API RAM — tous les champs » (generee depuis l'ELF :
 ↪  scalaires seulement)
# Pages doubles (osmocom-bb include/calypso/dsp_api.h:18-23) : mot DSP = 0x0800 + (adresse ARM - 0xFFD00000)/2
#   W0 0x0800..0x0813  W1 0x0814..0x0827  (T_DB_MCU_TO_DSP, 17 mots utiles + 3 de bourrage, dsp_api.h:32-80)
#   R0 0x0828..0x083b  R1 0x083c..0x084f  (T_DB_DSP_TO_MCU, 20 mots, dsp_api.h:82-107)   NDB 0x08d4.. (268)  PARAM
 ↪  0x0c31.. (57)
api_page() { # api_page <mot DSP> -> W0|W1|R0|R1|NDB|PARAM|API
    a=$1; if [ $a -lt $(( 0x0814 )) ]; then echo W0; elif [ $a -lt $(( 0x0828 )) ]; then echo W1
    elif [ $a -lt $(( 0x083c )) ]; then echo R0; elif [ $a -lt $(( 0x0850 )) ]; then echo R1
    elif [ $a -ge $(( 0x08d4 )) ] && [ $a -lt $(( 0x08d4 + 268 )) ]; then echo NDB
    elif [ $a -ge $(( 0x0c31 )) ] && [ $a -lt $(( 0x0c31 + 57 )) ]; then echo PARAM; else echo API; fi
}
api_table() { # "nom 0xADDR page" : la carte generee + les tableaux que l'ELF ne nomme pas (dsp_api.h, API 36xx)
    grep '^| [^|]* | [a-z][a-z0-9_]* | 0x[0-9a-fA-F]* | 0xFFD[0-9A-F]* | 0x[0-9a-f]*' "$CARTE" 2>/dev/null | awk
     ↪  -F'|' '{gsub(/ /,"",$2); gsub(/ /,"",$3); gsub(/ /,"",$6); print $3, $6, $2}'
    for pp in 0x0800:W0 0x0814:W1; do pb=$(( ${pp%%:*} )); ppg=${pp#*:}                                  #
     ↪  dsp_api.h:67-70, bourrage 17..19
        printf 'a_a5fn[0] 0x%04x %s\na_a5fn[1] 0x%04x %s\n' $(( pb + 12 )) "$ppg" $(( pb + 13 )) "$ppg"
        printf '(bourrage) 0x%04x %s\n(bourrage) 0x%04x %s\n(bourrage) 0x%04x %s\n' $(( pb + 17 )) "$ppg" $(( pb +
         ↪  18 )) "$ppg" $(( pb + 19 )) "$ppg"; done
    for pp in 0x0828:R0 0x083c:R1; do pb=$(( ${pp%%:*} )); ppg=${pp#*:}; pi=0                            #
     ↪  dsp_api.h:97-99 (DSP 33..36)
       for nm in 'a_serv_demod[0]=D_TOA' 'a_serv_demod[1]=D_PM' 'a_serv_demod[2]=D_ANGLE' 'a_serv_demod[3]=D_SNR'
         ↪  'a_pm[0]' 'a_pm[1]' 'a_pm[2]' 'a_sch[0]' 'a_sch[1]' 'a_sch[2]' 'a_sch[3]' 'a_sch[4]'; do
           printf '%s 0x%04x %s\n' "$nm" $(( pb + 8 + pi )) "$ppg"; pi=$(( pi + 1 )); done; done
}
api_adr() { # api_adr <nom|adresse> -> entier (mot DSP), 1 si inconnu / hors fenetre
```

```
    case $1 in
        [a-z]*) v=$(api_table | awk -v n="$1" '$1==n || (n !~ /\[/ && $1 ~ ("^" n "(\\[0\\])?(=|$)")) {print $2;
         ↪  exit}'); [ -n "$v" ] || { rouge "champ inconnu ($CARTE + dsp_api.h) : $1   -> d@ noms <motif>" >&2;
         ↪  return 1; }; v=$(( v )) ;;
        *) v=$(hex "$1") || { rouge "adresse invalide : $1" >&2; return 1; } ;;
    esac
    [ $v -ge $API_BASE ] && [ $v -lt $(( API_BASE + API_MOTS )) ] || { rouge "$(printf '%04x' $v) : hors fenetre
     ↪  API (0800-27ff)" >&2; return 1; }
    printf '%d' $v
}
api_nom() { api_table | awk -v a="$(printf '0x%04x' $1)" '$2==a {print $3 " " $1; exit}'; }
api_vivant() { [ -r "$API_SHM" ] || { rouge "pas de segment $API_SHM (lancer ./run.sh ou ./c54x_exe --arm)";
 ↪  return 1; }
    pgrep -f 'c54x_exe.*--arm' > /dev/null || echo "  (aucun c54x_exe --arm en marche : contenu fige du dernier
     ↪  run)"; }
# Decodage des champs (sources : dsp_api.h:46-79 pour les bits, l1_environment.h:36-80 pour les taches,
# :339-353 et :370-386 pour d_ctrl_tch, :355-367 pour d_ctrl_abb / d_ctrl_system). Rien n'est devine.
TACHES="0=NO 1=IDLE1 5=FB 6=SB 8=TCH_FB 9=TCH_SB 10=RACH 11=AUL 12=DUL 13=TCHT 14=TCHA 15=TCH_DTX_UL 17=NBN 18=EBN
 ↪  19=NBS 20=EBS 21=NP 22=EP 24=ALLC 25=CB 26=DDL 27=ADL 28=TCHD 30=PNP 31=PEP 32=PALLC 33=CHECKSUM/PBS
 ↪  35=TST_NDB 36=TST_DB 37=INIT_VEGA 38=DSP_LOOP_C"
MODES="0=SIG_ONLY 1=TCH_FS 2=TCH_HS 3=TCH_96 4=TCH_48F 5=TCH_48H 6=TCH_24F 7=TCH_24H 8=TCH_EFR 9=TCH_144"
TYPES="0=INVALID 1=TCH_F 2=TCH_H 3=SDCCH_4 4=SDCCH_8"
decoder() { # decoder <nom> <hex16> -> texte (vide si rien a dire)
    n=${1%%=*}; v=$(( 0x$2 ))
    case $n in
        d_task_d|d_task_u|d_task_md|d_task_ra) t=$(printf '%s\n' $TACHES | awk -F= -v k=$v '$1==k {print $2}');
         ↪  printf 'tache %d%s' $v "${t:+ = $t}" ;;
        d_fn) printf 'fn_report=%d (FN dans la periode de rapport)  fn_sid=%d (FN%%104)' $(( v & 0xff )) $(( v >>
         ↪  8 )) ;;
        d_ctrl_tch) m=$(printf '%s\n' $MODES | awk -F= -v k=$(( v & 0xf )) '$1==k {print $2}'); ty=$(printf '%s\n'
         ↪  $TYPES | awk -F= -v k=$(( v >> 4 & 3 )) '$1==k {print $2}')
            printf 'chan_mode=%d%s chan_type=%d%s' $(( v & 0xf )) "${m:+($m)}" $(( v >> 4 & 3 )) "${ty:+($ty)}"
            [ $(( v >> 6 & 1 )) = 1 ] && printf ' RESET_SACCH'; [ $(( v >> 7 & 1 )) = 1 ] && printf ' VOCODER_ON'
            [ $(( v >> 8 & 1 )) = 1 ] && printf ' SYNC_TCH_UL'; [ $(( v >> 9 & 1 )) = 1 ] && printf ' SYNC_TCH_DL'
           [ $(( v >> 10 & 1 )) = 1 ] && printf ' STOP_TCH_UL'; [ $(( v >> 11 & 1 )) = 1 ] && printf ' STOP_TCH_DL'
            [ $(( v >> 12 & 3 )) != 0 ] && printf ' tch_loop=%d' $(( v >> 12 & 3 )); [ $(( v >> 15 & 1 )) = 1 ] &&
             ↪  printf ' SUBCHANNEL' ;;
        d_ctrl_abb) printf 'bits:'; [ $(( v & 1 )) = 1 ] && printf ' b_ramp(a_ramp dans la NDB)'; [ $(( v >> 3 & 1
         ↪  )) = 1 ] && printf ' b_apcdel(delais dans la NDB)'
            [ $(( v >> 4 & 1 )) = 1 ] && printf ' b_afc(d_afc valide)'; [ $(( v & ~0x19 )) != 0 ] && printf '
             ↪  +inconnus(%04x)' $(( v & ~0x19 & 0xffff )) ;;
        'a_a5fn[0]') printf 'T2=%d T3=%d (chiffrement A5)' $(( v & 0x1f )) $(( v >> 5 & 0x3f )) ;;
        'a_a5fn[1]') printf 'T1=%d (chiffrement A5)' $(( v & 0xfff )) ;;
        d_ctrl_system) printf 'b_tsq=%d (sequence d apprentissage)' $(( v & 7 )); [ $(( v >> 3 & 1 )) = 1 ] &&
         ↪  printf ' BCCH_FREQ_IND'; [ $(( v >> 15 & 1 )) = 1 ] && printf ' TASK_ABORT' ;;
        d_afc) [ $v -ge 32768 ] && v=$(( v - 65536 )); printf 'signe %d' $v ;;
        'a_serv_demod[0]') [ $v -ge 32768 ] && v=$(( v - 65536 )); printf 'TOA %d' $v ;;
        'a_serv_demod[2]') [ $v -ge 32768 ] && v=$(( v - 65536 )); printf 'angle %d (signe)' $v ;;
    esac
}
cmd_d_chaud() { # d@ <adr|nom> [n] : lire l'API RAM du DSP en marche, etiquetee par adresse absolue et decodee
    [ $# -ge 1 ] || { echo "usage : d@ <adresse|nom> [n]   ex : d@ 0800 20 | d@ d_task_d | d@ W1 | d@ noms d_task";
     ↪  return 1; }
    case $1 in
        noms) api_table | grep "${2:-.}" | awk '{printf "    %-5s %s  %s\n", $3, $2, $1}'; return ;;
        W0) set -- 0800 17 ;; W1) set -- 0814 17 ;; R0) set -- 0828 20 ;; R1) set -- 083c 20 ;;
    esac
    api_vivant || return 1; a=$(api_adr "$1") || return 1; n=${2:-8}
    api_table > "$DIR/table.txt"
    od -An -v -tx2 -w2 -j $(( (a - API_BASE) * 2 )) -N $(( n * 2 )) "$API_SHM" > "$DIR/mots.txt"
    while read -r mot; do
        [ -n "$mot" ] || continue
        set -- $(awk -v ad="$(printf '0x%04x' $a)" '$2==ad {print $3, $1; exit}' "$DIR/table.txt")
        pg=${1:-$(api_page $a)}; nom=${2:-}
        printf '    %-5s [%04x]=%s  %-18s %s\n' "$pg" $a "$mot" "$nom" "$([ -n "$nom" ] && decoder "$nom" "$mot")"
        a=$(( a + 1 ))
```

```
    done < "$DIR/mots.txt"
}
cmd_d_chaud_ecrire() { # d@! <adr|nom> <mots...> : ecrire dans l'API RAM du DSP en marche (vu par le DSP et par
 ↪  l'ARM)
    [ $# -ge 2 ] || { echo "usage : d@! <adresse|nom> <mot> [mot...]"; return 1; }
    api_vivant || return 1; a=$(api_adr "$1") || return 1; shift
    [ -w "$API_SHM" ] || { rouge "segment en lecture seule pour cet utilisateur"; return 1; }
    for m in "$@"; do v=$(hex "$m") || { rouge "mot invalide : $m"; return 1; }; v=$(( v & 0xffff ))
        [ $a -lt $(( API_BASE + API_MOTS )) ] || { rouge "debordement de la fenetre"; return 1; }
        printf "\\$(printf '%03o' $(( v & 255 )))\\$(printf '%03o' $(( v >> 8 )))" \
            | dd of="$API_SHM" bs=1 seek=$(( (a - API_BASE) * 2 )) conv=notrunc status=none || { rouge "ecriture
             ↪  refusee"; return 1; }
        printf '    [%04x] <- %04x  %s\n' $a $v "$(api_nom $a)"; a=$(( a + 1 )); done
    echo "  (ecrit dans le DSP en marche ; l'ARM voit le meme segment)"
}
# ------------------------------------------------------------------ « run » : les outils du banc tels quels
lancer() { # lancer <cmd...> : execute, journalise, montre la fin
    gras "  \$ $*"; "$@" > "$DIR/dernier_run.log" 2>&1; rc=$?
    n=$(wc -l < "$DIR/dernier_run.log")
    [ $n -gt 60 ] && echo "  ... ($n lignes, tout dans $DIR/dernier_run.log)"
    tail -n 60 "$DIR/dernier_run.log"; echo "  (code de retour $rc)"; return $rc
}
cmd_rejeu() { # rejeu [trames] [-v..] : c54x_exe --rejouer, acquisition FB/SB deterministe
    t=${1:-200}; shift 2>/dev/null; (cd "$HERE" && lancer ./c54x_exe --rejouer --trames "$t" --verbeux "$@"); }
cmd_banc() { # banc [test|--all|--list] [options dsp_tester]
    [ $# -ge 1 ] || set -- --list; (cd "$HERE" && lancer ./dsp_tester "$@"); }
cmd_isa() { # isa [-k MOT] : les exemples SPRU172C contre le coeur
    (cd "$HERE" && lancer ./isa_test tools/isa_tests.txt "$@"); }
RUNDIR_OMBRE=${RUNDIR_OMBRE:-/tmp/c54x-pont}
cmd_ombre() { # ombre [cell_id|off] : relance la chaine COMPLETE (run_si.sh : BTS + echafaudage + pont) avec
 ↪  l'ombre SI3
    n=${1:-${DEMO_CELL_ID:-777}}
    case $n in off|OFF) sv=""; lib="desactivee" ;; *[!0-9]*) rouge "usage : ombre [cell_id|off]  (defaut 777)";
     ↪  return 1 ;; *) sv="$n"; lib="LAC SI3 = $n" ;; esac
    # run_si.sh amene le mobile a decoder la BCCH (corrrelateur natif non verrouille -> echafaudage
     ↪  CAN_TOA/CAN_SB/AFC/LOCKSTEP).
    # run.sh nu (PONT=0) ne fait arriver AUCUN burst du BTS : pas de camp, pas de SI3, l'ombre ne se declenche
     ↪  jamais.
    lanceur="$HERE/run_si.sh"; mode_full=1
    [ -x "$lanceur" ] || { lanceur="$HERE/run.sh"; mode_full=0; rouge "  run_si.sh absent : run.sh nu (sans pont,
     ↪  le mobile ne campera pas)"; }
    gras "  ombre : relance de la chaine complete ($(basename "$lanceur")) avec CALYPSO_SHADOW_LAC=${sv:-<off>}
     ↪  ($lib)"
    logdir=${RUNDIR_OMBRE:-/tmp/c54x-pont}
    : > "$DIR/run_ombre.log"
    # [2026-10-06] eviter l'empilement : tuer tout lanceur fantome (run_si.sh/run.sh) AVANT de relancer,
    # sinon plusieurs chaines se disputent le socket DSP/le shm/le BTS et le mobile n'acquiert jamais
    # (FBSB result=255). On ne tue QUE les lanceurs, pas ce shell.
    for q in $(pgrep -f 'run_si\.sh|c54x_exe/run\.sh' 2>/dev/null); do [ "$q" = "$$" ] || kill "$q" 2>/dev/null;
     ↪  done
    ( cd "$HERE" && ./run.sh --stop ) >/dev/null 2>&1; sleep 1
    if [ "$mode_full" = 1 ]; then
        ( cd "$HERE" && CALYPSO_SHADOW_LAC="$sv" CALYPSO_SHADOW_SB="${OMBRE_SB:-}" setsid "$lanceur" --secondes
         ↪  "${OMBRE_SECS:-300}" ) > "$DIR/run_ombre.log" 2>&1 &
    else
        ( cd "$HERE" && ./run.sh --stop ) >/dev/null 2>&1
        ( cd "$HERE" && env MODE=dsp PONT=1 IQ=none CALYPSO_BSP_DIRECT_FEED=1 CALYPSO_PONT_LOCKSTEP=1 \
            CALYPSO_SHADOW_LAC="$sv" CALYPSO_SHADOW_SB="${OMBRE_SB:-}" setsid ./run.sh ) > "$DIR/run_ombre.log"
 ↪  2>&1 &
    fi
    echo "  banc en relance (BTS + echafaudage + pont ; journal $DIR/run_ombre.log)..."
    i=0; annonce=""
    while [ $i -lt ${OMBRE_ATTENTE:-50} ]; do
        [ -z "$sv" ] && { [ -S /tmp/calypso_dsp.sock ] && { echo "  banc relance (ombre off)"; return 0; }; sleep
         ↪  1; i=$((i+1)); continue; }
        annonce=$(grep -a "ombre-dl.*LAC SI3 force a $sv" "$logdir/dsp.log" 2>/dev/null | tail -1)
```

```
        [ -n "$annonce" ] && break
        sleep 1; i=$((i+1))
    done
    [ -n "$annonce" ] && vert "  ombre active : ${annonce#  }" || { echo "  (ombre pas encore annoncee apres
     ↪  ${OMBRE_ATTENTE:-50}s ; voir $DIR/run_ombre.log et $logdir/dsp.log)"; return 1; }
}
cmd_statut() { (cd "$HERE" && lancer ./run.sh --status); }

# ------------------------------------------------------------------ REPL
aide() {
    cat <<'FIN'
  cote P (programme)                          cote D (donnees)
    pas <insn|mots>   executer, garder l'etat     reg [REG val]   voir / fixer un registre
    essai <insn>      executer sans garder        d <adr> [n]     lire la memoire donnees
    p <adr> [n]       desassembler la ROM         d! <adr> <mots> poser des mots en D
    p! <adr> <insn>   poser des mots en P         drom <adr> [n]  lire la ROM donnees
    asm <insn>        assembler seulement         etat | raz      tout afficher | etat a zero
    dis <mots hex>    desassembler des mots
    cours [N]         courir dans la ROM jusqu'a arret / IDLE / N pas (etat du shell)
    arret [adr..|-]   points d'arret de « cours »
  DSP EN MARCHE (c54x_exe --arm)              ARM (firmware layer1 sous gdb, telnet 44444)
    stop              pause du DSP vivant + adresses   demo [tout|arm|dsp]  la main sur l'ARM (gdb) puis le DSP
    go                reprise                          gdb <cmd>   une commande du panneau (fn, l1, help_osmo)
    dsp               instantane sans arreter                     (demo arm etat|trame|hello|go : etape par etape)
    sb                DEAD BEEF dans a_sch (SB)        si3 [id]    relabellise le cell_id de SI3 (defaut 777)
    lac [id]          force le LAC du SI3 EN VIF (shm, sans relance ; course avec le DSP)
    ombre [id|off]    relance le banc, LAC du SI3 force a la SOURCE (deterministe ; mobile.log CGI 3e groupe)
    d@ <adr|nom|W0..R1> [n]  API RAM decodee     d@! <adr|nom> <mots>  y ecrire     d@ noms [motif]
  run                                           divers
    rejeu [trames]    c54x_exe --rejouer           journal [n]     derniers pas
    banc [test|--all] dsp_tester                   log [n]         avertissements du coeur
    isa [-k MOT]      isa_test SPRU172C            sauve|charge <f> etat sur disque
    statut            run.sh --status              menu | aide | q
  adresses : 1234, 0x1234, 1:8000 (page 1).  insn : mnemonique TI (LD #0x1234, A) ou mots hex (f020 1234).
FIN
}
dispatch() { # dispatch <ligne de commande>
    c=$1; shift
    case $c in
        pas) cmd_pas "$@" ;;      essai) cmd_essai "$@" ;;
        p) cmd_p "$@" ;;          'p!') cmd_p_pose "$@" ;;
        d) cmd_d "$@" ;;          'd!') cmd_d_pose "$@" ;;
        'd@') cmd_d_chaud "$@" ;;  'd@!') cmd_d_chaud_ecrire "$@" ;;
        drom) cmd_drom "$@" ;;    asm) cmd_asm "$@" ;;      dis) cmd_dis "$@" ;;
        reg) cmd_reg "$@" ;;      etat) cmd_etat ;;         raz) cmd_raz ;;
        cours) cmd_cours "$@" ;;  arret) cmd_arret "$@" ;;
        stop) cmd_stop ;;         go) cmd_go ;;             dsp) cmd_dsp ;;
        demo) cmd_demo "$@" ;;    gdb) cmd_gdb "$@" ;;
        sb|demo_sb) demo_sb ;;   si3|demo_si3) demo_si3 "$@" ;;   lac|demo_lac) demo_lac "$@" ;;
        demo_dsp) demo_dsp ;;    demo_arm) cmd_gdb demo "$@" ;;
        journal) cmd_journal "$@" ;; log) cmd_log "$@" ;;
        sauve) cmd_sauve "$@" ;;  charge) cmd_charge "$@" ;;
        rejeu) cmd_rejeu "$@" ;;  banc) cmd_banc "$@" ;;    isa) cmd_isa "$@" ;;  statut) cmd_statut ;;
        ombre) cmd_ombre "$@" ;;
        aide|'?'|help) aide ;;
        *) rouge "commande inconnue : $c (aide)"; return 1 ;;
    esac
}
repl() {
    verifier_outils; gras "shell Harvard c54x - « aide » pour les commandes, « menu » pour le menu, « q » pour
     ↪  sortir"
    while :; do
        pc=$(etat_reg PC); if [ -t 1 ]; then printf '\033[1mP:%s D>\033[0m ' "${pc:-9000}"; else printf 'P:%s D> '
         ↪  "${pc:-9000}"; fi
        read -r ligne || { echo; return 0; }
        set -- $ligne; [ $# -eq 0 ] && continue
```

```
        case $1 in q|quit|exit) return 0 ;; menu) return 2 ;; esac
        dispatch "$@"
    done
}

# ------------------------------------------------------------- menu interactif
pause() { printf '\n  [Entree pour continuer] '; read -r _; }
demander() { # demander <invite> [defaut] -> REP
    printf '  %s' "$1"; [ -n "${2:-}" ] && printf ' [%s]' "$2"; printf ' : '
    read -r REP; [ -z "$REP" ] && REP=${2:-}
}
menu() {
    verifier_outils
    while :; do
        clear 2>/dev/null
        pc=$(etat_reg PC)
        gras "================  SHELL HARVARD - DSP TMS320C54x (Calypso)  ================"
        printf '  PC=%s  A=%s  B=%s  SP=%s   %s\n' "${pc:-9000 (init)}" "$(etat_reg A)" "$(etat_reg B)"
         ↪  "$(etat_reg SP)" "$(drapeaux "$(etat_reg ST0)" "$(etat_reg ST1)")"
        cat <<'FIN'

  ESPACE P (programme)                    ESPACE D (donnees)
   1) executer une instruction (un pas)     5) voir les registres et les mots poses
   2) essayer une instruction (sans garder)  6) fixer un registre
   3) desassembler la ROM programme          7) lire la memoire donnees du coeur
   4) poser des mots en P (sous-routine)     8) poser des mots en D
                                             9) lire la ROM donnees (DROM / PDROM)
  COURIR DANS LA ROM (etat du shell)        DSP EN MARCHE (c54x_exe --arm, /dev/shm)
  10) cours : jusqu'a arret / IDLE / N pas  12) stop : pause du DSP vivant, registres, adresses
  11) arret : points d'arret                13) go : reprise          14) dsp : instantane
                                            15) lire l'API RAM a chaud (adresse ou nom)
                                            16) ecrire dans l'API RAM a chaud
  ARM (firmware layer1 sous gdb, telnet 44444)
  17) demo : la main sur l'ARM puis le DSP  18) gdb : une commande du panneau osmocom
  RUN (les outils du banc, tels quels)      DIVERS
  19) c54x_exe --rejouer (acquisition FB/SB) 23) journal des pas / avertissements du coeur
  20) dsp_tester (liste, un test, --all)     24) etat a zero / sauver / charger
  21) isa_test (exemples SPRU172C, filtre)   r) invite de commandes (REPL)
  22) run.sh --status                        q) quitter
FIN
        demander "choix"
        case $REP in
            1) demander "instruction (mnemonique TI ou mots hex)" "NOP"; cmd_pas $REP; pause ;;
            2) demander "instruction" "NOP"; cmd_essai $REP; pause ;;
            3) demander "adresse (7000.., e000.., 1:8000)" "7000"; a=$REP; demander "nombre de mots" 24; cmd_p
             ↪  "$a" "$REP"; pause ;;
            4) demander "adresse P" "9100"; a=$REP; demander "instruction ou mots"; cmd_p_pose "$a" $REP; pause ;;
            5) cmd_etat; pause ;;
            6) demander "registre" "AR3"; r=$REP; demander "valeur hex" "0"; cmd_reg "$r" "$REP"; pause ;;
            7) demander "adresse D" "0100"; a=$REP; demander "nombre de mots (<=24)" 8; cmd_d "$a" "$REP"; pause ;;
            8) demander "adresse D" "0100"; a=$REP; demander "mots hex (separes par des espaces)"; cmd_d_pose "$a"
             ↪  $REP; pause ;;
            9) demander "adresse (9000.., e000..)" "9000"; a=$REP; demander "nombre de mots" 32; cmd_drom "$a"
             ↪  "$REP"; pause ;;
            10) cmd_arret; demander "nombre de pas max" 200; cmd_cours "$REP"; pause ;;
            11) cmd_arret; demander "arret <adr..> | - (effacer) | vide" ""; [ -n "$REP" ] && cmd_arret $REP;
             ↪  pause ;;
            12) cmd_stop; pause ;;
            13) cmd_go; pause ;;
            14) cmd_dsp; pause ;;
            15) demander "adresse, nom, page W0|W1|R0|R1, ou noms <motif>" "W0"; a=$REP; demander "nombre de mots"
             ↪  8; case $a in noms*) cmd_d_chaud $a ;; *) cmd_d_chaud "$a" "$REP" ;; esac; pause ;;
            16) demander "adresse ou nom" "d_task_d"; a=$REP; demander "mots hex (separes par des espaces)";
             ↪  cmd_d_chaud_ecrire "$a" $REP; pause ;;
            17) demander "demo tout | dsp | arm [etat|trame|hello|go]" "tout"; cmd_demo $REP; pause ;;
            18) demander "commande gdb (fn, l1, dsp, help_osmo...)" "help_osmo"; cmd_gdb $REP; pause ;;
```

```
            19) demander "trames" 200; t=$REP; demander "verbosite (-v .. -vvvvvv, vide = aucune)" ""; cmd_rejeu
             ↪  "$t" $REP; pause ;;
            20) cmd_banc --list; demander "test (nom, --all, ou vide)" ""; [ -n "$REP" ] && { demander "options
             ↪  (--trace N, -v, --insns N...)" ""; o=$REP; cmd_banc $REP $o; }; pause ;;
            21) demander "filtre -k (vide = tout)" ""; [ -n "$REP" ] && cmd_isa -k "$REP" || cmd_isa; pause ;;
            22) cmd_statut; pause ;;
            23) cmd_journal 20; cmd_log 10; pause ;;
            24) demander "raz | sauve <fichier> | charge <fichier>" "raz"; dispatch $REP; pause ;;
            r|R) repl; [ $? = 2 ] || return 0 ;;
            q|Q) return 0 ;;
            *) ;;
        esac
    done
}

# ------------------------------------------------------------------ point d'entree
case ${1:-menu} in
    menu) menu ;;
    repl) repl ;;
    aide|-h|--help) sed -n '2,/^$/p' "$0" | sed 's/^# \{0,1\}//'; aide ;;
    *) dispatch "$@" ;;
esac
```

3.5 /opt/GSM/c54x_exe/install.sh

26582 octets, 489 lignes → 489 lignes

```
#!/bin/bash
# ============================================================================
#  install.sh - construit c54x_exe (le DSP Calypso hors QEMU) et pose sa ROM
# ============================================================================
#
#  Autonome : il faut les SOURCES de qosmo (clonees, pas construites :
#  c54x_exe compile hw/arm/calypso/l1-dsp/ de cet arbre) et libosmocore avec
#  libosmocoding, visibles de pkg-config - ce qu a deja qui a sa pile Osmocom.
#  C est aussi CE script qu appellent le Dockerfile d osmo-operator (stage l1),
#  son Dockerfile.run, start-docker.sh, l ISO et l installation native
#  (install_modules/45-calypso.sh) : une seule liste de commandes.
#
#      ./install.sh                tout : build, rom, verify
#      ./install.sh --check        prerequis et etat, NE MODIFIE RIEN
#      ./install.sh --list         les etapes et les valeurs retenues
#      ./install.sh --print-deps   les paquets apt, sur une ligne
#      sudo ./install.sh --deps    installe les paquets apt manquants, rien d autre
#      sudo ./install.sh --with-deps   les paquets manquants, puis tout
#      ./install.sh --only rom     une ou des etapes (liste a virgules)
#      ./install.sh --skip verify  toutes sauf celles-la
#      ./install.sh --reinstall    rejoue meme ce qui est deja fait
#      ./install.sh -v             la sortie des commandes a l ecran (sinon : journal)
#
#  Options, et la variable qui fait la meme chose (l option l emporte) :
#      --qosmo DIR      QOSMO        $GSM_ROOT/qosmo     les sources compilees
#      --rom-dir DIR    ROM_DIR      $GSM_ROOT           ou poser la ROM (repetable) ; sans
#                                                        l option : ROM_DIR ET rom/ de ce depot
#                                                        (Dockerfile:736). c54x_exe la lit dans
#                                                        --rom-dir, /opt/GSM par defaut (src/main.c:79)
#      --portable       PORTABLE=1   0                  CFLAGS du Dockerfile:735, sans -march=native :
#                                                        pour un binaire qui tournera sur un autre CPU
#      --dest DIR                                        copie ce depot dans DIR (sans .git ni binaires)
#                                                        et y construit : l arbre courant n est pas touche
#                       GSM_ROOT     /opt/GSM     ROM_VERSION 3606     LOG_DIR /tmp/c54x_exe-install
#                       FREECALYPSO_URL : miroir de la ROM (rom/fetch-rom.sh)
#
#  Correspondance avec osmo-operator/Dockerfile : chaque etape cite ses lignes,
```

```
#  celles du Dockerfile au commit 4266c8b, quand il portait encore ces commandes
#  (stage l1, 723-741 ; paquets 156-205). Il appelle desormais ce script.
# ------------------------------------------------------------------------------
set -uo pipefail
ICI="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
# ==============================================================================
#  LE CONTRAT - celui de osmo-operator/install_modules/_lib/inst.sh
# ==============================================================================
#  Copie IDENTIQUE dans qosmo/install.sh, c54x_exe/install.sh et
#  grgsm_exe/install.sh : chaque depot doit pouvoir s installer seul, sans
#  osmo-operator. Une correction ici se reporte dans les deux autres.
#
#  Quatre fonctions par etape, `run` seule obligatoire :
#     inst_<etape>_check    prerequis, LECTURE SEULE        0 ok · 1 manque · 4 sans objet
#     inst_<etape>_done     deja fait ?                     0 oui · 1 non
#     inst_<etape>_run      fait le travail                 0 ok · 1 echec · 3 deja fait
#     inst_<etape>_verify   controle APRES coup             0 ok · 1 echec
# ------------------------------------------------------------------------------
readonly INST_RC_OK=0 INST_RC_FAIL=1 INST_RC_DONE=3 INST_RC_NA=4
declare -a INST_ORDER=()
declare -A INST_DESC=()
INST_REGISTER() { INST_ORDER+=("$1"); INST_DESC[$1]="$2"; }

_INST_REASON=""
_INST_HINT=""
inst_ok()   { _INST_REASON=""   ; return $INST_RC_OK; }
inst_fail() { _INST_REASON="$*" ; return $INST_RC_FAIL; }
inst_done() { _INST_REASON="$*" ; return $INST_RC_DONE; }
inst_na()   { _INST_REASON="$*" ; return $INST_RC_NA; }
inst_hint() { _INST_HINT="$*"; }
inst_say()  { printf '%s\n' "$*"; }

have_cmd()  { command -v "$1" >/dev/null 2>&1; }
have_file() { [ -f "$1" ]; }
have_pkg()  { dpkg -s "$1" >/dev/null 2>&1; }
# Un chemin est inscriptible s il l est, ou si son plus proche parent existant
# l est (on pourra le creer).
ecrivable() { local d="$1"; while [ ! -e "$d" ] && [ "$d" != / ]; do d="$(dirname "$d")"; done; [ -w "$d" ]; }
# Chemin absolu sans exiger qu il existe (configure, make -C et les --dest
# changent de dossier : un chemin relatif n y voudrait plus rien dire).
absolu() { case "$1" in /*) printf '%s\n' "$1" ;; *) printf '%s\n' "$PWD/$1" ;; esac; }

# exige "ce qu il faut" "conseil si absent" commande... : une condition de
# prerequis. En --check elle est imprimee (ok / MANQUE + conseil) ; sinon elle
# est memorisee, et la premiere qui manque donne la raison de l echec.
_MANQUE=()
exige() {
    local quoi="$1" conseil="$2"; shift 2
    if "$@" >/dev/null 2>&1; then
        [ "$ACTION" = check ] && printf '      ok      %s\n' "$quoi"
        return 0
    fi
    if [ "$ACTION" = check ]; then
        printf '      %sMANQUE%s  %s\n' "$C_KO" "$C_Z" "$quoi"
        [ -n "$conseil" ] && printf '              → %s\n' "$conseil"
    fi
    [ ${#_MANQUE[@]} -eq 0 ] && [ -n "$conseil" ] && inst_hint "$conseil"
    _MANQUE+=("$quoi")
    return 1
}
# Verdict d une fonction check qui a appele exige.
bilan_exige() {
    [ ${#_MANQUE[@]} -eq 0 ] && { inst_ok; return $INST_RC_OK; }
    local plus=""; [ ${#_MANQUE[@]} -gt 1 ] && plus=" (+$(( ${#_MANQUE[@]} - 1 )) autre(s), --check)"
    inst_fail "manque : ${_MANQUE[0]}$plus"
}
```

```
# ═══════════════════════════════════════════════════════════════════
#  LE COMPOSANT - c54x_exe
# ═══════════════════════════════════════════════════════════════════
COMPOSANT=c54x_exe
: "${GSM_ROOT:=/opt/GSM}"
: "${QOSMO:=$GSM_ROOT/qosmo}"
: "${ROM_DIR:=$GSM_ROOT}"
: "${ROM_VERSION:=3606}"
: "${PORTABLE:=0}"
: "${LOG_DIR:=/tmp/c54x_exe-install}"
ROM_DIRS=()
DEST=""
# Dockerfile:735 - les drapeaux du Makefile, sans -march=native : un binaire
# -march=native construit ailleurs meurt en SIGILL au demarrage (Dockerfile:726-728).
CFLAGS_PORTABLE="-O3 -g -Wall -Werror=format -Werror=format-extra-args -Wno-unused-function -Wno-unused-variable
 ↪  -Wno-unused-but-set-variable -Wno-sign-compare"
ROMS="DROM PDROM PROM0 PROM1 PROM2 PROM3"

# Paquets apt, repris de la liste unique d osmo-operator/Dockerfile :
#   :159      build-essential pkg-config curl       compilation ; curl pour rom/fetch-rom.sh
#   :166      python3                               tools/dsp_txt2bin.py (conversion de la ROM)
# libosmocore / libosmocoding ne sont PAS des paquets apt dans l image : elle les
# compile (Dockerfile:260-311). Ici on les exige seulement, par pkg-config.
PAQUETS=(build-essential pkg-config python3 curl)

options_composant() {
    case "$1" in
        --qosmo)    _val "$@"; QOSMO="$2";       _n=2 ;;
        --rom-dir)  _val "$@"; ROM_DIRS+=("$2"); _n=2 ;;
        --dest)     _val "$@"; DEST="$2";        _n=2 ;;
        --portable) PORTABLE=1;                  _n=1 ;;
    esac
}

apres_options() {
    QOSMO="$(absolu "$QOSMO")"
    if [ ${#ROM_DIRS[@]} -eq 0 ]; then ROM_DIRS=("$ROM_DIR" "$ICI/rom"); fi
    local d l=(); for d in "${ROM_DIRS[@]}"; do
        d="$(absolu "${d%/}")"; in_liste "$d" "${l[@]+"${l[@]}"}" || l+=("$d")
    done
    ROM_DIRS=("${l[@]}")
    # --dest : on construit une COPIE. Le binaire en service (un banc qui tourne
    # sur $ICI/c54x_exe) n est pas reecrit sous ses pieds.
    if [ -n "$DEST" ] && [ "$ACTION" = install ]; then
        DEST="$(absolu "$DEST")"
        if [ "$DEST" != "$ICI" ]; then
            mkdir -p "$DEST" || { printf 'impossible de creer %s\n' "$DEST" >&2; exit 1; }
            tar -C "$ICI" --exclude=./.git --exclude=./c54x_exe --exclude='./c54x_exe.*' \
                --exclude=./isa_test -cf - . | tar -C "$DEST" -xf - \
                || { printf 'copie vers %s impossible\n' "$DEST" >&2; exit 1; }
            printf 'arbre copie dans %s (sans .git ni binaires) - la suite s y deroule\n' "$DEST"
            local a sauter=0 args=()
            for a in "${_ARGS_ORIG[@]}"; do
                [ "$sauter" = 1 ] && { sauter=0; continue; }
                [ "$a" = --dest ] && { sauter=1; continue; }
                args+=("$a")
            done
            # sans --rom-dir, la ROM va dans ROM_DIR et rom/ de la COPIE
            exec bash "$DEST/install.sh" "${args[@]+"${args[@]}"}"
        fi
    fi
}
in_liste() { local x="$1"; shift; local y; for y in "$@"; do [ "$x" = "$y" ] && return 0; done; return 1; }

resume_composant() {
    printf 'depot=%s\nQOSMO=%s\nROM (%s) -> %s\nCFLAGS=%s\n' "$ICI" "$QOSMO" "$ROM_VERSION" "${ROM_DIRS[*]}" \
```

```
        "$([ "$PORTABLE" = 1 ] && echo "portables (Dockerfile:735)" || echo "ceux du Makefile (-march=native)")"
}
avant_etapes() { :; }
apres_etapes() {
    [ -x "$ICI/c54x_exe" ] && printf '  binaire : %s/c54x_exe\n' "$ICI"
    printf '  ROM     : %s\n' "${ROM_DIRS[*]}"
    [ "${ROM_DIRS[0]}" = /opt/GSM ] || printf '  (hors /opt/GSM : lancer c54x_exe avec --rom-dir %s)\n'
     ↪  "${ROM_DIRS[0]}"
    printf '  lancer  : PONT=1 %s/run.sh   (wiki osmo-operator, Installer-c54x_exe)\n' "$ICI"
}

# La ROM d un dossier est-elle complete et conforme a rom/SHA256SUMS.<version> ?
# (le meme test que bon() de rom/fetch-rom.sh, plus l instantane de registres)
rom_valide() {
    local s="$ICI/rom/SHA256SUMS.$ROM_VERSION" r
    [ -f "$s" ] || return 1
    for r in $ROMS Registers; do [ -f "$1/calypso_dsp.$r.bin" ] || return 1; done
    ( cd "$1" && sha256sum -c --quiet "$s" ) >/dev/null 2>&1
}
# Une copie deja verifiee sur cette machine, parmi les destinations et rom/.
rom_source_locale() {
    local d; for d in "${ROM_DIRS[@]}" "$ICI/rom"; do rom_valide "$d" && { printf '%s\n' "$d"; return 0; }; done
    return 1
}

# ── build ──────────────────────────────────────────────────────────────
# Pas de « deja fait » : le Makefile recompile a CHAQUE appel, c est voulu
# (Makefile:46-52, les sources viennent d un autre depot).
INST_REGISTER build "Compilation de c54x_exe (sources de qosmo)"
_conseil_osmo() {
    if [ -f /usr/local/lib/pkgconfig/libosmocoding.pc ]; then
       printf 'export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig  (libosmocore est en /usr/local, Dockerfile:92-94)'
    else
        printf 'installez libosmocore (sources : https://gitea.osmocom.org/osmocom/libosmocore, ou le paquet
         ↪  libosmocore-dev de votre depot Osmocom)'
    fi
}
inst_build_check() {
    _MANQUE=()
    exige "sources qosmo ($QOSMO/hw/arm/calypso/l1-dsp/calypso_c54x.c)" \
        "git clone https://github.com/bbaranoff/qosmO $QOSMO, ou --qosmo DIR" \
        test -f "$QOSMO/hw/arm/calypso/l1-dsp/calypso_c54x.c"
    exige "cales hors QEMU ($QOSMO/contrib/hors-qemu/cales-qemu.c)" "qosmo trop ancien : git -C $QOSMO pull" \
        test -f "$QOSMO/contrib/hors-qemu/cales-qemu.c"
    exige "compilateur C (${CC:-gcc})" "apt-get install build-essential" have_cmd "${CC:-gcc}"
    exige "make" "apt-get install build-essential" have_cmd make
    exige "pkg-config" "apt-get install pkg-config" have_cmd pkg-config
    # Le Makefile (ligne 14) avale l erreur de pkg-config (2>/dev/null) : sans
    # ce controle, l echec n arrive qu au link, sur des gsm0503_* introuvables.
    exige "libosmocoding + libosmocore vus de pkg-config" "$(_conseil_osmo)" \
        pkg-config --exists libosmocoding libosmocore
    exige "$ICI inscriptible (le binaire y est ecrit)" "--dest DIR pour construire une copie ailleurs" test -w
     ↪  "$ICI"
    bilan_exige
}
inst_build_run() {
    # Dockerfile:733-735 : cd c54x_exe && make QOSMO=/opt/GSM/qosmo CFLAGS="<sans -march=native>"
    local args=("QOSMO=$QOSMO")
    [ "$PORTABLE" = 1 ] && args+=("CFLAGS=$CFLAGS_PORTABLE")
    make -C "$ICI" "${args[@]}" || { inst_fail "echec de compilation"; return $INST_RC_FAIL; }
    inst_ok
}
inst_build_verify() {
    [ -x "$ICI/c54x_exe" ] || { inst_fail "$ICI/c54x_exe absent apres make"; return $INST_RC_FAIL; }
    local nf
    nf="$(ldd "$ICI/c54x_exe" 2>/dev/null | awk '/not found/ {print $1}' | tr '\n' ' ')"
```

```
    [ -z "$nf" ] || { inst_hint "sudo ldconfig, ou LD_LIBRARY_PATH=/usr/local/lib"; inst_fail "bibliotheques
     ↪  introuvables : $nf"; return $INST_RC_FAIL; }
    inst_ok
}

# ── rom : la mask-ROM TI, telechargee (pas dans le depot) ──────────────────
INST_REGISTER rom "ROM DSP $ROM_VERSION (FreeCalypso, somme verifiee)"
inst_rom_done() { local d; for d in "${ROM_DIRS[@]}"; do rom_valide "$d" || return 1; done; }
inst_rom_check() {
    _MANQUE=()
    exige "rom/fetch-rom.sh et rom/SHA256SUMS.$ROM_VERSION" "" test -f "$ICI/rom/fetch-rom.sh" -a -f
     ↪  "$ICI/rom/SHA256SUMS.$ROM_VERSION"
    exige "sha256sum" "apt-get install coreutils" have_cmd sha256sum
    local d; for d in "${ROM_DIRS[@]}"; do
        exige "$d inscriptible (ROM)" "sudo, ou --rom-dir DIR" ecrivable "$d"
    done
    # Le reseau, python3 et curl ne servent que s il n y a nulle part de copie valide.
    if ! rom_source_locale >/dev/null; then
        exige "python3 (tools/dsp_txt2bin.py)" "apt-get install python3" have_cmd python3
        exige "curl ou wget (telechargement FreeCalypso)" "apt-get install curl" sh -c 'command -v curl || command
         ↪  -v wget'
    fi
    bilan_exige
}
inst_rom_run() {
    # 1. Une copie deja verifiee sur cette machine ? on la recopie, sans reseau.
    #    (Dockerfile.run:188 et start-docker.sh faisaient cp c54x_exe/rom/*.bin /opt/GSM/.)
    local src d r args=()
    if src="$(rom_source_locale)"; then
        for d in "${ROM_DIRS[@]}"; do
            [ "$d" = "$src" ] && continue
            rom_valide "$d" && continue
            mkdir -p "$d" || { inst_fail "impossible de creer $d"; return $INST_RC_FAIL; }
            for r in $ROMS Registers; do
                install -m644 "$src/calypso_dsp.$r.bin" "$d/calypso_dsp.$r.bin" \
                    || { inst_fail "copie de calypso_dsp.$r.bin vers $d impossible"; return $INST_RC_FAIL; }
            done
            inst_say "ROM DSP $ROM_VERSION recopiee de $src vers $d (somme verifiee)"
        done
    fi
    # 2. rom/fetch-rom.sh fait le reste : telecharge, convertit, verifie, depose
    #    (Dockerfile:736). Tout deja en place : il le dit et ne telecharge rien.
    for d in "${ROM_DIRS[@]}"; do args+=(--dest "$d"); done
    bash "$ICI/rom/fetch-rom.sh" --version "$ROM_VERSION" "${args[@]}" \
        || { inst_hint "reseau ? un miroir : FREECALYPSO_URL=..."; inst_fail "rom/fetch-rom.sh en echec"; return
         ↪  $INST_RC_FAIL; }
    inst_ok
}
inst_rom_verify() { inst_rom_done && inst_ok || inst_fail "ROM incomplete ou somme differente dans
 ↪  ${ROM_DIRS[*]}"; }

# ── verify : le DSP seul ──────────────────────────────────────────────────
# README.md:66 : « la ROM seule, sans ARM : boote-t-elle ? ». Hors --arm,
# c54x_exe travaille sur une API RAM PRIVEE : ni /dev/shm, ni socket, ni
# /tmp/c54x-pont - un banc en marche a cote n en voit rien. Les CALYPSO_* de
# l appelant (sondes, traces) sont retires pour que ca le reste.
INST_REGISTER verify "Controle : c54x_exe --trames 200 (la ROM seule)"
sans_calypso_env() {
    local v u=(); for v in $(compgen -e); do case "$v" in CALYPSO_*) u+=(-u "$v") ;; esac; done
    env "${u[@]+"${u[@]}"}" "$@"
}
inst_verify_check() {
    _MANQUE=()
    etape_retenue build || exige "binaire $ICI/c54x_exe" "./install.sh --only build" test -x "$ICI/c54x_exe"
    etape_retenue rom   || exige "ROM DSP dans ${ROM_DIRS[0]}" "./install.sh --only rom" rom_valide
     ↪  "${ROM_DIRS[0]}"
```

```
    bilan_exige
}
inst_verify_run() {
    local out rc
    out="$(sans_calypso_env timeout 300 "$ICI/c54x_exe" --rom-dir "${ROM_DIRS[0]}" --trames 200 2>&1)"; rc=$?
    printf '%s\n' "$out" | tail -n 14
    [ $rc -eq 0 ] || { inst_fail "c54x_exe --trames 200 rend $rc"; return $INST_RC_FAIL; }
    grep -q 'sections chargees' <<<"$out" && grep -q 'bilan sur 200 trames' <<<"$out" \
        || { inst_fail "sortie inattendue (ni « sections chargees » ni « bilan sur 200 trames »)"; return
        ↪  $INST_RC_FAIL; }
    inst_ok
}
# ============================================================================
#  LE MOTEUR - copie IDENTIQUE dans les trois depots (voir LE CONTRAT)
# ============================================================================
#  Meme deroule que osmo-operator/install.sh : pour chaque etape retenue,
#  « deja fait ? » (sauf --reinstall), prerequis, travail, controle. Le premier
#  echec arrete tout, dit pourquoi et montre la fin du journal de l etape.
#  Le composant fournit : COMPOSANT, PAQUETS, LOG_DIR, les etapes, et les
#  crochets options_composant / apres_options / resume_composant /
#  avant_etapes / apres_etapes.
# ----------------------------------------------------------------------------
ACTION=install ONLY="" SKIP="" WITH_DEPS=0
: "${REINSTALL:=0}" "${VERBOSE:=0}"
_ARGS_ORIG=("$@")
# _val OPTION [VALEUR] : une option qui attend une valeur doit l avoir.
_val() { [ $# -ge 2 ] && [ -n "$2" ] || { printf '%s : valeur attendue   (--help)\n' "$1" >&2; exit 2; }; }
while [ $# -gt 0 ]; do
    case "$1" in
        -h|--help)    ACTION=help ;;
        --check)      ACTION=check ;;
        --list)       ACTION=list ;;
        --deps)       ACTION=deps ;;
        --print-deps) ACTION=print-deps ;;
        --with-deps)  WITH_DEPS=1 ;;
        --only)       _val "$@"; ONLY="$2"; shift ;;
        --skip)       _val "$@"; SKIP="$2"; shift ;;
        --reinstall)  REINSTALL=1 ;;
        -v|--verbose) VERBOSE=1 ;;
        *)  _n=0; options_composant "$@"
            [ "$_n" -gt 0 ] || { printf 'option inconnue : %s   (--help)\n' "$1" >&2; exit 2; }
            shift $((_n - 1)) ;;
    esac
    shift
done

if [ "$ACTION" = help ]; then
    awk 'NR > 2 && /^# -{10}/ { exit } NR > 2 && !/^# =+$/ { sub(/^# ?/, ""); print }' "$0"
    exit 0
fi
apres_options

if [ -t 1 ] && [ -z "${NO_COLOR:-}" ]; then
    TTY=1; C_OK=$'\033[32m'; C_KO=$'\033[31m'; C_SK=$'\033[33m'; C_DIM=$'\033[2m'; C_Z=$'\033[0m'
else
    TTY=0; C_OK=""; C_KO=""; C_SK=""; C_DIM=""; C_Z=""
fi
begin() { if [ $TTY -eq 1 ] && [ "$VERBOSE" != 1 ]; then printf '[ %s..%s ] %s' "$C_DIM" "$C_Z" "$1"; else printf
 ↪  '[ .. ] %s\n' "$1"; fi; }
end()   { [ $TTY -eq 1 ] && [ "$VERBOSE" != 1 ] && printf '\r\033[K'
          printf '[%s%s%s] %s' "$2" "$1" "$C_Z" "$3"
          [ -n "${4:-}" ] && printf ' %s(%s)%s' "$C_DIM" "$4" "$C_Z"; printf '\n'; }

# --- selection des etapes (--only / --skip, listes a virgules) ---------------
in_csv() { case ",$1," in *",$2,"*) return 0;; esac; return 1; }
for _s in ${ONLY//,/ } ${SKIP//,/ }; do
```

```
    [ -n "${INST_DESC[$_s]+x}" ] || { printf 'etape inconnue : %s   (etapes : %s)\n' "$_s" "${INST_ORDER[*]}" >&2;
     ↪  exit 2; }
done
SELECTED=()
for _s in "${INST_ORDER[@]}"; do
    [ -n "$ONLY" ] && ! in_csv "$ONLY" "$_s" && continue
    [ -n "$SKIP" ] && in_csv "$SKIP" "$_s" && continue
    SELECTED+=("$_s")
done
# Une etape qui produit ce qu une autre exige : en --check, on ne reproche pas
# a la seconde l absence de ce que la premiere va justement fabriquer.
etape_retenue() { in_csv "$(IFS=,; echo "${SELECTED[*]}")" "$1"; }

# --- paquets apt -------------------------------------------------------------
# rend la liste des absents ; code 2 sans dpkg (pas une Debian/Ubuntu)
paquets_absents() {
    have_cmd dpkg || return 2
    local p; for p in "${PAQUETS[@]}"; do have_pkg "$p" || printf '%s ' "$p"; done
    return 0
}
installer_paquets() {
    local manque rc
    manque="$(paquets_absents)"; rc=$?
    if [ $rc -eq 2 ]; then
        printf 'dpkg absent : installez l equivalent de : %s\n' "${PAQUETS[*]}" >&2; return 1
    fi
    if [ -z "${manque// }" ]; then
        printf 'paquets apt : les %d sont presents\n' "${#PAQUETS[@]}"; return 0
    fi
    if [ "$(id -u)" -ne 0 ]; then
        printf 'paquets apt manquants : %s\n  sudo apt-get install -y --no-install-recommends %s\n' "$manque"
         ↪  "$manque" >&2
        return 1
    fi
    printf 'apt-get install : %s\n' "$manque"
    apt-get update -qq && DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends $manque ||
     ↪  return 1
    manque="$(paquets_absents)"
    [ -z "${manque// }" ] || { printf 'toujours absents apres apt-get : %s\n' "$manque" >&2; return 1; }
}

case "$ACTION" in
    print-deps) printf '%s\n' "${PAQUETS[*]}"; exit 0 ;;
    deps)       installer_paquets; exit $? ;;
    list)
        printf '%s - %d etape(s), dans cet ordre\n\n' "$COMPOSANT" "${#SELECTED[@]}"
        for _s in "${SELECTED[@]}"; do printf '  %-10s %s\n' "$_s" "${INST_DESC[$_s]}"; done
        printf '\n'; resume_composant | sed 's/^/  /'
        exit 0 ;;
    check)
        printf '%s : prerequis et etat (rien n est modifie)\n\n' "$COMPOSANT"
        resume_composant | sed 's/^/  /'
        _manque="$(paquets_absents)"; _rc=$?
        if [ $_rc -eq 2 ]; then printf '\n  paquets apt : dpkg absent, non verifies (%s)\n' "${PAQUETS[*]}"
        elif [ -n "${_manque// }" ]; then printf '\n  paquets apt absents (installes autrement ? sinon --deps) :
         ↪  %s\n' "$_manque"
        else printf '\n  paquets apt : les %d sont presents\n' "${#PAQUETS[@]}"; fi
        _rc=0
        for _s in "${SELECTED[@]}"; do
            _p="inst_${_s//-/_}"
            if ! declare -F "${_p}_done" >/dev/null; then _etat="pas de controle"
            elif "${_p}_done" >/dev/null 2>&1; then _etat="deja fait"
            else _etat="a faire"; fi
            printf '\n  %-10s %s  %s[%s]%s\n' "$_s" "${INST_DESC[$_s]}" "$C_DIM" "$_etat" "$C_Z"
            declare -F "${_p}_check" >/dev/null || continue
            _INST_REASON=""; _INST_HINT=""; _MANQUE=()
```

```
            "${_p}_check"; _r=$?
            case $_r in
                "$INST_RC_FAIL") _rc=1 ;;
                "$INST_RC_NA")   printf '      sans objet : %s\n' "$_INST_REASON" ;;
            esac
        done
        printf '\n'
        if [ $_rc -eq 0 ]; then printf '%sprerequis satisfaits%s\n' "$C_OK" "$C_Z"
        else printf '%sprerequis manquants%s (voir MANQUE ci-dessus)\n' "$C_KO" "$C_Z"; fi
        exit $_rc ;;
esac

# --- installation -----------------------------------------------------------
if [ "$WITH_DEPS" = 1 ]; then
    installer_paquets || { printf 'paquets apt : echec - rien n a ete construit\n' >&2; exit 1; }
else
    _manque="$(paquets_absents 2>/dev/null)"
    [ -n "${_manque// }" ] && printf '%snote : paquets apt absents (installes autrement ? sinon --with-deps) :
     ↪  %s%s\n' "$C_DIM" "$_manque" "$C_Z"
fi
if [ "$VERBOSE" != 1 ]; then
    mkdir -p "$LOG_DIR" || { printf 'journal impossible : %s (LOG_DIR=...)\n' "$LOG_DIR" >&2; exit 1; }
fi
# _jouer FONCTION : la sortie va au journal de l etape, ou a l ecran en -v.
# Pas de tube : la fonction tourne dans CE shell (sa raison d echec survit).
_jouer() { if [ "$VERBOSE" = 1 ]; then "$1"; else "$1" >>"$_log" 2>&1; fi; }
_echec() {   # _echec ETAPE RAISON [journal] : dit tout ce qu on sait, et s arrete
    end FAIL "$C_KO" "${INST_DESC[$1]}" "$2"
    [ -n "$_INST_HINT" ] && printf '       → %s\n' "$_INST_HINT"
    # la fin du journal, quand c est le travail (run, verify) qui a echoue
    if [ -n "${3:-}" ] && [ "$VERBOSE" != 1 ] && [ -s "$_log" ]; then
        printf '       %sjournal : %s (fin ci-dessous)%s\n' "$C_DIM" "$_log" "$C_Z"
        tail -n 25 "$_log" | sed 's/^/       | /'
    fi
    printf '\n%s : installation interrompue a l etape %s\n' "$COMPOSANT" "$1"
    exit 1
}
avant_etapes
_nb_ok=0; _nb_skip=0
for _s in "${SELECTED[@]}"; do
    _p="inst_${_s//-/_}"; _log="$LOG_DIR/$_s.log"
    _INST_REASON=""; _INST_HINT=""; _MANQUE=()
    [ "$VERBOSE" = 1 ] || printf '\n===== %s  %s =====\n' "$(date '+%F %T')" "$_s" >>"$_log"
    begin "${INST_DESC[$_s]}"
    if [ "$REINSTALL" != 1 ] && declare -F "${_p}_done" >/dev/null && _jouer "${_p}_done"; then
        end SKIP "$C_SK" "${INST_DESC[$_s]}" "deja fait"; _nb_skip=$((_nb_skip + 1)); continue
    fi
    if declare -F "${_p}_check" >/dev/null; then
        _jouer "${_p}_check"; _r=$?
        case $_r in
            "$INST_RC_OK")   ;;
            "$INST_RC_NA")   end SKIP "$C_SK" "${INST_DESC[$_s]}" "${_INST_REASON:-sans objet}"
                             _nb_skip=$((_nb_skip + 1)); continue ;;
            *)               _echec "$_s" "${_INST_REASON:-prerequis non satisfait}" ;;
        esac
    fi
    _jouer "${_p}_run"; _r=$?
    case $_r in
        "$INST_RC_OK")   ;;
        "$INST_RC_DONE"|"$INST_RC_NA")
                         end SKIP "$C_SK" "${INST_DESC[$_s]}" "${_INST_REASON:-rien a faire}"
                         _nb_skip=$((_nb_skip + 1)); continue ;;
        *)               _echec "$_s" "${_INST_REASON:-echec}" journal ;;
    esac
    # installer n est pas avoir installe : on controle apres coup
    if declare -F "${_p}_verify" >/dev/null && ! _jouer "${_p}_verify"; then
```

```
        _echec "$_s" "faite, mais le controle echoue : ${_INST_REASON:-}" journal
    fi
    end " OK " "$C_OK" "${INST_DESC[$_s]}" "${_INST_REASON:-}"; _nb_ok=$((_nb_ok + 1))
done
printf '\n%s : %d ok · %d ignoree(s) · 0 echec\n' "$COMPOSANT" "$_nb_ok" "$_nb_skip"
apres_etapes
exit 0
```

3.6 /opt/GSM/c54x_exe/mobile_pont.cfg

4216 octets, 167 lignes → 167 lignes

```
# mobile.cfg.template — avec placeholders
# KI : "ki comp128 XX XX XX ... XX"  16 octets, octet 15=ms_idx, 16=op_id
#
# AUDIO [2026-09-22] : comme le run sans --dsp (/root/.osmocom/bb/mobile.cfg)
# sauf le format de trame TCH, plus la voie donnees.
#   tch-voice  io-handler gapk, io-tch-format ti (le DSP vocode ; c'est
#              « rtp » sans --dsp, ou c'est pont.py qui code), ALSA
#              gsm_out/gsm_in.
#   tch-data   io-handler unix-sock, /tmp/ms_data (CSD 9600 async, 8N1).
#   L'ordre tch-voice puis tch-data est celui que le mobile lui-meme ecrit.
#   `io-tch-format` n'est valide que derriere gapk ou unix-sock : sous l1phy le
#   mobile la refuse (« This parameter is only valid for GAPK ») et abandonne
#   tout le fichier.
#
# SOCKETS : les memes que le run sans --dsp (/tmp/osmocom_l2, /tmp/osmocom_sap)
# et non plus les variantes « _pont ». En montage DSP le module `l2` du fork
# n'est pas joue, donc aucun autre mobile ne les prend. Lancer a la main un
# banc grgsm ET ce banc-ci en meme temps les ferait se disputer : c'est le
# compromis assume pour n'avoir qu'un seul jeu de noms.
!
line vty
 no login
 bind 127.0.0.1 4247
!
gps device /dev/ttyACM0
gps baudrate default
no gps enable
!
no hide-default
!
!
log stderr
 logging filter all 1
 logging color 1
 logging timestamp 1
 logging print category 1
 logging level mm debug
 logging level rr debug
 logging level cc debug
 logging level sms debug
!
log gsmtap 172.20.0.1
 logging filter all 1
 logging color 0
 logging timestamp 0
 logging print category 1
 logging level mm debug
 logging level rr debug
 logging level cc debug
 logging level sms debug
!
ms 1
```

```
 layer2-socket /tmp/osmocom_l2
 sap-socket /tmp/osmocom_sap
 sim test
 network-selection-mode auto
 imei 358925008967594 0
 imei-fixed
 no emergency-imsi
 sms-service-center +3366612340001
 no call-waiting
 no auto-answer
 no force-rekey
 no clip
 no clir
 tx-power auto
 no simulated-delay
 stick 1
 location-updating
 neighbour-measurement
 codec full-speed prefer
 codec half-speed
 no abbrev
 support
  sms
  a5/1
  a5/2
  p-gsm
  e-gsm
  no r-gsm
  no gsm-850
  dcs
  no pcs
  class-900 4
  class-850 4
  class-dcs 1
  class-pcs 1
  channel-capability sdcch+tchf+tchh
  full-speech-v1
  full-speech-v2
  half-speech-v1
  min-rxlev -106
  dsc-max 90
  no skip-max-per-band
 test-sim
  imsi 001010001000001
  ki comp128 00 11 22 33 44 55 66 77 88 99 aa bb cc dd 01 01
  no barred-access
  rplmn 001 01
 tch-voice
  io-handler gapk
! [2026-09-22] FORMAT DE TRAME TCH : « ti » ICI, « rtp » SANS --dsp.
! Ce n'est pas une preference, c'est QUI fait le vocodage. Sur ce banc-ci le
! vrai DSP tourne (c54x_exe, mask-ROM TI) et rend ses trames dans SA
! disposition, celle des telephones Calypso : « ti ».
! Le montage gr-gsm, lui, n'a pas de DSP — c'est pont.py qui code avec
! libosmocoding (gsm0503_tch_fr_encode/decode), laquelle parle RTP/RFC3551,
! 33 octets ouverts par la signature 0xD. D'ou le « rtp » de
! /root/.osmocom/bb/mobile.cfg et sa note du 2026-08-27 : elle vaut pour CE
! montage-la, pas pour celui-ci. Se tromper de cote ne leve aucun compteur,
! les trames passent — elles sont simplement lues avec le mauvais plan, et le
! son sort robotise.
  io-tch-format ti
  alsa-output-dev gsm_out
  alsa-input-dev gsm_in
 tch-data
  io-handler unix-sock
  io-tch-format ti
```

```
  unix-socket /tmp/ms_data
  call-params type-rate 71
  call-params ce transparent
  call-params async
  call-params async nr-stop-bits 1
  call-params async nr-data-bits 8
  call-params async parity none
 no shutdown
!
! GSMTAP configuration
!
gsmtap
 remote-host 172.20.0.1
 no local-host
 lchan sacch
 lchan lsacch
 lchan sacch/4
 lchan sacch/8
 lchan sacch/f
 lchan sacch/h
 lchan unknown
 lchan bcch
 lchan ccch
 lchan rach
 lchan agch
 lchan pch
 lchan sdcch
 lchan sdcch/4
 lchan sdcch/8
 lchan facch/f
 lchan facch/h
 lchan pacch
 lchan cbch
 lchan pdch
 lchan pttch
 lchan tch/f
 lchan tch/h
 category gprs dl-unknown
 category gprs dl-dummy
 category gprs dl-ctrl
 category gprs dl-data-gprs
 category gprs dl-data-egprs
 category gprs ul-unknown
 category gprs ul-dummy
 category gprs ul-ctrl
 category gprs ul-data-gprs
 category gprs ul-data-egprs
end
```

3.7 /opt/GSM/c54x_exe/repro_toa.sh

5217 octets, 102 lignes → 102 lignes

```
#!/bin/bash
# Reproduces the "magic 23" evidence and writes everything to toa.log.
# Idempotent: skips downloads and builds already present.
# Needs: curl, python3, gcc, make; osmocom-bb is cloned if absent.
set -u
LOG=toa.log
: > "$LOG"
say() { printf '%s\n' "$*" | tee -a "$LOG"; }
run() { say "\$ $*"; "$@" 2>&1 | tee -a "$LOG"; say ""; }

say "# magic 23 reproduction — $(date -u +%Y-%m-%dT%H:%MZ) — $(uname -srm)"
```

```
say ""

# ------------------------------------------------------------- 1. ROM
say "## 1. ROM dump -> PROM0.bin"
[ -f dsp-rom-3606-dump.txt ] || run curl -sO ftp://ftp.freecalypso.org/pub/GSM/Calypso/dsp-rom-3606-dump.txt
run sha256sum dsp-rom-3606-dump.txt
python3 - <<'E' 2>&1 | tee -a "$LOG"
import re,struct
sec=None;w={}
for l in open("dsp-rom-3606-dump.txt",errors="replace"):
    m=re.match(r"DSP dump: (\w+) \[",l)
    if m: sec=m.group(1); continue
    m=re.match(r"([0-9a-f]{5}) : ((?:[0-9a-f]{4} ?)+)",l)
    if m and sec=="PROM0":
        a=int(m.group(1),16)
        for i,x in enumerate(m.group(2).split()): w[a+i]=int(x,16)
open("PROM0.bin","wb").write(b"".join(struct.pack("<H",w[a]) for a in range(0x7000,0xe000)))
print("PROM0.bin: %d words (0x7000-0xdfff)" % len(range(0x7000,0xe000)))
for a in (0x7cbd,0x84a4,0xb21d,0xb2b1,0xb32d):
    print("  %04x: %04x %04x" % (a,w[a],w[a+1]))
E
run sha256sum PROM0.bin
say ""

# ---------------------------------------------------------- 2. disassembler
say "## 2. objdump tic54x (binutils 2.21.1)"
OBJDUMP=build-tic54x/binutils/objdump
if [ ! -x "$OBJDUMP" ]; then
    [ -f binutils-2.21.1.tar.bz2 ] || run curl -sO https://ftp.gnu.org/gnu/binutils/binutils-2.21.1.tar.bz2
    [ -d binutils-2.21.1 ] || tar xjf binutils-2.21.1.tar.bz2
    mkdir -p build-tic54x && ( cd build-tic54x && \
      ../binutils-2.21.1/configure --target=tic54x-coff --disable-werror --disable-nls --disable-gdb \
        --disable-gas --disable-ld --disable-gold --disable-gprof --disable-sim MAKEINFO=true \
        CFLAGS='-O1 -w -std=gnu89' >/dev/null && make -j8 all-binutils MAKEINFO=true >/dev/null ) \
      && say "built $OBJDUMP" || say "BUILD FAILED"
fi
run "$OBJDUMP" --version
"$OBJDUMP" -D -b binary -m tms320c54x --adjust-vma=0x7000 PROM0.bin > PROM0.dis 2>&1
say "full disassembly: PROM0.dis ($(wc -l < PROM0.dis) lines)"
say ""
say "### expected: 7cbd sub #11236,a | 84a4 stm #10791,ar3 | b21d stm #191,ar3 | b2b1 stm #151,ar3 | b32d stm
 ↪  #48,ar3"
grep -E '^ *(7cbd|84a4|b21d|b2b1|b32d):' PROM0.dis | tee -a "$LOG"
say ""
say "### SB TOA chain"
for a in 7c3d 81c8 7c49 7a1c 84a1 84ca 84dd 7c5f 7ca6 7cbc 7e8a b1e7; do
    grep -E "^ *$a:" PROM0.dis | tee -a "$LOG"
done
say ""
say "### FB TOA: 0x793e-0x795a"
awk '/^ *793e:/,/^ *795a:/' PROM0.dis | tee -a "$LOG"
say ""
say "### DMA programming routine: 0xa5cd-0xa5f6"
awk '/^ *a5cd:/,/^ *a5f6:/' PROM0.dis | tee -a "$LOG"
say ""

# ------------------------------------------------------------------ 3. git
say "## 3. layer1 git history"
[ -d osmocom-bb ] || run git clone -q https://gitea.osmocom.org/phone-side/osmocom-bb
run git -C osmocom-bb log --format='%h %ad %an %s' --date=short -S"toa -= 23" -- src/target/firmware/layer1
run git -C osmocom-bb log --format='%h %ad %an %s' --date=short -S"magic 23" -- src/target/firmware/layer1
say "### cb71b972: toa -= 23 / fbinfo2cellinfo lines (expect added fbinfo2cellinfo, no removed toa -= 23)"
git -C osmocom-bb show cb71b972 -- src/target/firmware/layer1/prim_fbsb.c | grep -nE '^[-+].*(toa -=
 ↪  23|fbinfo2cellinfo)|^@@' | tee -a "$LOG"
say ""
say "### current constants"
```

```
grep -nE 'L1_(SB|NB)_MARGIN_Q|L1_TAIL_DURATION_Q|L1_(SB|NB)_DURATION_Q'
 ↪  osmocom-bb/src/target/firmware/layer1/tpu_window.c | tee -a "$LOG"
grep -nE 'toa -= 23|magic 23' osmocom-bb/src/target/firmware/layer1/prim_fbsb.c | tee -a "$LOG"
grep -nE 'tpu_shift \+= 75|SB TOA of 4' osmocom-bb/src/target/firmware/layer1/sync.c | tee -a "$LOG"
say ""

# ------------------------------------------------------------- 4. logs
say "## 4. hardware logs: qbits check ((TOA-46) mod 1250)*4"
say "sources: http://lists.osmocom.org/pipermail/baseband-devel/2011-September/002526.html"
say "         https://www.mail-archive.com/baseband-devel@lists.osmocom.org/msg01046.html"
python3 - <<'E' 2>&1 | tee -a "$LOG"
fb1 = [(9651,3420),(8751,4820),(8755,4836),(10003,4828),(10007,4844),(8755,4836)]
sb  = [(29,4932),(29,4932),(27,4924),(27,4924),(24,4912),(26,4920)]
ok = 0
for toa,q in fb1+sb:
    p46 = ((toa-46)%1250)*4; p23 = ((toa-23)%1250)*4
    ok += (p46==q)
    print("  TOA=%5d printed=%4d  -46:%4d %s  -23:%4d" % (toa,q,p46,"OK " if p46==q else "NO ",p23))
print("  %d/%d lines match the -46 rule" % (ok,len(fb1)+len(sb)))
fb0=[7200,2544,8784,10032,48,1296,8784]; m1=[9651,8751,8755,10003,10007,12507,12507,1259,10007,8755]
print("  FB mode 0 multiples of 48: %d/%d" % (sum(t%48==0 for t in fb0),len(fb0)))
print("  FB mode 1 == 3 mod 4:      %d/%d" % (sum(t%4==3 for t in m1),len(m1)))
d=[t-27.25 for t,_ in sb]
print("  first SB after FB sync: %s  mean delta %.2f" % ([t for t,_ in sb], sum(d)/len(d)))
print("  148 + 2*23 - 3 = %d ; 148 + 2*3 - 3 = %d ; (191-148+3)/2 = %g" % (148+2*23-3,148+2*3-3,(191-148+3)/2))
E
say ""
say "# done — $(wc -l < "$LOG") lines in $LOG"
```

3.8 /opt/GSM/c54x_exe/run.sh

65933 octets, 931 lignes → 931 lignes

```
#!/usr/bin/env bash
# run.sh - le cote MOBILE du banc Calypso, processus par processus, sans le reseau.
#
# Deux montages (MODE) :
#   dsp    (defaut)  1. c54x_exe --arm   le DSP TMS320C54x, mask-ROM TI, hors QEMU
#                    2. qosmo            l'ARM + layer1 osmocom-bb (CALYPSO_DSP_EXTERN=1)
#                    3. osmocon          romload sur le pty serial0, relais L1CTL
#                    4. mobile           couche 2/3 osmocom-bb
#                    5. pont_dsp.py      (PONT=1) bursts du BTS -> DSP (--dsp-port 6702) ;
#                                        le pont DSP (pont/dsp/), bascule TCH suivie par le firmware
#   grgsm            2. qosmo            l'ARM + layer1 avec la couche 1 gr-gsm (shunt)
#                    3. osmocon  4. mobile  5. pont.py (PONT=1) bursts du BTS -> L1 gr-gsm
# Le pont a besoin du BTS (osmo-bts-trx, TRXD 5700) : sans reseau il demarre et attend.
#   OSMO_TRX=1       [2026-10-19] LE TELEPHONE EST LE TRX DE LA BTS#0 (start-native.sh --osmo-trx) :
#                    2. qosmo sur trx.highram.elf (osmocom-bb-transceiver, appli « trx » de Sylvain Munaut)
#                    4. transceiver (hote, osmocom-bb-transceiver, patches/osmocom-bb-transceiver-qemu-bench)
#                       A LA PLACE de mobile : L1CTL sur $L2_SOCK, TRXD 5700-5702 face a osmo-bts-trx,
#                       --no-sync (aucune cellule a ecouter), un seul timeslot RX (OSMO_TRX_TS_MASK=0x01) ;
#                       le masque TX reste 0xe3 (TS 5,6,7,0,1) : prim_bts.c ne sert le TS0 BCCH que si
#                       tx_start == 5 (fenetre TX decalee de 3 slots). [2026-10-20] -m 0x01 posait aussi
#                       TX=0x01 -> tx_start 0 -> dummy a chaque trame (DUMMY 18000, FB 0, SB 0).
#                    5. pas de pont : il tiendrait les memes ports 5700-5702 (PONT force a 0)
#                    OSMO_TRX_ARGS (defaut « --no-sync -e 3 », + « -r $OSMO_TRX_RT » SCHED_RR 20), TRANSCEIVER
 ↪  (chemin du binaire).
#   INSTANCE=2       [2026-10-19] UNE DEUXIEME CHAINE COMPLETE a cote de la premiere (banc-mobile pendant
#                    que banc-trx tourne) : run.sh se relance dans un espace de montage prive ou
#                    /dev/shm EST /dev/shm/calypso2 (vu de l'hote), et le vrai /dev/shm de l'hote est
#                    visible en /run/calypso-shm-hote ; aucun chemin /dev/shm/... du code ne change.
#                    RUNDIR /tmp/c54x-pont2, L1CTL /tmp/osmocom_l2_2, socket DSP /tmp/calypso_dsp2.sock,
#                    moniteur, gdb 1235/44445, clavier /tmp/calypso_keys2.sock, BSP udp 6712, VTY mobile
```

```
#                   +100, 4347 (copie de mobile_pont.cfg ; 4257 est a osmo-ggsn), PANNEAU_LOGS=none. AIR=1 (defaut
 ↪  en instance 2) :
#                    etape 5 = pont/air.py, l'air entre les deux Calypsos (bursts TX de l'un vers le BSP
#                    de l'autre) a la place du pont.
#   FIRMWARE_APP=rssi  [2026-10-19] une appli firmware SEULE sur l'ARM + le DSP (rssi, menu,
#                    hello_world, snake_game... : board/compal_e88/<app>.highram.elf) : etapes 1-3,
#                    ni mobile ni pont. L'ecran et le clavier du C123 sont emules (qosmo :
#                    calypso_st7558.c -> /dev/shm/calypso_lcd, calypso_armio.c <- /tmp/calypso_keys.sock) ;
#                    la fenetre : /usr/bin/python3 /opt/GSM/osmo-operator/tools/osmo-c123-gtk.py
#
#   ./run.sh                 tout lancer            ./run.sh --status    qui tourne
#   MODE=grgsm ./run.sh      montage gr-gsm         ./run.sh --logs      suivre les journaux
#   PONT=1 ./run.sh          avec le pont           ./run.sh --stop      tout arreter, nettoyer
#   ./run.sh --step N        une seule etape (les precedentes doivent tourner)
# Variables : MODE, PONT, LOCKSTEP (1 : QEMU attend le DSP a chaque trame), INSNS (80000),
#   VERB (-v), IQ (none|fcch|cell|...), AMP (30000),
#   QOSMO, FIRMWARE_ELF, FIRMWARE_BIN, OSMOCON, MOBILE, MOBILE_CFG, PONT_PY, RUNDIR, L2_SOCK.
#   UL_RAW (1 : le pont emet les bursts finaux de la ROM tels quels ; 0 : codage hote), MONTANT_EFFACE_TASK_RA.
#   Temps reel : RT_NICE (-10 par defaut, vide = sans), RT_FIFO (SCHED_FIFO, ex. 20, root), RT_CPUS (taskset,
#   ex. 1-3), RT_RENICE_AUTRES (ex. 15 : renice des outils du panneau) ; ./run.sh --charge  qui consomme quoi.
# Details, attendus et verifications : LAUNCH.md a cote. Bruit (BRUIT_MODE, inactif par defaut) : bloc BRUIT_ en
 ↪  fin.
set -uo pipefail
HERE="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
MODE="${MODE:-dsp}"
PONT="${PONT:-0}"
# [2026-10-19] INSTANCE=N (N != 1) : /dev/shm prive = /dev/shm/calypsoN de l'hote ; le vrai /dev/shm
# reste lisible en /run/calypso-shm-hote (l'air y lit l'anneau TX de l'instance 1). Les processus
# lances ici heritent de l'espace ; --stop/--status relances avec INSTANCE=N y entrent aussi.
INSTANCE="${INSTANCE:-1}"
# [2026-10-21] REGLAGES PAR INSTANCE : instances/instance-$INSTANCE.env (ecrit par la page web
 ↪  tools/osmo-reglages.py,
# patron instances/instance.env.example). Lignes `: "${VAR:=valeur}"` : l'environnement (start-native.sh, ligne de
# commande) garde la priorite, puis le fichier, puis les defauts ci-dessous. Exporte (set -a) pour
 ↪  c54x_exe/QEMU/pont.
INSTANCE_ENV="${INSTANCE_ENV:-/opt/GSM/osmo-operator/instances/instance-$INSTANCE.env}"
if [ -r "$INSTANCE_ENV" ]; then
    set -a; . "$INSTANCE_ENV"; set +a
    printf '\033[1m[run dsp]\033[0m instance %s : reglages de %s\n' "$INSTANCE" "$INSTANCE_ENV"
fi
HOTE_SHM=/run/calypso-shm-hote
AIR_DEMANDE="${AIR:-}"   # [2026-10-23] AIR pose par l appelant (sinon defaut par instance / appli, plus bas)
if [ "$INSTANCE" != 1 ] && [ -z "${CALYPSO_NS:-}" ]; then
    command -v unshare >/dev/null || { echo "[run] ECHEC : unshare absent (util-linux)" >&2; exit 1; }
    _d="/dev/shm/calypso$INSTANCE"; mkdir -p "$_d" "$HOTE_SHM" && chmod 1777 "$_d"
    exec unshare -m --propagation private bash -c '
        mount --bind /dev/shm "$4" && mount --bind "$1" /dev/shm && export CALYPSO_NS="$2" && exec bash "$3"
 ↪  "${@:5}"' \
        _ "$_d" "$INSTANCE" "$0" "$HOTE_SHM" "$@"
fi
if [ "$INSTANCE" != 1 ]; then
    RUNDIR="${RUNDIR:-/tmp/c54x-pont$INSTANCE}"
    L2_SOCK="${L2_SOCK:-/tmp/osmocom_l2_$INSTANCE}"
    MONITOR="${MONITOR:-/tmp/qemu-monitor-pont$INSTANCE.sock}"
    DSP_SOCK="${DSP_SOCK:-/tmp/calypso_dsp$INSTANCE.sock}"
    GDB_STUB="${GDB_STUB:-$((1233 + INSTANCE))}"
    GDB_TELNET="${GDB_TELNET:-$((44443 + INSTANCE))}"
    PANNEAU_LOGS="${PANNEAU_LOGS:-none}"
    # [2026-10-21] instance >= 3 : +30, car 6722 (= 6702 + 20) est le port TRXD « MS » de fake_trx.py (-p 6720,
     ↪  sidecar de
    # la BTS 1) : il se liait sur 127.0.0.1:6722 et volait les bursts montants du telephone 2 (instance 3 : 6752).
    export CALYPSO_BSP_PORT="${CALYPSO_BSP_PORT:-$((6702 + 10 * (INSTANCE - 1) + (INSTANCE >= 3 ? 30 : 0)))}"
    export CALYPSO_KEYS_SOCK="${CALYPSO_KEYS_SOCK:-/tmp/calypso_keys$INSTANCE.sock}"
    AIR="${AIR:-1}"
fi
AIR="${AIR:-0}"
```

```
AIR_PY="${AIR_PY:-/opt/GSM/osmo-operator/pont/air.py}"
QOSMO="${QOSMO:-/opt/GSM/qosmo}"
QEMU="${QEMU:-$QOSMO/build/qemu-system-arm}"
# Firmware : layer1.highram d'origine, non modifie. [2026-10-04] La variante layer1.highram_patched
# (trames B_BFI du TCH descendant passees a charge nulle, masquees par l'ECU de GAPK ; be10de1) est
# retiree : le B_BFI de la ROM est juste depuis qosmo a19dde59, et le but est le firmware tel quel.
# FIRMWARE_ELF=... pour un autre binaire.
_FW=/opt/GSM/firmware/board/compal_e88
# [2026-10-19] OSMO_TRX=1 : firmware trx.highram (le Calypso emet la BTS), appli hote transceiver.
OSMO_TRX="${OSMO_TRX:-0}"
FIRMWARE_APP="${FIRMWARE_APP:-}"
if [ -n "$FIRMWARE_APP" ]; then
    FIRMWARE_ELF="$_FW/$FIRMWARE_APP.highram.elf"   # s'impose aussi face a /etc/default/osmo-banc
    FIRMWARE_BIN="${FIRMWARE_ELF%.elf}.bin"
    PONT=0
fi
# [2026-10-23] RSSI : l'air realiste par defaut (calypso_bsp.c CALYPSO_AIR_NIVEAUX : la cellule seulement sur son
# ARFCN, niveaux DL/UL, bruit ailleurs) et, en BTS classique (pont_dsp.py vivant), pas d'air.py : le descendant
 ↪  arrive
# par la copie du pont (PONT_DSP_COPIES, udp 6762). Ici et non dans banc-rssi : un terminal ouvert avant la mise a
# jour des alias, ou un autre lanceur, relancait rssi avec l'ancien comportement.
if [ "$FIRMWARE_APP" = rssi ]; then
    export CALYPSO_AIR_NIVEAUX="${CALYPSO_AIR_NIVEAUX:-1}"
    export CALYPSO_FAUX_AIR="${CALYPSO_FAUX_AIR:-$( . /etc/osmocom/radio-plan.env 2>/dev/null; echo
     ↪  "${PLAN_ARFCN:-}")}"
    [ -z "$AIR_DEMANDE" ] && pgrep -f 'pont/pont_dsp\.py' >/dev/null && AIR=0
fi
# [2026-10-20] OSMO_TRX_TS1=1 (instance 1, avec OSMO_TRX=1) : un DEUXIEME Calypso TRX pour le TS1 (TCH/F, les
 ↪  appels) :
# l'instance OSMO_TRX_TS1_INSTANCE (3) est lancee avant le transceiver (etapes 1-3 seules, OSMO_TRX_PHONE=2 :
 ↪  firmware trx,
# L1CTL /tmp/osmocom_l2_3, pas de transceiver, pas d'air) et le transceiver prend « -2 -L /tmp/osmocom_l2_3 »
 ↪  (telephone 2 =
# TS1 TX+RX, prim_bts.c SLOT 0 / tn 1). Son BSP (udp 6752 depuis le 21/10, 6722 avant) n'a pas de SB pour se caler
 ↪  : CALYPSO_BSP_TRX_OFFSET=-1 (fn du
# firmware = tick - 1, mesure sur le telephone 1). L'air (instance 2) relie aussi cet anneau (--c-ring/--c-port).
# [2026-10-20 23:10] -2 par defaut : OSMO_TRX_TS1=1 (start-native.sh --ts 1 ou OSMO_TRX_TS1=0 pour un seul
 ↪  telephone)
# [2026-10-21] retour a OSMO_TRX_TS1=0 par defaut : le 2e telephone a un fn independant (TCH descendant a ~80 bits
# faux pour le mobile, ASSIGNMENT FAILURE), seul le montage UN Calypso TS0+TS1 (OSMO_TRX_TCH=1) passe un appel.
# start-native.sh --ts 2 (ou OSMO_TRX_TS1=1) pour le montage a deux telephones.
OSMO_TRX_TS1="${OSMO_TRX_TS1:-0}"
OSMO_TRX_TS1_INSTANCE="${OSMO_TRX_TS1_INSTANCE:-3}"
OSMO_TRX_PHONE="${OSMO_TRX_PHONE:-1}"
# [2026-10-20] Le mobile par l'air (INSTANCE=N, AIR=1) n'est jamais le TRX : start-native.sh --bts exporte
 ↪  OSMO_TRX=1
# et l'instance 2 en heritait (firmware trx + second transceiver -> « port UDP 5700 deja pris »).
if [ "$OSMO_TRX" = 1 ] && [ "$AIR" = 1 ] && [ "$INSTANCE" != 1 ] && [ "$OSMO_TRX_PHONE" != 2 ]; then
    printf '\033[1m[run dsp]\033[0m INSTANCE=%s AIR=1 : OSMO_TRX=1 herite ignore, ce Calypso est le mobile (le TRX
     ↪  est l instance 1)\n' "$INSTANCE"
    OSMO_TRX=0
fi
if [ "$OSMO_TRX" = 1 ]; then
    # Le firmware trx S'IMPOSE : /etc/default/osmo-banc (profil GPRS) pose FIRMWARE_ELF=layer1.highram.gprs
    # dans l'environnement, et un « ${FIRMWARE_ELF:-trx} » le laissait passer : le transceiver parlait
    # BTS_MODE (0x1f) a un layer1 qui ne le connait pas. OSMO_TRX_FIRMWARE=<elf> pour un autre trx.
    [ -n "${FIRMWARE_ELF:-}" ] && [ "${FIRMWARE_ELF##*/}" != "${OSMO_TRX_FIRMWARE:-trx.highram.elf}" ] \
        && printf '\033[1m[run dsp]\033[0m OSMO_TRX=1 : firmware %s ignore, trx.highram.elf a la place\n'
         ↪  "$FIRMWARE_ELF"
    FIRMWARE_ELF="${OSMO_TRX_FIRMWARE:-$_FW/trx.highram.elf}"
    FIRMWARE_BIN="${FIRMWARE_ELF%.elf}.bin"
    # BSIC de la cellule : osmo-bts-trx n'envoie que SETTSC (BCC), le NCC vient du bts 0 du BSC
    OSMO_TRX_BSIC="${OSMO_TRX_BSIC:-$(awk '/^ *bts 0$/{f=1} f&&/base_station_id_code/{print $2; exit}'
     ↪  /etc/osmocom/osmo-bsc.cfg 2>/dev/null)}"
    TRANSCEIVER="${TRANSCEIVER:-/opt/GSM/osmocom-bb-transceiver/src/host/layer23/src/transceiver/transceiver}"
    [ -x "$TRANSCEIVER" ] || TRANSCEIVER="$(command -v transceiver || echo "$TRANSCEIVER")"
```

```
    # [2026-10-20] OSMO_TRX_TCH=1 (defaut) : le telephone sert TS0 ET TS1 (TCH/F, les appels) : masques 0x03,
    # le firmware patche emet les deux slots de sa fenetre TX (prim_bts.c) et remonte le TS1 depuis rx[2] ;
    # c54x_exe publie les deux bursts depuis la page (CALYPSO_TRX_TX_SLOTS=2) et ecrit le montant TS1 dans
    # la page. OSMO_TRX_TCH=0 : TS0 seul (0x01).
    # [2026-10-20 23:10] avec le telephone 2 (OSMO_TRX_TS1=1, defaut) le telephone 1 garde le TS0 seul :
     ↪  OSMO_TRX_TCH=0
    OSMO_TRX_TCH="${OSMO_TRX_TCH:-$([ "$OSMO_TRX_TS1" = 1 ] && echo 0 || echo 1)}"
    if [ "$OSMO_TRX_TCH" = 1 ] && [ "$OSMO_TRX_PHONE" != 2 ]; then
        OSMO_TRX_TS_MASK="${OSMO_TRX_TS_MASK:-0x03}"
        OSMO_TRX_TX_MASK="${OSMO_TRX_TX_MASK:-0x03}"
        export CALYPSO_TRX_TX_SLOTS="${CALYPSO_TRX_TX_SLOTS:-2}"
    fi
    OSMO_TRX_TS_MASK="${OSMO_TRX_TS_MASK:-0x01}"
    # [2026-10-20] masque TX 0x01 : UNE fenetre TX (tx_start 0, slot TPU 5 = TS0 descendant), celle que la
    # ROM joue comme un RACH et que c54x_exe capture (script TSP 0x3cbb). Le 0xe3 d'origine (fenetre
    # multiple 5,6,7,0,1) n'est pas servi par le banc : seul le premier burst (dummy) sortait. Le firmware
    # trx corrige (prim_bts.c, SLOT/tn) vise le TS0 quel que soit tx_start.
    OSMO_TRX_TX_MASK="${OSMO_TRX_TX_MASK:-0x01}"
    # [2026-10-20 22:25] -r PRIO (SCHED_RR, main.c rt_prio ; OSMO_TRX_RT= vide pour s'en passer) ; -e 1 pour le
     ↪  debug (23:10 : -e 3 par defaut)
    OSMO_TRX_RT="${OSMO_TRX_RT-20}"
    OSMO_TRX_ARGS="${OSMO_TRX_ARGS:---no-sync -e 3}${OSMO_TRX_RT:+ -r $OSMO_TRX_RT}"
    PONT=0
    # la BTS repete ses bursts (dummy, FB) : c54x_exe publie chaque trame avec tache TX, pas que les nouveaux
    export MONTANT_ROM_TX_CHAQUE="${MONTANT_ROM_TX_CHAQUE:-1}"
    # [2026-10-20] le BSP joue les bursts montants (air.py, fn GSM du mobile) a la trame du firmware trx,
    # calee sur le fn du SB emis (pont.c -> calypso_bsp_trx_calage) ; CALYPSO_BSP_TRX_DELTA ajuste (defaut -2 avec
     ↪  le
    # firmware patche BTS_RX_LAG 3 : le burst montant fn X est joue une trame apres la fenetre native, marge pour
     ↪  l'air)
    export CALYPSO_BSP_TRX="${CALYPSO_BSP_TRX:-1}"
    # [2026-10-21] firmware trx BTS_RX_LAG 12 (prim_bts.c) : le burst montant fn X est joue 11 trames apres sa
     ↪  fenetre
    # native ; marge pour la latence aller-retour de l'air + le retard du mobile emule (RACH du Magnetite arrives
    # 27 trames trop tard avec LAG 4 / delta -3)
    export CALYPSO_BSP_TRX_DELTA="${CALYPSO_BSP_TRX_DELTA:--11}"
    # telephone 2 (TS1) : pas de SB emis, offset fn/tick fixe (voir OSMO_TRX_TS1)
    [ "$OSMO_TRX_PHONE" = 2 ] && export CALYPSO_BSP_TRX_OFFSET="${CALYPSO_BSP_TRX_OFFSET:--1}"
fi
FIRMWARE_ELF="${FIRMWARE_ELF:-$_FW/layer1.highram.elf}"
FIRMWARE_BIN="${FIRMWARE_BIN:-${FIRMWARE_ELF%.elf}.bin}"
# [2026-10-21] FIRMWARE_FLASH=<image 4 MiB> : un VRAI firmware en flash (FreeCalypso Magnetite,
# cible c139 : bootloader Compal en secteur 0 + fwimage.bin a 0x10000, voir
# osmo-operator/scripts/magnetite-c139.sh et doc/magnetite.md) a la place du layer1 osmocom-bb.
# QEMU demarre a 0 dans la flash (pas de -kernel, CPU ti925t = ARMv4T comme l'ARM7TDMI),
# l'ecran est celui du C139 (CALYPSO_LCD=c139, uWire), la pile L23 et l'UI sont DANS le
# firmware : pas d'osmocon (rvinterf decode les traces RVTMUX sur le pty modem, socket
# $RUNDIR/rvinterf.sock pour fc-shell / fc-fsio), pas de mobile. Le DSP (etape 1) et l'air
# ou le pont (etape 5) sont inchanges. start-native.sh --ms-magnetite le pose.
FIRMWARE_FLASH="${FIRMWARE_FLASH:-}"
# [2026-10-21 11:10] le TRX de la BTS (OSMO_TRX=1) ou le telephone 2 du TRX ne sont JAMAIS Magnetite
if [ -n "$FIRMWARE_FLASH" ] && { [ "$OSMO_TRX" = 1 ] || [ "$OSMO_TRX_PHONE" = 2 ]; }; then
    printf '\033[1m[run dsp]\033[0m OSMO_TRX=1 : FIRMWARE_FLASH=%s ignore, cette instance est le TRX de la BTS
     ↪  (firmware trx)\n' "$FIRMWARE_FLASH"
    FIRMWARE_FLASH=""
fi
# [2026-10-21] Magnetite (vrai firmware TI en flash) : pas de rattrapage du pacer. Avec le rattrapage (20),
# l ARM enchaine les trames en retard, la L1 sature et le chien de garde reset le firmware (gel).
if [ -n "$FIRMWARE_FLASH" ]; then
    export CALYPSO_PACER_RATTRAPAGE="${CALYPSO_PACER_RATTRAPAGE:-0}"
    export CALYPSO_PONT_ARM_LOCK="${CALYPSO_PONT_ARM_LOCK:-1}"
    # [2026-10-21] L1 TI : fenetre SB ouverte en fin de trame precedente (sinon BCCH decale d'une trame,
    # CRC KO sur chaque bloc) et TOA NB vise a 3 (marge 1) ; voir calypso_bsp.c sb_fin_trame()
    export CALYPSO_BSP_SB_FIN_TRAME="${CALYPSO_BSP_SB_FIN_TRAME:-1}"
    export CALYPSO_BSP_NB_MARGE="${CALYPSO_BSP_NB_MARGE:-1}"
    # TOA NB canne a la cible TI (pont.c). [21/10 17:05] coupe par defaut tant que c54x_exe n'est pas
```

```
    # recompile avec la garde init_done : sans elle le cannage ecrase le code DSP pendant le boot.
    export PONT_CAN_NB_TOA="${PONT_CAN_NB_TOA:-}"
    # [21/10] faux air : bruit hors de l'ARFCN de la cellule (calypso_bsp.c faux_air_*), ARFCN du plan radio
    # [21/10 19:50] SB module comme FB/NB : les reglages SB du 30/09 (instant 0.35, elargissement 1.0, bruit 3000,
    # faits pour la layer1 osmocom qui ignore l'angle du SB) biaisaient l'angle mesure (+166 a 0 Hz) ; la L1 TI
    # s'en sert pour l'AFC (-700 -> -673, ~224 Hz de vrai decalage) et perdait SB puis BCCH
    export CALYPSO_BSP_SB_DEC="${CALYPSO_BSP_SB_DEC:-0.5}" CALYPSO_BSP_SB_SYM="${CALYPSO_BSP_SB_SYM:-0.3}"
     ↪  CALYPSO_BSP_SB_NOISE="${CALYPSO_BSP_SB_NOISE:-0}"
    export CALYPSO_BSP_BRUIT_FOND="${CALYPSO_BSP_BRUIT_FOND:-1}"   # bruit au lieu de zeros (PM = 0 -> recovery L1)
    # rattraper le direct du BTS (calypso_bsp.c) : PAR L'AIR SEULEMENT (AIR=1, la BTS a sa propre horloge). [21/10
     ↪  21:37]
    # en BTS classique pont_dsp.py cadence osmo-bts sur les trames livrees par le DSP : chaque saut de +33
     ↪  accelerait la
    # BTS (pont a ~350 trames/s), d'ou plus de retard et plus de sauts ; RA_C 1708/1933/2165 contre RACH du pont a
    # 1737/2059/2457, IMM ASS jamais reconnues.
    export CALYPSO_BSP_RATTRAPE="${CALYPSO_BSP_RATTRAPE:-$AIR}"
    export CALYPSO_RACH_VERS_AIR="${CALYPSO_RACH_VERS_AIR:-1}"   # RACH du firmware TI publie vers l'air (pont.c)
    # [21/10 21:55] BTS classique : les bursts SDCCH/SACCH du firmware TI sont bien dans calypso_tx_rom (TSC 7, fn
     ↪  de
    # la ROM), mais pont_dsp les jetait faute de canal dedie annonce (calypso_dcch_cfg vient du tap L1CTL de QEMU,
    # absent sans layer1 osmocom) : « aucun canal dedie connu apres 300 ms, perdu », « DL: >N200 of SABM ».
    # PONT_UL_TN_DEFAUT=0 : TS0 (SDCCH/4 du banc) faute d'annonce. PONT_UL_RAW_DCCH_HOTE=1 (codage hote) reste
     ↪  dispo.
    export PONT_UL_TN_DEFAUT="${PONT_UL_TN_DEFAUT:-0}"
    # [21/10 23:05] TCH montant par l'HOTE (FACCH, SACCH/T, parole codees par le pont depuis a_fu/a_du) : le
     ↪  script TSP
    # de la ROM ne sort que 2 bursts « inconnus » sur le TCH du firmware TI, le SABM FACCH n'atteignait jamais la
     ↪  BTS
    # (BSC : rll_ready=no, ASSIGNMENT FAILURE, « >N200 of SABM »). 14-demarrage.sh exporte PONT_UL_RAW_TCH=1 pour
     ↪  le
    # vocodeur ROM du mobile osmocom : on force 0 ici (PONT_UL_RAW_TCH_MAGNETITE pour revenir dessus).
    export PONT_UL_RAW_TCH="${PONT_UL_RAW_TCH_MAGNETITE:-0}"
    # [21/10 21:35] PAS de correction de fn ni de RA (AIR_B_FN_DELTA, MONTANT_RACH_DIFFERE : 0). Les ecarts vus a
     ↪  21:05
    # et 21:25 comparaient au RA_R de la L1 TI, qui trace la DEMANDE (MPHC_RA_REQ) ; le RACH part apres le tirage
    # aleatoire, juste avant RA_C (RA_R 9394 ra 12 -> RA_C 9403 ; c54x_exe publie ra 12 a 9403) : fn et RA etaient
     ↪  justes.
    export AIR_B_FN_DELTA="${AIR_B_FN_DELTA:-0}"
    export MONTANT_RACH_DIFFERE="${MONTANT_RACH_DIFFERE:-0}"
    export CALYPSO_FAUX_AIR="${CALYPSO_FAUX_AIR:-$( . /etc/osmocom/radio-plan.env 2>/dev/null; echo
     ↪  "${PLAN_ARFCN:-}")}"
    [ -r "$FIRMWARE_FLASH" ] || { printf '\033[1;31m[run] ECHEC :\033[0m image flash absente : %s\n'
     ↪  "$FIRMWARE_FLASH" >&2; exit 1; }
    QEMU_CPU="${QEMU_CPU:-ti925t}"
    export CALYPSO_LCD="${CALYPSO_LCD:-c139}"
    RVINTERF_BIN="${RVINTERF_BIN:-/opt/GSM/fc-host-tools/rvinterf/lowlevel/rvinterf}"
fi
OSMOCON="${OSMOCON:-/opt/GSM/osmocom-bb/src/host/osmocon/osmocon}"
MOBILE="${MOBILE:-$(command -v mobile || echo /usr/local/bin/mobile)}"
MOBILE_CFG="${MOBILE_CFG:-$HERE/mobile_pont.cfg}"
# [2026-09-23] Deux points d'entree pour le meme paquet pont/ : pont_dsp.py pour le
# montage dsp (--dsp-port 6702), pont.py pour le montage grgsm, qui
# garde ses defauts d'avant le decoupage. PONT_PY force l'un ou l'autre.
if [ "$MODE" = grgsm ]; then
    PONT_PY="${PONT_PY:-/opt/GSM/osmo-operator/pont/pont.py}"
else
    PONT_PY="${PONT_PY:-/opt/GSM/osmo-operator/pont/pont_dsp.py}"
fi
RUNDIR="${RUNDIR:-/tmp/c54x-pont}"
# [2026-10-04] TEMPS REEL. Sous charge (campagne de tests, panneau, ffmpeg) la chaine c54x_exe <-> QEMU
# <-> osmocon/mobile <-> pont rate des ticks et le son se coupe : la trame fait 4,615 ms, le travail
# mesure ~1,2 ms ([chrono] 2026-09-23 : A 0.33 + go 0.40 + B 0.16 + qemu <0.32), et l'ordonnanceur CFS
# met volontiers un processus de cote 5 a 10 ms quand les coeurs sont pris. Les cinq processus de la
# chaine sont donc lances avec une priorite : RT_NICE=-10 par defaut (RT_NICE= vide pour ne rien faire),
# RT_FIFO=<prio> pour SCHED_FIFO (chrt -f, root ; rester sous 50, les threads noyau sont au-dessus),
# RT_CPUS=<liste> pour les epingler (taskset -c, ex. 1-3 : le coeur 0 reste au reste du systeme).
```

```
# RT_RENICE_AUTRES=<n> abaisse en plus les outils du panneau (osmo-fft-snap, osmo-panel, osmo-topzone,
# osmo-ts-probe, conky) a nice n. ./run.sh --charge montre qui consomme quoi.
RT_NICE="${RT_NICE--10}"
RT_FIFO="${RT_FIFO:-}"
RT_CPUS="${RT_CPUS:-}"
RT_RENICE_AUTRES="${RT_RENICE_AUTRES:-}"
PRIO=()
[ -n "$RT_CPUS" ] && PRIO+=(taskset -c "$RT_CPUS")
if [ -n "$RT_FIFO" ]; then PRIO+=(chrt -f "$RT_FIFO"); elif [ -n "$RT_NICE" ]; then PRIO+=(nice -n "$RT_NICE"); fi
renice_autres() {
    [ -n "$RT_RENICE_AUTRES" ] || return 0
    local pids; pids="$(pgrep -f 'osmo-fft-snap|osmo-panel|osmo-topzone|osmo-ts-probe|conky' | tr '\n' ' ')"
    [ -n "$pids" ] && renice -n "$RT_RENICE_AUTRES" -p $pids >/dev/null 2>&1 && dire "   outils du panneau a nice
     ↪  $RT_RENICE_AUTRES : $pids"
    return 0
}
charge() {
    echo "chaine (pid %cpu nice politique) :"
    local n
    for n in dsp qemu osmocon rvinterf mobile transceiver pont air; do
        if vivant "$n"; then printf '  %-8s %s\n' "$n" "$(ps -o pid=,pcpu=,ni=,cls= -p "$(pid_de "$n")" | tr -s '
         ↪  ')"; else printf '  %-8s arrete\n' "$n"; fi
    done
    echo "processus les plus gourmands :"
    ps -eo pid,pcpu,ni,cls,comm --sort=-pcpu | head -12 | sed 's/^/  /'
    echo "charge moyenne $(cut -d' ' -f1-3 /proc/loadavg) sur $(nproc) coeurs ; priorite de la chaine :
     ↪  ${PRIO[*]:-aucune}"
}
L2_SOCK="${L2_SOCK:-/tmp/osmocom_l2}"
MONITOR="${MONITOR:-/tmp/qemu-monitor-pont.sock}"
GDB="${GDB:-1}"                              # gdbstub QEMU + console telnet (etape 2)
GDB_STUB="${GDB_STUB:-1234}"
GDB_TELNET="${GDB_TELNET:-44444}"
GDB_TELNET_PY="${GDB_TELNET_PY:-$QOSMO/tools/gdb-telnet.py}"   # [2026-10-03] etait /opt/GSM/qosmo-dsp/... (depot
 ↪  disparu)
# [2026-10-21] 120000 par defaut : c est le budget du DSP du TRX (instance 1, boote au premier IDLE) ;
# 16000 laissait le DSP de l instance Magnetite en « occupe boot » pour toujours.
INSNS="${INSNS:-120000}"   # [2026-09-23] 60000 debordait en TCH (jusqu a 87000 insn/trame), voir start-native.sh
# [2026-09-20] Pas-a-pas DSP/QEMU par defaut (LOCKSTEP=0 pour le mode horloge murale) :
# le C54x emule coute ~6,7 ms par trame contre 4,615 ms de temps reel, QEMU sautait
# donc 3 trames sur 4 (« DSP en retard, tick saute »), la ROM ne voyait qu'une trame
# sur 4 a 6, son compteur de blocs FB n'avancait pas et le TOA valait 1251 quelle que
# soit la distance de la FCCH. En pas-a-pas QEMU n'avance la trame que quand le DSP
# a fini la precedente : TOA = 23 + n x 1250, delay=10, SB decodable.
LOCKSTEP="${LOCKSTEP:-1}"
[ "$MODE" = dsp ] && [ "$LOCKSTEP" = 1 ] && export CALYPSO_PONT_LOCKSTEP=1
VERB="${VERB:--v}"
IQ="${IQ:-none}"
AMP="${AMP:-30000}"
DSP_SOCK="${DSP_SOCK:-/tmp/calypso_dsp.sock}"
DSP_SHM=/dev/shm/calypso_api_ram

# [2026-09-23] LA VITRINE : LES MEMES JOURNAUX QU'EN MODE SHUNT. Le panneau
# (tmux calypso, osmo-fft-snap) suit /run/user/0/osmo-nitb/logs/{qemu,osmocon,
# mobile}.log et /dev/shm/pont.log, ceux qu'ecrit le montage grgsm. Ce banc-ci
# n'ecrivait que dans $RUNDIR : les volets restaient a 0 octet. Le VRAI fichier
# est donc pose au chemin du panneau et $RUNDIR/<nom>.log devient un lien vers
# lui -- pas l'inverse : un `tail -F` deja ouvert refuse un fichier remplace par
# un lien (« untailable symbolic link »). PANNEAU_LOGS=none pour ne rien toucher.
PANNEAU_LOGS="${PANNEAU_LOGS:-/run/user/0/osmo-nitb/logs}"
# La FFT du panneau lit /tmp/iq_fft_ms.fifo, que pont.py ne remplit que si
# l'enregistrement est actif : --no-record la rendait muette. PONT_AIRREC=0 le coupe.
PONT_AIRREC="${PONT_AIRREC:-1}"
vitrine() {   # vitrine <nom> : a appeler AVANT le lancement qui ecrit $RUNDIR/<nom>.log
    rm -f "$RUNDIR/$1.log"
    [ "$PANNEAU_LOGS" = none ] && return 0
```

```
    local cible="$PANNEAU_LOGS/$1.log"
    [ "$1" = pont ] && cible=/dev/shm/pont.log
    [ -d "$(dirname "$cible")" ] || return 0
    rm -f "$cible" && : > "$cible" && ln -s "$cible" "$RUNDIR/$1.log"
    return 0
}

dire()  { printf '\033[1m[run %s]\033[0m %s\n' "$MODE" "$*"; }
rater() { printf '\033[1;31m[run] ECHEC :\033[0m %s\n' "$*" >&2; exit 1; }
pid_de() { [ -f "$RUNDIR/$1.pid" ] && cat "$RUNDIR/$1.pid"; }
vivant() { local p; p="$(pid_de "$1")"; [ -n "$p" ] && kill -0 "$p" 2>/dev/null; }
attendre() { local n=$(( $1 * 10 )); shift; while [ "$n" -gt 0 ]; do "$@" && return 0; sleep 0.1; n=$((n - 1));
 ↪  done; return 1; }
[ "$MODE" = dsp ] || [ "$MODE" = grgsm ] || rater "MODE=$MODE inconnu (dsp|grgsm)"

etape1() {   # le DSP (montage dsp seulement)
    renice_autres
    [ "$MODE" = dsp ] || { dire "1. (montage grgsm : pas de c54x_exe, la couche 1 est dans QEMU)"; return; }
    vivant dsp && { dire "1. c54x_exe deja lance (pid $(pid_de dsp))"; return; }
    [ -x "$HERE/c54x_exe" ] || make -C "$HERE" >/dev/null || rater "make c54x_exe"
    rm -f "$DSP_SHM" "$DSP_SOCK"
    # [2026-09-23] L'annonce TCH d'une session morte (pont/dsp/tch.py) ne doit
    # pas etre relue par montant.c au demarrage.
    rm -f /dev/shm/calypso_tch_cfg
    # [2026-09-23] Attendre (au plus 40 ms) une trame que la BTS livre en
    # retard plutot que la jouer en effacement (calypso_bsp.c, bsp_attendre_trame).
    ( cd "$HERE" && bruit_env_dsp && CALYPSO_IQDUMP_FCCH=1 CALYPSO_BSP_ATTENTE_MS="${CALYPSO_BSP_ATTENTE_MS:-40}"
     ↪  exec ${PRIO[@]+"${PRIO[@]}"} "${C54X_BIN:-./c54x_exe}" --arm "$DSP_SOCK" --insns "$INSNS" --iq "$IQ"
     ↪  --amp "$AMP" $VERB ) > "$RUNDIR/dsp.log" 2>&1 &
    echo $! > "$RUNDIR/dsp.pid"
    attendre 5 test -S "$DSP_SOCK" || rater "c54x_exe n'a pas ouvert $DSP_SOCK (voir $RUNDIR/dsp.log)"
    # [2026-10-06] shm en 0666 : le umask rabaisse la demande 0666 de c54x_exe a 0644, ce qui empeche
    # un harvard.sh lance sous le compte utilisateur d'ecrire l'API RAM (d@!, demos sb/si3). On le force.
    attendre 3 test -e "$DSP_SHM" && chmod 0666 "$DSP_SHM" 2>/dev/null || true
    dire "1. c54x_exe --arm  pid $(pid_de dsp)  ($INSNS insn/trame, iq=$IQ, $DSP_SHM, $DSP_SOCK)"; bruit_dire_dsp
}

etape2() {   # l'ARM
    vivant qemu && { dire "2. QEMU deja lance (pid $(pid_de qemu))"; return; }
    [ -x "$QEMU" ] || rater "QEMU absent : $QEMU (ninja -C $QOSMO/build)"
    [ -r "$FIRMWARE_ELF" ] || rater "firmware absent : $FIRMWARE_ELF"
    local extern=""
    if [ "$MODE" = dsp ]; then [ -S "$DSP_SOCK" ] || rater "le DSP ne tourne pas (etape 1 d'abord)";
     ↪  extern="$DSP_SOCK"; fi
    rm -f "$MONITOR"
    vitrine qemu
    # [2026-09-23] gdb sur l'ARM : gdbstub QEMU en tcp::$GDB_STUB, et la console
    # telnet de qosmo-dsp (tools/gdb-telnet.py) sur le port $GDB_TELNET :
    #     telnet 0 44444      (Ctrl-C arrete l'ARM, « continue & » le relance)
    # Tant que personne n'est connecte, gdb ne tourne pas et l'ARM n'est pas
    # touche. GDB=0 : ni stub ni console.
    local gdb_opt=()
    [ "$GDB" = 1 ] && gdb_opt=(-gdb "tcp:127.0.0.1:$GDB_STUB")
    # [2026-09-23] ASSEMBLY_LOGS=1 (start-native.sh --assembly-logs) : trace asm
    # de l'ARM dans qemu-asm.log, chaque bloc une fois, a sa traduction.
    # [2026-10-19] in_asm seul par defaut. ASSEMBLY_LOGS_FLAGS=in_asm,exec,nochain
    # journalise aussi chaque bloc execute (nochain sinon les blocs chaines ne
    # sont vus qu'une fois) : l'ARM passait de 0,4 a 62 ms par trame ([chrono]
    # go), le DSP en pas-a-pas suivait, mais l'horloge asservie du pont
    # (osmo-operator/pont/trx.py, plafond 4 x FRAME_DUR) prenait +1159 a +3897
    # trames d'avance, le BSP ne trouvait plus les bursts (« le burst a existe
    # puis a disparu ») et mobile ne voyait aucune cellule (FBSB result=255).
    # [2026-10-19] LA TRACE VA SUR LE DISQUE, PAS SUR LE TMPFS. $RUNDIR est dans
    # /tmp (tmpfs de 4,7 Go) : avec exec,nochain la trace a rempli /tmp en cinq
    # minutes (3,8 Go), et un rm ne libere rien tant que QEMU garde le fichier
    # ouvert (il faut tronquer /proc/<pid>/fd/N). On ecrit dans
```

```
# ASSEMBLY_LOGS_DIR (/var/tmp/c54x-pont, sur le disque) et $RUNDIR/qemu-asm.log
# devient un lien ; les archives n'y touchent pas (garder_journaux).
if [ "${ASSEMBLY_LOGS:-0}" = 1 ]; then
    local asm_dir="${ASSEMBLY_LOGS_DIR:-/var/tmp/c54x-pont}"
    mkdir -p "$asm_dir" 2>/dev/null || asm_dir="$RUNDIR"
    rm -f "$RUNDIR/qemu-asm.log" "$asm_dir/qemu-asm.log"
    [ "$asm_dir" != "$RUNDIR" ] && ln -s "$asm_dir/qemu-asm.log" "$RUNDIR/qemu-asm.log"
    local asm_cible="$asm_dir/qemu-asm.log"
    # [2026-10-19] ASSEMBLY_LOGS_FIFO=1 : QEMU ecrit dans un FIFO sur /dev/shm et un
    # cat draine vers le fichier sur le disque. qemu_log_unlock() fait un fflush() a
    # CHAQUE ligne (util/log.c) : avec exec,nochain c'est un write(2) par bloc
    # execute (~830 par trame), une dizaine de us sur le disque contre 1 a 2 us
    # sur un pipe. Le lecteur est lance AVANT QEMU : son fopen(fifo, "w") bloque
    # tant que personne ne lit.
    if [ "${ASSEMBLY_LOGS_FIFO:-0}" = 1 ]; then
        local asm_fifo="/dev/shm/qemu-asm.fifo"
        rm -f "$asm_fifo"; mkfifo "$asm_fifo"
        ${PRIO[@]+"${PRIO[@]}"} cat "$asm_fifo" > "$asm_cible" 2>/dev/null &
        echo $! > "$RUNDIR/asmcat.pid"
        asm_cible="$asm_fifo"
    fi
    gdb_opt+=(-d "${ASSEMBLY_LOGS_FLAGS:-in_asm}" -D "$asm_cible")
    [ -n "${ASSEMBLY_LOGS_FILTRE:-}" ] && gdb_opt+=(-dfilter "$ASSEMBLY_LOGS_FILTRE")
    # [2026-10-19] QEMU_LOG_BUFFERED (util/log.c de qosmo) : fichier -D pleinement
    # bufferise, plus de fflush() par ligne. 4 Mio par defaut quand on trace.
    export QEMU_LOG_BUFFERED="${QEMU_LOG_BUFFERED:-4194304}"
    # [2026-10-19] ANNEAU DES BLOCS EXECUTES, PAR DEFAUT AVEC ASSEMBLY_LOGS=1. Le plugin
    # TCG tbring (qosmo/tools/qosmo-tbring) pousse le PC de chaque bloc EXECUTE, blocs
    # chaines compris, dans un anneau mmap : 8 octets par bloc, pas de texte dans la
    # boucle, pas de nochain, l'ARM tourne comme en production (-d exec,nochain : 150 Mo/s
    # de texte, 20 Go en 130 s). Anneau de 64 Gio = 2^33 blocs, soit ~12 h d'ARM a
    # 180 000 blocs/s ; fichier CREUX sur le disque ($asm_dir, pas le tmpfs), il ne pese
    # que ce qui a ete ecrit. Relecture a chaud, a tout moment, sans arreter QEMU :
    #     tbring-dump -n 500 -f $asm_dir/qemu-tbring -e $FIRMWARE_ELF -a $asm_dir/qemu-asm.log
    # (la trace in_asm, un bloc par traduction, sert de table de desassemblage).
    # ASSEMBLY_LOGS_RING=0 coupe ; ASSEMBLY_LOGS_RING_ENTRIES (puissance de 2) et
    # ASSEMBLY_LOGS_RING_FILE changent la taille et l'emplacement.
    if [ "${ASSEMBLY_LOGS_RING:-1}" = 1 ]; then
        local tbring="${TBRING_SO:-$QOSMO/tools/qosmo-tbring/libtbring.so}"
        [ -f "$tbring" ] || tbring=/usr/local/lib/qosmo/libtbring.so
        if [ -f "$tbring" ]; then
            local ring="${ASSEMBLY_LOGS_RING_FILE:-$asm_dir/qemu-tbring}"
            rm -f "$ring" "$RUNDIR/qemu-tbring"
            [ "$(dirname "$ring")" != "$RUNDIR" ] && ln -s "$ring" "$RUNDIR/qemu-tbring"
            # anneau= et pas file= : file= est le mot-cle de QEMU pour le chemin du plugin
            gdb_opt+=(-plugin "$tbring,anneau=$ring,entries=${ASSEMBLY_LOGS_RING_ENTRIES:-8589934592}")
        else
            dire "   anneau tbring ignore : libtbring.so introuvable (make -C $QOSMO/tools/qosmo-tbring)"
        fi
    fi
fi
# [2026-10-19] QEMU_DEBUG_FLAGS=int,guest_errors : journal des exceptions de l'ARM (PC du
# Data Abort, acces a une adresse non mappee) dans $RUNDIR/qemu-debug.log. Sans
# ASSEMBLY_LOGS seulement (un seul -d/-D par ligne QEMU).
if [ -n "${QEMU_DEBUG_FLAGS:-}" ] && [ "${ASSEMBLY_LOGS:-0}" != 1 ]; then
    gdb_opt+=(-d "$QEMU_DEBUG_FLAGS" -D "$RUNDIR/qemu-debug.log")
fi
# [2026-10-19] QEMU_ICOUNT=auto|shift=N,... (start-native.sh --icount) : -icount.
case "${QEMU_ICOUNT:-}" in ''|0|off) ;; *) gdb_opt+=(-icount "$QEMU_ICOUNT") ;; esac
# [2026-09-23] Relance de QEMU vers le DSP toutes les trame/64 (0,07 ms) au
# lieu de trame/16 (0,29 ms) : la partie en serie d'une trame (DONE de la
# phase A, GO) payait deux fois cette latence -- mesure [chrono] en TCH.
CALYPSO_PONT_RETRY_DIV="${CALYPSO_PONT_RETRY_DIV:-64}" \
# [2026-10-21] FIRMWARE_FLASH : la flash (copie de travail dans $RUNDIR, gardee entre deux
# lancements pour le FFS ; FLASH_RESET=1 ou une image plus recente la remplacent) remplace -kernel
local fw_opt=(-kernel "$FIRMWARE_ELF")
```

```
    if [ -n "$FIRMWARE_FLASH" ]; then
        if [ "${FLASH_RESET:-0}" != 0 ] || [ ! -f "$RUNDIR/flash.img" ] || [ "$FIRMWARE_FLASH" -nt
         ↪  "$RUNDIR/flash.img" ]; then
            cp "$FIRMWARE_FLASH" "$RUNDIR/flash.img" || rater "copie de $FIRMWARE_FLASH impossible"
            dire "   flash de travail $RUNDIR/flash.img (re)creee depuis $FIRMWARE_FLASH"
        fi
        fw_opt=(-drive "if=pflash,file=$RUNDIR/flash.img,format=raw")
    fi
    CALYPSO_DSP_EXTERN="$extern" ${PRIO[@]+"${PRIO[@]}"} "$QEMU" -M calypso -cpu "${QEMU_CPU:-arm946}" -display
     ↪  none -parallel none \
        -serial pty -serial pty -monitor "unix:$MONITOR,server,nowait" "${gdb_opt[@]}" \
        "${fw_opt[@]}" > "$RUNDIR/qemu.log" 2>&1 &
    echo $! > "$RUNDIR/qemu.pid"
    if [ "$GDB" = 1 ] && [ -f "$GDB_TELNET_PY" ] && ! vivant gdb; then
        python3 "$GDB_TELNET_PY" --port "$GDB_TELNET" --stub "$GDB_STUB" --elf "$FIRMWARE_ELF" \
            > "$RUNDIR/gdb.log" 2>&1 &
        echo $! > "$RUNDIR/gdb.pid"
    fi
    attendre 10 grep -aq "label serial0" "$RUNDIR/qemu.log" || rater "QEMU n'a pas publie son pty (voir
     ↪  $RUNDIR/qemu.log)"
    if [ "$MODE" = dsp ]; then
        attendre 10 grep -aq "pont DSP : API RAM partagee" "$RUNDIR/qemu.log" || rater "QEMU n'a pas rejoint le
         ↪  DSP (voir $RUNDIR/qemu.log)"
    else
        attendre 5 grep -aq "backend gr-gsm" "$RUNDIR/qemu.log" || rater "la couche 1 gr-gsm ne s'est pas annoncee
         ↪  (voir $RUNDIR/qemu.log)"
    fi
    sed -n 's/.*redirected to \(\/dev\/pts\/[0-9]*\) (label serial0).*/\1/p' "$RUNDIR/qemu.log" | head -1 >
     ↪  "$RUNDIR/modem.pty"
    dire "2. qemu-system-arm  pid $(pid_de qemu)  pty modem $(cat "$RUNDIR/modem.pty")  moniteur $MONITOR  L1=$([
     ↪  "$MODE" = dsp ] && echo "DSP externe" || echo "gr-gsm (udp 4730/4731)")"
    [ "$GDB" = 1 ] && dire "   gdb ARM : telnet 0 $GDB_TELNET  (stub tcp::$GDB_STUB, journal $RUNDIR/gdb.log)"
    case "${QEMU_ICOUNT:-}" in ''|0|off) ;; *) dire "   icount : $QEMU_ICOUNT" ;; esac
    [ -n "${QEMU_DEBUG_FLAGS:-}" ] && [ "${ASSEMBLY_LOGS:-0}" != 1 ] && dire "   journal QEMU -d $QEMU_DEBUG_FLAGS
     ↪  : $RUNDIR/qemu-debug.log"
    # [2026-10-19] moniteur mailbox (CALYPSO_MAILBOX=1) : deux fichiers, un par cote
    if [ -n "${CALYPSO_MAILBOX:-}" ] && [ "${CALYPSO_MAILBOX}" != 0 ]; then
        local _mb="${CALYPSO_MAILBOX_FILE:-${LOG_DIR:-/tmp/calypso/logs}/mailbox.log}"
        dire "   mailbox ARM<->DSP : ${_mb} (cote DSP, c54x_exe) et ${_mb%.log}-arm.log (cote ARM, QEMU) ;
         ↪  CALYPSO_MAILBOX=$CALYPSO_MAILBOX"
    fi
    [ "${ASSEMBLY_LOGS:-0}" = 1 ] && dire "   trace asm ARM : $(readlink -f "$RUNDIR/qemu-asm.log") (-d
     ↪  ${ASSEMBLY_LOGS_FLAGS:-in_asm}${ASSEMBLY_LOGS_FILTRE:+, -dfilter
     ↪  $ASSEMBLY_LOGS_FILTRE}${ASSEMBLY_LOGS_FIFO:+, via FIFO}, QEMU_LOG_BUFFERED=$QEMU_LOG_BUFFERED)$([
     ↪  "${ASSEMBLY_LOGS_RING:-1}" = 1 ] && echo " ; anneau des blocs executes : $(readlink -f
     ↪  "$RUNDIR/qemu-tbring" 2>/dev/null) (tbring-dump)")"
}

etape3_rvinterf() {   # FIRMWARE_FLASH : rvinterf (FreeCalypso) a la place d'osmocon
    vivant rvinterf && { dire "3. rvinterf deja lance (pid $(pid_de rvinterf))"; return; }
    [ -x "$RVINTERF_BIN" ] || { dire "3. (rvinterf absent : $RVINTERF_BIN ; pas de traces RVTMUX)"; return; }
    local pty; pty="$(cat "$RUNDIR/modem.pty")"
    rm -f "$RUNDIR/rvinterf.sock"
    "$RVINTERF_BIN" -n -l "$RUNDIR/rvinterf.log" -s "$RUNDIR/rvinterf.sock" "$pty" > "$RUNDIR/rvinterf.out" 2>&1 &
    echo $! > "$RUNDIR/rvinterf.pid"
    dire "3. rvinterf  pid $(pid_de rvinterf)  traces RVTMUX -> $RUNDIR/rvinterf.log  socket $RUNDIR/rvinterf.sock
     ↪  (fc-shell -s / fc-fsio -s)"
}
etape3() {   # osmocon
    [ -n "$FIRMWARE_FLASH" ] && { etape3_rvinterf; return; }
    vivant osmocon && { dire "3. osmocon deja lance (pid $(pid_de osmocon))"; return; }
    vivant qemu || rater "QEMU ne tourne pas (etape 2 d'abord)"
    [ -x "$OSMOCON" ] || rater "osmocon absent : $OSMOCON"
    [ -r "$FIRMWARE_BIN" ] || rater "image .bin absente : $FIRMWARE_BIN"
    local pty; pty="$(cat "$RUNDIR/modem.pty")"
    rm -f "$L2_SOCK"
```

```
    vitrine osmocon
    ${PRIO[@]+"${PRIO[@]}"} stdbuf -oL -eL "$OSMOCON" -m romload -i 100 -p "$pty" -s "$L2_SOCK" "$FIRMWARE_BIN" >
     ↪  "$RUNDIR/osmocon.log" 2>&1 &
    echo $! > "$RUNDIR/osmocon.pid"
    if ! attendre 30 grep -aq "your code is running now" "$RUNDIR/osmocon.log"; then
        rater "osmocon n'a pas fini le romload (voir $RUNDIR/osmocon.log). Un osmocon tue a mi-bloc laisse
       le stub romload de l'UART en attente : ./run.sh --stop puis ./run.sh"
    fi
    dire "3. osmocon  pid $(pid_de osmocon)  L1CTL sur $L2_SOCK"
}

# [2026-10-15] BANC_VOCODEUR=rom : la ROM vocode (port voix modelise, CALYPSO_VSP=1).
# Le mobile doit alors etre en `io-handler l1phy` -- seul mode ou le firmware leve les
# mutes du vocodeur (prim_tch.c:101-111) -- et SANS io-tch-format ni ALSA dans le bloc
# tch-voice : sous l1phy le mobile refuse io-tch-format et abandonne TOUT le fichier.
# La variante est donc generee ici depuis MOBILE_CFG, jamais ecrite a la main.
config_vocodeur_rom() {
    local src="$1" dst="$RUNDIR/mobile_pont_rom.cfg"
    awk '
        /^ tch-voice/ { voix = 1; print; next }
        /^ [^ ]/      { voix = 0 }
        voix && /^  io-handler gapk/ { print "  io-handler l1phy"; next }
        voix && /^  (io-tch-format|alsa-output-dev|alsa-input-dev)/ { next }
        { print }' "$src" > "$dst" || rater "variante l1phy impossible : $dst"
    grep -q "^  io-handler l1phy" "$dst" || rater "variante l1phy : pas de bloc tch-voice gapk dans $src"
    echo "$dst"
}

etape4_trx() {   # OSMO_TRX=1 : l'appli hote transceiver, a la place du mobile
    [ "$OSMO_TRX_PHONE" = 2 ] && { dire "4. (OSMO_TRX_PHONE=2 : telephone 2 du transceiver de l'instance 1, TS1 ;
     ↪  pas de transceiver ici)"; return; }
    vivant transceiver && { dire "4. transceiver deja lance (pid $(pid_de transceiver))"; return; }
    [ -S "$L2_SOCK" ] || rater "pas de socket L1CTL $L2_SOCK (etape 3 d'abord)"
    local args_ts1=""
    if [ "$OSMO_TRX_TS1" = 1 ]; then
        local l2_3="/tmp/osmocom_l2_$OSMO_TRX_TS1_INSTANCE"
        if [ ! -S "$l2_3" ]; then
            dire "4a. telephone 2 (TS1) : instance $OSMO_TRX_TS1_INSTANCE (c54x_exe + QEMU trx + osmocon)"
            INSTANCE="$OSMO_TRX_TS1_INSTANCE" OSMO_TRX=1 OSMO_TRX_PHONE=2 AIR=0 PONT=0 bash "$0" || rater
             ↪  "l'instance $OSMO_TRX_TS1_INSTANCE (telephone 2) n'a pas demarre"
        fi
        attendre 10 test -S "$l2_3" || rater "pas de socket L1CTL $l2_3 pour le telephone 2"
        args_ts1="-2 -L $l2_3"
    fi
    [ -x "$TRANSCEIVER" ] || rater "transceiver absent : $TRANSCEIVER (make -C
     ↪  /opt/GSM/osmocom-bb-transceiver/src/host/layer23/src/transceiver)"
    local p
    # [2026-10-19] Colonne LOCALE de ss ($4 ; $5 est le peer), tous etats : avec « -uan | grep » la colonne peer
     ↪  matchait
    # aussi (osmo-bts-trx 5800 -> 127.0.0.1:5700 passait pour un tenant), et « -uln » ne voit pas
    # les sockets du transceiver, qui sont CONNECTEES (ESTAB) vers la BTS, pas en ecoute.
    for p in 5700 5701 5702; do
        ss -uanH 2>/dev/null | awk -v p="$p" '$4 ~ ("[:.]" p "$") {found=1} END {exit !found}' \
            && rater "port UDP $p deja pris par $(ss -uanpH 2>/dev/null | awk -v p="$p" '$4 ~ ("[:.]" p "$")' |
             ↪  grep -o 'users:.*' | head -1) (un pont.py qui traine ? ./run.sh --stop)"
    done
    vitrine transceiver
    # -l : socket L1CTL d'osmocon ; -m : masque de timeslots RX (un seul pour commencer) ; le masque TX
    # garde son defaut 0xe3 (OSMO_TRX_TX_MASK pour le forcer : -M), voir l'en-tete
    ${PRIO[@]+"${PRIO[@]}"} stdbuf -oL -eL "$TRANSCEIVER" -l "$L2_SOCK" -m "$OSMO_TRX_TS_MASK"
     ↪  ${OSMO_TRX_TX_MASK:+-M "$OSMO_TRX_TX_MASK"} ${OSMO_TRX_BSIC:+-b "$OSMO_TRX_BSIC"} $args_ts1
     ↪  $OSMO_TRX_ARGS > "$RUNDIR/transceiver.log" 2>&1 &
    echo $! > "$RUNDIR/transceiver.pid"
    attendre 10 sh -c "ss -uanH 2>/dev/null | awk '\$4 ~ /[:.]5702\$/ {f=1} END {exit !f}'" || rater "transceiver
     ↪  n'ecoute pas en 5700-5702 (voir $RUNDIR/transceiver.log)"
    vivant transceiver || rater "transceiver s'est arrete : $(tail -1 "$RUNDIR/transceiver.log")"
```

```
    # [2026-10-20] osmo-bts-trx (BTS#0) SORT quand le transceiver disparait (« No more clock from
    # transceiver », bts_shutdown exit 1) : apres un --stop la BTS n'est plus la, et le plan du fork
    # (qosmo/run.sh --only bts) refuse de la relancer seule (dependance bsc). On la relance ici comme
    # le module 60-bts.sh, dans le journal du panneau. OSMO_TRX_BTS_RELANCE=0 pour ne rien faire.
    if [ "${OSMO_TRX_BTS_RELANCE:-1}" = 1 ]; then
        local bts_cfg="${BTS_CFG:-/etc/osmocom/osmo-bts-trx.cfg}" bts_bin bts_log bts_pid _i
        # La BTS d'avant peut etre encore en train de s'eteindre (bts_shutdown : jusqu'a 70 s mesures) :
        # on attend qu'elle soit partie, ou qu'elle ait allume le firmware (POWERON -> « BTS MODE: 7 514 »).
        for _i in $(seq 1 "${OSMO_TRX_BTS_ATTENTE:-90}"); do
            pgrep -f -- "osmo-bts-trx -c $bts_cfg\$" >/dev/null 2>&1 || break
            grep -aq 'BTS MODE: [0-9]* [1-9]' "$RUNDIR/osmocon.log" 2>/dev/null && break
            [ "$_i" = 1 ] && dire "   BTS#0 presente mais muette : on attend qu'elle allume le firmware ou qu'elle
             ↪  finisse de s'eteindre (${OSMO_TRX_BTS_ATTENTE:-90} s max)"
            sleep 1
        done
        if ! pgrep -f -- "osmo-bts-trx -c $bts_cfg\$" >/dev/null 2>&1; then
            bts_bin="${OSMO_BTS_TRX_BIN:-$(command -v osmo-bts-trx || echo /usr/local/bin/osmo-bts-trx)}"
            bts_log="$(ls -d /run/user/*/osmo-nitb/logs 2>/dev/null | head -1)";
             ↪  bts_log="${bts_log:-$RUNDIR}/bts.log"
            if [ -x "$bts_bin" ] && [ -r "$bts_cfg" ]; then
               setsid stdbuf -oL -eL $(chrt -r 1 true 2>/dev/null && echo "chrt -r 1") "$bts_bin" -c "$bts_cfg" >>
                ↪  "$bts_log" 2>&1 </dev/null &
                bts_pid=$!
                for bts_piddir in /run/user/*/osmo-nitb; do [ -d "$bts_piddir" ] && echo "$bts_pid" >
                 ↪  "$bts_piddir/bts.pid"; done
                dire "4b. osmo-bts-trx relancee (BTS#0 absente : elle sort quand le transceiver disparait)  pid
                 ↪  $bts_pid  $bts_cfg  journal $bts_log"
            else
                dire "4b. BTS#0 absente et non relancable (binaire $bts_bin ou config $bts_cfg) :
                 ↪  ./start-native.sh --osmo-trx"
            fi
        fi
    fi
    dire "4. transceiver  pid $(pid_de transceiver)  L1CTL $L2_SOCK  TRXD 5700-5702 <- osmo-bts-trx  timeslots RX
     ↪  $OSMO_TRX_TS_MASK TX ${OSMO_TRX_TX_MASK:-0xe3}  bsic ${OSMO_TRX_BSIC:-?}  ($OSMO_TRX_ARGS)${args_ts1:+  +
     ↪  telephone 2 TS1 : $args_ts1}"
}

etape4() {   # le mobile
    [ -n "$FIRMWARE_FLASH" ] && { dire "4. (FIRMWARE_FLASH : pile L23 et UI dans le firmware Magnetite, pas de
     ↪  mobile ; ecran C139 : tools/osmo-c123-gtk.py --modele c139 --instance $INSTANCE)"; return; }
    [ -n "$FIRMWARE_APP" ] && { dire "4. (FIRMWARE_APP=$FIRMWARE_APP : pas de mobile ; ecran /dev/shm/calypso_lcd,
     ↪  touches /tmp/calypso_keys.sock, fenetre : tools/osmo-c123-gtk.py)"; return; }
    [ "$OSMO_TRX" = 1 ] && { etape4_trx; return; }
    vivant mobile && { dire "4. mobile deja lance (pid $(pid_de mobile))"; return; }
    if [ "${BANC_VOCODEUR:-hote}" = rom ]; then
        MOBILE_CFG="$(config_vocodeur_rom "$MOBILE_CFG")"
        dire "4. vocodeur ROM (BANC_VOCODEUR=rom) : tch-voice en l1phy -> $MOBILE_CFG"
    fi
    [ -S "$L2_SOCK" ] || rater "pas de socket L1CTL $L2_SOCK (etape 3 d'abord)"
    if [ "$INSTANCE" != 1 ]; then
        # [2026-10-19] sockets et VTY propres a l'instance (layer2/sap/ms_data suffixes _N, VTY +10 par instance)
        local _src="$MOBILE_CFG" _dst="$RUNDIR/mobile_instance$INSTANCE.cfg"
        # le port VTY est decale de 100 par instance (4257 = osmo-ggsn ; awk additionne, sed non ; sub() garde
        # l'indentation : « $3 = ... » reconstruisait la ligne sans l'espace de tete et le VTY la refusait)
        awk -v inc=$((100 * (INSTANCE - 1))) '/^[[:space:]]*bind 127\.0\.0\.1 [0-9]+/ { n = $3 + inc;
         ↪  sub(/[0-9]+$/, n) } { print }' "$_src" \
            | sed -e "s#/tmp/osmocom_l2\b#$L2_SOCK#" -e "s#/tmp/osmocom_sap\b#/tmp/osmocom_sap_$INSTANCE#" -e
             ↪  "s#/tmp/ms_data\b#/tmp/ms_data_$INSTANCE#" > "$_dst"
        # [2026-10-21] « stick » suit l'ARFCN du plan radio (BTS#0) quand il a change (gabarit : 514)
        local _arfcn; _arfcn="$( . /etc/osmocom/radio-plan.env 2>/dev/null; echo "${PLAN_ARFCN:-}")"
        [ -n "$_arfcn" ] && sed -i "s|^\([[:space:]]*\)stick [0-9]*\$|\1stick $_arfcn|" "$_dst"
        MOBILE_CFG="$_dst"
        dire "4. instance $INSTANCE : config mobile derivee -> $MOBILE_CFG"
    fi
    [ -x "$MOBILE" ] || rater "mobile absent : $MOBILE"
```

