目录

一、前言

前几天系统地将32单片机学习了一下，学习的视频是bilibili的江科大自化协c8t6的教学，为了方便以后使用，在这里和b站视频联动写下一篇笔记，以便自己查阅资料和调用函数。

二、必要资料

1、C语言类型

C语言数据类型

关键字	位数	表示范围	stdint关键字	ST关键字
char	8	-128 ~ 127	int8_t	s8
unsigned char	8	0 ~ 255	uint8_t	u8
short	16	-32768 ~ 32767	int16_t	s16
unsigned short	16	0 ~ 65535	uint16_t	u16
int	32	-2147483648 ~ 2147483647	int32_t	s32
unsigned int	32	0 ~ 4294967295	uint32_t	u32
long	32	-2147483648 ~ 2147483647		
unsigned long	32	0 ~ 4294967295		
long long	64	-(2^64)/2 ~ (2^64)/2-1	int64_t	
unsigned long long	64	0 ~ (2^64)-1	uint64_t	
float	32	-3.4e38 ~ 3.4e38		
double	64	-1.7e308 ~ 1.7e308		

- int在51单片机中是16位的，在STM32中32位的，如果要用16位的数据要用short来表示
- float和double都是带符号的，没有不带符号的
- 枚举enum的使用，类似于struct结构体，只是赋值且引用是有范围限制的

C语言枚举

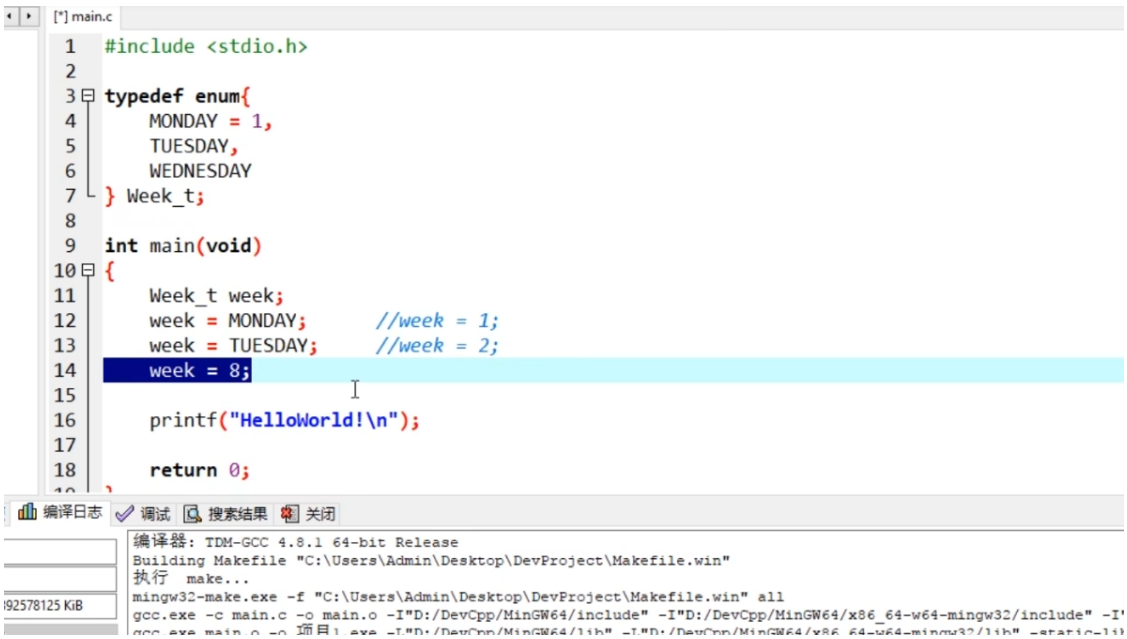
- 关键字：enum
- 用途：定义一个取值受限制的整型变量，用于限制变量取值范围；宏定义的集合
- 定义枚举变量：

```
enum{FALSE = 0, TRUE = 1} EnumName;
```

