电子科学与技术专业课

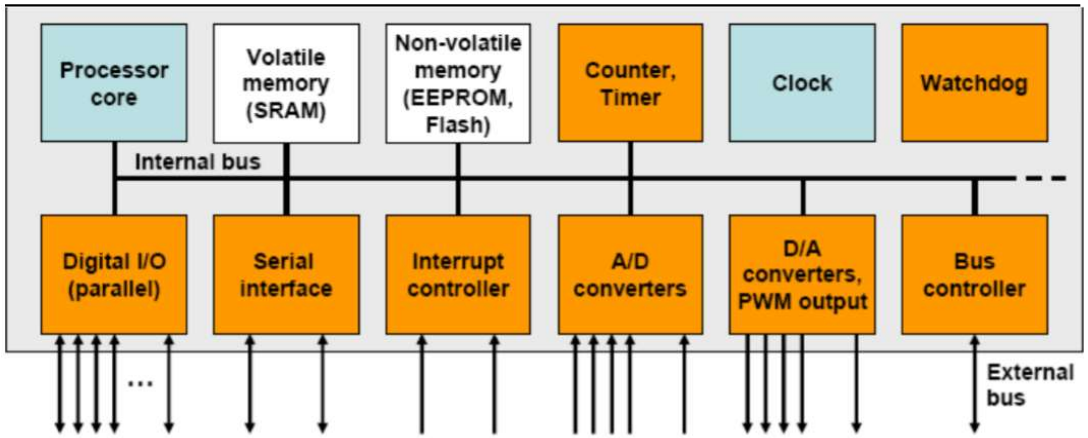
嵌入式系统

Embedded System

信息学院 光电子系

回顾 – 第2章MCU硬件基础

- **MCU – 完备计算机系统**
  - **CPU**
    -- 寄存器组、指令……
  - **存储器（Memory）**
    -- 代码、数据
    -- Flash、SRAM
  - **哈佛总线（Bus）**
    -- 指令总线、数据总线
    -- AHB、APB
  - **时钟树（ClockTree）**
    -- 系统时钟(最大)
    -- 外设时钟(各取所需)



  - **片上外设（Peripheral）**
    -- 负责与外部连接
    -- 寄存器地址映射到存储空间

第3章 STM32开发初步

— IO外设及软件基础

慕课视频内容

☆ **1、嵌入式开发的基本概念与工具链**（清华慕课 – 易）

—24min。嵌入式开发领域的基本概念以及嵌入式开发流程中所需的工具

☆ **2、嵌入式开发的进阶知识**（清华慕课- 易）

—20min。微控制器的启动过程，微控制器软件开发的流程，开发过程中的各种文件及文件之间的逻辑关系

☆ **3、基于Keil μ5的STM32开发过程及IO寄存器编程**（教师录制 - 中）

—30min。工程建立、点灯程序编写和下载的全过程

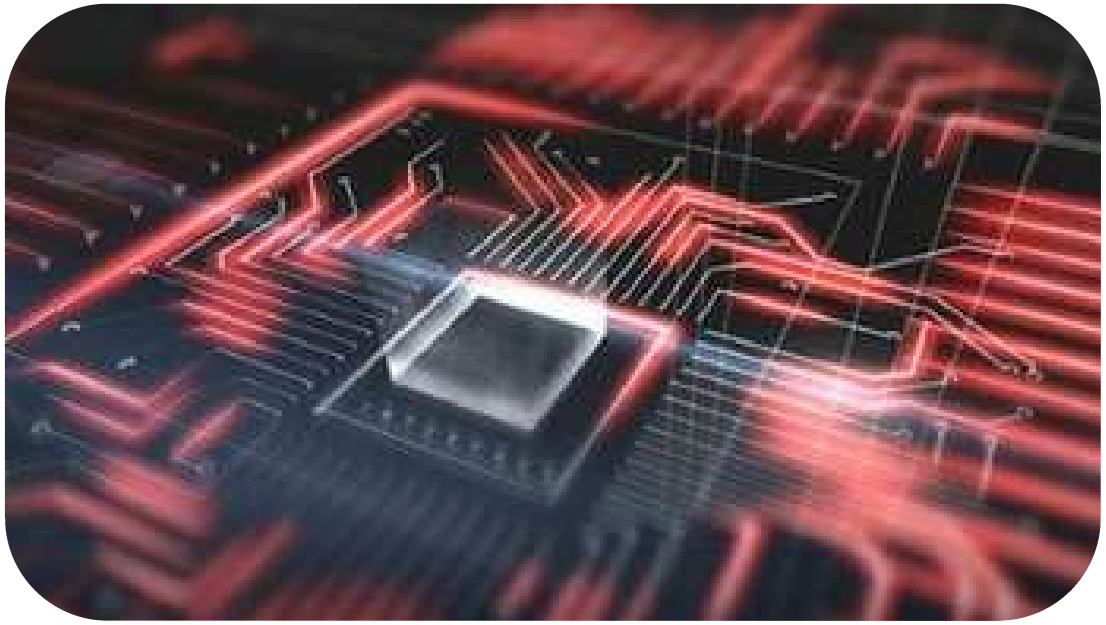
☆ **4、嵌入式开发中的C语言—上、下篇**（清华慕课 – 中、难）

—25+30min。上篇介绍C语言的历史渊源；C语言的数据类型及使用时的注意事项；位操作及其在嵌入式中的应用代码举例。下篇介绍强制类型转换的意义和volatile关键字；如何使用指针访问寄存器，读写特定地址；编写中断服务子程序的注意事项；程序执行时的起始代码以及微控制器C语言编程的特点

☆ **5、IO库函数编程实现按键控制LED**（教师录制 – 易）

—20min。标准库文件的添加，用户硬件库文件的建立与编写，在主程序(main.c)中调用函数

嵌入式系统(EMBEDDED SYSTEM)

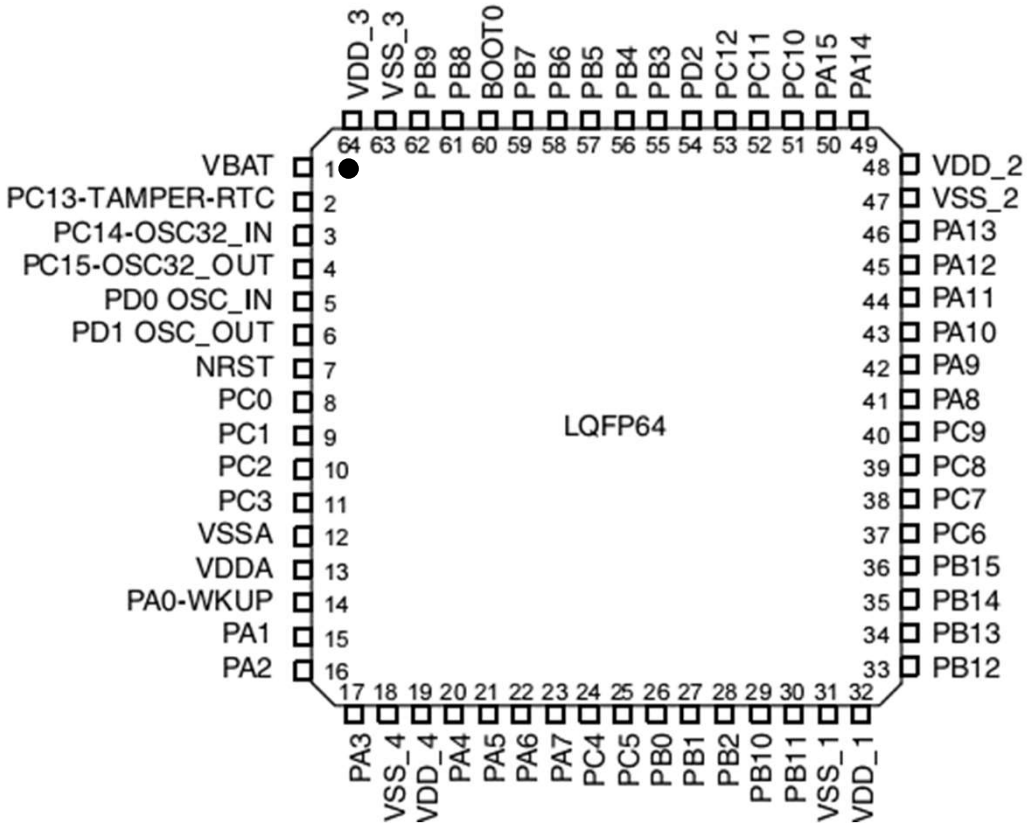


第3章 STM32开发初步

❀ **3.1 第一种外设GPIO**

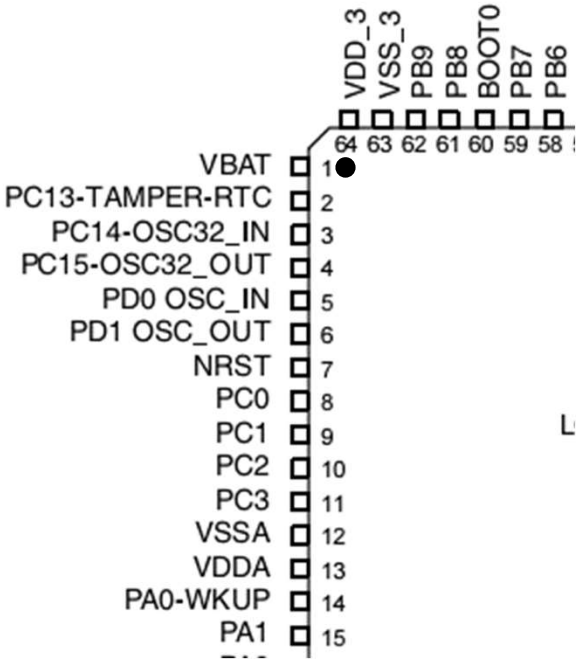
STM32F103R 引脚图

64-pin LQFP pinout diagram



STM32F103R 引脚图

局部



端口-PORT

PC0：PORT-C-0#

PC15：PORT-C-15#

.....

PA1：PORT-A-1#

PB9：PORT-B-9#

STM32F103x 引脚功能定义

Table 5. High-density STM32F103xC/D/E pin definitions

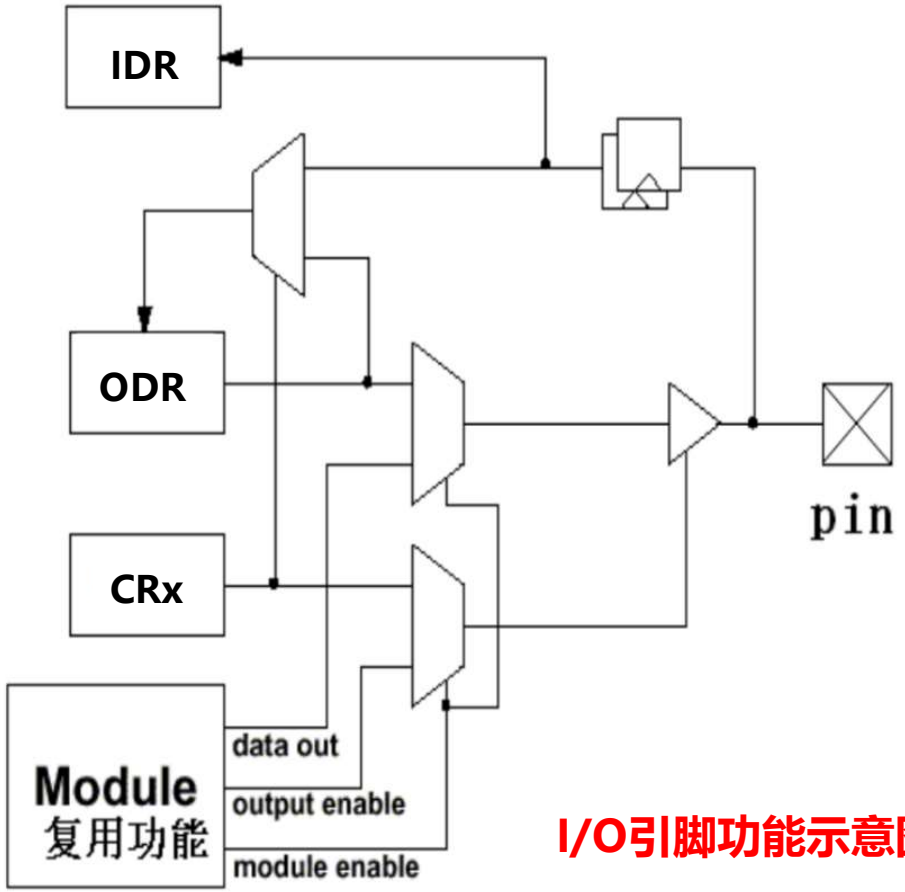
Pins						Pin name	Type(1)	I / O Level(2)	Main function(3) (after reset)	Alternate functions(4)	
LFBGA144	LFBGA100	WLCSP64	LQFP64	LQFP100	LQFP144				• 主功能	Default • 复用功能 节省成本	Remap
C2	B2	C6	1	6	6	V_{BAT}	S	-	V_{BAT}	-	-
A1	A2	C8	2	7	7	PC13-TAMPER-RTC(5)	I/O	-	PC13(6)	TAMPER-RTC	-
B1	A1	B8	3	8	8	PC14-OSC32_IN(5)	I/O	-	PC14(6)	OSC32_IN	-
C1	B1	B7	4	9	9	PC15-OSC32_OUT(5)	I/O	-	PC15(6)	OSC32_OUT	-

STM32F103x 引脚功能定义

- 更多复用功能

K2	H2	E6	15	24	35	PA1	I/O	-	PA1	USART2_RTS(9) ADC123_IN1/ TIM5_CH2/TIM2_CH2(9)	-
L2	J2	H8	16	25	36	PA2	I/O	-	PA2	USART2_TX(9)/TIM5_CH3 ADC123_IN2/ TIM2_CH3 (9)	-
M2	K2	G7	17	26	37	PA3	I/O	-	PA3	USART2_RX(9)/TIM5_CH4 ADC123_IN3/TIM2_CH4(9)	-

Digital I/O pins



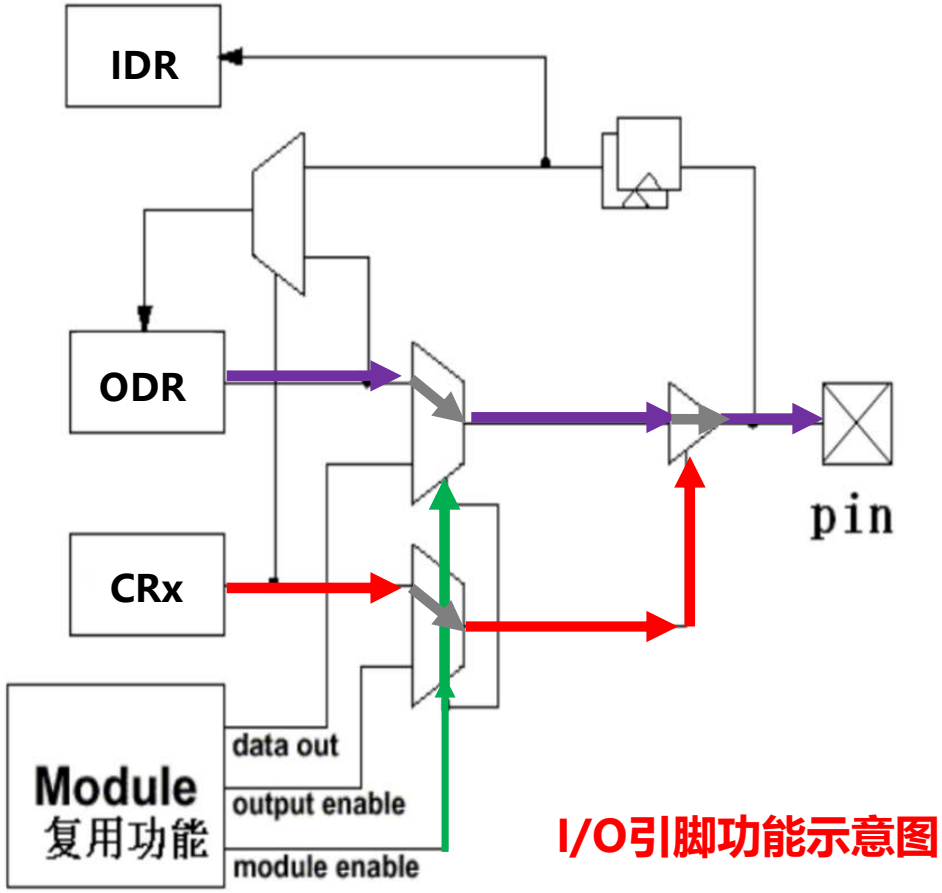
芯片外围引脚

I/O引脚功能示意图

Digital I/O

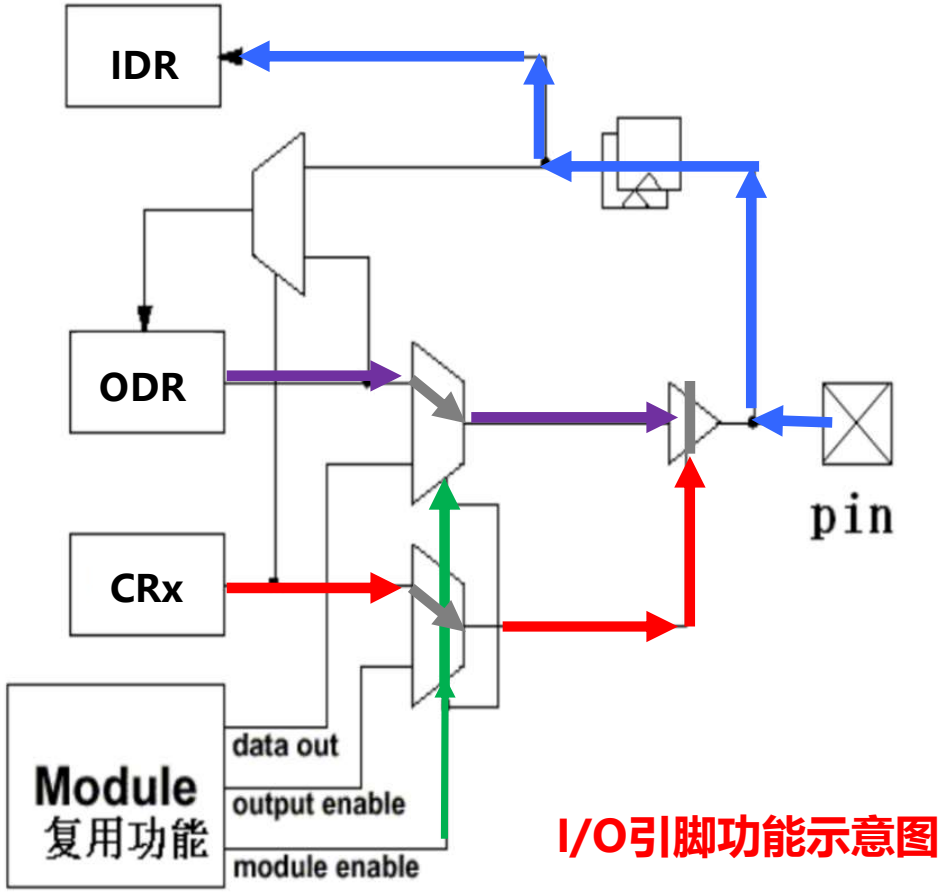
- IO是单片机和外部接口的最基本的手段
- 通常在微控制器中
  - 将8/16/32个IO口合成一组
  - IO通常是双向的（个别管脚是单向的）
  - IO引脚通常还和其他外设引脚复用
- IO相关基础寄存器
  - 端口(方向)配置寄存器（CRL,CRH）-Configuration Register
  - 端口数据输出寄存器（ODR）-Output Data Register
  - 端口数据输入寄存器（IDR） -Input Data Register
  - 端口其他寄存器（BSRRL, BSRRH）