```
    [ -r "$MOBILE_CFG" ] || rater "config mobile absente : $MOBILE_CFG"
    local vty; vty="$(sed -n 's/^ *bind 127.0.0.1 \([0-9]*\).*/\1/p' "$MOBILE_CFG" | head -1)"
    if [ -n "$vty" ] && ss -ltn 2>/dev/null | grep -q ":$vty "; then
        rater "le port VTY $vty de $MOBILE_CFG est deja pris par : $(ss -ltnp 2>/dev/null | grep ":$vty " | grep
        ↪  -o 'users:.*' | head -1)
       changez la ligne « bind 127.0.0.1 $vty » de la config (les 42xx sont ceux du reseau du banc)"
    fi
    vitrine mobile
    # [2026-09-23] Resynchro sur la cellule choisie : 8 essais au lieu de 1
    # (gsm322.c, sync_retries_selection). Le DSP ne passe le SB qu'une fois sur
    # trois a cinq, et deux echecs suffisaient a la boucle « no service ».
    # Montage grgsm : defaut du binaire, inchange.
    local retries=""
    [ "$MODE" = dsp ] && retries="${L23_SYNC_RETRIES_SELECTION:-8}"
    L23_SYNC_RETRIES_SELECTION="$retries" \
    ${PRIO[@]+"${PRIO[@]}"} stdbuf -oL "$MOBILE" -c "$MOBILE_CFG" > "$RUNDIR/mobile.log" 2>&1 &
    echo $! > "$RUNDIR/mobile.pid"
    sleep 2
    vivant mobile || rater "mobile s'est arrete : $(sed 's/\x1b\[[0-9;]*m//g' "$RUNDIR/mobile.log" | grep -iE
    ↪  'cannot|error|unable' | tail -1)"
    dire "4. mobile  pid $(pid_de mobile)  config $MOBILE_CFG  vty telnet 127.0.0.1 ${vty:-?}"
}

etape5_air() {   # AIR=1 : l'air entre les deux Calypsos (pont/air.py), a la place du pont
    if vivant air; then dire "5. air deja lance (pid $(pid_de air))"; etape5_vsp; return; fi
    [ -r "$AIR_PY" ] || rater "air.py absent : $AIR_PY"
    local autre_shm="$HOTE_SHM" autre_port=6702
    [ "$INSTANCE" = 1 ] && { autre_shm=/dev/shm/calypso2; autre_port=6712; }
    vitrine air
    # A = l'autre instance (son anneau TX, son port BSP), B = celle-ci
    # [2026-10-20] le telephone 2 du TRX (TS1, instance 3 : /dev/shm/calypso3, BSP udp 6722) s'il tourne, et
    ↪  l'annonce du
    # TCH au mobile (calypso_tch_cfg de CE /dev/shm) quand l'air decode l'ASSIGNMENT COMMAND
    local args_c=""
    if [ -S "/tmp/osmocom_l2_${OSMO_TRX_TS1_INSTANCE}" ] && [ -f "/tmp/c54x-pont${OSMO_TRX_TS1_INSTANCE}/dsp.pid"
    ↪  ] \
       && kill -0 "$(cat "/tmp/c54x-pont${OSMO_TRX_TS1_INSTANCE}/dsp.pid" 2>/dev/null)" 2>/dev/null; then
        args_c="--c-ring $HOTE_SHM/calypso${OSMO_TRX_TS1_INSTANCE}/calypso_tx_rom --c-port $((6702 + 10 *
        ↪  (OSMO_TRX_TS1_INSTANCE - 1) + (OSMO_TRX_TS1_INSTANCE >= 3 ? 30 : 0)))"
    fi
    ${PRIO[@]+"${PRIO[@]}"} python3 -u "$AIR_PY" --a-ring "$autre_shm/calypso_tx_rom" --a-port "$autre_port" \
        --b-ring /dev/shm/calypso_tx_rom --b-port "${CALYPSO_BSP_PORT:-6702}" $args_c --tch-cfg
        ↪  /dev/shm/calypso_tch_cfg > "$RUNDIR/air.log" 2>&1 &
    echo $! > "$RUNDIR/air.pid"
    attendre 5 grep -aq "air :" "$RUNDIR/air.log" || rater "air.py ne s'est pas annonce (voir $RUNDIR/air.log)"
    dire "5. air  pid $(pid_de air)  $autre_shm/calypso_tx_rom -> udp $CALYPSO_BSP_PORT (ici) ;
    ↪  /dev/shm/calypso_tx_rom (ici) -> udp $autre_port (l'autre)${args_c:+ ; TS1 : $args_c}"
    etape5_vsp
}

# [2026-10-20] le port voix (CALYPSO_VSP=1) : anneaux PCM du DSP <-> PulseAudio, aussi en mode air (le son
# de l'appel passait par le pont ; sans lui, appel etabli mais muet). Lance dans l'espace de l'instance :
# /dev/shm/calypso_pcm_ul et _dl sont ceux de CE mobile.
etape5_vsp() {
    [ "${CALYPSO_VSP:-0}" = 1 ] || return 0
    vivant vsp && { dire "5b. pont audio VSP deja lance (pid $(pid_de vsp))"; return 0; }
    ( cd /opt/GSM/osmo-operator && exec python3 -m pont.vsp_audio ) > "$RUNDIR/vsp.log" 2>&1 &
    echo $! > "$RUNDIR/vsp.pid"
    dire "5b. pont audio VSP  pid $(pid_de vsp)  gsm_mic.monitor -> calypso_pcm_ul, calypso_pcm_dl -> gsm_audio
    ↪  (journal $RUNDIR/vsp.log)"
}

etape5() {   # le pont TRX (PONT=1) : bursts du BTS vers la couche 1
    [ "$AIR" = 1 ] && { etape5_air; return; }
    [ "$OSMO_TRX" = 1 ] && { dire "5. (OSMO_TRX=1 : pas de pont, transceiver tient 5700-5702)"; return; }
    [ "$OSMO_TRX_PHONE" = 2 ] && { dire "5. (telephone 2 : rien)"; return; }
```

```
    [ -n "$FIRMWARE_APP" ] && { dire "5. (FIRMWARE_APP=$FIRMWARE_APP : pas de pont)"; return; }
    [ "$PONT" = 1 ] || { dire "5. (PONT=0 : pas de pont.py, aucun burst du BTS n'arrive)"; return; }
    vivant pont && { dire "5. pont.py deja lance (pid $(pid_de pont))"; return; }
    [ -r "$PONT_PY" ] || rater "pont.py absent : $PONT_PY"
    local extra=""; [ "$MODE" = dsp ] && extra="--dsp-port 6702"
    [ "$PONT_AIRREC" = 0 ] && extra="$extra --no-record"
    vitrine pont
    # [2026-10-04] UL_RAW=1 (defaut) : le pont emet les bursts montants finaux de la ROM (calypso_tx_rom,
    # src/tsp_tx.c) tels quels, a son numero de trame -- RACH, SDCCH, SACCH, FACCH, parole, A5 compris.
    # UL_RAW=0 : codage et A5 cote hote comme avant (PONT_UL_ROM reste le chemin xCCH de 2026-10-03).
    # [2026-10-19] Trace asm de l'ARM : le couple ARM+DSP peut tourner a 14 fois le
    # temps reel (exec,nochain). L'horloge asservie du pont (pont/trx.py) ne
    # descend que 4 fois sous le temps reel par defaut ; on lui permet 16 fois
    # (mesure : 10 a 14 fois ; au-dela de 16 la pause demandee a osmo-bts-trx
    # entre deux IND CLOCK approche sa limite de 50 trames, voir trx.py
    # run_clock). PONT_HORLOGE_MAX deja pose dans l'environnement : on le respecte.
    if [ "${ASSEMBLY_LOGS:-0}" = 1 ]; then export PONT_HORLOGE_MAX="${PONT_HORLOGE_MAX:-16}"; fi
    ( cd "$(dirname "$PONT_PY")/.." && bruit_env_pont && export PONT_UL_RAW="${UL_RAW:-1}" && exec
     ↪  ${PRIO[@]+"${PRIO[@]}"} python3 "$PONT_PY" $extra ) > "$RUNDIR/pont.log" 2>&1 &
    echo $! > "$RUNDIR/pont.pid"
    attendre 10 grep -aq "pont TRX : ports" "$RUNDIR/pont.log" || rater "pont.py ne s'est pas annonce (voir
     ↪  $RUNDIR/pont.log)"
    dire "5. pont.py  pid $(pid_de pont)  TRXD 5700-5702 <- BTS ; vers $([ "$MODE" = dsp ] && echo "le DSP (udp
     ↪  6702)" || echo "la L1 gr-gsm (udp 4730/4731)")"
    [ -n "${PONT_HORLOGE_MAX:-}" ] && dire "   horloge asservie : ralentissement jusqu'a ${PONT_HORLOGE_MAX} x le
     ↪  temps reel (PONT_HORLOGE_MAX)"
    # [2026-10-15] port voix (CALYPSO_VSP=1) : anneaux PCM du DSP <-> PulseAudio
    if [ "${CALYPSO_VSP:-0}" = 1 ] && ! vivant vsp; then
        ( cd /opt/GSM/osmo-operator && exec python3 -m pont.vsp_audio ) > "$RUNDIR/vsp.log" 2>&1 &
        echo $! > "$RUNDIR/vsp.pid"
       dire "5b. pont audio VSP  pid $(pid_de vsp)  gsm_mic.monitor -> calypso_pcm_ul, calypso_pcm_dl -> gsm_audio"
    fi
}

arreter() {
    local n
    for n in vsp pont air bruit mobile transceiver osmocon rvinterf gdb qemu dsp asmcat; do
        if vivant "$n"; then kill "$(pid_de "$n")" 2>/dev/null; dire "arret $n (pid $(pid_de "$n"))"; fi
        rm -f "$RUNDIR/$n.pid"
    done; bruit_arreter
    sleep 1
    # [2026-10-23] ARRET COMPLET D'UNE INSTANCE N != 1 (banc-rssi --stop) : les processus sans fichier pid ou dont
    # le pid a ete perdu (relance par-dessus une instance vivante) gardaient le port BSP (udp 6762 tenu par un
    # c54x_exe fantome, le nouveau ne recevait rien). On balaie ce qui porte les ressources propres a l'instance.
    if [ "$INSTANCE" != 1 ]; then
        # Pas « $RUNDIR/ » : un simple tail -F de ses journaux, ou le shell qui a lance --stop, y passaient.
        local motif="$DSP_SOCK|$MONITOR|$L2_SOCK|--stub $GDB_STUB |--port $GDB_TELNET "
        local p=$$ ancetres=" $$ "
        while p="$(awk '/^PPid:/ {print $2}' "/proc/$p/status" 2>/dev/null)" && [ -n "$p" ] && [ "$p" -gt 1 ]; do
            ancetres="$ancetres$p "
        done
        local restes="" r
        for r in $(pgrep -f -- "$motif"); do
            case "$ancetres" in *" $r "*) ;; *) restes="$restes$r " ;; esac
        done
        if [ -n "${restes// /}" ]; then
            dire "instance $INSTANCE : restes arretes (pid $restes)"
            kill $restes 2>/dev/null; sleep 1; kill -9 $restes 2>/dev/null
        fi
    fi
    rm -f "$DSP_SHM" "$DSP_SOCK" "$L2_SOCK" "$MONITOR" "$RUNDIR/modem.pty" /dev/shm/qemu-asm.fifo
    # [2026-09-22] Sockets du mobile : elles ne disparaissent pas avec lui. Le
    # « sap » etait deja oublie, et « ms_data » (tch-data, mobile_pont.cfg)
    # arrive avec la meme faiblesse : un fichier reste fait echouer le bind du
    # run suivant sur EADDRINUSE, et le mobile demarre sans sa voie donnees
    # sans le dire.
```

```
    # [2026-10-23] SEULEMENT pour l'instance 1 : ces sockets sont dans le /tmp COMMUN et appartiennent au mobile de
    # l'instance 1 ; banc-rssi --stop (instance 4) les effacait sous le mobile en marche (SAP, tch-data).
    [ "$INSTANCE" = 1 ] && rm -f /tmp/osmocom_sap /tmp/ms_data
    # [2026-09-22] Horloge du banc (calypso_bsp.c -> pont/trx.py) : laissee en
    # place, pont.py asservirait sa premiere seconde sur la trame d'une session
    # morte et la BTS resterait figee en attendant un DSP qui n'existe plus.
    rm -f /dev/shm/calypso_horloge
    # Side-bands du lien montant (src/montant.c) : sinon pont.py relit le
    # dernier enregistrement d'une session precedente au demarrage.
    rm -f /dev/shm/calypso_rach /dev/shm/calypso_sdcch_ul \
          /dev/shm/calypso_tch_facch_ul /dev/shm/calypso_tch_sacch_ul \
          /dev/shm/calypso_tch_ul
    # [2026-09-23] Annonce du TCH par le pont DSP (pont/dsp/tch.py -> montant.c
    # scruter_tch). Montage dsp seulement : en grgsm ce fichier est celui de la
    # L1 gr-gsm de QEMU, on n'y touche pas.
    [ "$MODE" = dsp ] && rm -f /dev/shm/calypso_tch_cfg
}

statut() {
    local n
    for n in dsp qemu osmocon rvinterf mobile transceiver pont air; do
        if vivant "$n"; then printf '  %-8s pid %-7s  %s\n' "$n" "$(pid_de "$n")" "$RUNDIR/$n.log"
        else printf '  %-8s arrete\n' "$n"; fi
    done; bruit_statut
    [ -f "$RUNDIR/dsp.log" ] && grep -a "fn=" "$RUNDIR/dsp.log" | tail -1
    [ -f "$RUNDIR/osmocon.log" ] && grep -a "FB0\|FB1\|SB\|BSIC" "$RUNDIR/osmocon.log" | tail -1
}

# [2026-09-23] GARDER LES JOURNAUX DU DSP ET DU PONT. dsp.log est ecrase a
# chaque lancement et pont.log est vide par vitrine ; aucun des deux n'est dans
# les archives du panneau (osmo-nitb/archives). Deux appels de suite ont ete
# perdus ainsi (13:11 et 13:21 : SP-CORRUPT et FACCH ko sans trace). On range
# la session precedente dans $RUNDIR/archives/<date de dsp.log>/ et on en garde
# JOURNAUX_GARDES (10). JOURNAUX_GARDES=0 ne garde rien.
JOURNAUX_GARDES="${JOURNAUX_GARDES:-10}"
garder_journaux() {
    [ "$JOURNAUX_GARDES" -gt 0 ] 2>/dev/null || return 0
    [ -s "$RUNDIR/dsp.log" ] || [ -s /dev/shm/pont.log ] || return 0
    local d="$RUNDIR/archives/$(date -r "$RUNDIR/dsp.log" +%Y%m%d-%H%M%S 2>/dev/null || date +%Y%m%d-%H%M%S)"
    [ -d "$d" ] && return 0                    # deja range (--stop puis relance)
    mkdir -p "$d" || return 0
    local n
    for n in dsp pont mobile qemu osmocon bruit; do
        [ -s "$RUNDIR/$n.log" ] && cp -L "$RUNDIR/$n.log" "$d/" 2>/dev/null
    done
    # etat du magasin dedie du BSP (stockes/joues/manques/replis/perdues)
    [ -s /dev/shm/calypso_bsp_dedie ] && cp /dev/shm/calypso_bsp_dedie "$d/bsp_dedie.txt" 2>/dev/null
    ls -1dt "$RUNDIR"/archives/*/ 2>/dev/null | tail -n +$((JOURNAUX_GARDES + 1)) | xargs -r rm -rf
    dire "journaux de la session precedente ranges dans $d"
}

# [2026-10-03] INJECTEUR DE BRUIT, INACTIF PAR DEFAUT (BRUIT_MODE et BRUIT_SNR_DB vides :
# rien ne change, aucune ligne de plus a l'ecran). Place ici, en fin de fichier, pour que
# les numeros de ligne cites par le wiki (Lancer-a-la-main.md...) restent justes ; les
# etapes ci-dessus n'appellent que bruit_env_dsp, bruit_dire_dsp, bruit_env_pont,
# bruit_arreter et bruit_statut, sans effet par defaut.
# Source unique : $QOSMO/tools/injecteur_bruit.py, repli sur tools/injecteur_bruit.py d'ici
# (un lien vers elle).
#   BRUIT_MODE     ber | souple | iq (EXPERIMENTAL) | relais : etape b (--step b), entre le
#                  DSP et QEMU dans la sequence, entre le pont et la L1 dans le flux.
#     dsp   : c54x_exe ecoute BRUIT_PORT_DSP (16702, CALYPSO_BSP_PORT), l'injecteur prend
#             127.0.0.1:6702 a sa place ; le pont ne change pas (--dsp-port 6702). Le BSP
#             apprend son pair montant sur l'emetteur du descendant : son montant passe par
#             l'injecteur, qui le rend tel quel au pont (5702) depuis 6702.
#     grgsm : l'injecteur ecoute BRUIT_PORT_GRGSM (14730) et relaie vers QEMU 4730 ; le pont
#             y envoie (PONT_GSMTAP_PORT, pose a l'etape 5). ber | relais seulement : QEMU
```

```
#             recoit des blocs L2 deja decodes.
#     iq    : refuse si CALYPSO_BSP_STREAM=1 (defaut de start-native.sh --dsp).
#   BRUIT_BER (0..1), BRUIT_RAFALES (L bits), BRUIT_SNR_DB (dB), BRUIT_PERTE (0..1),
#   BRUIT_TN (0 ou 1,2...), BRUIT_GRAINE, BRUIT_STATS (10 s), BRUIT_OPTS (options en plus),
#   BRUIT_PORT_DSP, BRUIT_PORT_GRGSM, BRUIT_PY.
#   BRUIT_SNR_DB SANS BRUIT_MODE (dsp) : pas d'injecteur, AWGN du coeur C54x -- c54x_exe
#   recoit CALYPSO_BSP_SNR_DB (et CALYPSO_BSP_BRUIT_GRAINE <- BRUIT_GRAINE), calypso_c54x.c.
# Une configuration refusee (--verifier) arrete run.sh AVANT tout lancement.
# Journal : $RUNDIR/bruit.log. Detail : /opt/GSM/osmo-operator/wiki/Injecteur-bruit.md.
BRUIT_MODE="${BRUIT_MODE:-}"
BRUIT_SNR_DB="${BRUIT_SNR_DB:-}"
BRUIT_PORT_DSP="${BRUIT_PORT_DSP:-16702}"
BRUIT_PORT_GRGSM="${BRUIT_PORT_GRGSM:-14730}"
if [ -z "${BRUIT_PY:-}" ]; then
    BRUIT_PY="$QOSMO/tools/injecteur_bruit.py"
    [ -r "$BRUIT_PY" ] || BRUIT_PY="$HERE/tools/injecteur_bruit.py"
fi
BRUIT_ACTIF=0      # 1 : l'injecteur est intercale pour ce lancement (bruit_preparer)
BRUIT_ARGS=()

bruit_args() {   # remplit BRUIT_ARGS
    BRUIT_ARGS=(--mode "$BRUIT_MODE" --etiquette c54x --stats "${BRUIT_STATS:-10}")
    if [ "$MODE" = dsp ]; then
        BRUIT_ARGS+=(--cible dsp --ecoute 127.0.0.1:6702 --vers "127.0.0.1:$BRUIT_PORT_DSP")
    else
        BRUIT_ARGS+=(--cible grgsm --ecoute "127.0.0.1:$BRUIT_PORT_GRGSM" --vers 127.0.0.1:4730)
    fi
    [ -n "${BRUIT_BER:-}" ]     && BRUIT_ARGS+=(--ber "$BRUIT_BER")
    [ -n "${BRUIT_RAFALES:-}" ] && BRUIT_ARGS+=(--rafales "$BRUIT_RAFALES")
    [ -n "$BRUIT_SNR_DB" ]      && BRUIT_ARGS+=(--snr-db "$BRUIT_SNR_DB")
    [ -n "${BRUIT_PERTE:-}" ]   && BRUIT_ARGS+=(--perte "$BRUIT_PERTE")
    [ -n "${BRUIT_TN:-}" ]      && BRUIT_ARGS+=(--tn "$BRUIT_TN")
    [ -n "${BRUIT_GRAINE:-}" ]  && BRUIT_ARGS+=(--graine "$BRUIT_GRAINE")
    [ -n "${BRUIT_OPTS:-}" ]    && BRUIT_ARGS+=($BRUIT_OPTS)
    return 0
}

bruit_preparer() {   # avant l'etape 1 : decide BRUIT_ACTIF, verifie la configuration
    [ -n "$BRUIT_MODE" ] || return 0
    if vivant bruit; then BRUIT_ACTIF=1; return 0; fi
    if [ "$MODE" = dsp ] && vivant dsp; then
        # Un c54x_exe deja lance n'est intercalable que s'il ecoute deja le port deplace.
        if ! tr '\0' '\n' < "/proc/$(pid_de dsp)/environ" 2>/dev/null | grep -qx
         ↪  "CALYPSO_BSP_PORT=$BRUIT_PORT_DSP"; then
            dire "b. BRUIT_MODE=$BRUIT_MODE ignore : c54x_exe tourne deja sur 6702, sans injecteur (./run.sh
             ↪  --stop puis ./run.sh)"
            return 0
        fi
    fi
    [ -r "$BRUIT_PY" ] || rater "injecteur de bruit absent : $BRUIT_PY (posez BRUIT_PY)"
    bruit_args
    local msg
    msg="$(python3 "$BRUIT_PY" "${BRUIT_ARGS[@]}" --verifier 2>&1)" || rater "injecteur de bruit : $(printf '%s\n'
     ↪  "$msg" | tail -1)"
    BRUIT_ACTIF=1
}

bruit_env_dsp() {   # dans le sous-shell de c54x_exe ; ne pose rien par defaut
    [ "$BRUIT_ACTIF" = 1 ] && export CALYPSO_BSP_PORT="$BRUIT_PORT_DSP"
    if [ -z "$BRUIT_MODE" ] && [ -n "$BRUIT_SNR_DB" ]; then
        export CALYPSO_BSP_SNR_DB="${CALYPSO_BSP_SNR_DB:-$BRUIT_SNR_DB}"
       [ -n "${BRUIT_GRAINE:-}" ] && export CALYPSO_BSP_BRUIT_GRAINE="${CALYPSO_BSP_BRUIT_GRAINE:-$BRUIT_GRAINE}"
    fi
    return 0
}
```

```
bruit_dire_dsp() {   # apres l'etape 1 ; muet par defaut
    [ "$BRUIT_ACTIF" = 1 ] && dire "   BSP deplace sur udp/$BRUIT_PORT_DSP (CALYPSO_BSP_PORT) : 6702 est a
     ↪  l'injecteur de bruit"
    [ -z "$BRUIT_MODE" ] && [ -n "$BRUIT_SNR_DB" ] && dire "   bruit du coeur :
     ↪  CALYPSO_BSP_SNR_DB=${CALYPSO_BSP_SNR_DB:-$BRUIT_SNR_DB} dB${BRUIT_GRAINE:+, graine $BRUIT_GRAINE}"
    return 0
}

bruit_env_pont() {   # dans le sous-shell du pont ; ne pose rien par defaut
    [ "$MODE" = grgsm ] && [ -n "$BRUIT_MODE" ] || return 0
    # pont/pont.py viserait de lui-meme BRUIT_PORT_GRGSM : on le dit explicitement, et
    # 4730 (QEMU) si l'injecteur ne tourne pas, pour ne pas envoyer dans le vide.
    if vivant bruit; then export PONT_GSMTAP_PORT="$BRUIT_PORT_GRGSM"; else export PONT_GSMTAP_PORT=4730; fi
    return 0
}

etapeb() {   # l'injecteur de bruit (BRUIT_MODE), entre le pont et la couche 1
    [ "$BRUIT_ACTIF" = 1 ] || return 0
    vivant bruit && { dire "b. injecteur de bruit deja lance (pid $(pid_de bruit))"; return; }
    bruit_args
    : > "$RUNDIR/bruit.log"
    python3 -u "$BRUIT_PY" "${BRUIT_ARGS[@]}" >> "$RUNDIR/bruit.log" 2>&1 &
    echo $! > "$RUNDIR/bruit.pid"
    attendre 5 grep -aq "PRET ecoute=" "$RUNDIR/bruit.log" || rater "l'injecteur de bruit ne s'est pas annonce
     ↪  (voir $RUNDIR/bruit.log)"
    dire "b. injecteur de bruit  pid $(pid_de bruit)  mode $BRUIT_MODE  $(sed -n 's/.*PRET \(ecoute=[^ ]*\)
     ↪  \(vers=[^ ]*\).*/\1 \2/p' "$RUNDIR/bruit.log" | head -1)  journal $RUNDIR/bruit.log"
}

bruit_arreter() {   # --stop : l'injecteur sans pid (lance a la main avec --etiquette c54x)
    pkill -f -- "injecteur_bruit\.py .*--etiquette c54x" 2>/dev/null
    return 0
}

bruit_statut() {   # --status : muet par defaut
    if vivant bruit; then
        printf '  %-8s pid %-7s  %s\n' bruit "$(pid_de bruit)" "$RUNDIR/bruit.log"
        grep -a "STATS\|BILAN" "$RUNDIR/bruit.log" 2>/dev/null | tail -1
    elif [ -n "$BRUIT_MODE" ]; then
        printf '  %-8s arrete\n' bruit
    fi
    return 0
}

mkdir -p "$RUNDIR"
case "${1:-}" in
    --stop)   arreter; garder_journaux
              # [2026-10-20] le telephone 2 du TRX (instance 3) tombe avec l'instance 1
              if [ "$INSTANCE" = 1 ] && [ -d "/tmp/c54x-pont$OSMO_TRX_TS1_INSTANCE" ] && [ -f
               ↪  "/tmp/c54x-pont$OSMO_TRX_TS1_INSTANCE/dsp.pid" ]; then
                  INSTANCE="$OSMO_TRX_TS1_INSTANCE" OSMO_TRX=1 OSMO_TRX_PHONE=2 AIR=0 bash "$0" --stop
              fi
              # [2026-10-20 22:30] le mobile par l'air (instance 2) aussi : il survivait aux relances de
              # start-native.sh --bts --ms (c54x_exe + air.py de 22:12 encore la a 22:28), et l'air gardait
              # ouvert l'ANCIEN anneau TX de l'instance 1 (fichier supprime) : A->B fige, plus de descendant
              # pour le mobile, donc plus de LU.
              if [ "$INSTANCE" = 1 ] && [ -d /tmp/c54x-pont2 ] && ls /tmp/c54x-pont2/*.pid >/dev/null 2>&1; then
                  INSTANCE=2 AIR=1 OSMO_TRX=0 bash "$0" --stop
              fi
              # vides une fois ranges : le lancement suivant ne les range pas deux fois
              [ -f "$RUNDIR/dsp.log" ] && : > "$RUNDIR/dsp.log"
              [ -f /dev/shm/pont.log ] && : > /dev/shm/pont.log ;;
    --status) statut ;;
    --charge) charge ;;
    --logs)   # [2026-10-19] en couleur a l'affichage (qosmo/tools/qosmo-couleurs.py, QOSMO_COULEURS=0 coupe)
              _coul="$QOSMO/tools/qosmo-couleurs.py"
```

```
            if [ "${QOSMO_COULEURS:-1}" != 0 ] && [ -f "$_coul" ] && [ -t 1 ]; then
                exec tail -n 5 -F "$RUNDIR"/dsp.log "$RUNDIR"/qemu.log "$RUNDIR"/osmocon.log
                 ↪  "$RUNDIR"/mobile.log "$RUNDIR"/transceiver.log "$RUNDIR"/pont.log "$RUNDIR"/air.log $([ -n
                 ↪  "$BRUIT_MODE" ] || [ -f "$RUNDIR/bruit.pid" ] && echo "$RUNDIR/bruit.log") 2>/dev/null |
                 ↪  python3 -u "$_coul" --force
            fi
            exec tail -n 5 -F "$RUNDIR"/dsp.log "$RUNDIR"/qemu.log "$RUNDIR"/osmocon.log "$RUNDIR"/mobile.log
             ↪  "$RUNDIR"/transceiver.log "$RUNDIR"/pont.log "$RUNDIR"/air.log $([ -n "$BRUIT_MODE" ] || [ -f
             ↪  "$RUNDIR/bruit.pid" ] && echo "$RUNDIR/bruit.log") 2>/dev/null ;;
    --step)   case "${2:-}" in 1) bruit_preparer; etape1;; b) bruit_preparer; etapeb;; 2) etape2;; 3) etape3;; 4)
     ↪  etape4;; 5) etape5;; a) etape5_air;; *) rater "--step 1|b|2|3|4|5|a";; esac ;;
    -h|--help) sed -n '2,22p' "$0" ;;
    "")       garder_journaux; bruit_preparer; etape1; etapeb; etape2; etape3; etape4; etape5
             dire "tout tourne. Journaux : $RUNDIR/*.log   suivre : ./run.sh --logs   arreter : ./run.sh --stop" ;;
    *)        rater "option inconnue : $1 (voir --help)" ;;
esac
```

3.9 /opt/GSM/c54x_exe/run_real.sh

3888 octets, 57 lignes → 57 lignes

```
#!/usr/bin/env bash
# run_real.sh - le banc REEL : BTS virtuelle (osmo-bts-trx + coeur osmocom) ->
# pont.py (TRX) -> DSP c54x_exe (--iq none, bursts UDP 6702) -> QEMU/ARM ->
# osmocon -> mobile. Memes executables que run.sh, sans aucune bequille
# (pas de CAN_TOA / CAN_SB / AFC forcee) : FB, SB et BCCH natifs.
#
#   ./run_real.sh              lance tout, observe 120 s, verdict SI / LAI
#   ./run_real.sh --secondes N duree d'observation
#   ./run_real.sh --logs       suivre les journaux      ./run_real.sh --stop  tout arreter
#
# Reglages (mesures 2026-09-21) : INSNS=60000 (cadence DSP proche du temps reel
# du BTS), CALYPSO_BSP_STREAM=1 (un burst TS0 par trame, ordre FN),
# CALYPSO_RHEA_DMA_XFER=1 CALYPSO_BSP_DIRECT_FEED=1 (livraison DMA/RIF),
# CALYPSO_BSP_NB_SYM=0.3 (elargissement des bursts normaux, calypso_bsp.c).
set -u
HERE="$(cd "$(dirname "$0")" && pwd)"
RUNDIR=/tmp/c54x-pont
SECS=120
case "${1:-}" in
  --stop)  "$HERE/run.sh" --stop; systemctl stop osmo-bts-trx 2>/dev/null; exit 0 ;;
  --logs)  exec "$HERE/run.sh" --logs ;;
  --secondes) SECS="$2" ;;
  -h|--help) sed -n '2,14p' "$0"; exit 0 ;;
esac
E_MCC="$(grep -m1 -oE 'network country code [0-9]+' /etc/osmocom/osmo-msc.cfg | grep -oE '[0-9]+$')"
E_MNC="$(grep -m1 -oE 'mobile network code [0-9]+'  /etc/osmocom/osmo-msc.cfg | grep -oE '[0-9]+$')"
E_LAC="$(grep -m1 -oE 'location_area_code 0x[0-9a-fA-F]+' /etc/osmocom/osmo-bsc.cfg | grep -oE '0x[0-9a-fA-F]+')"
E_LAC=$((E_LAC)); E_MCC=$((10#$E_MCC)); E_MNC=$((10#$E_MNC))
BSIC="$(grep -m1 -oE 'base_station_id_code [0-9]+' /etc/osmocom/osmo-bsc.cfg | grep -oE '[0-9]+$')"
echo "== run_real : LAI attendu MCC=$E_MCC MNC=$E_MNC LAC=$E_LAC (BSIC=${BSIC:-?}), aucune bequille"
"$HERE/run.sh" --stop >/dev/null 2>&1; sleep 1
for u in osmo-hlr osmo-stp osmo-msc osmo-mgw osmo-bsc; do systemctl is-active --quiet "$u" || systemctl start
 ↪  "$u"; done
systemctl restart osmo-bts-trx; sleep 3
mkdir -p "$RUNDIR"; : > "$RUNDIR/mobile.log"; : > "$RUNDIR/osmocon.log"; : > "$RUNDIR/pont.log"
export MODE=dsp PONT=1 IQ=none INSNS="${INSNS:-60000}" LOCKSTEP="${LOCKSTEP:-1}"
export CALYPSO_RHEA_DMA_XFER=1 CALYPSO_BSP_DIRECT_FEED=1 CALYPSO_BSP_STREAM=1
export PONT_NB_DEBUG="${PONT_NB_DEBUG:-1}"
"$HERE/run.sh" || { echo "== run.sh a echoue"; exit 1; }
echo "== observation ${SECS}s ..."
sleep "$SECS"
M="$RUNDIR/mobile.log"; O="$RUNDIR/osmocon.log"
```

```
echo "== SB decodees (osmocon) : $(sed 's/\x1b\[[0-9;]*m//g' "$O" | grep -ac '=> SB')   FBSB sans SB : $(sed
 ↪  's/\x1b\[[0-9;]*m//g' "$M" | grep -ac 'result=255')"
echo "== blocs BCCH jetes (fire) : $(sed 's/\x1b\[[0-9;]*m//g' "$M" | grep -ac 'Dropping frame')"
echo "== System Information vus :"
sed 's/\x1b\[[0-9;]*m//g' "$M" | grep -aoE "New SYSTEM INFORMATION [0-9a-z]+" | sort | uniq -c || echo "  (aucun)"
echo "== montant : RACH publies par le DSP : $(sed 's/\x1b\[[0-9;]*m//g' "$RUNDIR/dsp.log" | grep -ac '\[montant\]
 ↪  RACH')   compteurs du pont : $(grep -ao 'UL bursts=[0-9]* tard=[0-9]* rach=[0-9]*' "$RUNDIR/pont.log" | tail
 ↪  -1)"
echo "== IMM ASS : $(sed 's/\x1b\[[0-9;]*m//g' "$M" | grep -ac 'IMMEDIATE ASSIGNMENT')   LU ACCEPT : $(sed
 ↪  's/\x1b\[[0-9;]*m//g' "$M" | grep -ac 'LOCATION UPDATING ACCEPT')"
LAI="$(sed 's/\x1b\[[0-9;]*m//g' "$M" | grep -aoE "lai=[0-9]+-[0-9]+-[0-9]+" | tail -1)"
echo "== VERDICT :"
if [ -z "$LAI" ]; then echo "  ECHEC : aucun SI lu"; else
  D_MCC="$(echo "$LAI" | cut -d= -f2 | cut -d- -f1)"; D_MNC="$(echo "$LAI" | cut -d- -f2)"; D_LAC="$(echo "$LAI" |
   ↪  cut -d- -f3)"
  if [ "$((10#$D_MCC))" = "$E_MCC" ] && [ "$((10#$D_MNC))" = "$E_MNC" ] && [ "$D_LAC" = "$E_LAC" ]; then
    echo "  VRAI : $LAI = la config du reseau, le mobile a lu la BCCH du BTS"
  else echo "  FAUX POSITIF : $LAI ne correspond pas a MCC=$E_MCC MNC=$E_MNC LAC=$E_LAC"; fi
  sed 's/\x1b\[[0-9;]*m//g' "$M" | grep -aq "camping normally" && echo "  et il campe (MM IDLE)"
fi
echo "== arret : ./run_real.sh --stop"
```

3.10 /opt/GSM/c54x_exe/run_si.sh

4063 octets, 72 lignes → 72 lignes

```
#!/usr/bin/env bash
# run_si.sh — amener le mobile (pile DSP c54x_exe) jusqu'à décoder la BCCH du
# VRAI réseau et lire un System Information, PUIS vérifier que le LAI décodé
# correspond à la config du réseau. Sinon = FAUX POSITIF, dit tel quel.
#
# C'est un banc SOUS ÉCHAFAUDAGE (« canning »), PAS un fonctionnement natif :
#   - PONT_CAN_TOA=23      TOA FB figée (le corrélateur natif ne verrouille pas)
#   - PONT_CAN_SB=<bsic>   résultat SB fabriqué (BSIC + FN du BTS), comme qemu-src
#   - CALYPSO_TWL3025_AFC_HZ  rotation AFC forcée (annule l'offset de fréquence)
#   - CALYPSO_PONT_LOCKSTEP=1  trame QEMU cadencée sur le DSP
# Chaque hack actif est imprimé. Un SI lu avec des hacks n'est pas une preuve du
# DSP natif ; c'est un test d'intégration de l'aval (sync -> BCCH -> L2/L3).
#
#   ./run_si.sh              # chaîne réelle (BTS + pont), échafaudage complet
#   ./run_si.sh --secondes N # durée d'observation (défaut 90)
#   ./run_si.sh --stop       # tout arrêter
set -u
HERE="$(cd "$(dirname "$0")" && pwd)"
RUNDIR=/tmp/c54x-pont
SECS=90
BSIC="$(grep -m1 -oE 'base_station_id_code [0-9]+' /etc/osmocom/osmo-bsc.cfg 2>/dev/null | grep -oE '[0-9]+$')";
 ↪  BSIC="${BSIC:-7}"
# LAI attendu, lu depuis la config (MCC/MNC/LAC)
E_MCC="$(grep -m1 -oE 'network country code [0-9]+' /etc/osmocom/osmo-msc.cfg | grep -oE '[0-9]+$')"
E_MNC="$(grep -m1 -oE 'mobile network code [0-9]+'  /etc/osmocom/osmo-msc.cfg | grep -oE '[0-9]+$')"
E_LAC="$(grep -m1 -oE 'location_area_code 0x[0-9a-fA-F]+' /etc/osmocom/osmo-bsc.cfg | grep -oE '0x[0-9a-fA-F]+')"
E_LAC=$((E_LAC)); E_MCC=$((10#$E_MCC)); E_MNC=$((10#$E_MNC))

case "${1:-}" in
  --stop) "$HERE/run.sh" --stop; systemctl stop osmo-bts-trx 2>/dev/null; exit 0 ;;
  --secondes) SECS="$2" ;;
  -h|--help) sed -n '2,25p' "$0"; exit 0 ;;
esac

echo "== run_si : cible LAI attendu MCC=$E_MCC MNC=$E_MNC LAC=$E_LAC (BSIC=$BSIC, ARFCN 514)"
"$HERE/run.sh" --stop >/dev/null 2>&1; sleep 1
systemctl start osmo-bts-trx 2>/dev/null; sleep 3
: > "$RUNDIR/mobile.log" 2>/dev/null; : > "$RUNDIR/osmocon.log" 2>/dev/null
```

```
# chaîne réelle, NATIF. [2026-10-06] L'echafaudage de resultat (CAN_TOA/CAN_SB/AFC_HZ/RHEA_DMA) a ete
# RETIRE : le corrélateur FB, l'AFC et la DMA de la ROM font le travail. On ne garde que le cadencement
# et l'alimentation des bursts, qui ne fabriquent aucun resultat. (Historique des hacks : git.)
export MODE=dsp PONT=1 IQ=none
export CALYPSO_BSP_DIRECT_FEED=1 CALYPSO_PONT_LOCKSTEP=1   # cadencement / alimentation (pas des triches)
echo "== NATIF : aucun echafaudage (CAN_TOA/CAN_SB/AFC_HZ/RHEA_DMA retires) ; LOCKSTEP + DIRECT_FEED"
"$HERE/run.sh" >/dev/null 2>&1 &
sleep 12
grep -aq "code is running" "$RUNDIR/osmocon.log" && echo "== firmware lancé" || echo "== firmware PAS lancé (voir
 ↪  $RUNDIR/osmocon.log)"
echo "== observation ${SECS}s ..."
sleep "$SECS"

M="$RUNDIR/mobile.log"
echo "== System Information vus (BCCH) :"
sed 's/\x1b\[[0-9;]*m//g' "$M" | grep -aoE "New SYSTEM INFORMATION [0-9a-z]+" | sort | uniq -c || echo "  (aucun)"
LAI="$(sed 's/\x1b\[[0-9;]*m//g' "$M" | grep -aoE "lai=[0-9]+-[0-9]+-[0-9]+" | tail -1)"
echo "== [can-sb] premières lignes :"; grep -a "\[can-sb\]" "$RUNDIR/dsp.log" 2>/dev/null | head -3 | cut -c1-120

echo "== VERDICT :"
if [ -z "$LAI" ]; then
  echo "  ÉCHEC : aucun SI lu -> pas de camp. (attendu tant que le démod NB/BCCH natif n'est pas bon)"
else
  D_MCC="$(echo "$LAI" | cut -d= -f2 | cut -d- -f1)"
  D_MNC="$(echo "$LAI" | cut -d- -f2)"
  D_LAC="$(echo "$LAI" | cut -d- -f3)"
  echo "  LAI décodé : MCC=$D_MCC MNC=$D_MNC LAC=$D_LAC"
  if [ "$D_MCC" = "$E_MCC" ] && [ "$D_MNC" = "$E_MNC" ] && [ "$D_LAC" = "$E_LAC" ]; then
    echo "  ✓ VRAI : le LAI correspond à la config -> le mobile a réellement lu la BCCH du réseau."
  else
    echo "  ✗ FAUX POSITIF : le LAI ne correspond pas (config MCC=$E_MCC MNC=$E_MNC LAC=$E_LAC)."
    echo "    Un bloc décodé qui ne porte pas l'identité du réseau n'est pas un camp, c'est du bruit qui a passé
     ↪  un CRC."
  fi
fi
echo "== arrêt : ./run_si.sh --stop"
```

3.11 /opt/GSM/c54x_exe/sign.sh

2268 octets, 52 lignes → 52 lignes

```
#!/usr/bin/env bash
# sign.sh - empreinte SHA-256 et signature GPG detachee d'un fichier (par defaut proof_toa_23.md).
#
#   ./sign.sh                      signe proof_toa_23.md
#   ./sign.sh FICHIER              signe FICHIER
#   ./sign.sh --verify [FICHIER]   verifie l'empreinte et la signature
#
# Cle : si aucune cle secrete GPG ne correspond a SIGN_UID, une cle ed25519 (signature seule, sans
# expiration) est generee. Variables : SIGN_UID (identite), SIGN_PASSPHRASE (vide par defaut),
# GNUPGHOME (trousseau, defaut ~/.gnupg). Produits, a cote du fichier :
#   FICHIER.sha256     empreinte (format sha256sum)
#   FICHIER.asc        signature detachee ASCII
#   FICHIER.pubkey.asc cle publique a diffuser avec le document
set -euo pipefail
HERE="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
SIGN_UID="${SIGN_UID:-proof_toa_23 <bastienbaranoff@gmail.com>}"
SIGN_PASSPHRASE="${SIGN_PASSPHRASE:-}"
export GNUPGHOME="${GNUPGHOME:-$HOME/.gnupg}"
mkdir -p "$GNUPGHOME"; chmod 700 "$GNUPGHOME"
GPG=(gpg --batch --yes --pinentry-mode loopback --passphrase "$SIGN_PASSPHRASE")

verify=0
```

```
[ "${1:-}" = "--verify" ] && { verify=1; shift; }
F="${1:-$HERE/proof_toa_23.md}"
[ -r "$F" ] || { echo "fichier illisible : $F" >&2; exit 2; }
D="$(cd "$(dirname "$F")" && pwd)"; B="$(basename "$F")"

if [ "$verify" = 1 ]; then
    (cd "$D" && sha256sum -c "$B.sha256")
    [ -r "$F.pubkey.asc" ] && gpg --batch --quiet --import "$F.pubkey.asc" 2>/dev/null || true
    gpg --batch --verify "$F.asc" "$F"
    exit $?
fi

# 1. empreinte
(cd "$D" && sha256sum "$B" > "$B.sha256")
cat "$F.sha256"

# 2. cle (generee une seule fois)
if ! gpg --batch --list-secret-keys "=$SIGN_UID" >/dev/null 2>&1; then
    echo "aucune cle secrete pour « $SIGN_UID » : generation d'une cle ed25519 (signature, sans expiration)"
    "${GPG[@]}" --quick-gen-key "$SIGN_UID" ed25519 sign 0
fi
FPR="$(gpg --batch --with-colons --list-secret-keys "=$SIGN_UID" | awk -F: '$1=="fpr"{print $10; exit}')"
echo "cle : $FPR  ($SIGN_UID)"

# 3. signature detachee + cle publique
"${GPG[@]}" --armor --detach-sign --local-user "$FPR" --output "$F.asc" "$F"
gpg --batch --armor --export "$FPR" > "$F.pubkey.asc"
echo "signature : $F.asc"
echo "cle publique : $F.pubkey.asc"
echo "verification : $0 --verify $F"
```

3.12 /opt/GSM/c54x_exe/tout-en-un.py

20083 octets, 397 lignes → 397 lignes

```
#!/usr/bin/env python3
"""tout-en-un.py — rassemble tous les fichiers texte (sources, scripts, configs,
docs, logs) d'un ou plusieurs dossiers dans UN fichier Markdown, raccourci SANS PERTE :

  * fichiers identiques (meme contenu) : une seule copie, les autres renvoient
    a la premiere ;
  * lignes consecutives identiques : « ligne  ×N » ;
  * lignes consecutives qui ne different que par des nombres (horodatage,
    compteur, fn...) : un gabarit ou les nombres constants restent en place et
    les nombres variables deviennent ⟨1⟩ ⟨2⟩..., suivi de la liste ordonnee des
    valeurs. On reconstruit chaque ligne en remettant les valeurs dans l'ordre.
  * codes couleur ANSI et retours chariot retires (seule perte, volontaire).

    ./tout-en-un.py [dossier...]      defaut : /opt/GSM/c54x_exe et le dernier /root/c54x_exe-*
    SORTIE=/chemin.md  EXT="sh py c"  (restreint aux extensions ; defaut : tout fichier texte)
    SEUIL=3 (taille mini d'un groupe)

  Sont ecartes : .git, dossiers caches et caches, binaires (ELF, images, .pyc, archives),
  sauvegardes (.bak*, ~, .orig, .avant-*), build-tic54x et binutils-*, desassemblage (.dis)
  et dumps (-dump.txt) de la ROM, signatures (.asc, .sha256), LICENSE/COPYING et les
  sorties precedentes de ce script.
"""
import glob
import hashlib
import os
import re
import sys
import time
```

```
EXT = os.environ.get("EXT", "").split()             # vide : tout fichier texte
SEUIL = int(os.environ.get("SEUIL", "3"))
FORMAT = os.environ.get("FORMAT", "qmd")           # qmd (Quarto) ou md
SORTIE = os.environ.get("SORTIE") or "/root/c54x_exe-%s.%s" % (time.strftime("%Y%m%d-%H%M%S"), FORMAT)
AUTEUR = os.environ.get("AUTEUR", "Banc GSM émulé — banc-max")
IGNORES = {".git", "__pycache__", ".pytest_cache", ".mypy_cache", ".ruff_cache", "node_modules", ".venv", "venv",
          "rom", "roms",                            # dumps de ROM (DSP Calypso) : pas a nous, pas dans le dossier
           "osmo-operator-qmd",                        # compilation du depot par un autre outil : tout en double
           "build-tic54x"}                             # arbre de construction binutils (config.log, Makefile...)
IGNORES_PREFIXES = (".", "binutils-")                  # dossiers caches (.avant-isa-*...), sources binutils
# Fichiers texte inutiles : desassemblage et dumps de la ROM DSP, signatures/empreintes, sauvegardes .avant-*
INUTILES = re.compile(r"(\.dis|-dump\.txt|\.asc|\.sha256|\.avant-[\w-]+)$")
BINAIRES = set("png jpg jpeg gif bmp ico webp pdf zip gz tgz bz2 xz 7z tar bin elf o a so pyc pyo pyd "
               "wav mp3 ogg mp4 sqlite db img iso woff woff2 ttf otf".split())
SAUVEGARDES = re.compile(r"(~|\.orig|\.rej|\.swp|\.bak(-\w+)?)$")
TITRE_SORTIE = "title: \"Banc GSM émulé — dossier de run\"".encode()   # une sortie precedente de ce script
ANSI = re.compile(r"\x1b\[[0-9;]*[A-Za-z]")
TELNET = re.compile(rb"\xff[\xfb-\xfe].|\xff.")           # negociation telnet (IAC) des captures VTY
CONTROLE = re.compile("[\x00-\x08\x0b\x0c\x0e-\x1f\x7f\ufffd]")  # caracteres de commande, octets indecodables
NUM = re.compile(r"\d+")
# Preambule LaTeX du format pdf (xelatex). Sans lui : « Dimension too large » (framed
# mesure tout le bloc de code avant de le couper), lignes de code non repliees,
# diagrammes Mermaid plus larges que la page, glyphes manquants.
PDF_PREAMBULE = "".join("        %s\n" % l for l in r"""
% Code : police reduite, retour a la ligne (y compris dans les mots longs)
\usepackage{fvextra}
\fvset{fontsize=\small,breaklines,breakanywhere}
\RecustomVerbatimEnvironment{verbatim}{Verbatim}{fontsize=\small,breaklines,breakanywhere}
% Images (diagrammes Mermaid) : jamais plus larges que la ligne ni plus hautes que la page
\usepackage{adjustbox}
\usepackage{letltxmacro}
\LetLtxMacro\ORIGincludegraphics\includegraphics
\renewcommand{\includegraphics}[2][]{\adjustimage{#1,max width=\linewidth,max totalheight=.85\textheight}{#2}}
% Sans cadre : framed/snugshade mesure tout le bloc et plante ("Dimension too large")
\renewenvironment{Shaded}{\medskip}{\medskip}
% Glyphes absents de Latin Modern Roman -> DejaVu Sans
\usepackage{newunicodechar}
\newfontfamily\fallbackfont{DejaVu Sans}
\newunicodechar{↔}{{\fallbackfont ↔}}
\newunicodechar{⟨}{{\fallbackfont ⟨}}
\newunicodechar{⟩}{{\fallbackfont ⟩}}
\newunicodechar{≠}{{\fallbackfont ≠}}
\newunicodechar{∈}{{\fallbackfont ∈}}
\newunicodechar{ᵉ}{{\fallbackfont ᵉ}}
\newunicodechar{✓}{{\fallbackfont ✓}}
\newunicodechar{✗}{{\fallbackfont ✗}}
\newunicodechar{⚠}{{\fallbackfont ⚠}}
\newunicodechar{☎}{{\fallbackfont ☎}}
\newunicodechar{↩}{{\fallbackfont ↩}}
""".strip("\n").split("\n"))

LANG = {"sh": "bash", "bash": "bash", "py": "python", "md": "markdown", "qmd": "markdown", "mmd": "mermaid",
        "txt": "text", "log": "text", "c": "c", "h": "c", "cpp": "cpp", "cc": "cpp", "hpp": "cpp",
        "ini": "ini", "env": "ini", "service": "ini", "desktop": "ini", "toml": "toml", "yml": "yaml",
        "yaml": "yaml", "json": "json", "patch": "diff", "diff": "diff", "mak": "makefile", "mk": "makefile",
        "html": "html", "xml": "xml", "svg": "xml", "js": "javascript", "css": "css", "sql": "sql", "rs": "rust"}
NOMS_LANG = {"Makefile": "makefile", "GNUmakefile": "makefile", "Dockerfile": "dockerfile"}


def extension(nom):
    return nom.rsplit(".", 1)[-1].lower() if "." in nom[1:] else ""


def langage(chemin):
    """Langage de coloration : extension, puis nom du fichier, puis shebang."""
    nom = os.path.basename(chemin)
```

```
    e = extension(nom)
    if e in LANG:
        return LANG[e]
    if nom in NOMS_LANG:
        return NOMS_LANG[nom]
    try:
        premiere = open(chemin, "rb").readline().decode("utf-8", "replace")
    except OSError:
        premiere = ""
    if premiere.startswith("#!"):
        if "python" in premiere:
            return "python"
        if re.search(r"\b(ba|z|da|k)?sh\b", premiere):
            return "bash"
    return "text"


def texte_legitime(chemin, nom):
    """Fichier texte a nous : ni binaire, ni sauvegarde, ni licence, ni sortie de ce script."""
    if extension(nom) in BINAIRES or SAUVEGARDES.search(nom) or INUTILES.search(nom) or nom in ("LICENSE",
     ↪  "COPYING"):
        return False
    if nom.startswith("tout-en-un-"):
        return False
    try:
        with open(chemin, "rb") as f:
            debut = f.read(8192)
    except OSError:
        return False
    if b"\0" in debut:                          # ELF, images, .pyc, archives...
        return False
    if TITRE_SORTIE in debut:
        return False
    return True


def dossiers_par_defaut():
    runs = sorted(glob.glob("/root/c54x_exe-*"), key=os.path.getmtime)
    runs = [r for r in runs if os.path.isdir(r)]
    return ["/opt/GSM/c54x_exe"] + ([runs[-1]] if runs else [])


def fichiers(dossier):
    for racine, dirs, noms in os.walk(dossier):
        dirs[:] = sorted(d for d in dirs if d not in IGNORES and not d.startswith(IGNORES_PREFIXES))
        for n in sorted(noms):
            chemin = os.path.join(racine, n)
            if EXT and extension(n) not in EXT:
                continue
            if texte_legitime(chemin, n):
                yield chemin


LIGNE_MAX = int(os.environ.get("LIGNE_MAX", "1000"))


def couper(l):
    if len(l) <= LIGNE_MAX:
        return [l]
    return [l[i:i + LIGNE_MAX] + ("↩" if i + LIGNE_MAX < len(l) else "") for i in range(0, len(l), LIGNE_MAX)]


def compacter(lignes):
    """Rend (lignes de sortie, nb de groupes compactes)."""
    out, groupes, i, n = [], 0, 0, len(lignes)
    while i < n:
```

```
        cle = NUM.sub("\0", lignes[i])
        j = i + 1
        while j < n and NUM.sub("\0", lignes[j]) == cle:
            j += 1
        taille = j - i
        if taille < SEUIL:
            out.extend(lignes[i:j])
            i = j
            continue
        groupes += 1
        if "\0" not in cle:                      # repetition exacte
            out.append("%s  ×%d" % (lignes[i], taille))
            i = j
            continue
        valeurs = [NUM.findall(l) for l in lignes[i:j]]
        nb = len(valeurs[0])
        varie = [len({v[k] for v in valeurs}) > 1 for k in range(nb)]
        # gabarit : nombres constants remis en place, variables numerotes ⟨k⟩
        morceaux, k, idx = cle.split("\0"), 0, 0
        gab = morceaux[0]
        for m in morceaux[1:]:
            if varie[k]:
                idx += 1
                gab += "⟨%d⟩" % idx
            else:
                gab += valeurs[0][k]
            gab += m
            k += 1
        out.append("%s  ×%d" % (gab, taille))
        if idx:
            tuples = [",".join(v[k] for k in range(nb) if varie[k]) for v in valeurs]
            # lignes de valeurs de ~100 colonnes
            ligne, courant = [], "    ⟨⟩ ="
            for t in tuples:
                if len(courant) + len(t) + 3 > 110:
                    ligne.append(courant)
                    courant = "       "
                courant += " (" + t + ")"
            ligne.append(courant)
            out.extend(ligne)
        i = j
    return out, groupes


def synthese(tous):
    """Run, echelle, echecs, bilan de couverture : lus dans verdict.txt / couverture.txt."""
    lignes, vus = [], set()
    for f in tous:
        nom = os.path.basename(f)
        if nom not in ("verdict.txt", "couverture.txt"):
            continue
        run = os.path.basename(os.path.dirname(f))
        if (run, nom) in vus:                # meme run copie a deux endroits
            continue
        vus.add((run, nom))
        try:
            txt = ANSI.sub("", open(f, "rb").read().decode("utf-8", "replace"))
        except OSError:
            continue
        for l in txt.splitlines():
            l = re.sub(r"\s+", " ", l.strip())
            if l.startswith("ELEMENT MAX"):
                lignes.append((0, "| %s | échelle | %s |" % (run, l)))
            elif l.startswith("bilan :"):
                lignes.append((2, "| %s | couverture | %s |" % (run, l[8:])))
            elif l.startswith("couverture couche 1"):
```

```
                lignes.append((1, "| %s | mode, date | %s |" % (run, l.split("—", 1)[-1].strip())))
            else:
                m = re.match(r"(\d+) (\S+) (ECHEC|SAUTE) ?(.*)", l)
                if m:
                    lignes.append((3, "| %s | barreau %s %s | %s %s |" % (run, m.group(1), m.group(2), m.group(3),
↪   m.group(4))))
    return [l for _, l in sorted(lignes, key=lambda x: x[0])]


MERMAID_OUVRE = "```{mermaid}" if FORMAT == "qmd" else "```mermaid"


def rendre_mmd(lignes):
    return "%s\n%s\n```" % (MERMAID_OUVRE, "\n".join(lignes))


def rendre_md(lignes):
    out, dans_bloc, cloture = [], False, ""
    i = 0
    if lignes and lignes[0].strip() == "---":           # front matter YAML : montre en bloc, pas interprete
        j = next((k for k in range(1, len(lignes)) if lignes[k].strip() in ("---", "...")), None)
        if j:
            out.append("```yaml"); out.extend(lignes[1:j]); out.append("```")
            i = j + 1
    while i < len(lignes):
        l = lignes[i]; i += 1
        m = re.match(r"^(\s*)(`{3,}|~{3,})(.*)$", l)
        if m and not dans_bloc:
            dans_bloc, cloture = True, m.group(2)
            info = m.group(3).strip()
            if info.startswith("{mermaid}") or info.startswith("mermaid"):
                l = m.group(1) + MERMAID_OUVRE
            elif info.startswith("{"):                    # ```{r ...} ou {.bash filename=...} : langage seul,
                l = m.group(1) + m.group(2) + info.strip("{}").split()[0].split(",")[0].lstrip(".")  # ni
↪   execution ni flottant
            out.append(l); continue
        if m and dans_bloc and m.group(2)[0] == cloture[0] and len(m.group(2)) >= len(cloture) and not
         ↪  m.group(3).strip():
            dans_bloc = False; out.append(l); continue
        if not dans_bloc:
            h = re.match(r"^(#{1,6})\s", l)
            if h:
                l = "#" * min(6, len(h.group(1)) + 3) + l[len(h.group(1)):]
            elif l.strip() in ("---", "..."):            # pas de bloc YAML ni de regle au milieu du dossier
                l = "* * *"
        out.append(l)
    if dans_bloc:
        out.append(cloture)
    return "\n".join(out)


def main():
    dossiers = sys.argv[1:] or dossiers_par_defaut()
    vus, sections, table = {}, [], []
    total_in = total_out = 0
    # Ordre : les resultats des tests (.txt .md .mmd : verdicts, couverture, rapports, grafcets)
    # d'abord, les fichiers du banc (sources, scripts, configs...) ensuite, les .log en dernier
    CATEGORIES = (("Verdict et couverture", ()), ("Resultats des tests", ("txt", "md", "mmd", "tsv", "csv")),
                  ("Fichiers", None), ("Logs", ("log",)))          # Fichiers : tout le reste
    EN_TETE = ("verdict.txt", "couverture.txt")     # tout en haut, dans cet ordre
    tous = []
    for d in dossiers:
        if not os.path.isdir(d):
            table.append("| (absent) %s | | | | |" % d)
            continue
        tous.extend(fichiers(d))
```

```
def rang(f):
    if os.path.basename(f) in EN_TETE:
        return 0
    e = extension(os.path.basename(f))
    for i, (_, exts) in enumerate(CATEGORIES):
        if exts and e in exts:
            return i
    return 2                                    # Fichiers
categorie_courante = None
def cle(f):
    r = rang(f)
    return (r, EN_TETE.index(os.path.basename(f)) if r == 0 else 0, tous.index(f))
for f in sorted(tous, key=cle):
    if rang(f) != categorie_courante:
        categorie_courante = rang(f)
        nom = CATEGORIES[categorie_courante][0] if categorie_courante < len(CATEGORIES) else "Autres"
        sections.append("## %s\n" % nom)
    if True:
        try:
            brut = open(f, "rb").read()
        except OSError as e:
            table.append("| %s | | | | illisible : %s |" % (f, e))
            continue
        texte = ANSI.sub("", TELNET.sub(b"", brut).decode("utf-8", "replace")).replace("\r", "")
        texte = CONTROLE.sub("", texte)
        lignes = texte.split("\n")
        if lignes and lignes[-1] == "":
            lignes.pop()
        # lignes demesurees (blob base64, minifie...) : coupees en morceaux de LIGNE_MAX
        # termines par ↩ ; sinon xelatex mesure la ligne entiere et plante (Dimension too large)
        lignes = [m for l in lignes for m in couper(l)]
        h = hashlib.sha1(texte.encode()).hexdigest()
        total_in += len(brut)
        if h in vus:
            sections.append("### %s\n\nidentique à %s\n" % (f, vus[h]))
            continue
        vus[h] = f
        ext = extension(os.path.basename(f))
        if ext in ("md", "qmd", "mmd"):
            # Rendu, pas cite : les .mmd deviennent des diagrammes Mermaid, les .md
            # sont inclus tels quels (titres retrogrades sous le titre du fichier,
            # blocs mermaid rendus, blocs {r}/{python} neutralises). Pas de compactage.
            total_out += len(texte) + 1
            sections.append("### %s\n\n%d octets, %d lignes\n\n%s\n"
                            % (f, len(brut), len(lignes), rendre_md(lignes) if ext in ("md", "qmd") else
                             ↪  rendre_mmd(lignes)))
            continue
        compact, groupes = compacter(lignes)
        total_out += sum(len(l) + 1 for l in compact)
        note = "%d groupes compactés" % groupes if groupes else ""
        # cloture plus longue que toute suite de ` du contenu : un ``` dans un
        # .py ou un .md ne referme plus le bloc (pandoc lisait alors la suite
        # comme du Markdown : « Could not fetch resource »)
        plus_long = max((len(m) for m in re.findall(r"`+", "\n".join(compact))), default=0)
        cloture = "`" * max(3, plus_long + 1)
        sections.append("### %s\n\n%d octets, %d lignes → %d lignes%s\n\n%s%s\n%s\n%s\n"
                        % (f, len(brut), len(lignes), len(compact), (" (%s)" % note) if note else "",
                           cloture, langage(f), "\n".join(compact), cloture))
runs = sorted({os.path.basename(os.path.dirname(f)) for f in tous if os.path.basename(f) == "verdict.txt"})
with open(SORTIE, "w") as o:
    o.write("---\n")
    o.write("title: \"Banc GSM émulé — dossier de run\"\n")
    o.write("subtitle: \"%s\"\n" % (", ".join(runs) if runs else ", ".join(dossiers)))
    o.write("author: \"%s\"\n" % AUTEUR)
    o.write("date: \"%s\"\n" % time.strftime("%Y-%m-%d %H:%M"))
    o.write("lang: fr\n")
```

```
        if FORMAT == "qmd":
            o.write("engine: markdown\n")
            o.write("code-annotations: false\n")  # un "<100>" dans un commentaire n'est pas une annotation     #
↪  pandoc seul : knitr voyait des ```{r} dans les rapports inclus et s'arretait
        if FORMAT == "qmd":
            o.write("format:\n  html:\n    toc: true\n    toc-depth: 3\n    toc-location: left\n"
                    "    number-sections: true\n    embed-resources: true\n    theme: cosmo\n"
                    "    code-overflow: wrap\n    fontsize: 0.9em\n")
            o.write("  pdf:\n    toc: true\n    toc-depth: 3\n    number-sections: true\n"
                    "    shift-heading-level-by: -1\n"      # ## Synthese = section 1, pas 0.1
                    "    papersize: a4\n    geometry: margin=2cm\n    fontsize: 10pt\n"
                   "    monofont: DejaVu Sans Mono\n"      # trace de boites, ✓ ✗ ⟨⟩ ⚠ absents de Latin Modern Mono
                    "    include-in-header:\n      text: |\n" + PDF_PREAMBULE)
        o.write("---\n\n")
        o.write("## Synthèse\n\n")
        o.write("| run | élément | valeur |\n|---|---|---|\n")
        lignes_syn = synthese(tous)
        o.write("\n".join(lignes_syn) + "\n\n" if lignes_syn else "| (pas de verdict.txt) | | |\n\n")
        o.write("| dossier de run | contenu | |\n|---|---|---|\n")
        o.write("| Verdict et couverture | échelle des barreaux et tableau de couverture couche 1 | |\n")
        o.write("| Résultats des tests | captures VTY, diagnostics, rapports pytest, grafcets (.txt .md .mmd) |
↪  |\n")
        o.write("| Fichiers | sources, scripts, configs et docs du banc (tout fichier texte) | |\n")
        o.write("| Logs | journaux (.log), en dernier | |\n\n")
        o.write("::: {.callout-note collapse=\"true\"}\n## Méthode\n\n" if FORMAT == "qmd" else "### Méthode\n\n")
        o.write("Dossiers : %s. Extensions : %s. Entrée %d Ko, sortie %d Ko.\n\n"
                % (", ".join(dossiers), " ".join(EXT) if EXT else "tous les fichiers texte (hors .git, caches, "
                   "binaires, sauvegardes, LICENSE et sorties précédentes)", total_in // 1024, total_out // 1024))
        o.write("Compactage sans perte : fichiers identiques cités une fois ; lignes consécutives identiques "
                "« ×N » ; lignes ne différant que par des nombres : gabarit avec ⟨k⟩ puis la liste ordonnée "
                "des valeurs (k-uplets), chaque ligne se reconstruit en remettant les valeurs dans l'ordre. "
                "Seuls les codes couleur, la négociation telnet des captures VTY et les caractères de commande "
                "sont retirés ; les lignes de plus de %d caractères sont coupées en morceaux terminés par ↩. Les
                 ↪  .md sont inclus tels quels (titres rétrogradés) et les .mmd rendus en "
                "diagrammes Mermaid, sans compactage.\n" % LIGNE_MAX)
        o.write(":::\n\n" if FORMAT == "qmd" else "\n")
        for t in table:                       # ne reste que les dossiers absents / fichiers illisibles
            o.write(t.strip("| ").split(" |")[0] + "\n")
        o.write("\n")
        o.write("\n".join(sections))
    print("%s : %d fichiers (%d uniques), %d Ko -> %d Ko" % (SORTIE, len(tous), len(vus), total_in // 1024,
                                                            os.path.getsize(SORTIE) // 1024))


if __name__ == "__main__":
    main()
```

3.13 /opt/GSM/c54x_exe/src/carte_tx.h

2195 octets, 36 lignes → 36 lignes

```
/* genere par tests/rom-tx/carte.py : bit de sortie p du tampon ROM data[0x4280] (mot p/16, bit 15-p%16)
 * -> indice i dans les 456 bits de gsm0503_xcch_encode ; -1 = inconnu. Mesure du 2026-10-03, apres correction du
↪  coeur. */
static const short g_carte_tx[512] = {
  -1,1,2,3,4,5,6,-1,8,9,10,11,-1,13,14,15,
  16,17,18,19,20,-1,22,23,24,25,-1,27,28,29,30,31,
  32,33,34,-1,36,37,38,39,-1,41,42,43,44,45,46,47,
  48,-1,50,51,52,53,54,55,56,59,60,61,62,63,64,-1,
  66,-1,68,69,70,71,-1,73,74,75,76,77,78,-1,80,81,
  82,83,84,85,-1,87,88,89,90,91,92,-1,94,95,96,97,
  98,99,-1,101,102,103,104,105,106,107,108,109,110,111,112,113,
  -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
  116,117,118,119,120,-1,122,123,124,125,-1,127,128,129,130,131,
```

```
  132,133,134,-1,136,137,138,139,-1,141,142,143,144,145,146,147,
  148,-1,150,151,152,153,154,155,156,157,158,159,160,161,162,-1,
  164,-1,166,167,168,169,-1,171,172,175,176,177,178,-1,180,181,
  182,183,184,185,-1,187,188,189,190,191,192,-1,194,195,196,197,
  198,199,-1,201,202,203,204,205,206,207,208,209,210,211,212,213,
  -1,215,-1,217,218,219,220,221,222,-1,224,225,226,227,-1,229,
  230,231,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
  232,233,234,-1,236,237,238,239,-1,241,242,243,244,245,246,247,
  248,-1,250,251,252,253,254,255,256,257,258,259,260,261,262,-1,
  264,-1,266,267,268,269,-1,271,272,273,274,275,276,-1,278,279,
  280,281,282,283,-1,285,286,287,288,291,292,-1,294,295,296,297,
  298,299,-1,301,302,303,304,305,306,307,308,309,310,311,312,313,
  -1,315,316,317,318,319,320,321,322,-1,324,325,326,327,-1,329,
  -1,331,332,333,334,335,336,-1,338,339,340,341,-1,343,344,345,
  346,347,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
  348,-1,350,351,352,353,354,355,356,357,358,359,360,361,362,-1,
  364,-1,366,367,368,369,-1,371,372,373,374,375,376,-1,378,379,
  380,381,382,383,-1,385,386,387,388,389,390,-1,392,393,394,395,
  396,397,-1,399,400,401,402,403,404,407,408,409,410,411,412,413,
  -1,415,416,417,418,419,420,421,422,-1,424,425,426,427,-1,429,
  -1,431,432,433,434,435,436,-1,438,439,440,441,-1,443,444,445,
  446,447,448,449,450,-1,452,453,454,455,-1,-1,-1,-1,-1,-1,
  -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,
};
```

3.14 /opt/GSM/c54x_exe/src/cellule.c

38011 octets, 698 lignes → 700 lignes (2 groupes compactés)

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * cellule.c - cell synchronisation bursts for the DSP.
 *
 * FCCH = 148 zero bits (45.002 5.2.4). SCH = 3 tail + 39 + 64 extended training
 * sequence (45.002 5.2.5) + 39 + 3 tail; the 78 coded bits come from
 * gsm0503_sch_encode() of libosmocoding fed with sb_info (BSIC + T1'/T2/T3',
 * 44.018 9.1.30). Dummy burst: the fixed pattern of 45.002 5.2.6. Burst
 * assembly follows osmo-bts sched_lchan_fcch_sch.c and scheduler.c.
 */
#include <string.h>
#include <math.h>
#include "hw/arm/calypso/calypso_debug.h"
#include <stdlib.h>
#include <osmocom/core/bits.h>
#include <osmocom/coding/gsm0503_coding.h>
#include "calypso_gmsk.h"
#include "cellule.h"

static const uint8_t train_sb[64] = {
    1,0,1,1,1,0,0,1,0,1,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,
    0,0,1,0,1,1,0,1,0,1,0,0,0,1,0,1,0,1,1,1,0,1,1,0,0,0,0,1,1,0,1,1,
};
static const uint8_t factice[148] = {
    0,0,0,
    ⟨1⟩,⟨2⟩,⟨3⟩,⟨4⟩,⟨5⟩,⟨6⟩,⟨7⟩,⟨8⟩,0,⟨9⟩,⟨10⟩,1,⟨11⟩,⟨12⟩,⟨13⟩,⟨14⟩,⟨15⟩,⟨16⟩,⟨17⟩,⟨18⟩,⟨19⟩,⟨20⟩,⟨21⟩,⟨22⟩,⟨23⟩,⟨24⟩,⟨25⟩,⟨26⟩,⟨
 ↪  27⟩,⟨28⟩,⟨29⟩,⟨30⟩,
 ↪  ×4
    ⟨⟩ = (1,1,1,1,1,0,1,1,1,1,0,1,1,0,0,0,0,0,1,0,1,0,0,1,0,0,1,1,1,0)
         (0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,1,1,1,0,0)
         (0,1,0,1,1,1,0,0,1,0,1,1,0,0,0,1,0,1,0,1,1,1,0,1,0,0,1,0,1,0)
         (0,0,1,1,0,0,1,1,0,1,1,0,0,1,1,1,1,1,0,1,0,0,1,1,1,1,1,0,0,0,1)
    0,0,1,0,1,1,1,1,1,0,1,0,1,0,
    0,0,0,
};
```

```
int cellule_sch_partout;      /* diagnostic: emit SCH on every non-FCCH frame */

/* ---- BCCH / CCCH : normal bursts of the synthetic cell [2026-09-20] ----
 *
 * Downlink TN0 of a combined CCCH+SDCCH/4 cell (44.018 6.3.1.3, 45.002 clause
 * 7 table 3): BCCH norm on p51 2..5, CCCH on 6..9, 12..15, 16..19, SDCCH/4
 * elsewhere (left as dummy bursts). The BCCH block of 51-multiframe TC =
 * (fn / 51) % 8 carries SI1 (TC 0, 4), SI2 (1, 5), SI3 (2, 6), SI4 (3, 7);
 * the CCCH blocks carry an empty PAGING REQUEST TYPE 1. Coding is the one of
 * osmo-bts sched_lchan_xcch.c: gsm0503_xcch_encode() (Fire CRC, r=1/2
 * convolutional code, 4-burst diagonal interleaving) then 3 tail, 57 data,
 * hl, 26-bit training sequence BCC of the BSIC (45.002 5.2.3 set 1), hu, 57
 * data, 3 tail. The ROM demodulates with the TSC the ARM hands it in
 * dsp_load_rx_task(ALLC_DSP_TASK, burst_id, tsc), decodes the four bursts and
 * leaves 23 octets in a_cd[3..] with the Fire result in a_cd[0]; prim_rx_nb.c
 * copies them into an L1CTL_DATA_IND and mobile's rr reads the SI.
 *
 * Identity: MCC 001 MNC 01 LAC 1 CI 6001, ARFCN 514, the bench network of
 * /etc/osmocom (run_si.sh checks the decoded LAI against it). Override with
 * CELLULE_MCC / CELLULE_MNC / CELLULE_LAC / CELLULE_CI / CELLULE_ARFCN. */
#include <arpa/inet.h>
#include <osmocom/gsm/gsm48.h>
#include <osmocom/gsm/gsm48_ie.h>
#include <osmocom/gsm/gsm23003.h>
#include <osmocom/gsm/protocol/gsm_04_08.h>
#include <osmocom/gsm/sysinfo.h>

static const uint8_t train_nb[8][26] = {   /* 45.002 table 5.2.3a */
    { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0,16,17,18,19,20,21,22,23,24,25 },  ×8
    () = (0,0,1,0,0,1,0,1,1,1,0,0,0,1,0,0,1,0,0,1,0,1,1,1)
         (0,0,1,0,1,1,0,1,1,1,0,1,1,1,0,0,1,0,1,1,0,1,1,1)
         (0,1,0,0,0,0,1,1,1,0,1,1,1,0,1,0,1,0,0,0,0,1,1,1,0)
         (0,1,0,0,0,1,1,1,1,0,1,1,0,1,0,0,1,0,0,0,1,1,1,1,0)
         (0,0,0,1,1,0,1,0,1,1,1,0,0,1,0,0,0,0,1,1,0,1,0,1,1)
         (0,1,0,0,1,1,1,0,1,0,1,1,0,0,0,0,1,0,0,1,1,1,0,1,0)
         (1,0,1,0,0,1,1,1,1,1,0,1,1,0,0,1,0,1,0,0,1,1,1,1,1)
         (1,1,1,0,1,1,1,1,1,0,0,0,1,0,0,1,1,1,1,0,1,1,1,1,0,0)
};

int cellule_marge_nb = -1;    /* head margin of a normal burst; < 0: bare 148 samples */
int cellule_tsc_force = -1;   /* training sequence of the normal bursts; < 0: BCC of the BSIC */
double cellule_dec_nb = -1;
double cellule_phase_nb = 0;  /* carrier phase (deg) of the last normal burst (probe) */  /* sampling instant used
 ↪  for the last normal burst (probe) */
int cellule_fenetre_nb = 0;   /* one-shot NB window length in samples (0: 148 + 2 x margin) */
int cellule_sans_bcch;        /* 1: dummy bursts on BCCH/CCCH, the old cell */

static int env_int(const char *nom, int defaut)
{
    const char *e = calypso_getenv(nom);
    return (e && *e) ? (int)strtol(e, NULL, 0) : defaut;
}

/* The 23 octets of the L2 frame carried by BCCH block TC (0..7), or by a CCCH
 * block (tc < 0). Buffers are static, the caller copies. */
static const uint8_t *cellule_l2(int tc)
{
    static uint8_t si1[23], si2[23], si3[23], si4[23], pag[23];
    static int pret;
    if (!pret) {
        pret = 1;
        struct osmo_location_area_id lai = {
            .plmn = { .mcc = (uint16_t)env_int("CELLULE_MCC", 1),
                      .mnc = (uint16_t)env_int("CELLULE_MNC", 1), .mnc_3_digits = false },
            .lac = (uint16_t)env_int("CELLULE_LAC", 1) };
```

```
uint16_t ci = (uint16_t)env_int("CELLULE_CI", 6001);
int arfcn = env_int("CELLULE_ARFCN", 514) & 0x3ff;
struct gsm48_rach_control rach = { .re = 1, .cell_bar = 0, .tx_integer = 9, .max_trans = 3,
                                   .t2 = 0x00, .t3 = 0x00 };
struct gsm48_cell_sel_par csp = { .ms_txpwr_max_ccch = 0, .cell_resel_hyst = 2,
                                  .rxlev_acc_min = 0, .neci = 1, .acs = 0 };
memset(si1, GSM_MACBLOCK_PADDING, 23); memset(si2, GSM_MACBLOCK_PADDING, 23);
memset(si3, GSM_MACBLOCK_PADDING, 23); memset(si4, GSM_MACBLOCK_PADDING, 23);
memset(pag, GSM_MACBLOCK_PADDING, 23);

/* SI1: cell channel description in variable bitmap format (44.018
 * 10.5.2.13.7, the layout gsm48_decode_freq_list() reads): ORIG-ARFCN
 * = our carrier, no further bit set. Rest octet 0x2b: L, no NCH; L,
 * band indicator 1800. */
struct gsm48_system_information_type_1 *s1 = (void *)si1;
s1->header.l2_plen = (uint8_t)((21 << 2) | 1);
s1->header.rr_protocol_discriminator = GSM48_PDISC_RR;
s1->header.skip_indicator = 0;
s1->header.system_information = GSM48_MT_RR_SYSINFO_1;
memset(s1->cell_channel_description, 0, 16);
s1->cell_channel_description[0] = (uint8_t)(0x8e | ((arfcn >> 9) & 1));
s1->cell_channel_description[1] = (uint8_t)((arfcn >> 1) & 0xff);
s1->cell_channel_description[2] = (uint8_t)((arfcn & 1) << 7);
s1->rach_control = rach;
{   /* self-check: decode what we encoded */
    static struct gsm_sysinfo_freq f[1024];
    memset(f, 0, sizeof f);
    gsm48_decode_freq_list(f, s1->cell_channel_description, 16, 0xce, 1);
    int n = 0, ok = 0;
    for (int i = 0; i < 1024; i++) if (f[i].mask) { n++; if (i == arfcn) ok = 1; }
    if (!ok || n != 1)
        printf("cellule : SI1 cell channel description FAUSSE (%d ARFCN decodes, %d attendu %s)\n",
               n, arfcn, ok ? "present" : "ABSENT");
}

/* SI2: no neighbour (bitmap 0 all clear), all NCC permitted */
struct gsm48_system_information_type_2 *s2 = (void *)si2;
s2->header.l2_plen = (uint8_t)((22 << 2) | 1);
s2->header.rr_protocol_discriminator = GSM48_PDISC_RR;
s2->header.skip_indicator = 0;
s2->header.system_information = GSM48_MT_RR_SYSINFO_2;
memset(s2->bcch_frequency_list, 0, 16);
s2->ncc_permitted = 0xff;
s2->rach_control = rach;

/* SI3: identity, combined CCCH, IMSI attach, no periodic LU. Rest
 * octets 0x2b: every optional element absent (all L). */
struct gsm48_system_information_type_3 *s3 = (void *)si3;
s3->header.l2_plen = (uint8_t)((18 << 2) | 1);
s3->header.rr_protocol_discriminator = GSM48_PDISC_RR;
s3->header.skip_indicator = 0;
s3->header.system_information = GSM48_MT_RR_SYSINFO_3;
s3->cell_identity = htons(ci);
gsm48_generate_lai2(&s3->lai, &lai);
s3->control_channel_desc.ccch_conf = 1;        /* 1 CCCH combined with SDCCH/4 */
s3->control_channel_desc.bs_ag_blks_res = 1;
s3->control_channel_desc.att = 1;
s3->control_channel_desc.bs_pa_mfrms = 0;      /* 2 multiframes */
s3->control_channel_desc.t3212 = 0;
s3->cell_options.radio_link_timeout = 7;       /* 32 */
s3->cell_options.dtx = 2;
s3->cell_options.pwrc = 0;
s3->cell_sel_par = csp;
s3->rach_control = rach;

/* SI4: identity again, no CBCH; rest octets all L */
```

```
        struct gsm48_system_information_type_4 *s4 = (void *)si4;
        s4->header.l2_plen = (uint8_t)((12 << 2) | 1);
        s4->header.rr_protocol_discriminator = GSM48_PDISC_RR;
        s4->header.skip_indicator = 0;
        s4->header.system_information = GSM48_MT_RR_SYSINFO_4;
        gsm48_generate_lai2(&s4->lai, &lai);
        s4->cell_sel_par = csp;
        s4->rach_control = rach;

        /* CCCH: PAGING REQUEST TYPE 1, page mode normal, no identity (the
         * fill osmo-bts sends on an idle paging block) */
        static const uint8_t vide[] = { 0x15, 0x06, 0x21, 0x00, 0x01, 0xf0 };
        memcpy(pag, vide, sizeof vide);

        printf("cellule : BCCH SI1-4 MCC=%03u MNC=%02u LAC=%u CI=%u ARFCN=%d, CCCH = paging vide\n",
               lai.plmn.mcc, lai.plmn.mnc, lai.lac, ci, arfcn);
    }
    if (tc < 0) return pag;
    switch (tc & 3) { case 0: return si1; case 1: return si2; case 2: return si3; default: return si4; }
}

/* Normal burst bid (0..3) of the block starting at fn0, or -1 if no block
 * starts there. The four bursts of one block are cached. */
static int cellule_nb(uint32_t fn0, int bid, uint8_t bsic, uint8_t bits[148])
{
    static uint32_t fn_cache = 0xffffffffu;
    static ubit_t bursts[4 * 116];
    if (fn0 != fn_cache) {
        uint32_t p51 = fn0 % 51;
        const uint8_t *l2;
        if (p51 == 2)                                   l2 = cellule_l2((int)((fn0 / 51) % 8));
        else if (p51 == 6 || p51 == 12 || p51 == 16)    l2 = cellule_l2(-1);
        else return -1;
        gsm0503_xcch_encode(bursts, l2);
        fn_cache = fn0;
    }
    /* CELLULE_NB_REPEAT=<k>: burst k of the block in all four positions, to
     * hand the ROM's decoder four bursts it is known to demodulate. */
    { static int rep = -2; if (rep == -2) rep = env_int("CELLULE_NB_REPEAT", -1); if (rep >= 0 && rep < 4) bid =
    ↪  rep; }
    const ubit_t *b = bursts + bid * 116;
    memset(bits, 0, 3);
    memcpy(bits + 3, b, 58);                    /* 57 data + hl */
    memcpy(bits + 61, train_nb[cellule_tsc_force >= 0 ? cellule_tsc_force & 7 : bsic & 7], 26);
    memcpy(bits + 87, b + 58, 58);              /* hu + 57 data */
    memset(bits + 145, 0, 3);
    return 0;
}

/* sb_info of 44.018 9.1.30 for frame fn, byte layout of osmo-bts
 * sched_lchan_fcch_sch.c. T3' = (T3 - 1) / 10: the SCH sits on T3 in
 * {1,11,21,31,41}, so T3' in 0..4. Previously computed as T3 / 10 here and as
 * (T3 - 1) / 10 in pont.c: identical on real SCH frames, different at T3 = 50
 * under cellule_sch_partout. One encoder now, shared by cellule_burst() and
 * cellule_code_attendu(). */
static void sb_info_de_fn(uint32_t fn, uint8_t bsic, uint8_t sb_info[4])
{
    uint32_t t1 = fn / 1326, t2 = fn % 26, t3 = fn % 51;
    uint32_t t3p = t3 ? (t3 - 1) / 10 : 0;
    sb_info[0] = (uint8_t)(((bsic & 0x3f) << 2) | ((t1 & 0x600) >> 9));
    sb_info[1] = (uint8_t)((t1 & 0x1fe) >> 1);
    sb_info[2] = (uint8_t)(((t1 & 0x001) << 7) | ((t2 & 0x1f) << 2) | ((t3p & 0x6) >> 1));
    sb_info[3] = (uint8_t)(t3p & 0x1);
}
/* Caller sets the head margin; the tail margin is trimmed to keep 190 complex
 * samples in total. Sliding the burst inside that fixed-size buffer separates an
```

```
 * influence window that is an ABSOLUTE buffer index (late bits lose influence,
 * early ones gain it) from one RELATIVE to the burst start (the profile moves
 * with the burst and keeps its shape). */
int cellule_marge_fin = -1;

char cellule_burst(uint32_t fn, uint8_t bsic, int amp, double decalage, int marge, int16_t *iq, int *n_iq)
{
    uint8_t bits[148];
    uint32_t p51 = fn % 51;
    char type;
    if (p51 % 10 == 0 && p51 <= 40) {
        memset(bits, 0, sizeof(bits));
        type = 'F';
    } else if ((p51 % 10 == 1 && p51 <= 41) || cellule_sch_partout) {
        uint8_t sb_info[4];
        sb_info_de_fn(fn, bsic, sb_info);
        ubit_t code[78];
        gsm0503_sch_encode(code, sb_info);
        /* Demodulator impulse response: flip exactly ONE of the 78 coded bits and
         * watch which soft-bit positions at 0x2c72 move.
         *   1 position          -> pure permutation, 78 runs give the whole table
         *   3 or 4 around one   -> normal ISI, the chain is sound there
         *   all of them         -> global, wrong traceback (on C54x: CMPS/TRN) */
        {
            /* The flip must stay inside ONE frame: otherwise it perturbs the whole
             * run history (scheduling, AFC) and the differential measurement then
             * compares two histories instead of two bursts.
             * REJEU_INVERSER_FN=<f> restricts the flip to frame f. */
            static int inv = -2; static long invfn = -2;
            if (inv == -2) { const char *e = calypso_getenv("REJEU_INVERSER_BIT"); inv = e ? atoi(e) : -1; }
            if (invfn == -2) { const char *e = calypso_getenv("REJEU_INVERSER_FN"); invfn = e ? atol(e) : -1; }
            if (inv >= 0 && inv < 78 && (invfn < 0 || (long)fn == invfn)) code[inv] ^= 1;
        }
        memset(bits, 0, 3);
        memcpy(bits + 3, code, 39);
        memcpy(bits + 42, train_sb, 64);
        memcpy(bits + 106, code + 39, 39);
        memset(bits + 145, 0, 3);
        /* Midamble probe: the 78 coded bits cannot reach the training sequence, yet
         * that sequence is what channel estimation is supposed to use. If flipping
         * one midamble bit leaves the soft bits untouched, the demodulator ignores
         * the known sequence and equalises against an estimate built elsewhere. */
        {
            /* Same one-frame confinement as the coded-bit flip; without it every SCH
             * burst of the run is hit. REJEU_INVERSER_FN is shared with
             * REJEU_INVERSER_BIT, so only one thing is probed at a time. */
            static int im = -2; static long imfn = -2;
            if (im == -2) { const char *e = calypso_getenv("REJEU_INVERSER_MIDAMBULE");
                            im = e ? atoi(e) : -1; }
            if (imfn == -2) { const char *e = calypso_getenv("REJEU_INVERSER_FN");
                              imfn = e ? atol(e) : -1; }
            if (im >= 0 && im < 64 && (imfn < 0 || (long)fn == imfn)) bits[42 + im] ^= 1;
        }
        type = 'S';
    } else if (!cellule_sans_bcch && p51 >= 2 && p51 <= 49 && p51 % 10 >= 2 &&
               cellule_nb(fn - ((p51 % 10 - 2) & 3), (int)((p51 % 10 - 2) & 3), bsic, bits) == 0) {
        type = (p51 <= 5) ? 'B' : 'C';
    } else {
        memcpy(bits, factice, 148);
        type = '.';
    }
    /* CELLULE_NB_FINE=1 : burst position swept from 2.3 to 4.7 samples in 0.1
     * steps by multiframe (head margin + sampling instant together) */
    if (type == 'B' || type == 'C') {
        static int fine = -2; if (fine == -2) fine = env_int("CELLULE_NB_FINE", 0);
        if (fine && cellule_marge_nb >= 0) {
```

```
        double total = 2.3 + 0.1 * (double)((fn / 51u) % 25u);
        cellule_marge_nb = (int)total; decalage = total - (double)cellule_marge_nb;
        cellule_dec_nb = decalage;
    }
}
int m_tete = -1, m_fin = 0;
if (type == 'S' && marge > 0) {
    m_tete = marge;
    m_fin = (cellule_marge_fin >= 0) ? cellule_marge_fin : marge;
} else if ((type == 'B' || type == 'C') && cellule_marge_nb >= 0) {
    /* the ROM takes 151 samples for an NB window (ALGTH 604): 3 + 148 */
    /* exactly the window: a frame longer than the DMA window leaves
     * samples in the RIF and the next burst starts on them */
    m_tete = cellule_marge_nb;
    m_fin = cellule_fenetre_nb > m_tete + 148 ? cellule_fenetre_nb - m_tete - 148 : 0;
}
/* CELLULE_NB_AMP=<n>: amplitude of the normal bursts alone (FCCH/SCH keep
 * amp), to probe the ROM's fixed-point headroom on the NB path. */
if (type == 'B' || type == 'C') { static int nb_amp = -2; if (nb_amp == -2) nb_amp = env_int("CELLULE_NB_AMP",
 ↪  -1); if (nb_amp > 0) amp = nb_amp; }
/* CELLULE_NB_DEC=<x>|auto : sampling instant of the normal bursts alone
 * (FCCH/SCH keep decalage); auto sweeps 0, 0.25, 0.5, 0.75 by multiframe */
if (type == 'B' || type == 'C') {
    static int mode = -2; static double d = 0;
    if (mode == -2) { const char *e = calypso_getenv("CELLULE_NB_DEC"); mode = 0;
                      if (e && !strcmp(e, "auto")) mode = 2; else if (e && *e) { mode = 1; d = atof(e); } }
    if (mode == 1) decalage = d; else if (mode == 2) decalage = 0.25 * (double)((fn / 51u) % 4u);
    cellule_dec_nb = decalage;
}
/* CELLULE_NB_SHIFT=<n> (experiment): transmit the normal burst with its
 * bits shifted n positions earlier (bit i+n at position i, zeros at the
 * end). Why: the ROM reads the training sequence of its equaliser output
 * at index 58 of the decision buffer (bit 61 with 3 tail bits skipped),
 * but the emulated equaliser puts bit i at index i-2, so the 26 TSC
 * decisions agree with TSC2 at 12/26 as read and 26/26 shifted by one:
 * the residual estimate then sees no signal, the soft-bit scale collapses
 * to 2 and every quantised soft bit is +1. Shifting the bits by one is the
 * test of that reading; it is not a fix. */
if (type == 'B' || type == 'C') {
    static int nsh = -2; if (nsh == -2) nsh = env_int("CELLULE_NB_SHIFT", 0);
    if (nsh > 0) { for (int i = 0; i < 148; i++) bits[i] = (i + nsh < 148) ? bits[i + nsh] : 0; }
    else if (nsh < 0) { for (int i = 147; i >= 0; i--) bits[i] = (i + nsh >= 0) ? bits[i + nsh] : 0; }
}
/* CELLULE_NB_PHASE=<deg>|auto : carrier phase of the normal bursts (auto:
 * 0, 22.5, 45, 67.5 degrees by multiframe). Samples exactly on the I/Q
 * axes (decalage 0, phase 0) or exactly on the diagonals (decalage 0.5)
 * are what a synthetic GMSK gives and what no radio ever gives. */
double phase0 = 0.0;
if (type == 'B' || type == 'C') {
    static int pm = -2; static double pd = 0;
    if (pm == -2) { const char *e = calypso_getenv("CELLULE_NB_PHASE"); pm = 0;
                    if (e && !strcmp(e, "auto")) pm = 2; else if (e && *e) { pm = 1; pd = atof(e); } }
    if (pm == 1) phase0 = pd * M_PI / 180.0; else if (pm == 2) phase0 = 22.5 * (double)((fn / 51u) % 4u) * M_PI
     ↪  / 180.0;
    /* CELLULE_NB_PHASE=quad : 0, 90, 180, 270 degrees by multiframe */
    { static int q = -1; if (q < 0) { const char *e = calypso_getenv("CELLULE_NB_PHASE"); q = (e && !strcmp(e,
     ↪  "quad")) ? 1 : 0; }
      if (q) phase0 = 90.0 * (double)((fn / 51u) % 4u) * M_PI / 180.0; }
    cellule_phase_nb = phase0 * 180.0 / M_PI;
}
/* CELLULE_NB_MSK=1 (experiment): pure MSK for the normal bursts, no
 * Gaussian filter: the phase advances linearly by +-90 degrees per bit,
 * so a sample taken at decalage 0.5 sits EXACTLY on a diagonal, with
 * |I| = |Q|. Tests whether the ROM's NB demodulator hard-decides on the
 * signs of I and Q (measured: only decalage 0.5 ever works, 0.4 and 0.6
 * fail on every burst, an equaliser would degrade smoothly). */
```

```
static int nb_msk = -2; if (nb_msk == -2) nb_msk = env_int("CELLULE_NB_MSK", 0);
if (nb_msk && (type == 'B' || type == 'C')) {
    int16_t *o = iq + 2 * (m_tete >= 0 ? m_tete : 0);
    double ph = phase0; int prev = 1;
    for (int k = 0; k < 148; k++) {
        int d = (bits[k] & 1) ^ prev; prev = bits[k] & 1;
        double al = 1.0 - 2.0 * d;
        double pk = ph + al * (M_PI / 2.0) * decalage;     /* phase at the sampling instant */
        o[2*k] = (int16_t)lrint(amp * cos(pk)); o[2*k+1] = (int16_t)lrint(amp * sin(pk));
        ph += al * (M_PI / 2.0);
    }
    if (m_tete >= 0) {
        memset(iq, 0, (size_t)m_tete * 2 * sizeof(int16_t));
        memset(iq + 2 * (m_tete + 148), 0, (size_t)m_fin * 2 * sizeof(int16_t));
        *n_iq = 2 * (148 + m_tete + m_fin);
    } else *n_iq = 2 * 148;
    return type;
}
/* [2026-09-30] CELLULE_SB_AMP=<n> : amplitude du seul SCH (FCCH et NB gardent
 * amp). Le SNR de la ROM sur le SCH sature a 16384 ; le seul cas de banc a
 * SNR non sature (6974) avait decode. A mesurer. */
if (type == 'S') { static int sba = -2; if (sba == -2) sba = env_int("CELLULE_SB_AMP", -1); if (sba > 0) amp =
 ↪  sba; }
/* CELLULE_SB_PHASE=<deg> : carrier phase of the SCH burst */
if (type == 'S') { static int sp = -2; if (sp == -2) sp = env_int("CELLULE_SB_PHASE", 0); phase0 = sp * M_PI /
 ↪  180.0; }
int16_t *burst_iq = iq;
int fen_n = 148;              /* echantillons sur lesquels portent elargissement et bruit du SCH */
/* [2026-09-30] CELLULE_SB_GARDE=<mode> : LA GARDE N'EST PAS DU SILENCE. En
 * descendant GSM la BTS emet en continu : le SCH est precede du signal de TS7
 * et suivi de TS1, pas de 21 echantillons nuls. Mesure en rejeu (SYM 1.0, DEC
 * 0.35, bruit 3000) : les 20 % de SCH que la ROM rate le sont a burst bien
 * place (TOA 23, SNR sature), et les bits faux se concentrent sur les
 * PREMIERS bits d'information (bits 5-15 du mot, 65-84 %), c'est-a-dire le
 * debut du burst : l'egaliseur demarre sur un front silence -> signal qu'il
 * ne voit jamais sur silicium. Ici la fenetre entiere (marge, burst, marge)
 * est modulee d'un seul trait, les marges portant des bits de garde :
 * 1 = zeros, 2 = pseudo-aleatoires (graine = fn), 3 = uns. 0 = comme avant. */
if (m_tete >= 0) {
    static int garde = -2; if (garde == -2) garde = env_int("CELLULE_SB_GARDE", 0);
    int tot = m_tete + 148 + m_fin;
    if (type == 'S' && garde > 0 && tot <= 512) {
        uint8_t ext[512]; uint32_t seed = fn * 2654435761u + 99u;
        for (int k = 0; k < tot; k++) {
            int i = k - m_tete;
            if (i >= 0 && i < 148) ext[k] = bits[i];
            else if (garde == 1) ext[k] = 0;
            else if (garde == 3) ext[k] = 1;
            else { seed = seed * 1103515245u + 12345u; ext[k] = (uint8_t)((seed >> 16) & 1); }
        }
        { static double fc = -2; if (fc == -2) { const char *e = calypso_getenv("CELLULE_SB_FC"); fc = (e &&
         ↪  *e) ? atof(e) : 0; }
          if (fc > 0) gmsk_moduler_filtre(ext, tot, amp, phase0, decalage, fc, iq);
          else gmsk_moduler(ext, tot, amp, phase0, decalage, iq); }
        *n_iq = 2 * tot;
        burst_iq = iq;            /* elargissement et bruit sur toute la fenetre */
        fen_n = tot;
    } else {
        memset(iq, 0, (size_t)m_tete * 2 * sizeof(int16_t));
        { static double fc = -2; if (fc == -2) { const char *e = calypso_getenv("CELLULE_SB_FC"); fc = (e &&
         ↪  *e) ? atof(e) : 0; }
          /* [2026-09-30] CELLULE_SB_FC=<kHz> : chaine de reception modelisee (voir gmsk_moduler_filtre) */
         if (type == 'S' && fc > 0) gmsk_moduler_filtre(bits, 148, amp, phase0, decalage, fc, iq + 2 * m_tete);
          else gmsk_moduler(bits, 148, amp, phase0, decalage, iq + 2 * m_tete); }
        memset(iq + 2 * (m_tete + 148), 0, (size_t)m_fin * 2 * sizeof(int16_t));
        *n_iq = 2 * (148 + m_tete + m_fin);
```

```
        burst_iq = iq + 2 * m_tete;
    }
} else {
    gmsk_moduler(bits, 148, amp, phase0, decalage, iq);
    *n_iq = 2 * 148;
}
/* [2026-09-21] CELLULE_NB_ISI=<h1>[,<h2>[,<h3>]] : causal tail on the normal
 * bursts, y[n] = x[n] + h1 x[n-1] + h2 x[n-2] + h3 x[n-3]. Why: the ROM
 * correlates the 16 central TSC bits over 10 lags, then picks the 7-lag
 * window of maximum energy (0x8551, sliding sum) and cuts its 5-tap
 * channel estimate from it. Our GMSK at 1 sample/symbol has energy on 3
 * lags only (main + the +-1 ISI), so three of the four windows tie to 0.1 %
 * and the choice is decided by e[lag 8] against e[lag 1] = the GMSK +-2
 * tap (912) against the data leakage: measured over 9 bursts, the window
 * was right (s=2) exactly when e[8] > e[1]. A receiver's analogue filter
 * spreads energy over the following lags and settles it; this tail does
 * the same for the synthetic cell. */
if (type == 'B' || type == 'C') {
    static int isi_n = -2; static double h[4];
    if (isi_n == -2) { isi_n = 0; const char *e = calypso_getenv("CELLULE_NB_ISI");
        if (e && *e) { char tmp[64]; strncpy(tmp, e, sizeof tmp - 1); tmp[sizeof tmp - 1] = 0;
            for (char *t = strtok(tmp, ","); t && isi_n < 3; t = strtok(NULL, ",")) h[++isi_n] = atof(t); } }
    if (isi_n > 0) {
        double xi[148], xq[148];
        for (int k = 0; k < 148; k++) { xi[k] = burst_iq[2*k]; xq[k] = burst_iq[2*k+1]; }
        for (int k = 0; k < 148; k++) {
            double yi = xi[k], yq = xq[k];
            for (int d = 1; d <= isi_n; d++) if (k - d >= 0) { yi += h[d] * xi[k-d]; yq += h[d] * xq[k-d]; }
            if (yi > 32767) yi = 32767;
            if (yi < -32768) yi = -32768;
            if (yq > 32767) yq = 32767;
            if (yq < -32768) yq = -32768;
            burst_iq[2*k] = (int16_t)lrint(yi); burst_iq[2*k+1] = (int16_t)lrint(yq);
        }
    }
}
/* [2026-09-21] CELLULE_NB_SYM=<a> : symmetric spread y[n] = x[n] + a (x[n-1]
 * + x[n+1]) on the normal bursts, timing unchanged. Why: the ROM zeroes
 * every channel tap whose energy is below 1/16 of the window energy
 * (0x7f0c-0x7f1c, threshold = total >> 4, i.e. 25 % in amplitude). The
 * +-1 taps of our GMSK at 1 sample/symbol are 25 % (6467..7189 against a
 * main tap of 27000): measured, the pre-cursor tap was kept (e = 4.70e7 >
 * 4.50e7) on the bursts that decoded and zeroed (4.18e7 < 4.25e7) on the
 * ones that did not, and a 5-tap model without its 25 % pre-cursor gives
 * 45 % errors. A receiver's channel filter widens the pulse; a = 0.3 puts
 * the +-1 taps near 55 % and the +-2 taps near 10 %, both far from the
 * threshold, whatever the data. */
if (type == 'B' || type == 'C') {
    /* default 0.3 since 2026-09-21 (the value that decodes SI1-4); 0 disables */
    static double a = -2; if (a == -2) { const char *e = calypso_getenv("CELLULE_NB_SYM"); a = (e && *e) ?
     ↪  atof(e) : 0.3; }
    gmsk_elargir(burst_iq, 148, a);
}
/* [2026-09-29] CELLULE_SB_SYM=<a> : le meme elargissement sur le SCH. Mesure
 * du banc (run de 23:50, 129 fenetres SB natives, TOA 23-24) : la ROM rend
 * CRC OK sur la moitie des SCH et, sur la plupart des echecs, seuls les
 * bits 0-2 du mot SB sont faux (...1b/1f/1d au lieu de ...1c) : les memes
 * positions du burst basculent selon le contenu, le profil « fil du
 * rasoir » des NB avant CELLULE_NB_SYM. 0 par defaut tant que le rejeu ne
 * l'a pas mesure. */
if (type == 'S') {
    static double a = -2; if (a == -2) { const char *e = calypso_getenv("CELLULE_SB_SYM"); a = (e && *e) ?
     ↪  atof(e) : 0.0; }
    if (a != 0.0) gmsk_elargir(burst_iq, fen_n, a);
}
/* [2026-09-30] CELLULE_SB_ISI=<h1>[,<h2>[,<h3>]] : la traine CAUSALE de
```

```
 * CELLULE_NB_ISI, sur le SCH. Mesure : les 20 % de SCH rates le sont sur
 * la moitie AVANT la sequence d'apprentissage, et l'ensemble des rates est
 * fixe par le contenu (31/33 communs entre deux instants d'echantillonnage).
 * C'est le mecanisme decrit pour les NB : la ROM choisit la fenetre de son
 * estimation de canal au maximum d'energie sur des lags, notre GMSK a
 * 1 ech/symbole met les fenetres a egalite et les donnees voisines
 * tranchent. Une traine causale (comme le filtre analogique d'un vrai
 * recepteur) leve l'egalite. Normalisee par 1+sum(h) pour rester en 16 bits. */
if (type == 'S') {
    static int isi_n = -2; static double h[4];
    if (isi_n == -2) { isi_n = 0; const char *e = calypso_getenv("CELLULE_SB_ISI");
        if (e && *e) { char tmp[64]; strncpy(tmp, e, sizeof tmp - 1); tmp[sizeof tmp - 1] = 0;
            for (char *t = strtok(tmp, ","); t && isi_n < 3; t = strtok(NULL, ",")) h[++isi_n] = atof(t); } }
    if (isi_n > 0) {
        double norm = 1.0; for (int d = 1; d <= isi_n; d++) norm += fabs(h[d]);
        double xi[512], xq[512];
        for (int k = 0; k < fen_n; k++) { xi[k] = burst_iq[2*k]; xq[k] = burst_iq[2*k+1]; }
        for (int k = 0; k < fen_n; k++) {
            double yi = xi[k], yq = xq[k];
            for (int d = 1; d <= isi_n; d++) if (k - d >= 0) { yi += h[d] * xi[k-d]; yq += h[d] * xq[k-d]; }
            burst_iq[2*k] = (int16_t)lrint(yi / norm); burst_iq[2*k+1] = (int16_t)lrint(yq / norm);
        }
    }
}
/* [2026-09-30] CELLULE_SB_NOISE=<sigma> : le meme bruit gaussien sur le SCH
 * (voir CELLULE_NB_NOISE ci-dessous : sans bruit, l'echelle des bits
 * souples de la ROM s'effondre). 0 = inchange. */
if (type == 'S') {
    static double sigma = -2; if (sigma == -2) { const char *e = calypso_getenv("CELLULE_SB_NOISE"); sigma =
     ↪  (e && *e) ? atof(e) : 0.0; }
    if (sigma > 0) {
        static uint32_t graine = 0xffffffffu;   /* CELLULE_SB_NOISE_SEED : autre realisation du bruit */
        if (graine == 0xffffffffu) graine = (uint32_t)env_int("CELLULE_SB_NOISE_SEED", 0);
        uint32_t seed = fn * 2654435761u + 777u + graine * 7919u;
        for (int k = 0; k < 2 * fen_n; k++) {
            seed = seed * 1103515245u + 12345u; double u1 = ((seed >> 8) & 0xffff) / 65536.0 + 1e-6;
            seed = seed * 1103515245u + 12345u; double u2 = ((seed >> 8) & 0xffff) / 65536.0;
            double g = sqrt(-2.0 * log(u1)) * cos(2.0 * M_PI * u2);
            double v = burst_iq[k] + sigma * g;
            if (v > 32767) v = 32767;
            if (v < -32768) v = -32768;
            burst_iq[k] = (int16_t)lrint(v);
        }
    }
}
/* [2026-09-21] CELLULE_NB_NOISE=<sigma> : Gaussian noise on the normal
 * bursts (deterministic seed per frame). Why: the ROM scales its soft bits
 * by a noise estimate before the 4-bit quantiser (0x8168 -> 0x82d0, a
 * 129-entry table indexed by soft >> 8); with a noiseless burst the scaled
 * values are 2..5 instead of thousands, every index is 0 and all 116
 * quantised soft bits come out +1 (measured in the storage at 0x4200 +
 * 29 x burst): the sign is lost before the deinterleaver. A radio always
 * carries noise; the cell now does too. */
if (type == 'B' || type == 'C') {
    static double sigma = -2; if (sigma == -2) { const char *e = calypso_getenv("CELLULE_NB_NOISE"); sigma =
     ↪  (e && *e) ? atof(e) : 0.0; }
    if (sigma > 0) {
        uint32_t seed = fn * 2654435761u + 12345u;
        for (int k = 0; k < 296; k++) {
            /* Box-Muller on a small LCG */
            seed = seed * 1103515245u + 12345u; double u1 = ((seed >> 8) & 0xffff) / 65536.0 + 1e-6;
            seed = seed * 1103515245u + 12345u; double u2 = ((seed >> 8) & 0xffff) / 65536.0;
            double g = sqrt(-2.0 * log(u1)) * cos(2.0 * M_PI * u2);
            double v = burst_iq[k] + sigma * g;
            if (v > 32767) v = 32767;
            if (v < -32768) v = -32768;
```

```
                burst_iq[k] = (int16_t)lrint(v);
            }
        }
    }
    /* [2026-09-21] CELLULE_NB_ZERO_DC=1 (experiment): remove the mean of the
     * 148 samples of a normal burst. Measured: the ROM demodulates a normal
     * burst perfectly when the data-induced mean of its samples is below ~7 %
     * of the amplitude and loses it entirely above ~10 % (16 bursts of SI1-4,
     * no exception), the SCH being immune. */
    if (type == 'B' || type == 'C') {
        static int zdc = -2; if (zdc == -2) zdc = env_int("CELLULE_NB_ZERO_DC", 0);
        if (zdc) {
            long si = 0, sq = 0;
            for (int k = 0; k < 148; k++) { si += burst_iq[2*k]; sq += burst_iq[2*k+1]; }
            int mi = (int)(si / 148), mq = (int)(sq / 148);
            for (int k = 0; k < 148; k++) { burst_iq[2*k] = (int16_t)(burst_iq[2*k] - mi); burst_iq[2*k+1] =
             ↪  (int16_t)(burst_iq[2*k+1] - mq); }
        }
    }
    return type;
}

/* Expected 78 coded bits for a frame, to compare sign by sign with the soft bits
 * the DSP produces. */
void cellule_code_attendu(uint32_t fn, uint8_t bsic, unsigned char *code78)
{
    uint8_t sb_info[4];
    sb_info_de_fn(fn, bsic, sb_info);
    ubit_t code[78];
    gsm0503_sch_encode(code, sb_info);
    for (int i = 0; i < 78; i++) code78[i] = (unsigned char)code[i];
}

/* Reference demodulator, in C, to prove the samples handed to the DSP do carry the
 * message. GMSK at 1 sample/symbol: the phase advances by (pi/2)*alpha_n per symbol,
 * so alpha_n = sign(arg(x[n] * conj(x[n-1]))). Recover the alpha sequence, correlate
 * it with the midamble to find the offset, then read the data bits from there. */
#include <math.h>
int cellule_demod_reference(const int16_t *x, int n_ech, unsigned char *bits148, int *offset)
{
    static double alpha[512];
    if (n_ech > 512) n_ech = 512;
    for (int n = 1; n < n_ech; n++) {
        double i0 = x[2*(n-1)], q0 = x[2*(n-1)+1];
        double i1 = x[2*n],     q1 = x[2*n+1];
        /* x[n] * conj(x[n-1]) */
        double re = i1*i0 + q1*q0, im = q1*i0 - i1*q0;
        alpha[n] = atan2(im, re);
    }
    alpha[0] = 0;
    /* reference alpha of the midamble: d_j = t_j XOR t_{j-1}, burst positions 43..105 */
    double ref[63];
    for (int j = 1; j < 64; j++) ref[j-1] = (train_sb[j] ^ train_sb[j-1]) ? -1.0 : 1.0;
    int best = -1; double bestv = -1e30;
    for (int k = 0; k + 63 < n_ech; k++) {
        double acc = 0;
        for (int j = 0; j < 63; j++) acc += ref[j] * alpha[k + j];
        if (acc > bestv) { bestv = acc; best = k; }
    }
    if (best < 0) return -1;
    /* the midamble starts 43 symbols into the burst */
    int b0 = best - 43;
    *offset = b0;
    if (b0 < 0 || b0 + 148 > n_ech) return -1;
    int prev = 1;
    for (int i = 0; i < 148; i++) {
```

```
        int d = (alpha[b0 + i] < 0) ? 1 : 0;   /* alpha = 1-2d */
        bits148[i] = (unsigned char)(d ^ prev);
        prev = bits148[i];
    }
    return 0;
}

/* Raw DIFFERENTIAL bits d_i, with no differential decoding: an isolated error does
 * not propagate here, unlike in b_i = d_i XOR b_{i-1}. This is the honest measure of
 * demodulator quality. */
int cellule_demod_d(const int16_t *x, int n_ech, int b0, unsigned char *d148)
{
    if (b0 < 1 || b0 + 148 > n_ech) return -1;
    for (int i = 0; i < 148; i++) {
        int n = b0 + i;
        double i0 = x[2*(n-1)], q0 = x[2*(n-1)+1];
        double i1 = x[2*n],     q1 = x[2*n+1];
        double im = q1*i0 - i1*q0, re = i1*i0 + q1*q0;
        d148[i] = (unsigned char)(atan2(im, re) < 0 ? 1 : 0);
    }
    return 0;
}

int cellule_train_sb(int i) { return (i >= 0 && i < 64) ? train_sb[i] : -1; }

/* The dummy burst (45.002 5.2.6) as 148 GMSK samples: what the BCCH carrier
 * transmits on every timeslot that carries nothing else. The FB search of the
 * ROM receives the whole frame, so the seven other timeslots must look like a
 * real C0 carrier, not like silence. */
void cellule_factice(int amp, double decalage, int16_t *iq)
{
    gmsk_moduler(factice, 148, amp, 0.0, decalage, iq);
}

/* The 148 bits of the TN0 burst of frame fn as cellule_burst() sends them, for
 * probes that look for them inside the DSP memory. 0 if it is a normal burst. */
int cellule_bits_attendus(uint32_t fn, uint8_t bsic, uint8_t bits[148])
{
    uint32_t p51 = fn % 51;
    if (cellule_sans_bcch || p51 < 2 || p51 > 49 || p51 % 10 < 2) return -1;
    return cellule_nb(fn - ((p51 % 10 - 2) & 3), (int)((p51 % 10 - 2) & 3), bsic, bits);
}

/* Expected decoder-side vectors for the block that ends at frame fn (burst 3):
 * the 456 convolutionally coded bits in deinterleaved order (45.003 4.1.4:
 * bit k of the coded block sits in burst k mod 4 at position 2*((49k) mod 57)
 * + ((k mod 8) div 4) of the 114 data bits) and the 184 information bits +
 * 40 parity + 4 tail (the 23 octets as sent, MSB first, then the Fire parity
 * as gsm0503 computes it: we only need the 184 here). */
int cellule_bloc_attendu(uint32_t fn, uint8_t bsic, uint8_t code456[456], uint8_t info184[184])
{
    uint32_t p51 = fn % 51;
    if (cellule_sans_bcch || p51 < 2 || p51 > 49 || p51 % 10 < 2) return -1;
    uint32_t fn0 = fn - ((p51 % 10 - 2) & 3);
    uint8_t bits[148]; uint8_t data[4][114];
    for (int b = 0; b < 4; b++) {
        if (cellule_nb(fn0, b, bsic, bits) < 0) return -1;
        memcpy(data[b], bits + 3, 57); memcpy(data[b] + 57, bits + 88, 57);
    }
    for (int k = 0; k < 456; k++) {
        int b = k & 3, j = 2 * ((49 * k) % 57) + ((k % 8) / 4);
        code456[k] = data[b][j];
    }
    uint32_t tc = (fn0 / 51) % 8;
    const uint8_t *l2 = (p51 <= 5) ? cellule_l2((int)tc) : cellule_l2(-1);
    /* [2026-09-21] LSB first within each octet: that is how gsm0503_xcch_encode
```

```
     * unpacks the L2 frame before the Fire parity and the convolutional code
     * (osmo_pbit2ubit_ext(..., lsb_mode=1)), so it is the bit order the ROM's
     * Viterbi has to reproduce. MSB first gave a false "78 faux" on a correct
     * block. */
    for (int i = 0; i < 184; i++) info184[i] = (l2[i / 8] >> (i % 8)) & 1;
    return 0;
}

/* The 228 bits the Viterbi decoder must output: 184 information bits, 40 Fire
 * parity bits, 4 tail bits (45.003 4.1.1-4.1.2). */
#include <osmocom/core/crc64gen.h>
#include <osmocom/coding/gsm0503_parity.h>
int cellule_u228_attendu(uint32_t fn, uint8_t bsic, uint8_t u228[228])
{
    uint8_t code[456], info[184];
    if (cellule_bloc_attendu(fn, bsic, code, info) < 0) return -1;
    memcpy(u228, info, 184);
    osmo_crc64gen_set_bits(&gsm0503_fire_crc40, info, 184, u228 + 184);
    memset(u228 + 224, 0, 4);
    return 0;
}
```

3.15 /opt/GSM/c54x_exe/src/cellule.h

1880 octets, 29 lignes → 29 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
#ifndef CELLULE_H
#define CELLULE_H
#include <stdint.h>
/* Downlink TN0 burst of a GSM cell for frame fn (51-multiframe: FCCH on
 * 0/10/20/30/40, SCH on 1/11/21/31/41, dummy burst elsewhere), GMSK modulated,
 * 148 int16 I/Q samples. Returns the burst type: 'F', 'S', 'B' (BCCH, SI1-4),
 * 'C' (CCCH, empty paging) or '.' (dummy).
 * marge = silence samples prepended and appended to an SCH burst: the ROM SB
 * decoder reads a 190-sample window, 148 + 2 x 21 (BSP_IQ_MAX_I16 in
 * calypso_bsp.c, "SB en demande 190"). *n_iq gets the number of int16 written. */
extern int cellule_sch_partout;
extern int cellule_marge_fin;
extern int cellule_marge_nb;     /* head/tail margin of BCCH/CCCH normal bursts, < 0 = none (pont.c sets it per
 ↪  frame) */
extern int cellule_sans_bcch;
extern double cellule_dec_nb;
extern double cellule_phase_nb;
extern int cellule_fenetre_nb;   /* NB window length in samples, the frame is padded to it exactly */
extern int cellule_tsc_force;    /* TSC of the normal bursts, < 0 = BCC (CELLULE_TSC, pont.c) */    /* 1: the old
 ↪  cell, dummy bursts on BCCH/CCCH */
int cellule_train_sb(int i);
int cellule_demod_d(const int16_t *x, int n_ech, int b0, unsigned char *d148);
int cellule_demod_reference(const int16_t *x, int n_ech, unsigned char *bits148, int *offset);
void cellule_code_attendu(uint32_t fn, uint8_t bsic, unsigned char *code78);
char cellule_burst(uint32_t fn, uint8_t bsic, int amp, double decalage, int marge, int16_t *iq, int *n_iq);
int cellule_u228_attendu(uint32_t fn, uint8_t bsic, uint8_t u228[228]);
int cellule_bloc_attendu(uint32_t fn, uint8_t bsic, uint8_t code456[456], uint8_t info184[184]);
int cellule_bits_attendus(uint32_t fn, uint8_t bsic, uint8_t bits[148]);
void cellule_factice(int amp, double decalage, int16_t *iq);   /* 148 GMSK samples of the dummy burst */
#endif
```

3.16 /opt/GSM/c54x_exe/src/main.c

12822 octets, 278 lignes → 278 lignes

```c
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * c54x_exe - the Calypso TMS320C54x DSP, run without QEMU and without the ARM.
 *
 * Bench for the symptom recorded in the qemu-calypso README [2026-09-16]:
 * a_sch[0] = 0x8100 (B_BLUD | B_SCH_CRC), and a_sch[3] = 0xf8d8 CONSTANT over
 * 21/21 writes while 10 distinct burst contents were presented. A decoder
 * whose output does not depend on its input is not decoding.
 *
 * Reaching that point used to mean booting QEMU, the ARM and the osmocom-bb
 * firmware: seconds per run. The DSP needs almost none of it - out of 26631
 * lines of C54x L1, 14 symbols come from QEMU (two of them mutexes in the
 * core), and calypso_{arm2dsp,dma,fbsb,mailbox}.c need none. Here the TI mask
 * ROM runs alone in milliseconds and we watch what the DSP writes into API
 * RAM, which makes the question "does the output depend on the input?"
 * answerable in a loop, hence in CI.
 *
 * The sources are NOT copied here: this binary compiles those of
 * /opt/GSM/qosmo. Copying them would recreate the divergence this bench exists
 * to remove.
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "qemu/thread.h"
#include "calypso_c54x.h"
#include "hw/arm/calypso/calypso_api.h"
#include "hw/arm/calypso/calypso_dsp_pont.h"
#include "calypso_dma.h"
#include "calypso_bsp.h"
#include "verbosite.h"
#include "rejouer.h"
#include "pont.h"

/* -- what the platform would otherwise provide ------------------------------ */
extern int c54x_rapide;      /* calypso_c54x.h : fast path of the core */
extern int c54x_sondes;      /* c54x_internal.h : sondes pures, coupees par defaut */
uint32_t g_c54x_exe_fn;      /* non-static: pont.c updates it on every TICK */
uint32_t calypso_trx_get_fn(void) { return g_c54x_exe_fn; }

/* No interrupt controller here, so the ack the DSP raises has nowhere to go.
 * If DSP behaviour ever turns out to depend on that ack, this binary diverges
 * from QEMU at exactly this point. */
void calypso_inth_arm_ack(void) { }

/* The DARAM lock normally lives in l1-dsp/calypso_full_pcb.c, excluded from
 * this binary because it includes hw/core/cpu.h - all of QEMU behind it. Same
 * pthread_mutex: the DSP really locks. */
QemuMutex calypso_pcb_daram_lock;

/* Guest memory: with no ARM nobody writes into it, but shared sources
 * reference it. */
void cpu_physical_memory_rw(uint64_t addr, void *buf, uint64_t len, bool wr)
{
    (void)addr;
    if (!wr) {
        memset(buf, 0, len);
    }
}

/* -- ROMs at their silicon addresses (cf. calypso_trx.c of qosmo-dsp) ------- */
static const struct { const char *suffixe; uint32_t adresse; bool programme; }
ROMS[] = {
    { "PROM0",  0x07000, true  },
```

```
    { "PROM1",  0x18000, true  },   /* page 1, reached with XPC=1 */
    { "PROM2",  0x28000, true  },
    { "PROM3",  0x38000, true  },
    { "DROM",   0x09000, false },
    { "PDROM",  0x0E000, false },   /* mapped on the DATA side ... */
    { "PDROM",  0x0E000, true  },   /* ... AND on the PROGRAM side (IT vectors) */
};

static void usage(const char *prog)
{
    fprintf(stderr,
        "usage: %s [options]\n"
        "  --rom-dir DIR     ou sont calypso_dsp.*.bin   (defaut /opt/GSM)\n"
        "  --trames N        nombre de trames TDMA       (defaut 100, 4000 avec --rejouer)\n"
        "  --insns N         instructions par trame      (defaut 2300, 200000 avec --arm)\n"
        "  --verbeux         une ligne par trame\n"
        "  --iq MODE         avec --arm : injecter un burst I/Q a chaque trame :\n"
        "                    fcch (rotation +pi/2/ech.), noise, tone:<dphi>, none,\n"
        "                    cell[:bsic[:decalage]] = FCCH+SCH+factice GMSK (multitrame 51)\n"
        "  --amp N           amplitude int16 des echantillons injectes (defaut 30000)\n"
        "  --rejouer         rejeu DETERMINISTE de l'acquisition FB/SB (sans QEMU)\n"
        "  --bsic N          BSIC de la cellule injectee en rejeu (defaut 7)\n"
        "  --arm [SOCKET]    servir l'ARM de QEMU (qosmo, CALYPSO_DSP_EXTERN=1) :\n"
        "                    API RAM partagee dans /dev/shm%s, trame verrouillee\n"
        "                    sur %s\n"
        "  -v .. -vvvvvv     traces du coeur C54x (stderr), par niveau :\n"
        "                    (rien)  erreurs seulement\n"
        "                    -v      + avertissements\n"
        "                    -vv     + cycle de vie : boot, reset, gates ACTIF/INACTIVE\n"
        "                    -vvv    + taches, API RAM, interruptions, chemins FB/SB\n"
        "                    -vvvv   + sondes memoire : WATCH, DUMP, SCAN, MAP, SP-*\n"
        "                    -vvvvv  + pas a pas : BRANCH-TRACE, LOOPTRACE, TERM, CYCLE\n"
        "                    -vvvvvv tout, stderr brut\n",
        prog, CALYPSO_PONT_SHM, CALYPSO_PONT_SOCK);
}

int main(int argc, char **argv)
{
    const char *rom_dir = "/opt/GSM";
    const char *arm_sock = NULL;
    int rejeu = 0, bsic_rej = 7;
    const char *iq_mode = "none";
    int amp = 30000;
    long trames = -1, insns = -1;
    bool verbeux = false;
    int niveau = 0;

    for (int i = 1; i < argc; i++) {
        const char *a = argv[i];
        if (!strcmp(a, "--rom-dir") && i + 1 < argc)      rom_dir = argv[++i];
        else if (!strcmp(a, "--trames") && i + 1 < argc)  trames = atol(argv[++i]);
        else if (!strcmp(a, "--insns") && i + 1 < argc)   insns = atol(argv[++i]);
        else if (!strcmp(a, "--verbeux"))                 verbeux = true;
        else if (!strcmp(a, "--iq") && i + 1 < argc)      iq_mode = argv[++i];
        else if (!strcmp(a, "--amp") && i + 1 < argc)     amp = atoi(argv[++i]);
        else if (!strcmp(a, "--rejouer")) { rejeu = 1; }
        else if (!strcmp(a, "--bsic") && i + 1 < argc) bsic_rej = atoi(argv[++i]);
        else if (!strcmp(a, "--arm")) {
            arm_sock = (i + 1 < argc && argv[i + 1][0] != '-') ? argv[++i] : CALYPSO_PONT_SOCK;
        }
        else if (a[0] == '-' && a[1] == 'v' && strspn(a + 1, "v") == strlen(a + 1)) {
            niveau = (int)strlen(a + 1);
        }
        else if (!strcmp(a, "-h") || !strcmp(a, "--help")) { usage(argv[0]); return 0; }
        else { usage(argv[0]); return 2; }
    }
```

```
if (insns < 0) {
    /* [2026-09-20] 200000 under --arm: the FB search over whole 1250-symbol
     * frames costs the ROM 24-36k instructions per frame, and a frame cut
     * short by the budget leaves the ARM reading half-written results (a
     * zeroed a_sch read as a CRC-OK SB). Silicon has ~360k cycles per frame
     * at 78 MHz; 32000 was a bench constant, not a hardware one. */
    insns = arm_sock ? 200000 : 2300;
}
if (trames < 0) {
    trames = rejeu ? 4000 : 100;       /* an explicit --trames is honoured as is */
}
verbosite_installer(niveau);
/* Fast path of the core (calypso_c54x.h): on unless a probe is armed. */
{ const char *d = getenv("CALYPSO_DEBUG"), *r = getenv("CALYPSO_C54X_RAPIDE");
  c54x_rapide = (r && *r) ? (*r != '0') : !(d && *d); }
/* [2026-09-23] Sondes pures du coeur (c54x_internal.h) : coupees par defaut ;
 * -vvvv et plus les allument, puisque c'est a ce niveau qu'on les lit.
 * Sinon le coeur resout CALYPSO_SONDES / CALYPSO_DEBUG au premier c54x_init. */
if (niveau >= 4) c54x_sondes = 1;

qemu_mutex_init(&calypso_pcb_daram_lock);

/* API RAM: private, or shared with the QEMU ARM. It must be installed
 * BEFORE the ROMs, since the loader copies into it whatever falls inside
 * the 0x0800 window. Under --arm the shared segment IS data[0x0800..] and
 * api_ram aliases it, so every core write reaches the ARM by either
 * path. */
static uint16_t api_ram_privee[CALYPSO_API_WORDS];
uint16_t *api_ram = api_ram_privee;
C54xState *dsp;
if (arm_sock || rejeu) {
    /* A pending interrupt (IFR&IMR) is taken as soon as INTM drops: real
     * C54x behaviour (SPRU131 ch.6), not a workaround. The core gates it
     * behind CALYPSO_C54X_IRQ_LEVEL; turn it on here unless explicitly
     * overridden (empty CALYPSO_C54X_IRQ_LEVEL turns it off). Applied to
     * BOTH benches: the replay used to run the core without it and could
     * not reproduce the bridge. */
    const char *e = getenv("CALYPSO_C54X_IRQ_LEVEL");
    if (!e) setenv("CALYPSO_C54X_IRQ_LEVEL", "1", 1);
    else if (!*e) unsetenv("CALYPSO_C54X_IRQ_LEVEL");
}
if (arm_sock) {
    dsp = pont_allouer_dsp();
    if (!dsp) { return 1; }
    api_ram = &dsp->data[C54X_API_BASE];
} else {
    dsp = c54x_init();
    if (!dsp) { fprintf(stderr, "c54x_init a echoue\n"); return 1; }
}
c54x_set_api_ram(dsp, api_ram);

int charges = 0;
for (unsigned i = 0; i < sizeof(ROMS) / sizeof(ROMS[0]); i++) {
    char chemin[512];
    snprintf(chemin, sizeof(chemin), "%s/calypso_dsp.%s.bin", rom_dir, ROMS[i].suffixe);
    int n = c54x_load_section(dsp, chemin, ROMS[i].adresse, ROMS[i].programme);
    if (n < 0) {
        fprintf(stderr, "ROM manquante : %s\n", chemin);
        return 1;
    }
    printf("  %-6s %-5s 0x%05x  %6d mots\n", ROMS[i].suffixe,
           ROMS[i].programme ? "prog" : "data", ROMS[i].adresse, n);
    charges++;
}
{ char chemin[512];
  snprintf(chemin, sizeof(chemin), "%s/calypso_dsp.Registers.bin", rom_dir);
```

```
  int n = c54x_load_registers(dsp, chemin);
  printf("  %-6s %-5s %-7s  %6d mots\n", "Regs", "mmr", "", n); }

printf("%d sections chargees, reset...\n", charges);
c54x_reset(dsp);

if (rejeu) {
    calypso_dma_init();
    calypso_bsp_init(dsp);
    if (insns < 32000) insns = 32000;
    if (!iq_mode || !*iq_mode || !strcmp(iq_mode, "none")) iq_mode = "cell";
    int rc = rejouer(dsp, api_ram, trames, insns, iq_mode, amp, bsic_rej, verbeux);
    verbosite_retirer();
    verbosite_bilan(stdout);
    return rc;
}
if (arm_sock) {
    /* Same sequence as calypso_trx_init() of qosmo-dsp after reset. */
    calypso_dma_init();
    calypso_bsp_init(dsp);
    int rc = pont_serveur(dsp, api_ram, arm_sock, insns, verbeux, iq_mode, amp);
    verbosite_retirer();
    verbosite_bilan(stdout);
    return rc;
}

/* The README observables, sampled once per frame. */
uint16_t *d_fb_det = &api_ram[(API_NDB + NDB_D_FB_DET) / 2];
uint16_t *a_sch0   = &api_ram[(API_R_PAGE(0) + RP_A_SCH) / 2];

unsigned fb_vus = 0, sch_ecrits = 0;
uint16_t fb_precedent = 0, sch_sig_prec = 0xFFFF;
uint16_t sch3_premier = 0; bool sch3_varie = false, sch3_vu = false;

if (verbeux) {
    printf("\n  trame    d_fb_det   a_sch[0]  a_sch[1]  a_sch[2]  a_sch[3]\n");
}
for (long t = 0; t < trames; t++) {
    g_c54x_exe_fn = (uint32_t)t;
    c54x_run(dsp, (int)insns);

    /* Count TRANSITIONS, not frames where the value is non-zero: the bit
     * stays raised, so counting per frame reported "50 out of 50", which
     * only measured how long the run lasted. */
    if (*d_fb_det && !fb_precedent) fb_vus++;
    fb_precedent = *d_fb_det;

    uint16_t sch_sig = (uint16_t)(a_sch0[0] ^ a_sch0[3]);
    if ((a_sch0[0] || a_sch0[3]) && sch_sig != sch_sig_prec) {
        sch_ecrits++;
        if (!sch3_vu) { sch3_premier = a_sch0[3]; sch3_vu = true; }
        else if (a_sch0[3] != sch3_premier) sch3_varie = true;
        sch_sig_prec = sch_sig;
    }
    if (verbeux) {
        printf("  %5ld    %6u     0x%04x    0x%04x    0x%04x    0x%04x\n",
               t, *d_fb_det, a_sch0[0], a_sch0[1], a_sch0[2], a_sch0[3]);
    }
}

verbosite_retirer();
printf("\n--- bilan sur %ld trames ---\n", trames);
printf("  AUCUN burst injecte : le DSP tourne sur une API RAM vierge, sans\n"
       "  ARM ni TPU. Les valeurs ci-dessous disent que la mask-ROM EXECUTE,\n"
       "  pas encore qu'elle decode. Injecter des bursts est l'etape suivante.\n\n");
printf("  d_fb_det leve      : %u transition(s) 0 -> 1\n", fb_vus);
```

