西南交通大学

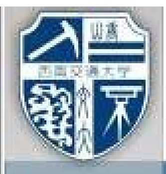
嵌入式系统及应用

西南交通大学

电气工程学院

孙永奎博士

E-mail: yksun@swjtu.edu.cn



第4章 STM32F407介绍及编程实例

- 4.1 STM32F407处理器介绍
- 4.2 STM32F407编程实例
- 5学时

4.1 STM32F407处理器介绍

- 4.1.1 STM32F4 MCU系列概述
- 4.1.2 STM32F407处理器功能
- 4.1.3 STM32F407最小系统
- 4.1.4 STM32F407实验系统

4.1.1 STM32F4 MCU系列概述

1、Cortex内核

Classic ARM Processors
Embedded Cortex Processors
Application Cortex Processors

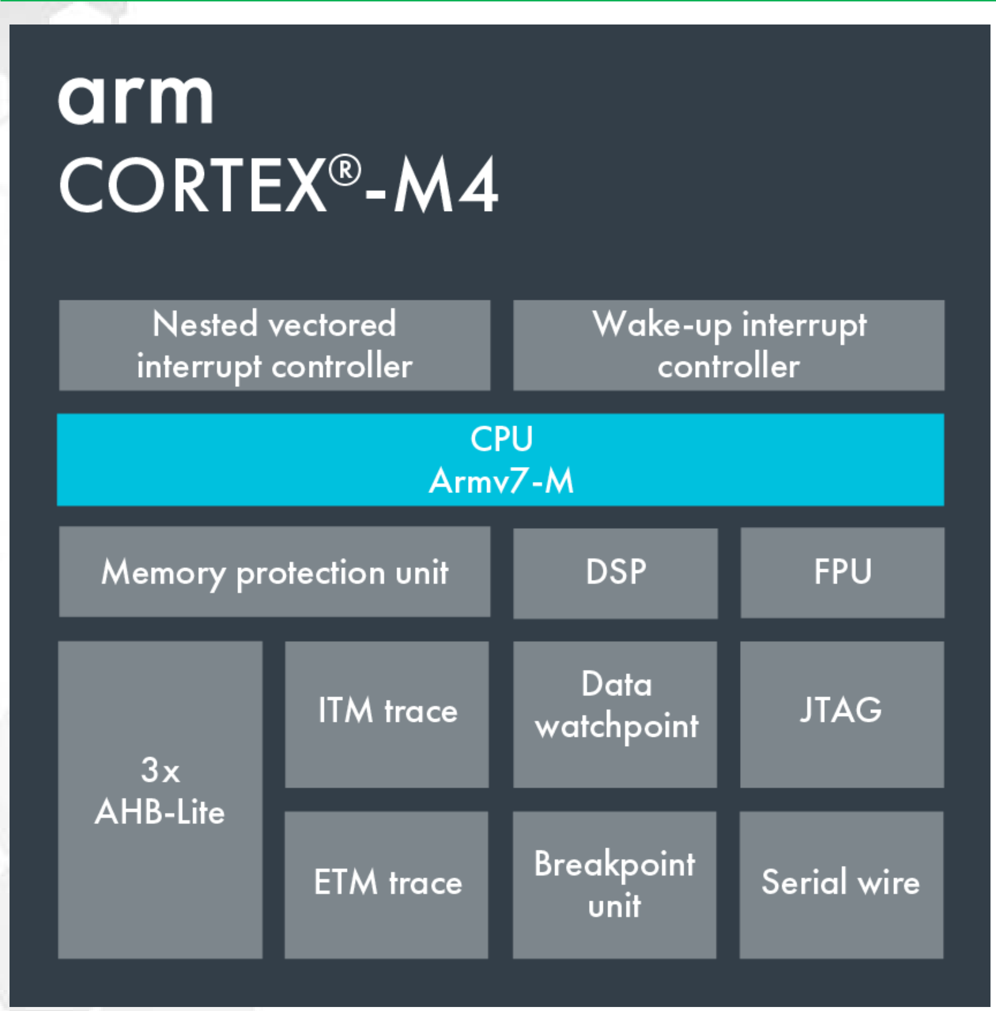
Cortex-A15
Cortex-A9
Cortex-A8
Cortex-A7
Cortex-A5
Cortex-R4
Cortex-M4
Cortex-M3
Cortex-M1
Cortex-M0

ARM11
ARM9
ARM7

Performance, Functionality
Capability

4.1.1 STM32F4 MCU系列概述

2、Cortex-M4应用系统架构



- Cortex-M4是采取了ARMv7-ME架构，
  - 支持DSP应用的指令
  - 一系列单周期MAC指令，
  - 单精度浮点运算指令(FPU)。

4.1.1 STM32F4 MCU系列概述

3、基于Cortex-M4处理器特点

- 32位控制与领先的数字信号处理技术集成来满足需要很高能效级别的市场
  - 单时钟周期乘法累加(MAC)单元、优化的单指令多数据（SIMD）指令、饱和运算指令和一个可选的单精度浮点单元(FPU)
  - RISC处理器内核：高性能32位CPU、具有确定性的运算、低延迟3阶段管道，可达1.25DMIPS/MHz
  - Thumb-2®指令集：16/32位指令的最佳混合、小于8位设备3倍的代码大小、对性能没有负面影响。

4.1.1 STM32F4 MCU系列概述

- 低功耗模式：集成的睡眠状态支持、多电源域、基于架构的软件控制
- 嵌套矢量中断控制器(NVIC)：低延迟、低抖动中断响应、不需要汇编编程、以纯C语言编写的中断服务例程
- 工具和RTOS支持：广泛的第三方工具支持、Cortex微控制器软件接口标准(CMSIS)、最大限度地增加软件成果重用
- CoreSight调试和跟踪：JTAG或2针串行线调试(SWD)连接、支持多处理器、支持实时跟踪
- 嵌入式开发者将得以快速设计并推出令人瞩目的终端产品，具备最多的功能以及最低的功耗和尺寸

4.1.1 STM32F4 MCU系列概述

■ 4、STM32F4 MCU系列

❖ 内核

- 32位 高性能ARM Cortex-M4处理器
- 时钟：高达180M(高级)、 168M(基础级)、100M(入门级
- 支持DSP和FPU（浮点运算）指令

❖ 存储器

- Flash Memory： 128~2056KB, RAM：96~384KB

❖ 外设

- 通用输出/输入口(GPIO)、串口（UART）、CAN、USB
- 真彩色LCD、摄像头接口、HS-SPI、网口、A/D、D/A等

4.1.1 STM32F4 MCU系列概述

STM32F4 MCU Series
32-bit Arm® Cortex®-M4 – Up to 180 MHz

- ART Accelerator™
- SDIO
- USART, SPI, I^2C
- I^2S + audio PLL
- 16 and 32-bit timers
- 12-bit ADC (0.41 µs)
- True Random Number Generator
- Batch Acquisition Mode
- Low voltage 1.7 to 3.6 V
- Temperature:
- -40 °C to 125 °C

Product line	F_{CPU} (MHz)	Flash (KB)	RAM (KB)	Ethernet I/F IEEE 1588	2x CAN	Camera I/F	SDRAM I/F	Dual Quad-SPI	SAI	SPDIF RX	Chrom-ART Graphic Accelerator™	TFT LCD Controller	MIPI DSI
Advanced lines													
STM32F469[2]	180	512 K to 2056 K	384	•	•	•	•	•	•		•	•	•
STM32F429[2]	180	512 K to 2056 K	256	•	•	•	•		•		•	•	
STM32F427[2]	180	1024 K to 2056 K	256	•	•	•	•		•		•		
Foundation lines													
STM32F446	180	256 K to 512 K	128		•	•	•	•	•	•			
STM32F407[2]	168	512 K to 1024 K	192	•	•	•							
STM32F405[2]	168	512 K to 1024 K	192		•								

Product line	F_{CPU} (MHz)	Flash (KB)	RAM (KB)	RUN current (µa/MHz)	STOP current (µa)	Small package (mm)	FSMC (NOR/PSRAM/LCD) support	QSPI	DFSDM	DAC	TRNG	DMA Batch Acquisition mode	USB 2.0 OTG FS
Access lines													
STM32F401	84	128 K to 512 K	up to 96	Down to 128	Down to 10	Down to 3x3							•
STM32F410	100	64 K to 128 K	32	Down to 89	Down to 6	Down to 2.553x 2.579				•	•	BAM	-
STM32F411	100	256 K to 512 K	128	Down to 100	Down to 12	Down to 3.034x 3.22						BAM	•
STM32F412	100	512 K to 1024 K	256	Down to 112	Down to 18	Down to 3.653x 3.651	•	•	•		•	BAM	• +LPM[1]
STM32F413[2]	100	1024 K to 1536 K	320	Down to 115	Down to 18	Down to 3.951x 4.039	•	•	•	•	•	BAM	• +LPM[1]

❖ https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html

4.1.2 STM32F407处理器功能

■1、STM32F407处理器简介

- 时钟：高达168M
- I/O:176引脚，140个I/O(PA~PH为16，PI为12)，大部分IO口都耐5V(模拟通道除外)
- 支持调试：SWD和JTAG，SWD只要2根数据线
- 1024K FLASH，192K SRAM
- 封装LQFP
- 电源：1.8~3.6V
- –40 to 85 ° C

STM32F407IGT6

4.1.2 STM32F407处理器功能

❖ **2 、STM32F407功能框图**

JTAG
Cortex-M4F

I/O

Timer
PWM
UART

A/D

USB

PWR
XTAL

Timer
UART

SPI/I2C

CAN

详见STM32F4x7datasheet

4.1.2 STM32F407处理器功能

❖ 3、F407引脚图

- 有的一个引脚定义多个功能

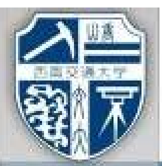
Figure 13. STM32F40x LQFP176 pinout

Pins					Pin name	I / O Level	Function(1) after reset	Alternate functions(2)
LQFP64	LQFP100	LQFP144	UFBGA176	LQFP176				
-	1	1	A2	1	PE2	FT	PE2	TRACECLK/ FSMC_A23 / ETH_MII_TXD3
-	2	2	A1	2	PE3	FT	PE3	TRACED0/FSMC_A19
-	3	3	B1	3	PE4	FT	PE4	TRACED1/FSMC_A20 / DCMI_D4
-	4	4	B2	4	PE5	FT	PE5	TRACED2 / FSMC_A21 / TIM9_CH1 / DCMI_D6
-	5	5	B3	5	PE6	FT	PE6	TRACED3 / FSMC_A22 / TIM9_CH2 / DCMI_D7

4.1.2 STM32F407处理器功能

4、F407总线架构

- 八条主控总线
  - Cortex™-M4F 内核 I 总线、D 总线和 S 总线
  - DMA1 存储器总线
  - DMA2 存储器总线
  - DMA2 外设总线
  - 以太网 DMA 总线
  - USB OTG HS DMA 总线



4.1.2 STM32F407处理器功能

4、F407总线架构

❖ 七条被控总线：

- 内部 Flash ICode 总线
- 内部 Flash DCode 总线
- 主要内部 SRAM1 (112 KB)
- 辅助内部 SRAM2 (16 KB)
- 辅助内部 SRAM3 (64 KB)（仅适用于 STM32F42xxx 和 STM32F43xxx 器件）
- AHB1 外设（包括 AHB-APB 总线桥和 APB 外设）
- AHB2 外设
- FSMC

4.1.2 STM32F407处理器功能

借助总线矩阵，可以实现主控总线到被控总线的访问

4.1.2 STM32F407处理器功能

5、F407存储器组织结构

- 程序存储器、数据存储器、寄存器和 **I/O** 端口排列在同一个顺序的 **4 GB** 地址空间内
- 各字节按小端格式在存储器中编码
- 可寻址的存储空间分为 **8** 个主要块，每个块为 **512 MB**
- 未分配给片上存储器和外设的所有存储区域均视为“保留区”

4.1.2 STM32F407处理器功能

- 存储器映射

Address	Block
0xFFFF FFFF – 0xE000 0000	512-Mbyte block 7 Cortex-M4's internal peripherals
0xDFFF FFFF – 0xC000 0000	512-Mbyte block 6 Not used
0xBFFF FFFF – 0xA000 0000	512-Mbyte block 5 FSMC registers
0x9FFF FFFF – 0x8000 0000	512-Mbyte block 4 FSMC bank 3 & bank4
0x7FFF FFFF – 0x6000 0000	512-Mbyte block 3 FSMC bank1 & bank2
0x5FFF FFFF – 0x4000 0000	512-Mbyte block 2 Peripherals
0x3FFF FFFF – 0x2000 0000	512-Mbyte block 1 SRAM
0x1FFF FFFF – 0x0000 0000	512-Mbyte block 0 Code

Region	Address
Reserved	0x2002 0000 - 0x3FFF FFFF
SRAM (16 KB aliased by bit-banding)	0x2001 C000 - 0x2001 FFFF
SRAM (112 KB aliased by bit-banding)	0x2000 0000 - 0x2001 BFFF
Reserved	0x1FFF C008 - 0x1FFF FFFF
Option Bytes	0x1FFF C000 - 0x1FFF C007
Reserved	0x1FFF 7A10 - 0x1FFF 7FFF
System memory + OTP	0x1FFF 0000 - 0x1FFF 7A0F
Reserved	0x1001 0000 - 0x1FFE FFFF
CCM data RAM (64 KB data SRAM)	0x1000 0000 - 0x1000 FFFF
Reserved	0x0810 0000 - 0x0FFF FFFF
Flash	0x0800 0000 - 0x080F FFFF
Reserved	0x0010 0000 - 0x07FF FFFF
Aliased to Flash, system memory or SRAM depending on the BOOT pins	0x0000 0000 - 0x000F FFFF

Region	Address
Reserved	0xE010 0000 - 0xFFFF FFFF
CORTEX-M4 internal peripherals	0xE000 0000 - 0xE00F FFFF
Reserved	0xA000 1000 - 0xDFFF FFFF
AHB3	0xA000 0FFF – 0x6000 0000
Reserved	0x5006 0C00 - 0x5FFF FFFF
AHB2	0x5006 0BFF – 0x5000 0000
Reserved	0x4008 0000 - 0x4FFF FFFF
AHB1	0x4007 FFFF – 0x4002 000
Reserved	0x4001 5800 - 0x4001 FFFF
APB2	0x4001 57FF – 0x4001 0000
Reserved	0x4000 7800 - 0x4000 FFFF
APB1	0x4000 7FFF – 0x4000 0000

4.1.2 STM32F407处理器功能

■ 6、灵活的静态存储控制器 (FSMC)

- FSMC 能够连接同步、异步存储器和 16 位 PC 存储卡
  - 将 AHB 数据通信事务转换为适当的外部器件协议
  - 满足外部器件的访问时序要求
- 所有外部存储器共享地址、数据和控制信号，但有各自的片选信号。FSMC 一次只能访问一 个外部器件
- FSMC 具有以下主要功能
  - 静态随机访问存储器 (SRAM)
  - 只读存储器 (ROM)
  - NOR Flash/OneNAND Flash
  - PSRAM（4 个存储区域）

4.1.2 STM32F407处理器功能

- FSMC 包含四个主要模块
  - AHB 接口(包括 FSMC 配置寄存器)
  - NOR Flash/PSRAM 控制器
  - NAND Flash/PC 卡控制器
  - 外部器件接口

图 403. FSMC 框图

FSMC 中断到 NVIC
来自时钟控制器
HCLK
AHB 总线
配置寄存器
NOR/PSRAM 存储器控制器
NAND/PC 卡存储器控制器
FSMC_NE[4:1]
FSMC_NL（或 NADV）
FSMC_NBL[1:0]
FSMC_CLK
NOR/PSRAM 信号
FSMC_A[25:0]
FSMC_D[15:0]
FSMC_NOE
FSMC_NWE
FSMC_NWAIT
共享信号
FSMC_NCE[3:2]
FSMC_INT[3:2]
NAND 信号
FSMC_INTR
FSMC_NCE4_1
FSMC_NCE4_2
FSMC_NIORD
FSMC_NIOWR
FSMC_NREG
FSMC_CD
PC 智能卡信号
ai15591b

