**优微电子科技**

**CC2530 基础实验**

**普通IO口配置**

# **一、**实验介绍

CC2530有21个IO口，可以配置为通用IO口或片上外设IO口，如配置为连接到ADC、定时器或USART外设。可以通过一系列寄存器对IO口进行配置。

I/O端口具备如下重要特性：

* 21 个数字I/O引脚
* 可以配置为通用I/O 或外部设备I/O
* 输入口具备上拉或下拉能力
* 具有外部中断能力。

21 个I/O引脚都可以用作于外部中断源输入口。

# **二、**实验目的

通过本实验了解并掌握CC2530通用IO口的使用及相关寄存器的配置。

# 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

CC2530一共有21个IO口可使用，其中P0和P1有8个（0-7），而P2只有5个（0-4）。其中P10和P11有20mA的输出驱动能力，而其他的IO口只有4mA的驱动能力。

每个IO口都可以作为通用IO口使用，也可以具备片上外设功能。

通用IO口指可执行输入或输出的端口，例如输出高或低电平使对应IO口的LED实现亮或灭。用作通用IO口时，引脚可以组成3个8位端口，端口0、端口1和端口2表示为P0、P1和P2。其中P0和P1是完全的8位端口，而P2仅有5位可用。

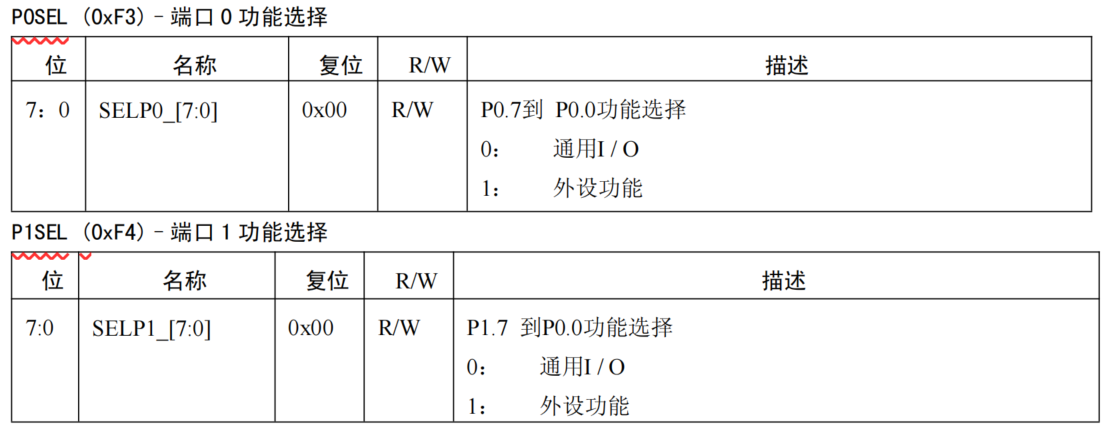
片上外设功能指IO口作为串口使用或ADC输入等。

配置IO口，首先配置PxSEL寄存器来设定IO口为通用IO口或具备片上外设功能的IO口，当配置为通用IO口时，通过配置PxDIR寄存器来设定IO口为输入模式或输出模式，如果为输入模式，则需要通过配置PxINP寄存器来设定为上、下拉输入或三态输入，如果为输出模式，只需配置IO口为高电平或低电平。