```
    printf("  a_sch change       : %u fois\n", sch_ecrits);
    if (sch3_vu) {
        printf("  a_sch[3]           : 0x%04x%s\n", sch3_premier,
               sch3_varie ? " puis VARIE" : " CONSTANT sur toutes les ecritures");
        if (!sch3_varie) {
            printf("\n  ^ c'est le symptome du README : une sortie qui ne depend\n"
                   "    pas de l'entree. Ici l'entree est vide, donc attendu.\n");
        }
    } else {
        printf("  a_sch              : jamais ecrit (la tache SB n'a pas tourne)\n");
    }
    verbosite_bilan(stdout);
    return 0;
}
```

3.17 /opt/GSM/c54x_exe/src/montant.c

77496 octets, 1640 lignes → 1640 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * montant.c - le lien montant du montage DSP : RACH, SDCCH, SACCH, FACCH, parole.
 *
 * POURQUOI CE FICHIER EXISTE
 * --------------------------
 * Sous CALYPSO_DSP_EXTERN=1, calypso_l1_do_init() appelle calypso_l1_disable()
 * (« couche 1 « grgsm » desactivee (DSP externe) ») : plus aucune couche 1
 * n'est enregistree dans QEMU, donc calypso_l1_do_rach_written() et
 * calypso_l1_do_page_written() (calypso_l1_dispatch.c) sont des no-op. Or
 * c'etaient eux qui, en montage grgsm, publiaient le montant dans les
 * side-bands /dev/shm que pont.py consomme (pont/uplink.py) :
 *
 *   /dev/shm/calypso_rach           RACH        (ra, bsic)
 *   /dev/shm/calypso_sdcch_ul       SDCCH UL    (bloc L2 de 23 octets)
 *   /dev/shm/calypso_tch_facch_ul   FACCH UL
 *   /dev/shm/calypso_tch_sacch_ul   SACCH UL
 *   /dev/shm/calypso_tch_ul         parole (anneau de trames FR)
 *
 * Resultat mesure le 2026-09-21 : le firmware emettait bien ses
 * L1CTL_RACH_REQ, le mobile comptait ses « RANDOM ACCESS (requests left 8..4) »,
 * mais pont.py affichait « UL bursts=0 rach=0 » et /dev/shm/calypso_rach
 * n'existait meme pas. Sans RACH il n'y a pas d'IMM ASS, donc pas de SDCCH,
 * donc jamais de LOCATION UPDATING ACCEPT.
 *
 * COMMENT
 * -------
 * Ce processus tient l'API RAM partagee : on reprend donc, a l'identique, les
 * captures de qosmo-grgsm/hw/arm/calypso/calypso_l1_grgsm.c, mais declenchees
 * par SCRUTATION une fois par trame au lieu des callbacks d'ecriture de QEMU.
 * Le point d'appel (pont.c, fin du PONT_TICK) correspond a la fin du scenario
 * de l'ARM : dsp_end_scenario() vient d'ecrire d_dsp_page = B_GSM_TASK | page,
 * et les mots de tache de cette page W sont a jour.
 *
 * Le RACH est le seul cas ou la scrutation n'est pas equivalente a un
 * callback : le firmware ecrit d_rach (prim_rach.c:72) puis d_task_ra, et
 * personne ne les efface ensuite (sync.c:307 ne les remet a zero que sur
 * ABORT). On declenche donc sur FRONT : premiere valeur non nulle, ou valeur
 * differente de la precedente. Angle mort assume : deux tentatives de suite
 * avec le meme (RA, BSIC), soit ~1/256 puisque la RA est tiree au hasard par
 * gsm48_rr ; la tentative suivante passe. MONTANT_CONSOMME_RACH=1 remet d_rach
 * a zero apres publication, ce qui rend le declenchement exact — au prix d'une
 * ecriture dans la fenetre API que la ROM pourrait lire.
 *
 * Ce module ne parle pas TRXD : calypso_bsp_send_rach_ra() existe (code mort
```

```
 * depuis le refactor « couche 1 enregistree ») mais enverrait l'access-burst
 * sur 127.0.0.1:5702, c'est-a-dire la socket DESCENDANTE de pont.py, dont
 * run_data() fait « self.bts_data = addr » sur tout paquet recu : le burst
 * serait relu comme une descente et l'adresse de la BTS ecrasee. La voie des
 * side-bands passe par la machinerie montante de pont.py, celle qui est deja
 * eprouvee en montage grgsm.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <stdint.h>
#include <unistd.h>
#include <fcntl.h>
#include "hw/arm/calypso/calypso_api.h"
#include "hw/arm/calypso/calypso_debug.h"
#include "calypso_bsp.h"
#include "montant.h"
#include <osmocom/core/bits.h>
#include <osmocom/coding/gsm0503_coding.h>   /* prothese FACCH descendante */

#define SHM_RACH        "/dev/shm/calypso_rach"
#define SHM_SDCCH_UL    "/dev/shm/calypso_sdcch_ul"
#define SHM_FACCH_UL    "/dev/shm/calypso_tch_facch_ul"
#define SHM_SACCH_UL    "/dev/shm/calypso_tch_sacch_ul"
#define SHM_TCH_UL      "/dev/shm/calypso_tch_ul"
#define SHM_KC          "/dev/shm/calypso_kc_l1"

/* Tailles et dispositions : pont/uplink.py. */
#define REC_RACH        16      /* lu 12 : seq(4) ra(1) bsic(1) ..(2) fn(4)   */
#define REC_L2          48      /* lu 39 : seq(4) l1s(4) fn(4) task(2) . p51(1) . l2(23) */
#define TCH_UL_SLOTS    16
#define TCH_UL_SLOT_SZ  64      /* TCH_UL_SLOT       */
#define TCH_UL_FR_OFS   16      /* TCH_UL_FR_OFS     */
#define FR_BYTES        33

/* Fenetre de recherche de l'en-tete L2 dans a_cu (SDCCH_UL_WINDOW_OFS de
 * calypso_l1_grgsm.c). */
#define SDCCH_UL_WINDOW_OFS  6

/* Anti-doublon SDCCH montant.
 *
 * [2026-09-21, mesure] La couche 1 gr-gsm republiait un bloc identique passe
 * 60 trames (SDCCH_UL_DEDUP_TICKS). Repris tel quel ici, ca tuait la
 * connexion : le firmware laisse son bloc dans a_cu, on le republiait, le
 * pont le reemettait, et le BSC repondait
 *
 *   lchan(0-0-1-SDCCH8-0){ESTABLISHED}: ERROR INDICATION
 *     cause=SABM frame with information not allowed in this state
 *
 * -- un deuxieme SABM sur un lien deja etabli. Le canal tombait, le MSC
 * passait en MSC_A_ST_RELEASING et repondait LOCATION UPDATING REJECT au
 * milieu de la procedure, apres avoir pourtant mene l'IDENTITY REQUEST et
 * l'AUTHENTICATION REQUEST a bien.
 *
 * Donc : un bloc n'est publie QUE si son contenu change. MONTANT_SDCCH_REPETE
 * = N retablit une republication du meme bloc au bout de N trames (0 = jamais,
 * le defaut). Une retransmission LAPDm du mobile porte les memes octets et
 * serait donc avalee ; c'est le compromis assume, l'inverse casse le lien a
 * coup sur. */
#define SDCCH_UL_REPETE_DEFAUT 0

/* Nombre de trames minimum entre deux RACH publies : une tentative du mobile
 * dure plusieurs trames et le meme d_rach reste lisible entre-temps. */
#define RACH_GARDE_TRAMES 4
```

```
static struct {
    unsigned long rach, sdcch, facch, sacch, parole;
    uint16_t prev_rach;
    uint32_t fn_rach;
    bool     rach_vu;      /* au moins un RACH publie              */
    bool     base_rach;    /* la valeur de reference a ete prise   */
    /* Dernier bloc SDCCH montant publie, pour l'anti-doublon. */
    uint8_t  sdcch_dernier[23];
    uint32_t sdcch_trame;
    bool     sdcch_a_dernier;
    bool     blud_vu;          /* a_cu a deja annonce un bloc par B_BLUD   */
    unsigned sans_blud;        /* taches montantes vues sans B_BLUD        */
    bool     dedie_arme;
    /* [2026-09-23] Bascule SDCCH <-> TCH suivie par le firmware (voir
     * suivre_tache_tch). Le dernier SDCCH lu dans calypso_dcch_cfg est garde
     * pour y revenir ; l'annonce du TCH vient du pont (calypso_tch_cfg). */
    int      sd_tn, sd_genre, sd_ss;
    bool     sd_valide;        /* un SDCCH est connu (dcch_cfg arme)       */
    int      tch_tn, tch_tsc;  /* annonce du pont : 0 = pas de TCH annonce */
    bool     sur_tch;          /* le BSP joue l'intervalle du TCH          */
} g;
static int g_tn_tch_sb;   /* TS du TCH vu par le side-band (0 : TS0 / pas de TCH) */

/* Taches de lecture que l'ARM pose dans d_task_d de la page W
 * (osmocom-bb firmware, include/calypso/l1_environment.h:45-52). La lecture
 * d'un bloc SDCCH/SACCH passe par ALLC (prim_rx_nb.c:200), comme la BCCH et
 * la CCCH ; DDL/ADL sont gardees par prudence. */
#ifndef DDL_DSP_TASK
#define DDL_DSP_TASK 26
#endif
#ifndef ADL_DSP_TASK
#define ADL_DSP_TASK 27
#endif
/* d_tch_mode : quatrieme mot du NDB (dsp_api.h:121, rejouer.c:623). */
#define NDB_D_TCH_MODE 0x006u

static int journal(void)
{
    static int n = -1;
    if (n < 0) {
        const char *e = calypso_getenv("MONTANT_DEBUG");
        n = (e && *e) ? atoi(e) : 20;   /* les 20 premiers evenements, par defaut */
    }
    return n;
}

static int sb_ouvrir(const char *chemin, off_t taille)
{
    int fd = open(chemin, O_CREAT | O_RDWR, 0644);
    if (fd >= 0 && ftruncate(fd, taille) < 0) {
        close(fd);
        return -1;
    }
    return fd;
}

static void sb_ecrire(int fd, const void *buf, size_t n, off_t off)
{
    if (fd >= 0 && pwrite(fd, buf, n, off) < 0) {
        return;
    }
}

static void publier_l2(int *fdp, const char *chemin, uint32_t *seq,
                       const uint8_t *l2, uint16_t task_u, uint32_t fn)
{
```

```
    if (*fdp == -2) {
        *fdp = sb_ouvrir(chemin, REC_L2);
    }
    uint8_t buf[REC_L2];
    memset(buf, 0, sizeof(buf));
    (*seq)++;
    memcpy(buf + 0, seq, 4);
    memcpy(buf + 4, &fn, 4);
    memcpy(buf + 8, &fn, 4);
    memcpy(buf + 12, &task_u, 2);
    buf[14] = (uint8_t)(fn % 51u);
    memcpy(buf + 16, l2, 23);
    sb_ecrire(*fdp, buf, sizeof(buf), 0);
}

/* [2026-09-30] LA PAROLE MONTANTE, ENREGISTREE. Run banc-max de 00:07 : l'appel
 * s'etablit, mais le ton 1 kHz injecte au micro ne revient pas ; les trames
 * descendantes a_dd, bit-exactes avec ce que la BTS a emis
 * (tools/comparer_parole.py, 1116/1117), decodees par libgsm ne le contiennent
 * pas non plus : ce qu'Asterisk a renvoye etait deja du bruit, le defaut est
 * MONTANT. La conversion TI -> FR ci-dessous est l'inverse exact de gapk
 * (aller-retour identique au bit pres, verifie) et le firmware ecrit a_du en
 * octet fort d'abord comme prendre_ul le lit. Reste : la capture GAPK, le
 * firmware, ou le pont (codage, A5 montant). Pour trancher au prochain appel,
 * chaque trame montante est notee ici, brute TI (type 1) et convertie FR
 * (type 0), meme format de 48 octets que noter_dl : tools/decoder_add.py les
 * decode avec libgsm et cherche le ton. */
static void noter_ul(uint32_t fn, uint8_t type, uint16_t etat, const uint8_t *data, int n)
{
    static FILE *f;
    static unsigned nrec;
    static uint32_t fn_prec;
    if (!f || (uint32_t)(fn - fn_prec) > 500u) {
        if (f) fclose(f);
        f = fopen("/dev/shm/calypso_add_ul.bin", "wb");
        nrec = 0;
    }
    fn_prec = fn;
    if (!f || nrec >= 16000) return;
    uint8_t r[48];
    memset(r, 0, sizeof r);
    memcpy(r, &fn, 4);
    r[4] = type; r[5] = (uint8_t)n;
    memcpy(r + 6, &etat, 2);
    memcpy(r + 12, data, n > 36 ? 36 : n);
    fwrite(r, 1, sizeof r, f);
    if ((++nrec % 32) == 0) fflush(f);
}
static void publier_parole(const uint8_t *fr, uint32_t fn)
{
    static int fd = -2;
    static uint32_t seq;
    if (fd == -2) {
        fd = sb_ouvrir(SHM_TCH_UL, 8 + TCH_UL_SLOTS * TCH_UL_SLOT_SZ);
        uint32_t hdr[2] = { 0, TCH_UL_SLOTS };
        sb_ecrire(fd, hdr, sizeof(hdr), 0);
    }
    uint8_t buf[TCH_UL_SLOT_SZ];
    memset(buf, 0, sizeof(buf));
    seq++;
    memcpy(buf + 0, &seq, 4);
    memcpy(buf + 4, &fn, 4);
    memcpy(buf + 8, &fn, 4);
    memcpy(buf + TCH_UL_FR_OFS, fr, FR_BYTES);
    /* L'entete (le compteur d'ecriture) en dernier : pont.py lit d'abord
     * l'entete, puis la case ; l'inverse lui livrerait une case a moitie ecrite. */
```

```
    sb_ecrire(fd, buf, sizeof(buf), 8 + (off_t)((seq - 1) % TCH_UL_SLOTS) * TCH_UL_SLOT_SZ);
    sb_ecrire(fd, &seq, 4, 0);
}

/* Bloc montant depose par le firmware dans le NDB : mot 0 = en-tete (B_BLUD
 * signale « bloc pret »), donnees a partir du mot 3. Les 33 octets de parole
 * sont ranges octet fort d'abord, les 23 octets L2 octet faible d'abord.
 *
 * [2026-10-04] B_BLUD EST A LA ROM, PAS A L'HOTE. Cette fonction effacait le drapeau
 * « comme le ferait le DSP » -- mais AVANT lui. Pour a_cu (SDCCH, SACCH) la ROM n'en a
 * pas besoin : elle code le bloc des la tache DUL/AUL posee, et le drapeau tombe au tick
 * suivant (rejeu REJEU_BLUD : pose a T, efface par la ROM a T+1, bursts de T+1 a T+4).
 * Pour a_fu (FACCH) et a_du (parole), la tache TCHT s'en sert pour decider FACCH ou
 * parole, et la ROM ne le lit qu'au tick suivant : montant_scruter, en fin de tick T,
 * l'avait deja efface -> la ROM n'a jamais emis UN SEUL burst FACCH ni de parole
 * (appels de 18:30, 18:31, 18:59, 19:02 : la BTS ne recoit sur le TS2 que les bursts
 * SACCH, fn%26 = 12, pas de UA au SABM, retour sur le SDCCH apres 6 SABM ; le rejeu, qui
 * ne joue pas montant.c, montre les 8 bursts FACCH a fn%26 = 4..11). Le drapeau est donc
 * laisse a la ROM ; un bloc n'est publie qu'une fois par pose (le drapeau vu retombe entre
 * deux, ou contenu different). MONTANT_CONSOMME_BLUD=1 : ancien comportement. */
static bool prendre_ul(uint16_t *api_ram, unsigned off, uint8_t *out, int n)
{
    uint16_t *w = &api_ram[(API_NDB + off) / 2];
    static int consomme = -1;
    static struct { unsigned off; bool publie; uint8_t contenu[FR_BYTES]; } vu[4];
    static int nvu;
    if (consomme < 0) {
        const char *e = calypso_getenv("MONTANT_CONSOMME_BLUD");
        consomme = (e && *e == '1') ? 1 : 0;
        printf("  [montant] B_BLUD des tampons montants (a_cu, a_fu, a_du) %s\n",
               consomme ? "efface par l'hote apres lecture (MONTANT_CONSOMME_BLUD=1) : la ROM n'emet ni FACCH ni
 ↪  parole"
                        : "laisse a la ROM, qui le consomme (MONTANT_CONSOMME_BLUD=1 pour l'ancien comportement)");
    }
    int k;
    for (k = 0; k < nvu && vu[k].off != off; k++) { }
    if (k == nvu && nvu < 4) { vu[nvu].off = off; vu[nvu].publie = false; nvu++; }
    if (!(w[0] & B_BLUD)) {
        if (k < nvu) vu[k].publie = false;     /* la ROM a consomme : la prochaine pose se publie */
        return false;
    }
    for (int i = 0; i < n; i += 2) {
        uint16_t v = w[3 + i / 2];
        uint8_t premier = (n == FR_BYTES) ? (uint8_t)(v >> 8) : (uint8_t)(v & 0xff);
        uint8_t second  = (n == FR_BYTES) ? (uint8_t)(v & 0xff) : (uint8_t)(v >> 8);
        out[i] = premier;
        if (i + 1 < n) {
            out[i + 1] = second;
        }
    }
    if (consomme) {
        w[0] &= (uint16_t)~B_BLUD;
        return true;
    }
    if (k < nvu) {
        if (vu[k].publie && memcmp(vu[k].contenu, out, (size_t)n) == 0) return false;   /* deja publie, pas encore
         ↪  consomme */
        vu[k].publie = true;
        memcpy(vu[k].contenu, out, (size_t)n);
    }
    return true;
}

/* [2026-09-23] LA PAROLE MONTANTE EST AU FORMAT TI, LE PONT ATTEND DU FR STANDARD.
 * mobile_pont.cfg : io-tch-format ti. gapk convertit donc la voix du micro
 * au format du DSP TI (osmo-gapk fmt_ti.c, ti_fr_from_canon), le firmware la
```

```
 * copie telle quelle dans a_du, et le pont la passait a
 * gsm0503_tch_fr_encode(..., net_order=1), qui attend du FR TS 101 318 (le
 * format RTP « gsm », nibble 0xd en tete). L'ordre des 260 bits n'est pas le
 * meme : la BTS recevait une parole melangee, l'echo test la renvoyait et le
 * decodeur du mobile la rendait en bruit sature (descendant colle a +-12882).
 * Le descendant, lui, reste TI de bout en bout : il etait propre tant qu'il ne
 * portait pas la voix de l'operateur. Conversion reprise de fmt_ti.c
 * (ti_fr_to_canon) puis fmt_gsm.c (gsm_from_canon). MONTANT_PAROLE_TI=0 :
 * passage brut, comme avant. */
#include <osmocom/codec/codec.h>
static int bit_msb(const uint8_t *b, int i) { return (b[i >> 3] >> (7 - (i & 7))) & 1; }
static void mettre_bit_msb(uint8_t *b, int i, int v)
{
    if (v) b[i >> 3] |= (uint8_t)(0x80 >> (i & 7));
    else   b[i >> 3] &= (uint8_t)~(0x80 >> (i & 7));
}
static void parole_ti_vers_fr(uint8_t fr[FR_BYTES])
{
    static int conv = -1;
    if (conv < 0) { const char *e = calypso_getenv("MONTANT_PAROLE_TI"); conv = !(e && *e == '0'); }
    if (!conv) return;
    uint8_t canon[FR_BYTES];
    memset(canon, 0, sizeof canon);
    for (int i = 0; i < 260; i++) {                /* ti_fr_to_canon */
        int si = i >= 182 ? i + 4 : i;
        mettre_bit_msb(canon, gsm610_bitorder[i], bit_msb(fr, si));
    }
    fr[0] = (uint8_t)(0xd0 | (canon[0] >> 4));      /* gsm_from_canon : 0xd + 260 bits */
    for (int i = 1; i < FR_BYTES; i++)
        fr[i] = (uint8_t)((canon[i - 1] << 4) | (canon[i] >> 4));
}

static int fd_facch = -2;
static uint32_t seq_facch;

/* [2026-10-22 13:50] LA FACCH MONTANTE AU GO. L'ARM pose a_fu (B_BLUD) entre la phase A et le GO ; la ROM le
 * lit et efface B_BLUD des la phase B de la MEME trame, montant_scruter (fin de trame) le trouvait le plus
 * souvent deja retombe : appel de 13:34, firmware « DL: >N200 of SABM » (34 SABM poses), un seul bloc FACCH
 * publie (et pas le SABM) -> la BTS n'a jamais repondu UA. Appele par pont.c au GO, avant que la ROM tourne ;
 * l'anti-doublon de prendre_ul() empeche montant_scruter de republier le meme bloc. MONTANT_FU_AU_GO=0 coupe. */
void montant_capter_fu(uint16_t *api_ram, uint32_t fn)
{
    static int on = -1;
    if (on < 0) { const char *e = calypso_getenv("MONTANT_FU_AU_GO"); on = !(e && *e == '0'); }
    if (!on || !g.sur_tch) return;
    uint8_t l2[23];
    if (prendre_ul(api_ram, NDB_A_FU, l2, 23)) {
        publier_l2(&fd_facch, SHM_FACCH_UL, &seq_facch, l2, TCHT_DSP_TASK, fn);
        if (g.facch++ < (unsigned long)journal())
            printf("  [montant] FACCH UL fn=%u (au GO)\n", fn);
    }
    /* [2026-10-22] LA PAROLE MONTANTE AU GO. a_du (NDB_A_DU_1) souffre du meme mal que a_fu : la ROM
     * pose B_BLUD puis l'efface des la phase B de la trame ; capture_tch_ul (montant_scruter, fin de
     * trame) arrivait trop tard, prendre_ul() rendait false -> publier_parole() jamais appelee, SHM parole
     * absent, pont ul=0 (aucune voix montante). On la capte ici au GO, symetrique a la FACCH ci-dessus.
     * L'anti-doublon de prendre_ul() (dedup par contenu, vu[]) empeche le double envoi quand
     * capture_tch_ul la retente en fin de trame. MONTANT_DU_AU_GO=0 coupe. */
    static int du = -1;
    if (du < 0) { const char *e = calypso_getenv("MONTANT_DU_AU_GO"); du = !(e && *e == '0'); }
    if (du) {
        uint8_t fr[FR_BYTES];
        if (prendre_ul(api_ram, NDB_A_DU_1, fr, FR_BYTES)) {
            noter_ul(fn, 1, TCHT_DSP_TASK, fr, FR_BYTES);   /* brute, format TI (a_du) */
            parole_ti_vers_fr(fr);
            noter_ul(fn, 0, TCHT_DSP_TASK, fr, FR_BYTES);   /* convertie FR, ce que recoit le pont */
            publier_parole(fr, fn);
```

```
            if (g.parole++ < (unsigned long)journal())
                printf("  [montant] parole UL fn=%u (au GO)\n", fn);
        }
    }
}

static bool capture_tch_ul(uint16_t *api_ram, uint16_t task_u, uint32_t fn)
{
    static int fd_sacch = -2;
    static uint32_t seq_sacch;
    uint8_t l2[23], fr[FR_BYTES];

    switch (task_u & 0x7FFF) {
    case TCHT_DSP_TASK:
        if (prendre_ul(api_ram, NDB_A_FU, l2, 23)) {
            publier_l2(&fd_facch, SHM_FACCH_UL, &seq_facch, l2, task_u, fn);
            if (g.facch++ < (unsigned long)journal())
                printf("  [montant] FACCH UL fn=%u task=0x%04x\n", fn, task_u);
        }
        if (prendre_ul(api_ram, NDB_A_DU_1, fr, FR_BYTES)) {
            noter_ul(fn, 1, task_u, fr, FR_BYTES);      /* brute, format TI (a_du) */
            parole_ti_vers_fr(fr);
            noter_ul(fn, 0, task_u, fr, FR_BYTES);      /* convertie FR, ce que recoit le pont */
            publier_parole(fr, fn);
            if (g.parole++ < (unsigned long)journal())
                printf("  [montant] parole UL fn=%u\n", fn);
        }
        return true;
    case TCHA_DSP_TASK:
        if (prendre_ul(api_ram, NDB_A_CU, l2, 23)) {
            publier_l2(&fd_sacch, SHM_SACCH_UL, &seq_sacch, l2, task_u, fn);
            if (g.sacch++ < (unsigned long)journal())
                printf("  [montant] SACCH UL fn=%u task=0x%04x\n", fn, task_u);
        }
        return true;
    case TCHD_DSP_TASK:
        return true;
    default:
        return false;
    }
}

/* SDCCH / SACCH montant.
 *
 * [2026-09-21] Le firmware ANNONCE son bloc, il n'y a rien a deviner :
 * prim_tx_nb.c:80-101 ecrit dans a_cu l'en-tete `(1 << B_BLUD)`, deux mots a
 * zero, puis les 23 octets L2 a partir du mot 3 -- la disposition exacte que
 * prendre_ul() sait lire. Le drapeau est a usage unique : on le consomme, et
 * un bloc = une publication.
 *
 * Avant d'avoir lu ce code, cette fonction reprenait la fenetre heuristique de
 * la couche 1 gr-gsm (balayage d'en-tete LAPDm dans a_cu+6) avec un anti-
 * doublon sur le contenu. Trois echecs de suite en sont sortis : republication
 * du meme SABM toutes les 60 trames -> « SABM frame with information not
 * allowed in this state » et canal casse en pleine procedure ; puis
 * comparaison sur 23 octets dont la friture de fin bouge -> 32 blocs
 * « neufs » ; puis verrou « un seul SABM par connexion » -> plus aucun SABM
 * des que le verrou restait arme. Le drapeau du firmware rend tout ca inutile.
 *
 * MONTANT_SDCCH_FENETRE=1 force l'ancienne voie heuristique, et elle prend le
 * relais toute seule si B_BLUD ne se leve jamais alors que le firmware pose
 * des taches montantes (le cas ou la ROM consommerait le drapeau avant nous).
 */
/* PEREMPTION DE L'ANTI-DOUBLON (voie heuristique seulement).
 *
 * [2026-09-21] La couche 1 gr-gsm republie un bloc identique passe 60 ticks
```

```
 * (calypso_l1_grgsm.c:673). Ce n'est PAS un moteur de renvoi : dans ce
 * montage-la, QEMU efface d_task_u a chaque tick (calypso_trx.c, branche non
 * pont), donc la couche 1 ne voit une tache montante que sur les trames ou le
 * firmware vient de la poser. Les 60 ticks ne font qu'empecher de publier
 * quatre fois le meme bloc (un par burst) tout en laissant passer une VRAIE
 * retransmission du mobile, quand son T200 le fait re-poster.
 *
 * Sur la voie B_BLUD ce probleme n'existe pas : le drapeau est a usage unique,
 * une pose = une publication, et une retransmission du mobile repose le
 * drapeau. Rien a temporiser.
 *
 * Avoir lu ces 60 ticks comme un renvoi a coute un banc : le SABM repartait
 * apres l'etablissement du lien, et le BTS repondait « SABM frame with
 * information not allowed in this state » -- 4 ERROR INDICATION pour 4
 * ESTABLISHED. */
#define SDCCH_TTL_DEFAUT      60   /* trames, comme SDCCH_UL_DEDUP_TICKS */

static int sabm_ttl(void)
{
    static int v = -1;
    if (v < 0) {
        const char *e = calypso_getenv("MONTANT_SDCCH_TTL");
        if (!e || !*e) {
            e = calypso_getenv("MONTANT_SDCCH_REPETE");   /* ancien nom */
        }
        v = (e && *e) ? atoi(e) : SDCCH_TTL_DEFAUT;
    }
    return v;
}

static void publier_sdcch(uint16_t task_u, uint32_t fn, const uint8_t *l2)
{
    static int fd = -2;
    static uint32_t seq;
    if (l2[1] == 0x03) {      /* trame vide (UI sans donnee) */
        return;
    }
    publier_l2(&fd, SHM_SDCCH_UL, &seq, l2, task_u, fn);
    if (g.sdcch++ < (unsigned long)journal()) {
        printf("  [montant] SDCCH UL fn=%u task=0x%04x L2=", fn, task_u);
        for (int k = 0; k < 23; k++) printf("%02x%s", l2[k], k == 22 ? "" : " ");
        printf("\n");
    }
}

/* L'ancienne voie : balayage de la fenetre, anti-doublon sur le contenu
 * utile, republication apres MONTANT_SDCCH_REPETE trames (0 = jamais). */
static void capture_sdcch_fenetre(uint16_t *api_ram, uint16_t task_u, uint32_t fn)
{
    uint8_t fen[30];
    const uint8_t *src = (const uint8_t *)api_ram + API_NDB + NDB_A_CU + SDCCH_UL_WINDOW_OFS;
    memcpy(fen, src, sizeof(fen));

    int kk = 0;
    for (int j = 0; j <= 6; j++) {
        uint8_t a = fen[j], c = fen[j + 1], l = fen[j + 2];
        int sapi = (a >> 2) & 7;
        bool addr_ok = (a & 0x01) && ((a & 0x60) == 0) && (sapi == 0 || sapi == 3);
        bool ctrl_ok = (c != 0x2b) && (c != 0xff);
        bool len_ok = (l & 0x01) && ((l >> 2) <= 20);
        if (addr_ok && ctrl_ok && len_ok) {
            kk = j;
            break;
        }
    }
    const uint8_t *l2 = fen + kk;
```

```
    int repete = sabm_ttl();
    unsigned utile = 3u + (unsigned)(l2[2] >> 2);
    if (utile > 23u) {
        utile = 23u;
    }
    if (g.sdcch_a_dernier && !memcmp(g.sdcch_dernier, l2, utile) &&
        (repete <= 0 || (uint32_t)(fn - g.sdcch_trame) < (uint32_t)repete)) {
        return;
    }
    memcpy(g.sdcch_dernier, l2, 23);
    g.sdcch_trame = fn;
    g.sdcch_a_dernier = true;
    publier_sdcch(task_u, fn, l2);
}

static void capture_sdcch_ul(uint16_t *api_ram, uint16_t task_u, uint32_t fn)
{
    static int fenetre = -1;
    if (fenetre < 0) {
        const char *e = calypso_getenv("MONTANT_SDCCH_FENETRE");
        fenetre = (e && *e == '1') ? 1 : 0;
    }
    if (!fenetre) {
        uint8_t l2[23];
        if (prendre_ul(api_ram, NDB_A_CU, l2, 23)) {
            g.blud_vu = true;
            publier_sdcch(task_u, fn, l2);
            return;
        }
        if (g.blud_vu) {
            return;               /* le drapeau fonctionne : rien a publier */
        }
        /* Jamais vu B_BLUD alors que le firmware pose des taches montantes :
         * la ROM le consomme peut-etre avant nous. On bascule sur la fenetre. */
        if (++g.sans_blud == 400) {
            printf("  [montant] a_cu : B_BLUD jamais vu en %u taches montantes, "
                   "bascule sur la fenetre heuristique\n", g.sans_blud);
        }
        if (g.sans_blud < 400) {
            return;
        }
    }
    capture_sdcch_fenetre(api_ram, task_u, fn);
}

/* Le canal dedie, lu directement dans le side-band que le tap L1CTL de QEMU
 * ecrit (calypso_dcch_tap.c) et que pont.py lit deja.
 *
 * [2026-09-21, mesure] Le canal etait annonce au DSP par un message du pont,
 * PONT_DCCH. Trace du banc : QEMU imprime bien « [dcch] canal dedie arme :
 * chan_nr=0x51 SDCCH/8 SS=2 TN=1 », et cote DSP, RIEN -- ni « pont : canal
 * dedie », ni « [BSP] canal dedie arme », ni message inconnu. Le message se
 * perd dans le pas-a-pas en deux phases (la boucle d'attente du PONT_GO jette
 * tout ce qui n'est pas un GO). Resultat : le BSP continuait de livrer TS0
 * pendant que l'ARM ecoutait TS1, et TOUS les blocs de la descente dediee
 * echouaient au code de Fire.
 *
 * Le fichier, lui, ne depend d'aucun protocole : une lecture de 8 octets par
 * trame, et le meme numero de sequence que pont.py utilise pour savoir s'il a
 * change. */
static void scruter_dcch(uint32_t fn)
{
    static int fd = -2;
    static uint32_t seq;
    static uint32_t dernier_essai = UINT32_MAX;
```

```
    if (fd < 0) {
        /* [2026-10-03] Un essai d'ouverture par trame au plus. L'ancienne garde `fn < prochain_essai`
         * n'etait pas modulaire : apres le bouclage de l'hypertrame (~3 h 29) elle bloquait toute
         * reouverture jusqu'a ce que fn rattrape l'ancienne valeur (revue de code 2026-10-03). */
        if (fn == dernier_essai) {
            return;
        }
        /* [2026-09-23] A CHAQUE TRAME, plus toutes les 200. run.sh efface les
         * side-bands au demarrage : le fichier n'apparait qu'a la premiere
         * ecriture (tap QEMU ici, pont pour calypso_tch_cfg), et on le voyait
         * jusqu'a 200 trames trop tard. Pour calypso_tch_cfg c'etait fatal au
         * premier appel apres chaque relance : le firmware posait sa tache TCH
         * 113 a 164 trames avant que l'annonce soit lue (« sans annonce du
         * pont »), l'UA de l'assignation se perdait, ASSIGNMENT FAILURE. Un
         * open() qui echoue coute une microseconde. */
        dernier_essai = fn;
        fd = open("/dev/shm/calypso_dcch_cfg", O_RDONLY);
        if (fd < 0) {
            return;
        }
    }
    uint8_t b[16];
    if (pread(fd, b, sizeof(b), 0) != (ssize_t)sizeof(b)) {
        return;
    }
    uint32_t s2;
    memcpy(&s2, b, 4);
    if (!s2 || s2 == seq) {
        return;
    }
    seq = s2;
    int genre = b[4], ss = b[5], tn = b[6];
    if (genre == 4) {
        /* [2026-10-15] PDCH (GPRS, firmware l1gprs_fw.c, tap calypso_dcch_tap.c) : le BSP
         * joue tout le TN, comme un TCH ; pas de SDCCH memorise, pas de bascule TCH. */
        printf("  [montant] canal dedie (side-band seq=%u) : PDCH TS%d (GPRS), toutes les trames au BSP\n", seq,
↪   tn);
        fflush(stdout);
        calypso_bsp_set_dedie(tn, 4, 0);
        g.dedie_arme = true;
        return;
    }
    /* [2026-09-23] Genre 2/3 : le tap QEMU (calypso_dcch_tap.c) annonce
     * desormais le TCH (TCH/F, TCH/H) au lieu de laisser le SDCCH perime
     * pendant tout l'appel. Simple constat ici : la bascule du BSP suit la
     * tache du firmware (suivre_tache_tch), l'intervalle et le TSC viennent
     * du pont (scruter_tch), et le SDCCH memorise ne doit PAS etre ecrase --
     * c'est lui que l'ASSIGNMENT FAILURE retrouve (le tap le republie quand
     * le firmware y revient). */
    if (genre == 2 || genre == 3) {
        g_tn_tch_sb = tn;   /* [2026-10-20] le TS du TCH, pour etiqueter les bursts montants publies (air) */
        /* [2026-10-21] le SDCCH est chiffre (A5/1) : ni le pont ni l'air ne lisent plus l'ASSIGNMENT COMMAND,
         * calypso_tch_cfg ne vient jamais. Le tap QEMU, lui, voit le TS que le firmware programme : on le prend
         * comme intervalle du TCH quand personne ne l'a annonce ; suivre_tache_tch bascule le BSP a la tache
↪   suivante. */
        if (!g.tch_tn && tn >= 1 && tn <= 7) {
            g.tch_tn = tn;
           printf("  [montant] TCH : intervalle TS%d pris sur le side-band (pas d'annonce calypso_tch_cfg)\n", tn);
        }
        printf("  [montant] canal dedie (side-band seq=%u) : TCH/%c TS%d SS=%d, firmware sur le TCH "
               "(%s%s)\n", seq, genre == 2 ? 'F' : 'H', tn, ss,
               g.sur_tch ? "BSP deja sur le TCH" : "BSP pas encore bascule",
               g.sd_valide ? ", SDCCH memorise" : "");
        fflush(stdout);
        return;
    }
```

```
    printf("  [montant] canal dedie (side-band seq=%u) : %s TS%d SDCCH/%d SS=%d%s\n",
           seq, genre == 0xFF ? "libere" : "arme", tn, genre == 1 ? 8 : 4, ss,
           (genre != 0xFF && g.sur_tch) ? " (memorise : le firmware est sur le TCH)" : "");
    g_tn_tch_sb = 0;   /* SDCCH (TS0) arme ou canal libere : le mobile n'emet plus sur le TCH */
    if (genre == 0xFF) {
        calypso_bsp_set_dedie(0, 0xFF, 0);
        g.dedie_arme = false;
        g.sd_valide = false;
        g.sur_tch = false;
        montant_canal_libere();
        return;
    }
    /* [2026-09-23] Le SDCCH est MEMORISE : c'est lui que le BSP retrouve quand
     * le firmware quitte le TCH (ASSIGNMENT FAILURE). Tant que le firmware est
     * sur le TCH, on ne touche pas au BSP. */
    g.sd_tn = tn;
    g.sd_genre = genre;
    g.sd_ss = ss;
    g.sd_valide = true;
    if (g.sur_tch) {
        return;
    }
    calypso_bsp_set_dedie(tn, genre, ss);
    g.dedie_arme = true;
}

/* Rendre au BSP le SDCCH memorise (ou le liberer s'il n'y en a pas). */
static void revenir_sdcch(const char *raison, uint32_t fn)
{
    g.sur_tch = false;
    if (g.sd_valide) {
        printf("  [montant] %s a fn=%u : BSP rendu au SDCCH TS%d SS=%d\n",
               raison, fn, g.sd_tn, g.sd_ss);
        calypso_bsp_set_dedie(g.sd_tn, g.sd_genre, g.sd_ss);
        g.dedie_arme = true;
    } else {
        printf("  [montant] %s a fn=%u : aucun SDCCH connu, canal dedie libere\n",
               raison, fn);
        calypso_bsp_set_dedie(0, 0xFF, 0);
        g.dedie_arme = false;
    }
    fflush(stdout);
}

/* [2026-09-23] 0 : armer le BSP des l'annonce du pont (comportement du
 * 2026-09-22). Voir scruter_tch. */
static bool tch_sur_tache(void)
{
    static int v = -1;
    if (v < 0) {
        const char *e = calypso_getenv("MONTANT_TCH_TACHE");
        v = (e && *e == '0') ? 0 : 1;
    }
    return v != 0;
}

/* [2026-09-22] L'INTERVALLE DU TCH, CHAINON QUI MANQUAIT.
 *
 * pont.py publie deja l'intervalle du canal de trafic dans
 * /dev/shm/calypso_tch_cfg (pont/state.py, Tch._write_cfg : seq, tn, tsc,
 * arfcn) des qu'il decode l'ASSIGNMENT COMMAND descendante. Mais PERSONNE ne
 * lisait ce fichier cote DSP : `calypso_bsp_set_dedie()` n'etait appelee que
 * depuis la bande laterale SDCCH ci-dessus. Le BSP continuait donc de servir
 * l'intervalle SDCCH pendant que le mobile ecoutait le TCH.
 *
 * Mesure du 2026-09-22, appel vers 600 : le BSC assigne un TCH/F sur TS2, le
```

```
 * mobile y bascule (« MON: ... TS=2 »), n'y entend RIEN -- trois secondes de
 * « MON: no cell info » avec rxlev-full=-110, le plancher -- revient sur TS1
 * et repond « ASSIGNMENT FAILURE (cause #1) » (gsm48_rr.c:4750). Cote BSC :
 * « Assignment failed in state WAIT_RR_ASS_COMPLETE, cause EQUIPMENT FAILURE:
 * Timeout ». La trace DSP ne montrait que « arme TS1 SDCCH/8 », jamais TS2.
 *
 * Ce n'etait donc pas un defaut de qualite du lien mais un chainon manquant.
 * MONTANT_TCH=0 coupe cette lecture.
 *
 * [2026-09-23] L'ANNONCE N'EST PAS LA BASCULE.
 * Le pont ecrit ce fichier au DECODAGE de l'ASSIGNMENT COMMAND, donc a
 * l'heure de la BTS, alors que le BSP joue les trames a l'heure du DSP -- en
 * retard de 12 a ~200 trames (decalage tick/fn BTS de 681 au run de 12:22).
 * Armer le BSP ici remplacait par des bursts TS2 TOUTES les trames pas
 * encore jouees (bsp_dedie_trame est vrai partout pour un TCH), l'ASSIGNMENT
 * COMMAND comprise, et TS0 avec. Releve du run de 12:22, appels 2 et 3 :
 * « TCH (seq=2) : arme TS2 » puis un seul a_cd « FIRE KO » et plus aucun
 * bloc SDCCH ; cote mobile 80-100 bit errors, jamais d'ASSIGNMENT COMMAND,
 * LOS apres ~20 s. Et apres une ASSIGNMENT FAILURE (appel 1) rien ne rendait
 * TS1 au BSP : 90-110 bit errors sur le SDCCH jusqu'a la liberation.
 *
 * Desormais ce fichier n'est plus qu'une ANNONCE (intervalle, TSC). La
 * bascule vient du firmware lui-meme : suivre_tache_tch() arme le TCH a la
 * premiere tache TCHT/TCHA/TCHD qu'il pose, et rend le SDCCH des qu'il repose
 * une tache de lecture de bloc (ALLC). Un seq non nul avec tn=0 est l'abandon
 * du pont (pont/dsp/tch.py, TchDsp.abandon) : retour au SDCCH sans lacher le
 * Kc. seq=0 reste la liberation complete.
 * MONTANT_TCH_TACHE=0 retablit l'armement a l'annonce. */
static void scruter_tch(uint32_t fn)
{
    static int fd = -1, coupe = -1;
    static uint32_t seq, dernier_essai = UINT32_MAX;

    if (coupe < 0) {
        const char *e = calypso_getenv("MONTANT_TCH");
        coupe = (e && *e == '0') ? 1 : 0;
    }
    if (coupe) {
        return;
    }
    if (fd < 0) {
        if (fn == dernier_essai) {          /* voir scruter_dcch */
            return;
        }
        dernier_essai = fn;
        fd = open("/dev/shm/calypso_tch_cfg", O_RDONLY);
        if (fd < 0) {
            return;
        }
    }
    uint8_t b[16];
    if (pread(fd, b, sizeof(b), 0) != (ssize_t)sizeof(b)) {
        return;
    }
    uint32_t s2;
    memcpy(&s2, b, 4);
    if (s2 == seq) {
        return;
    }
    seq = s2;
    int tn = b[4], tsc = b[5];
    if (!s2) {
        /* Liberation complete (Tch.close) : comme avant le 2026-09-23. */
        int etait = g.tch_tn;
        g.tch_tn = 0;
        if (g.sur_tch) {
```

```
            printf("  [montant] TCH (seq=0) : libere TS%d\n", etait);
            calypso_bsp_set_dedie(0, 0xFF, 0);
            g.sur_tch = false;
            g.dedie_arme = false;
            montant_canal_libere();
        } else if (etait) {
            printf("  [montant] TCH (seq=0) : annonce TS%d retiree (le firmware n'y etait pas)\n", etait);
        }
        return;
    }
    if (tn <= 0 || tn > 7) {
        /* Abandon par le pont (ASSIGNMENT FAILURE, retour du mobile sur le
         * SDCCH) : le SDCCH vit encore, le Kc aussi. */
        int etait = g.tch_tn;
        g.tch_tn = 0;
        printf("  [montant] TCH (seq=%u) : annonce TS%d abandonnee par le pont\n", seq, etait);
        if (g.sur_tch) {
            revenir_sdcch("abandon du TCH", fn);
        }
        return;
    }
    g.tch_tn = tn;
    g.tch_tsc = tsc;
    if (!tch_sur_tache()) {
        printf("  [montant] TCH (seq=%u) : arme TS%d TSC=%d - toutes ses trames (MONTANT_TCH_TACHE=0)\n",
               seq, tn, tsc);
        calypso_bsp_set_dedie(tn, 2 /* BSP_DEDIE_TCH */, 0);
        g.sur_tch = true;
        g.dedie_arme = true;
        return;
    }
    printf("  [montant] TCH (seq=%u) : TS%d TSC=%d annonce par le pont, "
           "bascule du BSP a la premiere tache TCH du firmware\n", seq, tn, tsc);
    fflush(stdout);
}

/* [2026-09-23] LA BASCULE SDCCH <-> TCH SUIT LE FIRMWARE.
 *
 * d_task_d de la page W que l'ARM vient d'ecrire dit ce qu'il ecoute a la
 * trame suivante : TCHT (13, trafic et FACCH), TCHA (14, SACCH/T) ou TCHD
 * (28) sur un TCH (prim_tch.c:520-585, 791-802), ALLC (24) pour un bloc
 * SDCCH/SACCH (prim_rx_nb.c:200). Cette tache est posee au rythme du DSP :
 * le BSP change donc d'intervalle sur la trame que l'ARM lit vraiment, quel
 * que soit le retard du DSP sur la BTS. Les autres trames (PM des voisines,
 * trame libre) ne changent rien. */
/* [2026-09-23] SONDE d_fn : la position que le firmware donne au DSP dans la
 * 104-multitrame (dsp.c:537, d_fn = fn_report | (fn%104) << 8), sur les taches
 * TCH. A comparer au fn BTS que le BSP joue au meme tick ([sacch_tf]) : un
 * ecart multiple de 26 laisse passer parole et FACCH mais desentrelace la
 * SACCH dans le desordre (a_cd FIRE KO a chaque bloc, LOS). 24 lignes. */
static void sonde_dfn(uint16_t *api_ram, unsigned pg, bool taches, uint32_t fn)
{
    static unsigned n;
    if (!taches || !g.sur_tch || n >= 24) {
        return;
    }
    uint16_t t = api_ram[(API_W_PAGE(pg) + WP_D_TASK_D) / 2] & 0x7fffu;
    uint16_t dfn = api_ram[(API_W_PAGE(pg) + WP_D_FN) / 2];
    if (t != TCHA_DSP_TASK) {
        return;
    }
    n++;
    printf("  [d_fn] tick=%u tache=%u fn_report=%u fn%%104=%u\n",
           fn, t, dfn & 0xffu, (dfn >> 8) & 0xffu);
}
```

```
static void suivre_tache_tch(uint16_t *api_ram, unsigned pg, bool taches, uint32_t fn)
{
    static bool sans_annonce_dit;
    if (!taches || !tch_sur_tache()) {
        return;
    }
    uint16_t t = api_ram[(API_W_PAGE(pg) + WP_D_TASK_D) / 2] & 0x7fffu;
    bool tache_tch = (t == TCHT_DSP_TASK || t == TCHA_DSP_TASK || t == TCHD_DSP_TASK);
    bool tache_bloc = (t == ALLC_DSP_TASK || t == DDL_DSP_TASK || t == ADL_DSP_TASK);

    if (tache_tch && !g.sur_tch) {
        if (!g.tch_tn) {
            /* [2026-10-21] pas d'annonce (SDCCH chiffre A5/1 : l'air ne lit plus l'ASSIGNMENT COMMAND) :
             * l'intervalle est celui de la fenetre RX que le firmware a programmee au TPU. */
            int tn_tpu = calypso_bsp_fenetre_tn();
            if (tn_tpu >= 1 && tn_tpu <= 7) {
                g.tch_tn = tn_tpu;
                printf("  [montant] TCH : tache %u postee a fn=%u sans annonce (calypso_tch_cfg) : "
                       "intervalle TS%d lu sur la fenetre TPU\n", t, fn, tn_tpu);
            } else {
                if (!sans_annonce_dit) {
                    sans_annonce_dit = true;
                    printf("  [montant] TCH : tache %u postee a fn=%u sans annonce du pont "
                           "(calypso_tch_cfg) : intervalle inconnu (TPU %d), BSP inchange\n", t, fn, tn_tpu);
                }
                return;
            }
        }
        sans_annonce_dit = false;
        printf("  [montant] TCH : le firmware poste la tache %u a fn=%u, BSP bascule sur TS%d (TSC=%d)\n",
               t, fn, g.tch_tn, g.tch_tsc);
        fflush(stdout);
        calypso_bsp_set_dedie(g.tch_tn, 2 /* BSP_DEDIE_TCH */, 0);
        g.sur_tch = true;
        g.dedie_arme = true;
        return;
    }
    /* [2026-10-22 21:50] FIRMWARE TI : ALLC APRES LE TCH N'EST PAS FORCEMENT LE SDCCH. La L1 TI lit aussi la BCCH
     ↪  et
     * le paging en ALLC : apres le CHANNEL RELEASE d'un appel (run de 21:42), « revenu sur le SDCCH » rendait le
↪  BSP
     * au SDCCH/8 TS1, qui remplace TS0 sur toutes les trames tant que le dedie TI dure (CALYPSO_BSP_SDCCH8_TOUT) :
     * NP_I tous en erreur des la sortie du dedie, RECOVERY, DSP relance en 2,4 s, cellule perdue. Ici le BSP
     * revient d'abord a TS0 ; on ne retourne au SDCCH que si une tache MONTANTE suit dans les 102 trames (echec
     * d'assignation : la L1 renvoie sur son SDCCH) ; en veille il n'y en a pas. MONTANT_TI_ALLC_DIRECT=1 :
     * ancien comportement. */
    static int ti = -1, direct = -1;
    static bool attente_ul; static uint32_t attente_fn;
    if (ti < 0) { ti = calypso_getenv("MS_MAGNETITE") != NULL;
                  const char *e = calypso_getenv("MONTANT_TI_ALLC_DIRECT"); direct = (e && *e == '1'); }
    if (tache_bloc && g.sur_tch) {
        if (!ti || direct) {
            revenir_sdcch("le firmware est revenu sur le SDCCH (tache ALLC)", fn);
        } else {
            g.sur_tch = false;
            g.dedie_arme = false;
            calypso_bsp_set_dedie(0, 0xFF, 0);
            attente_ul = g.sd_valide;
            attente_fn = fn;
            printf("  [montant] firmware TI : tache ALLC apres le TCH a fn=%u : BSP rendu a TS0, retour au SDCCH "
                   "seulement sur une tache montante%s\n", fn, g.sd_valide ? "" : " (aucun SDCCH connu)");
            fflush(stdout);
        }
        return;
    }
    if (attente_ul) {
```

```
        uint16_t u = api_ram[(API_W_PAGE(pg) + WP_D_TASK_U) / 2] & 0x7fffu;
        if (u) {
            attente_ul = false;
            revenir_sdcch("firmware TI : tache montante apres le TCH (echec d'assignation)", fn);
        } else if ((uint32_t)(fn - attente_fn) > 102u) {
            attente_ul = false;
            printf("  [montant] firmware TI : aucune tache montante 102 trames apres le TCH (fn=%u) : en
↪  veille\n", fn);
        }
    }
}

/* [2026-09-23] SONDE a_fd : LA FACCH DESCENDANTE SUR TCH.
 * Le firmware ne remonte une FACCH que si ((fn%13)%4)==3 ET a_fd[0] porte
 * B_BLUD (prim_tch.c:235-246), puis remet a_fd[0] = B_FIRE1 : seul le DSP
 * pose B_BLUD. Sur l'appel 1 du run de 12:22 le pont decodait 6 FACCH de la
 * BTS (UA) et le mobile n'en recevait aucune (T200, MDL-ERROR cause 1). Cette
 * sonde tranche entre ROM (B_BLUD jamais pose) et plomberie (pose mais
 * FIRE). MONTANT_AFD=0 la coupe. */
/* [2026-09-23] TRAMES DESCENDANTES LIVREES PAR LA ROM, POUR LA COMPARAISON.
 * Chaque parole (a_dd_0, convertie du format TI au FR standard comme le
 * montant) et chaque FACCH (a_fd) vue avec B_BLUD, avec le tick, le mot
 * d'etat et le nombre d'erreurs rapporte (mot 2). Enregistrements de 48
 * octets : fn LE32, type (0 parole, 1 FACCH), n, etat LE16, err LE16, 2 octets
 * de bourrage, 36 octets de donnees. Fichier remis a zero a chaque appel (plus
 * de 500 trames sans rien), 8000 enregistrements au plus.
 * tools/comparer_parole.py le met en regard de /dev/shm/calypso_tch_dl.bin
 * (bursts du BSP) et de /dev/shm/pont_tch_dl.bin (decodage du pont). */
static void noter_dl(uint32_t fn, uint8_t type, uint16_t etat, uint16_t err,
                     const uint8_t *data, int n)
{
    static FILE *f;
    static unsigned nrec;
    static uint32_t fn_prec;
    if (!f || (uint32_t)(fn - fn_prec) > 500u) {
        if (f) fclose(f);
        f = fopen("/dev/shm/calypso_add_dl.bin", "wb");
        nrec = 0;
    }
    fn_prec = fn;
    if (!f || nrec >= 8000) return;
    uint8_t r[48];
    memset(r, 0, sizeof r);
    memcpy(r, &fn, 4);
    r[4] = type; r[5] = (uint8_t)n;
    memcpy(r + 6, &etat, 2);
    memcpy(r + 8, &err, 2);
    memcpy(r + 12, data, n > 36 ? 36 : n);
    fwrite(r, 1, sizeof r, f);
    if ((++nrec % 32) == 0) fflush(f);
}

static void sonde_afd(uint16_t *api_ram, uint32_t fn)
{
    static int sonde = -1;
    static uint16_t prec;
    static unsigned long ok, ko;
    if (sonde < 0) {
        const char *e = calypso_getenv("MONTANT_AFD");
        sonde = (e && *e == '0') ? 0 : 1;
    }
    if (!sonde || !g.sur_tch) {
        return;
    }
    uint16_t etat = api_ram[(API_NDB + NDB_A_FD) / 2];
    if ((etat & B_BLUD) && etat != prec) {
```

```
        const uint8_t *d = (const uint8_t *)api_ram + API_NDB + NDB_A_FD + 6;
        uint16_t mode = api_ram[(API_NDB + NDB_D_TCH_MODE) / 2];
        bool fire = (etat & 0x0040) != 0;          /* B_FIRE1 */
        if (fire) ko++; else ok++;
        noter_dl(fn, 1, etat, api_ram[(API_NDB + NDB_A_FD) / 2 + 2], d, 23);
        if ((ok + ko) <= 60 || ((ok + ko) % 50) == 0) {
            printf("  [a_fd] fn=%u fn%%13=%u etat=%04x BLUD=1 FIRE=%d d_tch_mode=%04x "
                   "L2=%02x %02x %02x %02x | ok=%lu ko=%lu\n",
                   fn, fn % 13u, etat, fire ? 1 : 0, mode, d[0], d[1], d[2], d[3], ok, ko);
        }
    }
    prec = etat;
}

/* [2026-09-23] SONDE a_dd : LA PAROLE DESCENDANTE SUR TCH.
 * Le firmware ne remonte une trame de parole au mobile (L1CTL_TRAFFIC_IND) que
 * si a_dd_0[0] porte B_BLUD (prim_tch.c:322-327) ; GAPK ne code le montant
 * qu'en reponse. Appel de 15:02 : 19 trames « parole UL » puis plus rien, alors
 * que le pont decodait la parole de la BTS. Cette sonde dit si le DSP livre
 * encore la parole descendante. Une ligne pour les 20 premieres, puis toutes
 * les 100. MONTANT_ADD=0 la coupe.
 *
 * [2026-09-23] ko ne voyait rien sur la parole : 0x0040 est B_FIRE1, et le code
 * de Fire ne protege que les canaux de controle. Run de 20:22 : 40 etats releves
 * (c214, c204, 8084), ko=0, mais tous avec le bit 2 = B_BFI (l1_environment.h:272).
 * Le firmware ne teste que B_BLUD (prim_tch.c:327) et ne remonte pas le BFI :
 * GAPK decode les 33 octets tels quels. La sonde compte donc aussi le BFI et
 * affiche a_dd_0[2], le nombre d'erreurs que la ROM rapporte (num_biterr cote
 * firmware). */
static void sonde_add(uint16_t *api_ram, uint32_t fn)
{
    static int sonde = -1;
    static uint16_t prec;
    static unsigned long vues, ko, bfi;
    static uint32_t fn_dernier;
    if (sonde < 0) {
        const char *e = calypso_getenv("MONTANT_ADD");
        sonde = (e && *e == '0') ? 0 : 1;
    }
    if (!sonde || !g.sur_tch) {
        return;
    }
    uint16_t etat = api_ram[(API_NDB + NDB_A_DD_0) / 2];
    if ((etat & B_BLUD) && etat != prec) {
        bool fire = (etat & 0x0040) != 0;
        bool mauvaise = (etat & 0x0004) != 0;      /* B_BFI */
        uint16_t erreurs = api_ram[(API_NDB + NDB_A_DD_0) / 2 + 2];
        {
            const uint16_t *w = &api_ram[(API_NDB + NDB_A_DD_0) / 2];
            uint8_t fr[FR_BYTES];
            for (int k = 0; k < FR_BYTES; k += 2) {       /* octet fort d'abord, comme prendre_ul */
                fr[k] = (uint8_t)(w[3 + k / 2] >> 8);
                if (k + 1 < FR_BYTES) fr[k + 1] = (uint8_t)(w[3 + k / 2] & 0xff);
            }
            parole_ti_vers_fr(fr);
            noter_dl(fn, 0, etat, erreurs, fr, FR_BYTES);
        }
        vues++;
        if (fire) ko++;
        if (mauvaise) bfi++;
        if (vues <= 20 || vues % 100 == 0) {
            printf("  [a_dd] fn=%u etat=%04x FIRE=%d BFI=%d err=%u (ecart %u trames) | vues=%lu ko=%lu bfi=%lu\n",
                   fn, etat, fire ? 1 : 0, mauvaise ? 1 : 0, erreurs, fn - fn_dernier, vues, ko, bfi);
        }
        fn_dernier = fn;
    }
```

```
    prec = etat;
}

bool montant_sur_tch(void)
{
    return g.sur_tch;
}

/* [2026-10-20] L'intervalle sur lequel le firmware emet : le TS du TCH annonce par le side-band (genre 2/3)
 * tant qu'il n'est pas libere, 0 sinon (SDCCH/4 du TS0, RACH). Sert a etiqueter les bursts publies dans
 * calypso_tx_rom (bits 4-6 de l'octet src) pour que l'air les porte au bon intervalle du TRX. */
int montant_tn_dedie(void)
{
    return g_tn_tch_sb;
}

void tx_rach_vers_air(uint8_t ra, uint8_t bsic, uint32_t tick);   /* pont.c */

static void publier_rach(uint8_t ra, uint8_t bsic, uint32_t fn)
{
    tx_rach_vers_air(ra, bsic, fn);   /* [2026-10-21] Magnetite : vers l'air (CALYPSO_RACH_VERS_AIR) */
    static int fd = -2;
    static uint32_t seq;
    if (fd == -2) {
        fd = sb_ouvrir(SHM_RACH, REC_RACH);
    }
    uint8_t buf[REC_RACH];
    memset(buf, 0, sizeof(buf));
    seq++;
    memcpy(buf + 0, &seq, 4);
    buf[4] = ra;
    buf[5] = bsic;
    memcpy(buf + 8, &fn, 4);
    sb_ecrire(fd, buf, sizeof(buf), 0);
    /* Une tentative d'acces, c'est une nouvelle connexion : la memoire de
     * l'anti-doublon de la voie heuristique repart (sans effet sur la voie
     * B_BLUD, qui n'en a pas besoin). */
    g.sdcch_a_dernier = false;
    if (g.rach++ < (unsigned long)journal())
        printf("  [montant] RACH ra=0x%02x bsic=%u fn=%u -> %s\n", ra, bsic, fn, SHM_RACH);
}

/* ── LE Kc : LE PONT NE PEUT NI CHIFFRER NI DECHIFFRER SANS LUI ──────────
 *
 * [2026-09-22] Meme cause que tout ce fichier : sous CALYPSO_DSP_EXTERN=1 la
 * couche 1 gr-gsm est desactivee, et c'etait ELLE qui publiait
 * /dev/shm/calypso_kc_l1 (calypso_l1_grgsm.c, publish_kc). En montage DSP le
 * fichier n'existait donc pas, `Cipher.current()` de pont.py rendait None, et
 * `cipher.apply()` rendait le burst INCHANGE dans les deux sens -- releve sur
 * le banc : « A5 dl=0 ul=0 » a chaque STATS.
 *
 * Ce que ca coutait, mesure du 2026-09-22 avec ENCRYPTION="a5 1" : la
 * transaction allait jusqu'au bout de l'authentification en clair, puis
 *
 *     11:26:51  CIPHERING MODE COMMAND (sc=1, algo=A5/1 cr=1)
 *     11:26:51  CIPHERING MODE COMPLETE (cr 1)
 *     11:26:53  Dropping frame with 96 bit errors   (et sans fin ensuite)
 *
 * -- la descente chiffree par la BTS que personne ne dechiffre, et la montee
 * que personne ne chiffre. En « a5 0 » la meme transaction va au bout. Il n'y
 * a pas non plus d'A5 dans le modele Calypso (`d_a5mode` n'existe que dans
 * l1-grgsm/, rien dans l1-dsp/) : c'est bien au pont de le faire, comme il
 * fait deja le codage de canal.
 *
 * Disposition reprise telle quelle de publish_kc() pour que pont/cipher.py
 * (KC_RECLEN=32) la lise sans changement : seq(4) algo(1) longueur(1)
```

```
 * Kc[8] 0xFF. Les quatre mots de a_kc sortent en gros-boutiste ET a l'envers,
 * comme dans l'original -- on ne "corrige" pas une disposition que le lecteur
 * attend.
 *
 * MONTANT_KC=0 coupe la publication. */
#define KC_RECLEN        32
#define KC_PUBLIER_TOUTES 22   /* trames entre deux scrutations, comme grgsm */
#define KC_GRACE_CLAIR    5    /* cf. publish_kc : le firmware efface d_a5mode
                                  * a chaque DM_REL_REQ, y compris quand le Kc
                                  * revient juste apres (Assignment Command),
                                  * alors que la BTS, elle, chiffre toujours. */

/* Leve par montant_canal_libere() : la prochaine scrutation doit publier le
 * retour en clair SANS attendre la grace. Voir publier_kc(). */
static bool g_kc_liberer;

static void publier_kc(uint16_t *api_ram)
{
    static int actif = -1, fd = -1, tick, clair_en_attente;
    static uint32_t seq;
    static uint8_t dernier[KC_RECLEN];
    static bool a_dernier;

    if (actif < 0) {
        const char *e = calypso_getenv("MONTANT_KC");
        actif = (e && *e == '0') ? 0 : 1;
    }
    if (!actif) {
        return;
    }
    /* [2026-09-22] LE CHANGEMENT DE MODE NE PEUT PAS ATTENDRE LA SCRUTATION.
     * Version precedente : on ne lisait d_a5mode qu'une trame sur 22 (~128 ms).
     * Or le mobile bascule des qu'il traite le CIPHERING MODE COMMAND et emet
     * son CIPHERING MODE COMPLETE dans la foulee ; un bloc SDCCH montant tombe
     * toutes les 51 trames. Ce bloc-la -- le PREMIER message chiffre du montant
     * -- pouvait donc partir en clair alors que la BTS le dechiffrait deja.
     * Releve du 2026-09-22, run de 11:49 :
     *     fn=2438  01 64 35  06 32 17 ...   CIPHERING MODE COMPLETE
     *     fn=2591  01 74 35  06 32 17 ...   LE MEME, retransmis (bit P)
     * la BTS ne l'acquittait pas, LAPDm (fenetre de 1) restait bloque dessus,
     * le TMSI REALLOCATION COMPLETE n'etait jamais emis et le MSC repondait
     * LOCATION UPDATING REJECT alors que le mobile se croyait a jour.
     * On lit donc d_a5mode a CHAQUE trame -- deux acces memoire -- et la
     * scrutation complete n'est differee que tant que le mode ne change pas. */
    uint16_t mode = api_ram[(API_NDB + NDB_D_A5MODE) / 2];
    uint8_t mode_algo = (mode >= 1 && mode <= 3) ? (uint8_t)mode : 0;
    bool bascule = (a_dernier && mode_algo != dernier[4]) || g_kc_liberer;
    if (!bascule && ++tick < KC_PUBLIER_TOUTES) {
        return;
    }
    tick = 0;
    const uint16_t *kw = &api_ram[(API_NDB + NDB_A_KC) / 2];
    uint8_t rec[KC_RECLEN] = {0};
    bool nul = true;
    for (int i = 0; i < 4; i++) {
        rec[6 + 6 - 2 * i] = (uint8_t)(kw[i] >> 8);
        rec[6 + 7 - 2 * i] = (uint8_t)(kw[i] & 0xFF);
    }
    for (int i = 6; i < 14; i++) {
        if (rec[i]) {
            nul = false;
        }
    }
    uint8_t algo = (mode >= 1 && mode <= 3 && !nul) ? (uint8_t)mode : 0;
    if (!algo) {
        memset(rec + 6, 0, 8);
```

```
    }
    rec[4] = algo;
    rec[5] = algo ? 8 : 0;
    rec[14] = 0xFF;

    if (a_dernier && !memcmp(dernier + 4, rec + 4, KC_RECLEN - 4)) {
        clair_en_attente = 0;
        /* [2026-09-22] Ce retour anticipe doit CONSOMMER g_kc_liberer, sinon la
         * liberation reste armee indefiniment : chaque scrutation suivante
         * calcule bascule=vrai, republie le meme enregistrement et incremente
         * `seq`. Cote pont.py, un `seq` qui bouge veut dire « nouvelle cle » :
         * il rechargeait sans fin une cle inchangee. */
        g_kc_liberer = false;
        return;
    }
    /* [2026-09-22] LA GRACE NE DOIT PAS SURVIVRE A LA LIBERATION DU CANAL.
     * Elle vient de publish_kc() et sert au cas INTRA-connexion : le firmware
     * efface d_a5mode a chaque DM_REL_REQ, y compris pendant un Assignment
     * Command ou le Kc revient juste apres, alors que la BTS chiffre toujours.
     * Mais entre DEUX connexions elle est nuisible : l'enregistrement algo=1
     * restait lisible cinq scrutations de plus, et pont.py -- qui relache
     * pourtant sa cle a chaque IMMEDIATE ASSIGNMENT (downlink.py) -- la
     * relisait aussitot dans le fichier et la restaurait. Il chiffrait alors
     * le montant de la connexion SUIVANTE des son premier bloc, pendant que le
     * mobile emettait encore en clair.
     * Mesure du 2026-09-22, run de 11:53 : « A5 dl=0 ul=200 » -- tout le
     * montant chiffre, rien de descendant dechiffre -- et l'AUTHENTICATION
     * RESPONSE (fn=2525, `05 14`) retransmise a fn=2729 faute d'acquittement.
     * La liberation du canal est le bon signal, et il existe deja :
     * montant_canal_libere(), appele sur PONT_DCCH genre 0xFF. */
    if (!algo && a_dernier && dernier[4] && !g_kc_liberer &&
        ++clair_en_attente < KC_GRACE_CLAIR) {
        return;   /* chiffre -> clair : on attend, le Kc revient peut-etre */
    }
    clair_en_attente = 0;
    g_kc_liberer = false;

    if (fd < 0 && (fd = open(SHM_KC, O_WRONLY | O_CREAT, 0644)) < 0) {
        actif = 0;
        return;
    }
    seq++;
    memcpy(rec, &seq, 4);
    if (pwrite(fd, rec, sizeof rec, 0) != (ssize_t)sizeof rec) {
        close(fd); fd = -1; seq--;
        return;
    }
    memcpy(dernier, rec, sizeof rec);
    a_dernier = true;
    if (algo) {
        printf("  [montant] chiffrement A5/%u : Kc publie vers %s (seq=%u)\n",
               algo, SHM_KC, seq);
    } else {
        printf("  [montant] retour en clair (seq=%u)\n", seq);
    }
    fflush(stdout);
}

void calypso_bsp_set_dedie_ti(int actif);   /* calypso_bsp.c */
static bool g_ti_dedie;                      /* firmware TI en dedie (taches ALLC), voir plus bas */

/* [2026-10-22] PROTHESE FACCH DESCENDANTE (firmware TI). Sous la L1 TI de Magnetite la ROM decode la FACCH/F
 * avec les deux derniers bursts du bloc absents : l'UA qui repond au SABM de l'ASSIGNMENT sort « 01 8b 01 2b »
 * au lieu de « 01 73 01 2b » (reproduit hors ligne par libosmocoding en mettant les bursts 6 et 7 a zero),
 * Fire KO, RR Assignment Failure sur tous les appels du 22/10. Les bursts TS2 que le BSP livre, eux, portent
 * l'UA intact. On les decode ici a chaque fin de bloc FACCH/F (fn % 13 % 4 == 3) et, si c'est une FACCH au CRC
```

```
 * bon, on la pose dans a_fd : la L1 TI la lit a la trame suivante (l1_cmplx.c, fn_mod13 % 4 == 0), apres la
 * fin de ce tick. Le reste (parole) reste a la ROM. CALYPSO_PROTHESE_FACCH_DL=0 coupe (a chaud). */
/* [2026-10-22 21:00] PROTHESE PAROLE DESCENDANTE (firmware TI), meme mal que la FACCH : appel au 600 de 20:58,
 * vocodeur ROM arme (VSP rint 36923, SPC 0x00c8) mais toute la parole decodee par la ROM en BFI (a_dd 1500/1500,
 * ~30 bits faux par trame), crete DL 0. Le bloc FR decode ici des bursts du BSP (CRC bon) est pose dans a_dd au
 * format TI (inverse de parole_ti_vers_fr : gsm_to_canon puis ti_fr_from_canon), BFI a 0, pour que le decodeur
 * de parole de la ROM le prenne a la place du sien. Ne remplace que les trames que la ROM a jugees mauvaises.
 * CALYPSO_PROTHESE_PAROLE_DL=0 coupe (a chaud), =2 remplace aussi les bonnes. */
static void prothese_parole_dl(uint16_t *api_ram, const uint8_t fr[FR_BYTES], int ne, uint32_t e)
{
    static unsigned n_ok;
    int mode = calypso_reglage("CALYPSO_PROTHESE_PAROLE_DL", 1);
    if (!mode) return;
    uint16_t *dd = &api_ram[(API_NDB + NDB_A_DD_0) / 2];
    if (mode == 1 && (dd[0] & B_BLUD) && !(dd[0] & 0x0004)) return;   /* la ROM l'a deja bonne */
    uint8_t canon[FR_BYTES], ti[FR_BYTES + 1];
    memset(canon, 0, sizeof canon);
    for (int i = 0; i < 260; i++)                    /* gsm_to_canon : le quartet 0xd en tete saute */
        mettre_bit_msb(canon, i, bit_msb(fr, i + 4));
    memset(ti, 0, sizeof ti);
    for (int i = 0; i < 260; i++) {                  /* ti_fr_from_canon */
        int si = i >= 182 ? i + 4 : i;
        mettre_bit_msb(ti, si, bit_msb(canon, gsm610_bitorder[i]));
    }
    dd[0] = (uint16_t)((dd[0] & ~0x0044u) | B_BLUD);   /* bloc present, ni FIRE ni BFI */
    dd[2] = (uint16_t)ne;
    for (int k = 0; k < FR_BYTES; k += 2)              /* octet fort d'abord, comme sonde_add */
        dd[3 + k / 2] = (uint16_t)((ti[k] << 8) | (k + 1 < FR_BYTES ? ti[k + 1] : 0));
    if (n_ok++ < 40 || n_ok % 500 == 0)
        printf("  [parole-dl] fn=%u : trame FR de la BTS posee dans a_dd (%d err) [%u]\n", e, ne, n_ok);
}

static void prothese_facch_dl(uint16_t *api_ram)
{
    static uint32_t deja = 0xffffffffu;
    static unsigned n_ok;
    uint32_t e;
    if (!calypso_reglage("CALYPSO_PROTHESE_FACCH_DL", 1) || !g_ti_dedie) return;
    if (!calypso_bsp_tch_dernier_fn(&e) || e == deja) return;
    deja = e;
    if (e % 26u == 12u || e % 26u == 25u || (e % 13u) % 4u != 3u) return;
    uint32_t seq[8]; int n = 0;
    for (uint32_t f = e; n < 8; f--) {
        if (f % 26u == 12u || f % 26u == 25u) continue;
        seq[7 - n++] = f;
    }
    sbit_t soft[8 * 116];
    for (int k = 0; k < 8; k++) {
        uint8_t b[148];
        if (!calypso_bsp_tch_burst(seq[k], b)) return;
        for (int j = 0; j < 58; j++) {
            soft[k * 116 + j]      = b[3 + j]  ? -127 : 127;
            soft[k * 116 + 58 + j] = b[87 + j] ? -127 : 127;
        }
    }
    uint8_t l2[33]; int ne = 0, nb = 0;
    int rc = gsm0503_tch_fr_decode(l2, soft, 1, 0, &ne, &nb);
    if (rc == FR_BYTES) { prothese_parole_dl(api_ram, l2, ne, e); return; }
    if (rc != 23) return;
    uint16_t *fd = &api_ram[(API_NDB + NDB_A_FD) / 2];
    fd[0] = B_BLUD;                 /* bloc present, B_FIRE1/B_FIRE0 a 0 : CRC bon */
    fd[2] = (uint16_t)ne;
    for (int k = 0; k < 12; k++)
        fd[3 + k] = (uint16_t)(l2[2 * k] | (2 * k + 1 < 23 ? l2[2 * k + 1] << 8 : 0));
    if (n_ok++ < 40 || n_ok % 100 == 0)
        printf("  [facch-dl] fn=%u : FACCH de la BTS posee dans a_fd (%02x %02x %02x %02x, %d err) [%u]\n",
```

```
            e, l2[0], l2[1], l2[2], l2[3], ne, n_ok);
    fflush(stdout);
}

void montant_scruter(uint16_t *api_ram, uint32_t fn, unsigned page)
{
    static int coupe = -1;
    if (coupe < 0) {
        const char *e = calypso_getenv("MONTANT");
        coupe = (e && *e == '0') ? 1 : 0;
    }
    if (coupe || !api_ram) {
        return;
    }

    /* Quelle page W porte les taches ? dsp_end_scenario() (firmware,
     * calypso/dsp.c:471) ecrit d_dsp_page = B_GSM_TASK | w_page AVANT de
     * basculer w_page : le mot du NDB est donc la source fraiche, y compris
     * quand l1_sync() a tourne entre le TICK et le GO. L'argument `page` (le
     * d_dsp_page que QEMU avait echantillonne au TICK) ne sert que de repli. */
    uint16_t v_page = api_ram[(API_NDB + NDB_D_DSP_PAGE) / 2];
    bool taches = (v_page & B_GSM_TASK) != 0;
    unsigned pg = taches ? ((v_page & B_GSM_PAGE) ? 1u : 0u) : (page & 1u);

    {   /* [2026-09-23] SONDE CHIFFREMENT : chaque changement de d_a5mode ou de
         * a_kc tel que l'ARM les laisse (NDB 0x1ce / 0x2ce), avec le tick. Le
         * DSP chiffre et dechiffre lui-meme (calypso_a5.c) d'apres ces deux
         * champs ; une remise a zero sur le TCH arrete le dechiffrement alors
         * que la BTS chiffre toujours. */
        static uint16_t a5_prec = 0xffff, kc_prec[4];
        uint16_t a5 = api_ram[(API_NDB + 0x1ce) / 2];
        const uint16_t *kc = &api_ram[(API_NDB + 0x2ce) / 2];
        if (a5 != a5_prec || memcmp(kc, kc_prec, sizeof kc_prec)) {
            printf("  [a5-arm] tick=%u d_a5mode=%u a_kc=%04x %04x %04x %04x%s\n", fn, a5, kc[0], kc[1], kc[2],
↪   kc[3],
                   g.sur_tch ? "  (BSP sur le TCH)" : "");
            fflush(stdout);
            a5_prec = a5; memcpy(kc_prec, kc, sizeof kc_prec);
        }
    }
    scruter_dcch(fn);
    scruter_tch(fn);      /* l'annonce du TCH par le pont                  */
    suivre_tache_tch(api_ram, pg, taches, fn);   /* la bascule, par le firmware */
    sonde_dfn(api_ram, pg, taches, fn);
    if (taches) calypso_bsp_dfn_observe(fn, api_ram[(API_W_PAGE(pg) + WP_D_FN) / 2], api_ram[(API_W_PAGE(pg) +
     ↪  WP_D_TASK_D) / 2]);

    /* [2026-10-22] B_SYNC_TCH_DL POUR LA L1 TI. osmocom (calypso/dsp.c dsp_load_tch_param) pose au premier TCH
     * B_SYNC_TCH_UL (bit 8 de d_ctrl_tch) ET B_SYNC_TCH_DL (bit 9) ; la L1 TI de Magnetite pose le bit 8 seul, son
     * bit 9 s'appelle B_SYNC_AMR et reste a 0 (d_ctrl_tch 0x01d1 puis 0x0091). Sans synchro descendante la ROM
     * garde une phase de bloc TCH arbitraire : chaque FACCH (l'UA du SABM d'ASSIGNMENT) decodee avec ses deux
     * derniers bursts absents (« 01 8b 01 2b » au lieu de « 01 73 01 2b », reproduit hors ligne), Fire KO,
     * RR Assignment Failure (appels au 600 du 22/10). Pose ici, en fin de tick, sur la page W que la ROM lira au
     * tick suivant. CALYPSO_TI_SYNC_DL=0 coupe. */
    {
        /* n = 1 : sur la page du bit 8 ; n > 1 : sur la (n-1)-ieme page a tache suivante (decaler la phase). */
        int sync_dl = calypso_reglage("CALYPSO_TI_SYNC_DL", 1);   /* a chaud, /dev/shm/calypso_reglages */
        static int attente = -1;
        static bool fait_tch;   /* [2026-10-22 21:00] une fois par TCH */
        uint16_t *ctrl = &api_ram[(API_W_PAGE(pg) + 0x12u) / 2];   /* d_ctrl_tch, mot 9 de la page W */
        uint16_t td = api_ram[(API_W_PAGE(pg) + WP_D_TASK_D) / 2] & 0x7fffu;
        if (!g.sur_tch) fait_tch = false;
        if (sync_dl > 0 && taches) {
            /* [2026-10-22 21:00] AUSSI SUR LA PREMIERE TACHE TCH. Appels de 20:52 et 20:58 : la L1 TI pose
             * d_ctrl_tch=0x0091 (sans le bit 8) des la premiere page TCH, le declencheur ne partait jamais
             * (0 [sync-dl]) et toute la parole descendante sortait en BFI (1500/1500, ~30 bits faux). */
```

```
            if ((*ctrl & 0x0100) && !(*ctrl & 0x0200)) attente = sync_dl - 1;
            else if (td == TCHT_DSP_TASK && !fait_tch && !(*ctrl & 0x0200)) { attente = sync_dl - 1; fait_tch =
             ↪  true; }
            else if (attente > 0) attente--;
            if (attente == 0) {
                *ctrl |= 0x0200;
               printf("  [sync-dl] tick=%u page W%u : d_ctrl_tch |= B_SYNC_TCH_DL -> %04x (reglage %d)\n", fn, pg,
↪  *ctrl, sync_dl);
                fflush(stdout);
                attente = -1;
            }
        }
    }
    /* [2026-10-22 22:40] VOIX PAR L'HOTE : PAS DE CODEC ROM. Avec PONT_VOIX_HOTE=1 (pont/voix_hote.py, libgsm) la
     * parole ne passe plus par la ROM, mais la L1 TI pose B_VOCODER_ON (bit 7 de d_ctrl_tch, 0x0091) : la ROM
     * codait et decodait quand meme chaque trame (profil PDROM eb00/eec0/e8c0, PROM0 c6xx/cf00/d9c0), 8,4 ms par
     * trame pour 4,62, horloge du pont asservie au DSP -> tout le reseau a 55 % (DL 27 trames/s). Bit efface sur
     * la page W a tache. CALYPSO_TI_VOCODEUR_OFF (a chaud) : defaut 1 si PONT_VOIX_HOTE=1, sinon 0. */
    {
        static int defaut = -1;
        if (defaut < 0) { const char *e = calypso_getenv("PONT_VOIX_HOTE"); defaut = (e && *e == '1'); }
        if (taches && calypso_reglage("CALYPSO_TI_VOCODEUR_OFF", defaut)) {
            uint16_t *ctrl = &api_ram[(API_W_PAGE(pg) + 0x12u) / 2];   /* d_ctrl_tch, mot 9 de la page W */
            if (*ctrl & 0x0080) {
                static unsigned n_voc;
                *ctrl &= (uint16_t)~0x0080u;
                if (n_voc++ == 0) {
                    printf("  [vocodeur-off] tick=%u : B_VOCODER_ON efface de d_ctrl_tch (voix par l'hote) ->
↪  %04x\n", fn, *ctrl);
                    fflush(stdout);
                }
            }
        }
    }
    prothese_facch_dl(api_ram);   /* [2026-10-22] voir plus haut */

    /* [2026-09-21] LE CANAL DEDIE SE LIBERE AUSSI QUAND L'ARM RECHERCHE LA
     * SYNCHRO. Le tap L1CTL de QEMU n'annonce pas toujours la liberation ;
     * mesure : /dev/shm/calypso_bsp_dedie a garde « tn=1 ss=0 » bien apres la
     * fin de la communication. Le BSP continuait donc de remplacer TS0 sur les
     * trames du canal -- et pour SS=0 ce sont fn%%51 = 0..3, or fn%%51=0 porte
     * la FCCH et fn%%51=1 la SCH. Le mobile perdait sa synchro pour de bon :
     * « FBSB RESP: result=255 » en boucle, d_fb_det=0, DSP parque.
     *
     * Une tache FB (5) ou SB (6) postee par l'ARM veut dire qu'il cherche la
     * synchro sur TS0 : il n'est plus en mode dedie, quoi qu'en dise le tap.
     * Ce signal-la vient du firmware lui-meme. */
    {
        uint16_t md0 = api_ram[(API_W_PAGE(0) + WP_D_TASK_MD) / 2] & 0xff;
        uint16_t md1 = api_ram[(API_W_PAGE(1) + WP_D_TASK_MD) / 2] & 0xff;
        bool cherche_synchro = (md0 == FB_DSP_TASK || md0 == SB_DSP_TASK ||
                                md1 == FB_DSP_TASK || md1 == SB_DSP_TASK);
        if (g.dedie_arme && cherche_synchro) {
            printf("  [montant] tache %s postee : le mobile cherche la synchro, "
                   "canal dedie libere (TS0 rendu au FCCH/SCH)\n",
                   (md0 == FB_DSP_TASK || md1 == FB_DSP_TASK) ? "FB" : "SB");
            g.dedie_arme = false;
            g.sur_tch = false;
            g.sd_valide = false;
            calypso_bsp_set_dedie(0, 0xFF, 0);
            montant_canal_libere();
        }
    }
    sonde_afd(api_ram, fn);
    sonde_add(api_ram, fn);
```

```
    /* [2026-09-21] QUI RATE, ET QUAND. La descente dediee perd environ un bloc
     * sur deux (« Dropping frame with 110 bit errors », fire_crc >= 2 cote
     * layer23) alors que pont decode les MEMES blocs sans une seule erreur
     * (TS1/0:38/0 TS1/32:14/0) et que le BSP livre tous les bursts
     * (manques=0). C'est donc la demodulation dans la ROM qui flanche, pas la
     * plomberie. Cette sonde donne le verdict bloc par bloc : le mot d'etat de
     * a_cd (bit 15 = bloc present, bit 6 = erreur de Fire) avec la trame et sa
     * position dans la multitrame, de quoi voir si l'echec suit le SDCCH, la
     * SACCH, ou une position particuliere. MONTANT_ACD=0 la coupe. */
    {
        static int sonde = -1;
        if (sonde < 0) { const char *e = calypso_getenv("MONTANT_ACD"); sonde = (e && *e == '0') ? 0 : 1; }
        if (sonde && (g.dedie_arme || g_ti_dedie)) {
            static uint16_t prec;
            static unsigned long ok, ko;
            uint16_t etat = api_ram[(API_NDB + NDB_A_CD) / 2];
            /* [22:15] en dedie TI : chaque bloc (le filtre etat != prec cachait les FIRE KO consecutifs) */
            static uint8_t d_prec[4];
            const uint8_t *d = (const uint8_t *)api_ram + API_NDB + NDB_A_CD + 6;
            bool nouveau = (etat != prec) || (g_ti_dedie && memcmp(d, d_prec, 4) != 0);
            if ((etat & B_BLUD) && nouveau) {
                memcpy(d_prec, d, 4);
                bool fire = (etat & 0x0040) != 0;
                if (fire) ko++; else ok++;
                if ((ok + ko) <= 60 || ((ok + ko) % 50) == 0) {
                    printf("  [a_cd] fn=%u p51=%u p102=%u etat=%04x %s "
                           "L2=%02x %02x %02x %02x | ok=%lu ko=%lu\n",
                           fn, fn % 51u, fn % 102u, etat, fire ? "FIRE KO" : "ok",
                           d[0], d[1], d[2], d[3], ok, ko);
                }
            }
            prec = etat;
        }
    }

    uint16_t *wp = &api_ram[API_W_PAGE(pg) / 2];
    uint16_t task_u  = taches ? wp[WP_D_TASK_U / 2] : 0;
    uint16_t task_ra = wp[WP_D_TASK_RA / 2];
    uint16_t d_rach  = api_ram[(API_NDB + NDB_D_RACH) / 2];

    /* [2026-10-21 21:50] SONDE DU MONTANT DEDIE (Magnetite, BTS classique). Apres l'IMM ASS la L1 TI passe sur
     * le SDCCH (DBA_I) mais aucun bloc montant n'a ete publie : capture_sdcch_ul n'etait meme jamais appelee
     * (task_u toujours 0 ici). A la premiere tache ALLC vue dans une page W, 300 lignes : taches des deux pages,
     * d_dsp_page, page choisie, en-tete de a_cu. Et si a_cu porte B_BLUD sans tache montante vue, on le prend
     * quand meme (la L1 TI le pose avec le bloc a emettre, l1ddsp_load_info). */
    {
        static unsigned reste;
        static bool arme;
        uint16_t td0 = api_ram[(API_W_PAGE(0) + WP_D_TASK_D) / 2] & 0x7fffu;
        uint16_t td1 = api_ram[(API_W_PAGE(1) + WP_D_TASK_D) / 2] & 0x7fffu;
        uint16_t cu0 = api_ram[(API_NDB + NDB_A_CU) / 2];
        if (!arme && (td0 == ALLC_DSP_TASK || td1 == ALLC_DSP_TASK)) {
            arme = true;
            reste = 300;
            printf("  [ul-dedie] premiere tache ALLC (tick %u) : sonde du montant sur 300 trames\n", fn);
        }
        if (reste) {
            reste--;
            printf("  [ul-dedie] tick=%u dsp_page=%04x pg=%u W0 d=%u u=%u W1 d=%u u=%u a_cu[0]=%04x\n", fn,
↪   v_page, pg,
                   td0, api_ram[(API_W_PAGE(0) + WP_D_TASK_U) / 2] & 0x7fffu,
                   td1, api_ram[(API_W_PAGE(1) + WP_D_TASK_U) / 2] & 0x7fffu, cu0);
        }
        if (arme && task_u == 0 && (cu0 & B_BLUD))
            capture_sdcch_ul(api_ram, 0, fn);
        /* [2026-10-21 22:05] etat dedie du firmware TI : actif tant qu'une tache ALLC a ete vue dans les 102
```

```
     * dernieres trames. Arme la sonde a_cd et le decalage BSP en dedie (CALYPSO_BSP_DEDIE_DECAL). */
    /* [22:15] sur les TACHES MONTANTES (d_task_u, 11/12 vus a la sonde) : ALLC (24) sert aussi au BCCH et
     * au CCCH en veille (a_cd : SI3, paging lus « en dedie »), le decalage aurait frappe la veille. */
    static uint32_t derniere_allc;
    static bool vu_allc;
    uint16_t tu0 = api_ram[(API_W_PAGE(0) + WP_D_TASK_U) / 2] & 0x7fffu;
    uint16_t tu1 = api_ram[(API_W_PAGE(1) + WP_D_TASK_U) / 2] & 0x7fffu;
    if (tu0 || tu1) { derniere_allc = fn; vu_allc = true; }
    bool dedie = vu_allc && (uint32_t)(fn - derniere_allc) < 102u;
    if (dedie != g_ti_dedie) {
        g_ti_dedie = dedie;
        calypso_bsp_set_dedie_ti(dedie);
    }
}

/* RACH : la tache, pas la valeur.
 *
 * [2026-09-21, mesure sur le banc reel] Le mot NDB d_rach (octet 0x474
 * cote ARM) est AUSSI de la memoire du C54x (data[0x0A3A]) : la ROM y
 * laisse du residu. Echantillonnage a 4 ms pendant 90 s, une ligne par
 * changement :
 *
 *     d_rach   W0.task_ra W1.task_ra   occurrences
 *     0xfe00   0x0000     0x0000       3219   <- residu de la ROM
 *     0x0d54   0x000a     0x0000          1   <- vraie tentative
 *     0x0954   0x0000     0x000a          1   <- vraie tentative
 *     0x0c54   0x0000     0x000a          1   <- vraie tentative
 *
 * Declencher sur la valeur de d_rach publiait donc un access-burst bidon
 * (ra=0xfe, bsic=0) a chaque demarrage. Le signal juste est d_task_ra :
 * prim_rach.c:77 y ecrit dsp_task_iq_swap(RACH_DSP_TASK=10, arfcn, 1) au
 * moment ou il pose la RA, et rien d'autre ne vaut 10. On efface le mot
 * apres publication (comme qosmo-dsp/calypso_trx.c:1955) : le firmware ne
 * le relit jamais (seuls prim_rach.c:77 et sync.c:307 l'ecrivent) et c'est
 * ce qui donne un front propre a la tentative suivante.
 *
 * MONTANT_RACH_SUR_DRACH=1 retablit l'ancien declencheur (transition de
 * d_rach, premiere valeur prise comme reference) pour le cas ou quelque
 * chose consommerait d_task_ra avant cette scrutation. */
static int sur_drach = -1;
if (sur_drach < 0) {
    const char *e = calypso_getenv("MONTANT_RACH_SUR_DRACH");
    sur_drach = (e && *e == '1') ? 1 : 0;
}
bool tache_rach = ((task_ra & 0x7fffu) == RACH_DSP_TASK);
bool front_valeur = false;
if (!g.base_rach) {
    g.base_rach = true;
    g.prev_rach = d_rach;
} else if (d_rach != g.prev_rach) {
    front_valeur = (d_rach != 0);
    g.prev_rach = d_rach;
}
/* [2026-10-15] FRONT DE LA TACHE quand la ROM la garde (MONTANT_EFFACE_TASK_RA=0) :
 * personne n'efface alors d_task_ra, qui reste a RACH_DSP_TASK dans la page W, et la
 * garde de RACH_GARDE_TRAMES republiait le meme RACH toutes les 4 trames (7
 * publications pour 1 burst de la ROM, dsp.log du 15/10). On ne publie plus que sur
 * le front (tache absente -> presente) ou un nouveau d_rach. MONTANT_RACH_DEDUP=0 :
 * ancien comportement. */
static int front_seul = -1;
static bool tache_avant;
if (front_seul < 0) {
    const char *ea = calypso_getenv("MONTANT_EFFACE_TASK_RA");
    const char *ed = calypso_getenv("MONTANT_RACH_DEDUP");
    front_seul = (ea && *ea == '0') && !(ed && *ed == '0');
    if (front_seul)
```

```
            printf("  [montant] RACH publie sur le front de d_task_ra (MONTANT_RACH_DEDUP=0 : toutes les %d
↪  trames)\n",
                   RACH_GARDE_TRAMES);
        }
        bool nouveau_rach = !front_seul || !tache_avant || front_valeur;
        tache_avant = tache_rach;
        /* [2026-10-21 21:25] RACH DIFFERE (MONTANT_RACH_DIFFERE=N trames, Magnetite : 1 par run.sh). La L1 TI pose
         * d_task_ra AVANT d'ecrire le d_rach de la tentative : lu au front, d_rach etait celui de la tentative
         * PRECEDENTE (capture CCCH du 21/10 21:17 : RA_R 9868 ra 4 publie ra 8, 10114 ra 8 publie ra 4...), l'IMM ASS
         * portait le bon fn mais le RA d'avant et Magnetite l'ignorait. On garde le tick du front et on lit d_rach
         * N trames plus tard. */
        static int differe = -1;
        static bool rach_attente;
        static uint32_t fn_attente;
        if (differe < 0) {
            const char *e = calypso_getenv("MONTANT_RACH_DIFFERE");
            differe = (e && *e) ? atoi(e) : 0;
            if (differe > 0) printf("  [montant] RACH differe de %d trame(s) : d_rach lu apres le front de
             ↪  d_task_ra\n", differe);
        }
        if (rach_attente && (uint32_t)(fn - fn_attente) >= (uint32_t)differe) {
            rach_attente = false;
            if (d_rach != 0)
                publier_rach((uint8_t)(d_rach >> 8), (uint8_t)((d_rach & 0xff) >> 2), fn_attente);
            g.prev_rach = d_rach;
        }
        if (d_rach != 0 && (tache_rach || (sur_drach && front_valeur)) && nouveau_rach &&
            (!g.rach_vu || (uint32_t)(fn - g.fn_rach) >= RACH_GARDE_TRAMES)) {
            if (differe > 0) { rach_attente = true; fn_attente = fn; }
            else publier_rach((uint8_t)(d_rach >> 8), (uint8_t)((d_rach & 0xff) >> 2), fn);
            g.prev_rach = d_rach;
            g.fn_rach = fn;
            g.rach_vu = true;
            if (g.rach <= (unsigned long)journal())
                printf("  [montant]   declencheur : %s (task_ra=0x%04x d_rach=0x%04x)\n",
                       tache_rach ? "d_task_ra=RACH_DSP_TASK" : "transition de d_rach",
                       task_ra, d_rach);
            static int consomme = -1;
            if (consomme < 0) {
                const char *e = calypso_getenv("MONTANT_CONSOMME_RACH");
                consomme = (e && *e == '1') ? 1 : 0;
            }
            if (consomme) {
                api_ram[(API_NDB + NDB_D_RACH) / 2] = 0;
                g.prev_rach = 0;
            }
            /* [2026-10-03] Effacement de d_task_ra dans la page W : la ROM ne voit donc jamais la tache RACH
             * (elle la lirait a la trame suivante) et n'execute pas le RACH ; c'est le pont qui emet
             * l'access-burst (side-band calypso_rach). Ecart assume, desormais reglable :
             * MONTANT_EFFACE_TASK_RA=0 laisse la tache a la ROM (comportement d'un vrai Calypso, a valider). */
            /* [2026-10-04] Par defaut la ROM GARDE la tache : elle code l'access-burst (0x3f8a, 36 bits =
             * gsm0503_rach_ext_encode) et son emetteur le serialise dans le script TSP (dsp_tester rach,
             * 5/5), publie par pont.c (calypso_tx_rom). MONTANT_EFFACE_TASK_RA=1 : ancien comportement. */
            static int efface_ra = -1;
            if (efface_ra < 0) {
                const char *e = calypso_getenv("MONTANT_EFFACE_TASK_RA");
                efface_ra = (e && *e == '1') ? 1 : 0;
                printf("  [montant] d_task_ra %s\n", efface_ra ? "efface avant la ROM (MONTANT_EFFACE_TASK_RA=1) :
↪  RACH code par le pont"
                                                              : "laisse a la ROM : elle code l'access-burst
                                                               ↪  (MONTANT_EFFACE_TASK_RA=1 pour l'ancien
                                                               ↪  comportement)");
            }
            if (task_ra && efface_ra) {
                wp[WP_D_TASK_RA / 2] = 0;   /* comme qosmo-dsp/calypso_trx.c:1955 */
            }
```

```
    }

    /* SDCCH / SACCH / FACCH / parole : meme aiguillage que la couche 1 gr-gsm. */
    if (task_u != 0 && !capture_tch_ul(api_ram, task_u, fn)) {
        capture_sdcch_ul(api_ram, task_u, fn);
    }

    publier_kc(api_ram);
}

/* Le canal dedie vient d'etre libere (PONT_DCCH, genre 0xFF) : la memoire de
 * l'anti-doublon doit repartir a zero, sinon le SABM de la connexion SUIVANTE,
 * octet pour octet identique au precedent, serait pris pour un doublon et ne
 * partirait jamais. */
void montant_canal_libere(void)
{
    g.sdcch_a_dernier = false;
    g_kc_liberer = true;   /* le Kc de CETTE connexion ne vaut plus rien */
}

void montant_bilan(void)
{
    if (g.rach || g.sdcch || g.facch || g.sacch || g.parole) {
        printf("  montant publie : RACH %lu, SDCCH %lu, FACCH %lu, SACCH %lu, parole %lu\n",
               g.rach, g.sdcch, g.facch, g.sacch, g.parole);
    } else {
        printf("  montant : rien publie (aucun d_rach ni d_task_u vu dans l'API RAM)\n");
    }
}
```

3.18 /opt/GSM/c54x_exe/src/montant.h

1088 octets, 29 lignes → 29 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * montant.h - le lien montant du montage DSP (RACH, SDCCH, SACCH, FACCH, parole).
 */
#ifndef C54X_EXE_MONTANT_H
#define C54X_EXE_MONTANT_H

#include <stdint.h>
#include <stdbool.h>

/* A appeler une fois par trame, apres que l'ARM a rendu la main (fin de
 * scenario : d_dsp_page ecrit, taches posees dans la page W). `page` est le
 * bit 0 de d_dsp_page, celui que QEMU transporte deja dans PONT_TICK.m.b. */
void montant_scruter(uint16_t *api_ram, uint32_t fn, unsigned page);

/* [2026-10-22] FACCH montante lue au GO, avant que la ROM n'efface B_BLUD de a_fu. */
void montant_capter_fu(uint16_t *api_ram, uint32_t fn);

/* Le canal dedie est libere : oublier le dernier bloc SDCCH publie. */
void montant_canal_libere(void);

/* Une ligne de bilan en fin de session. */
void montant_bilan(void);

/* Vrai tant que le BSP joue l'intervalle du TCH (bascule suivie par le firmware). */
bool montant_sur_tch(void);
int  montant_tn_dedie(void);   /* [2026-10-20] TS du TCH (side-band), 0 sinon : etiquette des bursts publies */

#endif
```

3.19 /opt/GSM/c54x_exe/src/pcb-minimal.c

1380 octets, 44 lignes → 44 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * The four DARAM helpers of calypso_full_pcb.c, copied verbatim.
 *
 * calypso_full_pcb.c as a whole does not build outside QEMU: it includes
 * hw/core/cpu.h and all of QEMU behind it. The helpers the DSP needs do not
 * depend on any of that - one mutex and two array accesses - so they are copied
 * word for word from qosmo-dsp/hw/arm/calypso/calypso_full_pcb.c.
 *
 * This is the only copy in this binary: if the original changes, this file lies.
 * It goes away once calypso_full_pcb.c is decoupled from C54xState.
 */
#include "qemu/osdep.h"
#include "qemu/thread.h"
#include "calypso_c54x.h"

extern QemuMutex calypso_pcb_daram_lock;

uint16_t calypso_dsp_daram_read(void *dsp_void, uint16_t addr)
{
    C54xState *dsp = (C54xState *)dsp_void;
    qemu_mutex_lock(&calypso_pcb_daram_lock);
    uint16_t v = dsp->data[addr];
    qemu_mutex_unlock(&calypso_pcb_daram_lock);
    return v;
}

void calypso_dsp_daram_write(void *dsp_void, uint16_t addr, uint16_t val)
{
    C54xState *dsp = (C54xState *)dsp_void;
    qemu_mutex_lock(&calypso_pcb_daram_lock);
    dsp->data[addr] = val;
    qemu_mutex_unlock(&calypso_pcb_daram_lock);
}

void calypso_pcb_daram_lock_acquire(void)
{
    qemu_mutex_lock(&calypso_pcb_daram_lock);
}

void calypso_pcb_daram_lock_release(void)
{
    qemu_mutex_unlock(&calypso_pcb_daram_lock);
}
```

3.20 /opt/GSM/c54x_exe/src/pont.c

177349 octets, 2920 lignes → 2918 lignes (2 groupes compactés)

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * pont.c - server side of the ARM/DSP bridge.
 *
 * The ARM (osmocom-bb layer1) runs under QEMU (qosmo, CALYPSO_DSP_EXTERN=1);
 * the C54x runs in this process. Both share the API RAM through
 * /dev/shm/calypso_api_ram and lock step frame by frame over
 * /tmp/calypso_dsp.sock. The per-frame sequence below mirrors, section by
 * section, what qosmo-dsp/hw/arm/calypso/calypso_trx.c:calypso_tdma_tick()
 * does with its internal DSP; divergences between the two originate here.
```

```
 */
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <poll.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <math.h>
#include <sys/stat.h>
#include <osmocom/core/bits.h>
#include <osmocom/coding/gsm0503_coding.h>
#include "calypso_c54x.h"
#include "calypso_dma.h"
#include "calypso_bsp.h"
#include "calypso_twl3025.h"
#include "calypso_rhea_dma.h"
#include "calypso_a5.h"
#include "calypso_vsp.h"
#include "calypso_rif.h"
#include "hw/arm/calypso/calypso_api.h"
#include "hw/arm/calypso/calypso_dsp_pont.h"
#include "pont.h"
#include "tsp_tx.h"
#include "calypso_gmsk.h"
#include "cellule.h"
#include "montant.h"
#include "carte_tx.h"
#include "hw/arm/calypso/calypso_debug.h"

extern int g_toa_grille, g_toa_valeur;   /* c54x_mem.c : provenance du TOA */
extern uint32_t g_c54x_exe_fn;          /* main.c: value returned by calypso_trx_get_fn() */

static volatile sig_atomic_t g_stop;
static void sur_signal(int sig) { (void)sig; g_stop = 1; }

/* [2026-10-06] STOP / GO COOPERATIFS SUR LE DSP EN MARCHE (harvard.sh stop / go).
 * SIGUSR1 demande une pause, prise a la frontiere de trame suivante (en haut de
 * la boucle TICK : le DSP a rendu son DONE, il est en IDLE, l'ARM attend) ;
 * SIGUSR2 reprend. Pendant la pause, l'etat du coeur est ecrit dans
 * PONT_PAUSE_FICHIER (une ligne « CLE valeur » par element, lisible par un
 * shell) ; a la reprise la premiere ligne devient « ETAT reprise ». Rien n'est
 * ecrit dans l'API RAM ni dans le coeur : le firmware ne voit qu'une trame plus
 * longue, comme un SIGSTOP, mais a un point propre et avec les registres. */
#define PONT_PAUSE_FICHIER "/dev/shm/calypso_dsp_pause"
static volatile sig_atomic_t g_pause;
static void sur_pause(int sig) { g_pause = (sig == SIGUSR1); }
static void pause_ecrire(const C54xState *dsp, const char *etat)
{
    FILE *f = fopen(PONT_PAUSE_FICHIER ".tmp", "w");
    if (!f) return;
    fprintf(f, "ETAT %s\nFN %u\nPC %04x\nXPC %x\nIDLE %d\nRUNNING %d\nINSN %u\nLAST_PC %04x\nLAST_OP %04x\n",
            etat, g_c54x_exe_fn, dsp->pc & 0xffff, dsp->xpc, dsp->idle, dsp->running, dsp->insn_count,
            dsp->last_exec_pc, dsp->last_exec_op);
    fprintf(f, "A %llx\nB %llx\nT %x\nSP %x\nST0 %x\nST1 %x\nPMST %x\nIMR %x\nIFR %x\nBK %x\nBRC %x\nRSA %x\nREA
↪  %x\nTRN %x\n",
            (unsigned long long)(dsp->a & 0xFFFFFFFFFULL), (unsigned long long)(dsp->b & 0xFFFFFFFFFULL),
            dsp->t, dsp->sp, dsp->st0, dsp->st1, dsp->pmst, dsp->imr, dsp->ifr, dsp->bk, dsp->brc, dsp->rsa,
↪  dsp->rea, dsp->trn);
    for (int i = 0; i < 8; i++) fprintf(f, "AR%d %x\n", i, dsp->ar[i]);
```

```
    fprintf(f, "RPT %d %x %x\nDELAY %d %x\n", dsp->rpt_active, dsp->rpt_count, dsp->rpt_pc, dsp->delay_slots,
↪   dsp->delayed_pc);
    fclose(f);
    rename(PONT_PAUSE_FICHIER ".tmp", PONT_PAUSE_FICHIER);
}
static void pause_cooperative(C54xState *dsp)
{
    pause_ecrire(dsp, "pause");
    printf("pont : PAUSE (SIGUSR1) fn=%u pc=%04x xpc=%x idle=%d : etat dans %s, SIGUSR2 pour reprendre\n",
           g_c54x_exe_fn, dsp->pc & 0xffff, dsp->xpc, dsp->idle, PONT_PAUSE_FICHIER);
    fflush(stdout);
    while (g_pause && !g_stop) usleep(10000);
    pause_ecrire(dsp, "reprise");
    printf("pont : REPRISE fn=%u pc=%04x\n", g_c54x_exe_fn, dsp->pc & 0xffff);
    fflush(stdout);
}

C54xState *pont_allouer_dsp(void)
{
    /* Shift the start of the struct so that data[] lands on a page boundary,
     * and therefore data[C54X_API_BASE] too: the API window must be mmap-able
     * with MAP_FIXED. */
    size_t off = offsetof(C54xState, data) % 4096;
    size_t decal = off ? 4096 - off : 0;
    void *brut = NULL;
    if (posix_memalign(&brut, 4096, sizeof(C54xState) + 4096) != 0) {
        fprintf(stderr, "pont : posix_memalign\n");
        return NULL;
    }
    memset(brut, 0, sizeof(C54xState) + 4096);
    C54xState *s = (C54xState *)((char *)brut + decal);
    uint16_t *fenetre = &s->data[C54X_API_BASE];
    if (((uintptr_t)fenetre & 4095) != 0) {
        fprintf(stderr, "pont : fenetre API non alignee (%p)\n", (void *)fenetre);
        return NULL;
    }

    int fd = shm_open(CALYPSO_PONT_SHM, O_RDWR | O_CREAT, 0666);
    if (fd < 0) {
        fprintf(stderr, "pont : shm_open(%s) : %s\n", CALYPSO_PONT_SHM, strerror(errno));
        return NULL;
    }
    /* [2026-10-06] shm_open applique le umask : 0666 devient 0644 sous root, et un
     * harvard.sh lance sous le compte de l'utilisateur ne peut plus ECRIRE l'API RAM
     * (d@!, demos sb/si3). fchmod force 0666 pour que le shm soit ecrivable par tous. */
    if (fchmod(fd, 0666) < 0) {
        fprintf(stderr, "pont : fchmod(%s, 0666) : %s\n", CALYPSO_PONT_SHM, strerror(errno));
    }
    if (ftruncate(fd, CALYPSO_PONT_SHM_BYTES) < 0) {
        fprintf(stderr, "pont : ftruncate : %s\n", strerror(errno));
        close(fd);
        return NULL;
    }
    void *map = mmap(fenetre, CALYPSO_PONT_SHM_BYTES, PROT_READ | PROT_WRITE,
                     MAP_SHARED | MAP_FIXED, fd, 0);
    close(fd);
    if (map == MAP_FAILED || map != (void *)fenetre) {
        fprintf(stderr, "pont : mmap MAP_FIXED : %s\n", strerror(errno));
        return NULL;
    }
    memset(fenetre, 0, CALYPSO_PONT_SHM_BYTES);
    return s;
}

static bool envoyer(int fd, uint32_t type, uint32_t a, uint32_t b)
{
```

```
    CalypsoPontMsg m = { type, a, b };
    return send(fd, &m, sizeof(m), MSG_NOSIGNAL) == (ssize_t)sizeof(m);
}

/* PC profile: c54x_run is driven in slices of 64 instructions and the PC
 * sampled between two slices is charged to a 64-word bucket. No sampling
 * thread, no cost. Published every 217 frames. */
#define PROFIL_SEAU 64
static unsigned long g_profil[0x10000 / PROFIL_SEAU];
static uint32_t g_profil_hw;     /* high-water mark: highest PC seen (boot excluded) */
/* Trace window: PONT_TRACE_FB=N logs the next N instructions (pc, opcode, A,
 * AR3, DP, INTM, IMR/IFR) to /tmp/c54x-pont/trace-fb.txt, starting at the
 * first frame whose write page carries d_task_md=5 (FB_DSP_TASK). Exact
 * single-stepping (c54x_run with n=1). */
static long g_trace_reste = -1;
static FILE *g_trace_f;
static bool g_trace_ouverte;
static uint32_t g_trace_pc_lo, g_trace_pc_hi;   /* PONT_TRACE_PC=lo-hi: open the trace when PC enters [lo,hi] */
static void trace_armer(void)
{
    static bool fait;
    if (fait) return;
    fait = true;
    const char *e = calypso_getenv("PONT_TRACE_FB");
    const char *r = calypso_getenv("PONT_TRACE_PC");
    if (e && *e) { g_trace_reste = atol(e); g_trace_f = fopen("/tmp/c54x-pont/trace-fb.txt", "w"); }
    if (r && *r) {
        char *fin = NULL; g_trace_pc_lo = (uint32_t)strtoul(r, &fin, 0);
        g_trace_pc_hi = (fin && *fin == '-') ? (uint32_t)strtoul(fin + 1, NULL, 0) : g_trace_pc_lo;
        if (g_trace_reste <= 0) g_trace_reste = 4000;
        if (!g_trace_f) g_trace_f = fopen("/tmp/c54x-pont/trace-fb.txt", "w");
    }
}
/* DARAM cells watched during the trace: every change is logged with the PC of
 * the instruction that made it (before/after diff on each step). */
/* Cells watched during the trace. Rebuilt [2026-09-19]: the previous list had
 * accreted 13 addresses (0x3fc1 0x3fde 0x3fd3 0x0c3f 0x0906 0x058a 0x0e60
 * 0x435b 0x0e4f 0x2a01 ...) that appear NOWHERE else in the tree and whose
 * introducing commit (812c343) says only "commit". An address whose meaning
 * cannot be restated is not evidence: when it moves, nothing follows. Every
 * entry below carries where its meaning comes from.
 *
 * API window cells are derived from calypso_api.h rather than written raw, so
 * they cannot drift from the map: API base = C54X_API_BASE = 0x0800 words, and
 * the API_* offsets are in BYTES (hence the /2). */
#define CEL_W(p, off)   (uint16_t)(C54X_API_BASE + (API_W_PAGE(p) + (off)) / 2)
#define CEL_R(p, off)   (uint16_t)(C54X_API_BASE + (API_R_PAGE(p) + (off)) / 2)
#define CEL_NDB(off)    (uint16_t)(C54X_API_BASE + (API_NDB + (off)) / 2)

static const uint16_t g_cellules[] = {
    /* --- ARM -> DSP pages (calypso_api.h) ------------------------------- */
    CEL_W(0, WP_D_TASK_MD), CEL_W(1, WP_D_TASK_MD),   /* 0x0804/0x0818 the posted task */
    CEL_W(0, WP_D_TASK_D),  CEL_W(1, WP_D_TASK_D),    /* 0x0800/0x0814 */
    CEL_W(0, WP_D_FN),      CEL_W(1, WP_D_FN),        /* 0x0808/0x081c */

    /* --- NDB (calypso_api.h + calypso_fbsb.h) --------------------------- */
    CEL_NDB(NDB_D_DSP_PAGE),                          /* 0x08d4 page toggle */
    CEL_NDB(NDB_D_FB_DET),                            /* 0x08f8 FB found flag */
    CEL_NDB(NDB_A_SYNC_DEMOD + 2 * D_TOA),            /* 0x08fa */
    CEL_NDB(NDB_A_SYNC_DEMOD + 2 * D_PM),             /* 0x08fb */
    CEL_NDB(NDB_A_SYNC_DEMOD + 2 * D_ANGLE),          /* 0x08fc  <- was MISSING */
    CEL_NDB(NDB_A_SYNC_DEMOD + 2 * D_SNR),            /* 0x08fd  <- was MISSING */

    /* --- DSP -> ARM pages: a_sch[0,1,3,4], BOTH pages ------------------- */
    CEL_R(0, RP_A_SCH + 0), CEL_R(0, RP_A_SCH + 2),
    CEL_R(0, RP_A_SCH + 6), CEL_R(0, RP_A_SCH + 8),   /* 0x0837 38 3a 3b */
```

```
    CEL_R(1, RP_A_SCH + 0), CEL_R(1, RP_A_SCH + 2),
    CEL_R(1, RP_A_SCH + 6), CEL_R(1, RP_A_SCH + 8),   /* 0x084b 4c 4e 4f */

    /* --- FB correlator input, read out of the mask ROM [2026-09-19] ------ */
    0x3fb5,   /* pointer to the input buffer; PROM0 0xb2c4 ST #0x0cce,*(0x3fb5)
               *                              and 0xb2c9 ST #0x0d2e,*(0x3fb5) */
    0x0cce,   /* buffer A, same two instructions; also the AAD the DMA uses */
    0x0d2e,   /* buffer B, idem */
    0x0e4e,   /* second DARAM target seen in the rhea-dma RX transfers */
    0x2a00,   /* CALYPSO_BSP_DARAM_ADDR env default, before AAD_FOLLOW */

    /* --- cells whose meaning is stated elsewhere in the tree ------------- */
    0x3f92,   /* calypso_dma.h:9   source of d_error_status (0x08d5) */
    0x43d8,   /* calypso_bsp.c:1567  poked per burst on the FB/SB mission */
    0x43d5,   /* c54x_mem.c:2229   PROM 0xb4be stm #0x43d5 ; reada *AR1+ */
    0x098c,   /* calypso_mailbox.c:46  mailbox poll (0xde86 ld *(0x098c)) */
    0x435e,   /* calypso_c54x.c:4574  bit13 = DMA config lock */
    0x4368,   /* c54x_mem.c:1835   DISPATCH-CELL-RESEED */
    0x3fb0,   /* c54x_probes.c:505  BSP read window 0x3fb0..0x3fbf */
    0x0000 };
#define N_CELLULES (sizeof(g_cellules) / sizeof(g_cellules[0]))
static uint16_t g_cell_prev[N_CELLULES];
static uint16_t g_trace_pc_prev;
/* [2026-10-03] Zones surveillees en plus pendant une trace d'injection (PONT_TX_TRACE_BLOC) :
 * a_cu (entree du codeur) et le tampon de sortie 0x4280.. + 0x2be4.. ; chaque ecriture est
 * journalisee avec le PC de l'instruction qui l'a faite. */
static uint16_t g_zone_prev[0x10000];
static bool g_zone_active;
static void trace_zones(C54xState *dsp, bool init)
{
    static const struct { unsigned a, n; } z[] = { {0x0a06, 16}, {0x4280, 32}, {0x2be4, 52}, {0x2bc0, 8}, {0x3f8a,
     ↪  8} };
    for (unsigned r = 0; r < sizeof z / sizeof z[0]; r++)
        for (unsigned k = 0; k < z[r].n; k++) {
            unsigned a = z[r].a + k; uint16_t v = dsp->data[a];
            if (!init && v != g_zone_prev[a] && g_trace_f)
                fprintf(g_trace_f, "      Z data[%04x] %04x -> %04x   (par pc=%04x)\n", a, g_zone_prev[a], v,
 ↪  g_trace_pc_prev);
            g_zone_prev[a] = v;
        }
}
static void trace_cellules(C54xState *dsp, bool init)
{
    if (g_zone_active) trace_zones(dsp, init);
    for (unsigned i = 0; i < N_CELLULES; i++) {
        uint16_t v = dsp->data[g_cellules[i]];
        if (!init && v != g_cell_prev[i] && g_trace_f)
            fprintf(g_trace_f, "      W data[%04x] %04x -> %04x   (par pc=%04x)\n",
                    g_cellules[i], g_cell_prev[i], v, g_trace_pc_prev);
        g_cell_prev[i] = v;
    }
}

static inline void trace_pas(C54xState *dsp)
{
    if (g_trace_reste <= 0 || !g_trace_f || !g_trace_ouverte) return;
    trace_cellules(dsp, false);
    g_trace_pc_prev = dsp->pc & 0xffff;
    uint16_t op = dsp->prog[dsp->pc & 0x3ffff];
    uint16_t op2 = dsp->prog[(dsp->pc + 1) & 0x3ffff];
    fprintf(g_trace_f, "%04x  %04x %04x  A=%010llx B=%010llx AR1=%04x AR2=%04x AR3=%04x AR4=%04x DP=%03x INTM=%d
 ↪  IMR=%04x IFR=%04x SP=%04x T=%04x BRC=%04x RSA=%04x REA=%04x ST0=%04x ST1=%04x insn=%u\n",
            dsp->pc & 0xffff, op, op2,
            (unsigned long long)(dsp->a & 0xfffffffffULL), (unsigned long long)(dsp->b & 0xfffffffffULL),
            dsp->ar[1], dsp->ar[2], dsp->ar[3], dsp->ar[4], dsp->st0 & 0x1ff, !!(dsp->st1 & 0x800),
```

```
            dsp->imr, dsp->ifr, dsp->sp, dsp->t, dsp->brc, dsp->rsa, dsp->rea, dsp->st0, dsp->st1,
 ↪  dsp->insn_count);
    if (--g_trace_reste == 0) { fclose(g_trace_f); g_trace_f = NULL; printf("pont : trace FB terminee
     ↪  (/tmp/c54x-pont/trace-fb.txt)\n"); }
}

/* PONT_PC_COUNT=pc1,pc2,... : exact hit counters, published with the profile.
 * Forces single-stepping, about 3x slower; diagnostic use only. */
static uint16_t g_pcc[16]; static unsigned long g_pcc_n[16]; static int g_pcc_k = -1;
static void pcc_armer(void)
{
    if (g_pcc_k >= 0) return;
    g_pcc_k = 0;
    const char *e = calypso_getenv("PONT_PC_COUNT");
    while (e && *e && g_pcc_k < 16) {
        char *fin = NULL; long v = strtol(e, &fin, 0);
        if (fin == e) break;
        g_pcc[g_pcc_k++] = (uint16_t)v;
        e = (*fin == ',') ? fin + 1 : fin;
    }
}
/* PONT_DUMP_DATA=addr:n,addr:n : DARAM words printed along with the profile. */
static void dump_publier(C54xState *dsp)
{
    const char *e = calypso_getenv("PONT_DUMP_DATA");
    while (e && *e) {
        char *fin = NULL; long a = strtol(e, &fin, 0); long n = 8;
        if (fin == e) break;
        if (*fin == ':') n = strtol(fin + 1, &fin, 0);
        printf("  data[%04lx..] :", a);
        for (long i = 0; i < n && i < 32; i++) printf(" %04x", dsp->data[(a + i) & 0xffff]);
        printf("\n");
        e = (*fin == ',') ? fin + 1 : fin;
    }
}
static void pcc_publier(void)
{
    if (g_pcc_k <= 0) return;
    printf("  passages exacts :");
    for (int i = 0; i < g_pcc_k; i++) { printf(" %04x:%lu", g_pcc[i], g_pcc_n[i]); g_pcc_n[i] = 0; }
    printf("\n");
}

/* -- SONDE TX : la ROM execute-t-elle les taches montantes ? ---------------
 * [2026-10-03] PONT_TX_SONDE=N (N evenements, 0/absent = inerte). A chaque trame
 * ou l'ARM a pose d_task_u (ou d_task_ra) dans l'une des pages W, on compare la
 * memoire de donnees du DSP avant et apres l'execution de la trame et on cherche
 * ce que la ROM aurait ecrit : (a) les plages modifiees, (b) les suites de >= 100
 * mots ne valant que 0 ou 1 — l'allure des 148 bits durs d'un burst montant, que
 * calypso_bsp_tx_burst() lit a l'adresse CANDIDATE data[0x0900]. Ne modifie rien. */
static uint16_t *g_tx_snap;
static int g_tx_n = -1, g_tx_arme;
static int tx_sonde_active(void)
{
    if (g_tx_n < 0) { const char *e = calypso_getenv("PONT_TX_SONDE"); g_tx_n = (e && *e) ? atoi(e) : 0; }
    return g_tx_n > 0;
}
static void tx_sonde_avant(const C54xState *dsp, const uint16_t *api_ram)
{
    (void)api_ram;
    /* [2026-10-03] On photographie a CHAQUE trame : l'ARM pose ses taches APRES le
     * TICK (l1_sync tourne entre le TICK et le GO), donc d_task_u est encore a 0 ici
     * et armer sur sa valeur de ce moment ne declenchait jamais. La decision se
     * prend dans tx_sonde_apres(). */
    g_tx_arme = 0;
    if (!tx_sonde_active()) return;
```

```
    if (!g_tx_snap) g_tx_snap = malloc(sizeof(uint16_t) * C54X_DATA_SIZE);
    if (!g_tx_snap) return;
    memcpy(g_tx_snap, dsp->data, sizeof(uint16_t) * C54X_DATA_SIZE);
    g_tx_arme = 1;
}
static void tx_sonde_apres(const C54xState *dsp, const uint16_t *api_ram, uint32_t fn)
{
    if (!g_tx_arme || !g_tx_snap || g_tx_n <= 0) return;
    g_tx_arme = 0;
    /* declenche si une tache montante est posee dans l'une des pages W, et
     * AUSSI sur la trame suivante (la ROM peut l'executer une trame plus tard). */
    static int suite;
    int pose = api_ram[(API_W_PAGE(0) + WP_D_TASK_U) / 2] || api_ram[(API_W_PAGE(1) + WP_D_TASK_U) / 2] ||
               api_ram[(API_W_PAGE(0) + 0x06u) / 2] || api_ram[(API_W_PAGE(1) + 0x06u) / 2];
    if (!pose && !suite) return;
    suite = pose ? 1 : 0;
    unsigned changes = 0, a, nplages = 0, runs = 0;
    for (a = 0; a < C54X_DATA_SIZE; a++) if (dsp->data[a] != g_tx_snap[a]) changes++;
    /* une trame ou la ROM ne touche a rien (tache deja traitee, ROM au repos) ne
     * compte pas dans les N evenements : elle noyait la mesure. */
    if (changes == 0) return;
    g_tx_n--;
    printf("  [tx-sonde] fn=%u task_u: page0=0x%04x page1=0x%04x task_ra: page0=0x%04x page1=0x%04x pc=0x%04x\n",
 ↪  fn,
           api_ram[(API_W_PAGE(0) + WP_D_TASK_U) / 2], api_ram[(API_W_PAGE(1) + WP_D_TASK_U) / 2],
           api_ram[(API_W_PAGE(0) + 0x06u) / 2], api_ram[(API_W_PAGE(1) + 0x06u) / 2], dsp->pc);
    printf("  [tx-sonde]   %u mots de donnees modifies pendant la trame\n", changes);
    /* Les plages modifiees d'au moins 8 mots, ET LEUR CONTENU dans
     * /tmp/tx-sonde/fn<fn>_<adresse>.txt (un mot hexa par ligne) : de quoi
     * chercher hors ligne le bloc encode attendu, sous toutes ses formes (un bit
     * par mot, 16 bits par mot...). */
    mkdir("/tmp/tx-sonde", 0777);
    for (a = 0; a < C54X_DATA_SIZE && nplages < 12; ) {
        if (dsp->data[a] == g_tx_snap[a]) { a++; continue; }
        unsigned b = a; unsigned n = 0;
        while (b < C54X_DATA_SIZE && (dsp->data[b] != g_tx_snap[b] || (b + 1 < C54X_DATA_SIZE && dsp->data[b + 1]
         ↪  != g_tx_snap[b + 1]))) { b++; n++; }
        if (n >= 8) {
            char nom[96]; snprintf(nom, sizeof nom, "/tmp/tx-sonde/fn%u_%04x.txt", fn, a);
            FILE *f = fopen(nom, "w");
            if (f) { for (unsigned k = a; k < b; k++) fprintf(f, "%04x\n", dsp->data[k]); fclose(f); }
            printf("  [tx-sonde]   modifie data[0x%04x..0x%04x] (%u mots) -> %s\n", a, b - 1, n, nom);
            nplages++;
        }
        a = b + 1;
    }
    /* suites de >= 100 mots valant 0 ou 1, dont au moins 40 modifies */
    for (a = 0; a < C54X_DATA_SIZE && runs < 4; ) {
        if (dsp->data[a] > 1) { a++; continue; }
        unsigned b = a, ch = 0, uns = 0;
        while (b < C54X_DATA_SIZE && dsp->data[b] <= 1) { if (dsp->data[b] != g_tx_snap[b]) ch++; if
         ↪  (dsp->data[b]) uns++; b++; }
        if (b - a >= 100 && ch >= 40 && uns > 0) {
            printf("  [tx-sonde]   SUITE 0/1 data[0x%04x..0x%04x] long=%u modifies=%u uns=%u :", a, b - 1, b - a,
 ↪  ch, uns);
            for (unsigned k = 0; k < 24 && a + k < b; k++) printf(" %u", dsp->data[a + k]);
            printf("\n"); runs++;
        }
        a = b + 1;
    }
    if (!runs) printf("  [tx-sonde]   aucune suite de bits 0/1 (mot a mot)\n");
    fflush(stdout);
}

/* ── INJECTION DE BLOCS L2 DANS LA ROM (experience de linearite) ──────────────
 * [2026-10-03] PONT_TX_INJECT=<fichier> : une ligne par bloc de 23 octets hexa.
```

```
 * A chaque bloc montant (a_cu) que le firmware pose, APRES que montant.c l'a
 * publie (donc le lien reel n'est pas touche : la BTS recoit le vrai bloc), on
 * ecrit le bloc de test dans a_cu a sa place, pour que la ROM l'encode a la
 * trame suivante. Le codage etant lineaire, un bloc nul puis des blocs a un
 * seul bit donnent la carte entre bits d'entree et bits de sortie du tampon
 * lu par la sonde TX (PONT_TX_SONDE). Meme ordre d'octets que prendre_ul()
 * (montant.c) : mot 3 + i/2, octet faible d'abord. Ne fait rien si la
 * variable est absente. */
#define TX_INJ_MAX 600
static uint8_t g_inj_bloc[TX_INJ_MAX][23];
static C54xState *dsp_global_tx;
static int g_inj_n = -1, g_inj_i, g_inj_attente = -1;
static void tx_inject_charger(void)
{
    const char *f = calypso_getenv("PONT_TX_INJECT");
    g_inj_n = 0;
    if (!f || !*f) return;
    FILE *fp = fopen(f, "r");
    if (!fp) { printf("  [tx-inject] %s illisible\n", f); return; }
    char ligne[256];
    while (g_inj_n < TX_INJ_MAX && fgets(ligne, sizeof ligne, fp)) {
       unsigned v[23]; int k = sscanf(ligne, "%x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x",
            &v[0],&v[1],&v[2],&v[3],&v[4],&v[5],&v[6],&v[7],&v[8],&v[9],&v[10],&v[11],&v[12],&v[13],&v[14],&v[15
            ↪  ],&v[16],&v[17],&v[18],&v[19],&v[20],&v[21],&v[22]);
        if (k != 23) continue;
        for (int i = 0; i < 23; i++) g_inj_bloc[g_inj_n][i] = (uint8_t)v[i];
        g_inj_n++;
    }
    fclose(fp);
    printf("  [tx-inject] %d blocs de test charges depuis %s\n", g_inj_n, f);
}
/* avant = B_BLUD de a_cu AVANT montant_scruter() : un bloc vient d'etre pose. */
static void tx_inject_apres(uint16_t *api_ram, uint32_t fn, int avant)
{
    if (g_inj_n < 0) tx_inject_charger();
    if (g_inj_n == 0 || !avant) return;
    if (g_inj_i >= g_inj_n) {
       if (g_inj_i == g_inj_n) { printf("  [tx-inject] fn=%u fin : %d blocs injectes\n", fn, g_inj_n); g_inj_i++; }
        return;
    }
    uint16_t *w = &api_ram[(API_NDB + NDB_A_CU) / 2];
    const uint8_t *b = g_inj_bloc[g_inj_i];
    for (int i = 0; i < 23; i += 2)
        w[3 + i / 2] = (uint16_t)(b[i] | ((i + 1 < 23 ? b[i + 1] : 0) << 8));
    g_inj_attente = g_inj_i;   /* sa sortie sera lue a la fin de la trame suivante */
    { static long cible = -2;
      if (cible == -2) { const char *e = calypso_getenv("PONT_TX_TRACE_BLOC"); cible = (e && *e) ? atol(e) : -1; }
      if (cible == g_inj_i && !g_trace_f) {
          const char *e = calypso_getenv("PONT_TX_TRACE_N");
          g_trace_reste = (e && *e) ? atol(e) : 250000;
          g_trace_f = fopen("/tmp/c54x-pont/trace-tx.txt", "w");
          if (g_trace_f) { g_zone_active = true; g_trace_ouverte = true; trace_cellules(dsp_global_tx, true);
              fprintf(g_trace_f, "# trace du bloc injecte %d a fn=%u\n", g_inj_i, fn);
              printf("  [tx-inject] trace ouverte (bloc %d, %ld instructions) -> /tmp/c54x-pont/trace-tx.txt\n",
 ↪  g_inj_i, g_trace_reste); }
      } }
    printf("  [tx-inject] fn=%u bloc=%d ->", fn, g_inj_i);
    for (int i = 0; i < 23; i++) printf(" %02x", b[i]);
    printf("\n");
    g_inj_i++;
}

/* Fin de la trame qui suit l'injection : on ecrit, pour ce bloc, les plages OU
 * LA ROM A DEPOSE sa sortie (relevees par la sonde TX : 0x4280.. et 0x2be4..),
 * a ADRESSES FIXES, qu'elles aient change ou non. /tmp/tx-sonde/inj_<bloc>.txt :
 * « adresse mot » par ligne. Independant du compteur d'evenements de la sonde. */
```

```
static void tx_inject_fulldump(const C54xState *dsp, int bloc, uint32_t fn, int d)
{
    static int actif=-1;
    if (actif<0){const char*e=calypso_getenv("PONT_TX_FULLDUMP");actif=(e&&*e=='1');}
    if(!actif) return;
    if (d > 6) return;
    mkdir("/tmp/tx-full",0777);
    char nom[96]; snprintf(nom,sizeof nom,"/tmp/tx-full/full_%d_%d.txt",bloc,d);
    FILE*f=fopen(nom,"w");
    if(f){fprintf(f,"# fn=%u bloc=%d\n",fn,bloc);
        for(unsigned a=0x2800;a<0x5c00;a++) fprintf(f,"%04x %04x\n",a,dsp->data[a]);
        fclose(f);}
}
static void tx_inject_sortie(const C54xState *dsp, uint32_t fn)
{
    /* [2026-10-03] la ROM peut finir son travail sur PLUSIEURS trames (la parite FIRE, le remplissage
     * des bursts suivants) : on releve les memes plages pendant les 8 trames qui suivent l'injection,
     * /tmp/tx-sonde/injf_<bloc>_<d>.txt (d = trames apres l'injection), en plus du releve historique
     * inj_<bloc>.txt de la premiere trame. */
    static int restant = 0, bloc = -1, d = 0;
    if (g_inj_attente >= 0) { bloc = g_inj_attente; restant = 8; d = 0; }
    if (restant <= 0) { g_inj_attente = -1; return; }
    tx_inject_fulldump(dsp, bloc, fn, d + 1);
    static const struct { unsigned a, n; } plages[] = {
        {0x4280, 32}, {0x2be4, 52}, {0x2bc0, 8}, {0x3dc6, 9}, {0x5abb, 12}, {0x3f8a, 8}, {0x0809, 8}, {0x081d, 8},
        ↪  {0x0a06, 15}
    };
    mkdir("/tmp/tx-sonde", 0777);
    d++;
    char nom[96];
    if (d == 1) snprintf(nom, sizeof nom, "/tmp/tx-sonde/inj_%d.txt", bloc);
    else        snprintf(nom, sizeof nom, "/tmp/tx-sonde/injf_%d_%d.txt", bloc, d);
    FILE *f = fopen(nom, "w");
    if (f) {
        fprintf(f, "# fn=%u bloc=%d d=%d\n", fn, bloc, d);
        for (unsigned r = 0; r < sizeof plages / sizeof plages[0]; r++)
            for (unsigned k = 0; k < plages[r].n; k++)
                fprintf(f, "%04x %04x\n", plages[r].a + k, dsp->data[plages[r].a + k]);
        fclose(f);
    }
    restant--;
    g_inj_attente = -1;
}

/* ── MODE OMBRE : la ROM encode, l'hote encode, on compare ──────────────────
 * [2026-10-03] PONT_TX_OMBRE=1. A chaque bloc SDCCH montant REEL (a_cu, B_BLUD
 * pose par le firmware), apres que montant.c l'a publie, on garde ses 23 octets ;
 * a la fin de la trame suivante on lit le tampon que la ROM a rempli (data[0x4280],
 * 4 groupes de 8 mots = 4 bursts) et on le compare aux bits de gsm0503_xcch_encode
 * (celui que le pont utilise). Rien n'est ecrit dans le DSP ni envoye : le lien reste
 * celui de l'hote. Disposition mesuree : bit j du burst g a la colonne q = j - 2g dans
 * le groupe (premiere moitie, q < 57) et q = j - 2g - 2 (seconde moitie, apres le bit
 * de vol en q = 57). Compare q in [0,50) et [58,106) : zone sure pour g = 0..3.
 * Sortie : « [ombre] fn=.. g0 a/b g1 .. » + totaux, et le detail dans /tmp/tx-sonde/ombre.txt. */
static int g_omb = -1, g_omb_att;
static int g_txf_restant, g_txf_d; static uint8_t g_txf_l2[23];
static int g_rt_restant, g_rt_d, g_rt_n; static uint8_t g_rt_l2[23];
static uint8_t g_omb_l2[23];
static unsigned long g_omb_blocs, g_omb_parfaits, g_omb_ok[4], g_omb_n[4];
/* ── OMBRE DESCENDANTE : reecrire a la source ce que l'ARM va lire ──────────
 * [2026-10-06] CALYPSO_SHADOW_LAC=<n> et CALYPSO_SHADOW_SB=1. a_cd (NDB) porte
 * le bloc L2 descendant decode, a_sch le SCH decode ; l'ARM les lit APRES le DONE.
 * En lockstep le DSP ecrit puis l'ARM lit : on s'intercale juste avant le DONE, donc
 * ce qu'on pose est bien ce que le firmware recoit (deterministe, pas de course).
 *   - SI3 : si a_cd est un System Information 3 (octet type L2 = 0x1b), on remplace
 *     les 2 octets du LAC (LAI) par n -> mobile.log : CGI=MCC-MNC-n-CI. Flag FIRE
```

```
 *     inchange, aucun CRC ne protege le L3. (Le cell_id n'est pas touche.)
 *   - SB : a_sch (R0/R1) <- DEAD BEEF. DESTRUCTIF : le firmware perd la synchro SCH ;
 *     a n'activer que pour montrer la perte de cellule.
 * Offsets : a_cd[3]=debut L2 (dsp_memcpy_from_api be=0, 2 octets/mot, octet bas d'abord)
 * donc type=octet bas de a_cd[4], cell_id = octet haut de a_cd[4] (MSB) | octet bas de a_cd[5] (LSB). */
static void ombre_descendante(uint16_t *api_ram, uint32_t fn)
{
    static int lac = -2, sb = -2;
    if (lac == -2) { const char *e = calypso_getenv("CALYPSO_SHADOW_LAC");
                     lac = (e && *e) ? atoi(e) : -1;
                     if (lac >= 0) printf("  [ombre-dl] LAC SI3 force a %d (0x%04x)\n", lac, lac & 0xffff); }
    if (sb == -2)  { const char *e = calypso_getenv("CALYPSO_SHADOW_SB");
                     sb = (e && *e == '1') ? 1 : 0;
                     if (sb) printf("  [ombre-dl] SB a_sch force a 40C3 2026 (CCC 2026 ; DESTRUCTIF : perte de
                      ↪  synchro)\n"); }
    if (!api_ram) return;
    if (lac >= 0) {
        /* LAI dans SI3 : octets L2 8-9 = LAC (gros-boutiste). dsp_memcpy be=0 les place tous deux dans
         * a_cd[7] : octet 8 (LAC MSB) = octet bas de a_cd[7], octet 9 (LAC LSB) = octet haut. Donc
         * a_cd[7] = (LAC_LSB << 8) | LAC_MSB = octet-echange du LAC. On ne touche PAS le cell_id (a_cd[4..5]). */
        uint16_t *w = &api_ram[(API_NDB + NDB_A_CD) / 2];
        if ((w[4] & 0xff) == 0x1b) {                       /* SI3 (type L2 = octet bas de a_cd[4]) */
            uint16_t cible = (uint16_t)(((lac & 0xff) << 8) | ((lac >> 8) & 0xff));
            if (w[7] != cible) {
                int av = ((w[7] & 0xff) << 8) | ((w[7] >> 8) & 0xff);
                w[7] = cible;
                static unsigned long n;
                if (++n <= 5 || n % 200 == 0)
                    printf("  [ombre-dl] fn=%u SI3 LAC %d -> %d (#%lu)\n", fn, av, lac & 0xffff, n);
            }
        }
    }
    if (sb) {
        for (unsigned base = 0x0037; ; base = 0x004b) {    /* a_sch[0] R0=0x0837, R1=0x084b ; index = adr - 0x0800
         ↪  */
            api_ram[base] = 0x40c3; api_ram[base + 1] = 0x2026;   /* CCC 2026, marqueur lisible */
            if (base == 0x004b) break;
        }
    }
}
static void tx_ombre_pose(const uint16_t *api_ram, int blud_avant)
{
    if (g_omb < 0) { const char *e = calypso_getenv("PONT_TX_OMBRE"); g_omb = (e && *e == '1'); if (g_omb)
     ↪  printf("  [ombre] actif\n"); }
    if (!g_omb || !blud_avant) return;
    const uint16_t *w = &api_ram[(API_NDB + NDB_A_CU) / 2];
    for (int i = 0; i < 23; i += 2) {
        g_omb_l2[i] = (uint8_t)(w[3 + i / 2] & 0xff);
        if (i + 1 < 23) g_omb_l2[i + 1] = (uint8_t)(w[3 + i / 2] >> 8);
    }
    g_omb_att = 1;
    { int a=-1; static int actif=-1; if(actif<0){const
     ↪  char*e=calypso_getenv("PONT_TX_TXFRAMES");actif=(e&&*e=='1');}
      if(actif){ memcpy(g_txf_l2,g_omb_l2,23); g_txf_restant=10; g_txf_d=0; } (void)a; }
    { static int ar=-1; if(ar<0){const char*e=calypso_getenv("PONT_TX_REELFULL");ar=(e&&*e=='1');}
      if(ar){ memcpy(g_rt_l2,g_omb_l2,23); g_rt_restant=6; g_rt_d=0; } }
}
/* [2026-10-03] PONT_TX_TXFRAMES=1 : apres chaque bloc reel pose (a_cu), on enregistre data[0x4280]
 * (32 mots) a CHAQUE trame pendant 10 trames, avec le L2 et l'offset d, dans /tmp/tx-sonde/txf.txt.
 * But : les lanes de parite du burst peuvent n'etre ecrites qu'a la trame d'EMISSION de chaque burst
 * (une par burst), apres la trame de codage. */
static void tx_txframes(const C54xState *dsp)
{
    static int actif=-1;
    if(actif<0){const char*e=calypso_getenv("PONT_TX_TXFRAMES");actif=(e&&*e=='1');}
    if(!actif||g_txf_restant<=0) return;
```

```
    g_txf_d++;
    mkdir("/tmp/tx-sonde",0777);
    FILE*f=fopen("/tmp/tx-sonde/txf.txt","a");
    if(f){for(int i=0;i<23;i++)fprintf(f,"%02x",g_txf_l2[i]);
        fprintf(f," %d ",g_txf_d);
        for(int w=0;w<32;w++)fprintf(f,"%04x",dsp->data[0x4280+w]);
        fprintf(f,"\n");fclose(f);}
    g_txf_restant--;
}
static void tx_reelfull(const C54xState *dsp, uint32_t fn)
{
    static int actif=-1;
    if(actif<0){const char*e=calypso_getenv("PONT_TX_REELFULL");actif=(e&&*e=='1');}
    if(!actif||g_rt_restant<=0||g_rt_n>=600) return;
    g_rt_d++;
    mkdir("/tmp/tx-rt",0777);
    char nom[96]; snprintf(nom,sizeof nom,"/tmp/tx-rt/rt_%04d_%d.txt",g_rt_n,g_rt_d);
    FILE*f=fopen(nom,"w");
    if(f){for(int i=0;i<23;i++)fprintf(f,"%02x",g_rt_l2[i]);fprintf(f,"\n");
        for(unsigned a=0x0800;a<0x0e00;a++)fprintf(f,"%04x %04x\n",a,dsp->data[a]);
        for(unsigned a=0x2a00;a<0x2d00;a++)fprintf(f,"%04x %04x\n",a,dsp->data[a]);
        for(unsigned a=0x4200;a<0x4300;a++)fprintf(f,"%04x %04x\n",a,dsp->data[a]);
        fclose(f);}
    g_rt_restant--; if(g_rt_restant==0) g_rt_n++;
}
static void tx_ombre_compare(const C54xState *dsp, uint32_t fn)
{
    tx_txframes(dsp);
    tx_reelfull(dsp, fn);
    if (g_omb <= 0 || !g_omb_att) return;
    g_omb_att = 0;
    ubit_t e[1024] = {0};
    if (gsm0503_xcch_encode(e, g_omb_l2) != 0) return;
    unsigned ok = 0, n = 0, okg[4] = {0}, ng[4] = {0};
    mkdir("/tmp/tx-sonde", 0777);
    FILE *f = fopen("/tmp/tx-sonde/ombre.txt", "a");
    if (f) { fprintf(f, "# fn=%u L2=", fn); for (int i = 0; i < 23; i++) fprintf(f, "%02x ", g_omb_l2[i]);
     ↪  fprintf(f, "\n"); }
    for (int p = 0; p < 512; p++) {
        int i = g_carte_tx[p];
        if (i < 0) continue;
        unsigned bit = (dsp->data[0x4280 + p / 16] >> (15 - p % 16)) & 1u, att = e[i] & 1u;
        n++; ng[p / 128]++;
        if (bit == att) { ok++; okg[p / 128]++; }
        else if (f) fprintf(f, "  p=%d (g%d q=%d) i=%d rom=%u hote=%u\n", p, p / 128, p % 128, i, bit, att);
    }
    if (f) fclose(f);
    FILE *g = fopen("/tmp/tx-sonde/reel.txt", "a");
    if (g) {
        for (int i = 0; i < 23; i++) fprintf(g, "%02x", g_omb_l2[i]);
        fprintf(g, " ");
        for (int w = 0; w < 32; w++) fprintf(g, "%04x", dsp->data[0x4280 + w]);
        fprintf(g, "\n"); fclose(g);
    }
    g_omb_blocs++; g_omb_parfaits += (ok == n);
    for (int g = 0; g < 4; g++) { g_omb_ok[g] += okg[g]; g_omb_n[g] += ng[g]; }
    printf("  [ombre] fn=%u L2=%02x %02x %02x.. %u/%u bits connus identiques (g0 %u/%u g1 %u/%u g2 %u/%u g3 %u/%u)
 ↪  | blocs=%lu parfaits=%lu\n", fn,
           g_omb_l2[0], g_omb_l2[1], g_omb_l2[2], ok, n, okg[0], ng[0], okg[1], ng[1], okg[2], ng[2], okg[3],
 ↪  ng[3], g_omb_blocs, g_omb_parfaits);
    fflush(stdout);
}

/* [2026-10-03] BURSTS MONTANTS xCCH EMIS PAR LA ROM -> pont.py.
 * A chaque trame d'emission, la ROM (PROM0 0x890e-0x891b) copie le burst courant du tampon circulaire
 * 0x4280..0x429f (BK=32, 4 bursts de 8 mots) vers data[0x3f8a..0x3f91] : 114 bits + hl/hu dans les 2 bits
```

```
 * bas du mot 7, MSB d'abord ; puis remet a 0 le mot 7 du burst dans 0x4280 et range le pointeur avance de
 * 8 dans data[0x3d9b] (mvkd ar2,@0x1b, DP=0x7b). Le codage d'un bloc remet ce pointeur a 0x4280
 * (0x8820). Un changement de data[0x3d9b] en fin de trame = un burst vient de partir ; son rang est
 * ((p - 0x4280)/8 + 3) & 3. Mesure (trace du bloc ff x23) : les 4 copies dans 0x3f8a sont
 * gsm0503_xcch_encode bit a bit.
 * Les 4 bursts d'un bloc sont publies ensemble dans /dev/shm/calypso_xcch_ul_rom (anneau, format :
 * pont/dsp/uplink.py), avec le L2 pose dans a_cu juste avant le burst 0 (pour le journal ; le pont
 * ne s'y fie pas, il redecode les bursts). En mode chiffre la ROM XORe le flux de cle A5 montant
 * (data[0x3f9b..0x3fa2], calypso_a5.c) sur 0x3f8a meme : on publie ce flux a cote des bits, le pont
 * defait le XOR (la ROM chiffre a l'heure du DSP, pas au fn ou le pont emet) et rechiffre au fn de
 * l'air. MONTANT_ROM_UL=0 coupe la publication. */
#define ROM_UL_SHM      "/dev/shm/calypso_xcch_ul_rom"
#define ROM_UL_SLOTS    8
#define ROM_UL_SLOT_SZ  1024    /* seq(4) fn(4) l2(23) .(1) bits(4 x 116 ubits) flux(4 x 116 ubits) */
#define ROM_UL_FLUX     0x3f9b  /* flux de cle A5 montant que la ROM XORe sur 0x3f8a (PROM0 0x85d1) */
#define ROM_UL_PTR      0x3d9b
#define ROM_UL_BURST    0x3f8a
static uint8_t g_romul_l2_pose[23];     /* dernier L2 pose dans a_cu (tx_rom_pose) */
static void tx_rom_pose(const uint16_t *api_ram, int blud_avant)
{
    if (!blud_avant) return;
    const uint16_t *w = &api_ram[(API_NDB + NDB_A_CU) / 2];
    for (int i = 0; i < 23; i++)
        g_romul_l2_pose[i] = (uint8_t)(i & 1 ? w[3 + i / 2] >> 8 : w[3 + i / 2] & 0xff);
}

