# Xen with Linux For Functional Safety & Security Critical System

feat. Automotive

Paran Lee
p4ranlee@gmail.com
https://www.linkedin.com/in/paran-lee-055159216/

# Introduction

리눅스 커널에 관심이 많아서 시작한

Iamroot 16차 멤버

최근까지 보안 회사에서 커널 모듈 기반 안티 바이러스 개발을 하다가
현재는 5G 코어 망을 만드는 회사 다니는 중

uftrace 프로젝트 OSSCA 2022 멘티, 2023 멘토

프로그래머스 데브코스 시스템 소프트웨어 개발자 과정 리눅스 커널 커리큘럼 멘토

유튜브 채널 How To Be Contributor

# Contents

# Xen Overview
# (Perspective of Linux Usecase)

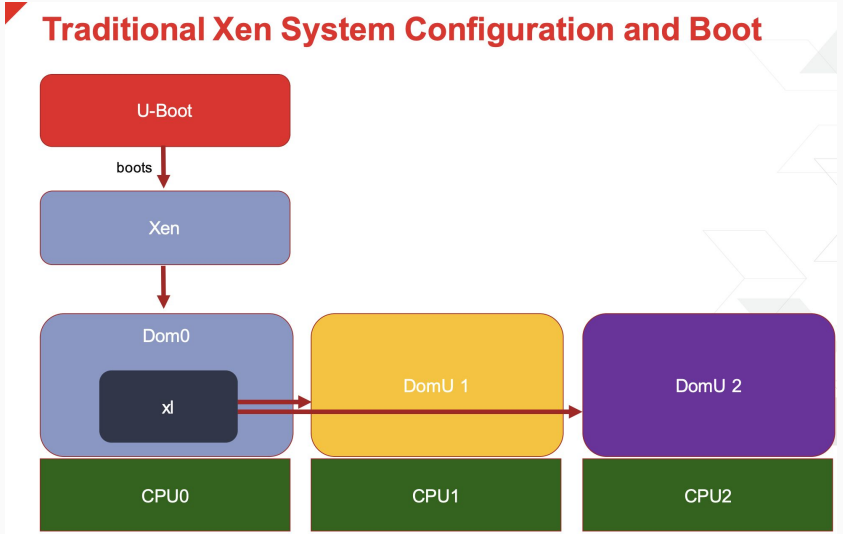
# Xen Overview

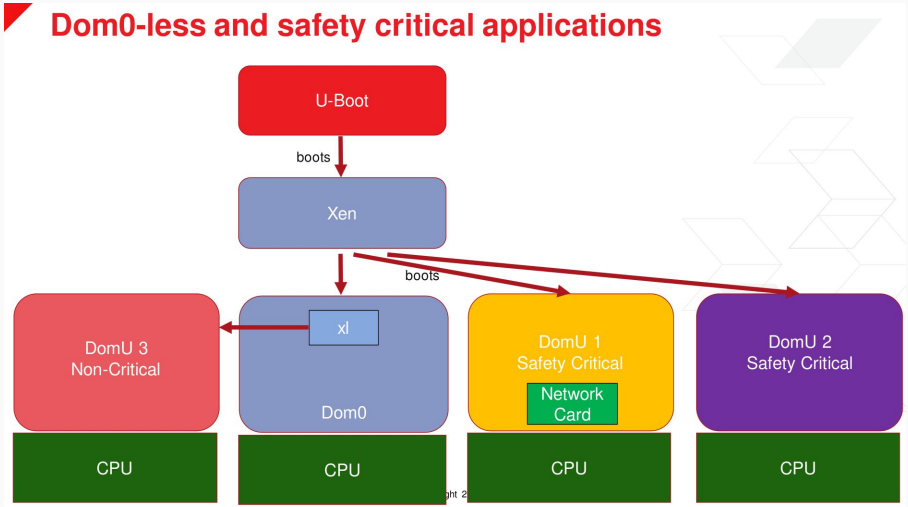
- Type 1 Hypervisor
- Flexible Virtualization Mode
- Driver disaggregation
- VM Isolation
- Dom_U Scalability
- Support A lot of Hardware
- Mature since 2003

