Principles and Practices of Microcontroller (Embedded System Design I) -IO1

Gang Chen (陈刚)

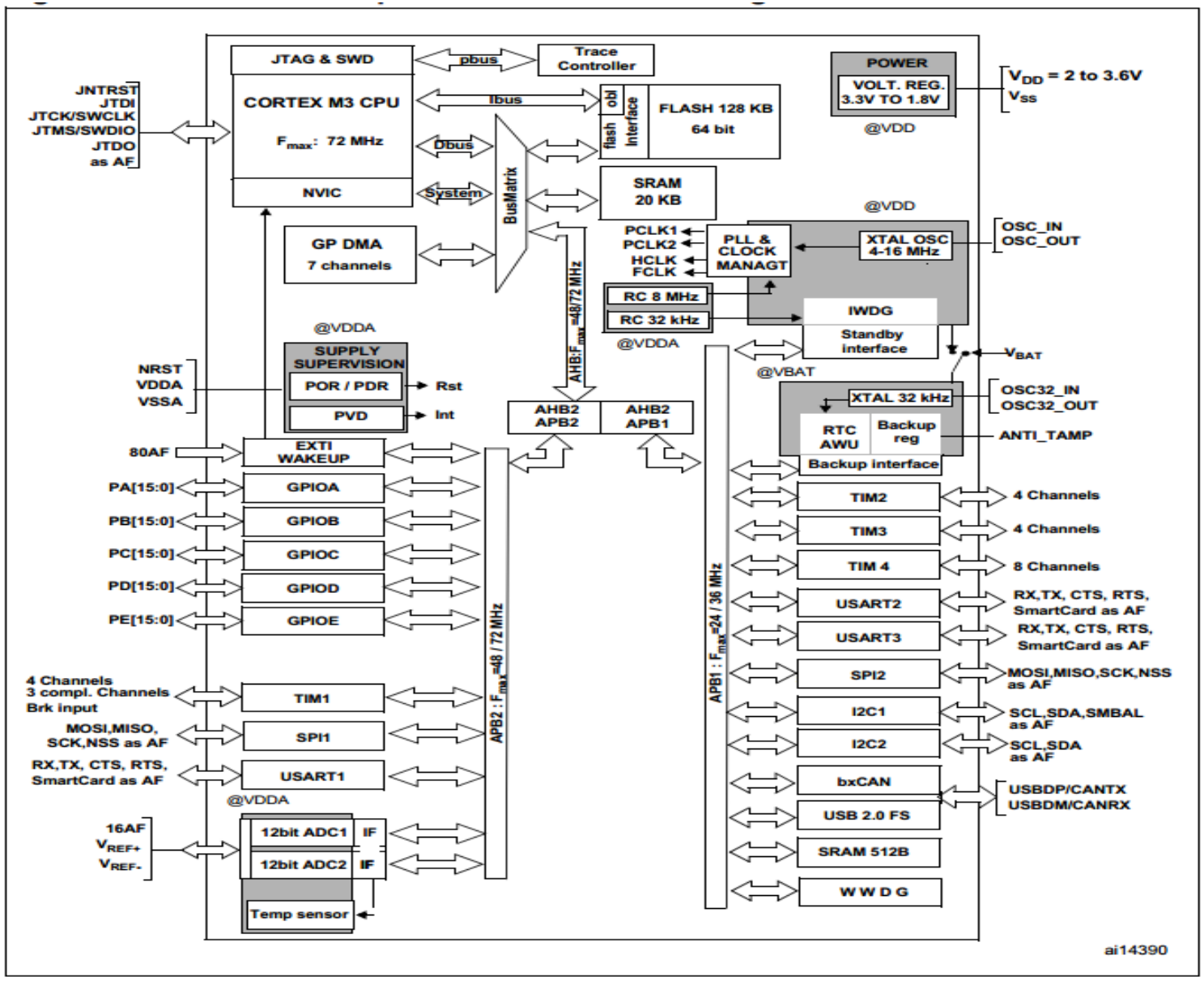
Associate Professor

Institute of Unmanned Systems
School of data and computer science
Sun Yat-Sen University

https://www.usilab.cn/team/chengang/

中山大學 SUN YAT-SEN UNIVERSITY 数据科学与计算机学院 School of Data and Computer Science

pin配置


JTAG & SWD
pbus
Trace Controller
JNTRST
JTDI
JTCK/SWCLK
JTMS/SWDIO
JTDO
as AF
CORTEX M3 CPU
Fmax: 72 MHz
NVIC
Ibus
Dbus
System
BusMatrix
obl
flash
Interface
FLASH 128 KB
64 bit
SRAM
20 KB
POWER
VOLT. REG.
3.3V TO 1.8V
@VDD
VDD = 2 to 3.6V
Vss
GP DMA
7 channels
PCLK1
PCLK2
HCLK
FCLK
PLL & CLOCK MANAGT
XTAL OSC
4-16 MHz
OSC_IN
OSC_OUT
RC 8 MHz
RC 32 kHz
IWDG
Standby interface
VBAT
@VBAT
@VDDA
SUPPLY SUPERVISION
POR / PDR
PVD
Rst
Int
NRST
VDDA
VSSA
AHB:Fmax=48/72 MHz
AHB2
APB2
AHB2
APB1
XTAL 32 kHz
OSC32_IN
OSC32_OUT
RTC
AWU
Backup reg
ANTI_TAMP
Backup interface
80AF
EXTI
WAKEUP
PA[15:0]
GPIOA
PB[15:0]
GPIOB
PC[15:0]
GPIOC
PD[15:0]
GPIOD
PE[15:0]
GPIOE
APB2 : Fmax=48 / 72 MHz
APB1 : Fmax=24 / 36 MHz
TIM2
4 Channels
TIM3
4 Channels
TIM 4
8 Channels
USART2
RX,TX, CTS, RTS, SmartCard as AF
USART3
RX,TX, CTS, RTS, SmartCard as AF
SPI2
MOSI,MISO,SCK,NSS as AF
I2C1
SCL,SDA,SMBAL as AF
I2C2
SCL,SDA as AF
bxCAN
USB 2.0 FS
USBDP/CANTX
USBDM/CANRX
SRAM 512B
W W D G
4 Channels
3 compl. Channels
Brk input
TIM1
MOSI,MISO, SCK,NSS as AF
SPI1
RX,TX, CTS, RTS, SmartCard as AF
USART1
@VDDA
16AF
VREF+
VREF-
12bit ADC1
12bit ADC2
IF
IF
Temp sensor
ai14390

嵌套矢量中断控制器（NVIC）

- **STM32F103xx**系列微控制器嵌入了一个嵌套矢量中断控制器，可以处理**43**个可屏蔽中断通道（不包括**CM3**的**16**根中断线），提供**16**个中断优先级。
- 紧密耦合的**NVIC**实现了更低的中断处理延迟。
- 直接向内核传递中断入口向量表地址。
- 紧密耦合的**NVIC**内核接口
- 允许中断提前处理。
- 对后到得更高优先级的中断进行处理。
- 支持尾链。
- 自动保存处理器状态。
- 中断入口在中断退出的时候自动恢复，不需要指令干预

