嵌入式系统 课程号：4240077

4. 库函数编程模式入门

- 以GPIO为例

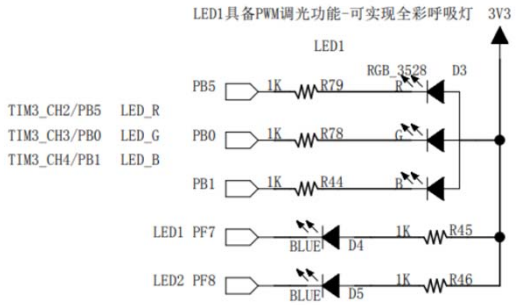
胡四泉

北京科技大学计算机与通信工程学院

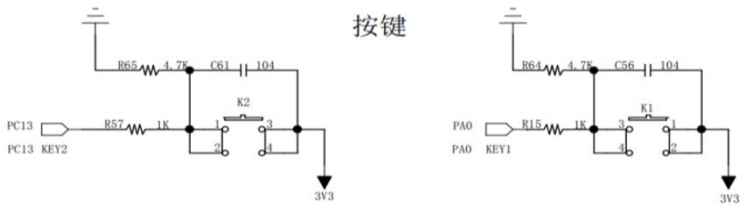
1

GPIO原理图

LED



按键



2

提纲

- 1. 寄存器成组地址封装 - 结构体
- 2. 寄存器内容封装 – 枚举
- 3. 寄存器操作封装&抽象 – 函数
- 4. 固件库工程结构分析

理解固件库实现封装和抽象的逻辑，让大家知其然，也知其所以然。

3

STM32寄存器组封装

使用结构体封装寄存器列表?

代码 6-6 使用结构体对 GPIO 寄存器组的封装

```
typedef unsigned           int uint32_t;  /*无符号 32 位变量*/
typedef unsigned short     int uint16_t;  /*无符号 16 位变量*/

/* GPIO 寄存器列表 */
typedef struct {
    uint32_t CRL;      /*GPIO 端口配置低寄存器     地址偏移: 0x00 */
    uint32_t CRH;      /*GPIO 端口配置高寄存器     地址偏移: 0x04 */
    uint32_t IDR;      /*GPIO 数据输入寄存器       地址偏移: 0x08 */
    uint32_t ODR;      /*GPIO 数据输出寄存器       地址偏移: 0x0C */
    uint32_t BSRR;     /*GPIO 位设置/清除寄存器    地址偏移: 0x10 */
    uint32_t BRR;      /*GPIO 端口位清除寄存器      地址偏移: 0x14 */
    uint16_t LCKR;     /*GPIO 端口配置锁定寄存器    地址偏移: 0x18 */
} GPIO_TypeDef;
```

4

STM32寄存器映射

定义GPIO端口基地址指针

代码 6-8 定义好 GPIO 端口首地址址针

```
/*使用 GPIO_TypeDef 把地址强制转换成指针*/
#define GPIOA                   ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB                   ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC                   ((GPIO_TypeDef *) GPIOC_BASE)
#define GPIOD                   ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOE                   ((GPIO_TypeDef *) GPIOE_BASE)
#define GPIOF                   ((GPIO_TypeDef *) GPIOF_BASE)
#define GPIOG                   ((GPIO_TypeDef *) GPIOG_BASE)
#define GPIOH                   ((GPIO_TypeDef *) GPIOH_BASE)


/*使用定义好的宏直接访问*/
/*访问 GPIOB 端口的寄存器*/
GPIOB->BSRR = 0xFFFF;           //通过指针访问并修改 GPIOB_BSRR 寄存器
GPIOB->CRL = 0xFFFF;            //修改 GPIOB_CRL 寄存器
GPIOB->ODR =0xFFFF;             //修改 GPIOB_ODR 寄存器

uint32_t temp;
temp = GPIOB->IDR;              //读取 GPIOB_IDR 寄存器的值到变量 temp 中

/*访问 GPIOA 端口的寄存器*/
GPIOA->BSRR = 0xFFFF;
GPIOA->CRL = 0xFFFF;
GPIOA->ODR =0xFFFF;

uint32_t temp;
temp = GPIOA->IDR;              //读取 GPIOA_IDR 寄存器的值到变量 temp 中
```

5

STM32寄存器组映射

使用结构体指针访问寄存器

代码 6-7 通过结构体指针访问寄存器

```
GPIO_TypeDef * GPIOx;           //定义一个 GPIO_TypeDef 型结构体指针 GPIOx
GPIOx = GPIOB_BASE;             //把指针地址设置为宏 GPIOH_BASE 地址
GPIOx->IDR = 0xFFFF;
GPIOx->ODR = 0xFFFF;


uint32_t temp;
temp = GPIOx->IDR;              //读取 GPIOB_IDR 寄存器的值到变量 temp 中
```

6

STM32寄存器封装定义文件 stm32f10x.h

stm32f10x.h

```
997   /**
998     * @brief General Purpose I/O
999     */
1000
1001  typedef struct
1002  {
1003    __IO uint32_t CRL;
1004    __IO uint32_t CRH;
1005    __IO uint32_t IDR;
1006    __IO uint32_t ODR;
1007    __IO uint32_t BSRR;
1008    __IO uint32_t BRR;
1009    __IO uint32_t LCKR;
1010  } GPIO_TypeDef;
1011
```

