第5章 GPIO结构及应用

5.1 GPIO 结构原理

5.2 GPIO相关寄存器

5.3 GPIO典型应用步骤及常用库函数

5.4 GPIO应用实例

电子与通信工程系

华东理工大学信息学院

5.1 GPIO 结构原理

5.1 GPIO 结构原理

GPIO（General-Purpose Input/Output，通用输入/输出）是微控制器和外部进行通信的最基本的通道，几乎所有微控制器上都有GPIO。

GPIO的每个引脚都能够被独立配合，在微控制器片内外设功能较多而外部引脚数量有限的情况下，GPIO引脚都具备复用功能。

STM32F系列微控制器中有多个GPIO端口，以英文字母进行编号，每个端口有16个引脚，根据芯片型号不同端口数量不同。

5.1 GPIO 结构原理

GPIO引脚的内部构造图如图5-1所示，每个GPIO相互独立，包括输入驱动器、输出驱动器、上拉/下拉控制电路和5V耐压保护电路

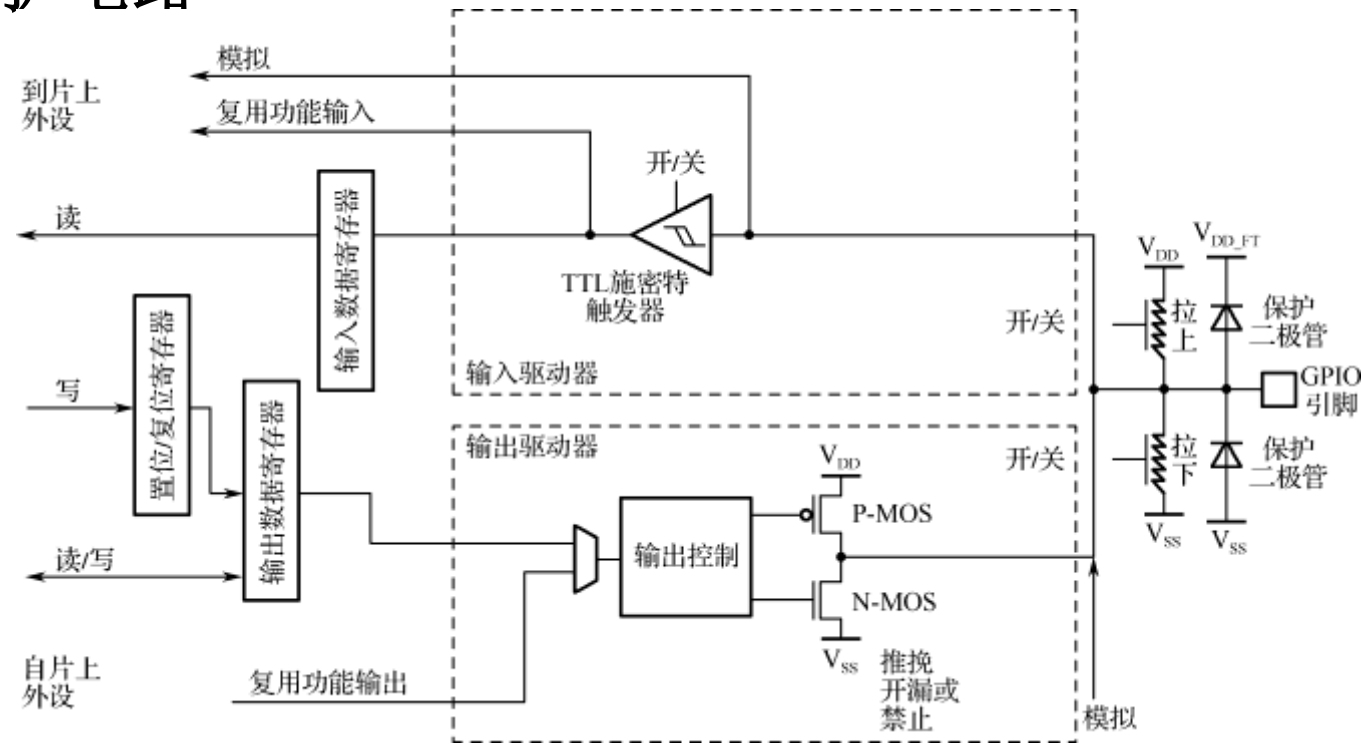


图5-1　GPIO引脚的内部构造图

5.1 GPIO 结构原理

5.1.1 GPIO 功能描述

GPIOA~GPIOK

每个通用 I/O 端口包括以下寄存器：

- 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）。

每个GPIO有16个引脚，每个引脚都可以单独配置。

5.1 GPIO 结构原理

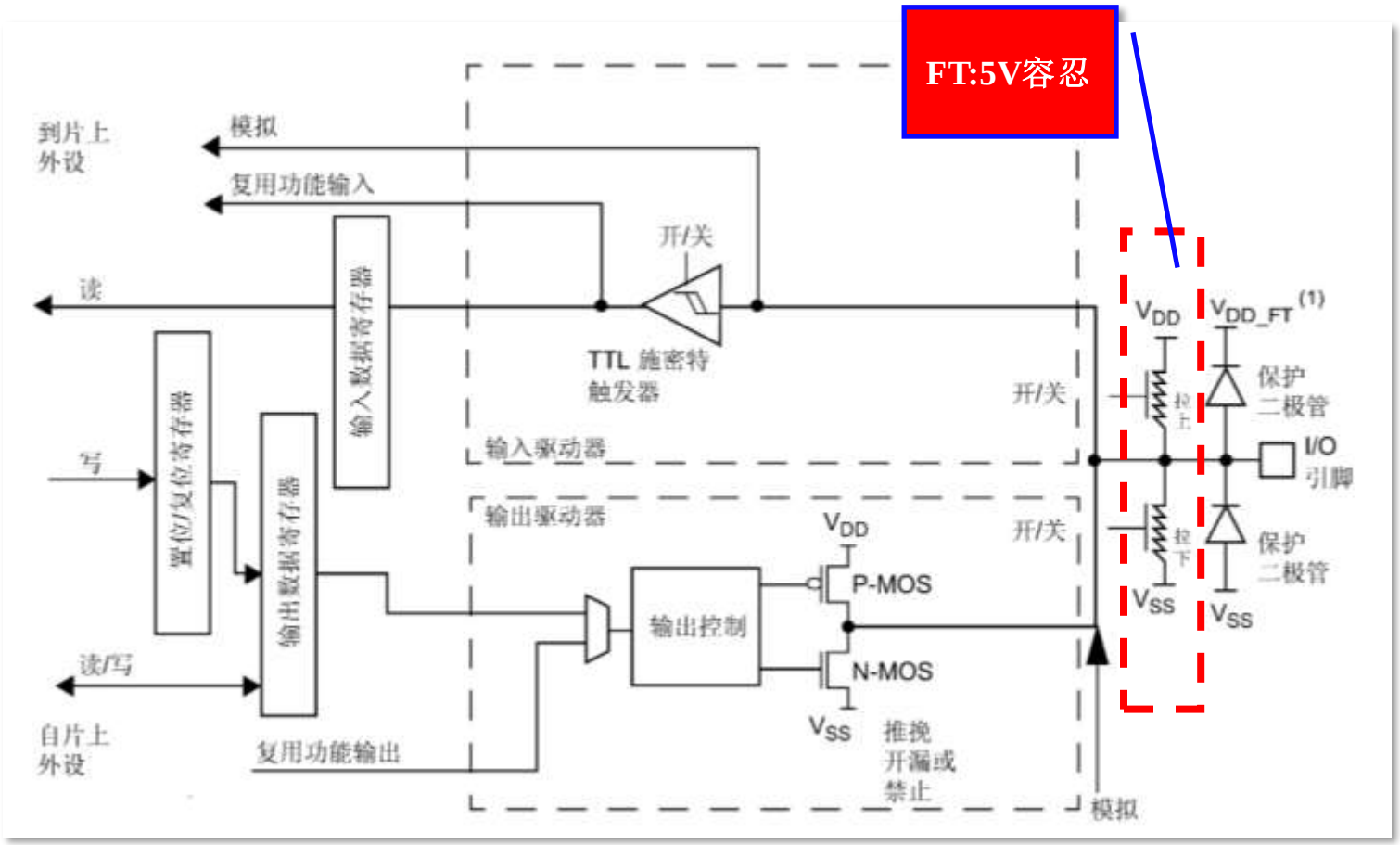
5.1.1 GPIO 功能描述

