目录

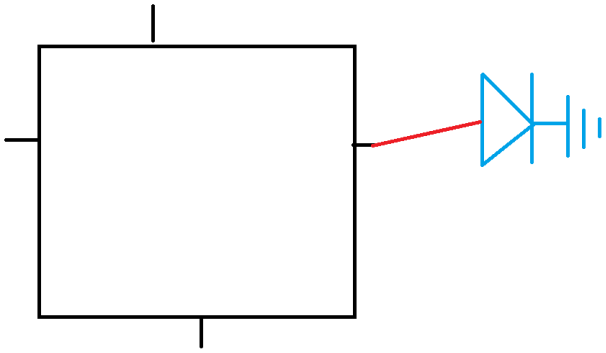
第2章 STM32-GPIO 口

2.1 GPIO 口概述

通用输入输出口 GPIO（General Purpose Input/Output）

2.1.1 GPIO 口作用

GPIO 口就是单片机与外部进行数据交流的窗口(相当于人的四肢)



2.1.2 STM32 的 GPIO 口

51 单片机：以数字进行命名，P0、P1、P2、P3，P0P1P2 都只有输入和输出功能，只有 P3 具有第二功能

STM32 单片机：以字母进行分组，M4 单片机从 GPIOA—GPIOI，但是我们当前使用的单片机 STM32F401RET6 有多少个 GPIO？50 个 GPIO 口(A/B/C：每组 16 个 + PH0 和 PH1)

在 STM32 单片机中，所有 GPIO 口都有四大模式 复用功能---复用成其他功能引脚。

1. 输入模式
   (1) 输入上拉 -----这个引脚空闲状态下为高电平
   (2) 输入下拉 ------这个引脚空闲状态为低电平
   (3) 输入浮空 -----高阻态
2. 输出模式
   (1) 推挽输出
   (2) 开漏输出
3. 复用模式(TX RX) 大部分 I/O 都有复用功能。
4. 模拟模式(数模转换器和模数转换器中)---ADC/DAC 温度传感器或者压力传感器等。

每个 GPIO 口都是多功能，使用前先选择相应的功能

所有 GPIO 除了输入输出功能外，还具有其它功能

PH0 和 PH1(接晶振)

STM32F401RET6：仿真 GPIO 50 个

GPIO 口一定是管脚

管脚不一定是 GPIO 口

分组：GPIO 分组

2.1.3 何谓寄存器

操作单片机的本质：操作寄存器

教学：

寄存器版本(必须要知道原理) ---寄存器板工程模板要知道寄存器的工作原理。

库函数版本(不用关心底层，并不需要知道原理) ---库函数工程模板，不需要考虑底层寄存器的工作方式。

寄存器每个地址代表什么含义，要查看它的数据手册和芯片手册。这样的话查找起来比较麻烦。另外的好处，当遇到问题你能很快的找到问题的原因。

int a; float b,c,d; 需要内存空间具有地址。

变量：一块内存空间，可读可写，地址随机分配

寄存器：一块内存空间，可读可写，地址固定不变，厂家出厂时已经固化

每个寄存器有什么功能都是已知的 --厂商已经设置好了。

可以找到哪个寄存器在哪个位置

每一个寄存都是有特定功能。

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

一个芯片有多少个寄存器，每个寄存器是什么功能。都是已知。---都是厂商固化好的

寄存器放在哪个地方？FLASH：掉电不丢失 ---U 盘、FLASH 这里存储的程序掉电不丢失

相当于放到了运行内存里面。

对单片机寄存器的操作不是随机，因为每个寄存器都是有特定功能的，所以你要实现某些功能时，首先找到能实现该功能寄存器，往里面写入特定值(按照参考手册)。核心根据你写入的值做出反应，从而实现你想要的功能。

实现定时功能：找到能实现定时器相关的寄存器，然后往里面写值

库函数：操作寄存器

2.1.4 STM32 的 GPIO 口特征

每一组通用 I/O 端口包括 4 个 32 位配置寄存器（GPIOx_MODER、 GPIOx_OTYPER、GPIOx_OSPEEDR 和 GPIOx_PUPDR）、 2 个 32 位数据寄存器（GPIOx_IDR 和 GPIOx_ODR）、 1 个 32 位置位/复位寄存器(GPIOx_BSRR)、 1 个 32 位锁定寄存器(GPIOx_LCKR) 和 2 个 32 位复用功能选择寄存器（GPIOx_AFRH 和 GPIOx_AFRL）能让它复用成什么功能。