因为枚举变量类型较长，所以通常用typedef更改变量类型名

- 引用枚举成员：

```
EnumName = FALSE;
EnumName = TRUE;
```

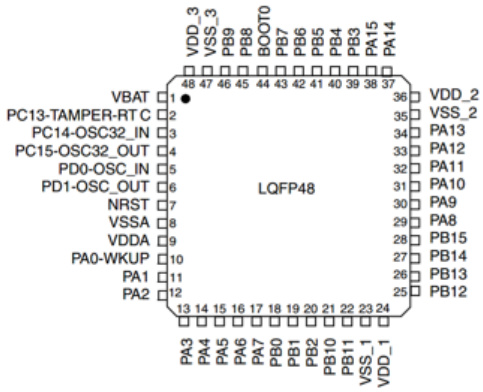


2、片上资源/外设

片上资源/外设

英文缩写	名称	英文缩写	名称
NVIC	嵌套向量中断控制器	CAN	CAN通信
SysTick	系统滴答定时器	USB	USB通信
RCC	复位和时钟控制	RTC	实时时钟
GPIO	通用IO口	CRC	CRC校验
AFIO	复用IO口	PWR	电源控制
EXTI	外部中断	BKP	备份寄存器
TIM	定时器	IWDG	独立看门狗
ADC	模数转换器	WWDG	窗口看门狗
DMA	直接内存访问	DAC	数模转换器
USART	同步/异步串口通信	SDIO	SD卡接口
I2C	I2C通信	FSMC	可变静态存储控制器
SPI	SPI通信	USB OTG	USB主机接口