外部中断/事件控制器（EXTI）

- 外部中断/事件控制器由用于19条产生中断/事件请求的边沿探测器线组成。每条线可以被单独配置用于选择触发事件（上升沿，下降沿或者两者都可以），也可以被单独屏蔽。有一个挂起寄存器来维护中断请求的状态。当外部线上出现长度超过内部APB2时钟周期的脉冲，EXTI能够探测到多达112个GPIO连接到16个外部中断线

时钟和启动

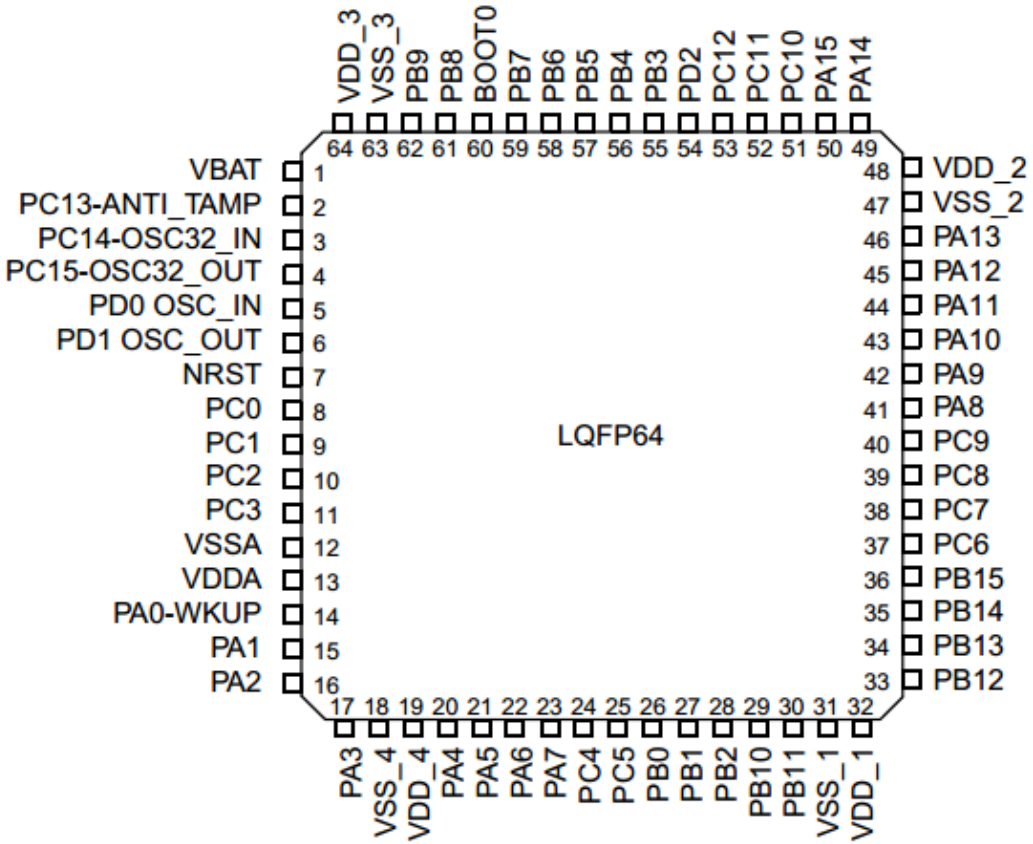
- 在启动的时候还是要进行系统时钟选择，但复位的时候内部8MHz的晶振被选作CPU时钟。可以选择一个外部的4-16MHz的时钟，并且会被监视来判定是否成功。在这期间，控制器被禁能并且软件中断管理也随后被禁能。同时，如果有需要（例如碰到一个间接使用的晶振失败），PLL时钟的中断管理完全可用。

低功耗模式

- STM32F101xx支持三种低功耗模式，从而在低功耗、短启动时间和可用唤醒源之间达到一个最好的平衡点。
- 休眠模式。在休眠模式中，只有CPU停止工作，所有的外设继续运行，在中断/事件发生的时候唤醒CPU。
- 停止模式。停止模式允许以最小的功耗来保持SRAM和寄存器的内容。1.8V区域的时钟都停止，PLL，HSI和HSE
- RC振荡器被禁能，调压器也被置为正常或者低功耗模式。设备可以通过外部中断线从停止模式唤醒。外部中断源可以使16个外部中断线之一、PVD输出或者TRC警告。
- 待机模式。

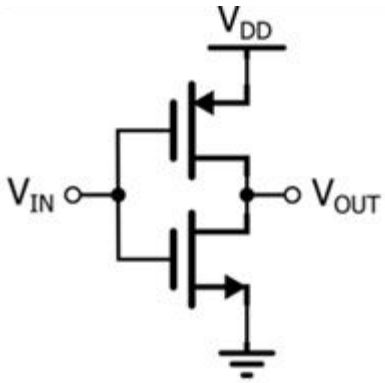
stm32f103xx

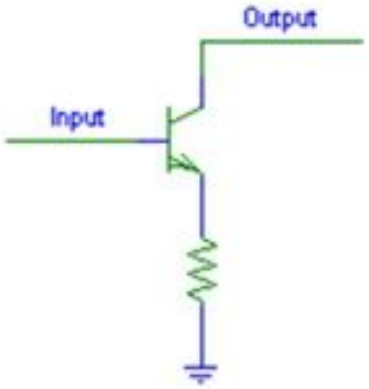
- 有PA，PB，PC，PD四组，由PA~PC每组各16(PA0~PA15，PB0~PB15，PC0~PC15)输入输出口
- 每个GPI/O端口有两个32位配置寄存器(GPIOx_CRL，GPIOx_CRH)，两个32位数据寄存器(GPIOx_IDR，GPIOx_ODR)，一个32位置位/复位寄存器(GPIOx_BSRR)，一个16位复位寄存器(GPIOx_BRR)和一个32位锁定寄存器(GPIOx_LCKR)



GPIO端口的每个位可以由软件分别配置成多种模式

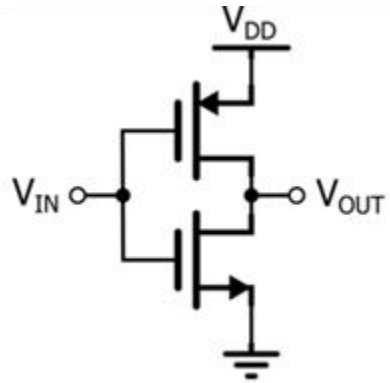
配置模式		CNF1	CNF0	MODE1	MODE0	PxODR寄存器
通用输出	推挽式(Push-Pull)	0	0	01 10 11		0 或 1
	开漏(Open-Drain)		1			0 或 1
复用功能输出	推挽式(Push-Pull)	1	0			不使用
	开漏(Open-Drain)		1			不使用
输入	模拟输入	0	0	00		不使用
	浮空输入		1			不使用
	下拉输入	1	0			0
	上拉输入					1

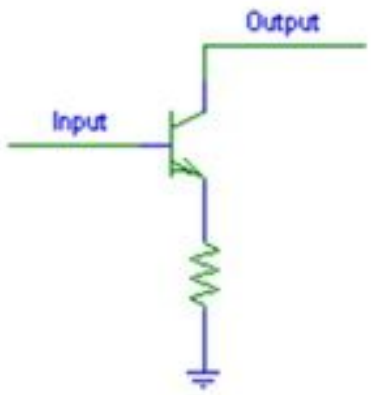




输出

- **Push out:** 能输出真正的高电平





- **Open Drian:** 对于开漏输出和推挽输出的区别最普遍的说法就是开漏输出无法真正输出高电平，即高电平时没有驱动能力，需要借助外部上拉电阻完成对外驱动
- 选哪种好？