4.1.2 STM32F407处理器功能

❖ 外部器件地址映射

- 从 **FSMC** 的角度，外部存储器被划分为 **4** 个固定大小的存储区域，每个存储区域的大小为 **256 MB**

图 404.　FSMC 存储区域

地址	存储区域	支持的存储器类型
6000 0000h – 6FFF FFFFh	Bank 1 4 × 64 MB	NOR / PSRAM
7000 0000h – 7FFF FFFFh	Bank 2 4 × 64 MB	NAND Flash
8000 0000h – 8FFF FFFFh	Bank 3 4 × 64 MB	NAND Flash
9000 0000h – 9FFF FFFFh	Bank 4 4 × 64 MB	PC 卡

ai14719

4.1.2 STM32F407处理器功能

❖ 外部存储器接口信号

■ **NOR Flash**，非复用 **I/O**、复用 **I/O**

表 190. 非复用 I/O NOR Flash

FSMC 信号名称	I/O	
CLK	O	时钟（用于同步突
A[25:0]	O	地址总线
D[15:0]	I/O	双向数据总线
NE[x]	O	片选，x = 1..4
NOE	O	输出使能
NWE	O	写入使能
NL(=NADV)	O	锁存使能（对于部
NWAIT	I	FSMC 的 NOR Fl

表 191. 复用 I/O NOR Flash

FSMC 信号名称	I/O	功能
CLK	O	时钟（用于同步突发）
A[25:16]	O	地址总线
AD[15:0]	I/O	16 位复用双向地址/数据总线
NE[x]	O	片选，x = 1..4
NOE	O	输出使能
NWE	O	写入使能
NL(=NADV)	O	锁存使能（对于部分 NOR Flash 器件，此信号也称为地址有效 (NADV)）
NWAIT	I	FSMC 的 NOR Flash 等待输入信号

NOR Flash 存储器采用 **16** 位字寻址。最大容量为 **256Mb**（**26** 个地址线, 4 个片选）

4.1.2 STM32F407处理器功能

❖ 外部存储器接口信号

■ **PSRAM/SRAM，非复用 I/O 、复用 I/O**

表 192. 非复用 I/O PSRAM/SRAM

FSMC 信号名称	I/O	
CLK	O	时钟（仅用于 P
A[25:0]	O	地址总线
D[15:0]	I/O	数据双向总线
NE[x]	O	片选，x = 1..4（ 即 CRAM））
NOE	O	输出使能
NWE	O	写入使能
NL(= NADV)	O	仅用于 PSRAM
NWAIT	I	PSRAM 发送给 I
NBL[1]	O	高字节使能（存
NBL[0]	O	低字节使能（存

表 193. 复用 I/O PSRAM

FSMC 信号名称	I/O	功能
CLK	O	时钟（用于同步突发）
A[25:16]	O	地址总线
AD[15:0]	I/O	16 位复用双向地址/数据总线
NE[x]	O	片选，x = 1..4（在 PSRAM 应用中被称作 NCE（Cellular RAM，即 CRAM））
NOE	O	输出使能
NWE	O	写入使能
NL(= NADV)	O	仅用于 PSRAM 输入的地址有效信号（存储器信号名称：NADV）
NWAIT	I	PSRAM 发送给 FSMC 的等待输入信号
NBL[1]	O	高字节使能（存储器信号名称：NUB）
NBL[0]	O	低字节使能（存储器信号名称：NLB）

PSRAM存储器采用 **16** 位字寻址。最大容量为 **256 Mb**（**26** 个地址线，4个片选）

4.1.2 STM32F407处理器功能

❖7、自举配置 （Boot configuration）

❖ 可通过 BOOT[1:0] 引脚选择三种不同的自举模式

表 3. 自举模式

自举模式选择引脚		自举模式	自举空间
BOOT1	BOOT0		
x	0	主 Flash	选择主 Flash 作为自举空间
0	1	系统存储器	选择系统存储器作为自举空间
1	1	嵌入式 SRAM	选择嵌入式 SRAM 作为自举空间

4.1.2 STM32F407处理器功能

8、复位

❖ 系统复位

- **NRST** 引脚低电平（外部复位）
- 窗口看门狗计数结束（**WWDG** 复位）
- 独立看门狗计数结束（**IWDG** 复位）
- 软件复位（**SW** 复位）
- 低功耗管理复位

除时钟控制寄存器CSR中的复位标志和备份域中的寄存器外，系统复位会将其它全部寄存器都复位为复位值

4.1.2 STM32F407处理器功能

8、复位

- 电源复位
  - 上电复位（POR）
  - 掉电复位（PDR）
  - 欠压复位（BOR）
  - 退出待机模式

除备份域内的寄存器以外，电源复位会将其它全部寄存器设置为复位值

4.1.2 STM32F407处理器功能

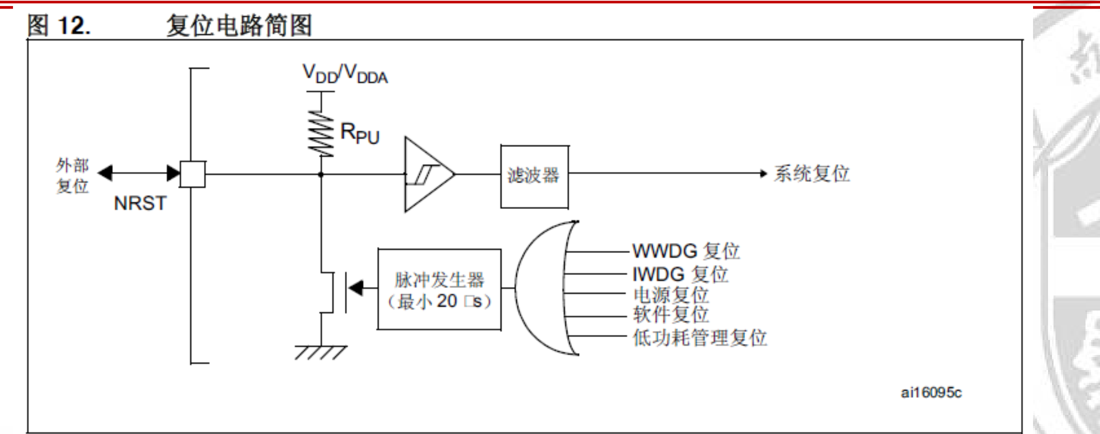
■ 8、复位

❖ 备份域复位

- 软件复位（ SW 复位）
- 在电源 VDD 和 VBAT 都已掉电后，其中任何一个又再上电欠压复位（BOR）

备份域复位会将所有 RTC 寄存器和 RCC_BDCR 寄存器复位为各自的复位值。

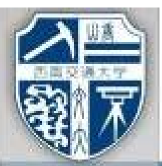
图 12.　　复位电路简图



4.1.2 STM32F407处理器功能

9、时钟

- 三种不同的时钟源来驱动系统时钟 (SYSCLK)
  - 高速内部振荡器时钟（HSI）——16MHz的高速RC振荡器
  - 高速外部振荡器时钟（HSE）——4~26M的外部高速晶振
  - 主 PLL 时钟(PLL) ——使用外部或者内部高速时钟作为PLL的输入
- 两个次级时钟源
  - 内部32kHz低速RC振荡器（LSI），看门狗时钟
  - 32.768 kHz 低速外部晶振（LSE 晶振），用于驱动 RTC 时钟 (RTCCLK



4.1.2 STM32F407处理器功能

时钟树

LSI
LSIRC
32 kHz
LSI
看门狗使能
IWDGCLK
到独立看门狗
RTCSEL[1:0]
LSE
OSC32_IN
OSC32_OUT
LSEOSC
32.768kHz
LSE
RTC
使能
RTCCLK
到 RTC
MCO2
/1 到 5
SYSCLK
LSE
HSE_RTC
MCO1
/1 到 5
/2到31
外设时钟使能
以太网
PTP 时钟
HSI
16 MHz
HSI RC
HSI
HCLK
最高 168 MHz
时钟
使能
到 AHB 总线、内核、
存储器和 DMA
/8
到 Cortex 系统
定时器
FCLK Cortex
自由运行时钟
SW
HSI
HSE
PLLCLK
AHB
PRESC
/1,2,..512
SYSCLK
最高 168 MHz
APBx
PRESC
/1,2,4,8,16
外设时钟使能
APBx
外设时钟
外设时钟使能
If (APBx presc = 1)x1
else x2
APBx
定时器时钟
HSE
OSC_OUT
OSC_IN
4-26 MHz
HSE OSC
HSE
/ M
PLL
VCO
xN
/ P
/ Q
/ R
PLL
PLL48CK
外设时钟使能
48 MHz
时钟
VCO
xN
/ P
/ Q
/ R
PLLI2S
I2SSRC
PLLI2SCLK
外部时钟
外设时钟使能
I2S 时钟
I2S_CKIN

4.1.2 STM32F407处理器功能

■ 10、嵌套向量中断控制器 (NVIC)

- ❖ 82 个可屏蔽中断通道
- ❖ 16 个可编程优先级
- ❖ 嵌套向量中断控制器 (NVIC) 和处理器内核接口紧密配合，可以实现低延迟的中断处理和晚到中断的高效处理

■ 11、外部中断/事件控制器 (EXTI)

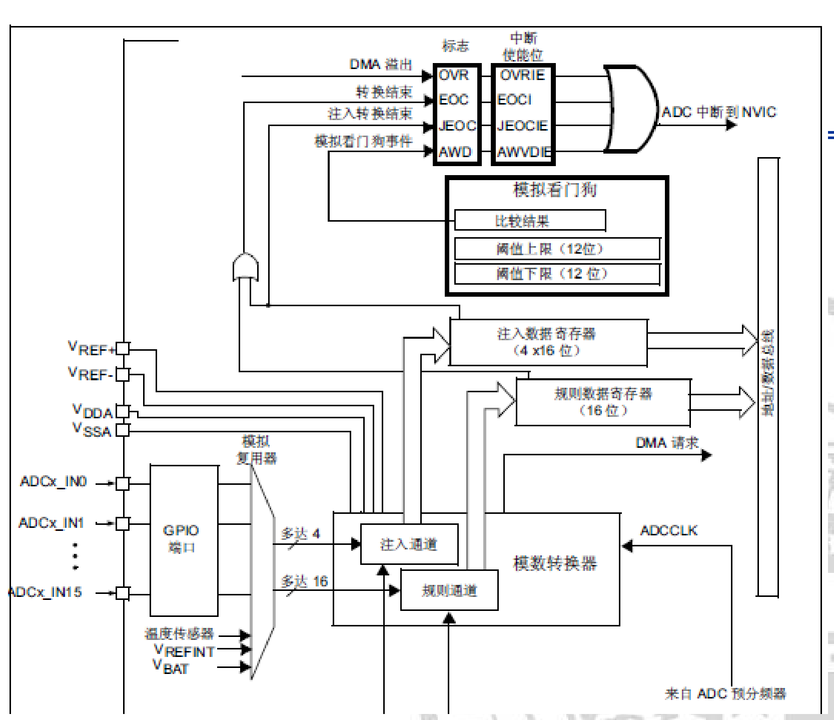
- ❖ 23 个用于产生事件/中断请求的边沿检测器
- ❖ 每根输入线都可单独进行配置，以选择类型（中断或事件）和相应的触发事件（上升沿触发、下降沿触发或边沿触发）
- ❖ 每根输入线还可单独屏蔽

4.1.2 STM32F407处理器功能

■ 12、A/D

❖ **A/D**

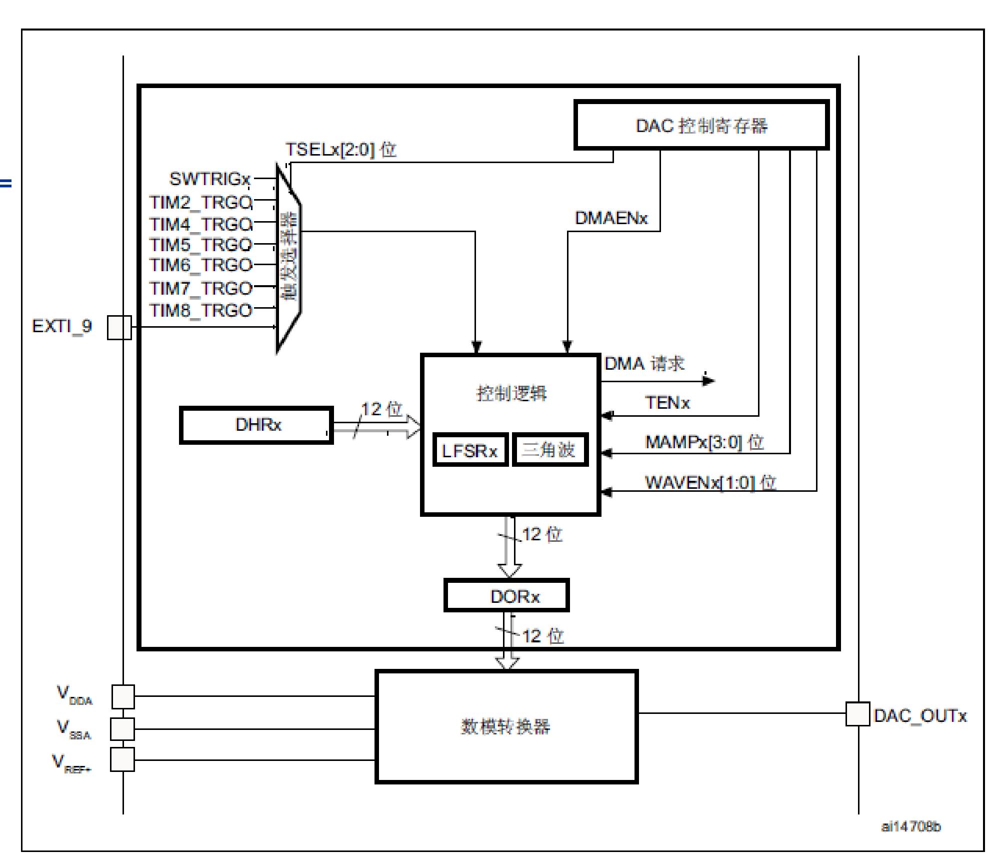
- 12位A/D(16个外部测量通道)，可配置为12位、10位、8位或6位分辨率
- 内部通道可以用于内部温度测量
- 内置参考电压



4.1.2 STM32F407处理器功

13、D/A

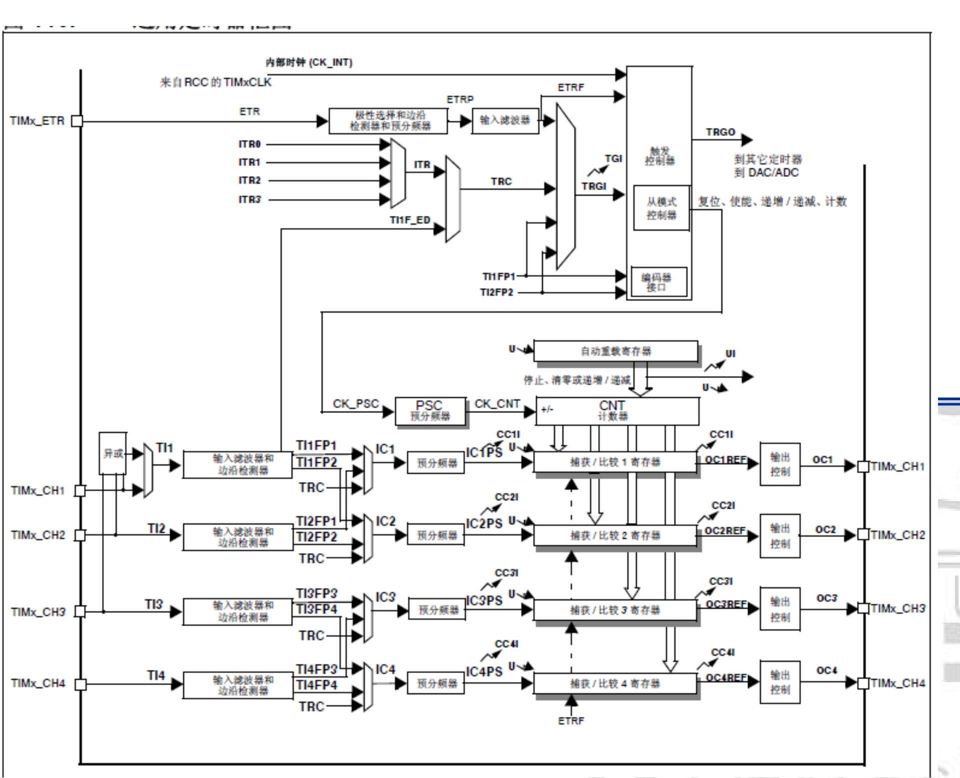
- 12 位电压输出数模转换器,可配置8 位或 12 位
- 2个输出通道
- 生成噪声波、生成三角波



