中南大学
无线传感器网络
实验报告

专　　业：物联网工程

指导老师：刘伟荣

班　　级：1802

学　　号：8213180228

姓　　名：王云鹏

2021年06月

目录

实验 1：两个灯交替闪烁

一、实验目的

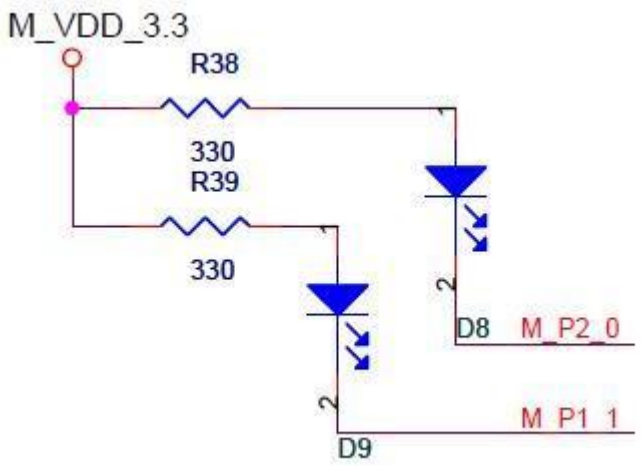
1.掌握 CC2530 单片机 C 语言编程方法；
2.掌握 P1 和 P2 口作为通用输出口的使用方法。

二、实验环境

1.装有 IAR 开发环境的 PC 机一台；
2.实验箱一台；
3.CCDebugger（以及 USB A-B 延长线）一个；
4.基础实验板一个；

三、实验原理

CC2530 主板有两枚 LED 灯。由《CC25xx 主板电路原理图》中的 CC2530 主板 LED 管脚图（图一）可以看出，这两枚 LED 灯（D8、D9）分别对应管脚 P2_0 和 P1_1。所以只要按照一定时序控制 P2_0 和 P1_1 的输出电压，就能够控制 LED 的闪烁。



四、实验要求

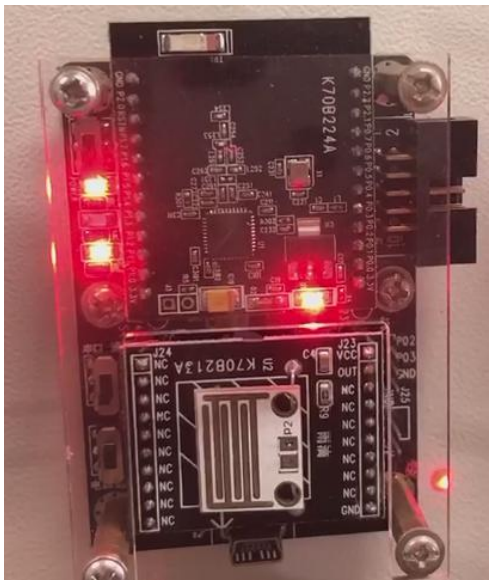
两个灯交替闪烁，并用定时器实现

五、实验现象

实验现象已上传到网络：https://www.bilibili.com/video/BV1Dq4y1L7HU/

如代码段的注释所述，代码段 1 在运行时，会在经过一定时间后，改变 P1_1 和 P2_0 的电平，进而控制 LED 等的闪烁。而代码段 2 实现的是在计时器计数溢出后，进入中断处理函数，改变 P1_1 和 P2_0 的电平，进而实现闪烁。

上面的 LED 灯亮起



下面的 LED 灯亮起

六、代码分析

代码段 1 两个灯交替闪烁（循环实现）

```
#include "ioCC2530.h"

void main(){
P1SEL=0;//设置 P1 为通用 I/O 口
P1DIR=2;//设置 P1_1 I/O 口为输出模式(00000010)
P2SEL=0;//设置 P2 为通用 I/O 口
P2DIR=1;//设置 P2_0 I/O 口为输出模式(00000001)
while(1){
P1_1=0;//P1_1 低电平
P2_0=1;//P2_0 高电平
for(int     i=1;i<100;i++){
for(int b=1;b<1000;b++){} //延时
}
P1_1=1;//P1_1 高电平
P2_0=0;//P2_0 低电平
for(int     i=1;i<100;i++){
for(int     b=1;b<1000;b++){} //延时
}
}
}
```