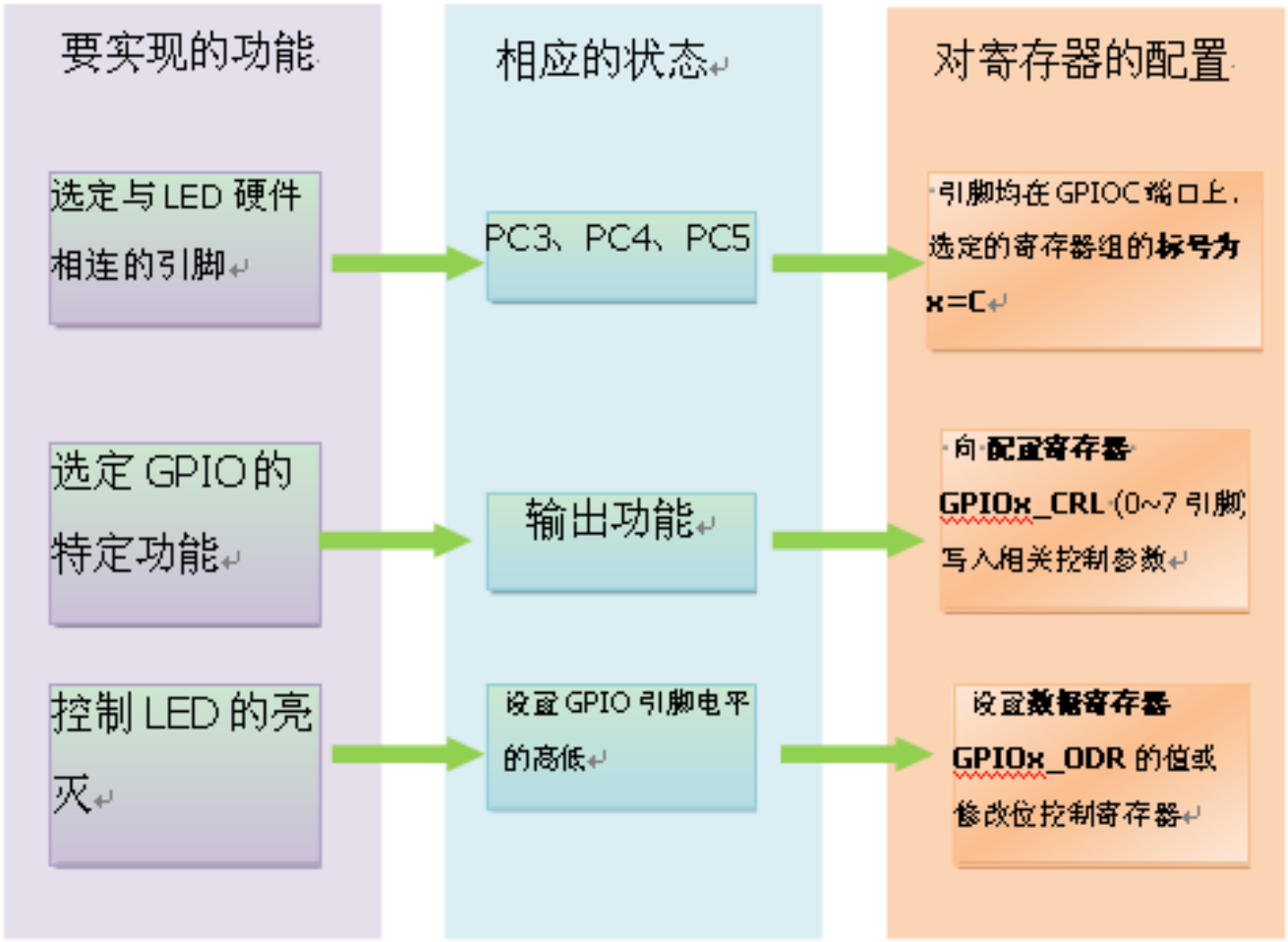
输出模式位含义

MODE[1:0]	意义
00	保留
01	最大输出速度为10MHz
10	最大输出速度为2MHz
11	最大输出速度为50MHz

- 举例:

复位期间和刚复位后，复用功能未开启，I/O端口被配置成浮空输入模式(CNFx[1:0]=01b，MODEx[1:0]=00b)

流程


要实现的功能
相应的状态
对寄存器的配置
选定与 LED 硬件相连的引脚
PC3、PC4、PC5
引脚均在 GPIOC 端口上，选定的寄存器组的标号为 x=C
选定 GPIO 的特定功能
输出功能
向配置寄存器 GPIOx_CRL (0~7 引脚) 写入相关控制参数
控制 LED 的亮灭
设置 GPIO 引脚电平的高低
设置数据寄存器 GPIOx_ODR 的值或修改位控制寄存器

配置IO

- 1）作为普通GPIO输入：根据需要配置该引脚为浮空输入、带弱上拉输入或带弱下拉输入，同时不要使能该引脚对应的所有复用功能模块。
- 2）作为普通GPIO输出：根据需要配置该引脚为推挽输出或开漏输出，同时不要使能该引脚对应的所有复用功能模块。
- 3）作为普通模拟输入：配置该引脚为模拟输入模式，同时不要使能该引脚对应的所有复用功能模块。

几种模式

- 上拉输入（GPIO_Mode_IPU）:区别在于没有输入信号的时候默认输入高电平（因为有弱上拉）。
- 下拉输入（GPIO_Mode_IPD）: 区别在于没有输入信号的时候默认输入低电平（因为有弱下拉）。
- 上拉下拉中的电阻作用就是为了达到以上目的。上拉电阻的目的是为了保证在没有信号输入时，输入端的电平为高电平。而在信号输入为低电平时，输入端的电平应该也为低电平。如果没有上拉电阻，在没有外界输入的情况下输入端是悬空的，它的电平是未知的无法保证的，上拉电阻就是为了保证没有信号输入时，输入端的电平为高电平，同样还有下拉电阻它是为了保证无信号输入时输入端的电平为低电平。
- 浮空输入（GPIO_Mode_IN_FLOATING）：顾名思义就是信号不确定，输入什么信号才是什么信号，所以要保证有明确的信号。可以做KEY识别，RX1。

几种模式

- 模拟输入（GPIO_Mode_AIN）：通常用在AD采样，或者低功耗下省电。
- 开漏输出（GPIO_Mode_Out_OD）：就是IO不输出电压，IO输出0（低电平）接GND，IO输出1（高电平）悬空，需要外接上拉电阻，才能实现输出高电平。当输出为1时，IO口的状态由上拉电阻拉高电平（拉到上拉电阻的电源电压），但由于是开漏输出模式，这样IO口也就可以由外部电路改变为低电平或不变。可以读IO输入电平变化，实现C51（微处理器）的IO双向功能。这种方式适合在连接的外设电压比微处理器电压低的时候，以及做电流型的驱动,其吸收电流的能力相对强(一般20ma以内)。（开漏的IO口上拉电平到不了5v，只有FT管脚才能达到5v，数据手册中有详细的说明。）

