
重庆邮电大学

物联网专业实验指导书

编制单位：自动化学院

编制人：付蔚

编制时间：2017 年 7 月

前言

一、实验课程的目的

物联网专业实验是物联网工程专业一门重要的实验教学课程。通过物联网专业实验课程的学习，使学生了解智能家居模型概念，并学会对物联网基础知识的初步应用，从而达到培养在实践中学习和应用知识的能力及创新意识，对学生进行综合工程素质教育和训练。

二、实验课前的准备工作

1. 预习好实验指导书，明确实验目的及要求，弄清实验原理，了解实验步骤和注意事项，做到对实验有一个概略性的认识。

2. 准备好实验指导书中规定自带的工具、纸张等。

3. 完成实验指导书规定在实验前应做完的工作。

三、遵守实验室的规章制度

1. 实验前必须了解实验设备、仪器的性能及使用操作规程，否则不得操作。

2. 严格按照规定，精心操作设备、仪器，如有损坏，按规定处理。

3. 实验室内与本实验无关的仪器设备，一律不得动用。

4. 在实验室内严守纪律，不得高声喧哗，保持室内整洁安静。

5. 实验完毕后，用过的仪器设备均应放回原处，并整理清洁，经教师同意后方可离开。

四、实验报告要求

实验报告是对实验过程进行描述、对实验数据和现象进行整理、分析并得出一定结论与看法的书面文件。学生在实验后必须按照要求，整理实验数据、绘制实验曲线、分析实验结果，写出正规的实验报告。

为了写好实验报告，应注意以下几点：

1. 实验结果记录应经实验指导教师过目签字。

2. 实验报告应用专门印制的实验报告纸，且用钢笔或圆珠笔书写，字体应工整，文字要简炼、通顺。

3. 报告中的结果分析及讨论应针对本实验的具体情况，防止不切实际的空谈。

4. 实验报告应在实验完毕后，由课代表统一汇集交给老师。

自动化与电气工程实验实训中心

2017 年 7 月

目 录

实验一：基础区-光敏传感器采集实验.....	7
1.1 实验目的	7
1.2 实验环境	7
1.3 实验原理	7
1.4 实验步骤	11
1.5 实验结果	12
实验二：基础区-人体红外传感器采集实验.....	14
2.1 实验目的	14
2.2 实验环境	14
2.3 实验原理	14
2.4 实验步骤	17
2.5 实验结果	18
实验三：基础区-震动传感器采集实验.....	20
3.1 实验目的	20
3.2 实验环境	20
3.3 实验原理	20
3.4 实验步骤	23
3.5 实验结果	24
实验四：基础区-可燃气体传感器采集实验.....	26
4.1 实验目的	26
4.2 实验环境	26
4.3 实验原理	26
4.4 实验步骤	30
4.5 实验结果	31
实验五：基础区-温湿度传感器采集实验.....	33
5.1 实验目的	33
5.2 实验环境	33
5.3 实验原理	33

5.4 实验步骤	41
5.5 实验结果	43
实验六：基础区-多彩 LED 控制实验	44
6.1 实验目的	44
6.2 实验环境	44
6.3 实验原理	44
6.4 实验步骤	47
6.5 实验结果	49
实验七：基础区-风扇控制实验	50
7.1 实验目的	50
7.2 实验环境	50
7.3 实验原理	50
7.4 实验步骤	53
7.5 实验结果	55
实验八：基础区-步进电机控制实验	57
8.1 实验目的	57
8.2 实验环境	57
8.3 实验原理	57
8.4 实验步骤	62
8.5 实验结果	64
实验九：基础区-I/O 口控制 LED 实验	65
9.1 实验目的	65
9.2 实验环境	65
9.3 实验原理	65
9.4 实验步骤	68
9.5 实验结果	69
实验十：基础区-Uart 串口通信实验	70
10.1 实验目的	70
10.2 实验环境	70
10.3 实验原理	70

10.4 实验步骤	75
10.5 实验结果	75
实验十一：基础区-无线自组织网络串口通信	77
11.1 实验目的	77
11.2 实验环境	77
11.3 实验原理	77
11.4 实验步骤	83
11.5 实验结果	83
实验十二：家居区-热释电传感器采集实验	85
12.1 实验目的	85
12.2 实验环境	85
12.3 实验原理	85
12.4 实验步骤	91
12.5 实验结果	92
实验十三：家居区-光照度传感器采集实验	93
13.1 实验目的	93
13.2 实验环境	93
13.3 实验原理	93
硬件原理	94
编程思路	95
源码解析	96
13.4 实验步骤	98
13.5 实验结果	99
实验十四：家居区-红外转发器应用实验	100
14.1 实验目的	100
14.2 实验环境	100
14.3 实验原理	100
14.4 实验步骤	105
14.5 实验结果	105
实验十五：家居区-智能窗帘控制实验	107

15.1 实验目的	107
15.2 实验环境	107
15.3 实验原理	107
15.4 实验步骤	112
15.5 实验结果	113
实验十六：家居区-智能门禁控制实验.....	114
16.1 实验目的	114
16.2 实验环境	114
16.3 实验原理	114
16.4 实验步骤	119
16.5 实验结果	120

实验一：基础区-光敏传感器采集实验

1.1 实验目的

1. 掌握光敏电阻的工作原理及应用电路；
2. 了解光敏传感器模块的工作原理；
3. 通过 STM32 采集光敏电阻的分压值，并通过串口显示出来。

1.2 实验环境

1. 硬件：1 个光敏传感器模块、1 个 ST-Link 调试器、2 根 USB2.0 方口线、1 根 USB3.0 数据线、1 个 RJ11 线，1 台 PC 机；
2. 软件：Windows 7/XP、MDK 集成开发环境、串口调试器。

1.3 实验原理



图 0.1 光敏传感器模块

光敏电阻简介

光敏传感器模块的核心采集部件为光敏电阻，如下图所示。光敏电阻在入射光强时阻值变小，入射光弱时阻值变大。只要人眼可感受的光的变化，都会引起它的阻值变化。



图 0.1 光敏电阻

光敏电阻的工作原理

在光敏电阻两端的金属电极加上电压，其中便有电流通过；当受到一定波长的光线照射时，电流就会随光强的增大而变大，从而实现光电转换。光敏电阻没有极性，是一个纯粹的电阻元件，使用时既可加直流电压，也可加交流电压，本节实验使用的是直流电压。

电路分析

本节实验中，用到了光敏传感器模块上的 Status 和 User1 指示灯，由【配套光盘\01-文档资料\01-原理图\03-传感器模块\02-光敏传感器】目录中的原理图文件“光敏传感器底板.pdf”，可以知道这两个指示灯的控制引脚配置如表 0.1 LED 引脚配置所示。

表 0.1 LED 引脚配置

名称	MCU 控制引脚	定义
Status	PB9	模块上电后点亮
User1	PB4	串口上传数据时闪烁

光敏电阻与 STM32 之间的接口电路如下图所示。光敏电阻的一端接 GND，另一端接到 STM32 的 PA4 引脚和一个 10kΩ 的电阻，电阻的另一端接 5V。由【配套光盘\01-文档资料\02-数据手册\02-STM32 相关资料\STM32F103x46 参考手册详细简介中文版.pdf】文件中的第 3 章引脚定义可知，PA4 可配置为外设模/数转换器 ADC1 或 ADC2 的通道 4。随着光敏电阻阻值的变化，形成了一个光敏电阻和 10kΩ 电阻之间电压的分配变化，所以产生了一个模拟变量，再通过 PA4 输入到 STM32 中。

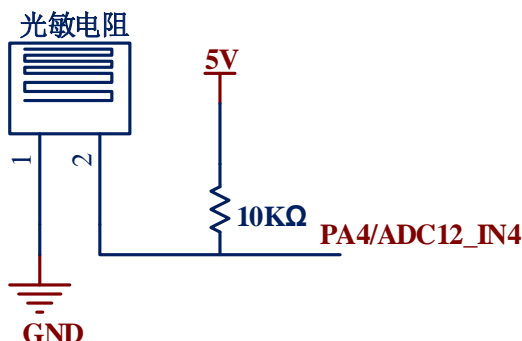


图 0.1 光敏电阻接口电路

当光照度变强时，光敏电阻的阻值会变小，导致光敏电阻的分压降低。相反，当光照度变弱时，光敏电阻的阻值会变大，导致光敏电阻的分压升高。这样，通过光敏电阻阻值的变化影响输入到 STM32 中的模拟量信号的变化，来达到检测光照强度变化的目的。

程序流程

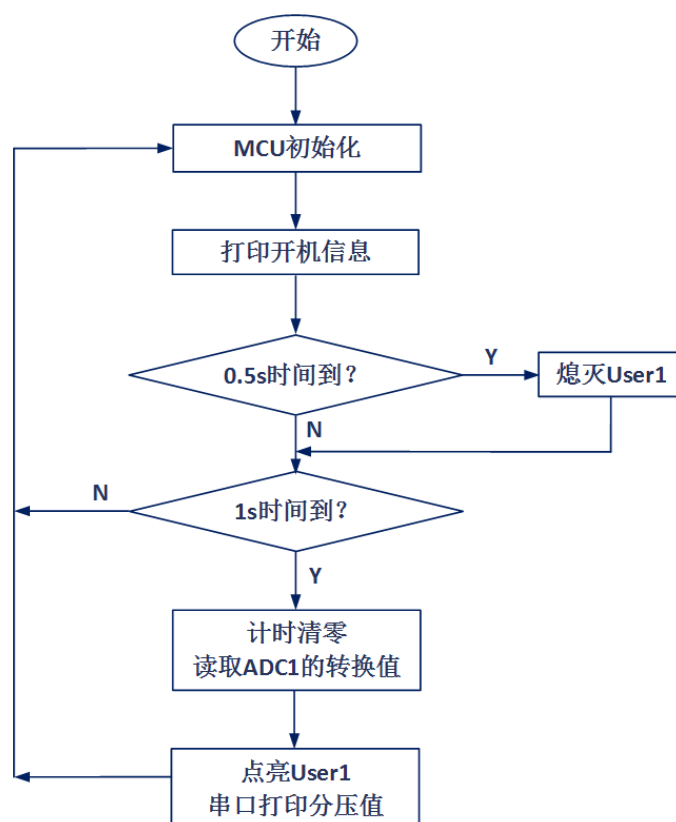


图 0.1 光敏传感器流程图

源码解析

本节实验的源码文件在【配套光盘\04-实验例程\01-物联网感知与控制技术\03-第三章传感器与数据采集实验\3.3 光敏传感器采集实验\photosensitive\USER】目录下，主要包括和ADC采样配置与转换有关的文件“sensor.c”以及主程序文件“main.c”。下面具体解析这些文件中的主要函数。

1. MCU 初始化

函数 McuInit()主要包括系统时钟和外设时钟的初始化配置、LED 的初始化配置、GPIO 的初始化配置、ADC1 的初始化配置、USART2 的初始化配置，以保证外设、LED、ADC 采样，以及 USART1 能够正常工作，具体代码请查看本节实验例程。

```

void McuInit(void)
{
    RCC_Configuration();           // 时钟配置;
    LedGpioInit('B',4,5,8,9);      // LED 灯初始化
    Sm_GPIOInit(4);                // 将 PA4 配置为 ADC1 的通道
4
    SmAdcInit(4);                  // ADC 采样初始化
    UartInit(2,115200);             // USART2 串口通信配置
}
  
```

2. ADC1 采样初始化

由图 0.1 光敏电阻接口电路知，STM32 是通过 PA4 对光敏电阻分压进行采集的，关于 ADC 初始化配置的具体代码如下。

```
void SmAdcInit(uint8_t channel)
```

```

{
    ADC_InitTypeDef ADC_InitStruct;

    ADC_DeInit(ADC1); // 复位 ADC1
    ADC_InitStruct.ADC_Mode = ADC_Mode_Independent; // ADC1 工作在独立模式
    ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right; // ADC 数据右对齐
    ADC_InitStruct.ADC_ContinuousConvMode = DISABLE; // 禁止连续转换
    ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; // 由软件触发模数转换
    ADC_InitStruct.ADC_NbrOfChannel = channel; // 选择通道号
    ADC_InitStruct.ADC_ScanConvMode = DISABLE; // 采用单通道模式
    ADC_Init(ADC1, &ADC_InitStruct); // 根据结构体
ADC_InitStruct 配置 ADC1

    ADC_RegularChannelConfig(ADC1, channel, 1, ADC_SampleTime_71Cycles5); // 设置 ADC1 的指定通道的转
化顺序和采样时间
    ADC_Cmd(ADC1, ENABLE); // 使能 ADC1
    ADC_ResetCalibration(ADC1); // 重置 ADC1 的校准寄存
器
    while(ADC_GetResetCalibrationStatus(ADC1)); // 等待重置完成;
    ADC_StartCalibration(ADC1); // 开启 ADC1 的校准状态
    while(ADC_GetCalibrationStatus(ADC1)); // 等待 ADC1 校准完成
    ADC_SoftwareStartConvCmd(ADC1, ENABLE); // 使能 ADC1 的软件转换
启动功能
}

```

3. 读取 ADC1 采样值

读取 ADC1 的采样值时,首先启动 ADC 转换,等待转换完成后即可读取,具体代码如下。该函数中读取的是数字量信号,在主函数中将被转换为电压信号。

```

uint16_t read_ADC(uint8_t channel)
{
    uint16_t temp = 0;

    ADC_SoftwareStartConvCmd(ADC1, ENABLE); // 启动 ADC 转换
    while(! ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC)); // 等待 ADC 转换完成
    temp = ADC_GetConversionValue(ADC1); // 读取 ADC 的值
    return (temp);
}

```

4. 主函数 main ()

主函数主要实现了以下功能:

- 1) MCU 初始化: 时钟、LED、ADC1、USART1 的初始化配置。
- 2) 打印开机信息: 系统初始化完成后通过串口打印开机信息。
- 3) 计算并上传光敏电阻的分压值。
- 4) 控制 USART1 发送指示灯的闪烁。

具体代码如下。

```

int main(void)
{
    McuInit();
    GPIO_SetBits(GPIOB, GPIO_Pin_0); //设置 485 控制引脚为高电平, 设置 485 串口为发送
模式
    printf("\r\n 光敏传感器采集实验\r\n");
    while(1)
    {
        if(delaytime == 500000) // 0.5s 时间到
            LedOff(4); // 熄灭 User1
    }
}

```

```

        if((delaytime++) >= 1000000)                // 1s 时间到
        {
            delaytime = 0;                          // 计时清零
            value = read_ADC(4) / 4096.0 * 3.3;      // 读取转换的 AD 值
            LedOn(4);                                // 点亮 User1
            GPIO_SetBits(GPIOB, GPIO_Pin_0); //设置 485 控制引脚为高电平，设置 485
串口为发送模式

            printf("光敏电阻分压: %0.1f V\r\n",value); // 通过串口 2 上传
        }
    }
}

```

1.4 实验步骤

1. 将 USB3.0 数据线的一端连接光敏传感器模块的 USB3.0 调试烧写口，另一端连接 ST-Link 调试器的“Debug”接口。
2. 将第 1 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-Debug”接口。
3. 将第 2 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-485”接口。
4. 将“RJ11”线的一端连接光敏传感器的“RJ11”的口，另一端连接 ST-Link 调试器的“RS-485”接口，连接正确后效果如下图所示。




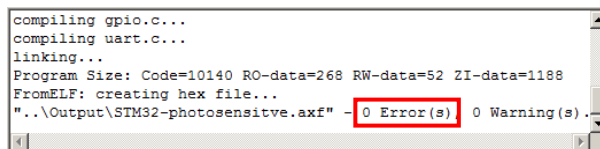
图 0.1 光敏传感器模块接线示意图

5. 双击打开【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】目录下的串口调试器，选择正确的端口号（可参照 1.2.2 查看串口端号），波特率设为 115200，其他均保持默认设置，效果见下图。点击按钮 **打开串口** 打开串口即可。




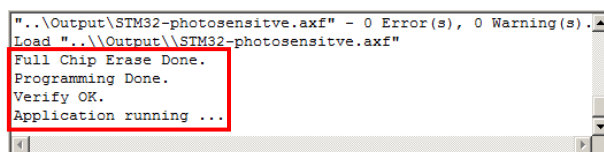
图 0.2 串口调试器配置

6. 在【配套光盘\04-实验例程\第 3 章基础技术实验区\3.1 光敏传感器采集实验\photosensitive\USER】目录下，双击打开工程文件“photosensitive.uvproj”。
7. 在工具栏中点击按钮，编译工程成功后，信息框会出现下图所示的信息。如果编译失败，请参照 1.2.1.2 中的内容，确认相关选项是否配置正确。



```
compiling gpio.c...
compiling uart.c...
linking...
Program Size: Code=10140 RO-data=268 RW-data=52 ZI-data=1188
FromELF: creating hex file...
"..\\Output\\STM32-photosensitive.axf" - 0 Error(s), 0 Warning(s)...
```

8. 参照 1.2.1.3 中的内容，确认与硬件调试有关的选项已设置正确。如果检测不到硬件，请参照 1.1.3.1 中的内容检查 ST-Link 驱动是否正确安装。
9. 点击按钮, 将程序下载到光敏传感器模块中。下载成功后，如果信息框显示下图所示的信息，表明程序下载成功并已自动运行。



```
"..\\Output\\STM32-photosensitive.axf" - 0 Error(s), 0 Warning(s)..  
Load "..\\Output\\STM32-photosensitive.axf"  
Full Chip Erase Done.  
Programming Done.  
Verify OK.  
Application running ...
```

10. 程序运行起来后，Status 指示灯被点亮，User1 指示灯开始每隔 0.5s 闪烁一次。
11. 观察串口调试器接收区接收到的数据，按一下光敏传感器模块的复位按钮重启模块，再观察串口调试器接收区接收到的数据。
12. 用手遮挡住光敏电阻，观察串口调试器接收区接收到的数据变化。

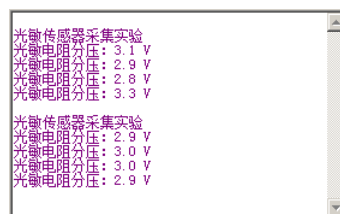
1.5 实验结果

1. 程序运行起来后，光敏传感器模块上的指示灯变化如下图所示。

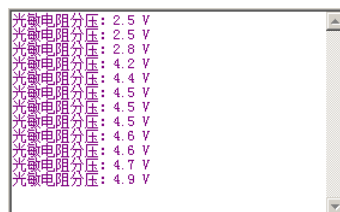


图 0.1 指示灯变化

2. 程序运行起来后，光敏传感器模块上传开机信息及光敏电阻分压，重启后，重新上传开机信息，并实时上传光敏电阻分压，如下图所示。



3. 用手逐渐去遮挡住光敏电阻时，串口调试器接收区接收到的数据如下图所示。可见，光照度变弱时，光敏电阻的阻值变大，分压变大。



实验二：基础区-人体红外传感器采集实验

2.1 实验目的

1. 了解 PM-7 人体红外模块的工作原理；
2. 通过 STM32 采集人体红外模块的输出信号，并通过串口显示在检测范围内是否有人。

2.2 实验环境

1. 硬件：1 个人体红外传感器模块、1 个 ST-Link 调试器、2 根 USB2.0 方口线、1 根 USB3.0 数据线、1 根 RJ11 线，1 台 PC 机；
2. 软件：Windows 7/XP、MDK 集成开发环境、串口调试器。

2.3 实验原理



图 0.1 人体红外传感器模块

PM-7 模块

人体红外传感器模块的核心采集部件为 PM-7 人体红外模块，如下图所示。

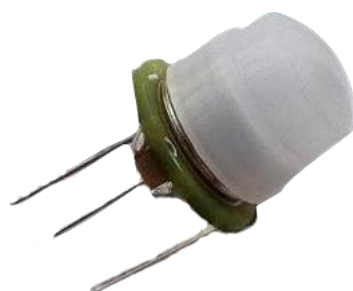


图 0.1 PM-7 模块

人体会发射 $10\mu\text{m}$ 左右的特定波长的红外线，人体红外模块可以针对性地检测这种红外线的存在与否。当人体红外线照射到传感器上后，该模块将输出高电平，平时为低电平。

电路分析

本节实验中，用到了人体红外传感器模块上的 Status、RS485-T 和 User1 指示灯，由【配套光盘\01-文档资料\01-原理图\03-传感器模块\14-人体红外传感器】目录中的原理图文件“人体红外传感器底板.pdf”，可以知道这三个指示灯的控制引脚配置如下表所示。

表 0.1 LED 引脚配置

名称	MCU 控制引脚	定义
Status	PB4	模块上电后点亮
RS485-T	PB6	串口上传数据时闪烁
User1	PB7	检测到有人时点亮

人体红外模块与 STM32 的接口电路如下图所示。

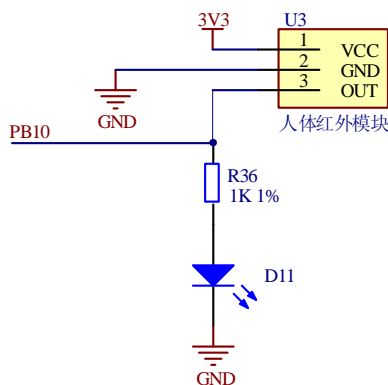


图 0.1 人体红外模块接口电路

当人体红外模块周围没人时，输出低电平；当人体红外模块周围有人时，输出高电平，并通过 PB10 输入到 STM32 中。

程序流程

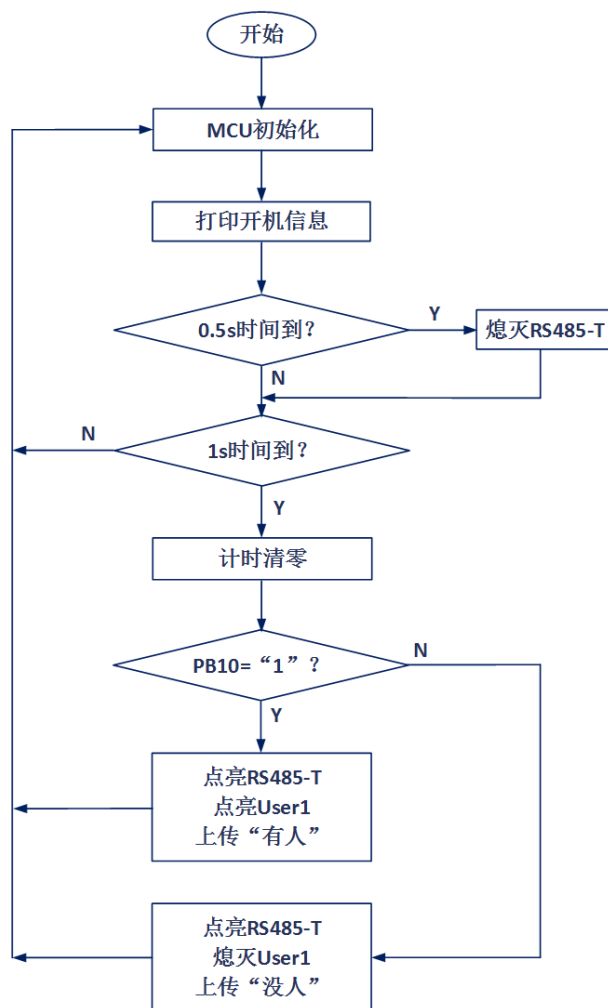


图 0.1 人体红外传感器流程图

源码解析

本节实验的源码文件在【配套光盘\04-实验例程\01-物联网感知与控制技术\03-第三章传感器与数据采集实验\3.16 人体红外传感器采集实验\body_infrare\USER】目录下，本节实验主要是通过检测 PB10 是高电平还是低电平，来查询在检测范围内是否有人，具体代码如下。

```
int main(void)
{
    McuInit(); // MCU 初始化
    GPIO_SetBits(GPIOA, GPIO_Pin_1); // 设置 485 串口为发送模式
    printf("\r\n 人体红外传感器采集实验\r\n"); // 打印开机信息
    while(1)
    {
        if(delaytime== 500000) // 0.5s 时间到
            LedOff(6); // 熄灭 RS485-T
        if((delaytime++) >= 1000000) // 1s 时间到
        {
            delaytime = 0; // 计时清零
            LedOn(6); // 点亮 RS485-T
            if(GetInput(10) == 1) // 查询 PB10 状态
```



```
{
    LedOn(7);                                // 点亮 User1
    GPIO_SetBits(GPIOA, GPIO_Pin_1);         //设置 485 串口为发送模式
    printf("有人\r\n");                       // 通过串口 2 上传数据
}
else
{
    LedOff(7);                                // 熄灭 User1
    GPIO_SetBits(GPIOA, GPIO_Pin_1);         //设置 485 串口为发送模式
    printf("没人\r\n");                       // 通过串口 2 上传数据
}
}
```

2.4 实验步骤

1. 将 USB3.0 数据线的一端连接人体红外传感器模块的 USB3.0 调试烧写口，另一端连接 ST-Link 调试器的“Debug”接口。
2. 将第 1 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-Debug”接口。
3. 将第 2 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-485”接口。
4. 将“RJ11”线的一端连接人体红外传感器的“RJ11”口，另一端连接 ST-Link 调试器的“RS-485”接口，连接正确后效果如下图所示。



图 0.1 人体红外传感器模块接线示意图



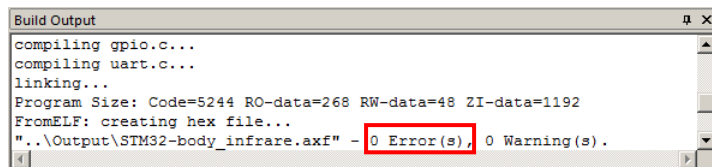

5. 双击打开【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】目录下的串口调试器,选择正确的端口号(可参照 1.2.2 节查看串口端号),波特率设为 115200,其他均保持默认设置,效果见下图。点击按钮  打开串口即可。

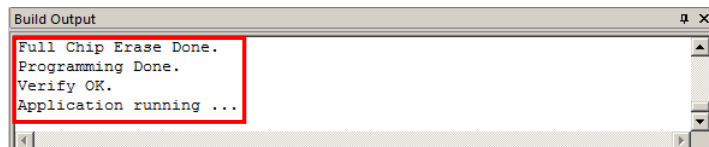


图 0.2 串口调试器配置

- 在【配套光盘\04-实验例程\第 3 章基础技术实验区\3.2 人体红外传感器采集实验\body_infrare\USER】目录下，双击打开工程文件“body_infrare.uvproj”。
- 在工具栏中点击按钮，编译工程成功后，信息框会出现下图所示的信息。如果编译失败，请参照 1.2.1.2 节中的内容，确认相关选项是否配置正确。



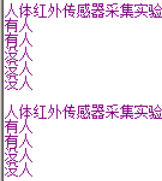
- 参照 1.2.1.3 节中的内容，确认与硬件调试有关的选项已设置正确。如果检测不到硬件，请参照 1.1.3.1 节中的内容检查 ST-Link 驱动是否正确安装。
- 点击按钮，将程序下载到人体红外传感器模块中。下载成功后，如果信息框显示下图中的信息，表明程序下载成功并已自动运行。



- 程序运行起来后，Status 指示灯被点亮，RS485-T 指示灯开始每隔 0.5s 闪烁一次。
- 观察串口调试器接收区接收到的数据，按一下人体红外传感器模块的复位按钮重启模块，再观察串口调试器接收区接收到的数据。
- 用手靠近人体红外模块，观察 User1 指示灯的变化，以及串口调试器接收区接收到的数据变化。

2.5 实验结果

- 程序运行起来后，人体红外传感器模块上传开机信息及检测范围内是否有人，重启后，重新上传开机信息，并实时上传检测范围内是否有人，如下图所示。



2. 用手靠近人体红外模块时，串口调试器接收区接收到的数据如下图所示，可以观察到，该模块在检测范围内检测到了人。此时，“User1”指示灯被点亮。



实验三：基础区-震动传感器采集实验

3.1 实验目的

1. 了解 TTP223 震动传感器的工作原理；
2. 通过 STM32 采集震动传感器的输出信号，并通过串口显示是否检测到震动。

3.2 实验环境

1. 硬件：1 个震动传感器模块、1 个 ST-Link 调试器、2 根 USB2.0 方口线、1 根 USB3.0 数据线、1 跟 RJ11 线，1 台 PC 机；
2. 软件：Windows 7/XP、MDK 集成开发环境、串口调试器。

