基础

状态设置代码

|= 置为1 &=~ 置为0 ^= 置为反

① PXSEL：端口特殊功能选择，例如P1SEL|=BIT3，即为选中P1.3的特殊功能，如果使用普通的输入输出功能，可以不设置，或者设置成P1SEL&=~BIT3。

②PXDIR：端口输入/输出寄存器。输出设置：P1DIR|=BIT3；输入设置：P1DIR&=~BIT3。其余的类似。

③PXOUT:端口输出寄存器。若想输出高电平，P1OUT|=BIT3；若想输出低电平，P1OUT&=~BIT3。

④PXIN：输入状态寄存器。可以读取相应端口的输入状态。

⑤PXREN：上拉电阻使能寄存器。如果想要使能相应端口的上拉电阻，只需要设置：P1REN|=BIT3。

⑥PXIES：端口触发寄存器。P1IES|=BIT3，下降沿触发；P1IES&=~BIT3，上升沿触发。

⑦PXIE：端口中断使能寄存器。P1IE|=BIT3，使能P1.3的中断。

⑧PXIFG:端口中断状态寄存器。用P1IFG&BIT3可以判断是哪一个端口触发了中断，同时需要在中断服务程序里软件清零。

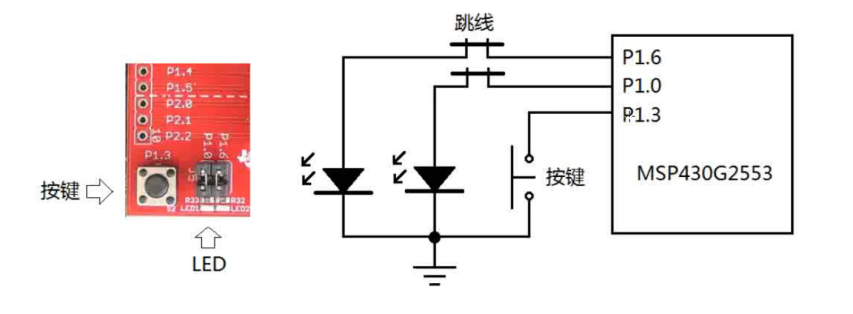
按键控制

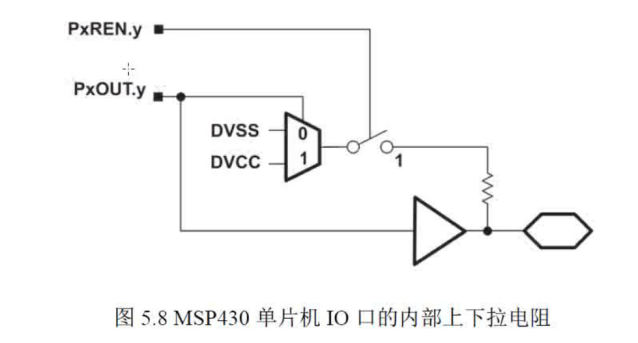
P1.0/p1.6 分别对应两个led灯，p1.3连接按键

P1.3如果为高电平则没有按下，p1.3如果为低电平则被按下、

If(P1IN&BIT3) P1OUT=-BIT6（p1.3的输出电平，1为高，0为低）（熄灭led2）

P1IN 反应了端口的输入状态 ，不能被赋值

  
上下拉电阻（解决外部影响）



代码实例：  
WDTCTL = WDTPW | WDTHODE;

PIDIR |= BIT6;//P1.6为输出

PIOUT &= ~BIT6;//LED2为灭、

PIDIR &= ~BIT3;//P1.3为输入

PIREN |= BIT3;//使p1.3能上或下拉电阻，1为能，0为否

PIOUT |= BIT3;//使p1.3上拉电阻，1为上拉，0为下拉

while(1)

{

if(P1IN & BIT3)// P1.3口为高电平，没被按下

{ P1OUT &=~ BIT6；熄灭LED2}

else

{P1OUT |=BIT6;}

}

时钟代码：

MCLK频率设置:

DCLCTL=CALDCO\_8MHZ；(可以选择1,8,12,16MHZ，1MHZ是初始值）

BCSCTL1=CALBC1\_8MHZ；

设置延迟：

\_ \_delay\_cycles(500000) (在设置时钟频率下执行需要500毫秒

串口（USCI）

分为 USCI\_AX 和USCI\_BX模块 （x为0,1,2,3），但g2553 只有A0和B0

P1.1/UCA0RXD 接收引脚

P1.2/UCA0TXD 发送引脚

UCSYNC被清零即选择了UART模式（串行通信协议）

UART基本参数配置：

波特率（Baud Rate）：波特率确定数据传输的速率，通常以位/秒（bps）为单位。可以设置波特率以适应通信的需求。（9600）

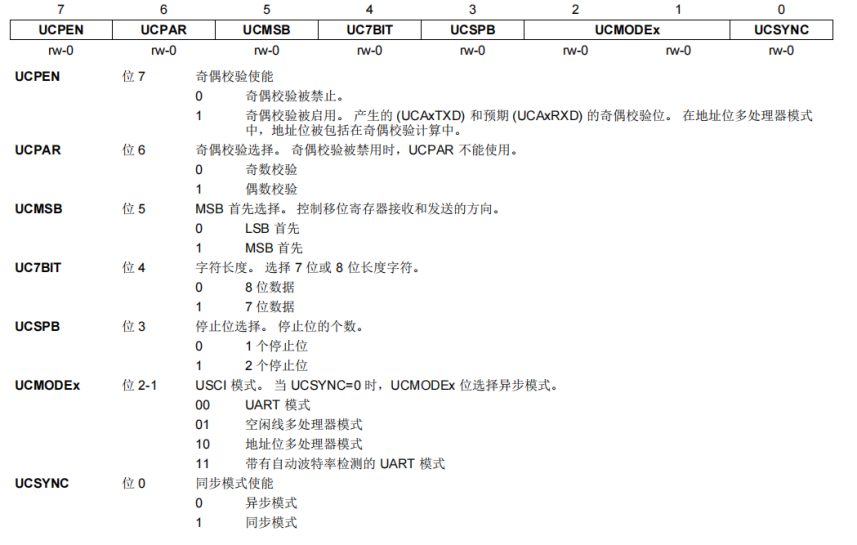
数据位：决定每个数据帧中的位数，通常为8位。

停止位：指定数据帧结束的位数，通常为1或2位。（1）

校验位：可选项，用于检测传输中的错误。



USCIAX 控制寄存器 0 配置



Lsb：第0位先发送Msb：第7位先发送

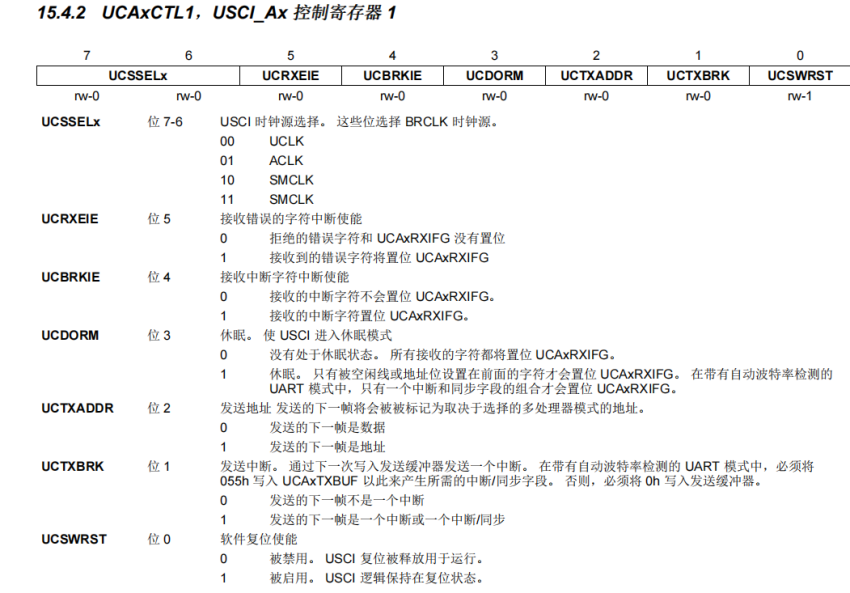
一般配8位（等于数据位）、

停止位一般为1

选择USCI模式

置为0（异步）

USCIAX 控制寄存器1 配置



1-5位不用配置  
配置0位和6-7位

串口配置代码（可以在用户手册第15个标题UART模式中寻找，具体代码如下）

WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

DCOCTL = CALDCO\_1MHZ;