几种模式

- 推挽输出（GPIO_Mode_Out_PP）：推挽输出就是微处理器引脚可以直接输出高电平电压。低电平时接地，高电平时输出微处理器电源电压（IO输出0-接GND， IO输出1 -接VCC，读输入值是未知的）。这种方式可以不接上拉电阻。
- 复用开漏输出（GPIO_Mode_AF_OD）：片内外设功能（TX1， MOSI,MISO.SCK.SS）.
- 复用推挽输出（GPIO_Mode_AF_PP）：片内外设功能（I2C的SCL,SDA）
- 也就是说复用开漏输出和复用推挽输出是GPIO口被用作第二功能时的配置情况（即并非作为通用IO口使用）。

端口配置低寄存器（GPIOx_CRL）(x=A..E)

- 在GPIO地址中的偏移地址为：0x00；复位值为：0x4444 4444。功能是配置每个端口的低八位，例如：配置PA口的PA0~PA7。

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

位31:30 27:26 23:22 19:18 15:14 11:10 7:6 3:2	**CNFy[1:0]**：端口x配置位(y = 0…7) 软件通过这些位配置相应的I/O端口，请参考表15端口位配置表。 在输入模式(MODE[1:0]=00)： 00：模拟输入模式 01：浮空输入模式(复位后的状态) 10：上拉/下拉输入模式 11：保留 在输出模式(MODE[1:0]>00)： 00：通用推挽输出模式 01：通用开漏输出模式 10：复用功能推挽输出模式 11：复用功能开漏输出模式
位29:28 25:24 21:20 17:16 13:12 9:8, 5:4 1:0	**MODEy[1:0]**：端口x的模式位(y = 0…7) 软件通过这些位配置相应的I/O端口，请参考表15端口位配置表。 00：输入模式(复位后的状态) 01：输出模式，最大速度10MHz 10：输出模式，最大速度2MHz 11：输出模式，最大速度50MHz

端口配置高寄存器（GPIOx_CRH）(x=A..E)

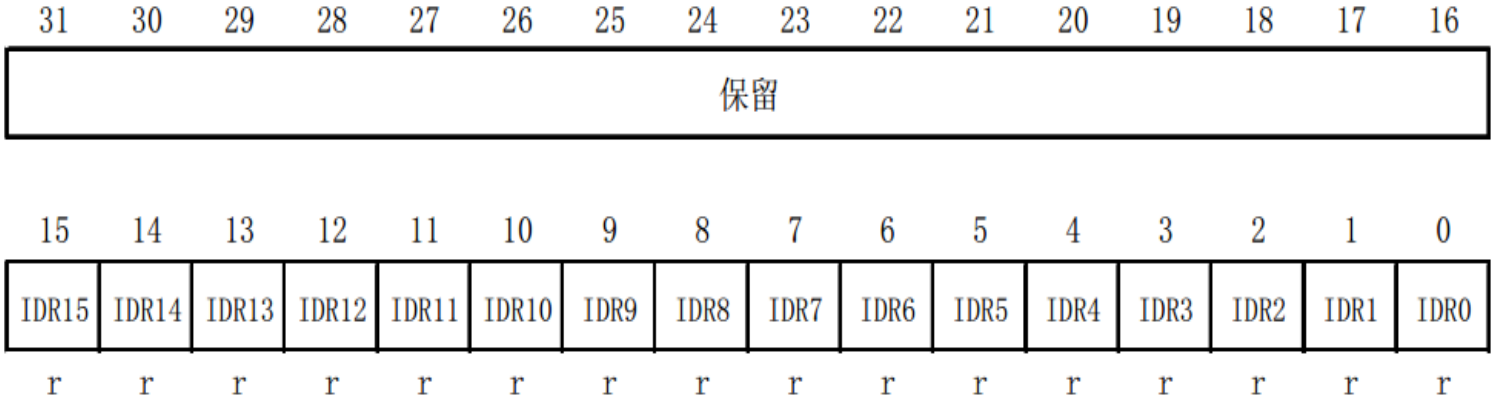
- 在GPIO地址中的偏移地址为：0x04；复位值为：0x4444 4444.
- 功能是配置每个端口的高八位，例如：配置PA口的PA8~PA15。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[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
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:30 27:26 23:22 19:18 15:14 11:10 7:6 3:2	**CNFy[1:0]**：端口x配置位(y = 8…15) 软件通过这些位配置相应的I/O端口，请参考表15端口位配置表。 在输入模式(MODE[1:0]=00)： 00：模拟输入模式 01：浮空输入模式(复位后的状态) 10：上拉/下拉输入模式 11：保留 在输出模式(MODE[1:0]>00)： 00：通用推挽输出模式 01：通用开漏输出模式 10：复用功能推挽输出模式 11：复用功能开漏输出模式
位9:28 25:24 21:20 17:16 13:12 9:8, 5:4 1:0	**MODEy[1:0]**：端口x的模式位(y = 8…15) 软件通过这些位配置相应的I/O端口，请参考表15端口位配置表。 00：输入模式(复位后的状态) 01：输出模式，最大速度10MHz 10：输出模式，最大速度2MHz 11：输出模式，最大速度50MHz

端口输入数据寄存器（GPIOx_IDR）(x=A..E)

- 在GPIO地址中的偏移地址为：0x08；复位值为：0x0000 XXXX.
- 功能是保存输入到端口的每个位的数据。



位31:16	保留，始终读为0。
位15:0	**IDRy[15:0]**：端口输入数据(y = 0…15) 这些位为只读并只能以字(16位)的形式读出。读出的值为对应I/O口的状态。

端口输出数据寄存器（GPIOx_ODR）(x=A..E)

- 在GPIO地址中的偏移地址为：0x0C；复位值为：0x0000 0000.
- 功能是保存输出到端口的每个位的数据

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

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

位31:16	保留，始终读为0。

位15:0	**ODRy[15:0]**：端口输出数据(y = 0…15) 这些位可读可写并只能以字(16位)的形式操作。 注：对GPIOx_BSRR(x = A…E)，可以分别地对各个ODR位进行独立的设置/清除。

端口位设置/清除寄存器(GPIOx_BSRR) (x=A..E)

- 在GPIO地址中的偏移地址为：0x10；复位值为：0x0000 0000.
- 功能是置位和清除在输出数据寄存器ODR的位，高16位写1完成清除，低16位写1完成置位。

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

位31:16	**BRy**: 清除端口x的位y (y = 0…15) 这些位只能写入并只能以字(16位)的形式操作。 0：对对应的ODRy位不产生影响 1：清除对应的ODRy位为0 注：如果同时设置了BSy和BRy的对应位，BSy位起作用。
位15:0	**BSy**: 设置端口x的位y (y = 0…15) 这些位只能写入并只能以字(16位)的形式操作。 0：对对应的ODRy位不产生影响 1：设置对应的ODRy位为1

端口位清除寄存器(GPIOx_BRR) (x=A..E)

- 在GPIO地址中的偏移地址为：0x14；复位值为：0x0000 0000
- 功能只有清除在输出数据寄存器ODR的位，高16位保留，低16位写1完成清除。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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

位31:16	保留。
位15:0	**BRy**: 清除端口x的位y (y = 0...15) 这些位只能写入并只能以字(16位)的形式操作。 0：对对应的ODRy位不产生影响 1：清除对应的ODRy位为0