4.1.2 STM32F407处理器功能

14、多定时器

- 17个定时器
- 10个通用定时器（TIM2和TIM5是32位）
- 2个基本定时器
- 2个高级定时器
- 1个系统定时器
- 2个看门狗定时器



4.1.2 STM32F407处理器功能

15、通信接口

- I2C接口
- 串口
- SPI接口
- CAN2.0
- USB OTG
- SDIO

- 其他外设：DMA、PWM、CRC等请见数据手册和参考手册

4.1.2 STM32F407处理器功能

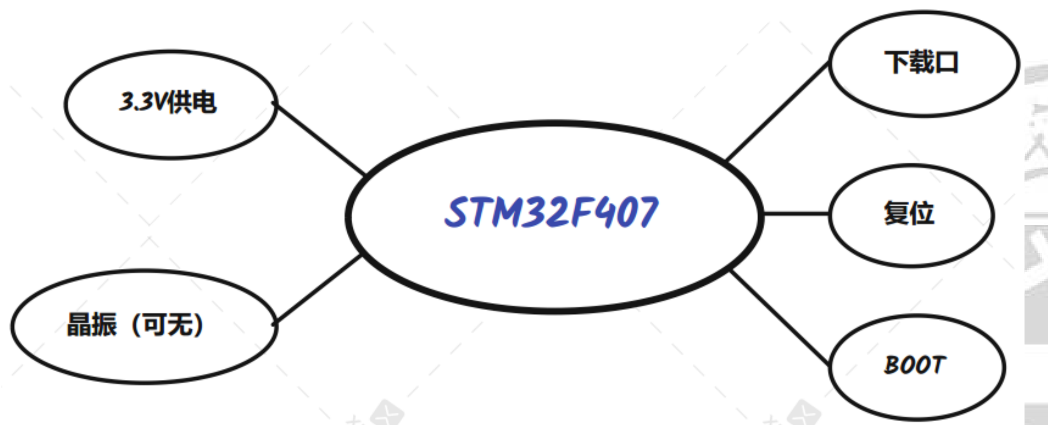
16、STM32F407典型应用领域

- 电机驱动器
- 医疗设备
- 工业应用：PLC、逆变器
- 打印机、扫描仪
- 警报系统、视频对讲设备、暖通空调
- 家庭音响

4.1.3 STM32F407最小系统

■ **STM32F407最小系统**

- 3.3V供电
- 复位电路
- 时钟电路（外部晶振（可选））
- Boot启动模式选择
- 下载电路（串口/JTAG/SWD)



4.1.3 STM32F407最小系统

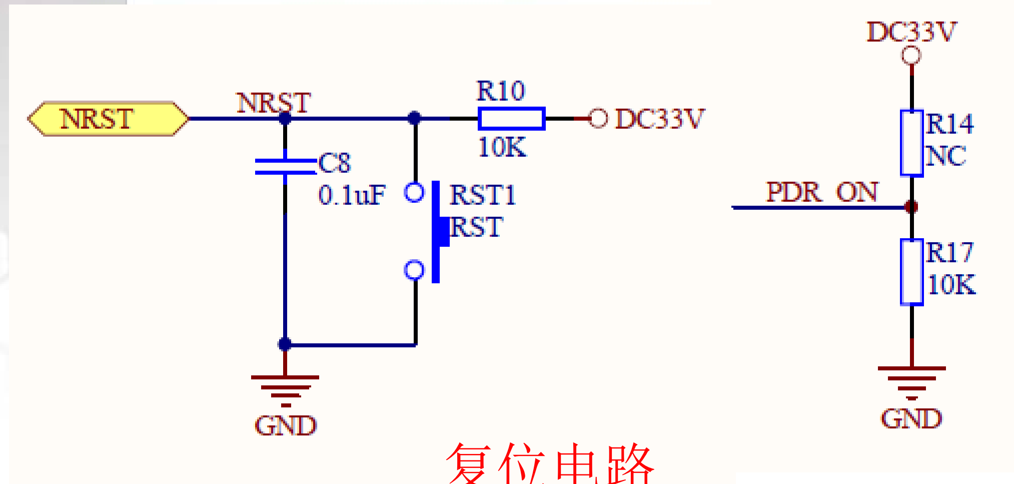
主晶振

C6 20pF
C7 20pF
Y1 8Mhz
R9 1M
GND
U1E
PH1 30
PH0 29
VREF+ 38
VDDA 39
VSSA 37
AGND
BOOT0166 BOOT0
R15 10K
GND
自举配置

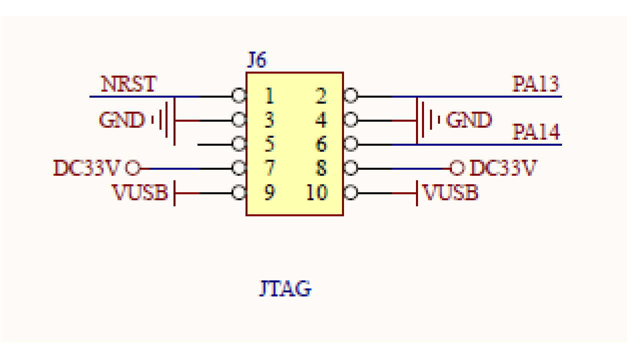
PC15 10
PC14 9
C9 6pF
M1 32.768KHz
C10 6pF
GND
VBAT 6 VBAT
复位
PDR_ON 171 PDR_ON EC2
EC3
VCAP_2 125 2.2uF/6.3V
VCAP_1 81 2.2uF/6.3V
NRST 31 NRST
复位
GND
GND

R4 0R
GND AGND
STM32F439

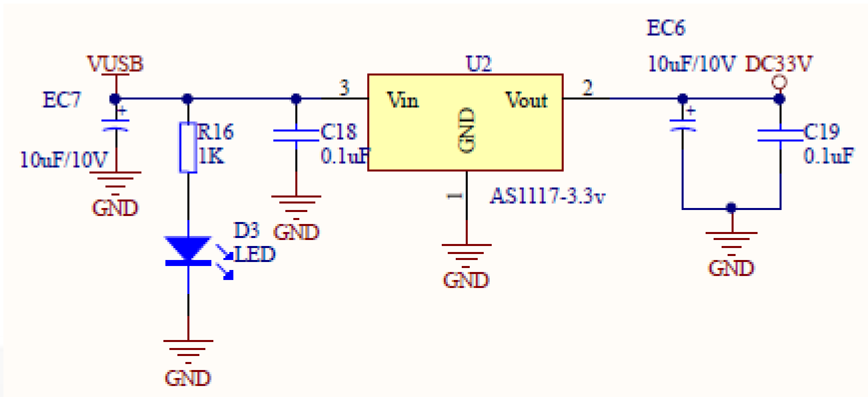
4.1.3 STM32F407最小系统

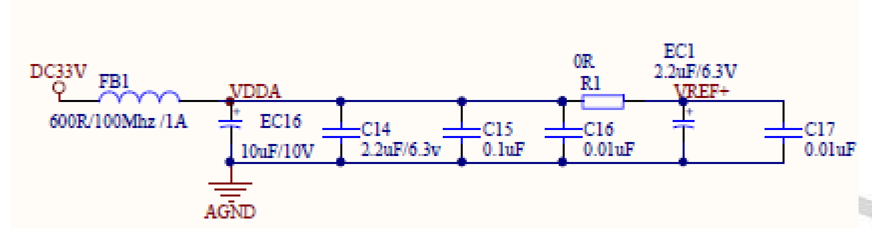


复位电路



JTAG电路





电源电路

4.1.4 STM32F407实验系统介绍

■1、实验系统组成

❖ 微控制器标配ARM Cortex-M4（STM32F407）

Arduino电机扩展板
Arduino键盘扩展板
（选配）Cortex-M4核心板
4G模块
Arduino 传感器扩展板
（标配）
Cortex-A53网关
可更换
WiFi接口
IPv6接口
433M接口
ZigBee/蓝牙接口\协调器
主板锁
RS232
RS232

4.1.4 STM32F407实验系统介绍

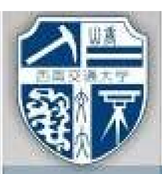
	功能部件	型号参数
Cortex-M4 配置	**CPU**	**- ARM Cortex-M4**内核的**STM32F407** **-**主频**168MHz**
	RAM容量	**-192K SRAM**
	Flash容量	**-1M Flash**
	外部资源	**-**外扩**8Mbit NOR Flash** **-**外扩**1Mbit SRAM** **- SWD**下载接口 **- 10M/100M**以太网**DP83848CVV** **- 1**路 **TTL UART**接口 **- 2**路贴片**LED**灯 **-**双排**50pin** 插针 **-**独立复位按键；

4.1.4 STM32F407实验系统介绍

	功能部件	型号参数
外设资源	按键数码管板	键数码管板
	电机板	- 1个直流电机 - 1个步进电机 - 1个舵机
	传感器板	- 1个酒精传感器 - 1个光敏传感器 - 1个温度传感器 - 1个烟雾传感器 1个火焰传感器 - 1个继电器传感器 - 1个光电开关传感器 - 1个蜂鸣器传感器 - 1个红外接收器传感器

第4章 STM32F407介绍及编程实例

- 4.1 STM32F407处理器介绍
- 4.2 STM32F407编程实例
- 8学时



4.2 STM32F407编程实例

- 4.2.1 C 语言复习
- 4.2.2 GPIO编程实例
- 4.2.3 中断编程实例
- 4.2.4 异步串口通信编程实例
- 4.2.5 ADC编程实例

4.2.1 C 语言复习

1、C语言介绍

- C 语言高级语言
  - 高效性
  - 灵活性
  - 可移植性
- C语言是面向过程的语言
  - 编程人员明确制定程序实现的步骤
  - 程序由函数/子程序组成

4.2.1 C 语言复习

❖ C 语言是一种接近硬件的编程语言
- 开发驱动程序
- 开发嵌入式系统程序
- 操作系统

❖ C语言非常流行编程语言
- 长期位于编程语言前三

Feb 2022	Feb 2021	Change	Programming Language	Ratings	Change
1	3	^	Python	15.33%	+4.47%
2	1	v	C	14.08%	-2.26%
3	2	v	Java	12.13%	+0.84%
4	4		C++	8.01%	+1.13%

4.2.1 C 语言复习

❖ C 程序包含4部分

■ 程序文档说明部分、预处理部分、全局声明部分、程序代码部分

文档说明部分

```
//**** 0. Documentation Section
//  This program calculates the area of square shaped rooms
//  Author:
//  Date:
//
```

预处理部分

```
// 1. Pre-processor Directives Section
#include <stdio.h>  // Diamond braces for sys lib: Standard I/O
#include "uart.h"   // Quotes for user lib: UART lib
```

全局声明部分

```
// 2. Global Declarations section
// 3. Subroutines Section
```

程序代码部分

```
// MAIN: Mandatory routine for a C program to be executable
int main(void)
{
    UART_Init();  // call subroutine to initialize the uart
    printf("This program calculates areas of square-shaped rooms\n");
}
```

Main函数

4.2.1 C 语言复习

❖ C 语言包括了很多库函数

- printf() 输出函数
- 头文件中的各种定义
- #include<header_file.h>

```
// ***** 1. Pre-processor Directives Section *****
#include <stdio.h>    // standard C library
#include "uart.h"     // functions to implment input/output
//#include "TExaS.h"    // Lab grader functions

// ***** 2. Global Declarations Section *****
```

4.2.1 C 语言复习

■2、C语言常用的基础知识

❖1）位操作

■ **6种位操作运算符**

运算符	含义	运算符	含义
&	按位与	~	取反
|	按位或	<<	左移
^	按位异或	>>	右移

4.2.1 C 语言复习

❖ 按位操作例子

```
unsigned int a = 60;  /* 60 = 0011 1100 */
unsigned int b = 13;  /* 13 = 0000 1101 */
int c = 0;

c = a & b;        /* 12 = 0000 1100 */

c = a | b;        /* 61 = 0011 1101 */

c = a ^ b;        /* 49 = 0011 0001 */

c = ~a;           195 = 1100 0011 */

c = a << 2;       /* 240 = 1111 0000 */

c = a >> 2;       /* 15 = 0000 1111 */
```

4.2.1 C 语言复习

❖ 按位操作例子

Set a bit:

x |= (1 << n)

Clear a bit:

x &= ~(1 << n)

Toggle a bit:

x ^= (1 << n)

Test a bit:

x & (1 << n)

```
unsigned char x = 0010 1000;

x |= (1<<4); // x = 0011 1000; x |= 0x10;

x &= ~(1<<5); // x= 0000 1000; x &= ~(0x20);

x ^= (1<<2); // 0010 1100;  x ^= 0x02;

if (x & (1 << 6))
     do a;
else do b;
```

4.2.1 C 语言复习

❖ 2） define宏定义关键词

- define是C语言中的预处理命令
- 宏定义是用宏名来表示一个字符串，在宏展开时又以该字符串取代宏名

#define 标识符　字符串

- “标识符”为宏名。“字符串”可以是常数、表达式、格式串等

4.2.1 C 语言复习

❖ 宏定义例子

```
#define  SYSCLK_FREQ_72MHz    72000000
```

标识符SYSCLK_FREQ_72MHz的值为72000000

4.2.1 C 语言复习

- 3）ifdef条件编译
  - **嵌入式程序开发中**，当满足某条件时对一组语句进行编译，而当条件不满足时则编译另一组语句
- 条件编译命令格式

```
#ifdef 标识符
     程序段1
#else
     程序段2
#endif
```

4.2.1 C 语言复习

❖ ifdef条件编译例子

```
ifdef STM32F10X_HD
    大容量芯片需要的一些变量定义
#end
```

STM32F10X_HD 通过#define 来定义

4.2.1 C 语言复习

- 4）extern变量申明
  - *extern*可以置于变量或者函数前，以表示变量或者函数的定义在别的文件中，提示编译器遇到此变量和函数时在其他模块中寻找其定义
  - 对于extern申明变量可以多次，但定义只有一次

4.2.1 C 语言复习

❖ extern变量申明例子

main.c文件

```
u8 id;//定义只允许一次
main()
{
	id=1;
	printf("d%",id);//id=1
	test();
	printf("d%",id);//id=2
}
```

test.c文件

```
extern u8 id;

void test(void)
{
	id=2;
}
```

4.2.1 C 语言复习

❖ 5） typedef类型别名

- 用于为现有的类型创建一个新名字（类型别名），用于简化变量的定义。

```
typedef unsigned   char          uint8_t;
typedef unsigned   short  int    uint16_t;
typedef unsigned   int           uint32_t;
typedef unsigned   __int64       uint64_t;
```

4.2.1 C 语言复习

- 6） 结构体
  - 构造类型
  - 表示不同类型的变量
- 声明结构体类型与定义变量

```
Struct 结构体名{
        成员列表1;
        成员变量2;
        …
}变量名列表;
```

Struct 结构体名字　结构体变量列表

4.2.1 C 语言复习

❖ 结构体变量例子

```
Struct UART_TYPE{
        int BaudRate
        int WordLength
}uart1, uart2;
        Struct UART_TYPE  uart1, uart2;
```

结构体成员变量的引用

```
        uart1. BaudRate= 9600;
        uart2.WordLength = 32;
```

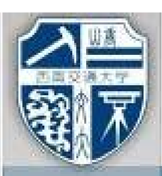
4.2.1 C 语言复习

❖ 结构体指针变量例子

```
Struct UART_TYPE  *uart3;
```

结构体指针成员变量的引用

```
uart3->BaudRate= 9600;
Uart3->WordLength = 32;
```



4.2.1 C 语言复习

❖ 结构体与函数相比的优点

例子：串口初始化程序

- 参数：端口号、波特率、极性、模式

函数实现串口初始化程序

Void UART_INIT(u8 uart_no, u32 BautRate, u8 Parity, u8 mode)

如果需要在初始化函数中增加一个参数：如字长

Void UART_INIT(u8 uart_no, u32 BautRate, u8 Parity, u8 mode， u8 WordLength)

重新定义函数

4.2.1 C 语言复习

❖ 使用结构体可以在不改变入口参数，通过改变成员变量来改变入口参数。

```
Typedef struct {
        uint32_t UART_BautRate;
        uint16_t UART_WordLength;
        uint16_t UART_StopBits;
        uint16_t UART_Parity;
        uint16_t UART_Mode;
        uint16_t UART_HWFlowcontrl;
}UART_InitType
```

4.2.1 C 语言复习

❖ 串口初始化程序中入口参数为UART_InitType类型的变量或者指针变量

Void UART_INIT(u8 uart_no, UART_InitType UART_InitStruct)

Void UART_INIT(u8 uart_no, UART_InitType * UART_InitStruct)

❖ 添加新参数，只需要在结构中加入新的成员变量，不用修改函数定义

4.2.1 C 语言复习

❖ 7）static关键字

- C 语言中，static声明变量具有隐藏功能
  - 变量自己仅仅在其的作用范围内可见（定义静态变量文件中可见）
- C 语言中， static声明的变量内容能够保持
  - 变量会被放在程序的全局存储区（静态存储区），下一次调用的时候还可以保持原来的赋值。
- C 语言中，static声明变量默认初始化为0

4.2.1 C 语言复习

静态变量的例子1

src1.c

```
 char a = 'A'; // 全局变量
void msg()
{

   printf("Hello\n");

}
```

src2.c

```
int main(void)
{
  extern char a;//外部变量声明
  printf("%c ", a);
  (void )msg();
   return 0;
}
```

程序的运行结果：A Hello

4.2.1 C 语言复习

src1.c

```
static char a = 'A'; // 全局变量
static void msg()
{

    printf("Hello\n");

}
```

src2.c

```
int main(void)
{
   extern char a;//外部变量声明
   printf("%c ", a);
   (void )msg();
    return 0;
}
```

程序的运行结果：?