BCSCTL1 = CALBC1\_1MHZ;

BCSCTL2 &=~SELS;//配置时钟频率

UCA0CTL1 |=UCSWRST; //复位USCI\_AX

UCA0CTL1 &= ~UCSWRST; // 启用UART模块

UCA0CTL1 |= UCSSEL1; // 选择SMCLK作为时钟源

UCA0BR0 = 0x68;

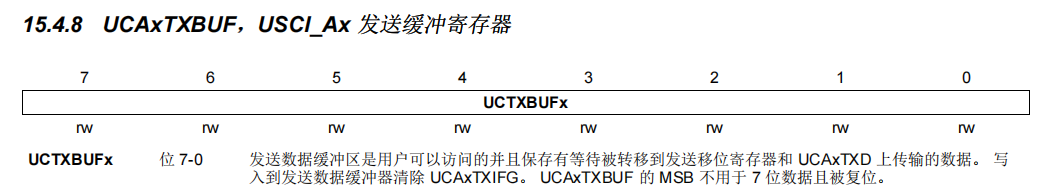
UCA0BR1 = 0x00;

UCA0MCTL = 1<<1; // 波特率9600(也可以用十进制数）

P1SEL |=BIT1+BIT2;

1SEL2 |=BIT1+BIT2;//使端口能复用

UCA0CTL1 &= ~UCSWRST; //清除复位，使能UART



while(1)

{UCA0TXBUF = 0x55; // 向电脑发送16进制数55

\_\_delay\_cycles(500000);}

return 0;

}

串口调试

uint8\_t 是一个无符号8位整数类型，大小为1字节，表示范围在0到255之间。uint16\_t：

uint16\_t 是一个无符号16位整数类型，大小为2字节，表示范围在0到65,535之间。

串口发送字符

void UARTsend\_char(uint8\_t \*p,uint8\_t num)

{

uint8\_t cnt=0;

for(cnt=0,cnt<num,cnt++)

{

while(UCA0STAR&UCBUSY);//xxxxxxx1&00000001(判断是否在发送信号）

UCA0TXBUF =\*(p+cnt);

}

串口发送数字

void UARTsend\_num(uint16\_t num)

{

unit8\_t buff6={0,0,0,0,0,‘/n’};

unit8\_t cnt=0;

for(cnt=0,cnt<5,cnt++)

{

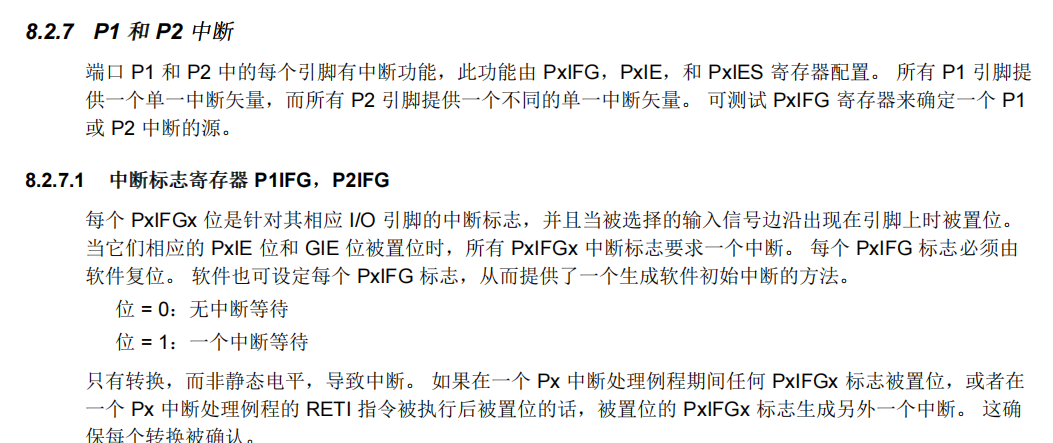
buff[4-cnt]=(unit8\_t) (num%10+‘0’);