# Xen Overview

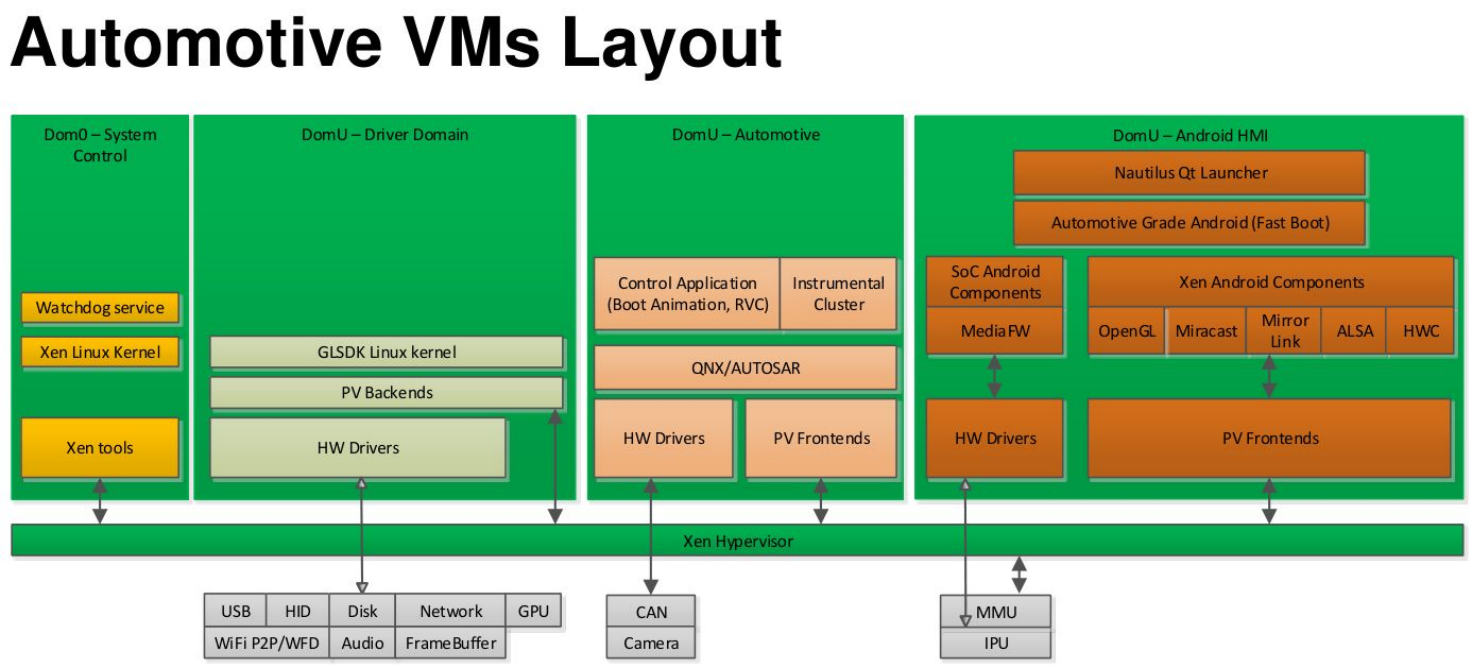
## Safety Certifications

- Dom0 doesn't have to be Linux but it typically is non-Linux Dom0



# Xen Use Case


Automotive VMs Layout
Dom0 – System Control
Watchdog service
Xen Linux Kernel
Xen tools
DomU – Driver Domain
GLSDK Linux kernel
PV Backends
HW Drivers
DomU – Automotive
Control Application (Boot Animation, RVC)
Instrumental Cluster
QNX/AUTOSAR
HW Drivers
PV Frontends
DomU – Android HMI
Nautilus Qt Launcher
Automotive Grade Android (Fast Boot)
SoC Android Components
MediaFW
Xen Android Components
OpenGL
Miracast
Mirror Link
ALSA
HWC
HW Drivers
PV Frontends
Xen Hypervisor
USB
HID
Disk
Network
GPU
WiFi P2P/WFD
Audio
FrameBuffer
CAN
Camera
MMU
IPU

# Functional Safety(FuSa) System

# 표준 가이드라인(ISO 26262)

소프트웨어 품질과 안전성: ISO 26262은 안전한 시스템 개발을 강조하며, Xen을 사용하는 경우 Xen 하이퍼바이저와 관련된 소프트웨어 개발 및 관리 프로세스를 구성해야 합니다. 이 프로세스에는 코드 리뷰, 테스트, 버그 수정 및 문서화가 포함됩니다.

ISO 26262 요구사항 준수: Xen은 자동차 시스템의 일부로 사용될 경우 해당 시스템의 ISO 26262 요구사항을 준수해야 합니다. 이러한 요구사항은 기능적 안전성과 품질 관리를 의미하며, Xen을 통해 제공되는 가상화 환경의 안전성에 관련이 있을 수 있습니다.

가상화의 안전성: Xen을 사용하는 가상화 환경은 안전한 동작을 보장해야 합니다. 이는 가상 머신 (VM)의 격리, 자원 할당 및 관리, 시스템의 부팅 및 종료 과정에서 안전한 프로세스 등을 포함합니다.