stm32f10x.h

```
1313  #define AFIO_BASE             (APB2PERIPH_BASE + 0x0000)
1314  #define EXTI_BASE             (APB2PERIPH_BASE + 0x0400)
1315  #define GPIOA_BASE            (APB2PERIPH_BASE + 0x0800)
1316  #define GPIOB_BASE            (APB2PERIPH_BASE + 0x0C00)
1317  #define GPIOC_BASE            (APB2PERIPH_BASE + 0x1000)
1318  #define GPIOD_BASE            (APB2PERIPH_BASE + 0x1400)
1319  #define GPIOE_BASE            (APB2PERIPH_BASE + 0x1800)
1320  #define GPIOF_BASE            (APB2PERIPH_BASE + 0x1C00)
1321  #define GPIOG_BASE            (APB2PERIPH_BASE + 0x2000)
1322  #define ADC1_BASE             (APB2PERIPH_BASE + 0x2400)
1323  #define ADC2_BASE             (APB2PERIPH_BASE + 0x2800)
1324  #define TIM1_BASE             (APB2PERIPH_BASE + 0x2C00)
1325  #define SPI1_BASE             (APB2PERIPH_BASE + 0x3000)
1326  #define TIM8_BASE             (APB2PERIPH_BASE + 0x3400)
1327  #define USART1_BASE           (APB2PERIPH_BASE + 0x3800)
1328  #define ADC3_BASE             (APB2PERIPH_BASE + 0x3C00)
```

stm32f10x.h

```
1408  #define GPIOA                 ((GPIO_TypeDef *) GPIOA_BASE)
1409  #define GPIOB                 ((GPIO_TypeDef *) GPIOB_BASE)
1410  #define GPIOC                 ((GPIO_TypeDef *) GPIOC_BASE)
1411  #define GPIOD                 ((GPIO_TypeDef *) GPIOD_BASE)
1412  #define GPIOE                 ((GPIO_TypeDef *) GPIOE_BASE)
1413  #define GPIOF                 ((GPIO_TypeDef *) GPIOF_BASE)
1414  #define GPIOG                 ((GPIO_TypeDef *) GPIOG_BASE)
1415  #define ADC1                  ((ADC_TypeDef *) ADC1_BASE)
1416  #define ADC2                  ((ADC_TypeDef *) ADC2_BASE)
1417  #define TIM1                  ((TIM_TypeDef *) TIM1_BASE)
1418  #define SPI1                  ((SPI_TypeDef *) SPI1_BASE)
1419  #define TIM8                  ((TIM_TypeDef *) TIM8_BASE)
1420  #define USART1                ((USART_TypeDef *) USART1_BASE)
1421  #define ADC3                  ((ADC_TypeDef *) ADC3_BASE)
```

7

STM32寄存器封装定义文件 stm32f10x.h

```
// 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)

typedef unsigned int      uint32_t;
typedef unsigned short    uint16_t;

typedef struct {
          uint32_t CRL;
          …
}GPIO_TypeDef;

typedef struct{
     …
     uint32_t AHBENR;
     uint32_t APB2ENR;
     …
}RCC_TypeDef;

#define GPIOB   ((GPIO_TypeDef*)GPIOB_BASE)
#define RCC     ((RCC_TypeDef*)RCC_BASE)
```

8

演示实验： 点亮LED－使用结构体访问寄存器

```
//main.c

#include “stm32f10x.h”
// stm32f10x.h包含了下面的RCC及GPIOB的地址定义和结构体定义

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 点亮或熄灭LED
        GPIOB->ODR &= ~(1<<0);
        //GPIOB->ODR |= (1<<0);
}
```

9

提纲

- 1. 寄存器成组地址封装 - 结构体
- 2. 寄存器内容封装 – 枚举
- 3. 寄存器操作封装&抽象 – 函数
- 4. 固件库工程结构分析

理解固件库实现封装和抽象的逻辑，让大家知其然，也知其所以然。

10

如何把寄存器位结构的细节隐藏起来

思路1：定义位掩码(Bit mask)指代pin

```
// stm32f10x_gpio.h
#ifndef __STM32F10X_GPIO_H
#define __STM32F10X_GPIO_H

#include "stm32f10x.h"

#define GPIO_Pin_0   ((uint16_t)0x0001)  /*!< 选择Pin0 */   //(00000000 00000001)b
#define GPIO_Pin_1   ((uint16_t)0x0002)  /*!< 选择Pin1 */   //(00000000 00000010)b
#define GPIO_Pin_2   ((uint16_t)0x0004)  /*!< 选择Pin2 */   //(00000000 00000100)b
#define GPIO_Pin_3   ((uint16_t)0x0008)  /*!< 选择Pin3 */   //(00000000 00001000)b
#define GPIO_Pin_4   ((uint16_t)0x0010)  /*!< 选择Pin4 */   //(00000000 00010000)b
#define GPIO_Pin_5   ((uint16_t)0x0020)  /*!< 选择Pin5 */   //(00000000 00100000)b
#define GPIO_Pin_6   ((uint16_t)0x0040)  /*!< 选择Pin6 */   //(00000000 01000000)b
#define GPIO_Pin_7   ((uint16_t)0x0080)  /*!< 选择Pin7 */   //(00000000 10000000)b
#define GPIO_Pin_8   ((uint16_t)0x0100)  /*!< 选择Pin8 */   //(00000001 00000000)b
#define GPIO_Pin_9   ((uint16_t)0x0200)  /*!< 选择Pin9 */   //(00000010 00000000)b
#define GPIO_Pin_10  ((uint16_t)0x0400)  /*!< 选择Pin10 */  //(00000100 00000000)b
#define GPIO_Pin_11  ((uint16_t)0x0800)  /*!< 选择Pin11 */  //(00001000 00000000)b
#define GPIO_Pin_12  ((uint16_t)0x1000)  /*!< 选择Pin12 */  //(00010000 00000000)b
#define GPIO_Pin_13  ((uint16_t)0x2000)  /*!< 选择Pin13 */  //(00100000 00000000)b
#define GPIO_Pin_14  ((uint16_t)0x4000)  /*!< 选择Pin14 */  //(01000000 00000000)b
#define GPIO_Pin_15  ((uint16_t)0x8000)  /*!< 选择Pin15 */  //(10000000 00000000)b
#define GPIO_Pin_All ((uint16_t)0xFFFF)  /*!< 选择全部引脚*/ //(11111111 11111111)b

void GPIO_SetBits(GPIO_TypeDef *GPIOx,uint16_t GPIO_Pin);
void GPIO_ResetBits( GPIO_TypeDef *GPIOx,uint16_t GPIO_Pin );

#endif /*    STM32F10X_GPIO_H */
```