/* [2026-10-04] LE BURST MONTANT FINAL DE LA ROM (script TSP, voir src/tsp_tx.h). Chaque trame ou le
 * script porte un burst nouveau, on le publie dans /dev/shm/calypso_tx_rom (anneau : en-tete w(4)
 * slots(4), puis des cases de 160 octets : seq(4) fn(4) type(1) tsc(1) offset(1) .(1) bits(148)).
 * fn = la trame du tick (m.a), celle que l'ARM a donnee au DSP pour ce burst : c'est aussi le
 * numero de trame avec lequel la ROM a chiffre (dsp_tester tx-sdcch-a5 : XOR = osmo_a5(kc, fn de la
 * commande), 24/24). Le pont peut donc emettre ce burst tel quel a ce fn, sans codage ni A5 cote
 * hote (PONT_UL_RAW=1 dans pont/dsp/uplink.py). MONTANT_ROM_TX=0 coupe la publication. */
/* [2026-10-04] LE NUMERO DE TRAME DU BURST. Le tick du pas-a-pas (m.a) n'est PAS le fn GSM : le
 * firmware se cale sur la SCH (run de 17:49 : SB fn=409 au tick ~1009) et le pont emettait les bursts
 * de la ROM a fn%51 = 5..8 au lieu de 15..18. Le fn vrai est dans la page W que le firmware remet au
 * DSP : a_a5fn[0] = (T3 << 5) | T2, a_a5fn[1] = T1 (calypso/dsp.c dsp_load_tch_param, ecrit avec
 * chaque tache RX/TX de next_time). On le releve a l'IT trame, sur la page que d_dsp_page designe ;
 * les pages sans tache RX/TX (RACH) gardent un a_a5fn perime : on prend alors tick + decalage, le
 * decalage etant celui de la derniere page datee. */
#define GSM_HYPERFRAME 2715648u
static struct { bool valide; uint32_t fn; bool dec_valide; int32_t dec; uint32_t tick, tick_dec; bool tache_tx; }
 ↪  g_tsp_fn;
static uint32_t gsmtime2fn(unsigned t1, unsigned t2, unsigned t3)
{
    return (1326u * t1 + 51u * ((t3 + 26u - t2) % 26u) + t3) % GSM_HYPERFRAME;
}
static void rach_dfn_noter(const C54xState *dsp, uint32_t tick);
static void scruter_ra_c(void);
static void rach_pend_echeance(uint32_t tick);
static struct { bool actif; uint32_t tick; uint8_t bits[148]; } g_rach_pend;
static uint32_t g_rom_emis_tick = 0xffffffffu;   /* dernier burst non NB emis par la ROM (RACH) */
static void rach_pend_publier(uint32_t tick_emis, const char *quoi);
static void tx_tsp_dater(const C54xState *dsp, uint32_t tick)
{
    const uint16_t *api = &dsp->data[0x800];
    uint16_t v = api[(API_NDB + NDB_D_DSP_PAGE) / 2];
    unsigned pg = (v & B_GSM_PAGE) ? 1u : 0u;
    const uint16_t *w = &api[API_W_PAGE(pg) / 2];
    bool datee = (w[WP_D_TASK_D / 2] & 0x7fff) != 0 || (w[WP_D_TASK_U / 2] & 0x7fff) != 0;
    g_tsp_fn.tick = tick; g_tsp_fn.valide = false;
    /* [2026-10-19] une tache d'emission commandee cette trame (RACH : c'est elle que l'extcode
     * du firmware trx detourne pour emettre FB/SB/dummy/NB) : tx_tsp_publier republie meme un
     * burst identique au precedent quand MONTANT_ROM_TX_CHAQUE=1 */
    g_tsp_fn.tache_tx = (v & B_GSM_TASK) && ((w[WP_D_TASK_RA / 2] & 0x7fff) != 0 || (w[WP_D_TASK_U / 2] & 0x7fff)
 ↪  != 0);
```

```
    if (v & B_GSM_TASK) rach_dfn_noter(dsp, tick);
    rach_pend_echeance(tick);
    if ((tick % 64u) == 0) scruter_ra_c();   /* [2026-10-22 14:15] RA_C lu au fil de l'eau, pas seulement au RACH
     ↪  suivant */
    if (!(v & B_GSM_TASK) || !datee) return;
    unsigned t2 = w[12] & 0x1f, t3 = (w[12] >> 5) & 0x3f, t1 = w[13] & 0x7ff;
    if (t2 >= 26 || t3 >= 51) return;
    /* [2026-10-20] firmware trx (mode BTS) : il ne pose jamais a_a5fn (pas de dsp_load_tch_param),
     * la page dit t1=t2=t3=0 a chaque trame et tous les bursts du TRX partaient avec fn=0 dans
     * l'anneau calypso_tx_rom. On garde alors le tick nu (source 0), monotone. */
    if (t1 == 0 && t2 == 0 && t3 == 0) return;
    g_tsp_fn.fn = gsmtime2fn(t1, t2, t3); g_tsp_fn.valide = true;
    g_tsp_fn.dec = (int32_t)((g_tsp_fn.fn + GSM_HYPERFRAME - (tick % GSM_HYPERFRAME)) % GSM_HYPERFRAME);
    g_tsp_fn.dec_valide = true;
    g_tsp_fn.tick_dec = tick;
    calypso_bsp_off_mobile_a5fn(tick, g_tsp_fn.fn);   /* [2026-10-22] l'offset de veille suit le dedie */
}
/* [2026-10-22] RACH DATE A UNE TRAME PRES. En veille le fn du mobile n'est connu que par le dernier SB ou le
 * dernier a_a5fn ; la L1 TI recale sa base de temps (tpu_offset) et revient d'un dedie avec l'un ou l'autre, a
 * une trame pres (22/10 : 494 a_a5fn, 495 attendu ; run precedent l'inverse). Un RACH date faux = IMM ASS
 * ignoree, le mobile renvoie un RACH. Donc : un nouveau RACH sans entree en dedie depuis le precedent (moins de
 * 1000 trames) -> correction suivante parmi 0, +1, -1 ; un dedie obtenu -> la correction reste.
 * CALYPSO_RACH_CORR=0 pour couper. */
/* [2026-10-22 12:10] RA_C : la trace L1 TI (rvinterf, CALYPSO_RVINTERF_LOG) dit a quel fn le MOBILE a emis
 * chaque RACH. Lue a chaque RACH (le RA_C du precedent est arrive depuis) ; le tick du RACH correspondant
 * vient de la file des derniers RACH. Apprend l'offset du mobile (calypso_bsp_off_mobile_rac) et remet la
 * correction tournante a 0. */
static uint32_t g_rach_ticks[16]; static unsigned g_rach_w;
static int rach_corr_idx_reset;
static void scruter_ra_c(void)
{
    static FILE *f; static long pos = -1;
    if (!f) {
        const char *e = calypso_getenv("CALYPSO_RVINTERF_LOG");
        f = fopen(e && *e ? e : "/tmp/c54x-pont/rvinterf.log", "r");
        if (!f) return;
        fseek(f, 0, SEEK_END); pos = ftell(f);
    }
    char l[256];
    clearerr(f); fseek(f, pos, SEEK_SET);
    while (fgets(l, sizeof l, f)) {
        pos = ftell(f);
        const char *p = strstr(l, "L1: RA_C");
        if (!p) continue;
        unsigned r = (unsigned)strtoul(p + 8, NULL, 10) % 42432u;
        /* le RACH dont le tick est le plus proche de r (mod 42432), parmi les 16 derniers */
        int best = -1; long bd = 1L << 30;
        for (unsigned k = 0; k < 16 && k < g_rach_w; k++) {
            uint32_t t = g_rach_ticks[(g_rach_w - 1 - k) % 16];
            uint32_t f0; if (!calypso_bsp_fn_mobile_du_tick(t, &f0)) f0 = t;
            long d = (long)((r + 42432u - f0 % 42432u) % 42432u); if (d > 21216) d -= 42432; if (d < 0) d = -d;
            if (d < bd) { bd = d; best = (int)((g_rach_w - 1 - k) % 16); }
        }
        if (best >= 0 && bd < 40) { calypso_bsp_off_mobile_rac(g_rach_ticks[best], r); rach_corr_idx_reset = 1; }
    }
}
static int rach_corr(uint32_t tick)
{
    static int on = -1, idx; static uint32_t t_prec; static unsigned dedie_prec; static bool vu;
    /* [2026-10-22 13:20] l'apprentissage RA_C ne remet PLUS la rotation a 0 : l'offset RA_C oscille de +-1
     ↪  (766/767)
     * et chaque RACH repartait a la correction 0, sans jamais explorer +1/-1 -- dix RACH sans canal, acces
 ↪  abandonne,
     * reselection (13:16). La rotation continue tant qu'aucun dedie n'est obtenu, et garde la valeur gagnante. */
    rach_corr_idx_reset = 0;
    static const int seq[5] = { 0, +1, -1, +2, -2 };   /* [2026-10-22] +-2 vus entre deux acces */
```

```
    if (on < 0) { const char *e = calypso_getenv("CALYPSO_RACH_CORR"); on = !(e && *e == '0'); }
    if (!on) return 0;
    /* [2026-10-22 14:05] ROTATION COUPEE PAR DEFAUT. Run de 14:02 : RA_C - tick = 760 pour les six RACH d'un meme
     * acces, offset appris des le 2e (« [off-mobile] 758 -> 760 ») ; c'est la rotation ajoutee par-dessus qui
     * datait faux les suivants (5630/5631, 5848/5850, 6097/6096, 6311/6309) : 5 SDCCH en Timeout a la BSC.
     * On garde la file des RACH et la lecture RA_C (apprentissage), sans correction. CALYPSO_RACH_ROTATION=1. */
    static int rot = -1;
    if (rot < 0) { const char *e = calypso_getenv("CALYPSO_RACH_ROTATION"); rot = (e && *e == '1'); }
    if (!vu || (uint32_t)(tick - t_prec) > 8) {      /* un nouveau RACH (les deux chemins tombent au meme tick) */
        scruter_ra_c();
        rach_corr_idx_reset = 0;
        g_rach_ticks[g_rach_w++ % 16] = tick;
        unsigned d = calypso_bsp_dedie_entrees();
        if (vu && d == dedie_prec && (uint32_t)(tick - t_prec) < 1000) {
            idx = (idx + 1) % 5;
           if (rot) printf("  [rach-corr] RACH sans canal depuis le tick %u : correction %+d\n", t_prec, seq[idx]);
        }
        dedie_prec = d; t_prec = tick; vu = true;
    }
    return rot ? seq[idx] : 0;
}
/* [2026-10-22 14:05] RACH DATE PAR d_fn. Run de 13:57 : 5 RACH pour une IMM ASS acceptee (BSC : « lchan allocation
 * failed ... WAIT_RLL_RTP_ESTABLISH: Timeout » pour chaque SDCCH perdu). Publie 2070/2290/2523/2747/2977, le
 * mobile (RA_C) 2069/2289/2521/2749/2977 : seul le 5e concordait. Le fn du tick (offset RA_C, 768..770) glisse
 * d'une trame a l'autre et la rotation en ajoute. La page W du RACH porte d_fn = fn_report | (fn%104) << 8, le
 * fn du MOBILE : on prend le fn le plus proche de l'estimation qui a ce reste modulo 104 (+ CALYPSO_RACH_DFN_K).
 * CALYPSO_RACH_DFN=0 : ancienne voie (estimation + rotation). */
static struct { bool valide; uint32_t tick; unsigned fn104; } g_rach_dfn;
static void rach_dfn_noter(const C54xState *dsp, uint32_t tick)
{
    if (!dsp) return;
    const uint16_t *api = &dsp->data[0x800];
    uint16_t v = api[(API_NDB + NDB_D_DSP_PAGE) / 2];
    const uint16_t *w = &api[API_W_PAGE((v & B_GSM_PAGE) ? 1u : 0u) / 2];
    if (!(w[WP_D_TASK_RA / 2] & 0x7fff)) return;
    unsigned f104 = (unsigned)(w[WP_D_FN / 2] >> 8);
    if (f104 >= 104 || w[WP_D_FN / 2] == 0) return;
    g_rach_dfn.valide = true; g_rach_dfn.tick = tick; g_rach_dfn.fn104 = f104;
}
static bool rach_dfn_caler(uint32_t tick, uint32_t estime, uint32_t *fn)
{
    static int on = -1, k = 0;
    /* 14:05 : le firmware TI ne pose pas d_fn sur la tache RACH (0 -> calage faux de +6) : coupe par defaut */
    if (on < 0) { const char *e = calypso_getenv("CALYPSO_RACH_DFN"); on = (e && *e == '1');
                  e = calypso_getenv("CALYPSO_RACH_DFN_K"); k = (e && *e) ? atoi(e) : 0; }
    if (!on || !g_rach_dfn.valide || (uint32_t)(tick - g_rach_dfn.tick + 2) > 4) return false;
    int d = ((int)g_rach_dfn.fn104 + k - (int)(estime % 104) + 2 * 104) % 104;
    if (d >= 52) d -= 104;
    if (d < -6 || d > 6) return false;
    *fn = (estime + GSM_HYPERFRAME + (uint32_t)d) % GSM_HYPERFRAME;
    static unsigned n;
    if (++n <= 40 || n % 100 == 0)
        printf("  [rach-dfn] tick=%u d_fn%%104=%u estime=%u -> fn %u (%+d)\n", tick, g_rach_dfn.fn104, estime,
 ↪  *fn, d);
    return true;
}
static uint32_t tx_tsp_fn(uint32_t tick, int *source)
{
    if (g_tsp_fn.valide && g_tsp_fn.tick == tick) { *source = 1; return g_tsp_fn.fn; }
    /* [2026-10-21 23:10] decalage a_a5fn PERIME au-dela de 102 trames (CALYPSO_TSP_DEC_TTL ; 0 = jamais) : la L1
     ↪  TI ne
     * pose a_a5fn qu'en connexion chiffree ; en veille le mobile se resynchronise (SB) et son fn glisse d'une trame
     * par rapport au decalage fige de la derniere connexion -> RACH date fn+1 (RA_C 33884 contre 33885 publie),
     * IMM ASS ignorees, echec d'acces, reselection (barres perdues), paging sans reponse (MT SMS en echec).
 ↪  Au-dela,
     * le calage BSP (source 3), qui suit les recalages de l'offset. */
```

```
    static int ttl = -1;
    if (ttl < 0) { const char *e = calypso_getenv("CALYPSO_TSP_DEC_TTL"); ttl = (e && *e) ? atoi(e) : 102; }
    if (g_tsp_fn.dec_valide && (ttl == 0 || (uint32_t)(tick - g_tsp_fn.tick_dec) <= (uint32_t)ttl)) {
        *source = 2; return (uint32_t)((tick + (uint32_t)g_tsp_fn.dec) % GSM_HYPERFRAME);
    }
    /* [2026-10-21 23:15] firmware sans a_a5fn en veille (L1 TI) : le fn que le MOBILE croit, appris au dernier SB
     * livre en fenetre SB (calypso_bsp.c g_off_mobile) ; c'est a lui que l'IMM ASS doit correspondre. */
    scruter_ra_c();   /* [2026-10-22 14:15] le RA_C du RACH precedent AVANT d'estimer (il arrivait un RACH trop
↪   tard) */
    { uint32_t f, c; if (calypso_bsp_fn_mobile_du_tick(tick, &f)) { *source = 4;
          if (rach_dfn_caler(tick, f, &c)) { rach_corr(tick); return c; }   /* rach_corr : file des RACH pour RA_C
           ↪  */
          return (uint32_t)((f + GSM_HYPERFRAME + rach_corr(tick)) % GSM_HYPERFRAME); } }
    /* [2026-10-21 18:45] firmware sans a_a5fn (L1 TI, Magnetite) : la trame BTS du calage TS0 */
    { uint32_t f; if (calypso_bsp_fn_bts_du_tick(tick, &f)) { *source = 3; return f; } }
    *source = 0; return tick;
}
#define ROM_TX_SHM   "/dev/shm/calypso_tx_rom"
#define ROM_TX_SLOTS 256
#define ROM_TX_SZ    160
/* [2026-10-20] L'anneau calypso_tx_rom est partage par tx_tsp_publier (bursts captures au script TSP) et
 * tx_page_publier (mode TRX : bursts synthetises depuis la page de l'extcode, un par slot TX). Case de
 * 160 octets : seq(4) fn(4) type(1) tsc(1) offset(1) src(1) bits(148) ; src : bit 7 = A5, bits 4-6 = tn. */
static int g_ring_fd = -1;
static uint32_t g_ring_w;
static bool ring_publier(int type, const uint8_t *bits, int tsc, int offset, uint8_t src, uint32_t fn_gsm)
{
    if (g_ring_fd < 0) {
        g_ring_fd = open(ROM_TX_SHM, O_RDWR | O_CREAT, 0666);
        if (g_ring_fd < 0 || ftruncate(g_ring_fd, 8 + ROM_TX_SLOTS * ROM_TX_SZ) < 0) return false;
        uint32_t h[2] = { 0, ROM_TX_SLOTS };
        if (pwrite(g_ring_fd, h, sizeof h, 0) < 0) return false;
    }
    uint8_t slot[ROM_TX_SZ] = {0};
    g_ring_w++;
    memcpy(slot, &g_ring_w, 4); memcpy(slot + 4, &fn_gsm, 4);
    slot[8] = (uint8_t)type; slot[9] = (uint8_t)(tsc < 0 ? 0xff : tsc); slot[10] = (uint8_t)(offset < 0 ? 0xff :
↪   offset);
    slot[11] = src;
    memcpy(slot + 12, bits, 148);
    off_t off = 8 + (off_t)((g_ring_w - 1) % ROM_TX_SLOTS) * ROM_TX_SZ;
    return pwrite(g_ring_fd, slot, sizeof slot, off) >= 0 && pwrite(g_ring_fd, &g_ring_w, 4, 0) >= 0;
}
static int trx_page_mode(void)
{
    static int m = -1;
    if (m < 0) { const char *e = calypso_getenv("CALYPSO_BSP_TRX"); m = (e && *e == '1') ? 1 : 0; }
    return m;
}
static void tx_tsp_publier(const C54xState *dsp, uint32_t fn)
{
    static int actif = -1;
    static unsigned n_rach, n_nb, n_inconnu, n_sb, n_dummy, n_fb;
    static int chaque = -1;
    if (actif < 0) {
        const char *e = calypso_getenv("MONTANT_ROM_TX"); actif = !(e && *e == '0');
        e = calypso_getenv("MONTANT_ROM_TX_CHAQUE"); chaque = (e && *e == '1') ? 1 : 0;
        if (chaque) printf("  [rom-tx] MONTANT_ROM_TX_CHAQUE=1 : un burst publie a CHAQUE trame avec tache TX
         ↪  (mode TRX)\n");
        if (trx_page_mode()) { actif = 0; printf("  [rom-tx] mode TRX : les bursts sont publies depuis la page de
         ↪  l'extcode (tx_page_publier), pas du script TSP\n"); }
        else printf("  [rom-tx] burst montant final de la ROM (script TSP 0x3cbb) -> %s : %s\n", ROM_TX_SHM,
               actif ? "actif" : "coupe (MONTANT_ROM_TX=0)");
    }
    if (!actif) return;
    struct tsp_tx_burst b; bool nouveau = false;
```

```
    if (!tsp_tx_lire(dsp->data, &b, &nouveau)) return;
    if (!nouveau && !(chaque && g_tsp_fn.tache_tx && g_tsp_fn.tick == fn)) return;
    /* [2026-10-20] LE SCH DE L'EXTCODE (firmware trx) SORT A MOITIE FAUX SUR CE BANC. Mesure du jour
     * (air TRX -> mobile, 999 SB livres dans la fenetre SB du mobile, 0 decode, [a_sch] crc_ko a TOA 23) :
     * compares a gsm0503_sch_encode() des mots sch[0..1] de la page etendue (0x2000/0x2200 + 32), les
     * 39 premiers bits codes du burst sont EXACTS et les 39 derniers (bits 106..144 : haut de l'info,
     * parite CRC-10, queue) sont faux a ~50 %. L'encodeur SCH est dans l'extcode du DSP (dsp_extcode.c,
     * 0x015c), pas dans la ROM : c'est une instruction de ce code-la que le coeur emule mal (a trouver :
     * la boucle de parite). En attendant, prothese derriere MONTANT_ROM_TX_SB_RECODE (1 par defaut en
     * mode TRX, 0 pour la couper) : on recode le SCH a partir des mots sch de la page dont la premiere
     * moitie correspond bit a bit au burst capture, et on remplace la seconde moitie dans ce qu'on
     * publie. Le compteur dit combien de bits differaient : l'ecart a surveiller quand le coeur sera
     * corrige (0 = la prothese ne sert plus). */
    if (b.type == TSP_TX_SB && chaque) {
        static int recode = -1; static unsigned n_sb, n_recodes, n_sans_page, n_bits_faux;
       if (recode < 0) { const char *e = calypso_getenv("MONTANT_ROM_TX_SB_RECODE"); recode = !(e && *e == '0'); }
        if (recode) {
            n_sb++;
            /* Les deux pages portent chacune un mot sch : celui de ce SB et celui du SB d'avant (10 ou
             * 11 trames plus tot). La premiere moitie codee ne les distingue pas toujours (elle ne porte
             * pas T2 ni le bit bas de T1/T3') : on prend, parmi les pages dont la premiere moitie colle,
             * celle au fn le plus recent. Mesure du jour avec « la premiere qui colle » : un SB sur
             * trois portait le fn du SB precedent (fn decode 234601 puis 234233 a 10 trames d'ecart). */
            int fait = 0; uint32_t fn_best = 0; ubit_t code_best[78];
            for (unsigned base = 0x2000; base <= 0x2200; base += 0x200) {
                uint32_t sch = dsp->data[base + 32] | ((uint32_t)dsp->data[base + 33] << 16);
                uint8_t info[4] = { sch & 0xff, (sch >> 8) & 0xff, (sch >> 16) & 0xff, (sch >> 24) & 0xff };
                ubit_t code[78];
                if (gsm0503_sch_encode(code, info) < 0) continue;
                if (memcmp(code, b.bits + 3, 39) != 0) continue;          /* pas la page de ce burst */
                /* inverse de sb_build() (prim_bts.c) : t1&1 -> bit 23, t1&0x1fe -> bits 8..15, t1&0x600 -> bits
                 ↪  0..1,
                 * t2 -> bits 18..22, t3'&1 -> bit 24, t3'&6 -> bits 16..17, bsic -> bits 2..7.
                 * Verifie sur 0x005a1c1c = BSIC 7, T1 56, T2 22, T3 41 (le SB que le mobile a decode). */
                uint32_t t1 = ((sch >> 23) & 1) | ((sch >> 7) & 0x1fe) | ((sch & 3) << 9);
                uint32_t t2 = (sch >> 18) & 0x1f, t3 = (((sch >> 24) & 1) | (((sch >> 16) & 3) << 1)) * 10 + 1;
               /* 05.02 3.3.2.2.1 : FN = 51 ((T3 - T2) mod 26) + T3 + 51 x 26 x T1 (le premier jet avait T2 - T3 :
                 * fn decode qui RECULAIT de 51 par 51 trames, offset de calage faux) */
                uint32_t fnsb = 51 * 26 * t1 + 51 * ((t3 + 26 - t2) % 26) + t3;
              if (!fait || (int32_t)(fnsb - fn_best) > 0) { fn_best = fnsb; memcpy(code_best, code, sizeof code); }
                fait = 1;
            }
            if (fait) {
                int faux = 0;
                for (int k = 0; k < 39; k++) faux += (b.bits[106 + k] != code_best[39 + k]);
                n_bits_faux += (unsigned)faux;
                if (faux) { memcpy(b.bits + 106, code_best + 39, 39); n_recodes++; }
                /* [2026-10-20] CALAGE DU MONTANT DU TRX. Ce SB (fn GSM fn_best) part au tick `fn` : on
                 * connait donc l'horloge du firmware trx. Le BSP de cette instance recoit les bursts du
                 * mobile tagues de leur fn GSM (a_a5fn du mobile) et doit les jouer a la trame ou le
                 * firmware les attend (calypso_bsp_trx_calage, CALYPSO_BSP_TRX=1). On n'applique que
                 * quand deux SB consecutifs donnent le meme ecart (les deux pages ne portent pas
                 * toujours le sch de CE burst : fn decode faux d'un SB sur trois sinon). */
                static int32_t ecart_prec = INT32_MIN; static unsigned n_cal;
                int32_t ecart = (int32_t)(fn_best - fn);
                if (ecart == ecart_prec) {
                    if (calypso_bsp_trx_calage(fn_best, fn) && (++n_cal <= 5 || n_cal % 500 == 0))
                        printf("  [rom-tx] calage TRX : SB fn=%u emis au tick %u (fn - tick = %d) -> BSP montant
↪  cale (%u)\n", fn_best, fn, ecart, n_cal);
                }
                ecart_prec = ecart;
            } else n_sans_page++;
            if (n_sb <= 3 || n_sb % 1000 == 0)
                printf("  [rom-tx] SB recode (prothese SCH) : %u SB, %u recodes, %u sans page sch, %.1f bits
↪  faux/SB en moyenne\n",
                       n_sb, n_recodes, n_sans_page, n_sb ? (double)n_bits_faux / n_sb : 0.0);
```

```
        }
    }
    if (b.type != TSP_TX_NB) g_rom_emis_tick = fn;
    if (g_rach_pend.actif && b.type != TSP_TX_NB && (uint32_t)(fn - g_rach_pend.tick) <= 2)
        rach_pend_publier(fn, "emission ROM");
    int src = 0;
    uint32_t fn_gsm = tx_tsp_fn(fn, &src);
    uint8_t octet_src = (uint8_t)src;
    /* [2026-10-15] bit 7 : burst NB chiffre par la ROM -- le coprocesseur A5 a
     * calcule un flux A5/1-2 pour ce fn (a une trame pres : recalage SACCH,
     * calypso_a5.c). Lu par le pont pour le compteur « A5 ul rom= ». */
    {
        uint32_t fa;
        if (b.type == TSP_TX_NB && calypso_a5_dernier(&fa)) {
            int32_t d = (int32_t)((fn_gsm + GSM_HYPERFRAME - fa) % GSM_HYPERFRAME);
            if (d <= 1 || d >= (int32_t)GSM_HYPERFRAME - 1) octet_src |= 0x80;
        }
    }
    /* [2026-10-20] bits 4-6 : l'intervalle sur lequel le firmware emet (TCH TS1..7 ; 0 = TS0) : l'air
     * porte le burst a cet intervalle du TRX */
    octet_src |= (uint8_t)((montant_tn_dedie() & 7) << 4);
    if (!ring_publier(b.type, b.bits, b.tsc, b.offset, octet_src, fn_gsm)) { actif = 0; return; }
    uint32_t w_compte = g_ring_w;
    if (b.type == TSP_TX_RACH) n_rach++; else if (b.type == TSP_TX_NB) n_nb++; else if (b.type == TSP_TX_SB)
     ↪  n_sb++; else if (b.type == TSP_TX_DUMMY) n_dummy++; else if (b.type == TSP_TX_FB) n_fb++; else
     ↪  n_inconnu++;
    static int nlog;
    if (nlog++ < 6 || (b.type == TSP_TX_RACH && n_rach <= 3))
        printf("  [rom-tx] #%u tick=%u fn=%u (%s) %s%s offset=%d @%04x : %s...\n", w_compte, fn, fn_gsm,
               src == 1 ? "a_a5fn" : src == 2 ? "tick+dec" : src == 3 ? "calage BSP" : "TICK NU",
              b.type == TSP_TX_RACH ? "RACH" : b.type == TSP_TX_NB ? "NB" : b.type == TSP_TX_SB ? "SB" : b.type ==
 ↪  TSP_TX_DUMMY ? "DUMMY" : b.type == TSP_TX_FB ? "FB" : "?", b.type == TSP_TX_NB ? (char[]){' ', 'T', 'S', 'C',
 ↪  (char)('0' + b.tsc), 0} : "",
               b.offset, b.adresse, (char[]){ '0' + b.bits[0], '0' + b.bits[1], '0' + b.bits[2], '0' + b.bits[3],
 ↪  '0' + b.bits[4], '0' + b.bits[5], '0' + b.bits[6], '0' + b.bits[7], 0 });
    if ((n_rach + n_nb + n_sb + n_dummy + n_fb + n_inconnu) % 2000 == 0)
        printf("  [rom-tx] %u bursts publies : RACH %u, NB %u, SB %u, DUMMY %u, FB %u, inconnus %u\n", n_rach +
 ↪  n_nb + n_sb + n_dummy + n_fb + n_inconnu, n_rach, n_nb, n_sb, n_dummy, n_fb, n_inconnu);
}

/* [2026-10-21 19:00] RACH VERS L'AIR (CALYPSO_RACH_VERS_AIR=1, Magnetite). La capture au script TSP de la
 * ROM (0x3cbb/0x3ccd, chemin du firmware osmocom) ne voit pas le burst d'acces du firmware TI : montant.c
 * detectait bien les RACH (d_task_ra, ra/bsic) mais ils n'allaient qu'a /dev/shm/calypso_rach, que l'air
 * ne lit pas -> B->A 0, pas de CHAN RQD, « Network forbidden ». On code le burst d'acces (05.03,
 * gsm0503_rach_ext_encode) et on le publie dans calypso_tx_rom, date de la trame BTS du calage. */
/* [2026-10-21 19:40] a_pm JAMAIS NUL (CALYPSO_BSP_BRUIT_FOND=1, Magnetite). La L1 TI lit pm = a_pm >> 5
 * (l1ddsp_meas_read) et compte chaque pm nul comme une panne (l1_check_pm_error : 100 -> recovery, DSP
 * relance). Le modele de puissance du banc (calypso_bsp_rssi_apm) ne passe que sur les ecritures du DSP ;
 * un a_pm laisse a 0 par la ROM partait tel quel. Avant de rendre la trame a l'ARM, chaque mot a_pm nul
 * des deux pages R recoit le modele (plancher 64). */
/* [2026-10-22 00:20] sonde [apmT] (voir calypso_apm_trace_on) : pages R au TICK (ce que le DSP a laisse
 * apres la trame precedente et ce que la L1 a remis a zero), puis au DONE avant/apres le modele. */
static uint16_t g_apm_tick_r[2][4];
static void apm_trace_tick(const uint16_t *api_ram, uint32_t tick, uint32_t b)
{
    for (int pg = 0; pg < 2; pg++) {
        const uint16_t *r = &api_ram[API_R_PAGE(pg) / 2];
        g_apm_tick_r[pg][0] = r[4]; g_apm_tick_r[pg][1] = r[12]; g_apm_tick_r[pg][2] = r[13]; g_apm_tick_r[pg][3]
 ↪  = r[14];
    }
    (void)tick; (void)b;
}
static void apm_trace_done(const uint16_t *api_ram, uint16_t avant[2][4]);
static void apm_jamais_nul_corps(uint16_t *api_ram);
static void apm_jamais_nul(uint16_t *api_ram)
{
```

```
    uint16_t done_avant[2][4];
    for (int pg = 0; pg < 2; pg++) {
        const uint16_t *r = &api_ram[API_R_PAGE(pg) / 2];
       done_avant[pg][0] = r[4]; done_avant[pg][1] = r[12]; done_avant[pg][2] = r[13]; done_avant[pg][3] = r[14];
    }
    apm_jamais_nul_corps(api_ram);
    apm_trace_done(api_ram, done_avant);
}
static void apm_jamais_nul_corps(uint16_t *api_ram)
{
    static int on = -1;
    if (on < 0) { const char *e = calypso_getenv("CALYPSO_BSP_BRUIT_FOND"); on = (e && *e == '1'); }
    if (!on) return;
    for (int pg = 0; pg < 2; pg++) {
        uint16_t *a = &api_ram[(API_R_PAGE(pg) + RP_A_PM) / 2];
        uint16_t avant[3] = { a[0], a[1], a[2] };
        int depile[3] = { -2, -2, -2 };
        /* [2026-10-21] une mesure par fenetre, dans l'ordre des fenetres livrees (file, voir calypso_bsp.c) */
        /* [2026-10-22 00:20] les mots 12..14 sont ecrits par le DSP et servis dans l'ordre par c54x_mem.c
         * (calypso_bsp_rssi_apm_suivante) ; un zero ici est une page remise a zero par la L1 : plancher sans
         * consommer la file (jamais 0 : l1_check_pm_error). */
        for (int k = 0; k < 3; k++)
            if ((a[k] >> 5) == 0) a[k] = calypso_bsp_rssi_apm();
        /* [2026-10-21 23:58] SONDE [apm-fort] : PM remplacees avec le MAV d'un VRAI burst (veille, ARFCN 1),
         * ou trame ou une fenetre en a recu un. 200 lignes. */
        /* bilan toutes les 255 trames (periode de paging du banc) : PM remplacees avec un burst, du bruit,
         * ou sans fenetre en file -- c'est la veille qui compte, la sonde detaillee s'epuise au balayage */
        {
            static unsigned long b_burst, b_bruit, b_vide, b_t0; static uint32_t b_tick;
            for (int k = 0; k < 3; k++) {
                if (depile[k] >= 5000) b_burst++; else if (depile[k] >= 0) b_bruit++; else if (depile[k] == -1)
                 ↪  b_vide++;
            }
            if (g_c54x_exe_fn - b_tick >= 255) {
                b_tick = g_c54x_exe_fn;
                printf("  [apm-bilan] fn=%u sur 255 trames : PM <- burst %lu, <- bruit %lu, file vide %lu\n",
                       g_c54x_exe_fn, b_burst, b_bruit, b_vide);
                b_burst = b_bruit = b_vide = 0; (void)b_t0;
            }
        }
        static unsigned n_fort;
        if ((calypso_bsp_pm_mav_fort() || depile[0] >= 5000 || depile[1] >= 5000 || depile[2] >= 5000) &&
            (n_fort++ < 20 || n_fort % 50 == 0)) {
            char t[256]; calypso_bsp_pm_mav_texte(t, sizeof t);
            uint16_t *r = &api_ram[API_R_PAGE(pg) / 2];
            printf("  [apm-fort] page=%d d_task_md=%04x d_task_d=%04x a_pm %04x->%04x %04x->%04x %04x->%04x
↪  depiles %d %d %d | mav %s\n",
                   pg, r[4], r[0], avant[0], a[0], avant[1], a[1], avant[2], a[2], depile[0], depile[1],
↪  depile[2], t);
        }
    }
    /* [2026-10-21 20:00] MESURE PLEINE LISTE (RXL_R 548) : la L1 TI (DSP 36) programme d_task_md = 0x200 + n
     * (n <= 8 mesures, l1_sync.c) et lit les n PM dans les mots 12..12+n-1 de la page R (l1ddsp_meas_read,
     * nbmeas = 8 vu au gdb) : au-dela des 3 mots a_pm, ce sont les mots de a_sch. Le modele de puissance du
     * banc ne couvrait que les mots 12..14 : 15..19 restaient a 0 (vu au moniteur, d_task_md 0x0208), 5 PM
     * nuls sur 8 a chaque trame du balayage -> recovery. La page R reprend d_task_md (mot 4) : on s'en sert. */
    /* [2026-10-21 21:50] MODE DEDIE : les taches SDCCH/SACCH (ADL 22, DDL 24 cote L1 TI) lisent la puissance dans
     * a_serv_demod[D_PM], mot 9 de la page R pour le DSP 36 (l1_defty.h : a_serv_demod 8..11). Le DSP emule l'y
     * laissait a 0 apres l'IMM ASS : « > PM 1778 22 », « > PM 1808 24 » a chaque trame, ABORT, retour en veille.
     * Meme plancher que a_pm, des qu'une tache descendante (d_task_d, mot 0) est rapportee. */
    for (int pg = 0; pg < 2; pg++) {
        uint16_t *r = &api_ram[API_R_PAGE(pg) / 2];
       /* [2026-10-22 00:58] + les taches de synchro (d_task_md : SB2, SBCONF...) : la L1 TI lit aussi leur PM dans
        * a_serv_demod[D_PM] (l1_cmplx.c, l1s_read_mon_result) ; mot 9 laisse a 0 -> 4 « > PM » (tache 8) a chaque
        * relecture SB de la cellule en veille, compteur cumule -> RECOVERY au bout de quelques minutes. */
       if (((r[0] & 0x7fff) || (r[4] & 0xff)) && (r[9] >> 5) == 0) r[9] = calypso_bsp_rssi_apm();
```

```
    }
    for (int pg = 0; pg < 2; pg++) {
        uint16_t *r = &api_ram[API_R_PAGE(pg) / 2];
        uint16_t md = r[4];
        if (!(md & 0x200)) continue;
        int n = md & 0xff; if (n > 8) n = 8;
        uint16_t avant[8];
        for (int k = 0; k < n; k++) avant[k] = r[12 + k];
        for (int k = 3; k < n; k++)      /* 15..19 : pas ecrits par le DSP sur ce chemin, suite de la file */
            if ((r[12 + k] >> 5) == 0) r[12 + k] = calypso_bsp_rssi_apm_suivante(NULL);
        /* [2026-10-21 23:55] SONDE [apm] : en veille (Magnetite) la mesure BA de l'ARFCN 1 tombe au plancher
         * meme avec un MAV par fenetre. PM rendues par le DSP (avant) et apres le modele, MAV des fenetres. */
        static unsigned long n_apm;
        if (++n_apm <= 60 || n_apm % 200 == 0) {
            char t[256]; calypso_bsp_pm_mav_texte(t, sizeof t);
            printf("  [apm] #%lu page=%d md=%04x n=%d :", n_apm, pg, md, n);
            for (int k = 0; k < n; k++) printf(" %04x->%04x", avant[k], r[12 + k]);
            printf(" | mav %s\n", t);
        }
    }
}

static void apm_trace_done(const uint16_t *api_ram, uint16_t avant[2][4])
{
    if (!calypso_apm_trace_on()) return;
    const uint16_t *w0 = &api_ram[0], *w1 = &api_ram[0x14];
    printf("  [apmT] fn=%u dsp_page=%04x W md %04x/%04x |", g_c54x_exe_fn,
           api_ram[(API_NDB + NDB_D_DSP_PAGE) / 2], w0[4], w1[4]);
    for (int pg = 0; pg < 2; pg++) {
        const uint16_t *r = &api_ram[API_R_PAGE(pg) / 2];
        printf(" R%d tick md=%04x pm=%04x,%04x,%04x done md=%04x pm=%04x,%04x,%04x -> %04x,%04x,%04x |", pg,
               g_apm_tick_r[pg][0], g_apm_tick_r[pg][1], g_apm_tick_r[pg][2], g_apm_tick_r[pg][3],
               avant[pg][0], avant[pg][1], avant[pg][2], avant[pg][3], r[12], r[13], r[14]);
    }
    printf("\n");
}

/* [2026-10-22 14:25] LE RACH DATE PAR SON EMISSION. Run de 14:21 : RA_C = tick ou la ROM EMET le burst (script TSP,
 * [rom-tx] 1304/1549/1761) + l'offset de livraison (762) : 2066/2311/2523, exact les trois fois. Le tick ou la
 ↪  tache
 * est POSEE (montant.c) precede l'emission de 0 ou 1 trame selon l'acces : c'etait toute la gigue (RACH datés
 ↪  fn-1,
 * IMM ASS ignorees, SDCCH en Timeout a la BSC). Le burst attend donc l'emission de la ROM (tx_tsp_publier) ; sans
 * emission sous 3 ticks, il part au tick de pose + 1. CALYPSO_RACH_EMISSION=0 : ancienne voie. */
static int rach_emission_on(void)
{
    static int on = -1;
    if (on < 0) { const char *e = calypso_getenv("CALYPSO_RACH_EMISSION"); on = !(e && *e == '0'); }
    return on;
}
static void rach_pend_publier(uint32_t tick_emis, const char *quoi)
{
    if (!g_rach_pend.actif) return;
    g_rach_pend.actif = false;
    uint32_t f; int src = 3;
    if (!calypso_bsp_fn_bts_du_tick(tick_emis, &f)) { f = tx_tsp_fn(tick_emis, &src); }
    rach_corr(g_rach_pend.tick);   /* file des RACH (lecture RA_C) */
    if (!ring_publier(TSP_TX_RACH, g_rach_pend.bits, -1, 8, (uint8_t)src, f)) return;
    static unsigned n;
    if (++n <= 40 || n % 100 == 0)
        printf("  [rach-air] RACH pose tick=%u, emis tick=%u (%s) -> fn BTS %u publie vers l'air [%u]\n",
               g_rach_pend.tick, tick_emis, quoi, f, n);
}
static void rach_pend_echeance(uint32_t tick)
{
    if (g_rach_pend.actif && (uint32_t)(tick - g_rach_pend.tick) >= 3)
```

```
        rach_pend_publier(g_rach_pend.tick + 1, "sans emission vue, pose+1");
}
void tx_rach_vers_air(uint8_t ra, uint8_t bsic, uint32_t tick)
{
    static int actif = -1;
    if (actif < 0) { const char *e = calypso_getenv("CALYPSO_RACH_VERS_AIR"); actif = (e && *e == '1'); }
    if (!actif) return;
    /* [2026-10-21 20:55] le BURST D'ACCES ENTIER (05.02 5.2.7), comme le chemin ROM (tsp_tx.c, offset 8) :
     * 8 bits de queue etendue, 41 de synchro, les 36 codes, 3 de queue. gsm0503_rach_ext_encode ne rend que
     * les 36 bits codes : publies seuls en tete, le TRX n'avait aucune synchro a correler (transceiver :
     * aucun « TOA : », osmo-bts rach:rcvd 0). */
    static const char queue[] = "00111010", sync[] = "01001011011111111100110011010101000111000";
    uint8_t code[36], bits[148] = {0};
    if (gsm0503_rach_ext_encode(code, ra, bsic, false) < 0) return;
    for (int i = 0; i < 8; i++) bits[i] = (uint8_t)(queue[i] - '0');
    for (int i = 0; i < 41; i++) bits[8 + i] = (uint8_t)(sync[i] - '0');
    memcpy(bits + 49, code, 36);
    if (rach_emission_on()) {
        if (g_rach_pend.actif) rach_pend_publier(g_rach_pend.tick + 1, "remplace");
        g_rach_pend.actif = true; g_rach_pend.tick = tick; memcpy(g_rach_pend.bits, bits, 148);
        if (g_rom_emis_tick == tick) rach_pend_publier(tick, "emission ROM, meme tick");   /* deja emis (14:25,
         ↪  RACH #1) */
        return;
    }
    int src = 0;
    rach_dfn_noter(dsp_global_tx, tick);
    uint32_t fn_gsm = tx_tsp_fn(tick, &src);
    if (!ring_publier(TSP_TX_RACH, bits, -1, 8, (uint8_t)src, fn_gsm)) return;
    static unsigned n;
    if (++n <= 20 || n % 100 == 0)
        printf("  [rach-air] RACH ra=0x%02x bsic=%u tick=%u -> fn BTS %u (%s) publie vers l'air [%u]\n",
               ra, bsic, tick, fn_gsm, src == 1 ? "a_a5fn" : src == 2 ? "tick+dec" : src == 3 ? "calage BSP" : src
↪  == 4 ? "fn du mobile (SB)" : "TICK NU", n);
}

/* [2026-10-20] MODE TRX : PUBLICATION DEPUIS LA PAGE DE L'EXTCODE. Le firmware trx remplit tx[p] (cmd,
 * pointeur de donnees) pour chaque slot p de sa fenetre TX (prim_bts.c patche : slot p = TS (tx_start + p)),
 * et l'extcode du DSP les module ; la ROM emulee ne joue qu'un burst par trame (le script TSP) et le
 * codeur SCH de l'extcode est faux sur ce coeur. On n'a pas besoin de la ROM : les bits se deduisent de
 * la page (FB = 148 zeros, dummy = 05.02 5.2.6, SB = gsm0503_sch_encode(sch[0..1]), NB = 8 mots de
 * donnees + TSC du NDB). Appele a la phase A du tick T+1 avec la page que le DSP a traitee au tick T
 * (index d_dsp_page) : les bursts publies portent fn = T. Le SB cale le BSP montant (calypso_bsp_trx_calage).
 * CALYPSO_TRX_TX_SLOTS (1) : nombre de slots publies ; CALYPSO_TRX_TX_START (0) : TS du slot 0. */
static void tx_page_publier(const C54xState *dsp, uint32_t tick, int w_page)
{
    static int nslots = -1, tx_start = 0; static unsigned n_pub, n_cal; static int32_t ecart_prec = INT32_MIN;
    static const char DUMMY_148[] =
     ↪  "00011111011011101100000101001001110000010010001000000011111000111000101110001011100010101110100101000110011001110011111
     ↪  ;
    static const char *const TSC8[8] = {
        "00100101110000100010010111", "00101101110111100010110111", "01000011101110100100001110",
        "01000111101101000100011110", "00011010111001000001101011", "01001110101100000100111010",
        "10100111110110001010011111", "11101111000100101110111100" };
    if (nslots < 0) {
        const char *e = calypso_getenv("CALYPSO_TRX_TX_SLOTS"); nslots = (e && *e) ? atoi(e) : 1;
        if (nslots < 1) nslots = 1;
        if (nslots > 8) nslots = 8;
        e = calypso_getenv("CALYPSO_TRX_TX_START"); tx_start = (e && *e) ? atoi(e) & 7 : 0;
        printf("  [rom-tx] publication depuis la page de l'extcode : %d slot(s) TX a partir du TS%d\n", nslots,
↪  tx_start);
    }
    const uint16_t *db = &dsp->data[w_page ? 0x2200 : 0x2000];
    uint32_t fn_tag = tick - 1;                       /* la trame DSP ou ces bursts sont partis */
    int tsc = dsp->data[0x2401] & 7;
    for (int p = 0; p < nslots; p++) {
        unsigned cmd = db[2 * p], ptr = db[2 * p + 1];
```

```
        uint8_t bits[148]; int type, offset = 0, tscp = -1;
        memset(bits, 0, sizeof bits);
        switch (cmd) {
        case 1: type = TSP_TX_FB; break;                                        /* 148 zeros */
        case 2: {                                                               /* SB */
            uint32_t sch = db[32] | ((uint32_t)db[33] << 16);
            uint8_t info[4] = { sch & 0xff, (sch >> 8) & 0xff, (sch >> 16) & 0xff, (sch >> 24) & 0xff };
            ubit_t code[78];
            if (gsm0503_sch_encode(code, info) < 0) continue;
            static const char SYNC_SB[] = "1011100101100010000001000000111100101101010001010111011000011011";
            for (int k = 0; k < 39; k++) { bits[3 + k] = code[k]; bits[106 + k] = code[39 + k]; }
            for (int k = 0; k < 64; k++) bits[42 + k] = (uint8_t)(SYNC_SB[k] - '0');
            type = TSP_TX_SB;
            /* calage du montant : ce SB (fn GSM) est parti au tick fn_tag ; applique quand deux SB d'affilee
             ↪  concordent */
            uint32_t t1 = ((sch >> 23) & 1) | ((sch >> 7) & 0x1fe) | ((sch & 3) << 9);
            uint32_t t2 = (sch >> 18) & 0x1f, t3 = (((sch >> 24) & 1) | (((sch >> 16) & 3) << 1)) * 10 + 1;
            uint32_t fnsb = 51 * 26 * t1 + 51 * ((t3 + 26 - t2) % 26) + t3;
            int32_t ecart = (int32_t)(fnsb - fn_tag);
            /* le calage garde la relation mesuree au script TSP (fn du SB = tick d'emission - 1) : on passe
             * le tick de la phase A (T + 1), pas fn_tag, pour que CALYPSO_BSP_TRX_DELTA garde son sens */
            if (ecart == ecart_prec && calypso_bsp_trx_calage(fnsb, tick) && (++n_cal <= 3 || n_cal % 500 == 0))
                printf("  [rom-tx] calage TRX (page) : SB fn=%u emis au tick %u (fn - tick = %d) -> BSP montant
↪  cale (%u)\n", fnsb, fn_tag, ecart, n_cal);
            ecart_prec = ecart;
            break; }
        case 3: for (int k = 0; k < 148; k++) bits[k] = (uint8_t)(DUMMY_148[k] - '0'); type = TSP_TX_DUMMY; break;
        case 5: {                                                               /* NB : 116 bits + TSC */
            if (ptr + 8 > 0x200) continue;
            const uint16_t *w = &db[ptr];
            uint8_t pb[15];
            for (int k = 0; k < 7; k++) { pb[2 * k] = w[k] >> 8; pb[2 * k + 1] = w[k] & 0xff; }
            pb[14] = w[7] >> 8;
           for (int k = 0; k < 58; k++) bits[3 + k] = (pb[k >> 3] >> (7 - (k & 7))) & 1;          /* 57 data + hl */
            for (int k = 0; k < 58; k++) bits[87 + k] = (pb[(58 + k) >> 3] >> (7 - ((58 + k) & 7))) & 1;   /* hu +
             ↪  57 data */
            for (int k = 0; k < 26; k++) bits[61 + k] = (uint8_t)(TSC8[tsc][k] - '0');
            type = TSP_TX_NB; tscp = tsc;
            break; }
        default: continue;                                                      /* 0 (rien) ou AB */
        }
        uint8_t src = (uint8_t)(((tx_start + p) & 7) << 4);
        ring_publier(type, bits, tscp, offset, src, fn_tag);
        n_pub++;
        if (n_pub <= 4 || n_pub % 20000 == 0)
            printf("  [rom-tx] page : #%u tick=%u slot %d (TS%d) cmd=%u -> %s\n", n_pub, fn_tag, p, (tx_start + p)
↪  & 7, cmd,
                   type == TSP_TX_FB ? "FB" : type == TSP_TX_SB ? "SB" : type == TSP_TX_DUMMY ? "DUMMY" : "NB");
    }
}

static void tx_rom_publier(const C54xState *dsp, uint32_t fn)
{
    static int actif = -1, fd = -1;
    static uint16_t prev;
    static unsigned masque;
    static uint8_t l2[23], bits[4 * 116], flux[4 * 116];
    static uint32_t w_compte, fn0;
    if (actif < 0) {
        const char *e = calypso_getenv("MONTANT_ROM_UL"); actif = !(e && *e == '0');
        prev = dsp->data[ROM_UL_PTR];
        printf("  [rom-ul] bursts xCCH montants de la ROM (0x3f8a) -> %s : %s\n", ROM_UL_SHM, actif ? "actif" :
↪  "coupe (MONTANT_ROM_UL=0)");
    }
    if (!actif) return;
    uint16_t p = dsp->data[ROM_UL_PTR];
```

```
    if (p == prev) return;
    prev = p;
    if (p < 0x4280 || p >= 0x42a0 || (p - 0x4280) % 8) { masque = 0; return; }
    unsigned b = ((p - 0x4280) / 8 + 3) & 3;
    if (b == 0) { masque = 0; memcpy(l2, g_romul_l2_pose, 23); fn0 = fn; }
    uint8_t *o = &bits[b * 116], *x = &flux[b * 116];
    for (int i = 0; i < 116; i++) {
        /* 57 donnees, hl, hu, 57 donnees. Les 114 bits de donnees occupent les mots 0..7 MSB d'abord
         * (bits 0..113) ; hl/hu sont les 2 bits BAS du mot 7 (releve : 0x3f91 = 0x?003 sur SDCCH, ou
         * hl = hu = 1), soit k = 126 et 127 -- pas 114/115 (bits 13/12, toujours 0 : revue 2026-10-03). */
        int k = i < 57 ? i : (i < 59 ? 126 + (i - 57) : i - 2);
        o[i] = (dsp->data[ROM_UL_BURST + k / 16] >> (15 - k % 16)) & 1;
        x[i] = (dsp->data[ROM_UL_FLUX + k / 16] >> (15 - k % 16)) & 1;
    }
    masque |= 1u << b;
    if (b != 3) return;
    if (masque != 0xf) { masque = 0; return; }
    masque = 0;
    if (fd < 0) {
        fd = open(ROM_UL_SHM, O_RDWR | O_CREAT, 0666);
        if (fd < 0 || ftruncate(fd, 8 + ROM_UL_SLOTS * ROM_UL_SLOT_SZ) < 0) { actif = 0; return; }
        uint32_t h[2] = { 0, ROM_UL_SLOTS };
        if (pwrite(fd, h, sizeof h, 0) < 0) { actif = 0; return; }
    }
    uint8_t slot[ROM_UL_SLOT_SZ] = {0};
    w_compte++;
    memcpy(slot, &w_compte, 4); memcpy(slot + 4, &fn0, 4);
    memcpy(slot + 8, l2, 23); memcpy(slot + 32, bits, sizeof bits); memcpy(slot + 32 + sizeof bits, flux, sizeof
 ↪  flux);
    off_t off = 8 + (off_t)((w_compte - 1) % ROM_UL_SLOTS) * ROM_UL_SLOT_SZ;
    if (pwrite(fd, slot, sizeof slot, off) < 0 || pwrite(fd, &w_compte, 4, 0) < 0) return;
    static int nlog;
    if (nlog++ < 4) printf("  [rom-ul] bloc %u publie (fn burst0=%u, L2=%02x %02x %02x..)\n", w_compte, fn0,
     ↪  l2[0], l2[1], l2[2]);
}

static int c54x_run_profile(C54xState *dsp, int budget)
{
    int fait = 0;
    pcc_armer();
    while (fait < budget && !dsp->idle && dsp->running) {
        int n = budget - fait < PROFIL_SEAU ? budget - fait : PROFIL_SEAU;
        static long minfn = -1;
        if (minfn < 0) { const char *e = calypso_getenv("PONT_TRACE_MINFN"); minfn = (e && *e) ? atol(e) : 0; }
        if (g_trace_reste > 0 && !g_trace_ouverte && g_trace_pc_hi && (long)g_c54x_exe_fn >= minfn &&
            (dsp->pc & 0xffff) >= g_trace_pc_lo && (dsp->pc & 0xffff) <= g_trace_pc_hi && g_trace_f) {
            g_trace_ouverte = true;
            trace_cellules(dsp, true);
            fprintf(g_trace_f, "# PC=%04x dans [%04x,%04x] a fn=%u ; IMR=%04x IFR=%04x\n",
                    dsp->pc & 0xffff, g_trace_pc_lo, g_trace_pc_hi, g_c54x_exe_fn, dsp->imr, dsp->ifr);
            printf("pont : trace ouverte sur PC=%04x fn=%u\n", dsp->pc & 0xffff, g_c54x_exe_fn);
        }
        if (g_trace_reste > 0 && g_trace_ouverte) n = 1;
        if (g_pcc_k > 0) {
            n = 1;
            uint16_t pc = dsp->pc & 0xffff;
            for (int i = 0; i < g_pcc_k; i++) if (g_pcc[i] == pc) g_pcc_n[i]++;
        }
        if (n == 1) trace_pas(dsp);
        /* Budget accounting uses the count c54x_run returns, not the insn_count
         * delta: RPT iterations do not bump insn_count, so a step or a whole
         * bucket landing inside a several-hundred-iteration "rpt *(lk); nop"
         * reads as a stalled DSP and aborts the frame mid-ISR. */
        int ex = c54x_run(dsp, n);
        fait += ex;
        g_profil[(dsp->pc & 0xffff) / PROFIL_SEAU]++;
```

```
        if ((dsp->pc & 0xffff) > g_profil_hw) g_profil_hw = dsp->pc & 0xffff;
        if (ex <= 0) break;
    }
    return fait;
}

static void profil_publier(void)
{
    unsigned long tot = 0; unsigned n = 0;
    for (unsigned i = 0; i < 0x10000 / PROFIL_SEAU; i++) tot += g_profil[i];
    if (!tot) return;
    printf("  profil PC (%lu tranches, high-water 0x%04x) :", tot, g_profil_hw);
    /* the 12 hottest buckets, in decreasing order */
    for (int k = 0; k < 12; k++) {
        unsigned best = 0; unsigned long bv = 0;
        for (unsigned i = 0; i < 0x10000 / PROFIL_SEAU; i++)
            if (g_profil[i] > bv) { bv = g_profil[i]; best = i; }
        if (!bv) break;
        printf(" %04x:%lu%%", best * PROFIL_SEAU, bv * 100 / tot);
        g_profil[best] = 0; n++;
    }
    printf("\n");
    memset(g_profil, 0, sizeof(g_profil)); g_profil_hw = 0;
}

/* One TDMA frame, in the order calypso_tdma_tick() uses: DMA tick, then boot
 * (run to the first IDLE) while init is pending, then the TPU-frame interrupt
 * if IMR arms it followed by one run budget. Returns the PONT_DONE flags and
 * the executed instruction count in *insns. */
/* [2026-09-20] MID-FRAME INJECTION (PONT_RX_MODE=milieu, the default). The ROM
 * arms DMA2 in its frame ISR and the receiver only keeps samples while a
 * window is open (calypso_rif: no window, no sample). Delivering the burst
 * AFTER the frame's run (the old PONT_RX_APRES=1) found the channel closed on
 * most frames: measured 13 transfers on one frame in six, the FB block counter
 * starved, TOA = 1251 for an FCCH four frames away. The replay bench delivers
 * after a short slice of the frame, once the ISR has armed; the bridge now does
 * the same: budget/8, inject (synthetic cell and UDP bursts), then the rest. */
static bool g_tick_irq_trame = true;   /* TICK.b bit 16 : l'ARM a arme l'interruption trame du DSP */
static struct { bool actif; const char *iq_mode; int amp; uint32_t fn; unsigned long *injectes; bool udp; char
 ↪  dernier_type; int dernier_n_iq; } g_inj;
static void injecter_burst(C54xState *dsp, const char *iq_mode, int amp, uint32_t fn, unsigned long *injectes);

uint16_t prog_fetch(C54xState *s, uint16_t pc);
static unsigned g_flags_entree; static uint16_t g_data_avant_b[C54X_DATA_SIZE];
static uint16_t g_snap_2be2[148]; static int g_snap_ok;
static uint16_t g_snap_2a00[456]; static int g_snap_2a00_ok;
#define s_or_dsp_st0(d) ((d)->st0)
static int16_t g_dernier_iq[2 * 256]; static int g_dernier_n_iq;
static uint16_t g_daram_apres_dma[384]; static uint16_t g_daram_aad;
/* phase : 0 = whole frame ; 1 = frame ISR only, up to the arming of the RX
 * window (then DONE|PHASE_A and wait for PONT_GO) ; 2 = burst delivery and
 * the rest of the frame. See CALYPSO_PONT_TICK_DEUX_PHASES. */
/* [2026-09-23] ENREGISTREUR DU TCH (cote ARM). Les ecritures de l ARM dans
 * l API RAM, par difference avec l instantane pris quand le DSP a rendu la
 * main : 'A', tick BE32, fenetre (0 = avant le TICK, 1 = entre phase A et GO),
 * nombre BE16, puis (mot BE16, valeur BE16). Et a chaque TICK : 'T', tick BE32,
 * drapeaux (bit0 irq trame, bit1 deux phases), budget BE32. Meme fichier que
 * les livraisons d I/Q du BSP (calypso_bsp_enreg_fichier), meme garde : TCH
 * actif, CALYPSO_REJEU_ENREG=0 coupe. Rejoue par tools/rejeu_banc.c. */
static uint16_t g_enreg_api[CALYPSO_API_WORDS];
static bool g_enreg_api_ok;
static void enreg_api_prendre(const uint16_t *api_ram)
{
    memcpy(g_enreg_api, api_ram, sizeof g_enreg_api);
    g_enreg_api_ok = true;
}
```

```
static void enreg_api_diff(const uint16_t *api_ram, uint32_t tick, int fenetre)
{
    FILE *f = calypso_bsp_enreg_fichier();
    if (!f || !g_enreg_api_ok) return;
    static uint8_t buf[8 + 4 * CALYPSO_API_WORDS];
    unsigned n = 0;
    for (unsigned i = 0; i < CALYPSO_API_WORDS; i++) {
        if (api_ram[i] == g_enreg_api[i]) continue;
        uint8_t *q = buf + 8 + 4 * n++;
        q[0] = i >> 8; q[1] = i; q[2] = api_ram[i] >> 8; q[3] = api_ram[i];
    }
    buf[0] = 'A'; buf[1] = tick >> 24; buf[2] = tick >> 16; buf[3] = tick >> 8; buf[4] = tick;
    buf[5] = (uint8_t)fenetre; buf[6] = n >> 8; buf[7] = n;
    fwrite(buf, 1, 8 + 4 * n, f);
    fflush(f);
}
/* Etat de depart, une fois, au premier TICK enregistre (DSP rendu, au repos) :
 * 'S' API RAM complete, 'D' registres puis memoire de donnees du C54x. */
typedef struct {
    int64_t a, b; uint16_t ar[8], t, trn, sp, bk, brc, rsa, rea, st0, st1, pmst, imr, ifr, xpc;
    uint32_t pc; uint8_t idle, running;
} EnregRegs;
static void enreg_base(FILE *f, const C54xState *dsp, const uint16_t *api_ram, uint32_t tick)
{
    uint8_t h[5] = { 'S', tick >> 24, tick >> 16, tick >> 8, tick };
    fwrite(h, 1, 5, f);
    fwrite(api_ram, sizeof(uint16_t), CALYPSO_API_WORDS, f);
    EnregRegs r = { .a = dsp->a, .b = dsp->b, .t = dsp->t, .trn = dsp->trn, .sp = dsp->sp, .bk = dsp->bk,
                    .brc = dsp->brc, .rsa = dsp->rsa, .rea = dsp->rea, .st0 = dsp->st0, .st1 = dsp->st1,
                    .pmst = dsp->pmst, .imr = dsp->imr, .ifr = dsp->ifr, .xpc = dsp->xpc, .pc = dsp->pc,
                    .idle = dsp->idle, .running = dsp->running };
    memcpy(r.ar, dsp->ar, sizeof r.ar);
    h[0] = 'D';
    fwrite(h, 1, 5, f);
    fwrite(&r, sizeof r, 1, f);
    fwrite(dsp->data, sizeof(uint16_t), C54X_DATA_SIZE, f);
    /* [2026-10-22 14:45] 'P' : la memoire PROGRAMME basse 0x0000-0x7FFF. La ROM y ecrit du code a l'execution
     * (prog_write : 0x2800-0x7FFF ne va que dans prog[]) ; sans lui le rejeu appelait 0x2EC5 vide a la 1re
     * tache TCH (« trou-prog ... depuis pc=0xbc43 »), pile perdue, DSP mort : aucun TCH rejouable. */
    h[0] = 'P';
    fwrite(h, 1, 5, f);
    fwrite(dsp->prog, sizeof(uint16_t), 0x8000, f);
}
static void enreg_tick(const C54xState *dsp, const uint16_t *api_ram, uint32_t tick, bool irq, bool deux, long
 ↪  budget)
{
    FILE *f = calypso_bsp_enreg_fichier();
    if (!f) return;
    static bool base;
    if (!base) { base = true; enreg_base(f, dsp, g_enreg_api_ok ? g_enreg_api : api_ram, tick); }
    uint8_t h[10] = { 'T', tick >> 24, tick >> 16, tick >> 8, tick, (uint8_t)((irq ? 1 : 0) | (deux ? 2 : 0)),
                      (uint8_t)(budget >> 24), (uint8_t)(budget >> 16), (uint8_t)(budget >> 8), (uint8_t)budget };
    fwrite(h, 1, sizeof h, f);
}

/* [2026-09-23] LE DSP ET L'ARM EN PARALLELE (PONT_DONE_TOT, 1 par defaut).
 * En pas-a-pas, une trame coutait la SOMME de QEMU (l'ARM) et du C54x : sur un
 * TCH le DSP monte a ~4 ms par trame (demodulation, Viterbi, parole), QEMU en
 * prend ~2, et le banc tombait a 174 trames/s au lieu de 216,7 -- la parole
 * arrivait a 40 trames/s pour un ALSA a 50 : echo test qui « part en live ».
 * Sur silicium l'ARM et le DSP tournent en meme temps et l'ARM ne relit les
 * resultats qu'a la trame suivante. On renvoie donc PONT_DONE des le burst
 * depose, et on finit la trame (reste du budget, pompe DMA, montant, sondes)
 * pendant que QEMU avance ; le TICK suivant attend dans la socket. Une trame
 * coute alors le plus long des deux, pas leur somme. Seul effet de bord :
 * PONT_DONE_API_IRQ n'est plus connu au moment du DONE (jamais leve sur ce
```

```
 * banc : irq=0 dans tous les bilans). PONT_DONE_TOT=0 retablit l'ancien ordre. */
static int g_done_tot = -1;
static int g_reste_b;

/* [2026-09-23] CHRONO PAR TRAME : ou passe une trame du banc, en ms moyennes,
 * imprime toutes les 1000 trames (« [chrono] »). qemu = attente du TICK apres
 * le DONE (l'ARM), A = phase A DSP, go = attente du GO (l1_sync de l'ARM),
 * B = phase B jusqu'au DONE, apres = fin de trame DSP apres le DONE. */
#include <time.h>
static double chrono_ms(void) { struct timespec t; clock_gettime(CLOCK_MONOTONIC, &t); return t.tv_sec * 1e3 +
 ↪  t.tv_nsec / 1e6; }
/* [2026-10-15] + PIRE TRAME : la moyenne ne peut pas depasser 4.62 (la trame est
 * cadencee par le TICK de QEMU, l'attente comble le reste) ; seuls le max et le
 * nombre de trames hors budget disent si la marge tient. dt = fin de trame a fin de
 * trame ; trop = dt > 5.54 ms (+20 % : la trame est cadencee sur 4.615, la moitie des
 * trames la depassent par simple gigue) ; dsp_trop = TICK -> fin de trame > 4.615 ms
 * (le DSP seul ne tient pas une trame). */
static struct { double t_fin, qemu, a, go, b, apres, dt_max, travail_max; unsigned n, trop, dsp_trop; } g_chrono;
static void chrono_trame(double t_tick, double t_done_a, double t_go, double t_done, double t_fin, uint32_t fn)
{
    if (g_chrono.t_fin > 0) {
        double dt = t_fin - g_chrono.t_fin;
        g_chrono.qemu += t_tick - g_chrono.t_fin;
        if (dt > g_chrono.dt_max) g_chrono.dt_max = dt;
        if (dt > 5.538) g_chrono.trop++;
    }
    if (t_fin - t_tick > g_chrono.travail_max) g_chrono.travail_max = t_fin - t_tick;
    if (t_fin - t_tick > 4.615) g_chrono.dsp_trop++;
    g_chrono.a += t_done_a - t_tick; g_chrono.go += t_go - t_done_a;
    g_chrono.b += t_done - t_go; g_chrono.apres += t_fin - t_done;
    g_chrono.t_fin = t_fin;
    if (++g_chrono.n == 1000) {
        double k = 1.0 / g_chrono.n;
        printf("  [chrono] fn=%u sur 1000 trames (ms) : qemu %.2f | A %.2f | go %.2f | B %.2f | apres DONE %.2f "
               "| trame %.2f (temps reel 4.62) | pire trame %.2f, DSP pire %.2f, trames >5.54 ms %u/1000, DSP >4.62
                ↪  ms %u/1000\n",
               fn, g_chrono.qemu * k, g_chrono.a * k, g_chrono.go * k,
               g_chrono.b * k, g_chrono.apres * k,
               (g_chrono.qemu + g_chrono.a + g_chrono.go + g_chrono.b + g_chrono.apres) * k,
               g_chrono.dt_max, g_chrono.travail_max, g_chrono.trop, g_chrono.dsp_trop);
        fflush(stdout);
        double tf = g_chrono.t_fin; memset(&g_chrono, 0, sizeof g_chrono); g_chrono.t_fin = tf;
    }
}

static uint32_t jouer_trame(C54xState *dsp, long budget, bool *init_done, uint32_t *insns, int phase)
{
    uint32_t drapeaux = 0;
    uint32_t avant = dsp->insn_count;
    static int fait_a;          /* phase A instructions, charged to phase B's budget */
    static bool etait_idle_a;   /* idle state at the frame start, for PONT_DONE_API_IRQ */

    if (phase == 2) goto phase_b;
    calypso_dma_tick(dsp);

    if (dsp->running && !*init_done) {
        if (!dsp->idle) {
            c54x_run(dsp, (int)budget);
        }
        if (dsp->idle) {
            *init_done = true;
            drapeaux |= PONT_DONE_INIT;
        }
    }
    if (dsp->running) {
        etait_idle_a = dsp->idle;
```

```
/* Wake on a level-held or pending interrupt. The core only vectors a
 * pending interrupt (IFR&IMR, INTM=0) on the next instruction
 * (c54x_irq_level_check, CALYPSO_C54X_IRQ_LEVEL=1), and an idle DSP
 * executes none; on silicon the interrupt itself wakes it (SPRU131).
 * INT10n (RHEA DMA completion, bit 14) is a level line: it stays
 * asserted while a channel holds IRQ_STATE, so present it on wake. */
if (dsp->idle && calypso_rhea_dma_irq_level() && (dsp->imr & (1u << 14)) &&
    !(dsp->ifr & (1u << 14)))
    c54x_interrupt_ex(dsp, 30, 14);
if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800))
    dsp->idle = false;
/* [2026-09-20] The DSP frame interrupt is TPU_CTRL_DSP_EN, a bit the
 * firmware sets in dsp_end_scenario() on EVERY scenario and that the
 * TPU consumes: the ROM gets a frame interrupt only on frames where
 * the ARM handed it a page. Raising it on every tick made the ROM
 * re-read the same page for 10+ frames (the firmware only flips the
 * write page on frames with a DSP item and the ROM never clears
 * d_task_md): the FB task restarted at each FCCH and the SB job never
 * ran (0xaba4 dispatched every frame, no 764-byte window armed).
 * QEMU now says in TICK.b bit 16 whether the ARM armed it.
 * PONT_IRQ_TRAME=1 restores the interrupt on every tick (A/B). */
static int irq_chaque = -1;
if (irq_chaque < 0) { const char *e = calypso_getenv("PONT_IRQ_TRAME"); irq_chaque = (e && *e == '1') ? 1
↪  : 0; }
if ((dsp->imr & (1u << C54X_IT_TPU_FRAME_BIT)) && (irq_chaque || g_tick_irq_trame)) {
    tx_tsp_dater(dsp, g_c54x_exe_fn);     /* le fn GSM de la page que la ROM va lire */
    c54x_interrupt_ex(dsp, C54X_IT_TPU_FRAME_VEC, C54X_IT_TPU_FRAME_BIT);
}
if (calypso_getenv("PONT_IRQ_DEBUG") && g_c54x_exe_fn > 5000 && g_c54x_exe_fn < 5012)
    printf("  [irq] fn=%u APRES vec28 : idle=%d pc=%04x INTM=%d IMR=%04x IFR=%04x\n",
           g_c54x_exe_fn, dsp->idle, dsp->pc & 0xffff, !!(dsp->st1 & 0x800), dsp->imr, dsp->ifr);
if (g_inj.actif || phase == 1) {
    int fait = 0;
    if (!dsp->idle) fait = c54x_run_profile(dsp, (int)budget / 8);   /* the ISR arms DMA2 */
    /* [2026-09-20] Deliver only once the receive window is armed. On a
     * frame where the ROM first finishes the previous burst's demod
     * (20-30k instructions of Viterbi) before it programs the one-shot
     * NB window, the fixed budget/8 slice delivered the frame BEFORE the
     * arming, the RIF discarded it (no window) and the result page stayed
     * empty: measured on the BTS bench, one normal-burst result in four
     * missing (b0 absent in 23 of 97 BCCH blocks), read by the firmware
     * as EMPTY / BURST ID n!=m. Keep running, in slices, until DMA2 is
     * armed or the DSP idles, half the budget at most. */
    /* [2026-09-23] SUR LE TCH, ALLER JUSQU'A L'IDLE AVANT LE DEPOT.
     * Pour une tache TCHA (SACCH/TF), la ROM arme la fenetre de la trame
     * N+1 PUIS demodule, au debut de N+1, le burst SACCH de N qu'elle
     * a laisse dans 0x0cce ; sur silicium le burst de N+1 n'arrive qu'au
     * passage de TS2, plus tard. Deposer des l'armement l'ecrasait : la
     * SACCH se decodait sur un burst de trafic (a_cd FIRE KO a chaque
     * bloc, 113-118 bits faux, LOS au 32e bloc). Reproducteur
     * tch_rejeu (scratchpad 2026-09-23) : depot apres l'IDLE -> SACCH
     * FIRE=0 « 07 00 03 », FACCH 10/10 bonnes (contre 6/12). Hors TCH
     * (FB, SB, SDCCH) on garde l'arret a l'armement.
     * PONT_TCH_DEPOT_IDLE=0 retablit l'ancien comportement. */
    int tch_idle = calypso_reglage("PONT_TCH_DEPOT_IDLE", 1);   /* [2026-10-22] a chaud */
    bool jusqua_idle = tch_idle && montant_sur_tch();
    while (!dsp->idle && (jusqua_idle || !calypso_rhea_dma_rx_armed()) && fait < (int)budget / 2)
        fait += c54x_run_profile(dsp, 256);
    if (jusqua_idle) {
        static unsigned n_tch;
        /* [2026-10-21 23:40] + irq_trame/IMR/IFR/INTM/pc : Magnetite sur TCH TS1, 3 trames puis
         * « 0 insn, idle=1 » et L1 TI « > COM » (d_debug=0) : IT trame non levee (QEMU) ou
         * bit TPU frame absent de l'IMR (ROM) ? 200 lignes pour couvrir plusieurs appels. */
        if (n_tch++ < 200)
            printf("  [depot_tch] fn=%u phase A : %d insn, idle=%d, fenetre armee=%d irq_trame=%d "
                   "IMR=%04x IFR=%04x INTM=%d pc=%04x%s\n",
```

```
                    g_c54x_exe_fn, fait, dsp->idle, calypso_rhea_dma_rx_armed(), g_tick_irq_trame,
                    dsp->imr, dsp->ifr, !!(dsp->st1 & 0x800), dsp->pc & 0xffff,
                    dsp->idle ? "" : "  <- BUDGET/2 ATTEINT, depot avant la fin");
            }
            fait_a = fait;
            if (phase == 1) {
                /* ISR played, R page written, window armed: the ARM may run
                 * l1_sync now. The burst comes with PONT_GO. */
                if (dsp->idle)    drapeaux |= PONT_DONE_IDLE;
                if (dsp->running) drapeaux |= PONT_DONE_RUNNING;
                if (*init_done)   drapeaux |= PONT_DONE_INIT;
                *insns = dsp->insn_count - avant;
                return drapeaux | PONT_DONE_PHASE_A;
            }
phase_b:
            fait = fait_a;
            if (dsp->api_ram) montant_capter_fu(dsp->api_ram, g_c54x_exe_fn);   /* avant que la ROM lise a_fu */
            /* PONT_NB_DEBUG: flags at the burst's entry, and a snapshot of the
             * data memory to list what the demod writes (regions), see
             * sonde_bits(). */
            { static int on = -1; if (on < 0) on = calypso_getenv("PONT_NB_DEBUG") ? 1 : 0;
              if (on) { g_flags_entree = ((s_or_dsp_st0(dsp) & ST0_OVA) ? 1 : 0) | ((dsp->st0 & ST0_OVB) ? 2 : 0) |
                ↪  ((dsp->st0 & ST0_C) ? 4 : 0) | ((dsp->st0 & ST0_TC) ? 8 : 0) | ((dsp->st1 & ST1_OVM) ? 16 : 0) |
                ↪  ((dsp->st1 & ST1_FRCT) ? 32 : 0) | ((dsp->st1 & ST1_SXM) ? 64 : 0);
                       memcpy(g_data_avant_b, dsp->data, sizeof g_data_avant_b); } }
            if (g_inj.udp) calypso_bsp_service(g_inj.fn);
            if (g_inj.iq_mode) injecter_burst(dsp, g_inj.iq_mode, g_inj.amp, g_inj.fn, g_inj.injectes);
            { uint16_t aad = calypso_rhea_dma_get_daram();
              memcpy(g_daram_apres_dma, &dsp->data[aad], sizeof g_daram_apres_dma); g_daram_aad = aad; }
            if (dsp->idle && calypso_rhea_dma_irq_level() && (dsp->imr & (1u << 14)) && !(dsp->ifr & (1u << 14)))
                c54x_interrupt_ex(dsp, 30, 14);
            if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
            /* PONT_NB_HIST=<dir>: opcode histogram of phase B (the burst's demod)
             * for the first 12 normal bursts, one file per frame, to name the
             * instructions the NB path leans on and cross them with the ISA
             * scorecard (isa_test). Single-stepped: prog_fetch() before each. */
            static const char *hist_dir = NULL; static int hist_init = 0; static unsigned hist_n;
            if (!hist_init) { hist_init = 1; hist_dir = calypso_getenv("PONT_NB_HIST"); }
            /* PONT_NB_HIST_SB=1 : trace the SCH bursts (SB task, d_task_md = 6) instead */
            static int hist_sb = -1; if (hist_sb < 0) hist_sb = calypso_getenv("PONT_NB_HIST_SB") ? 1 : 0;
            bool tache_nb = dsp->api_ram && (dsp->api_ram[API_R_PAGE(0) / 2] == 24 || dsp->api_ram[API_R_PAGE(1) /
             ↪  2] == 24);
            bool tache_sb = dsp->api_ram && ((dsp->api_ram[API_W_PAGE(0) / 2 + 4] & 0xff) == 6 ||
             ↪  (dsp->api_ram[API_W_PAGE(1) / 2 + 4] & 0xff) == 6);
            if (hist_dir && hist_n < 12 && !dsp->idle && dsp->api_ram &&
                ((!hist_sb && g_inj.dernier_type == 'B' && tache_nb) || (hist_sb && g_inj.dernier_type == 'S' &&
                 ↪  tache_sb))) {
                static unsigned hist[65536]; memset(hist, 0, sizeof hist);
                int reste = (int)budget - fait, k = 0;
                /* data watch: every write into the demod's working cells, with
                 * the PC of the instruction (taps 0x2cbb.., tracker 0x5aaa..,
                 * result cells 0x3fa4.., reference 0x2b28..) */
                static const struct { uint16_t lo, hi; } W[] = { {0x2cbb, 0x2d04}, {0x5aaa, 0x5ac8}, {0x3fa4,
                 ↪  0x3fa8}, {0x2b28, 0x2b58}, {0x2f00, 0x2f2c}, {0x2a00, 0x2bc8}, {0x2be0, 0x2c80} };
                #define NW 7
                static uint16_t prev[0x600]; int nw = 0;
                for (unsigned r = 0; r < NW; r++) for (unsigned a = W[r].lo; a < W[r].hi; a++) prev[nw++] =
                 ↪  dsp->data[a];
                char nomw[256]; snprintf(nomw, sizeof nomw, "%s/watch_%u.txt", hist_dir, g_inj.fn);
                FILE *fw = fopen(nomw, "w");
                char nomt[256]; snprintf(nomt, sizeof nomt, "%s/trace_%u.txt", hist_dir, g_inj.fn);
                FILE *ft = fopen(nomt, "w");
                while (!dsp->idle && k < reste) {
                    uint16_t w = prog_fetch(dsp, (uint16_t)dsp->pc);
                    uint16_t pc0 = (uint16_t)dsp->pc; uint16_t ar2 = dsp->ar[2], ar3 = dsp->ar[3];
                    hist[w]++;
```

```
                if (k == 12800) { memcpy(g_snap_2be2, &dsp->data[0x2be2], sizeof g_snap_2be2); g_snap_ok = 1; }
                 if (pc0 == 0x9a78 && !g_snap_2a00_ok) { memcpy(g_snap_2a00, &dsp->data[0x2a00], sizeof
                  ↪  g_snap_2a00); g_snap_2a00_ok = 1;
                    char nm[256]; snprintf(nm, sizeof nm, "%s/mem_%u_9a78.bin", hist_dir, g_inj.fn); FILE *fm =
                      ↪  fopen(nm, "wb"); if (fm) { fwrite(dsp->data, 2, C54X_DATA_SIZE, fm); fclose(fm); } }
                 if (ft) fprintf(ft, "%04x %04x %010llx %010llx %04x %04x %04x %04x %04x %04x %04x %04x %04x
                  ↪  %04x %04x %04x %04x %04x\n", pc0, w,
                                 (unsigned long long)(dsp->a & 0xFFFFFFFFFFULL), (unsigned long long)(dsp->b &
                                  ↪  0xFFFFFFFFFFULL),
                                 dsp->t, dsp->st0, dsp->st1, dsp->ar[1], ar2, ar3, dsp->ar[4], dsp->ar[5],
↪  dsp->ar[0],
                                 dsp->data[ar2], dsp->data[ar3], dsp->data[dsp->ar[4]], dsp->data[dsp->ar[5]],
↪  dsp->ar[6]);
                 c54x_run(dsp, 1); k++;
                 if (fw) { int i = 0;
                     for (unsigned r = 0; r < NW; r++) for (unsigned a = W[r].lo; a < W[r].hi; a++, i++)
                         if (dsp->data[a] != prev[i]) { fprintf(fw, "%d pc=%04x op=%04x %04x: %04x -> %04x (%d)
                          ↪  AR2=%04x AR3=%04x\n", k, pc0, w, a, prev[i], dsp->data[a], (int16_t)dsp->data[a],
                          ↪  ar2, ar3); prev[i] = dsp->data[a]; } }
             }
             if (fw) fclose(fw);
             if (ft) fclose(ft);
             { char nm[256]; snprintf(nm, sizeof nm, "%s/mem_%u_fin.bin", hist_dir, g_inj.fn); FILE *fm =
              ↪  fopen(nm, "wb"); if (fm) { fwrite(dsp->data, 2, C54X_DATA_SIZE, fm); fclose(fm); } }
             char nom[256]; snprintf(nom, sizeof nom, "%s/hist_%u.txt", hist_dir, g_inj.fn);
             FILE *f = fopen(nom, "w");
             if (f) { for (unsigned w = 0; w < 65536; w++) if (hist[w]) fprintf(f, "%04x %u\n", w, hist[w]);
              ↪  fclose(f); }
             hist_n++;
         }
         /* [2026-09-23] PONT_DONE_TOT : le reste de la trame (phase B) est joue
          * APRES le PONT_DONE, pendant que QEMU fait tourner l'ARM -- voir
          * servir(). */
         if (g_done_tot && phase == 2) g_reste_b = (int)budget - fait;
         else if (!dsp->idle) c54x_run_profile(dsp, (int)budget - fait);
     } else if (!dsp->idle) {
         c54x_run_profile(dsp, (int)budget);
     }
     if (!etait_idle_a && dsp->idle) {
         drapeaux |= PONT_DONE_API_IRQ;
     }
 }
 if (dsp->idle)    drapeaux |= PONT_DONE_IDLE;
 if (dsp->running) drapeaux |= PONT_DONE_RUNNING;
 if (*init_done)   drapeaux |= PONT_DONE_INIT;
 *insns = dsp->insn_count - avant;
 return drapeaux;
}

/* Synthetic I/Q injection.
 *
 * On silicon the DSP gets 148 complex int16 samples per burst, one per symbol,
 * written to DARAM 0x2a00 (296 words) by the BSP DMA and followed by the RX
 * interrupt; calypso_bsp_rx_burst() reproduces that sequence. An FCCH burst is
 * GMSK over 148 zero bits, i.e. a tone at +1625/24 kHz, which is +pi/2 of
 * phase per symbol at 270.833 ksym/s (GSM 45.004). */
#define IQ_N 148
static void iq_synthese(const char *mode, int amp, uint32_t fn, int16_t *iq)
{
    static unsigned seed = 12345;
    double dphi = 0;
    bool bruit = false;
    if (!strcmp(mode, "fcch")) {
        dphi = M_PI / 2;
    } else if (!strcmp(mode, "noise")) {
        bruit = true;
```

```
    } else if (!strncmp(mode, "tone:", 5)) {
        dphi = atof(mode + 5);
    }
    for (int k = 0; k < IQ_N; k++) {
        if (bruit) {
            seed = seed * 1103515245u + 12345u; int a = (int)((seed >> 16) % (2 * amp + 1)) - amp;
            seed = seed * 1103515245u + 12345u; int b = (int)((seed >> 16) % (2 * amp + 1)) - amp;
            iq[2 * k] = (int16_t)a; iq[2 * k + 1] = (int16_t)b;
        } else {
            double ph = dphi * k;
            iq[2 * k]     = (int16_t)lrint(amp * cos(ph));
            iq[2 * k + 1] = (int16_t)lrint(amp * sin(ph));
        }
    }
    (void)fn;
}

/* Recorded-cell replay, paced by the DSP frame clock.
 *
 * /opt/GSM/cellule_reelle.bin holds the TS0 of 311 CONSECUTIVE frames captured
 * off the air at 4 samples/symbol, each tagged with its real frame number, so
 * the 51-multiframe is intact: FCCH at fn%51 in {0,10,20,30,40}, SCH at
 * {1,11,21,31,41}.
 *
 * Why it is fed from here and not over UDP: the source and the DSP have
 * independent clocks. Measured on this bench, the DSP runs between 72 and 160
 * frames/s depending on load while rejouer_cellule.py streams at a rate of its
 * own, so the DSP swallowed about four cell frames per frame of its own. The
 * FB task survives that — it only ever correlates ONE window — but the SB task
 * needs the burst of the frame IMMEDIATELY AFTER the FCCH, and that frame was
 * never the SCH. Pulled from here, one burst per frame, the multiframe reaches
 * the DSP intact whatever the emulated clock does. */
typedef struct { uint32_t fn; int16_t *iq; int fcch; } ReelBurst;
static ReelBurst *g_reel;
static unsigned   g_reel_n, g_reel_util;
static int        g_reel_nsym;
static unsigned   g_reel_base;

/* Decimate a stored burst to the 1 sample/symbol the correlator works at. The
 * same decimation the UDP path applies through CALYPSO_BSP_IQ_DECIM, and the
 * same one reelle_injecter delivers: what is measured here is what the DSP
 * gets. Returns the number of int16 written. */
static int reelle_decimer(const ReelBurst *b, int16_t *iq, int max_i16)
{
    int decim = g_reel_nsym / 148;
    if (decim < 1) decim = 1;
    int n = 0;
    for (int k = 0; k * decim < g_reel_nsym && n <= max_i16 - 2; k++) {
        iq[n++] = b->iq[2 * (k * decim)];
        iq[n++] = b->iq[2 * (k * decim) + 1];
    }
    return n;
}

/* Is this burst an FCCH? At 1 sample/symbol an all-zeros GMSK burst is a pure
 * tone rotating by exactly +pi/2 per sample — the criterion calypso_bsp.c's
 * FCCH-PROBE already uses (coh ~ 1, dphi ~ +1.571). Nothing else on TS0 comes
 * close, so this labels the multiframe without decoding anything. */
static int reelle_est_fcch(const ReelBurst *b)
{
    int16_t iq[2 * 256];
    int n = reelle_decimer(b, iq, (int)(sizeof(iq) / sizeof(iq[0])));
    int ns = n / 2;
    if (ns < 32) return 0;
    double accr = 0, acci = 0, den = 0;
    for (int k = 1; k < ns; k++) {
```

```
        double i0 = iq[2*(k-1)], q0 = iq[2*(k-1)+1];
        double i1 = iq[2*k],     q1 = iq[2*k+1];
        accr += i1*i0 + q1*q0;
        acci += q1*i0 - i1*q0;
        den  += sqrt((i0*i0 + q0*q0) * (i1*i1 + q1*q1));
    }
    if (den <= 0) return 0;
    double coh  = sqrt(accr*accr + acci*acci) / den;
    double dphi = atan2(acci, accr);
    return coh > 0.90 && fabs(dphi - M_PI_2) < 0.30;
}

/* Where does the recording sit in the 51-multiframe?
 *
 * The frame numbers stored in the file cannot be trusted for this: measured on
 * cellule_reelle.bin, the FCCH bursts carry tags whose residues are
 * {0,10,20,31,41} where GSM 05.02 puts the FCCH at fn%51 in {0,10,20,30,40} —
 * a constant offset of 31. So the phase is taken from the SIGNAL instead: the
 * FCCH bursts are located by reelle_est_fcch(), and the one followed by a gap
 * of 11 frames is the last of its multiframe, i.e. true phase 40 (the sequence
 * of gaps is 10,10,10,10,11). g_reel_base then satisfies
 *
 *     entry_index  ==  (arm_fn - g_reel_base)  (mod 51)
 *
 * which is what reelle_injecter uses to pick a burst BY FRAME NUMBER. */
static void reelle_caler(void)
{
    unsigned fcch[64], nf = 0;
    for (unsigned i = 0; i < g_reel_n && nf < 64; i++)
        if (g_reel[i].fcch) fcch[nf++] = i;

    unsigned i40 = 0; int trouve = 0;
    for (unsigned k = 0; k + 1 < nf; k++)
        if (fcch[k + 1] - fcch[k] == 11) { i40 = fcch[k]; trouve = 1; break; }

    if (!trouve) {
        g_reel_base = 0;
        printf("pont : ATTENTION — phase de multitrame indeterminee (%u FCCH reperees, "
               "aucun ecart de 11) ; calage a 0, la lecture reste cadencee par fn\n", nf);
        return;
    }
    g_reel_base = (40u + 51u - (i40 % 51u)) % 51u;   /* i40 == 40 - base  (mod 51) */
    unsigned ecart_tag = (g_reel[0].fn + 51u - g_reel_base) % 51u;
    printf("pont : %u FCCH reperees dans la capture (ecarts 10/11), phase calee sur le signal : "
           "base=%u ; les tags fn du fichier sont decales de +%u mod 51\n",
           nf, g_reel_base, ecart_tag);
}

/* Recorded-cell replay, addressed by the ARM frame number.
 *
 * /opt/GSM/cellule_reelle.bin holds the TS0 of 311 CONSECUTIVE frames captured
 * off the air at 4 samples/symbol, each tagged with its real frame number.
 *
 * Why it is fed from here and not over UDP: the source and the DSP have
 * independent clocks. Measured on this bench, the DSP runs between 72 and 160
 * frames/s depending on load while rejouer_cellule.py streams at a rate of its
 * own, so the DSP swallowed about four cell frames per frame of its own. The
 * FB task survives that — it only ever correlates ONE window — but the SB task
 * needs the burst of the frame IMMEDIATELY AFTER the FCCH, and that frame was
 * never the SCH.
 *
 * [2026-09-19] Why it is addressed by fn and no longer by a running index:
 * calypso_trx.c:pont_echange() SKIPS a tick whenever the DSP has not returned
 * its DONE — about 3% of frames once the traces are off, 85% with -vvv on a
 * terminal. A running index does not advance on a skipped tick while the ARM
 * clock does, so every skip shifted the recording against the ARM by one frame,
```

```
 * FOR GOOD: measured 4202 frames of accumulated drift over a 5-minute run, i.e.
 * a multiframe phase wandering without bound. The ARM then armed its SB one
 * frame after an FCCH and got a burst from somewhere else entirely (11 SB CRC
 * OK in 68000 frames), and the deinterleaver assembled BCCH blocks out of
 * unrelated bursts (205-221 bit errors per block, every block dropped).
 * Indexing by fn makes a skipped tick skip a RECORDING burst too: the phase is
 * held whatever the emulated clock does. */
static void reelle_injecter(C54xState *dsp, uint32_t fn, unsigned long *injectes)
{
    if (!g_reel_util) return;
    unsigned idx = (fn + g_reel_util - (g_reel_base % g_reel_util)) % g_reel_util;
    const ReelBurst *b = &g_reel[idx];

    int16_t iq[2 * 256];
    int n_iq = reelle_decimer(b, iq, (int)(sizeof(iq) / sizeof(iq[0])));

    /* The burst keeps its OWN frame number: it is the provenance of the samples,
     * and calypso_bsp_rx_burst only ever logs it. */
    calypso_bsp_rx_burst(0, b->fn, iq, n_iq);
    (*injectes)++;
}

static int reelle_charger(const char *chemin)
{
    FILE *f = fopen(chemin, "rb");
    if (!f) { fprintf(stderr, "pont : cellule reelle introuvable : %s\n", chemin); return 0; }
    uint32_t hdr[4];
    if (fread(hdr, sizeof(hdr), 1, f) != 1) { fclose(f); return 0; }
    unsigned n = hdr[0], nsym = hdr[1], bsic = hdr[2];
    if (!n || !nsym || nsym > 4096) { fclose(f); return 0; }
    g_reel = calloc(n, sizeof(*g_reel));
    if (!g_reel) { fclose(f); return 0; }
    for (unsigned i = 0; i < n; i++) {
        uint32_t e[2];
        if (fread(e, sizeof(e), 1, f) != 1) { n = i; break; }
        int16_t *iq = malloc((size_t)nsym * 2 * sizeof(int16_t));
        if (!iq || fread(iq, sizeof(int16_t) * 2, nsym, f) != nsym) { free(iq); n = i; break; }
        g_reel[i].fn = e[0];
        g_reel[i].iq = iq;
    }
    fclose(f);
    g_reel_n = n; g_reel_nsym = (int)nsym;

    /* Only a WHOLE number of 51-multiframes may be looped: 311 = 6*51 + 5, so
     * wrapping on all 311 shifted the phase by 5 at every turn. */
    g_reel_util = (n / 51u) * 51u;

    for (unsigned i = 0; i < g_reel_n; i++)
        g_reel[i].fcch = reelle_est_fcch(&g_reel[i]);

    printf("pont : cellule REELLE %s — %u trames consecutives a %d ech/symbole, BSIC=%u, "
           "fn %u..%u, %u trames jouees (%u multitrames de 51, %u ecartees), adressage par fn\n",
           chemin, n, nsym / 148, bsic, n ? g_reel[0].fn : 0, n ? g_reel[n - 1].fn : 0,
           g_reel_util, g_reel_util / 51u, n - g_reel_util);
    reelle_caler();
    return (int)g_reel_util;
}

/* Inject one burst (synthetic cell or plain signal) into the RIF/BSP path. */
static int g_verif_sonde = -1;

static void injecter_burst(C54xState *dsp, const char *iq_mode, int amp, uint32_t fn,
                           unsigned long *injectes)
{
        int16_t iq[2 * 256];
        int n_iq = 2 * IQ_N;
```

```
        char t_dbg = '?';
        if (!strncmp(iq_mode, "reelle", 6)) {
            static int charge = 0;
            if (!charge) { const char *p = strchr(iq_mode, ':');
                charge = reelle_charger(p ? p + 1 : "/opt/GSM/cellule_reelle.bin"); if (!charge) charge = -1; }
            if (charge > 0) reelle_injecter(dsp, fn, injectes);
            return;
        }
        if (!strncmp(iq_mode, "cell", 4)) {
            /* cell[:bsic[:offset[:margin]]] : full cell, FCCH/SCH/dummy bursts */
            static int bsic = -1; static double dec = 0.5; static int marge = 21;
            if (bsic < 0) { bsic = 42; const char *p = strchr(iq_mode, ':');
                if (p) { bsic = atoi(p + 1) & 0x3f; p = strchr(p + 1, ':');
                    if (p) { dec = atof(p + 1); p = strchr(p + 1, ':'); if (p) marge = atoi(p + 1); } }
                if (marge > 50) marge = 50;
                if (strstr(iq_mode, "all")) cellule_sch_partout = 1;
                printf("pont : cellule BSIC=%d (NCC=%d BCC=%d), echantillonnage a %.2f symbole, "
                       "SCH dans une fenetre de %d echantillons (DARAM len=%u mots)%s\n",
                       bsic, bsic >> 3, bsic & 7, dec, 148 + 2 * marge, calypso_bsp_get_daram_len(),
                       cellule_sch_partout ? ", SCH sur toutes les trames non-FCCH" : ""); }
            /* the SCH burst is delivered as the 190-sample window block only when
             * the DSP has its one-shot SB window armed; otherwise it is a frame of
             * the FB stream like any other (same rule as rejouer.c) */
            int marge_eff = calypso_rhea_dma_one_shot() ? marge : 0;
            /* [2026-09-20] BCCH/CCCH normal bursts: framed like the BSP stream
             * assembler (bsp_livrer_trame), 3 silent samples ahead when the
             * one-shot window is the 151-sample NB one (tpu_window.c
             * L1_NB_MARGIN_Q), bare 148 samples in the continuous FB stream. */
            { int nwin = calypso_rhea_dma_one_shot() ? calypso_rhea_dma_get_len_words() / 2 : 0;
              /* PONT_NB_MARGE=<n> overrides the 3-sample head margin of a normal
               * burst; "auto" sweeps 0..7 by 51-multiframe (the [scan] probe
               * prints it), one run to find where the ROM's TSC search lands. */
              static int mnb = -2; static int mauto = 0;
              if (mnb == -2) { const char *e = calypso_getenv("PONT_NB_MARGE"); mnb = 3;
                               if (e && !strcmp(e, "auto")) mauto = 1; else if (e && *e) mnb = atoi(e); }
              int m_nb = mauto ? (int)((fn / 51u) % 8u) : mnb;
              /* CELLULE_TSC=<k>|auto : training sequence written in the bursts
               * (the ARM still tells the ROM tsc = BCC); auto sweeps 0..7 */
              static int tsc = -2; static int tauto = 0;
              if (tsc == -2) { const char *e = calypso_getenv("CELLULE_TSC"); tsc = -1;
                               if (e && !strcmp(e, "auto")) tauto = 1; else if (e && *e) tsc = atoi(e) & 7; }
              cellule_tsc_force = tauto ? (int)((fn / 51u) % 8u) : tsc;
              cellule_marge_nb = (nwin >= 150 && nwin < 190) ? m_nb : (nwin >= 190 ? marge : -1);
              cellule_fenetre_nb = nwin; }
            char t = cellule_burst(fn, (uint8_t)bsic, amp, dec, marge_eff, iq, &n_iq);
            t_dbg = t; g_inj.dernier_type = t; g_inj.dernier_n_iq = n_iq;
            static unsigned nS, nF, nB, nC, tot;
            if (t == 'S') nS++; else if (t == 'F') nF++; else if (t == 'B') nB++; else if (t == 'C') nC++;
            if (++tot % 5000 == 1) printf("pont : bursts injectes FCCH=%u SCH=%u BCCH=%u CCCH=%u (fn=%u)\n", nF,
             ↪  nS, nB, nC, fn);
        } else {
            iq_synthese(iq_mode, amp, fn, iq);
        }
        static int irqdbg = -1; static unsigned irqdbg_n;
        if (irqdbg < 0) irqdbg = calypso_getenv("PONT_IRQ_DEBUG") ? 1 : 0;
        bool dbg = irqdbg && fn > 5000 && irqdbg_n < 12;
        if (dbg) printf("  [irq] fn=%u AVANT rx_burst : idle=%d pc=%04x INTM=%d IMR=%04x IFR=%04x PMST=%04x
         ↪  SP=%04x\n",
                        fn, dsp->idle, dsp->pc & 0xffff, !!(dsp->st1 & 0x800), dsp->imr, dsp->ifr, dsp->pmst,
↪  dsp->sp);
        { static int dj = -1; if (dj < 0) dj = calypso_getenv("PONT_DEBUG_INJ") ? 1 : 0;
          int md0 = dsp->api_ram ? (dsp->api_ram[4] & 0xff) : 0, md1 = dsp->api_ram ? (dsp->api_ram[0x18] & 0xff)
           ↪  : 0;
          if (dj && (calypso_rhea_dma_one_shot() || md0 == 6 || md1 == 6 || (fn >= 300 && fn <= 312)))
              printf("  [inj] fn=%u p51=%u AVANT : type=%c n_iq=%d dma armee=%d one_shot=%d task_md=%d/%d rif=%d
↪  mots idle=%d pc=%04x"
```

```
                    " | W0=%04x %04x %04x %04x %04x ..%04x %04x  W1=%04x %04x %04x %04x %04x ..%04x %04x  NDB
                     ↪  page=%04x fb_mode=%04x fb_det=%04x\n",
                    fn, fn % 51u, t_dbg, n_iq, calypso_rhea_dma_rx_armed(), calypso_rhea_dma_one_shot(), md0,
↪  md1, calypso_rif_level(), dsp->idle, dsp->pc & 0xffff,
                    dsp->api_ram[0], dsp->api_ram[1], dsp->api_ram[2], dsp->api_ram[3], dsp->api_ram[4],
↪  dsp->api_ram[15], dsp->api_ram[16],
                    dsp->api_ram[0x14], dsp->api_ram[0x15], dsp->api_ram[0x16], dsp->api_ram[0x17],
↪  dsp->api_ram[0x18], dsp->api_ram[0x14+15], dsp->api_ram[0x14+16],
                    dsp->api_ram[0xd4], dsp->api_ram[0xd4+37], dsp->api_ram[0xd4+36]); }
        memcpy(g_dernier_iq, iq, (size_t)n_iq * sizeof(int16_t)); g_dernier_n_iq = n_iq;
        calypso_bsp_rx_burst(0, fn, iq, n_iq);
        { static int dj2 = -1; if (dj2 < 0) dj2 = calypso_getenv("PONT_DEBUG_INJ") ? 1 : 0;
          if (dj2 && fn >= 300 && fn <= 312)
              printf("  [inj] fn=%u APRES : rif=%d mots idle=%d IFR=%04x\n", fn, calypso_rif_level(), dsp->idle,
↪  dsp->ifr); }
        if (dbg) { printf("  [irq] fn=%u APRES rx_burst : idle=%d pc=%04x INTM=%d IMR=%04x IFR=%04x SP=%04x\n",
                          fn, dsp->idle, dsp->pc & 0xffff, !!(dsp->st1 & 0x800), dsp->imr, dsp->ifr, dsp->sp);
↪  irqdbg_n++; }
        (*injectes)++;
}


/* Register of active hacks. Every departure from native behaviour (canned TOA,
 * forced AFC rotation, direct feed, locks, ...) is listed by hacks_actifs() and
 * printed on every milestone line: a milestone reached with a non-empty list is
 * not a native milestone. The stimulus (synthetic cell) is reported separately,
 * being test input rather than a crutch. */

/* SB encoder: exact inverse of l1s_decode_sb (prim_fbsb.c), identical to
 * shunt_encode_sb in qemu-calypso. Maps {bsic, T1, T2, T3} to the 25-bit sb
 * word; the firmware reads sb = a_sch[3] | a_sch[4]<<16, then bsic = (sb>>2)
 * & 0x3f, and so on (GSM 45.002 SCH layout). */
static uint32_t pont_encode_sb(uint8_t bsic, uint16_t t1, uint8_t t2, uint8_t t3)
{
    uint8_t t3p = (t3 == 0) ? 0 : (uint8_t)((t3 - 1) / 10);
    uint32_t sb = 0;
    sb |= ((uint32_t)(bsic & 0x3f)) << 2;
    sb |= ((uint32_t)(t1 & 0x001)) << 23;
    sb |= ((uint32_t)(t1 & 0x1fe)) << 7;
    sb |= ((uint32_t)(t1 & 0x600)) >> 9;
    sb |= ((uint32_t)(t2 & 0x1f))  << 18;
    sb |= ((uint32_t)(t3p & 1))    << 24;
    sb |= ((uint32_t)(t3p & 6))    << 15;
    return sb;
}

/* [2026-09-21] CADENCEMENT, PAS BEQUILLE.
 *
 * STREAM et LOCKSTEP figuraient parmi les bequilles ; ils n'y ont pas leur
 * place. Aucun des deux ne falsifie une mesure : ils font cohabiter un DSP
 * emule, qui coute ~6,7 ms par trame, avec une BTS en temps reel dont la trame
 * dure 4,615 ms.
 *   - sans LOCKSTEP, QEMU avance a l'horloge murale et saute les trames que le
 *     DSP n'a pas finies : releve du 2026-09-21, « 2170 sauts, 3440 trames
 *     jouees », soit 39 % de perte ;
 *   - sans STREAM, chaque burst part vers le RIF des son arrivee : 217 bursts
 *     par seconde pour ~89 ticks, la ROM compte ~1526 symboles par trame au
 *     lieu de 1250, son TOA ne se stabilise jamais (2967, 3735, 4215, 48,
 *     13536 sur un meme run) et le pipeline se desaligne (« BURST ID 3!=2 »,
 *     « EMPTY »).
 * La vraie bequille etait ailleurs : le mode cadence ne livrait que TS0 et
 * bourrait les sept autres intervalles a zero. C'est corrige (calypso_bsp.c,
 * bsp_autres_stocker/bsp_autres_bits) : la trame remise au DSP porte
 * maintenant ce que la BTS a reellement emis sur les huit intervalles.
 *
 * Ils restent affiches, mais sur leur propre ligne. */
```

```
static const char *cadencement_actif(void)
{
    static char buf[128]; buf[0] = 0;
    const char *s1 = calypso_getenv("CALYPSO_BSP_STREAM");
    const char *s2 = calypso_getenv("CALYPSO_PONT_LOCKSTEP");
    if (s1 && *s1 == '1') strcat(buf, "STREAM ");
    if (s2 && *s2 == '1') strcat(buf, "LOCKSTEP ");
    return buf[0] ? buf : "aucun (le DSP ne suivra pas le temps reel)";
}

static const char *hacks_actifs(void)
{
    static char buf[512]; buf[0] = 0;
    struct { const char *env, *tag; int mode; } t[] = {   /* mode 0: set and non-empty; 1: =="1"; 2: =="0"; 3:
     ↪  integer != 0; 4: integer >= 0 */
        {"PONT_CAN_TOA","CAN_TOA",4}, {"PONT_CAN_SB_TOA","CAN_SB_TOA",4}, {"PONT_CAN_SB","CAN_SB_FULL",0},
        {"CALYPSO_TWL3025_AFC_HZ","AFC_HZ",3}, {"CALYPSO_TWL3025_AFC","AFC_OFF",2},
        {"CALYPSO_TWL3025_AFC_SIGN_OLD","AFC_SIGN_OLD",0},
        {"CALYPSO_BSP_VEC30","VEC30",1},
        {"CALYPSO_BSP_RX_LEAD","RX_LEAD",3}, {"CALYPSO_BSP_TPU_TRACK","TPU_TRACK",1},
        {"CALYPSO_BSP_TOA_LOCK","TOA_LOCK",1},
        {"CALYPSO_BSP_IQ_PASSTHROUGH","IQ_SYNTH",2},
        {"CALYPSO_RHEA_DMA_XFER","RHEA_DMA",1}, {"CALYPSO_BSP_RX_VEC","RX_VEC",0},
        {"CELLULE_SCH_ONLY","SCH_ONLY",1}, {"CELLULE_SCH_AMPDIV","SCH_AMPDIV",3},
        {"CALYPSO_FIXES","FIXES",0},
    };
    for (unsigned i = 0; i < sizeof t / sizeof t[0]; i++) {
        const char *v = calypso_getenv(t[i].env); if (!v || !*v) continue;
        int on = 0; long n = atol(v);
        switch (t[i].mode) { case 0: on = 1; break; case 1: on = (*v=='1'); break; case 2: on = (*v=='0'); break;
                             case 3: on = (n != 0); break; case 4: on = (n >= 0); break; }
        if (!on) continue;
        size_t l = strlen(buf);
        if (t[i].mode == 3 || t[i].mode == 4) snprintf(buf + l, sizeof buf - l, "%s%s=%ld", l ? "," : "",
         ↪  t[i].tag, n);
        else snprintf(buf + l, sizeof buf - l, "%s%s", l ? "," : "", t[i].tag);
    }
    /* a decoder fix turned off is a departure from native too */
    static const char *fx[] = {"NORM_SD","F7_DELAYED","MPY_MAC_LK","MACP_MACD","PAR_ST_DSTBAR","STL_STH_SHFT",
     ↪  "XCCD","ADDSUB_XSHFT","FIRS_RPT","RPT_COUNT"};
    for (unsigned i = 0; i < sizeof fx / sizeof fx[0]; i++) {
        char e[64]; snprintf(e, sizeof e, "CALYPSO_FIX_%s", fx[i]); const char *v = calypso_getenv(e);
       if (v && *v == '0') { size_t l = strlen(buf); snprintf(buf + l, sizeof buf - l, "%sFIX_%s=0", l ? "," : "",
         ↪  fx[i]); }
    }
    return buf[0] ? buf : "aucun";
}

/* PONT_NB_DEBUG=1: pages as seen at one instant of the frame (A = end of the
 * ROM's frame ISR, G = PONT_GO received i.e. after the ARM's l1_sync, B = end
 * of the frame). Who reads which W page and who writes which R page, when. */
static void sonde_pages(const char *quand, uint32_t fn, C54xState *dsp, const uint16_t *api_ram)
{
    static int on = -1; static unsigned n;
    if (on < 0) on = calypso_getenv("PONT_NB_DEBUG") ? 1 : 0;
    if (!on || n >= 240) return;
    const uint16_t *w0 = &api_ram[API_W_PAGE(0) / 2], *w1 = &api_ram[API_W_PAGE(1) / 2];
    const uint16_t *r0 = &api_ram[API_R_PAGE(0) / 2], *r1 = &api_ram[API_R_PAGE(1) / 2];
    if (!(w0[0] == 24 || w1[0] == 24 || r0[0] == 24 || r1[0] == 24)) return;
    n++;
    printf("  [pg%s] fn=%u p51=%u W0=%u/%u W1=%u/%u R0=%u/%u R1=%u/%u dsp_page=%04x idle=%d pc=%04x IFR=%04x
 ↪  IMR=%04x INTM=%d irq_trame=%d dma_armee=%d\n",
           quand, fn, fn % 51u, w0[0], w0[1], w1[0], w1[1], r0[0], r0[1], r1[0], r1[1],
           api_ram[(API_NDB + NDB_D_DSP_PAGE) / 2], dsp->idle, dsp->pc & 0xffff, dsp->ifr, dsp->imr,
           !!(dsp->st1 & 0x800), g_tick_irq_trame, calypso_rhea_dma_rx_armed());
}
```

```
/* PONT_NB_DEBUG=1, after a normal burst: look for the demodulated bits in the
 * DSP data memory. Two templates, the 116 data bits (57+hl, hu+57) and the
 * whole 148-bit burst, matched against the sign of each int16 word (soft bits,
 * both polarities) and against hard 0/1 words. Reports the best run. */
static uint32_t g_insn_a, g_insn_b;
static void sonde_bits(uint32_t fn, C54xState *dsp, uint8_t bsic)
{
    static int on = -1; static unsigned n;
    if (on < 0) on = calypso_getenv("PONT_NB_DEBUG") ? 1 : 0;
    if (!on || n >= 40) return;
    uint8_t bits[148];
    if (cellule_bits_attendus(fn, bsic, bits) < 0) return;
    uint8_t t116[116]; memcpy(t116, bits + 3, 58); memcpy(t116 + 58, bits + 87, 58);
    const struct { const uint8_t *t; int len; const char *nom; } tpl[2] = { { t116, 116, "116 data" }, { bits,
     ↪  148, "148 burst" } };
    n++;
    printf("  [scan] fn=%u p51=%u burst %d :", fn, fn % 51u, (int)((fn % 51u % 10 - 2) & 3));
    for (int k = 0; k < 2; k++) {
        int best[3] = {0,0,0}; unsigned bad[3] = {0,0,0};
        for (unsigned a = 0x60; a + tpl[k].len < C54X_DATA_SIZE; a++) {
            int m0 = 0, m1 = 0, m2 = 0;
            for (int i = 0; i < tpl[k].len; i++) {
                int16_t v = (int16_t)dsp->data[a + i];
                int neg = v < 0, one = (v == 1), zero = (v == 0);
                if ((v < 0) == (tpl[k].t[i] != 0)) m0++;
                if ((v > 0) == (tpl[k].t[i] != 0)) m1++;
                if ((one && tpl[k].t[i]) || (zero && !tpl[k].t[i])) m2++;
            }
            if (m⟨1⟩ > best[⟨2⟩]) { best[⟨3⟩] = m⟨4⟩; bad[⟨5⟩] = a; }  ×3
    ⟨⟩ = (0,0,0,0,0) (1,1,1,1,1) (2,2,2,2,2)
        }
        printf("  %s: neg=1 %d/%d @%04x  pos=1 %d/%d @%04x  hard %d/%d @%04x |", tpl[k].nom,
               best[0], tpl[k].len, bad[0], best[1], tpl[k].len, bad[1], best[2], tpl[k].len, bad[2]);
    }
    /* the ROM's burst buffer at 0x2be2 (found by this scan: 146/148 on good
     * bursts): mismatch map, one char per bit, '.' ok, 'x' wrong, '|' at the
     * data/TSC boundaries */
    { char map[160]; int k = 0, err = 0;
      for (int i = 0; i < 148; i++) {
          if (i == 3 || i == 61 || i == 87 || i == 145) map[k++] = '|';
          int16_t v = (int16_t)dsp->data[0x2be2 + i];
          int ok = ((v > 0) == (bits[i] != 0)); if (!ok) err++;
          map[k++] = ok ? '.' : 'x';
      }
      map[k] = 0;
      /* the same buffer against the two previous bursts: a demod deferred to
       * the next frame's ISR would leave burst N-1 here at the end of frame N */
      int e1 = -1, e2 = -1; uint8_t pb[148];
      if (cellule_bits_attendus(fn - 1, bsic, pb) == 0) { e1 = 0; for (int i = 0; i < 148; i++) if
       ↪  ((((int16_t)dsp->data[0x2be2 + i]) > 0) != (pb[i] != 0)) e1++; }
      if (cellule_bits_attendus(fn - 2, bsic, pb) == 0) { e2 = 0; for (int i = 0; i < 148; i++) if
       ↪  ((((int16_t)dsp->data[0x2be2 + i]) > 0) != (pb[i] != 0)) e2++; }
      /* regions of the data memory the demod wrote during phase B (beyond the
       * DARAM burst buffer and the API pages), with a few values each */
      { char z[900]; int k = 0; unsigned a = 0x60;
        while (a < C54X_DATA_SIZE && k < 800) {
            if (dsp->data[a] != g_data_avant_b[a]) {
                unsigned b = a; while (b < C54X_DATA_SIZE && b - a < 4096 && (dsp->data[b] != g_data_avant_b[b] ||
                 ↪  (b + 1 < C54X_DATA_SIZE && dsp->data[b+1] != g_data_avant_b[b+1]))) b++;
                k += snprintf(z + k, sizeof z - k, " %04x+%u", a, b - a); a = b;
            } else a++;
        }
        { const char *d = calypso_getenv("PONT_NB_HIST"); static unsigned nz;
          if (d && nz < 12) { char nom[256]; snprintf(nom, sizeof nom, "%s/zones_%u.txt", d, fn); FILE *f =
           ↪  fopen(nom, "w");
```

```
            if (f) { for (unsigned q = 0x60; q < C54X_DATA_SIZE; q++) if (dsp->data[q] != g_data_avant_b[q])
             ↪  fprintf(f, "%04x %04x %04x\n", q, g_data_avant_b[q], dsp->data[q]); fclose(f); nz++; } } }
        printf("  [zones] fn=%u flags(OVA=%d OVB=%d C=%d TC=%d OVM=%d FRCT=%d SXM=%d) ecrites:%s\n", fn,
               g_flags_entree & 1, !!(g_flags_entree & 2), !!(g_flags_entree & 4), !!(g_flags_entree & 8),
↪  !!(g_flags_entree & 16), !!(g_flags_entree & 32), !!(g_flags_entree & 64), z); }
      if (g_snap_ok) { int es = 0; for (int i = 0; i < 148; i++) if ((((int16_t)g_snap_2be2[i]) > 0) != (bits[i]
       ↪  != 0)) es++;
                       printf("  [scan] fn=%u 2be2 au pas 12800 (avant le decodeur) : erreurs=%d\n", fn, es);
↪  g_snap_ok = 0; }
      printf("  [scan] fn=%u marge=%d tsc=%d dec=%.2f phase=%.1f rif=%d 2be2 erreurs=%d (vs N-1: %d, N-2: %d)
↪  insnA=%u insnB=%u %s  v[0..3]=%d %d %d %d\n", fn, cellule_marge_nb, cellule_tsc_force, cellule_dec_nb,
↪  cellule_phase_nb, calypso_rif_level(), err, e1, e2, g_insn_a, g_insn_b, map,
             (int16_t)dsp->data[0x2be2], (int16_t)dsp->data[0x2be3], (int16_t)dsp->data[0x2be4],
              ↪  (int16_t)dsp->data[0x2be5]); }
    /* after burst 3: the decoder's vectors. Search the whole data memory for
     * the 456 coded bits in deinterleaved order and for the 184 information
     * bits, as signs of int16 words and as hard 0/1 words. */
    { uint8_t code[456], info[184];
      if (((fn % 51u) % 10u - 2) % 4 == 3 && cellule_bloc_attendu(fn, bsic, code, info) == 0) {
          const struct { const uint8_t *t; int len; const char *nom; } tp[2] = { { code, 456, "456 codes" }, {
           ↪  info, 184, "184 info" } };
          printf("  [bloc] fn=%u :", fn);
          for (int k = 0; k < 2; k++) {
              int best[3] = {0,0,0}; unsigned bad[3] = {0,0,0};
              for (unsigned a = 0x60; a + tp[k].len < C54X_DATA_SIZE; a++) {
                  int m0 = 0, m1 = 0, m2 = 0;
                  for (int i = 0; i < tp[k].len; i++) {
                      int16_t v = (int16_t)dsp->data[a + i];
                      if ((v < 0) == (tp[k].t[i] != 0)) m0++;
                      if ((v > 0) == (tp[k].t[i] != 0)) m1++;
                      if ((v == 1 && tp[k].t[i]) || (v == 0 && !tp[k].t[i])) m2++;
                  }
                  if (m⟨1⟩ > best[⟨2⟩]) { best[⟨3⟩] = m⟨4⟩; bad[⟨5⟩] = a; }  ×3
⟨⟩ = (0,0,0,0,0) (1,1,1,1,1) (2,2,2,2,2)
              }
              printf("  %s: neg=1 %d/%d @%04x  pos=1 %d/%d @%04x  hard %d/%d @%04x |", tp[k].nom,
                     best[0], tp[k].len, bad[0], best[1], tp[k].len, bad[1], best[2], tp[k].len, bad[2]);
          }
          /* also the coded bits packed 16 per word (MSB first) */
          { int bestp = 0; unsigned badp = 0;
            for (unsigned a = 0x60; a + 29 < C54X_DATA_SIZE; a++) {
                int m = 0;
                for (int i = 0; i < 456; i++) { int bit = (dsp->data[a + i / 16] >> (15 - i % 16)) & 1; if (bit ==
                 ↪  code[i]) m++; }
                if (m > bestp) { bestp = m; badp = a; } }
            printf("  packed %d/456 @%04x", bestp, badp);
            int pb[4] = {0,0,0,0};
            for (int i = 0; i < 456; i++) { int bit = (dsp->data[badp + i / 16] >> (15 - i % 16)) & 1; if (bit ==
             ↪  code[i]) pb[i & 3]++; }
            printf(" par burst %d %d %d %d /114", pb[0], pb[1], pb[2], pb[3]); }
          printf("\n");
          /* the decoder input at 0x2a00 (the trellis loop reads pairs from AR1 = 0x2a00) */
          if (g_snap_2a00_ok) { const char *d = calypso_getenv("PONT_NB_HIST"); if (d) { char nom[256];
           ↪  snprintf(nom, sizeof nom, "%s/entree_%u.txt", d, fn); FILE *f = fopen(nom, "w"); if (f) { for (int i
           ↪  = 0; i < 456; i++) fprintf(f, "%d\n", (int16_t)g_snap_2a00[i]); fclose(f); } } }
          if (g_snap_2a00_ok) { int mp = 0, mn = 0, m1 = 0, nz = 0; g_snap_2a00_ok = 0;
            for (int i = 0; i < 456; i++) { int16_t v = (int16_t)g_snap_2a00[i]; if (v) nz++;
                if ((v > 0) == (code[i] != 0)) mp++;
                if ((v < 0) == (code[i] != 0)) mn++;
                if ((v == 1) == (code[i] != 0)) m1++; }
            printf("  [entree] fn=%u 0x2a00..+456 : nonzero=%d  pos=1 %d/456  neg=1 %d/456  one=1 %d/456 |
↪  premiers:", fn, nz, mp, mn, m1);
            for (int i = 0; i < 48; i++) printf(" %d", (int16_t)g_snap_2a00[i]);
            printf("\n  [entree] attendu :"); for (int i = 0; i < 48; i++) printf(" %d", code[i]); printf("\n");
            /* per-burst view: bits k with k%4==b */
```

```
          for (int b = 0; b < 4; b++) { int m = 0, n = 0; for (int i = b; i < 456; i += 4) { int16_t v =
           ↪  (int16_t)g_snap_2a00[i]; if (v) { n++; if ((v < 0) == (code[i] != 0)) m++; } }
              printf("  [entree] burst %d : neg=1 %d/%d\n", b, m, n); } }
        /* the 228 decoder output bits: best match over memory, hard words and LSB */
        { uint8_t u[228];
          if (cellule_u228_attendu(fn, bsic, u) == 0) {
              int best[2] = {0,0}; unsigned bad[2] = {0,0};
              for (unsigned a = 0x60; a + 228 < C54X_DATA_SIZE; a++) {
                  int m0 = 0, m1 = 0;
                  for (int i = 0; i < 228; i++) { uint16_t v = dsp->data[a + i];
                      if ((v == 1 && u[i]) || (v == 0 && !u[i])) m0++;
                      if ((v & 1) == u[i]) m1++; }
                if (m0 > best[0]) { best[0] = m0; bad[0] = a; } if (m1 > best[1]) { best[1] = m1; bad[1] = a; } }
              printf("  [u228] fn=%u hard %d/228 @%04x  lsb %d/228 @%04x | ", fn, best[0], bad[0], best[1],
↪  bad[1]);
              /* mismatch map at 0x2d66 (hard) */
              unsigned a = 0x2d66; int e = 0; char map[240]; int k = 0;
              for (int i = 0; i < 228; i++) { uint16_t v = dsp->data[a + i]; int ok = (v == u[i]); if (!ok) e++;
               ↪  if (i == 184 || i == 224) map[k++] = '|'; map[k++] = ok ? '.' : (v > 1 ? '?' : 'x'); }
              map[k] = 0; printf("2d66: %d faux %s\n", e, map);
              /* packed forms of the 228 bits anywhere in memory: 16 per word MSB
               * first, LSB first, and the same with each word bit-reversed */
             { const char *nm[4] = { "msb", "lsb", "msb-rev", "lsb-rev" }; printf("  [u228] fn=%u packed:", fn);
                for (int f = 0; f < 4; f++) { int best = 0; unsigned bad = 0;
                  for (unsigned a = 0x60; a + 15 < C54X_DATA_SIZE; a++) { int m = 0;
                      for (int i = 0; i < 228; i++) { uint16_t w = dsp->data[a + i / 16]; int bi = i % 16;
                         int bit = (f == 0) ? (w >> (15 - bi)) & 1 : (f == 1) ? (w >> bi) & 1 : (f == 2) ? (w >>
                           ↪  bi) & 1 : (w >> (15 - bi)) & 1;
                          if (bit == u[i]) m++; }
                      if (m > best) { best = m; bad = a; } }
                  printf(" %s %d/228 @%04x", nm[f], best, bad); }
                printf(" | 2c3c:"); for (int i = 0; i < 16; i++) printf(" %04x", dsp->data[0x2c3c + i]);
↪  printf("\n"); } } }
        } }
    /* where did the delivered samples land in DARAM (AAD)? offset in words at
     * which the ROM's buffer equals our frame, and how many words match */
    { uint16_t aad = calypso_rhea_dma_get_daram(); int best = 0, boff = 0;
      for (int off = -8; off <= 8; off++) {
          int m = 0;
          for (int i = 0; i < g_dernier_n_iq; i++) {
              int a = (int)aad + off + i; if (a < 0 || a >= C54X_DATA_SIZE) continue;
              if ((int16_t)dsp->data[a] == g_dernier_iq[i]) m++;
          }
          if (m > best) { best = m; boff = off; }
      }
      int p0 = -1; for (int i = 0; i < 40; i++) if ((int16_t)dsp->data[aad + i] != 0) { p0 = i; break; }
      { /* runs of words that differ from what was delivered, at offset 0 */
        char runs[256]; int k = 0, i = 0;
        while (i < g_dernier_n_iq && k < 200) {
            if ((int16_t)dsp->data[aad + i] != g_dernier_iq[i]) {
                int j = i; while (j < g_dernier_n_iq && (int16_t)dsp->data[aad + j] != g_dernier_iq[j]) j++;
               k += snprintf(runs + k, sizeof runs - k, " %d+%d(%d)", i, j - i, (int16_t)dsp->data[aad + i]); i = j;
            } else i++;
        }
        runs[k] = 0;
        int d_dma = 0; for (int q = 0; q < g_dernier_n_iq && q < 384; q++) if ((int16_t)g_daram_apres_dma[q] !=
         ↪  g_dernier_iq[q]) d_dma++;
        printf("  [daram] fn=%u differences apres la trame (mot+longueur(valeur)):%s | juste apres le DMA, avant
↪  la ROM : %d mots differents (aad=%04x)\n", fn, runs, d_dma, g_daram_aad); }
       printf("  [daram] fn=%u aad=%04x livres=%d mots, identiques=%d a l'offset %d, premier mot non nul a +%d,
↪  mots 0..7: %d %d %d %d %d %d %d %d\n",
              fn, aad, g_dernier_n_iq, best, boff, p0,
              (int16_t)dsp->data[aad], (int16_t)dsp->data[aad+1], (int16_t)dsp->data[aad+2],
               ↪  (int16_t)dsp->data[aad+3],
              (int16_t)dsp->data[aad+4], (int16_t)dsp->data[aad+5], (int16_t)dsp->data[aad+6],
               ↪  (int16_t)dsp->data[aad+7]); }
```

```
    printf("\n");
}

static void servir(int fd, C54xState *dsp, uint16_t *api_ram, long insns, bool verbeux,
                   const char *iq_mode, int amp)
{
    bool injecter = iq_mode && *iq_mode && strcmp(iq_mode, "none") != 0;
    unsigned long injectes = 0;
    const uint16_t *a_sync = &api_ram[(API_NDB + NDB_A_SYNC_DEMOD) / 2];
    bool init_done = false;
    if (g_verif_sonde < 0) g_verif_sonde = calypso_getenv("CALYPSO_BSP_VERIF") ? 1 : 0;
    unsigned long trames = 0, irqs = 0, resets = 0;
    uint64_t insns_total = 0;
    const uint16_t *d_fb_det = &api_ram[(API_NDB + NDB_D_FB_DET) / 2];
    /* [2026-09-19] a_sch was read from R page 0 ONLY, hardcoded, while the DSP
     * writes its result to the page d_dsp_page designates, alternating. Stale
     * page-0 content read as a result is exactly what produces SB decodes that
     * are wrong yet REPEATABLE (measured: BSIC 44 six times, 22 five times out
     * of 17, never the 32 the capture carries). The PONT_CAN_SB hack in this
     * same file already writes BOTH pages, so the page was known to matter.
     * Both are kept here and the live one is picked per frame. */
    const uint16_t *a_sch_pg[2] = {
        &api_ram[(API_R_PAGE(0) + RP_A_SCH) / 2],
        &api_ram[(API_R_PAGE(1) + RP_A_SCH) / 2] };
    const uint16_t *a_sch0   = a_sch_pg[0];

    CalypsoPontMsg m;
    if (recv(fd, &m, sizeof(m), 0) != (ssize_t)sizeof(m) || m.type != PONT_HELLO ||
        m.a != CALYPSO_API_WORDS || m.b != CALYPSO_PONT_MAGIC) {
        fprintf(stderr, "pont : poignee de main invalide (type=%u a=%u)\n", m.type, m.a);
        return;
    }
    envoyer(fd, PONT_HELLO_OK, CALYPSO_API_WORDS, CALYPSO_PONT_MAGIC);
    printf("pont : ARM connecte, API RAM %u mots, %ld insn/trame\n", CALYPSO_API_WORDS, insns);
    fflush(stdout);

    /* [2026-10-21 18:35] PONT_RESET arrive pendant l'attente du PONT_GO (trame en deux phases) : il
     * etait JETE (« message 1 recu en attente de GO, ignore »). Le firmware TI relance le DSP en cours
     * de route (fn ~290) et attend alors LEAD_READY sans fin : l'ARM restait dans dsp_power_on+0x34.
     * Le reset est desormais garde et applique des la fin de la trame. */
    unsigned reset_differe = 0;
    uint32_t reset_differe_dl = 0;
    while (!g_stop) {
        if (g_pause) pause_cooperative(dsp);
        if (reset_differe) {
            reset_differe--;
            memset(&m, 0, sizeof m);
            m.type = PONT_RESET;
            m.a = reset_differe_dl;
            printf("pont : reset garde pendant l'attente du GO, applique maintenant\n");
        } else {
            struct pollfd pfd = { .fd = fd, .events = POLLIN };
            if (poll(&pfd, 1, 200) <= 0) {
                continue;
            }
            ssize_t n = recv(fd, &m, sizeof(m), 0);
            if (n != (ssize_t)sizeof(m)) {
                printf("pont : ARM deconnecte (%zd)\n", n);
                break;
            }
        }
        switch (m.type) {
        case PONT_RESET: {
            /* [2026-10-21 18:30] LE bootCode TI. Sur le Calypso C035 l'API est aussi l'espace PROGRAMME
             * 0xE000-0xFFFF du DSP (leadapi.h BASE_API_LEAD 0xE000). Le firmware TI (Magnetite,
             * LA_InitialLeadBoot, leadboot.c) y pose avant de relacher le reset un chargeur (0xf800), une
```

```
             * init (0xfe00) et le VECTEUR DE RESET (0xff80 : F073 FE00 = b 0xfe00) : chaque reset passe
             * par ce chargeur, qui affiche LEAD_READY. Ici 0xE000-0xFFFF etait la « PDROM » figee, et
             * c54x_reset recopie une image ROM dans la fenetre API des donnees : la ROM partait a chaud
             * (0xb3e7) au 2e reset, sans LEAD_READY, et l'ARM restait dans dsp_power_on+0x34 (attente sans
             * fin, NO_DWNLD). Signature presente : l'API est gardee telle que l'ARM l'a ecrite (un reset du
             * DSP ne touche pas la RAM partagee) et le DSP repart dans le chargeur, comme avec le bootCode.
             * Firmware osmocom (pas de signature) : inchange. */
            static uint16_t api_avant[0x2000];
            const bool boot_ti = api_ram[0xff80 - 0xe000] == 0xF073 && api_ram[0xff81 - 0xe000] == 0xFE00;
            if (boot_ti) memcpy(api_avant, api_ram, sizeof api_avant);
            c54x_reset(dsp);
            if (boot_ti) {
                /* [18:30] Executer le bootCode lui-meme (fenetre programme 0xE000) faisait planter le DSP
                 * (SP-CORRUPT a PC 0 : modes memoire du C035 non emules). On en garde la FONCTION : DSP
                 * pose dans le chargeur de la ROM (0xb410 : st #1,*(0xfff) puis attente du 2 et saut),
                 * avec les registres que la ROM y met (0xb3ff-0xb403). API gardee telle que l'ARM l'a ecrite. */
                memcpy(api_ram, api_avant, sizeof api_avant);
                for (int i = 0; i < 0x2000; i++) dsp->data[0x0800 + i] = api_avant[i];
                dsp->st1 = 0x2900; dsp->st0 = 0x1800; dsp->sp = 0x5AC8;
                dsp->pc = 0xb410;
                printf("pont : bootCode TI dans l'API (0xff80 = b 0xfe00) : API gardee, DSP dans le chargeur ROM
↪  0xb410\n");
            }
            dsp->running = true;
            init_done = false;
            api_ram[(API_NDB + NDB_D_DSP_PAGE) / 2] = 0;
            resets++;
            printf("pont : RESET #%lu (DL_STATUS=0x%04x) fn=%u pc=0x%04x\n",
                   resets, m.a, g_c54x_exe_fn, dsp->pc);
            break;
        }
        case PONT_TICK: {
            g_c54x_exe_fn = m.a;
            g_tick_irq_trame = (m.b & CALYPSO_PONT_TICK_IRQ_TRAME) != 0;
            apm_trace_tick(api_ram, m.a, m.b);
            bool deux_phases = (m.b & CALYPSO_PONT_TICK_DEUX_PHASES) != 0;
            bool done_envoye = false;
            double t_tick = chrono_ms(), t_done_a = t_tick, t_go = t_tick, t_done = 0;
            m.b &= 1u;
            enreg_tick(dsp, api_ram, m.a, g_tick_irq_trame, deux_phases, insns);
            enreg_api_diff(api_ram, m.a, 0);
            tx_sonde_avant(dsp, api_ram);
            calypso_bsp_set_tpu_offset((int)m.c);   /* firmware RX window */
            /* AFC relay, closing the loop. The ARM writes d_afc (word 15 of the W
             * page) into the shared API RAM; on silicon the DSP serialises it to
             * the TWL3025 over the TSP. Without this relay the sample rotation
             * stays frozen and the frequency error never converges below the SB
             * threshold. m.b carries d_dsp_page, bit 0 selecting the W page. */
            { unsigned wp = m.b & 1u;
              int16_t dac = (int16_t)api_ram[(wp ? 0x14u : 0x00u) + 15u];
              static int16_t prev; static int first = 1;
              /* [2026-09-19] Measurement before any fix: the AFC DAC never settles,
               * every correction is followed by a write of exactly -700
               * (afc_initial_dac_value). The suspicion is the dual-page write the
               * set_afc_dac filter already documents — page A holding the inherited
               * init while page B carries the correction — with the filter guarding
               * only against 0, not against -700. Print BOTH pages so the claim can
               * be checked instead of assumed. */
              { static unsigned na;
                int16_t d0 = (int16_t)api_ram[0x00u + 15u], d1 = (int16_t)api_ram[0x14u + 15u];
                static int16_t p0 = 0x7fff, p1 = 0x7fff;
                if ((d0 != p0 || d1 != p1) && na++ < 40)
                    printf("  [afc] fn=%u w_page=%u relaye=%d | page0=%d page1=%d\n",
                           m.a, wp, dac, d0, d1);
                p0 = d0; p1 = d1; }
              if (first || dac != prev) { calypso_twl3025_set_afc_dac(dac); prev = dac; first = 0; } }
```

```
{ static unsigned _to=0; if (calypso_getenv("PONT_TPU_DEBUG") && (_to<5 || _to%2000==0)) printf("
 ↪  [tpu] fn=%u tpu_offset=%u\n", m.a, m.c); _to++; }
trace_armer();
if (g_trace_reste > 0 && g_trace_f && !g_trace_ouverte && !g_trace_pc_hi) {
    /* wait for the first FB task posted by the ARM (W page 0 or 1) */
    static int md_cible = -1;
   if (md_cible < 0) { const char *e = calypso_getenv("PONT_TRACE_MD"); md_cible = (e && *e) ? atoi(e)
     ↪  : 5; }
    if (api_ram[(API_W_PAGE(0) + WP_D_TASK_MD) / 2] == md_cible ||
        api_ram[(API_W_PAGE(1) + WP_D_TASK_MD) / 2] == md_cible) {
        g_trace_ouverte = true;
        trace_cellules(dsp, true);
        fprintf(g_trace_f, "# d_task_md=%d vu a fn=%u page=%u ; IMR=%04x IFR=%04x pc=%04x\n",
                md_cible, m.a, m.b, dsp->imr, dsp->ifr, dsp->pc & 0xffff);
        printf("pont : trace FB ouverte a fn=%u\n", m.a);
    }
}
/* PONT_RX_APRES (default 1): deliver the burst AFTER the frame
 * interrupt, as on silicon, where the TPU programs the RX window
 * within the frame, the DSP arms DMA2 in the frame ISR, and the
 * samples land afterwards. Injecting before the interrupt instead
 * leaves DMA2 with ENABLE=0 when the RIF issues its RX request, and
 * the SB job then decodes the previous tick's window. */
/* PONT_RX_MODE : milieu (default, see jouer_trame) | apres | avant.
 * PONT_RX_APRES=0/1 is still honoured as avant/apres. */
static int rx_mode = -1;   /* 0 = milieu, 1 = apres, 2 = avant */
if (rx_mode < 0) { const char *e = calypso_getenv("PONT_RX_MODE"); const char *a =
 ↪  calypso_getenv("PONT_RX_APRES");
                  rx_mode = (e && !strcmp(e, "apres")) ? 1 : (e && !strcmp(e, "avant")) ? 2
                          : (a && *a == '1') ? 1 : (a && *a == '0') ? 2 : 0; }
bool rx_apres = (rx_mode == 1);
if (rx_mode == 2 && injecter && init_done) injecter_burst(dsp, iq_mode, amp, m.a, &injectes);
/* Real chain: drain UDP socket 6702 (bursts from the bridge/BTS) and
 * hand them to the DSP. This is the only source when synthetic
 * injection is off, and a no-op when the socket is empty. */
if (init_done && rx_mode != 0) calypso_bsp_service(m.a);
g_inj.actif = (rx_mode == 0 && init_done);
g_inj.iq_mode = (rx_mode == 0 && injecter) ? iq_mode : NULL;
g_inj.amp = amp; g_inj.fn = m.a; g_inj.injectes = &injectes; g_inj.udp = (rx_mode == 0 && init_done);
uint32_t ninsn = 0;
bool init_avant = init_done;
uint32_t drapeaux = jouer_trame(dsp, insns, &init_done, &ninsn, deux_phases ? 1 : 0);
if (deux_phases) {
    /* [2026-09-21] Phase A done: the ROM's frame ISR has written the
     * R page (previous burst) and armed the window. Tell QEMU, which
     * raises the ARM frame IRQ, waits for the end of l1_sync() and
     * sends PONT_GO; only then is the burst of this frame delivered.
     * That is the silicon order, and what keeps "BURST ID n!=m" and
     * "EMPTY" (prim_rx_nb.c) away. */
    sonde_pages("A", m.a, dsp, api_ram);
    enreg_api_prendre(api_ram);
    ombre_descendante(api_ram, m.a);
    /* [2026-10-20] mode TRX : les bits exacts du burst montant joue a la trame precedente,
     * dans la page de resultat que l'ARM lit apres ce DONE (calypso_bsp.c), et les bursts
     * descendants de tous les slots TX de cette page, publies dans l'anneau */
    {
        int w_page = (api_ram[(API_NDB + NDB_D_DSP_PAGE) / 2] & B_GSM_PAGE) ? 1 : 0;
        if (trx_page_mode()) tx_page_publier(dsp, m.a, w_page);
        calypso_bsp_trx_ul_prothese(dsp->data, m.a, w_page);
    }
    apm_jamais_nul(api_ram);
    envoyer(fd, PONT_DONE, drapeaux & ~PONT_DONE_API_IRQ, ninsn);
    t_done_a = chrono_ms();
    bool go = false;
    while (!g_stop && !go) {
        struct pollfd pg = { .fd = fd, .events = POLLIN };
```

