更多嵌入式 Linux 学习资料，请关注：一口 Linux 回复关键字:1024

一、CC2530 简介

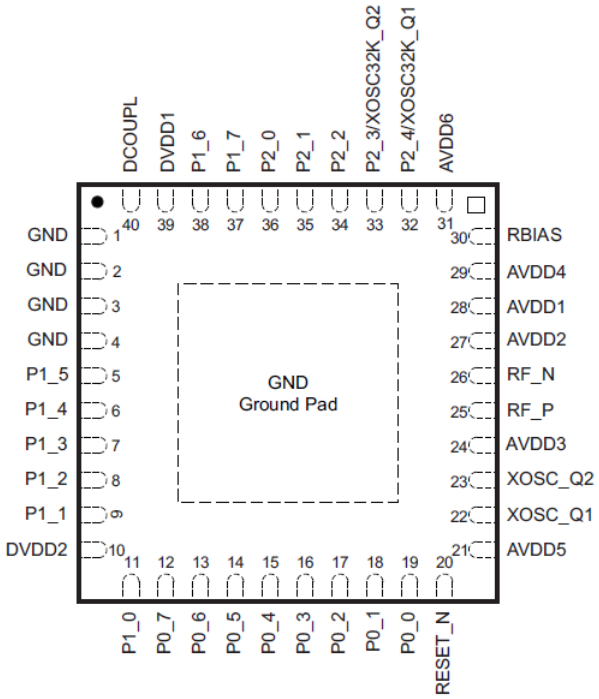
进行 ZigBee 无线传感器网络开发，首先，需要有相应的硬件支持（尤其是需要支持 ZigBee 协议栈的硬件）；此外还需要相应的软件支持（最好是相应的支持 ZigBee 的软件协议栈），当然，还需要下载器将程序下载到相应的硬件。

CC2530 单片机是一款完全兼容 8051 内核，同时支持 IEEE 802.15.4 协议的无线射频单片机。

CC2530 微控制器采用 QFN40 封装，有 40 个引脚。

其中，有 21 个数字 I/O 端口，其中 P0 和 P1 是 8 位端口，P2 仅有 5 位可以使用。这 21 个端口均可以通过编程进行配置。

实际上，在 P2 端口的 5 个引脚中，有 2 个需要用作仿真，有 2 个需要用作晶振，在 CC2530 的开发中真正能够使用的只有 17 个引脚。



在微控制器内部，有一些特殊功能的存储单元，这些单元用来存放控制微控制器内部器件的命令、数据或运行过程中的一些状态信息，这些寄存器统称为“特殊功能寄存器（SFR）”。

操作微控制器的本质，就是对这些特殊功能寄存器进行读写操作，并且某些特殊功能寄存器可以位寻址。

每一个特殊功能寄存器本质上就是一个内存单元，而标识每个内存单元的是内存地址，不容易记忆。

为了便于使用，每个特殊功能寄存器都会起一个名字，在程序设计时，只要引入头文件“ioCC2530.h”，就可以直接使用寄存器的名称访问内存地址了。

```
C:\Program Files (x86)\IAR Systems\Embedded Workbench 6.0 Evaluation\8051\inc\ioCC2530.h
```

该文件描述了所有的 CC2530 硬件信息。

可以在代码中字节包含该头文件

```
#include <ioCC2530.h>
```

CC2530 的通用 I/O 端口相关的常用寄存器有下面 4 个：

- <1> PxSEL：端口功能选择，设置端口是通用 I/O 还是外设功能。
- <2> PxDIR：作为通用 I/O 时，用来设置数据的传输方向。
- <3> PxINP：作为通用输入端口时，选择输入模式是上拉、下拉还是三态。
- <4> Px：数据端口，用来控制端口的输出或获取端口的输入。

三、如何操作硬件？

对于初次接触硬件的朋友，可能会有点不知所措，硬件的学习有自己的一个套路。

1. 用代码操作外设的一般步骤:

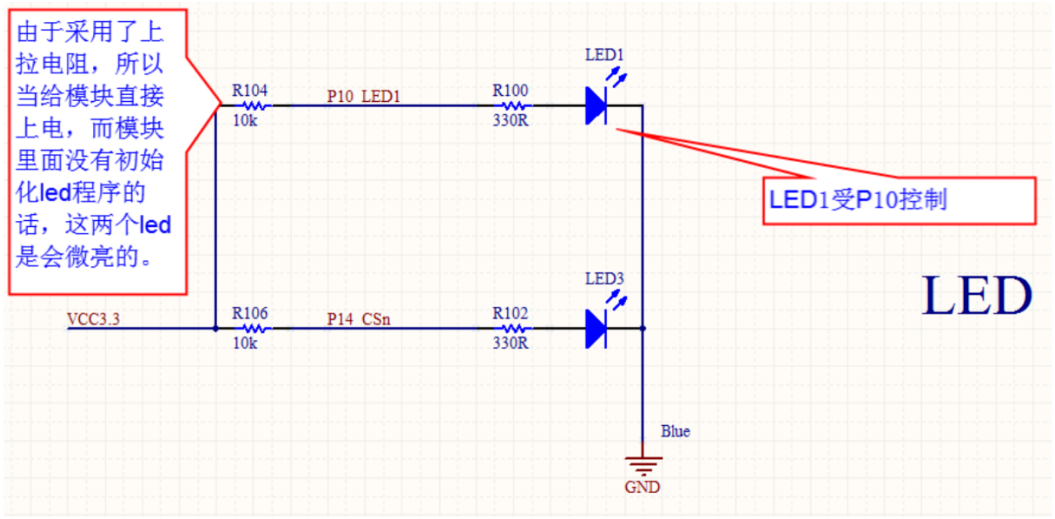
1. 将代码编译成 CPU 能识别的语言
2. cpu 解析执行代码流
3. 然后通过总线找到控制器的寄存器（即 SFR），通过设置这些寄存器，来指挥控制器工作。

2. 编写外设驱动程序的步骤

1. 先看电路图 板子一定会有电路图，就像房子的户型图，涵盖了所有的元器件连接关系， 查看外设数据线、信号线、片选、使能等， 这些线与 cpu 之间连接关系，往往这些外设都会连接某个控制器上，比如:GPIO、中断、SPI、IIC、USB、PCI。
2. 掌握外设连接的控制器原理 查找 soc 的 datasheet， 设置哪些寄存器，才能让外设正确地工作
3. 代码实现 代码必须按照一定顺序操作硬件，初始化->开启->读写->关闭 有的地方需要延时、阻塞，有的需要等待外设发送中断后再去读取。

四、led

1. 电路



LED1、LED3 是发光二极管，
功能比较简单，比如 LED1，只有一个 P10 与 soc 相连，
P10 为 1 的时候， 二极管导通，LED1 亮， P10 为 0 的时候， LED1 灭，
所以要控制 LED1、LED4 亮灭只需要控制 P10、P14 输出电平即可。

2. GPIO

微控制器的大部分 I/O 端口都是功能复用的，在使用的时候需要通过功能选择寄存器来配置端口的功能。
要使用一个 io 口，需要经过如下过程：

1. 配置 PxSEL，选择 io 功能
2. 配置 PxDIR，设置为输入还是输入
3. 如何设置为输入，则需要配置成上拉还是下拉还是高阻态。

P10 引脚对应寄存器操作如下：

1. P1SEL (0xF4) - 端口 1 功能选择

设置 P1 的某个口为 io 口还是外围功能

位	名称	复位	R/W	描述
7:0	SELP1_[7:0]	0x00	R/W	P1.7 到P0.0功能选择 0: 通用I/O 1: 外设功能

芯片 Reset 后，此寄存器默认值是 0x00，也就是 P1 的 8 个 io 口在 Reset 后，默认的设置成了 io 口。

2. P1DIR (0xFE) - 端口 1 方向

当 P1 的某个 io 口，被设置成 io 功能的时候，那么，这个 P1DIR 是把该 io 口设置成输入还是输出的。

位	名称	复位	R/W	描述
7:0	DIRP1_[7:0]	0x00	R/W	P1.7到P1.0的I/O方向 0: 输入 1: 输出

那么我们在做这个 P10 驱动 led 亮灭的实验的时候，需要把 P10 设置为输入，去驱动 led。
并且 P10 有 20ma 的驱动输出能力。

3. P1INP (0xF6) - 端口 1 输入模式

位	名称	复位	R/W	描述
7:2	MDP1_[7:2]	0000 00	R/W	P1.7到P1.2的I/O输入模式 0: 上拉/下拉(见P2INP (0xF7)–端口2输入模式) 1: 三态
1:0	-	00	R0	不使用