控制每组 GPIO 的寄存器有 10 个

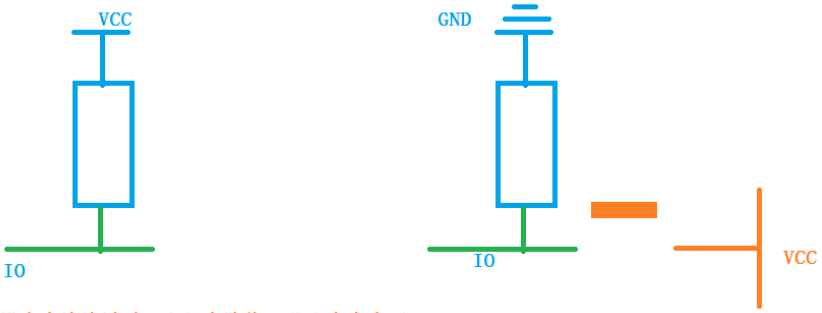
使用前要先配置好 GPIO 口的功能

受控 I/O 多达 16 个

- 输出状态：推挽或开漏 + 上拉/下拉 输出 I/O 高低电平---控制 MOS 管
- 从输出数据寄存器 (GPIOx_ODR) 或片上外设（复用功能输出）输出数据(输出数据的来源有两个)
- 可为每个 I/O 选择不同的速度(根据连接电路确定)
- 输入状态：浮空(无上下拉)、上拉/下拉、模拟 ----接入传感器等检测设备等
- 将数据输入到输入数据寄存器 (GPIOx_IDR) 或外设（复用功能输入）(输入数据可以有两个地方可以去)
- 置位和复位寄存器 (GPIOx_BSRR)，对 GPIOx_ODR 具有按位写权限
- 锁定机制 (GPIOx_LCKR)，可冻结 I/O 配置
- 模拟功能(使用 ADC 或者 DAC)
- 复用功能输入/输出选择寄存器（一个 I/O 最多可具有 16 个复用功能）选择余量很大
- 快速翻转，每次翻转最快只需要两个时钟周期
- 引脚复用非常灵活，允许将 I/O 引脚用作 普通输入输出或多种外设功能中的一种 –看模式怎么设置

2.1.5 STM32 的 GPIO 功能

- 输入浮空
- 输入上拉
- 输入下拉 （芯片内部）



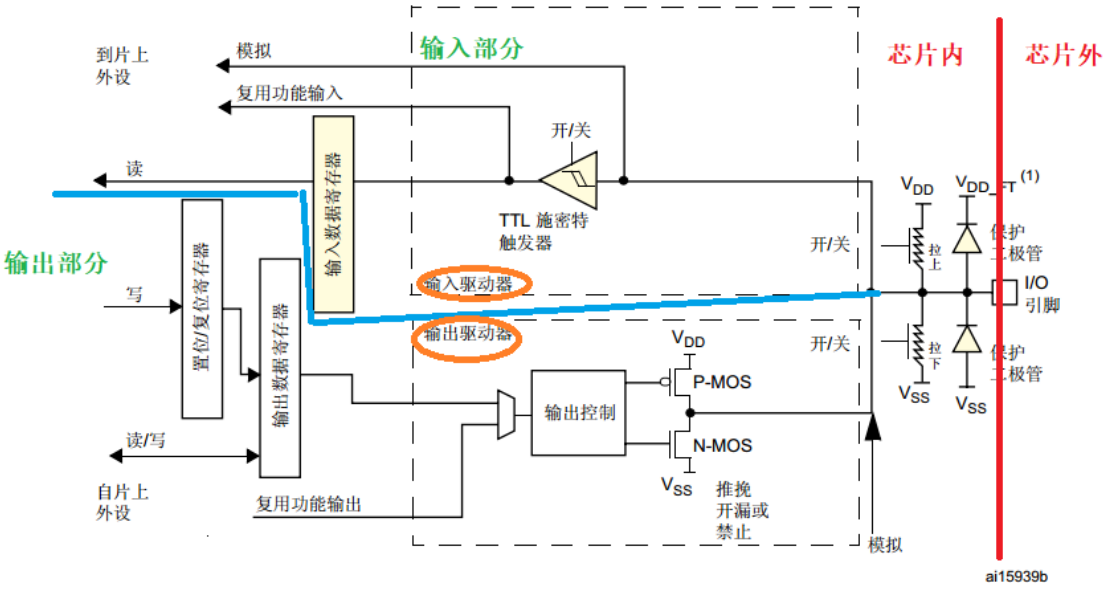
上拉：当IO没有电流流过时，上拉电路将IO稳定在高电平
下拉：当IO没有电流流过时，下拉电路将IO稳定在低电平