代码段 2 两个灯交替闪烁（定时器实现）

```
#include "ioCC2530.h"
#pragma vector = T1_VECTOR
```

```
  __interrupt     void T1_Handle(void)   //定时中断处理函数
{
  P1_1=~P1_1;//取 P1_1 电位反，实现闪烁
  P2_0=~P2_0;//取 P2_0 电位反，实现闪烁
     T1STAT&=~0x02;//(00000010)清除标志位
     IRCON&=~0x02;
}
void main(){
T1CTL =     0x0d;//f=32MHz/128=250000Hz,T=4us;模模式,从 0~T1CC0 反复计数
T1CCTL0 =     0x44;//Timer1 通道 0 中断允许,比较匹配模式,比较比配时输出置位
T1CC0L = 0xc4     ;//先写 T1CC0L,再写 T1CC0H,T1CNT=0 时更新
T1CC0H = 0x55     ;//2500*4us=10000us=10ms
TIMIF |=     0x40;//TIMIF.T1OVFIM Timer1 溢出中断允许
IEN1 |=     0x02;//IEN1.T1IETimer1 总中断允许
EA = 1; //打开全局总中断
P1SEL=0;//设置 P1 为通用 I/O 口
P1DIR=2;//设置 P1_1 I/O 口为输出模式(00000010)
P2SEL=0;//设置 P2 为通用 I/O 口
P2DIR=1;//设置 P2_0 I/O 口为输出模式(00000001)
P1_1=1;//P1_1 高电平
P2_0=0;//P2_0 低电平
while(1){} //死循环、处理事件
}
```

实验 2：流水灯动态效果

一、实验目的

1. 掌握 CC2530 单片机 C 语言编程方法
2. 掌握 P1 口作为通用输出口的使用方法

二、实验环境

1. 装有 IAR 开发环境的 PC 机一台
2. 实验箱一台
3. CCDebugger（以及 USB A-B 延长线）一个
4. 基础实验板一个

三、实验原理

程序流程图如图 1.40 所示。

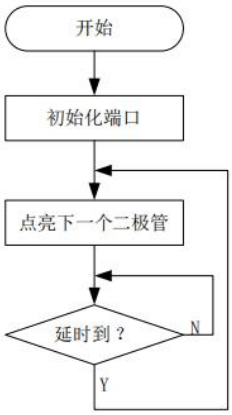


图 1.40 LED 控制流程图

四、实验要求

利用基础实验板上的 8 个 LED 灯实现：流水灯从两头到中间再中间到两头

五、实验现象

实验现象已上传到网络：https://www.bilibili.com/video/BV1z54y1H7QY/

这里修改了 LEDx_ON 对应的针脚 LED，使得同时亮关于中间对称的两个，然后再 main 函数的循环中从 LED1_ON 到 LED8_ON 挨个点亮

六、代码分析

```
//main.c
#include "led.h"
#include "Basic.h"
uint16 T_delay = 10;
```

```
void main(void){
   INIT_LED();
   for( ;; )//循环闪烁
   {   //LEDx_ON 对应的针脚定义在 led.h 中
       LEDprintf(LED1_ON,BYTE_5);//使 LED1_ON 对应的针脚 LED 亮
       Delay(T_delay);                                    //亮一会
       LEDprintf(LED2_ON,BYTE_5);
       Delay(T_delay);
       LEDprintf(LED3_ON,BYTE_5);
       Delay(T_delay);
       LEDprintf(LED4_ON,BYTE_5);
       Delay(T_delay);
       LEDprintf(LED5_ON,BYTE_5);
       Delay(T_delay);
       LEDprintf(LED6_ON,BYTE_5);
       Delay(T_delay);
       LEDprintf(LED7_ON,BYTE_5);
       Delay(T_delay);
       LEDprintf(LED8_ON,BYTE_5);
       Delay(T_delay);
   }
}
```

```
//led.h
#ifndef _LED_
#define _LED_
#include <ioCC2530.h>
#define HIGH 1
#define LOW 0
#define SER P1_0
#define RCK P1_1
#define SRCK P1_2
#define BYTE_5 0x80
#define LED1_ON 0x81//修改 LED1_ON 对应的针脚，使 LED1 和 LED8 都亮
#define LED2_ON 0x42//使 LED2 和 LED7 都亮
#define LED3_ON 0x24//使 LED3 和 LED6 都亮
#define LED4_ON 0x18//使 LED4 和 LED5 都亮
#define LED5_ON 0x18//使 LED4 和 LED5 都亮
#define LED6_ON 0x24//使 LED3 和 LED6 都亮
#define LED7_ON 0x42//使 LED2 和 LED7 都亮
#define LED8_ON 0x81//使 LED1 和 LED8 都亮
void INIT_LED(void);
void LEDprintf(unsigned char data,unsigned char byte);
#endif
```