11

如何把寄存器位结构的细节隐藏起来

```
// stm32f10x_gpio.h
//将PB端口中的某一个引脚GPIO_Pin置位
void GPIO_SetBits(GPIO_TypeDef *GPIOx,uint16_t GPIO_Pin) {
        GPIOx->BSRR |= GPIO_Pin;
}
//将PB端口中的某一个引脚GPIO_Pin清0
void GPIO_ResetBits( GPIO_TypeDef *GPIOx,uint16_t GPIO_Pin ) {
        GPIOx->BRR |= GPIO_Pin;
}

// main.c
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"

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 某个pin 输出低(高)电平
        GPIO_ResetBits( GPIOB,GPIO_Pin_0 );  //输出低电平
        //GPIO_SetBits(GPIOB,GPIO_Pin_0); //输出高电平

}
```

```
// 控制 ODR 寄存器 点亮或熄灭LED
GPIOB->ODR &= ~(1<<0);
//GPIOB->ODR |= (1<<0);
```

减少了神秘代码

12

定义初始化结构体 GPIO_InitTypeDef

思路2：用C语言中的枚举定义好寄存器或一部分的取值

```
typedef struct
{
        uint16_t GPIO_Pin; /*!< 选择要配置的 GPIO 引脚 */
        uint16_t GPIO_Speed;/*!< 选择 GPIO 引脚的速率 */
        uint16_t GPIO_Mode;/*!< 选择 GPIO 引脚的工作模式 */
} GPIO_InitTypeDef;
```

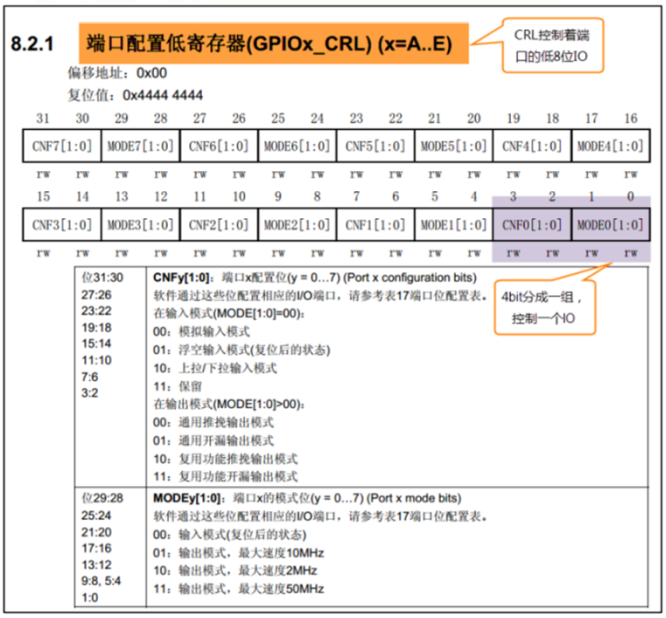
设计这个结构体用于GPIO 初始化的思路是：

1）先定义一个这样的结构体变量，根据需要配置 GPIO 的模式，对这个结构体的各个成员进行赋值

2）然后把这个变量作为“GPIO 初始化函数”的输入参数，该函数能根据这个变量值中的内容去配置寄存器，从而实现 GPIO 的初始化。

13

定义引脚模式的枚举类型



8.2.1 端口配置低寄存器(GPIOx_CRL) (x=A..E)

CRL控制着端口的低8位IO

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

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

4bit分成一组，控制一个IO

14

定义引脚模式的枚举类型

8.2.2 端口配置高寄存器(GPIOx_CRH) (x=A..E)

CRH控制着端口的高8位IO

偏移地址：0x04

复位值：0x4444 4444

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

4bit分成一组，控制一个IO

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) (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) (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

15

定义引脚模式的枚举类型

不希望每次用到的时候都要去查询手册，可以使用 C语言中的枚举定义功能，根据手册把每个成员的所有取值都定义好

思路2：用C语言中的枚举定义好寄存器或一部分的取值

```
// GPIO输出速率枚举定义
typedef enum {
  GPIO_Speed_10MHz = 1,          // 10MHZ        (01)b
  GPIO_Speed_2MHz,               // 2MHZ         (10)b
  GPIO_Speed_50MHz               // 50MHZ        (11)b
}GPIOSpeed_TypeDef;
//GPIO工作模式枚举定义
typedef enum{
  GPIO_Mode_AIN = 0x0,           // 模拟输入      (0000 0000)b
  GPIO_Mode_IN_FLOATING = 0x04,  // 浮空输入      (0000 0100)b
  GPIO_Mode_IPD = 0x28,          // 下拉输入      (0010 1000)b
  GPIO_Mode_IPU = 0x48,          // 上拉输入      (0100 1000)b

  GPIO_Mode_Out_OD = 0x14,       // 开漏输出      (0001 0100)b
  GPIO_Mode_Out_PP = 0x10,       // 推挽输出      (0001 0000)b
  GPIO_Mode_AF_OD = 0x1C,        // 复用开漏输出  (0001 1100)b
  GPIO_Mode_AF_PP = 0x18         // 复用推挽输出  (0001 1000)b
}GPIOMode_TypeDef;
```