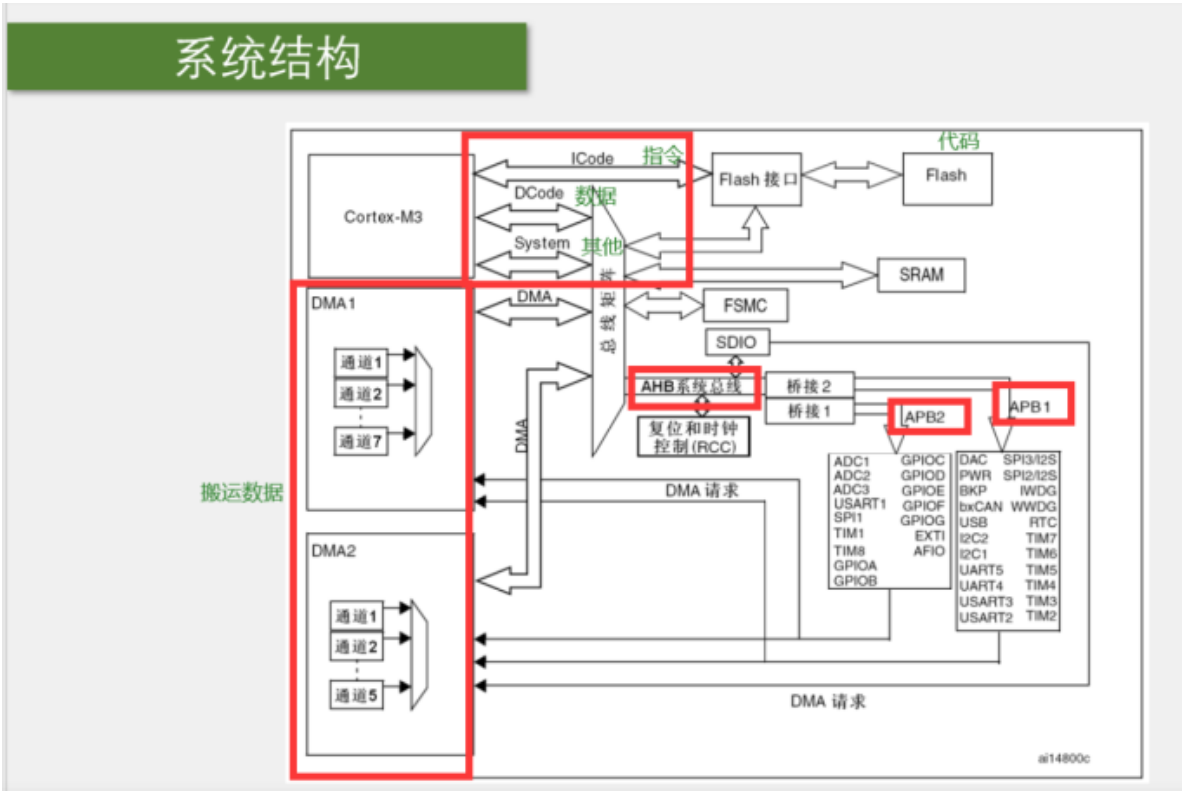
3、引脚定义



STM32F103C8T6
引脚定义

引脚号	引脚名称	类型	I/O口电平	主功能	默认复用功能	重定义功能
1	VBAT	S		VBAT		
2	PC13-TAMPER-RTC	I/O		PC13	TAMPER-RTC	
3	PC14-OSC32_IN	I/O		PC14	OSC32_IN	
4	PC15-OSC32_OUT	I/O		PC15	OSC32_OUT	
5	OSC_IN	I		OSC_IN		
6	OSC_OUT	O		OSC_OUT		
7	NRST	I/O		NRST		
8	VSSA	S		VSSA		
9	VDDA	S		VDDA		
10	**PA0-WKUP**	I/O		PA0	WKUP/USART2_CTS/ADC12_IN0/TIM2_CH1_ETR	
11	**PA1**	I/O		PA1	USART2_RTS/ADC12_IN1/TIM2_CH2	
12	**PA2**	I/O		PA2	USART2_TX/ADC12_IN2/TIM2_CH3	
13	**PA3**	I/O		PA3	USART2_RX/ADC12_IN3/TIM2_CH4	
14	**PA4**	I/O		PA4	SPI1_NSS/USART2_CK/ADC12_IN4	
15	**PA5**	I/O		PA5	SPI1_SCK/ADC12_IN5	
16	**PA6**	I/O		PA6	SPI1_MISO/ADC12_IN6/TIM3_CH1	TIM1_BKIN
17	**PA7**	I/O		PA7	SPI1_MOSI/ADC12_IN7/TIM3_CH2	TIM1_CH1N
18	**PB0**	I/O		PB0	ADC12_IN8/TIM3_CH3	TIM1_CH2N
19	**PB1**	I/O		PB1	ADC12_IN9/TIM3_CH4	TIM1_CH3N
20	PB2	I/O	FT	PB2/BOOT1		
21	**PB10**	I/O	FT	PB10	I2C2_SCL/USART3_TX	TIM2_CH3
22	**PB11**	I/O	FT	PB11	I2C2_SDA/USART3_RX	TIM2_CH4
23	VSS_1	S		VSS_1		
24	VDD_1	S		VDD_1		
25	**PB12**	I/O	FT	PB12	SPI2_NSS/I2C2_SMBAI/USART3_CK/TIM1_BKIN	
26	**PB13**	I/O	FT	PB13	SPI2_SCK/USART3_CTS/TIM1_CH1N	
27	**PB14**	I/O	FT	PB14	SPI2_MISO/USART3_RTS/TIM1_CH2N	
28	**PB15**	I/O	FT	PB15	SPI2_MOSI/TIM1_CH3N	
29	**PA8**	I/O	FT	PA8	USART1_CK/TIM1_CH1/MCO	
30	**PA9**	I/O	FT	PA9	USART1_TX/TIM1_CH2	
31	**PA10**	I/O	FT	PA10	USART1_RX/TIM1_CH3	
32	**PA11**	I/O	FT	PA11	USART1_CTS/USBDM/CAN_RX/TIM1_CH4	
33	**PA12**	I/O	FT	PA12	USART1_RTS/USBDP/CAN_TX/TIM1_ETR	
34	PA13	I/O	FT	JTMS/SWDIO		PA13
35	VSS_2	S		VSS_2		
36	VDD_2	S		VDD_2		
37	PA14	I/O	FT	JTCK/SWCLK		PA14
38	PA15	I/O	FT	JTDI		TIM2_CH1_ETR/PA15/SPI1_NSS
39	PB3	I/O	FT	JTDO		PB3/TRACESWO/TIM2_CH2/SPI1_SCK
40	PB4	I/O	FT	NJTRST		PB4/TIM3_CH1/SPI1_MISO
41	**PB5**	I/O		PB5	I2C1_SMBAI	TIM3_CH2/SPI1_MOSI
42	**PB6**	I/O	FT	PB6	I2C1_SCL/TIM4_CH1	USART1_TX
43	**PB7**	I/O	FT	PB7	I2C1_SDA/TIM4_CH2	USART1_RX
44	BOOT0	I		BOOT0		
45	**PB8**	I/O	FT	PB8	TIM4_CH3	I2C1_SCL/CAN_RX
46	**PB9**	I/O	FT	PB9	TIM4_CH4	I2C1_SDA/CAN_TX
47	VSS_3	S		VSS_3		
48	VDD_3	S		VDD_3		