- 以下举个例子让大家更好理解BSRR和BRR这两个寄存器。假设GPIOA的16个IO都被设置成输出，而每次操作仅需要改变低8位的数据而保持高8位不变，假设新的8位数据在变量data中，可以通过操作这两个寄存器实现，STM32的固件库中有两个函数GPIO_SetBits()和GPIO_ResetBits()使用了这两个寄存器操作端口。

- GPIO_SetBits(GPIOE, data & 0xff);
- GPIO_ResetBits(GPIOE, (~data & 0xff))
- 也可以直接操作这两个寄存器：
- GPIOE->BSRR = data & 0xff;
- GPIOE->BRR = ~data & 0xff;
- 还可以一次完成对8位的操作：
- GPIOE->BSRR = (data & 0xff) | (~data & 0xff)<<16;
- 位操作
- GPIO_ResetBits(GPIOC, GPIO_Pin_13);
- GPIO_SetBits(GPIOC, GPIO_Pin_13);

- 使用BRR和BSRR寄存器可以方便地快速地实现对端口某些特定位的操作，而不影响其它位的状态。例如如希望快速地对GPIOE的位7进行翻转，则可以：
- GPIOE->BSRR = 0x80; // 置'1'
- GPIOE->BRR = 0x80; // 置'0'

端口配置锁定寄存器(GPIOx_LCKR) (x=A..E)

- 在GPIO地址中的偏移地址为：0x18；复位值为：0x0000 0000。功能是锁定端口设置。

位31:17	保留。
位16	LCKK：锁键 该位可随时读出，它只可通过锁键写入序列修改。 0：端口配置锁键位激活 1：端口配置锁键位被激活，下次系统复位前GPIOx_LCKR寄存器被锁住。 锁键的写入序列： 写1 -> 写0 -> 写1 -> 读0 -> 读1 最后一个读可省略，但可以用来确认锁键已被激活。 注：在操作锁键的写入序列时，不能改变LCK[15:0]的值。 操作锁键写入序列中的任何错误将不能激活锁键。
位15:0	LCKy: 端口x的锁位y (y = 0…15) 这些位可读可写但只能在LCKK位为0时写入。 0：不锁定端口的配置 1：锁定端口的配置

复用I/O配置寄存器

◆端口复用器

STM32F4的大部分端口都具有复用功能。

所谓复用，就是一些端口不仅仅可以做为通用IO口，还可以复用为一些外设引脚，比如PA9,PA10可以复用为STM32F4的串口1引脚。

作用：最大限度的利用端口资源

复用功能	USART1_REMAP = 0	USART1_REMAP = 1
USART1_TX	PA9	PB6
USART1_RX	PA10	PB7

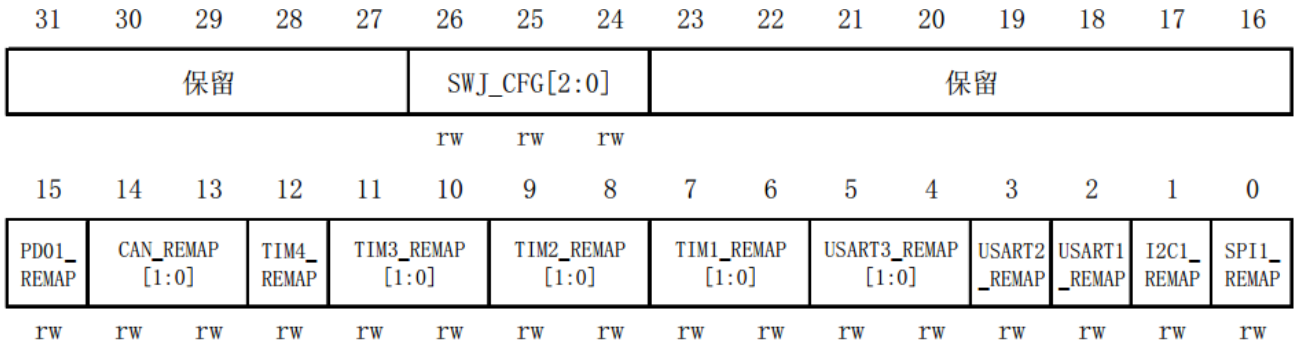
复用有关寄存器---事件控制寄存器(AFIO_EVCR)

- 在AFIO地址中的偏移地址为：0x00；复位值为：0x0000 0000.
- 功能控制CORTEX-M3内部事件输出到相应引脚

位31:8	保留。
位7	EVOE：允许事件输出 该位可由软件读写。当设置该位后，Cortex的EVENTOUT将连接到由PORT[2:0]和PIN[3:0]选定的I/O口。
位6:4	PORT[2:0]：端口选择 选择用于输出Cortex的EVENTOUT信号的端口： 000：选择PA　001：选择PB　010：选择PC　011：选择PD 100：选择PE
位3:0	PIN[3:0]：管脚选择 选择用于输出Cortex的EVENTOUT信号的管脚： 0000：选择Px0　0001：选择Px1　0010：选择Px2　0011：选择Px3 0100：选择Px4　0101：选择Px5　0110：选择Px6　0111：选择Px7 1000：选择Px8　1001：选择Px9　1010：选择Px10　1011：选择Px11 1100：选择Px12　1101：选择Px13　1110：选择Px14　1111：选择Px15

复用重映射和调试I/O配置寄存器(AFIO_MAPR)

- 在AFIO地址中的偏移地址为：0x04；复位值为：0x0000 0000.
- 功能实现引脚的重新映射,所有功能重新映射都是通过配置它来实现的



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留					SWJ_CFG[2:0]			保留							
					rw	rw	rw								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PD01_REMAP	CAN_REMAP[1:0]		TIM4_REMAP	TIM3_REMAP[1:0]		TIM2_REMAP[1:0]		TIM1_REMAP[1:0]		USART3_REMAP[1:0]		USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:27	保留。
位26:24	SWJ_CFG[2:0]：串行线JTAG配置 这些位可由软件读写，用于配置SWJ和跟踪复用功能的I/O口。SWJ(串行线JTAG)支持JTAG或SWD访问Cortex的调试端口。系统复位后的默认状态是启用SWJ但没有跟踪功能，这种状态下可以通过JTMS/JTCK脚上的特定信号选择JTAG或SW(串行线)模式。 000：完全SWJ(JTAG-DP + SW-DP)：复位状态 001：完全SWJ(JTAG-DP + SW-DP)但没有JNTRST 010：关闭JTAG-DP，启用SW-DP 100：关闭JTAG-DP，关闭SW-DP 其它组合：禁用
位23:16	保留。
位15	PD01_REMAP：端口D0/端口D1映像到OSC_IN/OSC_OUT 该位可由软件读写，它控制PD0和PD1的GPIO功能映像。当不使用主振荡器HSE时(系统运行于内部的8MHz阻容振荡器)PD0和PD1可以映像到OSC_IN和OSC_OUT引脚。此功能只能适用于36、48和64管脚的封装(PD0和PD1出现在TQFP100的封装上，不必重映像)。 0：不进行PD0和PD1的重映像 1：PD0映像到OSC_IN，PD1映像到OSC_OUT。
位14:13	CAN_REMAP[1:0]：CAN复用功能重映像 这些位可由软件读写，控制复用功能CANRX和CANTX的重映像 00：CANRX映像到PA11，CANTX映像到PA12 01：未用组合 10：CANRX映像到PB8，CANTX映像到PB9(不能用于36脚的封装) 11：CANRX映像到PD0，CANTX映像到PD1(只适用于100脚的封装)