16

提纲

- 1. 寄存器(组)地址封装 - 结构体
- 2. 寄存器内容封装 – 枚举
- 3. 寄存器操作封装&抽象 – 函数
- 4. 固件库工程结构分析

理解固件库实现封装和抽象的逻辑，让大家知其然，也知其所以然。

17

GPIO 端口初始化

```
//main.c
int main (void){
        GPIO_InitTypeDef  GPIO_InitStructure;

        // 打开 GPIOB 端口的时钟
        RCC->APB2ENR  |=  ( (1) << 3 );
        /* GPIO 端口初始化 */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOB, &GPIO_InitStructure);

        GPIO_ResetBits( GPIOB,GPIO_Pin_0 );
        //GPIO_SetBits(GPIOB,GPIO_Pin_0);
        }
```

```
// 配置IO口为输出
GPIOB->CRL &=  ~( (0x0f) << (4*0) );
GPIOB->CRL |=  ( (1) << (4*0) );
```

减少了神秘代码

18

GPIO 初始化函数

```
13  void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
14  {
15    uint32_t currentmode = 0x00, currentpin = 0x00, pinpos = 0x00, pos = 0x00;
16    uint32_t tmpreg = 0x00, pinmask = 0x00;
17
18  /*---------------------- GPIO 模式配置 --------------------------*/
19    // 把输入参数GPIO_Mode的低四位暂存在currentmode
20    currentmode = ((uint32_t)GPIO_InitStruct->GPIO_Mode) & ((uint32_t)0x0F);
21
22    // bit4是1表示输出，bit4是0则是输入
23    // 判断bit4是1还是0，即首选判断是输入还是输出模式
24    if ((((uint32_t)GPIO_InitStruct->GPIO_Mode) & ((uint32_t)0x10)) != 0x00)
25    {
26    // 输出模式则要设置输出速度
27      currentmode |= (uint32_t)GPIO_InitStruct->GPIO_Speed;
28    }
29  /*------------GPIO CRL 寄存器配置 CRL寄存器控制着低8位IO- -------*/
30    // 配置端口低8位，即Pin0~Pin7
31    if (((uint32_t)GPIO_InitStruct->GPIO_Pin & ((uint32_t)0x00FF)) != 0x00)
32    {
33    // 先备份CRL寄存器的值
34      tmpreg = GPIOx->CRL;
35
36    // 循环，从Pin0开始配对，找出具体的Pin
37      for (pinpos = 0x00; pinpos < 0x08; pinpos++)
38      {
39     // pos的值为1左移pinpos位
40        pos = ((uint32_t)0x01) << pinpos;
```

19

GPIO 初始化函数

```
41
42      // 令pos与输入参数GPIO_PIN作位与运算，为下面的判断作准备
43        currentpin = (GPIO_InitStruct->GPIO_Pin) & pos;
44
45      //若currentpin=pos,则找到使用的引脚
46        if (currentpin == pos)
47        {
48      // pinpos的值左移两位（乘以4），因为寄存器中4个寄存器位配置一个引脚
49          pos = pinpos << 2;
50          //把控制这个引脚的4个寄存器位清零，其它寄存器位不变
51          pinmask = ((uint32_t)0x0F) << pos;
52          tmpreg &= ~pinmask;
53
54          // 向寄存器写入将要配置的引脚的模式
55          tmpreg |= (currentmode << pos);
56
57      // 判断是否为下拉输入模式
58          if (GPIO_InitStruct->GPIO_Mode == GPIO_Mode_IPD)
59          {
60        // 下拉输入模式，引脚默认置0，对BRR寄存器写1可对引脚置0
61            GPIOx->BRR = (((uint32_t)0x01) << pinpos);
62          }
63          else
64          {
65            // 判断是否为上拉输入模式
66            if (GPIO_InitStruct->GPIO_Mode == GPIO_Mode_IPU)
67            {
68          // 上拉输入模式，引脚默认值为1，对BSRR寄存器写1可对引脚置1
69              GPIOx->BSRR = (((uint32_t)0x01) << pinpos);
70            }
71          }
72        }
73      }
74      // 把前面处理后的暂存值写入到CRL寄存器之中
75      GPIOx->CRL = tmpreg;
76    }
```

20

GPIO 初始化函数

```
/*----GPIO CRH 寄存器配置 CRH寄存器控制着高8位IO- ----*/
 // 配置端口高8位，即Pin8~Pin15
 if (GPIO_InitStruct->GPIO_Pin > 0x00FF)
 {
  // // 先备份CRH寄存器的值
  tmpreg = GPIOx->CRH;

 // 循环，从Pin8开始配对，找出具体的Pin
  for (pinpos = 0x00; pinpos < 0x08; pinpos++)
  {
   pos = (((uint32_t)0x01) << (pinpos + 0x08));

   // pos与输入参数GPIO_PIN作位与运算
   currentpin = ((GPIO_InitStruct->GPIO_Pin) & pos);

   //若currentpin=pos,则找到使用的引脚
   if (currentpin == pos)
   {
    //pinpos的值左移两位（乘以4），因为寄存器中4个寄存器位配置一个引脚
     pos = pinpos << 2;

    //把控制这个引脚的4个寄存器位清零，其它寄存器位不变
     pinmask = ((uint32_t)0x0F) << pos;
     tmpreg &= ~pinmask;

    // 向寄存器写入将要配置的引脚的模式
     tmpreg |= (currentmode << pos);

    // 判断是否为下拉输入模式
    if (GPIO_InitStruct->GPIO_Mode ==
GPIO_Mode_IPD) {
                    // 下拉输入模式，引脚默认置0，对BRR寄存器写1可对引脚置0
      GPIOx->BRR = (((uint32_t)0x01) << (pinpos +
0x08));
    }
     // 判断是否为上拉输入模式
    if (GPIO_InitStruct->GPIO_Mode ==
GPIO_Mode_IPU)
    {
    // 上拉输入模式，引脚默认值为1，对BSRR寄存器写1可对引脚置1
      GPIOx->BSRR = (((uint32_t)0x01) << (pinpos +
0x08));
    }
   }
  }
  // 把前面处理后的暂存值写入到CRH寄存器之中
  GPIOx->CRH = tmpreg;
 }
```