IO Output – 输出演示



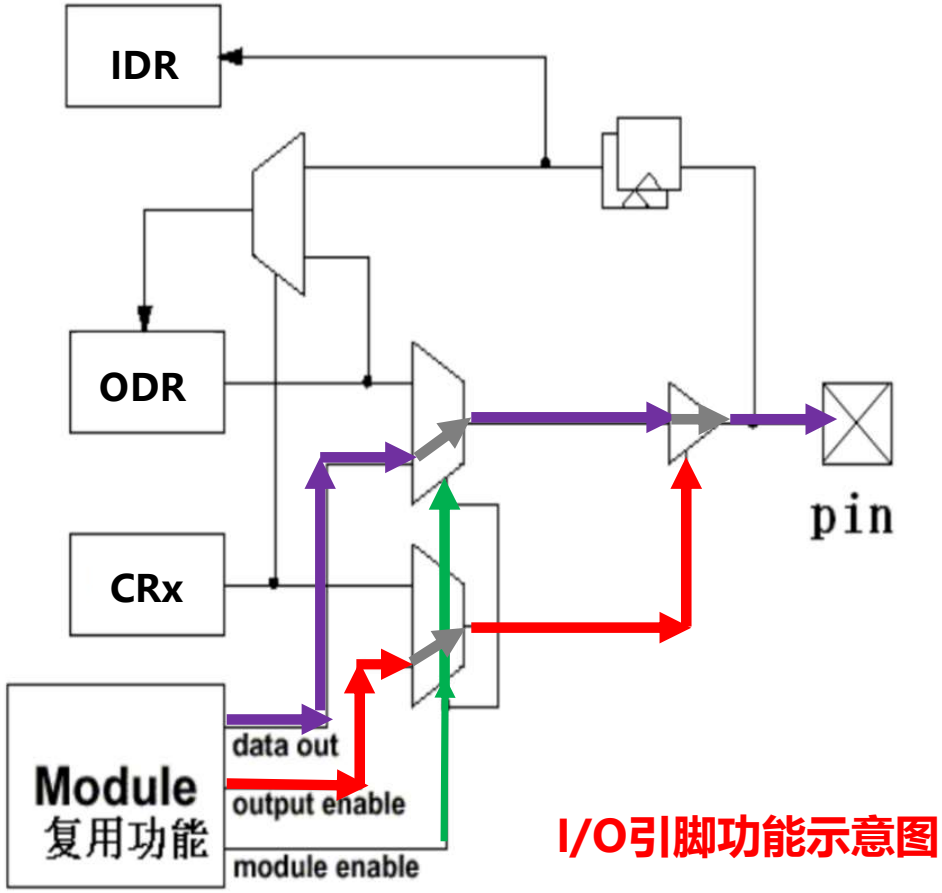
I/O引脚功能示意图

IO Input – 输入演示



I/O引脚功能示意图

IO shared function – 复用



I/O引脚功能示意图

IO Push Pull – 推挽(输出)

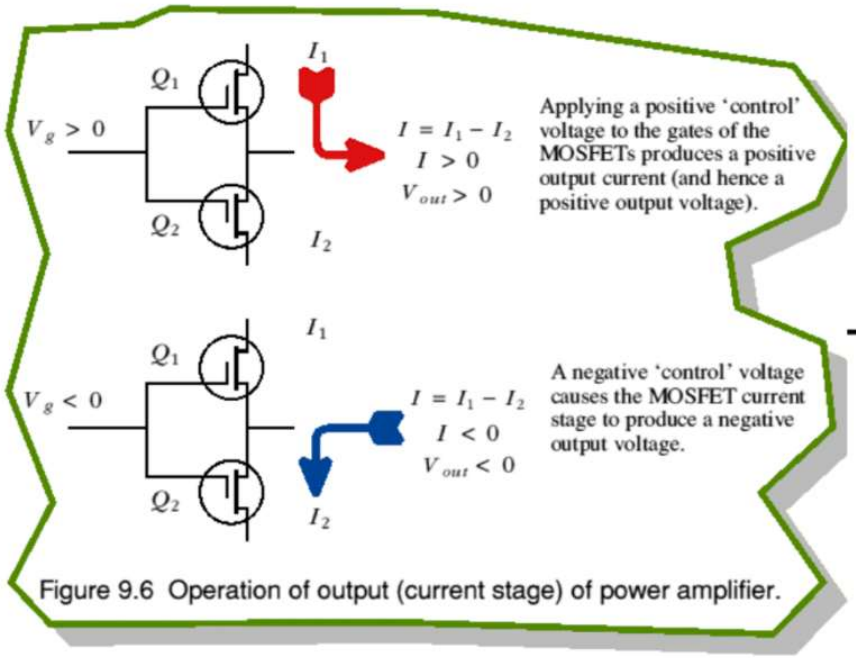
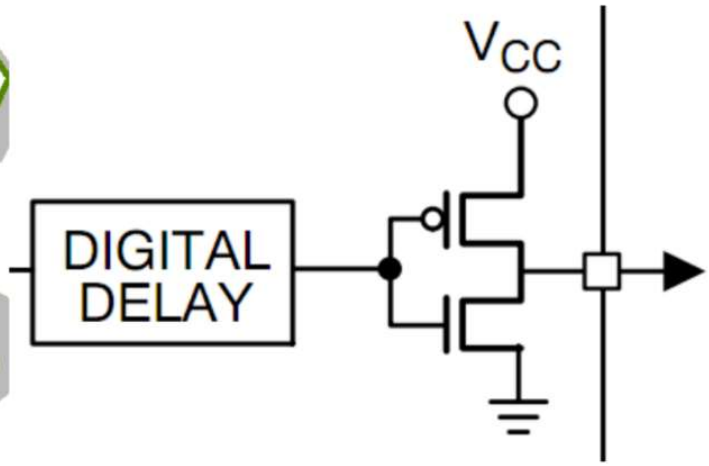


Figure 9.6 Operation of output (current stage) of power amplifier.



IO Current limit – 电流限制

Table 7. Voltage characteristics

Symbol	Ratings	Min	Max	Unit
V_{DD}–V_{SS}	External main supply voltage (including V_{DDA} and V_{DD})(1)	–0.3	4.0	V
V_{IN}(2)	Input voltage on five volt tolerant pin	V_{SS} –0.3	V_{DD} + 4.0	
	Input voltage on any other pin	V_{SS} –0.3	4.0	

Table 8. Current characteristics

Symbol	Ratings	Max.	Unit
I_{VDD}	Total current into V_{DD}/V_{DDA} power lines (source)(1)	150	mA
I_{VSS}	Total current out of V_{SS} ground lines (sink)(1)	150	
I_{IO}	Output current sunk by any I/O and control pin	25	
	Output current source by any I/Os and control pin	–25	

IO Registers – 格式

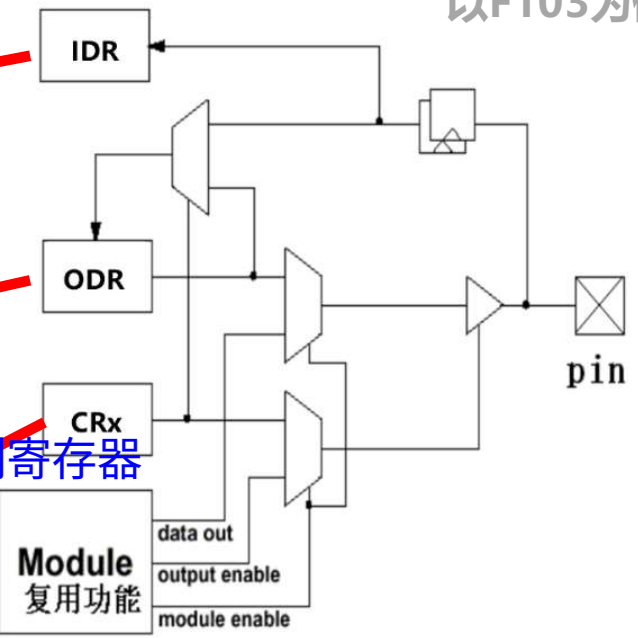
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[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
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
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
Reserved								EVOE	PORT[2:0]			PIN[3:0]			
								rw	rw	rw	rw	rw	rw	rw	rw

以F103为例



控制寄存器

IO模块也需要配置时钟

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
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

IO Registers – 描述

- 端口(方向)配置寄存器（CRL,CRH）
  - 可读/可写
  - 可控制每个IO的工作在输入还是输出模式
  - CRL控制IO的低8位脚，CRH控制IO的高8位脚
- 端口数据输出寄存器（ODR）
  - 可读/可写
  - 写操作，控制一组IO的输出高低电平状态
  - 读操作，获取一组IO的当前输出状态（回读）
- 端口数据输入寄存器（IDR）
  - 只读
  - 读取一组IO的当前输入值（外部输入）
  - 为啥需要IDR?

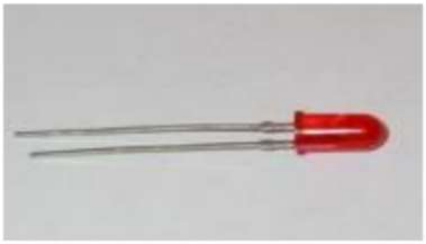
IO Registers – 描述

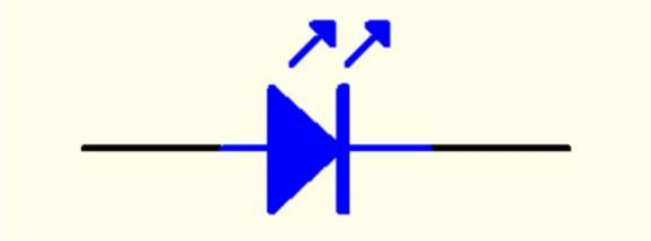
- 端口输出置位寄存器（BSRRL）
  - BSRR寄存器的低16位
  - 只能写入
  - 通过写入该寄存器的某个位，来改写ODR寄存器的对应位，进而控制每个IO引脚
    - 写0 – ODR位不变 – 引脚状态不变
    - 写1 – ODR位置1 – 引脚置位(1)
- 端口输出复位寄存器（BSRRH）
  - BSRR寄存器的高16位
  - 只能写入
  - 通过写入该寄存器的某个位，来改写ODR寄存器的对应位，进而控制每个IO引脚
    - 写0 – ODR位不变 – 引脚状态不变
    - 写1 – ODR位清0 – 引脚复位(0)

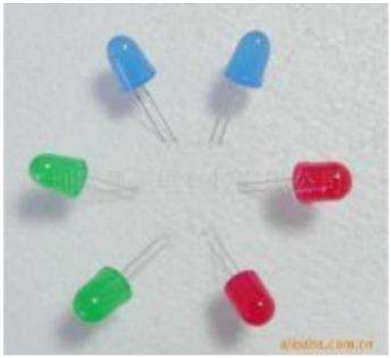
单独控制单个IO寄存器

IO应用 - LED

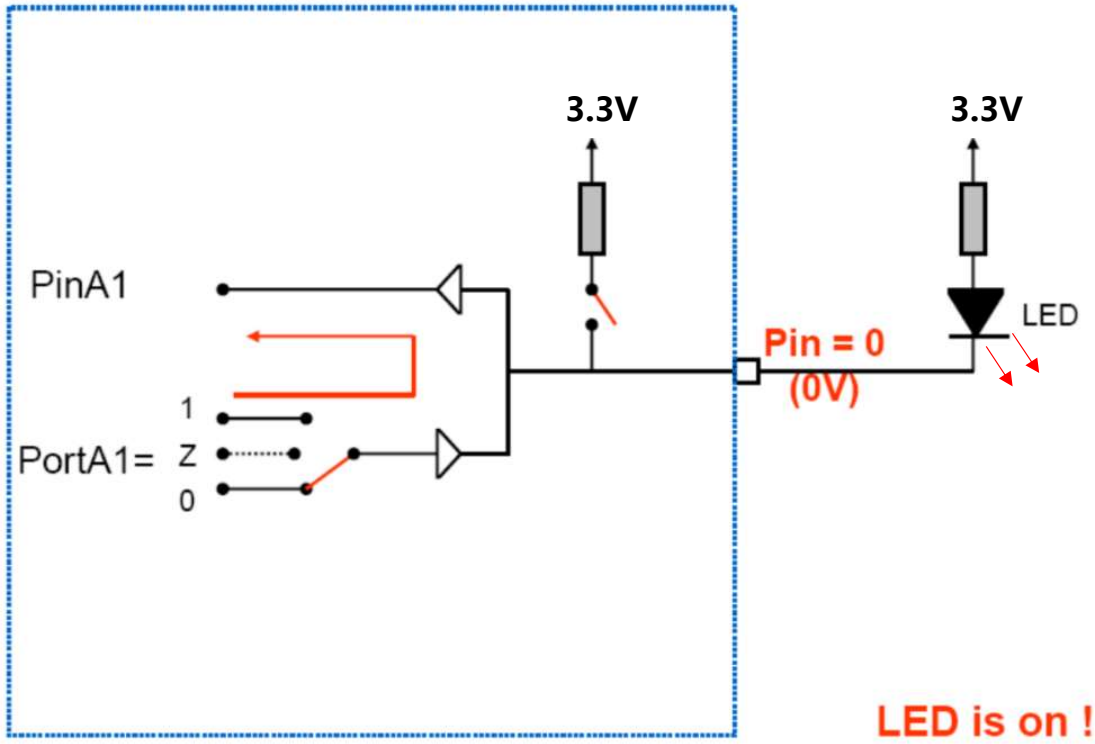
- LED（Light emitting diode/发光二极管）将电能转化为可见光
- PN节结构，正向导通发光，反向截止熄灭
- 多种颜色，多种尺寸，多种封装
- 流过的电流决定其亮度，需要使用限流电阻
- LED具有一定的响应速度



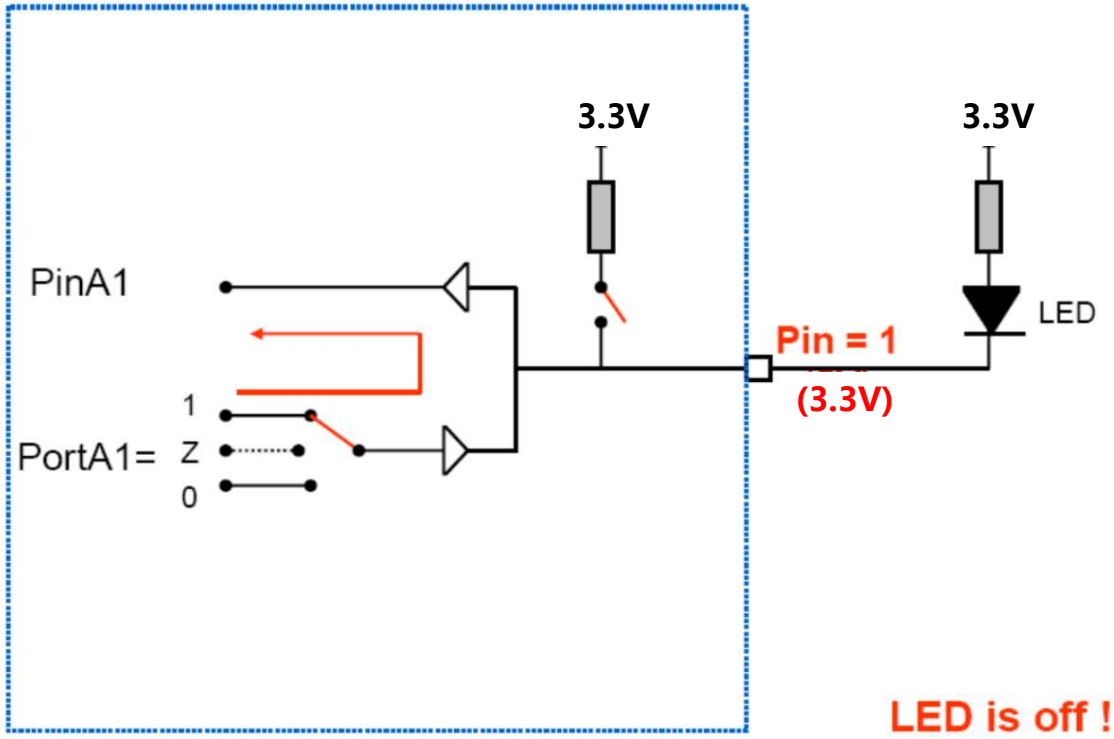




Turn on LED – 点亮

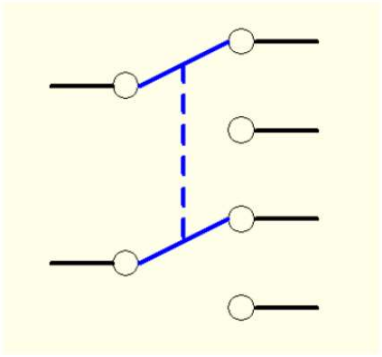


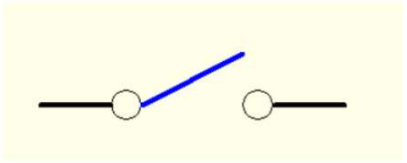
Turn off LED – 熄灭



IO应用 – 开关(switch)