上下拉用来确定 GPIO 电平的状态

- 模拟功能
- 普通功能的开漏输出
- 普通功能的推挽输出
- 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

- 输入浮空
- 输入上拉
- 输入下拉
- 模拟功能
- 具有上拉或下拉功能的开漏输出
- 具有上拉或下拉功能的推挽输出
- 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

2.2 STM32 的 GPIO 口框架（重点）



2.2.1 普通输出功能

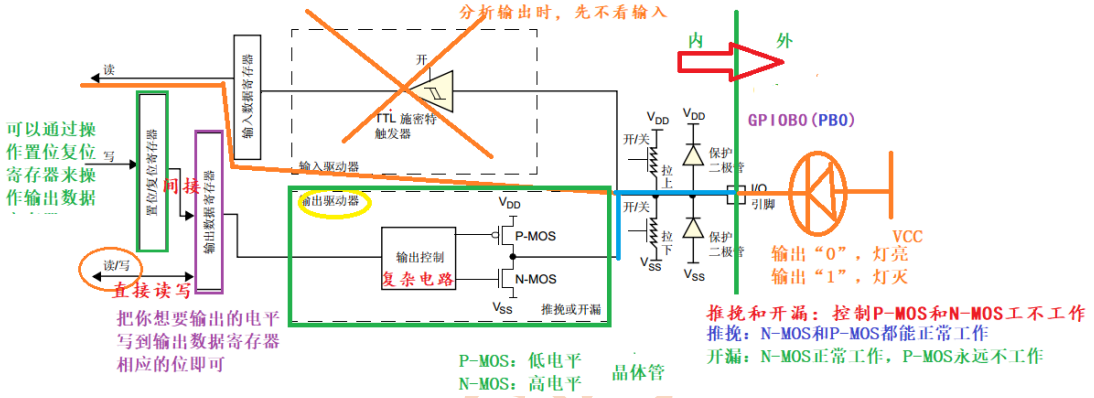
对 I/O 端口进行编程作为输出时：

- 输出缓冲器被打开：

— 开漏模式：输出寄存器中的“0”可激活 N-MOS，而输出寄存器中的“1”会使端口保持高组态 (Hi-Z)（P-MOS 始终不激活）。

— 推挽模式：输出寄存器中的“0”可激活 N-MOS，而输出寄存器中的“1”可激活 P-MOS。

- 施密特触发器输入被打开
- 根据 GPIOx_PUPDR 寄存器中的值决定是否打开弱上拉电阻和下拉电阻
- 输入数据寄存器每隔 1 个 AHB1 时钟周期对 I/O 引脚上的数据进行一次采样，频率越高次数越多。
- 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



推挽输出：能输出高电平也能输出电平　　驱动：LED、蜂鸣器....

开漏输出：只能输出低电平　　通信协议：IIC

配置为输出功能时，输入功能并没关闭

2.2.2 普通输入功能

对 I/O 端口进行编程作为输入时：

- 输出缓冲器被关闭(配置为输入功能，输出功能关闭)
- 施密特触发器输入被打开
- 根据 GPIOx_PUPDR 寄存器中的值决定是否打开上拉和下拉电阻
- 输入数据寄存器每隔 1 个 AHB1 时钟周期对 I/O 引脚上的数据进行一次采样
- 对输入数据寄存器的读访问可获取 I/O 状态，读输入数据寄存器可以判断 I/O 状态。