```
                if (poll(&pg, 1, 2000) <= 0) { static unsigned nt; if (nt++ < 3) printf("pont : PONT_GO attendu
                    ↪  (fn=%u)\n", m.a); continue; }
                CalypsoPontMsg g;
                ssize_t ng = recv(fd, &g, sizeof(g), 0);
                if (ng != (ssize_t)sizeof(g)) { printf("pont : ARM deconnecte en attente de GO (%zd)\n", ng);
                    ↪  g_stop = 1; break; }
                if (g.type == PONT_GO) { go = true; t_go = chrono_ms(); }
                else if (g.type == PONT_RESET) { reset_differe = 1; reset_differe_dl = g.a; }   /* voir
                    ↪  reset_differe */
                else { static unsigned nx; if (nx++ < 3) printf("pont : message %u recu en attente de GO,
                    ↪  ignore\n", g.type); }
            }
            if (!go) break;
            sonde_pages("G", m.a, dsp, api_ram);
            enreg_api_diff(api_ram, m.a, 1);
            if (calypso_getenv("PONT_NB_DEBUG") && ((m.a % 51u) % 10u - 2) % 4 == 3 && m.a % 51u <= 5 &&
                (api_ram[API_R_PAGE(0) / 2] == 24 || api_ram[API_R_PAGE(1) / 2] == 24)) {
                static unsigned nq;
                if (nq++ < 8) {
                    printf("  [garde] fn=%u avant le burst 3, bursts precedents en memoire :", m.a);
                    for (int b = 1; b <= 3; b++) {
                        uint8_t bits[148]; if (cellule_bits_attendus(m.a - b, 42, bits) < 0) continue;
                        uint8_t t116[116]; memcpy(t116, bits + 3, 58); memcpy(t116 + 58, bits + 87, 58);
                        int best[3] = {0,0,0}; unsigned bad[3] = {0,0,0};
                        for (unsigned a = 0x60; a + 116 < C54X_DATA_SIZE; a++) {
                            int m0 = 0, m1 = 0, m2 = 0;
                            for (int i = 0; i < 116; i++) { int16_t v = (int16_t)dsp->data[a + i];
                                if ((v < 0) == (t116[i] != 0)) m0++;
                                if ((v > 0) == (t116[i] != 0)) m1++;
                                if ((v == 1 && t116[i]) || (v == 0 && !t116[i])) m2++; }
                            if (m0 > best[0]) { best[0] = m0; bad[0] = a; } if (m1 > best[1]) { best[1] = m1;
                                ↪  bad[1] = a; } if (m2 > best[2]) { best[2] = m2; bad[2] = a; }
                        }
                        /* packed 16 per word too */
                        int bestp = 0; unsigned badp = 0;
                        for (unsigned a = 0x60; a + 8 < C54X_DATA_SIZE; a++) { int mm = 0;
                          for (int i = 0; i < 116; i++) { int bit = (dsp->data[a + i / 16] >> (15 - i % 16)) &
                              ↪  1; if (bit == t116[i]) mm++; }
                            if (mm > bestp) { bestp = mm; badp = a; } }
                       printf(" b%d(fn %u): neg %d@%04x pos %d@%04x hard %d@%04x packed %d@%04x |", 3 - b, m.a
↪  - b, best[0], bad[0], best[1], bad[1], best[2], bad[2], bestp, badp);
                    }
                    printf("\n");
                }
            }
            uint32_t n2 = 0;
            g_insn_a = ninsn;
            if (g_done_tot < 0) { const char *e = calypso_getenv("PONT_DONE_TOT"); g_done_tot = !(e && *e ==
                ↪  '0'); }
            g_reste_b = 0;
            drapeaux = jouer_trame(dsp, insns, &init_done, &n2, 2);
            ninsn += n2; g_insn_b = n2;
            if (g_done_tot) {
                /* DONE tout de suite : QEMU repart, le DSP finit la trame ici. */
                ombre_descendante(api_ram, m.a);
                envoyer(fd, PONT_DONE, drapeaux & ~PONT_DONE_API_IRQ, ninsn);
                done_envoye = true;
                t_done = chrono_ms();
                if (g_reste_b > 0 && !dsp->idle && dsp->running) {
                    uint32_t av = dsp->insn_count;
                    c54x_run_profile(dsp, g_reste_b);
                    ninsn += dsp->insn_count - av;
                }
                g_reste_b = 0;
            }
        }
```

```
g_inj.actif = false;
sonde_pages("B", m.a, dsp, api_ram);
if (g_inj.dernier_type == 'B' && (api_ram[API_R_PAGE(0) / 2] == 24 || api_ram[API_R_PAGE(1) / 2] ==
 ↪  24)) sonde_bits(m.a, dsp, 42);
/* Reference probe (CALYPSO_BSP_VERIF=1): compare DARAM against the
 * burst the BSP was handed, AFTER the DSP has run — the samples
 * only reach DARAM through the DSP's own DMA draining the RIF, so
 * there is nothing to compare before jouer_trame. */
if (g_verif_sonde) {
    uint32_t vfn = 0; uint16_t vad = 0; int vn = 0, vage = 0;
    int ident = calypso_bsp_verif_compare(&vfn, &vad, &vn, &vage);
    if (ident >= 0 && vn > 0) {
        static unsigned nv;
        if (nv++ < 4000) {
            uint16_t vpg = calypso_bsp_verif_last_page();
            printf("  [verif] fn=%u p51=%u age=%d page=0x%04x w_page=%d : "
                   "%d/%d en 0x%04x %s\n",
                   vfn, vfn % 51u, vage, vpg, (int)(vpg & 1u),
                   ident, vn, vad, ident == vn ? "VALIDE" : "partiel");
        }
    }
}
if (rx_apres && injecter && init_done && dsp->running) {
    injecter_burst(dsp, iq_mode, amp, m.a, &injectes);
    /* DMA completion inside the same frame: wake on the held INT10n
     * line, then let the DSP run the completion ISR and background
     * work through to IDLE. */
    uint32_t avant = dsp->insn_count;
    if (dsp->idle && calypso_rhea_dma_irq_level() && (dsp->imr & (1u << 14)) &&
        !(dsp->ifr & (1u << 14)))
        c54x_interrupt_ex(dsp, 30, 14);
    if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
    if (!dsp->idle) c54x_run_profile(dsp, (int)insns);
    ninsn += dsp->insn_count - avant;
}
/* [2026-09-20] STREAM PUMP (same as rejouer.c). DMA2 now fills its
 * double buffer with full pages and interrupts once per pair; the
 * ROM's ISR consumes both halves and the DSP idles. Then the next
 * pair is handed over, until the receiver holds less than a pair. A
 * 156.25-symbol frame is 3.25 pages, so this runs 1 or 2 times. */
for (int k = 0; k < 40 && dsp->running; k++) {   /* 13 page pairs per 1250-symbol frame */
    /* [2026-09-29] un SCH garde parce que la DMA n'etait pas armee au
     * depot est relivre ici, des que la ROM (phase B) l'a armee. */
    int relivre = calypso_bsp_sb_retenter();
    /* [2026-10-21 22:45] dedie TI : fenetre de mesure servie / NB garde relivre (calypso_bsp.c) ;
     * on laisse alors tourner le DSP (fin de DMA -> ISR -> fenetre suivante) dans le meme tick */
    relivre |= calypso_bsp_nb_retenter();
    if (!calypso_rhea_dma_pump(dsp) && !relivre) break;
    if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
    uint32_t av2 = dsp->insn_count;
    if (!dsp->idle) c54x_run_profile(dsp, (int)insns / 4);
    ninsn += dsp->insn_count - av2;
    drapeaux = (drapeaux & ~PONT_DONE_IDLE) | (dsp->idle ? PONT_DONE_IDLE : 0);
}
/* [2026-10-15] PORT VOIX (VSP, calypso_vsp.c, CALYPSO_VSP=1) : les 36 ou 37
 * echantillons PCM de la trame, chacun par l'ISR 0xc14a de la ROM (IT 20). */
if (init_done && dsp->running)
    ninsn += calypso_vsp_trame(dsp, c54x_run_profile, m.a);
drapeaux = (drapeaux & ~PONT_DONE_IDLE) | (dsp->idle ? PONT_DONE_IDLE : 0);
/* [2026-09-20] NB probe (PONT_NB_DEBUG=1): what the ROM leaves in the
 * R pages for a normal-burst task (ALLC/BCCH), per burst: d_task_d,
 * d_burst_d, a_serv_demod (TOA, PM, ANGLE, SNR) and, on burst 3,
 * the a_cd header of the NDB (Fire/CRC word, bit errors) plus the
 * first decoded octets. Read alongside mobile's "Dropping frame
 * with N bit errors". */
{
```

```
                static int nbdbg = -1; static unsigned nbn;
                if (nbdbg < 0) nbdbg = calypso_getenv("PONT_NB_DEBUG") ? 1 : 0;
                /* [2026-09-22] Plafond porte de 400 a 20000 : les 400 etaient
                 * consommees par le campement avant toute connexion dediee,
                 * et la sonde etait donc muette quand on en avait besoin. */
                if (nbdbg && nbn < 20000) {
                    static uint16_t prev[2][4];
                    for (int pg = 0; pg < 2; pg++) {
                        const uint16_t *r = &api_ram[API_R_PAGE(pg) / 2];
                        uint16_t cur[4] = { r[RP_D_TASK_D/2], r[RP_D_BURST_D/2], r[RP_A_SERV_DEMOD/2 + D_TOA],
                         ↪  r[RP_A_SERV_DEMOD/2 + D_PM] };
                        if (r[RP_D_TASK_D/2] && memcmp(cur, prev[pg], sizeof cur)) {
                            const uint16_t *w0 = &api_ram[API_W_PAGE(0) / 2], *w1 = &api_ram[API_W_PAGE(1) / 2];
                           printf("  [nb] fn=%u p51=%u R%d task_d=%u burst_d=%u TOA=%d PM=%d ANGLE=%d SNR=%u | W0
↪  task_d=%u burst=%u W1 task_d=%u burst=%u | livre=%c n_iq=%d one_shot=%d len=%u mots",
                                   m.a, m.a % 51u, pg, r[RP_D_TASK_D/2], r[RP_D_BURST_D/2],
                                   (int16_t)r[RP_A_SERV_DEMOD/2 + D_TOA], (int16_t)r[RP_A_SERV_DEMOD/2 + D_PM],
                                   (int16_t)r[RP_A_SERV_DEMOD/2 + D_ANGLE], r[RP_A_SERV_DEMOD/2 + D_SNR],
                                   w0[0], w0[1], w1[0], w1[1], g_inj.dernier_type, g_inj.dernier_n_iq,
                                   calypso_rhea_dma_one_shot(), calypso_rhea_dma_get_len_words());
                            if (r[RP_D_BURST_D/2] == 3) {
                                const uint16_t *cd = &api_ram[(API_NDB + NDB_A_CD) / 2];
                                printf(" | a_cd=%04x %04x %04x :", cd[0], cd[1], cd[2]);
                                for (int k = 3; k < 15; k++) printf(" %04x", cd[k]);
                            }
                            printf("\n"); nbn++;
                        }
                        memcpy(prev[pg], cur, sizeof cur);
                    }
                }
            }
            /* Canned results: scaffolding to prove the pipeline through to the
             * LU, not an end state. PONT_CAN_TOA=23 forces the reported TOA
             * (a_sync_demod[D_TOA]) to the on-time value the firmware expects
             * (prim_fbsb.c subtracts 23). Drop it once the correlator's native
             * TOA is correct. -1, the default, disables canning. */
            {
                static int can_toa = -2;
                if (can_toa == -2) { const char *e = calypso_getenv("PONT_CAN_TOA"); can_toa = (e && *e) ? atoi(e)
                 ↪  : -1; }
                if (can_toa >= 0 && *d_fb_det)
                    api_ram[(API_NDB + NDB_A_SYNC_DEMOD) / 2 + D_TOA] = (uint16_t)can_toa;
                /* SB: the firmware reads a_serv_demod[D_TOA] from the read page
                 * and expects about 4. PONT_CAN_SB_TOA=4 pins it on both R
                 * pages, which frames the SCH correctly. */
                /* [2026-10-21] PONT_CAN_NB_TOA=<v> (Magnetite : 3) : la L1 TI ne converge pas sur le TOA
                 * des bursts normaux (l1ctl_toa_update : ISH = (TOA+1)*4 - 16 qbits, le banc ne deplace
                 * pas le burst quand tpu_offset bouge) ; on le fixe sur les pages R portant une tache NB
                 * (d_task_d != 0). Le TOA du SB (a_sch, 23) reste natif : la synchro en depend. */
                static int can_nb = -2;
                if (can_nb == -2) { const char *e = calypso_getenv("PONT_CAN_NB_TOA"); can_nb = (e && *e) ? atoi(e)
                 ↪  : -1;
                                    if (can_nb >= 0) printf("  [pont] PONT_CAN_NB_TOA=%d : TOA des NB fixe\n",
                                     ↪  can_nb); }
                /* init_done : pendant le boot ces mots de l'API RAM portent le code DSP que l'ARM
                 * telecharge ; y ecrire cassait le telechargement (« occupe boot », 21/10 17:01). */
                if (can_nb >= 0 && init_done) {
                    for (int pg = 0; pg < 2; pg++) {
                        uint16_t *r = &api_ram[API_R_PAGE(pg) / 2];
                        if (r[RP_D_TASK_D / 2] && r[RP_D_TASK_D / 2] <= 40) r[RP_A_SERV_DEMOD / 2 + D_TOA] =
                         ↪  (uint16_t)can_nb;
                    }
                }
                static int can_sb = -2;
                if (can_sb == -2) { const char *e = calypso_getenv("PONT_CAN_SB_TOA"); can_sb = (e && *e) ? atoi(e)
                 ↪  : -1; }
```

```
                if (can_sb >= 0) {
                    api_ram[(API_R_PAGE(0) + RP_A_SERV_DEMOD) / 2 + D_TOA] = (uint16_t)can_sb;
                    api_ram[(API_R_PAGE(1) + RP_A_SERV_DEMOD) / 2 + D_TOA] = (uint16_t)can_sb;
                }
                /* Full SB canning (PONT_CAN_SB=<bsic>): write the SB result
                 * (a_sch) with CRC OK, the BSIC and the real frame number, the
                 * way qemu-calypso's shunt_encode_sb does. The frame number comes
                 * from the last delivered burst (real BTS, via
                 * calypso_bsp_get_last_fn) or from m.a. Applies only when the ARM
                 * asks for SB (d_task_md=6 on a W page) and on an SCH frame,
                 * fn%51 in {1,11,21,31,41}. */
                static int can_sb_full = -2, can_sb_bsic = 7;
                if (can_sb_full == -2) { const char *e = calypso_getenv("PONT_CAN_SB"); can_sb_full = (e && *e) ? 1
                 ↪  : 0; if (e && *e) can_sb_bsic = atoi(e) & 0x3f; }
                if (can_sb_full) {
                    int md0 = api_ram[4] & 0xff, md1 = api_ram[0x18] & 0xff;   /* d_task_md on W pages 0 and 1 */
                    if (md0 == 6 || md1 == 6) {
                        /* [2026-09-19] The frame MUST be the tick's own (m.a), not
                         * calypso_bsp_get_last_fn(): measured, the two diverged badly
                         * (the canned SB announced fn=5662 while the firmware sat at
                         * 11261), and Synchronize_TDMA then locked the ARM onto a frame
                         * number unrelated to what the injector delivers — every burst
                         * after the sync misaligned. A canned SB has to carry the clock
                         * the cell is actually generated from, or it proves nothing. */
                        uint32_t bfn = m.a;
                        uint32_t p51 = bfn % 51;
                        if (!(p51 % 10 == 1 && p51 <= 41)) bfn += (51 + 1 - (int)p51) % 51;  /* snap to an SCH frame */
                        uint32_t sb = pont_encode_sb((uint8_t)can_sb_bsic, bfn / 1326, bfn % 26, bfn % 51);
                        for (int pg = 0; pg < 2; pg++) {
                            uint16_t *a = &api_ram[(API_R_PAGE(pg) + RP_A_SCH) / 2];
                            a[0] = 0x8000;                /* B_SCH_CRC clear = CRC OK */
                            a[1] = 0x2034;                /* echo value, as the DSP writes it */
                            a[3] = (uint16_t)(sb & 0xffff);
                            a[4] = (uint16_t)(sb >> 16);
                        }
                        static int nlog = 0;
                        if (nlog < 8) { printf("  [can-sb] fn=%u -> sb=0x%08x BSIC=%d (T1=%u T2=%u T3=%u)
                         ↪  hacks=%s\n",
                                               bfn, sb, can_sb_bsic, bfn/1326, bfn%26, bfn%51, hacks_actifs());
↪  nlog++; }
                    }
                }
            }
            if (*d_fb_det) calypso_bsp_toa_feedback((int)(int16_t)a_sync[0]);  /* native TOA tracking loop */
            /* [2026-09-21] LIEN MONTANT. Sous CALYPSO_DSP_EXTERN=1 la couche 1
             * gr-gsm est desactivee, donc les hooks qui publiaient le montant
             * (calypso_l1_do_rach_written / _page_written) sont des no-op et le
             * RACH du mobile ne quittait jamais l'API RAM : pont.py comptait
             * « UL bursts=0 rach=0 », aucune IMM ASS, aucun LU ACCEPT. On
             * scrute ici la page W que l'ARM vient de remplir (m.b = d_dsp_page)
             * et on alimente les memes side-bands /dev/shm qu'en montage grgsm. */
            tx_inject_sortie(dsp, m.a);
            tx_ombre_compare(dsp, m.a);
            tx_rom_publier(dsp, m.a);
            tx_tsp_publier(dsp, m.a);
            tx_sonde_apres(dsp, api_ram, m.a);
            dsp_global_tx = dsp;
            { int blud_avant = (api_ram[(API_NDB + NDB_A_CU) / 2] & B_BLUD) != 0;
              montant_scruter(api_ram, m.a, m.b & 1u);
              tx_inject_apres(api_ram, m.a, blud_avant);
              tx_ombre_pose(api_ram, blud_avant && g_inj_n <= 0);
              tx_rom_pose(api_ram, blud_avant); }
            trames++;
            insns_total += ninsn;
            if (drapeaux & PONT_DONE_API_IRQ) irqs++;
            enreg_api_prendre(api_ram);
```

```
            if (!done_envoye) { apm_jamais_nul(api_ram); ombre_descendante(api_ram, m.a); envoyer(fd, PONT_DONE,
             ↪  drapeaux, ninsn); }
            { double t_fin = chrono_ms(); chrono_trame(t_tick, t_done_a, t_go, t_done > 0 ? t_done : t_fin, t_fin,
             ↪  m.a); }
            if (!init_avant && init_done) {
                printf("pont : DSP boote (premier IDLE) fn=%u insn=%u\n", m.a, dsp->insn_count);
            }
            /* frame at which the ARM posts the FB/SB task (W0 word 4, W1 word 0x18) */
            { static int prev5 = -1, prev6 = -1, ncmd = 0;
              int md0 = api_ram[4] & 0xff, md1 = api_ram[0x18] & 0xff;
              int has5 = (md0 == 5 || md1 == 5), has6 = (md0 == 6 || md1 == 6);
              if (has5 && prev5 != 1 && ncmd < 24) { printf("  [cmd] fn=%u tache FB postee par l'ARM\n", m.a);
               ↪  ncmd++; }
             { static int ncmd6; if (has6 && prev6 != 1 && ncmd6 < 60) { printf("  [cmd] fn=%u tache SB postee par
               ↪  l'ARM (W0 md=%d W1 md=%d)\n", m.a, md0, md1); ncmd6++; } }
              if (has6 && prev6 != 1 && ncmd < 24) { printf("  [cmd] fn=%u tache SB postee par l'ARM\n", m.a);
               ↪  ncmd++; }
              if (has6 && prev6 != 1) calypso_bsp_sb_trace(8);   /* [2026-09-29] et autour de chaque tache SB */
              prev5 = has5; prev6 = has6; }
            { static int fb_prev = 0; if (*d_fb_det && !fb_prev) { printf("  [jalon] fn=%u d_fb_det=1
             ↪  hacks=%s\n", m.a, hacks_actifs());
                                                                   calypso_bsp_sb_trace(60); }   /* [2026-09-29]
↪  chaque SCH trace jusqu'aux tentatives SB */
              fb_prev = *d_fb_det != 0; }
            /* [2026-09-19] ONE LINE PER SB ATTEMPT. Everything upstream of the SB
             * task is now measured correct — right frame, right window, right
             * sample offset — yet the CRC passes in 0.4% of attempts. A decode
             * that were simply broken would give 0%, so something DISCRIMINATES
             * the rare successes. This logs, for every frame the ARM has an SB
             * task posted, the TOA the FB left behind (a_sync[0], the value the
             * firmware positions the window from) next to the CRC outcome, so
             * the two populations can be compared directly. */
            /* The a_sync cells are already cleared by the time the SB task is
             * posted (measured: toa=pm=ang=0 on every attempt), so the FB result
             * has to be LATCHED when d_fb_det rises and carried to the attempt. */
            static int lat_toa, lat_pm, lat_ang; static uint32_t lat_fn;
            { static int fbl = 0;
              if (*d_fb_det && !fbl) {
                  lat_toa = (int)(int16_t)a_sync[0]; lat_pm = a_sync[1];
                  lat_ang = (int)(int16_t)a_sync[2]; lat_fn = m.a;
              }
              fbl = *d_fb_det != 0; }
            { static int sb_prev = 0;
              int md_0 = api_ram[4] & 0xff, md_1 = api_ram[0x18] & 0xff;
              int sb_now = (md_0 == 6 || md_1 == 6);
              if (sb_now && !sb_prev) {
                  static unsigned nsb;
                  if (nsb++ < 4000)
                      printf("  [sb] fn=%u p51=%u | FB a fn=%u toa=%d pm=%u ang=%d "
                             "| TOA-ROM=%d src=%s "
                             "| a_sch=%04x %s\n",
                             m.a, m.a % 51u, lat_fn, lat_toa, lat_pm, lat_ang,
                             g_toa_valeur,
                             g_toa_grille < 0 ? "?" : (g_toa_grille ? "GRILLE(0x0cce)" : "fine"),
                             a_sch0[0],
                             ((a_sch0[0] & 0x8100) == 0x8000) ? "CRC_OK" : "crc_ko");
              }
              sb_prev = sb_now; }
            /* CRC watched on BOTH R pages, and the page reported: reading page 0
             * alone cannot tell a real decode from stale content. A genuine SCH
             * gives the SAME BSIC every time (32 for cellule_reelle.bin); a
             * value that changes at each hit is a 10-bit CRC passing by chance
             * (1/1024) or a stale cell. */
            /* [2026-09-19] Who writes a_sch at all? The DSP never does: the
             * A_SCH-WR probe in c54x_mem.c (live, its ANGLE-WR neighbour fires)
             * counted ZERO stores by the ROM to 0x0837..0x083b / 0x084b..0x084f.
```

```
 * The only host writer is the PONT_CAN_SB block below, which is off.
 * Yet the cells change. The remaining writer is the ARM, through the
 * shared mapping, which no DSP-side probe can see -- so watch the
 * VALUES from here and name the frame. */
/* [2026-09-19] Who maintains d_dsp_page? calypso_api.h: an armed page
 * is B_GSM_TASK|page = 0x0002 or 0x0003; 0x0000 is the reset state
 * l1s_reset_hw() writes (sync.c:165), and it also puts the firmware
 * back on R page 0. Measured at 0x0000 on 39% of samples, because
 * the FBSB loop restarts ~12000 times. On silicon the DSP re-arms
 * the page itself; this names every transition and its writer. */
{ static uint16_t dpp; static int dfirst = 1; static unsigned ndp;
  uint16_t dp = api_ram[(API_NDB + NDB_D_DSP_PAGE) / 2];
  if (!dfirst && dp != dpp && ndp < 40) {
      printf("  [page] fn=%u d_dsp_page %04x -> %04x  (%s)\n", m.a, dpp, dp,
             (dp & 0x0002) ? ((dp & 1) ? "arme page 1" : "arme page 0")
                           : "NON ARME (etat de reset)");
      ndp++;
  }
  dpp = dp; dfirst = 0; }
{ static uint16_t prev[2][5]; static int first = 1; static unsigned nch;
  for (int pg = 0; pg < 2; pg++) {
      const uint16_t *a = a_sch_pg[pg];
      /* [2026-09-29] plafond 60 -> 600 et TOA de la page R : c'est la
       * mesure du cadrage SB (un SCH bien pose lit TOA=23). */
      if (!first && nch < 600 &&
          (a[0] != prev[pg][0] || a[3] != prev[pg][3] || a[4] != prev[pg][4])) {
          const uint16_t *rp = &api_ram[API_R_PAGE(pg) / 2];
          printf("  [a_sch] fn=%u page=%d : %04x %04x %04x %04x -> "
                 "%04x %04x %04x %04x  (d_dsp_page=%04x) TOA=%d PM=%d SNR=%u %s\n",
                 m.a, pg, prev[pg][0], prev[pg][1], prev[pg][3], prev[pg][4],
                 a[0], a[1], a[3], a[4],
                 api_ram[(API_NDB + NDB_D_DSP_PAGE) / 2],
                 (int16_t)rp[RP_A_SERV_DEMOD/2 + D_TOA], (int16_t)rp[RP_A_SERV_DEMOD/2 + D_PM],
                 rp[RP_A_SERV_DEMOD/2 + D_SNR],
                 ((a[0] & 0x8100) == 0x8000) ? "CRC_OK" : (a[0] & 0x0100) ? "crc_ko" : "");
          nch++;
      }
      prev[pg][0]=a[0]; prev[pg][1]=a[1]; prev[pg][3]=a[3]; prev[pg][4]=a[4];
  }
  first = 0; }
{ static int crc_prev[2] = {1, 1};
  for (int pg = 0; pg < 2; pg++) {
      const uint16_t *a = a_sch_pg[pg];
      /* [2026-09-19] The criterion (a[0] & 0x8100) == 0x8000 counts
       * SATURATED ACCUMULATORS as CRC OK: a_sch[0] has been seen
       * carrying plain numbers, and 0x8000 is exactly what a
       * saturated accumulator stores (rejouer.c:923). Every "SB CRC
       * OK" of this session rested on it, so it is replaced by a
       * test the arithmetic cannot pass by accident: GSM 04.08
       * bounds T2 <= 25 and T3' <= 4, so 6 of 32 T2 values and 3 of
       * 8 T3' values are IMPOSSIBLE in a real SCH. */
      uint32_t _sb = (uint32_t)a[3] | ((uint32_t)a[4] << 16);
      unsigned _t2  = (_sb >> 18) & 0x1f;
      unsigned _t3p = ((_sb >> 24) & 1) | ((_sb >> 15) & 6);
      int _plausible = (_t2 <= 25) && (_t3p <= 4);
      int crc_ok = ((a[0] & 0x8100) == 0x8000) && _plausible;
      if (crc_ok && !crc_prev[pg]) {
          uint32_t sb = (uint32_t)a[3] | ((uint32_t)a[4] << 16);
          printf("  [jalon] fn=%u SB PLAUSIBLE page=%d BSIC=%u "
                 "a_sch=%04x %04x %04x %04x  hacks=%s\n",
                 m.a, pg, (unsigned)((sb >> 2) & 0x3f),
                 a[0], a[1], a[3], a[4], hacks_actifs());
      }
      crc_prev[pg] = crc_ok;
  } }
if ((trames % 217) == 0) {
```

```
                profil_publier();
                pcc_publier();
                dump_publier(dsp);
            }
            if (verbeux || (trames % 217) == 0) {
                printf("  fn=%-7u page=%u insn=%-7u %s%s%s  d_fb_det=%-5u a_sch=%04x %04x %04x %04x"
                       " sync=%04x %04x %04x  | trames=%lu irq=%lu iq=%lu\n",
                       m.a, m.b, ninsn,
                       (drapeaux & PONT_DONE_IDLE) ? "IDLE " : "occupe ",
                       (drapeaux & PONT_DONE_API_IRQ) ? "IRQ-API " : "",
                       (drapeaux & PONT_DONE_INIT) ? "" : "boot ",
                       *d_fb_det, a_sch0[0], a_sch0[1], a_sch0[2], a_sch0[3],
                       a_sync[0], a_sync[1], a_sync[2], trames, irqs, injectes);
                if ((trames % (217*8)) == 0) printf("  hacks actifs : %s ; cadencement : %s ; stimulus : %s\n",
                                                    hacks_actifs(), cadencement_actif(), iq_mode ? iq_mode : "none");
            }
            fflush(stdout);
            break;
        }
        case PONT_DCCH:
            /* [2026-09-21] QEMU a lu le canal dedie dans le flux L1CTL du
             * firmware (calypso_dcch_tap.c) : a = TN, b = genre, c = sous-voie.
             * Le BSP en a besoin pour savoir quel intervalle de temps livrer :
             * le pont lui envoie les huit, il n'en joue qu'un par tick, et sans
             * ca c'est toujours TS0 - donc rien du SDCCH du mobile. */
            if (m.b == 4)
                printf("pont : canal dedie arme : TS%u PDCH (GPRS)\n", m.a);
            else
                printf("pont : canal dedie %s : TS%u SDCCH/%s SS=%u\n",
                       m.b == 0xFF ? "libere" : "arme", m.a, m.b ? "8" : "4", m.c);
            calypso_bsp_set_dedie((int)m.a, (int)m.b, (int)m.c);
            if (m.b == 0xFF || (int)m.a <= 0) {
                montant_canal_libere();
            }
            break;
        case PONT_BYE:
            printf("pont : BYE\n");
            return;
        default:
            fprintf(stderr, "pont : message inconnu type=%u\n", m.type);
            break;
        }
    }
    printf("pont : bilan de la session : %lu trames, %lu IRQ API, %lu reset, %llu insn\n",
           trames, irqs, resets, (unsigned long long)insns_total);
    montant_bilan();
}

int pont_serveur(C54xState *dsp, uint16_t *api_ram, const char *socket_path,
                 long insns, bool verbeux, const char *iq_mode, int amp)
{
    signal(SIGINT, sur_signal);
    signal(SIGTERM, sur_signal);
    signal(SIGUSR1, sur_pause);
    signal(SIGUSR2, sur_pause);
    { static int16_t rempl[2 * 148];         /* TS1..TS7 of the C0 carrier: dummy bursts */
      cellule_factice(amp, 0.5, rempl);
      calypso_bsp_set_remplissage(rempl, 2 * 148); }

    int srv = socket(AF_UNIX, SOCK_SEQPACKET, 0);
    if (srv < 0) {
        fprintf(stderr, "pont : socket : %s\n", strerror(errno));
        return 1;
    }
    struct sockaddr_un sa = { .sun_family = AF_UNIX };
    snprintf(sa.sun_path, sizeof(sa.sun_path), "%s", socket_path);
```

```
    unlink(socket_path);
    if (bind(srv, (struct sockaddr *)&sa, sizeof(sa)) < 0 || listen(srv, 1) < 0) {
        fprintf(stderr, "pont : bind/listen(%s) : %s\n", socket_path, strerror(errno));
        close(srv);
        return 1;
    }
    chmod(socket_path, 0666);
    if (iq_mode && *iq_mode && strcmp(iq_mode, "none") != 0)
        printf("pont : injection I/Q « %s » amplitude %d a chaque trame (DARAM 0x%04x, %u mots)\n",
               iq_mode, amp, calypso_bsp_get_daram_addr(), calypso_bsp_get_daram_len());
    printf("pont : en attente de l'ARM sur %s (API RAM : /dev/shm%s)\n"
           "       cote QEMU : CALYPSO_DSP_EXTERN=1 qemu-system-arm -M calypso ...\n",
           socket_path, CALYPSO_PONT_SHM);
    fflush(stdout);

    while (!g_stop) {
        struct pollfd pfd = { .fd = srv, .events = POLLIN };
        if (poll(&pfd, 1, 200) <= 0) {
            continue;
        }
        int fd = accept(srv, NULL, NULL);
        if (fd < 0) {
            continue;
        }
        servir(fd, dsp, api_ram, insns, verbeux, iq_mode, amp);
        close(fd);
        fflush(stdout);
    }
    close(srv);
    unlink(socket_path);
    return 0;
}
```

3.21 /opt/GSM/c54x_exe/src/pont.h

1062 octets, 23 lignes → 23 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
#ifndef PONT_H
#define PONT_H
#include <stdint.h>
#include <stdbool.h>
#include "calypso_c54x.h"

/* Allocates the C54xState with data[] page-aligned, then maps the shared segment
 * OVER data[0x0800..0x27FF] (protocol in calypso_dsp_pont.h). Replaces
 * c54x_init() in --arm mode; api_ram must then be &dsp->data[C54X_API_BASE]. */
C54xState *pont_allouer_dsp(void);

/* Serves QEMU's ARM: one connection after another, runs one frame per TICK,
 * answers DONE. Returns only on SIGINT/SIGTERM.
 * iq_mode: NULL/"none" = nothing, "fcch" = synthetic FCCH burst (+pi/2 rotation
 * per sample, 1 sample/symbol), "noise" = noise, "tone:<dphi>" = dphi radians
 * per sample, "cell[:bsic[:decalage[:marge]]]" = full 51-multiframe cell.
 * amp = int16 amplitude (real FCCH bursts run at ~32500 rms). Injected on every
 * frame through calypso_bsp_rx_burst(). */
int pont_serveur(C54xState *dsp, uint16_t *api_ram, const char *socket_path,
                 long insns, bool verbeux, const char *iq_mode, int amp);

#endif
```

3.22 /opt/GSM/c54x_exe/src/rejouer.c

161845 octets, 2596 lignes → 2593 lignes (2 groupes compactés)

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * rejouer.c - deterministic replay of FB/SB acquisition on the real DSP.
 *
 * Two identical runs of the full bench (QEMU + osmocon + mobile) diverged: the
 * ARM (TCG) advances at host speed between frame interrupts, so the frame on
 * which it posts the FB task changes from run to run. Here the ARM is NOT
 * emulated: its layer 1 is replayed in C (l1_sync + fb_sched_set/sb_sched_set
 * from prim_fbsb.c/sync.c), injection is locked to the frame counter, and
 * nothing reads the host clock. Same input => same output, always.
 *
 * This is not an operating mode: it is a DSP measurement bench (FB then SB)
 * driven by the same command sequence as the real firmware.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include "calypso_c54x.h"
#include "calypso_bsp.h"
#include "hw/arm/calypso/calypso_api.h"
#include "cellule.h"
#include "calypso_twl3025.h"
#include "calypso_rhea_dma.h"
#include "rejouer.h"
#include <osmocom/core/bits.h>
#include <osmocom/core/crcgen.h>
#include <osmocom/coding/gsm0503_parity.h>
#include "hw/arm/calypso/calypso_debug.h"

/* ---- API offsets in WORDS from the API base (= DSP 0x800) ---------------- */
#define W_PAGE(p)     ((p) ? 0x14u : 0x00u)   /* T_DB_MCU_TO_DSP, 17 words */
#define R_PAGE(p)     ((p) ? 0x3Cu : 0x28u)   /* T_DB_DSP_TO_MCU, 20 words */
#define W_SIZE        17u
#define R_SIZE        20u
#define W_TASK_D      0u
#define W_TASK_MD     4u
#define W_CTRL_ABB    11u
#define W_AFC         15u
#define W_CTRL_SYS    16u
#define NDB           0xD4u
#define NDB_PAGE      (NDB + 0u)
#define NDB_ERRSTAT   (NDB + 1u)
#define NDB_FB_DET    (NDB + 36u)
#define NDB_FB_MODE   (NDB + 37u)
#define NDB_SYNC      (NDB + 38u)   /* a_sync_demod[TOA,PM,ANGLE,SNR] */
#define R_SERV        8u            /* a_serv_demod[4] */
/* [2026-09-18] SYMBOL SAMPLING OFFSET. Replay sampled the GMSK at 0.0, i.e. on
 * the symbol BOUNDARY, while the bridge uses 0.5, the CENTER. With the gaussian
 * pulse of gmsk.c (BT=0.3), the weight of the target symbol in the phase
 * difference between two consecutive samples is:
 *
 *   offset   target symbol   previous neighbour   next neighbour
 *    0.0         0.454            0.023               0.492
 *    0.25        0.590            0.069               0.333
 *    0.5         0.653            0.158               0.189
 *
 * At 0.0 the NEXT neighbour outweighs the symbol being read: the offset is not
 * noisy, it is UNDETERMINED, and the demodulator may latch onto either one. That
 * accounts for offset=22 at margin=21, midamble at 56/63, data at 71-73/78 with
 * otherwise perfect samples, and the correlation peak jumping between adjacent
```

```
 * lags. */
/* Tunable for sweeps: 0.5 (symbol center) is argued above, but it is a
 * hypothesis and the bench must be able to test it.
 * REJEU_DECALAGE_SYMB=<x>, default 0.5. */
static double decalage_symb(void)
{
    static double d = -1.0;
    if (d < 0) { const char *e = calypso_getenv("REJEU_DECALAGE_SYMB");
                 d = (e && *e) ? atof(e) : 0.5;
                 if (d < 0 || d >= 1.0) d = 0.5; }
    return d;
}
#define DECALAGE_SYMB decalage_symb()
/* Tunable head margin (REJEU_MARGE, default 21); the tail fills out to 190 complex samples */
static int marge_tete(void)
{
    static int m = -1;
    if (m < 0) { const char *e = calypso_getenv("REJEU_MARGE"); m = e ? atoi(e) : 21;
                 if (m < 0 || m > 41) m = 21;
                 cellule_marge_fin = 190 - 148 - m; }
    return m;
}

#define R_SCH         15u           /* a_sch[5] */
#define B_SCH_CRC     8
/* B_GSM_TASK comes from calypso_api.h (bit mask, (1u << 1)) */
#define B_AFC         4
#define FB_DSP_TASK   5
#define SB_DSP_TASK   6
#define BITS_PER_TDMA 1250   /* tpu.h: a TDMA frame is 8 x 156.25 = 1250 bit periods */

/* firmware thresholds (prim_fbsb.c: SNR not gating, #else FB*_SNR_THRESH=0) */
#define THRESH1       (11000 - 1000)
#define THRESH2       (1000 - 200)
#define AFC_RETRY_MAX 30
#define FB0_RETRY_MAX 3
/* sync.h: ANGLE_TO_FREQ(a) = a * BITFREQ_DIV_PI / ANG2FREQ_SCALING */
#ifndef BITFREQ_DIV_PI
#define BITFREQ_DIV_PI   86208   /* sync.h:203: 270kHz/pi */
#endif
#ifndef ANG2FREQ_SCALING
#define ANG2FREQ_SCALING (2<<15) /* sync.h:204: fx1.15 */
#endif
#define ANGLE2FREQ(a) ((int)(int16_t)(a) * BITFREQ_DIV_PI / ANG2FREQ_SCALING)

/* ---- replayed ARM state ------------------------------------------------ */
typedef void (*cb_t)(int attempt);
struct item { int frame; cb_t cb; int attempt; };

static uint16_t *api;
static C54xState *dsp;
static unsigned w_page, r_page, r_page_used;
static uint32_t fn_cur;              /* l1s.current_time.fn */
static unsigned long histo_pc[64];
static unsigned long insn_total;
static unsigned char *vu_sb, *vu_hors;
static int16_t g_livre_iq[2*256]; static int g_livre_niq;
static char g_livre_type; static uint32_t g_livre_fn; static int g_livre_n;
static int      afc_dac = -700;      /* afc_initial_dac_value (compal) */
static int      fb_mode, afc_retries, fb0_retries;
static struct   item sched[32]; static int n_sched;
static struct   { int toa, pm, angle, snr, freq_diff, attempt; uint32_t fnr; } fb;
static int      verdict;             /* 0 running, 1 SB OK, -1 gave up */
static uint32_t sb_word; static int sb_bsic; static uint32_t sb_fn;
static int      n_fb_ok, n_sb_try, n_sb_crcfail, n_crc_ok, n_sb_vraies;
static int      n_err_dsp, n_err8;
```

```
static long     n_cmps_sb, n_cmps_hors, n_e0_sb, n_sttrn_sb;
static long     n_e0x_sb[4];
static int      n_firs_vus;
static long     n_cmps_reg[64];
static long     n_xc_reg[64];
static int      n_xc_vus;
static int      n_bal;
static int      n_sat_vus;
static int      n_ecr;
static int      n_2a0;
static int      g_bsic_injecte;   /* BSIC actually transmitted by the cell */
static int      trace;
static unsigned long hit_7c31, hit_9841, hit_84a1, hit_770a, hit_b219, hit_7a16;
static unsigned long hit_8478, hit_8492, hit_8493, hit_8497;
static unsigned long hit_7d1c, hit_7d1d, hit_7d1e, hit_81e4;
/* [2026-09-19] SB PATH OPCODE INVENTORY. The fault is confined to the window
 * 0x84a1..0x8497, but which instruction is mis-emulated is unknown. Record the
 * distinct opcodes executed inside the SB demod with their pass count and one
 * witness PC: that gives a finite list to audit against SPRU172C instead of an
 * intuition. REJEU_OPCODES=1. */
/* Environment flags read ONCE in rejouer(): calypso_getenv() inside the per-instruction
 * loop was a linear scan of environ per emulated instruction. */
static int env_div, env_softs_continu, env_firs, env_probe_t, env_decodeur;
static uint16_t g_ad_avant;
static uint32_t g_vie_fn; static uint16_t g_vie_ad;
static int g_dans_sb;            /* true between 0x7c31 (demod) and 0x9841 (decoder) */
static unsigned long g_op_n[65536];
static unsigned long g_op_dec[65536];   /* DECODER opcodes, region 0x9800-0x9bff */
static unsigned long g_op_eq[65536];    /* EQUALIZER opcodes, region 0x8400-0x84ff */
static unsigned long g_n_dec;           /* number of decodes (passes at 0x9841) */
static uint16_t      g_op_pc[65536];
static unsigned long g_op_tous[65536];   /* every executed opcode word, both paths */

/* [2026-09-18] REAL SCH BURSTS. Until now the DSP only ever saw our own
 * fixture, a synthetic GMSK SCH at 1 sample/symbol, so "is the ROM sound, or is
 * the fixture too poor for a coherent equalizer?" stayed open. Inject SCH bursts
 * extracted from a REAL capture (ptrkrysik/test_data, ARFCN 725, USRP decimation
 * 174 -> 574712.6 Hz) whose answer is known: BSIC=32, Viterbi decode with 0
 * errors, valid CRC. REJEU_SCH_REEL=<file>. The framing is unchanged -- same
 * margins, same timing -- ONLY the burst content differs. */
static int16_t (*g_reels)[296];
static uint8_t (*g_reels_code)[78];      /* the 78 bits actually transmitted */
static uint32_t *g_reels_fn; static uint16_t *g_reels_bsic;
static unsigned g_n_reels, g_reel_i;
static int g_reel_pour_fn[64];           /* which frame received which burst */
static int reels_charger(const char *chemin)
{
    FILE *f = fopen(chemin, "rb");
    if (!f) { fprintf(stderr, "SCH reels : %s illisible\n", chemin); return -1; }
    uint32_t n = 0, nsym = 0, ncode = 0;
    if (fread(&n, 4, 1, f) != 1 || fread(&nsym, 4, 1, f) != 1 ||
        fread(&ncode, 4, 1, f) != 1 || nsym != 148 || ncode != 78 || !n) {
        fprintf(stderr, "SCH reels : entete invalide\n"); fclose(f); return -1; }
    g_reels = calloc(n, sizeof *g_reels);
    g_reels_code = calloc(n, sizeof *g_reels_code);
    g_reels_fn = calloc(n, sizeof *g_reels_fn);
    g_reels_bsic = calloc(n, sizeof *g_reels_bsic);
    if (!g_reels || !g_reels_code || !g_reels_fn || !g_reels_bsic) { fclose(f); return -1; }
    for (uint32_t i = 0; i < n; i++) {
        uint32_t fn; uint16_t bsic; uint8_t pad[2];
        if (fread(&fn,4,1,f)!=1 || fread(&bsic,2,1,f)!=1 || fread(pad,1,2,f)!=2 ||
            fread(g_reels[i], sizeof(int16_t), 296, f) != 296 ||
            fread(g_reels_code[i], 1, 78, f) != 78) { n = i; break; }
        g_reels_fn[i] = fn; g_reels_bsic[i] = bsic;
    }
    fclose(f); g_n_reels = n;
```

```
    for (int i = 0; i < 64; i++) g_reel_pour_fn[i] = -1;
    printf("SCH reels : %u bursts charges depuis %s (BSIC attendu %u)\n",
           n, chemin, n ? g_reels_bsic[0] : 0);
    return n ? 0 : -1;
}
/* Read PROGRAM space the way the core does: with OVLY set, DARAM 0x0060..0x27FF
 * is aliased into program space, so reading dsp->prog[] directly misses the
 * alias -- exactly the trap the FIRS probe must avoid. */
static uint16_t prog_ovly(C54xState *d, uint16_t a)
{
    if ((d->pmst & (1u << 5)) && a >= 0x0060 && a < 0x2800) return d->data[a];
    return d->prog[a];
}

static void plan(int delay, cb_t cb, int attempt)
{
    if (n_sched >= 32) return;
    sched[n_sched].frame = (int)fn_cur + delay;
    sched[n_sched].cb = cb; sched[n_sched].attempt = attempt; n_sched++;
}
static void sched_reset(void) { n_sched = 0; }

static uint16_t *dbw(void) { return &api[W_PAGE(w_page)]; }
static uint16_t *dbr(void) { return &api[R_PAGE(r_page)]; }

/* ---- callbacks: faithful copy of prim_fbsb.c ---------------------------- */
static void fbdet_cmd(int unused);
static void fbdet_resp(int attempt);
static void sbdet_cmd(int attempt);

/* calypso_getenv("X") alone is TRUE even for X=0, so a line written
 * `REJEU_SCH_PARTOUT=0 REJEU_SB_FORCE=0` enabled every hack instead of cutting
 * them. Here "0", "", "non", "no" and "off" are false. */
static int drapeau_env(const char *nom)
{
    const char *e = calypso_getenv(nom);
    if (!e || !*e) return 0;
    if (e[0] == '0' && e[1] == 0) return 0;
    if (!strcmp(e, "non") || !strcmp(e, "no") || !strcmp(e, "off")) return 0;
    return 1;
}

static void plan_sb_annule(void) { }

static void sbdet_resp(int attempt);

static void plan_fb_set(int delay, int mode)
{
    fb_mode = mode;
    plan(delay + 0, fbdet_cmd, 0);
    for (int a = 1; a <= 12; a++) plan(delay + 1 + a, fbdet_resp, a);
}
static int sb_uniq(void)
{
    static int u = -1;
    if (u < 0) u = drapeau_env("REJEU_SB_UNIQUE") ? 1 : 0;
    return u;
}
/* Last scheduled SB attempt: that one must restart an acquisition. Hardcoding
 * it to "attempt == 2" leaves the bench silent under a single command, where
 * that attempt does not exist. */
static int sb_dernier_essai(void) { return sb_uniq() ? 1 : 2; }

static void plan_sb_set(int delay)
{
/* [2026-09-18] The DSP demodulates the burst of the frame FOLLOWING the command
```

```
     * (measured by perturbation: command at fn=11 -> only a perturbation at fn=12
     * changes the softs). Posting on TWO consecutive frames therefore consumes f and
     * f+1; SCH frames are 10 apart, so the second attempt always reads a DUMMY
     * burst, which is half the measurements taken on a constant input.
     * REJEU_SB_UNIQUE=1 posts a single command so every attempt sees a real SCH. */
    plan(delay + 0, sbdet_cmd, 1);
    if (!sb_uniq()) plan(delay + 1, sbdet_cmd, 2);
    plan(delay + 3, sbdet_resp, 1);
    if (!sb_uniq()) plan(delay + 4, sbdet_resp, 2);
}

static uint32_t g_sb_cmd_fn;   /* frame of the LAST SB command = the one the DSP demodulates */
static uint32_t fb_cmd_fn;      /* frame on which the ARM posted the FB task */
static void fbdet_cmd(int unused)
{
    (void)unused;
    fb_cmd_fn = fn_cur;
    dbw()[W_TASK_MD]  = FB_DSP_TASK;
    api[NDB_FB_MODE]  = (uint16_t)fb_mode;
}

static void fbdet_resp(int attempt)
{
    if (!api[NDB_FB_DET]) {
        if (attempt < 12) return;
        sched_reset();
        if (fb0_retries < FB0_RETRY_MAX) { fb0_retries++; plan_fb_set(1, 0); }
        else verdict = -1;
        return;
    }
    /* read_fb_result */
    fb.toa   = (int16_t)api[NDB_SYNC + 0];
    fb.pm    = (int16_t)api[NDB_SYNC + 1] >> 3;
    fb.angle = (int16_t)api[NDB_SYNC + 2];
    fb.snr   = (int16_t)api[NDB_SYNC + 3];
    fb.freq_diff = ANGLE2FREQ(fb.angle);
    fb.fnr = fn_cur; fb.attempt = attempt;
    api[NDB_FB_DET] = 0; api[NDB_SYNC + 0] = 0;
    n_fb_ok++;
    if (trace) {
        /* real FCCH = first frame >= cmd with p51 in {0,10,20,30,40} */
        uint32_t f = fb_cmd_fn; while ((f % 51) % 10 != 0 || (f % 51) > 40) f++;
        int dframe = (int)(f - fb_cmd_fn);                /* FCCH frame inside the window */
        int toa_attendu = dframe * BITS_PER_TDMA + 23;      /* what the firmware can read */
        int ntdma_lu = (fb.toa - 23) / BITS_PER_TDMA;
        printf("  FB%d att=%d fn=%u TOA=%d (df=%d) | cmd=%u FCCH reelle=%u (+%d trames) "
               "=> TOA attendu ~%d, ntdma lu=%d au lieu de %d\n",
               fb_mode, attempt, fn_cur, fb.toa, fb.freq_diff,
               fb_cmd_fn, f, dframe, toa_attendu, ntdma_lu, dframe);
    }
    /* afc_correct: delta = (norm * err)/slope; slope compal_e88 = 287 */
    afc_dac += (int)(((32768 / 947) * (long)fb.freq_diff) / 287);
    if (afc_dac > 4095) afc_dac = 4095;
    if (afc_dac < -4096) afc_dac = -4096;
    sched_reset();
    if (fb_mode == 0) {
        if (abs(fb.freq_diff) < THRESH1) plan_fb_set(1, 1);
        else if (afc_retries < AFC_RETRY_MAX) { afc_retries++; plan_fb_set(1, 0); }
        else verdict = -1;
    } else {
        int toa = fb.toa - 23, ntdma, qbits;
        if (toa < 0) { qbits = (toa + BITS_PER_TDMA) * 4; ntdma = -1; }
        else { ntdma = toa / BITS_PER_TDMA; qbits = (toa - ntdma * BITS_PER_TDMA) * 4; }
        int fn_offset = (int)fn_cur - attempt + ntdma;
        int delay = fn_offset + 11 - (int)fn_cur - 1;
        if (trace) printf("    -> toa-23=%d ntdma=%d qbits=%d delay=%d\n", toa, ntdma, qbits, delay);
```

```
        if (abs(fb.freq_diff) < THRESH2) {
            if (delay < 0) delay = 0;
            if (delay > 20) delay = 20;
            /* REJEU_SB_FORCE=1: aim at the NEXT SCH frame (p51 in {1,11,21,31,41})
             * instead of the computed delay. Separates the DEMODULATOR from the
             * TIMING: a CRC passing here means only the timing (ntdma) is at
             * fault; a failure means the SB demodulation really is wrong. */
            static int force = -1;
            if (force < 0) force = drapeau_env("REJEU_SB_FORCE") ? 1 : 0;
            if (force) {
                uint32_t f = fn_cur + 1;
                while (!((f % 51) % 10 == 1 && (f % 51) <= 41)) f++;
                delay = (int)(f - fn_cur);
                /* [2026-09-18] THE CONSUMED FRAME IS NOT THE FIRST COMMAND.
                 * plan_sb_set posts the task on TWO frames (delay+0 and delay+1) and the
                 * DSP demodulates the SECOND: measured by perturbing one raw sample,
                 * frame by frame, over fn 9..15 -- only fn=12 changes the soft bits read
                 * at fn=14/15, while [force] aimed at fn=11. The SCH was thus placed on
                 * frame 11 while the DSP demodulated frame 12, which carries a DUMMY
                 * burst (fixed pattern, 45.002 5.2.6), hence a CONSTANT input: frozen
                 * softs, identical range from frame to frame, and CRC OKs all returning
                 * the same word tens of frames apart. Stepping back one frame aligns the
                 * CONSUMED frame with the SCH. REJEU_SB_DECALAGE=<n> (default -1). */
                { static int d = -2; if (d == -2) { const char *e = calypso_getenv("REJEU_SB_DECALAGE");
                                                    d = e ? atoi(e) : -1; }
                  delay += d; if (delay < 0) delay = 0; }
                if (trace) printf("    [force] SB vise fn=%u (p51=%u), delay=%d\n", f, f % 51, delay);
            }
            plan_sb_set(delay);
        } else plan_fb_set(1, 1);
    }
}

static void sbdet_cmd(int attempt)
{
    g_sb_cmd_fn = fn_cur;
    if (trace) { unsigned p = fn_cur % 51;
        printf("  SBcmd att=%d fn=%u p51=%u %s\n", attempt, fn_cur, p,
               (p % 10 == 1 && p <= 41) ? "<- trame SCH (bon)" : "<- PAS une trame SCH"); }
    dbw()[W_TASK_MD] = SB_DSP_TASK;
    api[NDB_FB_MODE] = 0;
    if (trace) printf("  [SBcmd W] page=%u W=%04x %04x %04x %04x %04x ..%04x %04x  NDB page=%04x fb_mode=%04x
     ↪  fb_det=%04x\n",
                      w_page, dbw()[0], dbw()[1], dbw()[2], dbw()[3], dbw()[4], dbw()[15], dbw()[16],
 ↪  api[NDB_PAGE], api[NDB_FB_MODE], api[NDB_FB_DET]);
}

static void sbdet_resp(int attempt)
{
    n_sb_try++;
    r_page_used = 1;
    if (trace) {
        /* SB demodulator internals (RE 9.4 and 9.19):
         * 0x2f06 = correlation peak index
         * 0x2bf8 = internal CRC flag
         * 0x2c72 = THE 78 SOFT BITS. 0x2a00 is the 296-word FB correlator buffer, so
         * reading it there yields only 0x0000 and 0xffff; the real softs at 0x2c72
         * have varied magnitudes. Their range is printed too. */
        {
            int16_t mn = 32767, mx = -32768, nnul = 0;
            for (int k = 0; k < 78; k++) {
                int16_t v = (int16_t)dsp->data[0x2c72 + k];
                if (v) nnul++;
                if (v < mn) mn = v;
                if (v > mx) mx = v;
            }
```

```
/* THE FEED. Does the buffer the DSP demodulates really hold the
 * delivered samples, and at which shift? Compare the BSP drop word by word
 * with the last delivered burst and look for the best alignment: a maximum
 * away from 0 means the burst is shifted; low everywhere means it is not
 * our burst at all. */
if (calypso_getenv("REJEU_FEED")) {
    uint16_t ad = calypso_bsp_get_daram_addr();
    int nl = g_livre_niq;
    int best = 0, bestn = -1, n0 = 0;
    for (int sh = -80; sh <= 80; sh++) {
        int m = 0;
        for (int k = 0; k < nl; k++) {
            int j = k + sh; if (j < 0 || j >= 2048) continue;
            if ((int16_t)dsp->data[(ad + j) & 0x3fff] == g_livre_iq[k]) m++;
        }
        if (sh == 0) n0 = m;
        if (m > bestn) { bestn = m; best = sh; }
    }
    printf("    [feed] fn=%u (livre fn=%u type=%c) addr=0x%04x len=%u nl=%d"
           "  identiques a shift0=%d/%d  meilleur shift=%d avec %d/%d\n",
           fn_cur, g_livre_fn, g_livre_type ? g_livre_type : '?', ad,
           calypso_bsp_get_daram_len(), nl, n0, nl, best, bestn, nl);
}
/* [2026-09-18] PER-POSITION ERROR PROFILE. An aggregate percentage hides
 * what discriminates: WHERE the errors fall. The 78 coded bits occupy
 * symbols [3..41] and [106..144], the midamble sits at [42..105]. Errors at
 * the EDGES = channel estimate right at the midamble and drifting away from
 * it (phase ramp / too short an estimate). Uniform errors = wrong equalizer.
 * ~0 errors under one polarity = good softs and a bug downstream. Compared
 * against the frame ACTUALLY demodulated (last SB command), not the current
 * frame. */
/* [2026-09-19] FIRS reads its input at 0x2a8e..0x2a9a (zone 0x2a00) while
 * the BSP drops the burst at 0x0cce. Is that intermediate zone filled, and
 * does it carry our burst? */
if (calypso_getenv("REJEU_ZONE2A")) {
    uint16_t ad = calypso_bsp_get_daram_addr();
    int nz0 = 0, nz2 = 0;
    for (int k = 0; k < 296; k++) {
        if (dsp->data[(ad + k) & 0x3fff]) nz0++;
        if (dsp->data[(0x2a00 + k) & 0x3fff]) nz2++;
    }
    printf("    [zone] depot 0x%04x : %d/296 non nuls | zone 0x2a00 : %d/296 non nuls\n",
           ad, nz0, nz2);
    printf("      0x%04x : %04x %04x %04x %04x %04x %04x\n", ad,
           dsp->data[ad&0x3fff], dsp->data[(ad+1)&0x3fff], dsp->data[(ad+2)&0x3fff],
           dsp->data[(ad+3)&0x3fff], dsp->data[(ad+4)&0x3fff], dsp->data[(ad+5)&0x3fff]);
    printf("      0x2a8e : %04x %04x %04x %04x %04x %04x   (entree lue par FIRS)\n",
           dsp->data[0x2a8e], dsp->data[0x2a8f], dsp->data[0x2a90],
           dsp->data[0x2a91], dsp->data[0x2a92], dsp->data[0x2a93]);
}
if (calypso_getenv("REJEU_PROFIL")) {
    unsigned char att78[78];
    int ri = g_n_reels ? g_reel_pour_fn[g_sb_cmd_fn & 63] : -1;
    if (ri >= 0) memcpy(att78, g_reels_code[ri], 78);   /* real ground truth */
    else cellule_code_attendu(g_sb_cmd_fn, (uint8_t)g_bsic_injecte, att78);
    int acc[2] = {0, 0};
    char map[2][79];
    for (int pol = 0; pol < 2; pol++) {
        for (int k = 0; k < 78; k++) {
            int16_t v = (int16_t)dsp->data[0x2c72 + k];
            int bit = pol ? (v > 0) : (v < 0);   /* both sign conventions */
            int ok = (bit == (att78[k] & 1));
            map[pol][k] = ok ? '.' : 'X';
            acc[pol] += ok;
        }
        map[pol][78] = 0;
```

```
            }
            /* [2026-09-18] ALIGNMENT SWEEP. The peak moves from frame to frame
             * (23, 20, 19...): if it places the read window, comparing at the
             * canonical alignment compares good softs against a wrong framing, and
             * "uncorrelated" would be a false conclusion. Rebuild the full expected
             * burst (3 tail + 39 + 64 TSC + 39 + 3 tail) and find the shift s that
             * maximizes agreement. An s reaching ~78/78 means the softs are GOOD and
             * only the framing is wrong. */
            {
                unsigned char burst[148];
                memset(burst, 0, 3);
                for (int k = 0; k < 39; k++) burst[3 + k] = att78[k];
                for (int k = 0; k < 64; k++) burst[42 + k] = (unsigned char)cellule_train_sb(k);
                for (int k = 0; k < 39; k++) burst[106 + k] = att78[39 + k];
                memset(burst + 145, 0, 3);
                int bs = 0, bp = 0, bv = -1;
                for (int sh = -40; sh <= 40; sh++) {
                    for (int pol = 0; pol < 2; pol++) {
                        int m = 0, n = 0;
                        for (int k = 0; k < 78; k++) {
                            int pos = (k < 39 ? 3 + k : 106 + (k - 39)) + sh;
                            if (pos < 0 || pos >= 148) continue;
                            int16_t v = (int16_t)dsp->data[0x2c72 + k];
                            int bit = pol ? (v > 0) : (v < 0);
                            if (bit == (burst[pos] & 1)) m++;
                            n++;
                        }
                        if (n >= 60 && m > bv) { bv = m; bs = sh; bp = pol; }
                    }
                }
                printf("    [align] meilleur decalage=%+d polarite=%d -> %d/78\n", bs, bp, bv);
            }
            int best = acc[1] > acc[0] ? 1 : 0;
            printf("    [profil] fn_demod=%u (p51=%u) pic=%u : concordance pol0=%d/78 pol1=%d/78\n"
                   "      premiers39 |%.39s|\n"
                   "      derniers39 |%s|\n",
                   g_sb_cmd_fn, g_sb_cmd_fn % 51, dsp->data[0x2f06],
                   acc[0], acc[1], map[best], map[best] + 39);
        }
        if (calypso_getenv("REJEU_DUMP_SOUPLES")) {
            printf("    [souples] fn=%u pic=%u :", fn_cur, dsp->data[0x2f06]);
            for (int k = 0; k < 78; k++) printf(" %04x", dsp->data[0x2c72 + k]);
            printf("\n");
        }
        printf("    [sb-int] pic(2f06)=%u  crc(2bf8)=%u  souples(2c72)[0..5]=%04x %04x %04x %04x %04x %04x"
               "  non nuls=%d/78  etendue=[%d..%d]\n",
               dsp->data[0x2f06], dsp->data[0x2bf8],
               dsp->data[0x2c72], dsp->data[0x2c73], dsp->data[0x2c74],
               dsp->data[0x2c75], dsp->data[0x2c76], dsp->data[0x2c77], nnul, mn, mx);
    }
}
if (trace) { unsigned p = fn_cur % 51;
    /* [2026-09-30] mot decode MEME en CRC faux, et les mesures de la ROM sur le
     * burst (a_serv_demod : TOA, PM, ANGLE, SNR) : c'est la matiere pour
     * comparer les SCH decodes aux rates (voir MAILBOX 2026-09-30). */
    printf("  SBresp att=%d fn=%u p51=%u crc=%s a_sch=%04x %04x %04x %04x %04x toa=%d pm=%d angle=%d snr=%d
↪ sb_cmd_fn=%u\n", attempt, fn_cur, p,
           (dbr()[R_SCH + 0] & (1 << B_SCH_CRC)) ? "FAUX" : "OK",
           dbr()[R_SCH + 0], dbr()[R_SCH + 1], dbr()[R_SCH + 2], dbr()[R_SCH + 3], dbr()[R_SCH + 4],
           (int16_t)dbr()[R_SERV + 0], (int16_t)dbr()[R_SERV + 1], (int16_t)dbr()[R_SERV + 2],
           ↪  (int16_t)dbr()[R_SERV + 3], g_sb_cmd_fn); }
if (dbr()[R_SCH + 0] & (1 << B_SCH_CRC)) {
    n_sb_crcfail++;
    if (attempt == sb_dernier_essai()) { sched_reset(); plan_fb_set(1, 0); }
    return;
}
```

```
    sb_word = dbr()[R_SCH + 3] | ((uint32_t)dbr()[R_SCH + 4] << 16);
    sb_bsic = (sb_word >> 2) & 0x3f;
    unsigned t1 = ((sb_word >> 23) & 1) | ((sb_word >> 7) & 0x1fe) | ((sb_word << 9) & 0x600);
    unsigned t2 = (sb_word >> 18) & 0x1f;
    unsigned t3p = ((sb_word >> 24) & 1) | ((sb_word >> 15) & 6);
    unsigned t3 = t3p * 10 + 1;
    sb_fn = 51u * ((t3 - t2 + 26u) % 26u) + t3 + 26u * 51u * t1;
    /* A CRC OK is a decode only if it yields the INJECTED BSIC, a valid T3 (T3 <= 50
     * by construction) and a reconstructed FN equal to the current frame. Each pass
     * is qualified and the run continues instead of stopping on the first.
     * REJEU_ARRET_1ER=1 restores stopping.
     *
     * REJEU_SCH_PARTOUT=1 CORRUPTS THE REFERENCE TRUTH: T3' has three bits, so
     * a SCH emitted on p51=45 encodes T3'=4, which the decoder turns back into
     * T3=41 and sb_fn != fn_cur even for a PERFECT decode. FN can only be
     * validated without that flag. */
    int t3_ok = (t3 <= 50), bsic_ok = (sb_bsic == (unsigned)g_bsic_injecte);
    /* [2026-09-20] The SB word carries the frame number of the BURST that was
     * demodulated, i.e. the frame of the last SB command (g_sb_cmd_fn), not the
     * frame on which the ARM reads the result (fn_cur, 2-3 frames later). The
     * first genuine decodes (BSIC 7, FN 31 read at fn 34; FN 62 read at fn 64)
     * were being counted as false positives by the old fn_cur comparison. */
    int fn_ok = (sb_fn == g_sb_cmd_fn);
    n_crc_ok++;
    if (bsic_ok && t3_ok && fn_ok) n_sb_vraies++;
    printf("  SB%d fn=%u : sb=0x%08x BSIC=%d (injecte %d) T1=%u T2=%u T3=%u -> FN=%u (burst demodule fn=%u)
↪   %s%s\n",
           attempt, fn_cur, sb_word, sb_bsic, g_bsic_injecte, t1, t2, t3, sb_fn, g_sb_cmd_fn,
           (bsic_ok && t3_ok && fn_ok) ? "** VRAIE **"
           : !bsic_ok ? "FAUX POSITIF (BSIC ne colle pas)"
           : !t3_ok   ? "FAUX POSITIF (T3 > 50, impossible)"
           :            "FAUX POSITIF (FN != trame du burst demodule)",
           cellule_sch_partout ? "  [FN non qualifiable sous SCH_PARTOUT]" : "");
    if (drapeau_env("REJEU_ARRET_1ER") || ((bsic_ok && t3_ok && fn_ok) && !drapeau_env("REJEU_CONTINUER"))) {
     ↪  verdict = 1; return; }
    if (bsic_ok && t3_ok && fn_ok) { sched_reset(); plan_fb_set(1, 0); return; }   /* REJEU_CONTINUER: restart the
     ↪  acquisition */
    /* [2026-09-18] An unqualified CRC OK (false positive) must reschedule, or the
     * bench freezes and the freeze reads as a result: the item queue drains, no
     * command is posted again, and the replay crosses the remaining thousands of
     * frames asking the DSP nothing. The summary was then IDENTICAL at 1200 and at
     * 12000 frames -- not because the DSP stalled, but because the replayed ARM had
     * gone silent. The CRC failure branch already restarted (plan_fb_set on attempt
     * 2); only the "CRC OK but false positive" exit was a dead end. */
    sched_reset(); plan_fb_set(1, 0);
}


/* ---- ARM-side DSP init: what the firmware's dsp_power_on() does ---------
 * Without it the DSP runs on a blank API RAM: the FB detection thresholds
 * (d_fb_thr_det_iacq/track) and the margins (d_fb_margin_beg/end, which feed the
 * TOA formula at 0x794b/0x7956) are 0, and FB detection diverges from the first
 * frame. Values from dsp_params.c / dsp_ndb_init(). */
#define PARAM   0x431u          /* BASE_API_PARAM (API 0x862) in words */
#define A_SCH26 (NDB + 42u)
/* DSP bootloader: API offsets in WORDS (dsp.c: BASE_API_RAM + 0x0ff8..0x0ffe) */
#define BL_ADDR_HI_W  0x7FCu
#define BL_SIZE_W     0x7FDu
#define BL_ADDR_LO_W  0x7FEu
#define BL_STATUS_W   0x7FFu
#define DSP_START     0x7000u

static long pump(long max, int stop_on_idle)
{
    long b = 0;
    while (b < max && dsp->running) {
```

```
        int ex = c54x_run(dsp, 256);
        if (ex <= 0) break;
        b += ex;
        if (stop_on_idle && dsp->idle) break;
    }
    return b;
}

static void arm_dsp_init(void)
{
    memset(api, 0, 0x2000u * sizeof(uint16_t));          /* dsp_api_memset(API) */

    /* dsp_pre_boot(): the DSP has just been reset; wait for BL_STATUS_IDLE */
    long b = 0;
    while (api[BL_STATUS_W] != 1 && b < 8000000) {
        int ex = c54x_run(dsp, 256); if (ex <= 0) break; b += ex;
    }
    printf("  bootloader DSP : BL_STATUS=%u apres %ld insn\n", api[BL_STATUS_W], b);

    /* dsp_set_params(): NDB first */
    api[NDB + 8]  = 0x0074;  api[NDB + 9]  = 0x0001;
    api[NDB + 10] = 0x0154;  api[NDB + 11] = 0x17ff;
    api[NDB + 12] = 7;       api[NDB + 13] = 0;
    api[NDB + 14] = 3;                                   /* d_dsp_state = IDLE3 */
    /* then the parameter table (dsp_params.c). The first four FB fields feed the
     * TOA formula (0x794b/0x7956) and the detection thresholds. */
    static const int16_t P[] = {
        0x6666, 15, 12, 5, 4, 0x7002, 1, 0xE, 0, 0, 0, 0,
        24, 22, 296, 30,                                 /* margin_beg/end, nsubb_idle/dedic */
        0x3333, (int16_t)0x28f6,                         /* fb_thr_det_iacq / _track */
        0x7fff, 17408, 26624, 20152,
        7872, -4, 7872, 5772, 7872, 53, -892, 208,
    };
    for (unsigned i = 0; i < sizeof P / sizeof P[0]; i++) api[PARAM + i] = (uint16_t)P[i];

    /* dsp_bl_start_at(DSP_START) */
    api[BL_ADDR_HI_W] = 0; api[BL_ADDR_LO_W] = DSP_START; api[BL_SIZE_W] = 0;
    api[BL_STATUS_W]  = 2;                               /* BL_CMD_COPY_BLOCK */
    b = pump(4000000, 1);
    printf("  DSP demarre : %ld insn, idle=%d, version=0x%04x%04x\n",
           b, dsp->idle, api[NDB + 6], api[NDB + 7]);

    /* dsp_ndb_init(): what matters for FB/SB */
    api[NDB + 2]  = 0x0179;                              /* d_spcx_rif */
    api[NDB + 3]  = 0x0800 | ((8 - 4) << 7);             /* d_tch_mode */
    api[NDB_FB_MODE] = 1;
    api[NDB_FB_DET]  = 0;
    api[A_SCH26]     = (1u << B_SCH_CRC);
    /* dsp_db_init() */
    memset(&api[W_PAGE(0)], 0, W_SIZE * sizeof(uint16_t));
    memset(&api[W_PAGE(1)], 0, W_SIZE * sizeof(uint16_t));
    memset(&api[R_PAGE(0)], 0, R_SIZE * sizeof(uint16_t));
    memset(&api[R_PAGE(1)], 0, R_SIZE * sizeof(uint16_t));
    /* [2026-09-20] REJEU_DUMP_PARAM=1: the first eight parameter words after
     * the DSP has started, to compare with the live bench's shared API RAM
     * (od on /dev/shm/calypso_api_ram at word 0x431). */
    if (drapeau_env("REJEU_DUMP_PARAM"))
        printf("  PARAM apres demarrage : %04x %04x %04x %04x %04x %04x %04x %04x  (attendu 6666 000f 000c 0005
↪  0004 7002 0001 000e)\n",
               api[PARAM], api[PARAM+1], api[PARAM+2], api[PARAM+3], api[PARAM+4], api[PARAM+5], api[PARAM+6],
↪  api[PARAM+7]);
}

/* ---- one frame: l1_sync() then the DSP --------------------------------- */
static void l1_sync(void)
{
```

```
    r_page_used = 0;
    memset(dbw(), 0, W_SIZE * sizeof(uint16_t));          /* memset db_w */
    dbw()[W_AFC] = (uint16_t)(int16_t)afc_dac;            /* afc_load_dsp */
    dbw()[W_CTRL_ABB] |= (1u << B_AFC);
    /* [2026-09-17] CLOSED AFC LOOP. qemu-src relays the d_afc write to the TWL3025
     * model (calypso_trx.c, offsets 0x001E/0x0046); qosmo does it NOWHERE
     * (set_afc_dac is never called there), so the loop stayed open, sample rotation
     * never moved, and the frequency error never converged below the SB threshold
     * (800 Hz). */
    calypso_twl3025_set_afc_dac((int16_t)afc_dac);
    if (api[NDB_ERRSTAT]) {                               /* as in sync.c:249 */
        static int n; if (trace && n < 6) { printf("  DSP Error Status: %u\n", api[NDB_ERRSTAT]); n++; }
        api[NDB_ERRSTAT] = 0;
    }
    /* Run the items of THIS frame. A callback may schedule a new one for the
     * CURRENT frame (delay=0, which is what the firmware computes after FB1), so
     * rescan until exhaustion or that item is silently lost -- and it is precisely
     * the SB command aiming at the right SCH frame. */
    for (int tour = 0; tour < 8; tour++) {
        int fait = 0;
        for (int i = 0; i < n_sched; i++) {
            if (sched[i].frame == (int)fn_cur && sched[i].cb) {
                cb_t cb = sched[i].cb; int at = sched[i].attempt;
                sched[i].cb = NULL;
                cb(at); fait = 1;
                if (verdict) return;
            }
        }
        if (!fait) break;
    }
    if (r_page_used) {
        memset(dbr(), 0, R_SIZE * sizeof(uint16_t));
        dbr()[R_SCH + 0] = (1u << B_SCH_CRC);
        r_page ^= 1;
    }
    api[NDB_PAGE] = (uint16_t)(B_GSM_TASK | w_page);      /* dsp_end_scenario() */
    w_page ^= 1;
}

int rejouer(C54xState *d, uint16_t *api_ram, long trames, long insns,
            const char *iq_mode, int amp, int bsic, int verbeux)
{
    dsp = d; api = api_ram; trace = verbeux;
    env_div = calypso_getenv("REJEU_DIV") != NULL;
    env_softs_continu = calypso_getenv("REJEU_SOFTS_CONTINU") != NULL;
    env_firs = calypso_getenv("REJEU_FIRS") != NULL;
    env_probe_t = drapeau_env("REJEU_PROBE_T");
    env_decodeur = calypso_getenv("REJEU_DECODEUR") != NULL;
    w_page = r_page = r_page_used = 0; fn_cur = 0; afc_dac = -700;
    fb_mode = 0; afc_retries = fb0_retries = 0; n_sched = 0; verdict = 0;
    n_fb_ok = n_sb_try = n_sb_crcfail = n_crc_ok = n_sb_vraies = 0;
    n_err_dsp = n_err8 = 0;
    g_bsic_injecte = bsic;
    { const char *r = calypso_getenv("REJEU_SCH_REEL");
      if (r && *r && reels_charger(r) == 0) g_bsic_injecte = g_reels_bsic[0]; }
    memset(&fb, 0, sizeof fb);

    if (drapeau_env("REJEU_SCH_PARTOUT")) { cellule_sch_partout = 1; printf("  [stimulus] SCH sur toutes les
     ↪  trames non-FCCH (masque le cadencage)\n"
               "  [stimulus] ATTENTION : t3p = p51/10 n'est juste que sur p51 in {1,11,21,31,41},\n"
               "              donc la FN reconstruite ne peut PAS etre validee sous ce drapeau.\n"); }
    printf("rejeu deterministe : %ld trames max, %ld insn/trame, cellule BSIC=%d, iq=%s\n",
           trames, insns, bsic, iq_mode ? iq_mode : "cell");
    /* firmware dsp_power_on(): bootloader boot + parameters + NDB */
    arm_dsp_init();
    { static int16_t rempl[2 * 148];         /* TS1..TS7 of the C0 carrier: dummy bursts */
```

```
      cellule_factice(amp, DECALAGE_SYMB, rempl);
      calypso_bsp_set_remplissage(rempl, 2 * 148); }
    plan_fb_set(1, 0);                       /* first FBSB_REQ */

    int16_t iq[2 * 256]; int n_iq;
    for (long t = 0; t < trames && !verdict; t++) {
        fn_cur = (uint32_t)t;
        g_c54x_exe_fn = fn_cur;
        uint32_t insn_debut_trame = dsp->insn_count;
        l1_sync();
        /* [2026-09-18] The firmware (sync.c) reads d_error_status every frame, prints
         * it and clears it; replay ignored it entirely, hence its silence about the
         * "DSP Error Status: 8" seen on the firmware side. 8 = DSP_ERR_DMA_PROG:
         * overflow of the DMA job ring at 0x4330 (orm *(0x3f92),#8 at 0xaa83). */
        if (api[NDB_ERRSTAT]) {
            unsigned e = api[NDB_ERRSTAT];
            n_err_dsp++;
            if (e & 8) n_err8++;
            if (n_err_dsp <= 8)
                printf("  [erreur DSP] status=%u%s a fn=%u\n", e,
                       (e & 8) ? " (bit 3 = DSP_ERR_DMA_PROG, anneau DMA 0x4330 sature)" : "",
                       fn_cur);
            api[NDB_ERRSTAT] = 0;
        }
        if (verdict) break;
        /* [2026-09-17] HARDWARE ORDER. On silicon: the ARM posts the task -> the DSP
         * reads it on the frame interrupt and ARMS its RX window (DMA) -> the samples
         * arrive -> the DSP processes them. Injecting BEFORE the DSP has armed makes
         * the transfer use the PREVIOUS task's DMA programming (the FB one), so the SB
         * receives the wrong burst. REJEU_RX_AVANT=1 restores the old order for
         * comparison. */
        static int rx_avant = -1;
        if (rx_avant < 0) rx_avant = drapeau_env("REJEU_RX_AVANT") ? 1 : 0;
        n_iq = 2 * 148;
        int injecter = (!iq_mode || strcmp(iq_mode, "none") != 0);
        if (injecter && rx_avant) {
            g_livre_type = cellule_burst(fn_cur, (uint8_t)g_bsic_injecte, amp, DECALAGE_SYMB, marge_tete(), iq,
↪   &n_iq);
            g_livre_fn = fn_cur; g_livre_n = n_iq;
            g_ad_avant = calypso_bsp_get_daram_addr();
            { static int da = -1; static unsigned nd;
              if (da < 0) da = calypso_getenv("REJEU_ADR") ? 1 : 0;
              if (da && nd < 14) { nd++;
                  printf("  [adr] fn=%-4u type=%c n_iq=%-4d -> depot 0x%04x len=%u\n",
                         fn_cur, g_livre_type ? g_livre_type : '?', n_iq,
                         calypso_bsp_get_daram_addr(), calypso_bsp_get_daram_len()); } }
            calypso_bsp_rx_burst(0, fn_cur, iq, n_iq);
        }
        /* frame interrupt: the DSP reads the task and arms its RX window.
         * [2026-09-20] REJEU_IRQ_SCENARIO=1: raise it only on frames where the
         * replayed ARM ended a DSP scenario (a task on the page just handed
         * over), as dsp_end_scenario() does with tpu_dsp_frameirq_enable(),
         * a bit the firmware sets again on every scenario. Every other frame
         * the ROM gets no frame interrupt and does not re-read the page. */
        { static int irq_sc = -1;
          if (irq_sc < 0) irq_sc = drapeau_env("REJEU_IRQ_SCENARIO") ? 1 : 0;
          unsigned wp_donnee = (api[NDB_PAGE] & 1u);
          bool scenario = api[W_PAGE(wp_donnee) + W_TASK_MD] != 0 || api[W_PAGE(wp_donnee) + 0] != 0;
          if ((dsp->imr & (1u << 12)) && (!irq_sc || scenario)) c54x_interrupt_ex(dsp, 28, 12); }
        if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
        if (injecter && !rx_avant) {
            /* let the DSP arm (short budget), THEN deliver the samples */
            long arm = 0;
            while (arm < insns / 4 && dsp->running && !dsp->idle) {
                int ex = c54x_run(dsp, 64); if (ex <= 0) break; arm += ex;
            }
```

```
            g_livre_type = cellule_burst(fn_cur, (uint8_t)g_bsic_injecte, amp, DECALAGE_SYMB, marge_tete(), iq,
↪  &n_iq);
            /* [2026-09-20] FULL FRAME for the FB search. On silicon the FB task
             * streams 156.25 symbols per TDMA frame (148 burst + 8.25 guard); the
             * TOA it reports counts frames in those units and the firmware turns
             * it back into frames with BITS_PER_TDMA = 1250. Delivering 148
             * samples per frame made a 9-frame distance read as 5. Pad every
             * non-SCH burst with silence to 156 samples (157 on one frame in four,
             * so the average is 156.25). REJEU_TRAME_PLEINE=0 restores 148. */
            /* The SCH frame is delivered as the 190-sample WINDOW block (margins
             * 21/21) only while the DSP has its one-shot SB window armed; inside
             * an FB search (continuous DMA) it is a plain frame of the stream like
             * any other, or the stream would gain 34 symbols on every SCH frame.
             * The TOA origin (the firmware's "23") is not set here but by the DMA
             * model at arm time (CALYPSO_RHEA_DMA_ARM_SKIP): the burst position
             * inside the frame can only move within the 8.25 idle symbols. */
            { static int pleine = -1;
              if (pleine < 0) { const char *e = calypso_getenv("REJEU_TRAME_PLEINE"); pleine = (e && *e == '0') ? 0
↪  : 1; }
              bool fenetre_sb = calypso_rhea_dma_one_shot();
              if (pleine && !(g_livre_type == 'S' && fenetre_sb)) {
                  if (g_livre_type == 'S') {          /* drop the window margins: burst only */
                      int m = marge_tete();
                      memmove(iq, iq + 2 * m, 2 * 148 * sizeof(int16_t));
                      n_iq = 2 * 148;
                  }
                  int cible = 156 + ((fn_cur & 3) == 3 ? 1 : 0);
                  if (n_iq < 2 * cible) { memset(iq + n_iq, 0, (size_t)(2 * cible - n_iq) * sizeof(int16_t)); n_iq
↪  = 2 * cible; }
              } }
            g_livre_fn = fn_cur; g_livre_n = n_iq;
            /* Replace the SCH CONTENT with a real burst, without touching the framing. */
            if (g_n_reels && g_livre_type == 'S') {
                int m = marge_tete();
                unsigned r = g_reel_i % g_n_reels;
                { static double gn = -1;
                  if (gn < 0) { const char *e = calypso_getenv("REJEU_REEL_GAIN"); gn = e ? atof(e) : 1.0;
                                if (gn <= 0) gn = 1.0; }
                  if (gn == 1.0) memcpy(iq + 2 * m, g_reels[r], 296 * sizeof(int16_t));
                  else for (int q = 0; q < 296; q++) {
                      long v = lrint(g_reels[r][q] * gn);
                      iq[2 * m + q] = (int16_t)(v > 32767 ? 32767 : v < -32768 ? -32768 : v); } }
                /* Check: is the burst dropped really the REAL one? */
                static unsigned nv;
                if (nv < 4) { nv++;
                    printf("  [reel] trame %u <- burst #%u (fn reelle %u, BSIC %u) I/Q %d %d %d %d\n",
                           fn_cur, r, g_reels_fn[r], g_reels_bsic[r],
                           iq[2*m], iq[2*m+1], iq[2*m+2], iq[2*m+3]); }
                g_reel_pour_fn[fn_cur & 63] = (int)r;
                g_reel_i++;
            }
            /* [2026-09-18] Perturb ONE raw sample rather than a bit: this maps
             * influence index by index, with no confusion from burst packing nor from
             * the scheduling change a coded-bit flip causes. REJEU_PERTURBER_ECH=<n>
             * adds a delta to complex sample n of the delivered buffer. */
            { static int pe = -2; static long pf = -2;
              if (pe == -2) { const char *e = calypso_getenv("REJEU_PERTURBER_ECH"); pe = e ? atoi(e) : -1; }
              if (pf == -2) { const char *e = calypso_getenv("REJEU_PERTURBER_FN");  pf = e ? atol(e) : -1; }
              if (pe >= 0 && 2 * pe + 1 < n_iq && (pf < 0 || (long)fn_cur == pf)) {
                  iq[2 * pe]     = (int16_t)(iq[2 * pe]     + 3000);
                  iq[2 * pe + 1] = (int16_t)(iq[2 * pe + 1] - 3000);
              } }
            /* [2026-09-18] DELIVERED SAMPLE CONVENTION. Transport is exact ([feed]
             * probe: 380/380 identical at shift 0), so if the feed is at fault it is the
             * CONVENTION, not the routing. Four mutually exclusive hypotheses:
             *   derot-  : the DSP expects a signal DEROTATED by -pi/2 per symbol
```

```
             *   derot+  : ... by +pi/2
             *   swap    : I and Q swapped
             *   conj    : Q negated (conjugate)
             * REJEU_FEED_XFORM=derot-|derot+|swap|conj|none (default none). */
            { static const char *xf = NULL; static int init = 0;
              if (!init) { xf = calypso_getenv("REJEU_FEED_XFORM"); init = 1; }
              if (xf && *xf && strcmp(xf, "none")) {
                  int ns = n_iq / 2;
                  if (!strcmp(xf, "swap")) {
                      for (int k = 0; k < ns; k++) { int16_t t = iq[2*k]; iq[2*k] = iq[2*k+1]; iq[2*k+1] = t; }
                  } else if (!strcmp(xf, "conj")) {
                      for (int k = 0; k < ns; k++) iq[2*k+1] = (int16_t)(-iq[2*k+1]);
                  } else if (!strcmp(xf, "derot-") || !strcmp(xf, "derot+")) {
                      double sgn = (xf[5] == '-') ? -1.0 : 1.0;
                      for (int k = 0; k < ns; k++) {
                          double ph = sgn * (M_PI / 2.0) * k;
                          double c = cos(ph), sn = sin(ph);
                          double i0 = iq[2*k], q0 = iq[2*k+1];
                          iq[2*k]   = (int16_t)lrint(i0 * c - q0 * sn);
                          iq[2*k+1] = (int16_t)lrint(i0 * sn + q0 * c);
                      }
                  }
              } }
            memcpy(g_livre_iq, iq, (size_t)n_iq * sizeof(int16_t)); g_livre_niq = n_iq;
            { static int da = -1; static unsigned nd;
              if (da < 0) da = calypso_getenv("REJEU_ADR") ? 1 : 0;
              if (da && nd < 14) { nd++;
                  printf("  [adr] fn=%-4u type=%c n_iq=%-4d -> depot 0x%04x len=%u\n",
                         fn_cur, g_livre_type ? g_livre_type : '?', n_iq,
                         calypso_bsp_get_daram_addr(), calypso_bsp_get_daram_len()); } }
            /* [2026-09-20] PAGE-BY-PAGE DELIVERY (REJEU_PAGES=<words>, e.g. 96).
             * On silicon the RIF streams continuously and DMA2 completes one
             * 96-word page at a time, each completion interrupting the DSP; the
             * FB correlator runs per page. Handing the whole frame in one call
             * drains 3-4 pages in one pass with ONE interrupt, so the ROM
             * processes one page per frame and its TOA advanced by 96 per frame
             * instead of 156 symbols. Here the frame's samples are delivered in
             * chunks, the DSP running a slice after each, so every page gets its
             * own completion. */
            { static long pages_mots = -1;
              if (pages_mots < 0) { const char *e = calypso_getenv("REJEU_PAGES"); pages_mots = (e && *e) ?
               ↪  atol(e) : 0; }
              if (pages_mots > 0) {
                  int pos = 0, npage = 0;
                  while (pos < n_iq) {
                      int m = n_iq - pos < pages_mots ? n_iq - pos : (int)pages_mots;
                      uint16_t b34 = dsp->data[0x3fb4], b35 = dsp->data[0x3fb5];
                      calypso_bsp_rx_burst(0, fn_cur, iq + pos, m);
                      pos += m; npage++;
                      { static unsigned nb; if (nb < 40 && fn_cur >= 2 && fn_cur <= 14) { nb++;
                         printf("  [bloc] fn=%u page %d (%d mots) : 0x3fb4=%04x 0x3fb5=%04x avant", fn_cur, npage,
↪  m, b34, b35); } }
                      /* completion interrupt already pending: let the DSP serve it */
                      if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
                      long slice = 0;
                      while (slice < insns / 8 && dsp->running && !dsp->idle) {
                          int ex = c54x_run(dsp, 64); if (ex <= 0) break; slice += ex;
                      }
                      { static unsigned nb2; if (nb2 < 40 && fn_cur >= 2 && fn_cur <= 14) { nb2++;
                          printf(" -> apres 0x3fb4=%04x 0x3fb5=%04x (idle=%d, %ld insn)\n", dsp->data[0x3fb4],
↪  dsp->data[0x3fb5], dsp->idle, slice); } }
                  }
                  { static unsigned np; if (np < 3) { np++;
                      printf("  [pages] fn=%u : %d mots livres en %d pages de %ld\n", fn_cur, n_iq, npage,
↪  pages_mots); } }
              } else
```

```
            calypso_bsp_rx_burst(0, fn_cur, iq, n_iq);
        }
        /* [2026-09-20] STREAM PUMP. The DMA now fills its double buffer with
         * full pages only and interrupts once per pair; the ROM's ISR consumes
         * both halves and the DSP goes idle. Then the next pair is handed
         * over, as the hardware would once the ISR is out of the way. A frame
         * carries 3.25 pages, so this runs 1 or 2 times per frame. */
        for (int k = 0; k < 40; k++) {          /* a 1250-symbol frame is 26 pages = 13 pairs */
            if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
            long sl = 0;
            while (sl < insns / 4 && dsp->running && !dsp->idle) {
                int ex = c54x_run(dsp, 64); if (ex <= 0) break; sl += ex;
            }
            if (!calypso_rhea_dma_pump(dsp)) break;
        }
        if (calypso_getenv("CALYPSO_BSP_VERIF")) {
            static int dit;
            if (!dit) { dit = 1;
                printf("  [mem] api est-il un alias de data[0x0800] ? %s\n",
                       (void *)api == (void *)&dsp->data[0x0800] ? "OUI" : "NON — deux memoires distinctes");
                printf("        api[0x0cce-0x0800]=%04x   data[0x0cce]=%04x\n",
                       api[0x0cce - 0x0800], dsp->data[0x0cce]); }
            static unsigned nvr;
            uint32_t vfn; uint16_t vad; int vn, vage;
            int id = calypso_bsp_verif_compare(&vfn, &vad, &vn, &vage);
            if (id >= 0 && nvr < 8) { nvr++;
                printf("  [ref] fn=%-4u type=%c age=%d : %d/%d identiques en 0x%04x%s\n",
                       vfn, g_livre_type ? g_livre_type : '?', vage, id, vn, vad,
                       id == vn ? "   VALIDE" : "   <<< ECRITURE FAUSSE"); }
        }
        { static int vv = -1; static unsigned nv;
          if (vv < 0) vv = calypso_getenv("REJEU_VIE") ? 1 : 0;
          if (vv && g_livre_type == 'S' && nv < 6) { nv++;
              uint16_t ad = calypso_bsp_get_daram_addr();
              printf("  [vie] fn=%-4u adresse APRES l'appel : 0x%04x (avant : 0x%04x)\n",
                     fn_cur, ad, g_ad_avant);
              int ex = 0, best = 0, bex = -1;
              for (int sh = -8; sh <= 44; sh++) {
                  int e = 0;
                  for (int k = 0; k < 280; k++) {
                      int j = k + sh; if (j < 0 || j >= n_iq) continue;
                      if ((int16_t)dsp->data[(ad + k) & 0x3fff] == iq[j]) e++;
                  }
                  if (e > bex) { bex = e; best = sh; }
                  if (sh == 0) ex = e;
              }
             printf("  [vie] fn=%-4u APRES depot en 0x%04x : shift0=%d/280  meilleur shift=%+d avec %d/280\n",
                    fn_cur, ad, ex, best, bex);
              if (bex < 200) {   /* pas trouve la : ou est le burst ? */
                  int ba = -1, bn = 0;
                  for (unsigned a = 0; a + 280 < 0x4000; a++) {
                      int e = 0;
                      for (int k = 0; k < 64; k++)
                          if ((int16_t)dsp->data[a + k] == iq[k]) e++;
                      if (e > bn) { bn = e; ba = (int)a; }
                  }
                  if (bn >= 60) {
                      int tot = 0;
                      for (int k = 0; k < 280; k++)
                          if ((int16_t)dsp->data[(ba + k) & 0x3fff] == iq[k]) tot++;
                      printf("        -> burst TROUVE en 0x%04x : %d/280 identiques\n", ba, tot);
                  } else printf("        -> burst introuvable en DARAM (meilleur %d/64 en 0x%04x)\n", bn, ba);
              }
              g_vie_fn = fn_cur; g_vie_ad = ad; } }
        if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
    }
```

```
        { static int tf = -1; if (tf < 0) tf = calypso_getenv("REJEU_TRACE_FB") ? 1 : 0;
          if (tf && fn_cur < 60)
              printf("  [fb] fn=%-3u apres pompe : 0x3fb4=%04x 0x3fb3=%04x d_fb_det=%u fb_mode=%u task_md=%u/%u
↪  idle=%d insn_trame=%u  sync=%04x %04x %04x %04x\n",
                     fn_cur, dsp->data[0x3fb4], dsp->data[0x3fb3], api[NDB_FB_DET], api[NDB_FB_MODE],
                     api[W_PAGE(0) + W_TASK_MD], api[W_PAGE(1) + W_TASK_MD], dsp->idle, dsp->insn_count -
↪  insn_debut_trame,
                     api[NDB_SYNC], api[NDB_SYNC+1], api[NDB_SYNC+2], api[NDB_SYNC+3]); }
        long done = 0;
        static int probe = -1;
        if (probe < 0) probe = drapeau_env("REJEU_PROBE_TOA") ? 1 : 0;
        if (!probe) {
            /* Always count, step by step: running 256 instructions at a time without
             * looking at a single PC left hit_b219/7c31/84a1/9841/770a at zero while the
             * summary still printed them, so "job b219=0 ... corr FB 770a=0" only meant
             * that nobody was counting. The PC histogram says what the DSP really
             * executes. */
            while (done < insns && dsp->running && !dsp->idle) {
                uint16_t pc = dsp->pc & 0xffff;
                histo_pc[pc >> 10]++;
                switch (pc) {
                case 0x7c31: hit_7c31++; break;
                case 0x9841: hit_9841++; break;
                case 0x84a1: hit_84a1++; break;
                case 0x770a: hit_770a++; break;
                case 0xb219: hit_b219++; break;
                case 0x7a16: hit_7a16++; break;
                /* The two FIRS sites of the SB. The one at 0x8493 is followed by
                 * `sth *AR6+,B` at 0x8497: THAT is what writes the soft bits. If it does
                 * not run, the softs come out of a stale B. */
                case 0x⟨1⟩: hit_⟨2⟩++; break;  ×4
⟨⟩ = (8478,8478) (8492,8492) (8493,8493) (8497,8497)
                /* [2026-09-19] THE 16-STEP DIVISION. 0x7d1c RPT #15; 0x7d1d SUBC
                 * *(0x0b),A; 0x7d1e STL A,*(0x0a) = the quotient. A null divisor gives a
                 * degenerate quotient and the whole cascade follows. */
                /* [2026-09-19] The dividend is born here. It should be 1 shifted
                 * (ld #1,A; sfta A,<n>) and it arrives NULL at the division. Trace A over
                 * the whole prologue to see which instruction zeroes it. */
                case 0x7d10: case 0x7d11: case 0x7d12: case 0x7d13:
                case 0x7d14: case 0x7d15: case 0x7d16: case 0x7d17:
                case 0x7d18: case 0x7d19: case 0x7d1a: case 0x7d1b:
                    if (env_div) {
                        static int dumpe;
                        if (!dumpe) { dumpe = 1;
                            printf("    [rom] 0x7d10-0x7d20 :");
                            for (unsigned a = 0x7d10; a <= 0x7d20; a++)
                                printf(" %04x", prog_ovly(dsp, (uint16_t)a));
                            printf("\n"); }
                        static unsigned np;
                        if (np < 26) { np++;
                            printf("    [pro] pc=%04x op=%04x  A=%010llx  T=%04x ST0=%04x ST1=%04x\n",
                                   pc, prog_ovly(dsp, pc),
                                   (unsigned long long)(dsp->a & 0xfffffffffULL),
                                   dsp->t, dsp->st0, dsp->st1); }
                    }
                    break;
                case 0x7d1c: hit_7d1c++;
                    if (env_div) {
                        unsigned dp = dsp->st0 & 0x1FF;
                        uint16_t dv = dsp->data[(uint16_t)((dp << 7) | 0x0B)];
                        static unsigned nd;
                        if (nd < 12) { nd++;
                            printf("    [div] avant : dividende A=%010llx  diviseur=0x%04x (%d)%s\n",
                                   (unsigned long long)(dsp->a & 0xfffffffffULL), dv, (int16_t)dv,
                                   dv == 0 ? "   <<< DIVISEUR NUL" : ""); }
                    }
```

```
    break;
case 0x7d1d: hit_7d1d++; break;
case 0x7d1e: hit_7d1e++;
    if (env_div) {
        static unsigned nq;
        if (nq < 12) { nq++;
            printf("    [div] apres : quotient A=%010llx  (mot bas = %d)\n",
                   (unsigned long long)(dsp->a & 0xfffffffffULL),
                   (int16_t)(dsp->a & 0xffff)); }
    }
    break;
case 0x81e4: hit_81e4++; break;
default: break; }
/* [2026-09-19] WHO WRITES a_sch? The status word a_sch[0] sometimes
 * receives plain numbers (0x1111, 0x1388=5000, 0x142e) where only B_BLUD
 * (bit15) and B_SCH_CRC (bit8) have a meaning -- and the 0x8000 read as
 * "CRC OK" is more likely a saturated accumulator. a_sch[0..4] =
 * R_PAGE+15.., i.e. data[0x0837..0x083b] (page 0) and
 * data[0x084b..0x084f] (page 1). Record the PC of each write.
 * REJEU_QUI_ASCH=1. */
{ static int qa = -1; static uint16_t sh[10]; static int ini; static unsigned nqa;
  if (qa < 0) qa = calypso_getenv("REJEU_QUI_ASCH") ? 1 : 0;
  if (qa) {
      static const uint16_t adr[10] = {0x0837,0x0838,0x0839,0x083a,0x083b,
                                       0x084b,0x084c,0x084d,0x084e,0x084f};
      if (!ini) { for (int k=0;k<10;k++) sh[k]=dsp->data[adr[k]]; ini=1; }
      for (int k=0;k<10;k++) {
          uint16_t v = dsp->data[adr[k]];
          if (v != sh[k]) {
              /* On a CRC OK, print the FULL word: eight events
               * carrying the same word are not eight independent
               * draws but one attractor reached eight times, which
               * changes the statistics completely. */
              if (k % 5 == 0 && v == 0x8000) {
                  const uint16_t *b = &dsp->data[adr[k]];
                  printf("    [crcok] fn=%u  a_sch = %04x %04x %04x %04x %04x"
                         "  -> mot 0x%04x%04x  crc_interne(2bf8)=%u\n",
                         fn_cur, b[0], b[1], b[2], b[3], b[4],
                         b[4], b[3], dsp->data[0x2bf8]);
              }
              if (nqa < 4000) { nqa++;
                  int j = k % 5;
                  printf("    [asch] a_sch[%d] (page %d, 0x%04x) : %04x -> %04x"
                         "   ecrit juste avant pc=%04x  fn=%u%s\n",
                         j, k/5, adr[k], sh[k], v, pc, fn_cur,
                         (j==0 && v && v!=0x0100 && v!=0x8000 && v!=0x8100)
                           ? "   <<< PAS UN DRAPEAU" : ""); }
              sh[k] = v;
          }
      }
  } }
/* [2026-09-19] WHICH PC PINS A AT +-0x40000000 (A_high = +-16384)?
 * Everything else follows from it: the clipping at 0x2ac0, B zeroed by
 * the ADD at 0x8389, the degenerate softs. Record the PC of each
 * transition to that value. REJEU_QUI_A=1. */
{ static int qA = -1; static int64_t aprec; static unsigned long parpc[0x10000];
  static unsigned long tA; static int armA;
  if (qA < 0) qA = calypso_getenv("REJEU_QUI_A") ? 1 : 0;
  if (qA) {
      int64_t a40 = dsp->a & 0xfffffffffLL;
      int pin = (a40 == 0x0040000000LL || a40 == 0xffc0000000LL);
      int pin0 = (aprec == 0x0040000000LL || aprec == 0xffc0000000LL);
      static uint16_t pcp;
      if (pin && !pin0 && armA) parpc[pcp]++;
      aprec = a40; pcp = pc; armA = 1;
      if (++tA % 400000 == 0) {
```

```
                    printf("  [quiA] PC qui amenent A a +-0x40000000 :\n");
                    for (int rang=0; rang<8; rang++) {
                        unsigned best=0; unsigned long bv=0;
                        for (unsigned i=0;i<0x10000;i++) if (parpc[i]>bv){bv=parpc[i];best=i;}
                        if (!bv) break;
                        printf("    pc=%04x op=%04x : %lu fois\n", best,
                               prog_ovly(dsp,(uint16_t)best), bv);
                        parpc[best]=0;
                    }
                    tA = 1;
                }
              } }
            /* [2026-09-19] WHO WRITES THE +-16384 INTO 0x2ac0? Watch the buffer
             * word by word and record the PC of each write, separating clipped
             * values from the rest. REJEU_QUI2AC0=1. */
            { static int qc = -1; static uint16_t sh2[64]; static int ini3;
              static unsigned long par_pc_butee[0x10000], par_pc_autre[0x10000];
              static unsigned long tot2;
              if (qc < 0) qc = calypso_getenv("REJEU_QUI2AC0") ? 1 : 0;
              if (qc) {
                  if (!ini3) { for (int k=0;k<64;k++) sh2[k]=dsp->data[0x2ac0+k]; ini3=1; }
                  for (int k=0;k<64;k++) {
                      uint16_t v = dsp->data[0x2ac0+k];
                      if (v != sh2[k]) {
                          int16_t sv = (int16_t)v;
                          if (sv == 16384 || sv == -16384) par_pc_butee[pc]++;
                          else par_pc_autre[pc]++;
                          sh2[k] = v;
                      }
                  }
                  if (++tot2 % 300000 == 0) {
                      printf("  [2ac0] PC ecrivains (butee +-16384 | autres) :\n");
                      for (int rang=0; rang<8; rang++) {
                          unsigned best=0; unsigned long bv=0;
                          for (unsigned i=0;i<0x10000;i++)
                              if (par_pc_butee[i]+par_pc_autre[i] > bv) { bv=par_pc_butee[i]+par_pc_autre[i];
                                  ↪  best=i; }
                          if (!bv) break;
                          printf("    pc=%04x op=%04x : butee=%lu  autres=%lu\n",
                                 best, prog_ovly(dsp,(uint16_t)best),
                                 par_pc_butee[best], par_pc_autre[best]);
                          par_pc_butee[best]=0; par_pc_autre[best]=0;
                      }
                      tot2 = 1;
                  }
              } }
            /* [2026-09-19] DOES THE WORKING BUFFER HOLD THE BURST? The injected
             * samples are known (g_livre_iq); after the copy, 0x2a00 and 0x2ac0
             * should carry the I and Q channels. Compare instead of assuming.
             * REJEU_CMP2A=1: at the first pass at 0x84a0 (correlator entry, so the
             * copy is done) on an SCH frame. */
            { static int cm = -1; static int fait2;
              if (cm < 0) cm = calypso_getenv("REJEU_CMP2A") ? 1 : 0;
              if (cm && fait2 < 3 && pc == 0x84a0 && g_n_reels &&
                  g_reel_pour_fn[g_sb_cmd_fn & 63] >= 0) {
                  fait2++;
                  int ri = g_reel_pour_fn[g_sb_cmd_fn & 63];
                  const int16_t *bi = g_reels[ri];
                  int m = 0;
                 printf("  [cmp] burst reel #%d (fn %u, BSIC %u) vs tampons de travail\n", ri, g_reels_fn[ri],
↪  g_reels_bsic[ri]);
                  printf("    I injecte : "); for (int k=0;k<10;k++) printf(" %6d", bi[2*(m+k)]);
                  printf("\n    0x2a00    : "); for (int k=0;k<10;k++) printf(" %6d",
↪  (int16_t)dsp->data[0x2a00+k]);
                  printf("\n    0x2ac0    : "); for (int k=0;k<10;k++) printf(" %6d",
↪  (int16_t)dsp->data[0x2ac0+k]);
```

```
                printf("\n    Q injecte : "); for (int k=0;k<10;k++) printf(" %6d", bi[2*(m+k)+1]);
                printf("\n");
                /* normalized correlation between each buffer and each channel */
                /* how far the constancy goes: how many words are identical? */
                for (int buf=0; buf<2; buf++) {
                    uint16_t base = buf ? 0x2ac0 : 0x2a00;
                    int16_t v0 = (int16_t)dsp->data[base];
                    int same = 0, n = 0; int16_t mn=32767, mx=-32768;
                    for (int k=0;k<190;k++) { int16_t v=(int16_t)dsp->data[base+k];
                        if (v==v0) same++;
                        if(v<mn)mn=v;
                        if(v>mx)mx=v;
                        n++; }
                    printf("    0x%04x : %d/%d mots egaux au premier (%d) ; etendue [%d..%d]\n",
                           base, same, n, v0, mn, mx);
                }
                for (int buf=0; buf<2; buf++) {
                    uint16_t base = buf ? 0x2ac0 : 0x2a00;
                    for (int voie=0; voie<2; voie++) {
                        double sxy=0, sxx=0, syy=0;
                        for (int k=0;k<128;k++) {
                            double x=(int16_t)dsp->data[base+k], y=bi[2*(m+k)+voie];
                            sxy+=x*y; sxx+=x*x; syy+=y*y; }
                        printf("    correlation 0x%04x vs %c : %+.3f\n", base, voie?'Q':'I',
                               (sxx>0&&syy>0)? sxy/sqrt(sxx*syy) : 0.0);
                    }
                }
              } }
            /* [2026-09-19] THE burst -> working buffer COPY, the last link never
             * examined: the burst arrives exact at 0x0cce and buffer 0x2a80 is
             * filled by 0x81d0..0x81da. What does that loop READ? */
            { static int rc = -1; static unsigned n;
              if (rc < 0) rc = calypso_getenv("REJEU_RECOPIE") ? 1 : 0;
              if (rc && pc >= 0x81c8 && pc <= 0x81e0) {
                /* Does AR5 sweep a table, or stay on two cells? */
                static unsigned lo = 0xffff, hi = 0, vus[64], nv;
                if (dsp->ar[5] < lo) lo = dsp->ar[5];
                if (dsp->ar[5] > hi) hi = dsp->ar[5];
                { int trouve = 0; for (unsigned q = 0; q < nv; q++) if (vus[q] == dsp->ar[5]) trouve = 1;
                  if (!trouve && nv < 64) vus[nv++] = dsp->ar[5]; }
                /* Does the multiplicand change? A phasor kept in place would
                 * change value at every step; a constant would not. */
                { static unsigned long nval; static uint16_t vu_v[32]; static unsigned nvv;
                  uint16_t v = dsp->data[dsp->ar[5] & 0x3fff];
                  int t2 = 0; for (unsigned q = 0; q < nvv; q++) if (vu_v[q] == v) t2 = 1;
                  if (!t2 && nvv < 32) vu_v[nvv++] = v;
                  if (++nval % 4000 == 0) {
                      printf("    [mul] %u valeurs distinctes lues via AR5 :", nvv);
                      for (unsigned q = 0; q < nvv && q < 12; q++) printf(" %04x", vu_v[q]);
                      printf("\n"); } }
                static unsigned long tot;
                if (++tot % 4000 == 0)
                    printf("    [ar5] apres %lu pas : plage 0x%04x..0x%04x, %u cellules distinctes\n",
                           tot, lo, hi, nv);
              }
              if (rc && n < 16 && pc >= 0x81c8 && pc <= 0x81e0) {
                  n++;
                  printf("    [cp] pc=%04x op=%04x  A=%010llx B=%010llx"
                         "  AR2=%04x->%04x AR3=%04x->%04x AR4=%04x->%04x AR5=%04x->%04x\n",
                         pc, prog_ovly(dsp, pc),
                         (unsigned long long)(dsp->a & 0xffffffffffULL),
                         (unsigned long long)(dsp->b & 0xffffffffffULL),
                         dsp->ar[(1)], dsp->data[dsp->ar[(2)] & 0x3fff],  ×3
⟨⟩ = (2,2) (3,3) (4,4)
                         dsp->ar[5], dsp->data[dsp->ar[5] & 0x3fff]);
              } }
```

```
/* [2026-09-19] THE BLOCK THAT OVERWRITES THE LOWER HALF OF THE FIRS
 * WINDOW. 0x832b..0x8330 writes 0000 x4 then 4000 x2 there. Boundary
 * conditions, or clobbering? Print the program words of the area and the
 * registers on entry, once. REJEU_DUMP832=1. */
{ static int d8 = -1; static int fait;
  if (d8 < 0) d8 = calypso_getenv("REJEU_DUMP832") ? 1 : 0;
  if (d8 && !fait && pc == 0x8320) {
      fait = 1;
      printf("  [832] programme 0x8318-0x8340 (alias OVLY compris) :\n");
      for (unsigned a = 0x8318; a <= 0x8340; a += 8) {
          printf("    %04x:", a);
          for (int k = 0; k < 8 && a + k <= 0x8340; k++)
              printf(" %04x", prog_ovly(dsp, (uint16_t)(a + k)));
          printf("\n");
      }
      printf("    registres : A=%010llx B=%010llx T=%04x\n",
             (unsigned long long)(dsp->a & 0xffffffffffULL),
             (unsigned long long)(dsp->b & 0xffffffffffULL), dsp->t);
      printf("    AR0=%04x AR1=%04x AR2=%04x AR3=%04x AR4=%04x AR5=%04x AR6=%04x AR7=%04x\n",
             dsp->ar[0],dsp->ar[1],dsp->ar[2],dsp->ar[3],
             dsp->ar[4],dsp->ar[5],dsp->ar[6],dsp->ar[7]);
  } }
/* [2026-09-19] THE CORRELATOR BLOCK, PC BY PC, OVER A SINGLE DECODE.
 * 3100 MACs per decode reads either as "50 x 62" or "31 x 100"; only a
 * per-pass count decides. Count each PC in 0x84a0..0x84d0 during the
 * FIRST decode, then print. REJEU_BLOC=1. */
{ static int bl = -1; static unsigned long cnt[0x40]; static int fini, vu_dec;
  if (bl < 0) bl = calypso_getenv("REJEU_BLOC") ? 1 : 0;
  if (bl && !fini) {
      if (pc >= 0x84a0 && pc <= 0x84df) cnt[pc - 0x84a0]++;
      if (pc == 0x9841) {
          if (!vu_dec) { vu_dec = 1; }
          else {
              fini = 1;
              printf("  [bloc] correlateur 0x84a0-0x84df sur UN decodage :\n");
              for (int k = 0; k < 0x40; k++)
                  if (cnt[k]) printf("    pc=%04x  op=%04x  %6lu fois\n",
                                     0x84a0 + k, prog_ovly(dsp, 0x84a0 + k), cnt[k]);
          }
      }
  } }
/* [2026-09-19] IMPULSE RESPONSE OF 0x2c72. Force ONE position to an
 * extreme value, held over the whole window, and compare the output
 * between +v and -v. How many decoded bits move:
 *   1 position       -> a permutation, 0x2c72 really is one soft per bit
 *   3 or 4 adjacent  -> normal ISI, the chain is sound here
 *   all / none       -> these are not per-bit softs
 * REJEU_IMPULSION=<k> REJEU_IMPULSION_VAL=<v>. */
{ static int ik = -2, iv;
  if (ik == -2) { const char *e = calypso_getenv("REJEU_IMPULSION");
                  ik = e ? atoi(e) : -1;
                  const char *w = calypso_getenv("REJEU_IMPULSION_VAL");
                  iv = w ? atoi(w) : 20000; }
  if (ik >= 0 && ik < 78 && g_dans_sb)
      dsp->data[0x2c72 + ik] = (uint16_t)(int16_t)iv; }
/* [2026-09-19] IS THE PEAK USED AT ALL? The addresses FIRS reads do
 * not move when the burst moves, while the peak does follow. Force the
 * peak to an arbitrary value over the whole demodulation window: if
 * nothing downstream changes, it is not consumed and the equalizer is
 * aligned on nothing. REJEU_FORCER_PIC=<n>. */
{ static int fp = -2;
  if (fp == -2) { const char *e = calypso_getenv("REJEU_FORCER_PIC"); fp = e ? atoi(e) : -1; }
  if (fp >= 0 && g_dans_sb) dsp->data[0x2f06] = (uint16_t)fp; }
/* [2026-09-19] PERFECT SOFTS. The fault lies somewhere between the
 * correlator (sound) and the soft-bit write. Split the space in two:
 * just before the decoder (0x9841) reads 0x2c72, write the ideal soft
```

```
 * bits there ourselves, derived from the 78 bits the injected burst
 * REALLY carries. If the DSP then returns BSIC=32, everything downstream
 * (Viterbi, CRC, a_sch packing) is proven and the fault is strictly in
 * soft PRODUCTION; otherwise it is downstream.
 * REJEU_SOFTS_PARFAITS=<amplitude>, sign tested both ways via
 * REJEU_SOFTS_POLARITE=0|1. REJEU_SOFTS_CONTINU=1 holds the ideal softs
 * at EVERY step of the demodulation window, not only at the decoder
 * entry -- otherwise a read before 0x9841 escapes the injection and "no
 * effect" means nothing. */
if (pc == 0x9841 || (env_softs_continu && g_dans_sb)) {
    static int amp = -2, pol = -1;
    if (amp == -2) { const char *e = calypso_getenv("REJEU_SOFTS_PARFAITS");
                     amp = e ? atoi(e) : -1;
                     const char *q = calypso_getenv("REJEU_SOFTS_POLARITE");
                     pol = q ? atoi(q) : 0; }
    if (amp > 0) {
        int ri = g_n_reels ? g_reel_pour_fn[g_sb_cmd_fn & 63] : -1;
        unsigned char att[78];
        if (ri >= 0) memcpy(att, g_reels_code[ri], 78);
        else cellule_code_attendu(g_sb_cmd_fn, (uint8_t)g_bsic_injecte, att);
        /* REJEU_SOFTS_PERM: the order of the 78 values is a hypothesis.
         * none = as is; swap = the two halves of 39 exchanged;
         * rev = reversed order; entrelace = even/odd separated. */
        static const char *perm; static int perm_lu;
        if (!perm_lu) { perm = calypso_getenv("REJEU_SOFTS_PERM"); perm_lu = 1; }
        /* [2026-09-20] The decoder reads its 78 softs at 0x2a00
         * (ROM 0x984a `stm #0x2a00,AR1`; packing 0x7e65-0x7e7a),
         * NOT at 0x2c72. Writing 0x2c72 tested nothing.
         * REJEU_SOFTS_ADDR overrides (default 0x2a00). */
        static long sa = -1;
        if (sa < 0) { const char *e = calypso_getenv("REJEU_SOFTS_ADDR");
                      sa = (e && *e) ? strtol(e, NULL, 0) : 0x2a00; }
        for (int k = 0; k < 78; k++) {
            int j = k;
            if (perm && !strcmp(perm, "swap"))       j = (k < 39) ? k + 39 : k - 39;
            else if (perm && !strcmp(perm, "rev"))   j = 77 - k;
            else if (perm && !strcmp(perm, "entrelace")) j = (k < 39) ? 2*k : 2*(k-39)+1;
            int bit = att[j] & 1;
            int v = (pol ? bit : !bit) ? amp : -amp;
            dsp->data[(uint16_t)(sa + k)] = (uint16_t)(int16_t)v;
        }
        static unsigned ns; if (ns < 3) { ns++;
            printf("  [softs] fn_demod=%u : 78 souples IDEAUX ecrits en 0x%04lx "
                   "(amp=%d pol=%d, source=%s)\n", g_sb_cmd_fn, sa, amp, pol,
                   ri >= 0 ? "burst reel" : "fixture"); }
    }
}
/* [2026-09-19] WHO WRITES THE EQUALIZER INPUT WINDOW? FIRS reads
 * 0x2a8e..0x2a9a and its lower half is zero or 0x4000. Watch the area
 * word by word and record the PC of each write: that says who fills it,
 * in what order, and where the filling stops. REJEU_QUI2A=1. */
{ static int q2 = -1; static uint16_t shadow[0x30]; static int init2;
  static unsigned nq;
  if (q2 < 0) q2 = calypso_getenv("REJEU_QUI2A") ? 1 : 0;
  if (q2) {
      if (!init2) { for (int k = 0; k < 0x30; k++) shadow[k] = dsp->data[0x2a80 + k]; init2 = 1; }
      for (int k = 0; k < 0x30; k++) {
          uint16_t v = dsp->data[0x2a80 + k];
          if (v != shadow[k]) {
              if (nq < 4000) { nq++;
                  printf("    [qui] 0x%04x : %04x -> %04x   ecrit juste avant pc=%04x (fn=%u)\n",
                         0x2a80 + k, shadow[k], v, pc, fn_cur); }
              shadow[k] = v;
          }
      }
  } }
```

```
/* [2026-09-19] WHO FILLS B? The correlator is sound (the peak follows
 * the margin), FIRS carries nothing, and yet `sth *AR6+,B` at 0x8497 is
 * what writes the soft bits. Trace A and B over the whole window
 * 0x8470..0x84a0 to see where B takes its value. REJEU_TRACE_B=1. */
/* 0x847c / 0x8498 : op=0x4485 = LD Smem,16,A per tic54x-opc.c
 * (0x4400/0xFE00). No handler matches this mask in c54x_exec.c,
 * yet A changes across it. Dump A before/after and every AR with
 * the word it points at, to find where the value comes from. */
/* Dump the DSP work buffers at the correlator entry, together with
 * the real burst that was fed, so an external model can identify what
 * each buffer holds instead of guessing. REJEU_DUMP_BUF=<path>. */
/* Who writes the BSP deposit window 0x0cce, and when? At correlator
 * entry the window holds none of the injected bursts and its content
 * does not change between frames. Shadow the window, log each change
 * with the PC that made it. REJEU_QUI_CCE=1. */
/* Timeline of the deposit window on one SB frame: every change with
 * the PC and the instruction count, plus the correlator entry. Tells
 * whether the burst is destroyed before or after the SB reads it.
 * REJEU_CHRONO=<fn>. */
{ static long cf = -2; static uint16_t sh2[380]; static int ini2; static unsigned nl;
  if (cf == -2) { const char *e = calypso_getenv("REJEU_CHRONO"); cf = e ? atol(e) : -1; }
  if (cf >= 0 && (long)fn_cur == cf) {
      if (!ini2) { for (int k=0;k<380;k++) sh2[k]=dsp->data[0x0cce + k]; ini2=1;
                   printf("  [chrono] trame %ld\n", cf); }
      unsigned chg=0;
      for (int k=0;k<380;k++) { uint16_t v=dsp->data[0x0cce + k];
          if (v!=sh2[k]) { chg++; sh2[k]=v; } }
      if (chg && nl < 24) { nl++;
          printf("      insn=%-8u pc=%04x  %4u mots changes\n",
                 dsp->insn_count, pc, chg); }
      if (pc == 0x84a0 && nl < 30) { nl++;
          printf("      insn=%-8u pc=84a0  <== ENTREE DU CORRELATEUR SB\n",
                 dsp->insn_count); }
      if (pc == 0x7c31 && nl < 30) { nl++;
          printf("      insn=%-8u pc=7c31  <== entree du demodulateur SB\n",
                 dsp->insn_count); }
  } }
{ static int qc = -1; static uint16_t sh[380]; static int ini;
  static unsigned long par_pc[0x10000], tot; static unsigned long nfr[64];
  if (qc < 0) qc = calypso_getenv("REJEU_QUI_CCE") ? 1 : 0;
  if (qc) {
      if (!ini) { for (int k=0;k<380;k++) sh[k]=dsp->data[0x0cce + k]; ini=1; }
      unsigned chg = 0;
      for (int k=0;k<380;k++) {
          uint16_t v = dsp->data[0x0cce + k];
          if (v != sh[k]) { chg++; sh[k]=v; }
      }
      if (chg) { par_pc[pc] += chg; nfr[fn_cur & 63] += chg; }
      if (++tot % 500000 == 0) {
          printf("  [cce] ecrivains de 0x0cce (mots modifies) :\n");
          for (int r=0;r<6;r++) {
              unsigned best=0; unsigned long bv=0;
              for (unsigned i=0;i<0x10000;i++) if (par_pc[i]>bv){bv=par_pc[i];best=i;}
              if (!bv) break;
              printf("      pc=%04x op=%04x : %lu mots\n", best,
                     prog_ovly(dsp,(uint16_t)best), bv);
              par_pc[best]=0;
          }
          tot=1;
      }
  } }
{ static int vv2 = -1; static unsigned nv2;
  if (vv2 < 0) vv2 = calypso_getenv("REJEU_VIE") ? 1 : 0;
  if (vv2 && pc == 0x84a0 && g_vie_ad && nv2 < 5) { nv2++;
      int ex = 0;
      for (int k = 0; k < 296; k++)
```

```
            if ((int16_t)dsp->data[(g_vie_ad + k) & 0x3fff] == g_livre_iq[2*marge_tete() + k]) ex++;
        printf("  [vie] fn=%-4u ENTREE CORRELATEUR (depot de fn=%u) : %d/296 identiques\n",
               fn_cur, g_vie_fn, ex); } }
{ static int db = -1; static FILE *fb;
  if (db < 0) { const char *e = calypso_getenv("REJEU_DUMP_BUF");
                db = e ? 1 : 0; if (db) fb = fopen(e, "wb"); }
  if (db && fb && pc == 0x84a0 && g_n_reels) {
      int ri = g_reel_pour_fn[g_sb_cmd_fn & 63];
      if (ri >= 0) {
          static int nb;
          if (nb < 8) { nb++;
              uint32_t hdr[4] = { g_sb_cmd_fn, (uint32_t)ri,
                                  g_reels_fn[ri], g_reels_bsic[ri] };
              fwrite(hdr, 4, 4, fb);
              fwrite(g_reels[ri], 2, 296, fb);          /* le burst injecte */
              fwrite(&dsp->data[0x0cce], 2, 380, fb);   /* depot BSP */
              fwrite(&dsp->data[0x2a00], 2, 256, fb);   /* tampon 1 */
              fwrite(&dsp->data[0x2ac0], 2, 256, fb);   /* tampon 2 */
              fwrite(&dsp->data[0x2c72], 2, 78, fb);    /* bits souples */
              fwrite(g_reels_code[ri], 1, 78, fb);      /* bits emis */
              fflush(fb);
          }
      }
  } }
{ static int q4 = -1; static unsigned n4; static int64_t avant; static int arme;
  if (q4 < 0) q4 = calypso_getenv("REJEU_Q4485") ? 1 : 0;
  if (q4) {
      if (arme) { arme = 0;
          printf("        -> A apres = %010llx\n",
                 (unsigned long long)(dsp->a & 0xfffffffffULL)); }
      if ((pc == 0x847c || pc == 0x8498) && n4 < 6) {
          n4++; avant = dsp->a; arme = 1;
          printf("    [4485] pc=%04x op=%04x  A avant = %010llx\n",
                 pc, prog_ovly(dsp, pc), (unsigned long long)(avant & 0xfffffffffULL));
          for (int k = 0; k < 8; k++)
              printf("        AR%d=%04x -> %04x\n", k, dsp->ar[k],
                     dsp->data[dsp->ar[k] & 0x3fff]);
      }
  } }
{ static int tb = -1; static unsigned ntb;
  if (tb < 0) tb = calypso_getenv("REJEU_TRACE_B") ? 1 : 0;
  if (tb && ntb < 70 && pc >= 0x8470 && pc <= 0x84a0) {
      ntb++;
      printf("    [B] pc=%04x op=%04x A=%010llx B=%010llx AR2=%04x->%04x AR3=%04x->%04x\n",
             pc, prog_ovly(dsp, pc),
             (unsigned long long)(dsp->a & 0xfffffffffULL),
             (unsigned long long)(dsp->b & 0xfffffffffULL),
             dsp->ar[2], dsp->data[dsp->ar[2] & 0x3fff],
             dsp->ar[3], dsp->data[dsp->ar[3] & 0x3fff]);
  } }
{ uint16_t o = prog_ovly(dsp, pc); uint8_t h = o >> 8;
  static int dans_sb2;
  if (pc == 0x7c31) dans_sb2 = 1;
  if (pc == 0x9841) dans_sb2 = 0;
  g_dans_sb = dans_sb2;
  /* [2026-09-18] WHAT FIRS MULTIPLIES. FIRS (0xE0, 2 words) takes its
   * pmad from the following word and reads its coefficients in PROGRAM
   * space. Print pmad, the coefficients as the core sees them (OVLY alias
   * included) and data[] at the same address: null or constant
   * coefficients mean the equalizer output CANNOT depend on its input,
   * which would be the root cause. */
  if (env_firs && dans_sb2 && h == 0xE0) {
      static unsigned nf;
      if (nf < 10) {
          uint16_t pmad = prog_ovly(dsp, (uint16_t)(pc + 1));
          printf("    [firs] #%u pc=%04x pmad=%04x  coef(prog+ovly)=", ++nf, pc, pmad);
```

```
            for (int k = 0; k < 6; k++) printf(" %04x", prog_ovly(dsp, (uint16_t)(pmad + k)));
            printf("   data[pmad..]=");
            for (int k = 0; k < 6; k++) printf(" %04x", dsp->data[(pmad + k) & 0x3fff]);
            printf("\n");
        }
    }
    if (pc >= 0x9800 && pc <= 0x9bff) g_op_dec[o]++;
    if (pc >= 0x8400 && pc <= 0x84ff) g_op_eq[o]++;
    if (pc == 0x9841) g_n_dec++;
    if (dans_sb2) {
        if (g_op_n[o]++ == 0) g_op_pc[o] = pc;
        if (h == 0x8E || h == 0x8F) n_cmps_sb++;
        if (h >= 0xE0 && h <= 0xE3) { n_e0_sb++; n_e0x_sb[h - 0xE0]++; }
        /* [2026-09-18] FIRS reads its coefficients at Pmem[pmad], with
         * pmad ~ 0x0061. But 0x0060-0x007F is the C54x DARAM SCRATCH-PAD
         * and the core's OVLY window only starts at 0x0080, so
         * prog_read(0x61) falls back on prog[] where nothing is loaded
         * below 0x7000. If the coefficients only live in data[], FIRS
         * multiplies by nothing and its output cannot depend on its input.
         * Read BOTH spaces at the FIRS. */
        if (h == 0xE0 && n_firs_vus < 6) {
            uint16_t pmad = dsp->prog[(pc + 1) & 0xffff];
            int nzp = 0, nzd = 0;
            for (int k = 0; k < 6; k++) {
                if (dsp->prog[(pmad + k) & 0xffff]) nzp++;
                if (dsp->data[(pmad + k) & 0x3fff]) nzd++;
            }
            /* [2026-09-18] Reading prog[] RAW is misleading: with
             * PMST_OVLY set and the alias floor at 0x0060 (gate
             * CALYPSO_OVLY_SCRATCH, default 1), a PROGRAM read in
             * 0x0060-0x27FF is redirected to data[]. Reproduce the core's
             * translation to know what FIRS REALLY reads, rather than what
             * the prog[] array holds. */
            int ovly = (dsp->pmst & 0x0020) != 0;
            int alias = ovly && pmad >= 0x0060 && pmad < 0x2800;
            printf("    [firs] pc=%04x pmad=%04x PMST=%04x OVLY=%d alias=%s"
                   " -> FIRS lit", pc, pmad, dsp->pmst, ovly,
                   alias ? "data[] (scratch-pad visible)" : "prog[] (PAS d'alias)");
            for (int k = 0; k < 6; k++)
                printf(" %04x", alias ? dsp->data[(pmad + k) & 0x3fff]
                                      : dsp->prog[(pmad + k) & 0xffff]);
            printf("\n");
            printf("    [firs] pc=%04x pmad=%04x | prog[pmad..+5]=", pc, pmad);
            for (int k = 0; k < 6; k++) printf(" %04x", dsp->prog[(pmad + k) & 0xffff]);
            printf(" (%d non nuls)\n                       | data[pmad..+5]=", nzp);
            for (int k = 0; k < 6; k++) printf(" %04x", dsp->data[(pmad + k) & 0x3fff]);
            printf(" (%d non nuls)\n", nzd);
            n_firs_vus++;
        }
    } else if (h == 0x8E || h == 0x8F) n_cmps_hors++;
    /* [2026-09-18] The dans_sb window closes at 0x9841, so it EXCLUDES
     * the SCH decoder and its Viterbi (0x9a78): "0 CMPS in the SB demod"
     * said nothing about the Viterbi. CMPS is therefore also counted per PC
     * region, with no window, which is unambiguous. */
    if (h == 0x8E || h == 0x8F) n_cmps_reg[pc >> 10]++;
    /* [2026-09-18] WHO WRITES 0x2a00? The decoder reads 78 words at
     * 0x2a00 and finds them ALL NULL on entry: either the equalization
     * stage never writes them, or something erases them first. Watch
     * 0x2a00..0x2a8d over the WHOLE SB job (not just the demod), recording
     * the PC of each change and its direction (to a value, or to zero). */
/* [2026-09-18] 0x8389 = 0x4594 = `ADD *AR4+,16,A,B` (0x4400/0xFC00,
 * bit9=src, bit8=dst). The core has NO handler for 0x4400-0x47FF: the
 * only one in the area is (op & 0xFC00) == 0x4000, which covers SUB
 * alone. Check that B really changes across that instruction. */
{ static int nav; static int64_t bavant;
    if (pc == 0x8389) { bavant = dsp->b; nav = 1; }
```

```
  else if (nav == 1 && pc == 0x838a) {
      static int n=0;
      if (n < 4) {
          printf("    [add-4594] B avant=%010llx  B apres=%010llx  A=%010llx  %s\n",
                 (unsigned long long)(bavant & 0xfffffffffULL),
                 (unsigned long long)(dsp->b & 0xfffffffffULL),
                 (unsigned long long)(dsp->a & 0xfffffffffULL),
                 (dsp->b == bavant) ? "B INCHANGE => instruction NON EXECUTEE" : "B modifie");
          n++;
      }
      nav = 0;
  } }
{ static int nb838d;
  if (pc == 0x838e && nb838d < 6) {
      printf("    [B@838d] B=%010llx  A=%010llx  AR6=%04x  BRC=%u  ST1=%04x\n",
             (unsigned long long)(dsp->b & 0xfffffffffULL),
             (unsigned long long)(dsp->a & 0xfffffffffULL),
             dsp->ar[6], dsp->brc, dsp->st1);
      nb838d++;
  } }
{ static uint16_t omb2[142]; static int arme6; static long ecr[64], vers0[64];
  static uint16_t pmin[64], pmax[64];
  static struct { uint16_t pc; long v, z; } parpc2[12];
  if (pc == 0xb219) {
      for (int k = 0; k < 142; k++) omb2[k] = dsp->data[0x2a00 + k];
      arme6 = 1;
      for (int b = 0; b < 64; b++) { ecr[b] = 0; vers0[b] = 0; }
      for (int t = 0; t < 12; t++) { parpc2[t].pc = 0; parpc2[t].v = 0; parpc2[t].z = 0; }
  }
  if (arme6) {
      for (int k = 0; k < 142; k++) {
          if (dsp->data[0x2a00 + k] != omb2[k]) {
              /* [2026-09-18] Bucket by EXACT PC, and separate value
               * writes from zeroing: 284 changes of which 142 to zero
               * means something writes the 142 soft bits then ERASES
               * them all, and both instructions must be named. The PC
               * printed is that of the FOLLOWING instruction. */
              { int z = (dsp->data[0x2a00 + k] == 0);
                for (int t = 0; t < 12; t++) {
                    if (parpc2[t].pc == 0 || parpc2[t].pc == pc) {
                        parpc2[t].pc = pc;
                        if (z) parpc2[t].z++; else parpc2[t].v++;
                        break;
                    }
                } }
              int b = pc >> 10;
              if (!ecr[b]) { pmin[b] = pc; pmax[b] = pc; }
              if (pc < pmin[b]) pmin[b] = pc;
              if (pc > pmax[b]) pmax[b] = pc;
              ecr[b]++;
              if (dsp->data[0x2a00 + k] == 0) vers0[b]++;
              omb2[k] = dsp->data[0x2a00 + k];
          }
      }
  }
  if (pc == 0x9841 && arme6 && n_2a0 < 3) {
      printf("    [qui-2a00] modifications de 0x2a00..0x2a8d pendant le job SB :\n");
      long tot = 0;
      for (int b = 0; b < 64; b++)
          if (ecr[b]) {
            printf("       region 0x%04x : %ld modifs (dont %ld vers zero), PC 0x%04x a 0x%04x\n",
                   b << 10, ecr[b], vers0[b], pmin[b], pmax[b]);
              tot += ecr[b];
          }
      if (!tot) printf("        AUCUNE : rien n'ecrit jamais dans 0x2a00\n");
      printf("        par PC exact (PC imprime = instruction SUIVANTE) :\n");
```

```
      for (int t = 0; t < 12; t++)
          if (parpc2[t].pc)
              printf("           pc=%04x (donc ecrivain %04x) : %ld valeurs, %ld mises a ZERO\n",
                     parpc2[t].pc, parpc2[t].pc - 1, parpc2[t].v, parpc2[t].z);
      n_2a0++;
  } }
/* [2026-09-18] WHO WRITES THE 78 SOFT BITS, AND AT WHICH ADDRESS?
 * Sweeping the 78 coded bits shows the first half landing at position b+3
 * and the second at TWO positions 51 apart (b-38 and b+13), i.e. the two
 * data blocks OVERLAP instead of concatenating. Shadow the area and
 * record the PC as soon as its content changes. */
{ static uint16_t ombre[78]; static int arme5; static long parpc[64];
  static uint16_t pcmin[64], pcmax[64]; static long seq;
  if (dans_sb2) {
     if (!arme5) { for (int k = 0; k < 78; k++) ombre[k] = dsp->data[0x2c72 + k]; arme5 = 1; seq =
      ↪  0; }
      for (int k = 0; k < 78; k++) {
          if (dsp->data[0x2c72 + k] != ombre[k]) {
              int b = pc >> 10;
              if (!parpc[b]) { pcmin[b] = pc; pcmax[b] = pc; }
              if (pc < pcmin[b]) pcmin[b] = pc;
              if (pc > pcmax[b]) pcmax[b] = pc;
              parpc[b]++;
              /* [2026-09-18] EXACT CHRONOLOGY: sequence number, PC,
               * index touched, old and new value. Tells (a) in which
               * ORDER the two passes write, hence which destroys which,
               * and (b) whether two writes at the same index carry the
               * SAME value (mis-addressed copy) or DIFFERENT ones (two
               * halves of a sum that should have landed together). */
              if (n_ecr == 0 && seq < 100)
                  printf("    [w] %3ld pc=%04x k=%2d  %04x -> %04x  "
                         "AR0=%04x AR1=%04x AR2=%04x AR3=%04x AR4=%04x AR5=%04x AR6=%04x AR7=%04x
                          ↪  BRC=%u\n",
                         seq, pc, k, ombre[k], dsp->data[0x2c72 + k],
                         dsp->ar[0], dsp->ar[1], dsp->ar[2], dsp->ar[3],
                         dsp->ar[4], dsp->ar[5], dsp->ar[6], dsp->ar[7], dsp->brc);
              seq++;
              ombre[k] = dsp->data[0x2c72 + k];
          }
      }
  } else if (arme5 && n_ecr < 2) {
      printf("    [ecrit-2c72] sites qui modifient les 78 bits souples :\n");
      for (int b = 0; b < 64; b++)
          if (parpc[b])
              printf("        region 0x%04x : %ld ecritures, PC de 0x%04x a 0x%04x\n",
                     b << 10, parpc[b], pcmin[b], pcmax[b]);
      n_ecr++; arme5 = 0;
      for (int b = 0; b < 64; b++) parpc[b] = 0;
  } }
/* [2026-09-18] WHY THE EDGES ARE LOST. The max|delta| profile shows
 * three orders of magnitude between the middle of the burst (13452,
 * 19534, extreme values REPEATED) and its edges (2 to 30). Identical
 * extremes coming back means SATURATION. In a fixed-point MLSE the path
 * metrics MUST be renormalized at every step; without that they grow,
 * saturate, and the relative contribution of edge symbols collapses.
 * Count accumulator saturations and flag state during the SB demod. */
{ static long nsat, nova, novb; static int ovm_vu;
  if (dans_sb2) {
      int64_t a = dsp->a, b = dsp->b;
      if (a > 0x7FFFFFFFLL || a < -0x80000000LL) nsat++;
      if (b > 0x7FFFFFFFLL || b < -0x80000000LL) nsat++;
      if (dsp->st0 & 0x0400) nova++;      /* OVA */
      if (dsp->st0 & 0x0200) novb++;      /* OVB */
      if (dsp->st1 & 0x0200) ovm_vu = 1;  /* OVM: saturation mode */
  } else if ((nsat || nova || novb) && n_sat_vus < 3) {
      printf("    [saturation] demod SB : %ld depassements 32 bits,"
```