4.2.1 C 语言复习

静态变量的例子2

```
#include <stdio.h>
int fun(void)
{
    static int count = 10;
    return count--;
}
    int count = 1;

    int main(void)
{
    printf("global\t\tlocal static\n");
    for( ; count <= 10;++count)
    printf("%d\t\t%d\n" , count, fun());
    return 0;
}
```

只在程序刚运行时完成唯一一次初始化

程序的运行结果:

Global	local static
1	10
2	9
3	8
4	7
5	6
6	5
7	4
8	3
9	2
10	1

4.2.2 GPIO编程实例

■1、GPIO工作方式

❖ 输入模式

- 输入浮空、 输入上拉
- 输入下拉、 模拟输入

❖ 输出模式

- 带上拉或者下拉开漏输出、带上拉或者下拉推挽式输出
- 具有上拉或下拉功能的复用功能推挽、具有上拉或下拉功能的复用功能开漏

4.2.2 GPIO编程实例

❖5 V 容忍 I/O 端口位的基本结构

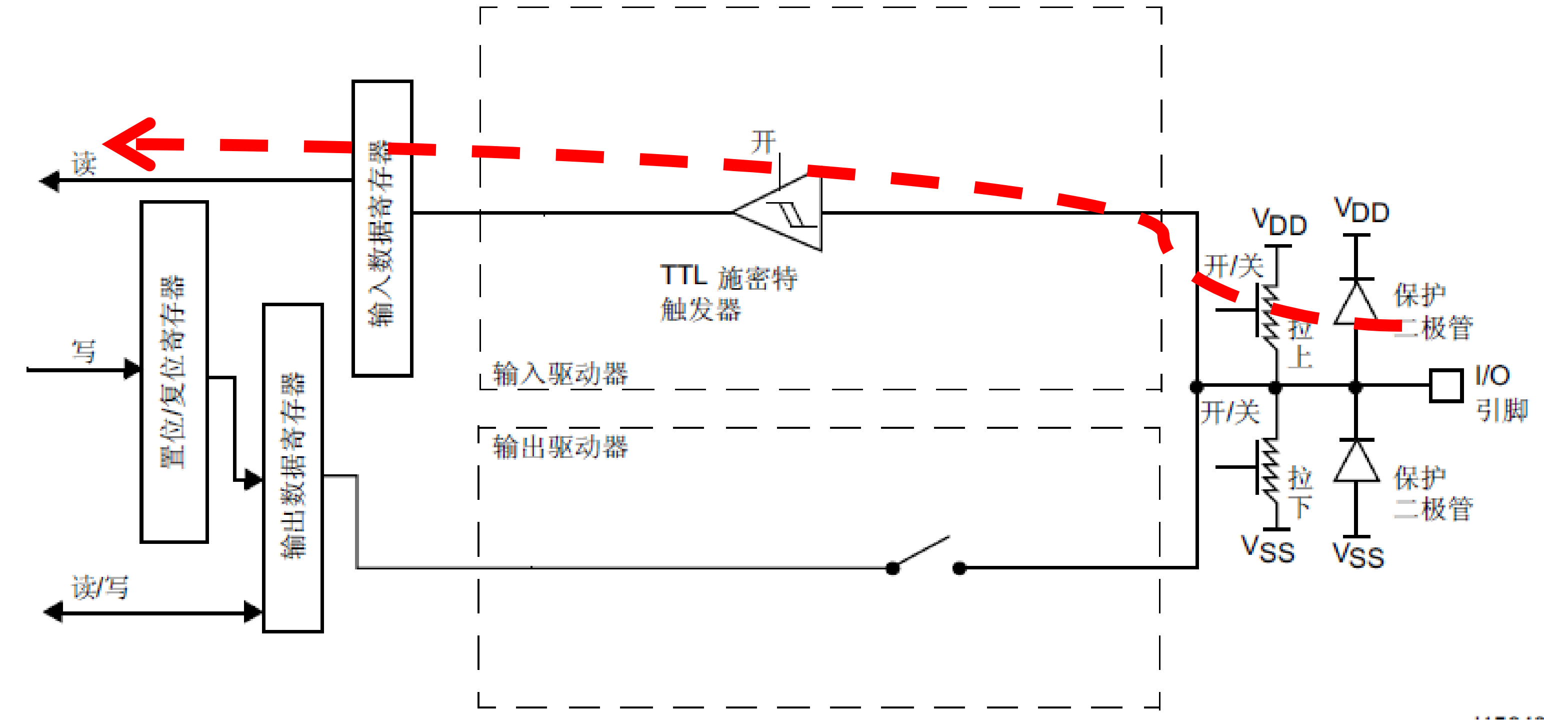
到片上外设
模拟
复用功能输入
开/关
读
输入数据寄存器
TTL 施密特触发器
置位/复位寄存器
写
输出数据寄存器
开/关
输入驱动器
输出驱动器
V_{DD}
开/关
P-MOS
输出控制
N-MOS
V_{SS}
推挽
开漏或
禁止
读/写
自片上外设
复用功能输出
模拟
V_{DD}
V_{DD_FT} (1)
保护二极管
I/O 引脚
保护二极管
V_{SS}
V_{SS}
ai15939b
FT:5V容忍

1. V_{DD_FT} 是和 5 V 容忍 I/O 相关的电位，与 V_{DD} 不同。

4.2.2 GPIO编程实例

❖ 输入配置（输入浮空/上拉/下拉）

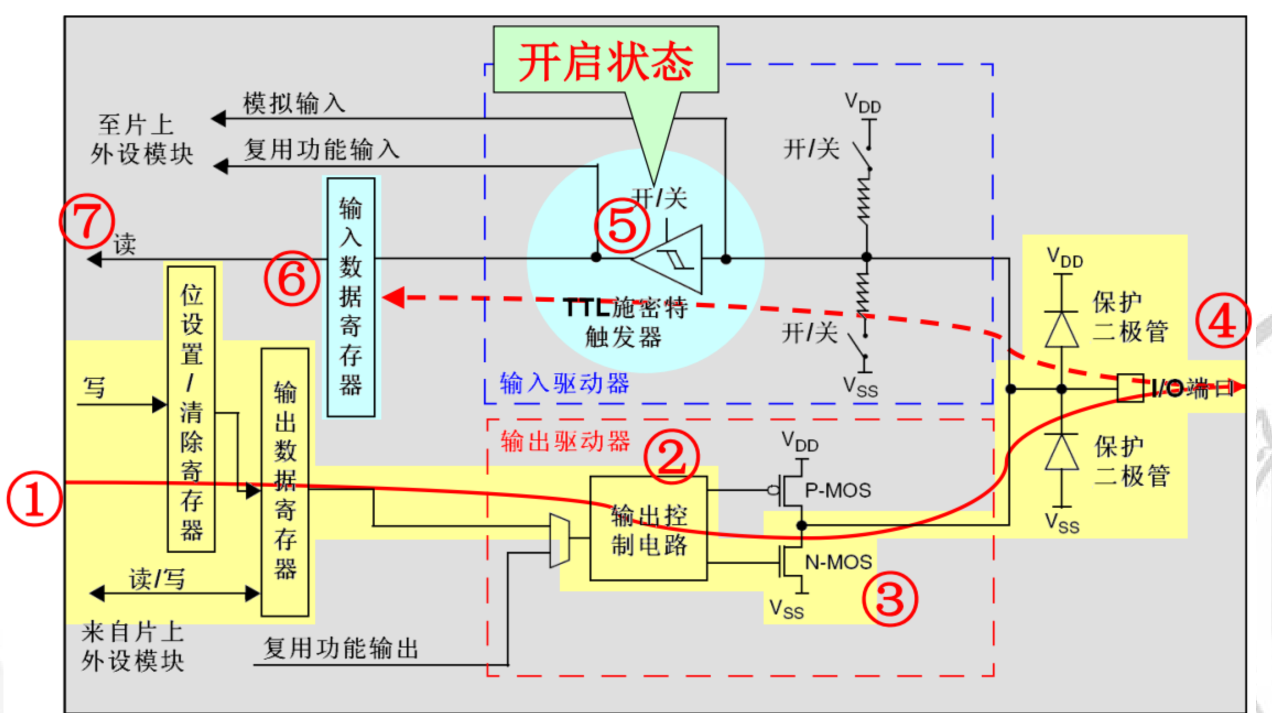
- 输出缓冲器被关闭、施密特触发器输入被打开
- 对输入数据寄存器的读访问可获取I/O状态
- GPIOx_PUPDR 寄存器中配置是否浮空、上拉下拉



4.2.2 GPIO编程实例

❖ 输出配置

- 输出缓冲器被打开、施密特触发器输入被打开
- 对输入数据寄存器的读访问可获取I/O 状态
- 对输出数据寄存器的读访问可获取最后的写入值
- GPIOx_PUPDR 寄存器中配置上拉或下拉



4.2.2 GPIO编程实例

■2、GPIO端口寄存器

- 端口模式寄存器（GPIOx_MODER）
- 端口输出类型寄存器(GPIOx_OTYPER)
- 端口输出速度寄存器（GPIOx_OSPEEDR)
- 端口上拉下拉寄存器（GPIOx_PUPDR)

4个32位控制寄存器

- 端口输入数据寄存器（GPIOx_IDR）
- 端口输出数据寄存器（GPIOx_ODR)

2个16位数据寄存器

- 端口置位/复位寄存器（GPIOx_BSRR)
- 端口配置锁存寄存器（GPIOx_LCKR)
- 两个复位功能寄存器（低位GPIOx_AFRL & GPIOx_AFRH)

每组I/O口用10个寄存器控制其16个I/O口。

4.2.2 GPIO编程实例

■3、GPIO端口初始化步骤

❖1）GPIO端口时钟使能

❖2）配置4个GPIO控制寄存器

- GPIOx_MODER
- GPIOx_OTYPER
- GPIOx_OSPEEDR
- GPIOx_PUPDR

4.2.2 GPIO编程实例

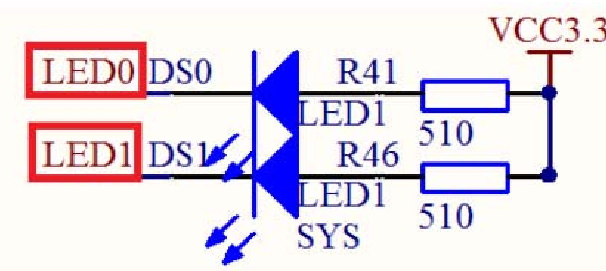
■4、GPIO端口数据操作

- 从端口输入数据
  - 读端口输入数据寄存器（GPIOx_IDR）
- 从端口输出数据
  - 写端口输出数据寄存器（GPIOx_ODR)

4.2.2 GPIO编程实例

■5、用GPIO端口控制LED

❖ GPIO端口控制LED原理图



- GPIO的F端口PIN9和PIN10控制2个LED
- PIN9和PIN10输出低时LED亮，高时LED灭
- 设F端口推挽输出

4.2.2 GPIO编程实例

❖1） GPIO F端口时钟使能

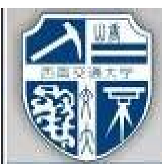
6.3.12 RCC AHB1 外设时钟使能寄存器 (RCC_AHB1ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	OTGHS ULPIEN	OTGHS EN	ETHMA CPTPE N	ETHMA CRXEN	ETHMA CTXEN	ETHMA CEN	Reserved		DMA2EN	DMA1EN	CCMDATA RAMEN	Res.	BKPSR AMEN	Reserved	
	rw	rw	rw	rw	rw	rw			rw	rw			rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved			GPIOIE N	GPIOH EN	GPIOGE N	GPIOFE N	GPIOEEN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

位 5 **GPIOFEN**：IO 端口 F 时钟使能 (IO port F clock enable 1：使能 IO 端口 E 时钟)

```
RCC->AHB1ENR|=1<<5//F端口时钟使能
```



4.2.2 GPIO编程实例

❖ 2）配置4个GPIO控制寄存器

■ （1）GPIO 端口模式寄存器 (GPIOx_MODER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_MODER位20~21： 端口 F 的PIN10（01：普通输出），

位**18~19**端口 F 的PIN9（01:普通输出）

```
PIN9: GPIOF->MODER&=~(3<<(9*2));//清除原来的设置
      GPIOF->MODER|=(1<<(9*2));//设置为输出模式
PIN10:
      GPIOF->MODER&=~(3<<(10*2));
      GPIOF->MODER|=(1<<(10*2));
```

4.2.2 GPIO编程实例

- (2) GPIO 端口输出类型寄存器 (GPIOx_OTYPER) 配置

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_OTYPER位9～10： 位10端口 F 的PIN10（0：推挽输出），位9端口 F 的PIN9（ 0：推挽输出）

```
PIN9: GPIOF->OTYPER&=~(1<<9);//清除原来的设置
      GPIOF-> OTYPER |=(0<<9);//设置为推挽输出
PIN10:
      GPIOF-> OTYPER &=~(1<<10);
      GPIOF-> OTYPER |=(0<<10);
```

4.2.2 GPIO编程实例

- （3）GPIO 端口输出速度寄存器 (GPIOx_OSPEEDR) 配置

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15[1:0]		OSPEEDR14[1:0]		OSPEEDR13[1:0]		OSPEEDR12[1:0]		OSPEEDR11[1:0]		OSPEEDR10[1:0]		OSPEEDR9[1:0]		OSPEEDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1[1:0]		OSPEEDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_OSPEEDR位20~21： 端口 F 的PIN10（01：25MHz），

位18~19端口 F 的PIN9（ 01：25MHz ）

```
PIN9: GPIOF->OSPEEDR&=~(3<<(9*2));//清除原来的设置
      GPIOF->OSPEEDR|=(1<<(9*2))//设置为25MHz
PIN10:
      GPIOF->OSPEEDR&=~(3<<(10*2));
      GPIOF->OSPEEDR|=(1<<(10*2));
```

4.2.2 GPIO编程实例

- （4） GPIO 端口上拉/下拉寄存器 (GPIOx_PUPDR) 配置

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_OSPPUPDR位20~21： 端口 F 的PIN10（00：无上拉下拉），**位18~19**端口 F 的PIN9（ 00：无上拉下拉 ）

```
PIN9:  GPIOF->PUPDR &=~(3<<(9*2));//清除原来的设置
       GPIOF-> PUPDR |=(0<<(9*2));//设置为无上拉下拉
PIN10:
       GPIOF-> PUPDR &=~(3<<(10*2));
       GPIOF-> PUPDR |=(0<<(10*2));
```

