Principles and Practices of Microcontroller (Embedded System Design I) -IO2 and Interrupt

Gang Chen (陈刚)

Associate Professor

Institute of Unmanned Systems

School of data and computer science

Sun Yat-Sen University

https://www.usilab.cn/team/chengang/

- C语言复习
- MDK中寄存器地址名称映射分析

C语言复习

- 位操作
- define宏定义关键词
- ifdef条件编译
- extern变量申明
- typedef类型别名
- 结构体
- static关键字

■ 位操作：6种位操作运算符

运算符	含义	运算符	含义
&	按位与	~	取反
|	按位或	<<	左移
^	按位异或	>>	右移

```
GPIOA->ODR|=1<<5;
TIMx->SR = (uint16_t)~TIM_FLAG;
```

■ define宏定义关键词

define是C语言中的预处理命令，它用于宏定义，可以提高源代码的可读性，为编程提供方便。

常见的格式：

#define 标识符　字符串

“标识符”为所定义的宏名。“字符串”可以是常数、表达式、格式串等。

例如：

#define SYSCLK_FREQ_72MHz 72000000

定义标识符SYSCLK_FREQ_72MHz的值为72000000。

ifdef条件编译

单片机程序开发过程中，经常会遇到一种情况，当满足某条件时对一组语句进行编译，而当条件不满足时则编译另一组语句。条件编译命令最常见的形式为：

```
#ifdef 标识符
程序段1
#else
程序段2
#endif
```

例如：

```
#ifdef STM32F10X_HD
大容量芯片需要的一些变量定义
#end
```

■ extern变量申明

C语言中*extern*可以置于变量或者函数前，以表示变量或者函数的定义在别的文件中，提示编译器遇到此变量和函数时在其他模块中寻找其定义。

这里面要注意，对于extern申明变量可以多次，但定义只有一次。

- extern变量申明

main.c文件

```
u8 id;//定义只允许一次
main()
{
id=1;
printf("d%",id);//id=1
test();
printf("d%",id);//id=2
}
```

test.c文件

```
extern u8 id;

void test(void){
id=2;
}
```

typedef类型别名

定义一种类型的别名，而不只是简单的宏替换。可以用作同时声明指针型的多个对象。

```
typedef unsigned   char      uint8_t;
typedef unsigned short  int uint16_t;
typedef unsigned   int       uint32_t;
typedef unsigned  __int64  uint64_t;
```

■ 结构体：构造类型

```
Struct 结构体名{
成员列表1;
成员变量2;
…
}变量名列表;
```

在结构体申明的时候可以定义变量，也可以申明之后定义，方法是：
Struct 结构体名字　结构体变量列表；

◆ C语言关键字： static

- Static申明的局部变量，存储在静态存储区。
- 它在函数调用结束之后，不会被释放。它的值会一直保留下来。
- 所以可以说static申明的局部变量，具有记忆功能。

◆每次调用**getValue**函数之后，返回值是多少？

```
int getValue(void)
{
  int  flag=0;
  flag++;
  return flag;
}
```

```
int getValue(void)
{
  static int  flag=0;
  flag++;
  return flag;
}
```

MDK中寄存器地址名称映射分析

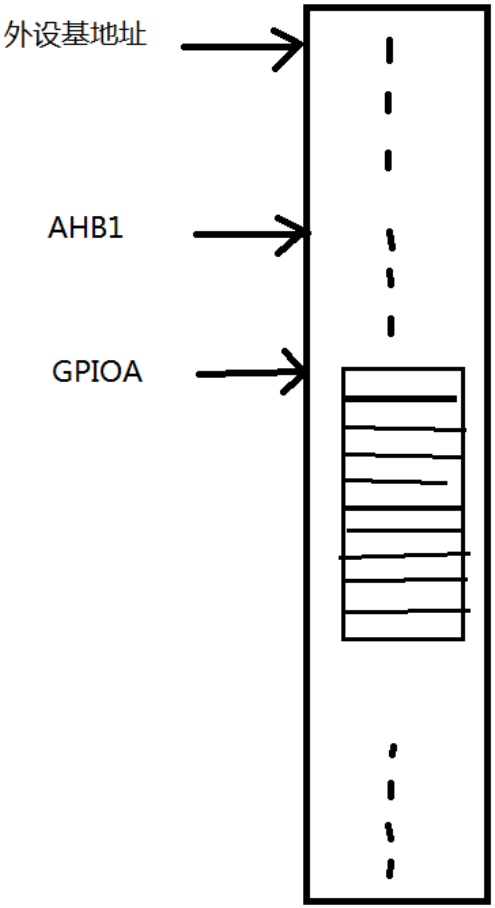
- **STM32中操作：**

GPIOA->ODR=0x00000000;