根据应用需求，可通过软件将通用 I/O (GPIO) 端口对应的各个引脚位分别配置为多种模式：

- 输入浮空-上电默认模式
- 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

7.1 GPIO 结构原理

7.1.1 GPIO 功能描述



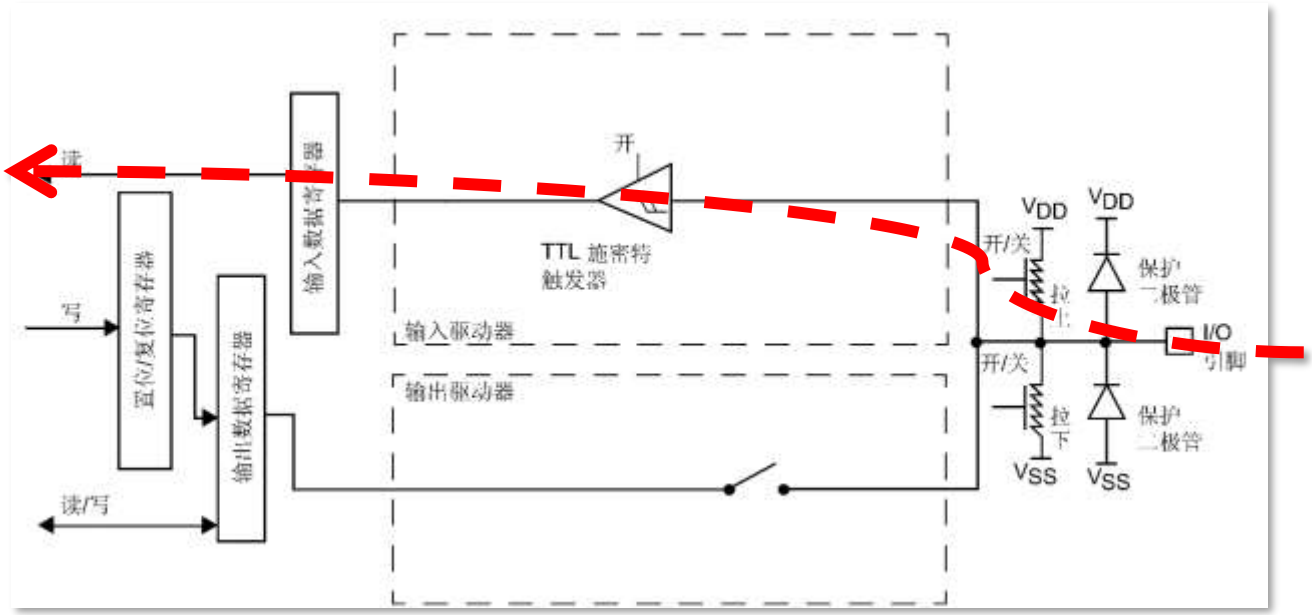
GPIO 结构图

5.1 GPIO 结构原理

5.1.2 GPIO 输入配置

对 I/O 端口进行编程作为输入时：

- 输出缓冲器被关闭
- 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

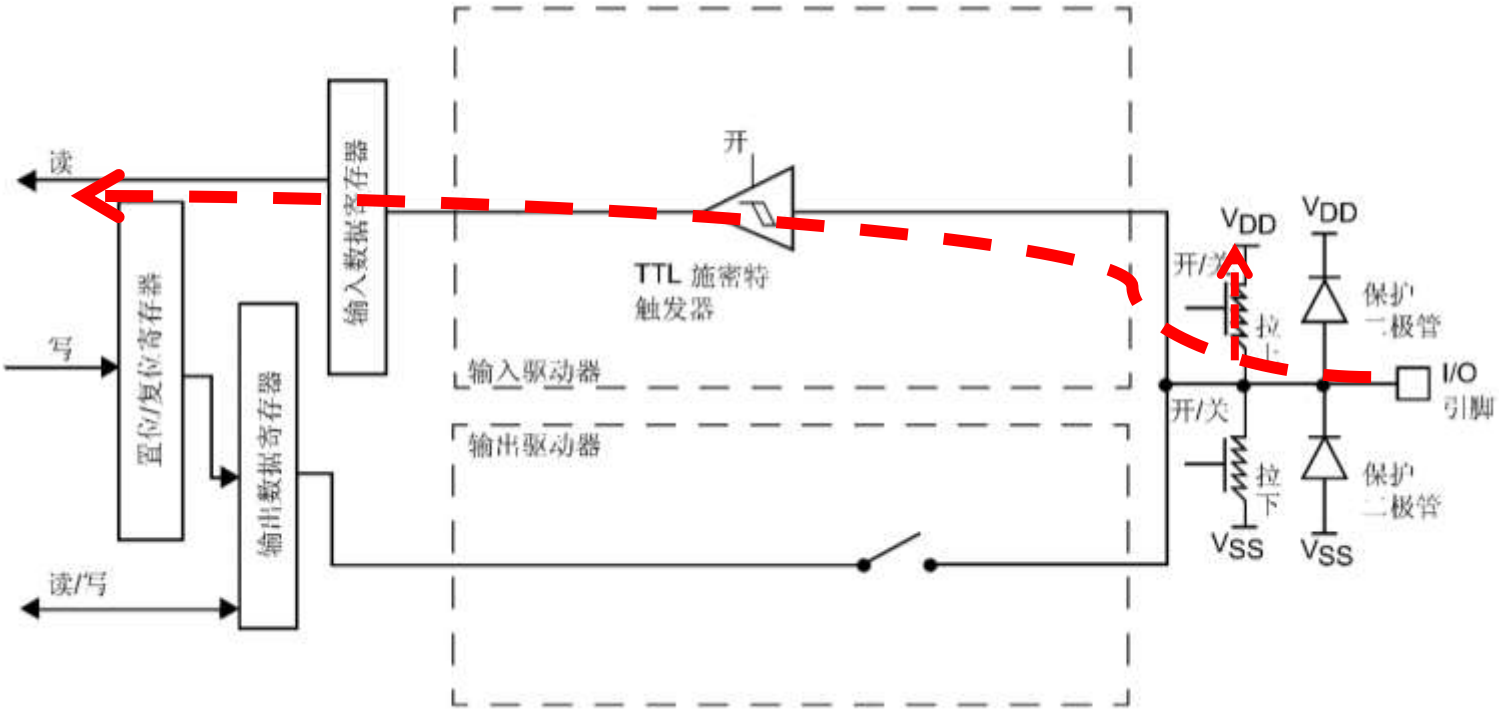


1、输入浮空模式

5.1 GPIO 结构原理

5.1.2 GPIO 输入配置

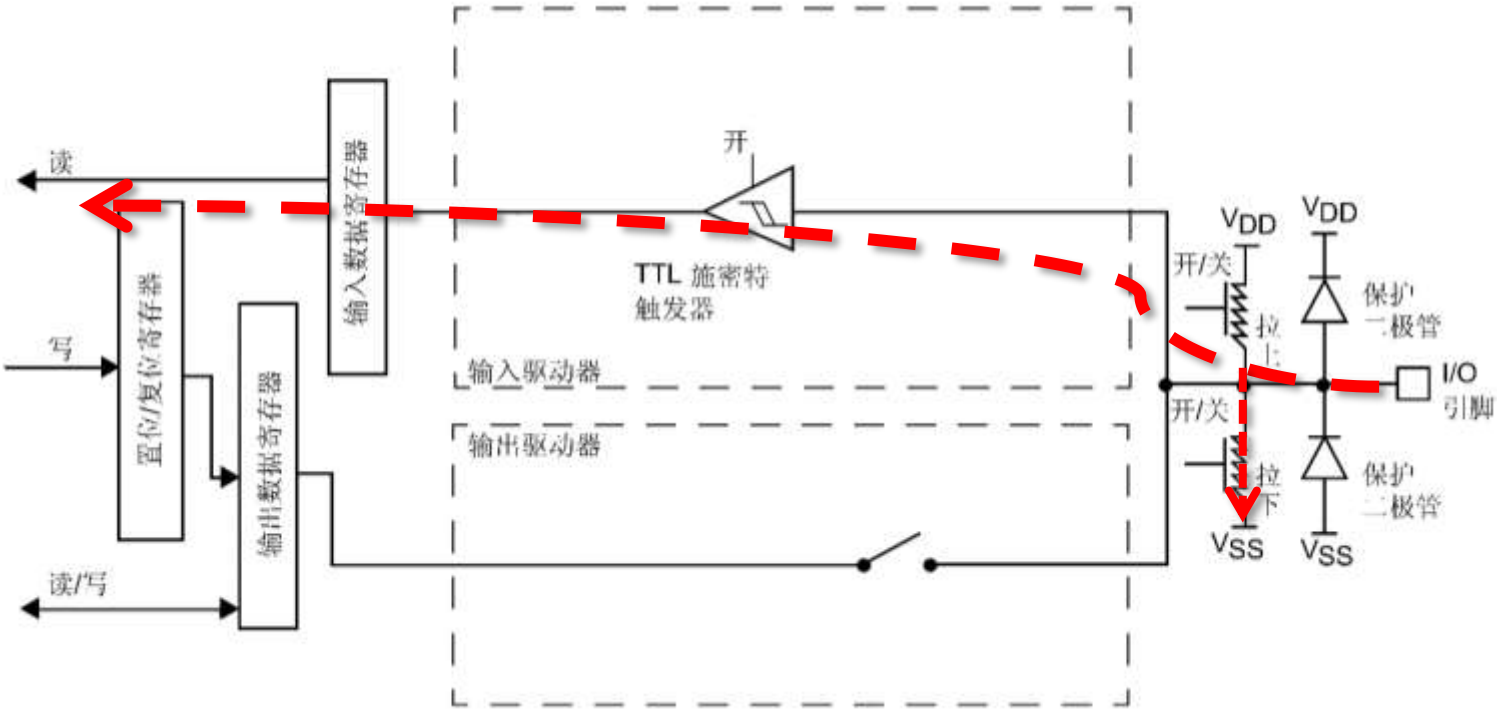
2、输入上拉模式



5.1 GPIO 结构原理

5.1.2 GPIO 输入配置

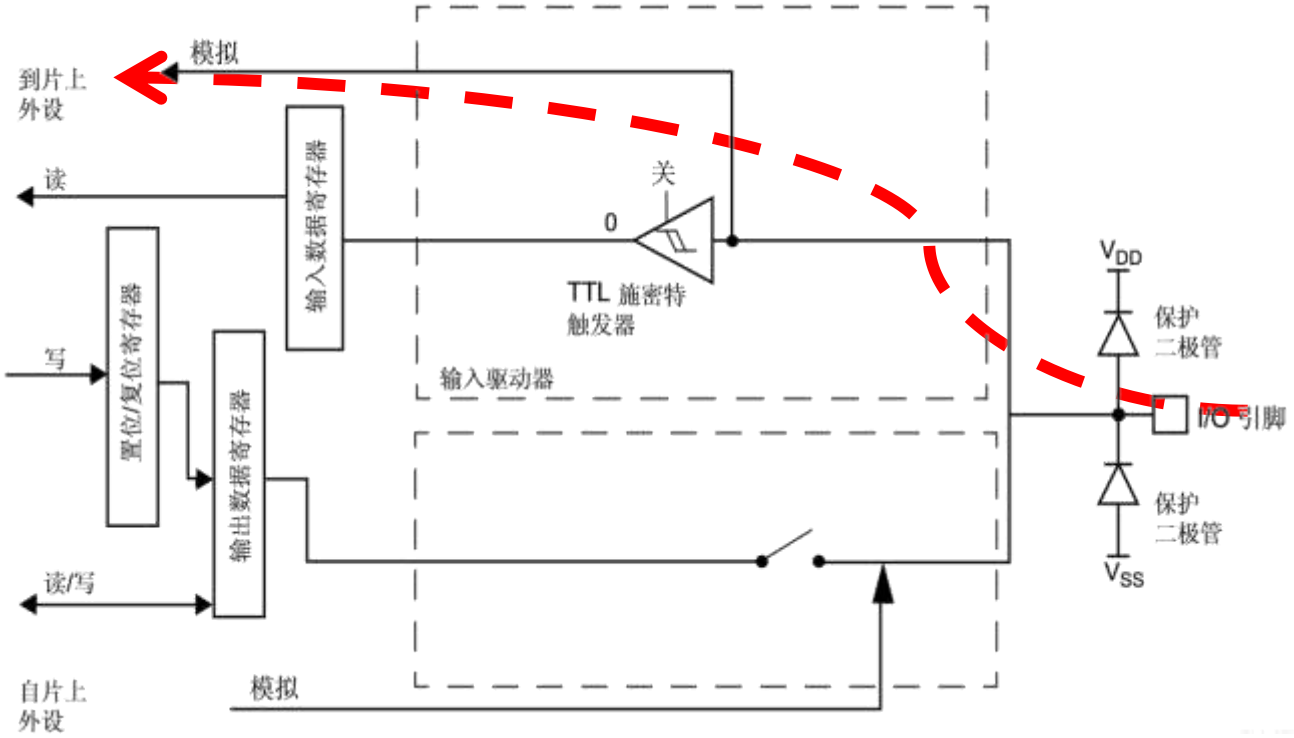
3、输入下拉模式



5.1 GPIO 结构原理

5.1.2 GPIO 输入配置

4、模拟模式



5.1 GPIO 结构原理

5.1.3 GPIO 输出配置

对 I/O 端口进行编程作为输出时：

- 输出缓冲器被打开： 开漏模式和推挽模式
- 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
- 对输出数据寄存器的读访问可获取最后的写入值