```
                    " OVA pose %ld fois, OVB %ld fois, mode OVM %s\n",
                    nsat, nova, novb, ovm_vu ? "ACTIF" : "inactif");
            n_sat_vus++; nsat = nova = novb = 0;
          } }
        /* [2026-09-18] WHERE DOES THE FIXED WINDOW ADDRESS COME FROM? Watch
         * the address registers that SWEEP the input buffer 0x0cce during the SB
         * demod and print, per register, the sample index range covered and the
         * PC that set it. If that range does NOT move by 10 when the margin goes
         * from 21 to 31, the address is computed once and reused as is. BK is
         * printed too: a mis-emulated circular block size would loop the access
         * over a fixed window of the right length with a correct base. */
        { static uint16_t amin[8], amax[8], apc[8]; static long an[8];
          static int arme4;
          if (dans_sb2) {
             if (!arme4) { for (int k = 0; k < 8; k++) { amin[k] = 0xffff; amax[k] = 0; an[k] = 0; } arme4
              ↪  = 1; }
             for (int k = 0; k < 8; k++) {
                 uint16_t v = dsp->ar[k];
                 if (v >= 0x0cce && v < 0x0cce + 380) {
                     uint16_t idx = (uint16_t)((v - 0x0cce) / 2);
                     if (idx < amin[k]) { amin[k] = idx; apc[k] = pc; }
                     if (idx > amax[k]) amax[k] = idx;
                     an[k]++;
                 }
             }
          } else if (arme4 && n_bal < 3) {
             printf("    [tampon-AR] parcours de 0x0cce pendant le demod SB (BK=%u) :\n", dsp->bk);
             for (int k = 0; k < 8; k++)
                 if (an[k])
                     printf("        AR%d : echantillons %u..%u (%u larges), %ld acces, 1er a pc=%04x\n",
                            k, amin[k], amax[k], amax[k] - amin[k] + 1, an[k], apc[k]);
             n_bal++; arme4 = 0;
          } }
        /* [2026-09-18] XC PIPELINE HAZARD. On the C54x the XC condition is
           * sampled two cycles before execution, so an instruction setting the
           * flag just before the XC is not yet visible; the core evaluates at
           * execution time. The hazard therefore only bites if the ROM places the
           * flag setter within two slots of the XC (code written for real silicon
           * normally does not). Measure the real DISTANCE instead of assuming it:
           * keep the last 4 PCs and the ST0 before each. */
          { static uint16_t ring_pc[4], ring_op[4], ring_st0[4]; static int ri;
            if ((h == 0xFD || h == 0xFF) && pc >= 0x8400 && pc < 0x8800 && n_xc_vus < 10) {
                printf("    [xc] pc=%04x op=%04x cc=%02x | 3 precedentes :", pc, o, o & 0xff);
                for (int k = 3; k >= 1; k--) {
                    int q = (ri - k) & 3;
                    printf("  %04x:%04x(ST0=%04x)", ring_pc[q], ring_op[q], ring_st0[q]);
                }
                printf(" | ST0 au XC=%04x\n", dsp->st0);
                n_xc_vus++;
           }
           ring_pc[ri] = pc; ring_op[ri] = o; ring_st0[ri] = dsp->st0;
           ri = (ri + 1) & 3;
           if (h == 0xFD || h == 0xFF) n_xc_reg[pc >> 10]++; } }
        /* [2026-09-20] DECODER ORACLE (REJEU_DECODEUR=1). Ideal softs at
         * 0x2a00 still give no CRC OK, so the fault is inside 0x9841..: ACS
         * (0x9a78), traceback (0x9aaf) or CRC (0x9887). Dump each stage's
         * output next to what libosmocoding says it should be: u[0..34] =
         * 25 info bits of sb_info + 10 parity bits (gsm0503_sch_crc10). */
        if (env_decodeur) {
            static unsigned nd;
            { static unsigned nt;
              if (pc == 0x9a7f && nt < 6) { nt++;
                  printf("  [dec] apres `sth @0x0e,B` (pc=9a7e) : T=%04x  B=%010llx (hi=%04x)  DP=%03x
↪ CPL=%d  ST0=%04x ST1=%04x  AR1=%04x\n",
                         dsp->t, (unsigned long long)(dsp->b & 0xfffffffffULL), (unsigned)((dsp->b >> 16)
↪ & 0xffff),
```

```
                            dsp->st0 & 0x1ff, !!(dsp->st1 & 0x4000), dsp->st0, dsp->st1, dsp->ar[1]); } }
                { static int cpl_prev = -1; static uint16_t pc_prev; static unsigned ncpl;
                  int cpl = !!(dsp->st1 & 0x4000);
                  if (cpl_prev >= 0 && cpl != cpl_prev && ncpl < 16) { ncpl++;
                      printf("  [cpl] CPL %d -> %d apres l'instruction pc=%04x op=%04x  (fn=%u ST1=%04x
↪  SP=%04x)\n",
                            cpl_prev, cpl, pc_prev, prog_ovly(dsp, pc_prev), fn_cur, dsp->st1, dsp->sp); }
                  cpl_prev = cpl; pc_prev = pc; }
                if (pc == 0x9866 && nd < 3) {
                    printf("  [dec] fn_demod=%u ACS termine : TRN[0..38] @0x2c82 =", g_sb_cmd_fn);
                    for (int k = 0; k < 39; k++) printf(" %04x", dsp->data[0x2c82 + k]);
                    printf("\n        metriques @0x2c00..0x2c1f =");
                    for (int k = 0; k < 32; k++) printf(" %04x", dsp->data[0x2c00 + k]);
                    printf("\n        softs @0x2a00[0..7] = %04x %04x %04x %04x %04x %04x %04x %04x  BK=%04x
↪  AR0=%04x ST1=%04x\n",
                           dsp->data[0x2a00], dsp->data[0x2a01], dsp->data[0x2a02], dsp->data[0x2a03],
                           dsp->data[0x2a04], dsp->data[0x2a05], dsp->data[0x2a06], dsp->data[0x2a07],
                           dsp->bk, dsp->ar[0], dsp->st1);
                }
                if (pc == 0x987b && nd < 3) {
                    int ri = g_n_reels ? g_reel_pour_fn[g_sb_cmd_fn & 63] : -1;
                    unsigned char u[35];
                    if (ri >= 0) { /* decode the real codeword back to u: not available, print softs only */
                        memset(u, 9, sizeof u);
                    } else {
                        uint32_t fn = g_sb_cmd_fn;
                        uint32_t t1 = fn / 1326, t2 = fn % 26, t3 = fn % 51, t3p = t3 ? (t3 - 1) / 10 : 0;
                        uint8_t sb_info[4] = {
                            (uint8_t)(((g_bsic_injecte & 0x3f) << 2) | ((t1 & 0x600) >> 9)),
                            (uint8_t)((t1 & 0x1fe) >> 1),
                            (uint8_t)(((t1 & 0x001) << 7) | ((t2 & 0x1f) << 2) | ((t3p & 0x6) >> 1)),
                            (uint8_t)(t3p & 0x1) };
                        ubit_t ub[35];
                        osmo_pbit2ubit_ext(ub, 0, sb_info, 0, 25, 1);
                        osmo_crc16gen_set_bits(&gsm0503_sch_crc10, ub, 25, ub + 25);
                        for (int k = 0; k < 35; k++) u[k] = ub[k];
                    }
                    unsigned long long um = 0, ul = 0;
                   for (int k = 0; k < 35; k++) { um = (um << 1) | u[k]; ul |= (unsigned long long)u[k] << k; }
                    printf("  [dec] traceback termine : mots @0x2c00 = %04x %04x %04x %04x | attendu u[0..34]
↪  MSB-first=0x%09llx LSB-first=0x%09llx\n",
                         dsp->data[0x2c00], dsp->data[0x2c01], dsp->data[0x2c02], dsp->data[0x2c03], um, ul);
                    printf("        u = "); for (int k = 0; k < 35; k++) printf("%d", u[k]); printf("\n");
                }
                if (pc == 0x98a2 && nd < 3) {
                    printf("  [dec] CRC : drapeau 0x2bf8=%u  A=%010llx  mots @0x2c00 = %04x %04x %04x\n",
                           dsp->data[0x2bf8], (unsigned long long)(dsp->a & 0xfffffffffULL),
                           dsp->data[0x2c00], dsp->data[0x2c01], dsp->data[0x2c02]);
                    nd++;
                }
            }
            g_op_tous[prog_ovly(dsp, pc)]++;
            int ex = c54x_run(dsp, 1);
            if (ex <= 0) break;
            done += ex;
            insn_total++;
        }
    } else {
        /* single-step around the TOA computation (0x7940..0x795c): A, B, T, 0x3fb4 */
        static int nlog;
        while (done < insns && dsp->running && !dsp->idle) {
            uint16_t pc = dsp->pc & 0xffff;
            /* The SB demod writes 78 soft bits at 0x2a00 (repack at 0x7e94) and
             * the decoder at 0x9841 reads them back. The FB correlator uses THE SAME
             * buffer. Compare both instants to prove the overwrite. */
            { static uint16_t apres_demod[8]; static int arme, np;
```

```
                  if (pc == 0x7e94 && !arme) {
                      for (int k = 0; k < 8; k++) apres_demod[k] = dsp->data[0x2a00 + k];
                      arme = 1;
                  } else if (pc == 0x9841 && arme && np < 6) {
                      int diff = 0;
                      for (int k = 0; k < 8; k++) if (dsp->data[0x2a00 + k] != apres_demod[k]) diff = 1;
                     printf("    [2a00] apres demod: %04x %04x %04x %04x | a l'entree du decodeur: %04x %04x %04x
↪  %04x  => %s\n",
                             apres_demod[0], apres_demod[1], apres_demod[2], apres_demod[3],
                             dsp->data[0x2a00], dsp->data[0x2a01], dsp->data[0x2a02], dsp->data[0x2a03],
                             diff ? "ECRASE" : "intact");
                      np++; arme = 0;
                  } }
                /* [2026-09-17] Proof that the samples carry the message: demodulate
                 * the SAME buffer 0x0cce the DSP reads, with a reference demodulator
                 * written in C. If it recovers the code word and the DSP does not, the
                 * signal is good and the fault is entirely in the emulated DSP. */
                /* Does the buffer hold EXACTLY the samples delivered for this frame,
                 * or a mix of several bursts? The midamble is identical in every SCH: a
                 * mix would reinforce it while drowning the data, which is precisely the
                 * symptom observed. */
                { static int nx;
                  if (pc == 0x9841 && nx < 4 && g_livre_niq) {
                      int ident = 0, n = g_livre_niq < 380 ? g_livre_niq : 380;
                      int prem_diff = -1;
                      for (int k = 0; k < n; k++) {
                          if ((int16_t)dsp->data[0x0cce + k] == g_livre_iq[k]) ident++;
                          else if (prem_diff < 0) prem_diff = k;
                      }
                      printf("    [tampon] identique aux echantillons livres : %d/%d mots"
                             " (1re difference au mot %d)\n", ident, n, prem_diff);
                      nx++;
                  } }
                /* [2026-09-18] At the decoder entry, measure what it actually READS,
                 * job by job: the signs of the 78 words packed at 0x2a00 against the
                 * expected code word, with the best frame searched as in [ref] (under
                 * REJEU_SCH_PARTOUT the delivered FN is not reliable). Both polarities
                 * are printed: on differential GMSK the sign convention is not known a
                 * priori. Links "the stage rewriting 0x2a00 outputs values" to "the
                 * decoder has something to work on". */
                { static int nb2;
                  if (pc == 0x9841 && nb2 < 20) {
                      int16_t sb[78]; int nz = 0, mn = 32767, mx = -32768;
                      for (int k = 0; k < 78; k++) {
                          sb[k] = (int16_t)dsp->data[0x2a00 + k];
                          if (!sb[k]) nz++;
                          if (sb[k] < mn) mn = sb[k];
                          if (sb[k] > mx) mx = sb[k];
                      }
                      int meil_f = -1, meil_s = -1, s0 = -1;
                      for (int df = -12; df <= 3; df++) {
                          long f = (long)g_livre_fn + df; if (f < 0) continue;
                          unsigned char a2[78]; cellule_code_attendu((uint32_t)f, (uint8_t)g_bsic_injecte, a2);
                          int ok = 0;
                          for (int k = 0; k < 78; k++) { int bit = sb[k] < 0 ? 1 : 0; if (bit == a2[k]) ok++; }
                          int meilleur = ok > 78 - ok ? ok : 78 - ok;
                          if (meilleur > meil_s) { meil_s = meilleur; meil_f = (int)f; }
                          if (df == 0) s0 = ok;
                      }
                      printf("    [2a00-lu] job %d : zeros=%d/78 etendue=[%d..%d] |"
                             " signes trame livree %d/78 | meilleure trame %d avec %d/78\n",
                             nb2 + 1, nz, mn, mx, s0, meil_f, meil_s);
                      nb2++;
                  } }
                { static int nr;
                  if (pc == 0x9841 && nr < 4) {
```

```
int16_t ech[2 * 190];
for (int k = 0; k < 380; k++) ech[k] = (int16_t)dsp->data[0x0cce + k];
unsigned char b148[148]; int off = 0;
unsigned char att[78];
cellule_code_attendu(g_livre_fn, (uint8_t)g_bsic_injecte, att);
if (cellule_demod_reference(ech, 190, b148, &off) == 0) {
    /* sanity of the reference demod: the midamble must come back out */
    int okm = 0;
    for (int i = 0; i < 64; i++) if (b148[42 + i] == cellule_train_sb(i)) okm++;
    /* raw differential bits: no error propagation */
    unsigned char d148[148]; int okd = 0, okdd = 0;
    if (cellule_demod_d(ech, 190, off, d148) == 0) {
        for (int i = 1; i < 64; i++) {
            int att_d = cellule_train_sb(i) ^ cellule_train_sb(i - 1);
            if (d148[42 + i] == att_d) okd++;
        }
        /* same bits, opposite polarity */
        for (int i = 1; i < 64; i++) {
            int att_d = cellule_train_sb(i) ^ cellule_train_sb(i - 1);
            if (d148[42 + i] != att_d) okdd++;
        }
    }
    /* The DATA too, as raw differential bits, against the full
     * expected burst (3 tail + 39 + midamble + 39 + 3 tail).
     * Comparing on b148 is meaningless: differential decoding
     * propagates any single error over the rest of the burst. */
    int okdat = 0, ndat = 0, meil_f = -1, meil_s = -1;
    for (int df = -12; df <= 3; df++) {
        long f = (long)g_livre_fn + df; if (f < 0) continue;
      unsigned char a2[78]; cellule_code_attendu((uint32_t)f, (uint8_t)g_bsic_injecte, a2);
        unsigned char burst[148];
        memset(burst, 0, 3);
        memcpy(burst + 3, a2, 39);
        for (int i = 0; i < 64; i++) burst[42 + i] = (unsigned char)cellule_train_sb(i);
        memcpy(burst + 106, a2 + 39, 39);
        memset(burst + 145, 0, 3);
        int ok2 = 0, n2 = 0, prev2 = 1;
        for (int i = 0; i < 148; i++) {
            int att_d = burst[i] ^ prev2; prev2 = burst[i];
            int est_donnee = (i >= 3 && i < 42) || (i >= 106 && i < 145);
            if (!est_donnee) continue;
            n2++; if (d148[i] == att_d) ok2++;
        }
        if (ok2 > meil_s) { meil_s = ok2; meil_f = (int)f; }
        if (df == 0) { okdat = ok2; ndat = n2; }
    }
    printf("    [d-brut] midambule %d/63 (inverse %d) | DONNEES trame livree %d/%d"
           " | meilleure trame %d avec %d/%d\n",
           okd, okdd, okdat, ndat, meil_f, meil_s, ndat);
    /* the midamble is IDENTICAL for every SCH frame: if it locks
     * but the data does not, the buffer holds the burst of ANOTHER
     * frame. Find out which. */
    int meilleure = -1, meilleur_score = -1;
    for (int df = -12; df <= 3; df++) {
        long f = (long)g_livre_fn + df; if (f < 0) continue;
      unsigned char a2[78]; cellule_code_attendu((uint32_t)f, (uint8_t)g_bsic_injecte, a2);
        int ok2 = 0;
        for (int i = 0; i < 39; i++) if (b148[3 + i] == a2[i]) ok2++;
        for (int i = 0; i < 39; i++) if (b148[106 + i] == a2[39 + i]) ok2++;
        if (ok2 > meilleur_score) { meilleur_score = ok2; meilleure = (int)f; }
    }
    int ok = 0;
    for (int i = 0; i < 39; i++) if (b148[3 + i] == att[i]) ok++;
    for (int i = 0; i < 39; i++) if (b148[106 + i] == att[39 + i]) ok++;
    printf("    [ref] fn livree=%u offset=%d | midambule %d/64 |"
           " mot de code de la trame livree %d/78 |"
```

```
                " MEILLEURE trame = %d avec %d/78\n",
                g_livre_fn, off, okm, ok, meilleure, meilleur_score);
        } else {
            printf("    [ref] fn=%u  demod de reference : pas de pic exploitable\n", g_livre_fn);
        }
        nr++;
    } }
/* [2026-09-17] Hunting the next ISA bug. The FB path works, so every
 * opcode it executes is valid by construction. List the opcodes only the
 * SB demodulator executes: the one family never validated, hence the pool
 * of suspects. */
{ static int dans_sb;
  if (!vu_sb) { vu_sb = calloc(65536, 1); vu_hors = calloc(65536, 1); }
  if (pc == 0x7c31) dans_sb = 1;
  if (pc == 0x9841) dans_sb = 0;
  /* [2026-09-18] Flipping ONE coded bit moves 78 positions out of 78
   * in 0x2c72: the traceback is global. On the C54x that points at CMPS
   * (0x8E/0x8F) and the TRN register. So count what really executes in
   * the SB path: the true CMPS, or the 0xE0 family (FIRS/LMS/SQDST/ABDST)
   * whose variant still carries pseudo-CMPS semantics in the core. */
  { uint16_t o = dsp->prog[pc]; uint8_t h = o >> 8;
    if (dans_sb) {
        if (h == 0x8E || h == 0x8F) n_cmps_sb++;
        if (h >= 0xE0 && h <= 0xE3) { n_e0_sb++; n_e0x_sb[h - 0xE0]++; }
        if (o == 0x8D00 || (o & 0xFF00) == 0x8D00) n_sttrn_sb++;
    } else {
        if (h == 0x8E || h == 0x8F) n_cmps_hors++;
    } }
  { uint16_t o = dsp->prog[pc];
    if (dans_sb) vu_sb[o] = 1; else vu_hors[o] = 1; }
}
/* [2026-09-17] The 78 soft bits are at 0x2c72 (NOT 0x2a00, which is
 * the 296-word FB correlator buffer). Compare the SIGN of each with the
 * bit actually transmitted: matching signs mean the demodulator is sound
 * and the fault is in the decoder; matching with a shift means the
 * correlator offset is wrong. */
{ static int nb;
  if (pc == 0x9841 && nb < 400) {
      unsigned char att[78];
      cellule_code_attendu(g_livre_fn, (uint8_t)g_bsic_injecte, att);
      /* Try the plausible conventions: polarity, halves swapped (the
       * DSP may return 39+39 in the other order), and a shift. A
       * combination clearly above chance means the demodulator is sound
       * and it is a matter of convention. */
         /* [2026-09-18] The family tested did NOT include even/odd
          * de-interleaving. The SCH convolutional code emits its 78 bits
          * in the order C(2k), C(2k+1) per trellis step, and the burst
          * stores them as two halves of 39. If the DSP stores its softs
          * in the INTERNAL order (all evens then all odds), no cyclic
          * shift, polarity or half swap recovers it: every combination
          * lands exactly at chance, which is precisely the plateau
          * observed. Add the permutation and its inverse.
          *   0 identity   1 halves swapped
          *   2 evens first   3 odds first */
         int best = -99, bestd = 0, bestv = 0;
         for (int v = 0; v < 8; v++) {
             int inv = v & 1, perm = v >> 1;
             for (int d = -8; d <= 8; d++) {
                 int ok = 0, tot = 0;
                 for (int i = 0; i < 78; i++) {
                     int src;
                     switch (perm) {
                     case 1:  src = (i + 39) % 78; break;
                     case 2:  src = (i % 2 == 0) ? (i / 2) : (39 + (i - 1) / 2); break;
                     case 3:  src = (i % 2 == 1) ? ((i - 1) / 2) : (39 + i / 2); break;
                     default: src = i; break;
```

```
                    }
                int j = src + d; if (j < 0 || j >= 78) continue;
                int16_t sv = (int16_t)dsp->data[0x2c72 + j];
                if (!sv) continue;
                tot++;
                int bit = att[i] != 0; if (inv) bit = !bit;
                if ((sv < 0) == bit) ok++;
            }
            if (tot >= 40) { int pc2 = ok * 100 / tot;
                if (pc2 > best) { best = pc2; bestd = d; bestv = v; } }
        }
    }
    int nznb = 0; for (int i = 0; i < 78; i++) if (dsp->data[0x2c72 + i]) nznb++;
    /* Is the DSP output a SLICE of the burst at some arbitrary
     * offset? Correlate the 78 signs against the differential
     * sequence of the 148 burst bits, at every shift, in both
     * polarities. */
    { unsigned char burst[148]; int prev3 = 1; unsigned char dref[148];
      memset(burst, 0, 3); memcpy(burst + 3, att, 39);
      for (int i = 0; i < 64; i++) burst[42 + i] = (unsigned char)cellule_train_sb(i);
      memcpy(burst + 106, att + 39, 39); memset(burst + 145, 0, 3);
      for (int i = 0; i < 148; i++) { dref[i] = burst[i] ^ prev3; prev3 = burst[i]; }
      int bs = -1, bo = 0, bp = 0;
      for (int o = -78; o <= 148; o++) {
          for (int pol = 0; pol < 2; pol++) {
              int ok = 0, tot = 0;
              for (int i = 0; i < 78; i++) {
                  int j = o + i; if (j < 0 || j >= 148) continue;
                  int16_t sv = (int16_t)dsp->data[0x2c72 + i];
                  if (!sv) continue;
                  tot++; int b = dref[j]; if (pol) b = !b;
                  if ((sv < 0) == b) ok++;
              }
             if (tot >= 50 && ok * 100 / tot > bs) { bs = ok * 100 / tot; bo = o; bp = pol; }
          }
      }
      printf("    [tranche] sortie DSP vs burst complet : max %d%% a l'offset %d (polarite
↪  %d)\n",
             bs, bo, bp); }
    /* [2026-09-18] Peak and agreement on the SAME line, so they can
     * be correlated frame by frame: if the frames whose peak is 21 (the
     * real burst start) agree better, peak PRECISION is still a problem
     * and the channel estimate sits beside the burst, which is enough
     * to ruin an MLSE equalizer on GMSK whose ISI spans three symbols.
     * A flat agreement whatever the peak clears the correlator. */
    /* fingerprint of the 78 soft bits, to diff two runs */
    { static int nemq; static long cible = -2;
      if (cible == -2) { const char *e = calypso_getenv("REJEU_EMPREINTE_FN");
                         cible = e ? atol(e) : -1; }
      /* [2026-09-18] THE FRAME MUST BE LOCKED. Taking whichever
       * occurrence comes first does not compare the same thing across
       * runs: the frame the SB attempt lands on depends on FB
       * scheduling, and under SCH_PARTOUT a different frame means a
       * different T1/T2/T3, hence an entirely different code word --
       * 190/380 samples differing and 38 signs out of 78 flipping,
       * i.e. chance. REJEU_EMPREINTE_FN=<n> prints frame n only. */
      if (nemq < 1 && (cible < 0 || (long)g_livre_fn == cible)) {
          /* [2026-09-18] Check the MODULATOR before blaming the
           * demodulator. Flipping coded bit 0 touches burst bit 3, so
           * alpha_3 and alpha_4 both change sign and their SUM is
           * preserved: the phase realigns after two symbols, and only
           * ~6 complex samples around index 24 should differ out of
           * 380. If all 380 differ, the modulator is at fault. */
          printf("    [empreinte-iq] fn=%u :", g_livre_fn);
          for (int k = 0; k < 380; k++) printf(" %04x", dsp->data[0x0cce + k]);
          printf("\n"); nemq++;
```

```
  } }
{ static int nemp; static long cible2 = -2;
  if (cible2 == -2) { const char *e = calypso_getenv("REJEU_EMPREINTE_FN");
                      cible2 = e ? atol(e) : -1; }
  if (nemp < 3 && (cible2 < 0 || (long)g_livre_fn == cible2)) {
      printf("    [empreinte] fn=%u (trame courante %u%s) pic=%u :", g_livre_fn, fn_cur,
             g_livre_fn == fn_cur ? "" : " DESALIGNE", dsp->data[0x2f06]);
      for (int k = 0; k < 78; k++) printf(" %04x", dsp->data[0x2c72 + k]);
      printf("\n"); nemp++;
  } }
/* [2026-09-18] Four independent decodings of the block
 * 0x84a0-0x84d8 show it is NOT the soft-bit writer but the MIDAMBLE
 * CORRELATOR: 50 lags, 64 taps of a fixed reference at 0x2cea
 * (64 = midamble length), Re output at 0x2c56 and Im at 0x2c88,
 * then |corr|^2 at 0x2be4 for the argmax. Its index is the LAG, not
 * a coded-bit rank, so there is nothing to look for there. They
 * point at 0x2a00 as the real buffer (loop 0x848d-0x849f, BRC=141
 * hence 142 iterations, `sth B,*AR6+` at 0x8497). 142 is close to
 * the 148 burst bits, hence the hypothesis "0x2a00 indexed by
 * POSITION IN THE BURST", where coded bit b is at 3+b for b<39 and
 * at 106+(b-39) beyond. */
/* [2026-09-18] SEARCH RATHER THAN GUESS. At the decoder entry,
 * sweep ALL of data memory for a zone whose SIGNS match the
 * transmitted code word, under two layouts: 78 contiguous words
 * indexed by coded-bit rank, and indexing by burst position (3+b
 * then 106+b-39). Print the best bases. If none stands clearly
 * above chance, the soft bits are not in data memory in either
 * form. */
/* [2026-09-18] INPUT / OUTPUT DISCRIMINANT for the 0x2ad5
 * candidate. Agreement does not separate the two: the derotated
 * input and the demodulator output are both indexed by burst
 * position and would answer a bit flip alike. What separates them
 * is CHANNEL dependence: a demodulator output depends on the
 * channel estimate, hence on the peak; an input representation does
 * not. So perturb the MIDAMBLE alone (which moves the peak without
 * touching the data) and see whether 0x2ad5 moves AT THE DATA
 * POSITIONS. Invariant => input. Moving => output, buffer found. */
/* [2026-09-18] THE REAL BUFFER, found in ROM: 0x2a00.
 * Two successive stages, both at 0x2a00:
 *   - equalizer output, 142 words, index = burst position - 3
 *     (`sth B,*AR6+`, AR6 init 0x2a00, BRC=0x8d hence 142 turns);
 *   - PACKED codeword, 78 contiguous words, index = coded-bit rank,
 *     produced by 0x7e65-0x7e7a: 39 copies, then
 *     `mar *+AR2(0x0040)` at 0x7e76 which SKIPS THE 64 MIDAMBLE
 *     BITS, then 39 copies. The +64 is literal in ROM.
 * The decoder confirms it: 0x984a `stm #0x2a00,AR1`, BRC=0x26, and
 * the body at 0x9a78 advances AR1 by 2 per trellis step, 39 steps =
 * 78 words.
 *
 * Zeros must NOT be discarded here: these soft bits take the values
 * 0x0000 and 0xffff, so an `if (!v) continue;` plus a `n >= 60`
 * requirement threw away half the samples and eliminated 0x2a00
 * before it could be scored. */
{ static int n2a0;
  if (n2a0 < 4) {
      int meil_c = -1, pol_c = 0, meil_b = -1, pol_b = 0;
      for (int pol = 0; pol < 2; pol++) {
          int ok = 0;
          for (int b = 0; b < 78; b++) {
              int16_t v = (int16_t)dsp->data[0x2a00 + b];
              int bit = att[b] != 0; if (pol) bit = !bit;
              if ((v < 0) == bit) ok++;
          }
          if (ok * 100 / 78 > meil_c) { meil_c = ok * 100 / 78; pol_c = pol; }
          ok = 0;
          for (int b = 0; b < 78; b++) {
```

```
            int j = (b < 39) ? b : (64 + b);
            int16_t v = (int16_t)dsp->data[0x2a00 + j];
            int bit = att[b] != 0; if (pol) bit = !bit;
            if ((v < 0) == bit) ok++;
        }
        if (ok * 100 / 78 > meil_b) { meil_b = ok * 100 / 78; pol_b = pol; }
      }
      int nz = 0, nff = 0, naut = 0;
      for (int k = 0; k < 142; k++) {
        uint16_t v = dsp->data[0x2a00 + k];
        if (v == 0) nz++; else if (v == 0xffff) nff++; else naut++;
      }
      printf("    [2a00] COMPACTE (0x2a00+b) : %d%% (polarite %d) | "
             "BRUT (saut de 64) : %d%% (polarite %d)\n",
             meil_c, pol_c, meil_b, pol_b);
      printf("    [2a00] contenu sur 142 mots : %d nuls, %d a 0xffff, %d autres |",
             nz, nff, naut);
      for (int k = 0; k < 10; k++) printf(" %04x", dsp->data[0x2a00 + k]);
      printf("\n");
      n2a0++;
  } }
{ static int n2d;
  if (n2d < 1) {
      printf("    [2ad5] pic=%u valeurs aux positions de burst 0..147 :\n      ",
             dsp->data[0x2f06]);
      for (int k = 0; k < 148; k++) {
        printf(" %04x", dsp->data[0x2ad5 + k]);
        if (k % 12 == 11) printf("\n      ");
      }
      printf("\n");
      n2d++;
  } }
{ static int nch;
  if (nch < 1) {
      struct { int sc, base, disp, pol; } top[6];
      for (int t = 0; t < 6; t++) { top[t].sc = -1; top[t].base = 0; }
      for (int base = 0; base < 0x3f00; base++)
        for (int disp = 0; disp < 2; disp++)
          for (int pol = 0; pol < 2; pol++) {
            int ok = 0, n = 0;
            for (int b = 0; b < 78; b++) {
              int j = disp ? ((b < 39) ? (3 + b) : (106 + b - 39)) : b;
              int16_t v = (int16_t)dsp->data[(base + j) & 0x3fff];
              if (!v) continue;
              n++; int bit = att[b] != 0; if (pol) bit = !bit;
              if ((v < 0) == bit) ok++;
            }
            if (n < 60) continue;
            int sc = ok * 100 / n;
            for (int t = 0; t < 6; t++)
              if (sc > top[t].sc) {
                for (int u = 5; u > t; u--) top[u] = top[u-1];
                top[t].sc = sc; top[t].base = base; top[t].disp = disp; top[t].pol = pol;
                break;
              }
          }
      printf("    [chasse] meilleure zone : base 0x%04x %s pol %d -> %d%%"
             "   (2e: 0x%04x %d%%, 3e: 0x%04x %d%%)\n",
             top[0].base, top[0].disp ? "burst" : "contigu", top[0].pol, top[0].sc,
             top[1].base, top[1].sc, top[2].base, top[2].sc);
      nch++;
  } }
{ static int n2a;
  if (n2a < 3) {
      int meil = -1, mo = 0, mp = 0;
      for (int o = -8; o <= 8; o++)
```

```
                for (int pol = 0; pol < 2; pol++) {
                    int ok = 0, n = 0;
                    for (int b = 0; b < 78; b++) {
                        int bp = (b < 39) ? (3 + b) : (106 + b - 39);
                        int j = bp + o; if (j < 0 || j >= 148) continue;
                        int16_t v = (int16_t)dsp->data[0x2a00 + j]; if (!v) continue;
                        n++; int bit = att[b] != 0; if (pol) bit = !bit;
                        if ((v < 0) == bit) ok++;
                    }
                    if (n >= 50) { int pc2 = ok * 100 / n;
                        if (pc2 > meil) { meil = pc2; mo = o; mp = pol; } }
                }
            int nz = 0; for (int k = 0; k < 148; k++) if (dsp->data[0x2a00 + k]) nz++;
            printf("    [2a00] indexe par position de burst : concordance max %d%%"
                   " (decalage %d, polarite %d) | %d/148 mots non nuls\n",
                   meil, mo, mp, nz);
            printf("    [2a00] valeurs :");
            for (int k = 0; k < 14; k++) printf(" %04x", dsp->data[0x2a00 + k]);
            printf("\n");
            n2a++;
        } }
    { static int nv;
        if (nv < 3) {
            int m1b = -1, m1o = 0, m1s = 0, m2b = -1, m2o = 0, m2s = 0;
            for (int o = 0; o < 100; o++)
                for (int sens = -1; sens <= 1; sens += 2)
                    for (int pol = 0; pol < 2; pol++) {
                        int ok1 = 0, n1 = 0, ok2 = 0, n2 = 0;
                        for (int b = 0; b < 39; b++) {
                            int j = o + sens * b; if (j < 0 || j >= 100) continue;
                            int16_t v = (int16_t)dsp->data[0x2c56 + j]; if (!v) continue;
                            n1++; int bit = att[b] != 0; if (pol) bit = !bit;
                            if ((v < 0) == bit) ok1++;
                        }
                        for (int b = 39; b < 78; b++) {
                            int j = o + sens * (b - 39); if (j < 0 || j >= 100) continue;
                            int16_t v = (int16_t)dsp->data[0x2c56 + j]; if (!v) continue;
                            n2++; int bit = att[b] != 0; if (pol) bit = !bit;
                            if ((v < 0) == bit) ok2++;
                        }
                        if (n1 >= 30) { int pc1 = ok1 * 100 / n1;
                            if (pc1 > m1b) { m1b = pc1; m1o = o; m1s = sens * (pol ? -2 : 1); } }
                        if (n2 >= 30) { int pc2 = ok2 * 100 / n2;
                            if (pc2 > m2b) { m2b = pc2; m2o = o; m2s = sens * (pol ? -2 : 1); } }
                    }
            printf("    [2c56] moitie 1 : meilleur %d%% a l'offset %d (code %d) | "
                   "moitie 2 : meilleur %d%% a l'offset %d (code %d)\n",
                   m1b, m1o, m1s, m2b, m2o, m2s);
            int nz = 0; for (int k = 0; k < 100; k++) if (dsp->data[0x2c56 + k]) nz++;
            printf("    [2c56] %d/100 mots non nuls, carte des nuls (. = nul, X = non nul) :\n
↪  ", nz);
            for (int k = 0; k < 100; k++) {
                printf("%c", dsp->data[0x2c56 + k] ? 'X' : '.');
                if (k % 50 == 49) printf("\n      ");
            }
            printf("\n");
            for (int lig = 0; lig < 10; lig++) {
                printf("      +%2d :", lig * 10);
                for (int k = lig * 10; k < lig * 10 + 10; k++) printf(" %04x", dsp->data[0x2c56 + k]);
                printf("\n");
            }
            nv++;
        } }
    printf("    [pic-conc] pic=%u concordance=%d%%\n", dsp->data[0x2f06], best);
    printf("    [bits] fn=%u  0x2c72 non-nuls=%d/78 : %04x %04x %04x %04x %04x %04x\n"
           "           concordance max = %d%% (decalage %d, polarite %d, ordre %s)\n",
```

```
            g_livre_fn, nznb, dsp->data[0x2c72], dsp->data[0x2c73], dsp->data[0x2c74],
            dsp->data[0x2c75], dsp->data[0x2c76], dsp->data[0x2c77],
            best, bestd, bestv & 1,
            (bestv >> 1) == 1 ? "moities echangees"
            : (bestv >> 1) == 2 ? "pairs d'abord"
            : (bestv >> 1) == 3 ? "impairs d'abord" : "identite");
        nb++;
  } }
/* [2026-09-18] The `sb` word returned is IDENTICAL for BSIC 7, 14 and
 * 49, while the soft bits at 0x2c72 do follow the payload: the word the
 * ARM reads back does NOT come from the decoder. Does the DSP ever write
 * the a_sch words of the R page? API at 0x0800, R_PAGE0=0x28 and
 * R_PAGE1=0x3C, a_sch at word +15. */
{ static uint16_t vu0[10], vu1[10]; static int arme3, nw;
  const uint16_t A0 = 0x800 + 0x28 + 15, A1 = 0x800 + 0x3C + 15;
  if (!arme3) {
      for (int k = 0; k < 10; k++) { vu0[k] = dsp->data[A0 + k]; vu1[k] = dsp->data[A1 + k]; }
      arme3 = 1;
  } else if (nw < 12) {
      for (int k = 0; k < 5; k++) {
          if (dsp->data[A0 + k] != vu0[k]) {
              printf("    [a_sch] page0 mot %d : %04x -> %04x  ecrit par pc=%04x\n",
                     k, vu0[k], dsp->data[A0 + k], pc);
              vu0[k] = dsp->data[A0 + k]; nw++;
          }
          if (dsp->data[A1 + k] != vu1[k]) {
              printf("    [a_sch] page1 mot %d : %04x -> %04x  ecrit par pc=%04x\n",
                     k, vu1[k], dsp->data[A1 + k], pc);
              vu1[k] = dsp->data[A1 + k]; nw++;
          }
      }
  } }
/* [2026-09-18] The magnitudes are present and the argmax is still
 * wrong. Next question: WHICH zone does it sweep? Follow the span of the
 * address registers during peak selection (0x84c8..0x84ef) and compare it
 * with the magnitude zones. */
{ static uint16_t armin[8], armax[8]; static int arme2;
  if (pc >= 0x84c8 && pc <= 0x84ef) {
      if (!arme2) { for (int k = 0; k < 8; k++) { armin[k] = 0xffff; armax[k] = 0; } arme2 = 1; }
      for (int k = 0; k < 8; k++) {
          uint16_t v = dsp->ar[k];
          if (v < armin[k]) armin[k] = v;
          if (v > armax[k]) armax[k] = v;
      }
  }
{ static int nq;
  if (pc == 0x84ef && nq < 5) {
      printf("    [balayage] registres d'adresse pendant 0x84c8-0x84ef :");
      for (int k = 0; k < 8; k++)
          if (armax[k] >= armin[k] && armax[k] > armin[k])
              printf(" AR%d=[0x%04x..0x%04x]", k, armin[k], armax[k]);
      printf("\n");
      arme2 = 0;
      printf("    [magn@84ef] plages non nulles entre 0x2b00 et 0x2d00 :\n");
      int deb = -1, fin = -1;
      for (int a = 0x2b00; a <= 0x2d00; a++) {
          int nz = (a <= 0x2cff) && dsp->data[a] != 0;
          if (nz && deb < 0) deb = a;
          if (nz) fin = a;
          if (!nz && deb >= 0) {
              /* [2026-09-18] Compare as int16_t, not uint16_t: the
               * reported "maxima" (65502, 65104, 65318) were in fact
               * -34, -432, -218, so the probe reported the value closest
               * to -1 and the rank with it was noise. The maximum in
               * absolute value is printed too: if 0x2be4 carries a SIGNED
               * correlation, |v| (or v*v) must decide, not v. The two
```

```
             * ranks side by side say which. */
            int n = fin - deb + 1;
            int16_t vmax = -32768; int imax = -1;
            int amax = -1, iamax = -1;
            for (int k = deb; k <= fin; k++) {
                int16_t v = (int16_t)dsp->data[k];
                int a = v < 0 ? -(int)v : (int)v;
                if (v > vmax) { vmax = v; imax = k - deb; }
                if (a > amax) { amax = a; iamax = k - deb; }
            }
            printf("        0x%04x-0x%04x : %3d mots, max signe=%d au rang %d,"
                   " max |v|=%d au rang %d\n",
                   deb, fin, n, vmax, imax, amax, iamax);
            deb = -1;
        }
    }
    printf("        argmax DSP 0x2f06=%u\n", dsp->data[0x2f06]);
    nq++;
  } } }
/* [2026-09-17] Where does the correlator ACTUALLY write? Snapshot
 * data RAM on entry to 0x84a1 and list what changed on exit, rather than
 * guessing the address of the energy buffer. */
{ static uint16_t *snap; static int armec, ns;
  if (pc == 0x84a1 && !armec && ns < 3) {
    if (!snap) snap = malloc(0x4000 * sizeof(uint16_t));
    memcpy(snap, dsp->data, 0x4000 * sizeof(uint16_t));
    armec = 1;
  } else if (pc == 0x7e94 && armec) {
    int deb = -1, fin = -1, nch = 0;
    printf("    [ecrit] plages modifiees par le correlateur SB :\n");
    for (int k = 0; k < 0x4000; k++) {
        int d = (dsp->data[k] != snap[k]);
        if (d && deb < 0) deb = k;
        if (d) { fin = k; nch++; }
        if (!d && deb >= 0 && k > fin + 8) {
            printf("        0x%04x-0x%04x (%d mots)  ex: %04x->%04x\n",
                   deb, fin, fin - deb + 1, snap[deb], dsp->data[deb]);
            deb = -1;
        }
    }
    if (deb >= 0) printf("        0x%04x-0x%04x\n", deb, fin);
    printf("        total %d mots changes\n", nch);
    armec = 0; ns++;
  } }
/* [2026-09-17] With a REAL SCH burst on the input, the soft bits at
 * 0x2a00 come out null. Look at the middle link: the midamble
 * correlation energy (0x2be4) and the argmax (0x2f06). */
{ static int nc;
  if (pc == 0x84ef && nc < 8) {
    /* same type fix as above: int16_t, plus |v| */
    int nzc = 0, emax = -1, iemax = -1, samax = -32768, isamax = -1;
    for (int k = 0; k < 64; k++) {
        int16_t v = (int16_t)dsp->data[0x2be4 + k];
        int a = v < 0 ? -(int)v : (int)v;
        if (v) nzc++;
        if (a > emax) { emax = a; iemax = k; }
        if (v > samax) { samax = v; isamax = k; }
    }
    int nzin = 0; for (int k = 0; k < 380; k++) if (dsp->data[0x0cce + k]) nzin++;
    printf("    [corr] entree 0x0cce non-nuls=%d/380 | 0x2be4 non-nuls=%d/64"
           " | max |v|=%d au rang %d, max signe=%d au rang %d"
           " | argmax 0x2f06=%u | 0x2be4: %04x %04x %04x %04x %04x %04x\n",
           nzin, nzc, emax, iemax, samax, isamax, dsp->data[0x2f06],
           dsp->data[0x2be4], dsp->data[0x2be5], dsp->data[0x2be6],
           dsp->data[0x2be7], dsp->data[0x2be8], dsp->data[0x2be9]);
    nc++;
```

```
            } }
          /* SB demod input: the DMA must have dropped 190 complex samples at 0x0cce */
          { static int ne;
            if (pc == 0x7c31 && ne < 6) {
                int nz = 0; for (int k = 0; k < 380; k++) if (dsp->data[0x0cce + k]) nz++;
                int z0 = 0; while (z0 < 380 && !dsp->data[0x0cce + z0]) z0++;
                printf("    [in-sb] fn=%u p51=%u  DERNIER BURST LIVRE: type=%c fn=%u n=%d\n"
                       "            0x0cce non-nuls=%d/380 zeros_tete=%d : %04x %04x %04x %04x %04x %04x\n",
                       fn_cur, fn_cur % 51, g_livre_type ? g_livre_type : '?', g_livre_fn, g_livre_n,
                       nz, z0, dsp->data[0x0cce], dsp->data[0x0ccf], dsp->data[0x0cd0],
                       dsp->data[0x0cd1], dsp->data[0x0cd2], dsp->data[0x0cd3]);
                ne++;
            } }
          switch (pc) {                       /* SB chain (RE 9.4) */
          case 0x7c31: hit_7c31++; break;     /* SB demodulator */
          /* Counters must be fed in BOTH execution paths: one that stays at zero
           * under REJEU_PROBE_TOA while the code runs reads as a measurement. */
          case 0x7d1c: hit_7d1c++; break;     /* rpt #15, the division */
          case 0x7d1d: hit_7d1d++; break;     /* subc */
          case 0x7d1e: hit_7d1e++; break;     /* the quotient */
          case 0x81e4: hit_81e4++; break;     /* the mpy that depends on T */
          case 0x8478: hit_8478++; break;     /* FIRS site 1 */
          case 0x8492: hit_8492++; break;     /* rpt of FIRS site 2 */
          case 0x8493: hit_8493++; break;     /* FIRS site 2 */
          case 0x8497: hit_8497++; break;     /* sth B -> soft bits */
          case 0x9841: hit_9841++; break;     /* SCH decoder */
          case 0x84a1: hit_84a1++; break;     /* midamble correlator */
          case 0x770a: hit_770a++; break;     /* FB correlator */
          case 0xb219: hit_b219++; break;     /* SB job (arms the DMA) */
          case 0x7a16: hit_7a16++; break;     /* fcall into the SB demod */
          default: break; }
          histo_pc[pc >> 10]++; insn_total++;
          { static uint16_t tprev, tpc, ppc; static int tn;
            if (dsp->t != tprev) { tpc = ppc; tprev = dsp->t; }
            ppc = pc;
            if (pc == 0x7947 && tn < 10 && env_probe_t) {
                printf("    [T] au mpya (0x7947) : T=%04x (%u), dernier ecrit par pc=%04x\n",
                       dsp->t, dsp->t, tpc); tn++; } }
          if (!env_probe_t && pc >= 0x7940 && pc <= 0x795c && nlog < 60) {
              printf("    [toa] pc=%04x A=%010llx B=%010llx T=%04x 3fb4=%04x AR4=%04x\n",
                     pc, (unsigned long long)(dsp->a & 0xffffffffffULL),
                     (unsigned long long)(dsp->b & 0xffffffffffULL),
                     dsp->t, dsp->data[0x3fb4], dsp->ar[4]);
              nlog++;
          }
          g_op_tous[prog_ovly(dsp, pc)]++;
          int ex = c54x_run(dsp, 1);
          if (ex <= 0) break;
          done += ex;
      }
  }
}
{   /* [2026-09-18] Raw dump of the program words around the soft-bit writer,
     * to read the ADDRESS COMPUTATION of both passes. */
  printf("  mots programme 0x8378-0x8396 (l'ecrivain 0x838d qui met 142 zeros) :\n");
  for (uint16_t a = 0x8378; a <= 0x8396; a++)
      printf("    %04x: %04x%s", a, dsp->prog[a], ((a - 0x8378) % 8 == 7) ? "\n" : "");
  printf("\n  mots programme 0x81c8-0x81e0 (les ecrivains des 142 valeurs) :\n");
  for (uint16_t a = 0x81c8; a <= 0x81e0; a++)
      printf("    %04x: %04x%s", a, dsp->prog[a], ((a - 0x81c8) % 8 == 7) ? "\n" : "");
  printf("\n");
  printf("  mots programme 0x84a0-0x84d8 (correlateur de midambule) :\n");
  for (uint16_t a = 0x84a0; a <= 0x84d8; a++)
      printf("    %04x: %04x%s", a, dsp->prog[a], ((a - 0x84a0) % 8 == 7) ? "\n" : "");
  printf("\n  mots programme 0x7cb0-0x7cc0 (site qui remet a zero) :\n");
  for (uint16_t a = 0x7cb0; a <= 0x7cc0; a++)
```

```
            printf("    %04x: %04x%s", a, dsp->prog[a], ((a - 0x7cb0) % 8 == 7) ? "\n" : "");
        printf("\n");
    }
    { const char *e = calypso_getenv("REJEU_OPCODES_TOUS");
      if (e && *e) { FILE *fo = fopen(e, "w");
          if (fo) { for (unsigned o = 0; o < 65536; o++) if (g_op_tous[o]) fprintf(fo, "%04x %lu\n", o,
           ↪  g_op_tous[o]);
                    fclose(fo); printf("  opcodes executes ecrits dans %s\n", e); } } }
    printf("\n─── bilan rejeu (deterministe) ───\n");
    {
        printf("  instructions DSP executees : %lu\n", insn_total);
        printf("  histogramme des PC (plages de 1024 mots, non vides) :\n");
        for (int b = 0; b < 64; b++)
            if (histo_pc[b])
                printf("      0x%04x-0x%04x : %10lu  (%4.1f%%)\n",
                       b << 10, ((b + 1) << 10) - 1, histo_pc[b],
                       insn_total ? 100.0 * histo_pc[b] / insn_total : 0.0);
    }
    if (vu_sb && vu_hors) {
        int n = 0;
        printf("  opcodes executes UNIQUEMENT par le demodulateur SB (suspects d'ISA,\n"
               "  jamais valides par le chemin FB qui, lui, fonctionne) :\n    ");
        int fam[256] = {0}, famt[256] = {0};
        for (int o = 0; o < 65536; o++) {
            if (vu_sb[o]) famt[o >> 8]++;
            if (vu_sb[o] && !vu_hors[o]) { fam[o >> 8]++; n++; }
        }
        printf("\n");
        for (int f = 0; f < 256; f++)
            if (fam[f]) printf("    famille %02x.. : %3d opcodes exclusifs SB / %3d executes en SB\n",
                               f, fam[f], famt[f]);
        printf("    (%d opcodes distincts exclusifs au demodulateur SB)\n", n);
    }
    /* [2026-09-19] WHERE DOES 0x01dbf46a COME FROM? The same word is returned by
     * the synthetic fixture, by the real capture in replay and by the bridge: three
     * inputs with nothing in common. A constant word looks like a CONSTANT, not a
     * computation. Sweep data space AND program space for the pattern. */
    if (calypso_getenv("REJEU_CHERCHER_MOT")) {
        unsigned long v = strtoul(calypso_getenv("REJEU_CHERCHER_MOT"), NULL, 0);
        uint16_t lo = (uint16_t)(v & 0xffff), hi = (uint16_t)(v >> 16);
        printf("  recherche de 0x%04x%04x (lo=%04x hi=%04x) :\n", hi, lo, lo, hi);
        int n = 0;
        for (unsigned a = 0; a + 1 < 0x4000; a++) {
            if (dsp->data[a] == lo && dsp->data[a+1] == hi && n < 12) {
                printf("    data[0x%04x] = %04x %04x   (lo puis hi)\n", a, lo, hi); n++; }
            if (dsp->data[a] == hi && dsp->data[a+1] == lo && n < 12) {
                printf("    data[0x%04x] = %04x %04x   (hi puis lo)\n", a, hi, lo); n++; }
        }
        for (unsigned a = 0; a + 1 < 0x10000; a++) {
            if (dsp->prog[a] == lo && dsp->prog[a+1] == hi && n < 24) {
                printf("    prog[0x%04x] = %04x %04x\n", a, lo, hi); n++; }
        }
        int nlo = 0, nhi = 0;
        for (unsigned a = 0; a < 0x4000; a++) { if (dsp->data[a]==lo) nlo++; if (dsp->data[a]==hi) nhi++; }
        printf("    occurrences isolees en data : lo(%04x) x%d, hi(%04x) x%d\n", lo, nlo, hi, nhi);
        if (!n) printf("    motif absent de la memoire : le mot est CALCULE, pas stocke\n");
    }
    if (calypso_getenv("REJEU_OPCODES")) {
        printf("  %lu decodages\n", g_n_dec);
        printf("  opcodes de l'EGALISEUR (0x8400-0x84ff), par decodage :\n");
        for (int rang = 0; rang < 16; rang++) {
            unsigned best = 0; unsigned long bv = 0;
            for (unsigned i = 0; i < 65536; i++) if (g_op_eq[i] > bv) { bv = g_op_eq[i]; best = i; }
            if (!bv) break;
            double q = g_n_dec ? (double)bv / g_n_dec : 0;
            printf("    op=%04x  %8lu   /dec = %8.2f %s\n", best, bv, q,
```

```
                (g_n_dec && bv % g_n_dec == 0) ? "  (entier)" : "  <<< NON ENTIER");
            g_op_eq[best] = 0;
        }
        printf("\n");
        printf("  opcodes du DECODEUR (0x9800-0x9bff) :\n");
        { unsigned long st_trn = 0, cmps = 0;
          for (unsigned i = 0; i < 65536; i++) {
              if ((i & 0xFF00) == 0x8D00) st_trn += g_op_dec[i];
              if ((i >> 8) == 0x8E || (i >> 8) == 0x8F) cmps += g_op_dec[i];
          }
          printf("    CMPS (0x8E/0x8F) : %lu     ST TRN (0x8D00) : %lu\n", cmps, st_trn); }
        for (int rang = 0; rang < 14; rang++) {
            unsigned best = 0; unsigned long bv = 0;
            for (unsigned i = 0; i < 65536; i++) if (g_op_dec[i] > bv) { bv = g_op_dec[i]; best = i; }
            if (!bv) break;
            printf("    op=%04x  %8lu fois\n", best, bv);
            g_op_dec[best] = 0;
        }
        printf("  opcodes du demod SB, par nombre de passages :\n");
        for (int rang = 0; rang < 26; rang++) {
            unsigned best = 0; unsigned long bv = 0;
            for (unsigned i = 0; i < 65536; i++) if (g_op_n[i] > bv) { bv = g_op_n[i]; best = i; }
            if (!bv) break;
            printf("    op=%04x  %8lu fois   vu en pc=%04x\n", best, bv, g_op_pc[best]);
            g_op_n[best] = 0;
        }
    }
    printf("  division 16 pas : rpt(7d1c)=%lu subc(7d1d)=%lu quotient(7d1e)=%lu mpy(81e4)=%lu\n",
           hit_7d1c, hit_7d1d, hit_7d1e, hit_81e4);
    printf("  sites FIRS du SB : 0x8478=%lu  0x8492(rpt)=%lu  0x8493=%lu  0x8497(sth B->softs)=%lu\n",
           hit_8478, hit_8492, hit_8493, hit_8497);
    printf("  trames jouees      : %u\n", fn_cur + 1);
    printf("  chaine SB (pas-a-pas) : job b219=%lu  fcall 7a16=%lu  demod 7c31=%lu  corr 84a1=%lu  decodeur
↪   9841=%lu  | corr FB 770a=%lu\n",
           hit_b219, hit_7a16, hit_7c31, hit_84a1, hit_9841, hit_770a);
    printf("  FB acceptes (ARM)  : %d\n", n_fb_ok);
    printf("  SB tentees / CRC KO: %d / %d\n", n_sb_try, n_sb_crcfail);
    printf("  CMPS (0x8E/0x8F) : %ld fois dans le demod SB, %ld hors | famille 0xE0-0xE3 en SB : %ld | ST TRN :
↪   %ld\n",
           n_cmps_sb, n_cmps_hors, n_e0_sb, n_sttrn_sb);
    {
        printf("  XC (0xFD/0xFF) par region de PC :");
        for (int b = 0; b < 64; b++) if (n_xc_reg[b]) printf(" 0x%04x=%ld", b << 10, n_xc_reg[b]);
        printf("\n");
        printf("  CMPS (0x8E/0x8F) par region de PC :");
        for (int b = 0; b < 64; b++)
            if (n_cmps_reg[b]) printf(" 0x%04x=%ld", b << 10, n_cmps_reg[b]);
        printf("\n");
    }
    printf("      detail : FIRS(E0)=%ld  LMS(E1)=%ld  SQDST(E2)=%ld  ABDST(E3)=%ld"
           "   <- inertes si CALYPSO_ISA_E0_FAM=1\n",
           n_e0x_sb[0], n_e0x_sb[1], n_e0x_sb[2], n_e0x_sb[3]);
    printf("  erreurs DSP signalees: %d, dont erreur 8 (anneau DMA): %d\n", n_err_dsp, n_err8);
    printf("  CRC OK: %d, dont VRAIES (BSIC injecte + T3 valide + FN = trame): %d%s\n",
           n_crc_ok, n_sb_vraies,
           cellule_sch_partout ? "   [FN non qualifiable : SCH_PARTOUT actif]" : "");
    if (verdict == 1) printf("  VERDICT : SB DECODEE  BSIC=%d FN=%u (sb=0x%08x)\n", sb_bsic, sb_fn, sb_word);
    else if (verdict == -1) printf("  VERDICT : ABANDON (le firmware a renonce)\n");
    else printf("  VERDICT : AUCUNE SB (ni decodee ni abandon) en %ld trames\n", trames);
    return verdict == 1 ? 0 : 1;
}
```

3.23 /opt/GSM/c54x_exe/src/rejouer.h

306 octets, 9 lignes → 9 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
#ifndef REJOUER_H
#define REJOUER_H
#include <stdint.h>
#include "calypso_c54x.h"
extern uint32_t g_c54x_exe_fn;
int rejouer(C54xState *dsp, uint16_t *api_ram, long trames, long insns,
            const char *iq_mode, int amp, int bsic, int verbeux);
#endif
```

3.24 /opt/GSM/c54x_exe/src/tsp_tx.c

3630 octets, 65 lignes → 65 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/* tsp_tx.c - voir tsp_tx.h. */
#include <string.h>
#include "tsp_tx.h"

/* DROM 0xa0c9..0xa0d8 : les 8 sequences d'apprentissage des bursts normaux (26 bits, 2 mots chacune),
 * telles que la ROM les copie dans son emetteur (0x85d6 : 0xa0c9 + 2 * tsc). */
static const char *const TSC_ROM[8] = {
    "00100101110000100010010111", "00101101110111100010110111", "01000011101110100100001110",
    "01000111101101000100011110", "00011010111001000001101011", "01001110101100000100111010",
    "10100111110110001010011111", "11101111000100101110111100",
};
/* DROM 0xa0d9..0xa0db : la sequence de synchronisation de l'access-burst (41 bits, 05.02 5.2.7). */
static const char SYNC_RACH[] = "01001011011111111100110011010102010001111000";
/* [2026-10-19] 05.02 5.2.5 : sequence d'apprentissage etendue du burst de synchronisation (64 bits,
 * a l'offset 42 du burst) ; 5.2.6 : le dummy burst entier (148 bits, queues comprises). */
static const char SYNC_SB[] = "10111001011000100000010000001111100101101010001010111011000011011";
static const char DUMMY_BURST[] =
 ↪  "00011111011011101100000101001001110000010010001000000011111000111000101110001011100010101110100101000110011001110011111010000
 ↪  ;

static bool motif_a(const uint8_t *flux, int pos, const char *m, int n)
{
    if (pos < 0 || pos + n > 160) return false;
    for (int i = 0; i < n; i++) if (flux[pos + i] != (uint8_t)(m[i] - '0')) return false;
    return true;
}

bool tsp_tx_lire(const uint16_t *data, struct tsp_tx_burst *b, bool *nouveau)
{
    static uint16_t prec[16];
    for (unsigned a = 0x3cbb; a + 18 <= 0x3d00; a++) {
        if (data[a] != 0x1c0a) continue;
        const uint16_t *w = &data[a + 2];
        int ok = 1;
        for (int i = 0; i < 16; i++) if ((w[i] & 0x3e) != 0x06 && (w[i] & 0x3e) != 0x04) { ok = 0; break; }
        if (!ok) continue;
        memset(b, 0, sizeof *b);
        b->adresse = (uint16_t)a;
        memcpy(b->mots, w, sizeof b->mots);
        for (int i = 0; i < 16; i++) for (int k = 0; k < 10; k++) b->flux[10 * i + k] = (w[i] >> (15 - k)) & 1;
        *nouveau = memcmp(w, prec, sizeof prec) != 0;
        memcpy(prec, w, sizeof prec);
        b->tsc = -1; b->type = TSP_TX_INCONNU; b->offset = -1;
        /* access-burst : [8 de garde][8 queue etendue][41 sync]... -> sync a 16, burst a 8 */
```

```
        if (motif_a(b->flux, 16, SYNC_RACH, 41)) { b->type = TSP_TX_RACH; b->offset = 8; }
        else {
            /* burst normal : 3 queue, 57, hl, 26 TSC, hu, 57, 3 -> TSC a offset + 61 */
            for (int t = 0; t < 8 && b->type == TSP_TX_INCONNU; t++)
                for (int p = 61; p + 26 <= 160 && p - 61 + 148 <= 160; p++)
                   if (motif_a(b->flux, p, TSC_ROM[t], 26)) { b->type = TSP_TX_NB; b->tsc = t; b->offset = p - 61;
                     ↪  break; }
            /* [2026-10-19] bursts de BTS (firmware trx) : SB (training 64 bits a +42), dummy, FB (148 zeros) */
            for (int p = 0; b->type == TSP_TX_INCONNU && p + 148 <= 160; p++) {
                if (motif_a(b->flux, p + 42, SYNC_SB, 64)) { b->type = TSP_TX_SB; b->offset = p; }
                else if (motif_a(b->flux, p, DUMMY_BURST, 148)) { b->type = TSP_TX_DUMMY; b->offset = p; }
            }
            if (b->type == TSP_TX_INCONNU) {
                int zeros = 0; for (int i = 0; i < 160; i++) zeros += !b->flux[i];
                if (zeros >= 148) { b->type = TSP_TX_FB; b->offset = 0; }
            }
        }
        if (b->offset >= 0) memcpy(b->bits, b->flux + b->offset, 148);
        else memcpy(b->bits, b->flux, 148);
        return true;
    }
    return false;
}
```

3.25 /opt/GSM/c54x_exe/src/tsp_tx.h

2467 octets, 42 lignes → 42 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * tsp_tx.h - le burst montant FINAL de la ROM, tel qu'elle le remet a l'ABB.
 *
 * [2026-10-04] Chaque trame, la ROM construit en data[0x3cbb..] le script TSP pour le TWL3025 (PROM0
 * 0xb5b8 `stm #15547,ar2`). Quand un burst part, le script contient le mot TOGBR2 = 0x1c0a, un mot de
 * puissance, puis les 16 mots BULDATA ecrits par l'emetteur 0x8605 : (10 bits de donnees) << 6 |
 * (registre BULDATA1 = 3) << 1, soit `xxxx xxxx xx00 0110`. Les 160 bits ainsi serialises sont la
 * sortie de la ROM apres sequence d'apprentissage (DROM 0xa0c9), synchro RACH (DROM 0xa0d9), queues
 * (DROM 0xa0dc), chiffrement A5 (XOR en 0x85d1) et inversion I/Q (XOR 0xaa80 en 0xb773) : ce que le
 * silicium envoie a l'antenne, au format pres. Mesure (dsp_tester rach, tx-sdcch, tx-sdcch-a5) :
 * identique a libosmocoding au bit pres.
 *
 * Deux bugs du coeur (CMPR absent, LD src,ASM sans decalage) empechaient cet emetteur de tourner en
 * emulation jusqu'au 2026-10-04 ; d'ou l'ancienne capture en 0x3f8a (bits avant modulateur).
 */
#ifndef TSP_TX_H
#define TSP_TX_H
#include <stdint.h>
#include <stdbool.h>

/* [2026-10-19] SB, DUMMY, FB : les bursts d'une BTS (firmware trx, start-native.sh --osmo-trx),
 * emis par la tache RACH que l'extcode detourne ; publies dans calypso_tx_rom pour l'air. */
enum tsp_tx_type { TSP_TX_INCONNU = 0, TSP_TX_RACH = 1, TSP_TX_NB = 2, TSP_TX_SB = 3, TSP_TX_DUMMY = 4, TSP_TX_FB
 ↪  = 5 };

struct tsp_tx_burst {
    uint16_t mots[16];      /* les 16 mots BULDATA tels quels */
    uint8_t  flux[160];     /* les 160 bits serialises (10 par mot, MSB d'abord) */
    uint8_t  bits[148];     /* le burst de 148 bits (0/1) : access-burst ou burst normal */
    int      type;          /* enum tsp_tx_type */
    int      tsc;           /* NB : sequence d'apprentissage reconnue (0..7), -1 sinon */
    int      offset;        /* position du burst dans flux[] (8 pour le RACH, sinon TSC - 61) */
    uint16_t adresse;       /* adresse du marqueur 0x1c0a dans la memoire de donnees */
};
```

```
/* Cherche le script TSP dans data[0x3cbb..0x3d00[ et y lit le burst. Rend true si un burst complet
 * (16 mots BULDATA) est present ; `nouveau` dit s'il differe des 16 mots de l'appel precedent (meme
 * processus) : c'est le signal « un burst vient d'etre emis » quand aucun pointeur d'anneau n'a bouge
 * (RACH). */
bool tsp_tx_lire(const uint16_t *data, struct tsp_tx_burst *b, bool *nouveau);

#endif
```

3.26 /opt/GSM/c54x_exe/src/verbosite.c

5631 octets, 153 lignes → 153 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * verbosite.c - the -v levels of c54x_exe.
 *
 * The C54x core (qosmo/hw/arm/calypso/l1-dsp/calypso_c54x.c) carries about a
 * hundred "[c54x] NAME ..." probes on stderr, some of them unconditional
 * (BRANCH-TRACE alone emits 700 lines at boot) [2026-09-16]. They serve QEMU
 * as much as this binary, so gating them one by one through environment
 * variables would mean touching the shared sources for every new probe.
 *
 * Instead the core is left alone: stderr is redirected into a pipe and a
 * thread re-reads the lines, passing through only those whose level is <= the
 * requested one. Classification is by keyword on the probe NAME rather than an
 * exhaustive table, so a new probe lands at a sensible level without being
 * declared. The summary reports how many lines each level hid, so the caller
 * knows what to ask for.
 */
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include "verbosite.h"

static int       g_niveau = 0;
static int       g_stderr_reel = -1;   /* dup(2) of the original */
static int       g_tube_lecture = -1;
static pthread_t g_fil;
static bool      g_actif = false;
static unsigned long g_masquees[VERBOSITE_MAX + 1];
static pthread_mutex_t g_mu = PTHREAD_MUTEX_INITIALIZER;

static bool contient(const char *l, const char *mot) { return strstr(l, mot) != NULL; }

/* Lowest -v level at which a line is shown. */
static int niveau_ligne(const char *l)
{
    /* memory probes whose NAME contains an error keyword: classify first */
    if (contient(l, "ERRWATCH"))
        return 4;
    /* 0: what breaks */
    if (contient(l, "FATAL") || contient(l, "ERR") || contient(l, "abort") ||
        contient(l, "CORRUPT") || contient(l, "TRAP") || contient(l, "manquante") ||
        contient(l, "cannot") || contient(l, "impossible") || contient(l, "a echoue"))
        return 0;
    /* 1: what is worrying */
    if (contient(l, "WARN") || contient(l, "echou") || contient(l, "failed") ||
        contient(l, "pas parque") || contient(l, "perdue"))
        return 1;
    /* 5: single-stepping */
    if (contient(l, "TRACE") || contient(l, "LOOP") || contient(l, "CYCLE") ||
```

```
        contient(l, "HIST") || contient(l, "RING") || contient(l, "-STACK"))
        return 5;
    /* 4: memory probes */
    if (contient(l, "WATCH") || contient(l, "DUMP") || contient(l, "SCAN") ||
        contient(l, "PROBE") || contient(l, "MAP") || contient(l, "SP-") ||
        contient(l, "FLOW") || contient(l, "HOT-OPS") || contient(l, "FIRST"))
        return 4;
    /* 3: tasks, API RAM, interrupts, FB/SB paths */
    if (contient(l, "TASK") || contient(l, "DISPATCH") || contient(l, "] FB") ||
        contient(l, "-FB") || contient(l, "FBDET") || contient(l, "FBWATCH") ||
        contient(l, "FBSB") || contient(l, "FBCALL") || contient(l, "FBROUTE") ||
        contient(l, "FBENTRY") || contient(l, "FBGATE") || contient(l, "FBMODE") ||
        contient(l, "SYNC") || contient(l, "FEED") || contient(l, "VEC") ||
        contient(l, "INTM") || contient(l, "IFR") || contient(l, "IMR") ||
        contient(l, "PMST") || contient(l, "MMR") || contient(l, "AFC") ||
        contient(l, "IRQ") || contient(l, "ACK") || contient(l, "IDLE") ||
        contient(l, "DMA") || contient(l, "RIF") || contient(l, "bsp") ||
        contient(l, "BSP") || contient(l, "TPU") || contient(l, "API"))
        return 3;
    /* 2: the rest - gate banners (ACTIF/INACTIVE), CALYPSO_* reads, boot,
     *    reset, [calypso-debug], and anything not coming from the core */
    return 2;
}

static void *fil_filtre(void *arg)
{
    (void)arg;
    FILE *in = fdopen(g_tube_lecture, "r");
    FILE *out = fdopen(g_stderr_reel, "w");
    if (!in || !out) {
        return NULL;
    }
    setvbuf(out, NULL, _IOLBF, 0);
    char ligne[4096];
    while (fgets(ligne, sizeof(ligne), in)) {
        int n = niveau_ligne(ligne);
        if (n <= g_niveau) {
            fputs(ligne, out);
        } else {
            pthread_mutex_lock(&g_mu);
            g_masquees[n]++;
            pthread_mutex_unlock(&g_mu);
        }
    }
    fflush(out);
    return NULL;
}

void verbosite_installer(int niveau)
{
    if (niveau < 0) niveau = 0;
    if (niveau > VERBOSITE_MAX) niveau = VERBOSITE_MAX;
    g_niveau = niveau;
    if (niveau >= VERBOSITE_MAX) {
        return;                         /* raw: nothing to install */
    }
    int tube[2];
    if (pipe(tube) < 0) {
        return;
    }
    fflush(stderr);
    g_stderr_reel = dup(STDERR_FILENO);
    g_tube_lecture = tube[0];
    dup2(tube[1], STDERR_FILENO);
    close(tube[1]);
    if (pthread_create(&g_fil, NULL, fil_filtre, NULL) != 0) {
```

```
        dup2(g_stderr_reel, STDERR_FILENO);
        return;
    }
    g_actif = true;
}

void verbosite_retirer(void)
{
    if (!g_actif) {
        return;
    }
    fflush(stderr);
    /* restore fd 2: the pipe write end closes, the thread sees EOF */
    dup2(g_stderr_reel, STDERR_FILENO);
    pthread_join(g_fil, NULL);
    g_actif = false;
}

void verbosite_bilan(FILE *out)
{
    unsigned long total = 0;
    for (int i = 0; i <= VERBOSITE_MAX; i++) total += g_masquees[i];
    if (!total) {
        return;
    }
    fprintf(out, "  traces DSP masquees : %lu  (", total);
    const char *nom[] = { "erreurs", "-v", "-vv", "-vvv", "-vvvv", "-vvvvv", "brut" };
    bool premier = true;
    for (int i = 1; i <= 5; i++) {
        if (!g_masquees[i]) continue;
        fprintf(out, "%s%lu avec %s", premier ? "" : ", ", g_masquees[i], nom[i]);
        premier = false;
    }
    fprintf(out, ")\n");
}
```

3.27 /opt/GSM/c54x_exe/src/verbosite.h

749 octets, 18 lignes → 18 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
#ifndef VERBOSITE_H
#define VERBOSITE_H
#include <stdio.h>

/* Levels: 0 = errors only (default)    -v = + warnings
 *         -vv    = + lifecycle (boot, reset, gate banner)
 *         -vvv   = + tasks, API RAM, interrupts
 *         -vvvv  = + memory probes (WATCH, DUMP, SCAN, MAP, SP-*)
 *         -vvvvv = + per-instruction traces (BRANCH, LOOP, TERM, CYCLE)
 *         -vvvvvv = everything, raw stderr, unfiltered */
#define VERBOSITE_MAX 6

void verbosite_installer(int niveau);   /* call before any DSP call           */
void verbosite_retirer(void);           /* drain the pipe, restore stderr     */
void verbosite_bilan(FILE *out);        /* count of suppressed lines, per level */

#endif
```

3.28 /opt/GSM/c54x_exe/tools/analyse_sb.py

1097 octets, 14 lignes → 14 lignes

```
#!/usr/bin/env python3
# analyse_sb.py - resume des tentatives SB d un rejeu verbeux (c54x_exe --rejouer --verbeux) :
#   SCH presents (burst dans la fenetre), taux CRC OK, angle moyen OK/KO, TOA.
# Usage : tools/analyse_sb.py rejeu.log "etiquette"
import re,sys,collections
L=open(sys.argv[1],errors='replace').read().split('\n')
att=[]
for l in L:
    m=re.search(r'SBresp att=(\d) fn=(\d+) p51=(\d+) crc=(\w+) a_sch=(\w+) (\w+) (\w+) (\w+) (\w+) toa=(-?\d+)
 ↪  pm=(-?\d+) angle=(-?\d+) snr=(-?\d+) sb_cmd_fn=(\d+)',l)
    if m: att.append(dict(att=int(m.group(1)),crc=m.group(4),toa=int(m.group(10)),angle=int(m.group(12)),snr=int
     ↪  (m.group(13))))
pres=[a for a in att if a['snr']>1000]           # burst present dans la fenetre
ok=[a for a in pres if a['crc']=='OK']; ko=[a for a in pres if a['crc']!='OK']
ma=lambda g: (sum(a['angle'] for a in g)/len(g)) if g else float('nan')
print(f"{sys.argv[2]:28s} SCH presents={len(pres):3d} OK={len(ok):3d} ({100*len(ok)/max(1,len(pres)):3.0f}%)
 ↪  angle OK={ma(ok):5.0f} KO={ma(ko):5.0f}  toa OK={collections.Counter(a['toa'] for a in ok).most_common(2)}")
```

3.29 /opt/GSM/c54x_exe/tools/cmp_codeur.py

2814 octets, 49 lignes → 49 lignes

```
"""cmp_codeur.py <trace> : codeur FR de la ROM contre libgsm, sur les memes 160 echantillons."""
import ctypes, sys, numpy as np
gsm = ctypes.CDLL("libgsm.so.1"); oc = ctypes.CDLL("/usr/local/lib/libosmocodec.so.4")
gsm.gsm_create.restype = ctypes.c_void_p
order = (ctypes.c_uint16 * 260).in_dll(oc, "gsm610_bitorder")
def bit(b, i): return (b[i >> 3] >> (7 - (i & 7))) & 1
def ti_vers_fr(words):
    ti = bytearray()
    for w in words[3:20]: ti += bytes([w >> 8, w & 0xff])
    canon = bytearray(33)
    for i in range(260):
        si = i + 4 if i >= 182 else i
        if bit(ti, si): canon[order[i] >> 3] |= 0x80 >> (order[i] & 7)
    fr = bytearray(33); fr[0] = 0xd0 | (canon[0] >> 4)
    for i in range(1, 33): fr[i] = ((canon[i-1] << 4) | (canon[i] >> 4)) & 0xff
    return bytes(fr)
def explode(g, fr):
    p = (ctypes.c_short * 76)(); gsm.gsm_explode(ctypes.c_void_p(g), fr, p); return list(p)
NOMS = ['LAR1','LAR2','LAR3','LAR4','LAR5','LAR6','LAR7','LAR8'] + [f'{n}{k}' for k in range(4) for n in
 ↪  ['Nc','bc','Mc','xmax'] + [f'x{j}' for j in range(13)]]
so, paires = None, []
for l in open(sys.argv[1]):
    if l.startswith('SO'): so = [int(v) for v in l.split()[1:]]
    elif l.startswith('ENC') and so is not None:
        a = l.split(' A ')[1].split(' B ')[0].split()
        paires.append((so, [int(w, 16) for w in a])); so = None
genc = gsm.gsm_create(); gdec = gsm.gsm_create(); gx = gsm.gsm_create()
diff = np.zeros(76); n = 0; sortie_rom, sortie_ref, entree = [], [], []
for so, w in paires:
    x = (ctypes.c_short * 160)(*[max(-32768, min(32767, v * 2)) for v in so])
    ref = (ctypes.c_ubyte * 33)(); gsm.gsm_encode(ctypes.c_void_p(genc), x, ref)
    fr = ti_vers_fr(w)
    pr, pf = explode(gx, bytes(ref)), explode(gx, fr)
    if max(abs(v) for v in so) > 200:
        diff += np.array(pr) != np.array(pf); n += 1
    o = (ctypes.c_short * 160)(); gsm.gsm_decode(ctypes.c_void_p(gdec), (ctypes.c_ubyte * 33)(*fr), o); sortie_rom
 ↪  += list(o)
    entree += [v * 2 for v in so]
print(f"{len(paires)} trames, {n} avec du signal ; fraction de trames ou le parametre ROM != libgsm :")
print('  ' + '  '.join(f"{NOMS[i]}={diff[i]/max(n,1):.2f}" for i in range(76) if i < 12 or i % 17 in (8, 9, 10,
 ↪  11)))
e = np.array(entree, float); r = np.array(sortie_rom, float)
```

```
def spec(x):
    S = np.zeros(513)
    for i in range(0, len(x) - 1024, 1024):
        s = x[i:i+1024]
        if np.sqrt((s**2).mean()) > 300: S += abs(np.fft.rfft(s * np.hanning(1024)))**2
    f = np.arange(513) * 8000 / 1024; t = S.sum() or 1
    return [round(100 * S[(f >= a) & (f < a + 1000)].sum() / t, 1) for a in (0, 1000, 2000, 3000)]
print('entree micro  energie % [0-1k,1-2k,2-3k,3-4k]', spec(e))
print('codeur ROM decode par libgsm               ', spec(r))
np.array(r, '<i2').tofile(sys.argv[1] + '.ul_rom_decode.raw')
```

3.30 /opt/GSM/c54x_exe/tools/comparer_parole.py

7942 octets, 200 lignes → 200 lignes

```
#!/usr/bin/env python3
# comparer_parole.py - la parole et la FACCH descendantes : ROM contre reference.
#
# [2026-09-23] Toutes les trames de parole que la ROM livre dans a_dd portent
# B_BFI, avec 15 a 90 erreurs rapportees par trame ; et au raccroche de 21:29
# elle a rate toutes les FACCH que le pont, lui, decodait. Ce script tranche
# entre « les bursts livres au DSP sont mauvais » et « le DSP decode mal » :
#
#   /dev/shm/calypso_tch_dl.bin   bursts TCH que le BSP a livres au DSP
#                                 (tick BE32, fn BTS BE32, 148 bits 0/1)
#   /dev/shm/calypso_add_dl.bin   trames que la ROM a rendues (montant.c,
#                                 noter_dl : parole convertie TI -> FR, FACCH)
#
# Les bursts sont dechiffres avec le Kc de la session (lignes [a5] de dsp.log)
# et le COUNT de leur propre fn, puis decodes par gsm0503_tch_fr_decode, comme
# le fait le pont. Chaque bloc est apparie a la trame que la ROM a livree juste
# apres ; pour la parole on compte les bits faux par classe (Ia 50, Ib 132,
# II 78), pour la FACCH on compare les 23 octets.
#
# Usage : tools/comparer_parole.py [--dsp-log /tmp/c54x-pont/dsp.log] [--kc HEX]
import argparse
import collections
import ctypes
import re
import struct
import sys

sys.path.insert(0, "/opt/GSM/osmo-operator")
from pont import gsm  # noqa: E402

FR = gsm.FR_BYTES
MAC = gsm.MACBLOCK_LEN


def lire_bursts(path):
    raw = open(path, "rb").read()
    out = []
    for o in range(0, len(raw) - 155, 156):
        tick, fn = struct.unpack(">II", raw[o:o + 8])
        out.append((tick, fn, bytes(raw[o + 8:o + 156])))
    return out


def lire_rom(path):
    raw = open(path, "rb").read()
    out = []
    for o in range(0, len(raw) - 47, 48):
        fn, typ, n, etat, err = struct.unpack_from("<IBBHH", raw, o)
        out.append((fn, typ, etat, err, bytes(raw[o + 12:o + 12 + n])))
```

```
    return out


def lire_kc(dsp_log):
    """(fn, Kc) de chaque nouveau Kc vu par le coprocesseur A5."""
    kcs = []
    for line in open(dsp_log, errors="replace"):
        m = re.search(r"\[a5\] #\d+ A5/\d fn=(\d+) .*Kc=([0-9a-f]{16})", line)
        if m:
            fn, kc = int(m.group(1)), bytes.fromhex(m.group(2))
            if not kcs or kcs[-1][1] != kc:
                kcs.append((fn, kc))
    return kcs


def kc_pour(kcs, fn):
    best = None
    for f, kc in kcs:
        if f <= fn + 2:
            best = kc
    return best


def classes_fr():
    """classe (0 Ia, 1 Ib, 2 II) de chaque bit du format FR standard (260 bits
    apres l'entete 0xd), d'apres gsm610_bitorder : d[k] = s[bitorder[k]]."""
    lib = ctypes.CDLL("libosmocodec.so")
    order = (ctypes.c_uint16 * 260).in_dll(lib, "gsm610_bitorder")
    cl = [0] * 260
    for k in range(260):
        cl[order[k]] = 0 if k < 50 else (1 if k < 182 else 2)
    return cl


def bits260(fr):
    v = int.from_bytes(fr[:FR], "big")
    s = bin(v)[2:].zfill(8 * FR)
    return s[4:4 + 260]


def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--dsp-log", default="/tmp/c54x-pont/dsp.log")
    ap.add_argument("--bursts", default="/dev/shm/calypso_tch_dl.bin")
    ap.add_argument("--rom", default="/dev/shm/calypso_add_dl.bin")
    ap.add_argument("--kc", help="Kc en hexa (sinon lu dans dsp.log)")
    ap.add_argument("-v", action="store_true", help="une ligne par trame")
    a = ap.parse_args()

    bursts = lire_bursts(a.bursts)
    rom = lire_rom(a.rom)
    if not bursts or not rom:
        sys.exit("rien a comparer : %d bursts, %d trames ROM" % (len(bursts), len(rom)))
    kcs = [(0, bytes.fromhex(a.kc))] if a.kc else lire_kc(a.dsp_log)
    offs = collections.Counter(t - f for t, f, _ in bursts)
    off = offs.most_common(1)[0][0]
    print("bursts : %d (fn BTS %d..%d, tick - fn = %s)" % (
        len(bursts), bursts[0][1], bursts[-1][1], dict(offs.most_common(3))))
    print("trames ROM : %d (parole %d, FACCH %d) ; Kc connus : %s" % (
        len(rom), sum(1 for r in rom if r[1] == 0), sum(1 for r in rom if r[1] == 1),
        [k.hex() for _, k in kcs]))

    # Decodage de reference : un bloc se termine sur le 4e burst d'un demi-bloc.
    par_fn = {fn: b for _, fn, b in bursts}
    blocs = {}   # fn de fin -> (rc, donnees, erreurs)
    for fn in sorted(par_fn):
```

```
    if not gsm.is_tch_carrier(fn) or gsm.tch_burst_index(fn) % 4 != 3:
        continue
    seq, f = [fn], fn
    while len(seq) < 8:
        f -= 1
        while not gsm.is_tch_carrier(f):
            f -= 1
        seq.append(f)
    seq.reverse()
    if any(x not in par_fn for x in seq):
        continue
    kc = kc_pour(kcs, fn)
    coded = []
    for x in seq:
        b = par_fn[x]
        if kc:
            ks, _ = gsm.a5_keystream(1, kc, x)
            b = gsm.a5_xor(b, ks)
        coded.append(gsm.coded_from_burst(b))
    buf = (gsm.sbit * (8 * 116)).from_buffer_copy(gsm._soft(coded))
    out = (ctypes.c_uint8 * FR)()
    ne, nb = ctypes.c_int(), ctypes.c_int()
    rc = gsm._cod.gsm0503_tch_fr_decode(out, buf, 1, 0, ctypes.byref(ne), ctypes.byref(nb))
    blocs[fn] = (rc, bytes(out), ne.value)
n_fr = sum(1 for r in blocs.values() if r[0] == FR)
n_fa = sum(1 for r in blocs.values() if r[0] == MAC)
print("reference : %d blocs, %d parole, %d FACCH, %d en echec" % (
    len(blocs), n_fr, n_fa, len(blocs) - n_fr - n_fa))

cl = classes_fr()
fins = sorted(blocs)
stats = collections.defaultdict(list)
facch = collections.Counter()
for fn_rom, typ, etat, err, data in rom:
    cible = fn_rom - off
    cand = [f for f in fins if cible - 8 <= f <= cible]
    if not cand:
        stats["sans_reference"].append(0)
        continue
    f = cand[-1]
    rc, ref, ne = blocs[f]
    if typ == 0:
        if rc != FR:
            stats["parole_ref_pas_parole"].append(rc)
            continue
        b1, b2 = bits260(ref), bits260(data)
        diff = [i for i in range(260) if b1[i] != b2[i]]
        par = [sum(1 for i in diff if cl[i] == c) for c in range(3)]
        stats["parole"].append((len(diff), par, err, ne, etat))
        if a.v:
            print("  parole fn=%d ref_fin=%d etat=%04x err_rom=%d err_ref=%d faux=%d (Ia %d, Ib %d, II %d)"
                  % (fn_rom, f, etat, err, ne, len(diff), *par))
    else:
        fire = bool(etat & 0x0040)
        if rc == MAC:
            facch["ref FACCH, ROM %s" % ("FIRE KO" if fire else
                  ("identique" if data[:MAC] == ref[:MAC] else "DIFFERENTE"))] += 1
        else:
            facch["ref pas FACCH (rc=%d), ROM %s" % (rc, "FIRE KO" if fire else "ok")] += 1
        if a.v:
            print("  facch  fn=%d ref_fin=%d etat=%04x rc_ref=%d ROM=%s REF=%s" % (
                fn_rom, f, etat, rc, data[:6].hex(" "), ref[:6].hex(" ")))

p = stats["parole"]
if p:
    tot = [x[0] for x in p]
```

```
        print("\nPAROLE : %d trames appariees" % len(p))
        print("  identiques aux bits pres : %d" % sum(1 for x in tot if x == 0))
        print("  bits faux par trame : moyenne %.1f, max %d (sur 260)" % (sum(tot) / len(tot), max(tot)))
        for c, nom in enumerate(("Ia (50)", "Ib (132)", "II (78)")):
            v = [x[1][c] for x in p]
            print("  classe %-8s : moyenne %.2f, trames touchees %d" % (nom, sum(v) / len(v), sum(1 for x in v if
             ↪  x)))
        print("  erreurs canal : ROM (a_dd[2]) moy %.1f | reference (Viterbi) moy %.1f" % (
            sum(x[2] for x in p) / len(p), sum(x[3] for x in p) / len(p)))
    for k in ("parole_ref_pas_parole", "sans_reference"):
        if stats[k]:
            print("  %s : %d" % (k, len(stats[k])))
    if facch:
        print("\nFACCH (trames a_fd vues par la ROM) :")
        for k, v in facch.most_common():
            print("  %-40s %d" % (k, v))


if __name__ == "__main__":
    main()
```

3.31 /opt/GSM/c54x_exe/tools/decoder_add.py

2430 octets, 46 lignes → 46 lignes

```
#!/usr/bin/env python3
# decoder_add.py - decode avec libgsm les trames FR notees par montant.c et cherche un ton.
#
#   /dev/shm/calypso_add_dl.bin   parole descendante rendue par la ROM (a_dd), convertie TI -> FR
#   /dev/shm/calypso_add_ul.bin   parole montante (a_du) : type 1 = brute TI, type 0 = convertie FR
#
# Enregistrements de 48 octets : fn u32, type u8, n u8, etat u16, err u16, [12..12+n] donnees.
# Le descendant a_dd est bit-exact avec ce que la BTS emet (comparer_parole.py) : s'il ne
# porte pas le ton injecte au micro, c'est le montant (ou l'echo) qui l'a perdu.
#
# Usage : tools/decoder_add.py [dl|ul] [--ton 1000] [--out fichier.raw]
import argparse, ctypes, math, struct, sys
import numpy as np
ap = argparse.ArgumentParser()
ap.add_argument("sens", nargs="?", default="dl", choices=["dl", "ul"])
ap.add_argument("--ton", type=float, default=1000.0, help="frequence cherchee (Hz)")
ap.add_argument("--out", help="PCM s16le 8 kHz decode")
a = ap.parse_args()
path = "/dev/shm/calypso_add_%s.bin" % a.sens
raw = open(path, "rb").read()
recs = []
for o in range(0, len(raw) - 47, 48):
    fn, typ, n, etat, err = struct.unpack_from("<IBBHH", raw, o)
    recs.append((fn, typ, n, etat, err, raw[o + 12:o + 12 + n]))
fr = [r for r in recs if r[1] == 0 and r[2] == 33]
print("%s : %d enregistrements, %d trames FR (fn %d..%d), nibble de tete %s" % (
    path, len(recs), len(fr), fr[0][0] if fr else 0, fr[-1][0] if fr else 0,
    sorted({hex(x[5][0] >> 4) for x in fr})))
g = ctypes.CDLL("libgsm.so.1"); g.gsm_create.restype = ctypes.c_void_p
g.gsm_decode.argtypes = [ctypes.c_void_p, ctypes.POINTER(ctypes.c_ubyte), ctypes.POINTER(ctypes.c_short)]
h = g.gsm_create()
pcm = []
for _, _, _, _, _, d in fr:
    out = (ctypes.c_short * 160)()
    g.gsm_decode(h, (ctypes.c_ubyte * 33)(*d), out)
    pcm.extend(out)
x = np.array(pcm, dtype=float); sr = 8000
if a.out:
    open(a.out, "wb").write(np.array(pcm, dtype="<i2").tobytes())
```

```
print(" sec    rms  raie(Hz)  %%dans %.0f+-50  fn" % a.ton)
for s in range(int(len(x) // sr)):
    v = x[s * sr:(s + 1) * sr]
    rms = math.sqrt(np.mean(v * v)) + 1e-9
    f = np.fft.rfftfreq(len(v), 1 / sr); sp = np.abs(np.fft.rfft(v * np.hanning(len(v)))) ** 2
    k = int(np.argmax(sp[1:])) + 1; band = (f >= a.ton - 50) & (f <= a.ton + 50)
    print("%4d %6.0f %9.0f %12.0f  %d" % (s, rms, f[k], 100 * sp[band].sum() / (sp[1:].sum() + 1e-9), fr[min(s *
     ↪  50, len(fr) - 1)][0]))
```

3.32 /opt/GSM/c54x_exe/tools/dsp_banc_commun.c

34118 octets, 825 lignes → 825 lignes (3 groupes compactés)

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * dsp_banc_commun.c - le DSP Calypso pilote comme par l'ARM osmocom-bb.
 * Voir dsp_banc_commun.h. Les sequences sont recopiees du firmware
 * (osmocom-bb src/target/firmware : calypso/dsp.c, layer1/sync.c,
 * layer1/tdma_sched.c, layer1/mframe_sched.c) et du banc (src/pont.c :
 * jouer_trame, servir ; qosmo calypso_bsp.c : bsp_ts0_livrer, sb_moduler).
 * Chaque ecart est dit a l'endroit ou il est fait.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "qemu/thread.h"
#include "dsp_banc_commun.h"
#include "calypso_dma.h"
#include "calypso_bsp.h"
#include "calypso_rhea_dma.h"
#include "calypso_rif.h"
#include "calypso_twl3025.h"
#include "calypso_gmsk.h"
#include <osmocom/core/bits.h>
#include <osmocom/coding/gsm0503_coding.h>

/* ---- ce que main.c de c54x_exe fournit d'ordinaire ------------------------ */
uint32_t g_c54x_exe_fn;
uint32_t calypso_trx_get_fn(void) { return g_c54x_exe_fn; }
void calypso_inth_arm_ack(void) { }
QemuMutex calypso_pcb_daram_lock;
void cpu_physical_memory_rw(uint64_t addr, void *buf, uint64_t len, bool wr)
{ (void)addr; if (!wr) memset(buf, 0, len); }
extern int c54x_rapide;

/* ---- etat ----------------------------------------------------------------- */
struct banc_dsp_api dsp_api;
C54xState *banc_dsp;
uint16_t *banc_api;
struct banc_temps banc_courant, banc_suivant;
struct banc_config banc_cfg = { .rom_dir = "/opt/GSM", .budget = 300000, .verbeux = 0, .afc_dac = -700 };
struct banc_stats banc_stats;
banc_source_fn banc_source = banc_cellule;
const uint8_t *(*banc_source_autre)(uint32_t fn, unsigned tn);
uint8_t banc_bsic = 7;
banc_cellule_nb_fn banc_cellule_nb;
int (*banc_crochet_l1s)(uint32_t fn);
void (*banc_crochet_fin)(uint32_t fn);
bool banc_sur_tch;
struct banc_ul_burst banc_ul[BANC_UL_MAX];
struct banc_tsp_burst banc_tsp[BANC_UL_MAX];
int banc_n_tsp;
static uint16_t g_tsp_prec[16];
```

```
int banc_n_ul;

static int g_irq_armee;            /* dsp_end_scenario() a la derniere l1_sync */
static uint16_t g_ul_prec;
static uint16_t g_ul_prec_ptr[6];   /* derniers pointeurs des anneaux TX : 0x3d91, 0x3d93, 0x3d95, 0x3d97, 0x3d99,
 ↪  0x3d9b */

/* API RAM, en mots depuis data[0x0800] (dsp_api.h : BASE_API_*) */
#define W_PAGE_MOT(p)   ((p) ? 0x14u : 0x00u)
#define R_PAGE_MOT(p)   ((p) ? 0x3Cu : 0x28u)
#define NDB_MOT         0xD4u
#define PARAM_MOT       0x431u
#define BL_ADDR_HI_W    0x7FCu     /* BL_ADDR_HI = BASE_API_RAM + 0x0ff8 */
#define BL_SIZE_W       0x7FDu
#define BL_ADDR_LO_W    0x7FEu
#define BL_STATUS_W     0x7FFu
#define BL_STATUS_IDLE  1
#define BL_CMD_COPY_BLOCK 2
#define DSP_START       0x7000

/* Registres du TWL3025 (osmocom-bb include/abb/twl3025.h ; ABB_VAL n'en garde
 * que les 5 bits bas, la page tombe). */
#define PAGE(n) ((n) << 7)
enum { R_AFCCTLADD = PAGE(1) | 21, R_APCDEL1 = PAGE(0) | 2, R_APCDEL2 = PAGE(1) | 26,
       R_APCRAM = PAGE(0) | 10, R_APCOFF = PAGE(0) | 11, R_AUXDAC = PAGE(0) | 12,
       R_BULIOFF = PAGE(1) | 2, R_BULQOFF = PAGE(1) | 3, R_BULGCAL = PAGE(1) | 14,
       R_BBCTRL = PAGE(1) | 6, R_VBCTRL1 = PAGE(1) | 8, R_VBCTRL2 = PAGE(1) | 11,
       R_VBUCTRL = PAGE(1) | 7, R_VBDCTRL = PAGE(0) | 6 };
#define ABB_RAMP_VAL(up, down)  ((((down) & 0x1F) << 5) | ((up) & 0x1F))

/* calypso/dsp_params.c du firmware, inclus tel quel : la table de parametres
 * d'un vrai telephone (ROM 3306/3606) que dsp_set_params() copie en API. */
#include "dsp_params.c"

void banc_fn2temps(struct banc_temps *t, uint32_t fn)
{
    fn %= 2715648u;
    t->fn = fn;
    t->t1 = (uint16_t)(fn / (26 * 51));
    t->t2 = (uint8_t)(fn % 26);
    t->t3 = (uint8_t)(fn % 51);
}

long banc_executer(long n)
{
    long k = 0;
    while (k < n && banc_dsp->running && !banc_dsp->idle) {
        long m = n - k;
        int r = c54x_run(banc_dsp, (int)(m > 0x40000000L ? 0x40000000L : m));
        if (r <= 0) break;
        k += r;
    }
    return k;
}

/* Un DSP a l'IDLE est reveille par une interruption pendante (SPRU131) ;
 * INT10n (fin de DMA RHEA, IMR bit 14) est une ligne de niveau. pont.c. */
static void reveil(void)
{
    C54xState *d = banc_dsp;
    if (d->idle && calypso_rhea_dma_irq_level() && (d->imr & (1u << 14)) && !(d->ifr & (1u << 14)))
        c54x_interrupt_ex(d, 30, 14);
    if (d->idle && (d->ifr & d->imr) && !(d->st1 & 0x800))
        d->idle = false;
}
```

```
static void pages(void)
{
    dsp_api.db_w = (T_DB_MCU_TO_DSP *)&banc_api[W_PAGE_MOT(dsp_api.w_page)];
    dsp_api.db_r = (T_DB_DSP_TO_MCU *)&banc_api[R_PAGE_MOT(dsp_api.r_page)];
}

int banc_init(void)
{
    /* Rien vers l'exterieur : pas de socket TRXD (un fichier de rejeu vide
     * court-circuite l'ecoute UDP de calypso_bsp_init), pas d'horloge publiee,
     * pas d'enregistreur, pas de moniteur de boite aux lettres. */
    setenv("CALYPSO_BSP_REPLAY_FILE", "/dev/null", 1);
    setenv("CALYPSO_BSP_HORLOGE", "0", 1);
    setenv("CALYPSO_REJEU_ENREG", "0", 1);
    setenv("CALYPSO_MAILBOX", "0", 1);
    /* Interruption pendante prise des que INTM retombe : comportement C54x
     * (SPRU131 ch. 6), active sur les deux bancs par main.c. */
    setenv("CALYPSO_C54X_IRQ_LEVEL", "1", 0);
    /* Le modele du coprocesseur A5 recale le COUNT des trames SACCH/TF quand il
     * vise la trame deja deposee (calypso_a5.c, CALYPSO_A5_RECALE) : c'est un
     * contournement d'un defaut de cadencement du banc. Le testeur le coupe
     * pour mesurer la ROM telle quelle (sauf si l'appelant l'a fixe). */
    setenv("CALYPSO_A5_RECALE", "0", 0);
    qemu_mutex_init(&calypso_pcb_daram_lock);
    { const char *d = getenv("CALYPSO_DEBUG"), *r = getenv("CALYPSO_C54X_RAPIDE");
      c54x_rapide = (r && *r) ? (*r != '0') : !(d && *d); }
    banc_dsp = c54x_init();
    if (!banc_dsp) { fprintf(stderr, "c54x_init a echoue\n"); return -1; }
    banc_api = &banc_dsp->data[C54X_API_BASE];
    c54x_set_api_ram(banc_dsp, banc_api);       /* AVANT les ROM (main.c) */
    static const struct { const char *s; uint32_t a; bool p; } R[] = {
        { "PROM0", 0x07000, true }, { "PROM1", 0x18000, true }, { "PROM2", 0x28000, true },
        { "PROM3", 0x38000, true }, { "DROM", 0x09000, false }, { "PDROM", 0x0E000, false },
        { "PDROM", 0x0E000, true } };
    for (unsigned i = 0; i < sizeof R / sizeof R[0]; i++) {
        char c[512];
        snprintf(c, sizeof c, "%s/calypso_dsp.%s.bin", banc_cfg.rom_dir, R[i].s);
        if (c54x_load_section(banc_dsp, c, R[i].a, R[i].p) < 0) {
            fprintf(stderr, "ROM manquante : %s\n", c);
            return -1;
        }
    }
    { char c[512]; snprintf(c, sizeof c, "%s/calypso_dsp.Registers.bin", banc_cfg.rom_dir);
      c54x_load_registers(banc_dsp, c); }
    c54x_reset(banc_dsp);
    calypso_dma_init();
    calypso_bsp_init(banc_dsp);
    calypso_twl3025_set_afc_dac(banc_cfg.afc_dac);
    dsp_api.ndb = (T_NDB_MCU_DSP *)&banc_api[NDB_MOT];
    dsp_api.param = (T_PARAM_MCU_DSP *)&banc_api[PARAM_MOT];
    dsp_api.r_page = dsp_api.w_page = dsp_api.r_page_used = 0;
    pages();
    banc_fn2temps(&banc_courant, 0);
    banc_fn2temps(&banc_suivant, 0);
    return 0;
}

/* ---- calypso/dsp.c ------------------------------------------------------- */
static void dsp_audio_init(void)
{
    T_NDB_MCU_DSP *ndb = dsp_api.ndb;
    ndb->d_vbctrl1 = ABB_VAL_T(R_VBCTRL1, 0x00B);
    ndb->d_vbctrl2 = ABB_VAL_T(R_VBCTRL2, 0x000);
    ndb->d_vbuctrl = ABB_VAL_T(R_VBUCTRL, 0x009);
    ndb->d_vbdctrl = ABB_VAL_T(R_VBDCTRL, 0x066);
    ndb->d_toneskb_init = 0;
```

```
    ndb->d_toneskb_status = 0;
    ndb->d_shiftul = 0x100;
    ndb->d_shiftdl = 0x100;
    ndb->d_melo_osc_used = 0;
    ndb->d_melo_osc_active = 0;
    ndb->a_melo_note0[0] = ndb->a_melo_note1[0] = ndb->a_melo_note2[0] = ndb->a_melo_note3[0] = 0xfffe;
    ndb->a_melo_note4[0] = ndb->a_melo_note5[0] = ndb->a_melo_note6[0] = ndb->a_melo_note7[0] = 0xfffe;
    dsp_api.param->a_fir31_downlink[0] = 0x4000;
    dsp_api.param->a_fir31_uplink[0] = 0x4000;
    for (int i = 1; i < 31; i++) {
        dsp_api.param->a_fir31_downlink[i] = 0;
        dsp_api.param->a_fir31_uplink[i] = 0;
    }
#define B_GSM_ONLY      ((1L << 13) | (1L << 11))
#define B_BT_CORDLESS   (1L << 12)
#define B_BT_HEADSET    (1L << 14)
#define B_FIR_LOOP      (1L << 1)
    ndb->d_audio_init &= ~(B_FIR_LOOP | B_GSM_ONLY | B_BT_HEADSET | B_BT_CORDLESS);
    ndb->d_audio_init |= B_GSM_ONLY;
    ndb->d_aec_ctrl = 0;
    dsp_api.param->d_gsm_bgd_mgt = 0;
    ndb->d_audio_compressor_ctrl = 0x0401;
    ndb->d_melody_selection = 0;
}

static void dsp_ndb_init(void)
{
    T_NDB_MCU_DSP *ndb = dsp_api.ndb;
#define APCDEL_DOWN (2 + 0)
#define APCDEL_UP   (6 + 3 + 1)
    for (int i = 0; i < 16; i++)
        ndb->a_ramp[i] = ABB_VAL(R_APCRAM, ABB_RAMP_VAL(0, 0));
    ndb->d_debug1 = ABB_VAL_T(0, 0x000);
    ndb->d_afcctladd = ABB_VAL_T(R_AFCCTLADD, 0x000);
    ndb->d_vbuctrl = ABB_VAL_T(R_VBUCTRL, 0x0C9);
    ndb->d_vbdctrl = ABB_VAL_T(R_VBDCTRL, 0x006);
    ndb->d_bbctrl = ABB_VAL_T(R_BBCTRL, 0x2C1);
    ndb->d_bulgcal = ABB_VAL_T(R_BULGCAL, 0x000);
    ndb->d_apcoff = ABB_VAL_T(R_APCOFF, 0x040);
    ndb->d_bulioff = ABB_VAL_T(R_BULIOFF, 0x0FF);
    ndb->d_bulqoff = ABB_VAL_T(R_BULQOFF, 0x0FF);
    ndb->d_dai_onoff = ABB_VAL_T(R_APCOFF, 0x000);
    ndb->d_auxdac = ABB_VAL_T(R_AUXDAC, 0x000);
    ndb->d_vbctrl1 = ABB_VAL_T(R_VBCTRL1, 0x00B);
    ndb->d_vbctrl2 = ABB_VAL_T(R_VBCTRL2, 0x000);
    ndb->d_apcdel1 = ABB_VAL_T(R_APCDEL1, ((APCDEL_DOWN - 2) << 5) | (APCDEL_UP - 6));
    ndb->d_apcdel2 = ABB_VAL_T(R_APCDEL2, 0x000);
    ndb->d_fb_mode = 1;
    ndb->d_fb_det = 0;
    ndb->a_cd[0] = (1 << B_FIRE1);
    ndb->a_dd_0[0] = 0;
    ndb->a_dd_0[2] = 0xffff;
    ndb->a_dd_1[0] = 0;
    ndb->a_dd_1[2] = 0xffff;
    ndb->a_du_0[0] = 0;
    ndb->a_du_0[2] = 0xffff;
    ndb->a_du_1[0] = 0;
    ndb->a_du_1[2] = 0xffff;
    ndb->a_fd[0] = (1 << B_FIRE1);
    ndb->a_fd[2] = 0xffff;
    ndb->d_a5mode = 0;
    ndb->d_tch_mode = 0x0800;
#define GUARD_BITS 8
    ndb->d_tch_mode |= (((GUARD_BITS - 4) & 0x000F) << 7);
    ndb->a_sch26[0] = (1 << B_SCH_CRC);
    ndb->d_spcx_rif = 0x179;
```

```
    dsp_audio_init();
}

int banc_dsp_power_on(struct banc_boot *b)
{
    struct banc_boot vide;
    if (!b) b = &vide;
    memset(b, 0, sizeof *b);
    memset(banc_api, 0, 0x2000u * sizeof(uint16_t));        /* dsp_api_memset(API) */

    /* dsp_pre_boot() : le DSP sort du reset et attend dans son chargeur
     * (BL_STATUS_IDLE). Ecart : le firmware y televerse d'abord dsp_bootcode
     * par le miroir 0xe000 ; ici c'est le chargeur de la ROM qui tourne. */
    long n = 0;
    while (banc_api[BL_STATUS_W] != BL_STATUS_IDLE && n < 8000000) {
        int r = c54x_run(banc_dsp, 256); if (r <= 0) break; n += r;
    }
    b->insn_chargeur = n;
    b->bl_status = banc_api[BL_STATUS_W];

    /* dsp_set_params() */
    T_NDB_MCU_DSP *ndb = dsp_api.ndb;
    ndb->d_background_enable = 0;
    ndb->d_background_abort = 0;
    ndb->d_background_state = 0;
    ndb->d_debug_ptr = 0x0074;
    ndb->d_debug_bk = 0x0001;
    ndb->d_pll_config = 0x154;
    ndb->p_debug_buffer = 0x17ff;
    ndb->d_debug_buffer_size = 7;
    ndb->d_debug_trace_type = 0;
    ndb->d_dsp_state = 3;
    ndb->d_audio_gain_ul = 0;
    ndb->d_audio_gain_dl = 0;
    ndb->d_es_level_api = 0x5213;
    ndb->d_mu_api = 0x5000;
    memcpy(dsp_api.param, &dsp_params, sizeof dsp_params);
    /* dsp_bl_start_at(DSP_START) */
    banc_api[BL_ADDR_HI_W] = 0;
    banc_api[BL_ADDR_LO_W] = DSP_START;
    banc_api[BL_SIZE_W] = 0;
    banc_api[BL_STATUS_W] = BL_CMD_COPY_BLOCK;
    n = 0;
    while (n < 4000000 && banc_dsp->running) {
        int r = c54x_run(banc_dsp, 256); if (r <= 0) break; n += r;
        if (banc_dsp->idle) break;
    }
    b->insn_demarrage = n;
    b->idle = banc_dsp->idle;
    b->version1 = ndb->d_version_number1;
    b->version2 = ndb->d_version_number2;

    dsp_ndb_init();
    /* dsp_db_init() */
    memset(&banc_api[W_PAGE_MOT(0)], 0, sizeof(T_DB_MCU_TO_DSP));
    memset(&banc_api[W_PAGE_MOT(1)], 0, sizeof(T_DB_MCU_TO_DSP));
    memset(&banc_api[R_PAGE_MOT(0)], 0, sizeof(T_DB_DSP_TO_MCU));
    memset(&banc_api[R_PAGE_MOT(1)], 0, sizeof(T_DB_DSP_TO_MCU));
    dsp_api.frame_ctr = 0;
    dsp_api.r_page = dsp_api.w_page = dsp_api.r_page_used = 0;
    /* layer1 init : l1s_reset_hw() */
    l1s_reset_hw();
    g_ul_prec = banc_dsp->data[0x3d9b];
    { static const uint16_t ptrs[6] = { 0x3d91, 0x3d93, 0x3d95, 0x3d97, 0x3d99, 0x3d9b };
      for (int i = 0; i < 6; i++) g_ul_prec_ptr[i] = banc_dsp->data[ptrs[i]]; }
    return (b->bl_status == BL_STATUS_IDLE && b->idle) ? 0 : -1;
```

```
}

void l1s_reset_hw(void)
{
    dsp_api.w_page = 0;
    dsp_api.r_page = 0;
    dsp_api.r_page_used = 0;
    pages();
    dsp_api.ndb->d_dsp_page = 0;
}

void dsp_end_scenario(void)
{
    dsp_api.ndb->d_dsp_page = B_GSM_TASK | dsp_api.w_page;
    dsp_api.w_page ^= 1;
    g_irq_armee = 1;            /* tpu_dsp_frameirq_enable() : IT trame a la trame suivante */
}

void dsp_load_rx_task(uint16_t task, uint8_t burst_id, uint8_t tsc)
{
    dsp_api.db_w->d_task_d = task;
    dsp_api.db_w->d_burst_d = burst_id;
    dsp_api.db_w->d_ctrl_system |= tsc & 0x7;
}

void dsp_load_tx_task(uint16_t task, uint8_t burst_id, uint8_t tsc)
{
    dsp_api.db_w->d_task_u = task;
    dsp_api.db_w->d_burst_u = burst_id;
    dsp_api.db_w->d_ctrl_system |= tsc & 0x7;
}

void dsp_load_tch_param(const struct banc_temps *next_time, uint8_t chan_mode, uint8_t chan_type,
                        uint8_t chan_sub, uint8_t tch_loop, uint8_t sync_tch, uint8_t tn)
{
    uint16_t d_ctrl_tch, fn;
    d_ctrl_tch = (chan_mode << B_CHAN_MODE) | (chan_type << B_CHAN_TYPE) | (chan_sub << B_SUBCHANNEL) |
                 (sync_tch << B_SYNC_TCH_UL) | (sync_tch << B_SYNC_TCH_DL) | (tch_loop << B_TCH_LOOP);
    if (chan_type == TCH_F)
        fn = (uint16_t)(((next_time->fn - (tn * 13) + 104) % 104) | ((next_time->fn % 104) << 8));
    else if (chan_type == TCH_H) {
        uint8_t tn_report = (tn & ~1) | chan_sub;
        fn = (uint16_t)(((next_time->fn - (tn_report * 13) + 104) % 104) | ((next_time->fn % 104) << 8));
    } else
        fn = 0;
    dsp_api.db_w->d_fn = fn;
    dsp_api.db_w->a_a5fn[0] = (uint16_t)(((uint16_t)next_time->t3 << 5) | next_time->t2);
    dsp_api.db_w->a_a5fn[1] = next_time->t1;
    dsp_api.db_w->d_ctrl_tch = d_ctrl_tch;
}

void dsp_load_ciph_param(int mode, const uint8_t *key)
{
    dsp_api.ndb->d_a5mode = mode;
    if (!mode || !key)
        return;
    dsp_api.ndb->a_kc[⟨1⟩] = (uint16_t)key[⟨2⟩] | ((uint16_t)key[⟨3⟩] << 8);  ×4
    ⟨⟩ = (0,7,6) (1,5,4) (2,3,2) (3,1,0)
}

void dsp_memcpy_to_api(volatile uint16_t *dsp_buf, const uint8_t *mcu_buf, int n, int be)
{
    int odd = n & 1;
    n >>= 1;
    for (int i = 0; i < n; i++) {
        uint16_t w;
```

```
        if (be) { w = (uint16_t)(*(mcu_buf++) << 8); w |= *(mcu_buf++); }
        else    { w = *(mcu_buf++); w |= (uint16_t)(*(mcu_buf++) << 8); }
        *(dsp_buf++) = w;
    }
    if (odd)
        *dsp_buf = be ? (uint16_t)(*mcu_buf << 8) : *mcu_buf;
}

void dsp_memcpy_from_api(uint8_t *mcu_buf, const volatile uint16_t *dsp_buf, int n, int be)
{
    int odd = n & 1;
    n >>= 1;
    for (int i = 0; i < n; i++) {
        uint16_t w = *(dsp_buf++);
        if (be) { *(mcu_buf++) = w >> 8; *(mcu_buf++) = (uint8_t)w; }
        else    { *(mcu_buf++) = (uint8_t)w; *(mcu_buf++) = w >> 8; }
    }
    if (odd)
        *mcu_buf = be ? (uint8_t)(*dsp_buf >> 8) : (uint8_t)*dsp_buf;
}

/* ---- layer1/tdma_sched.c ------------------------------------------------- */
#define TDMASCHED_NUM_FRAMES 25
#define TDMASCHED_NUM_CB     8
static struct { struct tdma_sched_item item[TDMASCHED_NUM_CB]; uint8_t num_items; }
 ↪  g_bucket[TDMASCHED_NUM_FRAMES];
static uint8_t g_cur_bucket;

int tdma_end_set(uint8_t p1, uint8_t p2, uint16_t p3) { (void)p1; (void)p2; (void)p3; return 0; }
static uint8_t wrap_bucket(uint8_t off) { return (uint8_t)((g_cur_bucket + off) % TDMASCHED_NUM_FRAMES); }

int tdma_schedule(uint8_t frame_offset, tdma_sched_cb *cb, uint8_t p1, uint8_t p2, uint16_t p3, int16_t prio)
{
    uint8_t nr = wrap_bucket(frame_offset);
    if (g_bucket[nr].num_items >= TDMASCHED_NUM_CB) { printf("tdma_schedule bucket overflow\n"); return -1; }
    struct tdma_sched_item *it = &g_bucket[nr].item[g_bucket[nr].num_items++];
    it->cb = cb; it->p1 = p1; it->p2 = p2; it->p3 = p3; it->prio = prio; it->flags = 0;
    return 0;
}

int tdma_schedule_set(uint8_t frame_offset, const struct tdma_sched_item *set, uint16_t p3)
{
    uint8_t nr = wrap_bucket(frame_offset);
    int j = 0;
    for (int i = 0; ; i++) {
        const struct tdma_sched_item *si = &set[i];
        if (si->cb == &tdma_end_set) break;
        if (si->cb == NULL) { nr = wrap_bucket(++frame_offset); j++; continue; }
        if (g_bucket[nr].num_items >= TDMASCHED_NUM_CB) { printf("tdma_schedule bucket overflow\n"); return -1; }
        g_bucket[nr].item[g_bucket[nr].num_items] = *si;
        g_bucket[nr].item[g_bucket[nr].num_items].p3 = p3;
        g_bucket[nr].num_items++;
    }
    return j;
}

static uint16_t tdma_sched_flag_scan(void)
{
    uint16_t f = 0;
    for (int i = 0; i < g_bucket[g_cur_bucket].num_items; i++) f |= g_bucket[g_cur_bucket].item[i].flags;
    return f;
}

static void tdma_sched_execute(void)
{
    int seq[TDMASCHED_NUM_CB];
```

```
    __typeof__(g_bucket[0]) *b = &g_bucket[g_cur_bucket];
    for (int i = 0; i < TDMASCHED_NUM_CB; i++) seq[i] = i;
    for (int i = 0; i < b->num_items; i++) {          /* _tdma_sched_bucket_sort, a l'identique */
        struct tdma_sched_item *ii = &b->item[seq[i]];
        for (int j = i + 1; j < b->num_items; j++) {
            struct tdma_sched_item *jj = &b->item[seq[j]];
            if (ii->prio > jj->prio) { ii = jj; int k = seq[i]; seq[i] = seq[j]; seq[j] = k; }
        }
    }
    for (int i = 0; i < b->num_items; i++) {
        struct tdma_sched_item *it = &b->item[seq[i]];
        if (it->cb(it->p1, it->p2, it->p3) < 0) break;
    }
    b->num_items = 0;
}

void tdma_sched_reset(void)
{
    for (int i = 0; i < TDMASCHED_NUM_FRAMES; i++)
        if (i != g_cur_bucket) g_bucket[i].num_items = 0;
}

/* ---- layer1/mframe_sched.c (SCHEDULE_AHEAD 2, SCHEDULE_LATENCY 1) -------------- */
static struct banc_mframe_item g_mf[32];
static int g_n_mf;
void banc_mframe_ajouter(const struct tdma_sched_item *set, uint16_t modulo, uint16_t frame_nr, uint16_t p3)
{
    if (g_n_mf < 32) g_mf[g_n_mf++] = (struct banc_mframe_item){ set, modulo, frame_nr, p3 };
}
void banc_mframe_vider(void) { g_n_mf = 0; }
static void mframe_schedule(void)
{
    for (int i = 0; i < g_n_mf; i++) {
        unsigned trigger = g_mf[i].frame_nr % g_mf[i].modulo;
        unsigned current = (banc_courant.fn + 2) % g_mf[i].modulo;
        if (current == trigger) tdma_schedule_set(1, g_mf[i].set, g_mf[i].p3);
    }
}

/* ---- layer1/sync.c l1_sync() ---------------------------------------------- */
static void l1_sync(void)
{
    banc_courant = banc_suivant;
    banc_fn2temps(&banc_suivant, banc_courant.fn + 1);
    dsp_api.frame_ctr++;
    dsp_api.r_page_used = 0;
    pages();
    memset((void *)dsp_api.db_w, 0, sizeof(*dsp_api.db_w));
    /* afc_load_dsp() */
    dsp_api.db_w->d_afc = (uint16_t)banc_cfg.afc_dac;
    dsp_api.db_w->d_ctrl_abb |= (1 << B_AFC);
    if (dsp_api.ndb->d_error_status) {
        banc_stats.erreurs_dsp++;
        banc_stats.derniere_erreur = dsp_api.ndb->d_error_status;
        if (banc_cfg.verbeux) printf("    [l1s] fn=%u DSP Error Status: %u\n", banc_courant.fn,
         ↪  dsp_api.ndb->d_error_status);
        dsp_api.ndb->d_error_status = 0;
    }
    uint16_t flags = tdma_sched_flag_scan();
    tdma_sched_execute();
    if (banc_crochet_l1s) flags |= (uint16_t)banc_crochet_l1s(banc_courant.fn);
    if (dsp_api.r_page_used) {
        memset((void *)dsp_api.db_r, 0, sizeof(*dsp_api.db_r));
        dsp_api.db_r->a_sch[0] = (1 << B_SCH_CRC);
        dsp_api.r_page ^= 1;
    }
```

```
    if (flags & TDMA_IFLG_DSP)
        dsp_end_scenario();
    mframe_schedule();
    g_cur_bucket = wrap_bucket(1);       /* tdma_sched_advance() */
}

/* ---- le canal radio ------------------------------------------------------- */
const uint8_t banc_tsc[8][26] = {      /* 45.002 table 5.2.3a */
    { (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),(12),(13),(14),(15),0,(16),(17),(18),(19),(20),(21),(22),(23),(24),(25) },  ×8
    () = (0,0,1,0,0,1,0,1,1,1,0,0,0,1,0,0,1,0,0,1,0,1,1,1)
         (0,0,1,0,1,1,0,1,1,1,0,1,1,1,0,0,1,0,1,1,0,1,1,1)
         (0,1,0,0,0,0,1,1,1,0,1,1,1,0,1,0,1,0,0,0,0,1,1,1,0)
         (0,1,0,0,0,1,1,1,1,0,1,1,0,1,0,0,1,0,0,0,1,1,1,1,0)
         (0,0,0,1,1,0,1,0,1,1,1,0,0,1,0,0,0,0,1,1,0,1,0,1,1)
         (0,1,0,0,1,1,1,0,1,0,1,1,0,0,0,0,1,0,0,1,1,1,0,1,0)
         (1,0,1,0,0,1,1,1,1,1,0,1,1,0,0,1,0,1,0,0,1,1,1,1,1)
         (1,1,1,0,1,1,1,1,0,0,0,1,0,0,1,1,1,1,0,1,1,1,1,0,0)
};
const uint8_t banc_train_sb[64] = {
    1,0,1,1,1,0,0,1,0,1,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,1,
    0,0,1,0,1,1,0,1,0,1,0,0,0,1,0,1,0,1,1,1,0,1,1,0,0,0,0,1,1,0,1,1,
};
const uint8_t banc_factice[148] = {    /* 45.002 5.2.6 */
    0,0,0,
    (1),(2),(3),(4),(5),(6),(7),(8),0,(9),(10),1,(11),(12),(13),(14),(15),(16),(17),(18),(19),(20),(21),(22),(23),(24),(25),(26),(
↪   27),(28),(29),(30),
↪   ×4
    () = (1,1,1,1,1,0,1,1,1,1,0,1,1,0,0,0,0,0,1,0,1,0,0,1,0,0,1,1,1,1,0)
         (0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,1,1,1,1,0,0)
         (0,1,0,1,1,1,0,0,1,0,1,1,0,0,0,1,0,0,1,0,1,1,1,0,1,0,0,1,0,1,0)
         (0,0,1,1,0,0,1,1,0,1,1,0,0,1,1,1,1,1,0,1,0,0,1,1,1,1,1,0,0,0,1)
    0,0,1,0,1,1,1,1,1,0,1,0,1,0,
    0,0,0,
};

void banc_burst_nb(const uint8_t *e116, uint8_t tsc, uint8_t bits[148])
{
    memset(bits, 0, 3);
    memcpy(bits + 3, e116, 58);
    memcpy(bits + 61, banc_tsc[tsc & 7], 26);
    memcpy(bits + 87, e116 + 58, 58);
    memset(bits + 145, 0, 3);
}

void banc_sb_info(uint32_t fn, uint8_t bsic, uint8_t sb_info[4])
{
    uint32_t t1 = fn / 1326, t2 = fn % 26, t3 = fn % 51;
    uint32_t t3p = t3 ? (t3 - 1) / 10 : 0;
    sb_info[0] = (uint8_t)(((bsic & 0x3f) << 2) | ((t1 & 0x600) >> 9));
    sb_info[1] = (uint8_t)((t1 & 0x1fe) >> 1);
    sb_info[2] = (uint8_t)(((t1 & 0x001) << 7) | ((t2 & 0x1f) << 2) | ((t3p & 0x6) >> 1));
    sb_info[3] = (uint8_t)(t3p & 0x1);
}

void banc_burst_sch(uint32_t fn, uint8_t bsic, uint8_t bits[148])
{
    uint8_t sb_info[4];
    ubit_t code[78];
    banc_sb_info(fn, bsic, sb_info);
    gsm0503_sch_encode(code, sb_info);
    memset(bits, 0, 3);
    memcpy(bits + 3, code, 39);
    memcpy(bits + 42, banc_train_sb, 64);
    memcpy(bits + 106, code + 39, 39);
    memset(bits + 145, 0, 3);
}
```

```
char banc_cellule(uint32_t fn, uint8_t bits[148], int *amp)
{
    (void)amp;
    uint32_t p51 = fn % 51;
    if (p51 % 10 == 0 && p51 <= 40) { memset(bits, 0, 148); return 'F'; }
    if (p51 % 10 == 1 && p51 <= 41) { banc_burst_sch(fn, banc_bsic, bits); return 'S'; }
    static const int debuts[] = { 2, 6, 12, 16, 22, 26, 32, 36, 42, 46 };
    for (unsigned i = 0; i < sizeof debuts / sizeof debuts[0]; i++) {
        int d = debuts[i];
        if ((int)p51 < d || (int)p51 >= d + 4) continue;
        uint32_t fn0 = fn - (p51 - (uint32_t)d);
        const uint8_t *l2 = banc_cellule_nb ? banc_cellule_nb(fn0, d) : NULL;
        if (!l2) break;
        static ubit_t e[4 * 116];
        gsm0503_xcch_encode(e, l2);
        banc_burst_nb(e + 116 * (p51 - (uint32_t)d), banc_bsic & 7, bits);
        return 'N';
    }
    memcpy(bits, banc_factice, 148);
    return 'D';
}

/* SCH : la forme d'onde du BSP (calypso_bsp.c sb_moduler, reglages par defaut :
 * elargissement 1.0, instant 0.35, bruit gaussien sigma 3000 seme par le fn,
 * forme A). Recopiee parce que statique dans calypso_bsp.c. */
static int g_sb_propre = -1;
static void sb_moduler(const uint8_t *bits, int16_t *dst, uint32_t fn, int amp)
{
    if (g_sb_propre < 0) { const char *e = getenv("BANC_SB_PROPRE"); g_sb_propre = (e && *e == '1'); }
    if (g_sb_propre) {                                  /* GMSK nue, comme un burst normal */
        gmsk_moduler(bits, 148, amp, 0.0, 0.5, dst);
        gmsk_elargir(dst, 148, 0.3);
        return;
    }
    { static double dsb = -9; if (dsb < -1) { const char *e = getenv("BANC_SB_DEC"); dsb = e ? atof(e) : 0.35; }
      gmsk_moduler(bits, 148, amp, 0.0, dsb, dst); }
    gmsk_elargir(dst, 148, 1.0);
    uint32_t seed = fn * 2654435761u + 777u;
    for (int k = 0; k < 296; k++) {
        seed = seed * 1103515245u + 12345u; double u1 = ((seed >> 8) & 0xffff) / 65536.0 + 1e-6;
        seed = seed * 1103515245u + 12345u; double u2 = ((seed >> 8) & 0xffff) / 65536.0;
        double g = sqrt(-2.0 * log(u1)) * cos(2.0 * M_PI * u2);
        double v = dst[k] + 3000.0 * g;
        if (v > 32767) v = 32767;
        if (v < -32768) v = -32768;
        dst[k] = (int16_t)lrint(v);
    }
}

static struct { uint32_t fn, tick; uint8_t bits[148]; int amp; bool valide, livre_sb; } g_sch;

/* bsp_ts0_livrer() : cadrage du burst selon la fenetre que la ROM a armee. */
static void deposer(uint32_t fn)
{
    static int16_t iq[2 * 512], iq2[2 * 148];
    uint8_t bits[148];
    int amp = 0;
    char type = banc_source ? banc_source(fn, bits, &amp) : 0;
    if (amp <= 0) amp = 30000;
    const bool one_shot = calypso_rhea_dma_one_shot();
    const int nwin = one_shot ? calypso_rhea_dma_get_len_words() / 2 : 0;
    int marge = nwin >= 190 ? 21 : nwin >= 150 ? 3 : 0;
    /* BANC_FB_DECAL=k : en recherche FB (DMA continue), le FCCH est depose k
     * echantillons apres l'ouverture de la fenetre (mesure TOA = f(k)). */
    static int fb_decal = -1;
```

```
    if (fb_decal < 0) { const char *e = getenv("BANC_FB_DECAL"); fb_decal = e ? atoi(e) : 0; }
    if (!one_shot && type == 'F') marge = fb_decal;
    /* BANC_SB_DECAL=k : en fenetre SB (one-shot), le SCH est depose k echantillons
     * apres l'ouverture de la fenetre (defaut 21). */
    static int sb_decal = -2;
    if (sb_decal == -2) { const char *e = getenv("BANC_SB_DECAL"); sb_decal = e ? atoi(e) : -1; }
    if (type == 'S' && one_shot && nwin >= 190 && sb_decal >= 0) marge = sb_decal;
    static int sb_neg = 0; { const char *e = getenv("BANC_SB_NEG"); sb_neg = e ? atoi(e) : 0; }
    if (type == 'S' && one_shot && nwin >= 190 && sb_neg > 0) marge = 0;
    /* BANC_NB_DECAL=k : burst normal (fenetre one-shot de 150..189 echantillons), defaut 3. */
    static int nb_decal = -2;
    if (nb_decal == -2) { const char *e = getenv("BANC_NB_DECAL"); nb_decal = e ? atoi(e) : -1; }
    if (type != 'S' && type != 'F' && type && one_shot && nwin >= 150 && nwin < 190 && nb_decal >= 0) marge =
    ↪  nb_decal;
    memset(iq, 0, sizeof iq);
    if (type == 'S') {
        g_sch.fn = fn; g_sch.tick = fn; g_sch.amp = amp;
        memcpy(g_sch.bits, bits, 148);
        g_sch.livre_sb = one_shot && nwin >= 190;
        g_sch.valide = true;
        if (g_sch.livre_sb) banc_stats.depots_fenetre_sb++;
        { static int cel = -1; if (cel < 0) cel = getenv("BANC_SB_CELLULE") ? 1 : 0;
          extern char cellule_burst(uint32_t, uint8_t, int, double, int, int16_t *, int *) __attribute__((weak));
          if (cel && cellule_burst) {   /* meme forme d'onde que le pont : src/cellule.c cellule_burst() */
              static int16_t tmp[2 * 512]; int n_iq = 0;
              const char *ed = getenv("BANC_SB_DEC");
              cellule_burst(fn, banc_bsic, amp, ed ? atof(ed) : 0.5, marge, tmp, &n_iq);
              memcpy(iq, tmp, (size_t)(n_iq < 2 * 256 ? n_iq : 2 * 256) * sizeof(int16_t));
          } else {
              sb_moduler(bits, iq + 2 * marge, fn, amp);
             if (sb_neg > 0) { memmove(iq, iq + 2 * sb_neg, (size_t)(2 * (512 - sb_neg)) * sizeof(int16_t)); marge
              ↪  = 1; }
          } }
    } else if (type) {
        /* BANC_NB_NEG=n : burst normal en avance de n echantillons (debut coupe) */
       static int nb_neg = -1; if (nb_neg < 0) { const char *e = getenv("BANC_NB_NEG"); nb_neg = e ? atoi(e) : 0; }
        const bool nb_win = type != 'F' && one_shot && nwin >= 150 && nwin < 190;
        if (nb_neg > 0 && nb_win) marge = 0;
        gmsk_moduler(bits, 148, amp, 0.0, 0.5, iq + 2 * marge);
        if (type != 'F') gmsk_elargir(iq + 2 * marge, 148, 0.3);
        if (nb_neg > 0 && nb_win) { memmove(iq, iq + 2 * nb_neg, (size_t)(2 * (512 - nb_neg)) * sizeof(int16_t));
         ↪  marge = 1; }
    }
    if (type == 'S' && getenv("BANC_NWIN_DBG")) { static int n; if (n++ < 3) { FILE *f =
    ↪  fopen(getenv("BANC_NWIN_DBG"), "a"); if (f) { fprintf(f, "[nwin] SB one_shot=%d nwin=%d marge=%d
    ↪  len_words=%d\n", one_shot, nwin, marge, calypso_rhea_dma_get_len_words()); fclose(f); } } }
    int total = marge > 0 ? (nwin > marge + 148 ? nwin : marge + 148) : 148;
    if (total > 256) total = 256;
    calypso_bsp_rx_burst(0, fn, iq, 2 * total);
    banc_stats.depots++;
    if (!one_shot) {            /* DMA continue (recherche FB) : la trame entiere */
        for (unsigned tn = 1; tn < 8; tn++) {
            const uint8_t *b = banc_source_autre ? banc_source_autre(fn, tn) : NULL;
            if (!b) b = banc_factice;
            memset(iq2, 0, sizeof iq2);
            gmsk_moduler(b, 148, 30000, 0.0, 0.5, iq2);
            gmsk_elargir(iq2, 148, 0.3);
            calypso_bsp_rx_burst((uint8_t)tn, fn, iq2, 2 * 148);
        }
    }
}

/* calypso_bsp_sb_retenter() : un SCH depose avant l'armement de la fenetre SB
 * est relivre dans la fenetre one-shot qui suit, au plus un tick apres. */
static void sb_retenter(uint32_t tick)
{
```

```
    static int16_t iq[2 * 512];
    if (!g_sch.valide || g_sch.livre_sb) return;
    if ((int32_t)(tick - g_sch.tick) > 1) { g_sch.valide = false; return; }
    if (!calypso_rhea_dma_rx_armed() || !calypso_rhea_dma_one_shot()) return;
    int total = calypso_rhea_dma_get_len_words() / 2;
    if (total < 190) return;
    if (total > 512) total = 512;
    int marge = 21;
    { const char *e = getenv("BANC_SB_DECAL"); if (e && atoi(e) >= 0) marge = atoi(e); }
    if (marge > total - 148) marge = total - 148;
    memset(iq, 0, sizeof iq);
    sb_moduler(g_sch.bits, iq + 2 * marge, g_sch.fn, g_sch.amp);
    calypso_rif_flush();
    calypso_bsp_rx_burst(0, g_sch.fn, iq, 2 * total);
    g_sch.livre_sb = true;
    banc_stats.depots_fenetre_sb++;
}

/* ---- une trame (src/pont.c : jouer_trame en deux phases + servir) -------------- */
uint32_t banc_fn(void) { return banc_suivant.fn; }
void banc_set_fn(uint32_t fn) { banc_fn2temps(&banc_suivant, fn); banc_fn2temps(&banc_courant, fn ? fn - 1 : 0); }

void banc_trame(void)
{
    C54xState *d = banc_dsp;
    const uint32_t fn = banc_suivant.fn;    /* la l1_sync de cette trame aura current_time = fn */
    const long budget = banc_cfg.budget;
    uint32_t i0 = d->insn_count;
    long fait = 0;
    g_c54x_exe_fn = fn;
    banc_stats.trames++;

    /* phase A : IT trame (si l'ARM a clos un scenario), la ROM lit la page W
     * et arme sa fenetre RX */
    calypso_dma_tick(d);
    reveil();
    if (g_irq_armee && (d->imr & (1u << C54X_IT_TPU_FRAME_BIT))) {
        c54x_interrupt_ex(d, C54X_IT_TPU_FRAME_VEC, C54X_IT_TPU_FRAME_BIT);
        banc_stats.irq_trame++;
    }
    g_irq_armee = 0;
    if (d->idle && (d->ifr & d->imr) && !(d->st1 & 0x800)) d->idle = false;
    if (!d->idle) fait = banc_executer(budget / 8);
    while (!d->idle && d->running && (banc_sur_tch || !calypso_rhea_dma_rx_armed()) && fait < budget / 2)
        fait += banc_executer(256);

    /* l'ARM : l1_sync() de la trame fn ; le DSP lira sa page a la trame suivante */
    l1_sync();
    calypso_twl3025_set_afc_dac(banc_cfg.afc_dac);   /* relais de d_afc (pont.c, TICK) */

    /* phase B : le burst de la trame, puis le reste du budget */
    deposer(fn);
    reveil();
    if (!d->idle && budget - fait > 0) banc_executer(budget - fait);
    for (int k = 0; k < 40 && d->running; k++) {
        sb_retenter(fn);
        if (!calypso_rhea_dma_pump(d)) break;
        if (d->idle && (d->ifr & d->imr) && !(d->st1 & 0x800)) d->idle = false;
        if (!d->idle) banc_executer(budget / 4);
    }
    if (!d->idle) banc_stats.pas_idle_fin++;
    long n = (long)(d->insn_count - i0);
    if (n > banc_stats.insn_max) banc_stats.insn_max = n;
    banc_ul_capturer(fn);
    if (banc_crochet_fin) banc_crochet_fin(fn);
}
```

```c
void banc_courir(uint32_t n)
{
    for (uint32_t i = 0; i < n; i++) banc_trame();
}

/* ---- bursts montants de la ROM (pont.c tx_rom_publier) -------------------------- */
static void banc_tsp_capturer(uint32_t fn)
{
    const uint16_t *m = banc_dsp->data;
    for (unsigned a = 0x3cbb; a < 0x3d00 - 18; a++) {
        if (m[a] != 0x1c0a) continue;
        const uint16_t *w = &m[a + 2];
        int ok = 1;
        for (int i = 0; i < 16; i++) if ((w[i] & 0x3e) != 0x06 && (w[i] & 0x3e) != 0x04) { ok = 0; break; }
        if (!ok) continue;
        if (!memcmp(w, g_tsp_prec, sizeof g_tsp_prec)) return;
        memcpy(g_tsp_prec, w, sizeof g_tsp_prec);
        struct banc_tsp_burst *t = &banc_tsp[banc_n_tsp % BANC_UL_MAX];
        t->fn = fn; memcpy(t->mots, w, sizeof t->mots);
        banc_n_tsp++;
        return;
    }
}

void banc_ul_capturer(uint32_t fn)
{
    banc_tsp_capturer(fn);
    const uint16_t *m = banc_dsp->data;
    static const uint16_t ptrs[6] = { 0x3d91, 0x3d93, 0x3d95, 0x3d97, 0x3d99, 0x3d9b };
    int bouge = 0;
    for (int i = 0; i < 6; i++) if (m[ptrs[i]] != g_ul_prec_ptr[i]) { g_ul_prec_ptr[i] = m[ptrs[i]]; bouge = 1; }
    if (!bouge) return;
    uint16_t p = m[0x3d9b];
    g_ul_prec = p;
    struct banc_ul_burst *u = &banc_ul[banc_n_ul % BANC_UL_MAX];
    u->fn = fn;
    u->ptr = p;
    u->mode = m[0x3fac];
    for (int i = 0; i < 8; i++) u->raw[i] = m[0x3f8a + i];
    u->rang = (u->mode == 3 && p >= 0x4280 && p < 0x42a0 && (p - 0x4280) % 8 == 0) ? (int)(((p - 0x4280) / 8 + 3) &
↪   3) : -1;
    for (int i = 0; i < 116; i++) {
        /* 57 donnees, hl, hu, 57 donnees ; hl/hu = 2 bits BAS du mot 7 (k = 126, 127), comme pont.c tx_rom_publier
         * (revue 2026-10-03) -- le banc gardait 114/115 et comptait 0/24 bursts justes. */
        int k = i < 57 ? i : (i < 59 ? 126 + (i - 57) : i - 2);
        u->bits[i] = (m[0x3f8a + k / 16] >> (15 - k % 16)) & 1;
        u->flux[i] = (m[0x3f9b + k / 16] >> (15 - k % 16)) & 1;
    }
    banc_n_ul++;
}

void banc_trame_vide(void)
{
    banc_source_fn s = banc_source;
    banc_source = NULL;
    banc_trame();
    banc_source = s;
}
```

3.33 /opt/GSM/c54x_exe/tools/dsp_banc_commun.h

9772 octets, 191 lignes → 191 lignes

```c
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * dsp_banc_commun.h - faire tourner le DSP Calypso (mask-ROM TI) comme le fait
 * l'ARM osmocom-bb, sans QEMU ni ARM. Module commun, reutilisable par les bancs
 * hors ligne (tools/dsp_tester.c en est le premier client).
 *
 * Ce que le module fournit :
 *   - le coeur C54x + la ROM + le DMA + le BSP, compiles depuis qosmo ;
 *   - dsp_power_on() du firmware (calypso/dsp.c) : chargeur, parametres
 *     (dsp_params.c du firmware, inclus tel quel), NDB, pages ;
 *   - les primitives du firmware : dsp_end_scenario, dsp_load_rx_task,
 *     dsp_load_tx_task, dsp_load_tch_param, dsp_load_ciph_param,
 *     dsp_memcpy_to/from_api, l1s_reset_hw ;
 *   - l'ordonnanceur TDMA du firmware (layer1/tdma_sched.c, recopie) et un
 *     ordonnanceur de multitrame minimal (layer1/mframe_sched.c : SCHEDULE_AHEAD
 *     2, SCHEDULE_LATENCY 1) ;
 *   - la trame du banc, dans l'ordre de src/pont.c (jouer_trame + servir) :
 *     IT trame si l'ARM a clos un scenario, phase A jusqu'a l'armement de la
 *     fenetre RX (jusqu'a l'IDLE sur TCH), l1_sync() de l'ARM, depot du burst
 *     de la trame (bits -> GMSK, cadre comme bsp_ts0_livrer), reste du budget,
 *     pompe DMA ;
 *   - la capture des bursts montants que la ROM emet (data[0x3f8a], pointeur
 *     data[0x3d9b], voir README « ROM and uplink probes »).
 *
 * Les structures de l'API RAM sont celles du firmware (dsp_api.h d'osmocom-bb,
 * DSP == 36), posees sur data[0x0800..] : le code des bancs s'ecrit donc comme
 * celui du firmware (dsp_api.ndb->a_cd[0], dsp_api.db_w->d_task_d, ...).
 */
#ifndef DSP_BANC_COMMUN_H
#define DSP_BANC_COMMUN_H

#include <stdint.h>
#include <stdbool.h>
#include "calypso_c54x.h"
#include "dsp_api.h"            /* osmocom-bb : include/calypso/dsp_api.h */

/* ---- etat cote ARM, comme la struct dsp_api du firmware ------------------ */
struct banc_dsp_api {
    T_NDB_MCU_DSP     *ndb;
    T_DB_DSP_TO_MCU   *db_r;
    T_DB_MCU_TO_DSP   *db_w;
    T_PARAM_MCU_DSP   *param;
    int r_page, w_page, r_page_used;
    uint32_t frame_ctr;
};
extern struct banc_dsp_api dsp_api;
extern C54xState *banc_dsp;
extern uint16_t  *banc_api;           /* = &banc_dsp->data[0x0800] */