值0x00000000是怎么赋值给了GPIOA的ODR寄存器地址的呢？

也就是说GPIOA->ODR这种写法，是怎么与GPIOA的ODR寄存器地址映射起来的？

MDK中寄存器地址名称映射分析



#define PERIPH_BASE ((uint32_t)0x40000000)

#define AHB1PERIPH_BASE (PERIPH_BASE + 0x00020000)

#define GPIOA_BASE (AHB1PERIPH_BASE + 0x0000)

#define GPIOA ((GPIO_TypeDef *) GPIOA_BASE)

```
typedef struct
{
  __IO uint32_t CRL;
  __IO uint32_t CRH;
  __IO uint32_t IDR;
  __IO uint32_t ODR;
  __IO uint32_t BSRR;
  __IO uint32_t BRR;
  __IO uint32_t LCKR;
} GPIO_TypeDef;
```

NVIC中断优先级分组

- CM4内核支持256个中断，其中包含了16个内核中断和240个外部中断，并且具有256级的可编程中断设置。
- STM32F4并没有使用CM4内核的全部东西，而是只用了它的一部分。

  -STM32F40xx/STM32F41xx总共有92个中断。

  -STM32F42xx/STM32F43xx则总共有96个中断
- STM32F40xx/STM32F41xx的92个中断里面，包括10个内核中断和82个可屏蔽中断，具有16级可编程的中断优先级，而我们常用的就是这82个可屏蔽中断。

NVIC中断优先级分组

STM32F103xx Datasheet

■STM32F103xx增强型产品内置嵌套的向量式中断控制器，能够处理多达43个可屏蔽中断通道(不包括16个Cortex™-M3的中断线)和16个优先级。

■几十个中断，怎么管理？

NVIC中断优先级分组

- 中断管理方法：

首先，对STM32中断进行分组，组0~4。同时，对每个中断设置一个抢占优先级和一个响应优先级值。

分组配置是在寄存器SCB->AIRCR中配置：

组	AIRCR[10：8]	IP bit[7：4]分配情况	分配结果
0	111	0：4	0位抢占优先级，4位响应优先级
1	110	1：3	1位抢占优先级，3位响应优先级
2	101	2：2	2位抢占优先级，2位响应优先级
3	100	3：1	3位抢占优先级，1位响应优先级
4	011	4：0	4位抢占优先级，0位响应优先级

NVIC中断优先级分组

◆抢占优先级 & 响应优先级区别：

- 高优先级的抢占优先级是可以打断正在进行的低抢占优先级中断的。
- 抢占优先级相同的中断，高响应优先级不可以打断低响应优先级的中断。
- 抢占优先级相同的中断，当两个中断同时发生的情况下，哪个响应优先级高，哪个先执行。
- 如果两个中断的抢占优先级和响应优先级都是一样的话，则看哪个中断先发生就先执行；

◆ 举例：

- 假定设置中断优先级组为2，然后设置

中断3(RTC中断)的抢占优先级为2，响应优先级为1。
中断6（外部中断0）的抢占优先级为3，响应优先级为0。中断7（外部中断1）的抢占优先级为2，响应优先级为0。

那么这3个中断的优先级顺序为：中断7>中断3>中断6。

◆特别说明：

一般情况下，系统代码执行过程中，只设置一次中断优先级分组，比如分组2，设置好分组之后一般不会再改变分组。随意改变分组会导致中断管理混乱，程序出现意想不到的执行结果。

NVIC中断优先级分组

◆ 中断优先级分组函数：

void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup);

```
void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)
{
  assert_param(IS_NVIC_PRIORITY_GROUP(NVIC_PriorityGroup));
  SCB->AIRCR = AIRCR_VECTKEY_MASK | NVIC_PriorityGroup;
}
```