外部中断配置寄存器（AFIO_EXTICRx）(x=1…4)

- 在AFIO地址中的偏移地址依次为：0x08,0x0C,0x10,0x14；复位值都为：0x0000。功能如下：
- AFIO_EXTICR1（外部中断配置寄存器1），配置外部中断EXTI0,1,2,3给Px0,Px1,Px2,Px3引脚(x=A,B,C,D,E…)，

位31:16	保留。
位15:0	EXTIx[3:0]：EXTIx配置(x = 0 … 3) 这些位可由软件读写，用于选择EXTIx外部中断的输入源。参看6.2.5节外部中断/事件线映像。 0000：PA[x]脚 0001：PB[x]脚 0010：PC[x]脚 0011：PD[x]脚 0100：PE[x]脚

通用I/O和AFIO使用的配置步骤

- 因为CortexM3是时钟驱动型，STM32的每一个端口在使用前都要将其时钟使能， GPIO的时钟统一挂接在APB2总线上，具体的使能寄存器为RCC_APB2ENR，该寄存器的2位到6位（该寄存器的第3位到第7位）分别控制GPIOx（x=A,B,C,D,E)端口的时钟使能，当外设时钟没有启用时，程序不能读出外设寄存器的数值。
- 例如：使能PA时钟： RCC->APB2ENR|=1<<2; //使能PORTA时钟

使用固件库时，GPIO的操作步骤

- 1）定义GPIO_InitTypeDef类型变量,
- 例如：GPIO_InitTypeDef GPIO_InitStructure;
- 2）调用RCC_APB2PeriphClockCmd()函数使能相应的GPIOx端口（所有GPIO端口都挂载到APB2总线上的）时钟。注意：使能GPIOx端口时钟的函数必须在GPIO端口配置函数之前调用，否则GPIO端口会初始化不成功；
- 3）调用GPIO_DeInit()函数初始化要使用的GPIOx端口所属寄存器为默认值；（可略）
- 4）调用GPIO_StructInit()函数初始化前面定义的GPIO_InitTypeDef类型变量为默认值；（可略）
- 5）按开发者的需求给GPIO_InitTypeDef类型变量相应成员赋值，然后调用GPIO_Init()函数实现GPIO端口的初始化；
- 6）以上步骤完成后，就可以对相应的端口进行操作了：读/写，或配置为复用功能（步骤中出现过的函数详情，请查看固件库）。

实例代码

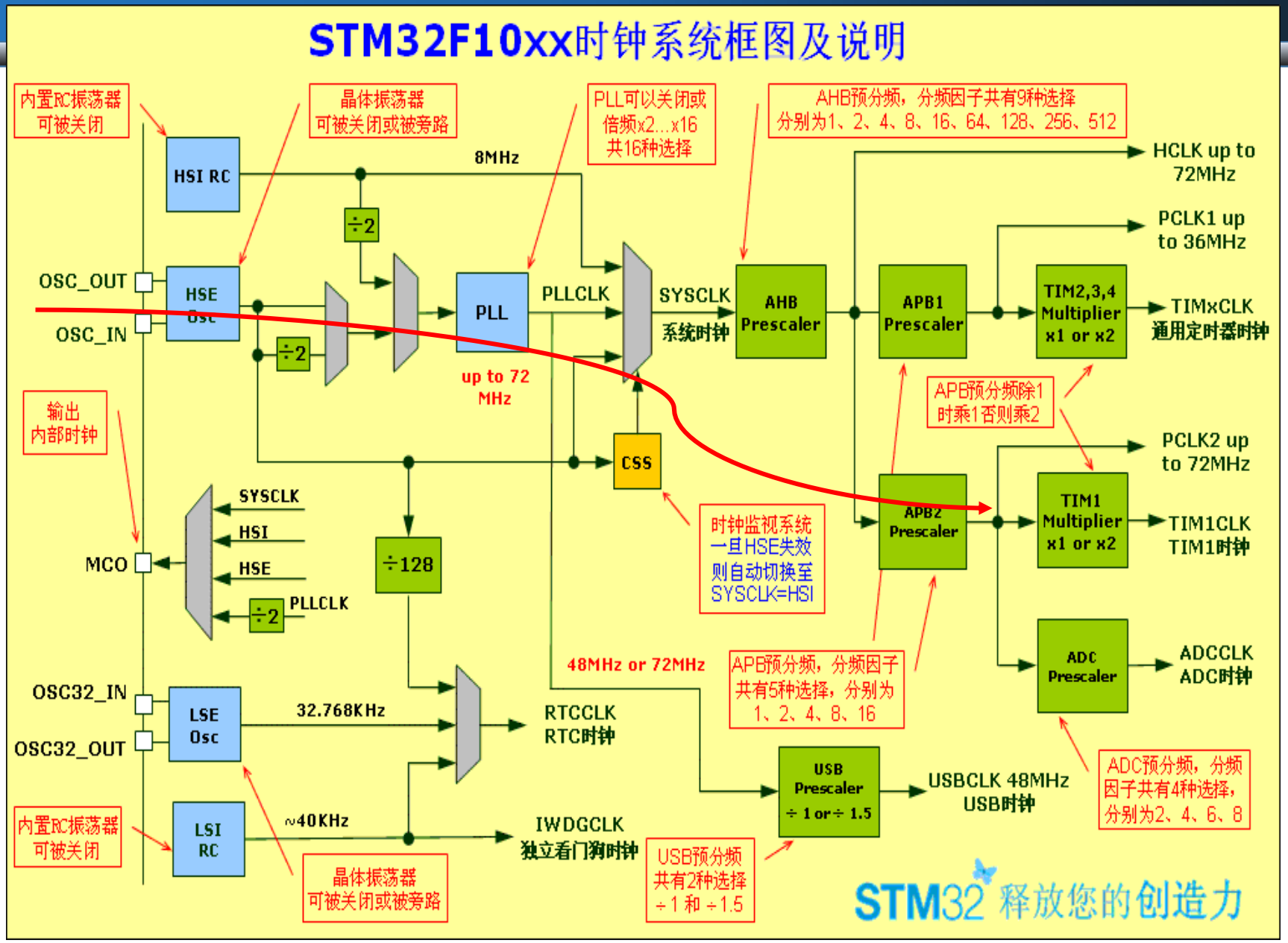
```
GPIO_InitTypeDef GPIO_InitStructure;
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC,ENABLE);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13；
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP；
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz；
GPIO_Init(GPIOC, &GPIO_InitStructure);
```

配置例子

- 1）时钟配置
- RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO, ENABLE);

- //重映射时钟
- RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,ENABLE);
- //GPIOC 时钟
- RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3,ENABLE); //USART时钟
- 也可写成
- RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOC, ENABLE);
- RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3,ENABLE);

STM32F10xx时钟系统框图及说明



- 2)GPIO配置
- GPIO_InitTypeDef GPIO_InitStructure;
- /*USART3管脚设置*/
- GPIO_PinRemapConfig(GPIO_PartialRemap_USART3, ENABLE);
- //将USART3局部重映射到PC10，PC11
-
- GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
- //管脚10
- GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
- GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
- //复用推挽输出
- GPIO_Init(GPIOC, &GPIO_InitStructure);
- //TX初始化
- GPIO_InitStructure.GPIO_Pin = GPIO_Pin_11;
- //管脚11
- GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
- //浮空输入
- GPIO_Init(GPIOC, &GPIO_InitStructure);