4、系统结构



三、GPIO初始化

1、首先使用RCC开启GPIO的时钟

从APB2总线，引出RCC_APB2PeriphClockCmd（GPIO口的名字，状态），如使GPIOA 使能：

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
```

2、其次使用GPIO_Init函数初始化GPIO

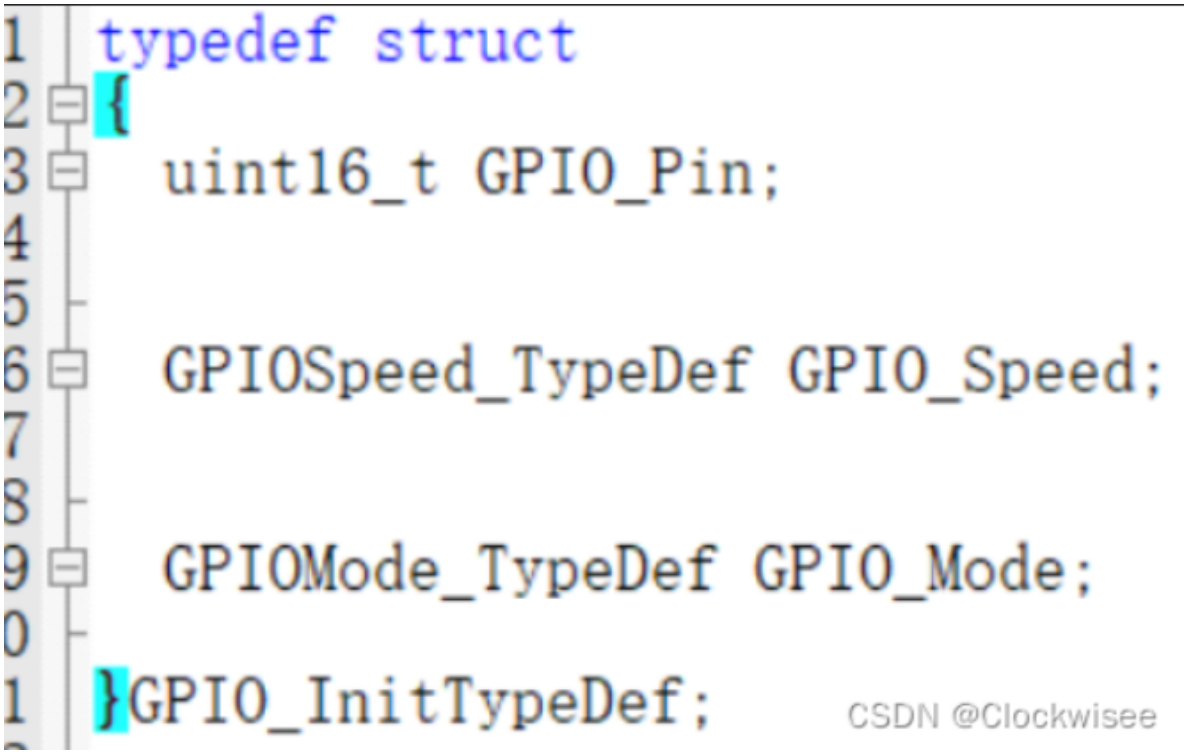
```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
```

第一个参数 GPIO_TypeDef* GPIOx 通过x选定GPIO， 如：GPIOA
第二个参数 GPIO_InitTypeDef* GPIO_InitStruct是GPIO的结构体地址 如：&GPIO_InitStructure

struct 关键字，定义结构体变量
用途：数据打包，不同类型的变量打包(可以数组，数组是同一个类型打包)

#define 新名字 旧名字（无脑式定义）
typedef 旧名字 新名字（定义 长变量类型名）；
因为结构体变量类型较长，所以通常用typedef更改变量类型名

比如：



将结构体变量 struct{...} 换一个别名叫GPIO_InitTypeDef

我们把GPIO_InitTypeDef取个名字叫GPIO_InitStructure放到void GPIO_Init（）上面并且在void GPIO_Init（）中取地址&GPIO_InitStructure

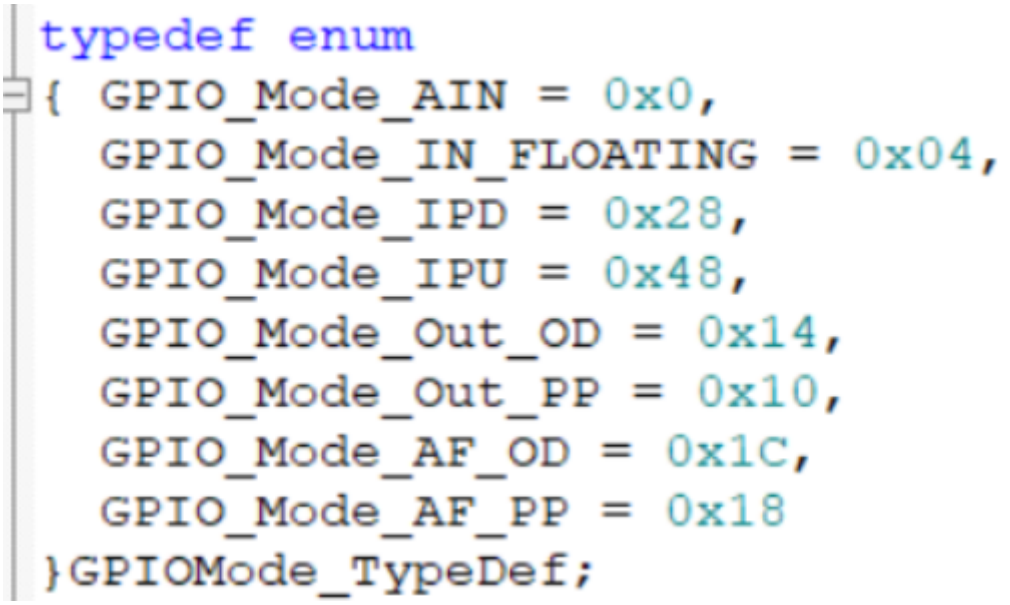
```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
GPIO_InitTypeDef GPIO_InitStructure