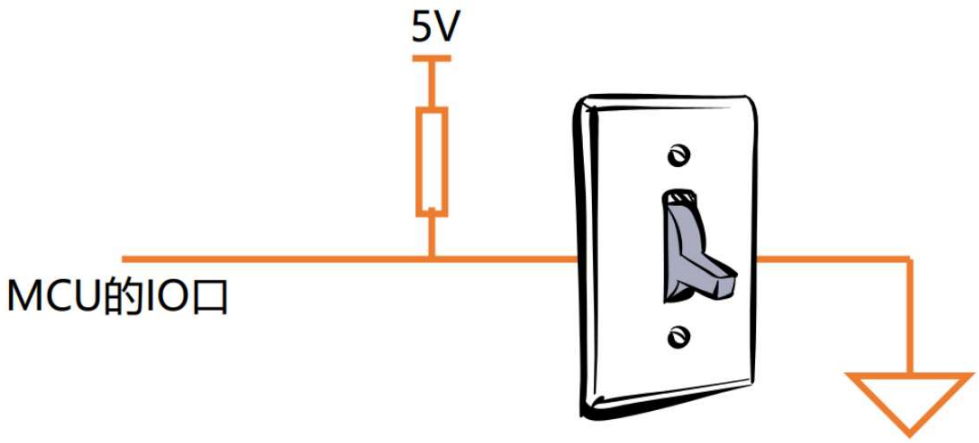
- 开关有两种状态 – 闭合和断开
- 船型开关，拨位开关
- 同一个开关可以提供多组触点





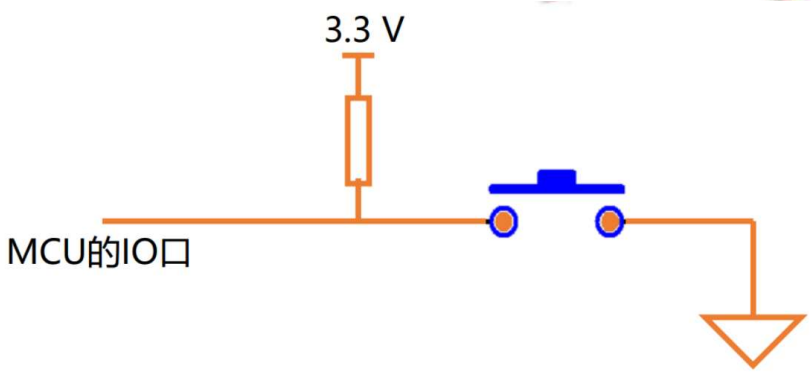
开关状态的读取

- 利用外部电路使开关的闭合和断开状态能得到两种不同的电平值
- 开关断开时要有默认电平值（上拉/下拉）



IO应用 – 按键

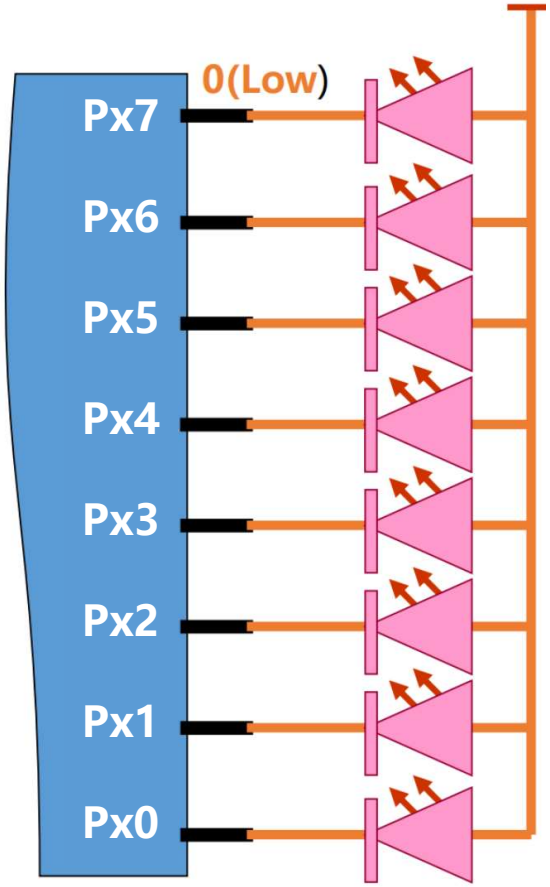
- 带有自恢复装置的开关，按一次产生一个脉冲
- 常开常闭型
- 各种形状，各种尺寸
- 耐高压，轻触式，带指示
- 一个按键也可以提供多组触点



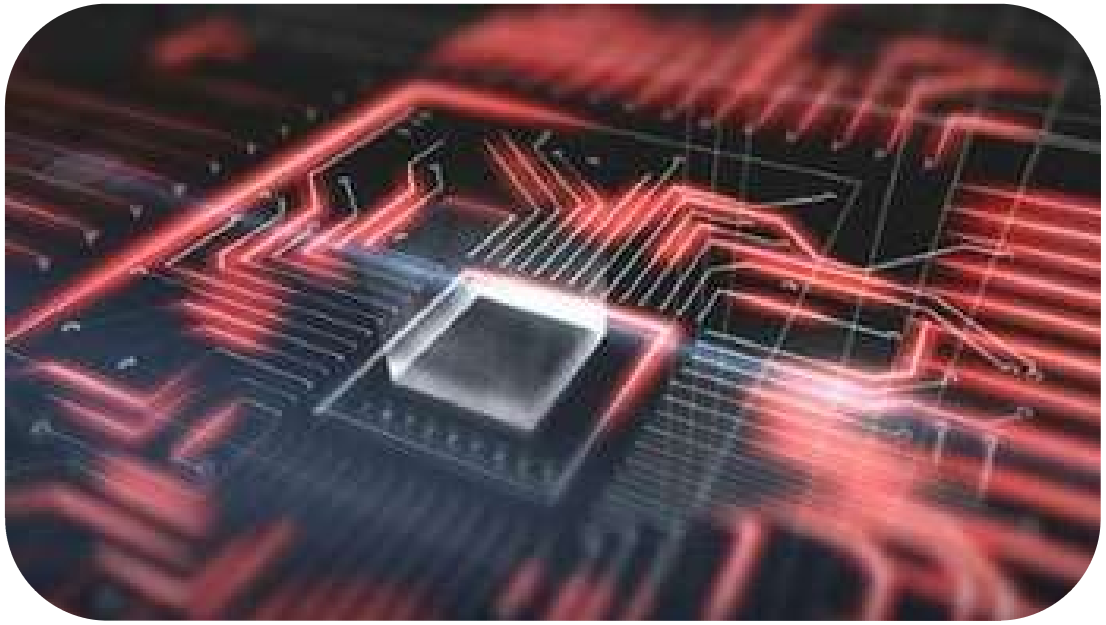
IO programming – 编程

```
main(void)
{
  ......
  ??


}
```



嵌入式系统(EMBEDDED SYSTEM)



第3章 STM32开发初步

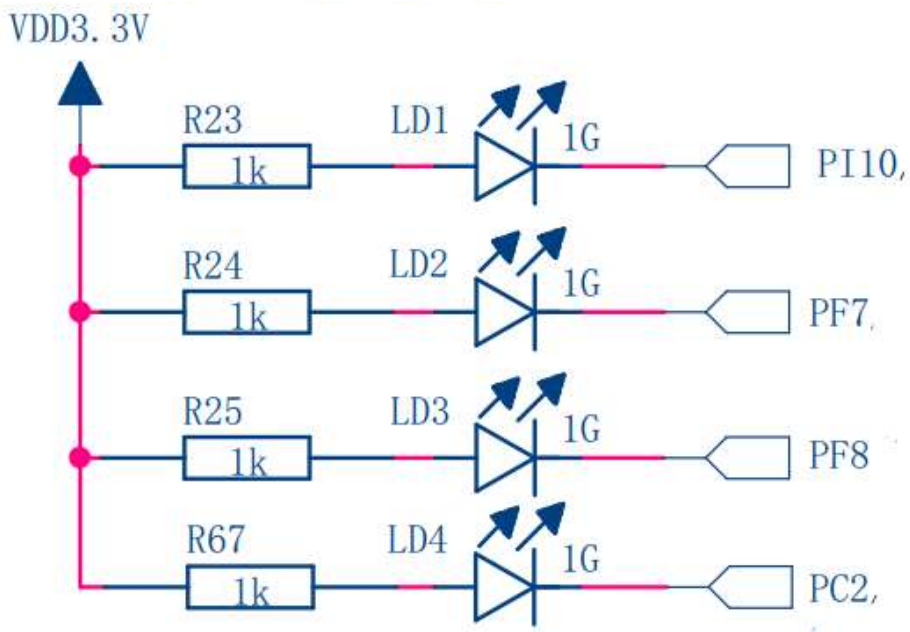
❀ **3.2 IO的寄存器编程**

IO programming – 编程

```
main(void)
{
  ......
  ??


}
```

F407开发板自定义LED指示灯



低电平点亮 高电平熄灭

GPIO编程三步走 – 输出

- Step 1：使能GPIO端口的时钟
- Step 2：将GPIO端口设置为通用IO及输出模式
- Step 3：设置ODR寄存器控制引脚输出0和1

F407 GPIO

- 详见F407的寄存器手册.pdf
Chapter 8 General-purpose IOs
8.4 GPIO registers

Control Registers：

GPIOx_MODER	**工作模式设置**
GPIOx_OTYPER	输出方式设置
GPIOx_OSPEEDR	输出速度设置
GPIOx_PUPDR	上拉/下拉电阻设置
GPIOx_IDR	端口输入数据
GPIOx_ODR	**端口输出数据**
GPIOx_BSRRL	**单引脚置位(1)**
GPIOx_BSRRH	**单引脚复位(0)**

GPIO 寄存器在哪?

Figure 18. STM32F40xxx memory map

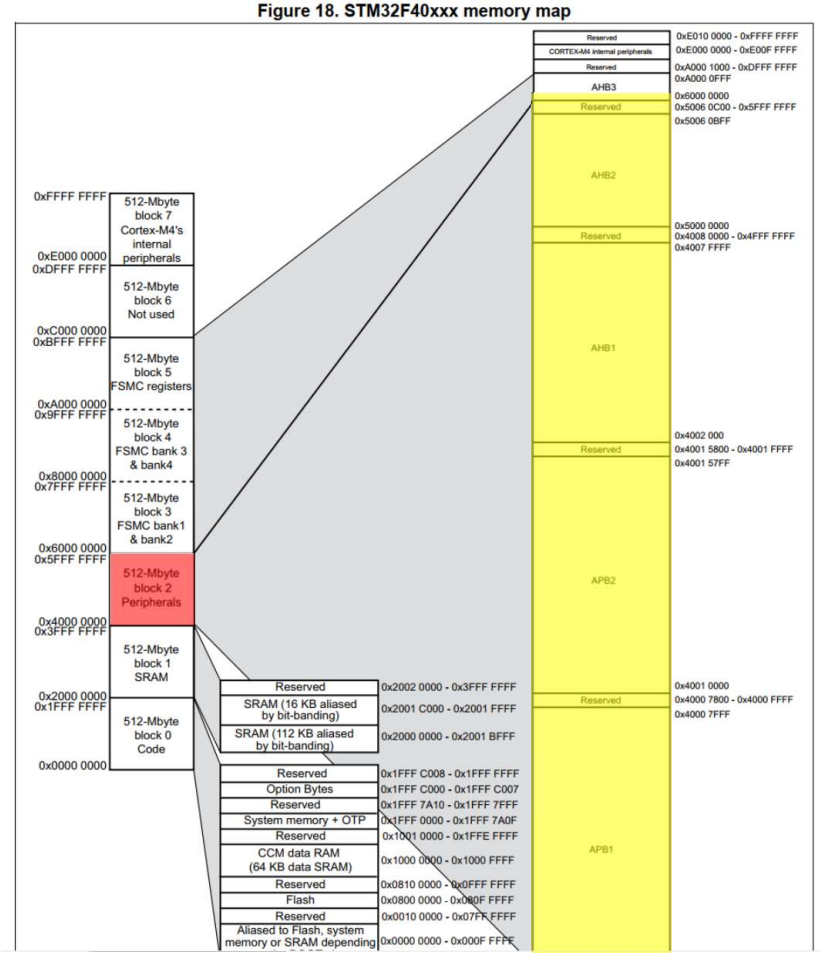


Table 1. STM32F4xx register boundary addresses

Boundary address	Peripheral
0x4002 2800 - 0x4002 2BFF	GPIOK
0x4002 2400 - 0x4002 27FF	GPIOJ
0x4002 2000 - 0x4002 23FF	GPIOI
0x4002 1C00 - 0x4002 1FFF	GPIOH
0x4002 1800 - 0x4002 1BFF	GPIOG
0x4002 1400 - 0x4002 17FF	GPIOF
0x4002 1000 - 0x4002 13FF	GPIOE
0x4002 0C00 - 0x4002 0FFF	GPIOD
0x4002 0800 - 0x4002 0BFF	GPIOC
0x4002 0400 - 0x4002 07FF	GPIOB
0x4002 0000 - 0x4002 03FF	GPIOA

GPIOx_MODER 工作模式

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
						0	0	0	0	0	1	0	0	0	0

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode 复用功能模式

11: Analog mode

问：PC2工作在输出模式，GPIOC_MODER = ?

key: = 0x0000010 = 0x10

GPIOx_ODR 端口输出

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
												0	1	0	0

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)

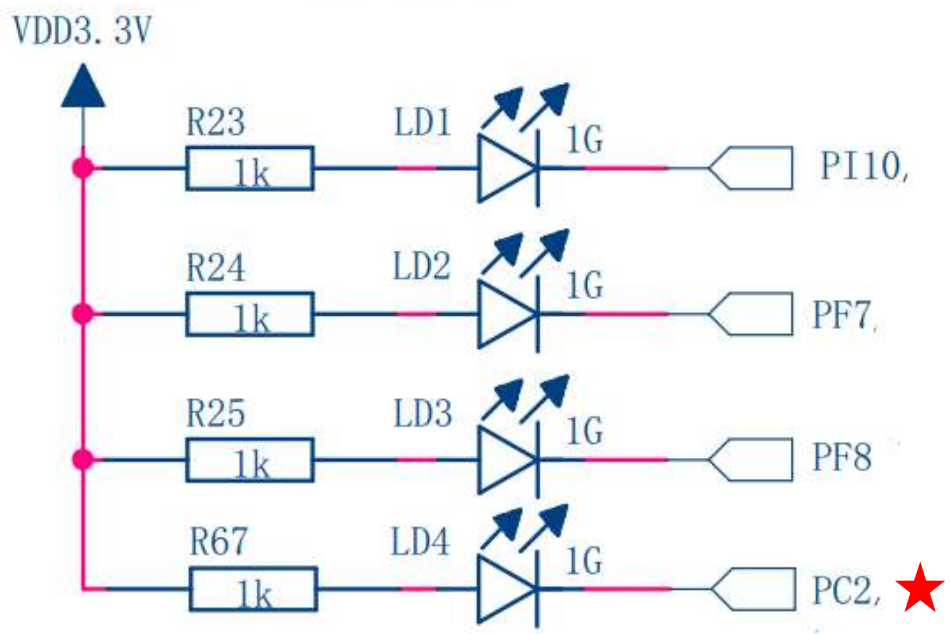
问：仅PC2 = 1，GPIOC_ODR = ?

Key: = 0x0004 = 0x04

IO programming

```
/*功能：通过PC2引脚控制LD4的亮灭*/
main(void){
//开启模块时钟
//指定用作GPIO输出
GPIOC->MODER = 0x10
//控制引脚输出电平(PC2=1,灯灭)
GPIOC->ODR = 0x04
//亮灭切换,间隔1秒
while(1){
  delayms(1000);  /*延迟1000ms*/
  GPIOC->ODR = ~ GPIOC->ODR;
  }
}
```

F407开发板自定义LED指示灯

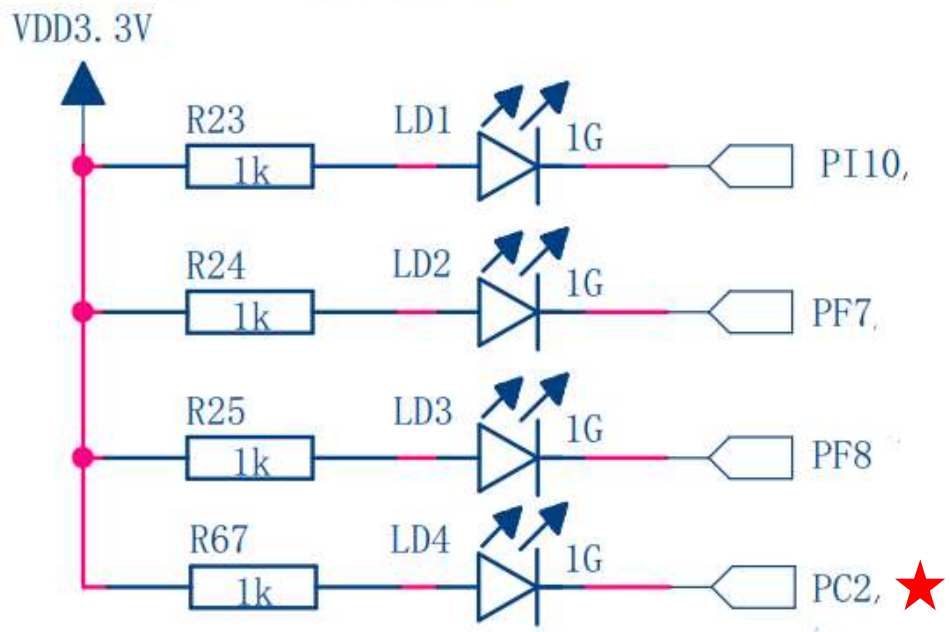


低电平点亮 高电平熄灭

IO programming

```
/*示例：只改变局部比特位，其他位不受影响*/
main(void){
//开启模块时钟
//指定用作GPIO输出（仅影响PC2, =1）
GPIOC->MODER |= 0x10
//控制引脚输出电平（仅影响PC2, =0, 亮灯）
GPIOC->ODR &= ~(0x04)
//亮灭切换,间隔1秒
while(1){
  delayms(1000);  /*延迟1000ms*/
  GPIOC->ODR = ~ GPIOC->ODR;
  }
}
```

F407开发板自定义LED指示灯



低电平点亮 高电平熄灭

IO programming

……

GPIOC->ODR = ~ GPIOC->ODR;

……

如果不想翻转GPIOC的所有端口，可以使用单引脚置位/复位寄存器。

BSRR（Port bit set/reset register）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

- 高16位：用于复位操作，写入1有效，写0无影响；
- 低16位：用于置位操作，写入1有效，写0无影响。

GPIO编程三步走 – 输出

- Step 1：使能GPIO端口的时钟
- Step 2：将GPIO端口设置为通用IO及输出模式
- Step 3：设置ODR寄存器控制引脚输出0和1

RCC – 复位与时钟控制寄存器

- 详见F407的寄存器说明手册.pdf
  Chapter 7 Reset and clock control
  7.3 RCC registers

Control Registers:

RCC_AHB1ENR	AHB1总线时钟使能设置
RCC_AHB2ENR	AHB2总线时钟使能设置
RCC_AHB3ENR	AHB3总线时钟使能设置
RCC_APB1ENR	APB1总线时钟使能设置
RCC_APB2ENR	APB2总线时钟使能设置

RCC_AHB1ENR AHB1外设时钟使能

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved		DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	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			GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

问：仅开放Port C时钟，RCC_AHB1ENR = ？

Key: = 0x04

问：C语句怎么写?