图 20. 输入浮空/上拉/下拉配置

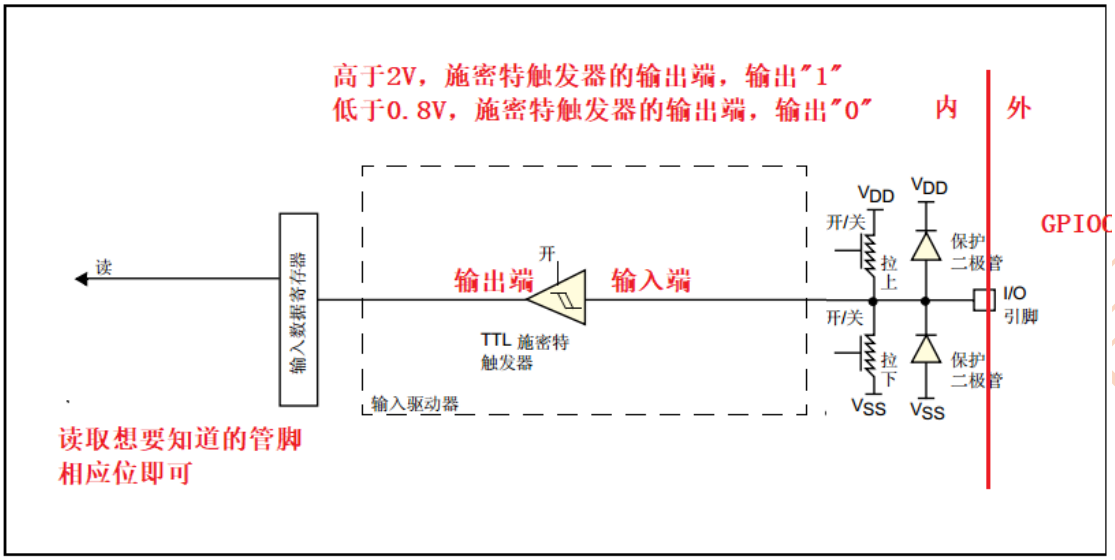
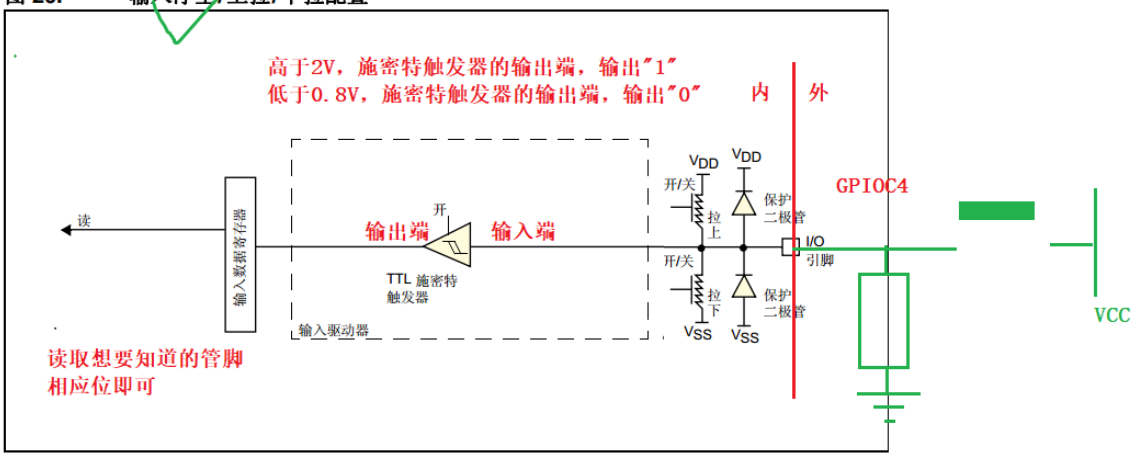