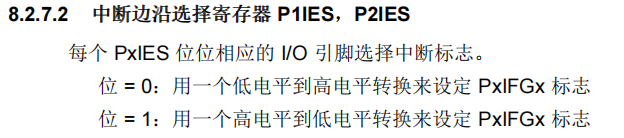
num=num/10;

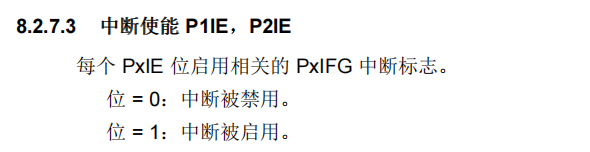
}

I/0中断

总中断开口：GIE







\_\_bic\_SR\_register(GIE);//打开全局中断

P1IE |=BIT3;打卡P1.3中断

P1IES |=BIT3;//设为下降沿触发

P1IEG &=~BI3;//清除中断标志

中断函数：  
#pragma vector=PORT1\_VECTOR//P1对应的中断向量

\_\_interrupt void 函数名(void)

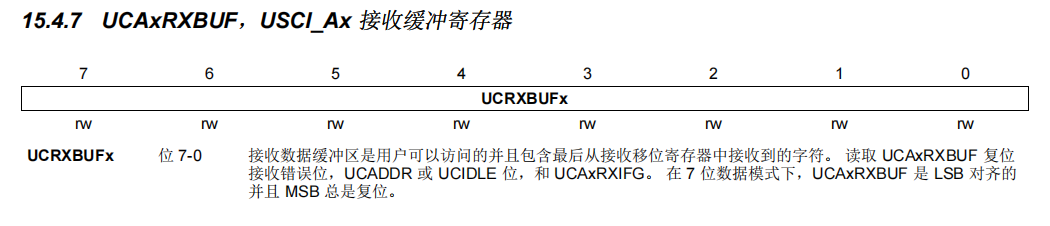
{

if(P1IFG & BIT3)//判断是否是p1.3中断

P1IFG &=~BIT3;//清除标志位

P10UT ^= BIT6;//写需要在中断中做的事  
}

串口接收和中断功能



前面与发送配置一样，剩下的位于清除复位位后

IE2 |= UCA0RXIE;//接收中断启用

IFG2 &= ~UCA0RXIFG//清空接收中断标志

while(1)

{  
if(iscomend=0)

{

iscomend=0//清除标志位，防止重复执行

Command(combuff);//执行命令

}

\_\_interrupt void UART\_receive(void)//接受中断服务函数

{

static unit8\_t cnt=0;//cnt设为静态

if(IFG2 & UCA0RXIFG)//检查是否有UCAI\_AO的中断，A0和B0接收中断共享一个向量

{

IFG2 &= UCAORXIFG;//清除标志位

combuff[cnt++]=UCA0RXBUF;//保存命令

cnt %= 20;//防止cnt大于20，导致缓冲区溢出命令结束符

if(combuff[cnt-1]==’\n’)//如果接收到结束符

{

cnt=0;//复位计数器;

iscomend=1;//命令接收完毕

}

}

void Command(uint8\_t\*comebuff)//检查是否收到命令

{

const uint8\_t charbuff[5][10]={”起床”，”吃饭”}；//让const的内容存在ROM中，ROM大一点

if(comebuff[0]==charbuff[1][0]&comebuff[1]==charbuff[1][1])

{

UARTsend\_char(“吃什么”,4)//每个汉字占有两个字节，也可以写其他命令

}

else

{

//省略

}

}