**优微电子科技**

**CC2530 基础实验**

**外部中断配置**

# 实验介绍

所谓中断是指CPU在正常执行程序的过程中，由于内部或外部事件的触发或由程序的预先安排，引起CPU暂时中断当前正在运行的程序，而转去执行为内部或外部事件或程序预先安排的事件的服务子程序，待中断服务子程序执行完毕后，CPU再返回到被暂时中断的程序处(断点)继续执行原来的程序，这一过程叫做中断。

CC2530有18个中断源。每个中断源都有它自己的位于一系列SFR寄存器中的中断请求标志。相应标志位请求的每个中断可以分别使能或禁用。

# 实验目的

（1）通过本实验了解CC2530 的外部中断的配置；

（2）通过本实验了解CC2530的外部中断功能的运用。

# **三、**实验设备

## 3.1、硬件设备

（1）ZIGBEE开发板

（2）SmartRF04EB仿真器

## 3.2、软件环境

（1）IAR Embedded Workbench for 8051集成开发环境。

# **四、**实验原理

## 4.1、IO口中断

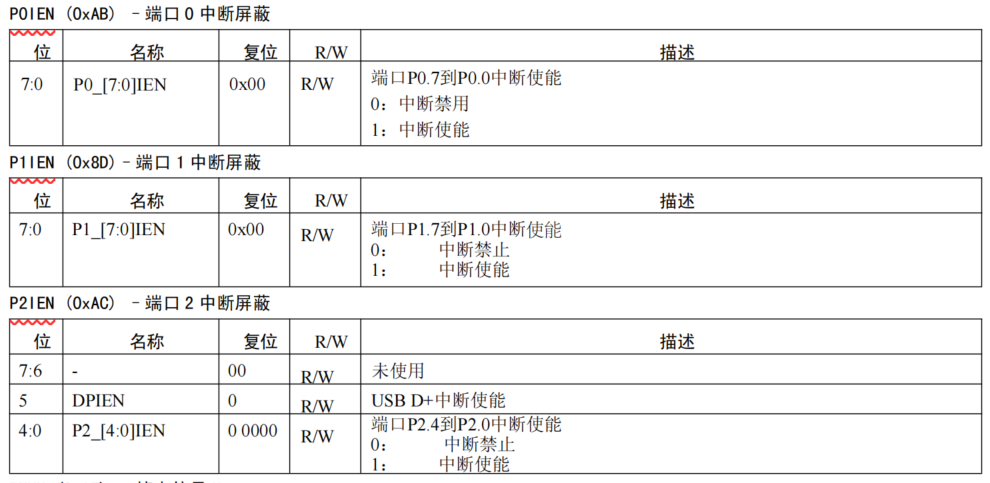
通用I/O 引脚设置为输入后，可以用于产生中断。中断可以设置在外部信号的上升或下降沿触发。P0、P1或P2端口都有中断使能位，对位于IENx(x=1,2)寄存器内的端口所有的位都是公共的。如下：

1、IEN1.P0IE：P0 中断使能；

2、IEN2.P1IE：P1 中断使能；

3、IEN2.P2IE：P2 中断使能。

除了这些公共中断使能之外，每个端口的为都有PxIEN(x=0,1,2)寄存器的单独中断使能。即使配置为通用IO口或片上外设功能IO口使能时都有中断产生。



当发生中断时，PxIFG(x=0,1,2)中断标志寄存器中相应位的中断状态标志将自动置1。



不管引脚是否设置了它的中断使能位，中断状态都被设置。当中断已经执行，硬件会自动清除该引脚的中断状态标志，即写0。若中断被禁用，则此标志不会自动清除，需软件手动清除。这个标志必须在清除CPU端口中断标志之前被清除，顺序为先清除PxIFG(x=0,1,2)寄存器的对应位引脚的中断状态标志，再清除CPU的端口中断标志PxIF(x=0,1,2)寄存器。这是因为PxIF寄存器是端口级汇总标志，只要器下属任意引脚的中断状态标志为1，PxIF寄存器就会保持置位。若先清除PxIF，但未清除引脚的状态标志，PxIF会因状态标志仍为1而立即重新置位。