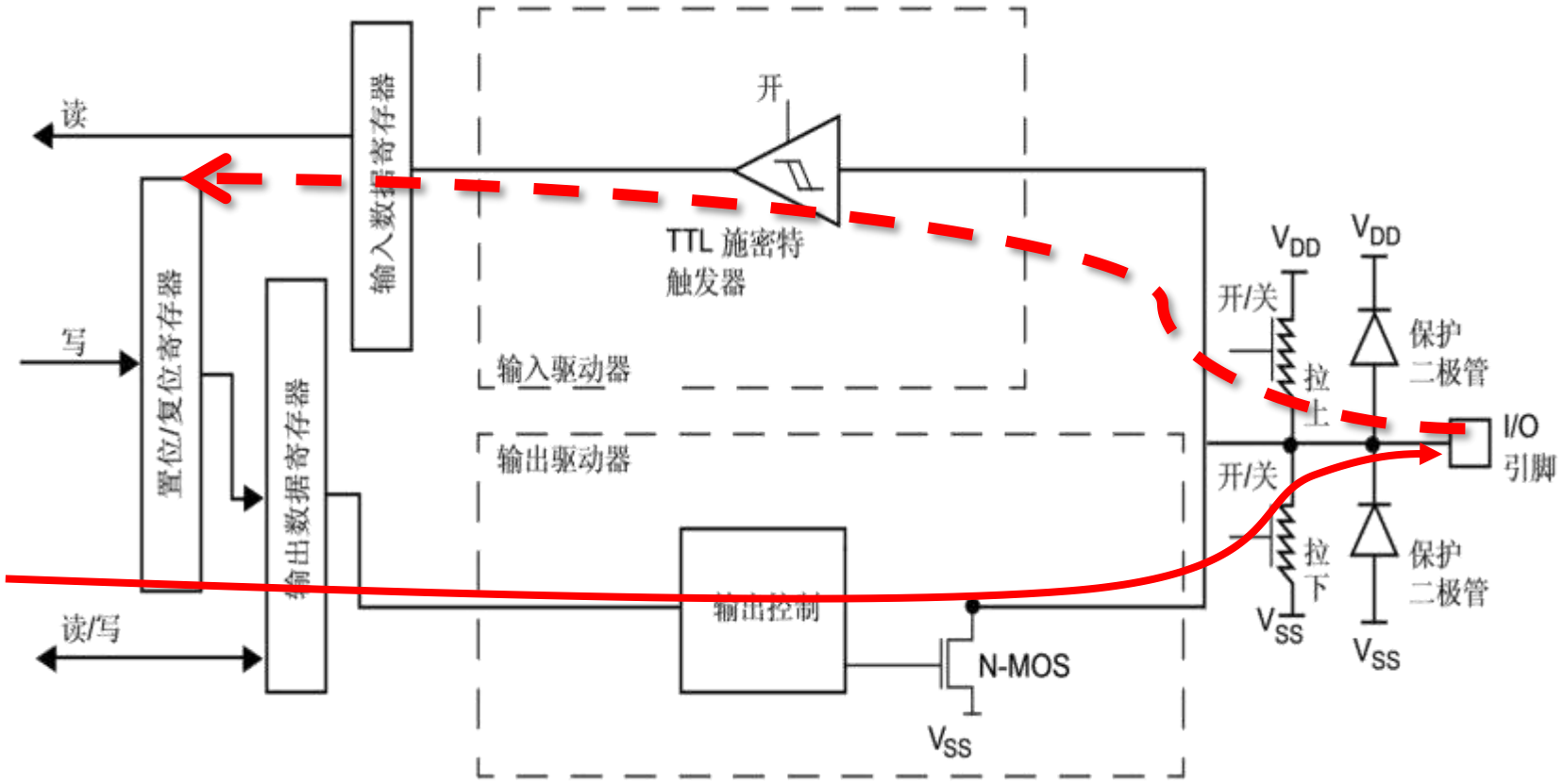
推挽输出：
可以直接输出高电平和低电平。

开漏输出：
默认只可以输出低电平
输出高电平需要加上拉电阻拉高。

5.1 GPIO 结构原理

5.1.3 GPIO 输出配置

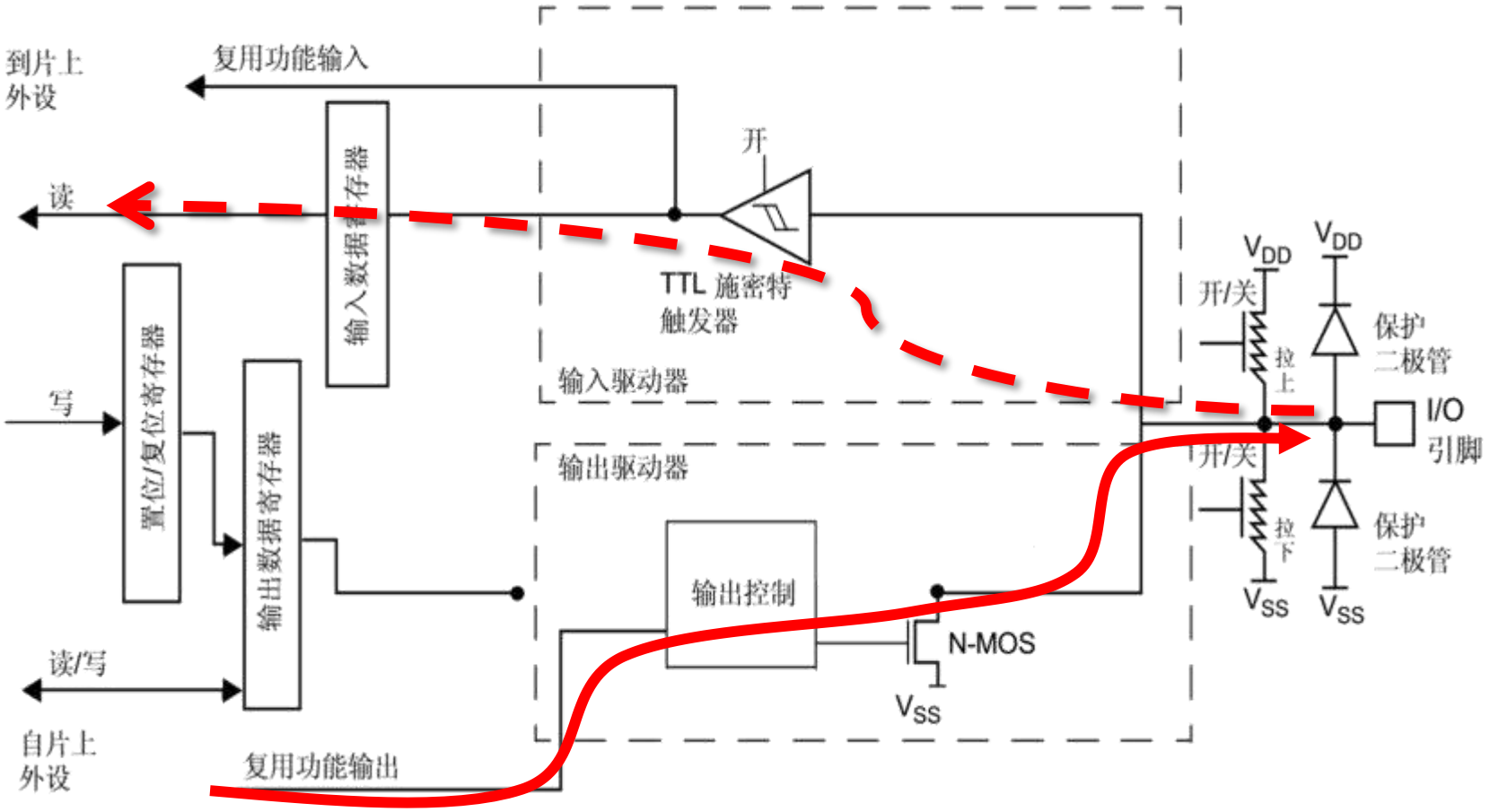
1、开漏输出模式



5.1 GPIO 结构原理

5.1.3 GPIO 输出配置

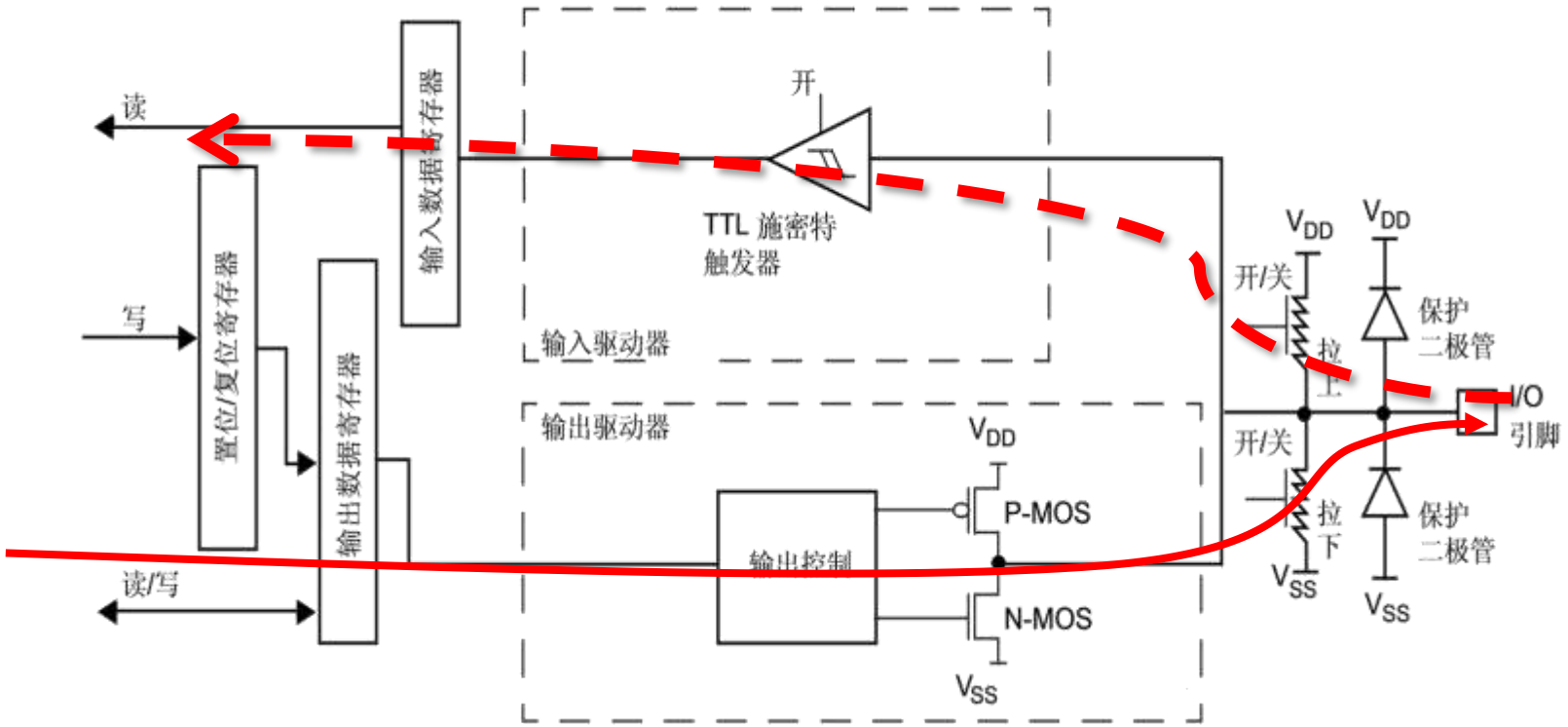
2、开漏复用输出模式



5.1 GPIO 结构原理

5.1.3 GPIO 输出配置

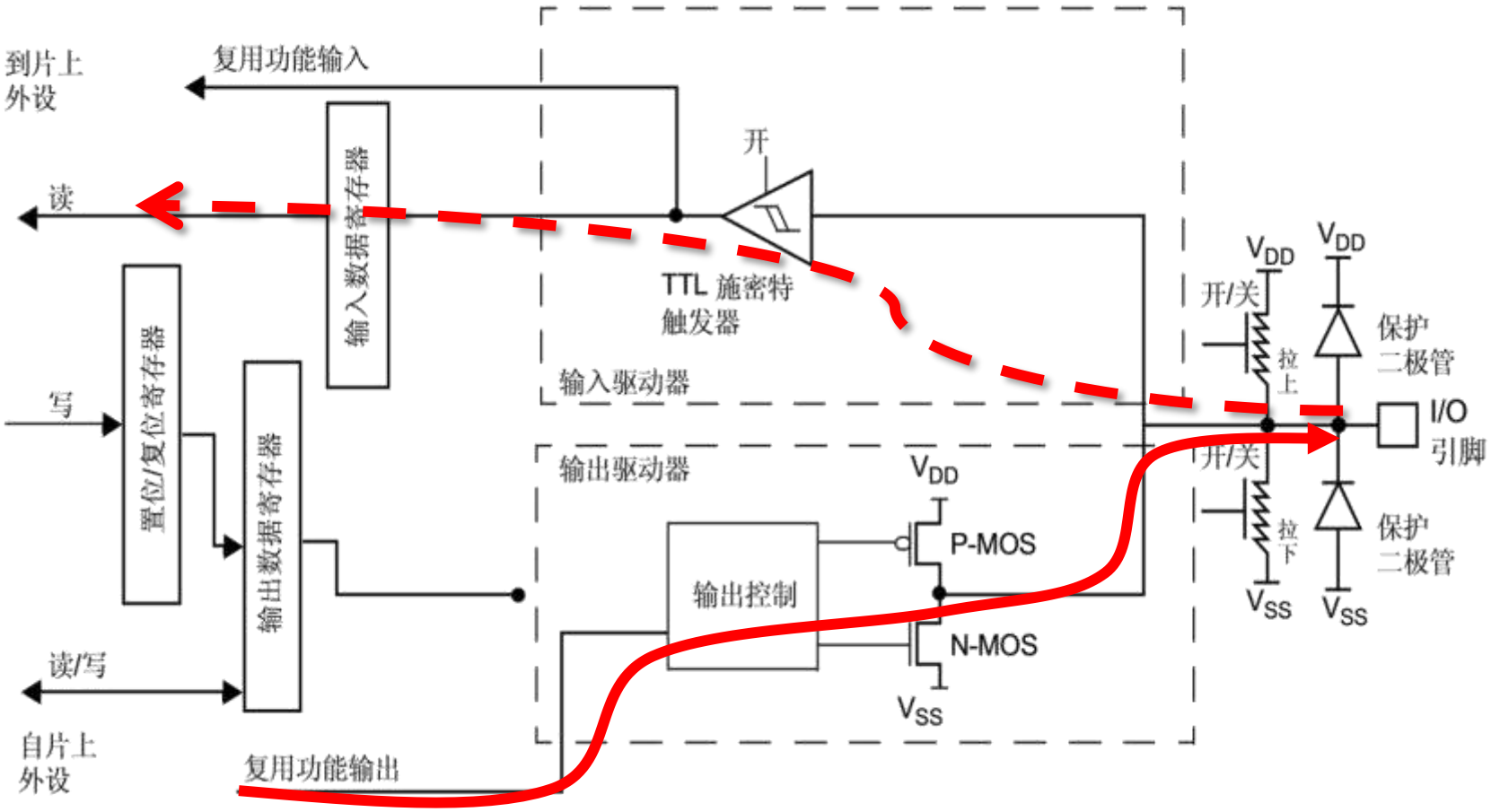
3、推挽输出模式



5.1 GPIO 结构原理

5.1.3 GPIO 输出配置

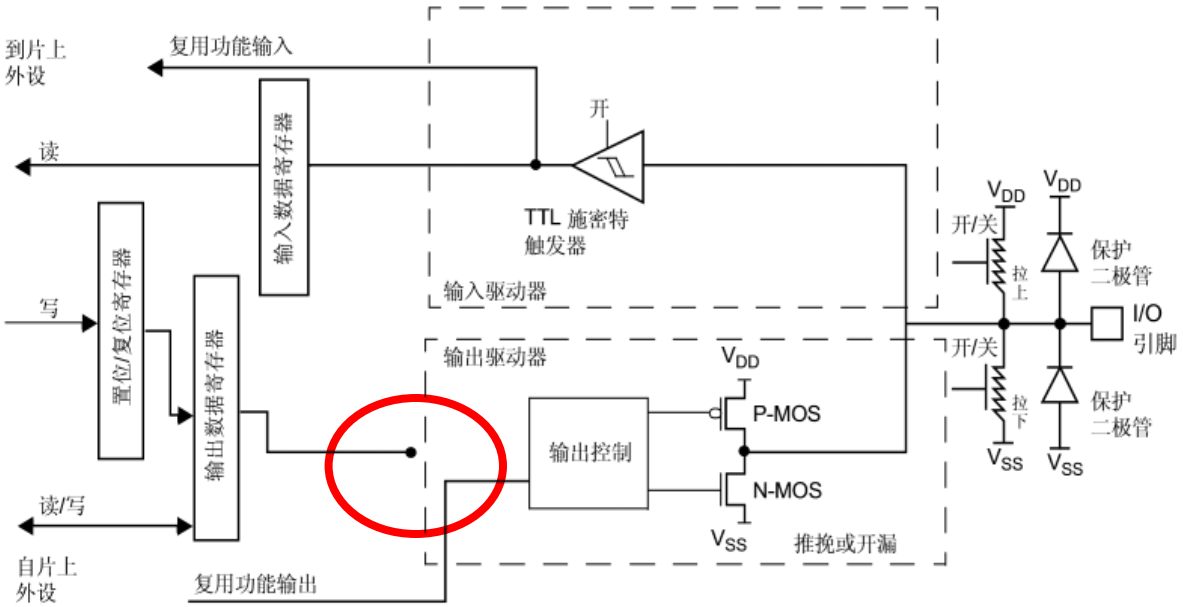
4、推挽复用输出模式



5.1 GPIO 结构原理

5.1.4 GPIO 复用功能配置

- 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

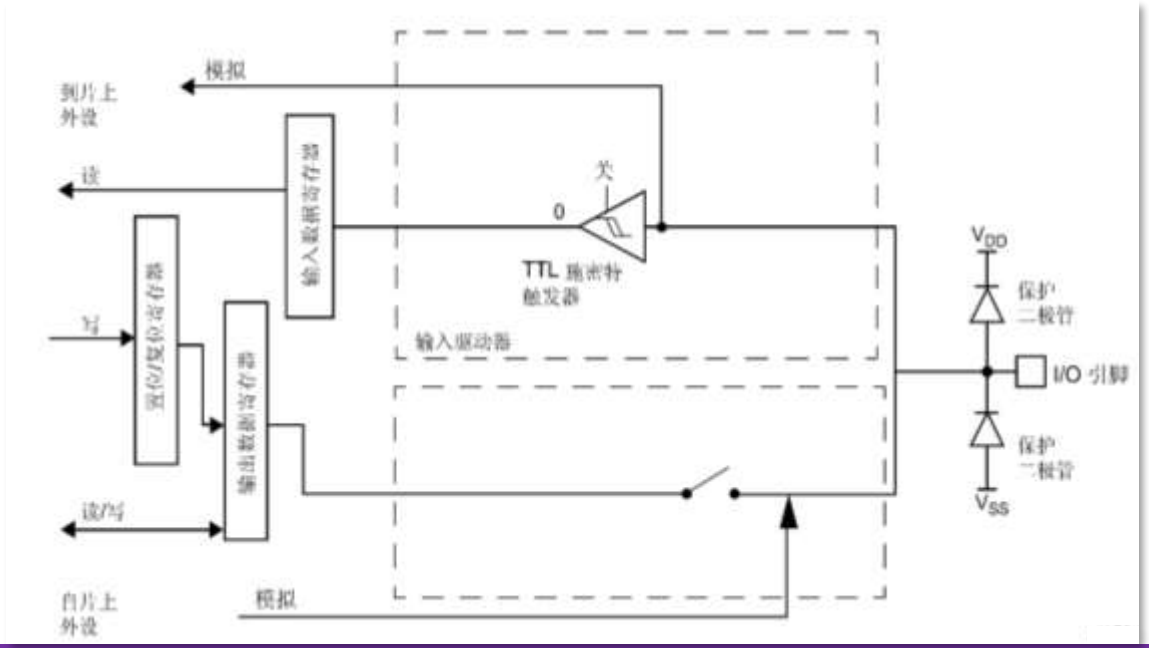


5.1 GPIO 结构原理

5.1.5 GPIO模拟配置

- 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”。

注意：在模拟配置中，*I/O* 引脚不能为 *5 V* 容忍。



5.2 GPIO相关寄存器

5.2 GPIO相关寄存器

每个GPIO端口有10个寄存器。

每个GPIO端口的寄存器相互独立！

1、GPIO端口的寄存器

1)、一个端口模式寄存器（GPIOx_MODER）
2)、一个端口输出类型寄存器(GPIOx_OTYPER)
3)、一个端口输出速度寄存器（GPIOx_OSPEEDR)
4)、一个端口上拉下拉寄存器（GPIOx_PUPDR)

} 4个32位配置寄存器

5)、一个端口输入数据寄存器（GPIOx_IDR）
6)、一个端口输出数据寄存器（GPIOx_ODR)

} 2个32位数据寄存器

7)、一个端口置位/复位寄存器（GPIOx_BSRR)
8)、一个端口配置锁存寄存器（GPIOx_LCKR)
9)、两个复用功能寄存器（低位GPIOx_AFRL & GPIOx_AFRH)

- 如果配置一个IO口需要2个位，那么刚好32位寄存器配置一组IO口16个IO口
- 如果配置一个IO口只需要1个位，一般高16位保留
- BSRR寄存器32位分为低16位BSRRL和高16位BSRRH，BSRRL配置一组IO口的16个IO口的置位状态（1），BSRRH配置复位状态（0）。

5.2 GPIO相关寄存器

2、端口模式寄存器(GPIOx_MODER)

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

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

5.2 GPIO相关寄存器

3、端口输出类型寄存器(GPIOx_OTYPER)

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

5.2 GPIO相关寄存器

4、端口输出速度寄存器(GPIOx_OSPEEDR)

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））