/* GSM time (l1s.current_time / l1s.next_time) */
struct banc_temps { uint32_t fn; uint16_t t1; uint8_t t2, t3; };
void banc_fn2temps(struct banc_temps *t, uint32_t fn);
extern struct banc_temps banc_courant, banc_suivant;

/* ---- configuration ------------------------------------------------------- */
struct banc_config {
    const char *rom_dir;    /* calypso_dsp.*.bin (defaut /opt/GSM) */
    long budget;            /* instructions par trame (defaut 300000, ~65 MHz) */
    int verbeux;
    int16_t afc_dac;        /* d_afc ecrit par l'ARM (-700 = pas d'erreur de frequence) */
};
extern struct banc_config banc_cfg;

/* Boot: what dsp_power_on() observed. */
struct banc_boot {
    long insn_chargeur;     /* instructions jusqu'a BL_STATUS_IDLE */
```

```
    long insn_demarrage;    /* instructions de 0x7000 jusqu'a l'IDLE */
    uint16_t bl_status;
    uint16_t version1, version2;
    bool idle;
};

int  banc_init(void);                          /* coeur, ROM, reset, DMA, BSP */
int  banc_dsp_power_on(struct banc_boot *b);   /* calypso/dsp.c dsp_power_on() */

/* ---- primitives du firmware (calypso/dsp.c, layer1/sync.c) ---------------- */
void dsp_end_scenario(void);
void dsp_load_rx_task(uint16_t task, uint8_t burst_id, uint8_t tsc);
void dsp_load_tx_task(uint16_t task, uint8_t burst_id, uint8_t tsc);
void dsp_load_tch_param(const struct banc_temps *next_time, uint8_t chan_mode,
                        uint8_t chan_type, uint8_t chan_sub, uint8_t tch_loop,
                        uint8_t sync_tch, uint8_t tn);
void dsp_load_ciph_param(int mode, const uint8_t *key);
void dsp_memcpy_to_api(volatile uint16_t *dsp_buf, const uint8_t *mcu_buf, int n, int be);
void dsp_memcpy_from_api(uint8_t *mcu_buf, const volatile uint16_t *dsp_buf, int n, int be);
void l1s_reset_hw(void);

/* ---- ordonnanceur TDMA (layer1/tdma_sched.c) ------------------------------ */
#define TDMA_IFLG_TPU   (1 << 0)
#define TDMA_IFLG_DSP   (1 << 1)
typedef int tdma_sched_cb(uint8_t p1, uint8_t p2, uint16_t p3);
struct tdma_sched_item {
    tdma_sched_cb *cb;
    uint8_t p1, p2;
    uint16_t p3;
    int16_t prio;
    uint16_t flags;
};
int tdma_end_set(uint8_t p1, uint8_t p2, uint16_t p3);
#define SCHED_ITEM(x, p, y, z)      { .cb = x, .p1 = y, .p2 = z, .prio = p, .flags = 0 }
#define SCHED_ITEM_DT(x, p, y, z)   { .cb = x, .p1 = y, .p2 = z, .prio = p, \
                                      .flags = TDMA_IFLG_TPU | TDMA_IFLG_DSP }
#define SCHED_END_FRAME()           { .cb = NULL, .p1 = 0, .p2 = 0 }
#define SCHED_END_SET()             { .cb = &tdma_end_set, .p1 = 0, .p2 = 0 }
int  tdma_schedule(uint8_t frame_offset, tdma_sched_cb *cb, uint8_t p1, uint8_t p2,
                   uint16_t p3, int16_t prio);
int  tdma_schedule_set(uint8_t frame_offset, const struct tdma_sched_item *set, uint16_t p3);
void tdma_sched_reset(void);

/* ---- multitrame (layer1/mframe_sched.c, reduit) --------------------------- */
#define MF_F_SACCH  (1 << 0)
struct banc_mframe_item { const struct tdma_sched_item *set; uint16_t modulo, frame_nr; uint16_t p3; };
void banc_mframe_ajouter(const struct tdma_sched_item *set, uint16_t modulo,
                         uint16_t frame_nr, uint16_t p3);
void banc_mframe_vider(void);

/* ---- le canal radio descendant -------------------------------------------- */
/* Le burst que la radio recoit a la trame fn (l'intervalle suivi : TS0 hors
 * dedie, l'intervalle du canal dedie sur ses trames, comme bsp_ts0_livrer).
 * Rend le type : 'F' FCCH, 'S' SCH, 'N' burst normal, 'D' factice, 0 = rien
 * (silence). amp : amplitude (defaut 30000 si 0). */
typedef char (*banc_source_fn)(uint32_t fn, uint8_t bits[148], int *amp);
extern banc_source_fn banc_source;
/* Bits de l'autre intervalle tn (1..7) en DMA continue (recherche FB) ; NULL =
 * burst factice (porteuse C0). */
extern const uint8_t *(*banc_source_autre)(uint32_t fn, unsigned tn);

/* Cellule synthetique TS0 combinee CCCH + SDCCH/4 (45.002 table 3) :
 * FCCH 0/10/20/30/40, SCH 1/11/21/31/41 (BSIC banc_bsic), le reste : le
 * contenu rendu par banc_cellule_nb (bloc xCCH de 23 octets, NULL = factice). */
extern uint8_t banc_bsic;
typedef const uint8_t *(*banc_cellule_nb_fn)(uint32_t fn0, int bloc_p51);
```

```
extern banc_cellule_nb_fn banc_cellule_nb;
char banc_cellule(uint32_t fn, uint8_t bits[148], int *amp);

/* Formes de burst (45.002 5.2) */
extern const uint8_t banc_tsc[8][26];
extern const uint8_t banc_train_sb[64];
extern const uint8_t banc_factice[148];
void banc_burst_nb(const uint8_t *e116, uint8_t tsc, uint8_t bits[148]);   /* 57+hl, TSC, hu+57 */
void banc_burst_sch(uint32_t fn, uint8_t bsic, uint8_t bits[148]);
void banc_sb_info(uint32_t fn, uint8_t bsic, uint8_t sb_info[4]);

/* ---- une trame TDMA ------------------------------------------------------ */
/* Crochets : banc_crochet_l1s est appele par l1_sync() de chaque trame, apres
 * l'execution des items de l'ordonnanceur et avant la bascule des pages (logique
 * de test cote ARM) ; s'il rend TDMA_IFLG_DSP, le scenario DSP est clos comme
 * pour un item SCHED_ITEM_DT. banc_crochet_fin : en fin de trame (sorties). */
extern int (*banc_crochet_l1s)(uint32_t fn);
extern void (*banc_crochet_fin)(uint32_t fn);
/* sur_tch : phase A jusqu'a l'IDLE avant le depot (pont.c, PONT_TCH_DEPOT_IDLE) */
extern bool banc_sur_tch;
void banc_trame(void);                /* une trame : phase A, l1_sync, depot, phase B, pompe */
void banc_courir(uint32_t n_trames);
uint32_t banc_fn(void);               /* trame du prochain banc_trame() */
void banc_set_fn(uint32_t fn);        /* recaler l'horloge (avant la premiere trame) */

/* Statistiques de la trame */
struct banc_stats {
    unsigned long trames, irq_trame, depots, depots_fenetre_sb, pas_idle_fin;
    unsigned long erreurs_dsp;          /* d_error_status non nul */
    uint16_t derniere_erreur;
    long insn_max;
};
extern struct banc_stats banc_stats;

/* ---- bursts montants emis par la ROM ------------------------------------- */
/* La ROM copie a chaque trame d'emission le burst courant (116 bits : 57,
 * hl, hu, 57 -> ordre libosmocoding) dans data[0x3f8a..0x3f91] et avance
 * data[0x3d9b] (pont.c tx_rom_publier). flux = flux A5 montant data[0x3f9b..]. */
/* Un burst serialise par la ROM dans data[0x3f8a..0x3f91] (serialiseur PROM0 0x8900) : anneau xCCH
 * (pointeur 0x3d9b, mode 0x3fac = 3), TCH (0x3d91, mode 0), mode 1 (0x3d93/0x3d95), RACH (0x3d97/0x3d99,
 * mode 2). rang : rang du burst dans le bloc xCCH (0..3), -1 sinon ; raw : les 8 mots tels quels. */
struct banc_ul_burst { uint32_t fn; int rang; int mode; uint16_t ptr; uint16_t raw[8]; uint8_t bits[116]; uint8_t
 ↪  flux[116]; };
#define BANC_UL_MAX 512
extern struct banc_ul_burst banc_ul[BANC_UL_MAX];
/* Le burst montant FINAL tel que la ROM le remet au BSP de l'ABB : 16 mots BULDATA (10 bits de donnees
 * en [15:6], registre BULDATA1 = 3 en [5:1]) ecrits par l'emetteur PROM0 0x8605 dans le script TSP
 * (data[0x3cbb..], juste apres le mot TOGBR2 = 0x1c0a et un mot de puissance). 160 bits = garde/queue
 * etendue + burst de 148 + garde. Capture a chaque trame ou ces 16 mots changent. */
struct banc_tsp_burst { uint32_t fn; uint16_t mots[16]; };
extern struct banc_tsp_burst banc_tsp[BANC_UL_MAX];
extern int banc_n_tsp;
extern int banc_n_ul;
void banc_ul_capturer(uint32_t fn);    /* appele en fin de trame par banc_trame() */

/* Helpers */
long banc_executer(long n);            /* c54x_run en boucle (rend la main tous les 32768) */
void banc_trame_vide(void);            /* une trame sans burst ni IT (DSP au repos) */

#endif
```

3.34 /opt/GSM/c54x_exe/tools/dsp_tester.c

39649 octets, 814 lignes → 810 lignes (1 groupes compactés)

```c
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * dsp_tester.c - testeur de fonctions du DSP Calypso (mask-ROM TI 3606).
 *
 * Sans QEMU ni ARM : le testeur JOUE l'ARM osmocom-bb (tools/dsp_banc_commun.c :
 * dsp_power_on, l1_sync, ordonnanceur TDMA, items de prim_*.c recopies) et la
 * BTS (bursts synthetiques codes par libosmocoding, modules en GMSK comme le
 * BSP du banc). Pour chaque fonction du DSP il pose les codes de tache dans
 * l'API RAM exactement comme le firmware, fait tourner les trames, lit les
 * sorties (pages R, NDB, bursts montants emis) et les compare a la reference
 * libosmocoding. Une ligne PASS / FAIL / NON-IMPL par fonction.
 *
 * Un test qui revele un defaut du DSP (ou du coeur emule) reste FAIL, avec le
 * diagnostic ; aucun contournement n'est applique pour le faire passer.
 *
 *   ./dsp_tester --list            les tests
 *   ./dsp_tester --all             tout
 *   ./dsp_tester fb sb nb ...      les tests nommes
 *   options : -v (detail par trame), --rom-dir DIR, --insns N (budget/trame),
 *             --bsic N
 * Code de retour : nombre de FAIL.
 *
 * Chaque test tourne dans un processus fils, a partir du DSP deja demarre
 * (fork apres dsp_power_on) : etat de depart identique, et un plantage du
 * coeur n'emporte que son test.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <math.h>
#include "dsp_banc_commun.h"
#include "calypso_rhea_dma.h"
#include "verbosite.h"
#include <osmocom/core/bits.h>
#include <osmocom/coding/gsm0503_coding.h>
#include <osmocom/gsm/a5.h>
#include <osmocom/codec/codec.h>

/* ---- verdicts ------------------------------------------------------------ */
enum { V_PASS = 0, V_FAIL = 1, V_NONIMPL = 2 };
static const char *NOMV[] = { "PASS", "FAIL", "NON-IMPL" };
static char g_detail[8192];
static size_t g_dl;
static int g_v;                     /* -v */
static int g_trace;                 /* --trace N : niveau des traces du coeur (src/verbosite.c) */
static void det(const char *fmt, ...) __attribute__((format(printf, 1, 2)));
static void det(const char *fmt, ...)
{
    va_list ap; va_start(ap, fmt);
    int n = vsnprintf(g_detail + g_dl, sizeof g_detail - g_dl, fmt, ap);
    va_end(ap);
    if (n > 0) { g_dl += (size_t)n; if (g_dl >= sizeof g_detail) g_dl = sizeof g_detail - 1; }
}
static void vlog(const char *fmt, ...) __attribute__((format(printf, 1, 2)));
static void vlog(const char *fmt, ...)
{
    if (!g_v) return;
    va_list ap; va_start(ap, fmt);
    printf("      ");
    vprintf(fmt, ap);
    va_end(ap);
}
```

```
static struct banc_boot g_boot;
static int g_boot_rc;

/* ---- outils ------------------------------------------------------------- */
static uint32_t g_alea = 0x1234567u;
static uint8_t alea8(void) { g_alea = g_alea * 1103515245u + 12345u; return (uint8_t)(g_alea >> 16); }
/* Bloc L2 de 23 octets determine par une graine : contenu unique par bloc,
 * de sorte qu'un a_cd perime ne puisse pas passer pour un bon decodage. */
static void l2_bloc(uint32_t graine, uint8_t l2[23])
{
    uint32_t s = graine * 2654435761u + 0x9e3779b9u;
    for (int i = 0; i < 23; i++) { s = s * 1103515245u + 12345u; l2[i] = (uint8_t)(s >> 16); }
    l2[0] = (uint8_t)(graine >> 8); l2[1] = (uint8_t)graine;     /* lisible dans les traces */
}
static void hex23(char *o, const uint8_t *b, int n)
{
    for (int i = 0; i < n; i++) sprintf(o + 3 * i, "%02x ", b[i]);
    if (n) o[3 * n - 1] = 0; else o[0] = 0;
}
static uint32_t gsmtime2fn(unsigned t1, unsigned t2, unsigned t3)
{
    return 51u * 26u * t1 + 51u * (((int)t3 - (int)t2 + 26 * 3) % 26) + t3;
}
#define BITFREQ_DIV_PI    86208
#define ANG2FREQ_SCALING  (2 << 15)
#define ANGLE_TO_FREQ(a)  ((int)(int16_t)(a) * BITFREQ_DIV_PI / ANG2FREQ_SCALING)
#define BITS_PER_TDMA     1250
static unsigned popdiff(const uint8_t *a, const uint8_t *b, int n)
{
    unsigned d = 0; for (int i = 0; i < n; i++) d += (a[i] & 1) != (b[i] & 1); return d;
}

/* source radio courante + contenu des blocs de la cellule */
static uint8_t g_l2_cell[64][23];
static const uint8_t *cellule_nb_alea(uint32_t fn0, int p51)
{
    (void)p51;
    uint8_t *l2 = g_l2_cell[(fn0 / 51 * 10 + (unsigned)p51) % 64];
    l2_bloc(fn0, l2);
    return l2;
}

/* ========================================================================== */
/*  boot / idle / checksum                                                    */
/* ========================================================================== */
static int t_boot(void)
{
    det("chargeur : BL_STATUS=%u apres %ld insn ; demarrage 0x7000 -> IDLE en %ld insn ; version API %04x %04x",
        g_boot.bl_status, g_boot.insn_chargeur, g_boot.insn_demarrage, g_boot.version1, g_boot.version2);
    if (g_boot_rc || g_boot.bl_status != 1 || !g_boot.idle) { det("\nle DSP n'a pas atteint l'IDLE"); return
     ↪  V_FAIL; }
    if (g_boot.version1 != 0x3606) { det("\nversion attendue 3606 (calypso_api.h API_VERSION_VALUE)"); return
     ↪  V_FAIL; }
    return V_PASS;
}

static int scenario_vide(uint32_t fn) { (void)fn; return TDMA_IFLG_DSP; }
static int t_idle(void)
{
    /* L'ARM clot un scenario vide a chaque trame (page W remise a zero par
     * l1_sync, aucune tache) : la ROM recoit l'IT trame, lit une page vide et
     * doit revenir a l'IDLE, sans erreur. */
    banc_source = banc_cellule;
    banc_crochet_l1s = scenario_vide;
    uint32_t i0 = banc_dsp->insn_count;
```

```
    banc_courir(100);
    long moy = (long)(banc_dsp->insn_count - i0) / 100;
    det("100 trames, %lu IT trame ; %ld insn/trame en moyenne, max %ld ; trames finies hors IDLE : %lu ;
↪   d_error_status : %lu",
        banc_stats.irq_trame, moy, banc_stats.insn_max, banc_stats.pas_idle_fin, banc_stats.erreurs_dsp);
    if (banc_stats.irq_trame < 99) { det("\nIT trame non prise (IMR=%04x)", banc_dsp->imr); return V_FAIL; }
    if (banc_stats.pas_idle_fin || banc_stats.erreurs_dsp) return V_FAIL;
    if (moy == 0) { det("\nle DSP n'execute rien sur l'IT trame"); return V_FAIL; }
    return V_PASS;
}

static int g_chk_vu;
static int chk_cmd(uint8_t p1, uint8_t p2, uint16_t p3)
{
    (void)p1; (void)p2; (void)p3;
    dsp_api.db_w->d_task_md = CHECKSUM_DSP_TASK;
    dsp_api.ndb->d_fb_mode = 1;
    return 0;
}
static const struct tdma_sched_item chk_set[] = { SCHED_ITEM_DT(chk_cmd, 0, 0, 0), SCHED_END_FRAME(),
↪   SCHED_END_SET() };
static int t_checksum(void)
{
    /* calypso/dsp.c dsp_checksum_task() : d_task_md = CHECKSUM_DSP_TASK (33),
     * puis a_pm[0] = version du code, a_pm[1] = somme, API 0x08DB = patch. */
    T_DB_DSP_TO_MCU *r0 = (T_DB_DSP_TO_MCU *)&banc_api[0x28], *r1 = (T_DB_DSP_TO_MCU *)&banc_api[0x3C];
    banc_source = banc_cellule;
    tdma_schedule_set(1, chk_set, 0);
    uint16_t av[2][3]; memcpy(av[0], r0->a_pm, 6); memcpy(av[1], r1->a_pm, 6);
    banc_courir(6);
    (void)g_chk_vu;
    det("R0 a_pm=%04x %04x %04x d_task_md=%u | R1 a_pm=%04x %04x %04x d_task_md=%u | patch (API 0x08DB)=%04x",
       r0->a_pm[0], r0->a_pm[1], r0->a_pm[2], r0->d_task_md, r1->a_pm[0], r1->a_pm[1], r1->a_pm[2], r1->d_task_md,
        banc_api[0xDB]);
    int ok = 0;
    for (int p = 0; p < 2; p++) {
        T_DB_DSP_TO_MCU *r = p ? r1 : r0;
        if (r->a_pm[0] == 0x3606 || (r->a_pm[0] && memcmp(av[p], r->a_pm, 6))) ok = 1;
    }
    if (!ok) { det("\naucune page R ne porte de version/somme : tache non executee"); return V_FAIL; }
    return V_PASS;
}

/* ================================================================== */
/*  FB / SB (layer1/prim_fbsb.c)                                      */
/* ================================================================== */
static struct { int det, attempt, mode; uint32_t fn_resp; int16_t toa, pm, angle, snr; int freq; } g_fb;
static uint32_t g_fb_cmd_fn;
static int fbdet_cmd(uint8_t p1, uint8_t p2, uint16_t fb_mode)
{
    (void)p1; (void)p2;
    g_fb_cmd_fn = banc_courant.fn;
    dsp_api.db_w->d_task_md = FB_DSP_TASK;
    dsp_api.ndb->d_fb_mode = fb_mode;
    return 0;
}
static int fbdet_resp(uint8_t p1, uint8_t attempt, uint16_t fb_mode)
{
    (void)p1;
    if (g_fb.det) return 0;
    if (!dsp_api.ndb->d_fb_det) {
        if (attempt == 12) g_fb.det = -1;
        return 0;
    }
    l1s_reset_hw();
    /* read_fb_result() */
```

```
    g_fb.det = 1; g_fb.attempt = attempt; g_fb.mode = fb_mode; g_fb.fn_resp = banc_courant.fn;
    g_fb.toa = (int16_t)dsp_api.ndb->a_sync_demod[D_TOA];
    g_fb.pm = (int16_t)(dsp_api.ndb->a_sync_demod[D_PM] >> 3);
    g_fb.angle = (int16_t)dsp_api.ndb->a_sync_demod[D_ANGLE];
    g_fb.snr = (int16_t)dsp_api.ndb->a_sync_demod[D_SNR];
    g_fb.freq = ANGLE_TO_FREQ(g_fb.angle);
    dsp_api.ndb->d_fb_det = 0;
    dsp_api.ndb->a_sync_demod[D_TOA] = 0;
    tdma_sched_reset();
    return 0;
}
static const struct tdma_sched_item fb_sched_set[] = {
    SCHED_ITEM_DT(fbdet_cmd, 0, 0, 0), SCHED_END_FRAME(),
    SCHED_END_FRAME(),
    SCHED_ITEM(fbdet_resp, -4, 0, ⟨1⟩), SCHED_END_FRAME(), SCHED_ITEM(fbdet_resp, -4, 0, ⟨2⟩), SCHED_END_FRAME(),
 ↪  ×6
    ⟨⟩ = (1,2) (3,4) (5,6) (7,8) (9,10) (11,12)
    SCHED_END_SET()
};

/* Une recherche FB en mode `mode`, commande posee a la prochaine l1_sync.
 * Rend la trame FCCH que le firmware en deduit (fn_offset de prim_fbsb.c, avec
 * l'arrondi que synchronize_tdma() fait par les qbits), -1 si rien ;
 * *residu = ecart du TOA a la position nominale (23 + n x 1250), en symboles. */
static long une_fb(int mode, uint32_t *debut, int *residu)
{
    memset(&g_fb, 0, sizeof g_fb);
    tdma_schedule_set(0, fb_sched_set, (uint16_t)mode);   /* hors l1_sync : 0 = la prochaine */
    banc_trame();                          /* la l1_sync de cette trame pose la commande */
    *debut = g_fb_cmd_fn + 1;              /* next_time de la commande : 1re trame de recherche */
    for (int k = 0; k < 14 && !g_fb.det; k++) banc_trame();
    if (g_fb.det <= 0) return -1;
    int toa = g_fb.toa - 23;
    int n = (int)lrint((double)toa / BITS_PER_TDMA);
    *residu = toa - n * BITS_PER_TDMA;
    /* prim_fbsb.c : fn_offset = current - attempt + ntdma, et current - attempt = debut */
    return (long)(g_fb.fn_resp - (uint32_t)g_fb.attempt) + n;
}
static int est_fcch(uint32_t f) { return (f % 51) % 10 == 0 && (f % 51) <= 40; }

static int t_fb_mode(int mode, int16_t dac, int hz_attendu, const char *quoi)
{
    banc_source = banc_cellule;
    banc_cfg.afc_dac = dac;
    int ok = 0, n = 0, sautees = 0;
    for (int essai = 0; essai < 6; essai++) {
        banc_courir(3 + 7 * (unsigned)essai);          /* positions differentes vis-a-vis de la FCCH */
        uint32_t debut; int residu = 0;
        long fcch = une_fb(mode, &debut, &residu);
        n++;
        uint32_t premiere = debut; while (!est_fcch(premiere)) premiere++;
        if (fcch < 0) { det("essai %d : recherche des fn=%u, pas de FB en 12 trames (FCCH en %u)\n", essai, debut,
         ↪  premiere); continue; }
        int fe = g_fb.freq - hz_attendu;
        int juste = fcch >= (long)debut && est_fcch((uint32_t)fcch) && abs(residu) <= 64 &&
                    abs(fe) <= (hz_attendu ? abs(hz_attendu) / 4 : 100);
        if (juste && fcch != (long)premiere) sautees++;
        det("essai %d : recherche des fn=%u -> FB%d att=%d TOA=%d (residu %+d symb.) PM=%d angle=%d (%+d Hz)
 ↪  SNR=%d ; FCCH deduite %ld%s%s\n",
              essai, debut, mode, g_fb.attempt, g_fb.toa, residu, g_fb.pm, g_fb.angle, g_fb.freq, g_fb.snr, fcch,
             fcch == (long)premiere ? "" : est_fcch((uint32_t)fcch) ? " (une FCCH sautee)" : " (PAS une trame FCCH)",
              juste ? "" : "  <- FAUX");
        ok += juste;
    }
    det("%s : %d/%d detections justes (FCCH reelle, TOA a +-64 symboles, frequence %+d Hz +-%d) ; %d fois la 1re
 ↪  FCCH sautee",
```

```
        quoi, ok, n, hz_attendu, hz_attendu ? abs(hz_attendu) / 4 : 100, sautees);
    return ok == n ? V_PASS : V_FAIL;
}
static int t_fb0(void) { return t_fb_mode(0, -700, 0, "FB mode 0 (acquisition)"); }
static int t_fb1(void) { return t_fb_mode(1, -700, 0, "FB mode 1 (poursuite)"); }
/* calypso_twl3025.c : (dac + 700) x 287 / (2^15/947) Hz, ~8.29 Hz/LSB. Le VCXO
 * tire de +60 LSB : la porteuse recue parait a -497 Hz, signe que la boucle du
 * firmware (afc_correct : dac += norm x err / pente) ramene vers -700. */
static int t_fb_afc(void)
{
    int hz = (int)lrint(-60.0 * 287.0 / (32768.0 / 947.0));
    return t_fb_mode(0, -700 + 60, hz, "FB mode 0, VCXO +60 LSB (d_afc -640)");
}

/* SB */
static struct { int fait, crc_ok; uint32_t sb; int16_t toa, pm, angle, snr; uint32_t fn_resp; } g_sb[3];
static int sbdet_cmd(uint8_t p1, uint8_t p2, uint16_t p3)
{
    (void)p1; (void)p2; (void)p3;
    dsp_api.db_w->d_task_md = SB_DSP_TASK;
    dsp_api.ndb->d_fb_mode = 0;
    return 0;
}
static int sbdet_resp(uint8_t p1, uint8_t attempt, uint16_t p3)
{
    (void)p1; (void)p3;
    if (attempt > 2) return 0;
    g_sb[attempt].fait = 1;
    g_sb[attempt].fn_resp = banc_courant.fn;
    dsp_api.r_page_used = 1;
    if (dsp_api.db_r->a_sch[0] & (1 << B_SCH_CRC)) return 0;
    g_sb[attempt].crc_ok = 1;
    g_sb[attempt].toa = (int16_t)dsp_api.db_r->a_serv_demod[D_TOA];
    g_sb[attempt].pm = (int16_t)(dsp_api.db_r->a_serv_demod[D_PM] >> 3);
    g_sb[attempt].angle = (int16_t)dsp_api.db_r->a_serv_demod[D_ANGLE];
    g_sb[attempt].snr = (int16_t)dsp_api.db_r->a_serv_demod[D_SNR];
    g_sb[attempt].sb = dsp_api.db_r->a_sch[3] | (uint32_t)dsp_api.db_r->a_sch[4] << 16;
    return 0;
}
static const struct tdma_sched_item sb_sched_set[] = {
    SCHED_ITEM_DT(sbdet_cmd, 0, 0, 1), SCHED_END_FRAME(),
    SCHED_ITEM_DT(sbdet_cmd, 0, 0, 2), SCHED_END_FRAME(),
    SCHED_END_FRAME(),
    SCHED_ITEM(sbdet_resp, -4, 0, 1), SCHED_END_FRAME(),
    SCHED_ITEM(sbdet_resp, -4, 0, 2), SCHED_END_FRAME(),
    SCHED_END_SET()
};
/* l1s_decode_sb() */
static uint8_t decode_sb(uint32_t sb, uint32_t *fn)
{
    unsigned t1 = ((sb >> 23) & 1) | ((sb >> 7) & 0x1fe) | ((sb << 9) & 0x600);
    unsigned t2 = (sb >> 18) & 0x1f;
    unsigned t3p = ((sb >> 24) & 1) | ((sb >> 15) & 6);
    *fn = gsmtime2fn(t1, t2, t3p * 10 + 1);
    return (sb >> 2) & 0x3f;
}

static int t_sb(void)
{
    /* La commande SB vise directement chaque trame SCH (p51 = 1, 11, .., 41) :
     * la 1re commande du sb_sched_set a next_time = trame SCH. */
    banc_source = banc_cellule;
    int n = 0, ok = 0, crc_faux = 0, faux = 0, rate = 0, fen = 0;
    char ligne[400];
    for (int k = 0; k < 20; k++) {
        uint32_t c = banc_fn();                /* current_time de la prochaine l1_sync */
```

```
        uint32_t s = c + 2; while ((s % 51) % 10 != 1 || (s % 51) > 41) s++;
        unsigned long fen0 = banc_stats.depots_fenetre_sb;
        memset(g_sb, 0, sizeof g_sb);
        while (banc_fn() < s - 2) banc_trame();
        /* l1_sync(s-2) programme le set a +1 : cmd1 a l1_sync(s-1), next_time s */
        tdma_schedule_set(1, sb_sched_set, 0);
        for (int i = 0; i < 6; i++) banc_trame();
        n++;
        bool dans_fenetre = banc_stats.depots_fenetre_sb > fen0;
        fen += dans_fenetre;
        uint32_t fn_dec = 0; uint8_t bsic = 0;
        if (g_sb[1].crc_ok) {
            bsic = decode_sb(g_sb[1].sb, &fn_dec);
            bool juste = bsic == banc_bsic && fn_dec == s;
            if (juste) ok++; else { faux++; crc_faux++; }
            snprintf(ligne, sizeof ligne, "SCH fn=%u : CRC ok, BSIC=%u fn=%u TOA=%d SNR=%d%s\n", s, bsic, fn_dec,
                    g_sb[1].toa, g_sb[1].snr, juste ? "" : "  <- CONTENU FAUX");
        } else {
            rate++;
            snprintf(ligne, sizeof ligne, "SCH fn=%u : CRC KO%s (2e essai sur trame factice : %s)\n", s,
                    dans_fenetre ? "" : ", jamais livre dans une fenetre SB", g_sb[2].crc_ok ? "CRC ok ?!" : "CRC
↪  KO");
            if (g_sb[2].crc_ok) faux++;
        }
        if (g_v || !g_sb[1].crc_ok || faux) det("%s", ligne);
        vlog("%s", ligne);
    }
    det("%d SCH vises : %d decodes justes, %d CRC KO, %d CRC ok au contenu faux ; %d livres dans une fenetre SB",
        n, ok, rate, faux, fen);
    (void)crc_faux;
    if (faux) return V_FAIL;
    /* un recepteur sain decode un SCH propre a coup sur ; on tolere 1 sur 20 */
    return (ok >= n - 1) ? V_PASS : V_FAIL;
}

/* FB0 -> FB1 -> SB, comme l1s_fbsb_req() + les reponses de prim_fbsb.c */
static int t_fbsb(void)
{
    banc_source = banc_cellule;
    banc_cfg.afc_dac = -700;
    banc_courir(7);
    uint32_t debut; int residu;
    long f0 = une_fb(0, &debut, &residu);
    if (f0 < 0) { det("FB0 introuvable"); return V_FAIL; }
    det("FB0 : TOA=%d %+d Hz -> FCCH %ld\n", g_fb.toa, g_fb.freq, f0);
    long f1 = une_fb(1, &debut, &residu);
    if (f1 < 0) { det("FB1 introuvable"); return V_FAIL; }
    det("FB1 : TOA=%d %+d Hz -> FCCH %ld\n", g_fb.toa, g_fb.freq, f1);
    /* delay = fn_offset + 11 - fn - 1, programme dans la l1_sync de la reponse */
    int toa = g_fb.toa - 23;
    int ntdma = toa < 0 ? -1 : toa / BITS_PER_TDMA;
    int fn_offset = (int)g_fb.fn_resp - g_fb.attempt + ntdma;
    /* nous sommes apres la trame de reponse : le set part de la prochaine l1_sync */
    int delay = fn_offset + 11 - (int)banc_fn() - 1;
    if (delay < 0) delay = 0;
    uint32_t s_vise = (uint32_t)(fn_offset + 11);
    memset(g_sb, 0, sizeof g_sb);
    tdma_schedule_set((uint8_t)(delay + 1), sb_sched_set, 0);
    for (int i = 0; i < delay + 8; i++) banc_trame();
    for (int a = 1; a <= 2; a++) {
        if (!g_sb[a].crc_ok) { det("SB essai %d : CRC KO\n", a); continue; }
        uint32_t fn_dec; uint8_t bsic = decode_sb(g_sb[a].sb, &fn_dec);
        det("SB essai %d : CRC ok BSIC=%u fn=%u (vise %u) TOA=%d\n", a, bsic, fn_dec, s_vise, g_sb[a].toa);
        if (bsic == banc_bsic && fn_dec == s_vise) { det("acquisition complete FB0/FB1/SB : BSIC %u, horloge %u",
         ↪  bsic, fn_dec); return V_PASS; }
    }
```

```
    det("SB non decode apres FB1 (SCH vise %u)", s_vise);
    return V_FAIL;
}

/* ========================================================================== */
/*  NB descendant (layer1/prim_rx_nb.c)                                       */
/* ========================================================================== */
static struct { int n_blocs, ok, fire, faux, vide, mauvais_id, err_tot; uint16_t tache; uint8_t chan_type;
                uint32_t blocs_fn[64]; int toa[4], snr[4]; char premier_ko[400]; } g_nb;
static uint8_t g_nb_l2_attendu[23];
static const uint8_t *(*g_nb_l2_de)(uint32_t fn0);
static int nb_resp(uint8_t p1, uint8_t burst_id, uint16_t p3)
{
    (void)p1; (void)p3;
    if (dsp_api.db_r->d_task_d == 0) { g_nb.vide++; return 0; }          /* "EMPTY" */
    if (dsp_api.db_r->d_burst_d != burst_id) { g_nb.mauvais_id++; return 0; }
    g_nb.toa[burst_id] = (int16_t)dsp_api.db_r->a_serv_demod[D_TOA];
    g_nb.snr[burst_id] = dsp_api.db_r->a_serv_demod[D_SNR];
    if (burst_id == 3) {
        uint32_t fn0 = banc_courant.fn - 5;       /* rx du burst 0 : current - 4 - 1 */
        uint8_t l2[23];
        uint16_t fire = (dsp_api.ndb->a_cd[0] & ((1 << B_FIRE1) | (1 << B_FIRE0))) >> B_FIRE0;
        uint16_t err = dsp_api.ndb->a_cd[2];
        dsp_memcpy_from_api(l2, &dsp_api.ndb->a_cd[3], 23, 0);
        const uint8_t *att = g_nb_l2_de(fn0);
        g_nb.n_blocs++;
        g_nb.err_tot += err;
        bool egal = att && !memcmp(l2, att, 23);
        if (!fire && egal) g_nb.ok++;
        else {
            if (fire) g_nb.fire++; else g_nb.faux++;
            if (!g_nb.premier_ko[0]) {
                char h1[80], h2[80]; hex23(h1, l2, 8); hex23(h2, att ? att : l2, 8);
                snprintf(g_nb.premier_ko, sizeof g_nb.premier_ko, "bloc fn0=%u : a_cd0=%04x FIRE=%u err=%u L2 %s..
↪  attendu %s..",
                         fn0, dsp_api.ndb->a_cd[0], fire, err, h1, h2);
            }
        }
        vlog("bloc fn0=%u a_cd0=%04x FIRE=%u err=%u %s TOA=%d/%d/%d/%d\n", fn0, dsp_api.ndb->a_cd[0], fire, err,
             egal ? "L2 egal" : "L2 DIFFERENT", g_nb.toa[0], g_nb.toa[1], g_nb.toa[2], g_nb.toa[3]);
        dsp_api.db_w->d_task_d = 0;
    }
    dsp_api.r_page_used = 1;
    return 0;
}
static int nb_cmd(uint8_t p1, uint8_t burst_id, uint16_t p3)
{
    (void)p1; (void)p3;
    dsp_load_tch_param(&banc_suivant, SIG_ONLY_MODE, g_nb.chan_type, 0, 0, 0, 0);
    dsp_load_rx_task(g_nb.tache, burst_id, banc_bsic & 7);
    return 0;
}
static const struct tdma_sched_item nb_sched_set[] = {
    SCHED_ITEM_DT(nb_cmd, 0, 0, 0), SCHED_END_FRAME(),
    SCHED_ITEM_DT(nb_cmd, 0, 0, 1), SCHED_END_FRAME(),
    SCHED_ITEM(nb_resp, -4, 0, 0), SCHED_ITEM_DT(nb_cmd, 0, 0, 2), SCHED_END_FRAME(),
    SCHED_ITEM(nb_resp, -4, 0, 1), SCHED_ITEM_DT(nb_cmd, 0, 0, 3), SCHED_END_FRAME(),
    SCHED_ITEM(nb_resp, -4, 0, 2), SCHED_END_FRAME(),
    SCHED_ITEM(nb_resp, -4, 0, 3), SCHED_END_FRAME(),
    SCHED_END_SET()
};
static const uint8_t *nb_l2_cellule(uint32_t fn0) { return cellule_nb_alea(fn0, (int)(fn0 % 51)); }

/* Lecture de n blocs xCCH de la cellule a la position p51 (2 = BCCH,
 * 6 = CCCH, 22 = SDCCH/4 n0, 42 = SACCH/4 ...) avec la tache `tache`. */
static int nb_lecture(uint16_t tache, uint8_t chan_type, int p51, int modulo, int n_blocs, const char *quoi)
```

```
{
    memset(&g_nb, 0, sizeof g_nb);
    g_nb.tache = tache; g_nb.chan_type = chan_type;
    g_nb_l2_de = nb_l2_cellule;
    banc_source = banc_cellule;
    banc_cellule_nb = cellule_nb_alea;
    banc_mframe_ajouter(nb_sched_set, (uint16_t)modulo, (uint16_t)p51, 0);
    uint32_t lim = banc_fn() + (uint32_t)(modulo * (n_blocs + 1) + 10);
    while (g_nb.n_blocs < n_blocs && banc_fn() < lim) banc_trame();
    banc_mframe_vider();
    det("%s (tache %u) : %d/%d blocs decodes justes, %d FIRE KO, %d contenu faux, %d erreurs canal au total ; "
        "R vide %d, d_burst_d faux %d ; TOA dernier bloc %d/%d/%d/%d",
        quoi, tache, g_nb.ok, g_nb.n_blocs, g_nb.fire, g_nb.faux, g_nb.err_tot, g_nb.vide, g_nb.mauvais_id,
        g_nb.toa[0], g_nb.toa[1], g_nb.toa[2], g_nb.toa[3]);
    if (g_nb.premier_ko[0]) det("\n1er echec : %s", g_nb.premier_ko);
    if (g_nb.n_blocs == 0) { det("\naucun bloc rendu"); return V_FAIL; }
    return (g_nb.ok == g_nb.n_blocs && g_nb.n_blocs >= n_blocs) ? V_PASS : V_FAIL;
}
static int t_nb_bcch(void) { return nb_lecture(ALLC_DSP_TASK, SDCCH_4, 2, 51, 8, "BCCH norm. (ALLC, comme
 ↪  osmocom-bb)"); }

/* ====================================================================== */
/*  NB montant (layer1/prim_tx_nb.c)                                      */
/* ====================================================================== */
static struct { uint16_t tache; uint8_t l2[16][23]; uint32_t fn_cmd[16][4]; int n_blocs; int sacch; } g_tx;
static int tx_resp(uint8_t p1, uint8_t burst_id, uint16_t p3)
{
    (void)p1; (void)burst_id; (void)p3;
    dsp_api.r_page_used = 1;
    return 0;
}
static int tx_cmd(uint8_t p1, uint8_t burst_id, uint16_t p3)
{
    (void)p3;
    if (burst_id == 0) {
        uint16_t *info = dsp_api.ndb->a_cu;
        int b = g_tx.n_blocs < 16 ? g_tx.n_blocs : 15;
        l2_bloc(0x7000u + (uint32_t)banc_suivant.fn, g_tx.l2[b]);
        info[0] = (1 << B_BLUD);
        info[1] = 0;
        info[2] = 0;
        dsp_memcpy_to_api(&info[3], g_tx.l2[b], 23, 0);
        g_tx.n_blocs++;
    }
    if (g_tx.n_blocs >= 1 && g_tx.n_blocs <= 16) g_tx.fn_cmd[g_tx.n_blocs - 1][burst_id & 3] = banc_suivant.fn;
    if (p1 == 2) dsp_load_tch_param(&banc_suivant, SIG_ONLY_MODE, SDCCH_4, 0, 0, 0, 0);
    else         dsp_load_tch_param(&banc_suivant, SIG_ONLY_MODE, INVALID_CHANNEL, 0, 0, 0, 0);
    dsp_load_tx_task(g_tx.tache, burst_id, banc_bsic & 7);
    return 0;
}
static const struct tdma_sched_item nb_sched_set_ul[] = {
    SCHED_ITEM_DT(tx_cmd, 3, 2, 0), SCHED_END_FRAME(),
    SCHED_ITEM_DT(tx_cmd, 3, 2, 1), SCHED_END_FRAME(),
    SCHED_ITEM(tx_resp, -4, 2, 0), SCHED_ITEM_DT(tx_cmd, 3, 2, 2), SCHED_END_FRAME(),
    SCHED_ITEM(tx_resp, -4, 2, 1), SCHED_ITEM_DT(tx_cmd, 3, 2, 3), SCHED_END_FRAME(),
    SCHED_ITEM(tx_resp, -4, 2, 2), SCHED_END_FRAME(),
    SCHED_ITEM(tx_resp, -4, 2, 3), SCHED_END_FRAME(),
    SCHED_END_SET()
};

/* Comparer les bursts emis (banc_ul) apres chaque commande de bloc avec
 * gsm0503_xcch_encode ; a5 : flux attendu XORe (algo, Kc) au fn du burst. */
static int ul_comparer_xcch(int a5_algo, const uint8_t *kc, const char *quoi)
{
    int blocs_ok = 0, bursts_ok = 0, bursts_n = 0, manquants = 0;
    char premier[600] = "";
```

```
    for (int b = 0; b < g_tx.n_blocs && b < 16; b++) {
        ubit_t e[4 * 116];
        gsm0503_xcch_encode(e, g_tx.l2[b]);
        int bon = 0;
        for (int k = 0; k < 4; k++) {
            uint32_t f = g_tx.fn_cmd[b][k];
            /* le burst emis pour la trame f : capture a la fin de la trame f */
            const struct banc_ul_burst *u = NULL;
            for (int i = 0; i < banc_n_ul && i < BANC_UL_MAX; i++)
                if (banc_ul[i].fn == f) u = &banc_ul[i];
            bursts_n++;
            if (!u) { manquants++; if (!premier[0]) snprintf(premier, sizeof premier, "bloc %d burst %d (fn %u) :
             ↪  aucun burst emis (0x3d9b immobile)", b, k, f); continue; }
            uint8_t att[116];
            memcpy(att, e + 116 * k, 116);
            if (a5_algo) {
                ubit_t dl[114], ul[114];
                osmo_a5(a5_algo, kc, f, dl, ul);
                for (int j = 0; j < 57; j++) { att[j] ^= ul[j]; att[59 + j] ^= ul[57 + j]; }
            }
            unsigned d = popdiff(u->bits, att, 116);
            if (!d && u->rang == k) { bon++; bursts_ok++; continue; }
            if (!premier[0]) {
                int dec = 99;
                if (a5_algo) {            /* chiffre a une autre trame ? */
                    for (int df = -3; df <= 3; df++) {
                        uint8_t a2[116]; memcpy(a2, e + 116 * k, 116);
                        ubit_t dl[114], ul[114]; osmo_a5(a5_algo, kc, f + df, dl, ul);
                        for (int j = 0; j < 57; j++) { a2[j] ^= ul[j]; a2[59 + j] ^= ul[57 + j]; }
                        if (!popdiff(u->bits, a2, 116)) { dec = df; break; }
                    }
                }
                unsigned dclair = popdiff(u->bits, e + 116 * k, 116);
               snprintf(premier, sizeof premier, "bloc %d burst %d (fn %u, rang ROM %d) : %u bits faux sur 116 (%u
 ↪  contre le burst en clair)%s",
                         b, k, f, u->rang, d, dclair,
                         dec != 99 ? (dec ? " ; chiffre avec le flux d'une autre trame" : "") : "");
                if (dec != 99 && dec) { size_t l = strlen(premier); snprintf(premier + l, sizeof premier - l, "
                 ↪  (fn%+d)", dec); }
            }
        }
        if (bon == 4) blocs_ok++;
    }
    det("%s : %d/%d blocs identiques a gsm0503_xcch_encode%s (%d/%d bursts, %d non emis)",
        quoi, blocs_ok, g_tx.n_blocs, a5_algo ? " + A5" : "", bursts_ok, bursts_n, manquants);
    if (premier[0]) det("\n1er ecart : %s", premier);
    return (g_tx.n_blocs > 0 && blocs_ok == g_tx.n_blocs) ? V_PASS : V_FAIL;
}

static int ul_bloc(uint16_t tache, int p51, int modulo, int n_blocs, int a5, const uint8_t *kc, const char *quoi)
{
    memset(&g_tx, 0, sizeof g_tx);
    g_tx.tache = tache;
    banc_source = banc_cellule;
    if (a5) dsp_load_ciph_param(a5, kc);
    banc_mframe_ajouter(nb_sched_set_ul, (uint16_t)modulo, (uint16_t)p51, 0);
    uint32_t lim = banc_fn() + (uint32_t)(modulo * (n_blocs + 1) + 10);
    while (banc_fn() < lim && g_tx.n_blocs < n_blocs) banc_trame();
    banc_mframe_vider();
    banc_courir(8);
    return ul_comparer_xcch(a5, kc, quoi);
}
static int t_tx_sdcch(void) { return ul_bloc(DUL_DSP_TASK, 22 + 15, 51, 6, 0, NULL, "SDCCH/4 n0 montant (DUL,
 ↪  comme osmocom-bb)"); }
/* Meme bloc, chiffre A5/1 par la ROM : ul_comparer_xcch compare au flux osmo_a5(kc, fn) de la trame
 * de la commande (next_time) et dit si la ROM a chiffre avec le flux d'une autre trame (fn+-d). C'est
```

```
 * la question ouverte « a quelle heure la ROM chiffre le montant » (99-couverture.sh:137). */
static int t_tx_sdcch_a5(void)
{
    static const uint8_t kc[8] = { 0x12, 0x34, 0x56, 0x78, 0x9a, 0xbc, 0xde, 0xf0 };
    return ul_bloc(DUL_DSP_TASK, 22 + 15, 51, 6, 1, kc, "SDCCH/4 n0 montant chiffre A5/1 (DUL)");
}

/* ====================================================================== */
/*  RACH montant (layer1/prim_rach.c)                                     */
/* ====================================================================== */
/* Le firmware ecrit d_rach = (ra << 8) | (bsic << 2) dans le NDB et d_task_ra = RACH_DSP_TASK (10) dans
 * la page W (prim_rach.c l1s_tx_rach_cmd). La ROM code l'access-burst (36 bits codes) dans son anneau
 * TX de mode 2 (pointeur 0x3d97), le serialiseur 0x8900 le copie dans data[0x3f8a] a la trame d'emission,
 * et l'emetteur 0x85a2 y ajoute la sequence de synchro (DROM 0xa0d9, 41 bits) et les queues. Ici on
 * compare les 36 bits codes trouves dans 0x3f8a a gsm0503_rach_ext_encode(ra, bsic) (bits 49..84 du
 * burst de 148). */
static struct { uint8_t ra, bsic; uint32_t fn_cmd; } g_rach;
static int rach_cmd(uint8_t p1, uint8_t p2, uint16_t p3)
{
    (void)p1; (void)p2; (void)p3;
    dsp_api.ndb->d_rach = (uint16_t)((g_rach.ra << 8) | (g_rach.bsic << 2));
    dsp_api.db_w->d_task_ra = RACH_DSP_TASK;
    g_rach.fn_cmd = banc_suivant.fn;
    return 0;
}
static int rach_resp(uint8_t p1, uint8_t p2, uint16_t p3) { (void)p1; (void)p2; (void)p3; dsp_api.r_page_used = 1;
 ↪  return 0; }
static const struct tdma_sched_item rach_sched_set[] = {
    SCHED_ITEM_DT(rach_cmd, 3, 1, 0), SCHED_END_FRAME(),
    SCHED_END_FRAME(),
    SCHED_ITEM(rach_resp, -4, 1, 0), SCHED_END_FRAME(),
    SCHED_END_SET()
};
static int t_rach(void)
{
    banc_source = banc_cellule;
    static const char *sync = "0100101101111111110011001101010100011111000";   /* 05.02 5.2.7, 41 bits */
    int n = 0, ok36 = 0, ok148 = 0;
    for (int essai = 0; essai < 5; essai++) {
        banc_courir(4);
        g_rach.ra = alea8(); g_rach.bsic = banc_bsic;
        int n0 = banc_n_tsp;
        tdma_schedule_set(0, rach_sched_set, 0);
        banc_courir(6);
        n++;
        ubit_t cod[36];
        gsm0503_rach_ext_encode(cod, g_rach.ra, g_rach.bsic, false);
        /* 1. les 36 bits codes dans data[0x3f8a..0x3f8c] (le codeur de la ROM) */
        const uint16_t *m = banc_dsp->data;
        int d36 = 0;
        for (int k = 0; k < 36; k++) d36 += ((m[0x3f8a + k / 16] >> (15 - k % 16)) & 1) != cod[k];
        /* 2. le burst final (script TSP, emetteur 0x8605) : [8 garde][8 queue etendue 00111010][41 sync][36][3
         ↪  queue][garde] */
        char att[8 + 41 + 36 + 3 + 1]; int p = 0;
        memcpy(att, "00111010", 8); p = 8;
        memcpy(att + p, sync, 41); p += 41;
        for (int k = 0; k < 36; k++) att[p++] = '0' + cod[k];
        memcpy(att + p, "000", 3); p += 3; att[p] = 0;
        int trouve = -1; char flux[161];
        for (int i = n0; i < banc_n_tsp; i++) {
            const struct banc_tsp_burst *t = &banc_tsp[i % BANC_UL_MAX];
            for (int w = 0; w < 16; w++) for (int b = 0; b < 10; b++) flux[10 * w + b] = '0' + ((t->mots[w] >> (15
             ↪  - b)) & 1);
            flux[160] = 0;
            const char *q = strstr(flux, att);
            if (q) { trouve = (int)(q - flux); break; }
```

```
        }
        det("essai %d : ra=%02x bsic=%u : 36 bits codes dans 0x3f8a : %d faux ; burst final (TSP, %d capture%s) :
 ↪  [queue 00111010][sync][36][000] %s\n",
            essai, g_rach.ra, g_rach.bsic, d36, banc_n_tsp - n0, banc_n_tsp - n0 > 1 ? "s" : "",
            trouve >= 0 ? "trouve" : "ABSENT");
        if (trouve >= 0) det("          flux de 160 bits : %s (burst a l'offset %d)\n", flux, trouve);
        ok36 += d36 == 0; ok148 += trouve >= 0;
    }
    det("RACH : %d/%d codages (36 bits) = gsm0503_rach_ext_encode ; %d/%d access-bursts complets dans le script
 ↪  TSP de la ROM", ok36, n, ok148, n);
    return (ok36 == n && ok148 == n) ? V_PASS : V_FAIL;
}

/* ================================================================== */
/*  table des tests                                                   */
/* ================================================================== */
/* TOA de la FB quand la FCCH est dans la TOUTE PREMIERE trame de recherche
 * (commande a p51 = 9, recherche des p51 = 10 : FCCH). FBPOS_MODE=0|1 (defaut 1).
 * Avec BANC_FB_DECAL=k (decalage du burst depuis l'ouverture de la fenetre), donne
 * le TOA de la FB en fonction de k, sans les termes de pages de 48 d'une FCCH lointaine. */
static int t_fb_pos(void)
{
    banc_source = banc_cellule;
    banc_cfg.afc_dac = -700;
    const char *em = getenv("FBPOS_MODE");
    int mode = em ? atoi(em) : 1;
    int vus = 0;
    for (int essai = 0; essai < 3; essai++) {
        uint32_t cible = (banc_fn() / 51 + 2) * 51 + 9;
        while (banc_fn() < cible) banc_trame();
        uint32_t debut; int residu = 0;
        long fcch = une_fb(mode, &debut, &residu);
        if (fcch < 0 || g_fb.det <= 0) { det("essai %d : debut=%u (p51=%u) : pas de FB\n", essai, debut, debut %
 ↪  51); continue; }
        vus++;
        det("essai %d : debut=%u (p51=%u) FB%d att=%d TOA=%d PM=%d angle=%d\n",
            essai, debut, debut % 51, mode, g_fb.attempt, g_fb.toa, g_fb.pm, g_fb.angle);
    }
    return vus ? V_PASS : V_FAIL;
}

typedef struct { const char *nom, *tache, *desc; int (*f)(void); } Test;
static const Test TESTS[] = {
    { "boot",      "-",            "chargeur + dsp_power_on() du firmware, IDLE, version API", t_boot },
    { "idle",      "0 (aucune)",   "scenario vide chaque trame : IT trame, retour IDLE, pas d'erreur", t_idle },
    { "checksum",  "33 CHECKSUM",  "dsp_checksum_task() : version/somme du code dans a_pm", t_checksum },
    { "fb0",       "5 FB, mode 0", "detection FCCH (d_fb_det, a_sync_demod : TOA -> trame, angle -> Hz)", t_fb0 },
    { "fb1",       "5 FB, mode 1", "detection FCCH en poursuite", t_fb1 },
    { "fbpos",     "5 FB",         "TOA FB, FCCH dans la 1re trame de recherche (FBPOS_MODE, BANC_FB_DECAL)",
 ↪  t_fb_pos },
    { "fb-afc",    "5 FB, mode 0", "erreur de frequence mesuree avec un VCXO decale (+60 LSB)", t_fb_afc },
    { "sb",        "6 SB",         "SCH : CRC (a_sch[0]), BSIC et T1/T2/T3 (a_sch[3..4]) sur 20 SCH", t_sb },
    { "fbsb",      "5,5,6",        "acquisition complete FB0 -> FB1 -> SB (prim_fbsb.c)", t_fbsb },
    { "nb",        "24 ALLC",      "BCCH : 8 blocs, a_cd FIRE + 23 octets == gsm0503_xcch_encode", t_nb_bcch },
    { "rach",      "10 RACH",      "access-burst montant : codage ROM (0x3f8a) et burst final (script TSP) ==
 ↪  05.03/05.02", t_rach },
    { "tx-sdcch",  "12 DUL",       "SDCCH montant : a_cu -> bursts 0x3f8a == gsm0503_xcch_encode", t_tx_sdcch },
    { "tx-sdcch-a5","12 DUL + A5", "SDCCH montant chiffre : bursts 0x3f8a == xcch_encode XOR osmo_a5(kc, fn de la
 ↪  commande)", t_tx_sdcch_a5 },
};
#define N_TESTS (int)(sizeof TESTS / sizeof TESTS[0])

/* Un test dans un fils, depuis le DSP demarre. */
static int lancer(const Test *t)
{
    int tube[2];
```

```
    /* DSP_TESTER_NOFORK=1 : le test tourne dans ce processus (gdb, gprof) ; un plantage emporte tout. */
    static int nofork = -1;
    if (nofork < 0) { const char *e = getenv("DSP_TESTER_NOFORK"); nofork = (e && *e == '1') ? 1 : 0; }
    if (nofork) {
        g_dl = 0; g_detail[0] = 0;
        int v = t->f();
        printf("[%-8s] %-10s %-14s %s\n", NOMV[v], t->nom, t->tache, g_detail);
        return v;
    }
    if (pipe(tube) < 0) { perror("pipe"); return V_FAIL; }
    fflush(stdout);
    pid_t p = fork();
    if (p == 0) {
        close(tube[0]);
        alarm(900);
        g_dl = 0; g_detail[0] = 0;
        int v = t->f();
        fflush(stdout);
        char buf[sizeof g_detail + 4];
        buf[0] = (char)v;
        memcpy(buf + 1, g_detail, g_dl + 1);
        if (write(tube[1], buf, g_dl + 2) < 0) _exit(3);
        close(tube[1]);
        _exit(0);
    }
    close(tube[1]);
    static char buf[sizeof g_detail + 4];
    ssize_t n = 0, r;
    while ((r = read(tube[0], buf + n, sizeof buf - 1 - (size_t)n)) > 0) n += r;
    close(tube[0]);
    int st = 0;
    waitpid(p, &st, 0);
    int v;
    const char *d;
    char crash[200];
    if (n >= 2) { v = buf[0]; buf[n] = 0; d = buf + 1; }
    else {
        v = V_FAIL;
        snprintf(crash, sizeof crash, "le test s'est arrete sans verdict (%s %d) : plantage du coeur ou delai
↪   depasse",
                 WIFSIGNALED(st) ? "signal" : "code", WIFSIGNALED(st) ? WTERMSIG(st) : WEXITSTATUS(st));
        d = crash;
    }
    printf("[%-8s] %-10s %-14s ", NOMV[v], t->nom, t->tache);
    /* premiere ligne du detail sur la ligne du verdict, le reste indente */
    const char *q = d;
    while (*q) {
        const char *e = strchr(q, '\n');
        size_t l = e ? (size_t)(e - q) : strlen(q);
        if (q != d) printf("%38s", "");
        printf("%.*s\n", (int)l, q);
        q += l + (e ? 1 : 0);
    }
    if (!*d) printf("\n");
    return v;
}

static void usage(const char *p)
{
    printf("usage : %s [-v] [--trace 0..6] [--rom-dir DIR] [--insns N] [--bsic N] (--all | --list | TEST...)\n",
↪   p);
}

int main(int argc, char **argv)
{
    int tous = 0, liste = 0, n_choix = 0;
```

```
    const char *choix[64];
    for (int i = 1; i < argc; i++) {
        const char *a = argv[i];
        if (!strcmp(a, "-v")) g_v = 1;
        else if (!strcmp(a, "--all")) tous = 1;
        else if (!strcmp(a, "--list")) liste = 1;
        else if (!strcmp(a, "--rom-dir") && i + 1 < argc) banc_cfg.rom_dir = argv[++i];
        else if (!strcmp(a, "--insns") && i + 1 < argc) banc_cfg.budget = atol(argv[++i]);
        else if (!strcmp(a, "--bsic") && i + 1 < argc) banc_bsic = (uint8_t)(atoi(argv[++i]) & 0x3f);
        else if (!strcmp(a, "--trace") && i + 1 < argc) g_trace = atoi(argv[++i]);
        else if (!strcmp(a, "-h") || !strcmp(a, "--help")) { usage(argv[0]); return 0; }
        else if (a[0] == '-') { usage(argv[0]); return 255; }
        else if (n_choix < 64) choix[n_choix++] = a;
    }
    if (liste) {
       for (int i = 0; i < N_TESTS; i++) printf("  %-10s %-14s %s\n", TESTS[i].nom, TESTS[i].tache, TESTS[i].desc);
        return 0;
    }
    if (!tous && !n_choix) { usage(argv[0]); return 255; }
    for (int k = 0; k < n_choix; k++) {
        int vu = 0;
        for (int i = 0; i < N_TESTS; i++) if (!strcmp(choix[k], TESTS[i].nom)) vu = 1;
        if (!vu) { fprintf(stderr, "test inconnu : %s (--list)\n", choix[k]); return 255; }
    }
    banc_cfg.verbeux = g_v;
    /* Les sondes du coeur ecrivent sur stderr ; filtrees comme dans c54x_exe
     * (src/verbosite.c). Les fils ecrivent dans le meme tube, le fil de
     * filtrage reste dans le pere. */
    verbosite_installer(g_trace);
    if (banc_init() < 0) return 255;
    g_boot_rc = banc_dsp_power_on(&g_boot);
    printf("dsp_tester : ROM %s (API %04x), %ld insn/trame, BSIC %u ; l'ARM et la BTS sont simules, rien n'est
↪   envoye au dehors\n",
           banc_cfg.rom_dir, g_boot.version1, banc_cfg.budget, banc_bsic);
    int n[3] = { 0, 0, 0 };
    for (int i = 0; i < N_TESTS; i++) {
        int pris = tous;
        for (int k = 0; k < n_choix; k++) if (!strcmp(choix[k], TESTS[i].nom)) pris = 1;
        if (!pris) continue;
        n[lancer(&TESTS[i])]++;
    }
    printf("bilan : %d PASS, %d FAIL, %d NON-IMPL\n", n[0], n[1], n[2]);
    verbosite_retirer();
    if (g_trace) verbosite_bilan(stdout);
    return n[1] > 254 ? 254 : n[1];
}
```

3.35 /opt/GSM/c54x_exe/tools/dsp_txt2bin.py

5630 octets, 156 lignes → 156 lignes

```
#!/usr/bin/env python3
"""
dsp_txt2bin.py — Convert calypso_dsp.txt (OsmocomBB Compal DSP dumper format)
to raw 16-bit little-endian binary.

Input format expected (per `c54x_load_rom`, calypso_c54x.c:9798-9854):

    DSP dump: PROM0
    07000 : XXXX XXXX XXXX ... XXXX
    07010 : XXXX XXXX XXXX ... XXXX
    ...
    DSP dump: PROM1
    18000 : ...
```

```
Each data line is "ADDR : WORD WORD WORD ..." where ADDR is hex (no prefix)
and each WORD is 4 hex chars. Sections: regs, DROM, PDROM, PROM0..PROM3.

Usage:
  dsp_txt2bin.py INPUT.txt                      # summary only, no output
  dsp_txt2bin.py INPUT.txt OUT.bin               # emit all sections → OUT.SECTION.bin
  dsp_txt2bin.py INPUT.txt OUT.bin --section S   # emit one section, zero-filled
  dsp_txt2bin.py INPUT.txt OUT.bin --slice A L   # emit L words from addr A (hex/dec OK)

Output is raw LE words. Loadable as -M calypso,dsp-blob=OUT.bin if the slice
falls in DARAM's OVLY-accessible range (0x80..0x27FF) and the PC override is
honored.

Examples:
  # split-by-section dump (OUT.bin base → OUT.PROM0.bin, OUT.PROM1.bin, ...)
  dsp_txt2bin.py calypso_dsp.txt calypso_dsp.bin

  # extract first 0x100 words of PROM0 (0x7000..0x70FF) as a probe blob
  dsp_txt2bin.py calypso_dsp.txt prom0_head.bin --slice 0x7000 0x100

  # full PROM0 dump
  dsp_txt2bin.py calypso_dsp.txt prom0.bin --section PROM0
"""

import argparse
import re
import struct
import sys
from collections import defaultdict

SECTION_RE = re.compile(r"DSP dump:\s+(\w+)")
DATA_RE    = re.compile(r"^\s*([0-9a-fA-F]+)\s*:\s*(.+?)\s*$")
WORD_RE    = re.compile(r"[0-9a-fA-F]{1,4}")


def parse_txt(path):
    """Return dict {section_name: {addr: word}}."""
    sections = defaultdict(dict)
    cur = None
    with open(path) as fp:
        for line in fp:
            m = SECTION_RE.search(line)
            if m:
                cur = m.group(1)
                continue
            if cur is None:
                continue
            m = DATA_RE.match(line)
            if not m:
                continue
            base = int(m.group(1), 16)
            for i, w in enumerate(WORD_RE.findall(m.group(2))):
                sections[cur][base + i] = int(w, 16) & 0xFFFF
    return sections


def merge(sections):
    """Flatten all sections into one {addr: word} dict (last writer wins)."""
    out = {}
    for s in sections.values():
        out.update(s)
    return out


def emit_slice(words, start_addr, length, out_path):
```

```
    with open(out_path, "wb") as fp:
        for a in range(start_addr, start_addr + length):
            fp.write(struct.pack("<H", words.get(a, 0)))
    filled = sum(1 for a in range(start_addr, start_addr + length) if a in words)
    print(f"[txt2bin] slice {length} words at {start_addr:#06x} → {out_path} "
          f"({filled} filled, {length - filled} zero-filled gaps)")


def emit_section(words, name, out_path):
    if not words:
        sys.exit(f"[txt2bin] section '{name}' is empty")
    lo, hi = min(words), max(words)
    n = hi - lo + 1
    with open(out_path, "wb") as fp:
        for a in range(lo, hi + 1):
            fp.write(struct.pack("<H", words.get(a, 0)))
    filled = len(words)
    print(f"[txt2bin] section {name}: {n} words [{lo:#06x}..{hi:#06x}] → "
          f"{out_path} ({filled} filled, {n - filled} zero-filled gaps)")


def main():
    ap = argparse.ArgumentParser(
        description="Convert calypso_dsp.txt to raw 16-bit LE binary.",
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog=__doc__)
    ap.add_argument("input", help="path to calypso_dsp.txt")
    ap.add_argument("output", nargs="?", help="output .bin path")
    grp = ap.add_mutually_exclusive_group()
    grp.add_argument("--section", help="emit one section (regs/DROM/PDROM/PROM0..3)")
    grp.add_argument("--slice", nargs=2, metavar=("ADDR", "LEN"),
                     help="emit LEN words starting at ADDR (hex or decimal)")
    args = ap.parse_args()

    sections = parse_txt(args.input)

    if not args.output:
        print(f"[txt2bin] sections found in {args.input}:")
        for name in sorted(sections):
            ws = sections[name]
            if not ws:
                print(f"  {name}: (empty)")
                continue
            lo, hi = min(ws), max(ws)
            print(f"  {name:8s}: {len(ws):6d} words, "
                  f"{lo:#07x}..{hi:#07x}  (span {hi - lo + 1})")
        return

    if args.section:
        if args.section not in sections:
            sys.exit(f"[txt2bin] no section named '{args.section}'. "
                     f"Available: {', '.join(sorted(sections))}")
        emit_section(sections[args.section], args.section, args.output)
    elif args.slice:
        addr = int(args.slice[0], 0)
        length = int(args.slice[1], 0)
        emit_slice(merge(sections), addr, length, args.output)
    else:
        # No flags + output given → split each section into OUTPUT.SECTION.bin.
        # Sections live in different address spaces (data vs program) and may
        # overlap in DSP address, so a single merged flat file would be
        # ambiguous. One file per section keeps things semantically clean.
        base = args.output
        if base.lower().endswith(".bin"):
            base = base[:-4]
        any_emitted = False
```

```
        for name in sorted(sections):
            if not sections[name]:
                continue
            emit_section(sections[name], name, f"{base}.{name}.bin")
            any_emitted = True
        if not any_emitted:
            sys.exit("[txt2bin] no non-empty sections found in input")


if __name__ == "__main__":
    main()
```

3.36 /opt/GSM/c54x_exe/tools/injecteur_bruit.py

32633 octets, 708 lignes → 708 lignes

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# injecteur_bruit.py - proxy UDP qui degrade le sens DESCENDANT entre le pont
# (osmo-operator/pont) et la couche 1 du telephone emule.
#
# [2026-10-03] Cree pour le banc Calypso. Un seul fichier source :
#   /opt/GSM/qosmo/tools/injecteur_bruit.py
# (c54x_exe/run.sh le prend la, avec repli sur c54x_exe/tools/injecteur_bruit.py,
# un lien vers ce fichier ; qosmo/run_modules/38-bruit.sh le lance pour le plan
# de qosmo).
#
# OU IL S'INTERCALE
#   cible dsp   : pont_dsp.py --dsp-port 6702 -> [injecteur 127.0.0.1:6702]
#                 -> BSP de c54x_exe deplace sur CALYPSO_BSP_PORT (16702 par
#                 defaut). Format : 8 octets d'en-tete [tn, fn BE32, att, 0, 0]
#                 puis 148 octets de bits DURS 0/1 (pont/trx.py Trx.run_data ;
#                 calypso_bsp.c bsp_trxd_readable).
#                 Le BSP apprend l'adresse de son pair montant sur l'emetteur du
#                 descendant : une fois l'injecteur intercale, ce qu'il renvoie
#                 (bursts UL TRXDv0, s'il en emet) arrive ici et repart SANS
#                 MODIFICATION vers la source (le pont, 127.0.0.1:5702), depuis
#                 le port d'ecoute -- exactement l'adresse qu'il avait avant.
#   cible grgsm : pont.py -> GSMTAP [injecteur 127.0.0.1:14730] -> L1 gr-gsm de
#                 QEMU (127.0.0.1:4730, port fige dans calypso_l1_grgsm.c).
#                 Le pont y envoie si PONT_GSMTAP_PORT=14730 (pont/config.py ;
#                 pont/pont.py le pose seul quand BRUIT_MODE vaut ber|relais).
#                 Format : en-tete GSMTAP v2 de 16 octets puis le bloc L2 DEJA
#                 DECODE (23 octets) -- PAS des bursts, PAS de valeurs souples.
#                 Le SCH (4731, « SCH2 ») et la parole TCH (/dev/shm/calypso_tch_dl)
#                 ne passent pas par ici et ne sont pas touches.
#
# MODES (sens descendant seulement ; le montant est toujours relaye tel quel)
#   relais  aucune alteration : sert a verifier que l'intercalation seule ne
#           change rien.
#   ber     inversion aleatoire de bits, taux moyen --ber. --rafales L : erreurs
#           groupees (Gilbert-Elliott, rafales de L bits en moyenne, taux
#           --ber-rafale dans une rafale), meme taux moyen.
#           dsp : bits durs inverses (0<->1). grgsm : bits des octets L2 inverses
#           APRES decodage -- non physique (ni code correcteur ni CRC ne
#           protegent plus le bloc) ; --perte est le modele realiste en grgsm.
#   souple  AWGN a --snr-db sur les valeurs souples. VERIFIE LE 2026-10-03 :
#           aucun des deux flux n'en porte (dsp : bits durs 0/1 ; grgsm : octets
#           L2). Sur des bits durs : modulation antipodale +-1, AWGN, decision
#           dure -- BER = Q(sqrt(2 snr)), annonce au demarrage ; la sortie reste
#           0/1, le seul format que le BSP sait lire. Si une source envoie un jour
#           des valeurs souples int8 (convention TRXD : +127 = bit 1), elles sont
#           bruitees et gardees souples. Refuse en grgsm.
#   iq      EXPERIMENTAL. Bits -> GMSK BT=0.3 a 4 echantillons/symbole (meme
```

```
#           modulateur que calypso_gmsk.c, instant 0,5) + AWGN a --snr-db, envoye
#           en I/Q int16 entrelaces (IQ_PASSTHROUGH du BSP, decimation par 4).
#           CONTOURNE la modulation reglee du BSP (sb_moduler pour le SCH :
#           instant 0,35, elargissement 1,0, bruit propre ; gmsk_elargir 0,3 sur
#           les NB -- seul ce dernier est imite ici, --iq-elargir).
#           INCOMPATIBLE avec CALYPSO_BSP_STREAM=1 (le defaut de
#           start-native.sh --dsp) : en STREAM le BSP range les 148 premiers
#           octets du paquet comme des bits AVANT la branche IQ_PASSTHROUGH
#           (calypso_bsp.c bsp_trxd_readable -> bsp_ts0_stocker). Refuse dans ce
#           cas, sauf --forcer-iq. Necessite numpy.
#   --perte P (tous modes) : un burst/bloc sur 1/P est efface. dsp : la charge
#           est remplacee par des bits aleatoires (evanouissement : le BSP garde
#           sa cadence, un burst jete le ferait attendre CALYPSO_BSP_ATTENTE_MS).
#           grgsm : le bloc n'est pas transmis (= echec CRC cote L1).
#
# DETERMINISME : chaque paquet tire son alea d'une graine derivee de
# (--graine, fn, tn) : meme graine + memes trames = memes erreurs, quel que soit
# l'ordre d'arrivee (sauf --rafales, dont l'etat suit l'ordre d'arrivee). Sans
# --graine, une graine est tiree et ECRITE au demarrage pour pouvoir rejouer.
#
# Arret propre sur SIGTERM/SIGINT (statistiques finales), SIGUSR1 = statistiques
# tout de suite. --verifier : controle la configuration et sort (0 = bon).
import argparse
import logging
import math
import os
import random
import selectors
import signal
import socket
import struct
import sys
import time

VERSION = "2026-10-03"
log = logging.getLogger("bruit")

PROFILS = {
    # cible : (ecoute = la ou la source envoie, vers = la couche 1)
    "dsp": ("127.0.0.1:6702", "127.0.0.1:16702"),
    "grgsm": ("127.0.0.1:14730", "127.0.0.1:4730"),
}
MODES = ("relais", "ber", "souple", "iq")
MODES_GRGSM = ("relais", "ber")

DSP_HDR = 8              # [tn, fn BE32, att, 0, 0]
NBITS = 148
IQ_MIN_OCTETS = 4 * 146  # seuil de calypso_bsp.c pour lire la charge en I/Q
GSMTAP_TYPE_UM = 0x01
MASQUE64 = (1 << 64) - 1


def _adresse(txt, option):
    hote, sep, port = txt.rpartition(":")
    if not sep or not hote:
        raise argparse.ArgumentTypeError("%s : HOTE:PORT attendu, pas %r" % (option, txt))
    try:
        p = int(port)
    except ValueError:
        raise argparse.ArgumentTypeError("%s : port invalide %r" % (option, port))
    if not 0 < p < 65536:
        raise argparse.ArgumentTypeError("%s : port hors bornes %d" % (option, p))
    return (hote, p)


def _q_gauss(x):
```

```
    """Q(x) = P(N(0,1) > x)."""
    return 0.5 * math.erfc(x / math.sqrt(2.0))


def _geometrique(rng, p):
    """Nombre d'essais jusqu'au premier succes (>= 1), succes de probabilite p."""
    if p >= 1.0:
        return 1
    if p <= 0.0:
        return 1 << 62
    u = rng.random()
    return 1 + int(math.log1p(-u) / math.log1p(-p))


class Rafales:
    """Gilbert-Elliott : etat bon (aucune erreur) / mauvais (taux q), sejours
    geometriques. Longueur moyenne d'une rafale L bits, taux moyen p."""

    def __init__(self, p, longueur, q):
        self.q = q
        pi_m = p / q                          # fraction du temps en rafale
        self.p_mb = 1.0 / longueur            # mauvais -> bon
        self.p_bm = self.p_mb * pi_m / (1.0 - pi_m) if pi_m < 1.0 else 1.0
        self.mauvais = False
        self.restant = None

    def positions(self, n, rng):
        if self.restant is None:
            self.restant = _geometrique(rng, self.p_bm)
        out = []
        pos = 0
        while pos < n:
            k = min(self.restant, n - pos)
            if self.mauvais:
                out.extend(i for i in range(pos, pos + k) if rng.random() < self.q)
            pos += k
            self.restant -= k
            if self.restant == 0:
                self.mauvais = not self.mauvais
                self.restant = _geometrique(rng, self.p_mb if self.mauvais else self.p_bm)
        return out


def positions_iid(n, p, rng):
    """Positions des bits inverses, independants de probabilite p (saut geometrique)."""
    if p <= 0.0:
        return []
    if p >= 1.0:
        return list(range(n))
    out = []
    lp = math.log1p(-p)
    i = -1
    while True:
        i += 1 + int(math.log1p(-rng.random()) / lp)
        if i >= n:
            return out
        out.append(i)


class ModulateurGmsk:
    """GMSK BT=0.3 (45.004) a `sps` echantillons par symbole, comme
    calypso_gmsk.c gmsk_moduler() : codage differentiel d_i = b_i ^ b_{i-1}
    (b_{-1} = 1), alpha_i = 1 - 2 d_i, phase(t) = pi/2 * sum alpha_i q(t - i),
    echantillon j a l'instant t = j/sps + decalage. Avec sps=4 et decalage=0,5,
    les echantillons 4k (ceux que garde le BSP, decimation par 4) sont ceux de
    gmsk_moduler(..., decalage=0.5)."""
```

```
    def __init__(self, np, sps=4, decalage=0.5):
        self.np = np
        self.sps = sps
        # La table q(t) est calculee EXACTEMENT comme calypso_gmsk.c q_init() :
        # somme de Riemann a gauche, pas T/16, normalisee, puis interpolation
        # lineaire (q_de). Une integration plus fine donnerait une autre forme
        # d'onde (q(0) = 0,500 au lieu de 0,477 : jusqu'a 0,05 rad d'ecart) et
        # le mode iq ne serait plus « le meme modulateur » que le BSP.
        bt, os_, lg = 0.3, 16, 3
        k = 2.0 * math.pi * bt / math.sqrt(math.log(2.0))
        dt = 1.0 / os_
        tab, acc = [], 0.0
        for i in range(lg * os_ + 1):
            t = -1.5 + i * dt
            tab.append(acc)
            acc += (_q_gauss(k * (t - 0.5)) - _q_gauss(k * (t + 0.5))) * dt
        tab = [v / acc for v in tab]

        def q(t):
            if t <= -1.5:
                return 0.0
            if t >= 1.5:
                return 1.0
            x = (t + 1.5) * os_
            i = int(x)
            f = x - i
            if i >= lg * os_:
                return 1.0
            return tab[i] * (1.0 - f) + tab[i + 1] * f

        # K(n) = q(n/sps + decalage) : 0 pour t <= -1,5, 1 pour t >= 1,5.
        n_lo = math.floor((-1.5 - decalage) * sps) + 1
        n_un = math.ceil((1.5 - decalage) * sps)
        self.n_lo = n_lo
        self.n_un = n_un
        self.k_trans = np.array([q(n / sps + decalage) for n in range(n_lo, n_un)])

    def phase(self, bits):
        np = self.np
        b = np.asarray(bits, dtype=np.int8) & 1
        prec = np.concatenate(([1], b[:-1]))
        alpha = 1.0 - 2.0 * (b ^ prec)
        n = len(b)
        long_ = n * self.sps
        d = np.zeros(long_)
        d[:: self.sps] = alpha
        conv = np.convolve(d, self.k_trans)               # conv[k] <-> j = k + n_lo
        j = np.arange(long_)
        part_trans = conv[j - self.n_lo]
        cumul = np.cumsum(d)
        idx = j - self.n_un
        part_marche = np.where(idx >= 0, cumul[np.clip(idx, 0, None)], 0.0)
        return (math.pi / 2.0) * (part_trans + part_marche)


class Stats:
    def __init__(self):
        self.t0 = time.monotonic()
        self.t_prec = self.t0
        self.dl_prec = 0
        self.dl = 0            # paquets descendants recus
        self.relayes = 0       # paquets descendants envoyes vers la L1
        self.alteres = 0       # paquets dont la charge a ete degradee
        self.effaces = 0       # --perte
        self.hors_tn = 0       # hors --tn, relayes tels quels
```

```
        self.autres = 0        # format non reconnu, relayes tels quels
        self.iq_deja = 0       # charge deja en I/Q, relayee telle quelle
        self.iq_produits = 0   # bursts convertis en I/Q (mode iq)
        self.souples = 0       # paquets a valeurs souples vus
        self.bits = 0          # bits durs examines (ber, souple sur bits durs)
        self.inverses = 0      # bits durs changes
        self.ul = 0            # paquets montants relayes vers la source
        self.ul_sans_dest = 0  # montant recu avant tout descendant
        self.err_envoi = 0

    def ligne(self, final=False):
        now = time.monotonic()
        dt = max(now - self.t_prec, 1e-9)
        pps = (self.dl - self.dl_prec) / dt
        self.t_prec, self.dl_prec = now, self.dl
        ber = (self.inverses / self.bits) if self.bits else 0.0
        return ("%s t=%.0fs DL recus=%d relayes=%d alteres=%d effaces=%d hors_tn=%d autres=%d iq_deja=%d"
                " iq_produits=%d souples=%d | bits=%d inverses=%d BER=%.3e | UL relayes=%d sans_dest=%d"
                " | erreurs_envoi=%d | %.0f pq/s"
                % ("BILAN" if final else "STATS", now - self.t0, self.dl, self.relayes, self.alteres,
                   self.effaces, self.hors_tn, self.autres, self.iq_deja, self.iq_produits, self.souples,
                   self.bits, self.inverses, ber, self.ul, self.ul_sans_dest, self.err_envoi, pps))


class Degradeur:
    def __init__(self, a, np=None):
        self.a = a
        self.cible = a.cible
        self.mode = a.mode
        self.stats = Stats()
        self.graine = a.graine
        self.tn = a.tn_set
        self.rafales = Rafales(a.ber, a.rafales, a.ber_rafale) if (a.mode == "ber" and a.rafales) else None
        self.snr_lin = 10.0 ** (a.snr_db / 10.0) if a.snr_db is not None else None
        self.p_souple = _q_gauss(math.sqrt(2.0 * self.snr_lin)) if self.snr_lin is not None else 0.0
        self.np = np
        self.mod = ModulateurGmsk(np) if a.mode == "iq" else None

    def _rng(self, fn, tn, sel=0):
        x = (self.graine * 0x9E3779B97F4A7C15 + (fn + 1) * 0x100000001B3 + tn * 0x2545F491 + sel * 0x632BE5AB) &
↪   MASQUE64
        return random.Random(x)

    # -- cible dsp ------------------------------------------------------------
    def _dsp(self, pkt):
        st = self.stats
        if len(pkt) < DSP_HDR + NBITS:
            st.autres += 1
            return pkt
        charge = len(pkt) - DSP_HDR
        if charge >= IQ_MIN_OCTETS:
            st.iq_deja += 1
            return pkt
        tn = pkt[0] & 0x07
        if self.tn is not None and tn not in self.tn:
            st.hors_tn += 1
            return pkt
        fn = struct.unpack_from(">I", pkt, 1)[0]
        rng = self._rng(fn, tn)
        ba = bytearray(pkt)
        dur = not bytes(ba[DSP_HDR:]).translate(None, b"\x00\x01")   # que des 0/1 : bits durs
        if not dur:
            st.souples += 1
        efface = self.a.perte > 0.0 and rng.random() < self.a.perte
        if efface:
            st.effaces += 1
```

```
            if dur:
                for i in range(DSP_HDR, len(ba)):
                    ba[i] = rng.getrandbits(1)
            else:
                for i in range(DSP_HDR, len(ba)):
                    ba[i] = 0                                   # souple : effacement = aucune information
        mode = self.mode
        if mode == "relais":
            if efface:
                st.alteres += 1
            return bytes(ba)
        if mode == "ber":
            n = len(ba) - DSP_HDR
            pos = self.rafales.positions(n, rng) if self.rafales else positions_iid(n, self.a.ber, rng)
            if dur:
                for i in pos:
                    ba[DSP_HDR + i] ^= 1
            else:
                for i in pos:                                   # souple : signe inverse
                    v = ba[DSP_HDR + i]
                    v = v - 256 if v > 127 else v
                    ba[DSP_HDR + i] = min(127, -v) & 0xFF
            st.bits += n
            st.inverses += len(pos)
            st.alteres += 1
            return bytes(ba)
        if mode == "souple":
            snr = self.snr_lin
            if dur:
                # +-1 + N(0, sigma^2), sigma^2 = 1/(2 snr), puis decision dure :
                # chaque bit change independamment avec la probabilite
                # Q(1/sigma) = Q(sqrt(2 snr)). On tire donc directement ces
                # positions -- meme loi, exactement, sans 148 gaussiennes.
                n = len(ba) - DSP_HDR
                pos = positions_iid(n, self.p_souple, rng)
                for i in pos:
                    ba[DSP_HDR + i] ^= 1
                st.bits += n
                st.inverses += len(pos)
            else:
                vals = [(v - 256 if v > 127 else v) for v in ba[DSP_HDR:]]
                amp = (sum(abs(v) for v in vals) / len(vals)) or 127.0
                sigma = amp / math.sqrt(2.0 * snr)
                for k, v in enumerate(vals):
                    w = int(round(v + rng.gauss(0.0, sigma)))
                    ba[DSP_HDR + k] = max(-127, min(127, w)) & 0xFF
            st.alteres += 1
            return bytes(ba)
        # mode iq
        if dur:
            bits = list(ba[DSP_HDR:DSP_HDR + NBITS])
        else:                                                   # souple int8 : decision dure (> 0 = 1)
            bits = [1 if 0 < v <= 127 else 0 for v in ba[DSP_HDR:DSP_HDR + NBITS]]
        return bytes(ba[:DSP_HDR]) + self._iq(bits, fn, tn)

    def _iq(self, bits, fn, tn):
        np = self.np
        a = self.a
        ph = self.mod.phase(bits)
        i_ = a.amp * np.cos(ph)
        q_ = a.amp * np.sin(ph)
        if a.iq_elargir and any(bits):
            # Imite gmsk_elargir() du BSP sur les echantillons qu'il garde
            # (voisins a +-1 symbole = +-sps echantillons) ; FCCH (tout a zero)
            # laisse tel quel, comme le BSP.
            s, e = self.mod.sps, a.iq_elargir
```

```
            for v in (i_, q_):
                w = v.copy()
                w[s:] += e * v[:-s]
                w[:-s] += e * v[s:]
                v[:] = w / (1.0 + 2.0 * e)
        if self.snr_lin is not None:
            sigma = math.sqrt(a.amp * a.amp / (2.0 * self.snr_lin))
            g = np.random.Generator(np.random.PCG64(self._rng(fn, tn, 1).getrandbits(64)))
            i_ = i_ + g.normal(0.0, sigma, i_.shape)
            q_ = q_ + g.normal(0.0, sigma, q_.shape)
        out = np.empty(2 * len(i_), dtype="<i2")
        out[0::2] = np.clip(np.rint(i_), -32768, 32767)
        out[1::2] = np.clip(np.rint(q_), -32768, 32767)
        self.stats.iq_produits += 1
        self.stats.alteres += 1
        return out.tobytes()

    # -- cible grgsm ------------------------------------------------------------
    def _grgsm(self, pkt):
        st = self.stats
        if len(pkt) < 16 or pkt[0] != 2 or pkt[2] != GSMTAP_TYPE_UM:
            st.autres += 1
            return pkt
        hdr = pkt[1] * 4
        if hdr < 16 or hdr >= len(pkt):
            st.autres += 1
            return pkt
        tn = pkt[3] & 0x07
        if self.tn is not None and tn not in self.tn:
            st.hors_tn += 1
            return pkt
        fn = struct.unpack_from(">I", pkt, 8)[0]
        rng = self._rng(fn, tn, pkt[12])
        if self.a.perte > 0.0 and rng.random() < self.a.perte:
            st.effaces += 1
            return None                                          # bloc perdu = echec CRC cote L1
        if self.mode == "relais":
            return pkt
        ba = bytearray(pkt)
        n = (len(ba) - hdr) * 8
        pos = self.rafales.positions(n, rng) if self.rafales else positions_iid(n, self.a.ber, rng)
        for i in pos:
            ba[hdr + (i >> 3)] ^= 0x80 >> (i & 7)
        st.bits += n
        st.inverses += len(pos)
        st.alteres += 1
        return bytes(ba)

    def descendant(self, pkt):
        self.stats.dl += 1
        return self._dsp(pkt) if self.cible == "dsp" else self._grgsm(pkt)


def _parser():
    p = argparse.ArgumentParser(
        prog="injecteur_bruit.py",
        description="Proxy UDP transparent pont -> couche 1 qui degrade le sens descendant "
                    "(bits inverses, AWGN, I/Q bruite). Le montant est relaye sans modification.")
    p.add_argument("--cible", choices=sorted(PROFILS), default="dsp",
                   help="format du flux : dsp (TRXD 8 o + 148 bits, BSP de c54x_exe) ou grgsm "
                        "(GSMTAP vers la L1 gr-gsm de QEMU) ; defaut dsp")
    p.add_argument("--mode", choices=MODES, default="ber", help="defaut ber")
    p.add_argument("--ecoute", help="HOTE:PORT ou la source envoie (dsp 127.0.0.1:6702, grgsm 127.0.0.1:14730)")
    p.add_argument("--vers", help="HOTE:PORT de la couche 1 (dsp 127.0.0.1:16702, grgsm 127.0.0.1:4730)")
    p.add_argument("--source", help="HOTE:PORT local d'emission vers la L1 (defaut : port libre)")
    p.add_argument("--retour", help="HOTE:PORT ou renvoyer le montant (defaut : l'emetteur du dernier descendant)")
```

```
    p.add_argument("--ber", type=float, default=0.0, help="mode ber : taux moyen d'inversion (0..1)")
    p.add_argument("--rafales", type=float, default=0.0, metavar="L",
                   help="mode ber : erreurs en rafales de L bits en moyenne (Gilbert-Elliott)")
    p.add_argument("--ber-rafale", type=float, default=0.5, help="taux d'erreur dans une rafale (defaut 0,5)")
    p.add_argument("--snr-db", type=float, default=None,
                   help="modes souple et iq : SNR en dB, P_signal / (2 sigma^2) par echantillon complexe "
                        "(meme definition que CALYPSO_BSP_SNR_DB)")
    p.add_argument("--perte", type=float, default=0.0,
                  help="probabilite d'effacer un burst (dsp : bits aleatoires) ou un bloc (grgsm : non transmis)")
    p.add_argument("--tn", default="tous", help="intervalles a degrader, ex. 0 ou 1,2 (defaut tous)")
    p.add_argument("--graine", type=lambda s: int(s, 0), default=None, help="graine (deterministe)")
    p.add_argument("--amp", type=int, default=30000, help="mode iq : amplitude (defaut 30000, comme le BSP)")
    p.add_argument("--iq-elargir", type=float, default=0.3,
                   help="mode iq : elargissement des NB comme gmsk_elargir (defaut 0,3 ; 0 = aucun)")
    p.add_argument("--forcer-iq", action="store_true",
                   help="mode iq malgre CALYPSO_BSP_STREAM=1 ou CALYPSO_BSP_IQ_PASSTHROUGH=0")
    p.add_argument("--stats", type=float, default=10.0, help="periode des statistiques en s (0 = seulement le
↪   bilan)")
    p.add_argument("--journal", help="fichier journal (en plus de stderr)")
    p.add_argument("--pidfile", help="ecrit le pid, efface a l'arret")
    p.add_argument("--etiquette", default="", help="libre, pour reconnaitre le processus (pkill -f)")
    p.add_argument("--verifier", action="store_true", help="controle la configuration et sort")
    p.add_argument("--version", action="version", version="%(prog)s " + VERSION)
    return p


def _valider(a, p):
    """Complete et controle les arguments ; rend numpy (mode iq) ou None."""
    ecoute, vers = PROFILS[a.cible]
    a.ecoute = _adresse(a.ecoute or ecoute, "--ecoute")
    a.vers = _adresse(a.vers or vers, "--vers")
    a.source = _adresse(a.source, "--source") if a.source else None
    a.retour = _adresse(a.retour, "--retour") if a.retour else None
    if a.ecoute == a.vers:
        p.error("--ecoute et --vers designent la meme adresse %s:%d : l'injecteur s'enverrait ses propres paquets"
↪   % a.ecoute)
    if a.cible == "grgsm" and a.mode not in MODES_GRGSM:
        p.error("cible grgsm : mode %s impossible. Le flux GSMTAP vers QEMU porte des blocs L2 deja decodes "
                "(16 o d'en-tete + 23 o), ni valeurs souples ni bursts a moduler (calypso_l1_grgsm.c "
                "gsmtap_readable). Modes possibles : %s (et --perte)." % (a.mode, ", ".join(MODES_GRGSM)))
    for nom in ("ber", "perte", "ber_rafale"):
        v = getattr(a, nom)
        if not 0.0 <= v <= 1.0:
            p.error("--%s doit etre entre 0 et 1 (%g)" % (nom.replace("_", "-"), v))
    if a.mode == "ber" and a.rafales:
        if a.rafales < 1.0:
            p.error("--rafales : longueur moyenne en bits >= 1 (%g)" % a.rafales)
        if not 0.0 < a.ber_rafale <= 1.0 or a.ber >= a.ber_rafale:
            p.error("--rafales : il faut --ber (%g) < --ber-rafale (%g) <= 1" % (a.ber, a.ber_rafale))
    if a.mode == "souple" and a.snr_db is None:
        p.error("mode souple : --snr-db obligatoire")
    if a.snr_db is not None and not -50.0 <= a.snr_db <= 100.0:
        p.error("--snr-db hors bornes raisonnables (%g)" % a.snr_db)
    if a.tn in ("", "tous", "all"):
        a.tn_set = None
    else:
        try:
            a.tn_set = {int(x) for x in a.tn.split(",") if x.strip() != ""}
        except ValueError:
            p.error("--tn : liste d'intervalles 0..7 attendue (%r)" % a.tn)
        if not a.tn_set or any(not 0 <= x <= 7 for x in a.tn_set):
            p.error("--tn : intervalles 0..7 (%r)" % a.tn)
    if a.graine is None:
        a.graine = int.from_bytes(os.urandom(4), "little")
        a.graine_tiree = True
    else:
```

```
        a.graine_tiree = False
    a.graine &= MASQUE64
    np = None
    if a.mode == "iq":
        if a.cible != "dsp":
            p.error("mode iq : cible dsp seulement")
        try:
            import numpy as np  # noqa: F811 - seul le mode iq en a besoin
        except ImportError:
            p.error("mode iq : numpy est necessaire et introuvable pour %s "
                    "(apt-get install python3-numpy, ou pip install numpy). "
                    "Les modes relais, ber et souple n'en ont pas besoin." % sys.executable)
        if not a.forcer_iq:
            if os.environ.get("CALYPSO_BSP_STREAM", "") == "1":
                p.error("mode iq refuse : CALYPSO_BSP_STREAM=1. En STREAM le BSP range les 148 premiers octets "
                        "de chaque paquet comme des BITS avant de regarder s'il s'agit d'I/Q "
                        "(calypso_bsp.c bsp_trxd_readable -> bsp_ts0_stocker) : l'I/Q y serait lu comme du bruit "
                        "de bits. Lancez c54x_exe sans STREAM, ou --forcer-iq (BRUIT_OPTS=--forcer-iq).")
            if os.environ.get("CALYPSO_BSP_IQ_PASSTHROUGH", "") == "0":
                p.error("mode iq refuse : CALYPSO_BSP_IQ_PASSTHROUGH=0, le BSP ne lit pas l'I/Q.")
        if not 1000 <= a.amp <= 32767:
            p.error("--amp : 1000..32767 (%d)" % a.amp)
        if not 0.0 <= a.iq_elargir <= 2.0:
            p.error("--iq-elargir : 0..2 (%g)" % a.iq_elargir)
    return np


def _ouvrir(a):
    amont = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        amont.bind(a.ecoute)
    except OSError as e:
        amont.close()
        raise SystemExit("injecteur_bruit : impossible d'ecouter sur %s:%d (%s) -- un autre processus "
                         "tient-il deja ce port ? ss -lunp | grep :%d" % (a.ecoute[0], a.ecoute[1], e.strerror,
                         ↪  a.ecoute[1]))
    aval = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    if a.source:
        loc = a.source
    else:
        loc = ("127.0.0.1", 0) if a.vers[0].startswith("127.") or a.vers[0] == "localhost" else ("0.0.0.0", 0)
    try:
        aval.bind(loc)
    except OSError as e:
        amont.close()
        aval.close()
        raise SystemExit("injecteur_bruit : impossible de lier le port d'emission %s:%d (%s)" % (loc[0], loc[1],
        ↪  e.strerror))
    for s in (amont, aval):
        s.setblocking(False)
        try:
            s.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 1 << 20)
        except OSError:
            pass
    return amont, aval


def _journal(a):
    fmt = logging.Formatter("%(asctime)s [bruit] %(message)s", "%H:%M:%S")
    log.setLevel(logging.INFO)
    h = logging.StreamHandler(sys.stderr)
    h.setFormatter(fmt)
    log.addHandler(h)
    if a.journal:
        f = logging.FileHandler(a.journal)
        f.setFormatter(fmt)
```

```
        log.addHandler(f)


def _annoncer(a, aval, np=None):
    loc = aval.getsockname()
    log.info("injecteur de bruit %s : cible %s, mode %s%s", VERSION, a.cible, a.mode,
             " (EXPERIMENTAL)" if a.mode == "iq" else "")
    if a.mode == "ber":
        if a.rafales:
            log.info("BER moyen %.3e en rafales de %.1f bits (taux %.2f dans une rafale, Gilbert-Elliott)",
                     a.ber, a.rafales, a.ber_rafale)
        else:
            log.info("BER %.3e (bits independants)", a.ber)
        if a.cible == "grgsm" and a.ber > 0:
            log.info("ATTENTION grgsm : bits inverses dans le bloc L2 DEJA DECODE, rien ne les detecte "
                     "(ni FEC ni CRC) ; --perte modelise l'echec CRC")
    if a.mode == "souple":
        snr = 10.0 ** (a.snr_db / 10.0)
        log.info("SNR %.2f dB ; sur bits durs (le cas du pont) : +-1 + AWGN + decision dure, "
                 "BER attendu Q(sqrt(2 snr)) = %.3e", a.snr_db, _q_gauss(math.sqrt(2.0 * snr)))
    if a.mode == "iq":
        log.info("I/Q GMSK 4 ech/symbole, amplitude %d, elargissement NB %.2f, %s ; "
                 "contourne sb_moduler/gmsk_elargir du BSP", a.amp, a.iq_elargir,
                 ("AWGN %.2f dB" % a.snr_db) if a.snr_db is not None else "SANS AWGN")
        if a.snr_db is not None and np is not None:
            # Ecretage int16 : a 30000 il ne reste que 2767 de marge. On l'estime
            # pour l'annoncer (le C, CALYPSO_BSP_SNR_DB, ecrete de la meme facon).
            g = np.random.default_rng(0)
            m = 20000
            s = a.amp * np.exp(1j * g.uniform(0.0, 2.0 * math.pi, m))
            sig = math.sqrt(a.amp * a.amp / (2.0 * 10.0 ** (a.snr_db / 10.0)))
            y = s + sig * (g.standard_normal(m) + 1j * g.standard_normal(m))
            ecr = float(np.mean((np.abs(y.real) > 32767) | (np.abs(y.imag) > 32767)))
            if ecr > 0.01:
                yc = np.clip(y.real, -32768, 32767) + 1j * np.clip(y.imag, -32768, 32767)
                eff = 10.0 * math.log10(a.amp * a.amp / float(np.mean(np.abs(yc - s) ** 2)))
                log.info("ATTENTION ecretage int16 : ~%.0f %% des echantillons, SNR effectif ~%.1f dB "
                         "(bruit non gaussien) ; --amp plus bas pour l'eviter", 100.0 * ecr, eff)
    if a.perte:
        log.info("effacement : %.3e des %s", a.perte, "bursts (bits aleatoires)" if a.cible == "dsp" else "blocs
↪   (non transmis)")
    if (a.mode == "relais" or (a.mode == "ber" and a.ber == 0.0)) and not a.perte:
        log.info("aucune degradation configuree : relais transparent")
    log.info("intervalles : %s ; graine %d%s", "tous" if a.tn_set is None else ",".join(map(str,
↪   sorted(a.tn_set))),
             a.graine, " (tiree : --graine %d pour rejouer)" % a.graine if a.graine_tiree else "")
    log.info("PRET ecoute=%s:%d vers=%s:%d emission=%s:%d retour=%s", a.ecoute[0], a.ecoute[1], a.vers[0],
↪   a.vers[1],
             loc[0], loc[1], ("%s:%d" % a.retour) if a.retour else "emetteur du descendant")


def main(argv=None):
    p = _parser()
    a = p.parse_args(argv)
    np = _valider(a, p)
    if a.verifier:
        amont, aval = _ouvrir(a)
        amont.close()
        aval.close()
        print("injecteur_bruit : configuration valide (cible %s, mode %s, ecoute %s:%d -> %s:%d)"
              % (a.cible, a.mode, a.ecoute[0], a.ecoute[1], a.vers[0], a.vers[1]))
        return 0
    _journal(a)
    amont, aval = _ouvrir(a)
    deg = Degradeur(a, np)
    st = deg.stats
```

```
if a.pidfile:
    with open(a.pidfile, "w") as f:
        f.write("%d\n" % os.getpid())

arret = []
flash = []
signal.signal(signal.SIGTERM, lambda s, f: arret.append(s))
signal.signal(signal.SIGINT, lambda s, f: arret.append(s))
signal.signal(signal.SIGUSR1, lambda s, f: flash.append(s))

_annoncer(a, aval, np)
sel = selectors.DefaultSelector()
sel.register(amont, selectors.EVENT_READ, "dl")
sel.register(aval, selectors.EVENT_READ, "ul")
source = None
prochain = time.monotonic() + a.stats if a.stats > 0 else None
try:
    while not arret:
        try:
            evts = sel.select(timeout=0.5)
        except InterruptedError:
            evts = []
        for cle, _ in evts:
            s = cle.fileobj
            for _ in range(512):
                try:
                    pkt, adr = s.recvfrom(65536)
                except (BlockingIOError, InterruptedError):
                    break
                except OSError:
                    st.err_envoi += 1            # ICMP remonte sur la socket : on continue
                    break
                if cle.data == "dl":
                    source = adr
                    sortie = deg.descendant(pkt)
                    if sortie is None:
                        continue
                    try:
                        aval.sendto(sortie, a.vers)
                        st.relayes += 1
                    except OSError:
                        st.err_envoi += 1
                else:
                    dest = a.retour or source
                    if dest is None:
                        st.ul_sans_dest += 1
                        continue
                    try:
                        amont.sendto(pkt, dest)  # montant : tel quel
                        st.ul += 1
                    except OSError:
                        st.err_envoi += 1
        now = time.monotonic()
        if flash or (prochain is not None and now >= prochain):
            del flash[:]
            log.info("%s", st.ligne())
            if prochain is not None:
                prochain = now + a.stats
finally:
    log.info("%s", st.ligne(final=True))
    log.info("arret (%s)", "signal %d" % arret[0] if arret else "fin")
    sel.close()
    amont.close()
    aval.close()
    if a.pidfile:
        try:
```

```
                os.unlink(a.pidfile)
            except OSError:
                pass
    return 0


if __name__ == "__main__":
    sys.exit(main())
```

3.37 /opt/GSM/c54x_exe/tools/isa_examples.py

19204 octets, 448 lignes → 448 lignes

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""isa_examples.py - turn the SPRU172C worked examples into ISA test vectors.

WHY. The C54x core (qosmo l1-dsp) has been fixed one instruction at a time,
each time a probe on the ROM's FB/SB path pointed at one. The manual carries
about 240 "Before Instruction / After Instruction" examples, one oracle per
instruction. This script reads them out of hw/arm/calypso/doc/spru172c.md,
assembles each example with the binutils opcode table (the same table the
disassembler uses, so encoding and decoding agree) and writes a flat test file
that tools/isa_test.c replays against the core.

    tools/isa_examples.py > tools/isa_tests.txt
    make isa_test && ./isa_test tools/isa_tests.txt

Test file format (one record per example):
    T <n> <section> | <instruction as printed>
    W <word> [<word> [<word>]]         assembled instruction
    B <REG> <hex>       B M <addr> <hex>    state before (registers, data memory)
    A <REG> <hex>       A M <addr> <hex>    state after
    E
Examples that cannot be assembled (labels, far calls, ports) are written as
    S <n> <section> | <instruction> | <reason>
so the count of what is NOT covered stays visible.
"""
import io, os, re, sys

DOC   = "/opt/GSM/qosmo-dsp/hw/arm/calypso/doc/spru172c.md"
TABLE = "/opt/GSM/qosmo-dsp/hw/arm/calypso/doc/opcodes/tic54x-opc.c"

# ------------------------------------------------------------------ the table --
ENTREE = re.compile(
    r'\{\s*"([^"]+)"\s*,\s*(\d+)\s*,[^,]*,[^,]*,\s*(0x[0-9A-Fa-f]+)\s*,\s*(0x[0-9A-Fa-f]+)\s*,\s*\{([^}]*)\}')

def charger_table(chemin):
    src = io.open(chemin, encoding="utf-8", errors="replace").read()
    fin = src.find("tic54x_paroptab")
    if fin > 0:
        src = src[:fin]
    tab = []
    for m in ENTREE.finditer(src):
        nom, mots, op, masque, ops = m.group(1), int(m.group(2)), \
            int(m.group(3), 16), int(m.group(4), 16), m.group(5).strip()
        if nom == "???":
            continue
        ops = [o.strip() for o in ops.split(",") if o.strip()]
        tab.append((nom, mots, op, masque, ops))
    return tab

# ------------------------------------------------------------------ operands ---
MMR = {"IMR": 0, "IFR": 1, "ST0": 6, "ST1": 7, "AL": 8, "AH": 9, "AG": 10, "BL": 11,
```

```
       "BH": 12, "BG": 13, "T": 14, "TREG": 14, "TRN": 15, "SP": 0x18, "BK": 0x19,
       "BRC": 0x1A, "RSA": 0x1B, "REA": 0x1C, "PMST": 0x1D, "XPC": 0x1E}
for _i in range(8):
    MMR["AR%d" % _i] = 0x10 + _i

COND = {"UNC": 0x00, "AEQ": 0x45, "ANEQ": 0x44, "ALT": 0x43, "ALEQ": 0x47, "AGT": 0x46,
        "AGEQ": 0x42, "BEQ": 0x4D, "BNEQ": 0x4C, "BLT": 0x4B, "BLEQ": 0x4F, "BGT": 0x4E,
        "BGEQ": 0x4A, "AOV": 0x70, "ANOV": 0x60, "BOV": 0x78, "BNOV": 0x68,
        "TC": 0x30, "NTC": 0x20, "C": 0x0C, "NC": 0x08, "BIO": 0x03, "NBIO": 0x02}

SBITS = {"BRAF": (1, 15), "CPL": (1, 14), "XF": (1, 13), "HM": (1, 12), "INTM": (1, 11),
         "OVM": (1, 9), "SXM": (1, 8), "C16": (1, 7), "FRCT": (1, 6), "CMPT": (1, 5),
         "TC": (0, 12), "C": (0, 11), "OVA": (0, 10), "OVB": (0, 9)}

# indirect Smem modifiers -> MOD nibble (SPRU172C table 3-?; binutils tic54x-dis.c)
SMOD = {"": 0x0, "-": 0x1, "+": 0x2, "-0B": 0x4, "-0": 0x5, "+0": 0x6, "+0B": 0x7,
        "-%": 0x8, "-0%": 0x9, "+%": 0xA, "+0%": 0xB}
XMOD = {"": 0, "-": 1, "+": 2, "+0%": 3}

def norm(t):
    return t.replace("–", "-").replace("−", "-").replace("—", "-").strip()

def nombre(t):
    """TI numbers: 0FFFEh, 1234h, 10h, 248, -8, +3, #..., 15-12 (bit expression)."""
    t = norm(t).lstrip("#").strip()
    if re.fullmatch(r'[+-]?[0-9A-Fa-f]+[hH]', t):
        return int(t[:-1], 16)
    if re.fullmatch(r'[+-]?\d+', t):
        return int(t, 10)
    if re.fullmatch(r'0[xX][0-9A-Fa-f]+', t):
        return int(t, 16)
    m = re.fullmatch(r'(\d+)\s*-\s*(\d+)', t)
    if m:
        return int(m.group(1)) - int(m.group(2))
    raise ValueError("nombre: %r" % t)

class Smem:
    """Parsed single data-memory operand."""
    def __init__(self, txt):
        t = norm(txt)
        self.lk = None
        if t.startswith("*"):
            m = re.fullmatch(r'\*\(\s*([^)]+)\)', t)
            if m:                                   # *(lk) absolute
                self.code = 0x80 | (0xF << 3); self.lk = nombre(m.group(1)); return
            m = re.fullmatch(r'\*\+AR(\d)\(([^)]+)\)(%?)', t)
            if m:                                   # *+ARn(lk) [%]
                self.code = 0x80 | ((0xE if m.group(3) else 0xD) << 3) | int(m.group(1))
                self.lk = nombre(m.group(2)); return
            m = re.fullmatch(r'\*AR(\d)\(([^)]+)\)', t)
            if m:                                   # *ARn(lk)
                self.code = 0x80 | (0xC << 3) | int(m.group(1)); self.lk = nombre(m.group(2)); return
            m = re.fullmatch(r'\*\+AR(\d)', t)
            if m:
                self.code = 0x80 | (0x3 << 3) | int(m.group(1)); return
            m = re.fullmatch(r'\*AR(\d)(.*)', t)
            if m and m.group(2) in SMOD:
                self.code = 0x80 | (SMOD[m.group(2)] << 3) | int(m.group(1)); return
            raise ValueError("Smem: %r" % txt)
        if t.startswith("@"):
            t = t[1:]
        if t.upper() in MMR:                        # MMR by name = direct address
            self.code = MMR[t.upper()]; return
        v = nombre(t)
        if not 0 <= v <= 0x7F:
            raise ValueError("Smem direct hors 0..7F: %r" % txt)
```

```
        self.code = v

def xmem(txt):
    t = norm(txt)
    m = re.fullmatch(r'\*AR(\d)(.*)', t)
    if not m or m.group(2) not in XMOD:
        raise ValueError("Xmem: %r" % txt)
    n = int(m.group(1))
    if not 2 <= n <= 5:
        raise ValueError("Xmem AR%d hors AR2..AR5" % n)
    return (XMOD[m.group(2)] << 2) | (n - 2)

def conds(tokens):
    v = 0
    for c in tokens:
        c = norm(c).upper()
        if c not in COND:
            raise ValueError("cond: %r" % c)
        v |= COND[c]
    return v

# ------------------------------------------------------------------ assembler ---
def split_operands(s):
    out, cur, depth = [], "", 0
    for ch in s:
        if ch == "(":
            depth += 1
        elif ch == ")":
            depth -= 1
        if ch == "," and depth == 0:
            out.append(cur.strip()); cur = ""
        else:
            cur += ch
    if cur.strip():
        out.append(cur.strip())
    return out

def est_acc(t):
    return norm(t).upper() in ("A", "B")

def assembler(tab, texte):
    """Returns list of words or raises ValueError."""
    texte = norm(texte.split(";")[0])
    m = re.match(r'([A-Za-z][A-Za-z0-9]*)\s*(.*)$', texte)
    if not m:
        raise ValueError("syntaxe")
    nom, reste = m.group(1).lower(), m.group(2)
    ops = split_operands(reste) if reste else []
    erreurs = []
    for (tnom, mots, opc, masque, types) in tab:
        if tnom != nom:
            continue
        try:
            return encoder(tnom, mots, opc, masque, types, list(ops))
        except ValueError as e:
            erreurs.append("%s%s: %s" % (tnom, types, e))
    raise ValueError("; ".join(erreurs) if erreurs else "mnemonique inconnu %s" % nom)

def encoder(nom, mots, opc, masque, types, ops):
    word = opc
    extra = []
    src = dst = None
    types = list(types)
    # optional operands are flagged OPT| in binutils; strip the flag but remember
    optional = [t.startswith("OPT|") for t in types]
    types = [t.replace("OPT|", "") for t in types]
```

```
# implicit-operand types consume nothing from the text
IMPLICIT = {"OP_None"}
ti = 0
for k, ty in enumerate(types):
    if ty in IMPLICIT:
        continue
    if ti >= len(ops):
        if optional[k]:
            continue
        raise ValueError("operande manquant pour %s" % ty)
    o = ops[ti]
    if ty in ("OP_Smem", "OP_Sind", "OP_Lmem", "OP_MMR"):
        if ty == "OP_MMR" and norm(o).upper() in MMR:
            word |= MMR[norm(o).upper()]
        else:
            sm = Smem(o)
            word |= sm.code
            if sm.lk is not None:
                extra.append(sm.lk & 0xFFFF)
    elif ty == "OP_Xmem":
        word |= xmem(o) << 4
    elif ty == "OP_Ymem":
        word |= xmem(o)
    elif ty == "OP_SRC":
        if not est_acc(o): raise ValueError("src attendu, %r" % o)
        src = 1 if norm(o).upper() == "B" else 0
        word |= src << 9
    elif ty == "OP_SRC1":
        if not est_acc(o): raise ValueError("src attendu, %r" % o)
        word |= (1 if norm(o).upper() == "B" else 0) << 8
    elif ty == "OP_DST":
        if not est_acc(o): raise ValueError("dst attendu, %r" % o)
        dst = 1 if norm(o).upper() == "B" else 0
        word |= dst << 8
    elif ty in ("OP_lk", "OP_lku", "OP_pmad", "OP_dmad", "OP_PA"):
        extra.append(nombre(o) & 0xFFFF)
    elif ty == "OP_xpmad":
        v = nombre(o)
        word |= (v >> 16) & 0x7F
        extra.append(v & 0xFFFF)
    elif ty in ("OP_k8u", "OP_k8"):
        word |= nombre(o) & 0xFF
    elif ty == "OP_k9":
        word |= nombre(o) & 0x1FF
    elif ty == "OP_k5":
        word |= nombre(o) & 0x1F
    elif ty == "OP_k3":
        word |= nombre(o) & 0x7
    elif ty == "OP_SHIFT":
        v = nombre(o)
        if not -16 <= v <= 15: raise ValueError("SHIFT hors -16..15")
        word |= v & 0x1F
    elif ty == "OP_SHFT":
        v = nombre(o)
        if not 0 <= v <= 15: raise ValueError("SHFT hors 0..15")
        word |= v & 0xF
    elif ty == "OP_031":
        v = nombre(o)
        if not 0 <= v <= 31: raise ValueError("0..31")
        word |= v & 0x1F
    elif ty == "OP_16":
        if norm(o) != "16": raise ValueError("16 attendu")
    elif ty in ("OP_T", "OP_TS", "OP_ASM", "OP_DP", "OP_ARP", "OP_TRN", "OP_A", "OP_B"):
        attendu = {"OP_T": "T", "OP_TS": "TS", "OP_ASM": "ASM", "OP_DP": "DP", "OP_ARP": "ARP",
                   "OP_TRN": "TRN", "OP_A": "A", "OP_B": "B"}[ty]
        if norm(o).upper() != attendu: raise ValueError("%s attendu" % attendu)
```

```
        elif ty == "OP_RND":
            raise ValueError("OP_RND non gere")
        elif ty in ("OP_CC", "OP_CC3"):
            # remaining operands are all conditions
            word |= conds(ops[ti:])
            ti = len(ops)
            continue
        elif ty == "OP_CC2":
            u = norm(o).upper()
            if nom in ("saccd", "srccd", "strcd"):  # 4-bit accumulator condition, bits 3-0
                if u not in COND or not (COND[u] & 0x40): raise ValueError("cond acc attendue")
                word |= COND[u] & 0x0F
            else:                                   # CMPR CC, ARx : cc in bits 9-8
                v = nombre(o) if not u.isalpha() else {"EQ": 0, "LT": 1, "GT": 2, "NEQ": 3}[u]
                word |= (v & 3) << 8
        elif ty == "OP_ARX":
            m = re.fullmatch(r'AR(\d)', norm(o).upper())
            if not m: raise ValueError("ARx attendu")
            word |= int(m.group(1)) & 7
        elif ty in ("OP_MMRX", "OP_MMRY"):       # AR0..AR7 = 0..7, SP = 8 ; X bits 7-4, Y bits 3-0
            u = norm(o).upper()
            m = re.fullmatch(r'AR(\d)', u)
            v = int(m.group(1)) if m else (8 if u == "SP" else None)
            if v is None: raise ValueError("ARx/SP attendu")
            word |= v << (4 if ty == "OP_MMRX" else 0)
        elif ty == "OP_N":
            u = norm(o).upper()
            if u in SBITS:                          # RSBX SXM : N and SBIT from the name
                n, b = SBITS[u]; word |= (n << 9) | b
                ti += 1
                if ti < len(ops): raise ValueError("SBIT apres nom")
                return [word] + extra
            word |= (nombre(o) & 1) << 9
        elif ty == "OP_SBIT":
            u = norm(o).upper()
            word |= (SBITS[u][1] if u in SBITS else nombre(o)) & 0xF
        elif ty == "OP_BITC":
            word |= nombre(o) & 0xF
        elif ty == "OP_123":
            word |= ((nombre(o) - 1) & 3) << 8
        elif ty == "OP_12":
            word |= ((nombre(o) - 1) & 1) << 9
        else:
            raise ValueError("type %s non gere" % ty)
        ti += 1
    if ti != len(ops):
        raise ValueError("operandes en trop: %r" % ops[ti:])
    # optional dst omitted: dst = src
    if "OP_DST" in types and dst is None and src is not None:
        word |= src << 8
    if len(extra) + 1 != mots and not (len(extra) + 1 == mots + 1):
        # the table's word count does not include the Smem lk word
        raise ValueError("longueur %d != %d" % (len(extra) + 1, mots))
    return [word & 0xFFFF] + extra

# ------------------------------------------------------------ example parsing ----
REG_OK = {"A", "B", "T", "TRN", "ASM", "C", "TC", "OVA", "OVB", "OVM", "SXM", "FRCT", "C16",
          "CMPT", "DP", "SP", "ARP", "BK", "BRC", "RSA", "REA", "PC", "XPC", "PMST", "ST0",
          "ST1", "IMR", "IFR", "INTM", "BRAF", "CPL", "XF", "HM", "AR0", "AR1", "AR2", "AR3",
          "AR4", "AR5", "AR6", "AR7", "TS"}
HEX = re.compile(r'^[0-9A-Fa-f]+$')

def valeur(tokens):
    """Value from the tokens following a register name. Returns (int, nbits)."""
    toks = [norm(t) for t in tokens]
    if len(toks) >= 3 and all(HEX.match(t) for t in toks[:3]) and len(toks[0]) == 2 \
```

```
            and len(toks[1]) == 4 and len(toks[2]) == 4:
        return int("".join(toks[:3]), 16), 40
    if toks and HEX.match(toks[0]) and len(toks[0]) == 4:
        return int(toks[0], 16), 16
    if toks and toks[0] in ("0", "1"):
        return int(toks[0]), 1
    if toks and toks[0].lower() == "x":
        return None, 1
    if toks and HEX.match(toks[0]) and len(toks[0]) in (1, 2, 3):
        return int(toks[0], 16), 16
    raise ValueError("valeur: %r" % toks)

def nettoyer_nom(t):
    return re.sub(r'[†‡‡*]+$', '', norm(t)).upper()

def parser(doc):
    L = io.open(doc, encoding="utf-8", errors="replace").read().split("\n")
    tests, section = [], "?"
    i = 0
    while i < len(L):
        l = L[i]
        m = re.match(r'^Syntax\s+(?:\d+:\s*)?(\S.*)$', l)
        if m:
            section = norm(m.group(1)); i += 1; continue
        m = re.match(r'^\s*(?:\d+:\s*)?Syntax\s+(\S.*)$', l)
        if m:
            section = norm(m.group(1)); i += 1; continue
        m = re.match(r'^Example(?:\s+\d+)?\s+([A-Z][A-Za-z0-9]*(?:\s.*)?)$', l)
        if not m:
            i += 1; continue
        instr = m.group(1).strip()
        t = {"section": section, "instr": instr, "before": {}, "after": {},
             "mem_b": {}, "mem_a": {}, "pmem_b": {}, "pmem_a": {}, "warn": []}
        zone = "reg"
        j = i + 1
        while j < len(L):
            lj = L[j]
            if re.match(r'^Example', lj) or re.match(r'^\s*(?:\d+:\s*)?Syntax\s', lj) \
                   or re.match(r'^Syntax', lj) or re.match(r'^[A-Z][a-z]+\s{2,}', lj) and "Instruction" not in lj:
                break
            if "Data Memory" in lj:
                zone = "mem"; j += 1; continue
            if "Program Memory" in lj:
                zone = "pmem"; j += 1; continue
            if "Before Instruction" in lj:
                j += 1; continue
            toks = lj.split()
            if not toks:
                j += 1; continue
            if zone in ("mem", "pmem"):
                mm = re.findall(r'([0-9A-Fa-f]+)h\s+([0-9A-Fa-f]{4})', lj)
                if len(mm) >= 1:
                    tgt_b = t["mem_b"] if zone == "mem" else t["pmem_b"]
                    tgt_a = t["mem_a"] if zone == "mem" else t["pmem_a"]
                    tgt_b[int(mm[0][0], 16)] = int(mm[0][1], 16)
                    if len(mm) >= 2:
                        tgt_a[int(mm[1][0], 16)] = int(mm[1][1], 16)
                    j += 1; continue
                # a register row can follow memory rows on some pages
            name = nettoyer_nom(toks[0])
            if name in REG_OK:
                # split at the second occurrence of the name
                idx = [k for k, tk in enumerate(toks) if nettoyer_nom(tk) == name]
                try:
                    if len(idx) >= 2:
                        vb, _ = valeur(toks[idx[0] + 1:idx[1]])
```

```
                    va, _ = valeur(toks[idx[1] + 1:])
                    if vb is not None: t["before"][name] = vb
                    if va is not None: t["after"][name] = va
                else:
                    t["warn"].append("ligne registre incomplete: %s" % lj.strip())
            except ValueError as e:
                t["warn"].append(str(e))
        j += 1
    tests.append(t)
    i = j
return tests

def main():
    tab = charger_table(TABLE)
    tests = parser(DOC)
    n_ok = n_skip = 0
    for k, t in enumerate(tests):
        instr = t["instr"]
        if not t["before"] and not t["after"] and not t["mem_a"]:
            continue                                # prose caught by the regex
        # symbolic pmad/dmad (COEFFS, DAT127...) : take the first Program Memory address
        if t["pmem_b"]:
            instr = re.sub(r'\b(COEFFS|DAT\d+)\b', "%04Xh" % min(t["pmem_b"]), instr)
        try:
            words = assembler(tab, instr)
        except ValueError as e:
            # dual-operand examples written with AR6/AR7: only AR2..AR5 can be encoded.
            # Rename AR6->AR4 and AR7->AR5 in the instruction AND the register rows.
            ren = {}
            if "AR6" in instr and "AR4" not in instr and "AR4" not in t["before"]: ren["AR6"] = "AR4"
            if "AR7" in instr and "AR5" not in instr and "AR5" not in t["before"]: ren["AR7"] = "AR5"
            if ren:
                instr2 = instr
                for a, b in ren.items(): instr2 = instr2.replace(a, b)
                try:
                    words = assembler(tab, instr2)
                    for a, b in ren.items():
                        for d in (t["before"], t["after"]):
                            if a in d: d[b] = d.pop(a)
                    instr = instr2 + " ;(" + ",".join("%s->%s" % kv for kv in ren.items()) + ")"
                    e = None
                except ValueError as e2:
                    e = e2
            if e is not None:
                print("S %d %s | %s | %s" % (k, t["section"], instr, str(e)[:120]))
                n_skip += 1
                continue
        print("T %d %s | %s" % (k, t["section"], instr))
        print("W " + " ".join("%04x" % w for w in words))
        for r, v in sorted(t["before"].items()):
            print("B %s %x" % (r, v))
        for a, v in sorted(t["mem_b"].items()):
            print("B M %04x %04x" % (a, v))
        for a, v in sorted(t["pmem_b"].items()):
            print("B P %04x %04x" % (a, v))
        for r, v in sorted(t["after"].items()):
            print("A %s %x" % (r, v))
        for a, v in sorted(t["mem_a"].items()):
            print("A M %04x %04x" % (a, v))
        for w in t["warn"]:
            print("# %s" % w)
        print("E")
        n_ok += 1
    sys.stderr.write("%d exemples assembles, %d non assembles\n" % (n_ok, n_skip))

if __name__ == "__main__":
```

```
    main()
```

3.38 /opt/GSM/c54x_exe/tools/isa_test.c

11637 octets, 223 lignes → 223 lignes

```
/* SPDX-License-Identifier: GPL-2.0-or-later */
/*
 * isa_test.c - replay the SPRU172C worked examples against the C54x core.
 *
 * Reads tools/isa_tests.txt (written by tools/isa_examples.py): for every
 * example the assembled words, the state before and the state after. Each
 * test runs on a fresh C54xState: registers and memory are set, the words are
 * placed at PC, ONE c54x_run(s, 1) executes the instruction, and every
 * register/memory word the manual lists in "After Instruction" is compared.
 *
 *     make isa_test && ./isa_test tools/isa_tests.txt 2>/dev/null
 *     ./isa_test tools/isa_tests.txt -v 2>/dev/null      # print every test
 *
 * The verdict is a scorecard per instruction, not a debugging aid: a FAIL
 * names the register that differs and both values, then the core's handler is
 * read against the manual page. Registers the manual does not list are not
 * compared, so a handler that corrupts an unlisted register can still pass.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <stdbool.h>
#include "qemu/thread.h"
#include "calypso_c54x.h"

/* what main.c normally provides */
uint32_t g_c54x_exe_fn;
uint32_t calypso_trx_get_fn(void) { return g_c54x_exe_fn; }
void calypso_inth_arm_ack(void) { }

/* Program word at a 16-bit address as the core will fetch it: 0x8000-0xDFFF is
 * banked by XPC (c54x_prog_xlate), so write the word in every page; an example
 * that starts with XPC != 0 (FBACCD B) or changes XPC before its delay slots
 * otherwise fetches 0x0000. */
static void poser_prog(C54xState *s, uint16_t a, uint16_t w)
{
    s->prog[a] = w;
    if (a >= 0x8000 && a < 0xE000)
        for (uint32_t pg = 0; pg < (C54X_PROG_SIZE >> 16); pg++)
            s->prog[((pg << 16) | a) & (C54X_PROG_SIZE - 1)] = w;
}
QemuMutex calypso_pcb_daram_lock;
void cpu_physical_memory_rw(uint64_t addr, void *buf, uint64_t len, bool wr)
{ (void)addr; if (!wr) memset(buf, 0, len); }

extern uint16_t data_read(C54xState *s, uint16_t addr);
extern void     data_write(C54xState *s, uint16_t addr, uint16_t val);

#define ST0_TC_B  (1u << 12)
#define ST0_C_B   (1u << 11)
#define ST0_OVA_B (1u << 10)
#define ST0_OVB_B (1u << 9)

static struct { const char *nom; int reg; uint16_t bit; } BITS[] = {
    { "TC", 0, ST0_TC_B }, { "C", 0, ST0_C_B }, { "OVA", 0, ST0_OVA_B }, { "OVB", 0, ST0_OVB_B },
    { "BRAF", 1, 1u << 15 }, { "CPL", 1, 1u << 14 }, { "XF", 1, 1u << 13 }, { "HM", 1, 1u << 12 },
    { "INTM", 1, 1u << 11 }, { "OVM", 1, 1u << 9 }, { "SXM", 1, 1u << 8 }, { "C16", 1, 1u << 7 },
```

```
    { "FRCT", 1, 1u << 6 }, { "CMPT", 1, 1u << 5 },
};

static bool set_reg(C54xState *s, const char *r, uint64_t v)
{
    if (!strcmp(r, "A")) { s->a = (int64_t)(v & 0xFFFFFFFFFFULL); if (s->a & 0x8000000000LL) s->a |=
     ↪  ~0xFFFFFFFFFFLL; return true; }
    if (!strcmp(r, "B")) { s->b = (int64_t)(v & 0xFFFFFFFFFFULL); if (s->b & 0x8000000000LL) s->b |=
     ↪  ~0xFFFFFFFFFFLL; return true; }
    if (!strcmp(r, "T"))   { s->t = v; return true; }
    if (!strcmp(r, "TS"))  { s->t = v; return true; }
    if (!strcmp(r, "TRN")) { s->trn = v; return true; }
    if (!strcmp(r, "SP"))  { s->sp = v; return true; }
    if (!strcmp(r, "BK"))  { s->bk = v; return true; }
    if (!strcmp(r, "BRC")) { s->brc = v; return true; }
    if (!strcmp(r, "RSA")) { s->rsa = v; return true; }
    if (!strcmp(r, "REA")) { s->rea = v; return true; }
    if (!strcmp(r, "PC"))  { s->pc = v; return true; }
    if (!strcmp(r, "XPC")) { s->xpc = v; return true; }
    if (!strcmp(r, "PMST")) { s->pmst = v; return true; }
    if (!strcmp(r, "IPTR")) { s->pmst = (s->pmst & 0x7F) | ((v & 0x1FF) << 7); return true; }
    if (!strcmp(r, "ST0")) { s->st0 = v; return true; }
    if (!strcmp(r, "ST1")) { s->st1 = v; return true; }
    if (!strcmp(r, "IMR")) { s->imr = v; return true; }
    if (!strcmp(r, "IFR")) { s->ifr = v; return true; }
    if (!strcmp(r, "ASM")) { s->st1 = (s->st1 & ~0x1F) | (v & 0x1F); return true; }
    if (!strcmp(r, "DP"))  { s->st0 = (s->st0 & ~0x1FF) | (v & 0x1FF); return true; }
    if (!strcmp(r, "ARP")) { s->st0 = (s->st0 & ~0xE000) | ((v & 7) << 13); return true; }
    if (r[0] == 'A' && r[1] == 'R' && r[2] >= '0' && r[2] <= '7' && !r[3]) { s->ar[r[2] - '0'] = v; return true; }
    for (unsigned i = 0; i < sizeof BITS / sizeof BITS[0]; i++)
        if (!strcmp(r, BITS[i].nom)) {
            uint16_t *st = BITS[i].reg ? &s->st1 : &s->st0;
            if (v) *st |= BITS[i].bit; else *st &= ~BITS[i].bit;
            return true;
        }
    return false;
}

static bool get_reg(C54xState *s, const char *r, uint64_t *v)
{
    if (!strcmp(r, "A")) { *v = (uint64_t)s->a & 0xFFFFFFFFFFULL; return true; }
    if (!strcmp(r, "B")) { *v = (uint64_t)s->b & 0xFFFFFFFFFFULL; return true; }
    if (!strcmp(r, "T") || !strcmp(r, "TS")) { *v = s->t; return true; }
    if (!strcmp(r, "TRN")) { *v = s->trn; return true; }
    if (!strcmp(r, "SP"))  { *v = s->sp; return true; }
    if (!strcmp(r, "BK"))  { *v = s->bk; return true; }
    if (!strcmp(r, "BRC")) { *v = s->brc; return true; }
    if (!strcmp(r, "RSA")) { *v = s->rsa; return true; }
    if (!strcmp(r, "REA")) { *v = s->rea; return true; }
    if (!strcmp(r, "PC"))  { *v = s->pc & 0xFFFF; return true; }
    if (!strcmp(r, "XPC")) { *v = s->xpc; return true; }
    if (!strcmp(r, "PMST")) { *v = s->pmst; return true; }
    if (!strcmp(r, "IPTR")) { *v = (s->pmst >> 7) & 0x1FF; return true; }
    if (!strcmp(r, "ST0")) { *v = s->st0; return true; }
    if (!strcmp(r, "ST1")) { *v = s->st1; return true; }
    if (!strcmp(r, "IMR")) { *v = s->imr; return true; }
    if (!strcmp(r, "IFR")) { *v = s->ifr; return true; }
    if (!strcmp(r, "ASM")) { *v = s->st1 & 0x1F; return true; }
    if (!strcmp(r, "DP"))  { *v = s->st0 & 0x1FF; return true; }
    if (!strcmp(r, "ARP")) { *v = (s->st0 >> 13) & 7; return true; }
    if (r[0] == 'A' && r[1] == 'R' && r[2] >= '0' && r[2] <= '7' && !r[3]) { *v = s->ar[r[2] - '0']; return true; }
    for (unsigned i = 0; i < sizeof BITS / sizeof BITS[0]; i++)
        if (!strcmp(r, BITS[i].nom)) {
            uint16_t st = BITS[i].reg ? s->st1 : s->st0;
            *v = (st & BITS[i].bit) ? 1 : 0;
            return true;
```

```
        }
    return false;
}

typedef struct { char reg[8]; uint64_t val; int mem; uint16_t addr; } Item;

int main(int argc, char **argv)
{
    if (argc < 2) { fprintf(stderr, "usage: %s tools/isa_tests.txt [-v] [-k MOT]\n", argv[0]); return 2; }
    bool verbeux = false; const char *filtre = NULL;
    for (int i = 2; i < argc; i++) {
        if (!strcmp(argv[i], "-v")) verbeux = true;
        else if (!strcmp(argv[i], "-k") && i + 1 < argc) filtre = argv[++i];
    }
    FILE *f = fopen(argv[1], "r");
    if (!f) { perror(argv[1]); return 2; }
    qemu_mutex_init(&calypso_pcb_daram_lock);

    char ligne[1024], titre[1024] = "";
    uint16_t mots[8]; int nmots = 0, npas = 1;
    Item avant[64], apres[64]; int na = 0, np = 0;
    int total = 0, ok = 0, ko = 0, ignores = 0;
    char bilan_ko[65536] = "";

    while (fgets(ligne, sizeof ligne, f)) {
        ligne[strcspn(ligne, "\n")] = 0;
        if (ligne[0] == 'T') {
            snprintf(titre, sizeof titre, "%s", ligne + 2);
            nmots = na = np = 0; npas = 1;
        } else if (ligne[0] == 'W') {
            char *p = ligne + 2; nmots = 0;
            while (*p && nmots < 8) { mots[nmots++] = (uint16_t)strtoul(p, &p, 16); while (*p == ' ') p++; }
        } else if (ligne[0] == 'N') {
            /* N <n>: run n instructions (multi-instruction examples, delay slots) */
            npas = atoi(ligne + 2); if (npas < 1) npas = 1;
        } else if (ligne[0] == 'B' || ligne[0] == 'A') {
            Item *it = ligne[0] == 'B' ? &avant[na] : &apres[np];
            int *n = ligne[0] == 'B' ? &na : &np;
            if (*n >= 64) continue;
            char k[8], r[8]; unsigned long long a, v;
            if (sscanf(ligne + 2, "%7s %llx %llx", k, &a, &v) == 3 && (k[0] == 'M' || k[0] == 'P') && !k[1]) {
                it->mem = k[0] == 'M' ? 1 : 2; it->addr = (uint16_t)a; it->val = v; it->reg[0] = 0; (*n)++;
            } else if (sscanf(ligne + 2, "%7s %llx", r, &v) == 2) {
                it->mem = 0; snprintf(it->reg, sizeof it->reg, "%s", r); it->val = v; (*n)++;
            }
        } else if (ligne[0] == 'E') {
            if (filtre && !strstr(titre, filtre)) continue;
            total++;
            C54xState *s = c54x_init();
            s->running = true;
            s->pc = 0x9000;
            s->pmst = 0x1FFu << 7;   /* IPTR = 1FFh, its reset value, assumed by the manual's vector examples */
            bool prob = false;
            for (int i = 0; i < na; i++) {
                if (avant[i].mem == 1) { if (avant[i].addr < 0x60) data_write(s, avant[i].addr,
                 ↪  (uint16_t)avant[i].val); else s->data[avant[i].addr] = (uint16_t)avant[i].val; }
                else if (avant[i].mem == 2) s->prog[avant[i].addr] = (uint16_t)avant[i].val;
                else if (!set_reg(s, avant[i].reg, avant[i].val)) prob = true;
            }
            uint16_t pc0 = (uint16_t)s->pc;
            for (int i = 0; i < nmots; i++) poser_prog(s, (uint16_t)(pc0 + i), mots[i]);
            /* the instruction after, so a delayed branch has something to execute */
            poser_prog(s, (uint16_t)(pc0 + nmots), 0xF495); poser_prog(s, (uint16_t)(pc0 + nmots + 1), 0xF495);
            for (int k = 0; k < npas; k++) c54x_run(s, 1);
            /* RPT arms a counter and returns 0 executed words: run the repeated
             * instruction as well so the example's "After" is observable. */
```

```
            { int _g = 70000; while (s->rpt_active && _g--) c54x_run(s, 1); }
            for (int k = 0; k < 4 && s->delay_slots; k++) c54x_run(s, 1);
            char detail[2048] = ""; int d = 0; bool echec = false;
            for (int i = 0; i < np; i++) {
                uint64_t got;
                if (apres[i].mem == 1) got = apres[i].addr < 0x60 ? data_read(s, apres[i].addr) :
                 ↪  s->data[apres[i].addr];
                else if (apres[i].mem == 2) got = s->prog[apres[i].addr];
                else if (!get_reg(s, apres[i].reg, &got)) { prob = true; continue; }
                uint64_t att = apres[i].val;
                if (!apres[i].mem && !strcmp(apres[i].reg, "XPC")) { att &= 3; got &= 3; }
                if (got != att) {
                    echec = true;
                    if (apres[i].mem) d += snprintf(detail + d, sizeof detail - d, "  %s[%04x]: attendu %04llx,
                     ↪  obtenu %04llx",
                                                    apres[i].mem == 1 ? "data" : "prog", apres[i].addr,
                                                    (unsigned long long)apres[i].val, (unsigned long long)got);
                    else d += snprintf(detail + d, sizeof detail - d, "  %s: attendu %llx, obtenu %llx",
                                       apres[i].reg, (unsigned long long)apres[i].val, (unsigned long long)got);
                }
            }
            if (echec) {
                ko++;
                printf("FAIL  %-60.60s  [%04x%s%04x%s]%s\n", titre, mots[0], nmots > 1 ? " " : "", nmots > 1 ?
↪  mots[1] : 0, prob ? " ?" : "", detail);
                size_t l = strlen(bilan_ko);
                snprintf(bilan_ko + l, sizeof bilan_ko - l, "  %s\n", titre);
            } else {
                ok++;
                if (verbeux) printf("ok    %-60.60s  [%04x]\n", titre, mots[0]);
            }
            free(s);
        } else if (ligne[0] == 'S') {
            ignores++;
        }
    }
    printf("\n%d exemples : %d ok, %d FAIL, %d non assembles (S)\n", total, ok, ko, ignores);
    return ko ? 1 : 0;
}
```

3.39 /opt/GSM/c54x_exe/tools/layer1_tester.c

28673 octets, 600 lignes → 600 lignes

```
/* layer1_tester.c - qui lit, qui ecrit chaque mot de la memoire de donnees du DSP
 * (partie mesure de tools/layer1_tester.py ; « api_carte » en est l'alias).
 *
 * Rejoue un enregistrement de canal dedie (/dev/shm/calypso_rejeu_tch.bin,
 * format de tools/rejeu_banc.c : 'S' 'D' 'T' 'A' 'B') sur le coeur C54x de
 * qosmo et la ROM TI, dans l'ordre de pont.c, et COMPTE pour chacun des 65536
 * mots de donnees :
 *   - les lectures et les ecritures faites par une instruction de la ROM
 *     (premier PC, premier tick, jusqu'a 6 PC distincts, valeurs typiques) ;
 *   - les ecritures de l'ARM (enregistrements 'A' : mots modifies cote QEMU entre
 *     deux trames, donc des MODIFICATIONS, pas toutes les ecritures) et la
 *     valeur de l'etat de depart ('S' pour l'API RAM, 'D' pour le reste) ;
 *   - les ecritures qui ne passent pas par une instruction : DMA, livraison
 *     d'I/Q du BSP, empilement d'interruption, bequilles du coeur qui ecrivent
 *     s->data[] en direct (difference d'instantanes avant/apres chaque etape).
 *
 * LE CROCHET. qosmo n'est pas modifie : data_read()/data_write() de c54x_mem.c
 * appellent calypso_mbx() (moniteur de la boite aux lettres) a chaque acces
 * quand c54x_rapide == 0, avec l'adresse, la valeur et le PC. On lie le coeur
 * SANS calypso_mailbox.c et on fournit ici calypso_mbx_actif/_init/_evt : chaque
```

