嵌入式系统 课程号： 4240077

3. GPIO编程

胡四泉

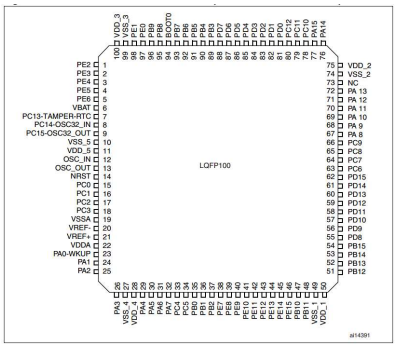
北京科技大学计算机与通信工程学院

1

GPIO简介

GPIO—general purpose intput output

是通用输入输出端口的简称，简单来说就是软件可控制的引脚，STM32芯片的GPIO引脚与外部设备连接起来，从而实现与外部通讯、控制以及数据采集的功能



STM32F103VET6引脚图

1、GPIO跟引脚有什么区别?

2、如何查找每一个GPIO的功能说明?

2

GPIO简介

STM32F10x系列引脚分类

引脚分类	引脚说明说明
电源	(VBAT)、(VDD VSS)、(VDDA VSSA)、(VREF+ VREF-)等
晶振 IO	主晶振 IO，RTC 晶振 IO
下载 IO	用于 JTAG 下载的 IO：JTMS、JTCK、JTDI、JTDO、NJTRST
BOOT IO	BOOT0、BOOT1，用于设置系统的启动方式
复位 IO	NRST，用于外部复位
上面 5 部分 IO 组成的系统我们也叫做最小系统	
GPIO	专用器件接到专用的总线，比如 I2C，SPI，SDIO，FSMC，DCMI 这些总线的器件需要接到专用的 IO
	普通的元器件接到 GPIO，比如蜂鸣器，LED，按键等元器件用普通的 GPIO 即
	如果还有剩下的 IO，可根据项目需要引出或者不引出

要学会查看每个引脚的功能，要知道查看什么资料

3

提纲

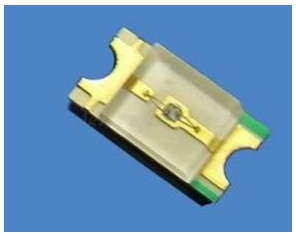
- 1. 点亮LED – 使用寄存器的绝对地址
- 2. 点亮LED –寄存器地址封装
- 3. GPIO输入编程 - 按键

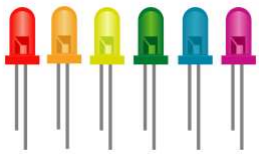
本章学习LED和按键的寄存器编程

4

LED发光二极管

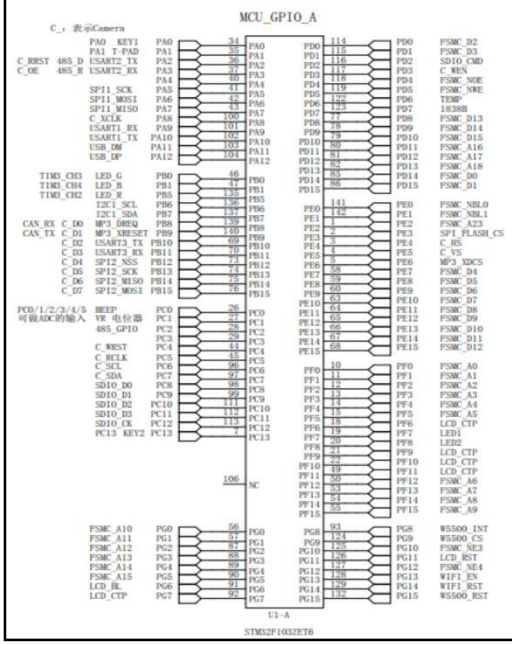
通常红色贴片**LED:**靠电流驱动，电压**1.8V~2.2V**，电流**1**到**20mA**，在**1**到**5mA**亮度有所变化，**5mA**以上亮度基本无变化。