# 표준 가이드라인(ISO 26262)

테스트 및 검증: Xen을 사용하는 경우, ISO 26262 요구사항을 충족하는 소프트웨어를 개발하고 검증하는 데 필요한 테스트 및 검증 절차를 수행해야 합니다. 이러한 테스트는 Xen 환경에서 가상화된 자원의 안전한 사용을 확인하기 위한 것일 수 있습니다.

배포 및 업데이트 관리: Xen을 사용하는 경우, 가상화 환경의 업데이트 및 배포는 안전하게 이루어져야 합니다. 시스템의 변경 사항이나 업데이트가 자동차 시스템의 안전성에 미치는 영향을 평가하고 관리해야 합니다.

문서화: ISO 26262는 문서화를 강조하며, Xen과 관련된 소프트웨어의 개발 및 운영 과정을 잘 문서화하여 추적 가능하게 관리해야 합니다.

# Xen FuSa

Code Quality

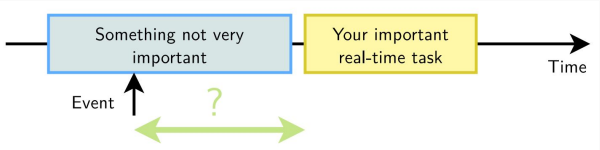
Improve Xen code quality and safety. Implement features to improve real-time and reduce interference. Improve Xen coding style and align it with MISRA-C.

Keywords: MISRA, code quality, static analysis, real-time

# Determinism

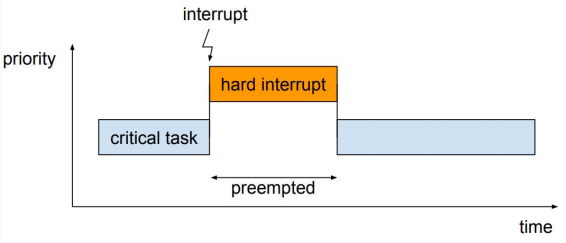
In an Realtime system, the timing for event and data processing must be consistent

- The same input must always yield the same output
- single-task single-core systems -> trivial
- On multi-tasking systems, critical tasks should be deterministic
  - The influence of CPU sharing and external interrupts must be fully predictable



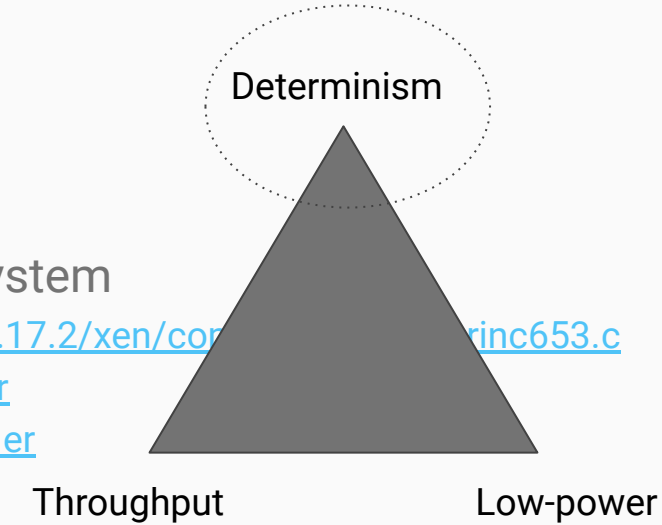

Latencies are the main focus for Realtime Operating Systems

- Time elapsed between an event and the reaction to the event
- The Worst Case Execution Time is very difficult to predict
- We therefore want bounds on the Worst Case Reaction Time



# Hard Real-time Determinism

- Avoid unpredictable effects
- Caches, Hardware Offload are hard to predict
- Avoid sleeping too deep, to wakeup fast
- Make the system fully preemptible
- Try to keep control over every aspect of the system
  - https://github.com/xen-project/xen/blob/RELEASE-4.17.2/xen/co rinc653.c
  - https://wiki.xenproject.org/wiki/ARINC653_Scheduler
  - https://wiki.xenproject.org/wiki/RTDS-Based-Scheduler