4.2.2 GPIO编程实例

（5）控制LED亮灭

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

```
PIN9（LED0）：GPIOF->ODR= 1<<9; //LED0灭
              GPIOF->ODR=0<<9; // LED0亮
PIN10（LED1）：
              GPIOF->ODR =1<<9;
              GPIOF->ODR =0<<9;
```

4.2.2 GPIO编程实例

❖LED0和LED1交替亮灭的流程图和程序

```
#include "sys.h"
#include "delay.h"
#include "led.h"

int main(void)
{
  Stm32_Clock_Init(336,8,2,7);//设置时钟,168Mhz
  delay_init(168);      //初始化延时函数
  LED_Init();         //初始化端口和LED
  while(1)
  {
    LED0=0;          //DS0亮
    LED1=1;          //DS1灭
    delay_ms(500);
    LED0=1;          //DS0灭
    LED1=0;          //DS1亮
    delay_ms(500);
  }
}
```

4.2.2 GPIO编程实例

❖ 关键代码

```
//LED端口定义
#define LED0 PFout(9) // PORTF_PIN9
#define LED1 PFout(10)  // PORTF_PIN10

void LED_Init(void);//初始化
#endif
```

```
//初始化PF9和PF10为输出口.并使能这两个口的时钟
//LED IO初始化
void LED_Init(void)
{
  RCC->AHB1ENR|=1<<5;//使能PORTF时钟
  GPIO_Set(GPIOF,PIN9|PIN10,GPIO_MODE_OUT,GPIO_OTYPE_PP,GPIO_SPEED_100M,GPIO_PUPD_PU); //PF9,PF10设置
  LED0=1;//LED0关闭
  LED1=1;//LED1关闭
}
```

4.2.2 GPIO编程实例

```
//GPIO通用设置
//GPIOx:GPIOA~GPIOI.
//BITx:0X0000~0XFFFF,位设置,每个位代表一个IO,第0位代表Px0,第1位代表Px1,依次类推.比如0X0101,代表同时设置Px0和Px8.
//MODE:0~3;模式选择,0,输入(系统复位默认状态);1,普通输出;2,复用功能;3,模拟输入.
//OTYPE:0/1;输出类型选择,0,推挽输出;1,开漏输出.
//OSPEED:0~3;输出速度设置,0,2Mhz;1,25Mhz;2,50Mhz;3,100Mh.
//PUPD:0~3:上下拉设置,0,不带上下拉;1,上拉;2,下拉;3,保留.
//注意:在输入模式(普通输入/模拟输入)下,OTYPE和OSPEED参数无效!!
void GPIO_Set(GPIO_TypeDef* GPIOx,u32 BITx,u32 MODE,u32 OTYPE,u32 OSPEED,u32 PUPD)
{
  u32 pinpos=0,pos=0,curpin=0;
  for(pinpos=0;pinpos<16;pinpos++)
  {
    pos=1<<pinpos;  //一个个位检查
    curpin=BITx&pos;//检查引脚是否要设置
    if(curpin==pos) //需要设置
    {
      GPIOx->MODER&=~(3<<(pinpos*2)); //先清除原来的设置
      GPIOx->MODER|=MODE<<(pinpos*2); //设置新的模式
      if((MODE==0X01)||(MODE==0X02))  //如果是输出模式/复用功能模式
      {
        GPIOx->OSPEEDR&=~(3<<(pinpos*2)); //清除原来的设置
        GPIOx->OSPEEDR|=(OSPEED<<(pinpos*2));//设置新的速度值
        GPIOx->OTYPER&=~(1<<pinpos) ;    //清除原来的设置
        GPIOx->OTYPER|=OTYPE<<pinpos;    //设置新的输出模式
      }
      GPIOx->PUPDR&=~(3<<(pinpos*2)); //先清除原来的设置
      GPIOx->PUPDR|=PUPD<<(pinpos*2); //设置新的上下拉
    }
  }
}
```

4.2.2 GPIO编程实例

```
//系统时钟初始化函数
//plln:主PLL倍频系数(PLL倍频),取值范围:64~432.
//pllm:主PLL和音频PLL分频系数(PLL之前的分频),取值范围:2~63.
//pllp:系统时钟的主PLL分频系数(PLL之后的分频),取值范围:2,4,6,8.(仅限这4个值!)
//pllq:USB/SDIO/随机数产生器等的主PLL分频系数(PLL之后的分频),取值范围:2~15.
void Stm32_Clock_Init(u32 plln,u32 pllm,u32 pllp,u32 pllq)
{
  RCC->CR|=0x00000001;      //设置HISON,开启内部高速RC振荡
  RCC->CFGR=0x00000000;     //CFGR清零
  RCC->CR&=0xFEF6FFFF;      //HSEON,CSSON,PLLON清零
  RCC->PLLCFGR=0x24003010;   //PLLCFGR恢复复位值
  RCC->CR&=~(1<<18);        //HSEBYP清零,外部晶振不旁路
  RCC->CIR=0x00000000;      //禁止RCC时钟中断
  Sys_Clock_Set(plln,pllm,pllp,pllq);//设置时钟
  //配置向量表
#ifdef  VECT_TAB_RAM
  MY_NVIC_SetVectorTable(1<<29,0x0);
#else
  MY_NVIC_SetVectorTable(0,0x0);
#endif
}
```

4.2.3 中断编程实例

■1、STM32F407外部中断/事件控制器

AMBA APB 总线

PCLK2

外设接口

23

挂起请求寄存器

中断屏蔽寄存器

软件中断事件寄存器

上升沿触发选择寄存器

下降沿触发选择寄存器

至 NVIC 中断控制器

脉冲发生器

事件屏蔽寄存器

边沿检测电路

输入线

4.2.3 中断编程实例

❖ 外部中断/事件控制器 (EXTI)

- STM32F407具有多达23个用于产生事件/中断请求的边沿检测器
- 每根输入线都可单独进行配置，以选择类型（中断或事件）和相应的触发事件（上升沿触发、下降沿触发或边沿触发）
- STM32F407 的140 个 GPIO都可以作为外部中断输入

4.2.3 中断编程实例

❖ 外部中断映射

- EXTI线0~15：对应外部IO口的输入中断
- EXTI线16：PVD输出
- EXTI线17：RTC闹钟事件
- EXTI线18：USB OTG FS唤醒事件
- EXTI线19：以太网唤醒事件
- EXTI线20：USB OTG HS(在FS中配置)唤醒事件
- EXTI线21：RTC入侵和时间戳事件
- EXTI线22：连接到RTC唤醒事件

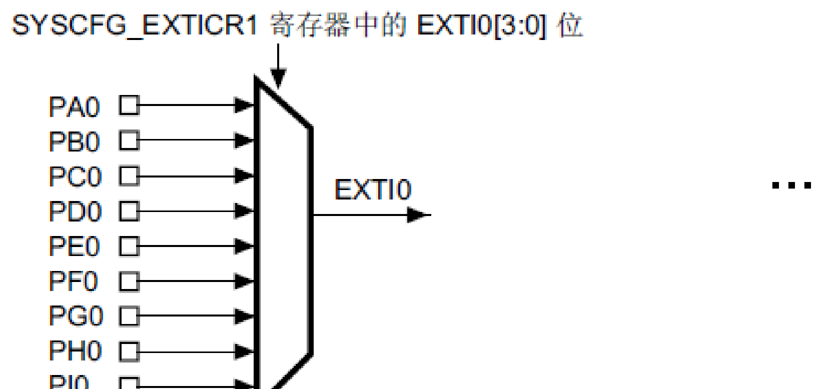
4.2.3 中断编程实例

- 产生中断条件：配置并使能中断线
  - 设置2个边沿触发寄存器
  - 设置中断屏蔽寄存器相应位“1”，使用中断请求
  - 挂起寄存器的对应位写“1”，将清除该中断请求
- 硬件中断
  - 配置23根中断线的屏蔽位(EXTI_IMR)
  - 配置中断线的触发选择位（EXTI_RTSR和EXTI_FTSR)
  - 配置对应到外部中断控制器(EXTI)的NVIC中断通道的使能和屏蔽位

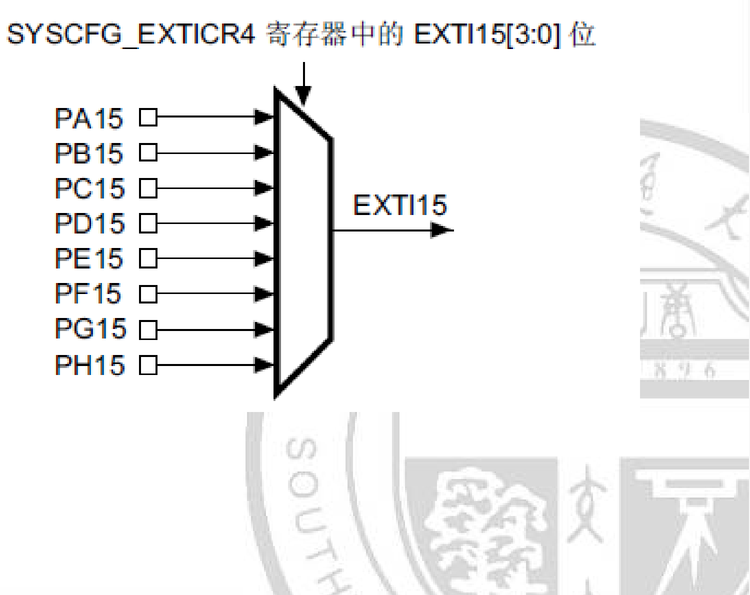
4.2.3 中断编程实例

❖ F407外部中断线与GPIO的映射

- **GPIOx.0映射到EXTI0**
- **GPIOx.1映射到EXTI1**
- **GPIOx.15映射到EXTI15**
- **SYSCFG 外部中断配置寄存器1 (SYSCFG_EXTICR1~4)**



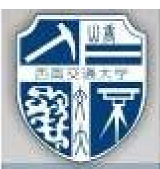
...



4.2.3 中断编程实例

- GPIO口外部中断的中断向量表
  - 7个中断向量
  - 使用7个中断服务函数

位置	优先级	优先级类型	名称	说明	地址
7	14	可设置	EXTI1	EXTI线1中断	0x0000_005C
8	15	可设置	EXTI2	EXTI线2中断	0x0000_0060
9	16	可设置	EXTI3	EXTI线3中断	0x0000_0064
10	17	可设置	EXTI4	EXTI线4中断	0x0000_0068
23	30	可设置	EXTI9_5	EXTI线[9:5]中断	0x0000_009C
40	47	可设置	EXTI15_10	EXTI线[15:10]中断	0x0000_00E0



4.2.3 中断编程实例

2、嵌套向量中断控制器 (NVIC)

- STM32F407具有 82 个可屏蔽中断通道和10个内核中断
- 16 个可编程优先级（使用了 4 位中断优先级）
- 0：最高优先级；15：最低优先级

位置	优先级	优先级类型	名称	说明	地址
	-	-	-	保留	0x0000 0000
	-3	固定	Reset	复位	0x0000 0004
	-2	固定	NMI	不可屏蔽中断。RCC 时钟安全系统 (CSS) 连接到 NMI 向量。	0x0000 0008
	-1	固定	HardFault	所有类型的错误	0x0000 000C
	0	可设置	MemManage	存储器管理	0x0000 0010
	1	可设置	BusFault	预取指失败，存储器访问失败	0x0000 0014
	2	可设置	UsageFault	未定义的指令或非法状态	0x0000 0018
	-	-	-	保留	0x0000 001C - 0x0000 002B
	3	可设置	SVCall	通过 SWI 指令调用的系统服务	0x0000 002C
	4	可设置	Debug Monitor	调试监控器	0x0000 0030
	-	-	-	保留	0x0000 0034
	5	可设置	PendSV	可挂起的系统服务	0x0000 0038
	6	可设置	SysTick	系统嘀嗒定时器	0x0000 003C

10个内核中断

位置	优先级	优先级类型	名称	说明	地址
0	7	可设置	WWDG	窗口看门狗中断	0x0000 0040
1	8	可设置	PVD	连接到 EXTI 线的可编程电压检测 (PVD) 中断	0x0000 0044
2	9	可设置	TAMP_STAMP	连接到 EXTI 线的入侵和时间戳中断	0x0000 0048
3	10	可设置	RTC_WKUP	连接到 EXTI 线的 RTC 唤醒中断	0x0000 004C
4	11	可设置	FLASH	Flash 全局中断	0x0000 0050
5	12	可设置	RCC	RCC 全局中断	0x0000 0054
77	84	可设置	OTG_HS	USB On The Go HS 全局中断	0x0000 0174
78	85	可设置	DCMI	DCMI 全局中断	0x0000 0178
79	86	可设置	CRYP	CRYP 加密全局中断	0x0000 017C
80	87	可设置	HASH_RNG	哈希和随机数发生器全局中断	0x0000 0180
81	88	可设置	FPU	FPU 全局中断	0x0000 0184

82个可屏蔽中断

4.2.3 中断编程实例

❖NVIC中断优先级分组

- **STM32**中断进行分组，组**0~4**
- 每个中断设置一个抢占优先级和一个响应优先级值
- 寄存器SCB-AIRCR、 NVIC-IPR中配置

组	AIRCR[10：8]	IP bit[7：4]分配情况	分配结果
0	111	0：4	0位抢占优先级，4位响应优先级
1	110	1：3	1位抢占优先级，3位响应优先级
2	101	2：2	2位抢占优先级，2位响应优先级
3	100	3：1	3位抢占优先级，1位响应优先级
4	011	4：0	4位抢占优先级，0位响应优先级

4.2.3 中断编程实例

❖ 抢占优先级与响应优先级区别

- 高优先级的抢占优先级是可以打断正在进行的低抢占优先级中断的
- 抢占优先级相同的中断，高响应优先级不可以打断低响应优先级的中断
- 抢占优先级相同的中断，当两个中断同时发生的情况下，哪个响应优先级高，哪个先执行
- 如果两个中断的抢占优先级和响应优先级都是一样的话，则看哪个中断先发生就先执行

4.2.3 中断编程实例

❖ 例子

- 假定设置中断优先级组为2，
- 设置中断3(RTC中断)的抢占优先级为2，响应优先级为1。中断6（外部中断0）的抢占优先级为3，响应优先级为0。中断7（外部中断1）的抢占优先级为2，响应优先级为0。

优先级顺序为：**中断7>中断3>中断6**

4.2.3 中断编程实例

❖NVIC中断设置相关寄存器

- NVIC_IPR， 中断优先级控制的寄存器组
- NVIC_ISER，中断使能寄存器组
- NVIC_ICER[8]，中断关闭寄存器组
- NVIC_ISPR[8]，中断挂起寄存器组
- NVIC_ICPR[8]，中断解挂寄存器组
- NVIC_IABR[8]，中断激活标志位寄存器组

4.2.3 中断编程实例

3、外部中断配置步骤

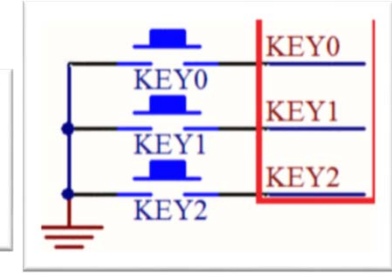
- （1）使能SYSCFG时钟:
- （2）初始化IO口为输入。
- （3）设置IO口与中断线的映射关系。
- （4）初始化线上中断，设置触发条件等。
- （5）配置中断分组（NVIC），并使能中断。
- （6）编写中断服务函数。
- （7）清除中断标志位

4.2.3 中断编程实例

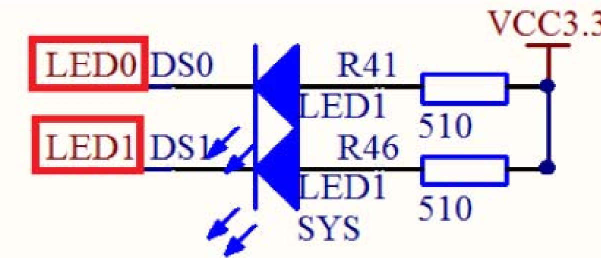
■4、用GPIO外部中断控制LED

❖GPIO外部中断控制LED的原理图

PE1/FSMC_NBL1/DCMI_D3
PE2/TRACECLK/FSMC_A23/ETH_MII_TXD3 — 1 PE2 KEY2
PE3/TRACED0/FSMC_A19 — 2 PE3 KEY1
PE4/TRACED1/FSMC_A20/DCMI_D4 — 3 PE4 KEY0