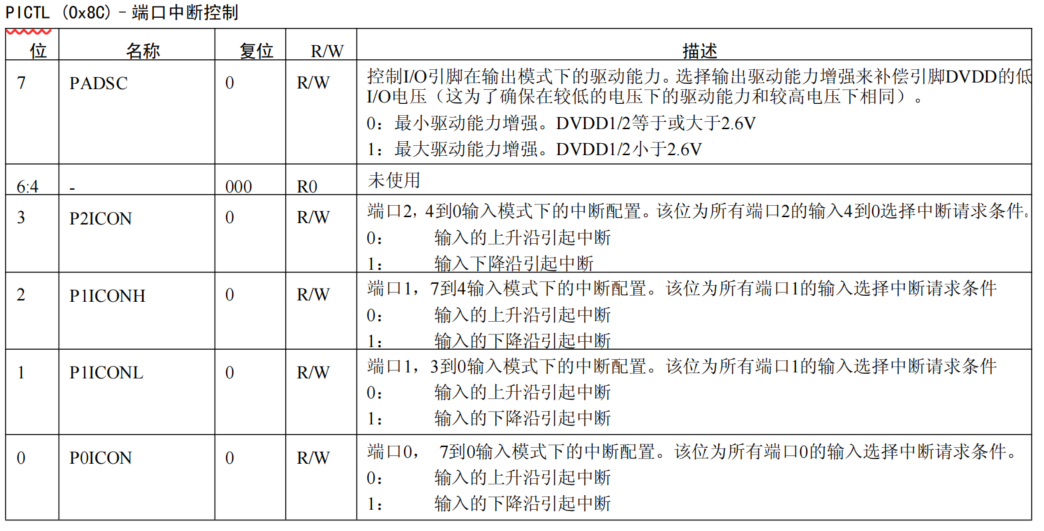
## 4.2、中断配置

CPU 有18个中断源。每个中断源都有它自己的位于一系列SFR寄存器中的中断请求标志。相应标志位请求的每个中断可以分别使能或禁用。

每个中断请求可以通过设置中断使能SFR寄存器的中断使能位IEN0，IEN1或者IEN2使能或禁止。CPU的中断使能相对应的SFR。

外部中断配置的步骤：

1. 初始化IO口工作在普通IO、上拉输入状态。
2. 通过配置PxIE(x=0,1,2)寄存器打开IO口组中断。
3. 通过配置PxIEN(x=0,1,2)寄存器打开组内对应的具体某个IO口中断。
4. 通过配置PICTL寄存器配置为上升沿或下降沿触发。
5. 打开CPU总中断。



# **五、**实验演示

本次实验演示的是利用P0\_0上的按键通过外部中断控制LED的状态。

首先在前面实验代码的基础上，初始化P0\_0口工作在普通IO、上拉输入状态，然后我们进行外部中断配置：

EA = 1; //开启总中断

P0IE = 1; //开启P0组中断

P0IEN |= 0x01; //开启P0\_0中断

PICTL |= 0x01; //配置P0\_0-P0\_7为下降沿触发

整合一下：

void key\_init(void)

{

P0SEL &= ~0x13;

P0DIR |= ~0x13;

P0INP |= ~0x13;

P2INP |= ~0x20;

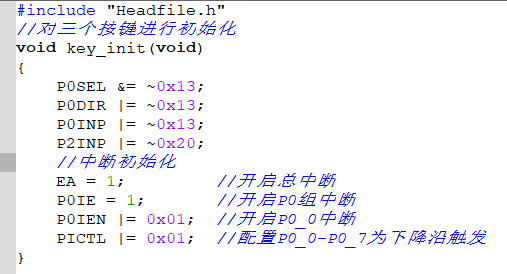
EA = 1; //开启总中断

P0IE = 1; //开启P0组中断

P0IEN |= 0x01; //开启P0\_0中断

PICTL |= 0x01; //配置P0\_0-P0\_7为下降沿触发

}



这样外部中断就配置好了。

接下来是中断函数。

中断函数是有一个固定的形式：

#pragma vector = PxINT\_VECTOR(x=0,1,2)

\_\_interrupt void 函数名(void)

{

执行语句;

}

我们先定义一个变量用于描述LED的状态：

uint8\_t status = 0;

我们用的是P0组的中断，，然后我们函数名简单命名：

#pragma vector = P0INT\_VECTOR

\_\_interrupt void interrupt(void)

{

}

当触发中断时，P0IFG寄存器的第0位将置1，判断是否进入中断只需判断对应位是否为1，可以用 P0IFG & 0x01进行判断。

if(P0IFG & 0x01) //判断是否进入中断

{

delayms(10); //按下消抖

while(KEY1 == 0);

delayms(10); //释放消抖

status = ~status; //改变状态

led\_proc(1,status);

}

执行完中断一定要消除中断标志，防止无法进入中断或一直卡在中断里。

P0IFG = 0;

P0IF = 0;

整合一下以上代码：

uint8\_t status=0;

#pragma vector = P0INT\_VECTOR

\_\_interrupt void interrupt(void)

{

if(P0IFG & 0x01) //判断是否进入中断

{

delayms(10); //按下消抖

while(KEY1 == 0);

delayms(10); //释放消抖

status = ~status; //改变状态

led\_proc(1,status);

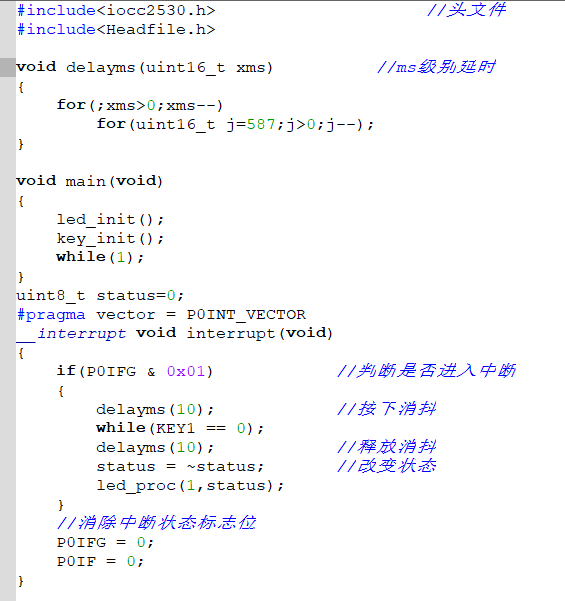
}

//消除中断状态标志位

P0IFG = 0;

P0IF = 0;

}



如果还想有更多的中断，如加入按键2的话，只需进行配置然后在中断函数内进行类似判断即可。