5.2 GPIO相关寄存器

5、端口上拉/下拉寄存器 (GPIOx_PUPDR)

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

5.2 GPIO相关寄存器

6、端口输入数据寄存器(GPIOx_IDR)

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

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。

5.2 GPIO相关寄存器

7、端口输出数据寄存器(GPIOx_ODR)

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)。

5.2 GPIO相关寄存器

8、端口置位/复位寄存器 (GPIOx_BSRR)

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**：端口 x 复位位 y (Port x reset bit y) (y = 0..15)

这些位为只写形式，只能在字、半字或字节模式下访问。读取这些位可返回值 0x0000。

0：不会对相应的 ODRx 位执行任何操作

1：对相应的 ODRx 位进行复位

注意：如果同时对 BSx 和 BRx 置位，则 BSx 的优先级更高。

位 15:0 **BSy**：端口 x 置位位 y (Port x set bit y) (y= 0..15)

这些位为只写形式，只能在字、半字或字节模式下访问。读取这些位可返回值 0x0000。

0：不会对相应的 ODRx 位执行任何操作

1：对相应的 ODRx 位进行置位

5.2 GPIO相关寄存器

9、复用功能低位寄存器(GPIOx_AFRL)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
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
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:0 **AFRLy**：端口 x 位 y 的复用功能选择 (Alternate function selection for port x bit y) (y = 0..7)
这些位通过软件写入，用于配置复用功能 I/O。

AFRLy 选择：

0000：AF0
0001：AF1
0010：AF2
0011：AF3
0100：AF4
0101：AF5
0110：AF6
0111：AF7
1000：AF8
1001：AF9
1010：AF10
1011：AF11
1100：AF12
1101：AF13
1110：AF14
1111：AF15

5.2 GPIO相关寄存器

10、复用功能高位寄存器 (GPIOx_AFRH)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
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
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

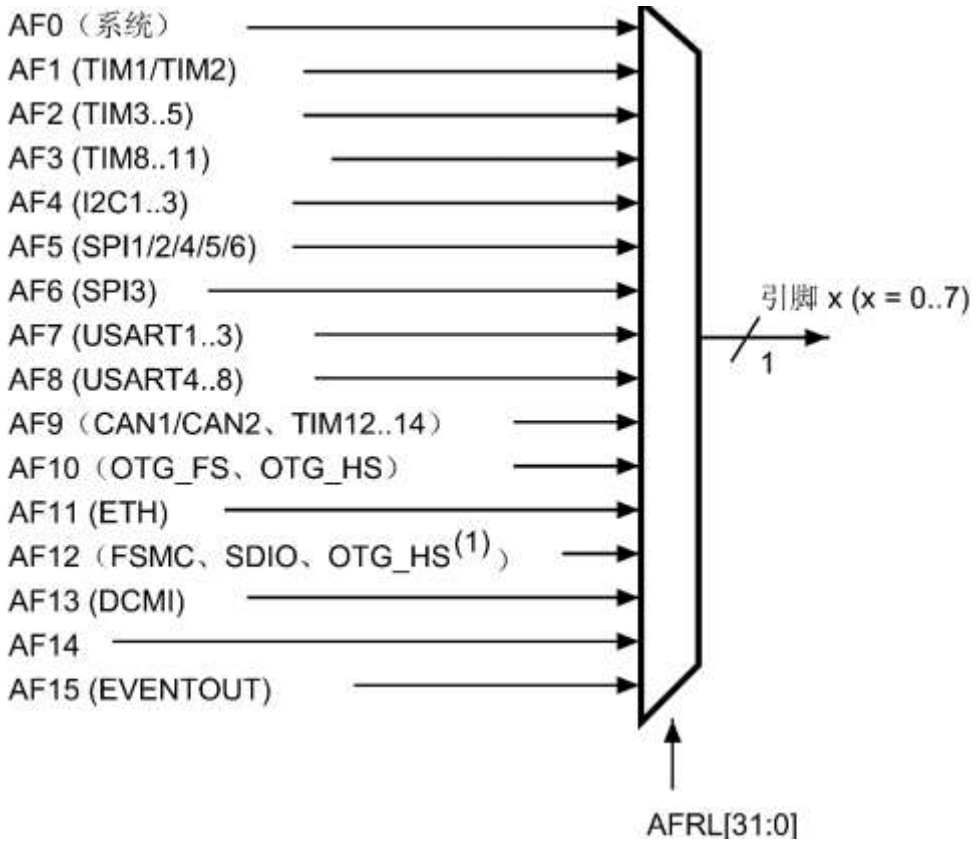
位 31:0 **AFRHy**：端口 x 位 y 的复用功能选择 (Alternate function selection for port x bit y) (y = 8.0.15)