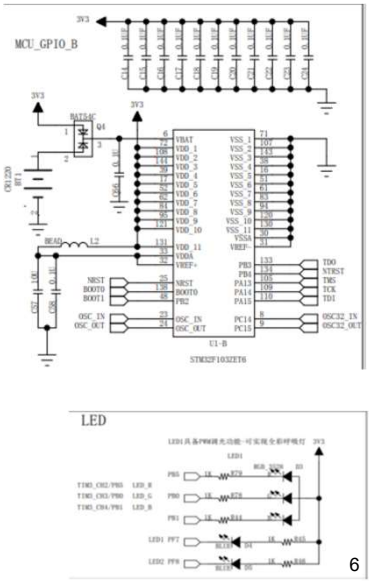


5

编程：用寄存器点亮LED

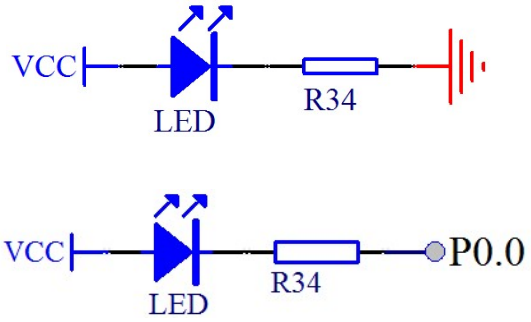


相关原理图



6

点亮一个LED发光二极管



7

LED限流电阻

- LED具有方向性（直插长脚为正，贴片彩标为负）。
- 发光二极管自身压降大概是2V， LED限流电阻承受的电压就是U=VCC-2V.
- LED要求电流范围是1~20mA的话，就可以根据欧姆定律R=U/I，把这个电阻的上限和下限值求出来。
- U=1.3V，当电流是1mA的时候，电阻值是1.3K；当电流是20mA的时候，电阻值是65欧，
- LED限流电阻的取值范围是65~1.3K欧姆。这个电阻值大小的变化，直接可以限制整条通路的电流的大小，因此这个电阻我们通常称之为“限流电阻”。
- 实际应用中以LED规格说明为准

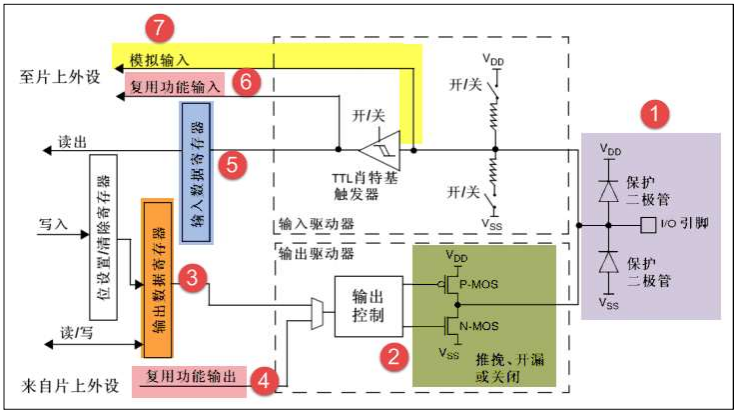
8

电平

- 单片机是一种数字集成芯片，数字电路中只有两种电平：高电平和低电平。
- 单片机输出与输入为TTL电平（晶体管一晶体管逻辑电平）或者CMOS逻辑电平。
- TTL电平信号用的最多，这是因为，数据表示通常采用二进制，高电平等价于逻辑1，低电平等价于逻辑0，这被称为TTL信号系统，这是计算机处理器控制的设备内部各部分之间道信的标准技术。
- 5V单片机采用5V TTL电平,高电平为+5V，低电平为0V。
- 3.3V单片机采用3.3V TTL电平,高电平为+3.3V，低电平为0V。
- 计算机的串口为RS-232C电平，其中高电平为-12V，低电平为+12V。因此，当计算机与单片机之间要通信时，需要加电平转换芯片，如MAX232。

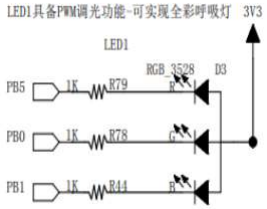
9

GPIO功能框图讲解



10

编程思路



目标：点亮PB0引脚上的LED_G

思路：

1）打开 GPIOB 端口的时钟

2）配置该IO引脚的输入输出方法为输出，（配置GPIO工作模式（CRL和CRH寄存器））

3）控制对应的数据寄存器(ODR、BRR和BSRR)让对应的IO引脚输出高低电平

11

寄存器映射

让GPIOB端口的16个引脚输出高电平，要怎么实现?