7/TIM11_CH1/FSMC_NREG/ADC3_IN5 — 19 PF7 LIGHT_SEN
/TIM13_CH1/FSMC_NIOWR/ADC3_IN6 — 20 PF8 BEEP
PF9/TIM14_CH1/FSMC_CD/ADC3_IN7 — 21 PF9 LED0
PF10/FSMC_INTR/ADC3_IN8 — 22 PF10 LED1
PF11/DCMI_D12 — 49 PF11 T_MOSI
50 PF12 FSMC_A6



- GPIO的F端口PIN9和PIN10控制2个LED
- PIN9和PIN10输出低时LED亮，高时LED灭
- KEY0与PE4、 KEY1与PE3、 KEY2与PE2相连作为中断输入
- KEY0控制LED0和LED1，KEY1控制LED1，KEY0控制LED0。

4.2.3 中断编程实例

❖ 设置IO口与中断线的映射关系

- **PE4、PE3、 PE2**相连作为中断输入
- SYSCFG 外部中断配置寄存器 1 (SYSCFG_EXTICR1 ~2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI7[3:0]				EXTI6[3:0]				EXTI5[3:0]				EXTI4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

0000：PA[x] 引脚
0001：PB[x] 引脚
0010：PC[x] 引脚
0011：PD[x] 引脚
0100：PE[x] 引脚
0101：PF[x] 引脚
0110：PG[x] 引脚
0111：PH[x] 引脚
1000：PI[x] 引脚

PE2：**SYSCFG_EXTICR1 _EXTI2**＝ 0100;
PE3：**SYSCFG_EXTICR1 _EXTI3**＝ 0100;
PE4：**SYSCFG_EXTICR1 _EXTI4**＝ 0100;

4.2.3 中断编程实例

❖ 外部中断控制LED流程图和程序

```
#include "sys.h"
#include "led.h"
#include "exti.h"
#include "key.h"

int main(void)
{
  Stm32_Clock_Init(336,8,2,7);//设置时钟,168Mhz
  LED_Init();           //初始化与LED连接的硬件接口
  KEY_Init();           //端口E初始化
  EXTIX_Init();         //初始化外部中断输入
  LED0=0;           //先点亮红灯
  while(1)
  {

  }
}
```

4.2.3 中断编程实例

❖ 中断服务程序和流程图

```
//外部中断2服务程序
void EXTI2_IRQHandler(void)
{
  delay_ms(10); //消抖
  if(KEY2==0)
  {
    LED0=!LED0;
  }
  EXTI->PR=1<<2;  //清除LINE2上的中断标志位
}
```

4.2.3 中断编程实例

```
//外部中断3服务程序
void EXTI3_IRQHandler(void)
{
  delay_ms(10); //消抖
  if(KEY1==0)
  {
    LED1=!LED1;
  }
  EXTI->PR=1<<3;  //清除LINE3上的中断标志位
}
```

```
//外部中断4服务程序
void EXTI4_IRQHandler(void)
{
  delay_ms(10); //消抖
  if(KEY0==0)
  {
    LED0=!LED0;
    LED1=!LED1;
  }
  EXTI->PR=1<<4;  //清除LINE4上的中断标志位
}
```

4.2.3 中断编程实例

❖ 端口E（按键）初始化程序

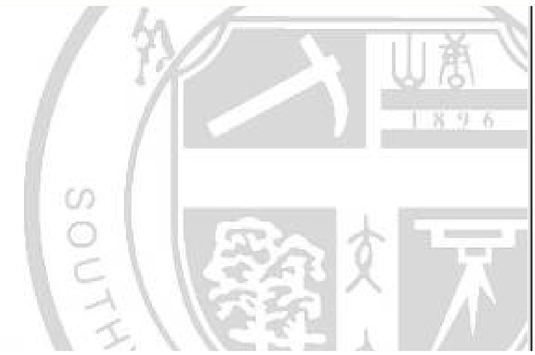
```
//按键初始化函数
void KEY_Init(void)
{
  RCC->AHB1ENR|=1<<0;      //使能PORTA时钟
  RCC->AHB1ENR|=1<<4;      //使能PORTE时钟
  GPIO_Set(GPIOE,PIN2|PIN3|PIN4,GPIO_MODE_IN,0,0,GPIO_PUPD_PU); //PE2~4设置上拉输入

}
```

4.2.3 中断编程实例

❖ 中断配置程序

```
//外部中断初始化程序
//初始化PE2~4为中断输入.
void EXTIX_Init(void)
{
   Ex_NVIC_Config(GPIO_E,2,FTIR);      //下降沿触发
   Ex_NVIC_Config(GPIO_E,3,FTIR);      //下降沿触发
   Ex_NVIC_Config(GPIO_E,4,FTIR);      //下降沿触发
   MY_NVIC_Init(3,2,EXTI2_IRQn,2);     //抢占3，子优先级2，组2
   MY_NVIC_Init(2,2,EXTI3_IRQn,2);     //抢占2，子优先级2，组2
   MY_NVIC_Init(1,2,EXTI4_IRQn,2);     //抢占1，子优先级2，组2
   MY_NVIC_Init(0,2,EXTI0_IRQn,2);     //抢占0，子优先级2，组2
}
```



4.2.3 中断编程实例

❖ 中断配置程序

```
//外部中断配置函数
//只针对GPIOA~I;不包括PVD,RTC,USB_OTG,USB_HS,以太网唤醒等
//参数:
//GPIOx:0~8,代表GPIOA~I
//BITx:需要使能的位;
//TRIM:触发模式,1,下升沿;2,上降沿;3, 任意电平触发
//该函数一次只能配置1个IO口,多个IO口,需多次调用
//该函数会自动开启对应中断,以及屏蔽线
void Ex_NVIC_Config(u8 GPIOx,u8 BITx,u8 TRIM)
{
  u8 EXTOFFSET=(BITx%4)*4;
  RCC->APB2ENR|=1<<14;                    //使能SYSCFG时钟
  SYSCFG->EXTICR[BITx/4]&=~(0x000F<<EXTOFFSET);//清除原来设置！！！
  SYSCFG->EXTICR[BITx/4]|=GPIOx<<EXTOFFSET; //EXTI.BITx映射到GPIOx.BITx
  //自动设置
  EXTI->IMR|=1<<BITx;              //开启line BITx上的中断(如果要禁止中断, 则反操作即可)
  if(TRIM&0x01)EXTI->FTSR|=1<<BITx; //line BITx上事件下降沿触发
  if(TRIM&0x02)EXTI->RTSR|=1<<BITx; //line BITx上事件上升降沿触发
}
```

4.2.3 中断编程实例

❖ 中断配置程序

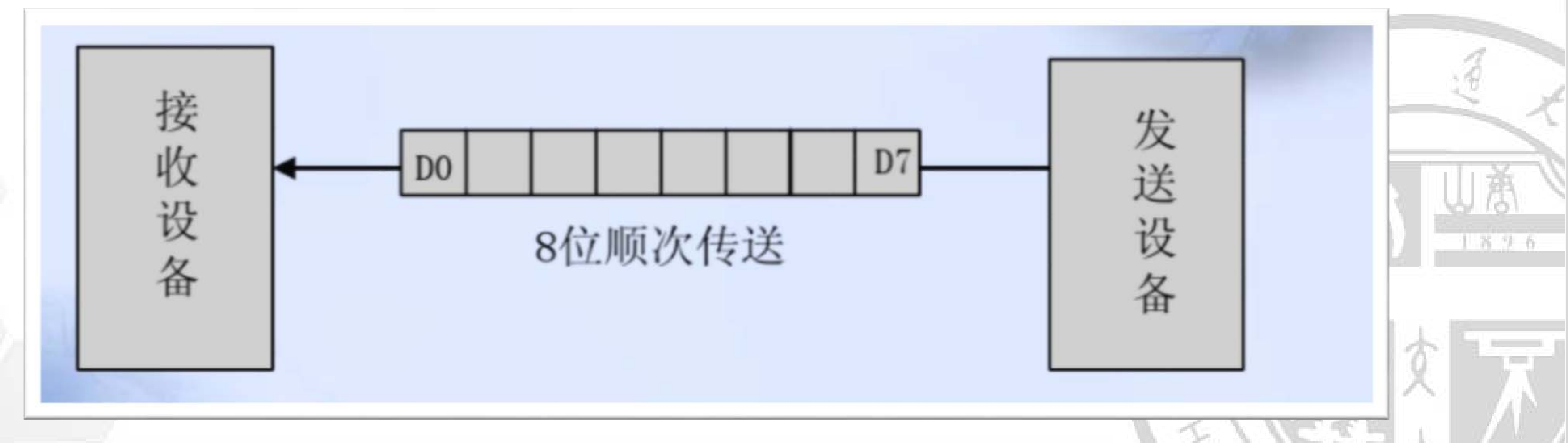
```
//设置NVIC
//NVIC_PreemptionPriority:抢占优先级
//NVIC_SubPriority       :响应优先级
//NVIC_Channel           :中断编号
//NVIC_Group             :中断分组 0~4
//注意优先级不能超过设定的组的范围!否则会有意想不到的错误
//组划分:
//组0:0位抢占优先级,4位响应优先级
//组1:1位抢占优先级,3位响应优先级
//组2:2位抢占优先级,2位响应优先级
//组3:3位抢占优先级,1位响应优先级
//组4:4位抢占优先级,0位响应优先级
//NVIC_SubPriority和NVIC_PreemptionPriority的原则是,数值越小,越优先
void MY_NVIC_Init(u8 NVIC_PreemptionPriority,u8 NVIC_SubPriority,u8 NVIC_Channel,u8 NVIC_Group)
{
  u32 temp;
  MY_NVIC_PriorityGroupConfig(NVIC_Group);//设置分组
  temp=NVIC_PreemptionPriority<<(4-NVIC_Group);
  temp|=NVIC_SubPriority&(0x0f>>NVIC_Group);
  temp&=0xf;                        //取低四位
  NVIC->ISER[NVIC_Channel/32]|=1<<NVIC_Channel%32;//使能中断位(要清除的话,设置ICER对应位为1即可)
  NVIC->IP[NVIC_Channel]|=temp<<4;         //设置响应优先级和抢断优先级
}
```

4.2.4 异步串口通信编程实例

■1、串行通信基本知识

❖ 串行通信特点

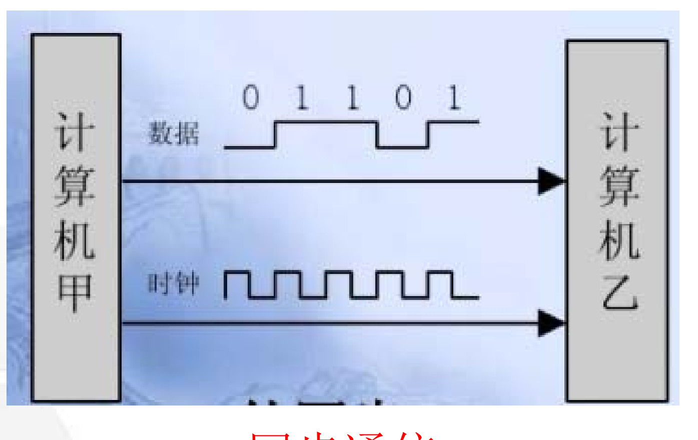
- 数据按位顺序在一条传输线逐个传输
- 占用引脚资源少
- 速度相对较慢



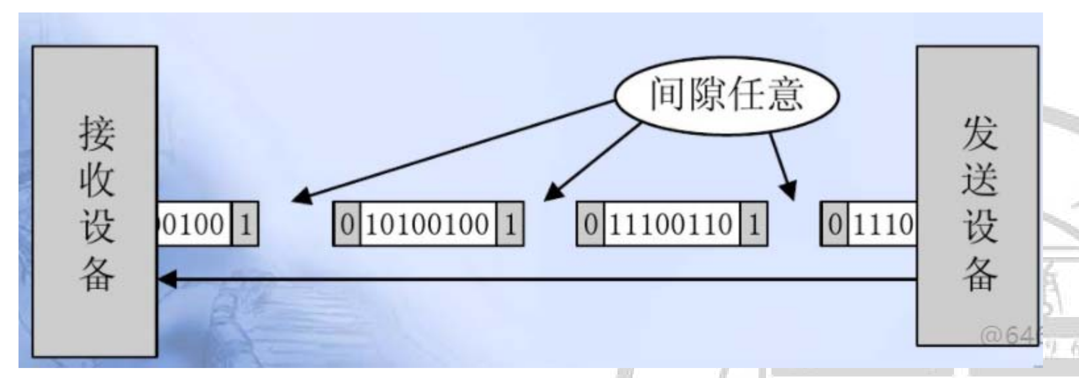
4.2.4 异步串口通信编程实例

❖ 串行通信方式

- 同步通信，要求发送和接收双方时钟同步
- 异步通信：不要求发送和接收双方时钟同步，但是需要附加2~3位起止位，各帧间需要有间隔



同步通信

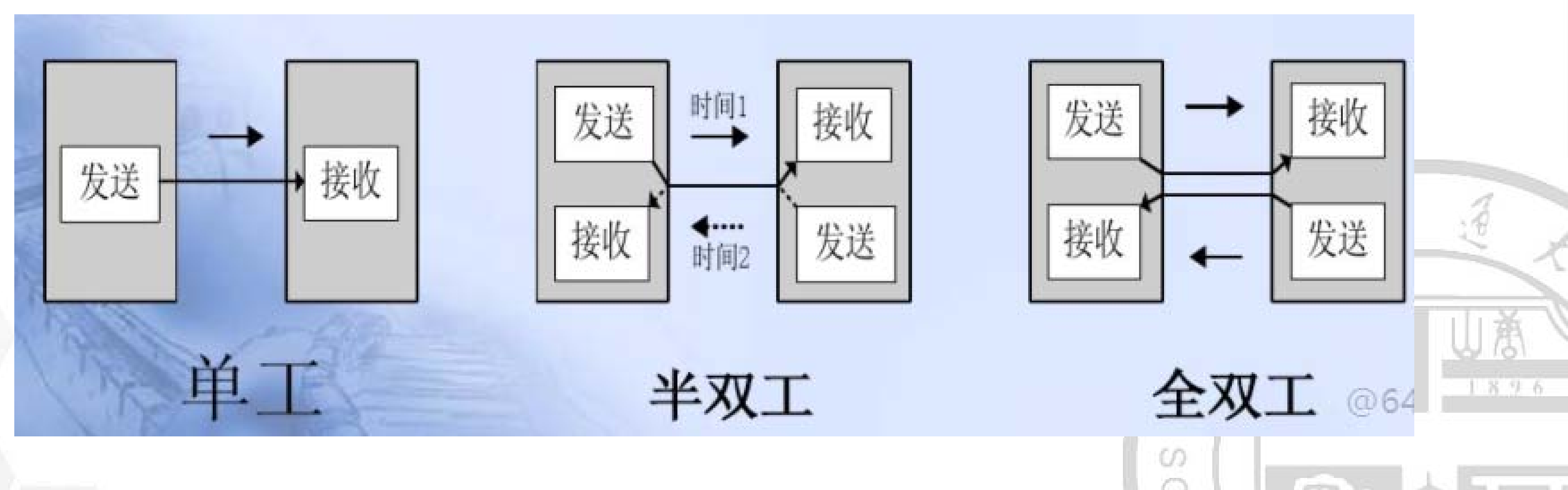


异步通信

4.2.4 异步串口通信编程实例

❖ 串行通信数据传输方向

- 单工：只支持数据在一个方向上传输
- 半双工：允许数据在两个方向上传输，但是需要分时进行
- 全双工：允许数据同时在两个方向上传输



4.2.4 异步串口通信编程实例

❖ 常见的串行通信接口

通信标准	引脚说明	通信方式	通信方向
UART (通用异步收发器)	TXD: 发送端 RXD: 接受端 GND: 公共地	异步通信	全双工
单总线 （1-wire)	DQ: 发送/接受端	异步通信	半双工
SPI	SCK: 同步时钟 MISO: 主机输入，从机输出 MOSI: 主机输出，从机输入	同步通信	全双工
I2C	SCL: 同步时钟 SDA: 数据输入/输出端	同步通信	半双工

