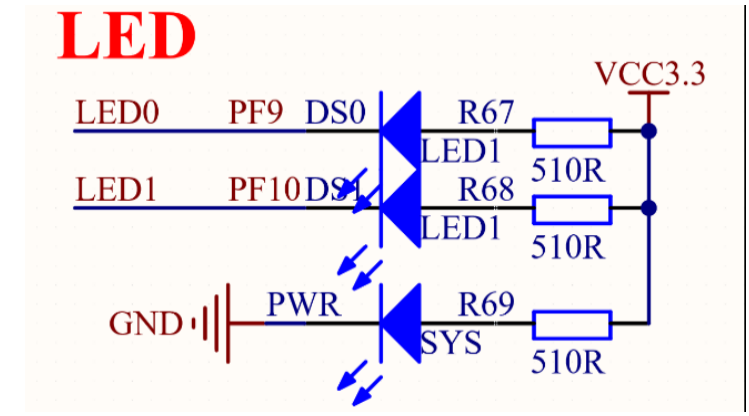
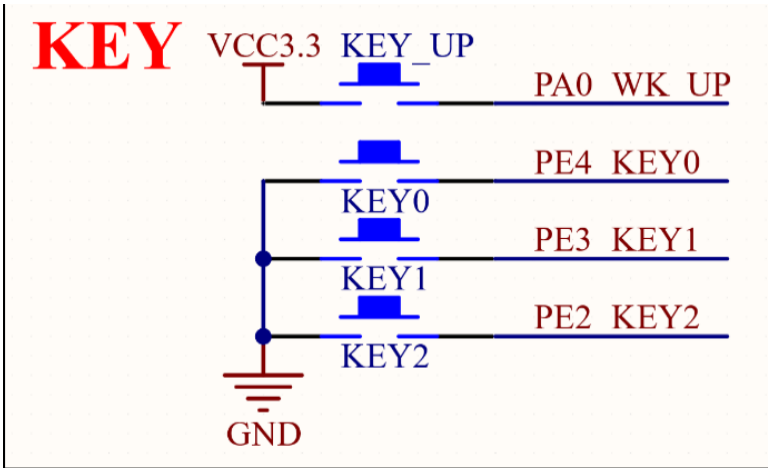
GPIO

1.实物与原理图的连接关系

观察部分外设的原理图LED KEY





我们可以看到LED和KEY是存在两个通道的，一个通道为VCC/GND，另一个通道连接GPIO引脚，GPIO可以通过代码来控制其模式

当我们试图点亮LED时，其本质是在LED所在电路产生电流，而产生电流需要电势差，LED作为发光二极管，电流只能单向导通，其右端已经连接了VCC3.3（高电平），GPIOF的GPIO_Pin_9此时接入低电平便可产生电势差，从而点亮LED。

2.GPIO寄存器

```
每个通用 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)
2 个 32 位复用功能选择寄存器(GPIOx_AFRH 和 GPIOx_AFRL)。
```

标准库代码如下：

```
//开启LED的GPIOF时钟
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);
//LED0、LED1初始化
GPIO_InitTypeDef  GPIO_InitStructure;                    //定义初始化结构体
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;  //配置引脚
//GPIO4个配置寄存器配置
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;            //配置GPIO_Mode为OUT
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;           //配置GPIO_OType为推挽输出
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;       //配置GPIO_Speed为100MHz
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;         //配置上拉为无上拉
GPIO_Init(GPIOF, &GPIO_InitStructure);
```

管脚定义如下：

```
#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 selected */
#define GPIO_Pin_4                 ((uint16_t)0x0010)  /* Pin 4 selected */
#define GPIO_Pin_5                 ((uint16_t)0x0020)  /* Pin 5 selected */
#define GPIO_Pin_6                 ((uint16_t)0x0040)  /* Pin 6 selected */
#define GPIO_Pin_7                 ((uint16_t)0x0080)  /* Pin 7 selected */
#define GPIO_Pin_8                 ((uint16_t)0x0100)  /* Pin 8 selected */
#define GPIO_Pin_9                 ((uint16_t)0x0200)  /* Pin 9 selected */
#define GPIO_Pin_10                ((uint16_t)0x0400)  /* Pin 10 selected */
#define GPIO_Pin_11                ((uint16_t)0x0800)  /* Pin 11 selected */
#define GPIO_Pin_12                ((uint16_t)0x1000)  /* Pin 12 selected */
#define GPIO_Pin_13                ((uint16_t)0x2000)  /* Pin 13 selected */
#define GPIO_Pin_14                ((uint16_t)0x4000)  /* Pin 14 selected */
#define GPIO_Pin_15                ((uint16_t)0x8000)  /* Pin 15 selected */
#define GPIO_Pin_All               ((uint16_t)0xFFFF)  /* All pins selected */
```