这些位通过软件写入，用于配置复用功能 I/O。

AFRHy 选择：

0000：AF0
0001：AF1
0010：AF2
0011：AF3
0100：AF4
0101：AF5
0110：AF6
0111：AF7
1000：AF8
1001：AF9
1010：AF10
1011：AF11
1100：AF12
1101：AF13
1110：AF14
1111：AF15

5.2 GPIO相关寄存器

STM32F4XX参考手册（RM0090）p180



GPIOx_AFRL定义的0-7号引脚复用功能

5.3 GPIO典型应用步骤及常用库函数

5.3 GPIO典型应用步骤及常用库函数

5.3.1 GPIO典型应用步骤

使用库函数实现片上外设的控制，一般需要以下几步：

1、使能相应片上外设的时钟（非常重要），设计到的文件有

头文件：stm32f4xx_rcc.h

源文件：stm32f4xx_rcc.c

使用的主要函数：

```
RCC_AHB1PeriphClockCmd(uint32_t RCC_AHB1Periph, FunctionalState NewState)
RCC_AHB2PeriphClockCmd(uint32_t RCC_AHB2Periph, FunctionalState NewState)
RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState NewState)
RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState NewState)
```

例如：RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE)

2、设置对应于片上外设使用的GPIO工作模式。

3、如果使用复用功能，需要单独设置每一个GPIO引脚的复用功能。

4、在应用程序中读取引脚状态、控制引脚输出状态或使用复用功能完成特定功能。

5.3 GPIO典型应用步骤及常用库函数

5.3.2 GPIO 常用函数

头文件：stm32f4xx_gpio.h
源文件：stm32f4xx_gpio.c

1、1个初始化函数：

```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
```

例：GPIOA　例：GPIO_Pin_1

2、2个读取输入电平函数：IDR

```
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
```

3、2个读取输出电平函数：ODR

```
uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
```

4、4个设置输出电平函数：

```
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin); //BSRRL
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin); //BSRRH
void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);//BSRR
void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal); //ODR
```