图 20. 输入浮空/上拉/下拉配置



按键识别的前提：明确按键按下前的电平状态和按键按下后的电平状态

按下前：低

按下后：高

图 20. 输入浮空/上拉/下拉配置

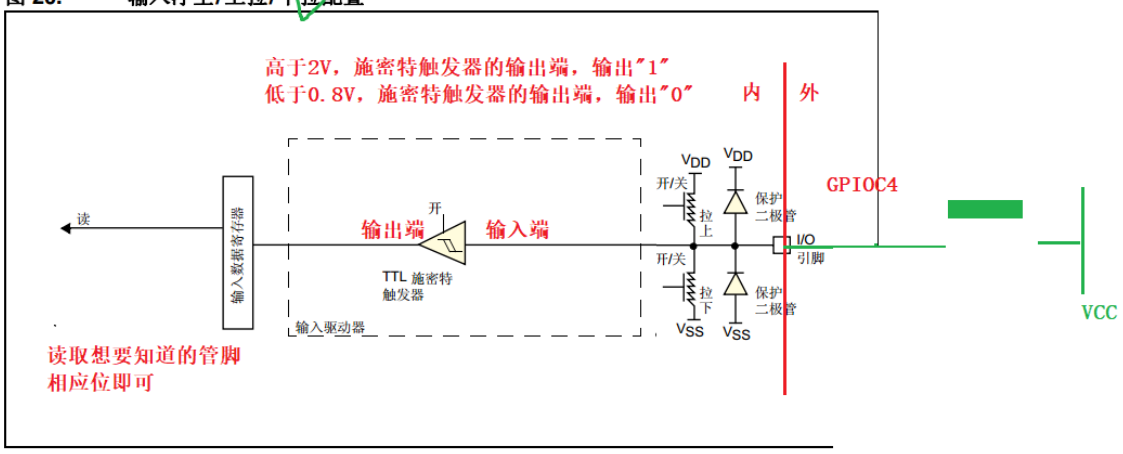
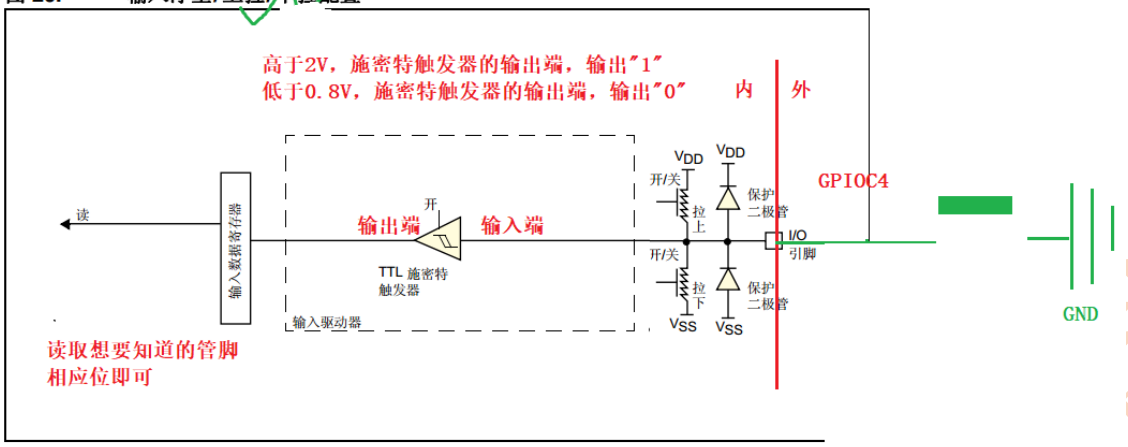