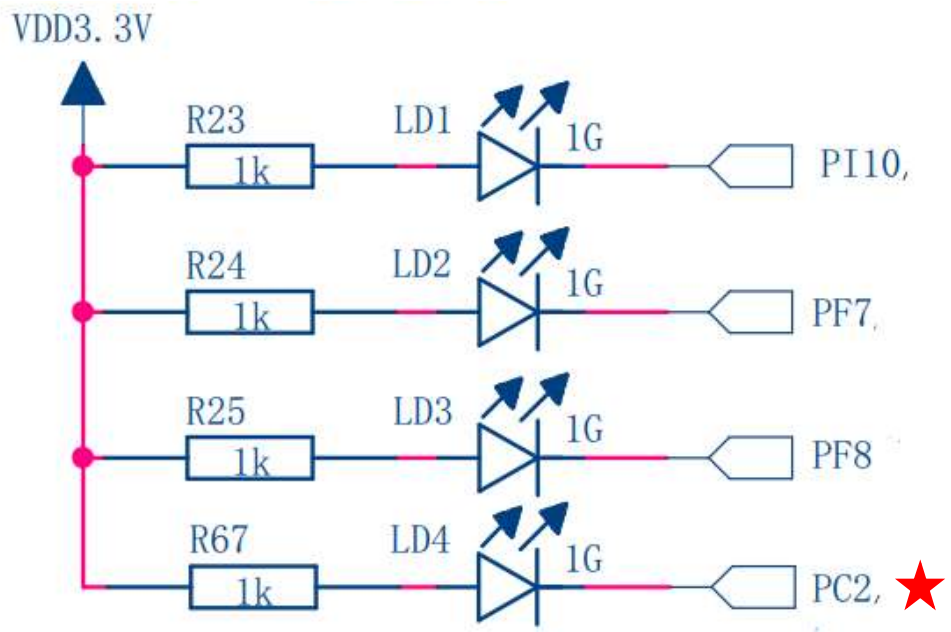
RCC->AHB1ENR |= (1<<2)　把1向左移动两位

Bit 2 **GPIOCEN:** IO port C clock enable
Set and cleared by software.
0: IO port C clock disabled
1: IO port C clock enabled

IO programming

```
main(void){
//开启模块时钟
RCC->AHB1ENR |= (1<<2)
//指定用作GPIO输出（仅影响PC2, =1）
GPIOC->MODER |= 0x10
//控制引脚输出电平（仅影响PC2, =0）
GPIOC->ODR &= ~(0x04)
//亮灭切换,间隔1秒
while(1){
  delayms(1000);  /*延迟1000ms*/
  GPIOC->ODR = ~ GPIOC->ODR;
  }
}
```

F407开发板自定义LED指示灯



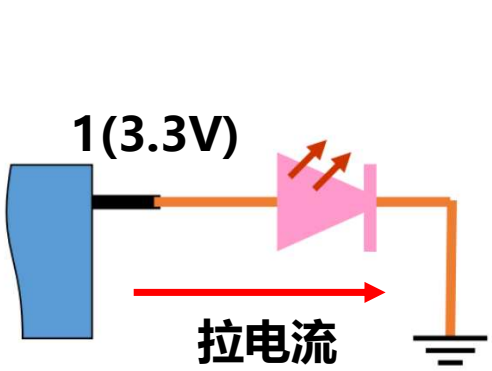
低电平点亮 高电平熄灭

IO。。。

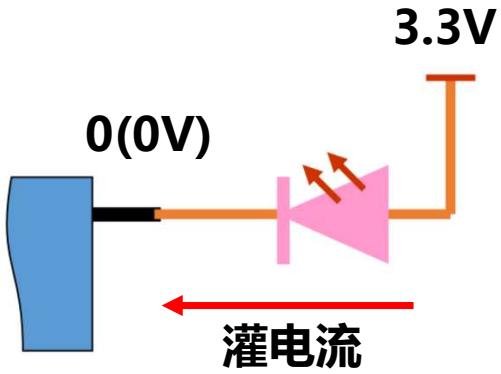
- IO is so simple?

IO进阶 - 驱动能力

- IO端口的灌电流能力高于拉电流能力



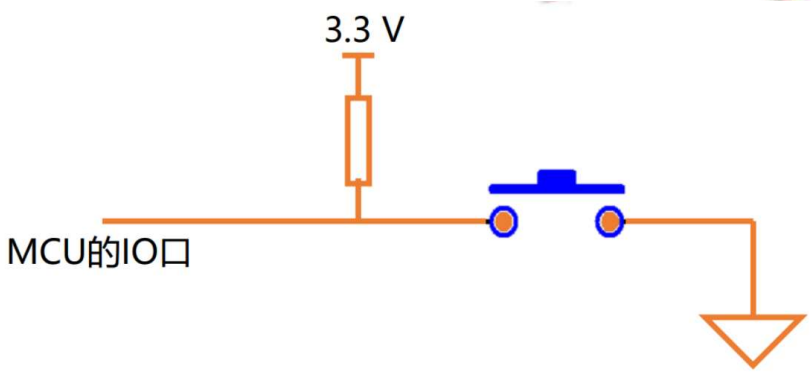
×



√

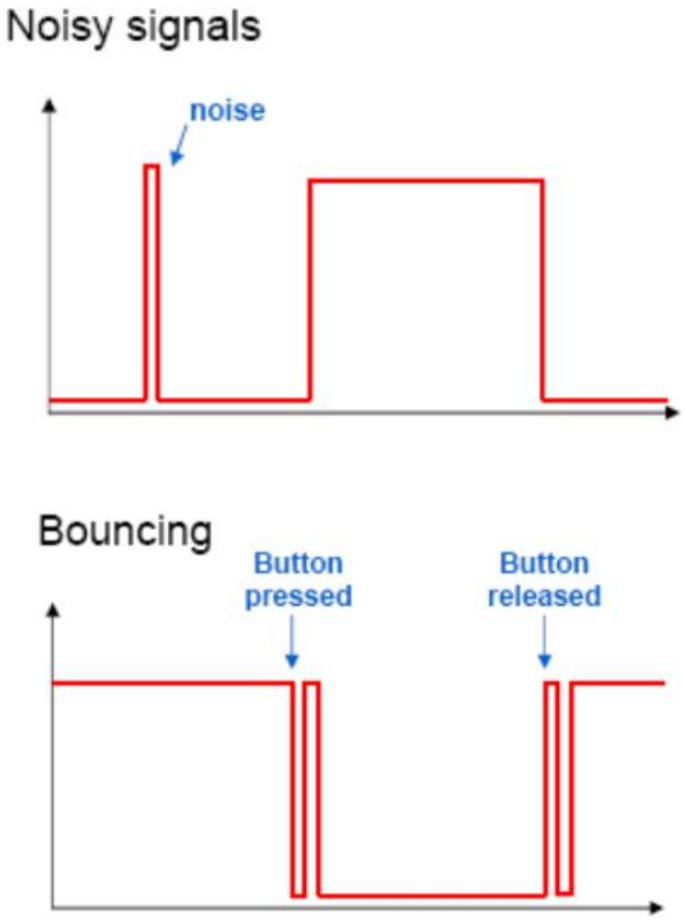
IO应用 – 按键

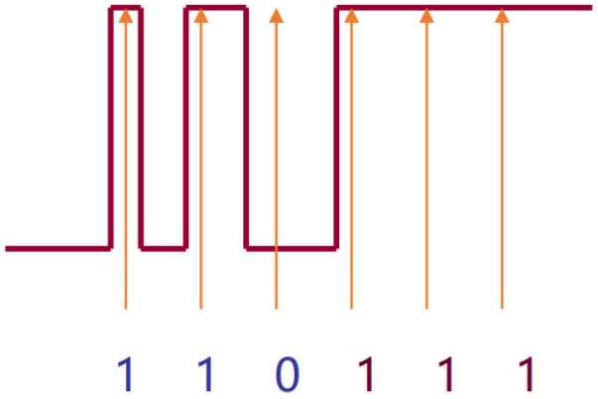
- 带有自恢复装置的开关，按一次产生一个脉冲
- 常开常闭型
- 各种形状，各种尺寸
- 耐高压，轻触式，带指示
- 一个按键也可以提供多组触点



IO 进阶

IO 进阶 – 采样

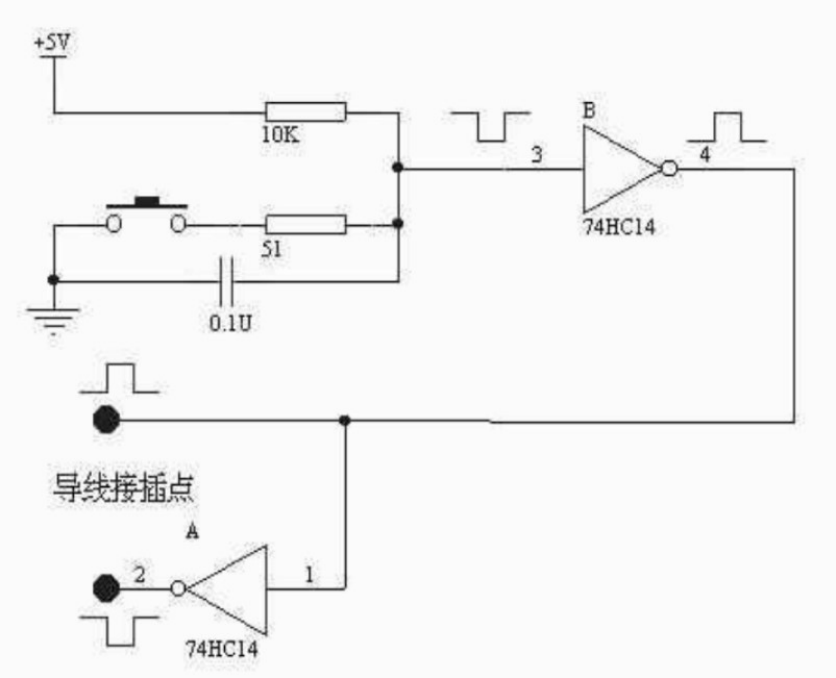




- 开关/按键通常会产生尖峰噪声和抖动，因此必须进行硬件或软件滤波。

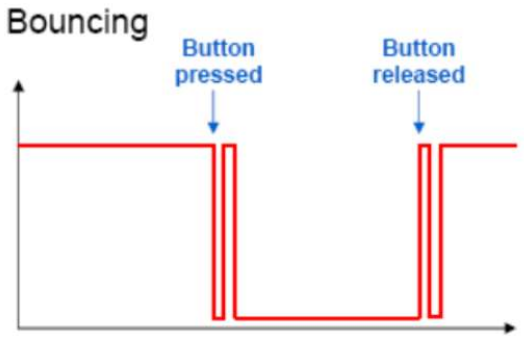
Filter – 滤波

硬件去抖动



软件去抖动

延时10ms再次读取端口， 如果前后两次结果相同，就认为状态稳定

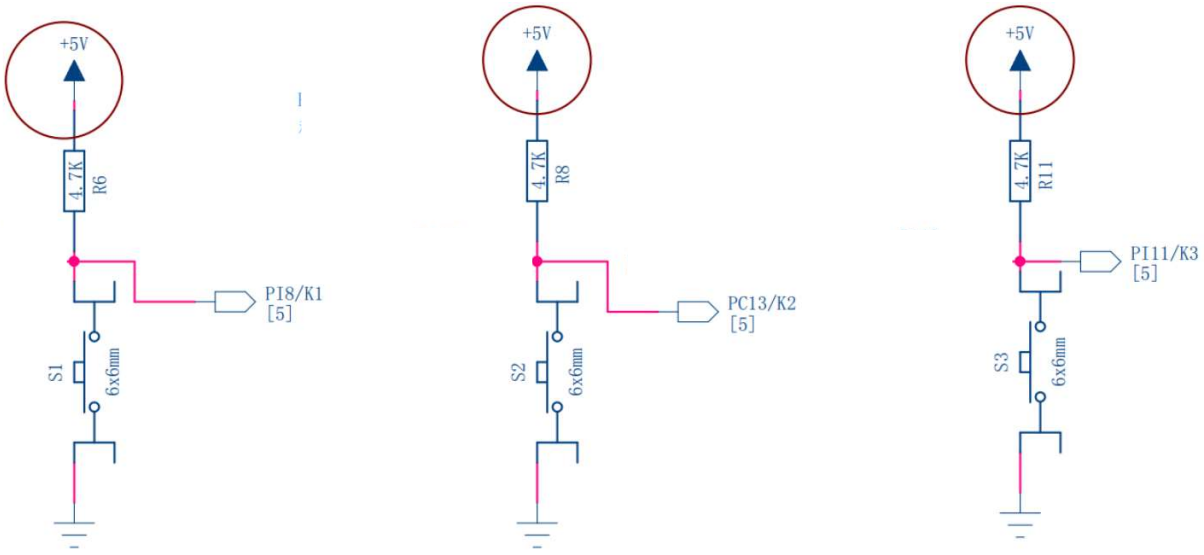


IO programming – 采样编程

```
main(void)
{
  ......
  ??


}
```

F407开发板自定义按钮



不按为高电平，按下为低电平

GPIO编程三步走 – 输入

- Step 1：使能GPIO端口的时钟
- Step 2：将GPIO端口设置为通用IO及输入模式
- Step 3：读取IDR寄存器

RCC_AHB1ENR AHB1外设时钟使能

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved		DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	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			GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

问：开放Port I时钟，RCC_AHB1ENR = ?

Key: = 0x100

问：C语句怎么写?

RCC->AHB1ENR |= (1<<8)

Bit 8 **GPIOIRST:** IO port I reset
Set and cleared by software.
0: does not reset IO port I
1: resets IO port I

F407 GPIO

- 详见F407的参考手册(寄存器说明).pdf
  Chapter 8 General-purpose IOs
  8.4 GPIO registers

Control Registers：

GPIOx_MODER	**工作模式设置**
GPIOx_OTYPER	输出方式设置
GPIOx_OSPEEDR	输出速度设置
GPIOx_PUPDR	上拉/下拉电阻设置
GPIOx_IDR	**端口输入数据**
GPIOx_ODR	端口输出数据
GPIOx_BSRRL	单引脚置位(1)
GPIOx_BSRRH	单引脚复位(0)

GPIOx_MODER 工作模式

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

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

问：PI8工作在输入模式，GPIOI_MODER = ?

Key: = 0x00

GPIOx_IDR 端口输入

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
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

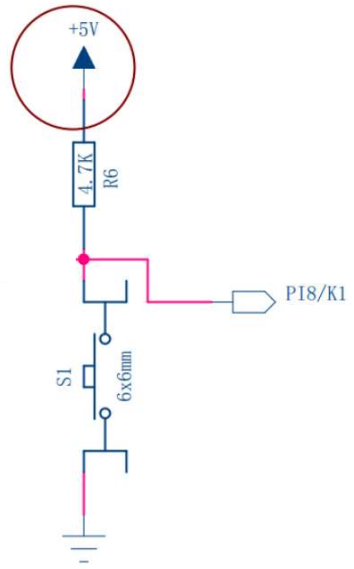
These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

只读寄存器，仅以16bit方式读取，每个bit对应一个引脚状态

IO programming – 编程

```
/*功能：通过按键控制LED亮灭*/
main(void){
  unsigned short keytemp1, keytemp2;
  //开启模块时钟
  RCC->AHB1ENR |= (1<<2);      /*Port C*/
  RCC->AHB1ENR |= (1<<8);      /*Port I*/
  //指定用作GPIO输出（PC2）
  GPIOC->MODER |= 0x10;
  //指定用作GPIO输入（PI8）
  GPIOI->MODER |= 0x00;
  控制引脚输出电平（PC2=0）
  GPIOC->ODR &= ~(0x04);
  (接下页)
```

F407开发板自定义按钮

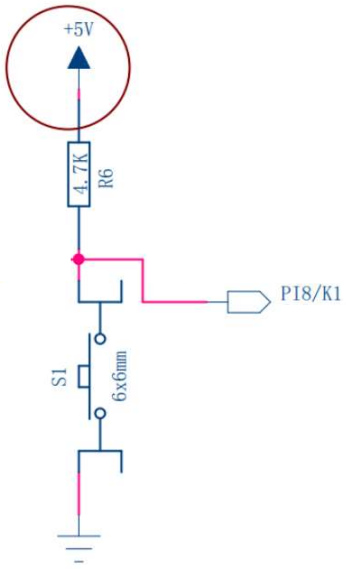


不按为高电平，按下为低电平

IO programming – 编程

(接上页)

```
//按键切换亮灭
while(1){
  keytemp1 = GPIOI->IDR & (1<<8);   /*仅留PI8*/
  delayms(10);
  keytemp2 = GPIOI->IDR & (1<<8);
  if(( keytemp1 | keytemp2 ) == 0)     /*两次采样一致，则键为按下*/
  {
     GPIOC->ODR = ~ GPIOC->ODR;
     delayms(1000);
   }
 }
}
```

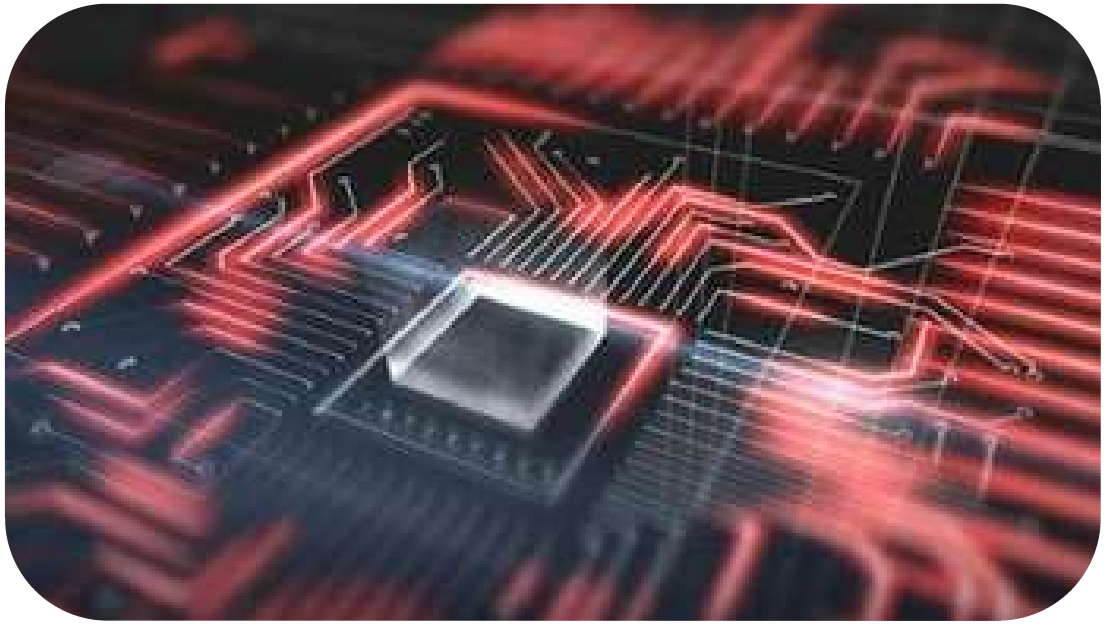


频繁采样的缺陷?