3.3 实验原理



图 0.1 震动传感器模块

SW-420 震动传感器

震动传感器模块的核心采集部件为 SW-420 震动传感器，如下图所示。



图 0.1 SW-420 震动传感器

该传感器为常闭型 360° 全方位滚珠开关，平时任何角度开关都是接通状态，受到振动或移动时，开关内导通电流的滚轴会产生移动或振动，从而导致通过的电流断开或电阻阻值的升高而触发电路变化。这种开关的特点是平时一般处于导通状态，振动时会短暂断开，所以它的灵敏度很高。

电路分析

本节实验中，用到了震动传感器模块上的 Status、RS485-T 和 User1 指示灯，由【配套光盘\01-文档资料\01-原理图\03-传感器模块\16-震动传感器】目录中的原理图文件“震动传感器底板.pdf”，可以知道这三个指示灯的控制引脚配置如下表所示。

表 0.1 LED 引脚配置

名称	MCU 控制引脚	定义
Status	PB4	模块上电后点亮
RS485-T	PB6	串口上传数据时闪烁
User1	PB7	检测到震动时点亮

SW-420 震动传感器与 STM32 的接口电路如下图所示。

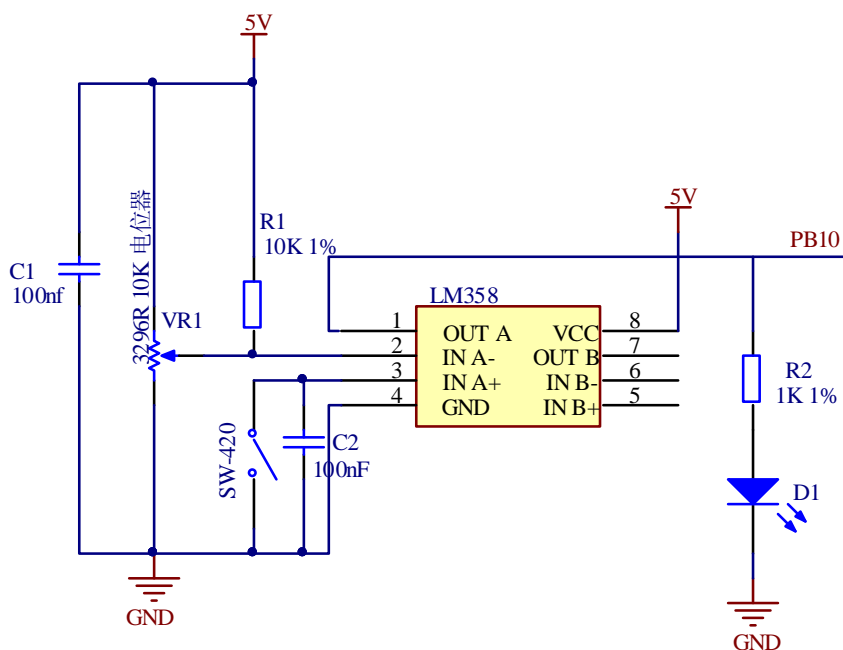


图 0.1 震动传感器接口电路

震动传感器的测量信号，经运放电路传送给 STM32 的 PB10，没有震动时，PB10 为低电平；当传感器感受到震动时，PB10 为高电平。

程序流程

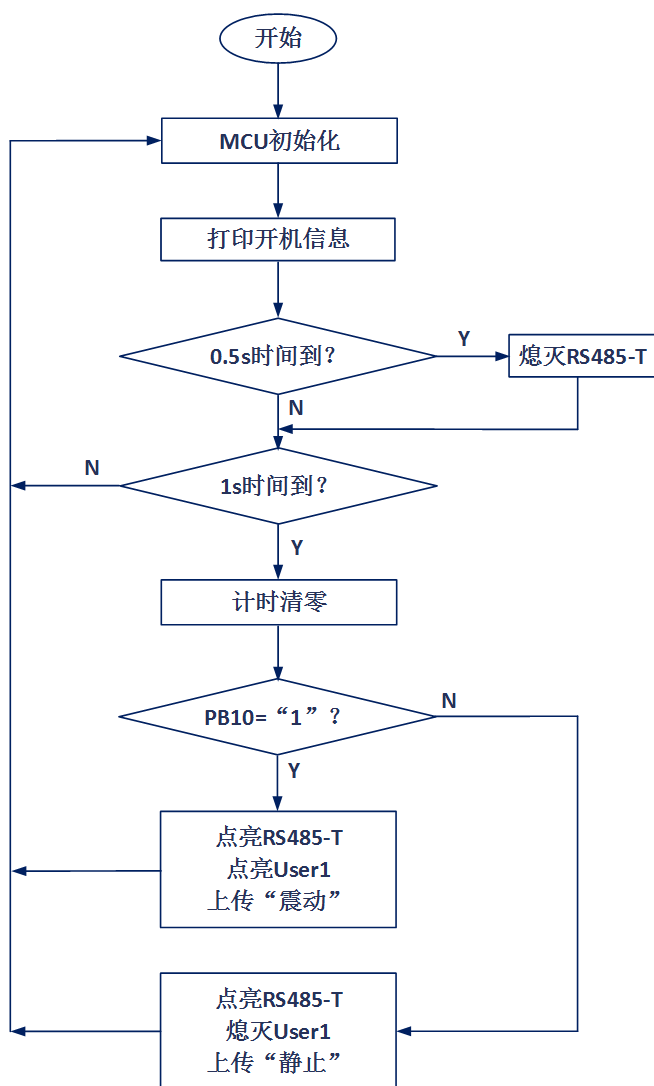


图 0.1 震动传感器流程图

源码解析

本节实验的源码文件在【配套光盘\04-实验例程\01-物联网感知与控制技术\03-第三章传感器与数据采集实验\3.18 震动传感器采集实验\shake\USER】目录下，本节实验主要是通过检测 PB10 是高电平还是低电平，来查询模块是否震动，具体代码如下。

```

int main(void)
{
    McuInit(); // MCU 初始化
    GPIO_SetBits(GPIOA,GPIO_Pin_1); //设置 485 串口为发送模
    printf("\r\n 震动传感器采集实验\r\n"); // 打印开机信息
    while(1)
    {
        if(delaytime== 500000) // 0.5s 时间到
            LedOff(6); // 熄灭 RS485-T
        if((delaytime++) >= 1000000) // 1s 时间到
        {
            delaytime = 0; // 计时清零
            LedOn(6); // 点亮 RS485-T
        }
    }
}

```

```

        if(GetInput(10) == 1)                // 查询 PB10 状态
        {
            LedOn(7);                        // 点亮 User1
            GPIO_SetBits(GPIOA,GPIO_Pin_1); //设置 485 串口为发送模式
            printf("震动\r\n");              // 通过串口 1 上传数据
        }
        else
        {
            LedOff(7);                       // 熄灭 User1
            GPIO_SetBits(GPIOA,GPIO_Pin_1); //设置 485 串口为发送模式
            printf("静止\r\n");
        }
    }
}

```

3.4 实验步骤

1. 将 USB3.0 数据线的一端连接震动传感器模块的 USB3.0 调试烧写口，另一端连接 ST-Link 调试器的“Debug”接口。
2. 将第 1 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-Debug”接口。
3. 将第 2 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-485”接口。
4. 将“RJ11”线的一端连接震动传感器的“RJ11”口，另一端连接 ST-Link 调试器的“RS-485”接口，连接正确后效果如下图所示。



图 0.1 震动传感器模块接线示意图



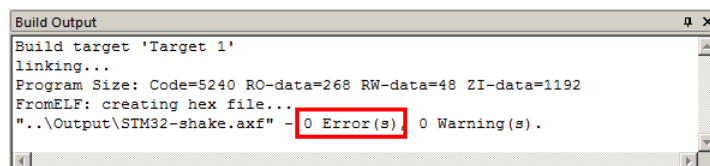

5. 双击打开【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】目录下的串口调试器，选择正确的端口号(可参照 1.2.2 节查看串口端号)，波特率设为 115200，其他均保持默认设置，效果见下图。点击按钮  打开串口即可。

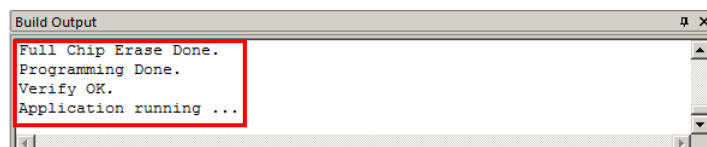


图 0.2 串口调试器配置

6. 在【配套光盘\04-实验例程\第 3 章基础技术实验区\3.5 震动传感器采集实验\shake\USER】目录下，双击打开工程文件“shake.uvproj”。
7. 在工具栏中点击按钮 ，编译工程成功后，信息框会出现下图所示的信息。如果编译失败，请参照 1.2.1.2 节中的内容，确认相关选项是否配置正确。



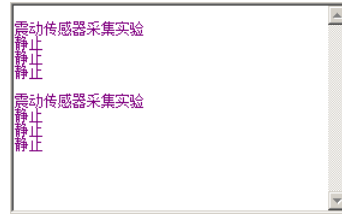
8. 参照 1.2.1.3 节中的内容，确认与硬件调试有关的选项已设置正确。如果检测不到硬件，请参照 1.1.3.1 节中的内容检查 ST-Link 驱动是否正确安装。
9. 点击按钮 ，将程序下载到震动传感器模块中。下载成功后，如果信息框显示下图所示的信息，表明程序下载成功并已自动运行。



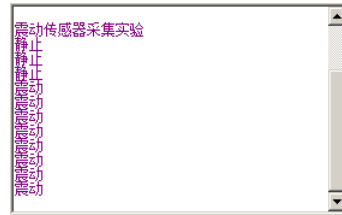
10. 程序运行起来后，Status 指示灯被点亮，RS485-T 指示灯开始每隔 0.5s 闪烁一次。
11. 观察串口调试器接收区接收到的数据，按一下震动传感器模块的复位按钮重启模块，再观察串口调试器接收区接收到的数据。
12. 移动或晃动震动传感器模块，观察 User1 指示灯的变化，以及串口调试器接收区接收到的数据变化。

3.5 实验结果

1. 程序运行起来后，震动传感器模块上传开机信息及是否检测到震动，重启后，重新上传开机信息，并实时上传是否检测到震动，如下图所示。



2. 移动或晃动模块时，串口调试器接收区接收到的数据如下图所示，可以观察到，该模块检测到了震动。此时，“User1”指示灯被点亮。



实验四：基础区-可燃气体传感器采集实验

4.1 实验目的

1. 掌握 TGS2610 传感器的工作原理及应用电路；
2. 了解可燃气体传感器模块的工作原理；
3. 通过 STM32 采集可燃气体传感器的输出信号，并通过串口上传可燃气体浓度。

4.2 实验环境

1. 硬件：1 个可燃气体传感器模块、1 个 ST-Link 调试器、2 根 USB2.0 方口线、1 根 USB3.0 数据线、1 根 RJ11 线，1 台 PC 机；
2. 软件：Windows 7/XP、MDK 集成开发环境、串口调试器。

4.3 实验原理



图 0.1 可燃气体传感器模块

可燃气体传感器

可燃气体传感器模块的核心采集部件为 TGS2610 传感器，如下图所示。



图 0.1 可燃气体传感器

TGS2610 传感器中的气敏电阻在清洁空气中阻值很大。当传感器所处环境中含有可燃气体时，气敏电阻的阻值随空气中可燃气体浓度的升高而减小。使用简单的电路即可将电导率的变化转换为该气体浓度相对应的输出信号。

电路分析

本节实验中，用到了可燃气体传感器模块上的 Status 和 User1 指示灯，由【配套光盘\01-文档资料\01-原理图\03-传感器模块\09-可燃气体传感器】目录中的原理图文件“可燃气体传感器.pdf”，可以知道这两个指示灯的控制引脚配置如下表所示。

表 0.1 LED 引脚配置

名称	MCU 控制引脚	定义
Status	PB9	模块上电后点亮
User1	PB4	串口上传数据时闪烁

TGS2610 传感器与 STM32 之间的接口电路如下图所示。其中，TGS2610 的输出信号通过 R_3 和 R_4 两个 $10k\Omega$ 的电阻分压后，连接到了 STM32 的 PA4 口。随着环境中可燃气体浓度的变化，产生一个变化的模拟量，再通过 PA4 输入到 STM32 中。

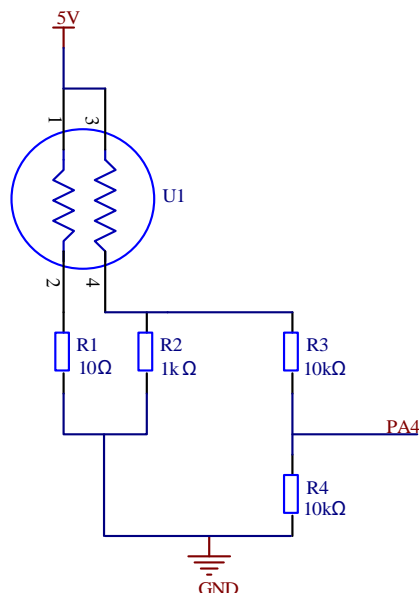


图 0.1 可燃气体传感器接口电路

当环境中的可燃气体浓度升高时，TGS2610 的气敏电阻阻值会减小，导致输出电压值增高。相反，当可燃气体浓度降低时，TGS2610 的气敏电阻阻值会增大，导致输出电压值降低。这样，通过输出电压值的变化影响输入到 STM32 中的模拟量信号的变化，来达到检测可燃气体浓度变化的目的。

程序流程

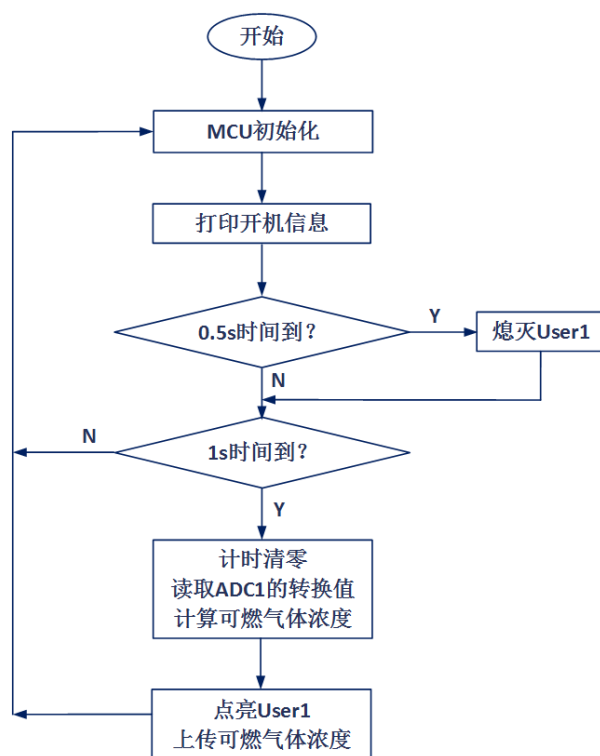


图 0.1 可燃气体传感器流程图

源码解析

本节实验的源码文件在【配套光盘\04-实验例程\01-物联网感知与控制技术\03-第三章传感器与数据采集实验\3.11 可燃气体传感器采集实验\combustible_gas\USER】目录下，主要包括和 ADC 采样配置与转换有关的文件“sensor.c”以及主程序文件“main.c”。下面对这些文件中的主要函数进行具体解析。

1. MCU 初始化

函数 McuInit()主要包括系统时钟和外设时钟的初始化配置、LED 的初始化配置、GPIO 的初始化配置、ADC1 的初始化配置、USART2 的初始化配置，以保证外设、LED、ADC 采样，以及 USART2 能够正常工作，具体代码请查看本节实验例程。

```
void McuInit(void)
{
    RCC_Configuration();           // 时钟配置;
    LedGpioInit('B',4,5,8,9);      // LED 灯初始
    Sm_GPIOInit(4);                // 将 PA4 配置为
    ADC1 的通道 4
    SmAdcInit(4);                  // ADC 采样初
    始化
    UartInit(2,115200);             // USART2 串
    口通信配置
}
```

2. ADC1 采样初始化

由实验一：基础区-光敏传感器采集实验节知，STM32 是通过 PA4 对传感器的输出信号进行采集的，关于 ADC 初始化配置的具体代码如下。

```

void SmAdcInit(uint8_t channel)
{
    ADC_InitTypeDef ADC_InitStruct;

    ADC_DeInit(ADC1); // 复位 ADC1
    ADC_InitStruct.ADC_Mode = ADC_Mode_Independent; // ADC1 工作在独立
模式
    ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right; // ADC 数据右
对齐
    ADC_InitStruct.ADC_ContinuousConvMode = DISABLE; // 禁止连续转换
    ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; // 由软
件触发模数转换
    ADC_InitStruct.ADC_NbrOfChannel = channel; // 选择通道号
    ADC_InitStruct.ADC_ScanConvMode = DISABLE; // 采用单通道模式
    ADC_Init(ADC1, &ADC_InitStruct); // 根据结构体 ADC_InitStruct 配
置 ADC1

    ADC_RegularChannelConfig(ADC1, channel, 1, ADC_SampleTime_71Cycles5);
// 设置 ADC1 的指定通道的转化顺
序和采样时间
    ADC_Cmd(ADC1, ENABLE); // 使能 ADC1
    ADC_ResetCalibration(ADC1); // 重置 ADC1 的校
准寄存器
    while(ADC_GetResetCalibrationStatus(ADC1)); // 等待重置完成;
    ADC_StartCalibration(ADC1); // 开启 ADC1 的校
准状态
    while(ADC_GetCalibrationStatus(ADC1)); // 等待 ADC1 校准
完成
    ADC_SoftwareStartConvCmd(ADC1, ENABLE); // 使能 ADC1 的软
件转换启动功能
}

```

3. 读取 ADC1 采样值

读取 ADC1 的采样值时,首先启动 ADC 转换,等待转换完成后即可读取,具体代码如下。
该函数中读取的是数字量信号,在主函数中将被转换为电压信号。

```

uint16_t read_ADC(uint8_t channel)
{
    uint16_t temp = 0;

    ADC_SoftwareStartConvCmd(ADC1, ENABLE); // 启动 ADC
转换
    while(! ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC)); // 等待 ADC 转换完
成
    temp = ADC_GetConversionValue(ADC1); // 读取 ADC
的值
    return (temp);
}

```

4. 主函数 main ()

主函数主要实现了以下功能:

- 1) MCU 初始化: 时钟、LED、ADC1、USART2 的初始化配置。
- 2) 打印开机信息: 系统初始化完成后通过串口打印开机信息。
- 3) 计算并上传输出电压值。
- 4) 控制 USART2 发送指示灯的闪烁。

具体代码如下。

```

int main(void)
{

```

```

        McuInit(); // 相关硬件初始化
        GPIO_SetBits(GPIOB,GPIO_Pin_0); //485 控制引脚为高电平，发送模
式
        printf("\r\n 可燃气体传感器采集实验\r\n"); //串口 2 打印开机信息
        while(1)
        {
            if(delaytime == 500000) // 0.5s 时间到
                LedOff(4); // 熄灭 User1
            if((delaytime++) >= 1000000) // 1s 时间到
            {
                delaytime = 0; // 计时清零
                value = read_ADC(4) / 4096.0 * 3.3; // 读取转换的 AD 值
                value = value * 4 * 100; // 电压每上升 0.01V，浓度上升
                4ppm
                LedOn(4); // 点亮 User1
                GPIO_SetBits(GPIOB,GPIO_Pin_0); //485 控制引脚为高电平，发送模式
                printf("可燃气体浓度: %d ppm\r\n", (uint16_t)value); // 通过串口 2 上传
                可燃气体浓度
            }
        }
    }
}

```

4.4 实验步骤

1. 将 USB3.0 数据线的一端连接可燃气体传感器模块的 USB3.0 调试烧写口，另一端连接 ST-Link 调试器的“Debug”接口。
2. 将第 1 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-Debug”接口。
3. 将第 2 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-485”接口。
4. 将“RJ11”线的一端连接可燃气体传感器的“RJ11”口，另一端连接 ST-Link 调试器的“RS-485”接口，连接正确后效果如下图所示。



图 0.1 可燃气体传感器模块接线示意图



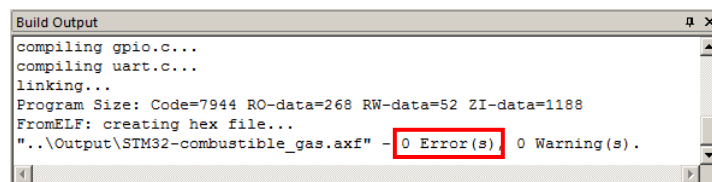

5. 双击打开【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】目录下的串口调试器，选择正确的端口号（可参照 1.2.2 查看串口端号），波特率设为 115200，其他均保持默认设置，效果见下图。点击按钮  打开串口即可。

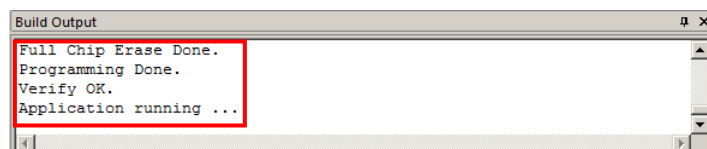


图 0.2 串口调试器配置

6. 在【配套光盘\04-实验例程\第 3 章基础技术实验区\3.4 可燃气体传感器采集实验\combustible_gas\USER】目录下，双击打开工程文件“combustible_gas.uvproj”。
7. 在工具栏中点击按钮 ，编译工程成功后，信息框会出现下图所示的信息。如果编译失败，请参照 1.2.1.2 节中的内容，确认相关选项是否配置正确。



8. 参照 1.2.1.3 节中的内容，确认与硬件调试有关的选项已设置正确。如果检测不到硬件，请参照 1.1.3.1 节中的内容检查 ST-Link 驱动是否正确安装。
9. 点击按钮 ，将程序下载到可燃气体传感器模块中。下载成功后，如果信息框显示下图中所示的信息，表明程序下载成功并已自动运行。



10. 程序运行起来后，Status 指示灯被点亮，User1 指示灯开始每隔 0.5s 闪烁一次，如图 0.1 所示。
11. 观察串口调试器接收区接收到的数据，按一下可燃气体传感器模块的复位按钮重启模块，再观察串口调试器接收区接收到的数据。
12. 用打火机对着传感器探头喷气，观察串口调试器接收区接收到的数据变化。

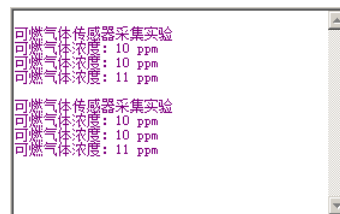
4.5 实验结果

1. 程序运行起来后，可燃气体传感器模块上的指示灯变化如下图所示。

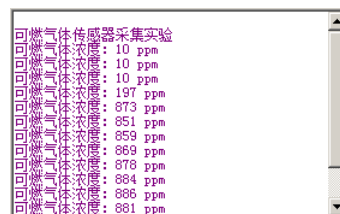


图 0.1 指示灯变化

2. 程序运行起来后，可燃气体传感器模块上传开机信息及可燃气体浓度，重启后，重新上传开机信息，并实时上传可燃气体浓度，如下图所示。



3. 用打火机对着传感器探头喷气时，串口调试器接收区接收到的数据如下图所示，可以观察到，空气中的可燃气体浓度升高了。



实验五：基础区-温湿度传感器采集实验

5.1 实验目的

1. 掌握温湿度传感器 DHT11 的接口原理及典型应用；
2. 熟悉 DHT11 与 STM32 间的通信过程；
3. 了解温湿度传感器模块的工作原理；
4. 通过 STM32 采集传感器数据，并通过串口显示出来。

5.2 实验环境

1. 硬件：1 个温湿度传感器模块、1 个 ST-Link 调试器、2 根 USB2.0 方口线、1 根 USB3.0 数据线、1 个 RJ11 线，1 台 PC 机；
2. 软件：Windows 7/XP、MDK 集成开发环境、串口调试器。

5.3 实验原理

温湿度传感器模块的核心采集部件为 DHT11，它是一款含有已校准数字信号输出的温湿度复合传感器。DHT11 的一体化结构能同时对相对湿度和温度进行测量。测量湿度范围从 20%RH 到 90%RH，测量温度范围从 0℃到 50℃。下面将详细介绍 DHT11 的工作原理。

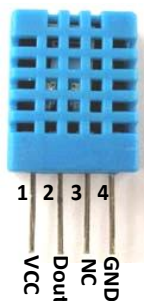


图 0.1 DHT11 温湿度传感器

接口原理

1. 引脚说明

传感器内部包括一个电阻式的感湿元件、一个 NTC 测温元件，和一个高性能的 8 位单片机与这两个元件相连接，外部采用如图 0.1 所示的 4 针单排引脚封装，方便连接。从有气孔的一侧正视 DHT11，从左到右依次为 1、2、3、4 脚，引脚说明如表 0.1 所示。

表 0.1 DHT11 引脚说明

引脚号	引脚名称	类型	引脚说明
Pin 1	VCC	电源	正电源输入，3V-5.5V DC（本节采用 3.3V）
Pin 2	Dout	输出	单总线，数据输入/输出引脚
Pin 3	NC	空脚	空脚，扩展未用

Pin 4	GND	地	电源地
-------	-----	---	-----

2. 典型应用

DHT11 温湿度传感器采用单总线方式与微处理器通信，只需要占用控制器的一个 I/O 口即可完成上下位的连接，典型应用电路如图 0.1 所示。

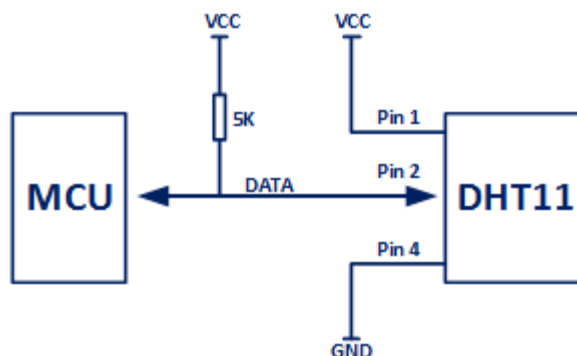


图 0.1 DHT11 典型应用电路

通信原理

空气温湿度传感器模块主要是通过 STM32 与 DHT11 之间的串行通信来获取温湿度数据的，DHT11 采用单总线数据格式，即由单个数据引脚端口完成输入输出双向传输，下面将详细介绍 DHT11 与微处理器间的通信原理。

1. 单总线说明

单总线即只有一根数据线，系统中的数据交换、控制均由单总线完成。在单总线通信中，微处理器与 DHT11 之间是主机与从机的关系。设备（主机或从机）通过一个漏极开路或三态端口连至该数据线，以允许设备在不发送数据时能够释放总线，而让其它设备使用总线；单总线通常要求外接一个约 $5.1k\Omega$ 的上拉电阻，这样，当总线闲置时，其状态为高电平。由于它们是主从关系，只有主机呼叫从机时，从机才能应答，因此主机访问器件都必须严格遵循单总线序列，如果出现序列混乱，器件将不响应主机。

2. 数据时序图

主机(本模块采用 STM32)发送一次开始信号后，DHT11 从低功耗模式转换到高速模式，待 STM32 开始信号结束后，DHT11 发送响应信号，然后送出 40bit 的数据，并触发一次信号采集。整个过程中 DATA 引脚的信号变化如图 0.1 所示。