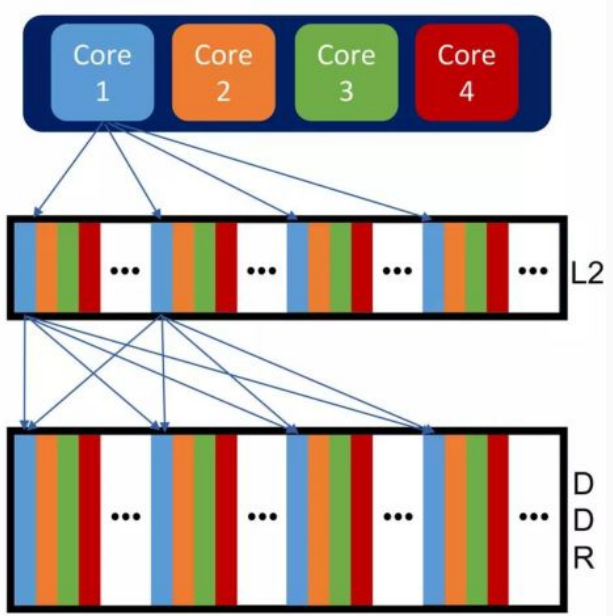
# Hard Real time Isolation Between VMs : Xen Cache Coloring

deterministic IRQ latency

- difficult to pull off as even small spikes lead to failure.
- No matter the activity in other VMs,
  the latency-sensitive app has to continue unaffected.

Xen can fully dedicate physical CPUs to VMs
to minimize latency and interference

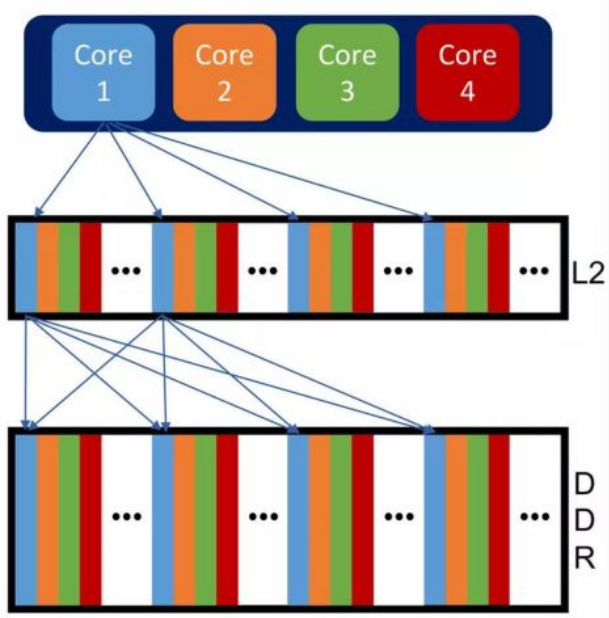
- real-time deadlines can still be missed due to
  the presence of a shared L2 cache across the ARM cores.
- One app on one CPU core can affect the performance of
  another app in a different VM by causing cache interference.



# Hard Real time Isolation Between VMs : Xen Cache Coloring

## Xen Cache Coloring

- CPUs clusters often share L2 cache
- Interference via L2 cache affects performance
  - App0 running on CPU0 can cause cache entries evictions, which affect App1 running on CPU1
  - App1 running on CPU1 could miss a deadline due to App0's behavior
  - It can happen between Apps running on the same OS & between VMs on the same hypervisor
- Hypervisor Solution: Cache Partitioning, AKA **Cache Coloring**
  - Each VM gets its own allocation of cache entries
  - No shared cache entries between VMs
  - Allows real-time apps to run with deterministic IRQ latency



# Xen MISRA C Check instrument

```
make -C xen \
  CPPCHECK=/path/to/cppcheck \
  CPPCHECK_HTMLREPORT=/path/to/cppcheck-htmlreport \
  CPPCHECK_MISRA=y \
  cppcheck-html
```

- https://github.com/xen-project/xen/blob/RELEASE-4.17.2/docs/misra/rules.rst
- https://github.com/xen-project/xen/blob/RELEASE-4.17.2/docs/misra/cppcheck.txt
- https://www.mail-archive.com/search?l=xen-devel%40lists.xenproject.org&q=MISRA+C&submit.x=0&submit.y=0

# Security Critical System

# 표준 가이드라인 몇 가지

STIG(Security Technical Implementation Guide)는 미국 국방부에서 개발한 표준으로, 다양한 정보 시스템 및 플랫폼의 보안 요구사항을 정의합니다.

CIS (Center for Internet Security) Controls: CIS Controls는 네트워크 및 정보 시스템 보안을 강화하기 위한 20가지 핵심 보안 조치를 정의합니다.

FIPS (Federal Information Processing Standards): FIPS는 미국 정부에서 사용하기 위한 정보 처리 및 보안 표준을 정의합니다. 자동차 시스템에서 FIPS를 준수하면 데이터 보호, 암호화 및 안전한 통신을 확보할 수 있습니다.

# 표준 가이드라인 몇 가지

OVA (Open Vulnerability and Assessment System): OVA는 취약성 스캐닝 도구 및 프로세스를 관리하기 위한 표준과 가이드라인을 제공합니다.