5.3 GPIO典型应用步骤及常用库函数

5.3.2 GPIO 常用函数

1、初始化函数

```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef*
GPIO_InitStruct);
```

作用：初始化一个或者多个IO口（同一组）的工作模式，输出类型，速度以及上下拉方式。也就是一组IO口的4个配置寄存器。

（GPIOx->MODER, GPIOx->OSPEEDR,GPIOx->OTYPER,GPIOx->PUPDR）

GPIOx: GPIOA~GPIOK

```
typedef struct
{
  uint32_t            GPIO_Pin;//指定要初始化的端口
  GPIOMode_TypeDef    GPIO_Mode;//端口模式
  GPIOSpeed_TypeDef   GPIO_Speed;//速度
  GPIOOType_TypeDef   GPIO_OType;//输出类型
  GPIOPuPd_TypeDef    GPIO_PuPd;//上拉或者下拉
}GPIO_InitTypeDef;
```

- 注意：外设（包括GPIO)在使用之前，几乎都要先使能对应的时钟。

5.3 GPIO典型应用步骤及常用库函数

5.3.2 GPIO 常用函数

◆ GPIO_Init函数初始化样例：
将GPIOF的9、10引脚配置为推挽输出模式，速度100MHz，还能上拉功能。

```
GPIO_InitTypeDef  GPIO_InitStructure;
 //stm32f4xx_rcc.h    时钟使能-第1步
 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);//使能GPIOF时钟

 //GPIOF9,F10初始化设置   功能初始化-第2步
 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;//需要配置的IO口引脚
 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;//普通输出模式
 GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;//推挽输出
 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;;//100MHz
 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;//上拉
 GPIO_Init(GPIOF, &GPIO_InitStructure);//初始化GPIOF9,F10
```

■ 可以一次初始化一个IO组下的多个IO，前提是这些IO口的配置方式一样。

5.3 GPIO典型应用步骤及常用库函数

5.3.2 GPIO 常用函数

2、读取输入电平函数

1）、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
```

2）、 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
```

5.3 GPIO典型应用步骤及常用库函数

5.3.2 GPIO 常用函数

3、读取输出电平函数

1）、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
```

2）、 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
```

5.3 GPIO典型应用步骤及常用库函数

5.3.2 GPIO 常用函数

4、设置输出电平函数

1）、 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

例如：

```
GPIO_SetBits(GPIOA, GPIO_Pin_5);//设置GPIOA.5输出高电平
```

2）、 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。

3）、 void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);

4）、 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。

5.4 GPIO应用实例

5.4 GPIO应用实例

5.4.1 GPIO输出应用实例

要求：控制LED灯，以1s为周期进行闪烁。

1. 电路图

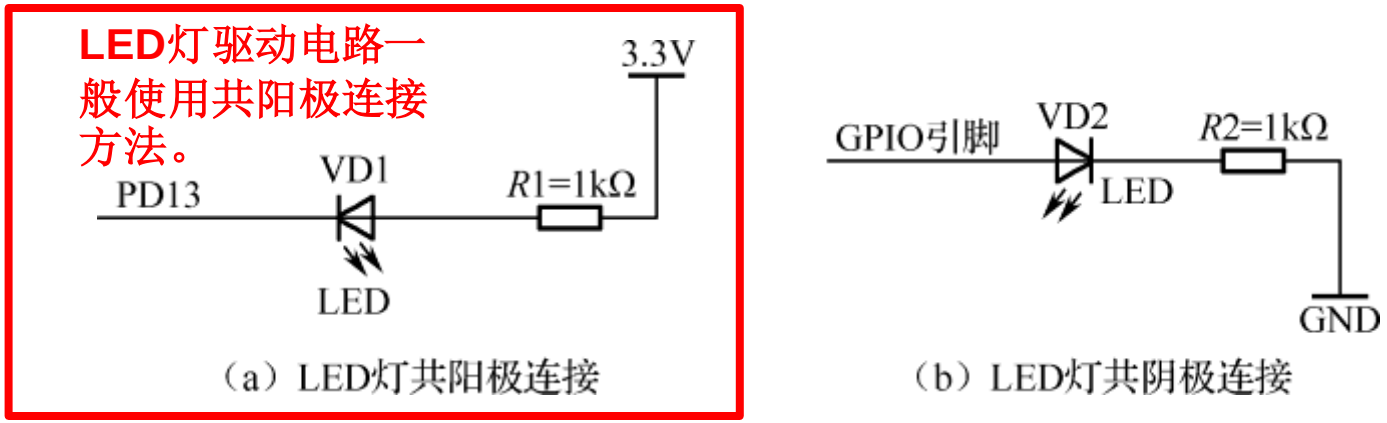


图5-10　LED灯共阳极和共阴极连接电路图

LED灯阴极连接GPIO引脚，阳极通过一个限流电阻连接到电源。在3.3V电压驱动下，限流电阻大小为330～1000Ω。

图5-10（a）中，当PD13被置为低电平时，LED灯VD1亮；

当PD13被置为高电平时，LED灯VD1灭。

5.4 GPIO应用实例

5.4.1 GPIO输出应用实例

2．编程要点

（1）使能GPIO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。

（2）初始化GPIO模式。调用函数GPIO_Init()。

（3）操作GPIO，设置引脚输出状态。调用函数GPIO_SetBits();或GPIO_ResetBits()或GPIO_ToggleBits()。

5.4 GPIO应用实例

5.4.1 GPIO输出应用实例

3. 程序实现

在main函数中完成如下功能:

（1）初始化SysTick。

（2）初始化LED灯的GPIO引脚。

（3）在无限循环中控制LED灯亮、灭。

在LED_Config函数中实现LED灯的GPIO引脚PD13的初始化：

（1）输出模式；

（2）推挽输出；

（3）连接上拉电阻；

（4）2MHz输出速度。

5.4 GPIO应用实例

5.4.1 GPIO输出应用实例

```
int main(void)
{
 /*初始化延时函数*/
 delay_init();
 /* LED 端口初始化 */
 LED_Config();
 /* 控制LED灯 */
 while (1)
 {
  LED_ON;  // 亮
  delay_ms(500);
  LED_OFF; // 灭
  delay_ms(500);

 }
}
```

```
void LED_Config(void)
{ /*定义一个GPIO_InitTypeDef类型的结构体*/
  GPIO_InitTypeDef GPIO_InitStructure;
  /*开启LED相关的GPIO外设时钟*/ 第一步
  RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOD, ENABLE);
  /*选择要控制的GPIO引脚*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
  /*设置引脚模式为模式*/
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  /*设置引脚的输出类型为推挽输出*/
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  /*设置引脚为上拉模式*/
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  /*设置引脚速率为2MHz */
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  /*调用库函数，使用上面配置的GPIO_InitStructure初始化GPIO*/
  GPIO_Init(GPIOD, &GPIO_InitStructure);// 第二步
  /*打开LED灯 */
  LED_ON;
}
```

设置工作方式

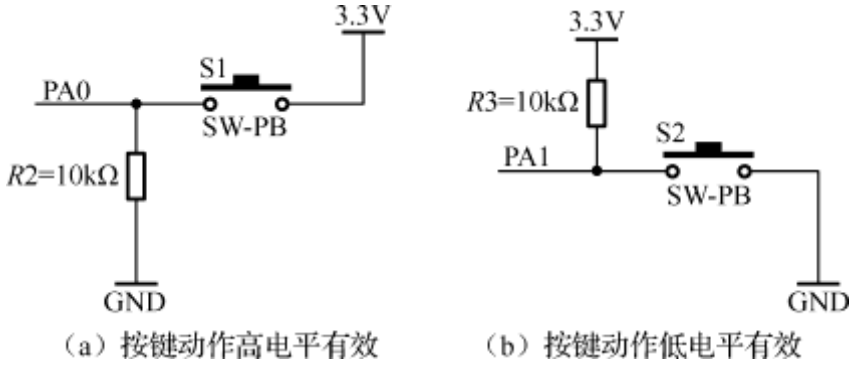
```
#define LED_OFF   GPIO_SetBits(GPIOD, GPIO_Pin_13);
#define LED_ON    GPIO_ResetBits(GPIOD, GPIO_Pin_13);
```

5.4 GPIO应用实例

5.4.2 GPIO输入应用实例

要求：使用独立按键S1和S2控制LED灯，按下S1点亮LED灯，按下S2熄灭LED灯。

1. 电路图



（a）按键动作高电平有效　　（b）按键动作低电平有效

图5-11　独立按键连接电路图

图5-11（a）中，独立按键有效按键动作检测电平是高电平。
图5-11（b）中，独立按键有效按键动作检测电平是低电平。

图5-12　机械按键抖动示意图

为了保证按键检测的正确性，需要在程序中加去抖动处理，防止抖动对检测的干扰。

5.4 GPIO应用实例

5.4.2 GPIO输入应用实例

独立按键软件扫描方法，需要在程序运行过程中循环或定时检测按键连接的引脚，①在首次检测到按键有效电平时，②延时10ms（不同按键延时不同）后，③再检测一次引脚电平，如能再次检测到有效电平，则是一次有效按键动作，*反之则认为是误操作。*