输入输出常用固件库函数

- **GPIO_DeInit函数**

函数名	GPIO_DeInit
函数原型	void GPIO_DeInit(GPIO_TypeDef* GPIOx)
功能描述	将GPIOx外设寄存器重设为它们的缺省值
输入	GPIOx：x可以是(A..G)，来选择GPIO外设
输出	无
返回值	无

- **GPIO_AFIODeInit函数**

函数名	GPIO_AFIODeInit
函数原型	void GPIO_AFIODeInit(void)
功能描述	将复用功能（重映射时间控制和EXTI配置）重设为默认值
输入	无
输出	无
返回值	无

GPIO_TypeDef

- **GPIO_Init函数**

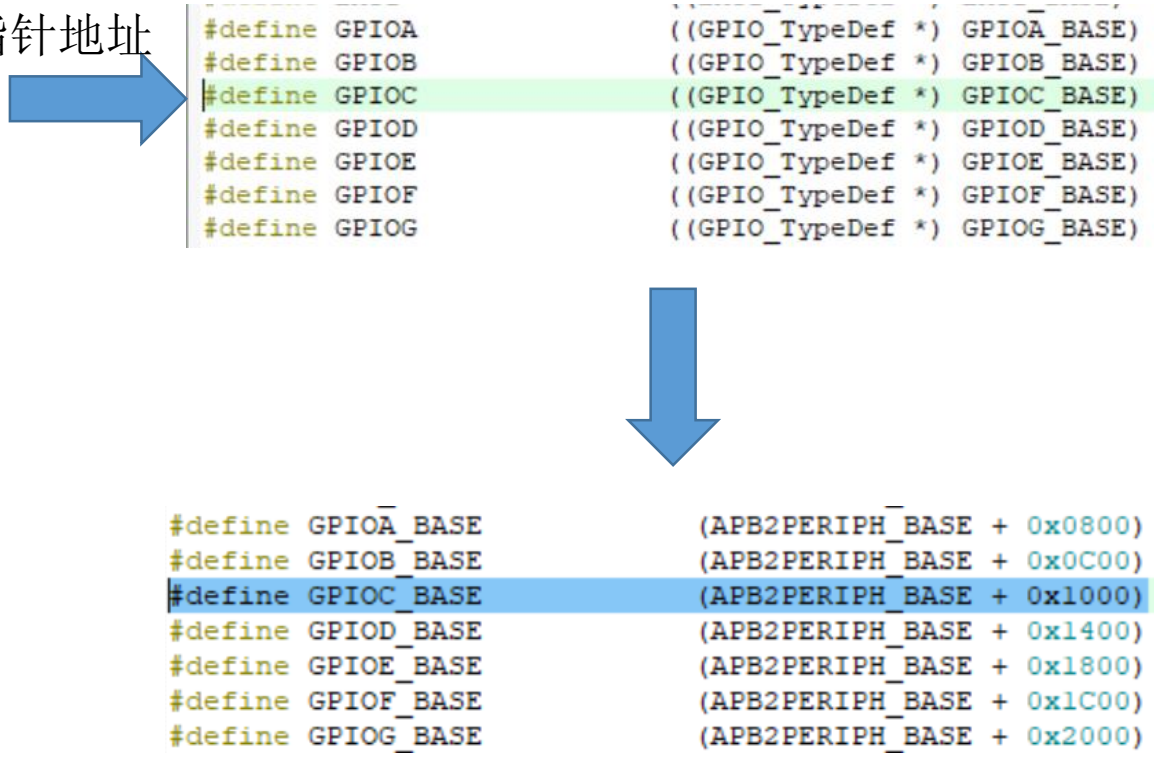
函数名	GPIO_Init
函数原型	void GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
功能描述	根据GPIO_IintStruct中指定已赋值初始化外设GPIOx寄存器
输入	GPIOx：x可以是(A..G)，来选择外设。 GPIO_IintStruct：指向结构GPIO_InitTypeDef的指针，包含了外设GPIO的配置信息。（）
输出	无
返回值	无