자동차 보안에서는 시스템 및 소프트웨어의 취약성을 식별하고 해결하기 위해 OVA를 사용할 수 있습니다.

ISO/SAE 21434: 이 표준은 "로드 차량 사이버 보안 엔지니어링"에 관한 것으로, 자동차의 사이버 보안 요구사항과 절차를 정의합니다.

ISO/SAE 21434는 ISO 26262와 함께 자동차의 안전성과 보안성을 고려하는 데 사용됩니다.

SAE J3061: Society of Automotive Engineers (SAE)에서 개발한 이 표준은 "사이버-물리 시스템 보안 절차 및 요구사항"에 관한 것으로, 자동차와 같은 사이버-물리 시스템의 보안을 다룹니다. 이 표준은 자동차의 사이버 보안을 평가하고 관리하는 데 적용됩니다.

# Secure Boot

You can’t trust your applications if you can’t trust the kernel or the hypervisor or the bootloader or the firmware or the CPU

- 커널을 통해서 어플리케이션의 데이터를 가로채어 동작 방식이나 로그를 바꿀 수 있습니다.

# Secure Boot

Host-based Secure Boot는 호스트 시스템의 Secure Boot 기능을 사용하여 가상 머신을 인증하는 방법입니다. 호스트 시스템이 Secure Boot를 지원하는 경우, KVM은 호스트 시스템의 Secure Boot 기능을 사용하여 가상 머신의 부트로더를 인증할 수 있습니다.

- Guest-based Secure Boot: 가상 머신의 Secure Boot 기능을 사용하여 가상 머신 자체를 인증할 수 있습니다.
- Hypervisor-based security: Xen hypervisor는 hypervisor-based security를 지원하여 하이퍼바이저가 가상 머신에 대한 액세스를 제어할 수 있습니다. hypervisor-based security는 가상 머신이 악성 코드에 감염되었더라도, 하이퍼바이저가 가상 머신을 보호할 수 있도록 도와줍니다.
- Memory protection: Xen hypervisor는 memory protection을 지원하여 가상 머신 간의 메모리 접근을 제어할 수 있습니다. memory protection은 가상 머신이 서로의 메모리에 접근하는 것을 방지하여, 악성 코드가 다른 가상 머신을 공격하는 것을 방지할 수 있습니다.

# VM Isolation

# VM Isolation

가상 머신의 리소스 액세스 제한

- 하이퍼바이저는 가상 머신의 CPU 사용량을 제한하거나, 가상 머신의 메모리 사용량을 제한하거나, 가상 머신의 네트워크 사용량을 제한할 수 있습니다. 이를 통해, 악성 코드가 가상 머신의 전체 시스템을 제어하는 것을 방지할 수 있습니다. 예를 들어, 하이퍼바이저는 가상 머신의 CPU 사용량을 10%로 제한함으로써, 악성 코드가 가상 머신의 CPU를 과도하게 사용하지 못하도록 막을 수 있습니다.

가상 머신의 메모리 액세스 제어

- 하이퍼바이저는 가상 머신의 메모리 액세스를 제어할 수 있습니다. 이를 통해, 악성 코드가 다른 가상 머신의 메모리에 접근하는 것을 방지할 수 있습니다. 예를 들어, 하이퍼바이저는 가상 머신의 메모리를 보호하는 가상 메모리 보호 기능을 제공합니다. 이 기능은 악성 코드가 다른 가상 머신의 메모리에 접근하지 못하도록 막아줍니다.

# VM Isolation

가상 머신의 운영 체제 보호

- 하이퍼바이저는 가상 머신의 운영 체제를 보호하기 위한 다양한 기술을 제공합니다. 예를 들어, 하이퍼바이저는 가상 머신의 운영 체제에 대한 패치를 자동으로 적용하거나, 가상 머신의 운영 체제에 대한 보안 검사를 수행할 수 있습니다. 예를 들어, 하이퍼바이저는 가상 머신의 운영 체제에 대한 패치가 배포되면, 자동으로 해당 패치를 적용합니다. 이를 통해, 악성 코드가 운영 체제의 취약점을 이용하여 가상 머신을 감염시키는 것을 방지할 수 있습니다.