21

我们需要理解内部逻辑吗？

GPIO 引脚工作模式真值表

STM32F103 GPIO 引脚工作模式真值表分析

			十六进制	二进制 bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
					上/下拉		输入/出	具体工作模式对照寄存器说明			
GPIOMode_TypeDef	GPIO_Mode_AIN	模拟输入	0X00	0	0	0	0	0	0	0	0
	GPIO_Mode_IN_FLOATING	浮空输入	0X04	0	0	0	0	0	1	0	0
	GPIO_Mode_IPD	下拉输入	0X28	0	0	1	0	0	0	0	0
	GPIO_Mode_IPU	上拉输入	0X48	0	1	0	0	0	0	0	0
	GPIO_Mode_Out_OD	开漏输出	0X14	0	0	0	1	0	1	0	0
	GPIO_Mode_Out_PP	推挽输出	0X10	0	0	0	1	0	0	0	0
	GPIO_Mode_AF_OD	复用开漏输出	0X1C	0	0	0	1	1	1	0	0
	GPIO_Mode_AF_PP	复用推挽输出	0X18	0	0	0	1	1	0	0	0

这8个宏的高4bit可随意设置，只要能在程序上帮助判断出模式即可，真正写到寄存器的值是bit2和bit3

```
// GPIO输出速率枚举定义
typedef enum {
  GPIO_Speed_10MHz = 1,    // 10MHZ      (01)b
  GPIO_Speed_2MHz,         // 2MHZ       (10)b
  GPIO_Speed_50MHz         // 50MHZ      (11)b
}GPIOSpeed_TypeDef;
```

```
//GPIO工作模式枚举定义
typedef enum{
  GPIO_Mode_AIN = 0x0,              // 模拟输入       (0000 0000)b
  GPIO_Mode_IN_FLOATING = 0x04,     // 浮空输入       (0000 0100)b
  GPIO_Mode_IPD = 0x28,             // 下拉输入       (0010 1000)b
  GPIO_Mode_IPU = 0x48,             // 上拉输入       (0100 1000)b

  GPIO_Mode_Out_OD = 0x14,          // 开漏输出       (0001 0100)b
  GPIO_Mode_Out_PP = 0x10,          // 推挽输出       (0001 0000)b
  GPIO_Mode_AF_OD = 0x1C,           // 复用开漏输出   (0001 1100)b
  GPIO_Mode_AF_PP = 0x18            // 复用推挽输出   (0001 1000)b
}GPIOMode_TypeDef;
```

22

我们需要理解内部逻辑吗？

1） 先取得 GPIO_Mode 的值，判断 bit4 是 1 还是 0 来判断是输出还是输入。如果是输出则设置输出速率，即加上 GPIO_Speed 的值，输入没有速率之说，不用设置。

2） 配置 CRL寄存器。通过 GPIO_Pin的值计算出具体需要初始化哪个引脚，算出后，然后把需要配置的值写入到 CRL 寄存器中，具体分析见代码注释。这里有一个比较有趣的是上/下拉输入并不是直接通过配置某一个寄存器来实现的，而是通过写BSRR 或者 BRR 寄存器来实现。这让很多只看手册没看固件库底层源码的人摸不着头脑，因为手册的寄存器说明中没有明确的指出如何配置上拉/下拉，具体见图 9-8。