GPIO引脚的定义就是16位的值，这样可以非常方便的将引脚的值配置给对应的寄存器

Q1.GPIO_Mode为什么是输出模式呢？如果按照电势差理论，LED的供电是由VCC3.3提供，GPIO_Pin_9并未参与电压输出

A1.个人理解为这里的GPIO_Mode的输入输出只是输出控制命令，进而控制对应模块的高低电平。

GPIO配置寄存器

GPIO_MODER

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

GPIO port mode register

偏移地址：0x00

复位值：

- 0xA800 0000（端口 A）
- 0x0000 0280（端口 B）
- 0x0000 0000（其它端口）

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

```
typedef enum
{
  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;
```

以上为GPIO_Mode的结构体定义，当我们使用GPIO_Mode_IN/OUT时，可以配合stm32f4xx_gpio.c中的函数读取输入的信息（KEY）或者控制输入的信息，对GPIO_Mode的配置，本质上是操作GPIO_MODER寄存器

```
/******************  Bits definition for GPIO_MODER register  *****************/
#define GPIO_MODER_MODER0                    ((uint32_t)0x00000003)     //0x3--
-0011
#define GPIO_MODER_MODER0_0                  ((uint32_t)0x00000001)
#define GPIO_MODER_MODER0_1                  ((uint32_t)0x00000002)

#define GPIO_MODER_MODER1                    ((uint32_t)0x0000000C)     //0xC--
-1100
#define GPIO_MODER_MODER1_0                  ((uint32_t)0x00000004)
#define GPIO_MODER_MODER1_1                  ((uint32_t)0x00000008)
//配置GPIO_MODER寄存器的值，GPIO_MODER寄存器是32位寄存器，要对最多16个引脚进行控制，所以每两个位控制一个引脚
//翻阅参考手册可知 00为输入 01为输出 10位AF复用 11为模拟
//00输入模式在stm32f4xx.h中并没有定义 00为复位值
```

GPIO_OTYPER

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

GPIO port output type register

偏移地址：0x04

复位值：0x0000 0000

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

```
/**
  * @brief  GPIO Output type enumeration
  */
typedef enum
{
  GPIO_OType_PP = 0x00,
  GPIO_OType_OD = 0x01
}GPIOOType_TypeDef;
#define IS_GPIO_OTYPE(OTYPE) (((OTYPE) == GPIO_OType_PP) || ((OTYPE) ==
GPIO_OType_OD))
```

配置输出模式，注意在GPIO_MODER寄存器配置为输入和模拟输入(00/11)时，GPIO_OTYPER和GPIO_OSPEEDR寄存器不起作用

这两个配置寄存器的都是以O开头的

GPIOx_OSPEEDR

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

GPIO port output speed register

偏移地址：0x08

复位值：

- 0x0000 00C0（端口 B）
- 0x0000 0000（其它端口）

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））

- ```
  /* @brief  GPIO Output Maximum frequency enumeration  */
     typedef enum
     {
     GPIO_Low_Speed     = 0x00, /*!< Low speed    */
     GPIO_Medium_Speed  = 0x01, /*!< Medium speed */
     GPIO_Fast_Speed    = 0x02, /*!< Fast speed   */
     GPIO_High_Speed    = 0x03  /*!< High speed   */
     }GPIOSpeed_TypeDef;
  /* Add legacy definition */
  #define  GPIO_Speed_2MHz    GPIO_Low_Speed
  #define  GPIO_Speed_25MHz   GPIO_Medium_Speed
  #define  GPIO_Speed_50MHz   GPIO_Fast_Speed
  #define  GPIO_Speed_100MHz  GPIO_High_Speed

  #define IS_GPIO_SPEED(SPEED) (((SPEED) == GPIO_Low_Speed) || ((SPEED) ==
  GPIO_Medium_Speed) || \
                                ((SPEED) == GPIO_Fast_Speed)||  ((SPEED) ==
  GPIO_High_Speed))
  ```