Xen hypervisor의 memory protection

- 가상 머신마다 고유한 메모리 공간을 할당: Xen hypervisor는 가상 머신마다 고유한 메모리 공간을 할당합니다. 이를 통해, 가상 머신은 서로의 메모리 공간에 접근할 수 없습니다.
- 예를 들어, 가상 머신 A와 가상 머신 B가 있다고 가정합니다. 가상 머신 A는 0x00000000~0x0000FFFF까지의 메모리 공간을 할당받고, 가상 머신 B는 0x00010000~0x0001FFFF까지의 메모리 공간을 할당받습니다. 따라서, 가상 머신 A는 가상 머신 B의 메모리 공간에 접근할 수 없습니다.
- 가상 머신의 메모리 접근을 제어: Xen hypervisor는 가상 머신의 메모리 접근을 제어할 수 있습니다. 이를 통해, 악성 코드가 다른 가상 머신의 메모리에 접근하는 것을 방지할 수 있습니다. 예를 들어, Xen hypervisor는 악성 코드가 특정 메모리 영역에 접근하는 것을 제한할 수 있습니다. 예를 들어, 가상 머신 A에 악성 코드가 감염되었다고 가정합니다. 악성 코드는 가상 머신 B의 메모리 공간에 접근하여 데이터를 삭제하려고 합니다. 그러나, Xen hypervisor는 악성 코드가 가상 머신 B의 메모리 공간에 접근하는 것을 제한하여, 데이터가 삭제되지 않도록 보호할 수 있습니다.

# 부록

# QEMU 에서 실습해보기 : 빌드

$ mkdir build_xen_yocto

$ sudo apt -y update && sudo apt -y upgrade && sudo apt -y install curl chrpath diffstat gawk lz4 build-essential python3-dev pzstd zstd

$ sudo update-alternatives --install /usr/bin/python python /usr/bin/python3 10

$ sudo update-alternatives --config python

$ git config --global user.name "..." && git config --global user.email "...@gmail.com"

$ curl -k https://storage.googleapis.com/git-repo-downloads/repo > repo

$ chmod a+x repo

$ ./repo init -u https://github.com/Xilinx/yocto-manifests.git -b rel-v2023.1

$ ./repo sync

$ source setupsdk

# QEMU 에서 실습해보기 : 빌드

$ curl -k https://storage.googleapis.com/git-repo-downloads/repo > repo

$ chmod a+x repo

$ ./repo init -u https://github.com/Xilinx/yocto-manifests.git -b rel-v2023.1

$ ./repo sync

$ source setupsdk

$ vim conf/local.conf

# Edit conf/local.conf and Add the below line

IMAGE_FSTYPES += "cpio.gz"

$ MACHINE=vck190-versal bitbake xen-image-minimal

# QEMU 에서 실습해보기 : 빌드

$ git clone https://gitlab.com/ViryaOS/imagebuilder.git

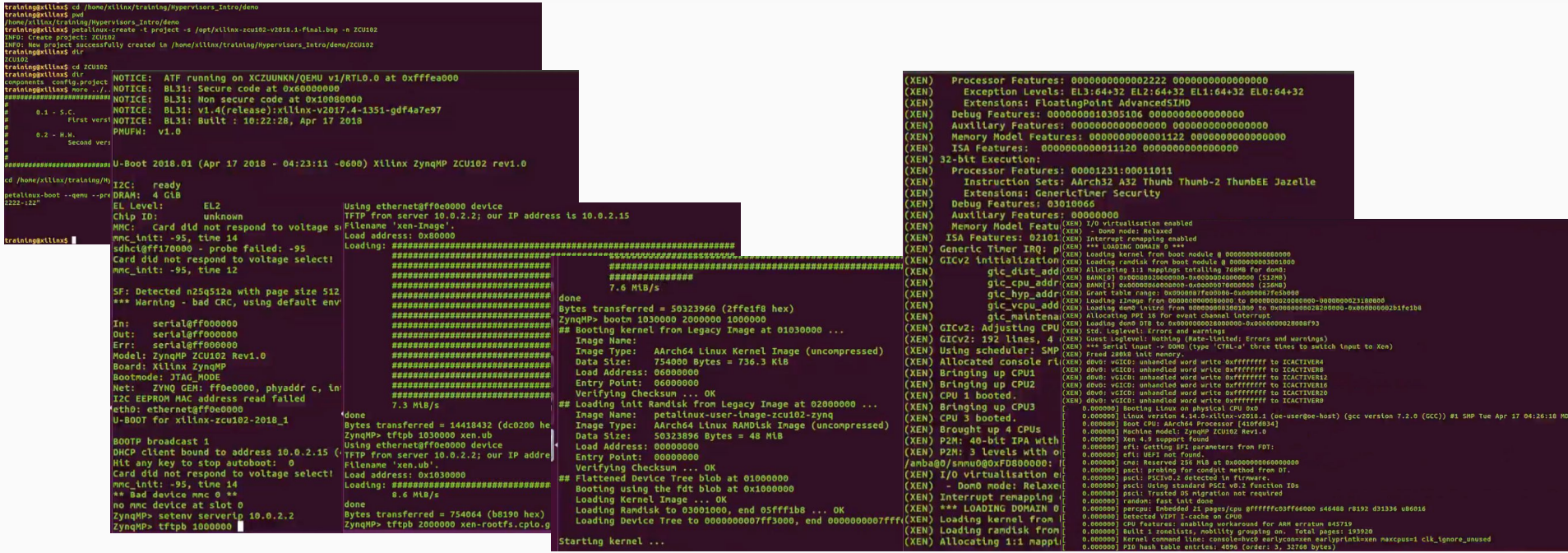
$ vim build/tmp/deploy/images/vck190-versal/config

```
MEMORY_START="0x0"
MEMORY_END="0x80000000"

DEVICE_TREE="system.dtb"
XEN="xen"
DOM0_KERNEL="Image"
DOM0_RAMDISK="xen-image-minimal-vck190-versal.cpio.gz"

NUM_DOMUS=0

UBOOT_SOURCE="boot_xen.source"
UBOOT_SCRIPT="boot_xen.scr"
```