图 20. 输入浮空/上拉/下拉配置

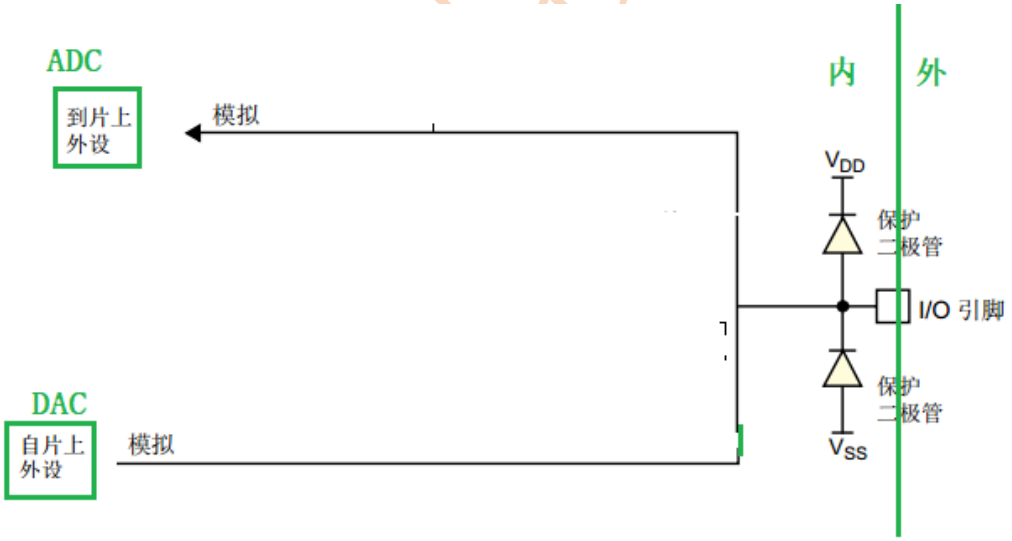


输入配置要根据管脚外接电路来决定。

2.2.3 模拟功能

对 I/O 端口进行编程作为模拟配置时：

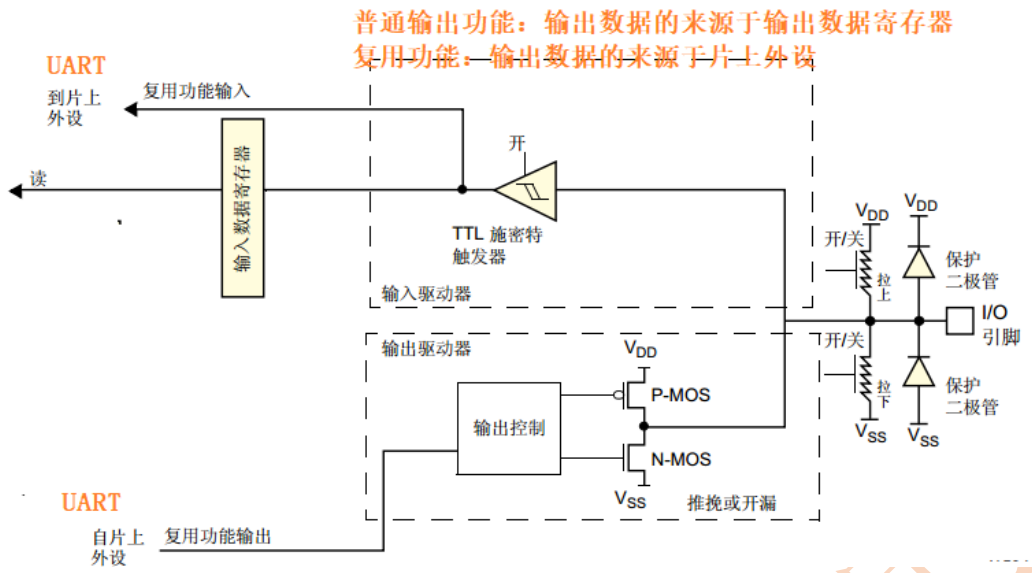
- 输出缓冲器被禁止。
- 施密特触发器输入停用， I/O 引脚的每个模拟输入的功耗变为零。施密特触发器的输出被强制处理为恒定值 (0)。
- 弱上拉和下拉电阻被关闭。
- 对输入数据寄存器的读访问值为“0”。



2.2.4 复用功能

对 I/O 端口进行编程作为复用功能时：

- 可将输出缓冲器配置为开漏或推挽
- 输出缓冲器由来自外设的信号驱动（发送器使能和数据）
- 施密特触发器输入被打开
- 根据 GPIOx_PUPDR 寄存器中的值决定是否打开弱上拉电阻和下拉电阻
- 输入数据寄存器每隔 1 个 AHB1 时钟周期对 I/O 引脚上的数据进行一次采样
- 对输入数据寄存器的读访问可获取 I/O 状态



2.3 STM32 的 GPIO 口相关寄存器

2.3.1 GPIO 端口模式寄存器 (GPIOx_MODER) (x = A..I)

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

位 2y:2y+1 MODERy[1:0]： 端口 x 配置位 (Port x configuration bits) (y = 0..15)
这些位通过软件写入，用于配置 I/O 方向模式。
00：输入（复位状态）
01：通用输出模式
10：复用功能模式
11：模拟模式

2.3.2 GPIO 端口输出类型寄存器 (GPIOx_OTYPER) (x = A..I)

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

位 31:16 保留，必须保持复位值。
位 15:0 OTy[1:0]： 端口 x 配置位 (Port x configuration bits) (y = 0..15)
这些位通过软件写入，用于配置 I/O 端口的输出类型。
0：输出推挽（复位状态）
1：输出开漏

2.3.3 GPIO 端口输出速度寄存器 (GPIOx_OSPEEDR) (x = A..I/)

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

位 2y:2y+1 OSPEEDRy[1:0]： 端口 x 配置位 (Port x configuration bits) (y = 0..15)
这些位通过软件写入，用于配置 I/O 输出速度。
00： 2 MHz（低速）
01： 25 MHz（中速）
10： 50 MHz（快速）
11： 30 pF 时为 100 MHz（高速）（15 pF 时为 80 MHz 输出（最大速度））