3） 配置 CRH 寄存器过程同 CRL。

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

如何区分上拉或者下拉？？？

图 9-8 上拉/下拉寄存器说明

23

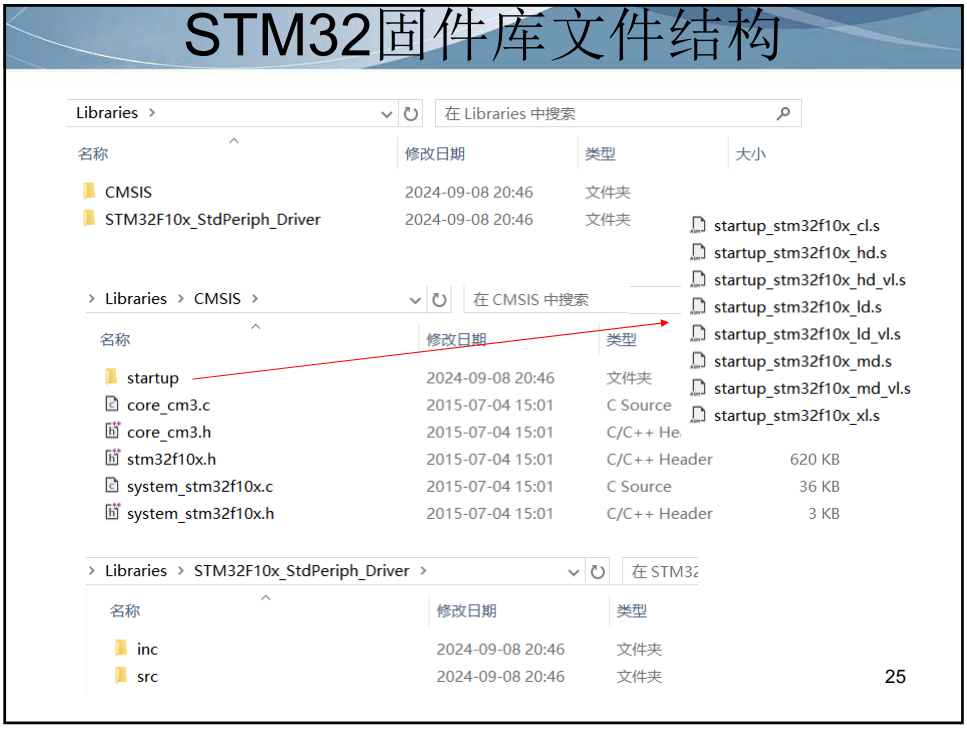
提纲

- 1. 寄存器成组地址封装 - 结构体
- 2. 寄存器内容封装 – 枚举
- 3. 寄存器操作封装&抽象 – 函数
- 4. 固件库工程结构分析

理解固件库实现封装和抽象的逻辑，让大家知其然，也知其所以然。

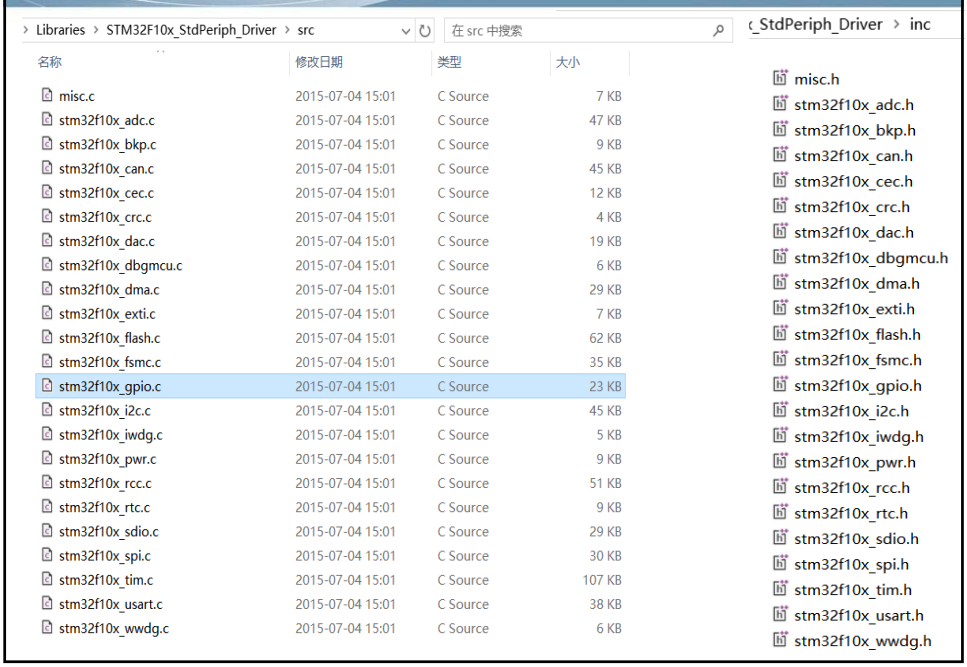
24

STM32固件库文件结构



25

STM32固件库文件结构



STM32固件库文件结构

1-汇编编写的启动文件
startup_stm32f10x_hd.s:设置堆栈指针、设置PC指针、初始化中断向量表、配置系统时钟、对用C库函数_main最终去到C的世界

2-时钟配置文件
system_stm32f10x.c：把外部时钟HSE=8M，经过PLL倍频为72M。

3-外设相关的
stm32f10x.h：实现了内核之外的外设的寄存器映射
xxx：GPIO、USRAT、I2C、SPI、FSMC
stm32f10x_xx.c：外设的驱动函数库文件
stm32f10x_xx.h：存放外设的初始化结构体，外设初始化结构体成员的参数列表，外设固件库函数的声明

4-内核相关的
CMSIS - Cortex 微控制器软件接口标准
core_cm3.h：实现了内核里面外设的寄存器映射
core_cm3.c：内核外设的驱动固件库

NVIC(嵌套向量中断控制器)、SysTick(系统滴答定时器)
misc.h
misc.c

27

STM32固件库文件结构

5-头文件的配置文件
stm32f10x_conf.h：头文件的头文件
//stm32f10x_usart.h
//stm32f10x_i2c.h
//stm32f10x_spi.h
//stm32f10x_adc.h
//stm32f10x_fsmc.h
.....

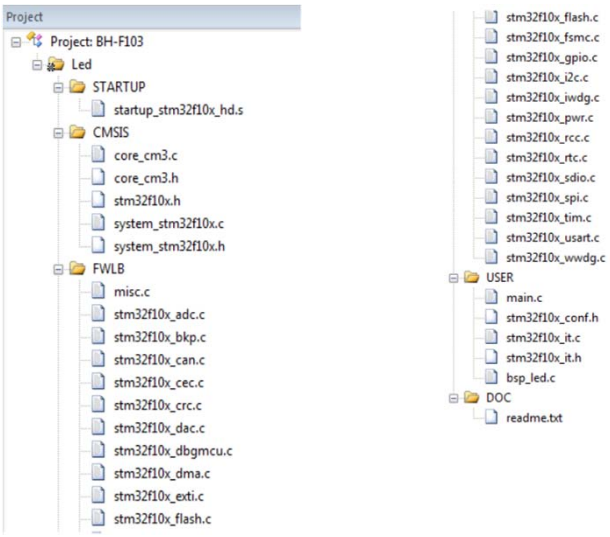
6-专门存放中断服务函数的C文件
stm32f10x_it.c
stm32f10x_it.h