# QEMU 에서 실습해보기 : 빌드

```
$ cd {Yocto_dir}
$ source build
# Create a tmp directory For QEMU.
$ mkdir tmp

# NOTE: Please check if the qemu sd image is of size 2. If size is not power of 2, we need to resize it to next power of 2 size in MB. Example:
 xen-image-minimal-vck190-versal.wic.qemu-sd size is 768MB, we need to resize it to 1024MB or 1GB. Please do the following:
/scratch/devops/yocto/yocto_test/build/tmp/work/x86_64-linux/qemu-xilinx-helper-native/1.0-r0/recipe-sysroot-native/usr/bin/qemu-img resize
 tmp/deploy/images/vck190-versal/xen-image-minimal-vck190-versal.wic.qemu-sd 1G

# Launch QEMU for VCK190-Versal
tmp/work/x86_64-linux/qemu-xilinx-helper-native/1.0-r0/recipe-sysroot-native/usr/bin/qemu-system-aarch64 \
-drive if=sd,index=1,file=tmp/deploy/images/vck190-versal/xen-image-minimal-vck190-versal.wic.qemu-sd,format=raw \
-serial null -serial null -serial mon:stdio -display none -boot mode=5 \
-hw-dtb tmp/deploy/images/vck190-versal/qemu-hw-devicetrees/multiarch/board-versal-ps-vck190.dtb \
-net nic -net user,tftp=tmp/deploy/images/vck190-versal \
```

# QEMU 에서 실습해보기
# : Start Xen Hypervisor



# QEMU 에서 실습해보기 : Dom-0

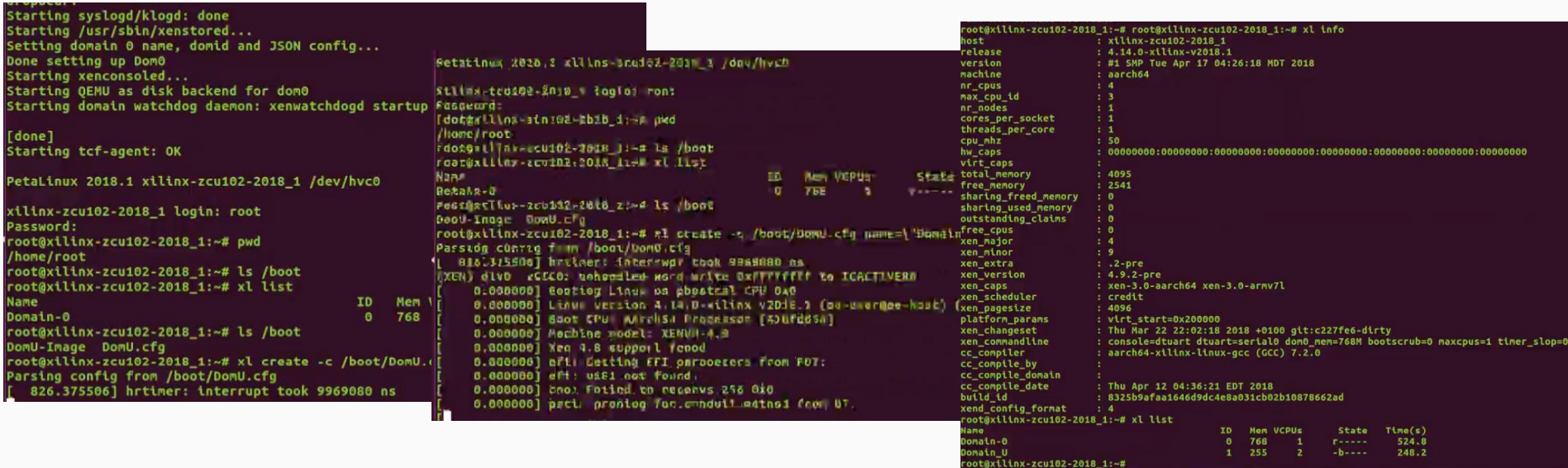
Dom-0 가 부팅되면 Dom-0 를 옮겨서 Domain_U 머신을 생성한다.

```
training@xilinx$ source /home/xilinx/training/Hypervisors_Intro/support/ImportFi
les.sh
Warning: Permanently added '[localhost]:2222' (RSA) to the list of known hosts.
root@localhost's password:
DomU.cfg                                      100%  413     0.4KB/s   00:00
DomU-Image                                     23% 6880KB 593.6KB/s   00:37 ETA
```

```
Starting syslogd/klogd: done
Starting /usr/sbin/xenstored...
Setting domain 0 name, domid and JSON config...
Done setting up Dom0
Starting xenconsoled...
Starting QEMU as disk backend for dom0
Starting domain watchdog daemon: xenwatchdogd startup

[done]
Starting tcf-agent: OK

PetaLinux 2018.1 xilinx-zcu102-2018_1 /dev/hvc0

xilinx-zcu102-2018_1 login: root
Password:
root@xilinx-zcu102-2018_1:~# pwd
/home/root
root@xilinx-zcu102-2018_1:~# ls /boot
root@xilinx-zcu102-2018_1:~# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0   768     1      r-----     381.6
root@xilinx-zcu102-2018_1:~# ls /boot
DomU-Image  DomU.cfg
root@xilinx-zcu102-2018_1:~# xl create -c /boot/DomU.cfg name=\"Domain_U\"
Parsing config from /boot/DomU.cfg
```