通过绝对地址访问内存单元

```
1 // GPIOB 端口全部输出 高电平
2 *(unsigned int*)(0x40010C0C) = 0xFFFF;
```

1、0X40010C0C 是GPIOB输出数据寄存器ODR的地址，如何找到?

2、(unsigned int*)的作用是什么?

2、学会使用C语言的 * 号

12

STM32寄存器映射

总线基地址（总线是什么）

- 1. 总线基地址

表格 6-5 总线基地址

总线名称	总线基地址	相对外设基地址的偏移
APB1	0x4000 0000	0x0
APB2	0x4001 0000	0x0001 0000
AHB	0x4001 8000	0x0001 8000

13

STM32寄存器映射

GPIO基地址（外设是什么）

表格 6-6 外设 GPIO 基地址

外设名称	外设基地址	相对 APB2 总线的地址偏移
GPIOA	0x4001 0800	0x0000 0800
GPIOB	0x4001 0C00	0x0000 0C00
GPIOC	0x4001 1000	0x0000 1000
GPIOD	0x4001 1400	0x0000 1400
GPIOE	0x4001 1800	0x0000 1800
GPIOF	0x4001 1C00	0x0000 1C00
GPIOG	0x4001 2000	0x0000 2000

14

STM32寄存器映射

GPIOB端口的寄存器列表

表格 6-7 GPIOB 端口的 寄存器地址列表

寄存器名称	寄存器基地址	相对 GPIOB 基址的偏移
GPIOB_CRL	0x4001 0C00	0x00
GPIOB_CRH	0x4001 0C00	0x04
GPIOB_IDR	0x4001 0C00	0x08
GPIOB_ODR	0x4001 0C00	0x0C
GPIOH_BSRR	0x4001 0C00	0x10
GPIOH_BRR	0x4001 0C00	0x14
GPIOH_LCKR	0x4001 0C00	0x18

15

GPIOx端口数据输出寄存器ODR描述

8.2.4 端口输出数据寄存器(GPIOx_ODR) (x=A..E)

地址偏移：0Ch

复位值：0x0000 0000

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) (Port output data) 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。

16

演示实验1：使用寄存器绝对地址点亮LED

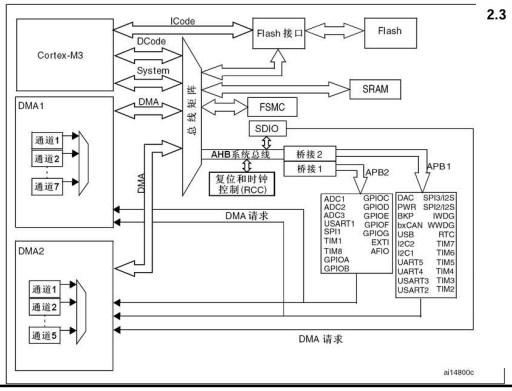
```
int main (void)
{
        // 打开 GPIOB 端口的时钟
        *( unsigned int * )0x40021018 |=  ( (1) << 3 );

        // 配置IO口为输出
        *( unsigned int * )0x40010C00 |=  ( (1) << (4*0)
);

        // 控制 ODR 寄存器
        *( unsigned int * )0x40010C0C &= ~(1<<0);

}
```

17

打开时钟的代码内涵

```
// 打开 GPIOB 端口的时钟
*( unsigned int * ) 0x40021018
|=  ( (1) << 3 );
```

图1　系统结构



6.3.7　APB2 外设时钟使能寄存器(RCC_APB2ENR)

偏移地址：0x18

复位值：0x0000 0000

访问：字，半字和字节访问

通常无访问等待周期。但在APB2总线上的外设被访问时，将插入等待状态直到AP
问结束。

注：　当外设时钟没有启用时，软件不能读出外设寄存器的数值，返回的数值始终是0x0。

31	30	29	28	27	26	25	24	23	22	21	20	19	18
保留													
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**
ADC3 EN	USART1 EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