## 4.2、PxSEL寄存器

PxSEL（x=0，1，2）寄存器，叫功能选择寄存器，用于选择IO口位通用IO口或外设功能，有8位，每一位对应IO组中相应的IO口。

P0SEL寄存器和P1SEL寄存器的8位都可以用来配置IO口的功能。



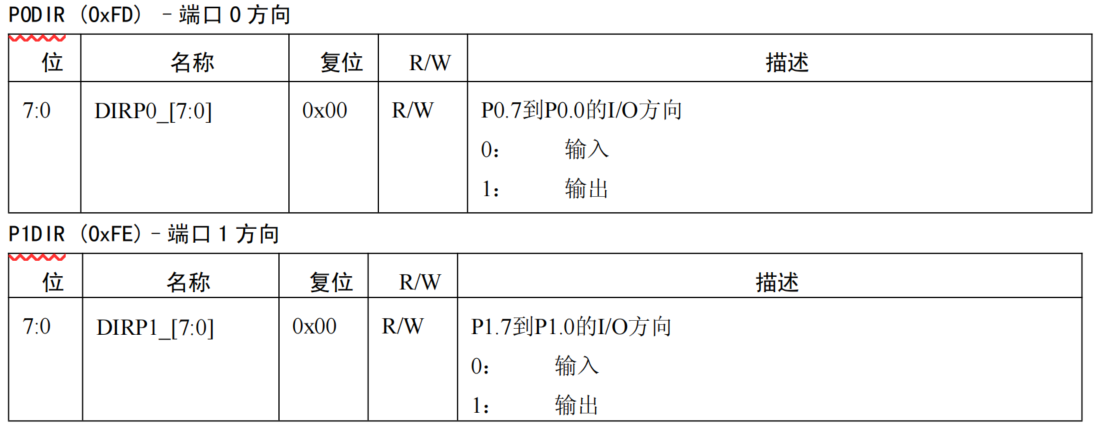
由于P2组IO口只有5个引脚可用，其中P2\_1和P2\_2用于下载程序使用，所以这两个IO口不需要配置，即P2SEL寄存器的低3位为P2\_4、P2\_3、P2\_0。P2SEL寄存器高5位用于端口1外设优先级控制。