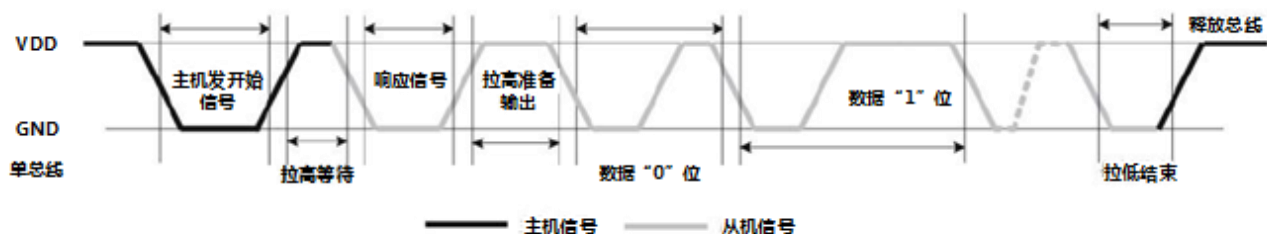


图 0.1 读 DHT11 数据时序图

3. 外设读取步骤

主机和从机之间的通信可通过如下几个步骤完成（STM32 读取 DHT11 的数据的步骤）。

步骤一：

DHT11 上电后（DHT11 上电后要等待 1S 以越过不稳定状态，在此期间不能发送任何指令），测试环境中的温湿度，并记录数据，同时 DHT11 的 DATA 数据线由上拉电阻拉高并一直保持高电平，此时 DHT11 的 DATA 引脚处于输入状态，时刻检测外部信号。

步骤二：

STM32 中与 DATA 脚连接的 I/O 引脚设置为输出，同时输出低电平，且低电平保持时间不能小于 18ms。然后将该 I/O 引脚设置为输入状态释放总线，由于上拉电阻的影响，该 I/O 口即 DHT11 的 DATA 数据线也随之变高，等待 DHT11 作出回答信号。在这个过程中 I/O 引脚的信号变化如图 0.2 所示。

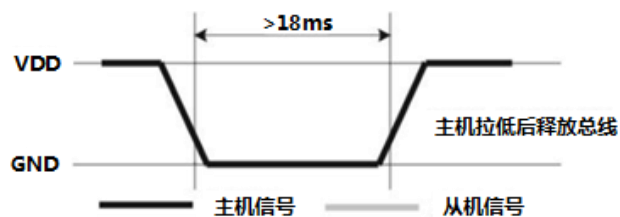


图 0.2 主机发送起始信号

步骤三：

DHT11 的 DATA 引脚检测到外部信号有低电平时，等待外部信号低电平结束，延迟后 DHT11 的 DATA 引脚处于输出状态，输出 80 微秒的低电平作为应答信号，紧接着输出 80 微秒的高电平通知主机准备接收数据，此时 STM32 的 I/O 引脚处于输入状态，STM32 检测到低电平（DHT11 回应信号）后，等待 80 微秒的高电平后的数据接收，DATA 引脚的信号变化如图 0.3 所示。

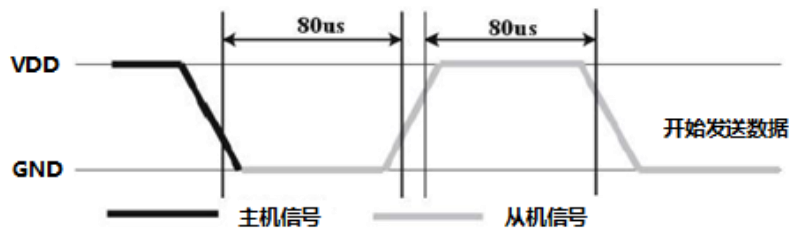


图 0.3 DHT11 响应起始信号

步骤四：

由 DHT11 的 DATA 引脚输出 40 位数据，STM32 根据 I/O 电平的变化接收 40 位数据，位数据“0”的格式为：50 微秒的低电平和 26-28 微秒的高电平，位数据“1”的格式为：50 微秒的低电平加 70 微秒的高电平。位数据“0”、“1”格式信号分别如图 0.4 和图 0.5 所示。

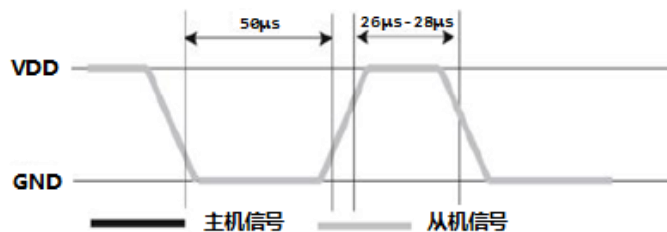


图 0.4 数字 0 信号表示方法

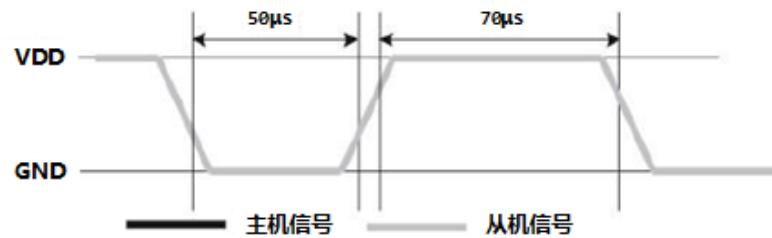


图 0.5 数字 1 信号表示方法

4. 单总线传送数据位定义

DHT11 通过 DATA 引脚与 STM32 进行通讯，在发送温湿度数据时一次传送 40bit 数据，高位先出。

数据格式：8bit 湿度整数数据+8bit 湿度小数数据+8bit 温度整数数据+8bit 温度小数数据+8bit 校验位。

注：其中，小数部分用于 DHT11 以后的扩展，现读出为零。

5. 校验位数据定义

“8bit 湿度整数数据+8bit 湿度小数数据+8bit 温度整数数据+8bit 温度小数数据”，8bit 校验位等于所得结果的最后一个字节。

示例一：

接收到的 40bit 数据为：

0011 0101 0000 0000 0001 1000 0000 0000 0100 1101

湿度高 8bit 湿度低 8bit 温度高 8bit 温度低 8bit 校验位

计算：0011 0101+0000 0000+0001 1000+0000 0000= 0100 1101。

接收数据正确。

湿度：0011 0101=35H=53%RH；

温度：0001 1000=18H=24℃。

示例二：

接收到的 40bit 数据为：

0011 0101 0000 0000 0001 1000 0000 0000 0100 1001

湿度高 8bit 湿度低 8bit 温度高 8bit 温度低 8bit 校验位

计算：0011 0101+0000 0000+0001 1000+0000 0000 = 0100 1101。

01001101 不等于 01001001。

本次接收的数据不正确，放弃，重新接收数据。



图 0.6 温湿度传感器模块

电路分析与程序配置

本节实验中，用到了温湿度传感器模块上的两个指示灯，指示灯定义及控制引脚如下表所示。

表 0.1 LED 引脚配置

名称	MCU 控制引脚	定义
Status	PB4	模块上电后点亮
User1	PB7	串口上传数据时闪烁

DHT11 与 STM32 间的接口电路如下图所示，PB10 用于 STM32 与 DHT11 之间的通信和同步，采用单总线数据格式，一次通讯时间 4ms 左右，数据分小数部分和整数部分，具体格式在 0 中已说明，当前小数部分用于以后扩展，现读出为零。STM32 采集到温湿度数据后，再通过 USART1 与 PC 机进行串口通信上传数据。

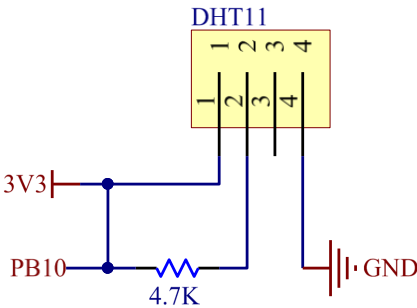


图 0.1 DHT11 接口电路

程序流程

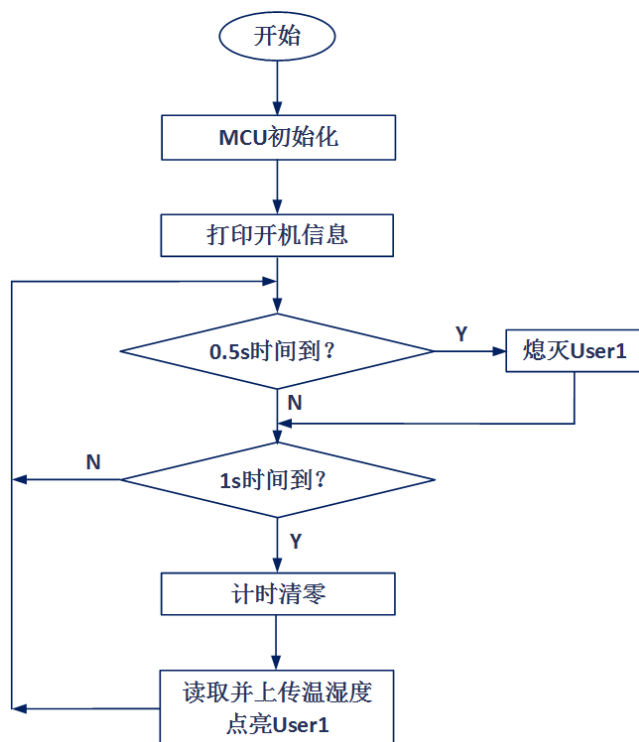


图 0.1 温湿度采集流程图

源码解析

本节实验的源码文件在【配套光盘\04-实验例程\01-物联网感知与控制技术\03-第三章传感器与数据采集实验\3.1 温湿度传感器采集实验\temp_humi\USER】目录下，主要包括 DHT11 相关配置及通信文件 sensor.c、以及主程序文件 main.c。下面具体解析这些文件中的主要函数。

1. MCU 初始化

函数 McuInit()主要包括系统时钟和外设时钟的初始化配置、滴答时钟的初始化配置、LED 的初始化配置、DHT11 的初始化配置、USART2 的初始化配置，以保证外设、延时函数、LED、DHT11，以及 USART2 能够正常工作，具体代码请查看本节实验例程。

```

void McuInit(void)
{
    RCC_Configuration();           // 配置系统时钟, 打开
    外设时钟;
    TickDelayInit(72);             // 延时函数初始化, 利用
    滴答时钟;
    LedGpioInit('B', 4, 5, 6, 7);  // Led 灯初始化;
    DHT11_Init();                  // 温湿度传感器初始
    化;
    UartInit(2, 115200);            // 初始化 USART2, 波特
    率设为 115200
}
  
```

2. DHT11 初始化

由图 0.1 知，STM32 是通过 PB10 控制 DHT11 的。所以，对 DHT11 进行初始化，也就是对 PB10 引脚进行初始化配置，具体代码如下。

```

void DHT11_Init(void)
{
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;    // 开漏输出
    GPIO_Init(GPIO_PORT, &GPIO_InitStructure);

    GPIO_SetBits(GPIOB,GPIO_Pin_10);                    // 开机置“1”
}

```

3. 温湿度数据的读取与上传

STM32 要读取温湿度数据，首先需要给 DHT11 发送复位信号，也就是请求 DHT11 发送数据的信号，对应 0 中的步骤二，实现代码如下。

```

void SmDHT11Reset(void)
{
    SmDHT11IOOut();                                // 将 PB10 设置为输出模
式

    GPIO_ResetBits(GPIOB,GPIO_Pin_10);            // PB10 输出低电平
    TickDelayMs(18);                                // 延时 18ms
    GPIO_SetBits(GPIOB,GPIO_Pin_10);                // PB10 输出高电平
    TickDelayUs(30);                                // 延时 30ms
}

```

DHT11 接收到请求信号后，会通过 PB10 向 DTM32 发送 40~80μs 的低电平响应信号，然后发送 40~80μs 的高电平信号通知 STM32 准备接收数据。该过程与 0 中的步骤三相对应，STM32 检测 DHT11 响应信号的代码如下。

```

uint8_t SmDHT11Check(void)
{
    u8 retry=0;

    SmDHT11IOIn();                                // 将 PB10 设置为输入模
式

    while(!SmDHT11ReadBitState() && (retry<100))    // 检测 PB10 持续为高电
平的时间
    {
        retry++;
        TickDelayUs(1);
    }
    if(retry>=100)return 1;                        // 如果该时间过长，不符合时序，返回 1，表示读
取失败

    else retry=0;                                // retry 清零
    while(SmDHT11ReadBitState() && (retry<100))    // 检测 PB10 持续为低电
平的时间
    {
        retry++;
        TickDelayUs(1);
    }
    if(retry>=100)return 1;                        // 如果该时间过长，不符合时序，返回 1，表示读
取失败

    return 0;                                    // 接收完 DHT11 的低电平和高电平信号，准备接收
数据
}

```

接收完 DHT11 的准备信号，STM32 开始接收数据。接收数据时是一位一位地接收，参照图 0.4 和图 0.5 所示的位数据格式判断是“0”还是“1”，然后将每 8 个位保存为一个字节，在该过程中用到的函数如下。

```

uint8_t SmDHT11ReadBit(void)                    // 接收位
{
    u8 retry=0;

```

```

while(SmDHT11ReadBitState() && (retry<100))    // 等待 PB10 变为低电平，因为
DHT11 会先发送低电平
{
    retry++;
    TickDelayUs(1);
}
retry=0;                                         // retry 清零
while(!SmDHT11ReadBitState() && (retry<100))    // 等待 PB10 变为高电平
{
    retry++;
    TickDelayUs(1);
}
TickDelayUs(40);                               // 延时 40μs
if(SmDHT11ReadBitState())return 1;              // 如果此时仍为高电平，该位数据
为“1”
else return 0;                                  // 如果此时为低电平，该位数据为“0”
}

uint8_t SmDHT11ReadByte(void)                   // 每 8 个位保存为一个字节
{
    uint8_t i, dat = 0;

    for(i=0; i<8; i++)                          // 将接收到的位数据依次保存到 dat 中
    {
        dat<<=1;
        dat|=SmDHT11ReadBit();
    }
    return dat;                                  // 接收并保存完 8 位数据，返回 dat，完
成一个字节的接收
}

```

通过调用函数 DHT11_Read_Byte()，接收完 5 个字节数据。由 0 中的步骤四知，第 5 个字节为 8 个校验位。根据校验位的数据定义进行校验，验证通过即完成了一次温湿度数据的读取，否则数据无效。函数 SmDHT11ReadData()调用了上面提到的几个函数，可实现一次完整的温湿度数据读取及上报，同时点亮串口发送指示灯 User1。

```

uint8_t SmDHT11ReadData(void)
{
    uint8_t buf[5];
    uint8_t i;
    static unsigned int t, h;

    SmDHT11Reset();                             // 给 DHT11 发送复位命令，等待应答信
号;
    if(SmDHT11Check())                           // 检查是否应答成功;
    {
        return 1;                                // 未检测到正确的应答信号，返回
    }
    for(i=0; i<5; i++)                          // 存储 DHT11 发送的 5 个字节到数组
buf[5]中
    {
        buf[i]=SmDHT11ReadByte();
    }
    if((buf[0]+buf[1]+buf[2]+buf[3])==buf[4])    // 验证校验位
    {
        h = buf[0];
        t = buf[2];
        LedOn(7);                               // 验证成功，点亮
发送指示灯 User1
        printf("humi:  %u%% %RH,temp:  %u °C\r\n", h, t); // 通过 USART2 上
传温湿度数据
    }
}

```



```
    return 0;
}
```

4. 主函数 main()

主函数主要实现了以下功能：

- 1) MCU 初始化：时钟、LED、DHT11、USART1 的初始化配置。
- 2) 打印开机信息：系统初始化完成后通过串口打印开机信息。
- 3) 读取并上传温湿度数据。
- 4) 控制 USART2 发送指示灯的闪烁。

具体代码如下。

```
int main(void)
{
    McuInit(); // MCU 初始化

    GPIO_SetBits(GPIOA, GPIO_Pin_1); //设置 485 引脚为高电平，设置 485 串口为发送模式
    printf("\r\n 温湿度采集实验\r\n"); // 打印开机信息

    while(1)
    {
        if(delaytime == 500000) // 0.5s 时间到
            LedOff(7); // 熄灭 User1
        if((delaytime++) >= 1000000) // 1s 时间到
        {
            delaytime = 0; // 计时清零
            GPIO_SetBits(GPIOA, GPIO_Pin_1); //设置 485 引脚为高电平，设置 485 串口
            SmdHT11ReadData(); // 读取并上传温湿度数据
        }
    }
}
```

5.4 实验步骤

1. 将 USB3.0 数据线的一端连接温湿度传感器模块的 USB3.0 调试烧写口，另一端连接 ST-Link 调试器的“Debug”接口。
2. 将第 1 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-Debug”接口。
3. 将第 2 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-485”接口。
4. 将“RJ11”线的一端连接温湿度传感器的“RJ11”口，另一端连接 ST-Link 调试器的“RS-485”接口，连接正确后效果如下图所示。




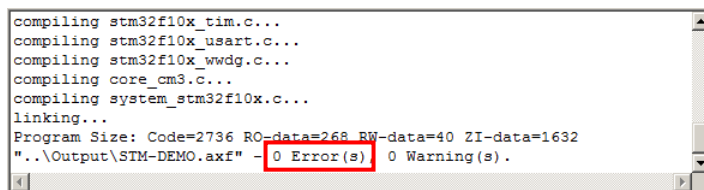
图 0.1 调试感知执行模块接线示意图


5. 双击打开【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】目录下的串口调试器，选择正确的端口号（可参照 1.2.2 查看串口端号），波特率设为 115200，其他均保持默认设置，效果见下图。点击按钮 **打开串口(C)** 打开串口即可。

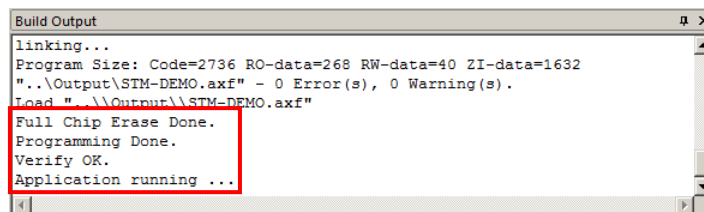


图 0.2 串口调试器配置

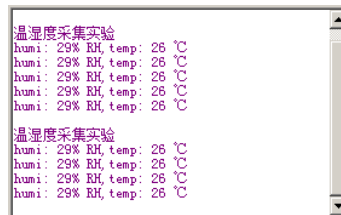
6. 在【配套光盘\04-实验例程\第 3 章基础技术实验区\3.5 温湿度传感器采集实验\temp_humi\USER】目录下，双击打开工程文件“temp_humi.uvproj”。
7. 在工具栏中点击按钮 ，编译工程成功后，信息框会出现下图所示的信息。如果编译失败，请参照 1.2.1.2 中的内容，确认相关选项是否配置正确。



8. 参照 1.2.1.3 中的内容，确认与硬件调试有关的选项已设置正确。
9. 点击按钮 ，将程序下载到温湿度传感器模块中。下载成功后，信息框显示下图中的信息，表明程序下载成功并已自动运行。



10. 程序运行起来后，Status 指示灯被点亮，User1 指示灯开始每隔 0.5s 闪烁一次。
11. 观察串口调试器接收区接收到的数据，按一下温湿度传感器模块的复位按钮，再观察串口调试器接收区接收到的数据，如下图所示。



5.5 实验结果

1. 程序运行起来后，温湿度传感器模块上的指示灯变化如下图所示。

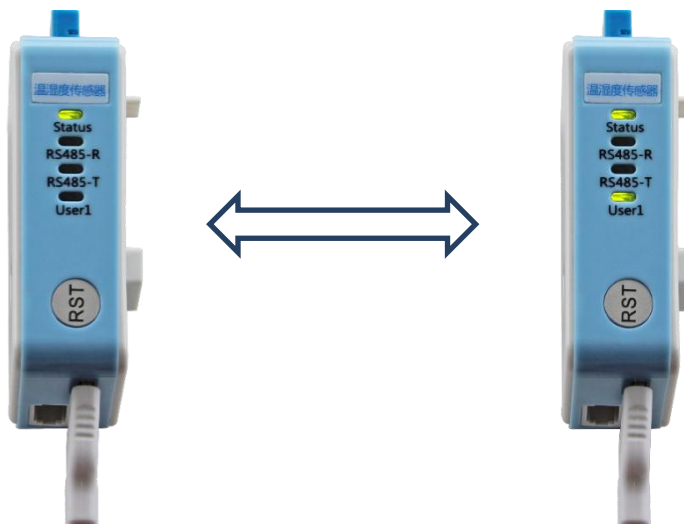


图 0.1 指示灯变化

2. 改变温湿度的过程中，串口调试器接收区数据如下图所示。

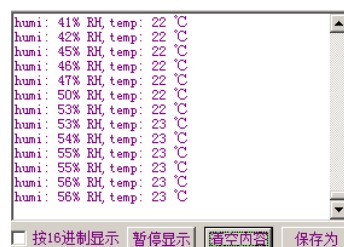


图 0.2 串口接收数据变化

实验六：基础区-多彩 LED 控制实验

6.1 实验目的

1. 了解多彩 LED 模块的控制原理；
2. 通过 STM32 接收上位机发送的控制命令，进而对 RGB 三色 LED 灯进行控制。

6.2 实验环境

1. 硬件：1 个多彩 LED 模块、1 个 ST-Link 调试器、2 根 USB2.0 方口线、1 根 USB3.0 数据线、1 根 RJ11 电话线、1 台 PC 机；
2. 软件：Windows 7/XP、MDK 集成开发环境、串口调试器。

6.3 实验原理



图 0.1 多彩 LED 模块

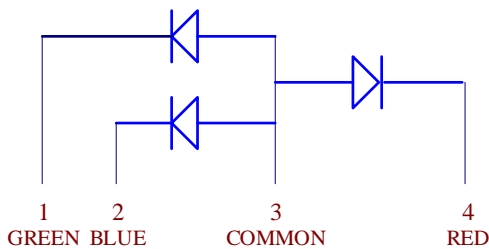
多彩 LED

多彩 LED 模块的核心执行部件为 RGB 三色灯，如下图所示。



图 0.1 RGB 三色灯

RGB 三色灯的内部原理如下图所示，第 3 脚为公共端，用来与电源 3.3V 相连，其它三个引脚分别为三个颜色的二极管的阴极，与 GND 相连时，对应颜色的发光二极管则点亮。例如，当第 3 脚与 3.3V 相连，第 2 脚与 GND 相连时，绿色灯将被点亮。



电路分析

RGB 三色灯与 STM32 的接口电路如下图所示。

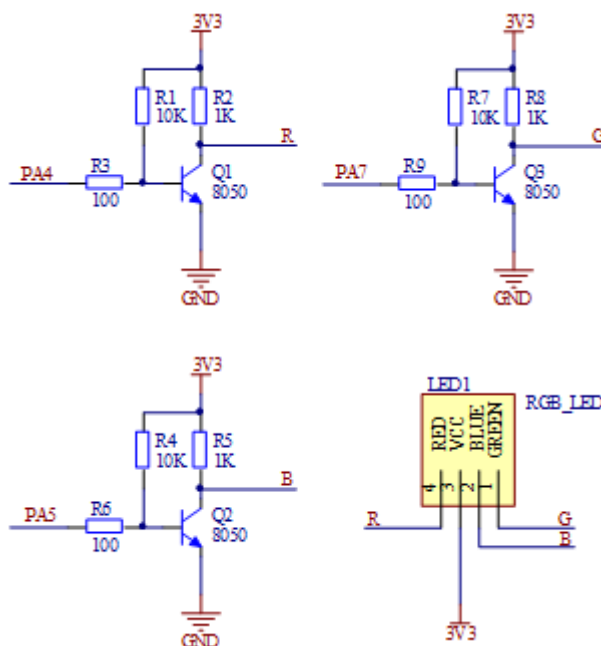


图 0.1 RGB 三色灯接口电路

下面以红色灯为例进行说明，当 PA4 为高电平时，三极管 Q1 导通，R 引脚与 GND 接通，红色灯亮；当 PA4 为低电平时，三极管 Q1 截止，R 引脚被上拉为高电平，红色灯灭。

因此，可以利用上位机通过 485 串口向多彩 LED 模块发送控制命令，使 STM32 控制 PA4、PA7、PA5，进而改变 RGB 三色灯的状态。

程序流程

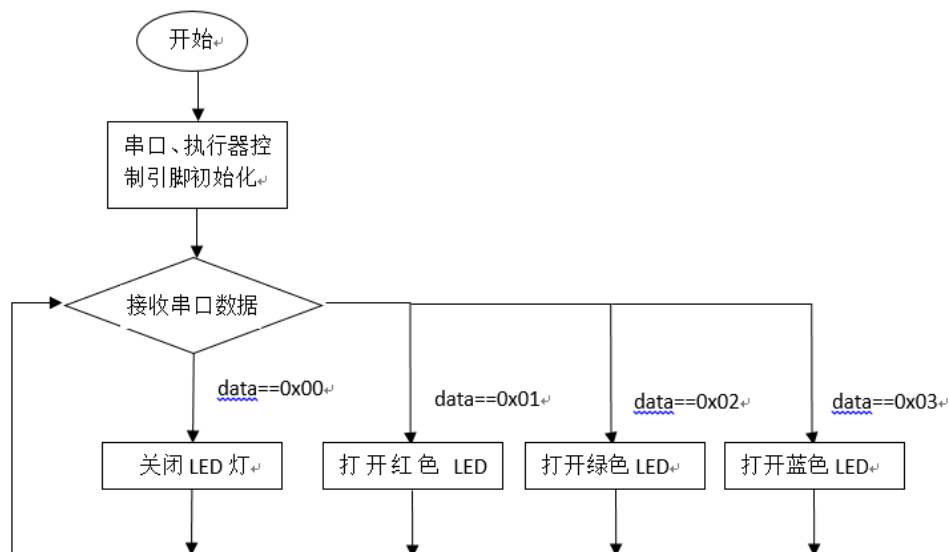


图 0.1 多彩 LED 流程图

源码解析

本节实验的源码文件在【配套光盘\04-实验例程\01-物联网感知与控制技术\04-第四章执行器与控制实验\4.2 多彩 LED 控制实验\rge_led\USER】目录下。下面对这些文件中的主要函数进行具体解析。

1. 串口中断

串口中断函数 `USART2_IRQHandler()` 从串口接收数据，具体代码请查看本节实验例程。

```

void USART2_IRQHandler(void)
{
    if(USART_GetITStatus(USART2, USART_IT_RXNE) == SET)
    {
        USART_ClearITPendingBit(USART2, USART_IT_RXNE); //清除中断标志位
        data = USART_ReceiveData(USART2);                //从串口 2 接收数据，赋值给临时变量 data;
    }
}

```

2. 多 LED 灯控制

由图 0.1 知，通过 PA4、PA5、PA7 引脚控制多彩 LED 灯。使用 `GPIO_SetBits(GPIOA,GPIO_Pin_4)` 来控制 PA4 输出高电平，使红色 LED 灯点亮，使用 `GPIO_ResetBits(GPIOA,GPIO_Pin_5)` 来控制红色 LED 灯熄灭。同理可以控制绿色与蓝色 LED 灯。具体代码请查看本节实验例程。

```

/*****
函数名      : red_on;
功能        : 打开红色 LED 灯
*****/
/
void red_on(void)
{
    GPIO_SetBits(GPIOA,GPIO_Pin_4);
}
/*****
函数名      : red_off;
功能        : 关闭红色 LED 灯
*****/

```

```

*****
/
void red_off(void)
{
    GPIO_ResetBits(GPIOA,GPIO_Pin_4);
}

```

3. 主函数 main ()

本节实验主要是通过 485 串口通讯，根据上位机传来的值，控制 LED 状态，具体代码如下。

```

int main(void) //主函数
{
    sensor_init(); //GPIO 初始化
    UartInit();    //串口参数设置
    UartNvicInit(); //串口中断设置
    GPIO_ResetBits(GPIOB,GPIO_Pin_0); //设置 485 为输入状态
    all_off();     //默认三种颜色的 LED 灯都为关闭状态
    while(1)
    {
        switch(data)
        {
            case 0x00:all_off(); //关闭三种颜色的 LED 灯
                break;
            case 0x01:all_off();red_on(); //只点亮红色 LED 灯
                break;
            case 0x02:all_off();green_on(); //只点亮绿色 LED 灯
                break;
            case 0x03:all_off();blue_on(); //只点亮蓝色 LED 灯
                break;
            default:
                break;
        }
    }
}