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
```

中断优先级设置

◆ 中断参数初始化函数

void NVIC_Init(NVIC_InitTypeDef NVIC_InitStruct);*

```
typedef struct
{
  uint8_t NVIC_IRQChannel; //设置中断通道
  uint8_t NVIC_IRQChannelPreemptionPriority;//设置响应优先级
  uint8_t NVIC_IRQChannelSubPriority; //设置抢占优先级
  FunctionalState NVIC_IRQChannelCmd; //使能/使能
} NVIC_InitTypeDef;
```

```
NVIC_InitTypeDef   NVIC_InitStructure;
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;//串口1中断
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1 ;// 抢占优先级为1
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;// 子优先级位2
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;//IRQ通道使能
NVIC_Init(&NVIC_InitStructure);    //根据上面指定的参数初始化NVIC寄存器
```

中断优先级设置

作用：用来挂起中断

- 中断解挂控制寄存器组：ICPR[8]

作用：用来解挂中断

static __INLINE void NVIC_SetPendingIRQ(IRQn_Type IRQn)；
static __INLINE uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)；
static __INLINE void NVIC_ClearPendingIRQ(IRQn_Type IRQn)

中断优先级设置

- 中断激活标志位寄存器组：IABR [8]

  作用：只读，通过它可以知道当前在执行的中断是哪一个

  如果对应位为1，说明该中断正在执行。

  static __INLINE uint32_t NVIC_GetActive(IRQn_Type IRQn)

NVIC总结

◆ 中断优先级设置步骤

①系统运行后先设置中断优先级分组。调用函数：

void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup);

整个系统执行过程中，只设置一次中断分组。

②针对每个中断，设置对应的抢占优先级和响应优先级：

void NVIC_Init(NVIC_InitTypeDef NVIC_InitStruct);*

③ 如果需要挂起/解挂，查看中断当前激活状态，分别调用相关函数即可。

外部中断概述

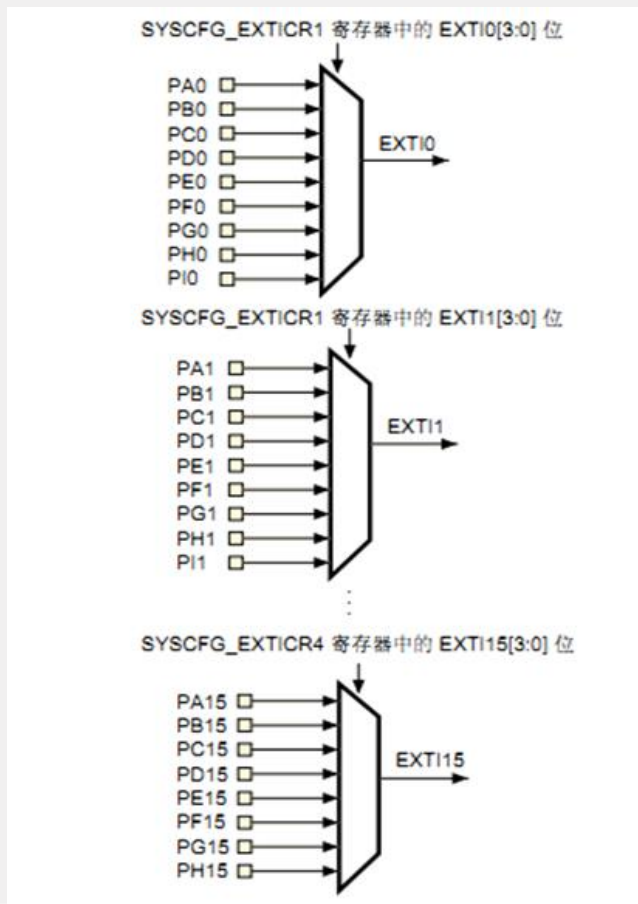
- STM32F4的每个IO都可以作为外部中断输入。
- STM32F4的中断控制器支持22个外部中断/事件请求：

EXTI线0~15：对应外部IO口的输入中断。
EXTI线16：连接到PVD输出。
EXTI线17：连接到RTC闹钟事件。
EXTI线18：连接到USB OTG FS唤醒事件。
EXTI线19：连接到以太网唤醒事件。
EXTI线20：连接到USB OTG HS(在FS中配置)唤醒事件。
EXTI线21：连接到RTC入侵和时间戳事件。
EXTI线22：连接到RTC唤醒事件。

每个外部中断线可以独立的配置触发方式（上升沿，下降沿或者双边沿触发），触发/屏蔽，专用的状态位。

从上面可以看出，STM32F4供IO使用的中断线只有16个，但是STM32F4xx系列的IO口多达上百个，STM32F103ZGT6(112)，那么中断线怎么跟io口对应呢？

- 对于每个中断线，我们可以设置相应的触发方式（上升沿触发，下降沿触发，边沿触发）以及使能。



GPIOx.0映射到EXTI0
GPIOx.1映射到EXTI1
…
GPIOx.15映射到EXTI15