效率低, 浪费CPU

不按为高电平，按下为低电平

嵌入式系统(EMBEDDED SYSTEM)



第3章 STM32开发初步

❀ **3.3 STM32软件开发基础**

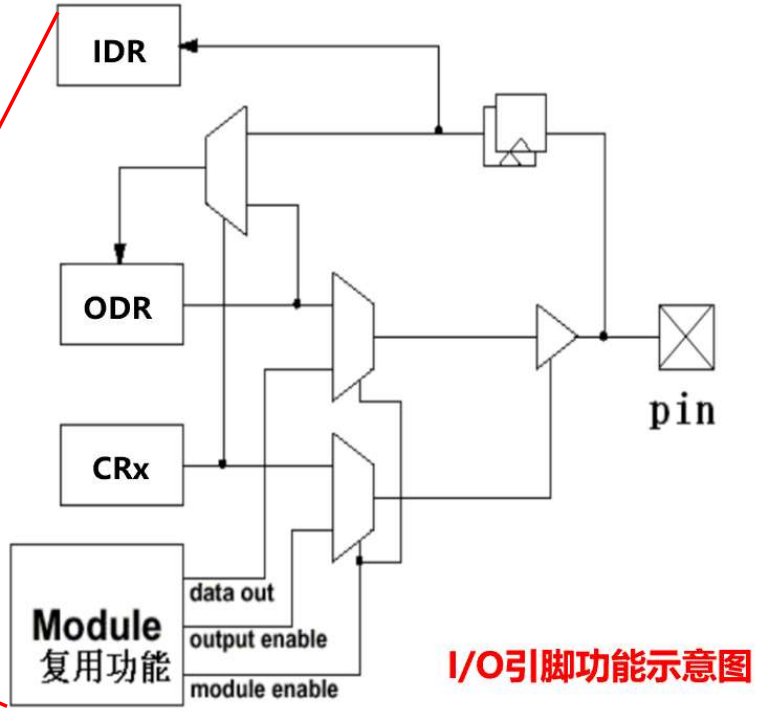
回顾 – IO寄存器编程

- 向寄存器写入/读出数据就可以控制IO引脚输出或读取IO引脚。

Table 1. STM32F4xx register boundary addresses

Boundary address	Peripheral
0x4002 2800 - 0x4002 2BFF	GPIOK
0x4002 2400 - 0x4002 27FF	GPIOJ
0x4002 2000 - 0x4002 23FF	GPIOI
0x4002 1C00 - 0x4002 1FFF	GPIOH
0x4002 1800 - 0x4002 1BFF	GPIOG
0x4002 1400 - 0x4002 17FF	GPIOF
0x4002 1000 - 0x4002 13FF	GPIOE
0x4002 0C00 - 0x4002 0FFF	GPIOD
0x4002 0800 - 0x4002 0BFF	GPIOC
0x4002 0400 - 0x4002 07FF	GPIOB
0x4002 0000 - 0x4002 03FF	GPIOA

16组



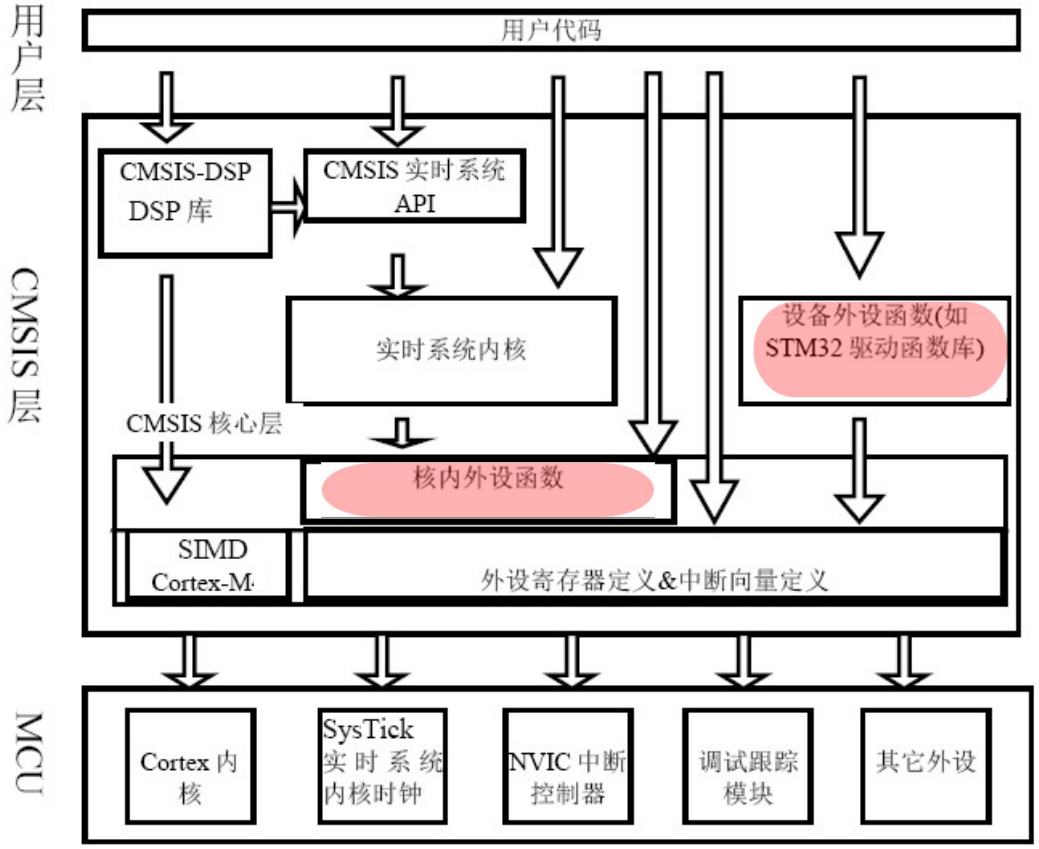
I/O引脚功能示意图

本节安排

- 嵌入式开发的基本概念与工具链（清华慕课）
- 嵌入式开发的进阶知识（清华慕课）
- 基于Keil μ5的STM32开发过程（教师自录）
- 编程练习E1.1~E1.4：建立工程、IO寄存器编程（配套视频E1.2&1.3）
- CMSIS 与 STM32标准库（本次课）

Cortex微控制器软件接口标准(CMSIS)

CMSIS软件架构



承上启下的作用

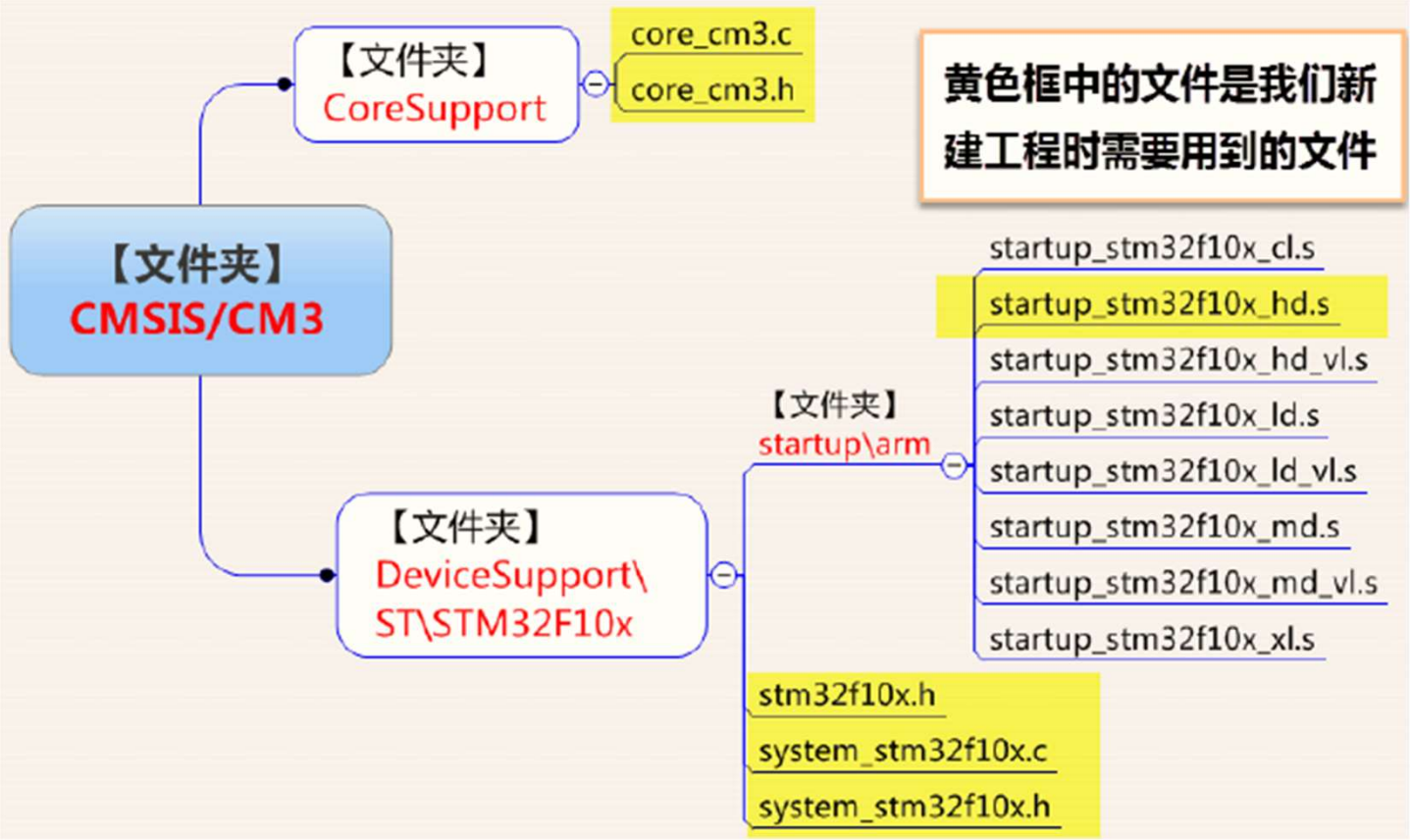
- 对寄存器统一定义
- 为用户提供接口

两个核心层

- ARM公司提供的核内访问层
- 芯片厂商提供的外设访问层

CMSIS的官方文件系统

- 内核文件、启动文件和外设支持文件（以CM3内核、STM32F103为例）



内核相关文件

- core_cm3.c
  - 操作内核寄存器的函数

    较少使用

内核相关文件

- core_cm3.h
  - 内核寄存器及类型的定义
    - Core
    - MPU
    - SysTik
    - FPU

标志位

```
//CMSIS_CORE  Status and Control Registers
 struct
  {
#if (__CORTEX_M != 0x04)
    uint32_t _reserved0:27;       /*!< bit:  0..26  Reserved   */
#else
    uint32_t _reserved0:16;       /*!< bit:  0..15  Reserved   */
    uint32_t GE:4;                /*!< bit: 16..19  ≥ flags    */
    uint32_t _reserved1:7;        /*!< bit: 20..26  Reserved */
#endif
    uint32_t  Q:1;      /* bit:    27  Saturation condition flag  */
    uint32_t  V:1;      /* bit:    28  Overflow condition code flag */
    uint32_t  C:1;      /* bit:    29  Carry condition code flag */
    uint32_t  Z:1;      /* bit:    30  Zero condition code flag */
    uint32_t  N:1;      /* bit:    31  Negative condition code flag*/
  } b;
```

系统启动文件（.s）

启动文件	区别
startup_stm32f10x_ld.s	ld: low-density 小容量， FLASH 容量在 16-32K 之间
startup_stm32f10x_md.s	md: medium-density 中容量，FLASH 容量在 64-128K 之间
startup_stm32f10x_hd.s	hd: high-density大容量， FLASH 容量在 256-512K 之间
startup_stm32f10x_xl.s	xl: 超大容量， FLASH 容量在 512-1024K 之间
以上四种都属于基本型，包括 STM32F101xx、STM32F102xx、STM32F103xx 系列	
startup_stm32f10x_cl.s	cl:connectivity line devices 互联型，特指 STM32F105xx 和 STM32F107xx 系列
startup_stm32f10x_ld_vl.s	vl:value line devices 超值型系列，特指 STM32F100xx 系列
startup_stm32f10x_md_vl.s	
startup_stm32f10x_hd_vl.s	

- **功能：是系统上电后运行的一段汇编程序**

  -- 设置堆栈指针SP

  -- 设置程序指针PC

  ……

  -- 引导至主程序main()

以Startup作为开头

.s系统启动文件

外设支持文件

- system_stm32f10x.c
  - STM32的时钟配置
  - 允许修改
- stm32f10x.h，stm32f4xx.h
  - 片上外设寄存器的定义、映射
  - 涉及外设寄存器操作时需引用
    #include "stm32f4xx.h"

```
/* AHB1 所挂外设基地址 */
#define GPIOB_BASE          (AHB1PERIPH_BASE + 0x0400)
#define GPIOC_BASE          (0x4002 0000          + 0x0800)
……
/* GPIO 内部寄存器定义 */
typedef struct
{
  __IO uint32_t  MODER;    /* Port mode register  */
  __IO uint32_t  ODR;      /* Output data register  */
  ……
} GPIO_TypeDef;   /*结构体*/

/* 外设声明(定义) */
#define GPIOB       ((GPIO_TypeDef *)  GPIOB_BASE)
#define GPIOC       ((GPIO_TypeDef *)  GPIOC_BASE)
……
```

结构体的数据类型

进行宏定义

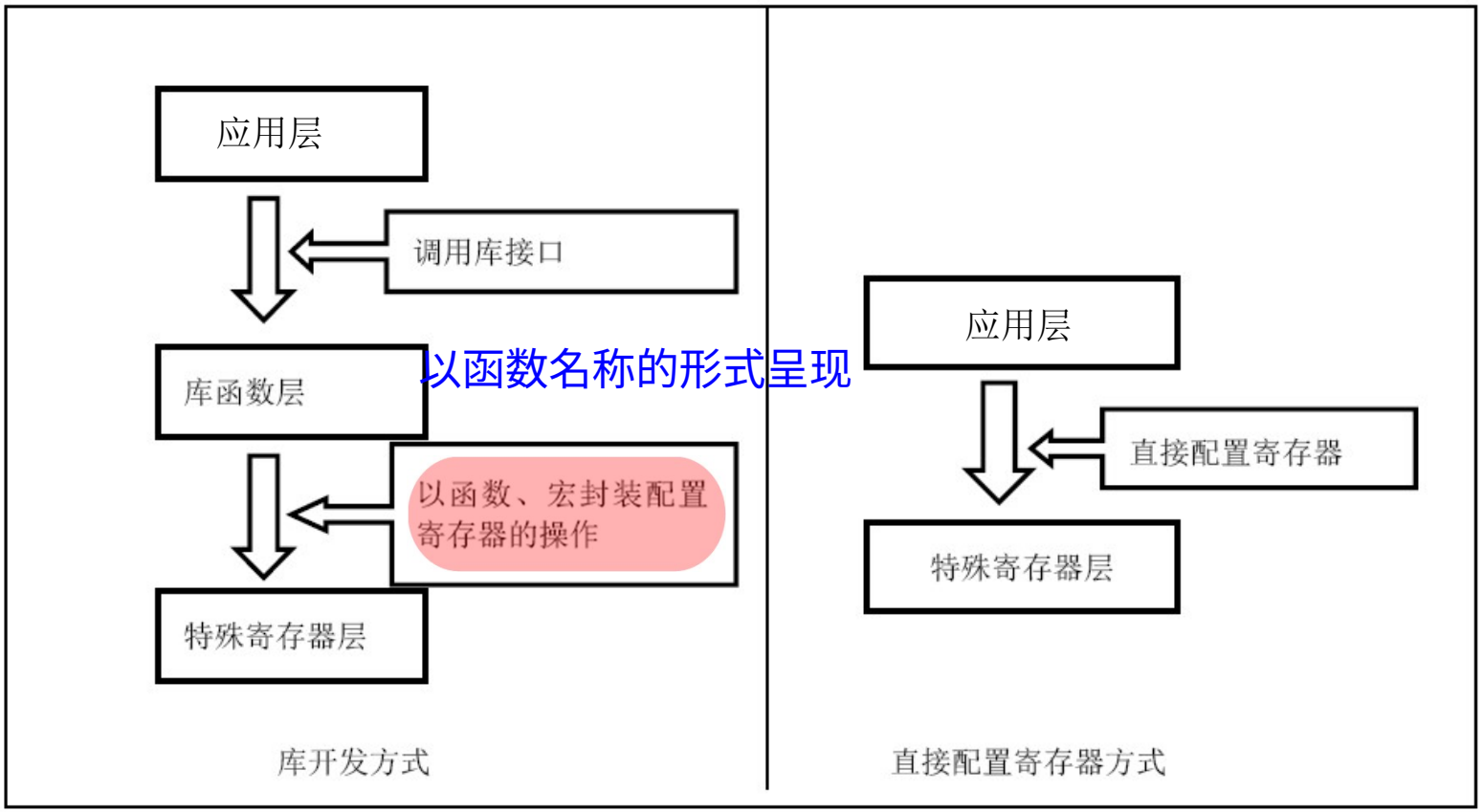
声明后，外设名(GPIOC)既是端口的基地址指针，又是端口各寄存器结构体指针。访问格式：GPIOC->MODER = ……