2.3　存储器映像

请参考相应器件的数据手册中的存储器映像图。表1列出了所用STM32F10xxx中内置外设的起始地址。

表1　寄存器组起始地址

起始地址	外设	总线	寄存器映像
0x5000 0000 – 0x5003 FFFF	USB OTG 全速	AHB	参见26.14.6节
0x4002 1000 - 0x4002 13FF	复位和时钟控制(RCC)		参见6.3.11节
0x4002 0800 - 0x4002 0FFF	保留		
0x4002 0400 - 0x4002 07FF	DMA2		参见10.4.7节
0x4002 0000 - 0x4002 03FF	DMA1		参见10.4.7节
0x4001 8400 - 0x4001 7FFF	保留		
0x4001 3800 - 0x4001 3BFF	USART1		参见25.6.8节
0x4001 3400 - 0x4001 37FF	TIM8定时器		参见13.4.21节
0x4001 3000 - 0x4001 33FF	SPI1		参见23.5节
0x4001 2C00 - 0x4001 2FFF	TIM1定时器		参见13.4.21节
0x4001 2800 - 0x4001 2BFF	ADC2		参见11.12.15节
0x4001 2400 - 0x4001 27FF	ADC1		参见11.12.15节
0x4001 2000 - 0x4001 23FF	GPIO端口G	APB2	参见8.5节
0x4001 2000 - 0x4001 23FF	GPIO端口F		参见8.5节
0x4001 1800 - 0x4001 1BFF	GPIO端口E		参见8.5节
0x4001 1400 - 0x4001 17FF	GPIO端口D		参见8.5节
0x4001 1000 - 0x4001 13FF	GPIO端口C		参见8.5节
0X4001 0C00 - 0x4001 0FFF	GPIO端口B		参见8.5节
0x4001 0800 - 0x4001 0BFF	GPIO端口A		参见8.5节

18

配置IO方向的代码内涵

```
// 配置PB0为输出
*( unsigned int * )0x40010C00 |= ( (1) << (4*0) );
```

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) (Port x configuration bits) 软件通过这些位配置相应的I/O端口，请参考表17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) (Port x mode bits) 软件通过这些位配置相应的I/O端口，请参考表17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

8.2.1 端口配置低寄存器(GPIOx_CRL) (x=A..E

偏移地址：0x00

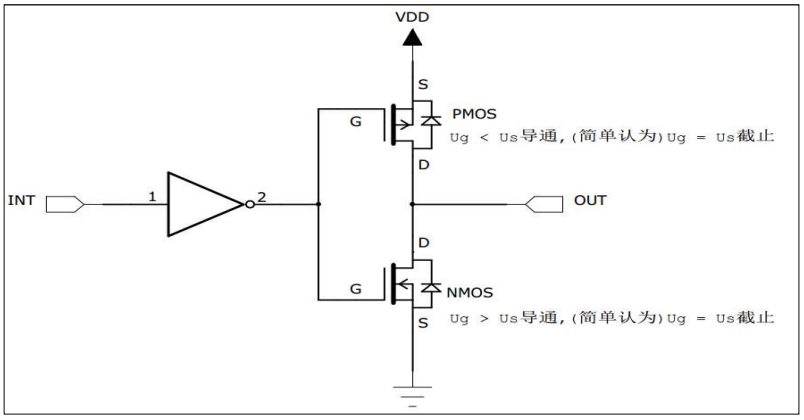
复位值：0x4444 4444

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

19

GPIO功能框图讲解

推挽输出

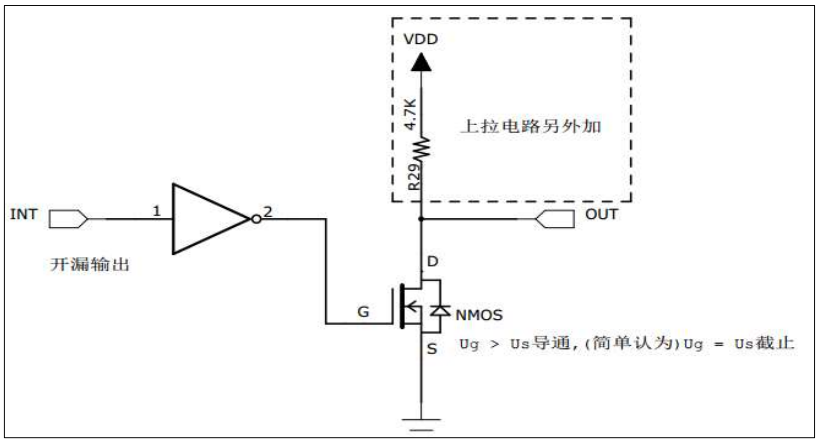


推挽模式时双MOS管以轮流方式工作，输出数据寄存器可控制I/O输出高低电平

20

GPIO功能框图讲解

开漏输出



开漏模式时，只有N-MOS管工作，输出数据寄存器可控制I/O 输出高阻态或低电平。

21

GPIO功能框图讲解

总结：什么叫推挽输出?