```

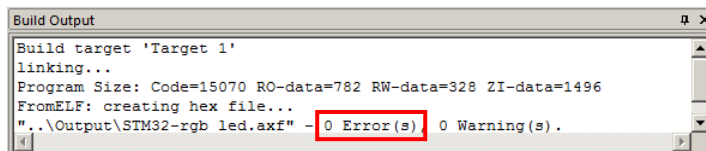
6.4 实验步骤


1. 将 USB3.0 数据线的一端连接多彩 LED 模块的 USB3.0 调试烧写口，另一端连接 ST-Link 调试器的“Debug”接口。
2. 将第 1 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-Debug”接口。
3. 将第 2 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-485”接口。
4. 将 RJ11 电话线的一端连接多彩 LED 模块的 RJ11-485 通信接口，另一端连接 ST-Link 调试器的“RS-485”接口，连接正确后效果如下图所示。

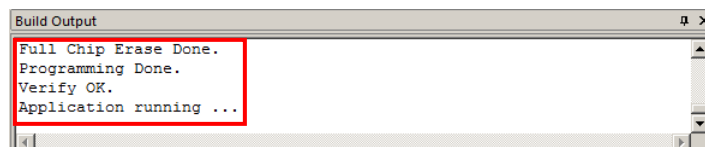



图 0.1 多彩 LED 模块接线示意图

5. 在【配套光盘\04-实验例程\第 3 章基础技术实验区\3.6 多彩 LED 控制实验\rge_led\USER】目录下，双击打开工程文件“rgb_led.uvproj”。
6. 在工具栏中点击按钮，编译工程成功后，信息框会出现下图所示的信息。如果编译失败，请参照 1.2.1.2 节中的内容，确认相关选项是否配置正确。



7. 参照 1.2.1.3 节中的内容，确认与硬件调试有关的选项已设置正确。如果检测不到硬件，请参照 1.1.3.1 节中的内容检查 ST-Link 驱动是否正确安装。
8. 点击按钮，将程序下载到多彩 LED 模块中。下载成功后，如果信息框显示下图中的信息，表明程序下载成功并已自动运行。



9. 双击打开【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】目录下的串口调试器，选择正确的端口号(可参照 1.2.2 节查看串口端号)，波特率设为 115200，点击按钮即可。



10. 利用串口调试器，勾选“按 16 进制显示或发送”，向多彩 LED 模块发送十六进制 00 或 01 或 02 或 03，进而改变多彩 LED 模块上的 RGB 三色灯的状态。

6.5 实验结果

1. 在发送区输入报文“01”，点击[发送]按钮，可观察到 RGB 三色灯的红色灯开始点亮。



2. 在发送区输入报文“00”，点击[发送]按钮，可观察到 LED 灯被熄灭了。在发送区输入报文“02”，点击[发送]按钮，可观察到 RGB 三色灯的绿色灯开始点亮。在发送区输入报文“03”，点击[发送]按钮，可观察到 RGB 三色灯的蓝色灯开始点亮。

实验七：基础区-风扇控制实验

7.1 实验目的

1. 了解风扇模块的控制原理；
2. 通过 STM32 接收上位机发送的控制命令，进而对继电器模块的输出进行控制。

7.2 实验环境

1. 硬件：1 个风扇模块、1 个风扇、1 个 ST-Link 调试器、2 根 USB2.0 方口线、1 根 USB3.0 数据线、1 根 RJ11 电话线、1 台 PC 机；
2. 软件：Windows 7/XP、MDK 集成开发环境、串口调试器。

7.3 实验原理



图 0.1 继电器模块

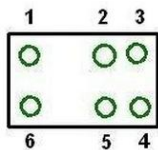
继电器

风扇模块的核心执行部件为 HT4100F 继电器，如下图所示。



图 0.1 HT4100F 继电器

继电器的引脚图如下，其中，引脚 2、5 为线圈，引脚 1、6 为公共端，引脚 3 为常开触点，引脚 4 为常闭触点。当线圈得电时，常开触点与公共端接通，常闭触点与公共端断开；当线圈失电时，常开触点与公共端断开，常闭触点与公共端接通；



电路分析

继电器与 STM32 的接口电路如下图所示。

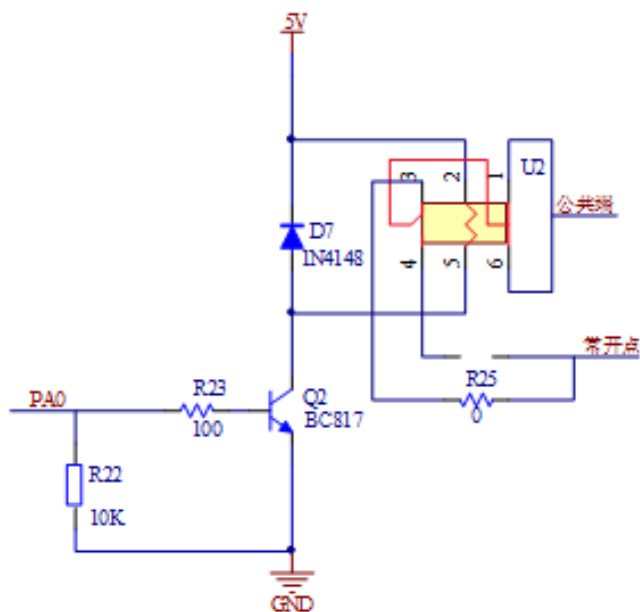


图 0.1 继电器接口电路

当 PA0 输出高电平时，三极管 Q2 导通，则引脚 5 与 GND 接通，使得线圈得电，常开点与公共端接通；当 PA0 输出低电平时，三极管 Q2 截止，则引脚 5 被 D7 上拉为高电平，使得线圈失电，常开点与公共端断开。

风扇模块的输出端子的端子定义如下图所示，连接时，公共端与 5V 连接，然后利用上位机通过 485 串口向风扇模块发送控制命令，使 STM32 控制 PA0 输出高低电平来控制风扇模块内部继电器线圈的得电或失电，进而改变输出端子上连接的执行器的状态。



程序流程

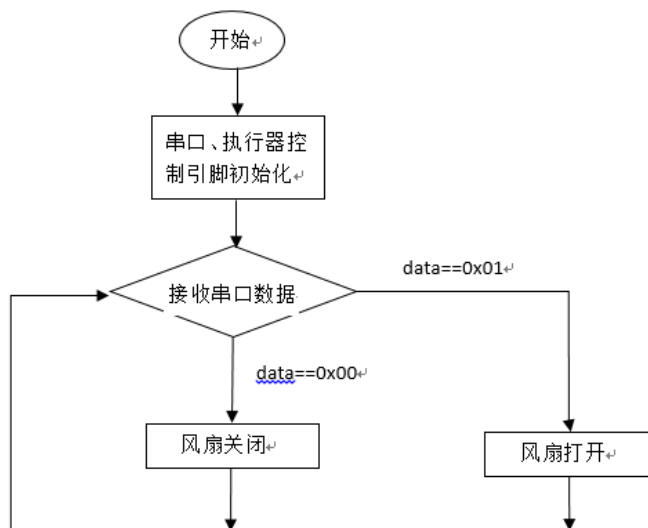


图 0.1 继电器流程图

源码解析

本节实验的源码文件在【配套光盘\04-实验例程\01-物联网感知与控制技术\04-第四章执行器与控制实验\4.3 继电器控制实验\relay\USER】目录下。下面对这些文件中的主要函数进行具体解析。

1. 串口中断

串口中断函数 `USART2_IRQHandler()` 从串口接收数据，具体代码请查看本节实验例程。

```
void USART2_IRQHandler(void)
{
    if(USART_GetITStatus(USART2, USART_IT_RXNE) == SET)
    {
        USART_ClearITPendingBit(USART2, USART_IT_RXNE); //清除中断标志位
        data = USART_ReceiveData(USART2);                //从串口 2 接收数据，赋值给临时变量 data; }
    }
```

2. 继电器控制

由图 0.1 知，PA0 为高电平继电器得电，PA0 为低电平时继电器失电。使用 `GPIO_SetBits(GPIOA,GPIO_Pin_0)` 来控制 PA0 输出高电平，使继电器得电，使用 `GPIO_ResetBits(GPIOA,GPIO_Pin_0)` 来控制继电器的失电，具体代码请查看本节实验例程。

```
/*
函数名      : relay_on;
功能        : 打开继电器
*/
void relay_on(void)
{
    GPIO_SetBits(GPIOA,GPIO_Pin_0);
}

/*
函数名      : relay_off;
```

```

        功能          : 关闭继电器
        *****
/
void relay_off(void)
{
    GPIO_ResetBits(GPIOA,GPIO_Pin_0);
}

```

3. 主函数 main ()

本节实验主要是通过 485 串口通讯，根据上位机传来的值，控制继电器状态，具体代码如下。

```

int main(void)          //主函数
{
    sensor_init();      //GPIO 初始化
    UartInit();          //串口参数设置
    UartNvicInit();      //串口中断设置
    GPIO_ResetBits(GPIOB,GPIO_Pin_0); //设置 485 为输入状态
    relay_off();          //风扇默认关闭
    while(1)
    {
        if(data == 1)    //当数据为 01 时，则打开风扇
        {
            relay_on();
        }
        else if(data == 0) //当数据为 00 时，则关闭风扇
        {
            relay_off();
        }
    }
}


```

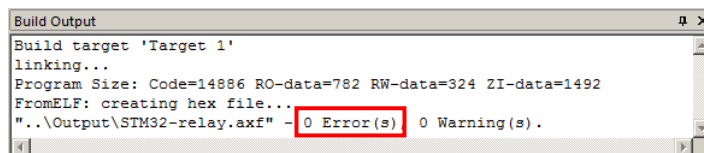
7.4 实验步骤


1. 将 USB3.0 数据线的一端连接继电器模块的 USB3.0 调试烧写口，另一端连接 ST-Link 调试器的“Debug”接口。
2. 将第 1 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-Debug”接口。
3. 将第 2 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-485”接口。
4. 将 RJ11 电话线的一端连接继电器模块的 RJ11-485 通信接口，另一端连接 ST-Link 调试器的“RS-485”接口，连接正确后效果如下图所示。




图 0.1 继电器模块接线示意图

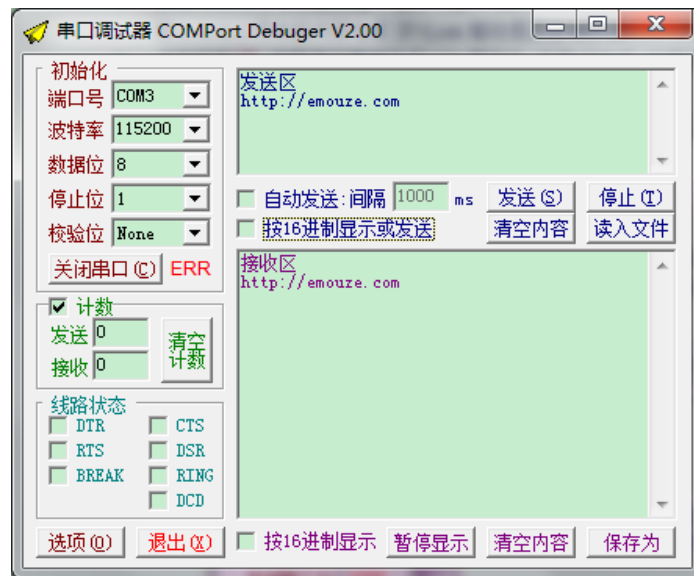
5. 在【配套光盘\04-实验例程\第 3 章基础技术实验区\3.7 风扇控制实验\relay\USER】目录下，双击打开工程文件“relay.uvproj”。
6. 在工具栏中点击按钮，编译工程成功后，信息框会出现下图所示的信息。如果编译失败，请参照 1.2.1.2 节中的内容，确认相关选项是否配置正确。



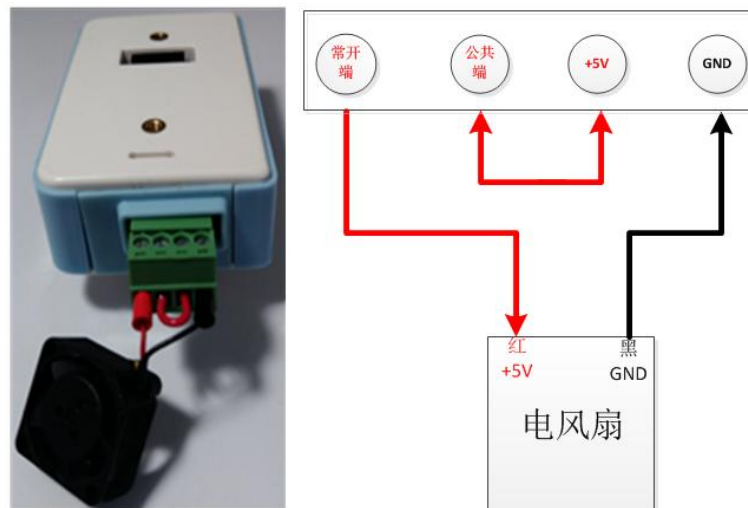
7. 参照 1.2.1.3 节中的内容，确认与硬件调试有关的选项已设置正确。如果检测不到硬件，请参照 1.1.3.1 节中的内容检查 ST-Link 驱动是否正确安装。
8. 点击按钮，将程序下载到风扇模块中。下载成功后，如果信息框显示下图中的信息，表明程序下载成功并已自动运行。



9. 双击打开【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】目录下的串口调试器，选择正确的端口号(可参照 1.2.2 节查看串口端号)，波特率设为 115200，点击按钮即可。



10. 按照下图将风扇连接到继电器模块的输出端子上。



11. 利用串口调试器，勾选“按 16 进制显示或发送”，向继电器模块发送 01 打开风扇；发送 00 关闭风扇。

7.5 实验结果

1. 在发送区输入报文“01”，点击[发送]按钮，可观察到风扇转动。



2. 在发送区输入报文“00”，点击按钮，可观察到风扇停止。

实验八：基础区-步进电机控制实验

8.1 实验目的

1. 了解步进电机驱动模块的控制原理；
2. 通过 STM32 接收上位机发送的控制命令，进而对步进电机进行控制。

8.2 实验环境

1. 硬件：1 个步进电机驱动模块、1 个步进电机、1 个 ST-Link 调试器、2 根 USB2.0 方口线、1 根 USB3.0 数据线、2 根 RJ11 电话线、1 台 PC 机；
2. 软件：Windows 7/XP、MDK 集成开发环境、串口调试器。

8.3 实验原理



图 0.1 步进电机驱动模块

步进电机

如下图所示。可以通过一根 RJ11 电话线来连接步进电机驱动模块的电机接口与步进电机模块上的 RJ11 接口。



图 0.1 步进电机模块连接图

本次实验中使用到的步进电机为四相步进电机，采用单极性直流电源供电。只要对步进电机的各相绕组按合适的时序通电，就能使步进电机步进转动。下图是该四相反应式步进电机工作原理示意图。

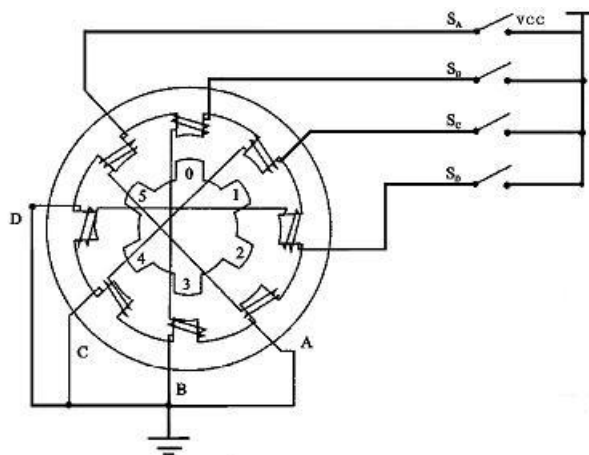


图 0.2 四相步进电机工作原理示意图

开始时，开关 S_B 接通电源， S_A 、 S_C 、 S_D 断开，B 相磁极和转子 0、3 号齿对齐，同时，转子的 1、4 号齿就和 C、D 相绕组磁极产生错齿，2、5 号齿就和 D、A 相绕组磁极产生错齿。

当开关 S_C 接通电源， S_B 、 S_A 、 S_D 断开时，由于 C 相绕组的磁力线和 1、4 号齿之间磁力线的作用，使转子转动，1、4 号齿和 C 相绕组的磁极对齐。而 0、3 号齿和 A、B 相绕组产生错齿，2、5 号齿就和 A、D 相绕组磁极产生错齿。依次类推，A、B、C、D 四相绕组轮

流供电，则转子会沿着 A、B、C、D 方向转动。

电路分析

在本节实验中，利用 PB0、PB1、PB10、PB11 通过两个驱动芯片 L9110S 分别取控制步进电机的 A、B、C、D 四相绕组，进而实现对步进电机的控制。

驱动芯片 L9110S 与 STM32 的接口电路如下图所示。

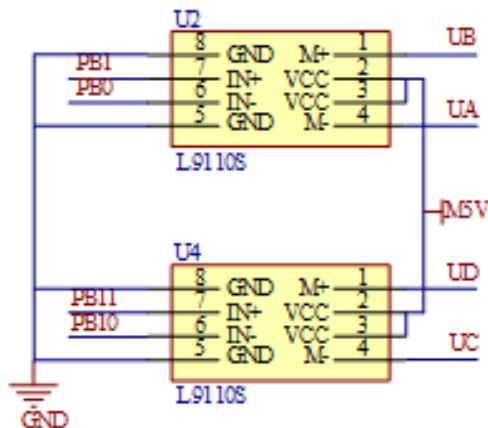


图 0.1 步进电机接口电路

当 PB0 输出高电平时，A 相通电；PB0 输出低电平时，A 相断电。当 PB1 输出高电平时，B 相通电；PB1 输出低电平时，B 相断电。当 PB10 输出高电平时，C 相通电；PB10 输出低电平时，C 相断电。当 PB11 输出高电平时，D 相通电；PB11 输出低电平时，D 相断电。控制步进电机的具体代码请查看本节实验例程。

程序流程

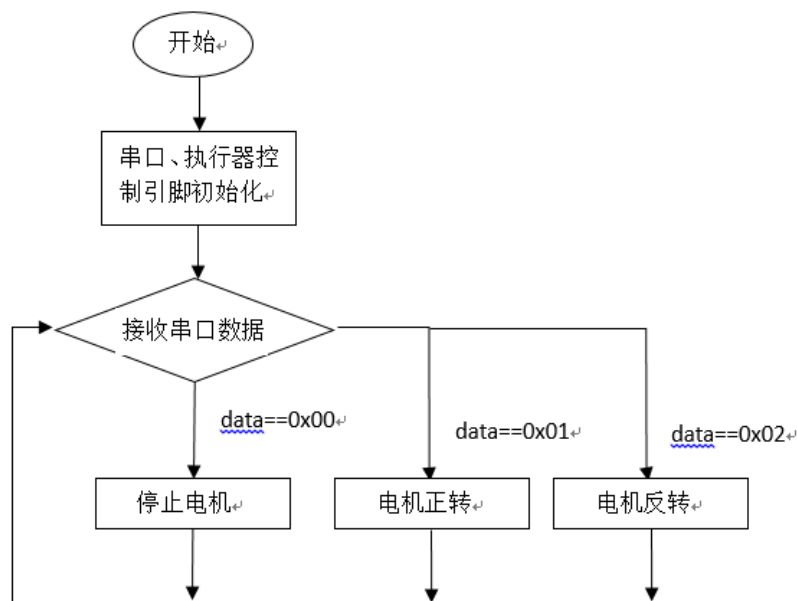


图 0.1 步进电机流程图

源码解析

本节实验的源码文件在【配套光盘\04-实验例程\01-物联网感知与控制技术\04-第四章执行器与控制实验\4.4 步进电机控制实验\step_motor\USER】目录下。下面对这些文件中的主要函数进行具体解析。

1. 串口中断

串口中断函数 USART2_IRQHandler() 从串口接收数据，具体代码请查看本节实验例程。

```
void USART2_IRQHandler(void)
{
    if(USART_GetITStatus(USART2, USART_IT_RXNE) == SET)
    {
        USART_ClearITPendingBit(USART2, USART_IT_RXNE); //清除中断标志位
        data = USART_ReceiveData(USART2);                //从串口 2 接收数据，赋值给临时变量 data;
    }
}
```

2. 步进电机控制

由图 0.1 步进电机接口电路知，PB0、PB1、PB10、PB11 控制步进电机。具体代码请查看本节实验例程。

```

/*****
函数名      : foreward;
功能        : 步进电机正转
*****/
/
void foreward(void)
{
    M1_H;
    M2_L;
    M3_L;
    M4_L;
    DelayMs(1);

    M1_H;
    M2_H;
    M3_L;
    M4_L;
    DelayMs(1);

    M1_L;
    M2_H;
    M3_L;
    M4_L;
    DelayMs(1);

    M1_L;
    M2_H;
    M3_H;
    M4_L;
    DelayMs(1);

    M1_L;
    M2_L;
    M3_H;
    M4_L;
    DelayMs(1);

    M1_L;
    M2_L;
    M3_H;
}
```

```

        M4_H;
        DelayMs(1);

        M1_L;
        M2_L;
        M3_L;
        M4_H;
        DelayMs(1);

        M1_H;
        M2_L;
        M3_L;
        M4_H;
        DelayMs(1);
    }
    /*****
    函数名      : back;
    功能        : 步进电机反转
    *****/
/

void back(void)
{
    M1_L;
    M2_H;
    M3_H;
    M4_H;
    DelayMs(1);

    M1_L;
    M2_L;
    M3_H;
    M4_H;
    DelayMs(1);

    M1_H;
    M2_L;
    M3_H;
    M4_H;
    DelayMs(1);

    M1_H;
    M2_L;
    M3_L;
    M4_H;
    DelayMs(1);

    M1_H;
    M2_H;
    M3_L;
    M4_H;
    DelayMs(1);

    M1_H;
    M2_H;
    M3_L;
    M4_L;
    DelayMs(1);

    M1_H;
    M2_H;
    M3_H;
    M4_L;
    DelayMs(1);

    M1_L;
    M2_H;

```

```

        M3_H;
        M4_L;
        DelayMs(1);
    }
    /*****
        函数名      : stop;
        功能        : 步机电机停止
        *****/
    /
    void stop(void)
    {
        M1_L;
        M2_L;
        M3_L;
        M4_L;
    }

```

3. 主函数 main ()

本节实验主要是通过 485 串口通讯，根据上位机传来的值，控制步进电机状态，具体代码如下。

```

int main(void)          //主函数
{
    sensor_init();      //GPIO 初始化
    UartInit();          //串口参数设置
    UartNvicInit();      //串口中断设置
    GPIO_ResetBits(GPIOA,GPIO_Pin_1); //设置 485 为输入状态
    stop();
    while(1)
    {
        if(data == 0x01)    //当数据为 01 时，则步进电机正转
        {
            foreward();
        }
        else if(data == 0x02) //当数据为 02 时，则步进电机反转
        {
            back();
        }
        else if(data == 0x00) //当数据为 00 时，则步进电机停止
        {
            stop();
        }
    }
}


```

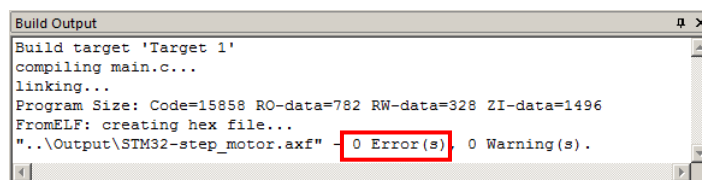
8.4 实验步骤


1. 将 USB3.0 数据线的一端连接步进电机驱动模块的 USB3.0 调试烧写口，另一端连接 ST-Link 调试器的“Debug”接口。
2. 将第 1 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-Debug”接口。
3. 将第 2 根 USB2.0 方口线的一端连接 PC 机的 USB 口，另一端连接 ST-Link 调试器的“USB-485”接口。
4. 将 RJ11 电话线的一端连接步进电机驱动模块的 RJ11-485 通信接口，另一端连接 ST-Link 调试器的“RS-485”接口，连接正确后效果如下图所示。

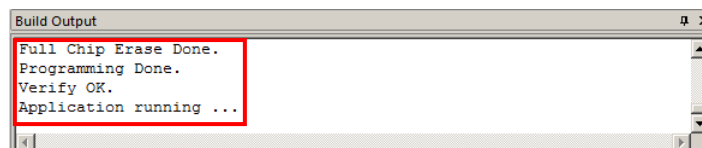



图 0.1 步进电机驱动模块接线示意图

5. 在【配套光盘\04-实验例程\第 3 章基础技术实验区\3.9 步进电机控制实验\step_motor\USER】目录下，双击打开工程文件“step_motor.uvproj”。
6. 在工具栏中点击按钮，编译工程成功后，信息框会出现下图所示的信息。如果编译失败，请参照 1.2.1.2 节中的内容，确认相关选项是否配置正确。



7. 参照 1.2.1.3 节中的内容，确认与硬件调试有关的选项已设置正确。如果检测不到硬件，请参照 1.1.3.1 节中的内容检查 ST-Link 驱动是否正确安装。
8. 点击按钮，将程序下载到步进电机驱动模块中。下载成功后，如果信息框显示下图所示的信息，表明程序下载成功并已自动运行。



9. 双击打开【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】目录下的串口调试器，选择正确的端口号（可参照 1.2.2 节查看串口端号），波特率设为 19200，点击按钮即可。



10. 将电机模块通过 RJ11 电话线连接到步进电机驱动模块上。
11. 利用串口调试器, 勾选“按 16 进制显示或发送”, 向步进电机驱动模块发送十六进制 00 或 01 或 02, 观察电机状态。

8.5 实验结果

1. 在发送区输入报文“01”, 点击[发送]按钮, 可看到步进电机正转。



2. 发送“02”, 可看到步进电机反转, 发送“00”, 可看到步进电机停止转动。

实验九：基础区-I/O 口控制 LED 实验

9.1 实验目的

1. 了解 CC2530 I/O 口的特点与特性；
2. 学习并掌握 CC2530 I/O 口的基本配置；
3. 学习并掌握 CC2530 I/O 口的程序设计；
4. 在 Zigbee 模块上运行 I/O 口控制程序。

9.2 实验环境

1. 硬件平台：ZigBee 模块 1 个、ZB-LINK 调试器 1 个、USB3.0 数据线、USB 方口线；
2. 软件平台：WinXP/Win7、IAR 集成开发环境。

9.3 实验原理

本节通过对 CC2530 的 I/O 口进行配置，将其配置为输出模式后，让 I/O 口输出高低电平来实现对共阳极 LED 灯的逐个控制来学习对 CC2530 单片机 I/O 类程序的编写方法与技巧。当被作为通用 I/O 使用时，所有引脚用三个 8 位端口寄存器表示，分别是 P0，P1 和 P2。P0 和 P1 是完全 8 位端口，P2 仅 5 位是有效引脚位。所有端口都是通过 P0，P1，P2 的三个 SFR 寄存器进行位寻址和字节寻址的。每个端口引脚都可单独设置成通用 I/O 或外围设备 I/O。

电路分析与程序配置

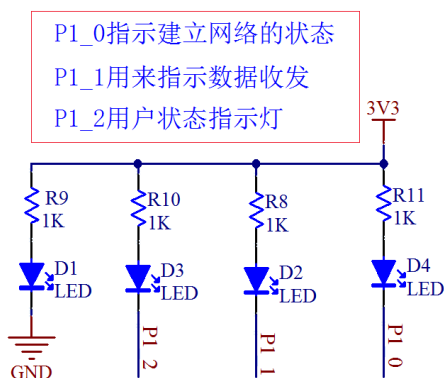


图 0.1 LED 接口电路

根据图 3.1.1 可以看出 D4 所对应的 I/O 为 P1_0，D2 所对应的 I/O 为 P1_1，D3 所对应的 I/O 为 P1_2，如图为 LED 灯的驱动电路，本实验选择 P1_0，P1_1 和 P1_2 I/O 引脚，P1_0 控制模块上的 LED 灯 Status/Data，P1_1 控制模块上的 LED 灯 Net，P1_2 控制模块上的 LED 灯 User1。故在软件上只要配置好 P1_0 口，P1_1 口与 P1_2 口即可。