void GPIO_Init(GPIOA, GPIO_InitTypeDef* GPIO_InitStruct)
```

由上面struct{...}结构体知道GPIO_InitStructure 有三个参数分别用.引出：
（也可以用->直接指向结构体成员即可，就不必用*取地址和.引出结构体）

3、最后使用输入或输出的函数控制GPIO口

```
GPIO_InitStructure.GPIO_Mode
GPIO_InitStructure.GPIO_Pin
GPIO_InitStructure.GPIO_Speed
```

第一个Mode有8种模式



```
typedef enum
{ GPIO_Mode_AIN = 0x0,
  GPIO_Mode_IN_FLOATING = 0x04,
  GPIO_Mode_IPD = 0x28,
  GPIO_Mode_IPU = 0x48,
  GPIO_Mode_Out_OD = 0x14,
  GPIO_Mode_Out_PP = 0x10,
  GPIO_Mode_AF_OD = 0x1C,
  GPIO_Mode_AF_PP = 0x18
}GPIOMode_TypeDef;
```

```
GPIO_Mode_AIN（Analog IN）//模拟输入
GPIO_Mode_IN_FLOATING  //浮空输入
GPIO_Mode_IPD (In Pull Down) //下拉输入
GPIO_Mode_IPU (In Pull Up) //上拉输入
GPIO_Mode_OUT_OD(Out Open Drain) //开漏输出
GPIO_Mode_PP_OD(Out Push Pull) //推挽输出
GPIO_Mode_AF_OD(Atl Open Drain) //复用开漏
GPIO_Mode_AF_PP(Atl Push Pull) //复用推挽
```

- 通过配置GPIO的端口配置寄存器，端口可以配置成以下8种模式

模式名称	性质	特征
浮空输入	数字输入	可读取引脚电平，若引脚悬空，则电平不确定
上拉输入	数字输入	可读取引脚电平，内部连接上拉电阻，悬空时默认高电平
下拉输入	数字输入	可读取引脚电平，内部连接下拉电阻，悬空时默认低电平
模拟输入	模拟输入	GPIO无效，引脚直接接入内部ADC
开漏输出	数字输出	可输出引脚电平，高电平为高阻态，低电平接VSS
推挽输出	数字输出	可输出引脚电平，高电平接VDD，低电平接VSS
复用开漏输出	数字输出	由片上外设控制，高电平为高阻态，低电平接VSS
复用推挽输出	数字输出	由片上外设控制，高电平接VDD，低电平接VSS

- 开漏 高电平没有驱动能力
- 推挽 高低电平都有驱动能力

  第二个GPIO_InitStructure.GPIO_Pin 的选择

```
#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 */
```

第三个GPIO_InitStructure.GPIO_Speed的选择