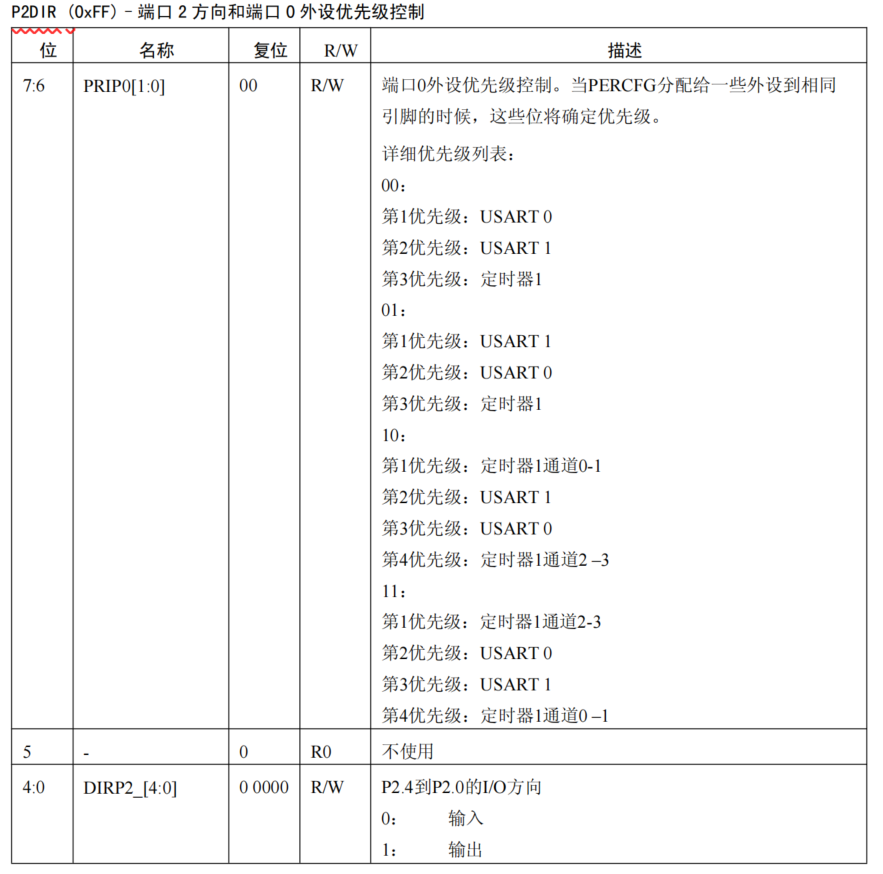
## 4.3、PxDIR寄存器

PxDIR（x=0，1，2）寄存器，叫方向寄存器，用于配置IO口为输入模式或输出模式，有8位，每一位对应IO组中相应的IO口。

P0DIR寄存器和P1DIR寄存器的8位都可以用来配置IO口的方向。



由于P2组IO口只有5个引脚可用，所以P2DIR寄存器的低5位用于配置P2\_4-P2\_0的方向。P2DIR寄存器高2位用于端口0外设优先级控制。



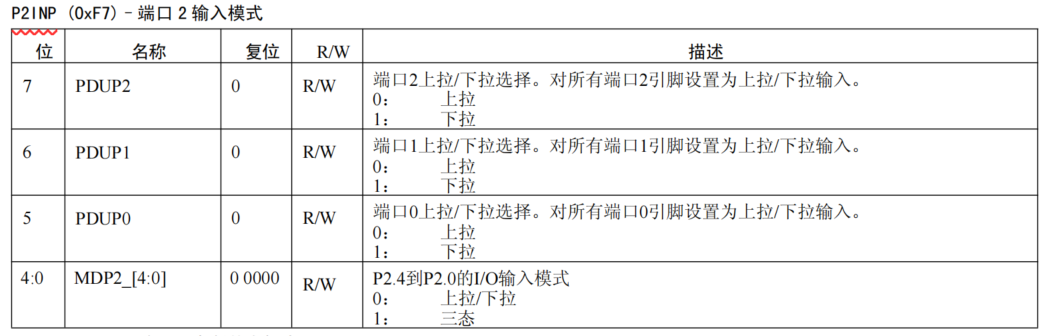
## 4.4、PxINP寄存器

PxINP（x=0，1，2）寄存器，叫输入模式寄存器，用于配置IO口为上、下拉输入模式或三态输入模式，有8位，每一位对应IO组中相应的IO口。

P0DIR寄存器和P1DIR寄存器的8位都可以用来配置IO口的输入模式。



由于P2组IO口只有5个引脚可用，所以P2INP寄存器的低5位用于配置P2\_4-P2\_0的输入模式，而高3位用于对3个端口的所有引脚设置为上拉或下拉输入模式。



# 五、实验演示

接下来我们解读一下第一次实验的代码。

#include<iocc2530.h> //头文件

void main(void)