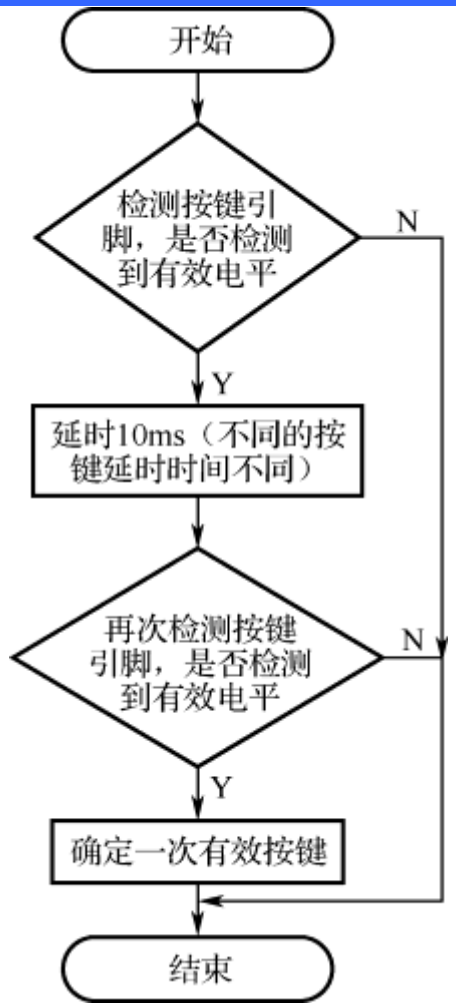
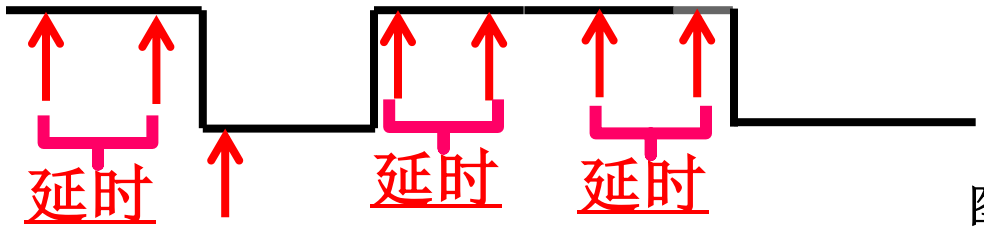


图5-13　一般独立按键检测程序流程图

如果按键检测有效电平为高电平



5.4 GPIO应用实例

5.4.2 GPIO输入应用实例

2. 编程要点

（1）使能GPIO时钟。调用函数RCC_AHB1PeriphClockCmd()。
（2）初始化GPIO模式。调用函数GPIO_Init()。
（3）操作GPIO，读取引脚状态。调用函数GPIO_ReadInputDataBit();

5.4 GPIO应用实例

5.4.2 GPIO输入应用实例

3. 主程序实现

在main函数中完成如下功能。

（1）初始化SysTick。

（2）初始化LED灯和按键的GPIO引脚。

（3）在无限循环中扫描按键，并在检测到有效按键动作后，控制LED灯亮或灭。

```
int main(void)
{
    delay_init();              //初始化SysTick，用于延时
    LED_Config();              //初始化LED灯的GPIO引脚
    Key_Config();              //初始化按键的GPIO引脚
    /*按键控制LED灯*/
    while (1)
    {          /*按键S1扫描判别，高电平有效*/
               if(Key_Scan(GPIOA,GPIO_Pin_0,1) == KEY_ON)
               {
                        LED_ON;              //点亮LED
               }
               /*按键S2扫描判别，低电平有效*/
               if(Key_Scan(GPIOA,GPIO_Pin_1,0) == KEY_ON)
               {
                        LED_OFF;             //熄灭LED
               }
    }
}
```

5.4 GPIO应用实例

5.4.2 GPIO输入应用实例

4. LED灯和按键的GPIO引脚初始化

LED灯的GPIO引脚初始化同5.4.1节。

按键的GPIO引脚PA0和PA1的初始化：
（1）输入模式；
（2）上拉/下拉电阻断开。

```
void Key_GPIO_Config(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
  /*开启按键GPIO口的时钟*/
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE)

  /*选择按键的引脚*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1;
  /*设置引脚为输入模式*/
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  /*设置引脚不上拉也不下拉*/
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  /*使用上面的结构体初始化按键*/
  GPIO_Init(GPIOA, &GPIO_InitStructure);

}
```

5.4 GPIO应用实例

5.4.2 GPIO输入应用实例

5. 按键扫描程序

```
uint8_t Key_Scan(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin,uint8_t Key_Lvl)
{
  /*检测是否有按键按下*/
  if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == Key_Lvl )//第一次检测电平
  {
     delay_ms(10);      //去抖动
     if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == Key_Lvl)//第二次检测电平
        return  KEY_ON;          //确认有效按键动作返回
     else
        return  KEY_OFF;         //无有效按键动作返回
  }
  else
        return  KEY_OFF; //无有效按键动作返回
}
```

```
#define KEY_OFF         0;
#define KEY_ON          1;
```

5.4 GPIO应用实例

5.4.3 GPIO复用应用实例

STM32有很多的内置外设，这些外设的外部引脚都是与GPIO复用的。也就是说，一个GPIO如果可以复用为内置外设的功能引脚，那么当这个GPIO作为内置外设使用的时候，就叫做端口复用。

例如：串口1 的发送接收引脚是PA9,PA10，当我们把PA9,PA10不用作GPIO，而用做复用功能串口1的发送接收引脚的时候，叫端口复用。

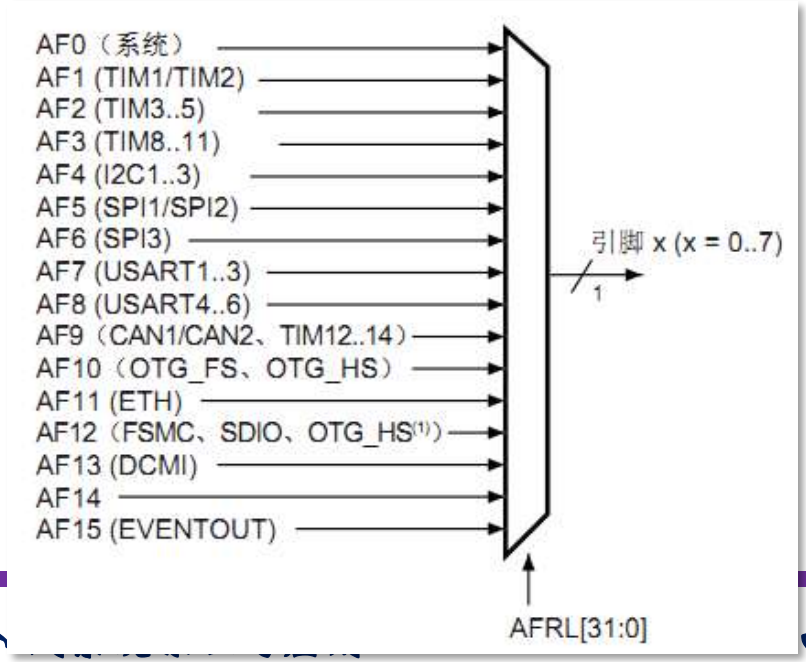
Pin number					Pin name (function after reset)(1)	Pin type	I / O structure	Notes	Alternate functions	Additional functions
LQFP64	LQFP100	LQFP144	UFBGA176	LQFP176						
41	67	100	F15	119	PA8	I/O	FT		MCO1 / USART1_CK/ TIM1_CH1/ I2C3_SCL/ OTG_FS_SOF/ EVENTOUT	
42	68	101	E15	120	PA9	I/O	FT		USART1_TX/ TIM1_CH2 / I2C3_SMBA / DCMI_D0/ EVENTOUT	OTG_FS_VBUS
43	69	102	D15	121	PA10	I/O	FT		USART1_RX/ TIM1_CH3/ OTG_FS_ID/DCMI_D1/ EVENTOUT	

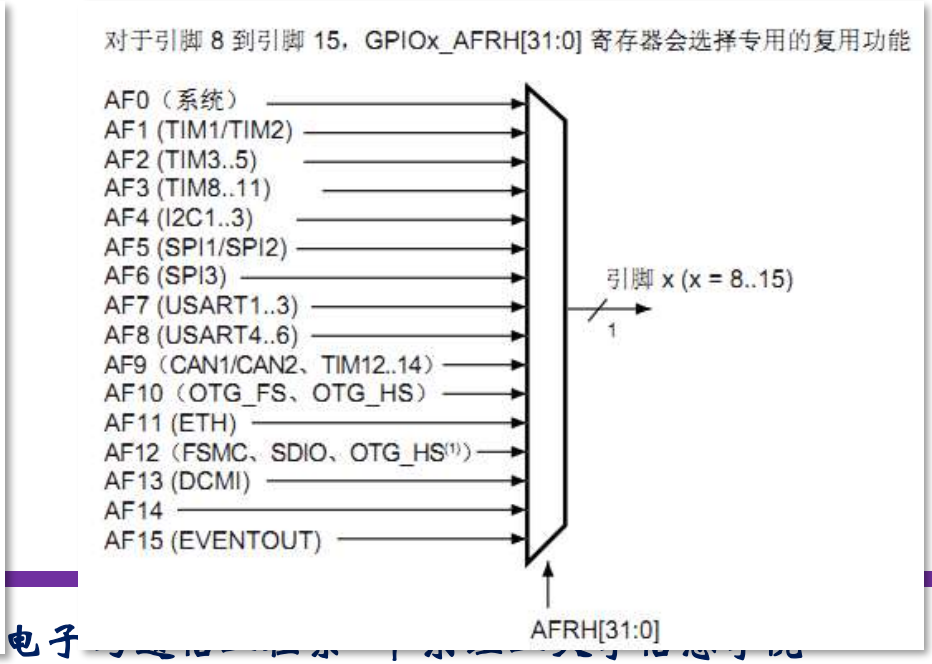
5.4 GPIO应用实例

5.4.3 GPIO复用应用实例

1STM32F4系列微控制器IO引脚通过一个复用器连接到内置外设或模块。该复用器一次只允许一个外设的复用功能（AF）连接到对应的IO口。这样可以确保共用同一个IO引脚的外设之间不会发生冲突。

每个IO引脚都有一个复用器，该复用器采用16路复用功能输入（AF0到AF15），可通过GPIOx_AFRL(针对引脚0-7)和GPIOx_AFRH（针对引脚8-15）寄存器对这些输入进行配置，每四位控制一路复用。





5.4 GPIO应用实例

5.4.3 GPIO复用应用实例

编程要点

端口复用为复用功能配置过程

-以PA9,PA10配置为USART1_TXD和USART1_RXD引脚为例

①GPIO端口时钟使能。

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);
```

复用外设时钟使能。

比如你要将端口PA9,PA10复用为串口，所以要使能串口时钟。

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
```

②端口模式配置为复用功能。GPIO_Init()函数。

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;//复用功能
```

5.4 GPIO应用实例

5.4.3 GPIO复用应用实例

③配置GPIOx_AFRL或者GPIOx_AFRH寄存器，将IO连接到所需的AFx。　对 复用功能高位寄存器操作