由于设置为输出，所有 P1INP 寄存器就不用设置了，而且 P1INP 中没有对 P10 和 P11 的配置，因为这两个口有超强的驱动能力。

4. P1

设置对应引脚则只需要通过寄存器 P1，操作对应的 bite 即可。

P1 (0x90) - 端口 1

位	名称	复位	R/W	描述
7:0	P1[7:0]	0xFF	R/W	端口1。通用I/O端口。可以从SFR位寻址。该CPU内部寄存器可以从XDATA (0x7090)读，但是不能写。

P10 设置为高、低电平,只需要将 P1 的 bit[0]设置为 1/0 即可。
而每一个 bit，头文件 ioCC2530.h 都已经给出对应的宏定义：

```
133 /* Port 1
134 SFRBIT( P1      ,  0x90, P1_7, P1_6, P1_5, P1_4, P1_3, P1_2, P1_1, P1_0 )
```

参考代码如下:

```
P1_0 = 1;
```

3. 编码

完整代码：

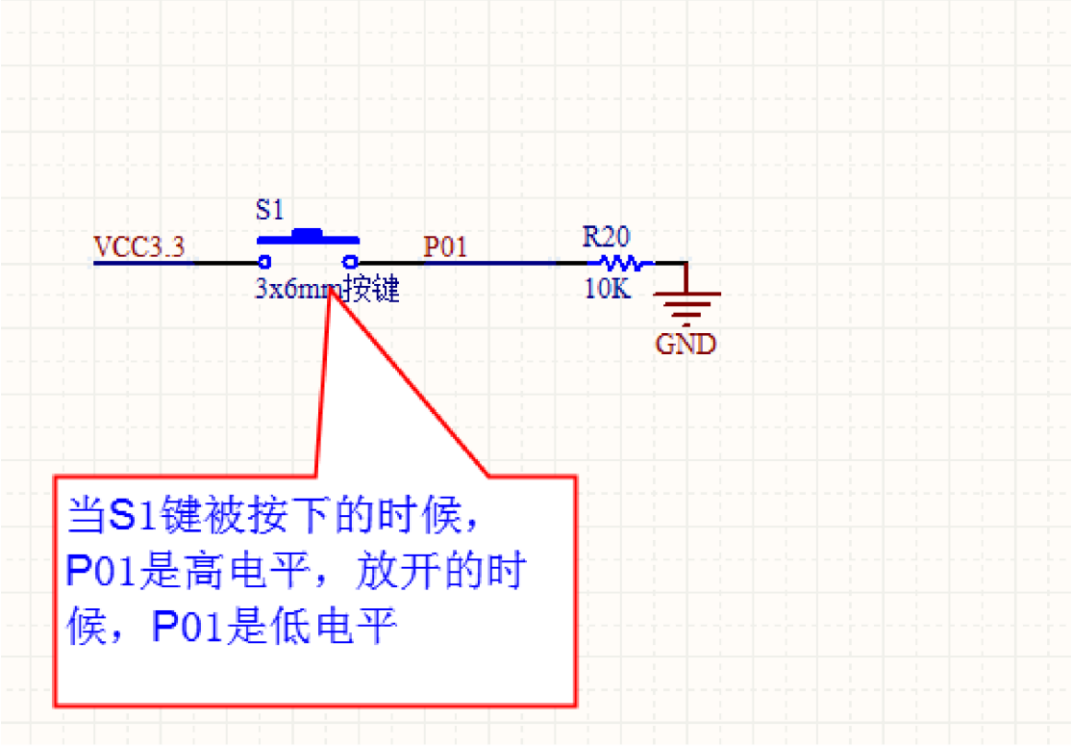
```
/*********************************************************************
* 文 件 名: main.c
* 作    者: Daniel Peng
* 修    订: 2022-4-10
* 版    本: 1.0
* 描    述: GPIO 输出控制实验 1 操作 IO 口控制 LED 灯的亮和灭
*********************************************************************/
#include <ioCC2530.h>

typedef unsigned char uchar;
typedef unsigned int  uint;

#define LED1 P1_0              //定义 P1.0 口为 LED1 控制端

/*********************************************************************
* 名    称: DelayMS()
* 功    能: 以毫秒为单位延时，系统时钟不配置时默认为 16M(用示波器测量相当精确)
* 入口参数: msec 延时参数，值越大，延时越久
* 出口参数: 无
*********************************************************************/
void DelayMS(uint msec)
{
  uint i,j;

  for (i=0; i<msec; i++)
    for (j=0; j<535; j++);
}

/*********************************************************************
* 名    称: InitLed()
* 功    能: 设置 LED 灯相应的 IO 口
* 入口参数: 无
* 出口参数: 无
*********************************************************************/
void InitLed(void)
{
  P1SEL &= 0xFE;            //P1.0 定义为 io 口，这么写，就是不改变 P1SEL 的其他 bit 的值，
                            //而只是改变 P1SEL 的 bit0 为 0，从而使 P1。0 为 io 口
  P1DIR |= 0x01;            //P1.0 定义为输出口
                            //由于设置为输出，所有 P1INP 寄存器就不用设置了，而且 P1INP 中没有对 P10 和 P11 的配置，
                            //因为这两个口有超强的驱动能力，
```

```
}

/*******************************************************************************
* 程序入口函数
*******************************************************************************/
void main(void)
{
  InitLed();                //设置 LED 灯相应的 IO 口

  while(1)                  //死循环
  {
    LED1 = 1;               //点亮 LED1
    DelayMS(1000);          //延时 1 秒

    LED1 = 0;               //LED1 熄灭
    DelayMS(1000);          //延时 1 秒
  }
}
```