1. I/O 简介

CC2530 有 21 个数字输入/输出引脚，可以配置为通用数字 I/O 或外设 I/O 信号，可配置为连接到 ADC、定时器或 USART 的外设。这些 I/O 口的使用可以通过一系列寄存器配置，由用户程序加以实现。

2. I/O 特性

CC2530 的 I/O 端口具备如下重要特性：

- 1) 21 个数字 I/O 引脚；
- 2) 可配置为通用 I/O 或外部设备 I/O；
- 3) 输入口具备上拉或下拉能力；
- 4) 具有外部中断能力；

说明：21 个 I/O 引脚都可以用作于外部中断源输入口。因此如果需要外部设备可以产生中断。外部中断功能也可以从睡眠模式唤醒设备。

3. I/O 引脚

由图 3.1.1 的 CC2530 的芯片引脚排布可知：CC2530 的 I/O 口一共有 21 个，分成 3 组，分别是 P0、P1 和 P2；其中：P0 包括 P0_0~P0_7，P1 包括 P1_0~P1_7，P2 包括 P2_0~P2_4。

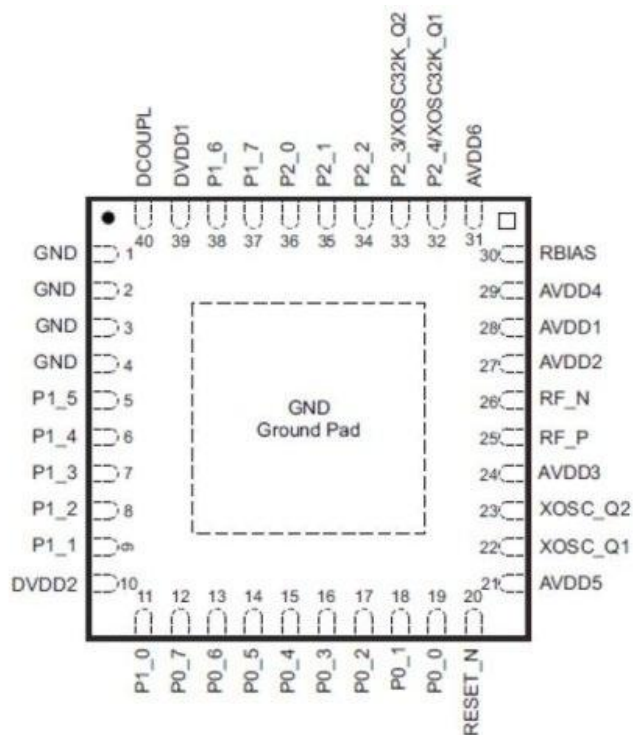


图 0.2 CC2530 的 I/O 引脚

4. 寄存器配置

本次实验所用到的控制寄存器中每一位的取值所对应的意义如下所示：

P1DIR (P1 方向寄存器，PODIR 同理)

表 0.1 P1DIR 寄存器

D7	D6	D5	D4	D3	D2	D1	D0
P1.7 的方向 0: 输入 1: 输出	P1.6 的方向 0: 输入 1: 输出	P1.5 的方向 0: 输入 1: 输出	P1.4 的方向 0: 输入 1: 输出	P1.3 的方向 0: 输入 1: 输出	P1.2 的方向 0: 输入 1: 输出	P1.1 的方向 0: 输入 1: 输出	P1.0 的方向 0: 输入 1: 输出

P1SEL (P1 功能选择寄存器, POSEL 同理)

表 0.2 P1SEL 寄存器

D7	D6	D5	D4	D3	D2	D1	D0
P1.7 0: I/O 1: 外设	P1.6 的功能 0: 普通 I/O 1: 外设功能	P1.5 的功能 0: 普通 I/O 1: 外设功能	P1.4 的功能 0: 普通 I/O 1: 外设功能	P1.3 的功能 0: 普通 I/O 1: 外设功能	P1.2 的功能 0: 普通 I/O 1: 外设功能	P1.1 的功能 0: 普通 I/O 1: 外设功能	P1.0 的功能 0: 普通 I/O 1: 外设功能

寄存器的配置方法:

1) 将控制寄存器的某一位置 1:

例: P1DIR |= 0X02;

解释: “|=” 表示按位或运算, 0X02 为十六进制数, 转换成二进制数为 0000 0010, 若 P1DIR 原来的值为 0011 0000, 或运算后 P1DIR 的值为 0011 0010。根据上面给出的取值表可知, 按位或运算后 P1_1 的方向改为输出, 其他 I/O 口方向保持不变。

2) 将控制寄存器某一位清 0:

例: P1DIR &= ~0x02;

解释: “&=” 表示按位与运算, “~” 运算符表示取反, 0x02 为 0000 0010, 即~0X02 为 1111 1101。若 P1DIR 原来的值为 0011 0010, 与运算后 P1DIR 的值为 0011 0000。

源码解析

```
void main(void)
{
    led_init();           // LED 灯初始化

    while(1)
    {
        led_test();       // LED 灯测试程序
    }
}
```

主函数中调用了 LED 指示灯初始化, 然后循环执行 LED 测试函数从而实现 LED 的闪烁控制。

```
void led_test(void)
{
    D4 = 0;           // D4 闪烁
    halWait(250);     // 延时 250ms
    D4 = 1;
    halWait(250);

    D2 = 0;           // D2 闪烁
    halWait(250);
    D2 = 1;
    halWait(250);

    D3 = 0;           // D3 闪烁
    halWait(250);
    D3 = 1;
    halWait(250);
}
```

其中: #define D4 P1_0 // Status/Data
#define D2 P1_1 // Net
#define D3 P1_2 // User1

即：定义原理图中的 D4,D2,D3 到对应的 I/O 口，LED 亮灭之间调用 `halWait()` 硬件延时库函数。

而 LED 初始化的函数实现为：

```
void led_init(void)
{
    P1SEL &= ~0x07; // P1.0 P1.1 P1.2 为普通 I/O 口
    P1DIR |= 0x07;  // 配置为输出

    D4=D2=D3=1;     // 正式运行前先灭灯。
}
```

程序流程

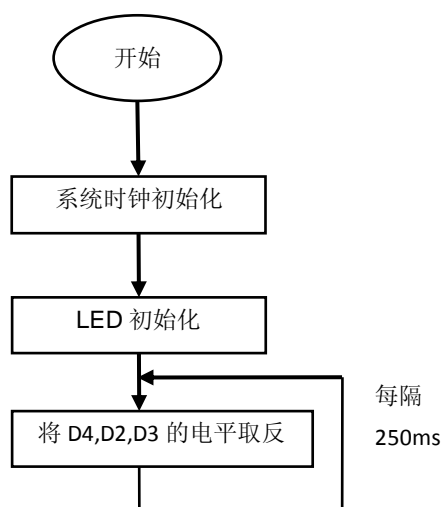



图 0.1 LED 程序流程图

由上图流程可知，实现 LED 的轮流闪烁，会经过系统时钟初始化的过程，而且系统时钟初始化是必须的，8051 微处理器的正常运行，必须要经过系统初始化，也就是调用 `xtal_init()` 函数，该函数在 `sys_init.c` 源文件中定义。在 `main.c` 中并没有看到调用系统时钟初始化的方法，这是因为官方库文件已经将系统时钟初始化的方法的调用过程写进启动文件中了（在后面的所有章节中也需要涉及到系统时钟的初始化，将不再重复说明）。

9.4 实验步骤

1. 正确连接 ZB-LINK 调试器到 PC 机和 ZigBee 模块，可参考第 1 章的 1.3 节进行连接；
2. 用 IAR 软件打开路径为：【配套光盘\04-实验例程\3 第三章基础技术实验区\3.11 IO 控制 LED 实验\实验代码】中的实验工程“LEDs.eww”，选择 Project→Rebuild All 重新编译工程；
3. 选择 Project→Download and debug 将程序下载到 ZigBee 模块，下载完成后点  退出；
4. 按下 ZB-LINK 调试器上的复位键。观察程序运行时 ZigBee 模块上 LED 灯的闪烁情况；
5. 修改延时函数，可以改变 LED 小灯的闪烁间隔时间。

9.5 实验结果

当程序烧写后，可以看到 LED 灯在 ZigBee 模块上至上而下地依次闪烁，说明该 I/O 程序已经正确地运行起来，从而完成了本节的 I/O 口控制 LED 实验。

实验十：基础区-Uart 串口通信实验

10.1 实验目的

1. 认识并了解 CC2530 串口应用；
2. 学习并掌握 CC2530 串口基本配置；
3. 学习并掌握 CC2530 串口程序编写。

10.2 实验环境

1. 硬件平台：ZigBee 模块一块、ZB-LINK 调试器、PC 机、USB3.0 线、USB 方口线 2 根。
2. 软件平台：Windows 7/Windows XP、IAR 集成开发环境、串口调试助手 Comdebug。

10.3 实验原理

在 WSN（无线传感器网络）中，CC2530 需要将采集到的数据发送给上位机 PC 进行处理，同时上位机需要向 CC2530 发送控制信息，这一切都离不开两者之间的数据通信传输接口——通用异步串行口，简称：“Uart 串口”。本实验通过正确配置 CC2530 的 Uart 从而实现与上、下位机之间的数据通信。所以，要正确地使用串口，就应该了解串口的工作模式以及如何实现 Uart 的数据收发。

1. Uart 模式

CC2530 有两个串行通信接口 UART0 和 UART1。两个串口既可以工作于 UART（异步通信）模式，也可以工作于 SPI（同步通信）模式，模式的选择有串口控制/状态寄存器的 UxCSR.MODE 决定。本实验采用 UART0 的标准 UART 模式。

UART 模式提供全双工传送。传送一个 UART 字节包含 1 个起始位、8 个数据位、1 个可选的第 9 位数据位或奇偶校验位再加上 1 个（或 2 个）停止位。

注意：串口数据帧格式为 11 位，即：起始位(1 位)，数据位(8 位)，校验位(1 位可选)，停止位(1 位)。而在应用中，真正所关心的是数据位。

UART 操作由 UART0 控制与状态寄存器 U0CSR 控制。当 U0CSR.MODE 设置为 1 时，即选择了 UART 模式。

2. UART 发送

当寄存器 U0DBUF 写入数据后，该数据被发送到输出引脚 TXD0。当传输开始时，U0CSR.ACTIVE 位变高，而当传输结束后 U0CSR.ACTIVE 位变低。传输结束时，TX_BYTE 位和 RX_BYTE 位置为 1，当收到新的数据，并在 USART 收/发数据寄存器 U0BUF 中就绪时，就产生一个中断。该中断在传输开始之后立刻发生，因此，当字节正在发送时，新的字节能够装入数据缓冲器。也就是说，数据通过 Uart 发送的同时，U0DBUF 中的数据也能同步更新。

3. UART 接收

当 1 写入 U0CSR.RE 位时，开始在 UART 上接收数据。USART 会在输入引脚 RXD0 中寻找有效起始位，并且设置 U0CSR.ACTIVE 位为 1。当检测到有效起始位时，收到的数据就传入

D7	D6	D5~D3	D2~D0
32KHZ 时间振荡器选择	系统时钟选择	定时器输出标记	系统主时钟选择

D7 位：32KHz 时钟振荡器选择，0 为 32KRC 震荡，1 为 32K 晶振。默认为 1。

D6 位：系统时钟选择。0 为 32M 晶振，1 为 16M RC 震荡。当 D7 位为 0 时 D6 必须为 1。

D5~D3：定时器输出标记位。000 为 32MHZ，001 为 16MHZ，010 为 8MHZ，011 为 4MHZ，100 为 2MHZ，101 为 1MHZ，110 为 500KHZ，111 为 250KHZ。默认为 001。需要注意的是：当 D6 为 1 时，定时器频率最高为 16MHZ。

D2~D0：系统主时钟选择：000 为 32MHZ，001 为 16MHZ，010 为 8MHZ，011 为 4MHZ，100 为 2MHZ，101 为 1MHZ，110 为 500KHZ，111 为 250KHZ。当 D6 为 1 时，系统主时钟最高频率为 16MHZ。

CLKCONSTA：时间频率状态寄存器。

表 0.2 CLKCONSTA 寄存器

D7	D6	D5~D3	D2~D0
当前 32KHZ 时间振荡器	当前系统时钟	当前定时器输出标记	当前系统主时钟

D7 位：当前 32KHZ 时间振荡器频率。0 为 32KRC 震荡，1 为 32K 晶振。

D6 位：当前系统时钟选择。0 为 32M 晶振，1 为 16M RC 震荡。

D5~D3：当前定时器输出标记。000 为 32MHZ，001 为 16MHZ，010 为 8MHZ，011 为 4MHZ，100 为 2MHZ，101 为 1MHZ，110 为 500KHZ，111 为 250KHZ。

D2~D0：当前系统主时钟。000 为 32MHZ，001 为 16MHZ，010 为 8MHZ，011 为 4MHZ，100 为 2MHZ，101 为 1MHZ，110 为 500KHZ，111 为 250KHZ。

UOCSR：USART0 控制与状态寄存器：

表 0.3 UOCSR 寄存器

D7	D6	D5	D4	D3	D2	D1	D0
模式选择	接收器使能	SPI 主/从模式	帧错误状态	奇偶错误状态	接受状态	传送状态	收发主动状态

D7：工作模式选择，0 为 SPI 模式，1 为 USART 模式。

D6：UART 接收器使能，0 为禁用接收器，1 为接收器使能。

D5：SPI 主/从模式选择，0 为 SPI 主模式，1 为 SPI 从模式。

D4：帧错误检测状态，0 为无错误，1 为出现出错。

D3：奇偶错误检测，0 为无错误出现，1 为出现奇偶校验错误。

D2：字节接收状态，0 为没有收到字节，1 为准备好接收字节。

D1：为字节传送状态，0 为字节没有被传送，1 为写到数据缓冲区的字节已经被发送。

D0：为 USART 接收/传送主动状态，0 为 USART 空闲，1 为 USART 忙碌。

UOGCR：USART0 通用控制寄存器。

表 0.4 UOGCR 寄存器

D7	D6	D5	D4~D0
SPI 时钟极性	SPI 时钟相位	传送位顺序	波特率指数值

D7：SPI 时钟极性：0 为负时钟极性，1 为正时钟极性；

D6：SPI 时钟相位：

D5：传送为顺序：0 为最低有效位先传送，1 为最高有效位先传送。

D4~D0：波特率设置：

表 0.5

波特率	指数值	小数部分
2400	6	59

4800	7	59
9600	8	59
14400	8	216
19200	9	59
28800	9	216
38400	10	59
57600	10	216
76800	11	59
115200	11	216
230400	12	216

U0BAUD: 波特率控制小数部分。(取值参考上表)

源码解析

```
/*主函数-----*/
void main(void)
{
    xtal_init();
    led_init();
    uart0_init(0x00, 0x00);           // 初始化串口:无奇偶校
    验, 停止位为 1 位
    Uart_Send_String("Please Input string ! \r\n"); // 复位后发出

    while(1)
    {
        Recv_Dat = Uart_Recv_char(); // 接收数据

        D3=0;                         // LED 指示
        halWait(250);
        D3=1;

        Uart_Send_char(Recv_Dat);     // 数据回发

        Uart_Send_char(0x0D);         // 回车换行
        Uart_Send_char(0x0A);

    }
}
```

主函数中主要实现了以下步骤:

4. 初始化 LED 灯 led_init(): 设置 P1.0 , P1.1, P1.2 为普通 I/O 口, 设置 P1 方向为输出。
5. 初始化串口 uart0_init(): 配置 I/O 口、设置波特率为 19200、奇偶校验位和停止位。
6. 在主函数中使用 while(1)等待串口数据的接收和显示即可。

其中, 串口初始化的实现代码如下:

```
/*uart0 初始化-----*/
void uart0_init(unsigned char StopBits,unsigned char Parity)
{
    P0SEL |= 0x0C;           // 初始化 UART0 端口
    PERCFG&= ~0x01;         // 设置 IO 功能为 UART0
    P2DIR &= ~0xC0;         // P0 优先作为串口 0
    U0CSR = 0xC0;           // 设置为 UART 模式, 而
    且使能接受器
```

```

        U0GCR = 0x09;
        U0BAUD = 0x3b;           // 设置 UART0 波特率为
19200bps(通过查表来设置)

        U0UCR |= StopBits|Parity;           // 设置停止位与奇偶校
验
    }

```

上述代码首先配置 UART0 所对应的 I/O 口:通过对 PECFRG.0 清零来设置 IO 功能为 UART0,即 RXD 对应 P0.2, TXD 对应 P0.3。然后配置 P0.2 和 P0.3 为外部设备 I/O。然后选择 UART 模式,并使能接收器。接着配置 UART0 的参数:波特率 19200,无奇偶校验、停止位为 1。

接下来就是实现串口接收和发送数据,通过下面的代码来解析串口收发数据的代码实现过程:

串口接收函数:

```

/*串口接收字节函数-----*/
int Uart_Recv_char(void)
{
    int ch;           // 定义变量

    while (URX0IF == 0);           // 接收标志为 0 则等待
    ch = U0DBUF;           // 读取数据到变量 ch
    URX0IF = 0;           // 接收标志清零
    return ch;           // 返回读取值
}

```

串口发送函数:

```

/*串口发送字节函数-----*/
void Uart_Send_char(char ch)
{
    U0DBUF = ch;           // 写数据到 U0DBUF
    while(UTX0IF == 0);           // 发送标志为零则等待
    UTX0IF = 0;           // 清发送标志
}

```

程序流程

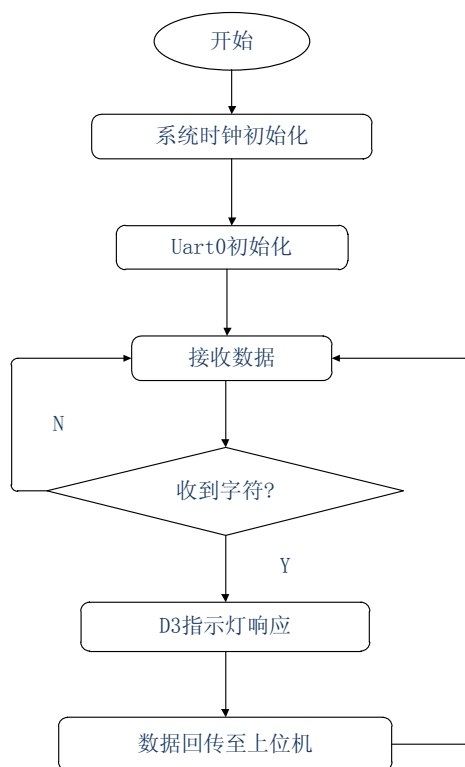


表 0.1 Uart 收发流程

10.4 实验步骤

1. 连接 ZB-LINK 调试器与 ZigBee 模块。即：用 RJ11 连接线将 ZB-LINK 的 RS-485 一端连接 ZigBee 模块 RJ11 对应口，用 USB3.0 数据线连接 ZB-LINK 的 Debug 端口与 ZigBee 模块 Debug 对应口。
2. 使用 USB2.0 方口线连接 ZB-LINK 调试器的 USB-Debug 端与 PC 机的 USB 口。
3. 使用 USB2.0 方口线连接 ZB-LINK 调试器的 USB-485 端与 PC 机的 USB 口。
4. 在 PC 机上打开串口终端软件，设置好波特率为 19200。打开：【配套光盘\04-实验例程\3 第三章基础技术实验区\3.13 Uart 串口通信实验\实验代码】中的：“uart.eww”，选择 Project→Rebuild All 重新编译工程。
5. 接下来选择 Project→Download and debug 将程序下载到 ZigBee 模块。
6. 下载完后按下 ZB-LINK 调试器上的复位按钮让刚才下载的程序重新运行。
7. 程序运行后，在 PC 机上的串口调试助手中会看到串口输出“Please Input string！”。在 PC 端通过键盘发送数据到 CC2530，然后观察 CC2530 回送给 PC 机的数据，并同时观察 ZigBee 模块上的 LED 指示灯 User1 的闪烁情况。

10.5 实验结果

如下图所示：

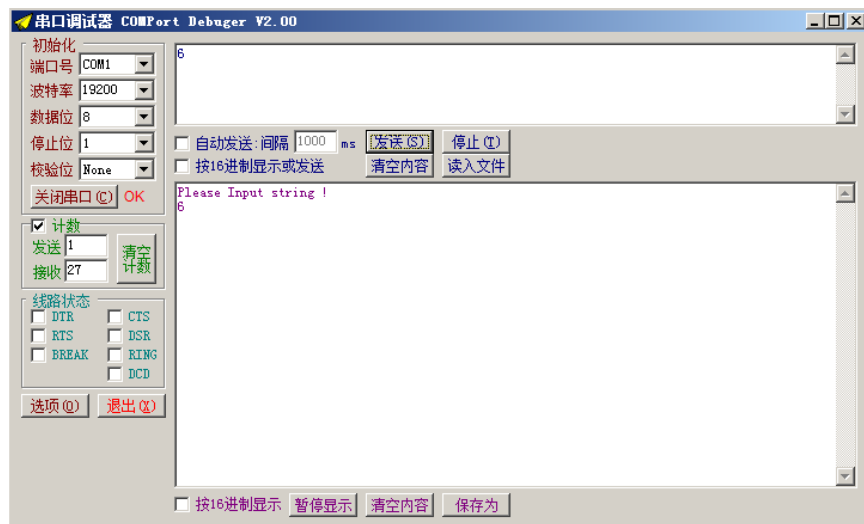


图 0.1 Uart 程序运行结果

系统复位后，首先显示一行提示字符串：“Please Input string !”提示输入数据，当上位机 PC 发送一个数据时，CC2530 就向上位机回传一个数据。与此同时，ZigBee 模块上的 User1 在每收到一个由 PC 端上位机发送过来的数据时就闪烁一次，表示 CC2530 已通过 Uart 串行口收到了数据，从而最终完成了本节的 Uart 串口通信实验。

实验十一：基础区-无线自组织网络串口通信

11.1 实验目的

1. 了解 Uart 串口应用的相关知识;
2. 了解 Z-Stack 协议栈及相关知识;
3. 掌握 Z-Stack 协议栈下 zigbee 终端节点与协调器串口通信原理。

11.2 实验环境

1. 硬件平台：ZigBee 模块 2 块（其中：一个作为节点 1，另一个作为节点 2）、ZB-LINK 调试器、ST-Link 调试器、USB3.0 数据线、USB2.0 方口线两根、RJ11 连接线;
2. 软件平台：WinXP/Win7、IAR 开发环境、串口调试工具。

11.3 实验原理

串口是开发板和用户电脑交互的一种工具，正确地使用串口对于 ZigBee 无线网络的学习具有较大的促进作用，使用串口的基本步骤：

1. 初始化串口，包括设置波特率，中断等;
2. 向发送缓冲区发送数据或者从接收缓冲区读取数据。

上述方法是使用串口的常用方法，但是由于 ZigBee 协议栈的存在，使得串口的使用略有不同，在 ZigBee 协议栈中已经对串口初始化所需要的函数进行了实现，用户只需要传递几个参数就可以使用串口，此外，ZigBee 协议栈还实现了串口的读取函数与写入函数。

因此，用户在使用串口时，只需要掌握 ZigBee 协议栈提供的串口操作相关的三个函数即可。ZigBee 协议栈中提供的与串口操作相关的三个函数为：

- (1) 接收串口数据及发送函数：static void rxCB(uint8 port,uint8 event);
- (2) 串口发送终端节点短地址函数：void Send_Info(void);
- (3) 串口打印函数：void MSG_Handler(aflIncomingMSGPacket_t *pkt);

ZigBee 协议栈中串口通信的配置使用一个结构体来实现，该结构体为 hal_UARTCfg_t，不必关心该结构体的具体定义形式，只需要对其功能有个了解，该结构体将串口初始化的参数集合在一起，只需要初始化各个参数即可最后使用 HalUARTOpen()函数对串口进行初始化。

电路分析与程序配置

本节实验所用到的是：LED 接口电路，Uart 通信接口及 RF 射频接口，原理图如下：



“Uart 与 RF 接口电路”中 CC2530 的 P0_2,P0_3 即串口 0 与 PC 机通信。

“Uart 与 RF 接口电路”中 CC2530 的 P0_2,P0_3 即串口 0 与 PC 机通信。

程序流程

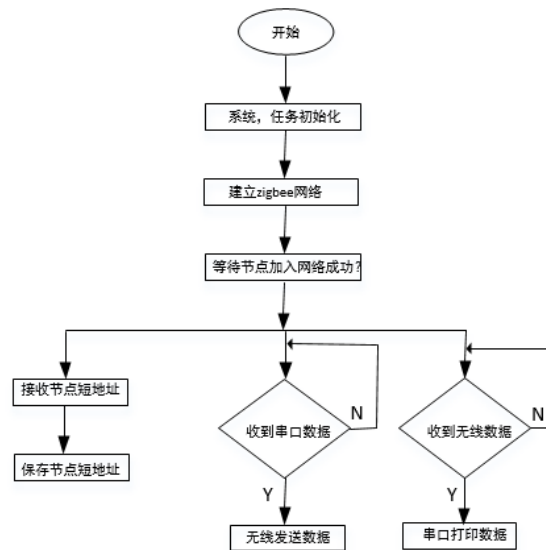


图 0.1 协调器通信流程

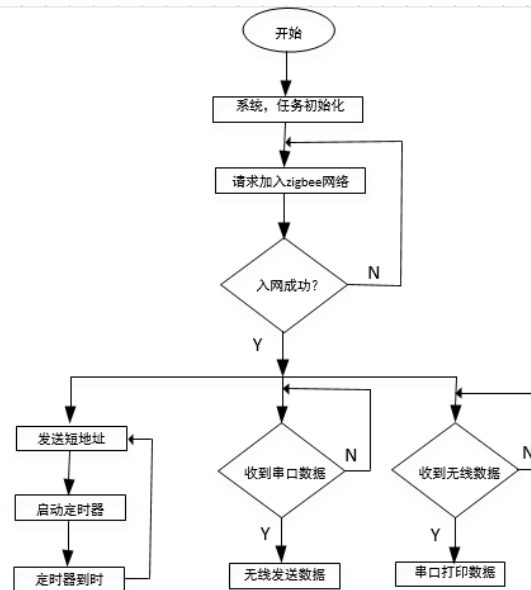


图 0.2 终端节点通信流程

使用 2 个 ZigBee 模块，并分别作为协调器及终端节点。程序运行后，通过上位机 PC 将数据通过串口发送给终端节点。终端节点收到数据后，再通过 ZigBee 网络将该数据发送至协调器，当协调器收到该数据后，通过串口打印出来，同样，通过上位机 PC 将数据通过串口发送给协调器，协调器收到数据后，再通过 ZigBee 网络将该数据发送至终端节点，当终端节点收到该数据后，通过串口打印出来从而完成协议栈下的串口通信过程，本节实验数据流程图如下：