1. 可以输出高低电平，用于连接数字器件，高电平由VDD决定，低电平由VSS决定。
2. 推挽结构指两个三极管受两路互补的信号控制，总是在一个导通的时候另外一个截止，优点开关效率高，电流大，驱动能力强。
3. 输出高电平时，电流输出到负载，叫灌电流，可以理解成推，输出低电平时，负载电流流向芯片，叫拉电流，即挽。

22

GPIO功能框图讲解

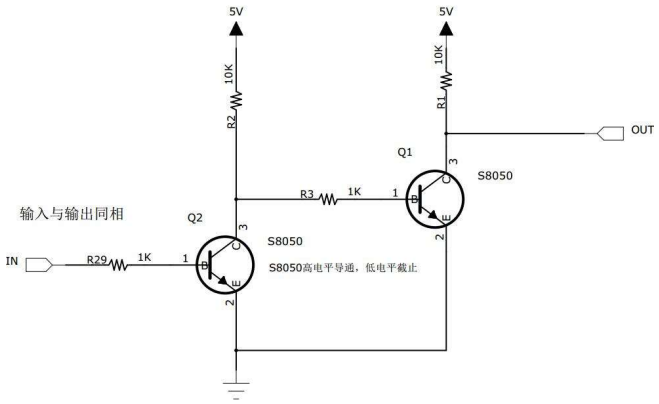
总结：什么叫开漏输出?

1. 只能输出低电平，不能输出高电平。
2. 如果要输出高电平，则需要外接上拉。
3. 开漏输出具有“线与”功能，一个为低，全部为低，多用于I2C和SMBUS总线。

23

GPIO功能框图讲解

STM32 IO 如何输出 与5V的传感器连接



24

启动文件

启动文件放在 startup/arm 这个文件夹下面，这里面启动文件有很多个，不同型号的单片机用的启动文件不一样，有关每个启动文件的详细说明见表

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 中容量， 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	

我们开发板中用的 STM32F103VET6 或者 STM32F103ZET6 的 FLASH 都是 512K，属于基本型的大容量产品，启动文件统一选择 startup_stm32f10x_hd.s。

25

提纲

- 1. 点亮LED – 使用寄存器的绝对地址
- 2. 点亮LED –寄存器地址封装
- 3. GPIO输入编程 - 按键

26

寄存器别名

通过寄存器别名方式访问内存单元

```
1 // GPIOB 端口全部输出 高电平
2 #define GPIOB_ODR     (unsigned int*)(0x40010C0C)
3 * GPIOB_ODR = 0xFF;
```

为了方便操作，我们干脆把指针操作“*”也定义到寄存器别名里面

```
1 // GPIOB 端口全部输出 高电平
2 #define GPIOB_ODR    *(unsigned int*)(0x40010C0C)
3 GPIOB_ODR = 0xFF;
```

27

STM32寄存器映射

让PB0输出低/高电平，要怎么实现?

```
#define PERIPH_BASE       ((unsigned int) 0x40000000)
#define APB2PERIPH_BASE   (PERIPH_BASE + 0x00010000)
#define GPIOB_BASE        (APB2PERIPH_BASE + 0x0C00)
#define GPIOB_ODR         *(unsigned int*)(GPIOB_BASE+0x0C)

// PB0输出输出低电平：   清0 &=~
GPIOB_ODR &= ~(1<<0);

// PB0输出输出高电平：置位    |=
GPIOB_ODR |= (1<<0);
```

28

演示实验2 寄存器别名实现点亮LED

stm32f10x.h

```
// 用来存放STM32寄存器映射的代码

// 外设  perirhral

#define  PERIPH_BASE              ((unsigned int)0x40000000)
#define  APB1PERIPH_BASE           PERIPH_BASE
#define  APB2PERIPH_BASE          (PERIPH_BASE + 0x10000)
#define  AHBPERIPH_BASE           (PERIPH_BASE + 0x20000)


#define  RCC_BASE                 (AHBPERIPH_BASE + 0x1000)
#define  GPIOB_BASE               (APB2PERIPH_BASE + 0x0C00)


#define  RCC_APB2ENR              *(unsigned int*)(RCC_BASE + 0x18)

#define  GPIOB_CRL                *(unsigned int*)(GPIOB_BASE + 0x00)
#define  GPIOB_CRH                *(unsigned int*)(GPIOB_BASE + 0x04)
#define  GPIOB_ODR                *(unsigned int*)(GPIOB_BASE + 0x0C)
```