例如：/* 配置所有的GPIOA管脚为输入浮动模式 */

```
GPIO_InitTypeDef  GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

- typedef struct
- {
- __IO uint32_t CRL;
- __IO uint32_t CRH;
- __IO uint32_t IDR;
- __IO uint32_t ODR;
- __IO uint32_t BSRR;
- __IO uint32_t BRR;
- __IO uint32_t LCKR;
- } GPIO_TypeDef; 寄存器地址

指针地址



- **GPIO_ResetBits函数**

函数名	GPIO_ResetBits
函数原型	void GPIO_ResetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	清除指定的GPIO端口位
输入	GPIOx：x可以是(A..G)，来选择外设。 GPIO_Pin：待清除的端口位，这个参数的值是GPIO_Pin_x，其中x 是 (0..15)。
输出	无
返回值	无

例如：/* 清除GPIOB端口的第7和第10号引脚 */
GPIO_ResetBits(GPIOB, GPIO_Pin_7 | GPIO_Pin_10);

- **GPIO_SetBits函数**

函数名	GPIO_SetBits
函数原型	void GPIO_SetBits(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	置位指定的GPIO端口位
输入	GPIOx：x可以是(A..G)，来选择外设。 GPIO_Pin：待清除的端口位，这个参数的值是GPIO_Pin_x，其中x 是 (0..15)。
输出	无
返回值	无

例如：/*置位GPIOB端口的第7和第10号引脚 */
GPIO_SetBits(GPIOB, GPIO_Pin_7 | GPIO_Pin_10);

- **GPIO_ReadInputDataBit函数**

函数名	GPIO_ReadInputDataBit
函数原型	U8 GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, u16 GPIO_Pin)
功能描述	置位指定的GPIO端口位
输入	GPIOx：x可以是(A..G)，来选择外设。 GPIO_Pin：待清除的端口位，这个参数的值是GPIO_Pin_x，其中x 是 (0..15)。
输出	
返回值	输出位的值

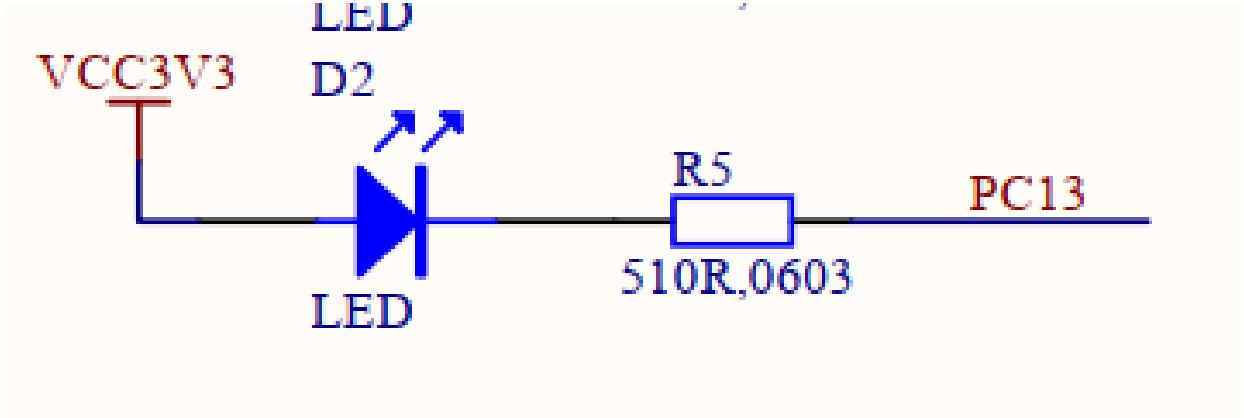
- **GPIO_ReadInputData函数**

函数名	GPIO_ReadInputData
函数原型	U8 GPIO_ReadInputData(GPIO_TypeDef* GPIOx)
功能描述	读取指定的GPIO端口
输入	GPIOx：x可以是(A..G)，来选择外设。 GPIO_Pin：待清除的端口位，这个参数的值是GPIO_Pin_x，其中x 是 (0..15)。
输出	
返回值	输出GPIO的数值

- **GPIO_ReadOutputData函数**

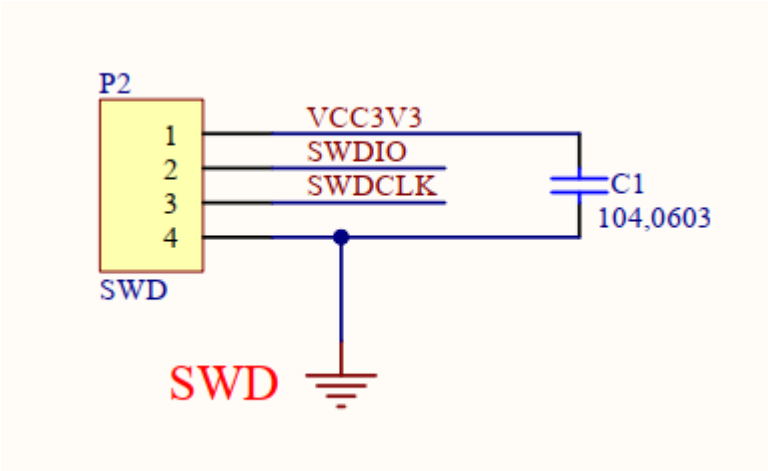
函数名	GPIO_ReadOutputData
函数原型	U8 GPIO_ReadOutputData(GPIO_TypeDef* GPIOx)
功能描述	置位指定的GPIO端口位
输入	GPIOx：x可以是(A..G)，来选择外设。 GPIO_Pin：待清除的端口位，这个参数的值是GPIO_Pin_x，其中x 是 (0..15)。
输出	
返回值	输出GPIO的数值

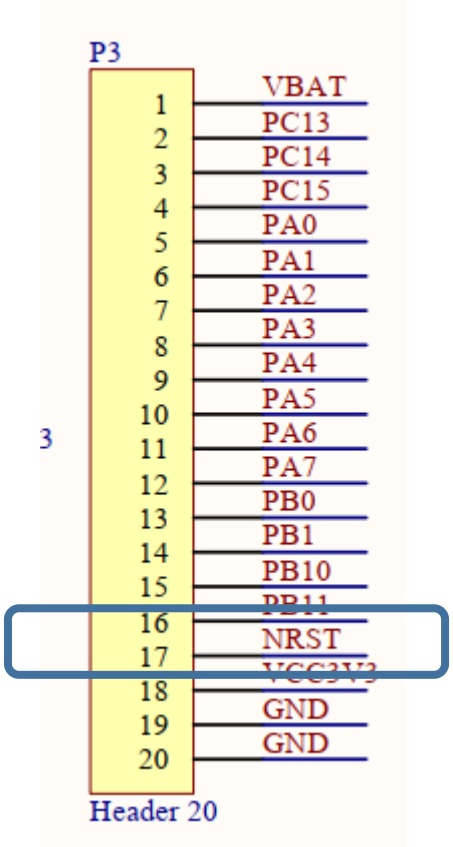
点灯实验



下载

- **SWD+Reset**
- **同时板子要供电（USB供电）**





软件准备

- 库函数介绍
  - 头文件： stm32f4xx_gpio.h
  - 源文件： stm32f4xx_gpio.c

- **http://www.st.com/en/embedded-software/stm32-standard-peripheral-libraries.html?querycriteria=productId=LN1939**

下载地址

- http://www.st.com/en/embedded-software/stm32-standard-peripheral-libraries.html?querycriteria=productId=LN1939

GPIO库函数介绍

◆重要函数：

- 1个初始化函数：
- void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);

- 2个读取输入电平函数：
- uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
- uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);

- 2个读取输出电平函数：
- uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
- uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);

- 4个设置输出电平函数：
- void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
- void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
- void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
- void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);

GPIO库函数介绍

◆ 2个读取输入电平函数：

- **uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);**
- 作用：读取某个GPIO的输入电平。实际操作的是GPIOx_IDR寄存器。

- 例如：
- GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_5);//读取GPIOA.5的输入电平

- **uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);**
- 作用：读取某组GPIO的输入电平。实际操作的是GPIOx_IDR寄存器。
- 例如：
- GPIO_ReadInputData(GPIOA);//读取GPIOA组中所有io口输入电平

GPIO库函数介绍

◆ 2个读取输出电平函数：

uint8_t GPIO_ReadOutputDataBit (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
作用：读取某个**GPIO**的输出电平。**实际操作的是GPIO_ODR寄存器。**

例如：
GPIO_ReadOutputDataBit(GPIOA, GPIO_Pin_5);//读取GPIOA.5的输出电平

uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
作用：读取某组**GPIO**的输出电平。**实际操作的是GPIO_ODR寄存器。**
例如：
GPIO_ReadOutputData(GPIOA);//读取GPIOA组中所有io口输出电平

2 GPIO库函数介绍

◆ 4个设置输出电平函数：

void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
作用：设置某个IO口输出为高电平（1）。**实际操作BSRRL寄存器**

void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
作用：设置某个IO口输出为低电平（0）。**实际操作的BSRRH寄存器。**

void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
这两个函数不常用，也是用来设置IO口输出电平。

初始化

- 使能IO口时钟。调用函数RCC_AHB1PeriphClockCmd();
  不同的外设调用的时钟使能函数可能不一样
- 初始化IO口模式。调用函数GPIO_Init();
- 操作IO口，输出高低电平。
  GPIO_SetBits();
  GPIO_ResetBits();

```
int main()
{
        uint32_t i = 0;
        uint32_t j = 0;

        GPIO_InitTypeDef GPIO_InitStructure;
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC,ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;//???????GPIOB??
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//?????????????
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//???????50MHZ
        GPIO_Init(GPIOC, &GPIO_InitStructure);

 while(1){
                for(i=0;i<0xffff;i++)
                        for(j=0;j<0xf;j++);
                GPIO_ResetBits(GPIOC, GPIO_Pin_13);
                for(i=0;i<0xffff;i++)
                        for(j=0;j<0xf;j++);
                GPIO_SetBits(GPIOC, GPIO_Pin_13);
        }
 return 0;
}
```

实验2：按键控制灯亮灭

- 按键按下是0，正常是1
- 功能1：按下键，灯熄灭，松开灯亮
- 功能2：按键一次，灯状态变化一次

- 板子上没有按键，用IO接VCC33何GND模拟
- 输入：PA0

- How？