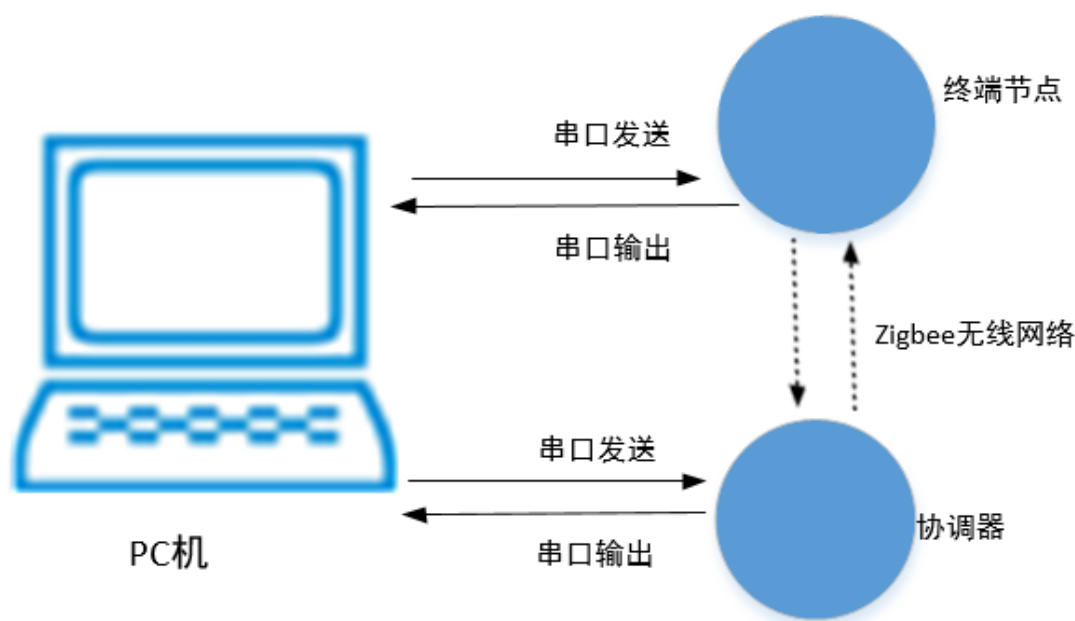


图 0.3 协议栈下串口应用数据流程图

源码分析

本节实验源码分为：协调器及终端节点，各文件相关代码如下：

1. 协调器(Coodinator.c):

/*函数功能：协调器协议栈相关功能初始化*/

```

void GenericApp_Init(byte task_id)
{
    halUARTCfg_t uartConfig;
    GenericApp_TaskID = task_id; //初始化任务的优先级
    GenericApp_TransID = 0; //将发送数据包的序号初始化为0
    GenericApp_epDesc.endPoint = GENERICAPP_ENDPOINT;
    GenericApp_epDesc.task_id = &GenericApp_TaskID;
    GenericApp_epDesc.simpleDesc = (SimpleDescriptionFormat_t
*)&GenericApp_SimpleDesc;
    GenericApp_epDesc.latencyReq = noLatencyReqs; //上述四行是对节点描述符进行初始化，不需要修改
    afRegister(&GenericApp_epDesc);
    uartConfig.configured = TRUE; //确定对串口参数进行配置
    uartConfig.baudRate = HAL_UART_BR_115200; //串口波特率为115200
    uartConfig.flowControl = FALSE; //没有流控制
    uartConfig.callBackFunc = rxCB; //串口回调函数
    HalUARTOpen(0,&uartConfig); //根据uartConfig结构体变量中的成员，初始化串口0
    HalLedBlink(HAL_LED_2,0,50,250); //使LED灯2开始闪烁
}

```

/*函数功能：协调器建立网络，并启动事件处理进程*/

```

UINT16 GenericApp_ProcessEvent(byte task_id,UINT16 events)
{
    afIncomingMSGPacket_t *MSGpkt; //定义一个指向接收消息结构体的指针
    MSGpkt;
    if(events & SYS_EVENT_MSG)
    {

```



```

MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive(GenericApp_TaskID);
while(MSGpkt)
{
    switch(MSGpkt->hdr.event)
    {
        case ZDO_STATE_CHANGE:
            HalLedSet(HAL_LED_2, HAL_LED_MODE_ON);    //协调器组网成功后, LED 灯 2
            break;
        case AF_INCOMING_MSG_CMD:
            GenericApp_MessageMSGCB(MSGpkt);          //对收到的数据进行处理
            break;
        default:
            break;
    }
    osal_msg_deallocate((uint8 *)MSGpkt);
    MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive(GenericApp_TaskID);
}
return (events ^ SYS_EVENT_MSG);
}
return 0;
}
void zb_StartConfirm( uint8 status )                // ZigBee 启动配置函数
{
    if ( status == ZB_SUCCESS )                    // 如果 ZigBee 设备启动成功
    {
        myAppState = APP_START;                    // 设置 APP 状态为启动
        HalLedSet( HAL_LED_2, HAL_LED_MODE_ON );    // 打开 LED2
    }
    else
    {
        osal_start_timerEx( sapi_TaskID, MY_START_EVT, myStartRetryDelay );
    }
}

```

/*函数功能:协调器收到终端点发过来的数据并从串口 0 打印出来, 协调器保存由终端节点上传的自身短地址*/

```

void GenericApp_MessageMSGCB(afIncomingMSGPacket_t *pkt)
{
    char buf[30] = 0;                                //接收数据缓冲区
    switch(pkt->clusterId)
    {
        case GENERICAPP_CLUSTERID_1:                //数据号为 GENERICAPP_CLUSTERID_2
            osal_memcpy(buf, pkt->cmd.Data, pkt->cmd.DataLength);
            HalUARTWrite(0, buf, pkt->cmd.DataLength); //从串口 0 打印出接收到的数据
            break;
        case GENERICAPP_CLUSTERID_2:                //节点自身地址信息的数据号为
            osal_memcpy(&info, pkt->cmd.Data, 2);
            break;
    }
}

```

/*函数功能: 用于协调器读取串口数据并将数据发送出去*/

```

static void rxCB(uint8 port, uint8 event)
{
    uint8 buf[128];
    uint8 len = 0;
    afAddrType_t my_DstAddr;
    len = HalUARTRead(0, buf, 128);                //读取串口数据, 并将数据存储在
    if(len != 0)
    {
        my_DstAddr.addrMode = (afAddrMode_t)Addr16Bit;
    }
}

```

```

        my_DstAddr.endPoint = GENERICAPP_ENDPOINT; //初始化端口号
        my_DstAddr.addr.shortAddr = info;          //目标节点短地址
        AF_DataRequest(&my_DstAddr,&GenericApp_epDesc,GENERICAPP_CLUSTERID_1,
            len,buf,&GenericApp_TransID,AF_DISCV_ROUTE,AF_DEFAULT_RADIUS);
    }
}

```

1. 终端节点(Enddevice.c):

终端节点串口接收来自协调器串口发过来的数据与协调器串口接收终端节点串口发过来的数据过程基本是一致的,只是多了往协调器上传自身短地址的步骤,具体代码如下:

/*函数功能: 获取自身短地址, 并且将自身短地址发送出去*/

```

void Send_Info(void)
{
    uint16 nwk;
    afAddrType_t my_DstAddr;
    nwk = NLME_GetShortAddr();
    my_DstAddr.addrMode = (afAddrMode_t)Addr16Bit;
    my_DstAddr.endPoint = GENERICAPP_ENDPOINT; //初始化端口号
    my_DstAddr.addr.shortAddr = 0x0000;       //目标节点短地址。协调器地
短地址始终为 0x0000

    AF_DataRequest(&my_DstAddr,&GenericApp_epDesc,GENERICAPP_CLUSTERID_2,
        2,(uint8
        *)&nwk,&GenericApp_TransID,AF_DISCV_ROUTE,AF_DEFAULT_RADIUS);
}

```

/*函数功能: 收到协调器过来的无线数据并将数据从串口 0 打印出来*/

```

void MSG_Handler(afIncomingMSGPacket_t *pkt)
{
    char buf[30] = 0; //接收数据缓冲区
    switch(pkt->clusterId)
    {
        case GENERICAPP_CLUSTERID_1: //数据号为
GENERICAPP_CLUSTERID_1 表明是用户数据
        {
            osal_memcpy(buf,pkt->cmd.Data,pkt->cmd.DataLength); //将收到的从协调器传过
来的数据拷贝到 buf
            HalUARTWrite(0,buf,pkt->cmd.DataLength); //从串口 0 打印出接收到的数据
            break;
        }
    }
}

```

/*函数功能: 终端节点读取从串口过来的数据并且将串口数据无线发送出去*/

```

static void rxCB(uint8 port,uint8 event)
{
    uint8 buf[128];
    uint8 len = 0;
    afAddrType_t my_DstAddr;
    len = HalUARTRead(0,buf,128); //读取串口数据, 并将数据存
储在 buf 缓冲区中
    if(len)
    {
        HalUARTWrite(0,buf,len); //从串口 0 打印出接收到的数
据

        my_DstAddr.addrMode = (afAddrMode_t)Addr16Bit;
        my_DstAddr.endPoint = GENERICAPP_ENDPOINT; //初始化端口号
        my_DstAddr.addr.shortAddr = 0x0000;       //目标节点短地址。协调器地短地
址始终为 0x0000
        AF_DataRequest(&my_DstAddr,&GenericApp_epDesc,GENERICAPP_CLUSTERID_1,
            len,buf,&GenericApp_TransID,AF_DISCV_ROUTE,AF_DEFAULT_RADIUS);
    }
}

```

11.4 实验步骤

注：由于出厂源码 ZigBee 网络 PANID 均设置为 0x2100，为了避免实验环境下多个实验平台之间网络互相串扰，每个实验平台需要修改 PAD ID，修改工程内文件：Tools -> f8wConfig.cfg，将 PANID 修改为个人学号的后四位（范围 0x0001~0x3FFF）。

1. 将：【配套光盘\04-实验例程\第 3 章基础技术实验区\3.16 无线自组织网络串口通信\实验代码】中的文件夹：End AND Coor 复制到路径：【C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples】中。

2. 使用 IAR 软件打开该文件夹中的工程“GenericApp.eww”，然后选择 Workspace 为 CoordinatorEB，之后点击 Project→Rebuild All 对工程进行全编译并等待编译完成。

1. 使用 USB3.0 数据线连接 1 个 ZigBee 模块与 ZB-LINK 调试器的 Debug 端，使用 USB 方口线连接 ZB-LINK 调试器的 US-Debug 端与 PC 机的 USB 口。

2. 打开程序下载软件 SmartRF Flash Programmer，并在 Flash image 中选择路径为：【C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples\End_AND_Coor \CC2530DB\CoordinatorEB\Exe】下的 GenericApp.hex 文件，然后点击 Perform actions 对其进行程序下载。

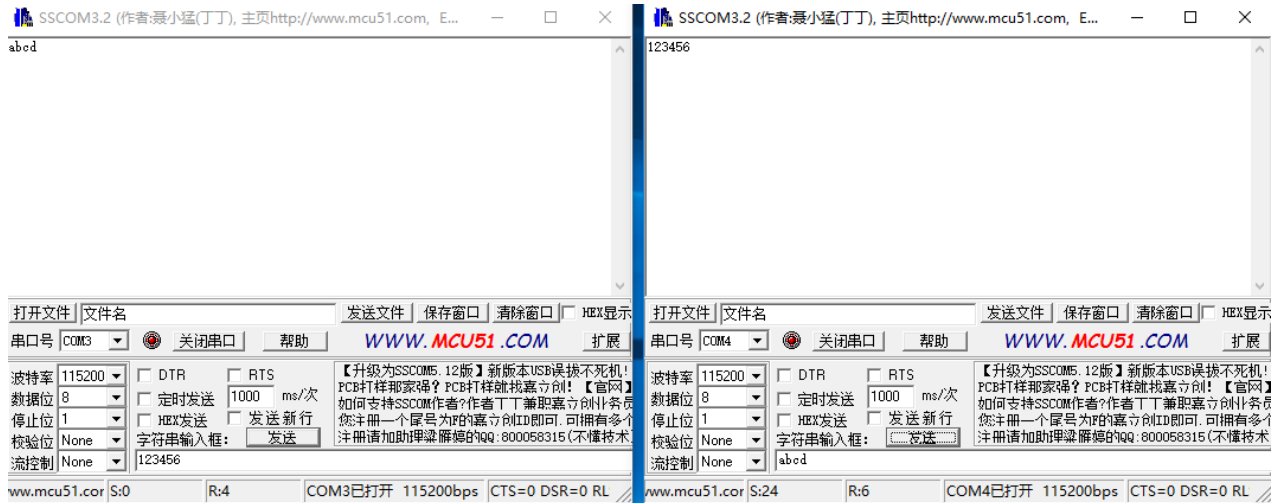
3. 断开此 ZigBee 上的 USB3.0 数据线，将其接到另外一个 ZigBee 模块上，再回到 IAR 开发环境，选择编译选项为 EndDeviceEB 然后参照步骤(2)~(4)将【C:\Texas Instruments\ZStack-CC2530-2.4.0-1.4.0\Projects\zstack\Samples\End_AND_Coor \CC2530DB\EndDeviceEB\Exe】GenericApp.hex 文件下载到该模块 ZigBee 中。

4. 所有模块下载完成后，用一根 RJ11 线的一端接到 ZigBee 模块 1，另一端连接到 ZB-LINK 调试器的 RS-485 端，然后再用另外一个 RJ11 线的一端连接到 Zigbee 模块 2，另一端线连接到 ST-LINK 调试器的 RS-485 端，再使用两根 USB 方口线分别一端连接 ZB-LINK 调试器和 STLINK 调试器的 USB-485 口另一端连接 PC 机的两个 USB 口。

5. 分别打开两个串口调试助手软件“串口调试工具”，将波特率设置为 115200，并选择好 COM 口后打开串口。

11.5 实验结果

当程序运行以后，打开两个串口助手，其中一个为终端节点的串口，一个是协调器的串口，两个串口助手之间可以互发数据。具体效果如下图所示。



实验十二：家居区-热释电传感器采集实验

12.1 实验目的

1. 了解 CC2530 I/O 口的特点与特性；
2. 学习并掌握 CC2530 I/O 口的基本配置；
3. CC2530 Uart 串口通信配置及程序编写；
4. 热释电红外传感器的工作原理及采集方式；

12.2 实验环境

1. 硬件：1 个 Zigbee 模块，1 个热释电红外传感器（智能家居区域面板）、1 个 ZB-Link 调试器、1 根 USB2.0 方口线、1 根 USB3.0 数据线、1 根 RJ11 电话线、1 根测试连接线，1 台 PC 机；
2. 软件：Windows 7/XP、IAR 集成开发环境、串口调试工具。

12.3 实验原理



图 4.2.3 Zigbee 模块



图 4.2.4 智能家居区域热释电红外传感器

热释电红外线传感器主要是由一种高热电系数的材料，如锆钛酸铅系陶瓷、钽酸锂、硫

酸三甘钛等制成尺寸为 $2 \times 1\text{mm}$ 的探测元件。在每个探测器内装入一个或两个探测元件，并将两个探测元件以反极性串联，以抑制由于自身温度升高而产生的干扰。由探测元件将探测并接收到的红外辐射转变成微弱的电压信号，经装在探头内的场效应管放大后向外输出。为了提高探测器的探测灵敏度以增大探测距离，一般在探测器的前方装设一个菲涅尔透镜，该透镜用透明塑料制成，将透镜的上、下两部分各分成若干等份，制成一种具有特殊光学系统的透镜，它和放大电路相配合，可将信号放大 70 分贝以上，这样就可以测出 10~20 米范围内人的行动。

菲涅尔透镜利用透镜的特殊光学原理，在探测器前方产生一个交替变化的“盲区”和“高灵敏区”，以提高它的探测接收灵敏度。当有人从透镜前走过时，人体发出的红外线就不断地交替从“盲区”进入“高灵敏区”，这样就使接收到的红外信号以忽强忽弱的脉冲形式输入，从而增强其能量幅度。

人体辐射的红外线中心波长为 $9 \sim 10\text{--}\mu\text{m}$ ，而探测元件的波长灵敏度在 $0.2 \sim 20\text{--}\mu\text{m}$ 范围内几乎稳定不变。在传感器顶端开设了一个装有滤光镜片的窗口，这个滤光片可通过光的波长范围为 $7 \sim 10\text{--}\mu\text{m}$ ，正好适合于人体红外辐射的探测，而对其它波长的红外线由滤光片予以吸收，这样便形成了一种专门用作探测人体辐射的红外线传感器。

人体都有恒定的体温，一般在 37 度，所以会发出特定波长 $10\mu\text{m}$ 左右的红外线，被动式红外探头就是靠探测人体发射的 $10\mu\text{m}$ 左右的红外线而进行工作的。人体发射的 $10\mu\text{m}$ 左右的红外线通过菲涅尔滤光片增强后聚集到红外感应源上。红外感应源通常采用热释电元件，这种元件在接收到人体红外辐射温度发生变化时就会失去电荷平衡，向外释放电荷，后续电路经检测处理后就能产生报警信号。

热释电红外传感器用来实时监测是否有人进入测量范围，如果没有检测到有人，将输出低电平信号，同时指示灯熄灭，如图 4.2.5 所示；如果检测到有人，将输出高电平信号，同时传感器中部的指示灯将被点亮为红色，如下图 4.2.6 所示。



图 0.5 热释电传感器指示灯熄灭



图 0.6 热释电传感器指示灯被点亮

热释电红外传感器具有以下几个特点：

- 启动时间：通电 60 秒后；
- 检测范围： 7.5×6 米（安装高度为 2.4 米）；
- 信号输出：开关量信号输出（高、低电平）；
- 警报显示：红色发光二极管，连续发光 3 秒。

硬件原理

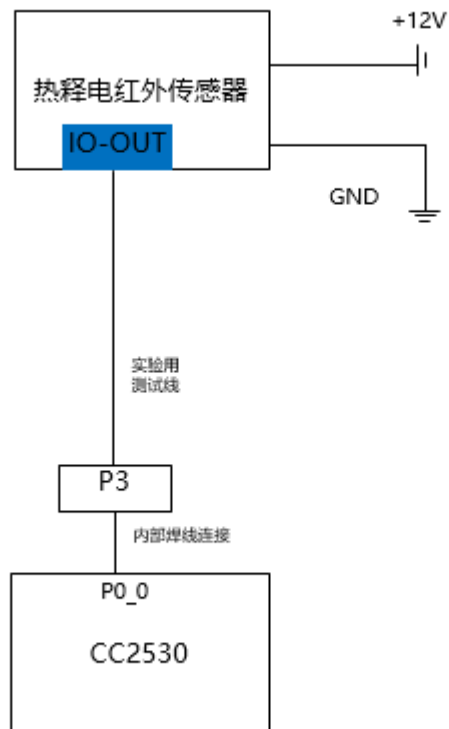


图 4.2.7 智能家居区域热释电红外传感器硬件接线图

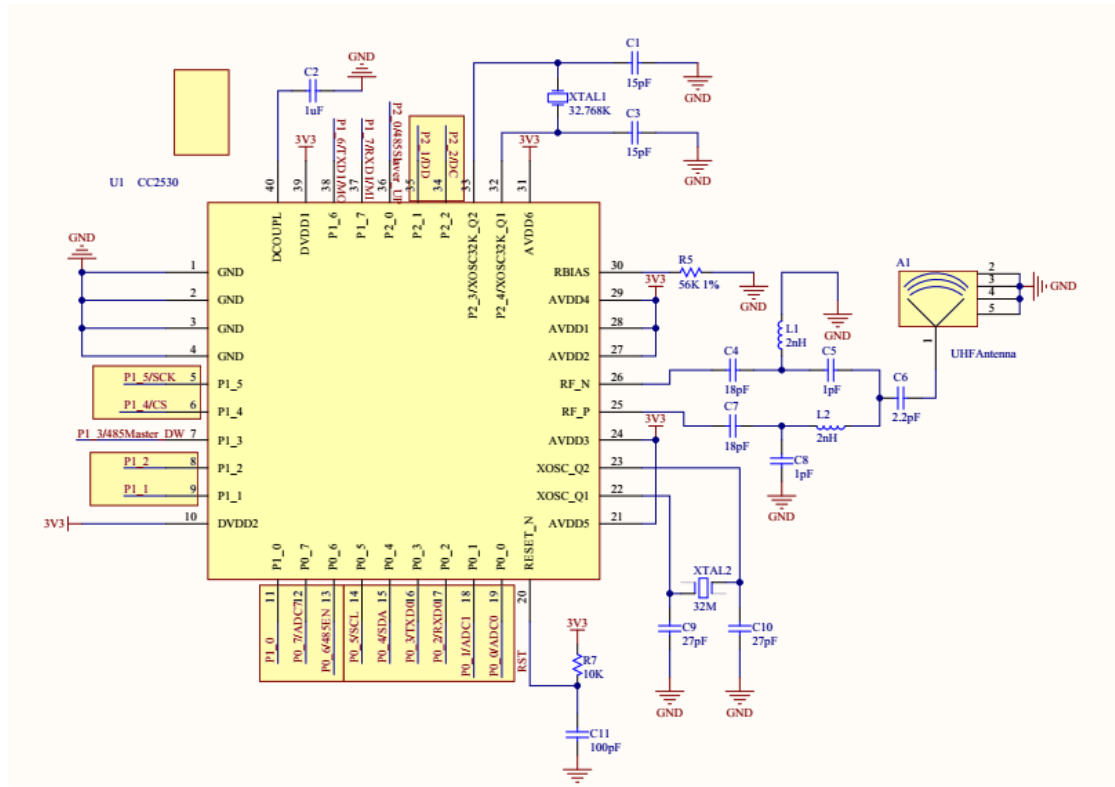


图 4.2.8 Zigbee 通用节点原理图

编程思路

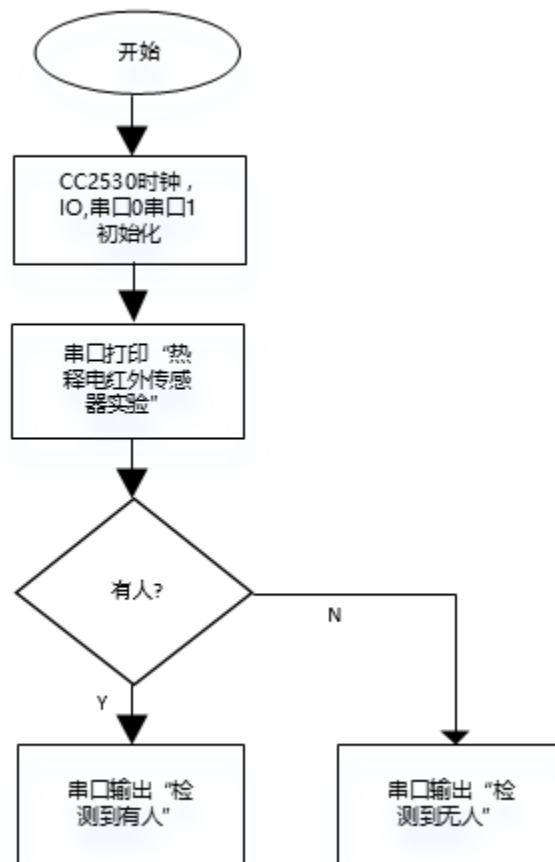


图 4. 2. 9 编程流程图

源码解析

本节实验的源码文件在【配套光盘\05-实验例程\04-第四章 智能家居实验\4.1 热释电传感器采集实验】目录下，工程里主要包括“sys_init.c”，“uart.c”和主程序文件“main.c”。下面对这些文件中的主要函数进行具体解析。

1.系统时钟初始化:

```
void xtal_init(void)
{
    SLEEP_CMD &= ~0x04;           //都上电
    while(!(CLKCONSTA & 0x40));    //晶体振荡器开启且稳定
    CLKCONCMD &= ~0x47;           //选择 32MHz 晶体振荡器
    SLEEP_CMD |= 0x04;
}
```

2.人体红外初始化:

```
void Infrared_init(void)
{
```

```
P0SEL &=~0x01; //配置 P0_0 为普通 IO
    P0DIR &=~0x01; //配置 P0_0 为 输入 IO
}
```

3. 串口 0（485 通信）初始化：

```
void uart0_init(unsigned char StopBits,unsigned char Parity)
{
    P0SEL |= 0x0C;           //初始化 UART0 端口 P0_2, P0_3
    PERCFG&= ~0x01;         //选择 UART0 为可选位置一
    U0CSR = 0xC0;            //设置为 UART 模式,而且使能接受器
    U0GCR = 0x09;
    U0BAUD = 0x3b;           //设置 UART0 波特率为 19200bps
    U0UCR |= StopBits|Parity; //设置停止位与奇偶校验
    P0SEL &= ~0x40;          //P0_6 为普通 I/O 口(485 控制端)
    P0DIR |= 0x40;           //P0_6 为输出端口(485 控制端)
    P0_6 = 1;
}
```

4. 串口发送字节函数：

```
void Uart_Send_char(char ch)
{
    U0DBUF = ch;
    while(UTX0IF == 0);
    UTX0IF = 0;
}
```

5. 串口接收字节函数：

```
int Uart_Recv_char(void)
{
    int ch;
    while (URX0IF == 0);
    ch = U0DBUF;
    URX0IF = 0;
    return ch;
}
```

6. 延时函数：

```
void delay(unsigned int i)
{
    unsigned int j ,k;
    for(j=i;j>0;j--)
        for(k=110;k>0;k--);
}
```


```

}

6. 主函数:
void main(void)
{
    int value;
    Infrared_init();          //人体红外硬件初始化
    xtal_init();              //系统时钟初始化
    uart0_init(0x00, 0x00);   //初始化串口: 无奇偶校验, 停止位为 1 位
    Uart_Send_String("人体红外传感器实验\r\n");
    P0_6 = 0;                 //设置为接收模式(485 控制端)
    while(1)
    {
        value = P0_0;
        if(value == 1)
        {
            Uart_Send_String("检测到有人\r\n"); //当 P0_0 上传为“1”时串口输出“检测到有人”
            delay(6000);
        }
        if(value == 0)
        {
            Uart_Send_String("检测到无人\r\n"); //当 P0_0 上传为“0”时串口输出“检测到无人”
            delay(6000);
        }
    }
}

```

12.4 实验步骤

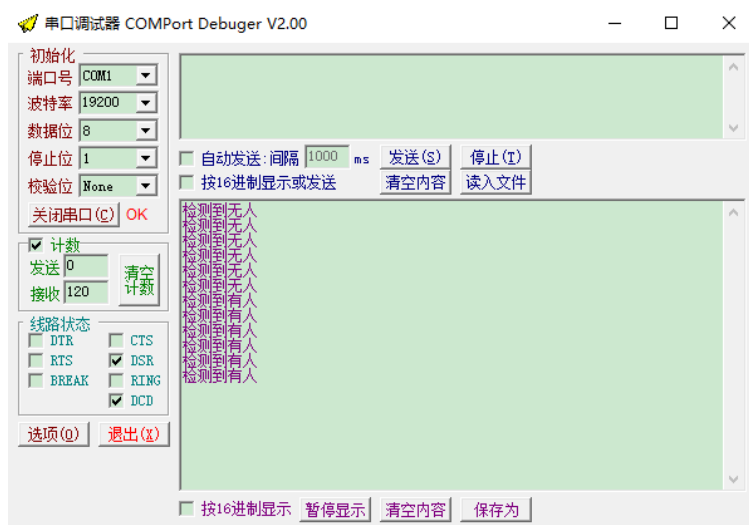
1. 将 USB 方口线的方口接 ZB-LINK 调试器的“USB-Debug”口，USB 口接电脑，USB3.0 下载线一端接 ZBLINK 调试器的“Debug”另一端接 Zigbee 模块 USB3.0 的下载接口。
2. 用 IAR 软件打开路径为：【配套光盘\04-实验例程\第 4 章 智能家居实训区\4.1 热释电红外传感器】中的实验工程“Infrared.eww”，选择 Project→Rebuild All 重新编译工程；
3. 选择 Project→Download and debug 将程序下载到 ZigBee 模块(如果下载出现问题，请按 ZB-LINK “RST” 复位键复位，再次下载)，下载完成后点退出调试模式，并按下 ZB-LINK 调试器上的复位按钮让刚才下载的程序重新运行。
4. 将下载好程序的节点安装在智能家居区域面板上的节点 2 上，用测试线将“IO-OUT”端接在节点 2 的“P3”端口上。
5. 将“RJ11 电话线”的一端连接“Zigbee 模块”的“RJ11-485”通信接口，另一端连

接“ZB-Link 调试器”的“RS-485”接口，用“USB-A 转 B 一端接”“ZB-Link 调试器”的“USB-485”，另一端接 PC 的 USB 口。