被定义了一个指针，指向PortC的基地址

通过指针访问必须使用箭头

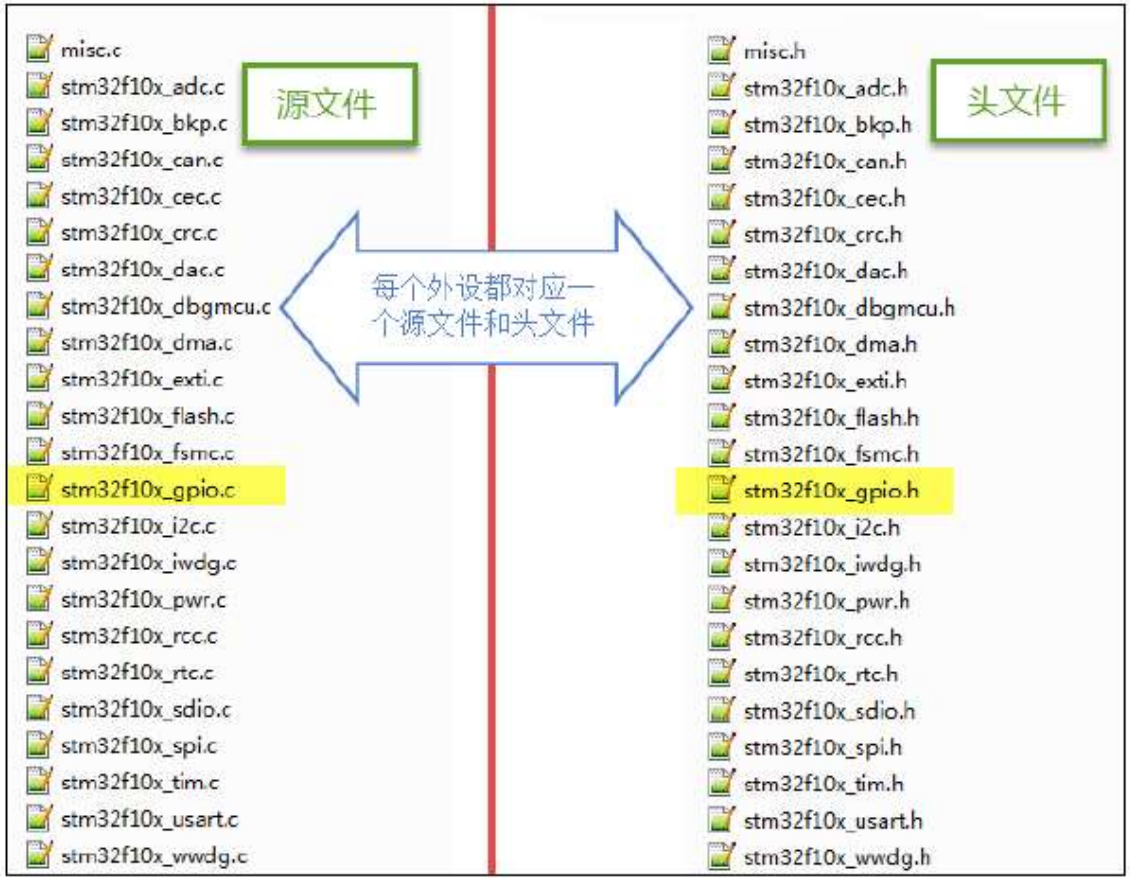
STM32标准外设库

- 固件库开发与寄存器开发对比

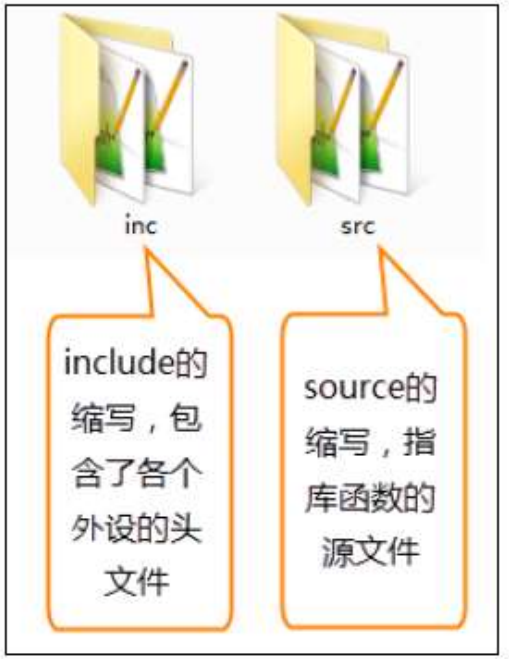


STM32标准外设库

- 固件库资源：**stm32fxxx_ppp.c & .h**　xxx表示型号　ppp表示片上处理设备



- 存放位置



源文件为可执行的具体函数　　头文件是对一些引脚的声明

STM32标准外设库

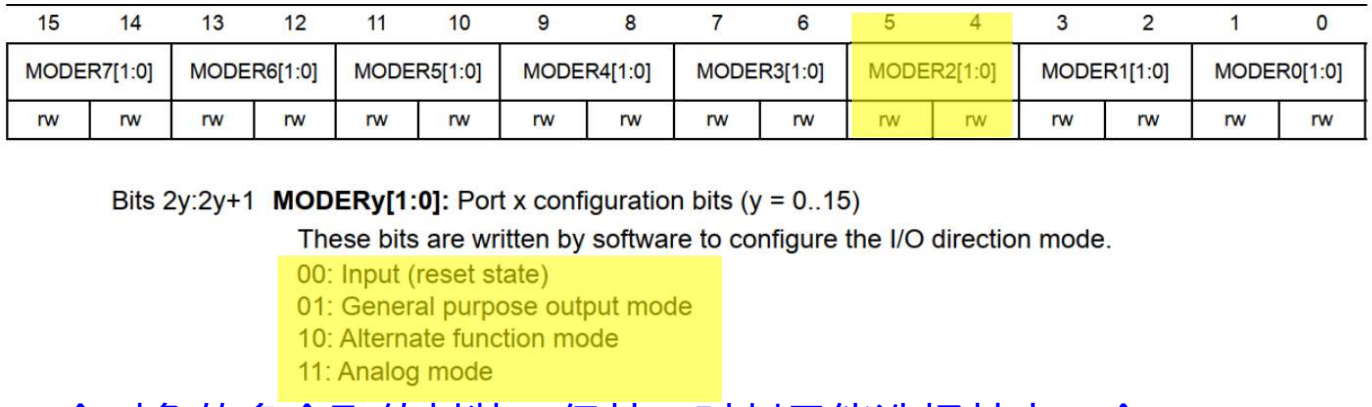
- 固件库头文件（.h）

  -- 封装寄存器的配置

  -- 定义外设的引脚

  ……

<示例stm32f4xx_gpio.h>



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

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

```
/* GPIO 配置模式枚举*/
typedef enum     枚举类型：一个对象的多个取值封装，但某一时刻只能选择其中一个
{
  GPIO_Mode_IN    = 0x00,   /* GPIO Input Mode */
  GPIO_Mode_OUT   = 0x01,   /* GPIO Output Mode */
  GPIO_Mode_AF    = 0x02,   /* GPIO Alternate function Mode */
  GPIO_Mode_AN    = 0x03    /* GPIO Analog Mode */
}GPIOMode_TypeDef;
```

STM32标准外设库

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

b0000 0000 0000 0100 = 0x0004

<示例stm32f4xx_gpio.h>

```
/* GPIO 引脚定义 */
#define GPIO_Pin_0      ((uint16_t) 0x0001)  /* Pin 0 selected */
#define GPIO_Pin_1      ((uint16_t) 0x0002)  /* Pin 1 selected */
#define GPIO_Pin_2      ((uint16_t) 0x0004)  /* Pin 2 selected */
.....
#define GPIO_Pin_15     ((uint16_t) 0x8000)  /* Pin 15 selected */
```

无符号十六位整形　宏定义

STM32标准外设库

- 固件库源文件（.c）

  -- 外设配置操作函数

  -- 外设读/写操作函数

  ……

<示例stm32f4xx_gpio.c>

```
/* GPIO 初始化配置函数*/
void GPIO_Init(GPIO_TypeDef*  GPIOx,   GPIO_InitTypeDef*  GPIO_InitStruct)
{ ……
      //模式寄存器配置
      GPIOx->MODER  &= ~(GPIO_MODER_MODER0 << (pinpos * 2));
      GPIOx->MODER |= (((uint32_t)GPIO_InitStruct->GPIO_Mode) << (pinpos * 2));
      //工作速度配置
      GPIOx->OSPEEDR &= ~(GPIO_OSPEEDER_OSPEEDR0 << (pinpos * 2));
      GPIOx->OSPEEDR |= ((uint32_t)(GPIO_InitStruct->GPIO_Speed) << (pinpos * 2));
  …… }
```

标准外设库之上

- 当用户添加的硬件较多时，应分类建立硬件的固件库

  -- Hardware.h 硬件的头文件（端口定义、函数声明）

  -- Hardware.c 硬件的源程序文件（函数代码）

  最普通的硬件，如

  -- Led灯：led.h，led.c

  -- 按键：key.h，key.c

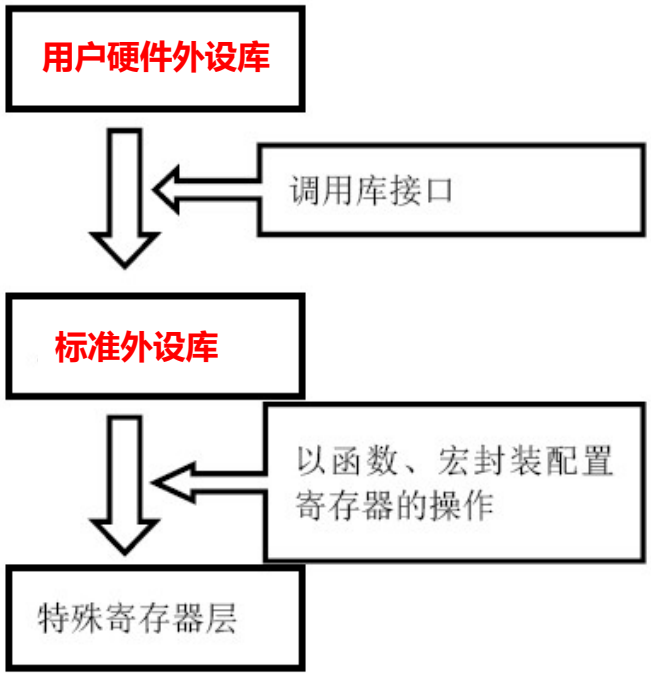
  其他

  -- 液晶屏：lcd.h，lcd.c

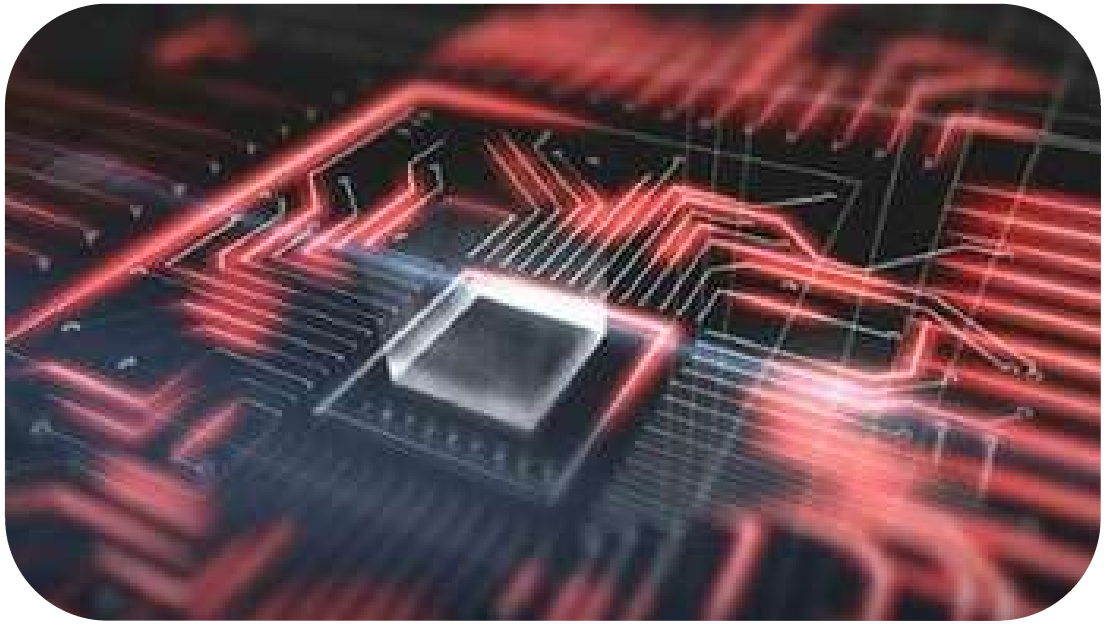
  -- 加速度传感器：MPU6050.h，MPU6050.c

自行编写外设库

程序的规范性



嵌入式系统(EMBEDDED SYSTEM)



第3章 STM32开发初步

❀ **3.3 嵌入式开发中的C语言**

本节安排

- 嵌入式开发中的C语言 – 上 （清华慕课）

  -- 25min。C语言的历史渊源；C语言的数据类型及使用时的注意事项；位操作及其在嵌入式中的应用代码举例。

- 嵌入式开发中的C语言 – 下 （清华慕课）

  -- 30min。强制类型转换的意义和volatile关键字；如何使用指针访问寄存器，读写特定地址；编写中断服务子程序的注意事项；程序执行时的起始代码以及微控制器C语言编程的特点。

- **STM32开发中的C语言**

  -- 数据类型，规范命名，文件引用，宏定义，位操作，结构体和枚举，指针（变量指针、数组指针、结构体指针）

C 数据类型

- typedef 定义准确宽度的数据类型

<stdint.h>中定义了：

char储蓄单个字符或者8位整形

```
/* 准确宽度的有符号整形 */
typedef   signed          char    int8_t;      /* -128~+127*/
typedef   signed  short   int     int16_t;     /* -32768~+32767*/
typedef   signed          int     int32_t;     /* -2,147,483,648~+2,147,483,647*/

/* 准确宽度的无符号整形 */
typedef   unsigned        char    uint8_t;     /* 0~255*/
typedef   unsigned short  int     uint16_t;    /* 0~65535*/
typedef   unsigned        int     uint32_t;    /* 0~4,294,967,295*/
```

变量及函数命名

- 规范命名
  - 有实际意义的英文单词或缩写
  - 单词首字母通常大写
  - 复杂命名考虑用多个单词组合

uint16_t Voltage; √	uint16_t V; ×
float Temperature; √	float t; ×
void SendDataToLcd (.....); √	void SDTL (.....); ×

学编程不仅仅是提高代码编写水平，还要注重代码编写的规范性！

头文件引用

- #include 引用必要文件
  - 引用C标准库 <*.h >

    -- #include <stdio.h>

    -- #include <math.h>

    ……
  - 引用STM32标准设备库或用户设备库 “*.h”

    -- #include “stm32f4xx_ppp.h”

    -- #include “led.h” 用户设备库

    ……

宏定义

- 用define对常量进行直观命名　　宏定义
  - 可读性高
  - 可移植性好

例如：#define　Pi　3.14　/* π值*/

STM32标准固件库中对端口、引脚进行了大量宏定义：

```
#define  GPIOC          ((GPIO_TypeDef *)  GPIOC_BASE)
#define  GPIO_Pin_2     ((uint16_t) 0x0004)
……
```

用户设备在占用GPIO时有必要进行二次宏定义：

```
#define   Led1Port   GPIOC          /*1号灯连在GPIOC端口*/
#define   Led1Pin    GPIO_Pin_2     /*1号灯连在C端口的2号脚*/
```

位操作

- 逻辑位操作
  - AND　　&　　(&&为条件与)
  - OR　　|　　(||为条件或)
  - XOR　　^
  - Inverter（反）　　~
  - Shift left　　<<　　(a<<b，a移位对象，b移位位数)
  - Shift right　　>>　　(同上)

“AND”常用于复位/屏蔽某些位，“OR”常用于置位某些位，左、右移常用于高低位调整。

结构体

◆ 封装归属于同一对象的属性

```
struct Student
{
  int    ID;
  char name;
  int    age;
  ……
};
```

struct – 结构体数据类型说明符

Student - 结构体类型名

ID、name、age、…… 结构体成员变量

● 也可以这样定义

```
typedef struct
{
  int    ID;
  char name;
  int    age;
  ……
} Student;
//定义结构体变量
Student  stu1, stu2;
//成员变量赋值
stu1.ID = 2020123456;
stu1. name =  '小强' ;
……
```

STM32开发中的结构体

- 封装归属于同一硬件的属性
  - GPIO初始化用结构体

```
typedef struct
{
   uint32_t   GPIO_Pin;            /* 待配置的GPIO引脚*/
   GPIOMode_TypeDef   GPIO_Mode;     /*工作模式*/
   GPIOSpeed_TypeDef   GPIO_Speed;   /*工作速度*/
   GPIOOType_TypeDef   GPIO_OType;   /*输出类型，如推挽、漏极开路*/
   GPIOPuPd_TypeDef    GPIO_PuPd;    /*上下拉电阻*/
} GPIO_InitTypeDef;
 GPIOMode_TypeDef   GPIO_InitStructure;     /*定义结构体变量 – IO初始化结构体变量 */
```

枚举

- 数据取值多选一

```
typedef enum
{
  male = 0;
  female = 1;
 }StudentGender;
```

枚举类型

enum – 枚举数据类型说明符

StudentGender – 枚举类型名

male、female - 枚举值

```
//定义枚举变量
StudentGender  sg1, sg2;
//枚举变量赋值
sg1 = male;
sg2 = female;
```

STM32开发中的枚举

- 封装某一配置项的不同配置参数
  - GPIO工作模式配置项

某一时刻多种选一

```
typedef enum
{
  GPIO_Mode_IN     = 0x00,    /* 输入模式 */
  GPIO_Mode_OUT    = 0x01,    /* 输出模式*/
  GPIO_Mode_AF     = 0x02,    /* 复用模式，第二、三.....种功能 */
  GPIO_Mode_AN     = 0x03     /* 模拟模式*/
}GPIOMode_TypeDef;

 GPIOMode_TypeDef   GPIO_Mode;     /*定义枚举变量 - IO工作模式*/
 GPIO_Mode = GPIO_Mode_IN;         /*枚举变量赋值 – IO输入模式*/
```

指针

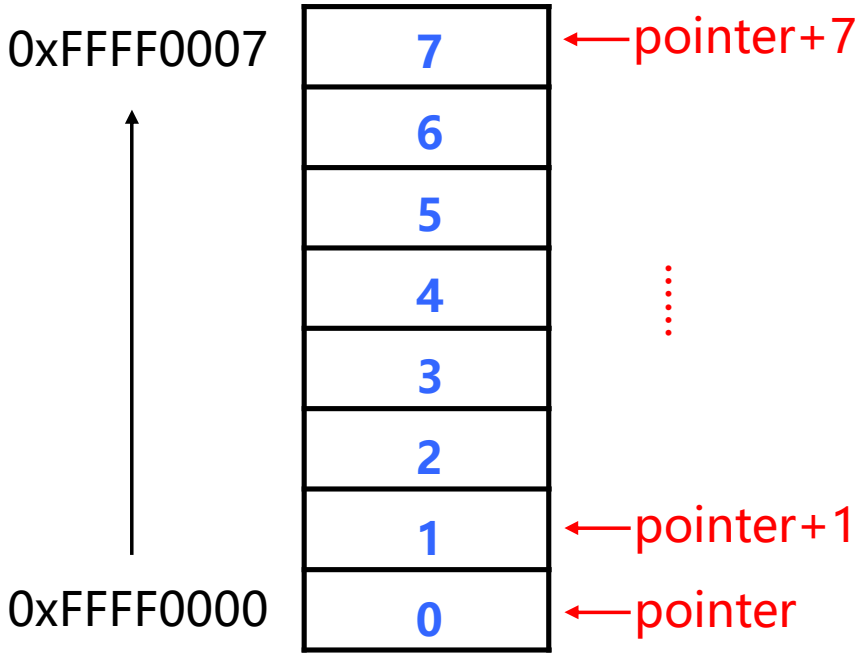
- 指针本质是地址，用于定位并操作变量、数组和函数

```
char *pointer;
pointer = 0xFFFF0000;
for( i = 0; i < 8; i++)
{
 *pointer = i;
  pointer++;
}
```

*pointer ++= i;

循环结束后，pointer = ?