```
typedef enum
{
  GPIO_Speed_10MHz = 1,
  GPIO_Speed_2MHz,
  GPIO_Speed_50MHz
}GPIOSpeed_TypeDef;
#define IS_GPIO_SPEED(SPEED) (((SPEED) == GPIO_Speed_10MHz) || ((SPEED) == GPIO_Speed_2MHz) || \
                              ((SPEED) == GPIO_Speed_50MHz))
```

最后结果如下：

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
    GPIO_InitTypeDef GPIO_InitStructure;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_All;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
```

这样我们就把GPIOA 上所有引脚都初始化好了。

这里总共涉及了RCC和GPIO两个外设

★常用的三个RCC外设

```
void RCC_AHBPeriphClockCmd(uint32_t RCC_AHBPeriph, FunctionalState NewState);
void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState NewState);
void RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState NewState);
```

★读写GPIO的8个函数

```
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
uint16_t GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
```

例1：

GPIO_SetBits　：　拉高引脚输出电平 //1

GPIO_ResetBits：拉低引脚输出电平 //0

```
        GPIO_ResetBits(GPIOA, GPIO_Pin_0); //将PA0口置0
        Delay_ms(500);
        GPIO_SetBits(GPIOA, GPIO_Pin_0);//将PA0口置1
        Delay_ms(500);
        GPIO_WriteBit(GPIOA, GPIO_Pin_0, Bit_RESET);//Bit_RESET设置低电平
        Delay_ms(500);
        GPIO_WriteBit(GPIOA, GPIO_Pin_0, Bit_SET);
        Delay_ms(500);
        GPIO_WriteBit(GPIOA, GPIO_Pin_0, (BitAction)0);//强制转换
BitAction类型 0 1
        GPIO_WriteBit(GPIOA, GPIO_Pin_0, (BitAction)1);
        Delay_ms(500);
```

例2：读取GPIO上 电平操作

```
GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_13);//读取PB13上引脚输入 一般是接一个按键 读它的输入 即是否按下
GPIO_ReadOutputDataBit(GPIOA, GPIO_Pin_1)//读取PA1上引脚输出 一般接光敏传感器读它的AO口输出  即是否被遮挡
```

例3：灯的函数使用

```
//灯的打开
void LED1_ON(void)
{
    GPIO_ResetBits(GPIOA, GPIO_Pin_1);
}
//灯的熄灭
void LED1_OFF(void)
{
    GPIO_SetBits(GPIOA, GPIO_Pin_1);
}
//灯的翻转
void LED1_Turn(void)
{
    if (GPIO_ReadOutputDataBit(GPIOA, GPIO_Pin_1) == 0)
    {
        GPIO_SetBits(GPIOA, GPIO_Pin_1);
    }
    else
    {
        GPIO_ResetBits(GPIOA, GPIO_Pin_1);
    }
}
```

main函数

```
#include "stm32f10x.h"
#include "Delay.h"

int main(void)
{
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);

        GPIO_InitTypeDef GPIO_InitStructure;
```

```
        GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
        GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_12;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOB,&GPIO_InitStructure);

        //GPIO_SetBits(GPIOB, GPIO_Pin_0);
        GPIO_ResetBits(GPIOB, GPIO_Pin_12);
        while(1)
        {
                GPIO_WriteBit(GPIOB, GPIO_Pin_12, Bit_SET );
                Delay_ms(500);
                GPIO_WriteBit(GPIOB, GPIO_Pin_12, Bit_RESET );
                Delay_ms(500);
//                GPIO_ResetBits(GPIOB, GPIO_Pin_12);
//                Delay_ms(100);
//                GPIO_SetBits(GPIOB, GPIO_Pin_12);
//                Delay_ms(100);
        }

}
```

四、OLED函数的调用

1、oled

OLED驱动函数

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	A		H	e	l	l	o	W	o	r	l	d	!			
2	1	2	3	4	5		-	6	6							
3	A	A	5	5												
4	1	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1

函数	作用
OLED_Init();	初始化
OLED_Clear();	清屏
OLED_ShowChar(1, 1, 'A');	显示一个字符
OLED_ShowString(1, 3, "HelloWorld!");	显示字符串
OLED_ShowNum(2, 1, 12345, 5);	显示十进制数字
OLED_ShowSignedNum(2, 7, -66, 2);	显示有符号十进制数字
OLED_ShowHexNum(3, 1, 0xAA55, 4);	显示十六进制数字
OLED_ShowBinNum(4, 1, 0xAA55, 16);	显示二进制数字

上述代码添加.c和.h文件后可以直接调用