中断服务函数你可以随意放在其他的地方，并不是一定要放在stm32f10x_it.c

```
#include "stm32f10x.h"   // 相当于51单片机中的  #include <reg51.h>

int main(void)
{
        // 来到这里的时候，系统的时钟已经被配置成72M。
}
```

28

固件库编程的工程结构示例



29

使用固件库点亮LED

```
//main.c
#include "stm32f10x.h"
#include "bsp_led.h"
#define SOFT_DELAY Delay(0x0FFFFF);
void Delay(__IO u32 nCount);
int main(void){
        LED_GPIO_Config();  /* LED 端口初始化 */

        while (1)       {
                LED1_ON;                        // 亮
                SOFT_DELAY;
                LED1_OFF;                       // 灭

                LED2_ON;                        // 亮
                SOFT_DELAY;
                LED2_OFF;                       // 灭

                LED3_ON;                       // 亮
                SOFT_DELAY;
                LED3_OFF;                       // 灭
        }
}
void Delay(__IO uint32_t nCount)          //简单的延时函数
{       for(; nCount != 0; nCount--);}
```

30

板级支持包BSP编写

```
#ifndef __LED_H
#define        __LED_H
#include "stm32f10x.h"

/* 定义LED连接的GPIO端口, 用户只需要修改下面的代码即可改变控制的LED引脚 */
// R-红色
#define LED1_GPIO_PORT              GPIOB                       /* GPIO端口 */
#define LED1_GPIO_CLK     RCC_APB2Periph_GPIOB          /* GPIO端口时钟 */
#define LED1_GPIO_PIN               GPIO_Pin_5                  /* 连接到SCL时钟线的GPIO */

// G-绿色
#define LED2_GPIO_PORT              GPIOB                       /* GPIO端口 */
#define LED2_GPIO_CLK     RCC_APB2Periph_GPIOB          /* GPIO端口时钟 */
#define LED2_GPIO_PIN               GPIO_Pin_0                  /* 连接到SCL时钟线的GPIO */

// B-蓝色
#define LED3_GPIO_PORT              GPIOB                       /* GPIO端口 */
#define LED3_GPIO_CLK     RCC_APB2Periph_GPIOB          /* GPIO端口时钟 */
#define LED3_GPIO_PIN               GPIO_Pin_1                  /* 连接到SCL时钟线的GPIO */
```

31

板级支持包BSP编写

```
//bsp.h
/** the macro definition to trigger the led on or off   */
#define ON  0
#define OFF 1

/* 使用标准的固件库控制IO*/
#define LED1(a)          if (a)      GPIO_SetBits(LED1_GPIO_PORT,LED1_GPIO_PIN);\
                         else        GPIO_ResetBits(LED1_GPIO_PORT,LED1_GPIO_PIN)
#define LED2(a)          if (a)      GPIO_SetBits(LED2_GPIO_PORT,LED2_GPIO_PIN);\
                         else        GPIO_ResetBits(LED2_GPIO_PORT,LED2_GPIO_PIN)
#define LED3(a)          if (a)      GPIO_SetBits(LED3_GPIO_PORT,LED3_GPIO_PIN);\
                         else        GPIO_ResetBits(LED3_GPIO_PORT,LED3_GPIO_PIN)

/* 定义控制IO的宏 */
#define LED1_OFF                     LED1(ON)
#define LED1_ON                      LED1(OFF)

#define LED2_OFF                    LED1(ON)
#define LED2_ON                     LED1(OFF)

#define LED3_OFF                    LED1(ON)
#define LED3_ON                     LED1(OFF)

void LED_GPIO_Config(void);

#endif /* __LED_H */
```

32

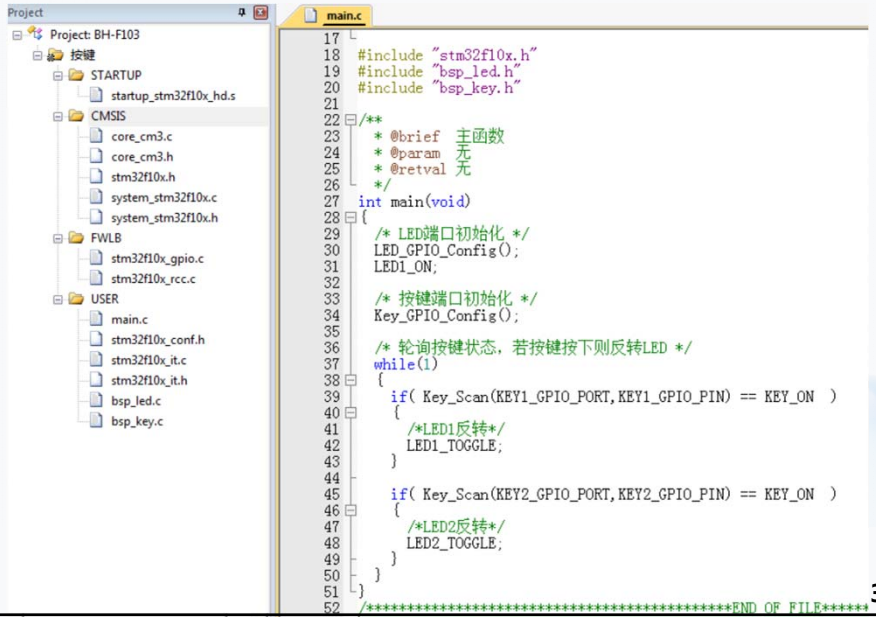
板级支持包BSP编写

```
//bsp_led.c
#include "bsp_led.h"
void LED_GPIO_Config(void){
                        GPIO_InitTypeDef GPIO_InitStructure; /*定义一个GPIO_InitTypeDef类型的结构体*/

                        /*开启LED相关的GPIO外设时钟*/
                        RCC_APB2PeriphClockCmd( LED1_GPIO_CLK | LED2_GPIO_CLK | LED3_GPIO_CLK, ENABLE);
                        /*设置引脚速率为50MHz */
                        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

                        /*选择要控制的GPIO引脚*/
                        GPIO_InitStructure.GPIO_Pin = LED1_GPIO_PIN;
                        /*设置引脚模式为通用推挽输出*/
                        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
                        /*调用库函数，初始化GPIO*/
                        GPIO_Init(LED1_GPIO_PORT, &GPIO_InitStructure);

                        GPIO_InitStructure.GPIO_Pin = LED2_GPIO_PIN;/*选择要控制的GPIO引脚*/
                        GPIO_Init(LED2_GPIO_PORT, &GPIO_InitStructure);/*调用库函数，初始化GPIO*/

                        GPIO_InitStructure.GPIO_Pin = LED3_GPIO_PIN;/*选择要控制的GPIO引脚*/
                        GPIO_Init(LED3_GPIO_PORT, &GPIO_InitStructure);/*调用库函数，初始化GPIOF*/

                        /* 关闭所有led灯              */
                        GPIO_SetBits(LED1_GPIO_PORT, LED1_GPIO_PIN);
                        GPIO_SetBits(LED2_GPIO_PORT, LED2_GPIO_PIN);
                        GPIO_SetBits(LED3_GPIO_PORT, LED3_GPIO_PIN);
}
```