  配置GPIO的输出速度

GPIOx_PUPDR

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

GPIO port pull-up/pull-down register

偏移地址：0x0C

复位值：

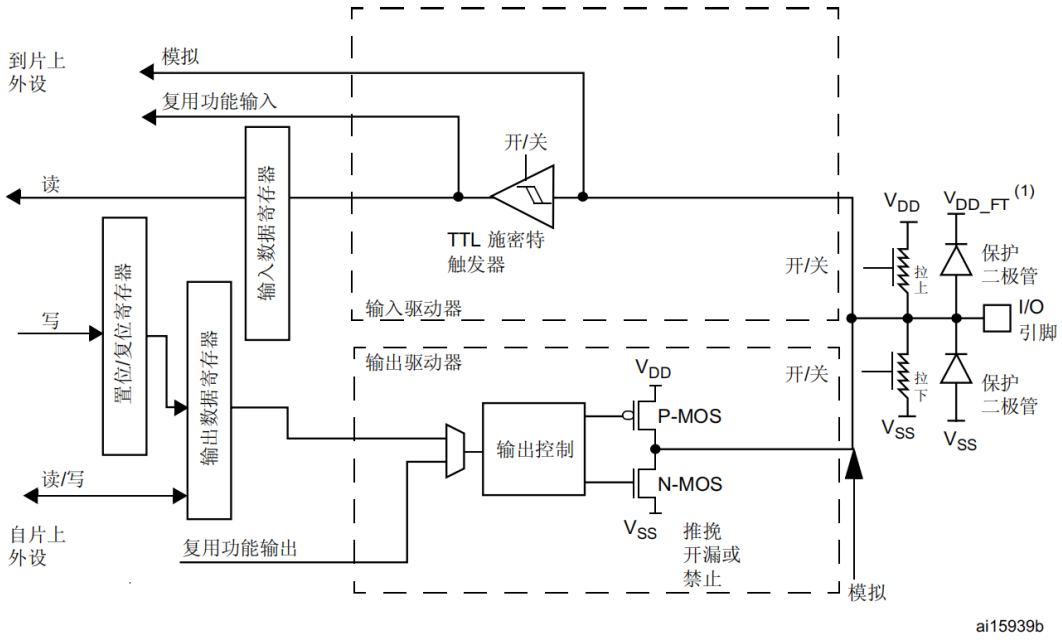
- 0x6400 0000（端口 A）
- 0x0000 0100（端口 B）
- 0x0000 0000（其它端口）

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

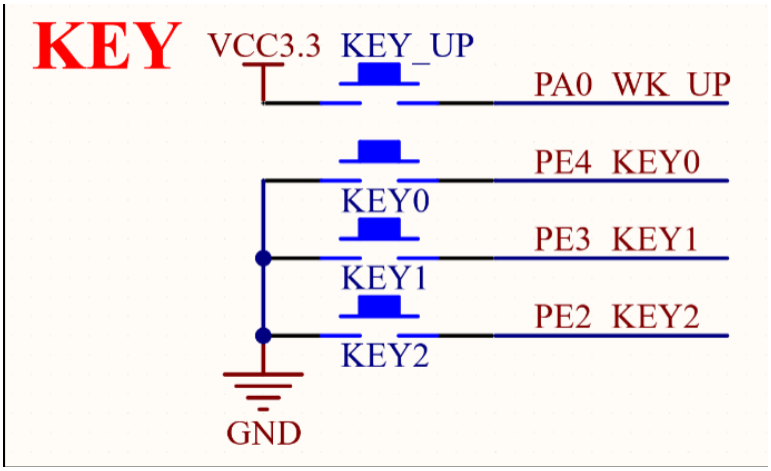
位 2y:2y+1 **PUPDRy[1:0]**：端口 x 配置位 (Port x configuration bits) (y = 0..15)
这些位通过软件写入，用于配置 I/O 上拉或下拉。
00：无上拉或下拉
01：上拉
10：下拉
11：保留

- ```
  /* @brief  GPIO Configuration PullUp PullDown enumeration  */
        typedef enum
        {
      GPIO_PuPd_NOPULL = 0x00,
      GPIO_PuPd_UP     = 0x01,
      GPIO_PuPd_DOWN   = 0x02
        }GPIOPuPd_TypeDef;
        #define IS_GPIO_PUPD(PUPD) (((PUPD) == GPIO_PuPd_NOPULL) || ((PUPD) ==
  GPIO_PuPd_UP) || \
                                    ((PUPD) == GPIO_PuPd_DOWN))
  ```