```
 * acces DSP de la memoire de donnees arrive dans compter(). Les acces faits hors
 * d'une instruction (DMA, BSP...) sont separes par g_dans_dsp. Une ecriture
 * directe s->data[] (bequille, DMA) echappe au crochet : la difference
 * d'instantanes la rattrape (colonne hc_*). Une LECTURE directe s->data[] du
 * coeur echappe aux deux : limite connue.
 *
 * PORTS I/O. PORTR/PORTW (c54x_exec.c) appellent d'abord calypso_a5_portr/_portw
 * pour TOUT port : enveloppes (--wrap) pour compter port, sens, PC, valeur
 * ecrite (lignes « io:xxxx » de la sortie). Seules les instructions 0x74xx/0x75xx
 * passent par la ; l'autre encodage PORTW (0x9Fxx) n'est que journalise par le
 * coeur et n'est pas compte.
 *
 * PC ACTIFS (-p). Nombre d'acces memoire par PC (xpc:pc) : dit si une routine
 * connue de la ROM a tourne pendant le rejeu (une routine sans acces memoire
 * n'y apparait pas).
 *
 * SANS EFFET DE BORD. Les sources du coeur ouvrent des fichiers /dev/shm et une
 * socket UDP (calypso_bsp.c) : l'edition de liens enveloppe fopen/open/socket
 * (-Wl,--wrap=...) ; toute ouverture en ecriture part sur /dev/null, socket()
 * echoue. Seul le fichier de sortie (-o) est ecrit. Rien n'est lance, aucun
 * processus du banc n'est touche.
 *
 * BOUCLE DE BUDGET. c54x_run rend la main de lui-meme toutes les
 * CALYPSO_DSP_YIELD instructions (32768) : courir() boucle jusqu'au budget,
 * comme pont.c et rejeu_banc.c (2026-10-03), sinon le DSP ne finit pas sa trame.
 *
 * CONSOMMATION. Pour chaque mot de l'API RAM, une modification ARM ('A') est
 * « en attente » jusqu'a ce qu'une instruction de la ROM lise le mot (consommee,
 * latence en ticks) ou que l'ARM le modifie de nouveau (ecrasee sans lecture).
 *
 *   make layer1_tester          (ou make api_carte, alias)
 *   tools/layer1_tester [-o sortie.tsv] [-j journal.tsv] [-n ticks_max] [fichier]
 *   python3 tools/layer1_tester.py --rejeu fichier   (lance ce binaire, juge, fait la carte)
 *
 * Sortie (-o) : TSV, une ligne par mot touche (voir l'en-tete), lignes '#' = bilan.
 * Journal (-j) : par tick, chaque mot de l'API RAM lu ou ecrit par la ROM :
 *   tick, mot (adresse DSP), lectures, derniere valeur lue, ecritures,
 *   derniere valeur ecrite, PC de la derniere ecriture.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdarg.h>
#include <fcntl.h>
#include <errno.h>
#include <time.h>
#include <sys/socket.h>
#include "qemu/thread.h"
#include "calypso_c54x.h"
#include "calypso_dma.h"
#include "calypso_bsp.h"
#include "calypso_rhea_dma.h"
#include "calypso_mailbox.h"
#include "hw/arm/calypso/calypso_api.h"

/* ce que main.c fournit d'habitude (meme jeu que tools/rejeu_banc.c) */
uint32_t g_c54x_exe_fn;
uint32_t calypso_trx_get_fn(void) { return g_c54x_exe_fn; }
void calypso_inth_arm_ack(void) { }
QemuMutex calypso_pcb_daram_lock;
void cpu_physical_memory_rw(uint64_t addr, void *buf, uint64_t len, bool wr)
{ (void)addr; if (!wr) memset(buf, 0, len); }

/* ---- garde-fous : aucune ecriture de fichier, aucune socket -------------- */
```

```
FILE *__real_fopen(const char *path, const char *mode);
int __real_open(const char *path, int flags, ...);
static int g_fichiers_detournes;
FILE *__wrap_fopen(const char *path, const char *mode)
{
    if (strpbrk(mode, "wa+")) { g_fichiers_detournes++; return __real_fopen("/dev/null", "w"); }
    return __real_fopen(path, mode);
}
int __wrap_open(const char *path, int flags, ...)
{
    if (flags & (O_WRONLY | O_RDWR | O_CREAT | O_TRUNC | O_APPEND)) {
        g_fichiers_detournes++;
        return __real_open("/dev/null", O_WRONLY);
    }
    return __real_open(path, flags);
}
int __wrap_socket(int domain, int type, int protocol)
{ (void)domain; (void)type; (void)protocol; errno = EACCES; return -1; }
/* ports I/O : voir plus bas, io_noter() */
static void io_noter(int ecrit, uint16_t pa, uint16_t val);
bool __real_calypso_a5_portw(C54xState *s, uint16_t pa, uint16_t val);
bool __real_calypso_a5_portr(C54xState *s, uint16_t pa, uint16_t *val);
bool __wrap_calypso_a5_portw(C54xState *s, uint16_t pa, uint16_t val)
{ io_noter(1, pa, val); return __real_calypso_a5_portw(s, pa, val); }
bool __wrap_calypso_a5_portr(C54xState *s, uint16_t pa, uint16_t *val)
{
    bool r = __real_calypso_a5_portr(s, pa, val);
    io_noter(0, pa, r ? *val : 0);
    return r;
}
ssize_t __wrap_sendto(int fd, const void *b, size_t n, int fl, const struct sockaddr *a, socklen_t l)
{ (void)fd; (void)b; (void)fl; (void)a; (void)l; return (ssize_t)n; }

/* ---- etat ------------------------------------------------------------------ */
#define API_WORDS 0x2000u                 /* fenetre API vue du DSP : data 0x0800..0x27ff */
#define ENREG_API_WORDS CALYPSO_API_WORDS  /* taille de l'API RAM enregistree par pont.c ('S') */
#define NDB 0xD4u
#define BL_ADDR_HI_W 0x7FCu
#define BL_SIZE_W 0x7FDu
#define BL_ADDR_LO_W 0x7FEu
#define BL_STATUS_W 0x7FFu
#define NPC 6                             /* PC distincts retenus par mot et par sens */
#define NVAL 4                            /* Misra-Gries : exact si <= 4 valeurs distinctes */

typedef struct {
    int64_t a, b; uint16_t ar[8], t, trn, sp, bk, brc, rsa, rea, st0, st1, pmst, imr, ifr, xpc;
    uint32_t pc; uint8_t idle, running;
} EnregRegs;                              /* = pont.c enreg_base() */

typedef struct { uint16_t v[NVAL]; uint32_t n[NVAL]; uint8_t plus; } Valeurs;
typedef struct {
    uint64_t n;                           /* acces */
    uint32_t pc1, tick1;                  /* premier acces */
    uint32_t pcs[NPC]; uint8_t npc, pcplus;
    Valeurs val;
    uint16_t dernier;
} Sens;

enum { SEG_BOOT = 0, SEG_DSP, SEG_IRQ, SEG_DMA, SEG_BSP, SEG_POMPE, SEG_N };
static const char *SEG_NOM[SEG_N] = { "boot", "dsp", "irq", "dma", "bsp", "pompe" };

static C54xState *dsp;
static uint16_t *api;
static uint8_t *fichier;
static size_t taille;
static int g_dans_dsp;                    /* 1 pendant c54x_run */
```

```
static int g_seg = SEG_BOOT;
static int g_phase_rejeu;                  /* 0 = boot local, 1 = rejeu */

static Sens *s_rd, *s_wr, *s_arm;          /* [65536] */
static uint64_t *boot_rd, *boot_wr;        /* acces pendant le boot local */
static uint64_t *hors_wr;                  /* ecritures par le crochet hors instruction */
static uint8_t *hors_src;
static uint64_t *hc_n;                     /* ecritures hors crochet (instantanes) */
static uint32_t *hc_tick1; static uint8_t *hc_src; static uint16_t *hc_der;
static uint16_t *val_s, *val_d; static uint8_t *a_s, *a_d;
static uint16_t *snap; static uint32_t *gen; static uint32_t g_gen = 1;
static uint64_t g_n_evt, g_n_rd, g_n_wr;
static Sens *io_rd, *io_wr;                /* [65536] ports */
static uint32_t *pc_acc;                   /* [0x40000] acces memoire par xpc:pc */
/* consommation des modifications ARM (mots de l'API RAM) */
static uint8_t *arm_pend; static uint32_t *arm_pend_tick;
static uint64_t *conso_n, *conso_lat, *ecrase_n;
/* journal par tick (mots de l'API RAM) */
static FILE *g_journal;
static uint32_t *j_r, *j_w; static uint16_t *j_vr, *j_vw; static uint32_t *j_pcw;
static uint16_t j_liste[0x2000]; static unsigned j_n;
static inline void j_toucher(uint16_t mot)
{
    unsigned i = mot - 0x800u;
    if (!j_r[i] && !j_w[i]) j_liste[j_n++] = (uint16_t)i;
}

static inline uint32_t pc_complet(void)
{
    uint32_t pc = dsp->pc & 0xffff;
    if (pc >= 0x8000 && dsp->xpc) pc |= (uint32_t)dsp->xpc << 16;
    return pc;
}

static inline void valeur(Valeurs *v, uint16_t x)
{
    for (int i = 0; i < NVAL; i++) if (v->n[i] && v->v[i] == x) { v->n[i]++; return; }
    for (int i = 0; i < NVAL; i++) if (!v->n[i]) { v->v[i] = x; v->n[i] = 1; return; }
    v->plus = 1;
    for (int i = 0; i < NVAL; i++) v->n[i]--;
}

static inline void noter(Sens *s, uint32_t pc, uint16_t v)
{
    if (!s->n) { s->pc1 = pc; s->tick1 = g_c54x_exe_fn; }
    s->n++;
    s->dernier = v;
    valeur(&s->val, v);
    if (pc != 0xffffffffu) {
        for (int i = 0; i < s->npc; i++) if (s->pcs[i] == pc) goto vu;
        if (s->npc < NPC) s->pcs[s->npc++] = pc; else s->pcplus = 1;
    vu:;
    }
}

static void io_noter(int ecrit, uint16_t pa, uint16_t val)
{
    if (!g_phase_rejeu || !io_rd) return;
    noter(ecrit ? &io_wr[pa] : &io_rd[pa], pc_complet(), val);
}

/* Le crochet : remplace calypso_mailbox.c (seuls symboles exportes). */
int calypso_mbx_actif = 1;
void calypso_mbx_init(void) { }
void calypso_mbx_evt(CalypsoMbxSens sens, uint16_t mot, uint16_t val, uint16_t avant,
                     uint32_t ctx, uint32_t fn, uint32_t insn)
```

```
{
    (void)avant; (void)ctx; (void)fn; (void)insn;
    g_n_evt++;
    if (g_phase_rejeu && g_dans_dsp) pc_acc[pc_complet() & 0x3ffff]++;
    if (sens == MBX_DSP_RD) {
        g_n_rd++;
        if (!g_phase_rejeu) { boot_rd[mot]++; return; }
        if (g_dans_dsp) {
            noter(&s_rd[mot], pc_complet(), val);
            if (mot >= 0x800 && mot < 0x2800) {
                if (arm_pend[mot]) { arm_pend[mot] = 0; conso_n[mot]++; conso_lat[mot] += g_c54x_exe_fn -
                 ↪  arm_pend_tick[mot]; }
                if (g_journal) { j_toucher(mot); j_r[mot - 0x800]++; j_vr[mot - 0x800] = val; }
            }
        }
    } else if (sens == MBX_DSP_WR) {
        g_n_wr++;
        gen[mot] = g_gen;
        if (!g_phase_rejeu) { boot_wr[mot]++; return; }
        if (g_dans_dsp) {
            noter(&s_wr[mot], pc_complet(), val);
            if (g_journal && mot >= 0x800 && mot < 0x2800) {
                j_toucher(mot); j_w[mot - 0x800]++; j_vw[mot - 0x800] = val; j_pcw[mot - 0x800] = pc_complet(); }
        }
        else { hors_wr[mot]++; hors_src[mot] |= (uint8_t)(1u << g_seg); }
    }
}

/* Ecritures qui n'ont pas pris le crochet depuis le dernier instantane. */
static void instantane(int seg)
{
    const uint64_t *a = (const uint64_t *)dsp->data, *b = (const uint64_t *)snap;
    for (unsigned q = 0; q < C54X_DATA_SIZE / 4; q++) {
        if (a[q] == b[q]) continue;
        for (unsigned k = 0; k < 4; k++) {
            unsigned m = q * 4 + k;
            if (dsp->data[m] == snap[m]) continue;
            if (m >= 0x60 && gen[m] != g_gen && g_phase_rejeu) {   /* MMR/peripheriques exclus */
                if (!hc_n[m]) hc_tick1[m] = g_c54x_exe_fn;
                hc_n[m]++; hc_src[m] |= (uint8_t)(1u << seg); hc_der[m] = dsp->data[m];
            }
            snap[m] = dsp->data[m];
        }
    }
    g_gen++;
}

static void synchro_instantane(void) { memcpy(snap, dsp->data, C54X_DATA_SIZE * 2); g_gen++; }

static uint32_t be32(const uint8_t *p) { return (uint32_t)p[0] << 24 | p[1] << 16 | p[2] << 8 | p[3]; }
static uint16_t be16(const uint8_t *p) { return (uint16_t)(p[0] << 8 | p[1]); }

static long courir(long n)
{
    long k = 0;
    g_dans_dsp = 1;
    while (k < n && dsp->running && !dsp->idle) {
        int r = c54x_run(dsp, (int)(n - k));
        if (r <= 0) break;
        k += r;
    }
    g_dans_dsp = 0;
    instantane(SEG_DSP);
    return k;
}
```

```
static void reveil(void)
{
    g_seg = SEG_IRQ;
    if (dsp->idle && calypso_rhea_dma_irq_level() && (dsp->imr & (1u << 14)) && !(dsp->ifr & (1u << 14)))
        c54x_interrupt_ex(dsp, 30, 14);
    if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
    instantane(SEG_IRQ);
}

static void boot(void)
{
    memset(api, 0, API_WORDS * 2);
    long b = 0;
    g_dans_dsp = 1;
    while (api[BL_STATUS_W] != 1 && b < 8000000) { int ex = c54x_run(dsp, 256); if (ex <= 0) break; b += ex; }
    api[BL_ADDR_HI_W] = 0; api[BL_ADDR_LO_W] = 0x7000; api[BL_SIZE_W] = 0; api[BL_STATUS_W] = 2;
    while (b < 12000000 && dsp->running) { int ex = c54x_run(dsp, 256); if (ex <= 0) break; b += ex; if
     ↪  (dsp->idle) break; }
    g_dans_dsp = 0;
    printf("# boot local : %ld insn, idle=%d\n", b, dsp->idle);
}

static void ecrire_arm(unsigned a, uint16_t v)
{
    if (a >= API_WORDS) return;
    api[a] = v;
    snap[0x800 + a] = v;
    noter(&s_arm[0x800 + a], 0xffffffffu, v);
    if (arm_pend[0x800 + a]) ecrase_n[0x800 + a]++;
    arm_pend[0x800 + a] = 1; arm_pend_tick[0x800 + a] = g_c54x_exe_fn;
}

static void pc_txt(char *o, uint32_t pc)
{
    if (pc > 0xffff) sprintf(o, "%x:%04x", pc >> 16, pc & 0xffff); else sprintf(o, "%04x", pc);
}

static void sens_txt(FILE *f, const Sens *s)
{
    char b[16];
    fprintf(f, "\t%llu", (unsigned long long)s->n);
    if (!s->n) { fprintf(f, "\t\t\t\t\t"); return; }
    if (s->pc1 != 0xffffffffu) { pc_txt(b, s->pc1); fprintf(f, "\t%s", b); } else fprintf(f, "\t");
    fprintf(f, "\t%u\t", s->tick1);
    for (int i = 0; i < s->npc; i++) { pc_txt(b, s->pcs[i]); fprintf(f, "%s%s", i ? "," : "", b); }
    if (s->pcplus) fprintf(f, ",+");
    fprintf(f, "\t");
    /* valeurs, de la plus frequente a la moins frequente */
    int ord[NVAL]; for (int i = 0; i < NVAL; i++) ord[i] = i;
    for (int i = 0; i < NVAL; i++) for (int j = i + 1; j < NVAL; j++)
        if (s->val.n[ord[j]] > s->val.n[ord[i]]) { int t = ord[i]; ord[i] = ord[j]; ord[j] = t; }
    int prem = 1;
    for (int i = 0; i < NVAL; i++) if (s->val.n[ord[i]]) {
        fprintf(f, "%s%04x:%u", prem ? "" : ",", s->val.v[ord[i]], s->val.n[ord[i]]); prem = 0; }
    if (s->val.plus) fprintf(f, ",+");
    fprintf(f, "\t%04x", s->dernier);
}

static void src_txt(FILE *f, uint8_t m)
{
    int prem = 1;
    for (int i = 0; i < SEG_N; i++) if (m & (1u << i)) { fprintf(f, "%s%s", prem ? "" : ",", SEG_NOM[i]); prem =
     ↪  0; }
}

typedef struct { size_t debut, fin; uint32_t tick; uint8_t drap; long budget; } Tick;
```

```
static void *alloue(size_t n) { void *p = calloc(1, n); if (!p) { perror("calloc"); exit(1); } return p; }

int main(int argc, char **argv)
{
    const char *nom = "/dev/shm/calypso_rejeu_tch.bin", *sortie = NULL, *f_pcs = NULL;
    long max_ticks = 1000000;
    for (int i = 1; i < argc; i++) {
        if (!strcmp(argv[i], "-o") && i + 1 < argc) sortie = argv[++i];
        else if (!strcmp(argv[i], "-p") && i + 1 < argc) f_pcs = argv[++i];
        else if (!strcmp(argv[i], "-j") && i + 1 < argc) {
            g_journal = __real_fopen(argv[++i], "w");
            if (!g_journal) { perror(argv[i]); return 1; }
        }
        else if (!strcmp(argv[i], "-n") && i + 1 < argc) max_ticks = atol(argv[++i]);
        else if (argv[i][0] == '-') { fprintf(stderr, "usage : %s [-o sortie.tsv] [-j journal.tsv] [-p pcs.tsv]
         ↪  [-n ticks] [fichier]\n", argv[0]); return 2; }
        else nom = argv[i];
    }
    FILE *f = __real_fopen(nom, "rb");
    if (!f) { perror(nom); return 1; }
    fseek(f, 0, SEEK_END); taille = (size_t)ftell(f); fseek(f, 0, SEEK_SET);
    fichier = alloue(taille);
    if (fread(fichier, 1, taille, f) != taille) { fprintf(stderr, "lecture courte\n"); return 1; }
    fclose(f);
    FILE *o = sortie ? __real_fopen(sortie, "w") : stdout;
    if (!o) { perror(sortie); return 1; }

   s_rd = alloue(65536 * sizeof(Sens)); s_wr = alloue(65536 * sizeof(Sens)); s_arm = alloue(65536 * sizeof(Sens));
    boot_rd = alloue(65536 * 8); boot_wr = alloue(65536 * 8); hors_wr = alloue(65536 * 8); hors_src =
↪  alloue(65536);
    hc_n = alloue(65536 * 8); hc_tick1 = alloue(65536 * 4); hc_src = alloue(65536); hc_der = alloue(65536 * 2);
    val_s = alloue(65536 * 2); val_d = alloue(65536 * 2); a_s = alloue(65536); a_d = alloue(65536);
    snap = alloue(65536 * 2); gen = alloue(65536 * 4);
    io_rd = alloue(65536 * sizeof(Sens)); io_wr = alloue(65536 * sizeof(Sens)); pc_acc = alloue(0x40000 * 4);
    arm_pend = alloue(65536); arm_pend_tick = alloue(65536 * 4);
    conso_n = alloue(65536 * 8); conso_lat = alloue(65536 * 8); ecrase_n = alloue(65536 * 8);
    j_r = alloue(0x2000 * 4); j_w = alloue(0x2000 * 4); j_vr = alloue(0x2000 * 2); j_vw = alloue(0x2000 * 2);
    j_pcw = alloue(0x2000 * 4);
    if (g_journal) fprintf(g_journal, "tick\tmot\tr\tval_r\tw\tval_w\tpc_w\n");

    /* meme environnement que tools/rejeu_banc.c ; les ecritures /dev/shm et la
     * socket sont de toute facon neutralisees par les enveloppes ci-dessus */
    setenv("CALYPSO_BSP_PORT", "16703", 1);
    setenv("CALYPSO_BSP_BIND_ADDR", "127.0.0.1", 1);
    setenv("CALYPSO_BSP_HORLOGE", "0", 1);
    setenv("CALYPSO_C54X_IRQ_LEVEL", "1", 0);
    setenv("CALYPSO_BSP_STREAM", "1", 0);
    setenv("CALYPSO_RHEA_DMA_XFER", "1", 0);
    setenv("CALYPSO_REJEU_ENREG", "0", 1);
    qemu_mutex_init(&calypso_pcb_daram_lock);
    dsp = c54x_init();
    api = &dsp->data[0x800];
    c54x_set_api_ram(dsp, api);
    static const struct { const char *s; uint32_t a; bool p; } R[] = {
        { "PROM0", 0x07000, true }, { "PROM1", 0x18000, true }, { "PROM2", 0x28000, true },
        { "PROM3", 0x38000, true }, { "DROM", 0x09000, false }, { "PDROM", 0x0E000, false }, { "PDROM", 0x0E000,
         ↪  true } };
    const char *rom = getenv("API_CARTE_ROM") ? getenv("API_CARTE_ROM") : "/opt/GSM";
    for (unsigned i = 0; i < 7; i++) {
        char c[512]; snprintf(c, sizeof c, "%s/calypso_dsp.%s.bin", rom, R[i].s);
        if (c54x_load_section(dsp, c, R[i].a, R[i].p) < 0) { fprintf(stderr, "ROM %s\n", c); return 1; }
    }
    { char c[512]; snprintf(c, sizeof c, "%s/calypso_dsp.Registers.bin", rom); c54x_load_registers(dsp, c); }
    c54x_reset(dsp);
    calypso_dma_init();
```

```
    calypso_bsp_init(dsp);
    struct timespec t0; clock_gettime(CLOCK_MONOTONIC, &t0);
    boot();
    g_phase_rejeu = 1;

    /* Premiere passe : etat de depart et decoupage en ticks. */
    Tick *ticks = alloue(sizeof(Tick) * 200000); int nt = 0;
    size_t p = 0;
    while (p < taille) {
        uint8_t k = fichier[p];
        size_t l;
        switch (k) {
        case 'S': l = 5 + (size_t)ENREG_API_WORDS * 2; break;
        case 'D': l = 5 + sizeof(EnregRegs) + C54X_DATA_SIZE * 2; break;
        case 'T': l = 10; break;
        case 'A': l = 8 + 4 * (size_t)be16(fichier + p + 6); break;
        case 'B': l = 15 + 2 * (size_t)be16(fichier + p + 13); break;
        default: fprintf(stderr, "enregistrement inconnu 0x%02x a %zu\n", k, p); goto fini;
        }
        if (p + l > taille) break;
        if (k == 'S') {
            memcpy(api, fichier + p + 5, API_WORDS * 2);
            for (unsigned i = 0; i < API_WORDS; i++) { val_s[0x800 + i] = api[i]; a_s[0x800 + i] = 1; }
            printf("# etat S pose (tick %u)\n", be32(fichier + p + 1));
        } else if (k == 'D') {
            EnregRegs r; memcpy(&r, fichier + p + 5, sizeof r);
            uint16_t *sauve = alloue(API_WORDS * 2); memcpy(sauve, api, API_WORDS * 2);
            memcpy(dsp->data, fichier + p + 5 + sizeof r, C54X_DATA_SIZE * 2);
            for (unsigned i = 0; i < C54X_DATA_SIZE; i++) { val_d[i] = dsp->data[i]; a_d[i] = 1; }
            memcpy(api, sauve, API_WORDS * 2); free(sauve);   /* 'S' fait foi pour l'API RAM */
            dsp->a = r.a; dsp->b = r.b; memcpy(dsp->ar, r.ar, sizeof r.ar); dsp->t = r.t; dsp->trn = r.trn;
            dsp->sp = r.sp; dsp->bk = r.bk; dsp->brc = r.brc; dsp->rsa = r.rsa; dsp->rea = r.rea;
            dsp->st0 = r.st0; dsp->st1 = r.st1; dsp->pmst = r.pmst; dsp->imr = r.imr; dsp->ifr = r.ifr;
            dsp->xpc = r.xpc; dsp->pc = r.pc; dsp->idle = r.idle; dsp->running = r.running;
            printf("# etat D pose : pc=%04x sp=%04x idle=%d imr=%04x\n", r.pc & 0xffff, r.sp, r.idle, r.imr);
        } else if (k == 'T') {
            if (nt > 0) ticks[nt - 1].fin = p;
            if (nt < 200000) {
                ticks[nt].debut = p; ticks[nt].tick = be32(fichier + p + 1);
                ticks[nt].drap = fichier[p + 5]; ticks[nt].budget = (long)be32(fichier + p + 6);
                nt++;
            }
        }
        p += l;
    }
fini:
    if (nt > 0) ticks[nt - 1].fin = p;
    printf("# %d ticks enregistres (%u..%u)\n", nt, nt ? ticks[0].tick : 0, nt ? ticks[nt - 1].tick : 0);
    synchro_instantane();

    uint16_t prec_cd = 0xffff, prec_fd = 0xffff, prec_dd = 0;
    int sacch_ok = 0, sacch_ko = 0, facch_ok = 0, facch_ko = 0;
    unsigned long parole_n = 0, parole_bfi = 0, arm_hors_fenetre = 0;
    uint64_t insns = 0;
    int joues = 0;
    for (int it = 0; it < nt && it < max_ticks; it++, joues++) {
        Tick *t = &ticks[it];
        g_c54x_exe_fn = t->tick;
        long budget = t->budget;
        /* ecritures ARM, fenetre 0 (avant le TICK) */
        for (size_t q = t->debut; q < t->fin;) {
            uint8_t k = fichier[q];
            size_t l = k == 'T' ? 10 : k == 'A' ? 8 + 4 * (size_t)be16(fichier + q + 6)
                     : k == 'B' ? 15 + 2 * (size_t)be16(fichier + q + 13) : 0;
            if (!l) break;
            if (k == 'A' && fichier[q + 5] == 0)
```

```
            for (unsigned i = 0, n = be16(fichier + q + 6); i < n; i++) {
                unsigned a = be16(fichier + q + 8 + 4 * i);
                if (a >= API_WORDS) arm_hors_fenetre++;
                ecrire_arm(a, be16(fichier + q + 10 + 4 * i));
            }
        q += l;
    }
    /* phase A */
    g_seg = SEG_DMA; calypso_dma_tick(dsp); instantane(SEG_DMA);
    reveil();
    g_seg = SEG_IRQ;
    if ((dsp->imr & (1u << C54X_IT_TPU_FRAME_BIT)) && (t->drap & 1))
        c54x_interrupt_ex(dsp, C54X_IT_TPU_FRAME_VEC, C54X_IT_TPU_FRAME_BIT);
    instantane(SEG_IRQ);
    g_seg = SEG_DSP;
    long fait = 0;
    if (!dsp->idle) fait = courir(budget / 8);
    while (!dsp->idle && fait < budget / 2) { long r = courir(256); if (r <= 0) break; fait += r; }
    /* ecritures ARM, fenetre 1, puis livraisons I/Q dans l'ordre */
    for (size_t q = t->debut; q < t->fin;) {
        uint8_t k = fichier[q];
        size_t l = k == 'T' ? 10 : k == 'A' ? 8 + 4 * (size_t)be16(fichier + q + 6)
                 : k == 'B' ? 15 + 2 * (size_t)be16(fichier + q + 13) : 0;
        if (!l) break;
        if (k == 'A' && fichier[q + 5] == 1)
            for (unsigned i = 0, n = be16(fichier + q + 6); i < n; i++) {
                unsigned a = be16(fichier + q + 8 + 4 * i);
                if (a >= API_WORDS) arm_hors_fenetre++;
                ecrire_arm(a, be16(fichier + q + 10 + 4 * i));
            }
        if (k == 'B') {
            int n = be16(fichier + q + 13);
            static int16_t iq[1024];
            memcpy(iq, fichier + q + 15, 2 * (size_t)(n > 1024 ? 1024 : n));
            g_seg = SEG_BSP;
            calypso_bsp_rx_burst(fichier[q + 5], be32(fichier + q + 6), iq, n);
            instantane(SEG_BSP);
        }
        q += l;
    }
    /* phase B : reste du budget, puis la pompe DMA de pont.c */
    reveil();
    g_seg = SEG_DSP;
    if (!dsp->idle && budget - fait > 0) fait += courir(budget - fait);
    for (int k = 0; k < 40 && dsp->running; k++) {
        g_seg = SEG_POMPE;
        int pompe = calypso_rhea_dma_pump(dsp);
        instantane(SEG_POMPE);
        if (!pompe) break;
        if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
        g_seg = SEG_DSP;
        if (!dsp->idle) fait += courir(budget / 4);
    }
    insns += (uint64_t)fait;
    /* controle de fidelite : memes bilans que tools/rejeu_banc.c */
    uint16_t *cd = &api[NDB + 0x1FC / 2], *fd = &api[NDB + 0x21A / 2], *dd = &api[NDB + 0x238 / 2];
    if (cd[0] != prec_cd && (cd[0] & 0x8000)) { if (cd[0] & (1u << 6)) sacch_ko++; else sacch_ok++; }
    if (fd[0] != prec_fd && (fd[0] & 0x8000)) { if (fd[0] & (1u << 6)) facch_ko++; else facch_ok++; }
    if (dd[0] != prec_dd && (dd[0] & 0x8000)) { parole_n++; if (dd[0] & 0x4) parole_bfi++; }
    prec_cd = cd[0]; prec_fd = fd[0]; prec_dd = dd[0];
    if (g_journal) {
        for (unsigned q = 0; q < j_n; q++) {
            unsigned i = j_liste[q]; char b[16];
            pc_txt(b, j_pcw[i]);
            fprintf(g_journal, "%u\t%04x\t%u\t%04x\t%u\t%04x\t%s\n", t->tick, 0x800 + i, j_r[i], j_vr[i],
↪   j_w[i], j_vw[i],
```

```
                    j_w[i] ? b : "");
                j_r[i] = j_w[i] = 0;
            }
            j_n = 0;
        }
    }
    if (g_journal) fclose(g_journal);
    struct timespec t1; clock_gettime(CLOCK_MONOTONIC, &t1);
    double sec = (double)(t1.tv_sec - t0.tv_sec) + (t1.tv_nsec - t0.tv_nsec) / 1e9;

    fprintf(o, "# layer1_tester : fichier=%s ticks_joues=%d/%d tick=%u..%u insn=%llu duree=%.1fs\n", nom, joues,
↪   nt,
            nt ? ticks[0].tick : 0, joues ? ticks[joues - 1].tick : 0, (unsigned long long)insns, sec);
    fprintf(o, "# controle : SACCH %d bonnes / %d Fire KO ; FACCH %d / %d KO ; PAROLE %lu trames, %lu BFI\n",
            sacch_ok, sacch_ko, facch_ok, facch_ko, parole_n, parole_bfi);
    fprintf(o, "# crochet : %llu evenements (%llu lectures, %llu ecritures DSP) ; ouvertures detournees vers
↪   /dev/null : %d ;"
            " mots ARM hors fenetre DSP : %lu\n", (unsigned long long)g_n_evt, (unsigned long long)g_n_rd,
            (unsigned long long)g_n_wr, g_fichiers_detournes, arm_hors_fenetre);
    fprintf(o, "adr\tdsp_r\tr_pc1\tr_tick1\tr_pcs\tr_vals\tr_der"
               "\tdsp_w\tw_pc1\tw_tick1\tw_pcs\tw_vals\tw_der"
               "\tarm_w\tarm_pc1\tarm_tick1\tarm_pcs\tarm_vals\tarm_der"
               "\thors_w\thors_src\thc_w\thc_src\thc_tick1\thc_der\tval_s\tval_d\tval_fin\tboot_r\tboot_w"
               "\tarm_conso\tarm_lat_moy\tarm_ecrase\n");
    for (unsigned a = 0; a < 65536; a++) {
        bool api_mot = a >= 0x800 && a < 0x800 + API_WORDS;
        if (!api_mot && !s_rd[a].n && !s_wr[a].n && !hors_wr[a] && !hc_n[a] && !boot_rd[a] && !boot_wr[a])
        ↪  continue;
        fprintf(o, "%04x", a);
        sens_txt(o, &s_rd[a]); sens_txt(o, &s_wr[a]); sens_txt(o, &s_arm[a]);
        fprintf(o, "\t%llu\t", (unsigned long long)hors_wr[a]); src_txt(o, hors_src[a]);
        fprintf(o, "\t%llu\t", (unsigned long long)hc_n[a]); src_txt(o, hc_src[a]);
        if (hc_n[a]) fprintf(o, "\t%u\t%04x", hc_tick1[a], hc_der[a]); else fprintf(o, "\t\t");
        if (a_s[a]) fprintf(o, "\t%04x", val_s[a]); else fprintf(o, "\t");
        if (a_d[a]) fprintf(o, "\t%04x", val_d[a]); else fprintf(o, "\t");
        fprintf(o, "\t%04x\t%llu\t%llu", dsp->data[a], (unsigned long long)boot_rd[a], (unsigned long
↪   long)boot_wr[a]);
        if (conso_n[a]) fprintf(o, "\t%llu\t%.2f", (unsigned long long)conso_n[a], (double)conso_lat[a] /
        ↪  conso_n[a]);
        else fprintf(o, "\t0\t");
        fprintf(o, "\t%llu\n", (unsigned long long)ecrase_n[a]);
    }
    for (unsigned a = 0; a < 65536; a++) {
        if (!io_rd[a].n && !io_wr[a].n) continue;
        fprintf(o, "io:%04x", a);
        sens_txt(o, &io_rd[a]); sens_txt(o, &io_wr[a]);
        static const Sens zero;
        sens_txt(o, &zero);                 /* pas de colonnes ARM pour un port */
        for (int k = 0; k < 15; k++) fputc('\t', o);
        fputc('\n', o);
    }
    if (o != stdout) fclose(o);
    if (f_pcs) {
        FILE *fp = __real_fopen(f_pcs, "w");
        if (!fp) { perror(f_pcs); return 1; }
        fprintf(fp, "pc\tacces\n");
        for (unsigned a = 0; a < 0x40000; a++) if (pc_acc[a]) {
            char b[16]; pc_txt(b, a); fprintf(fp, "%s\t%u\n", b, pc_acc[a]); }
        fclose(fp);
    }
    printf("# fini : %d ticks, %llu insn, %.1f s ; SACCH %d/%d KO ; PAROLE %lu (%lu BFI)\n", joues,
           (unsigned long long)insns, sec, sacch_ok, sacch_ko, parole_n, parole_bfi);
    return 0;
}
```

3.40 /opt/GSM/c54x_exe/tools/layer1_tester.py

20851 octets, 435 lignes → 435 lignes

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""layer1_tester.py - l'interface layer1 (ARM) <-> ROM du DSP Calypso, champ par champ.

Ce que l'outil fait (sorties PASS / FAIL / ECART / NON-EXERCE / INFO) :

 1. EXTRAIRE. Les structures de l'API RAM telles que COMPILEES pour le firmware :
    - DWARF de layer1.highram.elf (gdb-multiarch « ptype /o ») : offset et taille de
      chaque champ de T_DB_MCU_TO_DSP, T_DB_DSP_TO_MCU, T_NDB_MCU_DSP, T_PARAM_MCU_DSP ;
    - offsetof() sur dsp_api.h + l1_environment.h (memes #define : DSP=36, CHIPSET=12,
      ANLG_FAM=2) compile sur l'hote : les deux doivent coincider champ par champ ;
    - BASE_API_* (en-tete) contre l'initialiseur dsp_api de l'ELF, dsp_params (PARAM),
      macros B_*, *_DSP_TASK, BL_* (calypso/dsp.c).
    Chaque champ -> adresse ARM, offset octets dans l'API RAM, adresse mot DSP
    (0x0800 + octets/2).
 2. RECOUPER avec les constantes de qosmo (calypso_api.h, calypso_fbsb.h,
    calypso_arm2dsp.c, calypso_c54x.h, calypso_bsp.c...) et de c54x_exe (pont.c,
    montant.c, rejouer.c, tools/rejeu_banc.c...) : macros nommees (table d'unites
    ci-dessous, valeurs jamais recopiees) + balayage des lignes qui citent un champ
    a cote d'une adresse en dur (SUSPECT si l'adresse tombe sur un autre champ).
 3. FONCTIONS. Balayage statique des fonctions de la layer1 (layer1/*.c,
    calypso/dsp.c) : quels champs chacune ecrit / lit. Avec un rejeu (--rejeu ou
    --mesure), pour chaque champ et chaque fonction : l'ARM y ecrit-il (enregistrements
    'A'), des valeurs coherentes (codes de tache, burst_d/u, d_ctrl_system, d_ctrl_tch,
    d_fn, a_a5fn, d_dsp_page, B_BLUD...), la ROM le lit-elle (crochet de
    tools/layer1_tester.c), la ROM produit-elle ce que la layer1 lit ; page W
    ecrite = page annoncee par d_dsp_page ; echo page W -> page R ; PARAM et
    initialisations NDB de l'etat 'S' = valeurs du firmware.
 4. CARTE (--md, alias « api_carte ») : docs/carte-memoire.md, tables generees +
    significations de tools/layer1_tester.notes.tsv et adresses internes de
    tools/layer1_tester.internes.tsv (editables a la main, une source par ligne).

    make layer1_tester
    python3 tools/layer1_tester.py                          # 1-3 (statique)
    python3 tools/layer1_tester.py --rejeu FICHIER          # + mesure (≈ 1 min)
    python3 tools/layer1_tester.py --rejeu FICHIER --md docs/carte-memoire.md
    python3 tools/layer1_tester.py --mesure m.tsv --journal j.tsv --enreg FICHIER --md ...
    python3 tools/layer1_tester.py api_carte --rejeu FICHIER   # = --md docs/carte-memoire.md

Code de sortie 1 s'il y a au moins un FAIL. Lecture seule partout (firmware, qosmo,
enregistrement) ; n'ecrit que --md, --cache et les fichiers temporaires.
"""
import argparse
import collections
import datetime
import os
import re
import shutil
import struct
import subprocess
import sys
import tempfile

ICI = os.path.dirname(os.path.abspath(__file__))
RACINE = os.path.dirname(ICI)                       # c54x_exe
GSM = os.path.dirname(RACINE)                       # /opt/GSM
API_ARM = 0xFFD00000
API_DSP = 0x0800
API_MOTS = 0x2000

# ---------------------------------------------------------------------- resultats
class Resultats:
```

```
    ORDRE = ['FAIL', 'SUSPECT', 'ECART', 'NON-EXERCE', 'PASS', 'INFO']

    def __init__(self, bavard=False):
        self.lignes = []
        self.bavard = bavard

    def __call__(self, statut, cat, sujet, detail='', source=''):
        self.lignes.append((statut, cat, sujet, detail, source))
        if self.bavard or statut in ('FAIL', 'SUSPECT', 'ECART'):
            print('%-10s %-10s %-34s %s%s' % (statut, cat, sujet, detail, ('  [%s]' % source) if source else ''))

    def compte(self, cat=None):
        c = collections.Counter(l[0] for l in self.lignes if cat is None or l[1] == cat)
        return c

    def de(self, cat):
        return [l for l in self.lignes if l[1] == cat]


def rel(p):
    """Chemin relatif a /opt/GSM pour les sources citees."""
    p = os.path.abspath(p)
    return os.path.relpath(p, GSM) if p.startswith(GSM + os.sep) else p


def lancer(cmd, **kw):
    r = subprocess.run(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE, universal_newlines=True, **kw)
    if r.returncode != 0:
        raise RuntimeError('%s : %s' % (' '.join(cmd[:3]), r.stderr[-2000:]))
    return r.stdout


# ---------------------------------------------------------- evaluation C minimale
_SUFFIXE = re.compile(r'\b(0[xX][0-9a-fA-F]+|\d+)[uUlL]+\b')
_CAST = re.compile(r'\(\s*(?:const\s+)?(?:volatile\s+)?(?:unsigned\s+|signed\s+)?'
                   r'(?:u?int(?:8|16|32|64)_t|int|long|short|char|API|API_SIGNED)\s*\*?\s*\)')


def c_vers_py(expr):
    e = _CAST.sub('', expr)
    e = _SUFFIXE.sub(r'\1', e)
    e = e.replace('&&', ' and ').replace('||', ' or ').replace('!', ' not ').replace(' not =', '!=')
    # ternaire a ? b : c (un seul niveau)
    m = re.match(r'^\s*\(?\s*(.+?)\s*\?\s*(.+?)\s*:\s*(.+?)\s*\)?\s*$', e)
    if m and '?' in e:
        e = '((%s) if (%s) else (%s))' % (m.group(2), m.group(1), m.group(3))
    e = re.sub(r'(?<![\w])/(?![/=])', '//', e)
    return e


def evaluer(expr, env, prof=0):
    """Evalue une expression constante C ; env : nom -> (texte, params) ou entier."""
    if prof > 20:
        raise ValueError('recursion')
    expr = re.sub(r'/\*.*?\*/', '', expr)
    expr = re.sub(r'//.*', '', expr).strip()
    if not expr:
        raise ValueError('vide')

    def sub(m):
        n = m.group(0)
        if n in env:
            v = env[n]
            if isinstance(v, int):
                return str(v)
            texte, params = v
```

```
            if params is not None:
                raise ValueError('macro a parametres ' + n)
            return '(%d)' % evaluer(texte, env, prof + 1)
        if re.match(r'^(0[xX][0-9a-fA-F]+|\d+)$', n):
            return n
        if n in ('and', 'or', 'not', 'if', 'else'):
            return n
        raise ValueError('inconnu ' + n)
    e = c_vers_py(expr)
    e = re.sub(r'\b[A-Za-z_]\w*\b|\b0[xX][0-9a-fA-F]+\b|\b\d+\b', sub, e)
    if re.search(r'[^\s\d()+\-*/%<>&|^~xXa-fA-F,.]|\b(?!(if|else|and|or|not)\b)[g-wyzG-WYZ_]\w*', e):
        pass
    v = eval(e, {'__builtins__': {}}, {})
    if isinstance(v, bool):
        v = int(v)
    if not isinstance(v, int):
        raise ValueError('pas entier')
    return v


_DEFINE = re.compile(r'^\s*#\s*define\s+([A-Za-z_]\w*)(\(([^)]*)\))?(?:[ \t]+(.*?))?\s*$')


def lire_defines(chemin):
    """nom -> (texte, params|None, ligne) ; joint les lignes continuees par \\."""
    res = {}
    try:
        lignes = open(chemin, errors='replace').read().split('\n')
    except OSError:
        return res
    i = 0
    while i < len(lignes):
        l = lignes[i]
        n0 = i + 1
        while l.endswith('\\') and i + 1 < len(lignes):
            i += 1
            l = l[:-1] + ' ' + lignes[i]
        m = _DEFINE.match(l)
        if m:
            texte = (m.group(4) or '').strip()
            texte = re.sub(r'/\*.*?\*/', '', texte)
            texte = re.sub(r'//.*', '', texte).strip()
            params = [p.strip() for p in m.group(3).split(',')] if m.group(2) else None
            res.setdefault(m.group(1), (texte, params, n0))
        i += 1
    return res


def valeur_macro(defs, nom, args=None):
    texte, params, _ = defs[nom]
    env = {k: (v[0], v[1]) for k, v in defs.items()}
    if params is not None:
        for p, a in zip(params, args or []):
            texte = re.sub(r'\b%s\b' % re.escape(p), '(%d)' % a, texte)
    return evaluer(texte, env)


# ---------------------------------------------------------------- extraction
class Interface:
    """Les structures API de la layer1, telles que compilees."""
    TYPES = ['T_DB_MCU_TO_DSP', 'T_DB_DSP_TO_MCU', 'T_NDB_MCU_DSP', 'T_PARAM_MCU_DSP']
    ZONES = [('W0', 'T_DB_MCU_TO_DSP', 'BASE_API_W_PAGE_0'), ('W1', 'T_DB_MCU_TO_DSP', 'BASE_API_W_PAGE_1'),
             ('R0', 'T_DB_DSP_TO_MCU', 'BASE_API_R_PAGE_0'), ('R1', 'T_DB_DSP_TO_MCU', 'BASE_API_R_PAGE_1'),
             ('NDB', 'T_NDB_MCU_DSP', 'BASE_API_NDB'), ('PARAM', 'T_PARAM_MCU_DSP', 'BASE_API_PARAM')]

    def __init__(self, fw, elf, res, tmp):
```

```
        self.fw, self.elf, self.res, self.tmp = fw, elf, res, tmp
        self.inc = os.path.join(fw, 'include')
        self.hdr = os.path.join(self.inc, 'calypso', 'dsp_api.h')
        self.env_h = os.path.join(self.inc, 'calypso', 'l1_environment.h')
        self.dsp_c = os.path.join(fw, 'calypso', 'dsp.c')
        self.disp = {}          # type -> [(nom, off, taille, n, signe)]
        self.taille_elf = {}
        self.taille_hote = {}
        self.l1 = {}            # macro layer1 -> valeur (compilee)
        self.l1_src = {}        # macro -> fichier:ligne
        self.dspc = {}          # macros de calypso/dsp.c
        self.params = {}        # dsp_params (ELF)
        self.dsp_api_init = {}
        self.commentaires = {}  # (type, champ) -> texte
        self.champs = []        # dicts
        self.par_mot = collections.defaultdict(list)

    # -- DWARF
    def _gdb(self):
        cmds = []
        for t in self.TYPES:
            cmds += ['-ex', 'echo @@%s\\n' % t, '-ex', 'ptype /o %s' % t, '-ex', 'print sizeof(%s)' % t]
        cmds += ['-ex', 'echo @@params\\n', '-ex', 'print/x dsp_params', '-ex', 'echo @@api\\n', '-ex', 'print/x
↪   dsp_api']
        gdb = shutil.which('gdb-multiarch') or shutil.which('gdb')
        out = lancer([gdb, '-batch', '-nx'] + cmds + [self.elf])
        blocs = re.split(r'^@@(\w+)$', out, flags=re.M)
        for i in range(1, len(blocs) - 1, 2):
            nom, txt = blocs[i], blocs[i + 1]
            if nom in self.TYPES:
                ch = []
                for m in re.finditer(r'/\*\s*(\d+)\s*\|\s*(\d+)\s*\*/\s*(API_SIGNED|API)\s+(\w+)(?:\[(\d+)\])?;',
                 ↪  txt):
                    ch.append((m.group(4), int(m.group(1)), int(m.group(2)), int(m.group(5) or 1), m.group(3) ==
↪   'API_SIGNED'))
                self.disp[nom] = ch
                m = re.search(r'^\$\d+ = (\d+)', txt, re.M)
                self.taille_elf[nom] = int(m.group(1)) if m else None
            elif nom == 'params':
                self.params = self._gdb_struct(txt)
            elif nom == 'api':
                self.dsp_api_init = {k: v[0] for k, v in self._gdb_struct(txt).items()}

    @staticmethod
    def _gdb_struct(txt):
        txt = txt[txt.find('{') + 1: txt.rfind('}')]
        res = {}
        for m in re.finditer(r'(\w+) = (\{[^}]*\}|-?0x[0-9a-fA-F]+|-?\d+)', txt):
            v = m.group(2)
            if v.startswith('{'):
                vals = []
                for tok in v[1:-1].split(','):
                    tok = tok.strip()
                    r = re.match(r'(-?0x[0-9a-fA-F]+|-?\d+)(?:\s*<repeats (\d+) times>)?', tok)
                    if r:
                        vals += [int(r.group(1), 0)] * int(r.group(2) or 1)
                res[m.group(1)] = vals
            else:
                res[m.group(1)] = [int(v, 0)]
        return res

    # -- offsetof hote + macros
    def _hote(self):
        defs = {}
        for f in (self.env_h, self.hdr):
            for n, (t, p, l) in lire_defines(f).items():
```

```
                if p is None and t and not n.startswith('_'):
                    defs.setdefault(n, '%s:%d' % (rel(f), l))
        src = ['#include <stdio.h>', '#include <stddef.h>', '#include "dsp_api.h"', 'int main(void) {']
        for t in self.TYPES:
            for (nom, off, taille, n, sg) in self.disp[t]:
                src.append('printf("F %s %s %%zu %%zu\\n", offsetof(%s, %s), sizeof(((%s *)0)->%s));' % (t, nom,
↪  t, nom, t, nom))
            src.append('printf("S %s %%zu\\n", sizeof(%s));' % (t, t))
        for n in sorted(defs):
            src.append('#ifdef %s\nprintf("M %s %%lld\\n", (long long)(%s));\n#endif' % (n, n, n))
        src.append('return 0; }')
        c = os.path.join(self.tmp, 'offsets.c')
        open(c, 'w').write('\n'.join(src) + '\n')
        exe = os.path.join(self.tmp, 'offsets')
        r = subprocess.run(['gcc', '-w', '-I', os.path.join(self.inc, 'calypso'), '-I', self.inc, '-o', exe, c],
                           stdout=subprocess.PIPE, stderr=subprocess.PIPE, universal_newlines=True)
        if r.returncode != 0:
            # une macro non entiere casse la compilation : on retire les M fautives
            mauvaises = set(re.findall(r"error: [^\n]*?'(\w+)'", r.stderr))
            src = [s for s in src if not any(('M %s ' % m) in s for m in mauvaises)]
            open(c, 'w').write('\n'.join(src) + '\n')
            lancer(['gcc', '-w', '-I', os.path.join(self.inc, 'calypso'), '-I', self.inc, '-o', exe, c])
        hote = {}
        for l in lancer([exe]).split('\n'):
            p = l.split()
            if not p:
                continue
            if p[0] == 'F':
                hote[(p[1], p[2])] = (int(p[3]), int(p[4]))
            elif p[0] == 'S':
                self.taille_hote[p[1]] = int(p[2])
            elif p[0] == 'M':
                self.l1[p[1]] = int(p[2])
                self.l1_src[p[1]] = defs[p[1]]
        return hote

    def _commentaires(self):
        """Commentaires de chaque champ, depuis l'en-tete PREPROCESSE (branche active)."""
        c = os.path.join(self.tmp, 'pp.c')
        open(c, 'w').write('#include "dsp_api.h"\n')
        pp = lancer(['gcc', '-E', '-C', '-P', '-I', os.path.join(self.inc, 'calypso'), '-I', self.inc, c])
        for m in re.finditer(r'typedef\s+struct\s*\{(.*?)\}\s*(\w+)\s*;', pp, re.S):
            corps, t = m.group(1), m.group(2)
            dernier = None
            for l in corps.split('\n'):
                f = re.search(r'\b(API_SIGNED|API)\s+(\w+)\s*(\[[^\]]*\])?\s*;(.*)$', l)
                if f:
                    dernier = f.group(2)
                    com = f.group(4).strip()
                    com = re.sub(r'^//\s*|^/\*\s*|\s*\*/$', '', com)
                    self.commentaires[(t, dernier)] = com
                elif dernier and re.match(r'^\s*//', l):
                    self.commentaires[(t, dernier)] += ' ' + re.sub(r'^\s*//\s*', '', l).strip()
                elif re.match(r'^\s*$', l) or re.match(r'^\s*//', l):
                    continue
                else:
                    dernier = None

    def _dsp_c(self):
        d = lire_defines(self.dsp_c)
        for n in d:
            try:
                self.dspc[n] = valeur_macro(d, n)
            except Exception:
                pass
        self.dspc_src = {n: '%s:%d' % (rel(self.dsp_c), d[n][2]) for n in d}
```

```
    def extraire(self):
        r = self.res
        self._gdb()
        hote = self._hote()
        self._commentaires()
        self._dsp_c()
        # 1. offsets ELF == offsetof hote
        nb = collections.Counter()
        for t in self.TYPES:
            for (nom, off, taille, n, sg) in self.disp[t]:
                h = hote.get((t, nom))
                if h == (off, taille):
                    nb[t] += 1
                    if r.bavard:
                        r('PASS', 'extraction', '%s.%s' % (t, nom), 'offset %d taille %d (ELF = offsetof)' % (off,
↪   taille))
                else:
                    r('FAIL', 'extraction', '%s.%s' % (t, nom), 'ELF offset %d taille %d, offsetof hote %s' % (off,
↪   taille, h))
            r('PASS', 'extraction', t, '%d champs : offsets DWARF de l\'ELF = offsetof(dsp_api.h, DSP=%s)' %
↪   (nb[t], self.l1.get('DSP')),
              rel(self.elf))
            te, th = self.taille_elf.get(t), self.taille_hote.get(t)
            if te != th:
                dernier = self.disp[t][-1]
                r('ECART', 'extraction', 'sizeof(%s)' % t,
                  'ELF %s octets, hote %s : bourrage de fin (dernier champ %s finit a %d) ; offsets inchanges' %
                  (te, th, dernier[0], dernier[1] + dernier[2]), rel(self.elf))
        # 2. bases : en-tete == initialiseur dsp_api de l'ELF
        for z, t, b in self.ZONES:
            if b not in self.l1:
                r('FAIL', 'extraction', b, 'absent de dsp_api.h')
        for cle, base in (('ndb', 'BASE_API_NDB'), ('db_r', 'BASE_API_R_PAGE_0'), ('db_w', 'BASE_API_W_PAGE_0'),
         ↪  ('param', 'BASE_API_PARAM')):
            v = self.dsp_api_init.get(cle)
            st = 'PASS' if v == self.l1.get(base) else 'FAIL'
            r(st, 'extraction', 'dsp_api.%s' % cle, 'ELF %s, %s = 0x%08x' % (hex(v) if v is not None else None,
↪   base, self.l1.get(base, 0)),
              '%s ; %s' % (rel(self.dsp_c), self.l1_src.get(base, '')))
        # 3. table des champs
        for z, t, b in self.ZONES:
            base = self.l1[b]
            for (nom, off, taille, n, sg) in self.disp[t]:
                arm = base + off
                octet = arm - API_ARM
                mot = API_DSP + octet // 2
                c = dict(zone=z, type=t, nom=nom, off=off, taille=taille, n=n, signe=sg, arm=arm, octet=octet,
                         mot=mot, mots=list(range(mot, mot + taille // 2)),
                         commentaire=self.commentaires.get((t, nom), ''), trou='hole' in nom)
                self.champs.append(c)
                for i, w in enumerate(c['mots']):
                    self.par_mot[w].append((c, i))
        # 4. geometrie : pages disjointes, PARAM dans un trou du NDB, tout dans la fenetre DSP
        zones = {z: (self.l1[b] - API_ARM, self.l1[b] - API_ARM + self.taille_hote[t]) for z, t, b in self.ZONES}
        for a, b in (('W0', 'W1'), ('W1', 'R0'), ('R0', 'R1'), ('R1', 'NDB')):
            ok = zones[a][1] <= zones[b][0]
            r('PASS' if ok else 'FAIL', 'extraction', 'pages %s/%s' % (a, b),
              '%s 0x%03x..0x%03x, %s commence en 0x%03x' % (a, zones[a][0], zones[a][1] - 1, b, zones[b][0]))
        ndb0, ndb1 = zones['NDB']
        p0, p1 = zones['PARAM']
        trou = [c for c in self.champs if c['zone'] == 'NDB' and c['octet'] <= p0 < c['octet'] + c['taille']]
        if trou:
            tc = trou[0]
            utile = self.disp['T_PARAM_MCU_DSP'][-1]
            fin_utile = p0 + utile[1] + utile[2]
```

```
            ok = tc['trou'] and fin_utile <= tc['octet'] + tc['taille']
            r('PASS' if ok else 'FAIL', 'extraction', 'PARAM dans le NDB',
              'PARAM (0x%03x..0x%03x utiles) recouvre NDB.%s (0x%03x..0x%03x)%s' %
              (p0, fin_utile - 1, tc['nom'], tc['octet'], tc['octet'] + tc['taille'] - 1,
               '' if ok else ' : le recouvrement deborde sur un champ utile'))
        fin = max(v[1] for v in zones.values())
        r('PASS' if fin <= API_MOTS * 2 else 'FAIL', 'extraction', 'fenetre DSP',
          'structures 0x000..0x%03x octets = mots DSP 0x0800..0x%04x (fenetre 0x0800..0x%04x)' %
          (fin - 1, API_DSP + (fin - 1) // 2, API_DSP + API_MOTS - 1))
        # 5. chargeur (calypso/dsp.c)
        base_ram = self.dspc.get('BASE_API_RAM')
        r('PASS' if base_ram == API_ARM else 'FAIL', 'extraction', 'BASE_API_RAM', '0x%08x' % (base_ram or 0),
↪   self.dspc_src.get('BASE_API_RAM', ''))
        api_size = self.dspc.get('API_SIZE')
        r('PASS' if api_size == API_MOTS else 'FAIL', 'extraction', 'API_SIZE', '0x%x mots' % (api_size or 0),
↪   self.dspc_src.get('API_SIZE', ''))
        self.bl = {}
        for n in ('BL_CMD_STATUS', 'BL_ADDR_LO', 'BL_ADDR_HI', 'BL_SIZE'):
            if n in self.dspc:
                a = self.dspc[n]
                self.bl[n] = API_DSP + (a - API_ARM) // 2
                dessous = [c['nom'] for c, i in self.par_mot.get(self.bl[n], [])]
                r('INFO', 'extraction', n, 'ARM 0x%08x = mot DSP 0x%04x (recouvre NDB %s)' % (a, self.bl[n],
↪   ','.join(dessous) or '-'),
                  self.dspc_src[n])
        # 6. adresse en dur de la layer1 elle-meme (dsp.c SC_CHKSUM_VER)
        m = re.search(r'#define\s+SC_CHKSUM_VER\s+\(BASE_API_W_PAGE_0\s*\+\s*\(2\s*\*\s*\((0x[0-9A-Fa-f]+)\s*-
↪   \s*0x800\)\)\)',
                      open(self.dsp_c).read())
        if m:
            w = int(m.group(1), 16)
            noms = [c['nom'] for c, i in self.par_mot.get(w, []) if c['zone'] == 'NDB']
            r('PASS' if noms == ['d_version_number2'] else 'FAIL', 'extraction', 'SC_CHKSUM_VER (dsp.c)',
              'mot DSP 0x%04x = NDB.%s (« dsp patch version »)' % (w, ','.join(noms)), rel(self.dsp_c))
        return self

    # -- recherche
    def champ(self, zone, nom):
        for c in self.champs:
            if c['nom'] == nom and (c['zone'] == zone or (zone in ('W', 'R') and c['zone'].startswith(zone))):
                return c
        return None

    def zone_base(self, z):
        for zz, t, b in self.ZONES:
            if zz == z:
                return self.l1[b]
        raise KeyError(z)

    def noms(self):
        return set(c['nom'] for c in self.champs)

    def au_mot(self, w):
        """Champ(s) au mot DSP w, PARAM avant les trous du NDB."""
        l = self.par_mot.get(w, [])
        return sorted(l, key=lambda ci: (ci[0]['trou'], ci[0]['zone'] == 'NDB'))
```

3.41 /opt/GSM/c54x_exe/tools/rejeu_banc.c

43036 octets, 737 lignes → 737 lignes

```
/* rejeu_banc.c - rejoue hors banc un TCH enregistre par c54x_exe.
 *
 * Entree : /dev/shm/calypso_rejeu_tch.bin (calypso_bsp.c + pont.c,
```

```
 * CALYPSO_REJEU_ENREG). Enregistrements :
 *   'S' tick, API RAM complete        'D' tick, registres, memoire de donnees
 *   'T' tick, drapeaux, budget        'A' tick, fenetre, n x (mot, valeur)
 *   'B' tick, tn, fn, one_shot, nwin, n, I/Q
 * On boote le DSP comme tools/tch_rejeu, on pose l'etat 'S'/'D', puis chaque
 * tick dans l'ordre de pont.c : ecritures ARM d'avant le TICK, interruption
 * trame, phase A jusqu'a l'IDLE (TCH), ecritures ARM d'entre A et GO,
 * livraisons d'I/Q, reste du budget, pompe DMA. On imprime chaque resultat
 * SACCH (a_cd) et FACCH (a_fd).
 *
 * Si le rejeu reproduit le Fire KO du banc, on peut iterer ici sans relancer
 * le banc ; REJEU_SANS_D=1 garde l'etat du boot local au lieu de 'D'.
 *
 * Diagnostic (avec REJEU_TRACE=1) : REJEU_STOP_W=adr (ecritures d'un mot, PC fautif), REJEU_STOP_NW=n
 * (nombre d'ecritures avant arret, 6), REJEU_ANNEAU_W=n (n instructions avant la 1re ecriture),
 * REJEU_STOP_PC=pc, REJEU_STOP_TMIN=tick (ignorer avant ce tick), REJEU_PEEK=adr:n,... (memoire au
 * point d'arret). L'API RAM est dsp->data[0x0800..] : a_dd_0 = 0x09f0.
 *
 * [2026-10-04] Montant et descendant du TCH :
 *   REJEU_TX=1        chaque burst montant final (script TSP, src/tsp_tx.c) : fn de la page W, type,
 *                     TSC, hl/hu, 148 bits -- a decoder hors ligne (libosmocoding, A5 au fn de la ROM)
 *   REJEU_TX_PHASE=1  a quel moment du tick la ROM finit ce script (phase A, B, pompe DMA)
 *   REJEU_BLUD=1      qui pose / efface B_BLUD de a_cu, a_fu, a_du (ARM a T, ROM a T+1)
 *   REJEU_DL=1        chaque burst descendant livre (fenetre 0) : fn du pont, fn de la page W (a_a5fn,
 *                     les deux doivent etre egaux), bits demodules (grossierement : TSC a 14/26)
 * C'est REJEU_BLUD qui a montre que montant.c effacait B_BLUD de a_fu avant la ROM (appels sans FACCH).
 *
 *   ./rejeu_banc [fichier] [ticks max]
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <stdbool.h>
#include "qemu/thread.h"
#include "calypso_c54x.h"
#include "calypso_dma.h"
#include "calypso_bsp.h"
#include "calypso_rhea_dma.h"
#include "tsp_tx.h"
#include "calypso_vsp.h"
#include "hw/arm/calypso/calypso_api.h"
#include <math.h>
#include <osmocom/core/bits.h>
#include <osmocom/gsm/a5.h>

uint32_t g_c54x_exe_fn;
uint32_t calypso_trx_get_fn(void) { return g_c54x_exe_fn; }
void calypso_inth_arm_ack(void) { }
QemuMutex calypso_pcb_daram_lock;
void cpu_physical_memory_rw(uint64_t addr, void *buf, uint64_t len, bool wr)
{ (void)addr; if (!wr) memset(buf, 0, len); }

#define API_WORDS 0x2000u          /* fenetre API vue du DSP : data 0x0800..0x27ff */
#define ENREG_API_WORDS CALYPSO_API_WORDS   /* taille de l'API RAM enregistree par pont.c */
#define NDB 0xD4u
#define W_PAGE(p) ((p) ? 0x14u : 0x00u)   /* T_DB_MCU_TO_DSP, 17 mots (comme src/rejouer.c) */
static uint32_t g_page_it_fn; static unsigned g_page_it_pg; static uint16_t g_page_it_td, g_page_it_tu,
 ↪  g_page_it_tra, g_page_it_ctrl, g_it_tch_mode, g_it_du1;
#define PARAM 0x431u
#define BL_ADDR_HI_W 0x7FCu
#define BL_SIZE_W 0x7FDu
#define BL_ADDR_LO_W 0x7FEu
#define BL_STATUS_W 0x7FFu

typedef struct {
```

```
    int64_t a, b; uint16_t ar[8], t, trn, sp, bk, brc, rsa, rea, st0, st1, pmst, imr, ifr, xpc;
    uint32_t pc; uint8_t idle, running;
} EnregRegs;

static C54xState *dsp;
static uint16_t *api;
static uint8_t *fichier;
static size_t taille;

static uint32_t be32(const uint8_t *p) { return (uint32_t)p[0] << 24 | p[1] << 16 | p[2] << 8 | p[3]; }
static uint16_t be16(const uint8_t *p) { return (uint16_t)(p[0] << 8 | p[1]); }

static long pump(long max, int stop_idle)
{
    long b = 0;
    while (b < max && dsp->running) {
        int ex = c54x_run(dsp, 256); if (ex <= 0) break; b += ex;
        if (stop_idle && dsp->idle) break;
    }
    return b;
}

static void reveil(void)
{
    if (dsp->idle && calypso_rhea_dma_irq_level() && (dsp->imr & (1u << 14)) && !(dsp->ifr & (1u << 14)))
        c54x_interrupt_ex(dsp, 30, 14);
    if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
}

static void boot(void)
{
    memset(api, 0, API_WORDS * 2);
    long b = 0;
    while (api[BL_STATUS_W] != 1 && b < 8000000) { int ex = c54x_run(dsp, 256); if (ex <= 0) break; b += ex; }
    api[BL_ADDR_HI_W] = 0; api[BL_ADDR_LO_W] = 0x7000; api[BL_SIZE_W] = 0; api[BL_STATUS_W] = 2;
    b = pump(4000000, 1);
    printf("boot local : %ld insn, idle=%d\n", b, dsp->idle);
}

/* SONDE (REJEU_SONDE=1) : ou la ROM range-t-elle les bursts SACCH ?
 * Pour chaque burst du canal (slot 0 livre par le BSP), on retrouve les bits
 * par demodulation GMSK differentielle (polarite calee sur la TSC7), on les
 * dechiffre (Kc et d_a5mode pris dans l'API RAM), puis au tick ou a_cd change
 * on cherche dans la memoire du DSP ou sont les 116 bits de chacun des quatre
 * bursts du bloc (bits souples : signe negatif ou positif = 1). */
static const uint8_t TSC7[26] = {1,1,1,0,1,1,1,1,0,0,0,1,0,0,1,0,1,1,1,0,1,1,1,1,0,0};
static struct { uint32_t fn; uint8_t b[148]; } memo[4096];
static int nmemo;
/* Demodulation GMSK differentielle d'un burst livre (148 bits, polarite calee sur la TSC7) ;
 * rend le score TSC (0..26) ou -1 si la fenetre est trop courte. */
static int demod_burst(const int16_t *iq, int n, int marge, uint8_t *out, int *polarite)
{
    if (n / 2 < marge + 148) return -1;
    uint8_t d[148], b[2][148];
    for (int k = 0; k < 148; k++) {
        int i0 = 2 * (marge + k), i1 = i0 - 2;
        double ph1 = atan2(iq[i0 + 1], iq[i0]);
        double ph0 = k ? atan2(iq[i1 + 1], iq[i1]) : 0;
        double dp = ph1 - ph0;
        while (dp > M_PI) dp -= 2 * M_PI;
        while (dp < -M_PI) dp += 2 * M_PI;
        d[k] = dp > 0;
    }
    int best = -1, bs = -1;
    for (int pol = 0; pol < 2; pol++) {
        uint8_t prev = 0;
```

```
        for (int k = 0; k < 148; k++) { uint8_t dd = d[k] ^ pol; b[pol][k] = dd ^ prev; prev = b[pol][k]; }
        int sc = 0; for (int k = 0; k < 26; k++) sc += b[pol][61 + k] == TSC7[k];
        if (sc > bs) { bs = sc; best = pol; }
    }
    memcpy(out, b[best], 148); *polarite = best;
    return bs;
}
static void memo_burst(uint32_t fn, const int16_t *iq, int n, int marge)
{
    uint8_t b[148]; int pol;
    int bs = demod_burst(iq, n, marge, b, &pol);
    if (bs < 0) return;
    int slot = nmemo % 4096; nmemo++;
    memo[slot].fn = fn; memcpy(memo[slot].b, b, 148);
    static int nlog; if (nlog++ < 3) printf("  [sonde] burst fn=%u : TSC %d/26 (differentielle, pol=%d)\n", fn,
     ↪  bs, pol);
}
static const uint8_t *memo_trouver(uint32_t fn)
{
    /* Bits exacts de la sonde du BSP (calypso_sacch_tf.bin), s'il y en a. */
    static uint8_t (*vrai)[156]; static int nvrai = -1;
    if (nvrai < 0) {
        nvrai = 0;
        const char *nom = getenv("REJEU_BITS");
        FILE *f = nom ? fopen(nom, "rb") : NULL;
        if (f) { vrai = malloc(4096 * 156); while (nvrai < 4096 && fread(vrai[nvrai], 1, 156, f) == 156) nvrai++;
         ↪  fclose(f); }
        if (nom) printf("  [sonde] %d bursts exacts lus dans %s\n", nvrai, nom);
    }
    for (int i = 0; i < nvrai; i++) if (be32(vrai[i] + 4) == fn) return vrai[i] + 8;
    if (getenv("REJEU_BITS")) return NULL;
    for (int i = 0; i < 4096 && i < nmemo; i++) if (memo[i].fn == fn) return memo[i].b;
    return NULL;
}
static void sonde_bloc(uint32_t fn_dernier)
{
    uint8_t kc[8]; uint16_t *akc = &api[NDB + 0x2CE / 2]; int algo = api[NDB + 0x1CE / 2];
    for (int i = 0; i < 4; i++) { kc[7 - 2 * i] = akc[i] & 0xff; kc[6 - 2 * i] = akc[i] >> 8; }
    for (int k = 0; k < 4; k++) {
        uint32_t fn = fn_dernier - 26 * (3 - k);
        const uint8_t *b0 = memo_trouver(fn);
        if (!b0) { printf("    burst %d fn=%u : non enregistre\n", k, fn); continue; }
        uint8_t b[148]; memcpy(b, b0, 148);
        if (algo) { ubit_t dl[114], ul[114]; osmo_a5(algo, kc, fn, dl, ul);
            for (int j = 0; j < 57; j++) { b[3 + j] ^= dl[j]; b[88 + j] ^= dl[57 + j]; } }
        uint8_t t[116]; memcpy(t, b + 3, 58); memcpy(t + 58, b + 87, 58);
        int best[2] = {0, 0}; unsigned ad[2] = {0, 0};
        for (unsigned a = 0x60; a + 116 < C54X_DATA_SIZE; a++) {
            int m0 = 0, m1 = 0;
            for (int i = 0; i < 116; i++) { int16_t v = (int16_t)dsp->data[a + i];
                if (v != 0) { if ((v < 0) == (t[i] != 0)) m0++; if ((v > 0) == (t[i] != 0)) m1++; } }
            if (m0 > best[0]) { best[0] = m0; ad[0] = a; }
            if (m1 > best[1]) { best[1] = m1; ad[1] = a; }
        }
        /* aussi en 57+57 sans les bits de vol (e[] seuls) */
        uint8_t e[114]; memcpy(e, b + 3, 57); memcpy(e + 57, b + 88, 57);
        int be = 0; unsigned ae = 0;
        for (unsigned a = 0x60; a + 114 < C54X_DATA_SIZE; a++) {
           int m = 0; for (int i = 0; i < 114; i++) { int16_t v = (int16_t)dsp->data[a + i]; if (v != 0 && (v < 0)
            ↪  == (e[i] != 0)) m++; }
            if (m > be) { be = m; ae = a; }
        }
        printf("    burst %d fn=%u (fn%%104=%u) : 116b neg %d@%04x pos %d@%04x | 114b neg %d@%04x\n",
               k, fn, fn % 104, best[0], ad[0], best[1], ad[1], be, ae);
    }
}
```

```
/* REJEU_TRACE=1 : pas a pas, anneau des dernieres instructions, arret des que
 * SP sort de la pile ou que PC tombe en DARAM basse (< 0x0800, hors OVLY). */
uint16_t prog_fetch(C54xState *s, uint16_t pc);
uint16_t data_read(C54xState *s, uint16_t addr);
#define ANNEAU 4096
static struct { uint16_t pc, op, op2, sp, st0, st1, ar[8]; uint32_t tick; int64_t a, b; } anneau[ANNEAU];
static unsigned apos;
static uint16_t *g_pcs; static unsigned long *g_pcs_cpt; static int g_npcs;
static unsigned long *g_pcplage; static unsigned g_pclo, g_pchi;
static void pcplage_resume(void)
{
    for (unsigned a = g_pclo; a <= g_pchi; a++) if (g_pcplage[a - g_pclo]) printf("PCH %04x %lu\n", a, g_pcplage[a
     ↪  - g_pclo]);
}
static void pcs_resume(void)
{
    for (int q = 0; q < g_npcs; q++) printf("PCS %04x : %lu\n", g_pcs[q], g_pcs_cpt[q]);
}
static int trace_on = -1, plante;
static long courir(long n);
static int vsp_courir(C54xState *d, int n) { (void)d; return (int)courir(n); }
static long courir(long n)
{
    if (trace_on < 0) trace_on = getenv("REJEU_TRACE") != NULL;
    /* [2026-10-03] Boucler jusqu'au budget, comme le banc (pont.c c54x_run_profile) : c54x_run rend la
     * main de lui-meme a un point interruptible au bout de CALYPSO_DSP_YIELD instructions (32768 par
     * defaut). Un seul appel perdait le reste du budget de chaque phase : le DSP ne finissait pas son
     * travail (SACCH 70 bonnes / 69 Fire KO, a_dd[2] jamais ecrit = FFFF, BFI=0) la ou le banc et le pas
     * a pas donnent SACCH 132/33 et BFI 2654/2654. */
    if (!trace_on) {
        long k = 0;
        while (k < n && dsp->running && !dsp->idle) {
            int r = c54x_run(dsp, (int)(n - k));
            if (r <= 0) break;
            k += r;
        }
        return k;
    }
    long k = 0;
    for (; k < n && dsp->running && !dsp->idle && !plante; k++) {
        unsigned i = apos++ & (ANNEAU - 1);
        uint16_t pc = dsp->pc & 0xffff;
        anneau[i].pc = pc; anneau[i].op = prog_fetch(dsp, pc); anneau[i].op2 = prog_fetch(dsp, pc + 1);
       anneau[i].sp = dsp->sp; anneau[i].st0 = dsp->st0; anneau[i].st1 = dsp->st1; anneau[i].tick = g_c54x_exe_fn;
        memcpy(anneau[i].ar, dsp->ar, sizeof dsp->ar); anneau[i].a = dsp->a; anneau[i].b = dsp->b;
        {   /* [2026-10-15] REJEU_PCPLAGE=lo-hi : histogramme des PC executes dans la plage (fin de rejeu) */
            static int init; static unsigned lo, hi; static unsigned long *h;
            if (!init) { init = 1; const char *e = getenv("REJEU_PCPLAGE");
                if (e && sscanf(e, "%x-%x", &lo, &hi) == 2 && hi >= lo) { h = calloc(hi - lo + 1, sizeof *h);
                 ↪  g_pcplage = h; g_pclo = lo; g_pchi = hi; atexit(pcplage_resume); } }
            if (h && pc >= lo && pc <= hi && dsp->xpc == 0) h[pc - lo]++;
        }
        {   /* [2026-10-15] REJEU_PCS=pc1,pc2,... : nombre d'executions de chaque PC (resume en fin de rejeu) */
            static int npcs = -1; static uint16_t pcs[32]; static unsigned long cpt[32];
            if (npcs < 0) { npcs = 0; const char *e = getenv("REJEU_PCS");
                while (e && *e && npcs < 32) { pcs[npcs++] = (uint16_t)strtoul(e, (char **)&e, 16); if (*e == ','
                 ↪  e++; else break; }
                if (npcs) atexit(pcs_resume); }
            for (int q = 0; q < npcs; q++) if (pc == pcs[q]) cpt[q]++;
            g_pcs = pcs; g_pcs_cpt = cpt; g_npcs = npcs;
        }
        {   /* [2026-10-16] REJEU_CODEC=fichier (avec REJEU_TRACE=1) : le codeur FR de la ROM, trame par trame.
             * SO = les 160 echantillons qu'il lit (PDROM 0xe6af, deja (s>>3)<<2 comme Gsm_Preprocess),
             * ENC = la trame qu'il rend (0x0a24, format TI : c000 52b0 0000 puis 260 bits). Avec
             * CALYPSO_VSP_UL_FICHIER (micro de test), tools/cmp_codeur.py les compare a libgsm. */
```

```
            static int vt = -1; static FILE *fv;
            if (vt < 0) { const char *e = getenv("REJEU_CODEC"); vt = e && *e; if (vt) fv = fopen(e, "w"); if (!fv)
             ↪  vt = 0; }
            if (vt && dsp->xpc == 0) {
                if (pc == 0xe6af) { fprintf(fv, "SO"); for (int j = 0; j < 160; j++) fprintf(fv, " %d",
                 ↪  (int16_t)dsp->data[0x2bfd + j]); fprintf(fv, "\n"); }
                if (pc == 0xe3fe) { fprintf(fv, "ENC fn=%u A", g_page_it_fn); for (int j = 0; j < 20; j++)
                 ↪  fprintf(fv, " %04x", dsp->data[0x0a24 + j]); fprintf(fv, "\n"); }
            }
        }
        {   /* [2026-10-15] REJEU_MOTS=1 : le tampon de burst 0x3f8a..0x3f91 tel que la routine d'emballage
             * le recoit, AVANT le XOR A5 (PC 0x85cb) -- a correler avec les bits emis (verif_codage). */
            static int mots = -1; if (mots < 0) mots = getenv("REJEU_MOTS") != NULL;
            if (mots && pc == 0x85cb) {
                printf("MOTS tick=%u fn=%u :", g_c54x_exe_fn, g_page_it_fn);
                for (int j = 0; j < 8; j++) printf(" %04x", dsp->data[0x3f8a + j]);
                printf("\n");
            }
        }
        static int wa = -2; static uint16_t wv;
        if (wa == -2) { const char *e = getenv("REJEU_STOP_W"); wa = e ? (int)strtoul(e, NULL, 16) : -1; }
        if (wa >= 0) wv = dsp->data[wa];
        c54x_run(dsp, 1);
        uint16_t npc = dsp->pc & 0xffff;
        static long tmin = -2; if (tmin == -2) { const char *e = getenv("REJEU_STOP_TMIN"); tmin = e ? atol(e) :
         ↪  0; }
        if (wa >= 0 && dsp->data[wa] != wv && (long)g_c54x_exe_fn >= tmin) {
            printf("ECRITURE data[%04x] %04x -> %04x par pc=%04x tick=%u : AR0=%04x AR2=%04x AR3=%04x AR4=%04x
↪  BK=%04x "
                   "[4bcc]=%04x d_task_md(W0/W1)=%04x/%04x d_task_d=%04x/%04x\n",
                   wa, wv, dsp->data[wa], pc, g_c54x_exe_fn, dsp->ar[0], dsp->ar[2], dsp->ar[3], dsp->ar[4],
↪  dsp->bk,
                   dsp->data[0x4bcc], api[4], api[0x14 + 4], api[0], api[0x14]);
            if (getenv("REJEU_ANNEAU_W")) {
                int m = atoi(getenv("REJEU_ANNEAU_W"));
                for (int j = m; j > 0; j--) {
                    unsigned q = (apos - j) & (ANNEAU - 1);
                   printf("  %4d pc=%04x op=%04x %04x ar0=%04x ar2=%04x ar3=%04x ar4=%04x ar5=%04x sp=%04x\n", -j,
↪  anneau[q].pc,
                           anneau[q].op, anneau[q].op2, anneau[q].ar[0], anneau[q].ar[2], anneau[q].ar[3],
↪  anneau[q].ar[4], anneau[q].ar[5], anneau[q].sp);
                }
                exit(4);
            }
            static int nw; if (++nw >= (getenv("REJEU_STOP_NW") ? atoi(getenv("REJEU_STOP_NW")) : 6)) exit(4);
        }
        if (dsp->sp < 0x5900 || dsp->sp > 0x5c00 || (getenv("REJEU_STOP_PC") && (long)g_c54x_exe_fn >=
         ↪  (getenv("REJEU_STOP_TMIN") ? atol(getenv("REJEU_STOP_TMIN")) : 0) && npc ==
         ↪  (uint16_t)strtoul(getenv("REJEU_STOP_PC"), NULL, 16)) || (getenv("REJEU_STOP_DEBUG") &&
         ↪  dsp->data[0x08dc] != 0x0074 && g_c54x_exe_fn > 6200)) {
            plante = 1;
            printf("PLANTAGE tick=%u pc=%04x sp=%04x xpc=%d\n", g_c54x_exe_fn, npc, dsp->sp, dsp->xpc);
            {   /* REJEU_PEEK=debut:n[,debut:n...] : memoire lue par data_read au point d'arret */
                const char *pk = getenv("REJEU_PEEK");
                while (pk && *pk) {
                    unsigned a0 = strtoul(pk, (char **)&pk, 16), m = 1;
                    if (*pk == ':') m = strtoul(pk + 1, (char **)&pk, 10);
                    printf("PEEK %04x:", a0);
                    for (unsigned i = 0; i < m; i++) printf(" %04x", data_read(dsp, (uint16_t)(a0 + i)));
                    printf("\n");
                    if (*pk == ',') pk++; else break;
                }
            }
            int m = getenv("REJEU_ANNEAU") ? atoi(getenv("REJEU_ANNEAU")) : 120;
            for (int j = m; j > 0; j--) {
                unsigned q = (apos - j) & (ANNEAU - 1);
```

```
                printf("  %4d t=%u pc=%04x op=%04x %04x sp=%04x st0=%04x st1=%04x ar=%04x %04x %04x %04x %04x %04x
 ↪  %04x %04x a=%010llx b=%010llx\n",
                       -j, anneau[q].tick, anneau[q].pc, anneau[q].op, anneau[q].op2, anneau[q].sp, anneau[q].st0,
                        ↪  anneau[q].st1,
                       anneau[q].ar[0], anneau[q].ar[1], anneau[q].ar[2], anneau[q].ar[3], anneau[q].ar[4],
 ↪  anneau[q].ar[5],
                       anneau[q].ar[6], anneau[q].ar[7], (unsigned long long)(anneau[q].a & 0xFFFFFFFFFFULL),
                       (unsigned long long)(anneau[q].b & 0xFFFFFFFFFFULL));
            }
            exit(3);
        }
    }
    return k;
}

/* [2026-10-04] REJEU_TX_PHASE=1 : a quel moment du tick la ROM finit-elle le burst montant dans le
 * script TSP (apres la phase A, apres la phase B, a quelle iteration de la pompe DMA) ? Compare les 16
 * mots BULDATA a la derniere lecture, sans toucher au `nouveau` de tsp_tx_lire (copie locale). */
static void tsp_phase(uint32_t tick, const char *etape, int k)
{
    static uint16_t vu[16]; static int actif = -1;
    if (actif < 0) actif = getenv("REJEU_TX_PHASE") != NULL;
    if (!actif) return;
    struct tsp_tx_burst tb; bool nouveau;
    const uint16_t *d = dsp->data;
    uint16_t mots[16] = {0}; bool trouve = false;
    for (unsigned a = 0x3cbb; a + 18 <= 0x3d00 && !trouve; a++) {
        if (d[a] != 0x1c0a) continue;
        int ok = 1; for (int i = 0; i < 16; i++) if ((d[a + 2 + i] & 0x3e) != 0x06 && (d[a + 2 + i] & 0x3e) !=
         ↪  0x04) { ok = 0; break; }
        if (ok) { memcpy(mots, &d[a + 2], sizeof mots); trouve = true; }
    }
    if (!trouve || memcmp(mots, vu, sizeof vu) == 0) return;
    memcpy(vu, mots, sizeof vu);
    (void)tb; (void)nouveau;
    printf("tick=%u SCRIPT TSP nouveau : %s%s%d\n", tick, etape, k >= 0 ? " iteration " : "", k >= 0 ? k : 0);
}

/* Un tick = les enregistrements entre deux 'T'. */
typedef struct { size_t debut, fin; uint32_t tick; uint8_t drap; long budget; } Tick;

int main(int argc, char **argv)
{
    const char *nom = argc > 1 ? argv[1] : "/dev/shm/calypso_rejeu_tch.bin";
    long max_ticks = argc > 2 ? atol(argv[2]) : 100000;
    FILE *f = fopen(nom, "rb");
    if (!f) { perror(nom); return 1; }
    fseek(f, 0, SEEK_END); taille = (size_t)ftell(f); fseek(f, 0, SEEK_SET);
    fichier = malloc(taille);
    if (fread(fichier, 1, taille, f) != taille) { printf("lecture courte\n"); return 1; }
    fclose(f);

    setenv("CALYPSO_BSP_PORT", "16703", 1);
    setenv("CALYPSO_BSP_BIND_ADDR", "127.0.0.1", 1);
    setenv("CALYPSO_BSP_HORLOGE", "0", 1);
    setenv("CALYPSO_C54X_IRQ_LEVEL", "1", 0);
    setenv("CALYPSO_BSP_STREAM", "1", 0);
    setenv("CALYPSO_RHEA_DMA_XFER", "1", 0);
    setenv("CALYPSO_REJEU_ENREG", "0", 1);
    qemu_mutex_init(&calypso_pcb_daram_lock);
    dsp = c54x_init();
    api = &dsp->data[0x800];
    c54x_set_api_ram(dsp, api);
    static const struct { const char *s; uint32_t a; bool p; } R[] = {
        { "PROM0", 0x07000, true }, { "PROM1", 0x18000, true }, { "PROM2", 0x28000, true },
```

```
        { "PROM3", 0x38000, true }, { "DROM", 0x09000, false }, { "PDROM", 0x0E000, false }, { "PDROM", 0x0E000,
         ↪  true } };
    for (unsigned i = 0; i < 7; i++) {
        char c[256]; snprintf(c, sizeof c, "/opt/GSM/calypso_dsp.%s.bin", R[i].s);
        if (c54x_load_section(dsp, c, R[i].a, R[i].p) < 0) { printf("ROM %s\n", c); return 1; }
    }
    c54x_load_registers(dsp, "/opt/GSM/calypso_dsp.Registers.bin");
    c54x_reset(dsp);
    calypso_dma_init();
    calypso_bsp_init(dsp);
    boot();

    /* Premiere passe : l'etat de depart et le decoupage en ticks. */
    static Tick ticks[40000]; int nt = 0;
    size_t p = 0;
    while (p < taille) {
        uint8_t k = fichier[p];
        size_t l;
        switch (k) {
        case 'S': l = 5 + (size_t)ENREG_API_WORDS * 2; break;
        case 'D': l = 5 + sizeof(EnregRegs) + C54X_DATA_SIZE * 2; break;
        case 'P': l = 5 + 0x8000 * 2; break;   /* [2026-10-22] memoire programme 0x0000-0x7FFF */
        case 'T': l = 10; break;
        case 'A': l = 8 + 4 * (size_t)be16(fichier + p + 6); break;
        case 'B': l = 15 + 2 * (size_t)be16(fichier + p + 13); break;
        default: printf("enregistrement inconnu 0x%02x a %zu\n", k, p); goto fini;
        }
        if (p + l > taille) break;
        if (k == 'S') {
            memcpy(api, fichier + p + 5, API_WORDS * 2);
            printf("etat 'S' pose (tick %u)\n", be32(fichier + p + 1));
        } else if (k == 'P' && !getenv("REJEU_SANS_D")) {
            memcpy(dsp->prog, fichier + p + 5, 0x8000 * 2);
            printf("etat 'P' pose : memoire programme 0x0000-0x7FFF\n");
        } else if (k == 'D' && !getenv("REJEU_SANS_D")) {
            EnregRegs r; memcpy(&r, fichier + p + 5, sizeof r);
            uint16_t sauve_api[API_WORDS]; memcpy(sauve_api, api, sizeof sauve_api);
            memcpy(dsp->data, fichier + p + 5 + sizeof r, C54X_DATA_SIZE * 2);
            memcpy(api, sauve_api, sizeof sauve_api);    /* 'S' fait foi pour l'API RAM */
            dsp->a = r.a; dsp->b = r.b; memcpy(dsp->ar, r.ar, sizeof r.ar); dsp->t = r.t; dsp->trn = r.trn;
            dsp->sp = r.sp; dsp->bk = r.bk; dsp->brc = r.brc; dsp->rsa = r.rsa; dsp->rea = r.rea;
            dsp->st0 = r.st0; dsp->st1 = r.st1; dsp->pmst = r.pmst; dsp->imr = r.imr; dsp->ifr = r.ifr;
            dsp->xpc = r.xpc; dsp->pc = r.pc; dsp->idle = r.idle; dsp->running = r.running;
            printf("etat 'D' pose : pc=%04x sp=%04x idle=%d imr=%04x\n", r.pc & 0xffff, r.sp, r.idle, r.imr);
            {   /* REJEU_POKE=adr=val[,adr=val...] : corriger l'etat de depart */
                const char *e = getenv("REJEU_POKE");
                while (e && *e) {
                    unsigned a, v; if (sscanf(e, "%x=%x", &a, &v) != 2) break;
                    printf("poke data[%04x] = %04x (etait %04x)\n", a, v, dsp->data[a & 0xffff]);
                    dsp->data[a & 0xffff] = (uint16_t)v;
                    e = strchr(e, ','); if (e) e++;
                }
            }
        } else if (k == 'T') {
            if (nt > 0) ticks[nt - 1].fin = p;
            if (nt < 40000) {
                ticks[nt].debut = p; ticks[nt].tick = be32(fichier + p + 1);
                ticks[nt].drap = fichier[p + 5]; ticks[nt].budget = (long)be32(fichier + p + 6);
                nt++;
            }
        }
        p += l;
    }
fini:
    if (nt > 0) ticks[nt - 1].fin = p;
    printf("%d ticks enregistres (%u..%u)\n", nt, nt ? ticks[0].tick : 0, nt ? ticks[nt - 1].tick : 0);
```

```
uint16_t prec_cd = 0xffff, prec_fd = 0xffff;
int sacch_ok = 0, sacch_ko = 0;
unsigned long tx_n = 0, tx_rach = 0, tx_nb = 0, tx_inc = 0;   /* bursts montants finaux (script TSP) */
unsigned long tx_hist26[26] = {0};
static char tx_dernier[16] = "-";
uint16_t prec_dd = 0; unsigned long parole_n = 0, parole_bfi = 0, parole_err = 0;
for (int it = 0; it < nt && it < max_ticks; it++) {
    Tick *t = &ticks[it];
    g_c54x_exe_fn = t->tick;
    long budget = t->budget;
    /* [2026-10-22] REJEU_BUDGET=n : budget par tick impose (le banc enregistre INSNS) */
    if (getenv("REJEU_BUDGET")) budget = atol(getenv("REJEU_BUDGET"));
    uint32_t dernier_livre = 0xffffffffu;
    /* ecritures ARM, fenetre 0 */
    for (size_t q = t->debut; q < t->fin;) {
        uint8_t k = fichier[q];
        size_t l = k == 'T' ? 10 : k == 'A' ? 8 + 4 * (size_t)be16(fichier + q + 6)
                 : k == 'B' ? 15 + 2 * (size_t)be16(fichier + q + 13) : 0;
        if (!l) break;
        if (k == 'A' && fichier[q + 5] == 0)
            for (unsigned i = 0, n = be16(fichier + q + 6); i < n; i++) {
                unsigned a = be16(fichier + q + 8 + 4 * i);
                if (a < API_WORDS) api[a] = be16(fichier + q + 10 + 4 * i);
            }
        q += l;
    }
    {   /* [2026-10-15] REJEU_TCH_MODE_AND / _OR (hexa) : forcer des bits de d_tch_mode (NDB mot 3)
         * a chaque tick, apres les ecritures ARM -- p.ex. AND=7fff efface B_MUTE_VOCODEC_UL (phase C1). */
        static int lu; static unsigned m_and = 0xffff, m_or = 0;
        if (!lu) { lu = 1; const char *e;
                   if ((e = getenv("REJEU_TCH_MODE_AND"))) m_and = strtoul(e, NULL, 16);
                   if ((e = getenv("REJEU_TCH_MODE_OR")))  m_or  = strtoul(e, NULL, 16);
                   if (m_and != 0xffff || m_or) printf("d_tch_mode force a chaque tick : AND %04x OR %04x\n",
                    ↪  m_and, m_or); }
        if (m_and != 0xffff || m_or) api[NDB + 3] = (uint16_t)((api[NDB + 3] & m_and) | m_or);
    }
    {   /* [2026-10-15] REJEU_W16_BIT14=1 : quand l'ARM pose B_TASK_ABORT (bit 15 de d_ctrl_system,
         * mot 16 de la page W), ajouter le bit 14 -- la ROM (PROM0 0xa569) ne lance alors la tache 43
         * (decodeur de parole) que si ce bit l'accompagne. */
        static int on = -1; static unsigned long n;
        if (on < 0) on = getenv("REJEU_W16_BIT14") != NULL;
        for (int pg = 0; on && pg < 2; pg++) {
            uint16_t *w16 = &api[W_PAGE(pg) + 16];
            if (!(*w16 & 0x4000)) { *w16 |= 0x4000; if (n++ < 3) printf("tick=%u : d_ctrl_system page %d |=
             ↪  0x4000\n", t->tick, pg); }
        }
    }
    /* [2026-10-15] REJEU_MISC_TASK=1 : B_MISC_TASK (bit 3) dans d_dsp_page, comme la pile TI --
     * la ROM ne lance la tache 34 (commandes audio -> tache 64 -> decodeur de parole 0xc4f7)
     * que si ce bit est pose (PROM0 0xb0f9, 0x3fb0 = copie de d_dsp_page). */
    if (getenv("REJEU_MISC_TASK") && (api[NDB + 0] & 0x0002)) api[NDB + 0] |= 0x0008;
    /* [2026-10-22] REJEU_SYNC_DL=1 : la L1 TI pose B_SYNC_TCH_UL (bit 8 de d_ctrl_tch, W mot 9) au premier
     * TCH mais jamais le bit 9 (B_SYNC_AMR chez TI, B_SYNC_TCH_DL pour la ROM et osmocom dsp.c) : le poser. */
    /* [2026-10-22 21:40] REJEU_SYNC_DL=2 : comme montant.c, aussi sur la premiere page a tache TCH (task_d
     ↪  13) :
     * la L1 TI de Magnetite pose d_ctrl_tch=0x0091, sans le bit 8. */
    if (getenv("REJEU_SYNC_DL")) {
        static bool fait;
        int mode2 = atoi(getenv("REJEU_SYNC_DL")) == 2;
        for (int pg = 0; pg < 2; pg++) {
            uint16_t *c = &api[W_PAGE(pg) + 9];
            bool tch = (api[W_PAGE(pg) + 0] & 0x7fff) == 13;
            if (!(*c & 0x0200) && ((*c & 0x0100) || (mode2 && tch && !fait))) {
                *c |= 0x0200; fait = true;
                printf("tick=%u : d_ctrl_tch page %d |= B_SYNC_TCH_DL (%04x)\n", t->tick, pg, *c);
```

```
        }
    }
}
{   /* [2026-10-15] REJEU_DU1_VERS_DU0=1 : la trame de parole que le firmware pose dans a_du_1
     * (prim_tch.c : tch_sub ? a_du_0 : a_du_1) est recopiee dans a_du_0 -- la ROM relit a_du_0
     * a PC 0x909c une fois par bloc, jamais a_du_1 au-dela de l'aiguillage (phase C1). */
    static int on = -1; static unsigned long n;
    if (on < 0) on = getenv("REJEU_DU1_VERS_DU0") != NULL;
    uint16_t *du1 = &api[NDB + 0x134 / 2], *du0 = &api[NDB + 0x2A0 / 2];
    if (on && (du1[0] & 0x8000)) {
        memcpy(du0, du1, 22 * sizeof(uint16_t));
        du1[0] &= 0x7fff;
        if (n++ < 3) printf("tick=%u : a_du_1 -> a_du_0 (en-tete %04x)\n", t->tick, du0[0]);
    }
}
{   /* [2026-10-15] REJEU_DD0=1 : chaque trame de parole descendante (a_dd_0, B_BLUD, mots 3..19,
     * format TI) avec son tick -- reference libgsm pour juger la sortie du vocodeur ROM. */
    static int on = -1; static uint16_t prec[22];
    if (on < 0) on = getenv("REJEU_DD0") != NULL;
    uint16_t *dd0 = &api[NDB + 0x238 / 2];
    if (on && (dd0[0] & 0x8000) && memcmp(dd0 + 3, prec + 3, 17 * 2)) {
        printf("DD0 tick=%u hdr=%04x :", t->tick, dd0[0]);
        for (int i = 0; i < 17; i++) printf(" %04x", dd0[3 + i]);
        printf("\n");
        memcpy(prec, dd0, sizeof prec);
    }
}
{   /* [2026-10-15] REJEU_DU1=1 : chaque trame de parole posee par l'ARM (a_du_1 avec B_BLUD,
     * format TI, mots 3.. octet fort d'abord), avec le fn de la page W -- reference pour
     * comparer hors ligne au codage de la ROM (verif_codage.py). */
    static int on = -1; static uint16_t prec[22];
    if (on < 0) on = getenv("REJEU_DU1") != NULL;
    uint16_t *du1 = &api[NDB + 0x134 / 2];
    if (on && (du1[0] & 0x8000) && memcmp(du1 + 3, prec + 3, 17 * 2)) {
        uint16_t v = api[NDB + 0]; unsigned pg = (v & 1u) ? 1u : 0u; const uint16_t *w = &api[W_PAGE(pg)];
        unsigned t2 = w[12] & 0x1f, t3 = (w[12] >> 5) & 0x3f, t1 = w[13] & 0x7ff;
        uint32_t f = (1326u * t1 + 51u * ((t3 + 26u - t2) % 26u) + t3) % 2715648u;
        printf("DU1 tick=%u fn=%u :", t->tick, f);
        for (int i = 0; i < 17; i++) printf(" %04x", du1[3 + i]);
        printf("\n");
        memcpy(prec, du1, sizeof prec);
    }
}
/* phase A : comme jouer_trame(phase 1) */
calypso_dma_tick(dsp);
reveil();
if ((dsp->imr & (1u << C54X_IT_TPU_FRAME_BIT)) && (t->drap & 1))
    {   /* fn GSM de la page W que la ROM va lire a cette IT (comme pont.c tx_tsp_dater) */
        uint16_t v = api[NDB + 0]; unsigned pg = (v & 1u) ? 1u : 0u; const uint16_t *w = &api[W_PAGE(pg)];
        unsigned t2 = w[12] & 0x1f, t3 = (w[12] >> 5) & 0x3f, t1 = w[13] & 0x7ff;
        g_page_it_fn = (1326u * t1 + 51u * ((t3 + 26u - t2) % 26u) + t3) % 2715648u;
        g_page_it_pg = pg; g_page_it_td = w[0]; g_page_it_tu = w[2]; g_page_it_tra = w[7];
        g_page_it_ctrl = w[9];   /* d_ctrl_tch (W 0x0809 / 0x081d) : phase C1 du 2026-10-15 */
        g_it_tch_mode = api[NDB + 3];          /* d_tch_mode (NDB mot 3) : B_PLAY_UL, mutes */
        g_it_du1 = api[NDB + 0x134 / 2];       /* a_du_1[0] : B_BLUD = trame de parole de l'ARM */
    }
    c54x_interrupt_ex(dsp, C54X_IT_TPU_FRAME_VEC, C54X_IT_TPU_FRAME_BIT);
long fait = 0;
if (!dsp->idle) fait = courir(budget / 8);
while (!dsp->idle && fait < budget / 2) fait += courir(256);
tsp_phase(t->tick, "fin de phase A", -1);
/* ecritures ARM, fenetre 1, puis livraisons I/Q dans l'ordre */
for (size_t q = t->debut; q < t->fin;) {
    uint8_t k = fichier[q];
    size_t l = k == 'T' ? 10 : k == 'A' ? 8 + 4 * (size_t)be16(fichier + q + 6)
             : k == 'B' ? 15 + 2 * (size_t)be16(fichier + q + 13) : 0;
```

```
            if (!l) break;
            if (k == 'A' && fichier[q + 5] == 1) {
                for (unsigned i = 0, n = be16(fichier + q + 6); i < n; i++) {
                    unsigned a = be16(fichier + q + 8 + 4 * i);
                    if (a < API_WORDS) api[a] = be16(fichier + q + 10 + 4 * i);
                }
                if (getenv("REJEU_W16_BIT14")) { api[W_PAGE(0) + 16] |= 0x4000; api[W_PAGE(1) + 16] |= 0x4000; }
              if (getenv("REJEU_MISC_TASK") && (api[NDB + 0] & 0x0002)) api[NDB + 0] |= 0x0008;   /* B_MISC_TASK */
            }
            if (k == 'B') {
                int n = be16(fichier + q + 13);
                static int16_t iq[1024];
                size_t qb = q;
                /* [2026-10-22] REJEU_IQ_DECAL=n : sur les ticks de tache TCH (td=0x000d), livrer les I/Q
                 * enregistres au tick +n (fenetre 0) -- quel calage fait lire a la ROM l'UA de la FACCH ? */
                {   static int dec = 9999;
                    if (dec == 9999) { const char *e = getenv("REJEU_IQ_DECAL"); dec = e ? atoi(e) : 0; }
                    if (dec && fichier[q + 5] == 0 && g_page_it_td == 0x000d && it + dec >= 0 && it + dec < nt) {
                        Tick *u = &ticks[it + dec];
                        for (size_t r = u->debut; r < u->fin;) {
                            uint8_t kk = fichier[r];
                            size_t ll = kk == 'T' ? 10 : kk == 'A' ? 8 + 4 * (size_t)be16(fichier + r + 6)
                                      : kk == 'B' ? 15 + 2 * (size_t)be16(fichier + r + 13) : 0;
                            if (!ll) break;
                            if (kk == 'B' && fichier[r + 5] == 0) { qb = r; break; }
                            r += ll;
                        }
                    }
                }
                n = be16(fichier + qb + 13);
                memcpy(iq, fichier + qb + 15, 2 * (size_t)(n > 1024 ? 1024 : n));
                calypso_bsp_rx_burst(fichier[q + 5], be32(fichier + qb + 6), iq, n);
                if (fichier[q + 5] == 0) dernier_livre = be32(fichier + q + 6);
                if (fichier[q + 5] == 0 && getenv("REJEU_SONDE")) {
                    int nwin = be16(fichier + q + 11) / 2, marge = nwin >= 190 ? 21 : nwin >= 150 ? 3 : 0;
                    memo_burst(be32(fichier + q + 6), iq, n, marge);
                }
                /* [2026-10-04] REJEU_DL=1 : chaque burst descendant livre (fenetre 0), avec le fn que le
                 * pont lui a donne et le fn de la page W (a_a5fn) lu a l'IT de ce tick : les deux doivent
                 * etre egaux pour que la ROM dechiffre au bon fn. Bits demodules a dechiffrer/decoder
                 * hors ligne (libosmocoding). */
                if (fichier[q + 5] == 0 && getenv("REJEU_DL")) {
                    int nwin = be16(fichier + q + 11) / 2, marge = nwin >= 190 ? 21 : nwin >= 150 ? 3 : 0;
                    uint8_t b[148]; int pol; int sc = demod_burst(iq, n, marge, b, &pol);
                    if (sc >= 0) {
                        char s[149]; for (int i = 0; i < 148; i++) s[i] = '0' + b[i]; s[148] = 0;
                        printf("DL tick=%u fn=%u fnW=%u td=%04x nwin=%d tsc=%d/26 pol=%d : %s\n", t->tick,
↪ be32(fichier + q + 6),
                               g_page_it_fn, g_page_it_td, nwin, sc, pol, s);
                    }
                }
            }
            q += l;
        }
        /* phase B : reste du budget, puis la pompe DMA de pont.c */
        reveil();
        tsp_phase(t->tick, "apres les livraisons I/Q (avant phase B)", -1);
        if (!dsp->idle && budget - fait > 0) courir(budget - fait);
        tsp_phase(t->tick, "fin de phase B", -1);
        for (int k = 0; k < 40 && dsp->running; k++) {
            if (!calypso_rhea_dma_pump(dsp)) break;
            if (dsp->idle && (dsp->ifr & dsp->imr) && !(dsp->st1 & 0x800)) dsp->idle = false;
            if (!dsp->idle) courir(budget / 4);
            tsp_phase(t->tick, "pompe DMA", k);
        }
        /* [2026-10-15] port voix (CALYPSO_VSP=1, avec CALYPSO_VSP_SHM=0 hors banc) */
```

```
        if (dsp->running) calypso_vsp_trame(dsp, vsp_courir, g_page_it_fn);
        {   /* [2026-10-04] le burst montant final de la ROM (script TSP, src/tsp_tx.c), REJEU_TX=1 : chaque burst
         ↪  */
            struct tsp_tx_burst tb; bool nouveau = false;
            if (tsp_tx_lire(dsp->data, &tb, &nouveau) && nouveau) {
                tx_n++; if (tb.type == TSP_TX_RACH) tx_rach++; else if (tb.type == TSP_TX_NB) tx_nb++; else
↪  tx_inc++;
                snprintf(tx_dernier, sizeof tx_dernier, "%s%s", tb.type == TSP_TX_RACH ? "RACH" : tb.type ==
↪  TSP_TX_NB ? "NB" : "?",
                         tb.type == TSP_TX_NB ? (tb.bits[60] || tb.bits[87] ? "/vol" : "/parole") : "");
                /* fn GSM de la page W designee par d_dsp_page (a_a5fn, voir pont.c tx_tsp_dater) */
                uint32_t fn_gsm = g_page_it_fn;
                tx_hist26[fn_gsm % 26]++;
                if (getenv("REJEU_TX")) {   /* [2026-10-15] Kc/algo courants, pour dechiffrer hors ligne */
                    static char kc_prec[40];
                    char kcs[40]; uint16_t *akc = &api[NDB + 0x2CE / 2];
                    snprintf(kcs, sizeof kcs, "%d %04x%04x%04x%04x", api[NDB + 0x1CE / 2], akc[3], akc[2], akc[1],
↪  akc[0]);
                  if (strcmp(kcs, kc_prec)) { printf("KC algo,kc(a_kc[3..0])= %s\n", kcs); strcpy(kc_prec, kcs); }
                }
                if (getenv("REJEU_TX") || tx_n <= 12) {
                    char bits[149]; for (int i = 0; i < 148; i++) bits[i] = '0' + tb.bits[i]; bits[148] = 0;
                    printf("tick=%u fn=%u (a_a5fn page %u a l'IT, taches d/u/ra %04x/%04x/%04x) TX %s tsc=%d
↪  fn%%26=%u fn%%51=%u hl=%d hu=%d : %s\n", t->tick, fn_gsm, g_page_it_pg,
                           g_page_it_td, g_page_it_tu, g_page_it_tra, tb.type == TSP_TX_RACH ? "RACH" : tb.type ==
↪  TSP_TX_NB ? "NB" : "?", tb.tsc, fn_gsm % 26, fn_gsm % 51,
                           tb.bits[60], tb.bits[87], bits);
                }
            }
        }
        {   /* [2026-10-15] REJEU_CTRL=1 : par tick, ce que la ROM lit a l'IT (taches d/u, d_ctrl_tch de la
             * page W) et ce qu'elle emet a la fin du tick -- pour savoir si la parole montante manque
             * faute de d_ctrl_tch = TCH_F/FS (0x0011) ou pour une autre raison (phase C1). */
            static int ctrl_on = -1; static unsigned long tx_avant;
            if (ctrl_on < 0) ctrl_on = getenv("REJEU_CTRL") != NULL;
            char scr[64] = "-";
            if (ctrl_on && (g_page_it_tu || g_page_it_td)) {   /* tout script TSP (0x1c0a + 16 mots 0x04/0x06) */
                int k = 0; scr[0] = 0;
                for (unsigned a = 0x0080; a + 18 < 0x8000 && k < 4; a++) {
                    if (dsp->data[a] != 0x1c0a) continue;
                    int ok = 1; for (int i = 0; i < 16; i++) { uint16_t v = dsp->data[a + 2 + i] & 0x3e; if (v !=
                     ↪  0x06 && v != 0x04) { ok = 0; break; } }
                   if (ok) { k += snprintf(scr + strlen(scr), sizeof scr - strlen(scr), "%s%04x", k ? "," : "", a)
                     ↪  > 0; }
                }
                if (!scr[0]) strcpy(scr, "-");
            }
            if (ctrl_on && (g_page_it_tu || g_page_it_td))
                printf("SCR %s ", scr), printf("CTRL tick=%u fn=%u fn%%26=%u pg=%u td=%04x tu=%04x ctrl_tch=%04x
↪  tch_mode=%04x du1=%04x du1_fin=%04x tx=%s\n", t->tick, g_page_it_fn,
                       g_page_it_fn % 26, g_page_it_pg, g_page_it_td, g_page_it_tu, g_page_it_ctrl, g_it_tch_mode,
↪  g_it_du1,
                       api[NDB + 0x134 / 2],
                       tx_n == tx_avant ? "-" : tx_dernier);
            tx_avant = tx_n;
        }
        uint16_t *cd = &api[NDB + 0x1FC / 2], *fd = &api[NDB + 0x21A / 2];
        uint16_t *dd = &api[NDB + 0x238 / 2];   /* a_dd_0 : parole descendante */
        {   /* [2026-10-04] REJEU_BLUD=1 : qui efface B_BLUD des tampons montants (a_cu, a_fu, a_du) ?
             * En fin de tick (apres la ROM) ; les poses sont des ecritures ARM enregistrees. */
            static int blud_on = -1; static uint16_t blud_prec[3];
            if (blud_on < 0) blud_on = getenv("REJEU_BLUD") != NULL;
            if (blud_on) {
                static const unsigned off[3] = { NDB_A_CU, NDB_A_FU, NDB_A_DU_1 };
                static const char *nom[3] = { "a_cu", "a_fu", "a_du" };
                for (int i = 0; i < 3; i++) {
```

```
                uint16_t v = api[(API_NDB + off[i]) / 2] & B_BLUD;
                if (v != blud_prec[i]) { printf("tick=%u BLUD %s %s (fin de tick)\n", t->tick, nom[i], v ?
                 ↪  "pose" : "EFFACE"); blud_prec[i] = v; }
            }
        }
    }
    {   static int dit;
        if (!dsp->idle && dit < 5) { dit++; printf("tick=%u : DSP PAS A L'IDLE en fin de tick, pc=%04x
         ↪  sp=%04x\n", t->tick, dsp->pc & 0xffff, dsp->sp); } }
    {   /* REJEU_DUMP=t1-t2,dossier : memoire de donnees du DSP en fin de tick */
        const char *e = getenv("REJEU_DUMP");
        unsigned t1, t2; char dos[512];
        if (e && sscanf(e, "%u-%u,%511s", &t1, &t2, dos) == 3 && t->tick >= t1 && t->tick <= t2) {
            char nm[600]; snprintf(nm, sizeof nm, "%s/mem_%u.bin", dos, t->tick);
            FILE *fm = fopen(nm, "wb"); if (fm) { fwrite(dsp->data, 2, C54X_DATA_SIZE, fm); fclose(fm); }
        }
    }
    if (getenv("REJEU_SONDE") && dernier_livre != 0xffffffffu && dernier_livre % 104 == 12) {
        printf("tick=%u fn=%u (4e burst SACCH TS2) : a_cd0=%04x %s err=%u\n", t->tick, dernier_livre, cd[0],
               (cd[0] & 0x8000) ? ((cd[0] & 0x40) ? "FIRE KO" : "ok") : "pas de BLUD", cd[2]);
        sonde_bloc(dernier_livre);
    }
    if (cd[0] != prec_cd && (cd[0] & 0x8000)) {
        bool ko = cd[0] & (1u << 6);
        ko ? sacch_ko++ : sacch_ok++;
        printf("tick=%u SACCH a_cd0=%04x %s err=%u L2=%02x %02x %02x %02x\n", t->tick, cd[0],
               ko ? "FIRE KO" : "ok", cd[2], cd[3] & 0xff, cd[3] >> 8, cd[4] & 0xff, cd[4] >> 8);

    }
    if (fd[0] != prec_fd && (fd[0] & 0x8000))
        printf("tick=%u FACCH a_fd0=%04x %s L2=%02x %02x %02x\n", t->tick, fd[0],
               (fd[0] & (1u << 6)) ? "FIRE KO" : "ok", fd[3] & 0xff, fd[3] >> 8, fd[4] & 0xff);
    /* [2026-09-30] a_dd_0 : B_BLUD (bit 15), B_BFI (bit 2), mot 2 = erreurs
     * rapportees par la ROM. Une ligne par trame de parole (REJEU_PAROLE=1)
     * et un bilan ; sert a etudier le B_BFI hors banc. */
    if (dd[0] != prec_dd && (dd[0] & 0x8000)) {
        parole_n++; if (dd[0] & 0x4) parole_bfi++; parole_err += dd[2];
        if (getenv("REJEU_PAROLE"))
            printf("tick=%u PAROLE a_dd0=%04x BFI=%d err=%u\n", t->tick, dd[0], (dd[0] & 0x4) ? 1 : 0, dd[2]);
    }
    prec_cd = cd[0]; prec_fd = fd[0]; prec_dd = dd[0];
}
printf("SACCH : %d bonnes, %d Fire KO\n", sacch_ok, sacch_ko);
printf("TX (script TSP) : %lu bursts, RACH %lu, NB %lu, inconnus %lu\n", tx_n, tx_rach, tx_nb, tx_inc);
printf("TX par fn%%26 (a_a5fn) :"); for (int i = 0; i < 26; i++) printf(" %d:%lu", i, tx_hist26[i]);
 ↪  printf("\n");
if (parole_n) printf("PAROLE : %lu trames, %lu BFI, %.1f erreurs/trame\n", parole_n, parole_bfi,
 ↪  (double)parole_err / parole_n);
return 0;
}
```

3.42 /opt/GSM/c54x_exe/tools/sacch_tf_decode.c

4059 octets, 109 lignes → 109 lignes

```
/* sacch_tf_decode.c - decode hors DSP la SACCH/TF jouee par le BSP.
 *
 * Lit /dev/shm/calypso_sacch_tf.bin (calypso_bsp.c, sonde [sacch_tf] :
 * enregistrements de 156 octets = tick BE32, fn BE32, 148 bits 0/1), regroupe
 * les bursts en blocs selon 45.002 (TCH/F, bloc du TN a fn%104 = 12 + 26*TN/2
 * modulo 104, puis +26, +52, +78) et les passe a gsm0503_xcch_decode, en clair
 * et dechiffres avec le Kc de /dev/shm/calypso_kc_l1.
 *
 * Si la SACCH decode ici et pas dans la ROM, le defaut est dans le chemin
```

```
 * BSP -> ROM ; si elle ne decode pas ici non plus, dans ce que le BSP recoit.
 *
 * ./sacch_tf_decode [fichier] [TN=2] [Kc en hexa, A5/1 ; defaut : calypso_kc_l1]
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <osmocom/core/bits.h>
#include <osmocom/gsm/a5.h>
#include <osmocom/coding/gsm0503_coding.h>

#define MAXB 1024
static struct { uint32_t tick, fn; uint8_t b[148]; } r[MAXB];

static int kc_lire(uint8_t *algo, uint8_t kc[8])
{
    FILE *f = fopen("/dev/shm/calypso_kc_l1", "rb");
    uint8_t b[32];
    if (!f) return -1;
    size_t n = fread(b, 1, sizeof b, f);
    fclose(f);
    if (n < 14) return -1;
    *algo = b[4];
    memcpy(kc, b + 6, 8);
    return (*algo >= 1 && *algo <= 3) ? 0 : -1;
}

static int trouver(int n, uint32_t fn)
{
    for (int i = 0; i < n; i++) if (r[i].fn == fn) return i;
    return -1;
}

int main(int argc, char **argv)
{
    const char *nom = argc > 1 ? argv[1] : "/dev/shm/calypso_sacch_tf.bin";
    int tn = argc > 2 ? atoi(argv[2]) : 2;
    FILE *f = fopen(nom, "rb");
    if (!f) { perror(nom); return 1; }
    int n = 0;
    uint8_t rec[156];
    while (n < MAXB && fread(rec, 1, 156, f) == 156) {
        r[n].tick = (uint32_t)rec[0] << 24 | rec[1] << 16 | rec[2] << 8 | rec[3];
        r[n].fn = (uint32_t)rec[4] << 24 | rec[5] << 16 | rec[6] << 8 | rec[7];
        memcpy(r[n].b, rec + 8, 148);
        n++;
    }
    fclose(f);
    uint8_t algo = 0, kc[8] = { 0 };
    int a_kc = kc_lire(&algo, kc) == 0;
    if (argc > 3 && strlen(argv[3]) == 16) {
        for (int i = 0; i < 8; i++) sscanf(argv[3] + 2 * i, "%2hhx", &kc[i]);
        algo = 1; a_kc = 1;
    }
    printf("%d bursts, Kc %s", n, a_kc ? "" : "absent");
    if (a_kc) printf("A5/%u %02x%02x%02x%02x%02x%02x%02x%02x", algo, kc[0], kc[1], kc[2], kc[3], kc[4], kc[5],
     ↪  kc[6], kc[7]);
    printf("\n");

    unsigned debut = (12 + 26 * (tn / 2)) % 104;   /* TN2 : 38 */
    int ok[2] = { 0, 0 }, vus = 0;
    for (int i = 0; i < n; i++) {
        if (r[i].fn % 104 != debut) continue;
        int idx[4];
        int complet = 1;
```

```
        for (int k = 0; k < 4; k++) {
            idx[k] = trouver(n, r[i].fn + 26 * k);
            if (idx[k] < 0) complet = 0;
        }
        if (!complet) continue;
        vus++;
        printf("bloc fn=%u tick=%u :", r[i].fn, r[i].tick);
        for (int chiffre = 0; chiffre < 2; chiffre++) {
            if (chiffre && !a_kc) break;
            sbit_t sb[464];
            for (int k = 0; k < 4; k++) {
                uint8_t b[148];
                memcpy(b, r[idx[k]].b, 148);
                if (chiffre) {
                    ubit_t dl[114], ul[114];
                    osmo_a5(algo, kc, r[idx[k]].fn, dl, ul);
                    for (int j = 0; j < 57; j++) { b[3 + j] ^= dl[j]; b[88 + j] ^= dl[57 + j]; }
                }
                for (int j = 0; j < 58; j++) {
                    sb[k * 116 + j] = b[3 + j] ? -127 : 127;
                    sb[k * 116 + 58 + j] = b[87 + j] ? -127 : 127;
                }
            }
            uint8_t l2[23];
            int nerr = 0, nbits = 0;
            int rc = gsm0503_xcch_decode(l2, sb, &nerr, &nbits);
            if (rc == 0) ok[chiffre]++;
            printf("  %s rc=%d err=%d/%d L2=%02x %02x %02x %02x %02x %02x", chiffre ? "dechiffre" : "tel-quel",
                   rc, nerr, nbits, l2[0], l2[1], l2[2], l2[3], l2[4], l2[5]);
        }
        printf("\n");
    }
    printf("blocs complets=%d  bons tel-quel=%d  bons dechiffres=%d\n", vus, ok[0], ok[1]);
    return 0;
}
```

3.43 /opt/GSM/c54x_exe/tools/cch_ref/cch_interleave_ref.py

7736 octets, 171 lignes → 170 lignes (1 groupes compactés)

```
#!/usr/bin/env python3
# ================================================================
# GSM 05.03 xCCH block interleaving — REFERENCE
# Applies to BCCH / CCCH (PCH/AGCH) / SDCCH / SACCH.
# One 456-bit coded block -> 4 normal bursts of 114 data bits each,
# NO cross-block overlap (block-rectangular, unlike the 8-burst TCH).
#
# Byte-for-byte identical to libosmocore:
#     gsm0503_xcch_interleave() / gsm0503_xcch_deinterleave()
#         B = k & 3
#         j = 2*((49*k) % 57) + ((k % 8) >> 2)
#         iB[B*114 + j] = cB[k]        (deinterleave: cB[k] = iB[B*114+j])
#
# Burst data layout for j:
#     j =  0..56  -> first  57-bit half  (before the 26-bit midamble)
#     j = 57..113 -> second 57-bit half  (after  the midamble)
#   The 2 stealing flags are NOT part of these 114 bits.
# ================================================================
import csv
import random
from collections import Counter

N = 456
```

```
def fwd_map():
    """k -> (k, B, j, iB_index)."""
    rows = []
    for k in range(N):
        B = k & 3                                   # burst 0..3  (k mod 4)
        j = 2 * ((49 * k) % 57) + ((k % 8) >> 2)    # position 0..113
        rows.append((k, B, j, B * 114 + j))
    return rows

# ---- verify the map is a real interleaver (bijection + full coverage) ----
def verify(rows):
    idx = [r[3] for r in rows]
    assert sorted(idx) == list(range(N)), "iB index is NOT a bijection"
    pairs = Counter((r[1], r[2]) for r in rows)
    assert len(pairs) == N and all(v == 1 for v in pairs.values()), "duplicate (B,j)"
    for B in range(4):
        js = sorted(r[2] for r in rows if r[1] == B)
        assert js == list(range(114)), f"burst {B}: j not full 0..113"
    return True

# ---- rate-1/2 K=5 convolutional code (GSM 05.03 4.1.3) --------------------
# G0 = 1 + D^3 + D^4   (0b10011 = 0x13)  -> even coded index  c(2k)
# G1 = 1 + D + D^3 + D^4 (0b11011 = 0x1B) -> odd  coded index  c(2k+1)
def conv_encode(u):
    assert len(u) == 228                # 184 info + 40 Fire parity + 4 tail(=0)
    U = lambda i: u[i] if 0 <= i < 228 else 0
    c = [0] * N
    for k in range(228):
        c[2*k]   = U(k) ^ U(k-3) ^ U(k-4)
        c[2*k+1] = U(k) ^ U(k-1) ^ U(k-3) ^ U(k-4)
    return c

def interleave(c):
    iB = [0] * N
    for (k, B, j, idx) in fwd_map():
        iB[idx] = c[k]
    return iB

def deinterleave(iB):
    c = [0] * N
    for (k, B, j, idx) in fwd_map():
        c[k] = iB[idx]
    return c

# ============================================================================
if __name__ == "__main__":
    rows = fwd_map()
    verify(rows)

    # round-trip proof: u -> encode -> interleave -> deinterleave == c
    random.seed(0)
    u = [random.randint(0, 1) for _ in range(224)] + [0, 0, 0, 0]
    c = conv_encode(u)
    assert deinterleave(interleave(c)) == c
    print("OK  interleaver is a bijection over 0..455")
    print("OK  per-burst j fully covers 0..113")
    print("OK  conv-encode -> interleave -> deinterleave round-trip exact\n")

    # ---- main table: k -> (burst, pos), plus the two bug variants ----
    with open("cch_interleave_ref.csv", "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["k", "lane_G(0=G0/c0,1=G1/c1)", "burst_B", "pos_j",
                    "iB_index", "j_if_LSB_dropped", "j_if_LSB_inverted",
                    "iB_index_if_LSB_inverted"])
        for (k, B, j, idx) in rows:
            base = 2 * ((49 * k) % 57)
```

```
        j_drop = base
        j_inv = base + (1 - ((k % 8) >> 2))
        w.writerow([k, k & 1, B, j, idx, j_drop, j_inv, B * 114 + j_inv])

# ---- inverse table: what a deinterleaver actually indexes ----
inv = [0] * N
for (k, B, j, idx) in rows:
    inv[idx] = k
with open("cch_deinterleave_inverse_ref.csv", "w", newline="") as f:
    w = csv.writer(f)
    w.writerow(["iB_index", "burst_B", "pos_j", "source_coded_k", "lane_G"])
    for idx in range(N):
        k = inv[idx]
        w.writerow([idx, idx // 114, idx % 114, k, k & 1])

# ---- deinterleave with a deliberate bug, for stage isolation ----
def deinterleave_mode(iB, mode):
    c = [0] * N
    for k in range(N):
        B = k & 3
        base = 2 * ((49 * k) % 57)
        if mode == "correct":
            j = base + ((k % 8) >> 2)
        elif mode == "lsb_dropped":     # ((k%8)>>2) forced to 0
            j = base
        elif mode == "lsb_inverted":    # 0<->1 on the LSB term
            j = base + (1 - ((k % 8) >> 2))
        c[k] = iB[B * 114 + j]
    return c

# Known test vector. u is pseudo-random (dense in 1s) on purpose:
# any interleaver error then shows up massively at the cB level.
# Fire is NOT needed here - this isolates interleaver + conv-decoder only.
random.seed(1)
u_tv = [random.randint(0, 1) for _ in range(224)] + [0, 0, 0, 0]
c_tv = conv_encode(u_tv)                # 456 coded bits, correct order
iB_tv = interleave(c_tv)                # 4 x 114, from the CORRECT interleaver
cB_correct = deinterleave_mode(iB_tv, "correct")      # == c_tv (Viterbi(cB)->u)
cB_drop    = deinterleave_mode(iB_tv, "lsb_dropped")
cB_inv     = deinterleave_mode(iB_tv, "lsb_inverted")
assert cB_correct == c_tv

s = lambda bits: "".join(map(str, bits))
with open("cch_testvectors.txt", "w") as f:
    f.write("# GSM 05.03 xCCH stage-isolation test vector\n")
    f.write("# Fixed pseudo-random 228-bit u (u[224:228]=tail=0).\n")
    f.write("# TEST A (interleaver): feed burst0..3 into YOUR deinterleaver,\n")
    f.write("#   compare the 456-bit result against the three candidates below:\n")
    f.write("#     == cB_correct       -> interleaver OK, bug is downstream\n")
    f.write("#     == cB_lsb_dropped   -> you never apply the ((k%8)>>2) term\n")
    f.write("#     == cB_lsb_inverted  -> that term is inverted / off-by-one\n")
    f.write("#     == none of them     -> different bug (burst order? half swap?)\n")
    f.write("# TEST B (Viterbi/lane): feed cB_correct into YOUR Viterbi,\n")
    f.write("#   compare the 228-bit result against u. Mismatch with cB correct\n")
    f.write("#   => c0/c1 swap, wrong polynomials, or tail handling.\n\n")
    f.write("u_228           = " + s(u_tv) + "\n\n")
    f.write("burst{1}_114      = " + s(iB_tv[{2}:{3}]) + "\n")  ×3
{} = {0,0,114} {1,114,228} {2,228,342}
    f.write("burst3_114      = " + s(iB_tv[342:456]) + "\n\n")
    f.write("cB_correct      = " + s(cB_correct) + "\n\n")
    f.write("cB_lsb_dropped  = " + s(cB_drop) + "\n\n")
    f.write("cB_lsb_inverted = " + s(cB_inv) + "\n")

dd = sum(a != b for a, b in zip(cB_correct, cB_drop))
di = sum(a != b for a, b in zip(cB_correct, cB_inv))
print(f"test vector written: cB differs from correct in "
```

```
          f"{dd}/456 (dropped) and {di}/456 (inverted) positions\n")

    # ---- how big / what shape is the LSB bug ----
    diff = [k for (k, B, j, idx) in rows
            if (B * 114 + j) != (B * 114 + 2 * ((49 * k) % 57) + (1 - ((k % 8) >> 2)))]
    print(f"LSB-inverted bug moves {len(diff)}/456 coded bits to a wrong burst slot")
    print("  -> every k is displaced by exactly +/-1 in j (even<->odd swap),")
    print("     within its own burst, so the burst assignment (k&3) stays right.\n")

    print("spot check (k : burst, pos):")
    for k in [0, 1, 2, 3, 4, 5, 6, 7, 8, 57, 114, 228, 455]:
        B = k & 3
        j = 2 * ((49 * k) % 57) + ((k % 8) >> 2)
        print(f"  k={k:3d} -> burst {B}, pos {j:3d}   (lane {'G1/c1' if k & 1 else 'G0/c0'})")
```

4 Logs

4.1 /opt/GSM/c54x_exe/toa.log

7679 octets, 198 lignes → 191 lignes (4 groupes compactés)

```
```
magic 23 reproduction — 2026-10-04T16:50Z — Linux 6.8.0-146-generic x86_64

1. ROM dump -> PROM0.bin
$ sha256sum dsp-rom-3606-dump.txt
e35798f50fd0a29aa61cd53e417c010d7f172582c7bc055211daab8af1e05470 dsp-rom-3606-dump.txt

PROM0.bin: 28672 words (0x7000-0xdfff)
  7cbd: f010 2be4
  84a4: 7713 2a27
  b21d: 7713 00bf
  b2b1: 7713 0097
  b32d: 7713 0030
$ sha256sum PROM0.bin
75ce5375a07eac85ee42ba00af7ca82089d15b5df517710a756e6c26497021d7 PROM0.bin

2. objdump tic54x (binutils 2.21.1)
$ build-tic54x/binutils/objdump --version
GNU objdump (GNU Binutils) 2.21.1
Copyright 2011 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.

full disassembly: PROM0.dis (28665 lines)

expected: 7cbd sub #11236,a | 84a4 stm #10791,ar3 | b21d stm #191,ar3 | b2b1 stm #151,ar3 | b32d stm #48,ar3
    7cbd: f010   sub   #11236,a
    84a4:   7713   stm   #10791,ar3
    b21d: 7713   stm   #191,ar3
    b2b1:   7713   stm   #151,ar3
    b32d: 7713   stm   #48,ar3

SB TOA chain
    7c3d: 7712   stm   #3278,ar2
    81c8:   f272   rptbd   0x81db
    7c49:   7712   stm   #11498,ar2
    7a1c: fe00   retd
    84a1:   f7b6   ssbx   st1,frct
    84ca: f6b6   rsbx   st1,frct
    84dd: e900   ld   #0,b
```
```

```
    7c5f:   7210    mvdm   0x2f06,ar0
    7ca6:   f4bc    rsbx   st0,tc
    7cbc:   4817    ldm    ar7,a
    7e8a:   711f    mvdk   DP+0x1f,0x3fa4
    b1e7:   10f8    ld     *(0x3fa4),a

### FB TOA: 0x793e-0x795a
    793e:   1194    ld     *ar4+,b
    793f:   f762    sfta   b,2
    7940:   44f8    ld     *(0x3fb4),16,a
    7941:   3fb4
    7942:   f061    sub    #3,16,a
    7943:   0003
    7944:   3c84    add    *ar4,16,a
    7945:   f061    sub    #2,16,a
    7946:   0002
    7947:   f48c    mpya   a
    7948:   f500    add    a,b
    7949:   f310    sub    #5,b
    794a:   0005
    794b:   6ff8    sub    *(0xc3d),b,a
    794c:   0c3d
    794d:   0e20
    794e:   fc47    rc     aleq
    794f:   44f8    ld     *(0x3fb3),16,a
    7950:   3fb3
    7951:   f061    sub    #3,16,a
    7952:   0003
    7953:   770e    stm    #48,t
    7954:   0030
    7955:   f48c    mpya   a
    7956:   08f8    sub    *(0xc3e),a
    7957:   0c3e
    7958:   f620    sub    b,a
    7959:   fc47    rc     aleq
    795a:   81f8    stl    b,*(0x8fa)

### DMA programming routine: 0xa5cd-0xa5f6
    a5cd:   ea86    ld     #134,dp
    a5ce:   76f8    st     #-3980,*(0x158)
    a5cf:   0158
    a5d0:   f074
    a5d1:   76f8    st     #29250,*(0x159)
    a5d2:   0159
    a5d3:   7242
    a5d4:   76f8    st     #-3981,*(0x15a)
    a5d5:   015a
    a5d6:   f073
    a5d7:   4811    ldm    ar1,a
    a5d8:   80f8    stl    a,*(0x15b)
    a5d9:   015b
    a5da:   69f8    orm    #4,*(0x3fdc)
    a5db:   3fdc
    a5dc:   0004
    a5dd:   75f8    portw  *(0x3fdc),pa63744
    a5de:   3fdc
    a5df:   f900
    a5e0:   47f8    rpt    *(0xc32)
    a5e1:   0c32
    a5e2:   f495    nop
    a5e3:   4812    ldm    ar2,a
    a5e4:   f010    sub    #2048,a
    a5e5:   0800
    a5e6:   6f56    stl    a,1,DP+0x56
    a5e7:   0c81
    a5e8:   7556    portw  DP+0x56,pa64548
```

```
    a5e9:   fc24
    a5ea:   4813    ldm     ar3,a
    a5eb:   f110    sub     #48,a,b
    a5ec:   0030
    a5ed:   6f56    stl     a,2,DP+0x56
    a5ee:   0c82
    a5ef:   7556    portw   DP+0x56,pa64550
    a5f0:   fc26
    a5f1:   7656    st      #1197,DP+0x56
    a5f2:   04ad
    a5f3:   ff4d    xc      2,beq
    a5f4:   7656    st      #1193,DP+0x56
    a5f5:   04a9
    a5f6:   7556    portw   DP+0x56,pa64552

## 3. layer1 git history
$ git clone -q https://gitea.osmocom.org/phone-side/osmocom-bb

$ git -C osmocom-bb log --format=%h %ad %an %s --date=short -Stoa -= 23 -- src/target/firmware/layer1
8978ec57 2010-05-19 Harald Welte layer1/l1ctl: Split L1CTL_NEW_CCCH_REQ in FBSB_REQ nad SYNC_REQ
5f751d3d 2010-04-09 Harald Welte layer1: move FB and SB detection into prim_fbsb.c
fbe7b94c 2010-02-18 Harald Welte Initial import of OsmocomBB into git repository

$ git -C osmocom-bb log --format=%h %ad %an %s --date=short -Smagic 23 -- src/target/firmware/layer1
8978ec57 2010-05-19 Harald Welte layer1/l1ctl: Split L1CTL_NEW_CCCH_REQ in FBSB_REQ nad SYNC_REQ
5f751d3d 2010-04-09 Harald Welte layer1: move FB and SB detection into prim_fbsb.c
48dfd41e 2010-04-07 Harald Welte layer1: add 'struct mon_state' argument to read_{fb,sb}_result()

### cb71b972: toa -= 23 / fbinfo2cellinfo lines (expect added fbinfo2cellinfo, no removed toa -= 23)
20:@@ -87,7 +87,7 @@ static void dump_mon_state(struct mon_state *fb)
29:@@ -98,14 +98,16 @@ static int l1ctl_fbsb_resp(uint8_t res)
⟨1⟩:@@ -⟨2⟩,⟨3⟩ +⟨4⟩,⟨5⟩ @@ static int l1s_sbdet_resp(__unused uint8_t p1, uint8_t attempt,  ×3
    ⟨⟩ = (48,179,25,181,24) (80,225,25,226,14) (108,259,18,249,13)
134:@@ -279,6 +264,9 @@ static int l1s_sbdet_cmd(__unused uint8_t p1, __unused uint8_t p2,
144:@@ -330,6 +318,41 @@ static int read_fb_result(struct mon_state *st, int attempt)
148:+static void fbinfo2cellinfo(struct l1_cell_info *cinfo,
154:+   last_fb->toa -= 23;
186:@@ -354,6 +377,8 @@ static int l1s_fbdet_cmd(__unused uint8_t p1, __unused uint8_t p2,
⟨1⟩:@@ -⟨2⟩,⟨3⟩ +⟨4⟩,⟨5⟩ @@ static int l1s_fbdet_resp(__unused uint8_t p1, uint8_t attempt,  ×5
    ⟨⟩ = (195,363,16,388,7) (213,385,8,401,22) (236,398,7,428,8) (246,408,6,439,7) (254,423,8,455,15)
267:+               fbinfo2cellinfo(&l1s.serving_cell, last_fb);
271:@@ -458,8 +497,6 @@ static const struct tdma_sched_item fb_sched_set[] = {
280:@@ -467,35 +504,10 @@ static void l1a_fb_compl(__unused enum l1_compl c)
285:-   last_fb->toa -= 23;
313:+   fbinfo2cellinfo(&l1s.serving_cell, last_fb);
319:@@ -513,6 +525,9 @@ void l1s_fbsb_req(uint8_t base_fn, struct l1ctl_fbsb_req *req)

### current constants
38:#define L1_NB_MARGIN_Q       (3 * 4)
39:#define L1_SB_MARGIN_Q       (23 * 4)
40:#define L1_TAIL_DURATION_Q   (3 * 4)
43:#define L1_NB_DURATION_Q (L1_BURST_LENGTH_Q + 2 * L1_NB_MARGIN_Q - L1_TAIL_DURATION_Q)
44:#define L1_SB_DURATION_Q (L1_BURST_LENGTH_Q + 2 * L1_SB_MARGIN_Q - L1_TAIL_DURATION_Q)
54: [L1_RXWIN_SB]   = L1_SB_DURATION_Q,
55: [L1_RXWIN_NB]   = L1_NB_DURATION_Q,
206:    last_fb->toa -= 23;
333:    /* FIXME: where did this magic 23 come from? */
334:    last_fb->toa -= 23;
469:    /* FIXME: where did this magic 23 come from? */
470:    last_fb->toa -= 23;
119:     * is within 0...8. We have to add 75 to get an SB TOA of 4. */
120:    tpu_shift += 75;

## 4. hardware logs: qbits check ((TOA-46) mod 1250)*4
sources: http://lists.osmocom.org/pipermail/baseband-devel/2011-September/002526.html
         https://www.mail-archive.com/baseband-devel@lists.osmocom.org/msg01046.html
```

```
  TOA= ⟨1⟩ printed=⟨2⟩  -46:⟨3⟩ OK   -23:⟨4⟩  ×3
    ⟨⟩ = (9651,3420,3420,3512) (8751,4820,4820,4912) (8755,4836,4836,4928)
  TOA=10003 printed=4828  -46:4828 OK   -23:4920
  TOA=10007 printed=4844  -46:4844 OK   -23:4936
  TOA= 8755 printed=4836  -46:4836 OK   -23:4928
  TOA=   ⟨1⟩ printed=⟨2⟩  -46:⟨3⟩ OK   -23:  ⟨4⟩  ×4
    ⟨⟩ = (29,4932,4932,24) (29,4932,4932,24) (27,4924,4924,16) (27,4924,4924,16)
  TOA=   24 printed=4912  -46:4912 OK   -23:   4
  TOA=   26 printed=4920  -46:4920 OK   -23:  12
  12/12 lines match the -46 rule
  FB mode 0 multiples of 48: 7/7
  FB mode 1 == 3 mod 4:      10/10
  first SB after FB sync: [29, 29, 27, 27, 24, 26]  mean delta -0.25
  148 + 2*23 - 3 = 191 ; 148 + 2*3 - 3 = 151 ; (191-148+3)/2 = 23

# done — 195 lines in toa.log
```