33

GPIO输入固件库编程示例-按键

按键检测,KEY1按下控制绿灯闪烁



34

GPIO输入固件库编程示例-按键

bsp_led.h

```
/* 直接操作寄存器的方法控制IO */
#define digitalHi(p,i)      {p->BSRR=i;}  //输出为高电平
#define digitalLo(p,i)      {p->BRR=i;}   //输出低电平
#define digitalToggle(p,i)  {p->ODR ^=i;} //输出反转状态

                              // ^ 异或，C语言的一个二进制的运算符
                              // 与1异或改变，与0异或不变

/* 定义控制IO的宏 */
#define LED1_TOGGLE     digitalToggle(LED1_GPIO_PORT,LED1_GPIO_PIN)
#define LED1_OFF        digitalHi(LED1_GPIO_PORT,LED1_GPIO_PIN)
#define LED1_ON         digitalLo(LED1_GPIO_PORT,LED1_GPIO_PIN)

#define LED2_TOGGLE     digitalToggle(LED2_GPIO_PORT,LED2_GPIO_PIN)
#define LED2_OFF        digitalHi(LED2_GPIO_PORT,LED2_GPIO_PIN)
#define LED2_ON         digitalLo(LED2_GPIO_PORT,LED2_GPIO_PIN)

#define LED3_TOGGLE     digitalToggle(LED3_GPIO_PORT,LED3_GPIO_PIN)
#define LED3_OFF        digitalHi(LED3_GPIO_PORT,LED3_GPIO_PIN)
#define LED3_ON         digitalLo(LED3_GPIO_PORT,LED3_GPIO_PIN)
```

35

GPIO输入固件库编程示例-按键

bsp_key.c bsp_key.h

```
#ifndef __KEY_H
#define __KEY_H


#include "stm32f10x.h"

//  引脚定义
#define     KEY1_GPIO_CLK     RCC_APB2Periph_GPIOA
#define     KEY1_GPIO_PORT    GPIOA
#define     KEY1_GPIO_PIN     GPIO_Pin_0

#define     KEY2_GPIO_CLK     RCC_APB2Periph_GPIOC
#define     KEY2_GPIO_PORT    GPIOC
#define     KEY2_GPIO_PIN      GPIO_Pin_13


 /** 按键按下标置宏
   *  按键按下为高电平，设置 KEY_ON=1，  KEY_OFF=0
   *  若按键按下为低电平，把宏设置成KEY_ON=0 ，KEY_OFF=1
   */
#define KEY_ON  1
#define KEY_OFF 0

void Key_GPIO_Config(void);
uint8_t Key_Scan(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin);


#endif /* __KEY_H */
```

36

GPIO输入固件库编程示例-按键

```c
#include "./key/bsp_key.h"

/**
  * @brief  配置按键用到的I/O口
  * @param  无
  * @retval 无
  */
void Key_GPIO_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /*开启按键端口的时钟*/
  RCC_APB2PeriphClockCmd(KEY1_GPIO_CLK|KEY2_GPIO_CLK,ENABLE);

  //选择按键的引脚
  GPIO_InitStructure.GPIO_Pin = KEY1_GPIO_PIN;
  // 设置按键的引脚为浮空输入
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  //使用结构体初始化按键
  GPIO_Init(KEY1_GPIO_PORT, &GPIO_InitStructure);

  //选择按键的引脚
  GPIO_InitStructure.GPIO_Pin = KEY2_GPIO_PIN;
  //设置按键的引脚为浮空输入
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  //使用结构体初始化按键
  GPIO_Init(KEY2_GPIO_PORT, &GPIO_InitStructure);
}
```

37

GPIO输入固件库编程示例-按键

```c
/*
 * 函数名：Key_Scan
 * 描述  ：检测是否有按键按下
 * 输入  ：GPIOx：x 可以是 A，B，C，D或者 E
 *         GPIO_Pin：待读取的端口位
 * 输出  ：KEY_OFF(没按下按键)、KEY_ON（按下按键）
 */
uint8_t Key_Scan(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin)
{
  /*检测是否有按键按下 */
  if(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON )
  {
    /*等待按键释放 */
    while(GPIO_ReadInputDataBit(GPIOx,GPIO_Pin) == KEY_ON);
    return  KEY_ON;
  }
  else
    return KEY_OFF;
}
/*********************************************END OF FILE**********
```

38