配置GPIO上拉/下拉功能，可以根据以下原理图



可以根据需要配置GPIO输入输出信号的默认电平，例如GPIO输入实验时，KEY在未按下的时候是高阻态，由于我们需要按下时引脚输出低电平，所以应当配置上拉电阻使得在没有按下时输出高电平



GPIO_Init（）函数

以上对配置寄存器的值进行配置只是设置了对应的值，值还未写入对应的寄存器中，通过GPIO_Init()函数统一完成校验并写入值

```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
{
  uint32_t pinpos = 0x00, pos = 0x00 , currentpin = 0x00;
  //该函数在配置前还需要对设备、引脚号、模式进行检验
  /* Check the parameters */
  assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
  assert_param(IS_GPIO_PIN(GPIO_InitStruct->GPIO_Pin));
  assert_param(IS_GPIO_MODE(GPIO_InitStruct->GPIO_Mode));
  assert_param(IS_GPIO_PUPD(GPIO_InitStruct->GPIO_PuPd));

  /* ------------------------ Configure the port pins ---------------- */
  /*-- GPIO Mode Configuration --*/
  //通过对GPIO_MODER寄存器的配置我们可以发现，它通过一些代码技巧完成了默认值（置位）的功能
```

```
  for (pinpos = 0x00; pinpos < 0x10; pinpos++)
  {
    pos = ((uint32_t)0x01) << pinpos;
    /* Get the port pins position */
    currentpin = (GPIO_InitStruct->GPIO_Pin) & pos;
if (currentpin == pos)
{
  GPIOx->MODER  &= ~(GPIO_MODER_MODER0 << (pinpos * 2));
  GPIOx->MODER |= (((uint32_t)GPIO_InitStruct->GPIO_Mode) << (pinpos * 2));

  if ((GPIO_InitStruct->GPIO_Mode == GPIO_Mode_OUT) || (GPIO_InitStruct-
>GPIO_Mode == GPIO_Mode_AF))
  {
    /* Check Speed mode parameters */
    assert_param(IS_GPIO_SPEED(GPIO_InitStruct->GPIO_Speed));

    /* Speed mode configuration */
    GPIOx->OSPEEDR &= ~(GPIO_OSPEEDER_OSPEEDR0 << (pinpos * 2));
    GPIOx->OSPEEDR |= ((uint32_t)(GPIO_InitStruct->GPIO_Speed) << (pinpos * 2));

    /* Check Output mode parameters */
    assert_param(IS_GPIO_OTYPE(GPIO_InitStruct->GPIO_OType));

    /* Output mode configuration*/
    GPIOx->OTYPER  &= ~((GPIO_OTYPER_OT_0) << ((uint16_t)pinpos)) ;
    GPIOx->OTYPER |= (uint16_t)(((uint16_t)GPIO_InitStruct->GPIO_OType) <<
((uint16_t)pinpos));
  }

  /* Pull-up Pull down resistor configuration*/
  GPIOx->PUPDR &= ~(GPIO_PUPDR_PUPDR0 << ((uint16_t)pinpos * 2));
  GPIOx->PUPDR |= (((uint32_t)GPIO_InitStruct->GPIO_PuPd) << (pinpos * 2));
}
```

```
 }
}
```