◆是不是16个中断线就可以分配16个中断服务函数呢？

◆IO口外部中断在中断向量表中只分配了7个中断向量，也就是只能使用7个中断服务函数

位置	优先级	优先级类型	名称	说明	地址
7	14	可设置	EXTI1	EXTI线1中断	0x0000_005C
8	15	可设置	EXTI2	EXTI线2中断	0x0000_0060
9	16	可设置	EXTI3	EXTI线3中断	0x0000_0064
10	17	可设置	EXTI4	EXTI线4中断	0x0000_0068
23	30	可设置	EXTI9_5	EXTI线[9:5]中断	0x0000_009C
40	47	可设置	EXTI15_10	EXTI线[15:10]中断	0x0000_00E0

◆ 从表中可以看出，外部中断线5~9分配一个中断向量，共用一个服务函数
外部中断线10~15分配一个中断向量，共用一个中断服务函数。

■ 中断服务函数列表：

EXTI0_IRQHandler
EXTI1_IRQHandler
EXTI2_IRQHandler
EXTI3_IRQHandler
EXTI4_IRQHandler
EXTI9_5_IRQHandler
EXTI15_10_IRQHandler

◆ 外部中断常用库函数

①void SYSCFG_EXTILineConfig(uint8_t EXTI_PortSourceGPIOx, uint8_t EXTI_PinSourcex);
//设置IO口与中断线的映射关系

exp: SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE, EXTI_PinSource2);//区别M3

②void EXTI_Init(EXTI_InitTypeDef* EXTI_InitStruct);
//初始化中断线：触发方式等

③ITStatus EXTI_GetITStatus(uint32_t EXTI_Line);
//判断中断线中断状态，是否发生

④void EXTI_ClearITPendingBit(uint32_t EXTI_Line);
//清除中断线上的中断标志位

⑤***RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);//使能SYSCFG时钟***
//这个函数非常重要，在使用外部中断的时候一定要先使能SYSCFG时钟

外部中断常用库函数

◆ EXTI_Init函数

void EXTI_Init(EXTI_InitTypeDef EXTI_InitStruct)；*

```
typedef struct
{
  uint32_t EXTI_Line;   //指定要配置的中断线
  EXTIMode_TypeDef EXTI_Mode;   //模式：事件 OR中断
  EXTITrigger_TypeDef EXTI_Trigger;//触发方式：上升沿/下降沿/双沿触发
  FunctionalState EXTI_LineCmd;  //使能 OR失能
}EXTI_InitTypeDef;
```

```
 EXTI_InitStructure.EXTI_Line=EXTI_Line2;
 EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
 EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
 EXTI_InitStructure.EXTI_LineCmd = ENABLE;
 EXTI_Init(&EXTI_InitStructure);
```

外部中断一般配置过程

◆外部中断的一般配置步骤：

① 使能SYSCFG时钟：
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

② 初始化IO口为输入。
GPIO_Init();

③ 设置IO口与中断线的映射关系。
void SYSCFG_EXTILineConfig();

④ 初始化线上中断，设置触发条件等。
EXTI_Init();

⑤ 配置中断分组（NVIC），并使能中断。
NVIC_Init();

⑥ 编写中断服务函数。
EXTIx_IRQHandler();

⑦ 清除中断标志位
EXTI_ClearITPendingBit();

中断按键点灯实验