{

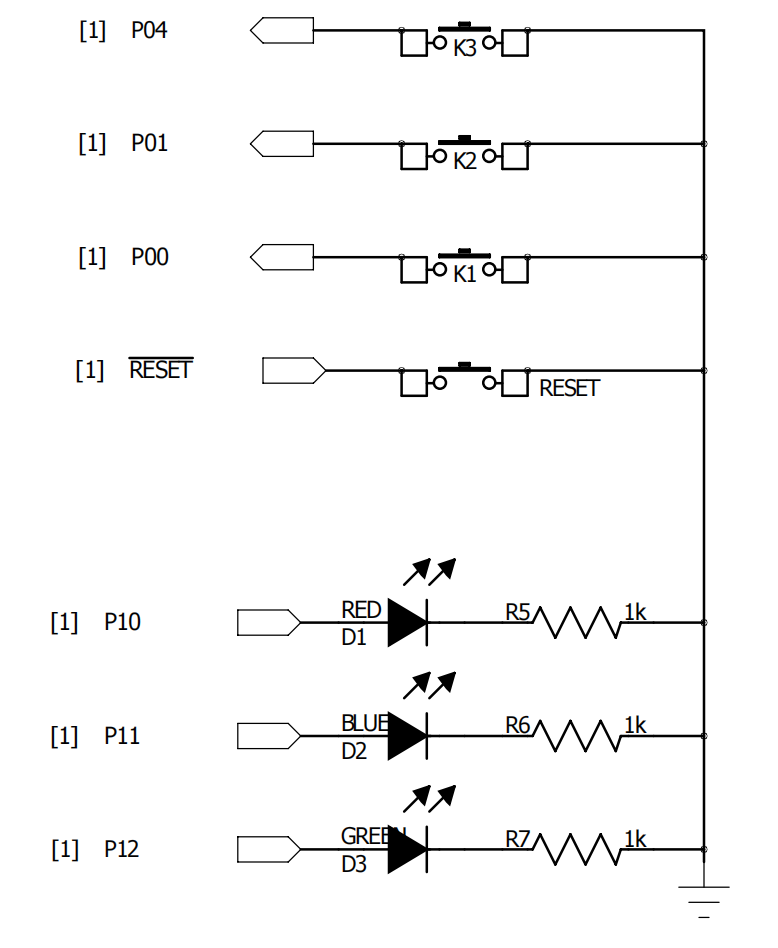
P1SEL &= ~0x01; //配置P1\_0为通用IO口

P1DIR |= 0x01; //配置P1\_0为输出模式

P1\_0 = 1; //配置P1\_0为高电平

}

再看一下原理图，我们可以看到P1\_0-P1\_2上都接有LED，接下来我们将利用P0\_0上的按键来控制3个LED的状态。



我们首先初始化LED和按键，配置P1\_2-P1\_0和P0\_0为通用IO口，且配置P1\_2-P1\_0为输出模式，P0\_0为输入模式。

P1SEL &= ~0x07; //1111 1000 配置P1\_2-P1\_0为通用IO口

P1DIR |= 0x07; //0000 0111 配置P1\_2-P1\_0为输出模式

P1 = 0x00; //0000 0000 配置P1\_2-P1\_0为低电平

P0SEL &= ~0x00; //1111 1110 配置P0\_0为通用IO口

P0DIR |= 0x00; //0000 0000 配置P0\_0为输入模式

P0INP |= 0x00; //0000 0000 配置P0\_0为上、下拉输入模式

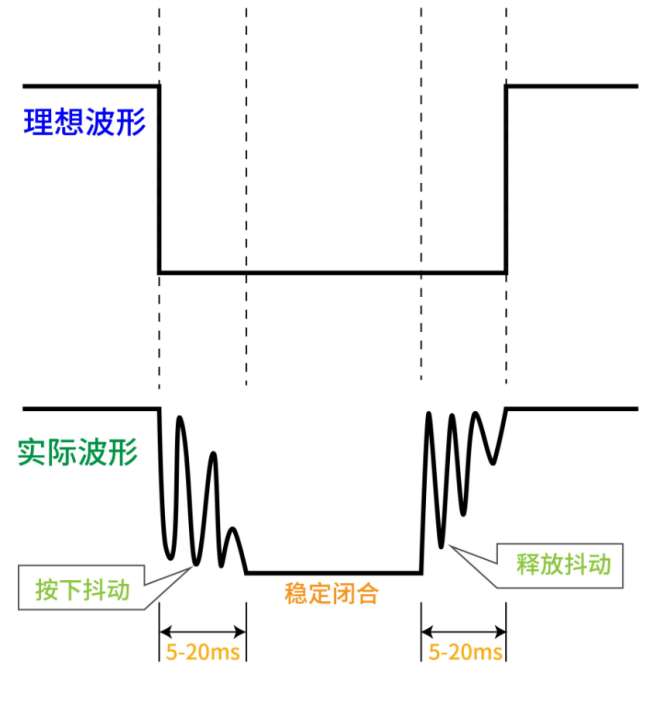
由于我们的按键模块没有接上电阻，所以我们不需要配置为上拉或下拉输入模式的一种。

当把按键按1下时，P0\_0口检测到的是低电平，也就是我们可以用if语句来判断P0\_0口的电平是否为低电平，

if(P0\_0 == 0)