GPIO配置寄存器总结

```
1/开启GPIO时钟/
2/定义结构体并初始化/
    要根据具体设备配置对应的模式
3/GPIO_Init();/
```

GPIO数据寄存器

每个 GPIO 都具有 2 个 16 位数据寄存器：输入和输出数据寄存器（GPIOx_IDR 和GPIOx_ODR）。GPIOx_ODR 用于存储待输出数据，可对其进行读/写访问。通过 I/O 输入的数据存储到输入数据寄存器(GPIOx_IDR) 中，它是一个只读寄存器。

```
/******************  Bits definition for GPIO_IDR register  ******************/
#define GPIO_IDR_IDR_0                      ((uint32_t)0x00000001)
//GPIO_Pin_0
#define GPIO_IDR_IDR_1                      ((uint32_t)0x00000002)
/******************  Bits definition for GPIO_ODR register  ******************/
#define GPIO_ODR_ODR_0                      ((uint32_t)0x00000001)
#define GPIO_ODR_ODR_1                      ((uint32_t)0x00000002)
```

GPIO_IDR

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

GPIO port input data register

偏移地址：0x10

复位值：0x0000 XXXX（其中 X 表示未定义）

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。

输入数据寄存器 (GPIOx_IDR) 每隔 1 个 AHB1 时钟周期捕获一次 I/O 引脚的数据。

[15:0]对应16个IO引脚 神奇的是只需要存放数据只需要一位，由此可见寄存器的每一位都是宝贵的，一位就可以确定引脚的输入信息

```
/**
  * @brief  Reads the specified input port pin.
  * @param  GPIOx: where x can be (A..K) to select the GPIO peripheral for
STM32F405xx/407xx and STM32F415xx/417xx devices
  * x can be (A..I) to select the GPIO peripheral for STM32F42xxx/43xxx devices.
  * x can be (A, B, C, D and H) to select the GPIO peripheral for STM32F401xx
devices.
  * @param  GPIO_Pin: specifies the port bit to read.
  * This parameter can be GPIO_Pin_x where x can be (0..15).
  * @retval The input port pin value.
    */
    uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
    {
      uint8_t bitstatus = 0x00;

  /* Check the parameters */
  assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
  assert_param(IS_GET_GPIO_PIN(GPIO_Pin));

  if ((GPIOx->IDR & GPIO_Pin) != (uint32_t)Bit_RESET)
  {
    bitstatus = (uint8_t)Bit_SET;
  }
  else
  {
    bitstatus = (uint8_t)Bit_RESET;
```

```
  }
  return bitstatus;
}
```

GPIO_ODR

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

GPIO port output data register

偏移地址：0x14

复位值：0x0000 0000

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)。

GPIO_ODR寄存器用来存放输出的数据，可以通过软件来控制读取和写入

`GPIOx_ODR` 中的每个数据位对应于 `GPIOx_BSRR` 中的两个控制位/高位和低位/：`BSRR(i)` 和 `BSRR(i+SIZE)`。当写入 `1` 时，`BSRR(i)` 位会/置位/对应的 `ODR(i)` 位。当写入 `1` 时，`BSRR(i+SIZE)` 位会/清零/`ODR(i)` 对应的位。

但是如果要进行置位/复位，应当去操作GPIO_BSRR寄存器

GPIO置位/复位寄存器

GPIO_BSRR

7.4.7 GPIO 端口置位/复位寄存器 (GPIOx_BSRR) (x = A..I)

GPIO port bit set/reset register

偏移地址：0x18

复位值：0x0000 0000

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

GPIO_BSRR中的控制位在置为1的时候才有控制效果

我们可以通过操作GPIO_ODR在控制GPIO_BSRR来进行置位或者复位操作

```
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  /* Check the parameters */
  assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
  assert_param(IS_GPIO_PIN(GPIO_Pin));
  GPIOx->BSRRH = GPIO_Pin;                    //将需要置位的引脚号写入置位寄存器
}
```

GPIOx->BSRRH = GPIO_Pin是将GPIO_Pin置为0

置位复位寄存器 (GPIOx_BSRR) 是一个 32 位寄存器，它允许应用程序在输出数据寄存器(GPIOx_ODR)中对各个单独的数据位执行置位和复位操作。

```
void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal)
{
  /* Check the parameters */
  assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
  assert_param(IS_GET_GPIO_PIN(GPIO_Pin));
  assert_param(IS_GPIO_BIT_ACTION(BitVal));

  if (BitVal != Bit_RESET)
  {
    GPIOx->BSRRL = GPIO_Pin;                  //写入置位引脚号
  }
  else
  {
    GPIOx->BSRRH = GPIO_Pin ;                 //写入复位引脚号
  }
}
```