2.3.4 GPIO 端口上拉/下拉寄存器 (GPIOx_PUPDR) (x = A..I/)

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

位 2y:2y+1 PUPDRy[1:0]： 端口 x 配置位 (Port x configuration bits) (y = 0..15)
这些位通过软件写入，用于配置 I/O 上拉或下拉。
00：无上拉或下拉
01：上拉
10：下拉
11：保留

2.3.5 GPIO 端口输入数据寄存器 (GPIOx_IDR) (x = A..I)

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

位 31:16 保留，必须保持复位值。
位 15:0 IDRy[15:0]： 端口输入数据 (Port input data) (y = 0..15)
这些位为只读形式，只能在字模式下访问。它们包含相应 I/O 端口的输入值。

2.3.6 GPIO 端口输出数据寄存器 (GPIOx_ODR) (x = A..I)

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

位 31:16 保留，必须保持复位值。

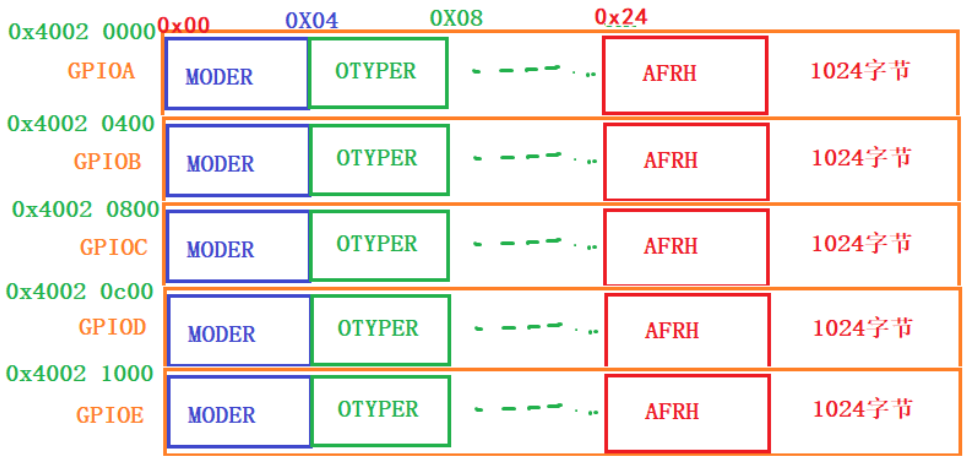
位 15:0 ODRy[15:0]： 端口输出数据 (Port output data) (y = 0..15)

这些位可通过软件读取和写入。

注意： 对于原子置位/复位，通过写入 GPIOx_BSRR 寄存器，可分别对 ODR 位进行置位和复位 (x = A..I/)。

2.4 寄存器地址偏移问题

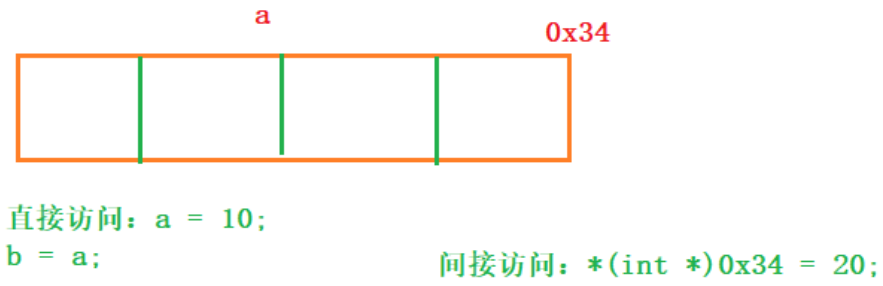
0x4002 2000 - 0x4002 23FF	GPIOI
0x4002 1C00 - 0x4002 1FFF	GPIOH
0x4002 1800 - 0x4002 1BFF	GPIOG
0x4002 1400 - 0x4002 17FF	GPIOF
0x4002 1000 - 0x4002 13FF	GPIOE
0x4002 0C00 - 0x4002 0FFF	GPIOD
0x4002 0800 - 0x4002 0BFF	GPIOC
0x4002 0400 - 0x4002 07FF	GPIOB
0x4002 0000 - 0x4002 03FF	GPIOA