- 在【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】路径下打开“Comdebug.exe”串口助手，在“设备管理器”-“端口”处查看串口号，在“串口调试工具”里选择对应的 COM 口，波特率设置为“19200”，点击“打开串口”即可。

12.5 实验结果

当有人在热释电红外传感器附近活动时候，热释电传感器灯亮，串口调试工具会显示“检测到有人”，当无人在热释电红外传感器附近活动时，热释电红外传感器灯熄灭，串口调试工具会显示“检测到无人”，如下图：



实验十三：家居区-光照度传感器采集实验

13.1 实验目的

1. 了解 CC2530 I/O 口的特点与特性；
2. 学习并掌握 CC2530 I/O 口的基本配置；
3. CC2530 485 串口通信配置及程序编写；
4. 光照传感器的工作原理及采集方式；

13.2 实验环境

1. 硬件：1 个 Zigbee 模块，1 个光照传感器（智能家居区域面板）、1 个 ZB-Link 调试器、1 根 USB2.0 方口线、1 根 USB3.0 数据线、1 根 RJ11 电话线、1 根测试连接线，1 台 PC 机；
2. 软件：Windows 7/XP、IAR 集成开发环境、串口调试工具。

13.3 实验原理



图 4.5.3 Zigbee 模块



图 4.5.4 智能家居区域光照度传感器

智能家居实训区共用了一个光照度传感器，用来模拟智能家居内光照强度的检测，并将光照强度数据通过 485 总线输出。

硬件原理

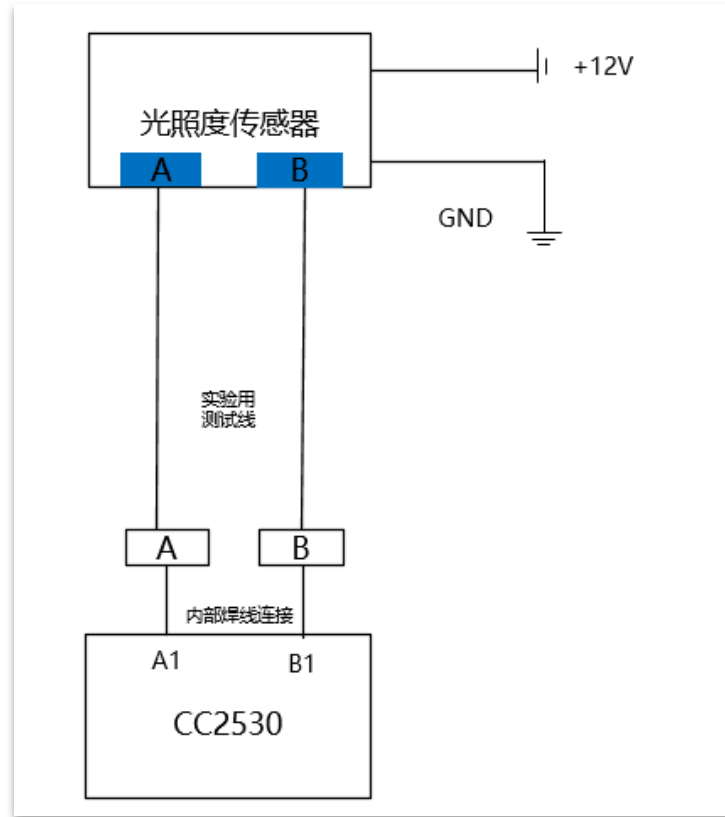


图 0.1 智能家居区光照传感器硬件接线图

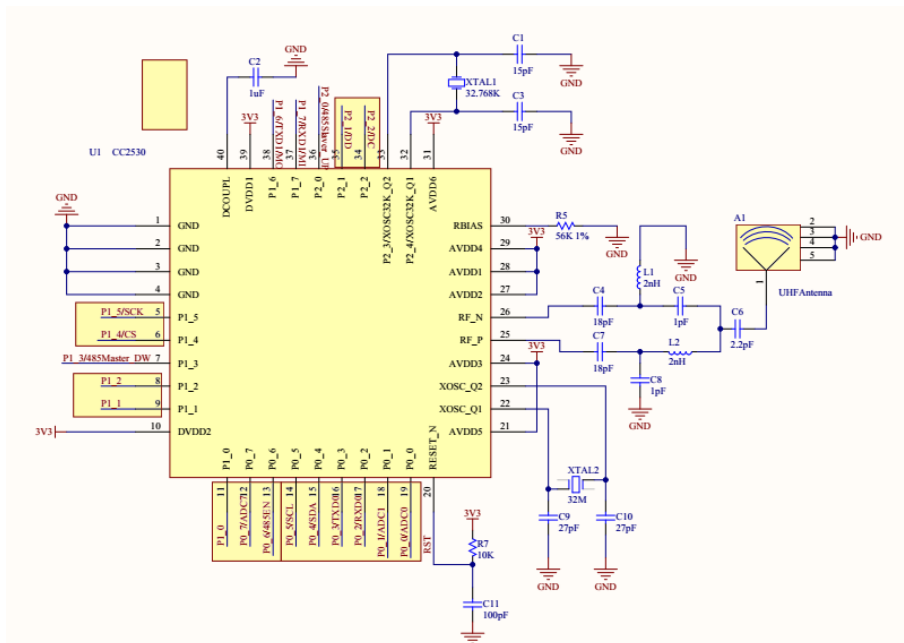
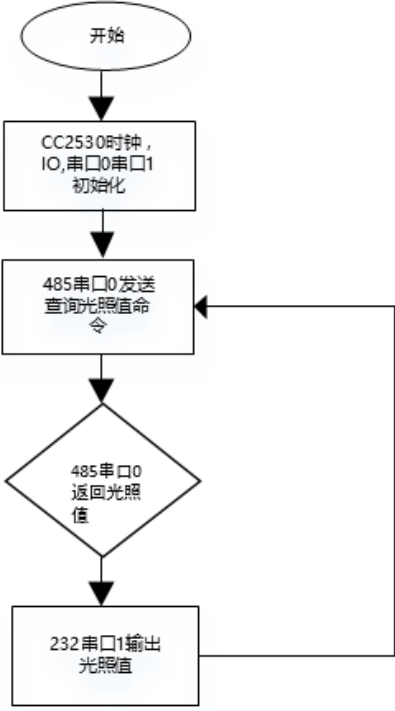
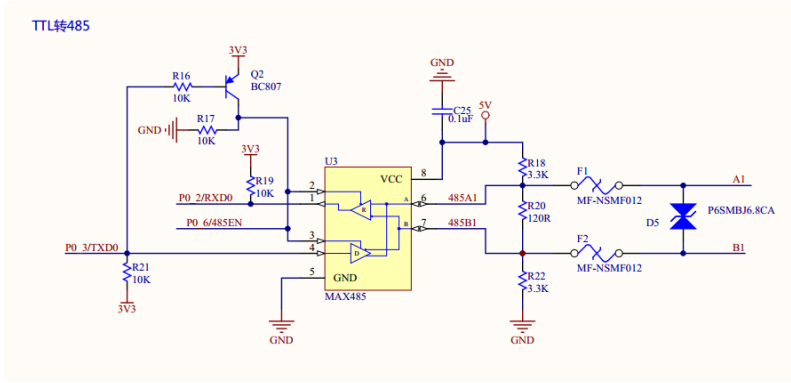
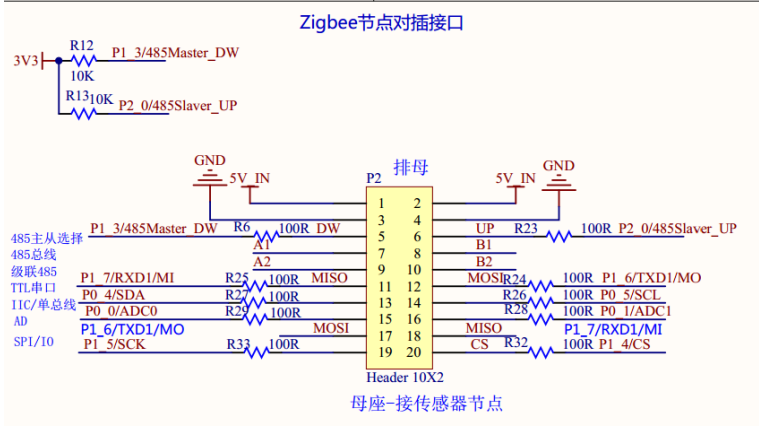


图 0.2 智能家居区光照传感器硬件接线图



源码解析

本节实验的源码文件在【配套光盘\05-实验例程\04-第四章 智能家居实验\4.4 光照度传感器】目录下，工程里主要包括“sys_init.c”，“uart.c”和主程序文件“main.c”。下面对这些文件中的主要函数进行具体解析。

1. 系统时钟初始化：

```
void xtal_init(void)
{
    SLEEPCMD &= ~0x04;           //都上电

    while(!(CLKCONSTA & 0x40));   //晶体振荡器开启且稳定

    CLKCONCMD &= ~0x47;           //选择 32MHz 晶体振荡器

    SLEEPCMD |= 0x04;
}
```

2. 串口初始化：

```
void uart1_init(void)
{
    P1SEL |= 0xC0;                //初始化 UART1 端口,P1_6, P1_7 作为串口

    PERCFG |= 0x02;               //选择 UART1 为可选位置二

    U1CSR = 0xC0;                 //设置为 UART 模式,而且使能接受器

    U1GCR = 0x09;

    U1BAUD = 0x3b;                //设置 UART0 波特率为 19200bps

    U1UCR |= 0x00;               //设置停止位与奇偶校验
}
```

3. 串口 0（485 通信）初始化：

```
void uart0_init(void)
{
    P0SEL |= 0x0C;                //初始化 UART0 端口 P0_2, P0_3 作为外设, 设置此即可

    PERCFG&= ~0x01;              //选择 UART0 为可选位置一

    U0CSR = 0xC0;                 //设置为 UART 模式,而且使能接受器

    U0GCR = 0x08;

    U0BAUD = 0x3b;                //设置 UART0 波特率为 9600bps

    U0UCR |= 0x00;               //设置停止位与奇偶校验

    UTX0IF = 0;                  //清除 USRT0 发送中断标志位

    URX0IF = 0;                  //清除 USRT0 接收中断标志位

    IEN0 |= 0x84;                //开总中断, 使能串口 0 接收中断
}
```



```
P0SEL &= ~0x40;          //P1.0 P1.1 为普通 I/O 口
P0DIR |= 0x40;           //输出
P0_6 = 1;
}
```

4. 串口发送字节函数:

```
void Uart_Send_char(char ch)
{
    U0DBUF = ch;
    while(UTX0IF == 0);
    UTX0IF = 0;
}
```

5. 串口接收字节函数:

```
int Uart_Recv_char(void)
{
    int ch;
    while (URX0IF == 0);
    ch = U0DBUF;
    URX0IF = 0;
    return ch;
}
```

6. 串口发送指定长度字符串函数

```
void Uart_Send_nString(char *Data,int len)
{
    while(len--)
        Uart_Send_char(*Data++);
}
```

7. 延时函数:

```
void delay(unsigned int i)
{
    unsigned int j ,k;
    for(j=i;j>0;j--)
        for(k=110;k>0;k--);
}
```

8. 主函数:


```
void main(void)
{
    xtal_init();          //系统时钟初始化
}
```

```

uart0_init();           //初始化串口：无奇偶校验，停止位为 1 位
uart1_init();           //初始化串口 1：无奇偶校验，停止位为 1 位
P0_6 = 0;               //设置为接收模式(485 控制端)
Uart_Send_nString(ReadLuxCmd,8); //发送获取光照度值的命令
while(1)
{
    if(Receive_len == 7) //接收的值为 7 个字节代表接收完毕
    {
        Receive_len = 0;
        Receive_state = 1; //接收完以后接收状态置 1
        pLux = (Receive_Value[3]<<8 | Receive_Value[4]); //转换接收的光照为十进制
        printf("光照值:%d Lux" ,pLux); //串口输出光照值
        delay(6000);
        P0_6 = 1; //设置 485 串口为接收模式
        Uart_Send_nString(ReadLuxCmd,8); //再次发送获取光照值的命令
        P0_6 = 0; //设置 485 串口为发送模式
        Receive_state = 0;
    }
}
}

```

13.4 实验步骤

1. USB A 转 B 连接线方口端接 ZB-LINK 调试器的“USB-Debug”口，USB 口接电脑，USB3.0 下载线一端接 ZBLINK 调试器的“Debug”另一端接“Zigbee”模块 USB3.0 的下载接口。
2. 用 IAR 软件打开路径为：【配套光盘\04-实验例程\第 4 章 智能家居实训区\4.4 光照传感器】中的实验工程“illumination.eww”，选择 Project→Rebuild All 重新编译工程；
3. 选择 Project→Download and debug 将程序下载到 ZigBee 模块（如果下载出现问题，请按 ZB-LINK “RST” 复位键复位，再次下载），下载完成后点  退出；
4. 将下载好程序的节点安装在智能家居区域面板上的节点 4 上，用测试线将光照度下面“A”端接在节点 4 的“A”端口上，“B”端接在节点 4 的“B”端口上。
5. 将“USB3.0”的一端连接“Zigbee 模块”的“USB3.0”的下载接口，另一端连接“ZB-Link 调试器”的“Debug”接口，用“USB-A 转 B 一端接”“ZB-Link 调试器”的“USB-232”，另一端接 PC 的 USB 口。
6. 在【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】路径下打开“Comdebug.exe”串口助手，在“设备管理器”-“端口”处查看串口号，在“串口调试工具”里选择对应的 COM 口，波特率设置为“19200”，点击“打开串口”即可。

13.5 实验结果

串口收到当前的光照值，当用手遮挡或者用手电筒照射光照传感器时会发现光照度值会改变，如下图：



实验十四：家居区-红外转发器应用实验

14.1 实验目的

1. 了解 CC2530 I/O 口的特点与特性；
2. 学习并掌握 CC2530 I/O 口的基本配置；
3. CC2530 Uart 串口通信配置及程序编写；
4. 红外转发器的工作原理及应用方式；

14.2 实验环境

1. 硬件：1 个 Zigbee 模块，1 个红外转发器（智能家居区域面板）、1 个 LED 灯（智能家居区域面板）、1 个 ZB-Link 调试器、1 根 USB2.0 方口线、1 根 USB3.0 数据线、1 根 RJ11 电话线、2 根测试连接线，1 台 PC 机；
2. 软件：Windows 7/XP、IAR 集成开发环境、串口调试工具。

14.3 实验原理



图 0.1 Zigbee 模块



图 0.2 红外转发器

红外转发器能在 360° 范围内模拟对智能家居中红外智能电器的控制。PC机通过 CC2530 的 Uart 串口 0 给 ZigBee 节点发送信号，ZigBee 节点再通过 Uart 串口 1 与红外转发器通信，红外转发器接收到的控制信号通过自身芯片转化为内部识别的控制电器的红外信号，实现控制电器的功能，还可以自动学习各种电器的遥控信号，控制距离为 0~20 米。

硬件原理

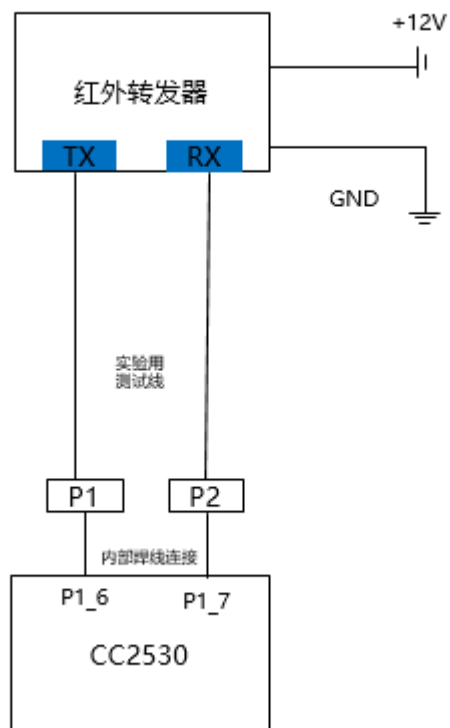


图 0.1 智能家居区域红外转发器器硬件接线图

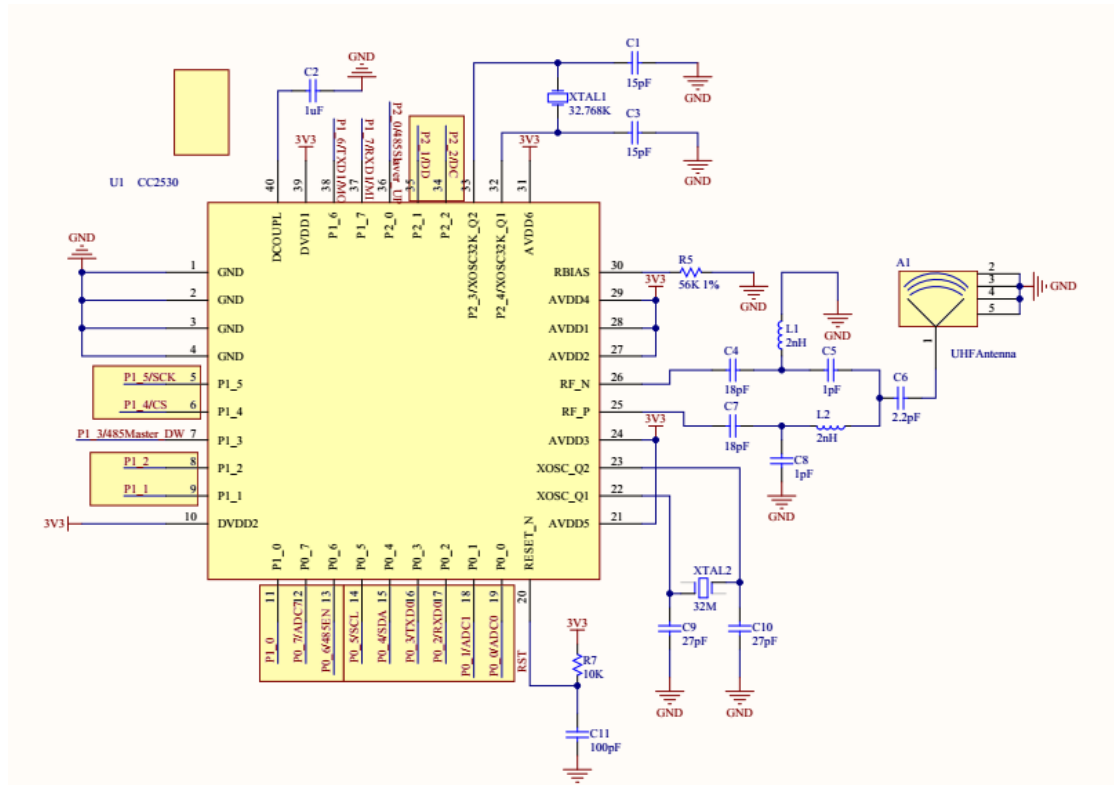


图 0.2 Zigbee 通用节点原理图

编程思路

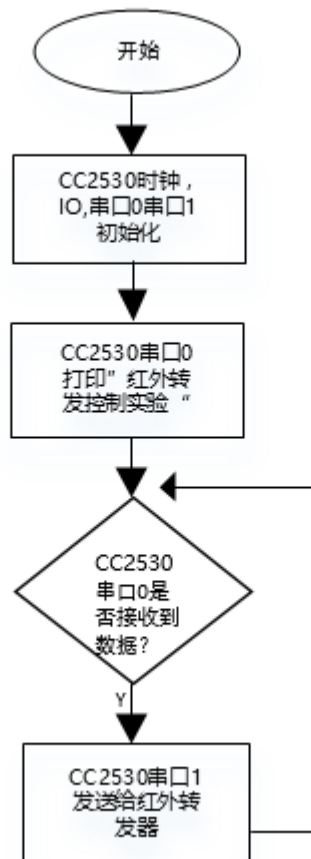


图 0.1 编程流程图

源码解析

本节实验的源码文件在【配套光盘\05-实验例程\04-第四章 智能家居实验\4.8 红外转发器应用实验】目录下，工程里主要包括“sys_init.c”，“uart.c”和主程序文件“main.c”。下面对这些文件中的主要函数进行具体解析。

1. 系统时钟初始化：

```
void xtal_init(void)
{
    SLEEPCMD &= ~0x04;           //都上电

    while(!(CLKCONSTA & 0x40));   //晶体振荡器开启且稳定

    CLKCONCMD &= ~0x47;           //选择 32MHz 晶体振荡器

    SLEEPCMD |= 0x04;
}
```

2. LED 灯初始化：

```
void led_init(void)
{
    P0SEL &= ~0x02;               //P0.1 为普通 I/O 口

    P0DIR |= 0x02;                //输出

    P0_1 = 1;                     //LED 灯默认打开
}
```

3. 串口 0（485 通信）初始化：

```
void uart0_init(unsigned char StopBits,unsigned char Parity)
{
    P0SEL |= 0x0C;                //初始化 UART0 端口 P0_2, P0_3

    PERCFG&= ~0x01;              //选择 UART0 为可选位置一

    U0CSR = 0xC0;                 //设置为 UART 模式,而且使能接受器

    U0GCR = 0x09;

    U0BAUD = 0x3b;                //设置 UART0 波特率为 19200bps

    U0UCR |= StopBits|Parity;     //设置停止位与奇偶校验

    P0SEL &= ~0x40;               //P0_6 为普通 I/O 口(485 控制端)

    P0DIR |= 0x40;                //P0_6 为输出端口(485 控制端)

    P0_6 = 1;
}
```

4. 串口 1 初始化

```
void uart1_init(void)
{
```

```
P1SEL |= 0xC0;           //初始化 UART1 端口
PERCFG|= 0x02;           //选择 UART1 为可选位置二
U1CSR = 0xC0;            //设置为 UART 模式,而且使能接受器

U1GCR = 0x08;
U1BAUD = 0x3b;           //设置 UART0 波特率为 9600bps
U1UCR |= 0x00;           //设置停止位与奇偶校验
}
```

5. 串口发送字节函数:

```
void Uart_Send_char(char ch)
{
    U0DBUF = ch;
    while(UTX0IF == 0);
    UTX0IF = 0;
}
```

6. 串口接收字节函数:

```
int Uart_Recv_char(void)
{
    int ch;
    while (URX0IF == 0);
    ch = U0DBUF;
    URX0IF = 0;
    return ch;
}
```


7. 主函数:

```
void main(void)
{
    xtal_init();
    led_init();           //LED 灯控制引脚初始化
    uart0_init();         //初始化串口 0,波特率 19200, 与 PC 机通信
    uart1_init();         //初始化串口 1,波特率 9600, 与红外转发传感器通信
    printf("*****红外转发控制实验*****\r\n");
    P0_6 = 0;             //串口 0 485 设置为接收模式
    while(1)
    {
        if(Receive_state == 1)//串口 0 接收到数据
        {
```



```
Uart_Send_char(Receive_Value);//将接收到的数据通过串口 1 发送给红外转发  
Receive_state = 0;  
}  
}  
}
```

14.4 实验步骤

6. USB A 转 B 连接线方口端接 ZB-LINK 调试器的“USB-Debug”口,USB 口接电脑,USB3.0 下载线一端接 ZBLINK 调试器的“Debug”另一端接“Zigbee”模块 USB3.0 的下载接口。
7. 用 IAR 软件打开路径为:【配套光盘\04-实验例程\第 4 章 智能家居实训区\4.7 红外转发控制器实验】中的实验工程“Infr_tran.eww”,选择 Project→Rebuild All 重新编译工程;
8. 选择 Project→Download and debug 将程序下载到 ZigBee 模块(如果下载出现问题,请按 ZB-LINK “RST”复位键复位,再次下载),下载完成后点退出;
9. 将下载好程序的节点安装在智能家居区域面板上的节点 2 上,用测试线将红外转发器下的“RX”端接在节点 5 的“P1”端口上,“TX”端接节点 5 的“P2”上,将节点 5 的“P4”端接在 LED 灯的“IO-IN”端,可以看到 LED 灯打开。
10. 将“RJ11 电话线”的一端连接“Zigbee 模块”的“RJ11-485”通信接口,另一端连接“ZB-Link 调试器”的“RS-485”接口,用“USB-A 转 B 一端接”“ZB-Link 调试器”的“USB-485”口,另一端接 PC 的 USB 口。
3. 【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】打开串口助手“Comdebug.exe”,在“设备管理器”-“端口”处查看串口号,在“串口调试工具”里选择对应的 COM 口,波特率设置为“19200”,点击“打开串口”即可。
11. 在串口助手发送区域里发送“f0”,红外转发器指示灯会常亮,此时红外转发器进入学习模式。
12. 在串口助手发送区域里发送“45”,红外转发器指示灯会灭一秒再常亮。
13. 此时拿着 LED 灯遥控器对着红外转发器内部黑色探头,按下 LED 遥控器的“OFF”键,此时红外转发器的指示灯状态为灭一秒,再亮,此时说明红外转发器已经学习 LED 灯的“OFF”键成功。
14. 在串口助手发送区域里发送“f2”,红外转发器指示灯会长灭,此时代表红外转发器退出学习模式进入遥控模式。

14.5 实验结果

在串口助手点击“hex 发送”,发送“45”,此时的现象和结果为红外转发器指示灯先亮一秒,然后长灭,此时打开状态的 LED 灯也熄灭。



实验十五：家居区-智能窗帘控制实验

15.1 实验目的

1. 了解 CC2530 I/O 口的特点与特性；
2. 学习并掌握 CC2530 I/O 口的基本配置；
3. CC2530 Uart 串口通信配置及程序编写；
4. 窗帘执行器的工作原理及使用方式；

15.2 实验环境

1. 硬件：1 个 Zigbee 模块，1 个窗帘（智能家居区域面板）、1 个 ZB-Link 调试器、1 根 USB2.0 方口线、1 根 USB3.0 数据线、1 根 RJ11 电话线、3 根测试连接线，1 台 PC 机；
2. 软件：Windows 7/XP、IAR 集成开发环境、串口调试工具。