五、key

按照相同的思路分析

1. 电路图



按键 S1 是一个输入设备，按下按键 P01 是高电平，放开时候 P01 是低电平。

2. GPIO

根据第一步分析，使能 S1 按键，操作步骤如下：

1. 配置 P0SEL，设置 P01 为普通 IO 口
2. 配置 P0DIR，设置 P0 口为输入
3. 配置 P0INP，设置 P01 上拉

寄存器设置与 LED 类似，不再截图。
按键值的读取，需要循环检测 P0_1 的值是否为 1，为 1 表示按下。
但是按键由于物理特性，按下瞬间会多次置为 1/0，俗称**抖动**。
所以我们第一次判断出 P0_1 为 1 时，需要给一个合理的延时，
然后再次判断 P0_1 是否为 1，
如果仍然为 1，表示按键按下，
延时时间太短起不到防抖效果，延时时间太长会丢失某次按键操作，
所以具体指需要多次测试得出。

3. 编码

按键初始化：

```
void InitKey(void)
{
    P0SEL &= ~0x02;      //设置P0.1为普通IO口
    P0DIR &= ~0x02;      //按键接在P0.1口上，设P0.1为输入模式
    P0INP &= ~0x02;      //打开P0.1上拉电阻
}
```

读取按键值：

```
void InitKey(void)
{
    P0SEL &= ~0x02;      //设置P0.1为普通IO口
    P0DIR &= ~0x02;      //按键接在P0.1口上，设P0.1为输入模式
    P0INP &= ~0x02;      //打开P0.1上拉电阻
}
uchar KeyScan(void)
{
    if (KEY1 == 1)
    {
        DelayMS(100);
        if (KEY1 == 1)
        {
            while(!KEY1); //松手检测
            return 1;     //有按键按下
        }
    }
    return 0;             //无按键按下
}
void main(void)
{
    InitLed();        //设置LED1相应的IO口
    InitKey();            //设置S1相应的IO口

    while(1)
    {
        if (KeyScan())    //按键按下则改变LED状态
            LED1 = ~LED1;
    }
}
```