```
/*PA9连接AF7，复用为USART1_TX */
GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_USART1);
/* PA10连接AF7,复用为USART1_RX*/
GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_USART1);
```

通过以上几个步骤就可以将PA9和PA10配置为USART1_TXD和USART1_RXD的复用功能。之后，再通过对USART1通信通能的初始化设置，即可和外部进行通信。

其他偏上外设的引脚复用配置过程和上面的讲解类似。

5.4 GPIO应用实例

5.4.3 GPIO复用应用实例

◆PA9,PA10复用为串口1的配置过程

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE); //使能GPIOA时钟 ①
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);//使能USART1时钟 ①

//USART1端口配置②
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10; //GPIOA9与GPIOA10
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;//复用功能
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;   //速度50MHz
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //推挽复用输出
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP; //上拉
GPIO_Init(GPIOA,&GPIO_InitStructure); //初始化PA9，PA10

//串口1对应引脚复用映射 ③
GPIO_PinAFConfig(GPIOA,GPIO_PinSource9,GPIO_AF_USART1); //GPIOA9复用为USART1
GPIO_PinAFConfig(GPIOA,GPIO_PinSource10,GPIO_AF_USART1); //GPIOA10复用为USART1
```

5.4 GPIO应用实例

5.4.4 矩阵按键应用

1. 矩阵按键原理

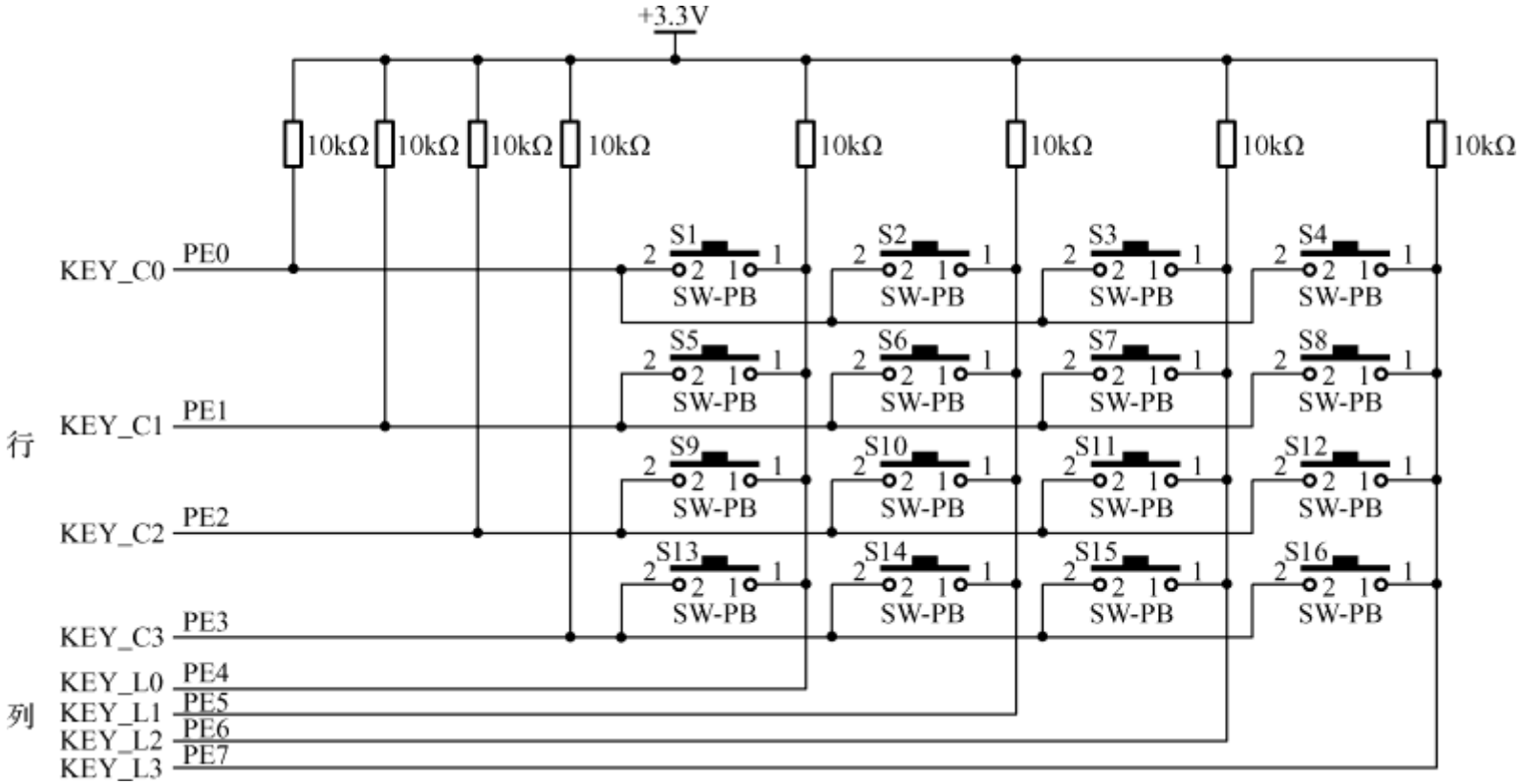


图5-14　4×4矩阵按键电路原理图

5.4 GPIO应用实例

5.4.4 矩阵按键应用

1. 矩阵按键原理

1）逐行扫描

通过在矩阵按键的每一条行线上轮流输出低电平，检测矩阵按键的列线，当检测到的列线不全为高电平的时候，说明有按键按下。然后，根据当前输出低电平的行号和检测到低电平的列号组合，判断是哪一个按键被按下。

5.4 GPIO应用实例

5.4.4 矩阵按键应用

1. 矩阵按键原理

2）行列扫描

首先，在全部行线上输出低电平，检测矩阵按键的列线，当检测到的列线不全为高电平的时候，说明有按键按下，并判断是哪一列有按键按下。

然后，反过来，在全部列线上输出低电平，检测矩阵按键的行线，当检测到的行线不全为高电平的时候，说明有按键按下，并判断是哪一行有按键按下。

最后，根据检测到的行号和检测的列号组合，以判断是哪一个按键被按下。

5.4 GPIO应用实例

5.4.4 矩阵按键应用

1．矩阵按键原理

2）行列扫描

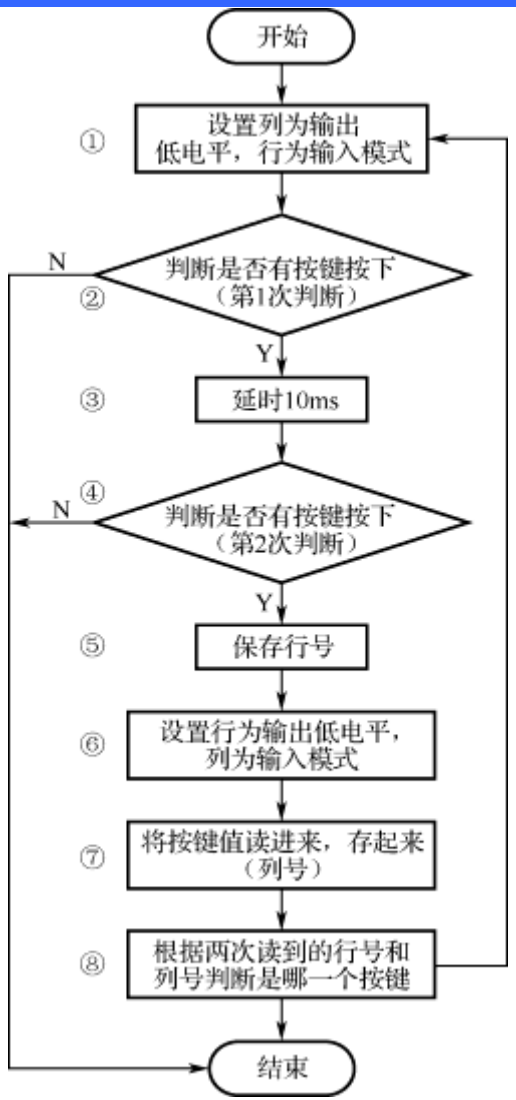


图5-15　行列扫描法的程序流程图

5.4 GPIO应用实例

5.4.4 矩阵按键应用

2. 矩阵按键程序实现

根据图5-13的矩阵原理，编写矩阵按键应用程序，轮徇S1～S16按键动作，若检测到S15（15号）按键按下，则反转LED灯

3. 编程要点

（1）使能LED灯和矩阵按键的GPIO时钟。调用函数：

```
RCC_AHB1PeriphClockCmd();
```

（2）初始化LED灯（同5.4.1节中“初始化程序”）和矩阵按键的GPIO模式。调用函数：

```
GPIO_Init();
```

（3）编写矩阵按键扫描程序。

5.4 GPIO应用实例

5.4.4 矩阵按键应用

4. 主程序

```
u8 key_value;
int main(void)
{
  delay_init();                                    //初始化延时函数
  LED_Config();                                    //初始化LED灯引脚
  Matrix_Key_Config();                                    //初始化矩阵按键引脚
   /*轮询矩阵按键状态，若15号（S15）按键按下，则反转LED灯*/
 while(1)
 {
        key_value=Matrix_Key_Scan();
        if( key_value !=0xff)
        {
             if(key_value==15)//如果检测到的按键是S15，则将LED灯状态反转
             LED_TOGGLE;          //反转LED灯状态
        }
 }
}
```

5.4 GPIO应用实例

5.4.4 矩阵按键应用

5. 矩阵按键GPIO初始化

```
void Matrix_Key_Config(void)
{
 GPIO_InitTypeDef GPIO_InitStructure;

 RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE,ENABLE);  /*使能GPIO的时钟*/

 GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|
 GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7; /*选择按键的引脚*/

 GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; /*设置引脚模式为输出模式*/

 GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;    /*设置引脚的输出类型为开漏输出*/

 GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  /*设置引脚模式为不上拉下拉模式*/

 GPIO_InitStructure.GPIO_Speed = GPIO_Speed_25MHz; /*设置引脚输出速度为25MHz*/

 GPIO_Init(GPIOE, &GPIO_InitStructure);  /*使用上面的结构体初始化按键*/
}
```

5.4 GPIO应用实例

5.4.4 矩阵按键应用

6. 矩阵按键扫描程序

略

请结合 图5-15 行列扫描法的程序流程图和教材中的代码进行分析。