15.3 实验原理



图 4.11.3 Zigbee 模块

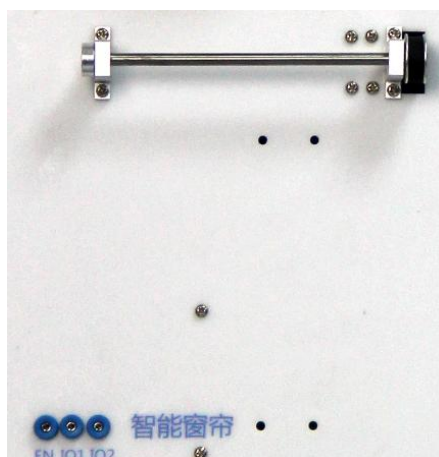


图 4.11.4 智能窗帘

硬件原理：

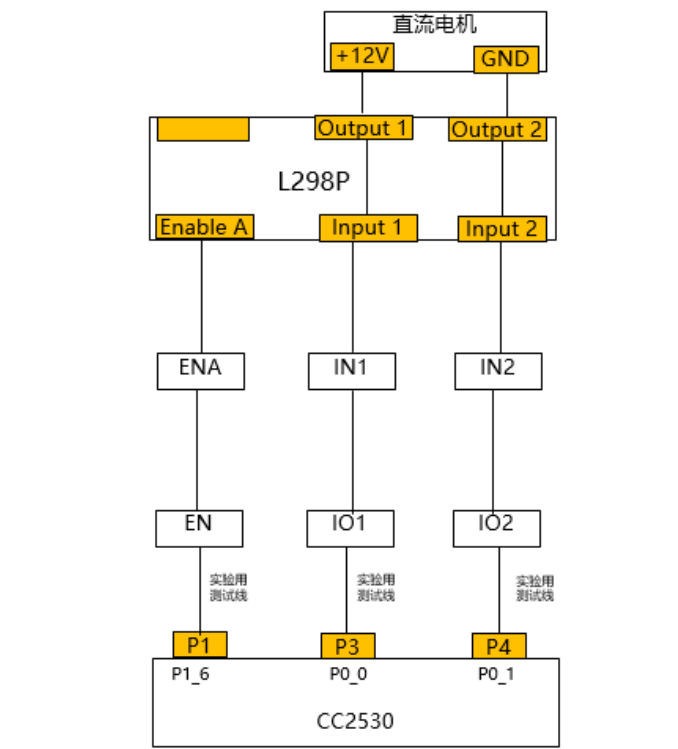


图 4.11.5 智能窗帘硬件接线图

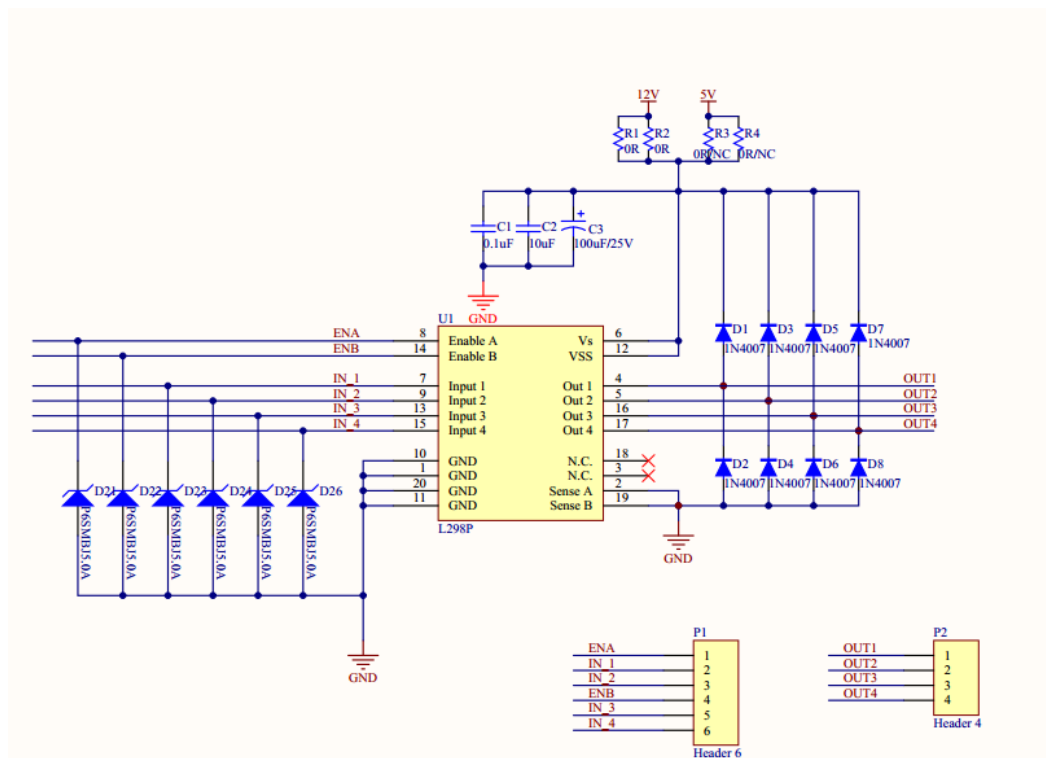


图 4.11.6 智能窗帘直流电机硬件原理图

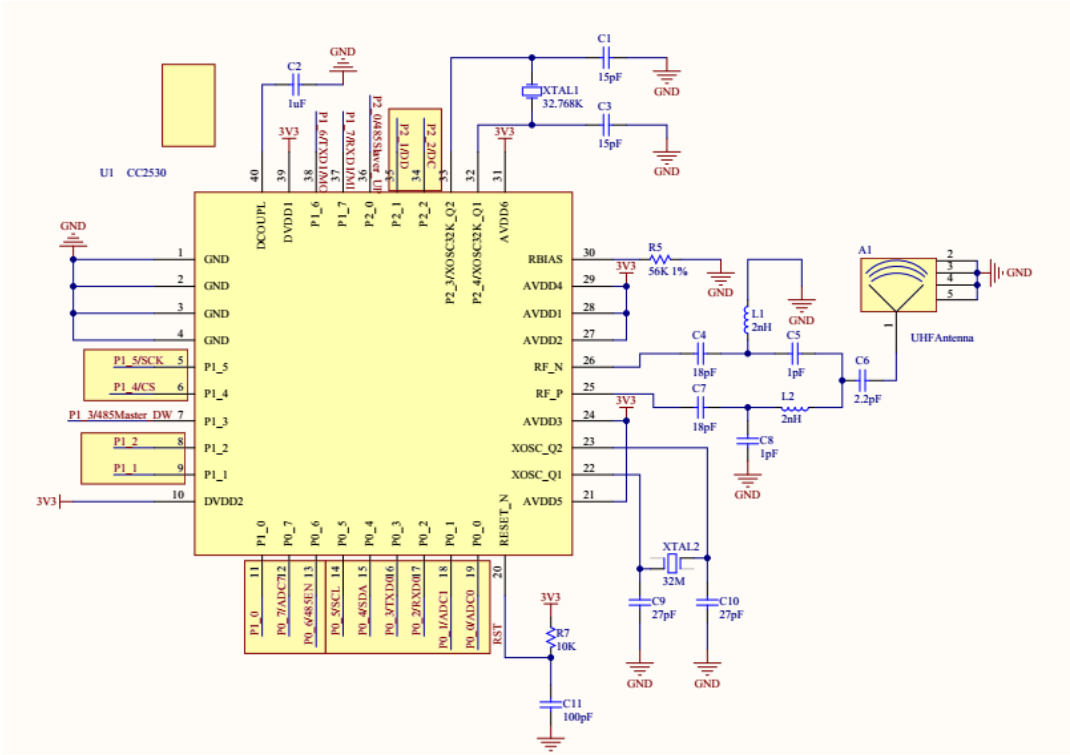


图 4.11.7 Zigbee 通用节点原理图

窗帘模块用来模拟智能家居中的智能窗帘，上图中上方的部件为直流电机驱动轴，直流电机用来驱动窗帘的正转，反转或关闭，它由三个开关量形式的输入信号 EN、IO1、IO2 来控制。下表列出了三个输入信号与窗帘状态之间的关系。

表 0.1 窗帘控制说明

EN	IO1	IO2	电机驱动轴 状态	窗帘状态
0	--	--	停止	停止
1	1	0	正转	打开
1	0	1	反转	关闭

“L298P”为直流电机驱动芯片，据驱动芯片原理可知，当“ENA”为低电平时，直流电机无输出，即窗帘停止；当“ENA”为高电平时，“IN1”为高电平，“IN2”为低电平，窗帘正转；“IN1”为低电平，“IN2”为“高电平”时，窗帘反转。

据“智能窗帘硬件接线图”可知当 CC2530 的“P1_6”为低电平时，直流电机无输出，窗帘停止；CC2530 的“P0_0”为高电平，“P0_1”为低电平时，直流电机正转，即窗帘正转；CC2530 的“P0_0”为低电平，“P0_1”为高电平时，直流电机反转，即窗帘反转。

编程思路：

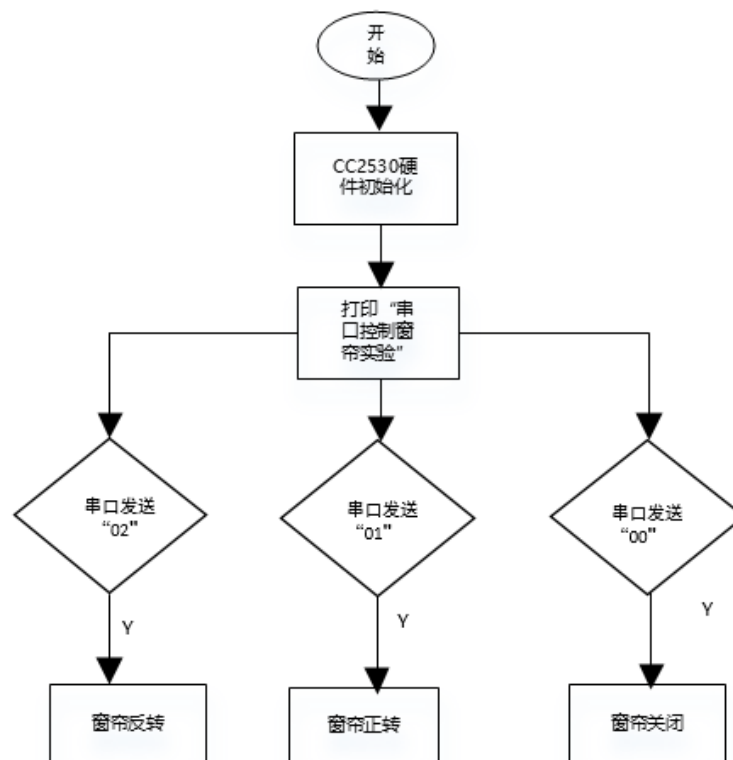


图 4.11.8 编程流程图

源码解析：

本节实验的源码文件在【配套光盘\05-实验例程\04-第四章 智能家居实验\4.11 门禁制实验】目录下，工程里主要包括“sys_init.c”，“uart.c”和主程序文件“main.c”。下面对这些文件中的主要函数进行具体解析。

1. 系统时钟初始化：

```
void xtal_init(void)
{
    SLEEP_CMD &= ~0x04;           //都上电

    while(!(CLKCONSTA & 0x40));    //晶体振荡器开启且稳定

    CLKCONCMD &= ~0x47;           //选择 32MHz 晶体振荡器

    SLEEP_CMD |= 0x04;
}
```

2. 窗帘初始化（CC2530 端口初始化）：

```
void Motor_init(void)
{
    P0SEL &= ~0x03;               //配置 P0_0, P0_1 为普通 IO

    P0DIR |= 0x03;                //配置 P0_0, P0_1 为输出 IO
}
```

```
P1SEL &=~0x40;          //配置 P1_6 为普通 IO
P1DIR |= 0x40;           //配置 P1_6 为 输出
P1_6 = 0;                //窗帘默认关闭
}
```

3. 串口 0（485 通信）初始化：

```
void uart0_init(unsigned char StopBits,unsigned char Parity)
{
    P0SEL |= 0x0C;          //初始化 UART0 端口 P0_2, P0_3
    PERCFG&= ~0x01;         //选择 UART0 为可选位置一
    U0CSR = 0xC0;           //设置为 UART 模式,而且使能接受器
    U0GCR = 0x09;
    U0BAUD = 0x3b;          //设置 UART0 波特率为 19200bps
    U0UCR |= StopBits|Parity; //设置停止位与奇偶校验
    P0SEL &= ~0x40;         //P0_6 为普通 I/O 口(485 控制端)
    P0DIR |= 0x40;          //P0_6 为输出端口(485 控制端)
    P0_6 = 1;
}
```

4. 串口发送字节函数：

```
void Uart_Send_char(char ch)
{
    U0DBUF = ch;
    while(UTX0IF == 0);
    UTX0IF = 0;
}
```

5. 串口接收字节函数：

```
int Uart_Recv_char(void)
{
    int ch;
    while (URX0IF == 0);
    ch = U0DBUF;
    URX0IF = 0;
    return ch;
}
```

6. 主函数：

```
void main(void)
{
    Motor_init();           //窗帘硬件初始化
}
```

```

xtal_init();                //系统时钟初始化

uart0_init(0x00, 0x00);     //初始化串口：无奇偶校验，停止位为 1 位


Uart_Send_String("串口控制窗帘实验\r\n");

P0_6 = 0;                   //设置为接收模式(485 控制端)

while(1)
{
    ch = Uart_Recv_char();   //串口接收字节函数
    if (ch == 1)
    {
        P1_6 = 1;           //串口接收到“01”窗帘正转
        P0_1 = 0;
        P0_0 = 1;
    }
    if(ch == 2)
    {
        //串口接收到“02”窗帘反转
        P1_6 = 1;
        P0_1 = 1;
        P0_0 = 0;
    }
    if(ch == 0)
    {
        //串口接收到“00”窗帘停止
        P1_6 = 0;
    }
    P0_6 = 1;               //设置为发送模式(485 控制端)
    Uart_Send_char(ch);
    P0_6 = 0;               //设置为接收模式(485 控制端)再次接收串口数据
}
}

```

15.4 实验步骤

1. USB A 转 B 连接线方口端接 ZB-LINK 调试器的“USB-Debug”口，USB 口接电脑，USB3.0 下载线一端接 ZBLINK 调试器的“Debug”另一端接“Zigbee”模块 USB3.0 的下载接口。
2. 用 IAR 软件打开路径为：【配套光盘\04-实验例程\第 4 章 智能家居实训区\4.10 窗帘控制实验】中的实验工程“Motor.eww”，选择 Project→Rebuild All 重新编译工程；
3. 选择 Project→Download and debug 将程序下载到 ZigBee 模块，下载完成后点  退出；

4. 将下载好程序的节点安装在智能家居区域面板上的节点 6 上,用测试线将智能窗帘的“EN”端接在节点 6 的“P4”端口;“IO1”接在节点 6 的“P3”端口;“IO2”接在节点 6 的“P1”端口。
5. 将“RJ11 电话线”的一端连接“Zigbee 模块”的“RJ11-485”通信接口,另一端连接“ZB-Link 调试器”的“RS-485”接口,用“USB-A 转 B 一端接”“ZB-Link 调试器”的“USB-485”,另一端接 PC 的 USB 口。
6. 在【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】路径下打开“Comdebug.exe”串口助手,在“设备管理器”-“端口”处查看串口号,在“端口号”里选择对应的 COM 口,波特率设置为“19200”,点击“打开串口”即可。如下图所示:



15.5 实验结果

通过串口调试工具发送“01”,观察窗帘会“正转”,发送“02”,观察窗帘会“反转”,发送“00”窗帘会“停止”。

实验十六：家居区-智能门禁控制实验

16.1 实验目的

1. 了解 CC2530 I/O 口的特点与特性；
2. 学习并掌握 CC2530 I/O 口的基本配置；
3. CC2530 Uart 串口通信配置及程序编写；
4. 门磁执行器的工作原理及使用方式；

16.2 实验环境

1. 硬件：1 个 Zigbee 模块，1 个门禁（智能家居区域面板）、1 个 ZB-Link 调试器、1 根 USB2.0 方口线、1 根 USB3.0 数据线、1 根 RJ11 电话线、1 根测试连接线，1 台 PC 机；
2. 软件：Windows 7/XP、IAR 集成开发环境、串口调试工具。

16.3 实验原理



图 4.11.3 Zigbee 模块



图 4.11.4 门禁

硬件原理：

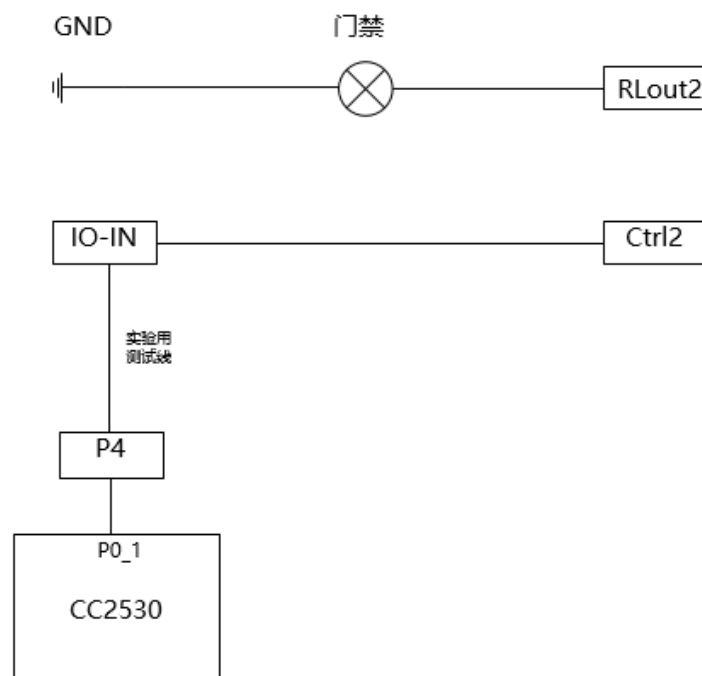


图 4.11.5 智能家居区域门禁硬件接线图

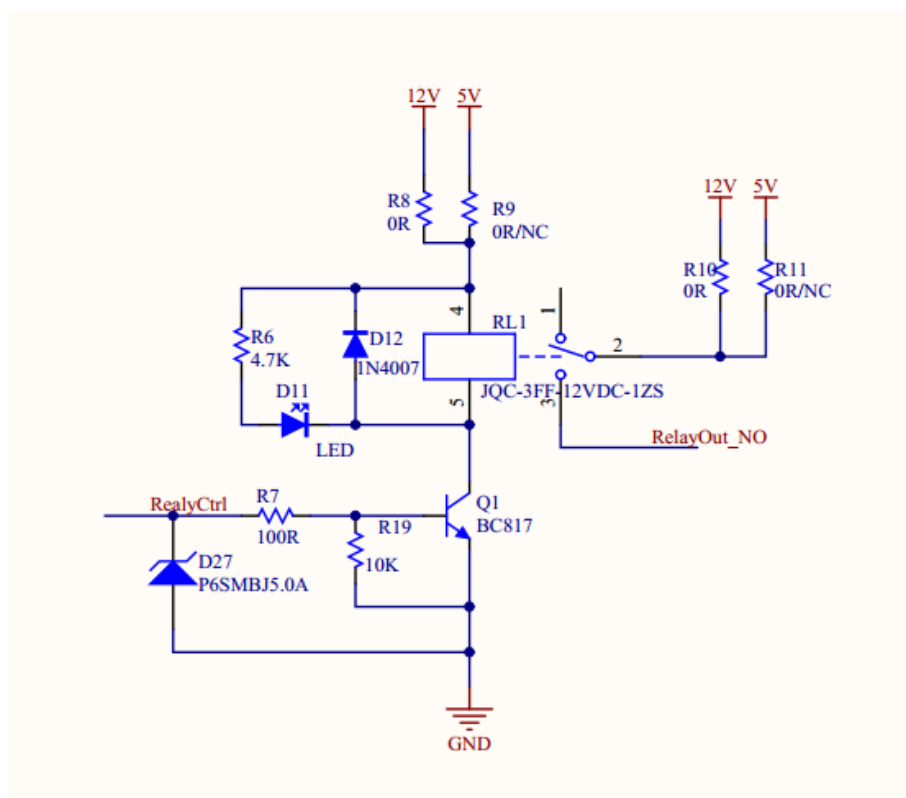


图 4.11.6 智能家居接线板连接图

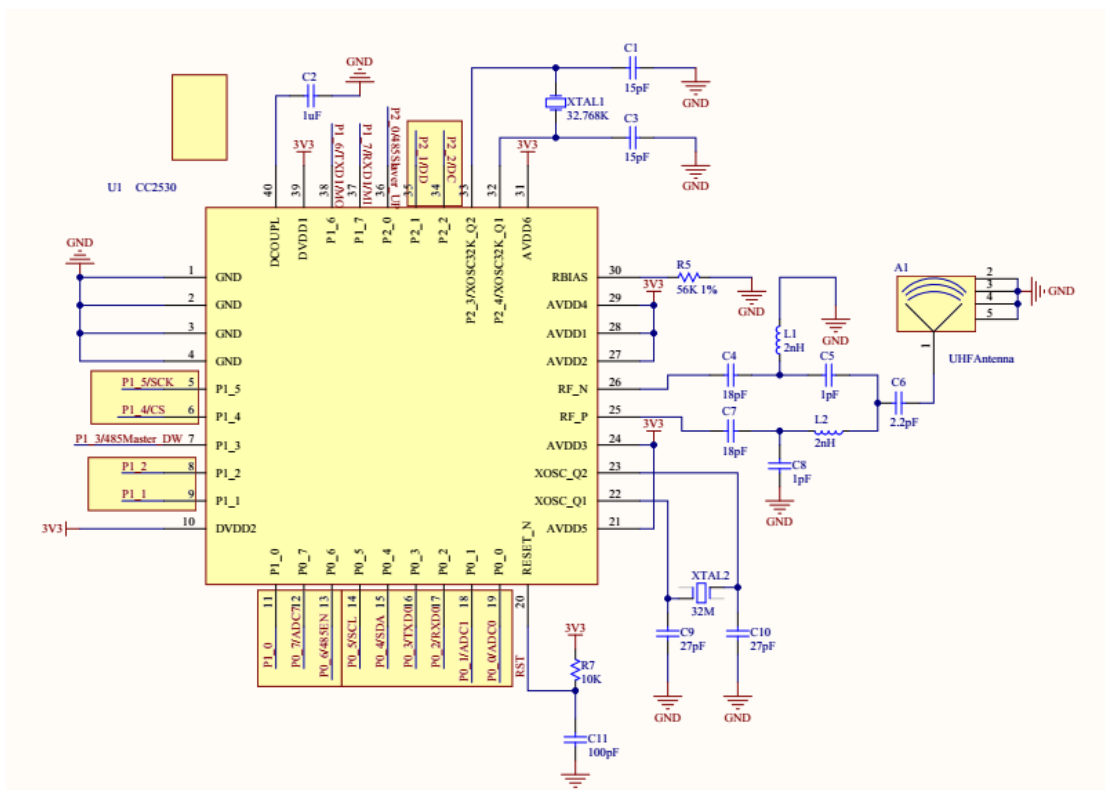


图 4.11.7 Zigbee 通用节点原理图

智能家居区域的门禁为直流 12V 供电门禁，也就是一端接 12V 电源，另一端接 GND，门禁可打开，也就是“RLout2”有 12V 电压即打开门禁，“RLout2”无电压即关闭门禁。

根据智能家居接线板连接图得知当“RealyCtrl”为高电平时，继电器“RL1”线圈内有电流，此时会将“1,2”常闭开关，转换到“1,3”常开，1,3 接通后“RLout2”即有 12V 电压输出，门禁，由“智能家居区域 LED 硬件接线图”可得知只需将“CC2530”的 PO_1 引脚置高电平即可打开门禁，置低电平即可关闭门禁。

编程思路：

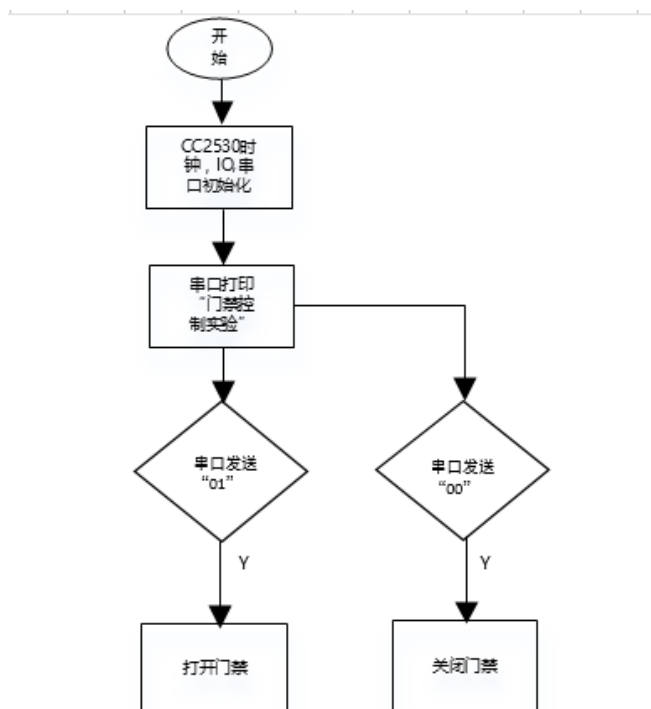


图 4.11.8 编程流程

源码解析：

本节实验的源码文件在【配套光盘\05-实验例程\04-第四章 智能家居实验\4.11 门禁控制实验】目录下，工程里主要包括“sys_init.c”，“uart.c”和主程序文件“main.c”。下面对这些文件中的主要函数进行具体解析。

1. 系统时钟初始化：

```
void xtal_init(void)
{
    SLEEP_CMD &= ~0x04;           // 都上电

    while(!(CLKCONSTA & 0x40));    // 晶体振荡器开启且稳定

    CLKCONCMD &= ~0x47;           // 选择 32MHz 晶体振荡器

    SLEEP_CMD |= 0x04;
}
```

2. 门禁初始化（配置 CC2530 输出管脚）

```
void Elect_Lock_init(void)
{
    P0SEL &= ~0x02;               // 配置 P0_1 为普通 IO

    P0DIR |= 0x02;                // 配置 P0_1 为 输出

    P0_1 = 0;                     // 门禁默认关闭
}
```

```
}

```

3. 串口0（485 通信）初始化:

```
void uart0_init(unsigned char StopBits,unsigned char Parity)
{
    P0SEL |= 0x0C;           //初始化 UART0 端口 P0_2, P0_3
    PERCFG&= ~0x01;         //选择 UART0 为可选位置一
    U0CSR = 0xC0;            //设置为 UART 模式,而且使能接受器
    U0GCR = 0x09;
    U0BAUD = 0x3b;           //设置 UART0 波特率为 19200bps
    U0UCR |= StopBits|Parity; //设置停止位与奇偶校验
    P0SEL &= ~0x40;          //P0_6 为普通 I/O 口(485 控制端)
    P0DIR |= 0x40;           //P0_6 为输出端口(485 控制端)
    P0_6 = 1;
}
```

4. 串口发送字节函数:

```
void Uart_Send_char(char ch)
{
    U0DBUF = ch;
    while(UTX0IF == 0);
    UTX0IF = 0;
}
```

5. 串口接收字节函数:

```
int Uart_Recv_char(void)
{
    int ch;
    while (URX0IF == 0);
    ch = U0DBUF;
    URX0IF = 0;
    return ch;
}
```

6. 主函数:

```
void main(void)
{
    Elect_Lock_init();           //CC2530 IO 硬件初始化
    xtal_init();                 //系统时钟初始化
    uart0_init(0x00, 0x00);      //初始化串口: 无奇偶校验, 停止位为 1 位
    Uart_Send_String("串口控制门禁实验\r\n");//串口打印 “串口控制门禁实验”
}
```

```

P0_6 = 0;          //485 控制端设置为低电平，485 串口设置为接收模式
while(1)
{
    ch = Uart_Recv_char(); //串口接收字节函数
    if (ch == 1)
    {
        P0_1= 1;          //串口接收到“01”打开门禁
    }
    if(ch == 0)
    {
        //串口接收到“00”关闭门禁
        P0_1 = 0;
    }
    P0_6 = 1;          //485 控制端设置为高电平，485 串口设置为发送模式
    Uart_Send_char(ch);
    P0_6 = 0;          //485 控制端设置为低电平，串口设置为接收模式，再次接收串口数据
}
}

```

16.4 实验步骤


1. USB A 转 B 连接线方口端接 ZB-LINK 调试器的“USB-Debug”口，USB 口接电脑，USB3.0 下载线一端接 ZBLINK 调试器的“Debug”另一端接“Zigbee”模块 USB3.0 的下载接口。
2. 用 IAR 软件打开路径为：【配套光盘\04-实验例程\第 4 章 智能家居实训区\4.1 门禁控制实验】中的实验工程“Elect_Lock.eww”，选择 Project→Rebuild All 重新编译工程；
3. 选择 Project→Download and debug 将程序下载到 ZigBee 模块，下载完成后点退出；
4. 将下载好程序的节点安装在智能家居区域面板上的节点 4 上，用测试线将门禁右侧“IO-IN”端接在节点 4 的“P4”端口上。
5. 将“RJ11 电话线”的一端连接“Zigbee 模块”的“RJ11-485”通信接口，另一端连接“ZB-Link 调试器”的“RS-485”接口，用“USB-A 转 B 一端接”“ZB-Link 调试器”的“USB-485”，另一端接 PC 的 USB 口。
6. 在【配套光盘\03-常用工具\01-硬件开发包\03-串口调试工具】路径下打开“Comdebug.exe”串口助手，在“设备管理器”-“端口”处查看串口号，在“端口号”里选择对应的 COM 口，波特率设置为“19200”，点击“打开串口”即可。如下图所示：



图 4. 11. 9 串口调试助手

16.5 实验结果

通过串口调试工具发送“01”，观察门禁会打开，发送“00”，观察门禁会关闭。