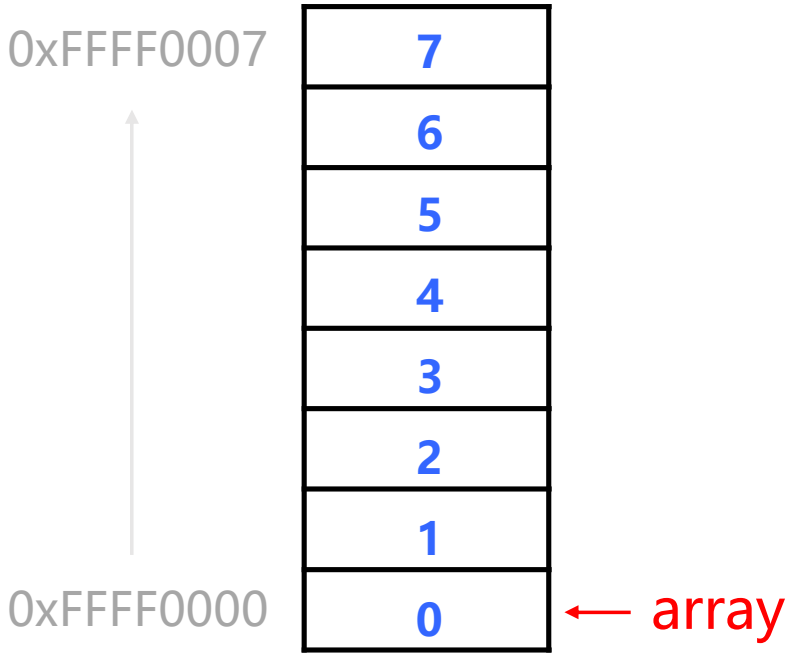
Key: 0xFFFF0008



数组

- 数据的一种组织形式，数组名等用于指针

```
char  array[8], *pointer;
pointer = array;
for( i = 0; i < 8; i++)
{
 *pointer ++= i;
 }
```



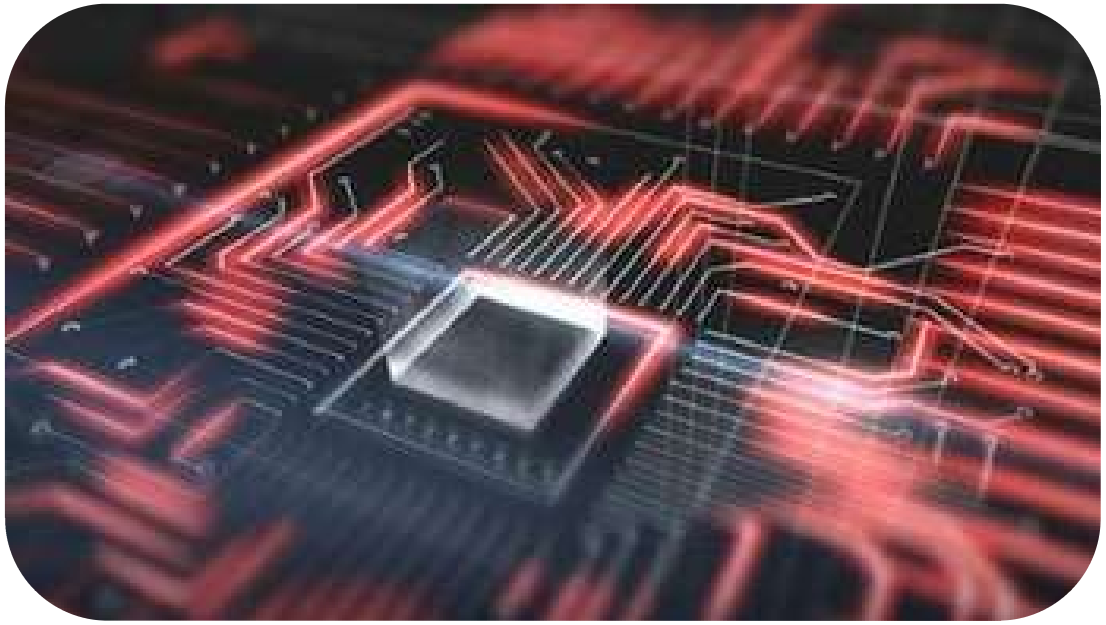
结构体指针

```
typedef struct
{
  int    ID;
  char name;
  int    age;
  ……
} Student;
//定义结构体变量/指针
Student  stu1, *pointer;
```

```
//成员变量直接赋值
stu1.ID = 2020123456;
stu1. name = ‘小强’ ;
……

//通过结构体指针赋值
pointer = &stu1;
pointer -> ID = 2020123456;
pointer -> name = ‘小强’ ;
……
```

嵌入式系统(EMBEDDED SYSTEM)

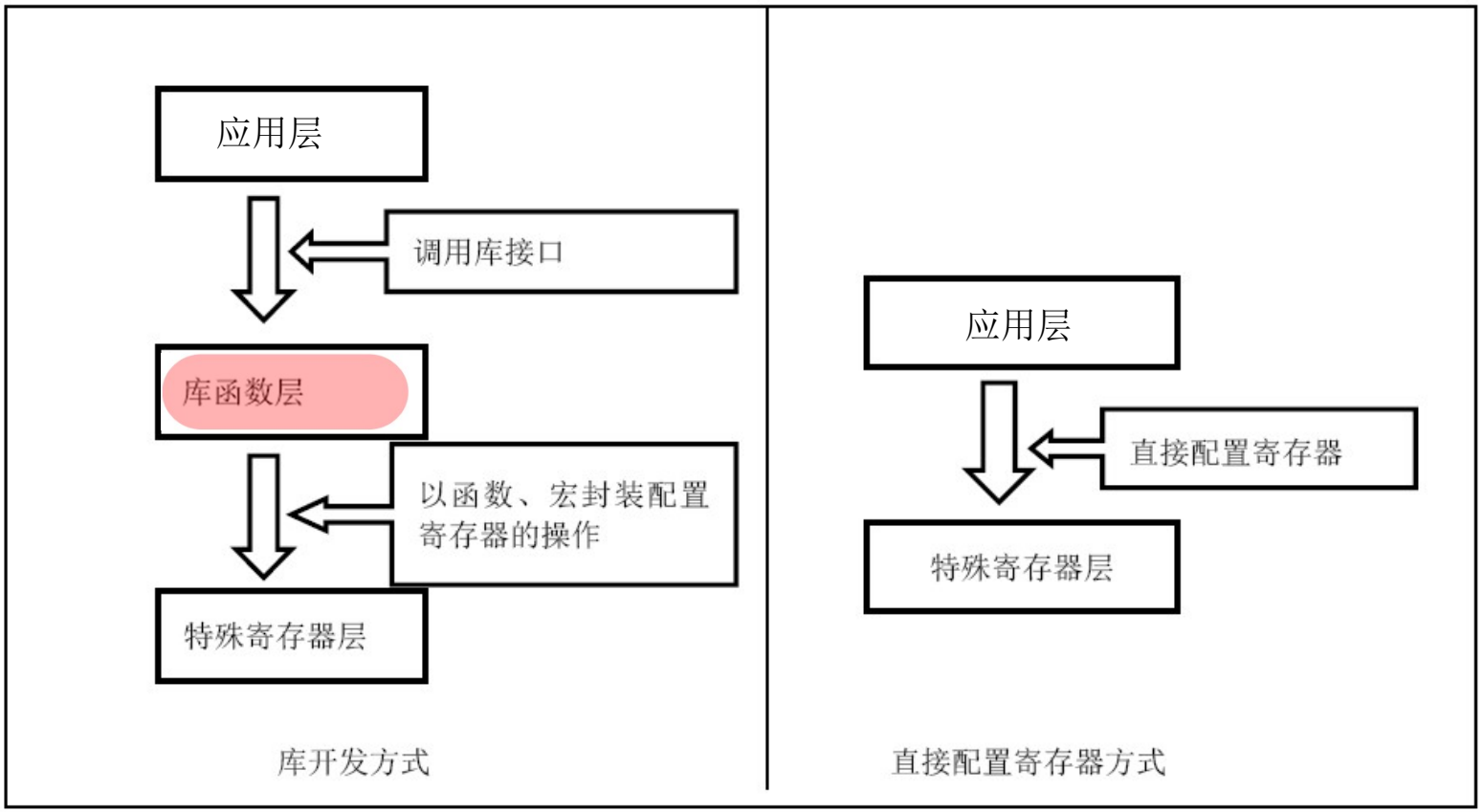


第3章 STM32开发初步

❀ **3.5 IO的库函数编程**

STM32的开发方式

- 标准库开发与寄存器开发对比



本节内容安排

- 常用GPIO标准库函数
- 标准库输出编程 – 点灯
- 标准库输入编程 – 按键
- 用户设备的建库及应用
- 课后编程练习E1.5：库函数编程实现按键控制LED

常用GPIO库函数

在stm32f4xx_gpio.c 中，对GPIO寄存器的操作进行了函数封装

/* 初始化与配置函数***/

- **void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)**

  功能：根据GPIO_InitStruct中指定的参数初始化GPIOx中的寄存器

/* GPIO 读和写函数 ***/

- **uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);**

  功能：读取GPIOx端口数据

- **uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);**

  功能：读取GPIOx的GPIO_Pin引脚上的数据

常用GPIO库函数

- **void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);**

  功能：向GPIOx写出数据PortVal　　16个引脚一起写

- **void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);**

  功能：置位GPIOx的GPIO_Pin引脚

- **void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);**

  功能：复位GPIOx的GPIO_Pin引脚

- **void GPIO_ToggleBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);**

  功能：翻转GPIOx的GPIO_Pin引脚

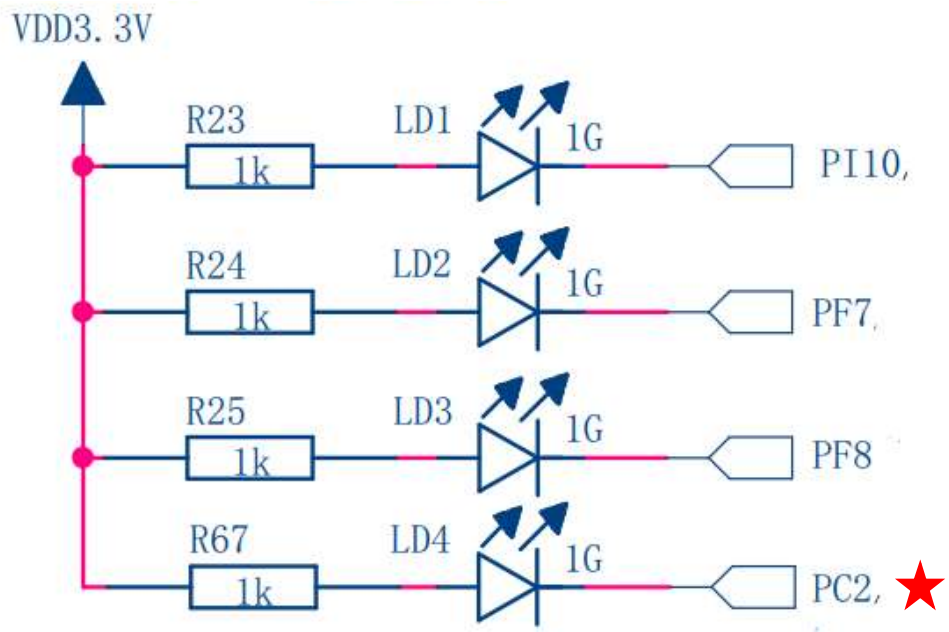
IO 库函数编程 – 点灯

```
main(void)
{
  //点亮LED4
   ……



}
```

F407开发板自定义LED指示灯



低电平点亮 高电平熄灭

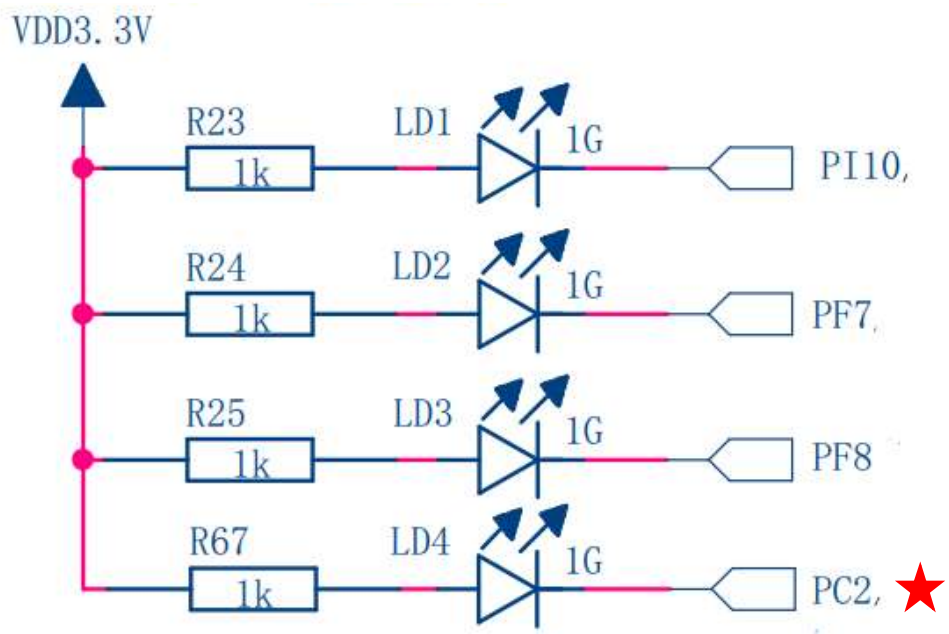
GPIO编程三步走 – 输出

- Step 1：使能GPIO端口的时钟
- Step 2：将GPIO初始化设置为通用IO输出模式
- Step 3：控制引脚输出0和1

回顾 IO寄存器编程代码

```
main(void){
//开启模块时钟
RCC->AHB1ENR |= (1<<2)
//指定用作GPIO输出（仅PC2）
GPIOC->MODER |= 0x10
//控制引脚输出电平（仅PC2=0，灯亮）
GPIOC->ODR &= ~(0x04)
//亮灭切换,间隔1秒
while(1){
  delayms(1000);  /*延迟1000ms*/
  GPIOC->ODR = ~ GPIOC->ODR;
  }
}
```

F407开发板自定义LED指示灯



低电平点亮 高电平熄灭

时钟配置函数

(stm32f4xx_rcc.c)

外部设备

- void RCC_AHB1PeriphClockCmd(uint32_t RCC_AHB1Periph, FunctionalState NewState)

功能：AHB1总线外设时钟控制

参数：-- RCC_AHB1Periph 需进行时钟开关控制的外设，如

具体值	含义
RCC_AHB1Periph_GPIOA	GPIOA clock
RCC_AHB1Periph_GPIOB	GPIOB clock
RCC_AHB1Periph_GPIOC	**GPIOC clock**
……	……

-- NewState 开关状态，ENABLE 或者 DISABLE

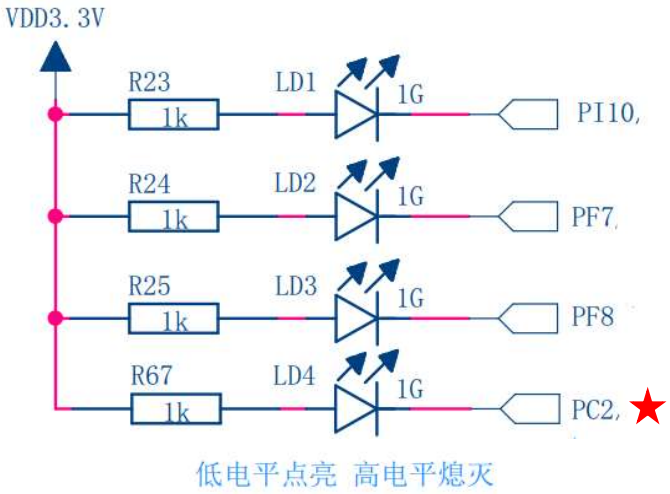
GPIO初始化结构体

```
typedef struct
{  uint32_t   GPIO_Pin;     /* 待配置的GPIO引脚*/
   GPIOMode_TypeDef   GPIO_Mode;     /*工作模式：输入(默认)，输出，复用*/
   GPIOSpeed_TypeDef   GPIO_Speed;   /*工作速度：2M(默认)、25M、50MHz*/
   GPIOOType_TypeDef   GPIO_OType;   /*输出类型：推挽(默认)、漏极开路(I2C总线用)*/
   GPIOPuPd_TypeDef    GPIO_PuPd;    /*上下拉电阻：上拉、下拉、无（默认）*/
} GPIO_InitTypeDef;
//定义初始化用结构体变量
 GPIO_InitTypeDef     GPIO_InitStructure;
//成员变量赋值（示例）
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
```

IO 库函数编程 – 点灯

```
main(void){
GPIO_InitTypeDef   GPIO_InitStructure;  /*定义GPIO初始化用的结构体变量*/
//开启模块时钟
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC , ENABLE);
//指定用作GPIO输出（PC2）
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_Init(GPIOC,  &GPIO_InitStructure);  //&为取地址
```