GPIO配置锁定寄存器

GPIO_LCKR

通过将特定的写序列应用到 GPIOx_LCKR 寄存器，可以冻结 GPIO 控制寄存器。
冻结的寄存器包括 GPIOx_MODER、GPIOx_OTYPER、GPIOx_OSPEEDR、GPIOx_PUPDR、GPIOx_AFRL 和 GPIOx_AFRH。
要对 GPIOx_LCKR 寄存器执行写操作，必须应用特定的写、读序列。当正确的 LOCK 序列应用到此寄存器的第16位后，会使用 LCKR[15:0] 的值来锁定 I/O 的配置（在写序列期间，LCKR[15:0] 的值必须相同）。
将 LOCK 序列应用到某个端口位后，在执行下一次复位之前，将无法对该端口位的值进行修改。
每个 GPIOx_LCKR 位都会冻结控制寄存器(GPIOx_MODER、GPIOx_OTYPER、GPIOx_OSPEEDR、GPIOx_PUPDR、GPIOx_AFRL 和 GPIOx_AFRH）中的对应位。
LOCK 序列（参见第 7.4.8 节：GPIO 端口配置锁定寄存器 (GPIOx_LCKR) (x = A..I)）只能 通过对 GPIOx_LCKR 寄存器进行字（32 位长）访问的方式来执行，因为 GPIOx_LCKR 的 第 16 位必须与 [15:0] 位同时置位。

不是每一个寄存器都是可以按位进行数据读写的，GPIOx_LCKR寄存器便是如此，操作一个GPIO引脚的锁存需要对至少两个位进行置位，即使能位--16位和对应的LCKR[15:0]操作位

```
/** @brief  Locks GPIO Pins configuration registers.
```

```
  * @note   The locked registers are GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR,
  * GPIOx_PUPDR, GPIOx_AFRL and GPIOx_AFRH.
  * @note   The configuration of the locked GPIO pins can no longer be modified
  * until the next reset.
  * @param  GPIOx: where x can be (A..K) to select the GPIO peripheral for
STM32F405xx/407xx and STM32F415xx/417xx devices
  * x can be (A..I) to select the GPIO peripheral for STM32F42xxx/43xxx devices.
  * x can be (A, B, C, D and H) to select the GPIO peripheral for STM32F401xx
devices.
  * @param  GPIO_Pin: specifies the port bit to be locked.
  * This parameter can be any combination of GPIO_Pin_x where x can be (0..15).
  * @retval None
    */
    void GPIO_PinLockConfig(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
    {
      __IO uint32_t tmp = 0x00010000;

  /* Check the parameters */
  assert_param(IS_GPIO_ALL_PERIPH(GPIOx));
  assert_param(IS_GPIO_PIN(GPIO_Pin));
//与操作是为了同时配置引脚对应的控制位和第16位使能位
  tmp |= GPIO_Pin;
  /* Set LCKK bit */
  GPIOx->LCKR = tmp;
  /* Reset LCKK bit */
  GPIOx->LCKR =  GPIO_Pin;
  /* Set LCKK bit */
  GPIOx->LCKR = tmp;
  /* Read LCKK bit*/
  tmp = GPIOx->LCKR;
  /* Read LCKK bit*/
  tmp = GPIOx->LCKR;
}
```

GPIO复用功能低位寄存器

GPIOx_AFRL

有两个寄存器可用来从每个 I/O 可用的 16 个复用功能输入/输出中进行选择。借助这些寄存器，可根据应用程序的要求将某个复用功能连接到其它某个引脚。
这意味着可使用 GPIOx_AFRL 和 GPIOx_AFRH 复用功能寄存器在每个 GPIO 上复用多个可用的外设功能。
这样一来，应用程序可为每个 I/O 选择任何一个可用功能。由于 AF 选择信号由复用功能输入和复用功能输出共用，所以只需为每个 I/O 的复用功能输入/输出选择一个通道即可。