4.2.4 异步串口通信编程实例

2、STM32F407串行通信接口

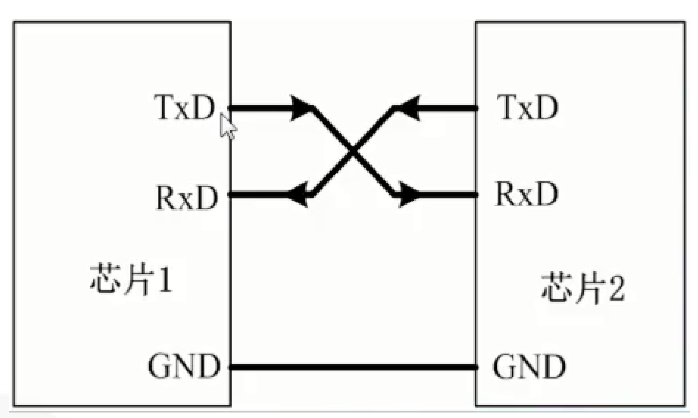
❖ 通用同步异步收发器 (USART) 特点

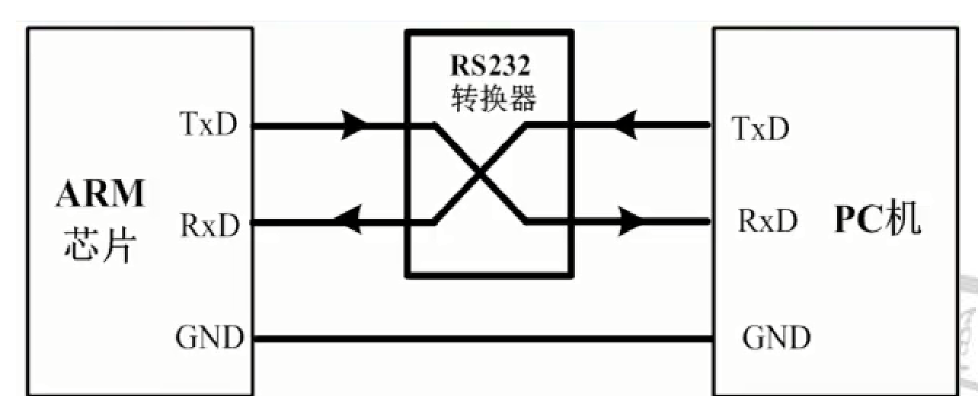
- 全双工异步通信
- 同步单向通信和半双工单线通信
- 数据字长度可编程（8 位或 9 位）
- 停止位可配置 - 支持 1 或 2 个停止位
- 校验控制，四个错误检测标志
- 触发中断
- 支持6个串口

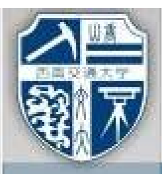
4.2.4 异步串口通信编程实例

❖ UART异步通信硬件连接

- RXD: 数据输入引脚，接受数据。
- TXD: 数据发送引脚，发送数据。







4.2.4 异步串口通信编程实例

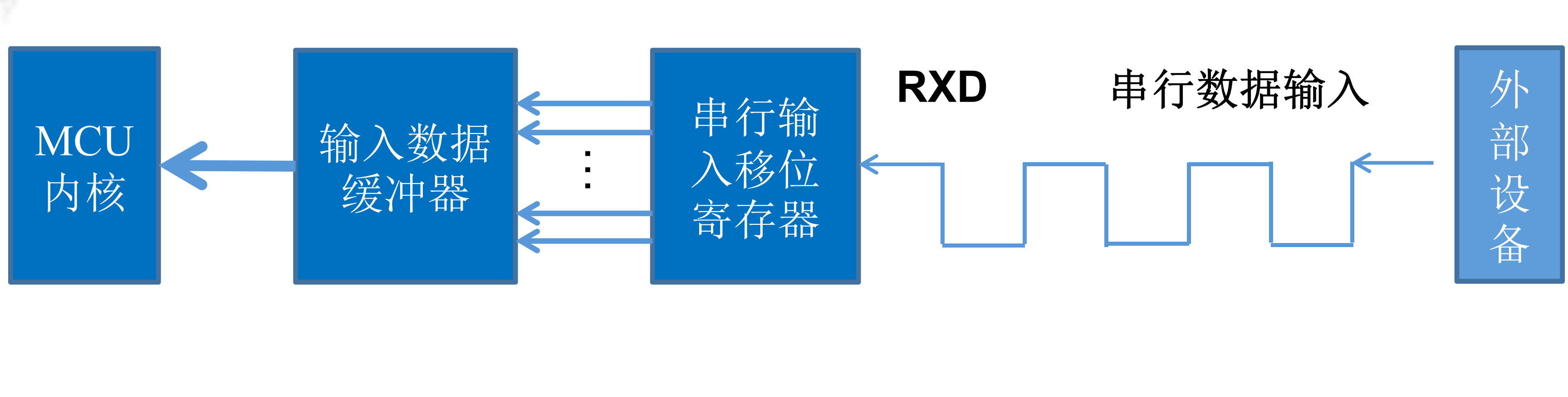
❖UART异步通信引脚(STM32F407)

串口号	RXD	TXD
1	PA10(PB7)	PA9（PB6)
2	PA3(PD6)	PA2(PD5)
3	PB11(PC11/PD9)	PB10(PC10/PD8)
4	PC11(PA1)	PC10(PA0)
5	PD2	PC12
6	PC7(PG9)	PC6(PG14)

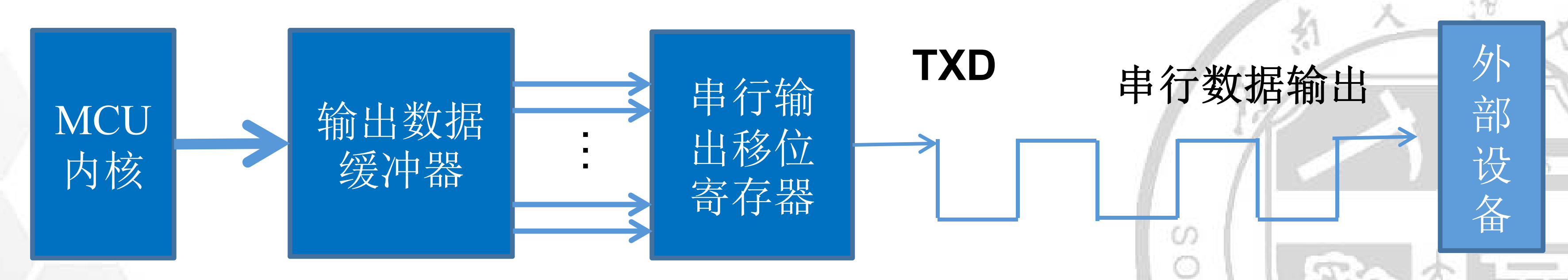
4.2.4 异步串口通信编程实例

❖ **STM32F407串口通信过程**

数据接收过程：



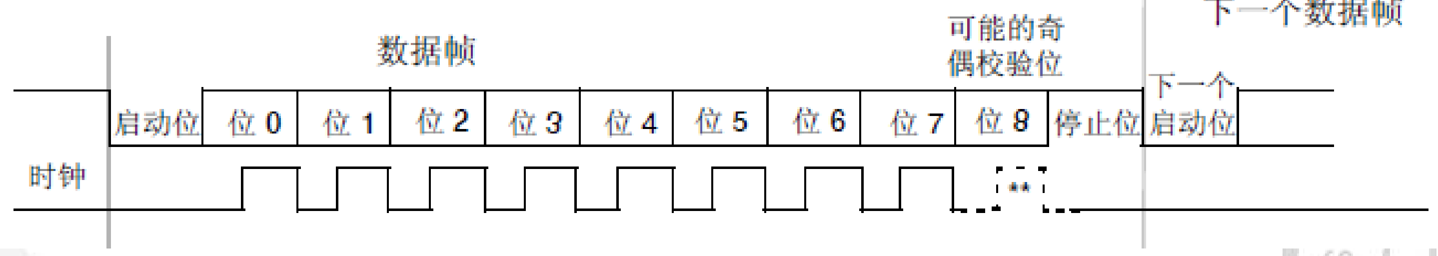
数据发送过程：



4.2.4 异步串口通信编程实例

❖ **STM32**串口异步通信需要定义的参数

- 起始位
- 数据位（8位或者9位）
- 奇偶校验位（第9位）
- 停止位（1或2位）
- 波特率设置



4.2.4 异步串口通信编程实例

❖ STM32串口异步通信寄存器

- 状态寄存器 (USART_SR)
- 数据寄存器 (USART_DR)
- 波特率寄存器 (USART_BRR)
- 控制寄存器 1(USART_CR1~3)
- 保护时间和预分频器寄存器 (USART_GTPR)

4.2.4 异步串口通信编程实例

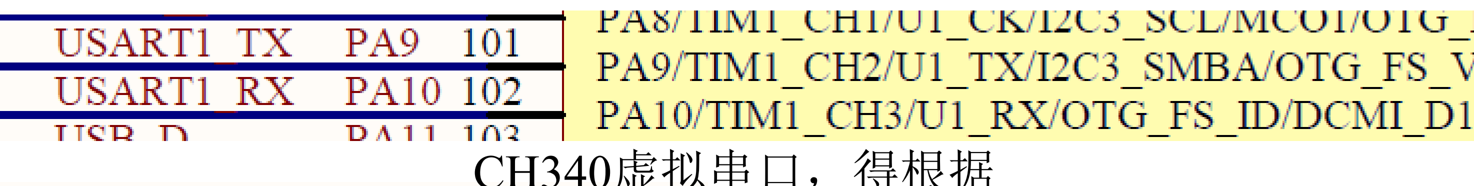
■3、STM32F407串行通信接口配置步骤

- 串口时钟使能
- GPIO时钟使能
- 引脚复用功能配置
- GPIO端口模式设置
- 串口参数初始化
- 开启中断并且初始化NVIC（使用中断才需要这个步骤）
- 使能串口
- 编写中断处理函数
- 串口数据收发
- 串口传输状态获取

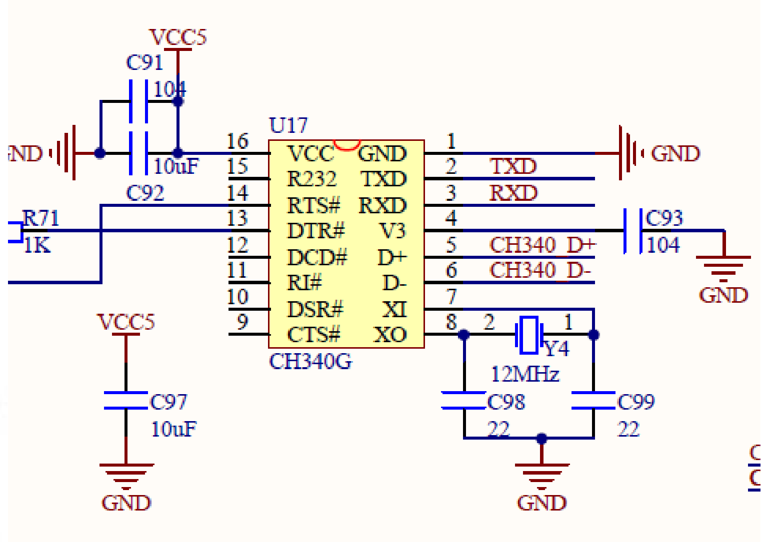
4.2.4 异步串口通信编程实例

4、STM32F407串行通信接口编程例子

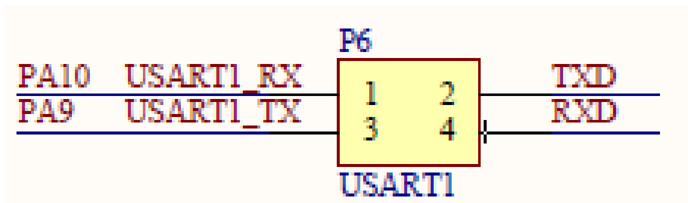
❖ 串口1的原理图



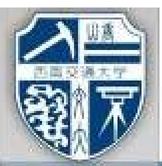
CH340虚拟串口，得根据



USB转串口电路



- **PA9：串口1发送**
- **PA10：串口1的接受**



4.2.4 异步串口通信编程实例

❖ 串口1通信流程图和程序

```
int main(void)
{
    u8 t;
    u8 len;
    u16 times=0;
    Stm32_Clock_Init(336,8,2,7);//设置时钟,168Mhz
    delay_init(168);    //延时初始化
    //端口A配置和复用功能配置
    GPIO_Set(GPIOA,PIN9|PIN10,GPIO_MODE_AF,GPIO_OTYPE_PP,GPIO_SPEED_50M,GPIO_PUPD_PU);//PA9,PA10,复用功能,上拉输出
    GPIO_AF_Set(GPIOA,9,7); //PA9,AF7
    GPIO_AF_Set(GPIOA,10,7);//PA10,AF7
    uart_init(84,115200); //串口初始化为115200
    while(1)
    {
        if(USART_RX_STA&0x8000)
        {
            len=USART_RX_STA&0x3fff;//得到此次接收到的数据长度
            printf("\r\n您发送的消息为:\r\n");
            for(t=0;t<len;t++)
            {
                USART1->DR=USART_RX_BUF[t];
                while((USART1->SR&0X40)==0);//等待发送结束
            }
            USART_RX_STA=0;
        }else
        {
            times++;
            if(times%200==0)printf("请输入数据,以回车键结束\r\n");
            delay_ms(10);
        }
    }
}
```

4.2.4 异步串口通信编程实例

```
//串口1中断服务程序
u8 USART_RX_BUF[USART_REC_LEN];     //接收缓冲,最大USART_REC_LEN个字节.
//接收状态, //bit15,  接收完成标志, //bit14,  接收到0x0d, //bit13~0,  接收到的有效字节数目
u16 USART_RX_STA=0;       //接收状态标记

void USART1_IRQHandler(void)
{
  u8 res;
  if(USART1->SR&(1<<5))//接收到数据
  {
    res=USART1->DR;
    if((USART_RX_STA&0x8000)==0)//接收未完成
    {
      if(USART_RX_STA&0x4000)//接收到了0x0d
      {
        if(res!=0x0a)USART_RX_STA=0;//接收错误,重新开始
        else USART_RX_STA|=0x8000;  //接收完成了
      }else //还没收到0X0D
      {
        if(res==0x0d)USART_RX_STA|=0x4000;
        else
        {
          USART_RX_BUF[USART_RX_STA&0X3FFF]=res;
          USART_RX_STA++;
          if(USART_RX_STA>(USART_REC_LEN-1))USART_RX_STA=0;//接收数据错误,重新开始接收
        }
      }
    }
  }

}
```

使用串口1的中断来接收数据

接受完成的数据放在USART_RX_BUF中

接收标志全局变量USART_RX_STA

4.2.4 异步串口通信编程实例

❖ 串口初始化程序

$$USARTDIV = DIV_Mantissa + (DIV_Fraction / 8 \times (2 - OVER8))$$

```
// 串口1初始化
//pclk2:PCLK2时钟频率(Mhz)
//bound:波特率
void uart_init(u32 pclk2,u32 bound)
{
  float temp;
  u16 mantissa;
  u16 fraction;
  temp=(float)(pclk2*1000000)/(bound*16);//得到USARTDIV@OVER8=0
  mantissa=temp;              //得到整数部分
  fraction=(temp-mantissa)*16; //得到小数部分@OVER8=0
    mantissa<<=4;
  mantissa+=fraction;
  RCC->AHB1ENR|=1<<0;      //使能PORTA口时钟
  RCC->APB2ENR|=1<<4;    //使能串口1时钟
  //波特率设置
  USART1->BRR=mantissa;    //波特率设置
  USART1->CR1&=~(1<<15);   //设置OVER8=0
  USART1->CR1|=1<<3;     //串口发送使能
  //使能接收中断
  USART1->CR1|=1<<2;     //串口接收使能
  USART1->CR1|=1<<5;      //接收缓冲区非空中断使能
  MY_NVIC_Init(3,3,USART1_IRQn,2);//组2，最低优先级
}
```