接下页

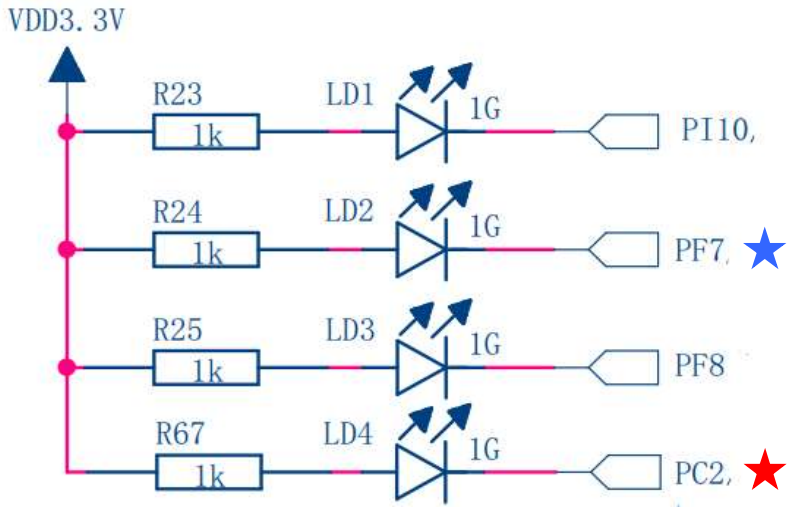


低电平点亮 高电平熄灭

IO 库函数编程 – 点灯

接上页

```
//控制引脚输出电平（仅PC2=0，灯亮）
GPIO_ResetBits(GPIOC, GPIO_Pin_2);
//亮灭切换,间隔1秒
while(1){
    delayms(1000);   /*延迟1000ms*/
    GPIO_ToggleBits(GPIOC, GPIO_Pin_2);
  }
}
```



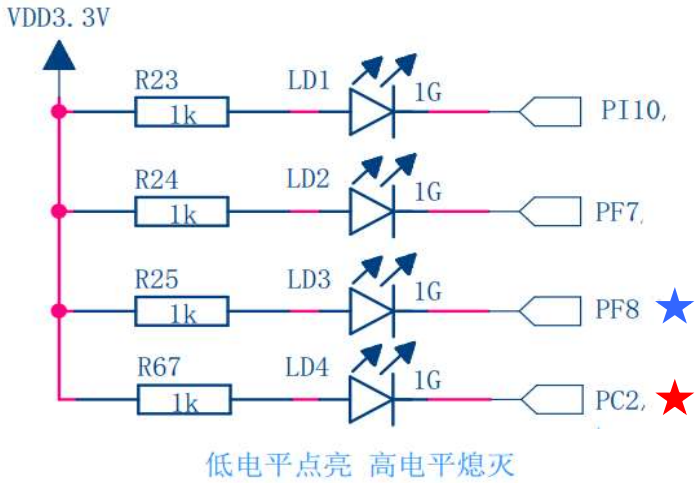
低电平点亮 高电平熄灭

若想把LED3也点亮闪烁，程序该怎么添加?

IO 库函数编程 – 点灯

- 同时闪烁LED2、LED4的代码

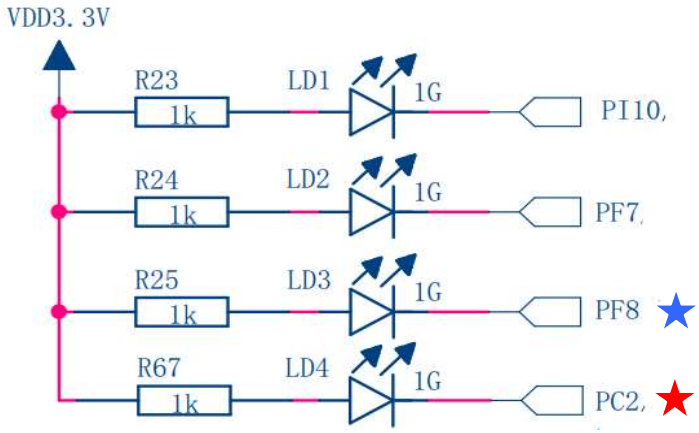
```
main(void){
GPIO_InitTypeDef   GPIO_InitStructure;     /*定义GPIO初始化用的结构体变量*/
//开启模块时钟
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC , ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF , ENABLE);
//指定用作GPIO输出（PC2、PF8）
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_Init(GPIOC,  &GPIO_InitStructure);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
GPIO_Init(GPIOF,  &GPIO_InitStructure);
```



低电平点亮 高电平熄灭

IO 库函数编程 – 点灯

```
//控制引脚输出电平（PC2=0）
GPIO_ResetBits(GPIOC, GPIO_Pin_2);
GPIO_ResetBits(GPIOF, GPIO_Pin_8);
//亮灭切换,间隔1秒
while(1){
    delayms(1000);   /*延迟1000ms*/
    GPIO_ToggleBits(GPIOC, GPIO_Pin_2);
    GPIO_ToggleBits(GPIOF, GPIO_Pin_8);
  }
}
```



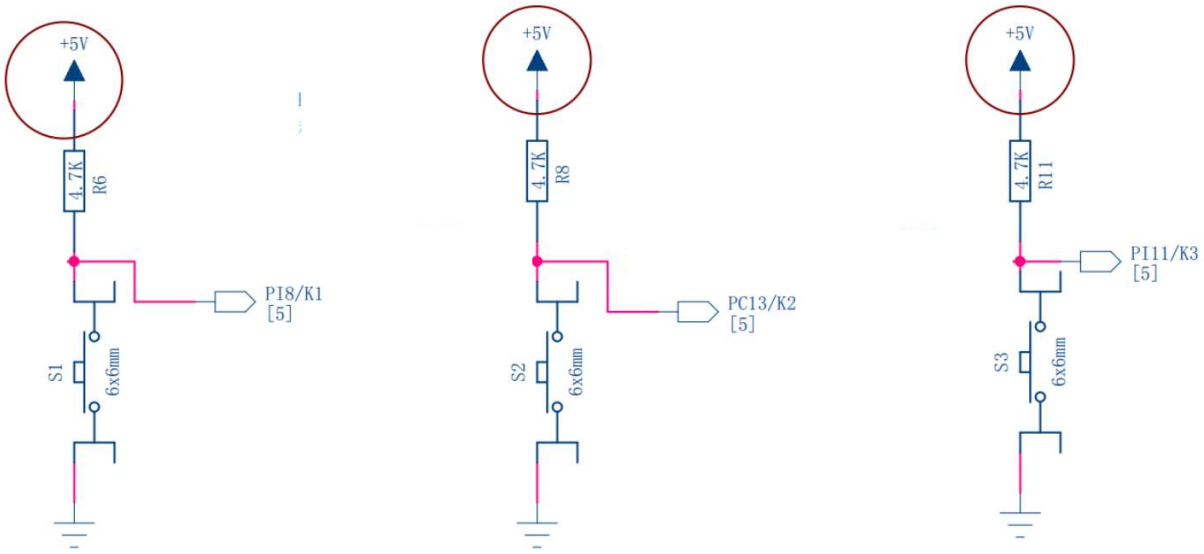
IO 库函数编程 – 按键

```
main(void)
{

  //捕获Key1
  ......

}
```

F407开发板自定义按钮



不按为高电平，按下为低电平

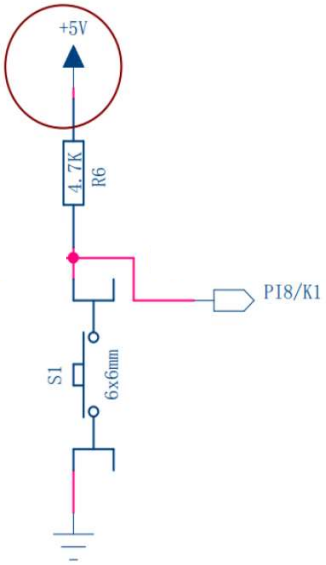
GPIO编程三步走 – 输入

- Step 1：使能GPIO端口的时钟
- Step 2：将GPIO初始化设置为通用IO输入模式
- Step 3：读取按键引脚

IO 库函数编程 – 按键

```
/*功能：通过按键1控制LED4亮灭*/
main(void){
uint8_t   keytemp1, keytemp2;
GPIO_InitTypeDef   GPIO_InitStructure;
//开启模块时钟
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC , ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOI  , ENABLE);
//指定用作GPIO输出（PC2/LED4）
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_Init(GPIOC, &GPIO_InitStructure);
```

（接下页）



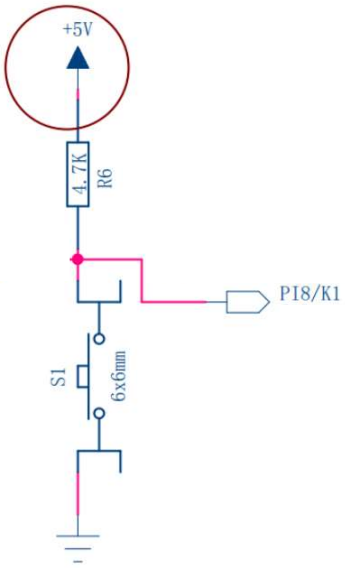
不按为高电平，按下为低电平

IO 库函数编程 – 按键

（接上页）

```
//指定用作GPIO输入（Key1/PI8）
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
GPIO_Init(GPIOI, &GPIO_InitStructure);
//控制引脚输出电平（PC2=0，LED4亮）
GPIO_ResetBits(GPIOC, GPIO_Pin_2);
//按键切换亮灭
while(1){
  keytemp1 = GPIO_ReadInputDataBit(GPIOI, GPIO_Pin_8);
  delayms(10);
  keytemp2 = GPIO_ReadInputDataBit(GPIOI, GPIO_Pin_8);
```

（接下页）



不按为高电平，按下为低电平

IO 库函数编程 – 按键

(接上页)

```
        if((keytemp1 | keytemp2) == 0)
        {
          GPIO_ToggleBits(GPIOC, GPIO_Pin_2);
          delayms(1000);
        }
    }
}
```

仅凭下面代码能否看出操作的何种器件及效果?

1）GPIO_ToggleBits(GPIOC, GPIO_Pin_2);

2）keytemp = GPIO_ReadInputDataBit(GPIOI, GPIO_Pin_8);

存在问题：代码还是不够直观

解决方法：对端口/引脚进行二次更明确的定义

用户设备的建库及应用

- **适用对象**
  - -- 片外设备，如LED、按键、显示部件，……
  - -- 片上外设，高于标准库函数之上的操作，如中断操作、定时操作，……
- **库文件包括什么?**
  - 头文件 (hardware.h)

    -- 硬件占用端口的宏定义、变量定义、操作函数声明，……
  - 源文件 (hardware.c)

    -- 操作函数的具体实现代码，如点灯、闪烁、按键获取，……

用户设备的建库 – LED

- led.h 头文件
  - **控制端口宏定义**

```
                用户宏定义      标准库中宏定义
#define   LED4_RCC      RCC_AHB1Periph_GPIOC   //所属外设时钟
#define   LED4_Port     GPIOC                  //所属端口
#define   LED4_Pin      GPIO_Pin_2             //所属引脚
 ……
```

  - **控制函数声明**

```
void LedInit (void);          //初始化函数
void LedOn (Led_t);           //亮灯函数
void LedOff (Led_t);          //灭灯函数
void LedToggle (Led_t);       //亮灭切换
```

用户设备的建库 – LED

- led.c 源文件
  - **初始化LED端口函数**

```
void LedInit (void)
{       GPIO_InitTypeDef    GPIO_InitStructure;
        /*打开时钟*/
        RCC_AHB1PeriphClockCmd(LED4_RCC , ENABLE);
        /*配置端口*/
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;   //输出模式
        GPIO_InitStructure.GPIO_Pin = LED4_Pin;                 //选定引脚
           /*完成配置*/
        GPIO_Init(LED4_Port,  &GPIO_InitStructure);
}
```

用户设备的建库 – LED

- led.c 源文件
  - **亮灯函数**

```
void LedOn (Led_t   led)   /*形参led为要点亮的灯*/
{
   if(led == LED3)   //是3号灯
        GPIO_ResetBits (LED3_Port , LED3_Pin);
   if(led == LED4)   //是4号灯
        GPIO_ResetBits (LED4_Port , LED4_Pin);
  .......
}
```

(led.h)

```
//定义LED枚举类型
typedef  enum
{
      LED1 = 1,
      LED2 = 2,
      LED3 = 3,
      LED4 = 4,
}Led_t;
```

用户设备的建库 – LED

- **灭灯函数**

```
void LedOff (Led_t   led)
 {
    if(led == LED4)
        GPIO_SetBits (LED4_Port , LED4_Pin);
  .......
}
```

- **亮灭切换**

```
void LedToggle (Led_t   led)
{
   if(led == LED4)
        GPIO_Toggle (LED4_Port , LED4_Pin);
  .......
}
```

用户设备的建库 – 按键

- key.h 头文件
  - **控制端口宏定义**

```
                用户宏定义      标准库中宏定义
#define   KEY1_RCC      RCC_AHB1Periph_GPIOI   //所属时钟
#define   KEY1_Port     GPIOI                  //所属端口
#define   KEY1_Pin      GPIO_Pin_8             //所属引脚
 ......
```

  - **控制函数声明**

```
void KeyInit (void);        //初始化函数
void GetKey (void);         //捕获按键
......
```

用户设备的建库 – 按键

- key.c 源文件
  - **初始化按键端口函数**

```
void KeyInit (void)
{       GPIO_InitTypeDef    GPIO_InitStructure;
        /*打开时钟*/
        RCC_AHB1PeriphClockCmd(KEY1_RCC , ENABLE);
        /*配置端口*/
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;      //输入模式
        GPIO_InitStructure.GPIO_Pin = KEY1_Pin;           //选定引脚
           /*完成配置*/
        GPIO_Init(KEY1_Port,  &GPIO_InitStructure);
}
```

用户设备的建库 – 按键

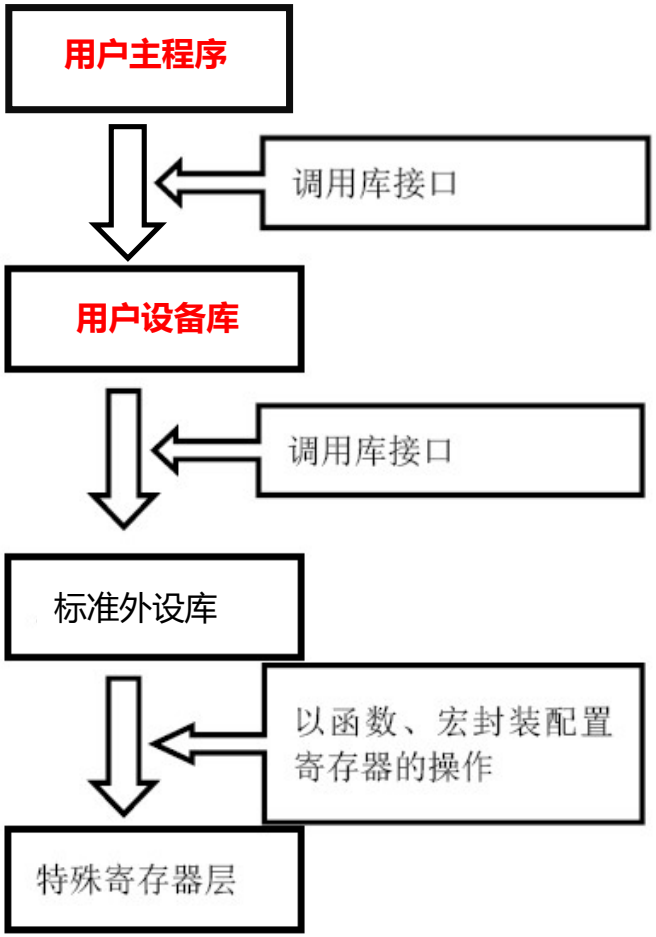
- key.c 源文件
  - **捕获按键函数**

```
Key_t  GetKey (void)          /*返回值代表按键(值)  */
{  uint8_t  temp1, temp2;
   Key_t   key = KEY_NO;      //默认无键按下
  /*读取按键引脚*/
   temp1 = GPIO_ReadInputDataBit (KEY1_PORT, KEY1_Pin);
   temp2 = GPIO_ReadInputDataBit (KEY2_PORT, KEY2_Pin);
  /*判断按键状态*/
   if (temp1 == 0x00)   key = KEY1;
   if (temp2 == 0x00)   key = KEY2;
   return key;    //返回键值
}
```

(key.h)

```
//定义Key枚举类型
typedef  enum
{
      KEY_NO = 0,
      KEY1 = 1,
      KEY2 = 2,
      ……
}Key_t;
```

用户设备库的应用



用户设备库的应用

- **用户主程序main.c – 不同按键闪烁不同LED**

```
int main(void)
{    Key_t   keytemp1, keytemp2;
     /*LED、Key端口初始化*/
     LedInit();
     KeyInit();
     /*点亮LED*/
     LedOn(LED4);
     /*按键切换亮灭*/
     while(1) {
          keytemp1 = GetKey();      //第一次按键检测
          delayms(10);
          keytemp2 = GetKey();      //第二次检测
```

用户设备库的应用

- **main.c – 续**

```
if(keytemp1 == keytemp2)
{   /*根据键值处理*/
    switch (keytemp1)
    { /*无键按下*/
      case  KEY_NO:
      break;
      /*1号键按下*/
      case  KEY1:
      LedToggle (LED4);
      delayms(1000);
      break;     } } } }
```

✓ 编程练习E1.5：库函数编程实现按键控制LED

(配套视频E1.5.1)

思考题

- STM32每组端口共多少个引脚？GPIO引脚的命名规则，如PA0三个字符各代表什么？
- IO引脚的控制逻辑中通常有哪三类寄存器？大致功能如何
- 解释什么是推挽输出，解释为何采用IO输出低电平点亮LED
- GPIOA被映射到存储空间的哪个地址段？
- 什么是CMSIS？它的两个核心层是什么？
- 什么是STM32的标准函数库？其文件命名规则，如STM32Fxxx_ppp.c，xxx和ppt代表什么？
- 解释寄存器操作语句“GPIOC->MODER = ...”的机理
- int16_t和uint16_t分别代表什么数据类型？表示的数据范围各是多少？
- 头文件和源文件的用途
- 结构体和枚举类型的区别？在GPIO初始化中有什么应用（可举例）？
- 库函数编程和寄存器编程的优劣比较
- 什么情况需要用户再建立设备库文件（标准库之上）？

致谢

- 部分图片和文字来自网络、学堂在线慕课《ARM微控制器与嵌入式系统》

第3章 STM32开发初步

— IO外设及软件基础