- 1. 总线基地址

表格 6-5 总线基地址

总线名称	总线基地址	相对外设基地址的偏移
APB1	0x4000 0000	0x0
APB2	0x4001 0000	0x0001 0000
AHB	0x4001 8000	0x0001 8000

```
int main (void){

    // 打开 GPIOB 端口的时钟
    RCC_APB2ENR  |=  ( (1) << 3 );

    // 配置IO口为输出
    GPIOB_CRL &=  ~( (0x0f) << (4*0) );
    GPIOB_CRL |=  ( (1) << (4*0) );

    // 控制 ODR 寄存器
    GPIOB_ODR &= ~(1<<0);
    //GPIOB_ODR |= (1<<0);
}
```

29

提纲

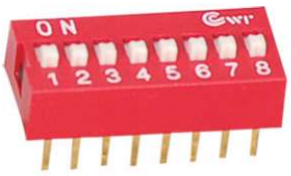
- 1. 点亮LED – 使用寄存器的绝对地址
- 2. 点亮LED –寄存器地址封装
- 3. GPIO输入编程 - 按键

30

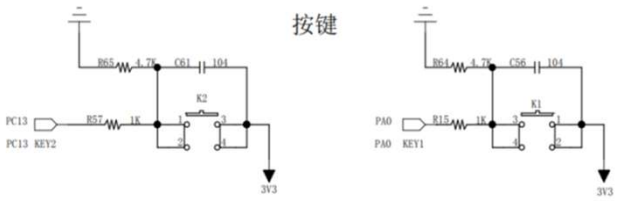
GPIO输入

- 按键

拨位开关



GPIO输入：
用来从外界器件获取输入的电信号，
也就是读取电平值是高(1)还是低(0)



没有被按下的时候，GPIO引脚的输入状态为低电平（按键所在的电路不通，引脚接地），当按键按下时，GPIO引脚的输入状态为高电平(按键所在的电路导通，引脚接到电源)。

编程目标：
按键检测,KEY1按下控制绿灯闪烁

31

GPIO-三部曲

- 第一步：使能GPIO端口时钟

- // 打开 GPIOA 端口的时钟 , RCC_APB2ENR bit 2
- RCC_APB2ENR |= ((1) << 2);

6.3.7 APB2 外设时钟使能寄存器(RCC_APB2ENR)

偏移地址：0x18
复位值：0x0000 0000
访问：字，半字和字节访问
通常无访问等待周期。但在APB2总线上的外设被访问时，将插入等待状态直到APB2的外设访问结束。

注：当外设时钟没有启用时，软件不能读出外设寄存器的数值，返回的数值始终是0x0。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
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
ADC3 EN	USART1 EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	保留	AFIO EN

32

GPIO-三部曲

- 第二步：设置IO引脚的方向和模式：

```
//KEY1 对应的引脚PA0
#define  GPIOA_BASE            (APB2PERIPH_BASE + 0x0800)
//PA对应的配置寄存器GPIOA_CRL， GPIOA_CRH
//PA0对应其中的低4位
#define  GPIOA_CRL             *(unsigned int*)(GPIOA_BASE + 0x00)


// 配置KEY1的IO口PA0为floating输入
                GPIOA_CRL &=  ~( (0x0f) << (4*0) );
        GPIOA_CRL |=  ( (0x04) << (4*0) );  //0b0100
```

8.2.1 端口配置低寄存器(GPIOx_CRL) (x=A..E)

偏移地址：0x00

复位值：0x4444 4444

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

33

端口配置寄存器

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) (Port x configuration bits) 软件通过这些位配置相应的I/O端口，请参考表17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) (Port x mode bits) 软件通过这些位配置相应的I/O端口，请参考表17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

- **上拉和下拉输入很好理解，默认的电平由上拉或者下拉决定。**
- **浮空输入的电平是不确定的，完全由外部的输入决定，一般接按键的时候用的是这个模式。**
- **模拟输入则用于ADC采集。**