# QEMU 에서 실습해보기 : Dom_U

옮긴 Dom_0 의 image, cfg 를 통해 Dom_U 를 부팅한다



# References


- Xen Yocto + QEMU
  - https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/1696137838/Building+Xen+Hypervisor+through+Yocto+Flow
- Xen For Functional Safety
  - https://xenproject.org/developers/teams/embedded-and-automotive/
  - https://wiki.xenproject.org/wiki/FuSa_SIG/Charter
- Xen CVEs
  - https://xenbits.xen.org/xsa/
- Xen VM Isolation
  - https://junjizhi.medium.com/vm-isolation-in-xen-and-kvm-e1f2c6173770
  - https://forum.odroid.com/viewtopic.php?t=18735
- Xen Driver Disaggregation
  - https://wiki.xenproject.org/wiki/Dom0_Disaggregation
- Xen Safety Certify Challenge
  - https://wiki.xenproject.org/wiki/Safety_Certification_Challenges
- Xen Dom0Less
  - https://xenbits.xen.org/docs/unstable/features/dom0less.html
  - https://patchwork.kernel.org/project/xen-devel/patch/20190821035315.12812-8-sstabellini@kernel.org/
  - https://xenproject.org/2019/12/16/true-static-partitioning-with-xen-dom0-less/
- Xen MISRA C
  - https://github.com/xen-project/xen/blob/RELEASE-4.17.2/docs/misra/rules.rst
- Xen PV HVM
  - https://serverfault.com/questions/222010/difference-between-xen-pv-xen-kvm-and-hvm

# References


- Xen Yocto + QEMU
  - https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/1696137838/Building+Xen+Hypervisor+through+Yocto+Flow
  - Xen + Petalinux
    - https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/183074865/Building+Xen+Hypervisor+with+Petalinux+2019.2
    - https://china.xilinx.com/video/soc/hypervisor-introduction.html
- Xen Raspberry pi 4
  - https://xenproject.org/2020/09/29/xen-on-raspberry-pi-4-adventures/
- Xen Real-time schedulers
  - https://wiki.xenproject.org/wiki/Xen_Project_Schedulers
  - https://xenproject.org/2013/12/16/what-is-the-arinc653-scheduler/
- RT-Xen
  - https://ieeexplore.ieee.org/abstract/document/6064510/
  - https://xenproject.org/2013/11/27/rt-xen-real-time-virtualization-in-xen/
- Linux PREEMPT_RT Kernel
  - https://bootlin.com/doc/training/preempt-rt/preempt-rt-slides.pdf
- uboot EL2
  - https://github.com/ARM-software/tf-issues/issues/353

# 감사합니다.

연락처:

회사 이름
도시 도로명 123,
우 12345

p4ranlee@gmail.com