每个组之间相差 1024 字节

每个寄存器之间相差 4 字节

2.5 寄存器访问方法



知道空间名：操作这个空间的方法就有两种

直接访问

间接访问

知道空间首地址和空间类型，可不可以访问该空间？可以 间接访问

方法一：

举例：把 10 写入 A 组 MODER 寄存器中 (4 字节)

A 组 MODER：*(unsigned int *)0x40020000 = 10;(指针类型：数据类型 *) 把 10 写入 GPIOA_MODER

不方便阅读代码

强转时：如果是地址→数据类型 *

如果是数据→数据类型

举例：把 10 写入 B 组 OTYPER 寄存器中 (4 字节)

*(unsigned int *)0x4002 0404 = 10;

方法二：宏定义：增加了代码的阅读性

#define GPIOA_MODER *(unsigned int *)0x40020000

GPIOA_MODER = 10;

因为芯片上的寄存器非常多

方法三：ST 公司弄好了

用到寄存器的文件都需要把头文件包含过来

芯片厂家已经做好了

有用到寄存器的地方就需要把单片机头文件包含过来(寄存器定义都在单片机头文件中)

```
GPIOA->MODER = 10;
#define   GPIOA                ((GPIO_TypeDef *) GPIOA_BASE)//0x4002 0000
#define GPIOA_BASE             (AHB1PERIPH_BASE + 0x0000)
#define AHB1PERIPH_BASE        (PERIPH_BASE + 0x00020000)
#define PERIPH_BASE            ((uint32_t)0x40000000)
                               0x4000 0000 + 0x0002 0000
                               0x4000 0000 + 0x0002 0000 + 0x0000
                               0x4002 0000(GPIOA 组寄存器的起始地址)
```

2.6 STM32 的 GPIO 口相关实验

补充：

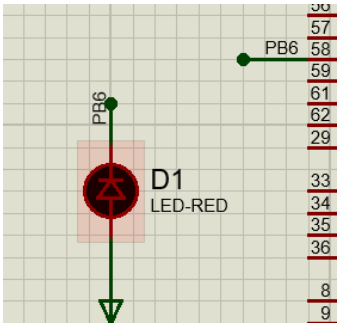
单片机是一种时序电路，要提供连续的脉冲信号才能正常工作

STM32 单片机默认各个外设时钟都是关闭：减少功耗

所以使用某个外设前，先要手动开启该外设的时钟

2.6.1 输出功能实验

2.6.1.1 硬件分析



发光二极管：PN 导通：灯亮 PN 不导通：灯不亮

因为发光二极管的阴极接了 PB6，阳极接了 VCC，所以要想灯亮，发光二极管的阴极为低电平

PB6 输出高电平，灯灭

PB6 输出低电平，灯亮

PB6 配置为推挽输出

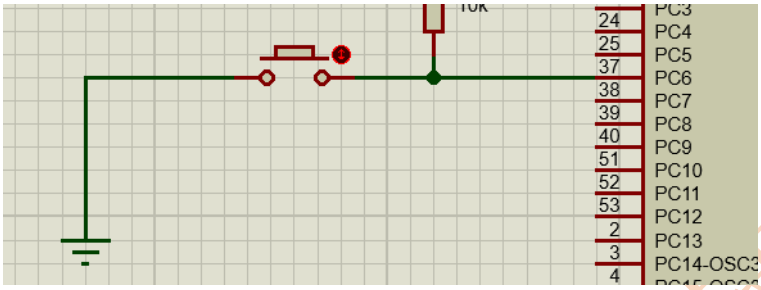
2.6.1.2 软件设计

配置流程 PB6

1. 打开 GPIOB 时钟(AHB1 上)
2. 将 PB6 配置为推挽输出(MODER(功能)、OTYPER(类型)、OSPEEDR(速度)、PUPDR(上下拉))
3. 操作输出数据寄存器(ODR) (写 1 高电平 写 0 低电平)

2.6.2 输入功能实验

2.6.2.1 硬件分析



有图可知

因为按键按下后是低电平，所以必须保证按键按下前是高电平

因为按键外部接上拉电路，所以我们不需要将内部的上拉打开

PC6 配置为浮空输入

按键按下前是高

按键按下后是低

2.6.2.2 软件设计

配置流程 PC6

初始化函数(将 PC6 管脚配置浮空输入)

1. 打开 GPIOC 时钟
2. 将 PC6 配置为浮空输入

尝试：多文件操作 key.c key.h

按键扫描函数(查看 PC6 的电平状态)