计算波特率寄存器(USART_BRR)中DIV中尾数和小数部分

配置串口1中断

4.2.4 异步串口通信编程实例

❖配置串口1中断

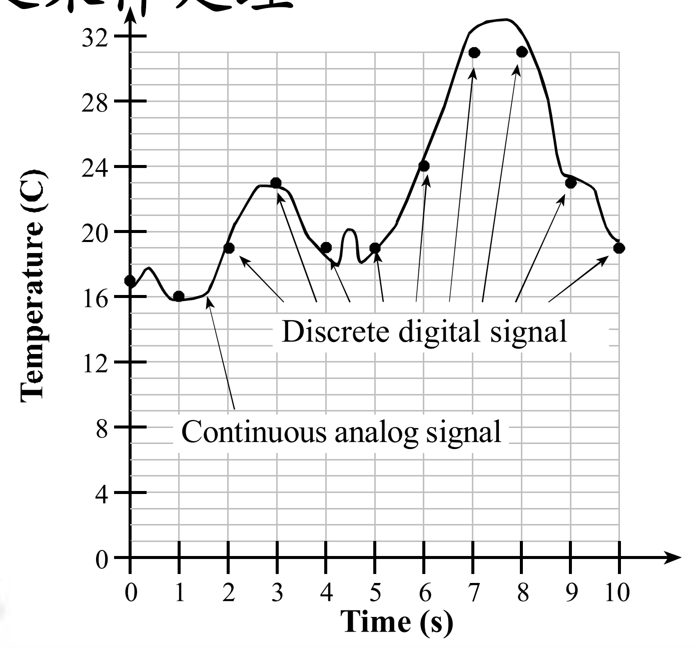
```
//设置NVIC
//NVIC_PreemptionPriority:抢占优先级
//NVIC_SubPriority       :响应优先级
//NVIC_Channel           :中断编号
//NVIC_Group             :中断分组 0~4
//注意优先级不能超过设定的组的范围!否则会有意想不到的错误
//组划分:
//组0:0位抢占优先级,4位响应优先级
//组1:1位抢占优先级,3位响应优先级
//组2:2位抢占优先级,2位响应优先级
//组3:3位抢占优先级,1位响应优先级
//组4:4位抢占优先级,0位响应优先级
//NVIC_SubPriority和NVIC_PreemptionPriority的原则是,数值越小,越优先
void MY_NVIC_Init(u8 NVIC_PreemptionPriority,u8 NVIC_SubPriority,u8 NVIC_Channel,u8 NVIC_Group)
{
  u32 temp;
  MY_NVIC_PriorityGroupConfig(NVIC_Group);//设置分组
  temp=NVIC_PreemptionPriority<<(4-NVIC_Group);
  temp|=NVIC_SubPriority&(0x0f>>NVIC_Group);
  temp&=0xf;                    //取低四位
  NVIC->ISER[NVIC_Channel/32]|=1<<NVIC_Channel%32;//使能中断位(要清除的话,设置ICER对应位为1即可)
  NVIC->IP[NVIC_Channel]|=temp<<4;          //设置响应优先级和抢断优先级
}
```

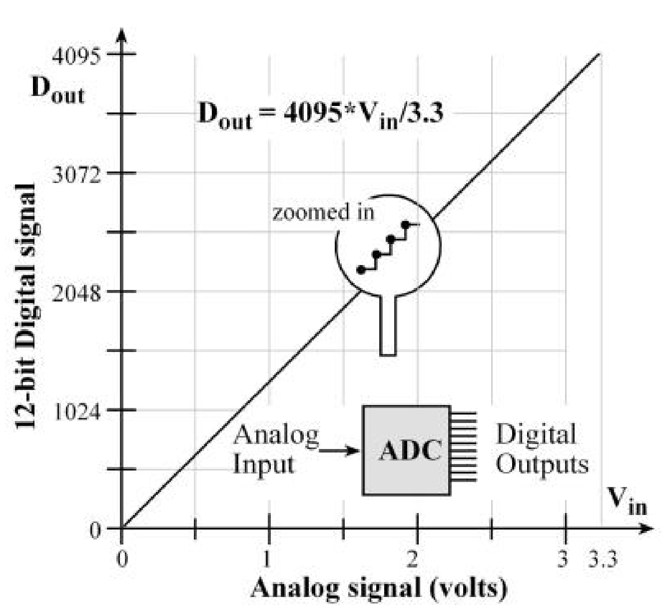
4.2.5 ADC编程实例

■1、ADC基本原理

❖ADC将模拟量转换为数字量

- 采样、保持、量化和编码
- 满足采样定理





4.2.5 ADC编程实例

❖ ADC技术参数

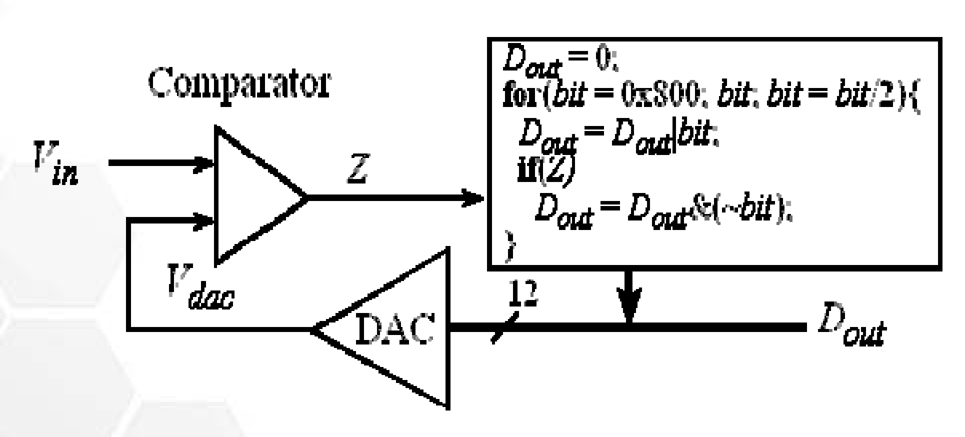
- 精度（位数）： 12 bits → 4096
- 输入电压范围： 0 to 3.3V
- 分辨率： 最小可检测的电压变化Δv

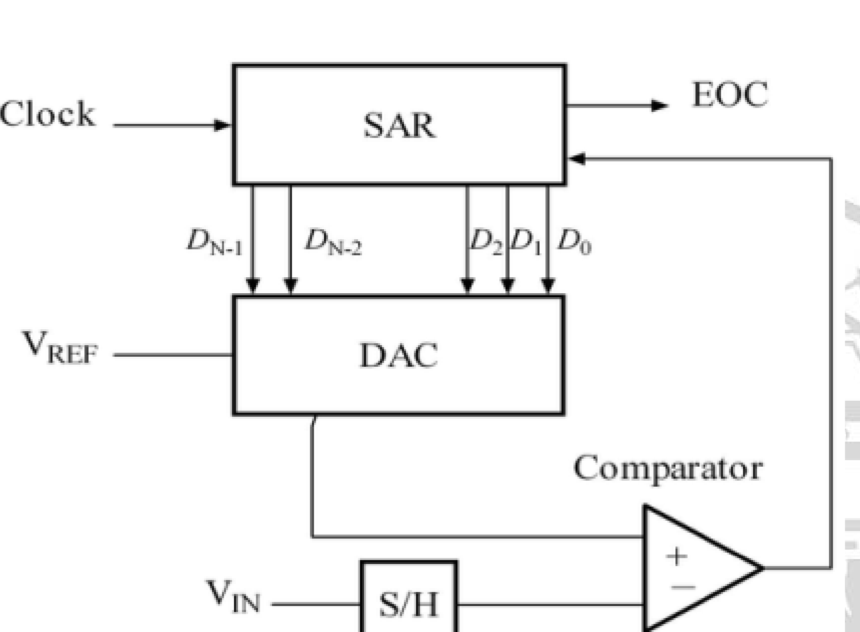
Δv = 电压范围/精度 = 3.3V-0V/4096 = 0.81mV

4.2.5 ADC编程实例

❖ 逐次趋近型ADC工作过程

- 逐次比较ADC由控制电路、数码寄存器、D/A转换器和电压比较器组成
- V_{IN} 为采样保持电路的输出
- SAR为一个计数器
- DAC的输出等于V_{IN}





4.2.5 ADC编程实例

2、STM32F407的ADC特点

- 12位ADC是逐次趋近型模数转换器
- 可配置12位、10位、8位或6位分辨率
- 单次和连续转换模式
- 转换结束产生中断
- 可独立设置各通道采样时间
- 数据可以左对齐或右对齐

4.2.5 ADC编程实例

- STM32F407有3个ADC控制器
  - 最多有24个外部通道
  - 最少有16个外部通道
- ADC通道选择
  - 一个规则转换组
  - 一个注入转换组

通道号	ADC1	ADC2	ADC3
通道0	PA0	PA0	PA0
通道1	PA1	PA1	PA1
通道2	PA2	PA2	PA2
通道3	PA3	PA3	PA3
通道4	PA4	PA4	PF6
通道5	PA5	PA5	PF7
通道6	PA6	PA6	PF8
通道7	PA7	PA7	PF9
通道8	PB0	PB0	PF10
通道9	PB1	PB1	PF3
通道10	PC0	PC0	PC0
通道11	PC1	PC1	PC1
通道12	PC2	PC2	PC2
通道13	PC13	PC13	PC13
通道14	PC4	PC4	PF4
通道15	PC5	PC5	PF5

4.2.5 ADC编程实例

❖ ADC引脚

表 48. ADC 引脚

名称	信号类型	备注
V_{REF+}	正模拟参考电压输入	ADC 高/正参考电压，1.8 V ≤ V_{REF+} ≤ V_{DDA}
V_{DDA}	模拟电源输入	模拟电源电压等于 V_{DD}， 全速运行时，2.4 V ≤ V_{DDA} ≤ V_{DD} (3.6 V) 低速运行时，1.8 V ≤ V_{DDA} ≤ V_{DD} (3.6 V)
V_{REF-}	负模拟参考电压输入	ADC 低/负参考电压，V_{REF-} = V_{SSA}
V_{SSA}	模拟电源接地输入	模拟电源接地电压等于 V_{SS}
ADCx_IN[15:0]	模拟输入信号	16 个模拟输入通道

❖ ADC时钟

- 用于模拟电路的时钟：ADCCLK，
- 用于数字接口的时钟：APB2

4.2.5 ADC编程实例

❖ ADC的采样时间

- 每个通道均可使用不同的采样时间进行采样
- 总转换时间的计算公式

  T_{conv} = 采样时间 + 12 个周期

示例：

ADCCLK = 30 MHz 且采样时间 = 3 个周期时：

T_{conv} = 3 + 12 = 15 个周期 = 0.5 μs（APB2 为 60 MHz 时）

4.2.5 ADC编程实例

❖ADC 寄存器

- ADC 状态寄存器 (ADC_SR)
- ADC 控制寄存器 (ADC_CR1~2)
- ADC 采样时间寄存器 (ADC_SMPR1~2)
- ADC 注入通道数据偏移寄存器 (ADC_JOFR1~4)
- ADC 看门狗高/低阈值寄存器 (ADC_HTR/ ADC_LTR)
- ADC 规则序列寄存器 (ADC_SQR1~3)
- ADC 注入序列寄存器 (ADC_JSQR)
- ADC 注入数据寄存器 (ADC_JDR1~4)
- ADC 规则数据寄存器 (ADC_DR)
- ADC 通用状态寄存器 (ADC_CSR)
- ADC 通用控制寄存器 (ADC_CCR)

4.2.5 ADC编程实例

4、STM32F407的ADC配置步骤

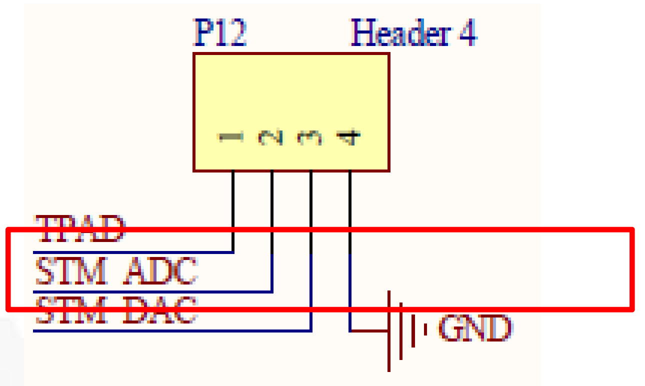
- 开启输入端口时钟
- 使能ADCx的时钟，配置分频因子
- 设置ADCx的工作模式
- 设置ADC1规则序列
- 开启AD转换器
- 读取ADC值

4.2.5 ADC编程实例

■5、ADC1通道5的AD转换（PA5引脚）

❖ ADC1通道5的ADC原理图

STM_ADC PA5 41 PA5/SPI1_SCK/OTG_HS_ULPI_CK/TIM2_CH1_ETR/
DCMI_PCLK PA6 42 PA6/SPI1_MISO/TIM1_BKIN/TIM3_CH1/TIM8_BKIN
PA7 43 PA7/SPI1_MOSI/TIM1_CH1N/TIM3_CH2/TIM8_CH1



- **PA5：ADC输入引脚**
- **0~3.3v**

4.2.5 ADC编程实例

❖ ADC1通道5的ADC流程图和程序

```
#include "sys.h"
#include "delay.h"
#include "adc.h"

int main(void)
{
  u16 adcx;
  Stm32_Clock_Init(336,8,2,7);//设置时钟,168Mhz
  delay_init(168);        //延时初始化
  Adc_Init();           //初始化ADC
  while(1)
  {
    adcx=Get_Adc_Average(ADC_CH5,20);
    delay_ms(250);
  }
}
```

开始
STM32F407时钟设置
ADC初始化
While(1)
是
读取ADC数据

4.2.5 ADC编程实例

❖ ADC初始化程序

```
//初始化ADC
//规则通道为例
//开启ADC1_CH5
void  Adc_Init(void)
{
    //先初始化IO口
    RCC->APB2ENR|=1<<8;       //使能ADC1时钟
    RCC->AHB1ENR|=1<<0;       //使能PORTA时钟
    GPIO_Set(GPIOA,PIN5,GPIO_MODE_AIN,0,0,GPIO_PUPD_PU);   //PA5,模拟输入,下拉

    RCC->APB2RSTR|=1<<8;      //ADCs复位
    RCC->APB2RSTR&=~(1<<8); //复位结束
    ADC->CCR=3<<16;       //ADCCLK=PCLK2/4=84/4=21Mhz,ADC时钟最好不要超过36Mhz

    ADC1->CR1=0;          //CR1设置清零
    ADC1->CR2=0;          //CR2设置清零
    ADC1->CR1|=0<<24;         //12位模式
    ADC1->CR1|=0<<8;        //非扫描模式

    ADC1->CR2&=~(1<<1);       //单次转换模式
    ADC1->CR2&=~(1<<11);      //右对齐
    ADC1->CR2|=0<<28;       //软件触发

    ADC1->SQR1&=~(0XF<<20);
    ADC1->SQR1|=0<<20;        //1个转换在规则序列中 也就是只转换规则序列1
    //设置通道5的采样时间
    ADC1->SMPR2&=~(7<<(3*5));//通道5采样时间清空
    ADC1->SMPR2|=7<<(3*5);   //通道5  480个周期,提高采样时间可以提高精确度
    ADC1->CR2|=1<<0;        //开启AD转换器
}
```

4.2.5 ADC编程实例

```
//获得ADC值
//ch:通道值 0~16
//返回值:转换结果
u16 Get_Adc(u8 ch)
{
  //设置转换序列
  ADC1->SQR3&=0XFFFFFFE0;//规则序列1 通道ch
  ADC1->SQR3|=ch;
  ADC1->CR2|=1<<30;        //启动规则转换通道
  while(!(ADC1->SR&1<<1));//等待转换结束
  return ADC1->DR;      //返回adc值
}
```

启动ADC并读取转换值

```
//ch:通道编号
//times:获取次数
//返回值:通道ch的times次转换结果平均值
u16 Get_Adc_Average(u8 ch,u8 times)
{
  u32 temp_val=0;
  u8 t;
  for(t=0;t<times;t++)
  {
    temp_val+=Get_Adc(ch);
    delay_ms(5);
  }
  return temp_val/times;
}
```

作业

- 简述STM32F407的最小系统包括哪些基本单元
- 概述ST32F407的GPIO的特点、相关寄存器和初始化步骤
- 设计编写一个按键控制5个LED的原理图，并编写相应的程序。