当检测为低电平时说明此刻按键被按下，就可以执行我们想要的效果。

但是按键在被按下和释放时是会发生抖动的，这个抖动可能会使得我们在按下或释放时，程序自动在这5-20ms内检测我们按下或释放多次，此时就无法达到预期的效果。



这种抖动并不是无法消除，我们可以用软件来消除该抖动，只需在我们按下和释放时各进行一次5-20ms的延时，使得在这段时间内程序进入延时而没有去检测抖动。

void delay\_ms(unsigned int xms) //简单的ms级别延时模块

{

for(;xms>0;xms--)

for(unsigned int j=587;j>0;j--);

}

//main函数里

if(P0\_0 == 0) //检测到按键1被是否按下

{

delay\_ms(10); //按键按下时进行一次5-20ms的延时

while(P0\_0 == 0); //当按键未被释放时进入死循环

delay\_ms(10); //按键释放时进行一次5-20ms的延时

}

回顾下我们要实现的效果，利用P0\_0上的按键来控制3个LED的状态，接下来我们把所有的代码给整合起来。

#include<iocc2530.h> //头文件

void delay\_ms(unsigned int xms) //ms级别延时

{

for(;xms>0;xms--)

for(unsigned int j=587;j>0;j--);

}

void main(void)

{

P1SEL &= ~0x07; //配置P1\_2-P1\_0为通用IO口

P1DIR |= 0x07; //配置P1\_2-P1\_0为输出模式

P1 = 0x00; //配置P1\_2-P1\_0为低电平

P0SEL &= ~0x01; //配置P0\_0为通用IO口

P0DIR |= 0x00; //配置P0\_0为输入模式

P0INP |= 0x00; //配置P0\_0为上、下拉输入模式

while(1)

{

if(P0\_0 == 0) //检测到按键1被是否按下

{

Delay\_ms(10); //按键按下时进行一次5-20ms的延时

while(P0\_0 == 0); //当按键未被释放时进入死循环

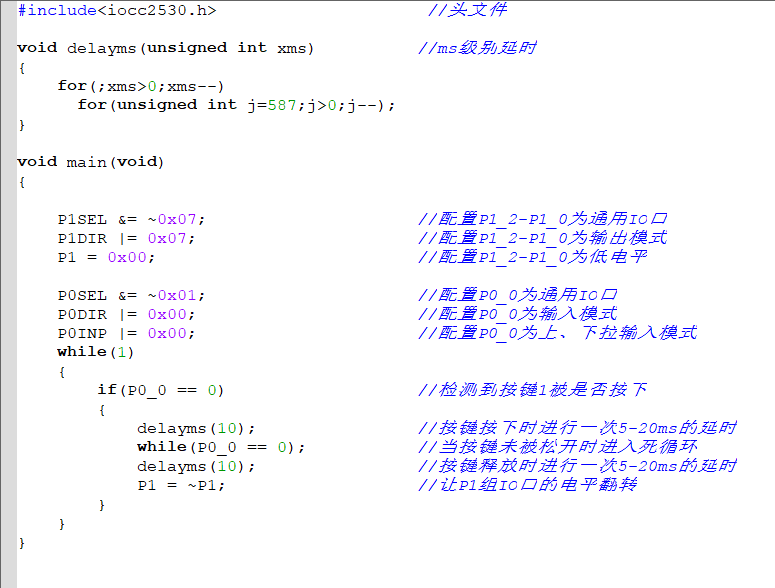
Delay\_ms(10); //按键释放时进行一次5-20ms的延时

P1 = ~P1; //让P1组IO口的电平翻转

}

}

}



以下是效果图。





当按键1被按下时，进入10ms的按下消抖，随后进入死循环，当按键1被释放时，跳出死循环并进入10ms的释放消抖，随后LED的状态发生改变。

注意的是我们实现的是当按键释放时LED的状态才发生改变。如果想实现按下的时候改变LED的状态，只需完成以下操作：

if(P0\_0 == 0) -> if(P0\_0 == 1)

while(P0\_0 == 0) -> while(P0\_0 == 1)