34

GPIO输入模式

- 当I/O引脚被配置成GPIO输入时，这些引脚可以设置为 “上拉”、“下拉”和“浮空”3种输入模式之一。

1. 上拉
   上拉是指单片机的引脚通过一个电阻连接到电源（高电平），当外界没有信号输入到引脚时，引脚被上拉电阻固定在高电平（逻辑值1）。
2. 下拉
   下拉是指单片机的引脚通过一个电阻连接到地（低电平），当外界没有信号输入到引脚时，引脚被下拉电阻固定在低电平（逻辑值0）。
3. 浮空
   此时输入引脚不连接任何内部上拉或下拉电阻，允许引脚悬空或者连接外部信号。在这种模式下，引脚的状态完全由外部条件决定，不受到内部电路的影响。这种模式常用于需要精确读取外部信号的情况，避免内部电阻对信号的干扰

35

GPIO-三部曲

- 第三步：Input动作 (以前是Output动作)：检测按键的状态{按下、抬起}，根据状态决定响应操作
- 读PA0的电平 = 读取GPIOA的输入数据寄存器的bit0的值
- #define GPIOA_IDR *(unsigned int*)(GPIOA_BASE + 0x08)
- If(GPIOA_IDR & GPIO_Pin) != 1) {…} else{… }

8.2.3 端口输入数据寄存器(GPIOx_IDR) (x=A..E)

地址偏移：0x08

复位值：0x0000 XXXX

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
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

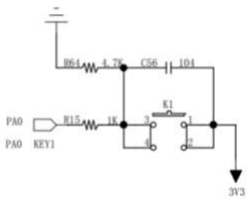
位31:16	保留，始终读为0。
位15:0	**IDRy[15:0]**：端口输入数据(y = 0…15) (Port input data) 这些位为只读并只能以字(16位)的形式读出。读出的值为对应I/O口的状态。

36

GPIO-三部曲第三步

- If(GPIOx->IDR & GPIO_Pin) != 1) …仅仅读取关心的bit，GPIO_Pin是掩码, 如：

```
#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_3                 ((uint16_t)0x0008)  /*!< Pin 3 s
#define GPIO_Pin_4                 ((uint16_t)0x0010)  /*!< Pin 4 s
#define GPIO_Pin_5                 ((uint16_t)0x0020)  /*!< Pin 5 s
```




```
//读取PA0的电平，按键
while(1){
        if( GPIOA_IDR & 0x01  ){ //读到PA0上的高电平，表明按键按下
                GPIOB_ODR &= ~(1<<0);//低电平，亮灯
        }else{  //否则按键没有按下
                GPIOB_ODR |= (1<<0);//高电平，熄灯
        }
}
```

37

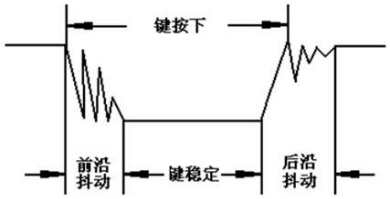
检测按键动作

- 以上的代码是检测按键状态
- 实际代码中更常见的检测按键动作(按下抬起为一个动作)：
  1. 按一下，On
  2. 再按一下，Off,
  3. 如此无限循环…

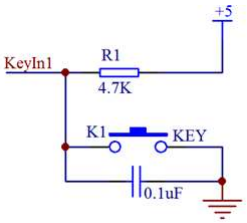
```
if( GPIOA_IDR & 0x01  ){  //按下
   while(GPIOA_IDR & 0x01 ); // 按键抬起检测
    LED_toggle(); // 自定义的翻转函数，亮变灭，灭变亮
}
```

38

按键消抖



按键抖动状态图



硬件电容消抖

39

按键消抖

- 软件消抖
- 最简单的消抖原理，就是当检测到按键状态变化后，先等待一个10ms左右的延时时间，让抖动消失后再进行一次按键状态检测，如果与刚才检测到的状态相同，就可以确认按键已经稳定的动作了。

```
if(SW1 == KEY_ON)   //判断按键被按下
     {
     DelayMS (10);     //为消抖进行延时
     if(SW1 == KEY_ON )  //经过延时后按键仍处在按下状态
       {
          while( SW1 = KEY_ON);          //等待按键松开
            。。。
       }
     }
```

40