实验 3：串口通信

一、实验目的

1. 理解串口通信原理
2. 掌握 CC2530 单片机与 PC 机串口通信的方法

二、实验环境

1. 装有 IAR 开发环境的 PC 机一台
2. 实验箱一台
3. CCDebugger（以及 USB A-B 延长线）一个
4. USB Mini 延长线一根

三、实验原理

本实验中 CC2530 节点通过串口调试助手向微机发送字符串,进行字符加 1 后，PC 机接收到串口数据后通过串口调试助手将接收到的内容显示出来。

CC2530 单片机使用的电平为 TTL 电平，而 PC 机使用的是 CMOS 电平，所以在与 PC 机进行通信时，需要电平转换电路来匹配逻辑电平。本实验选用串口转 USB 接口电路来匹配逻辑电平，同时使得单片机与 PC 机之间的硬件连接更加方便。硬件连接如图 1.60 所示。

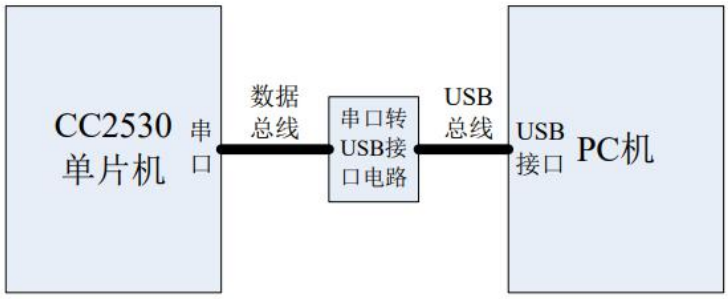


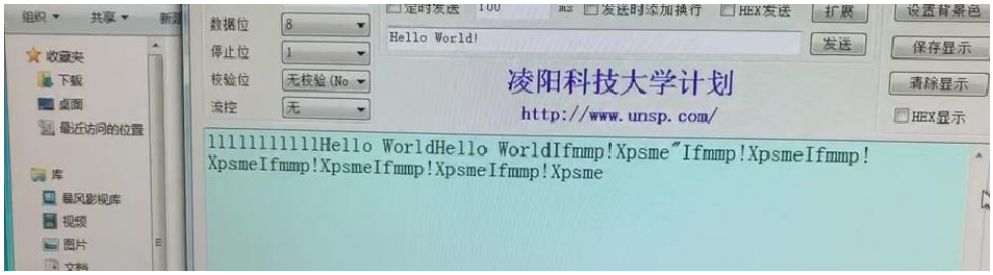
图 1.60 CC2530 单片机与 PC 机串口通信原理图

四、实验要求

输入框中输入字符串，将所有字符加 1 后 PC 机的串口调试助手显示，例如输入“Hello World！”，串口调试助手显示“IfmmpXpsme”。

五、实验现象

实验现象已上传到网络：https://www.bilibili.com/video/BV1PB4y1u76z/



六、代码分析

主要实现代码

至此就实现了指导书上的实验四_单片机与 PC 机串口通信实验PC 机的串口调试助手显示“Hello World！”。

在此基础上修改 mian.c 中屏蔽原 UART0_Send 方法；UART.c 中的 UART0_ISR 方法达到中断中实现字符加 1 及发送数据

mian.c 中：

```
//UART0_Send( SENDSTRING, sizeof(SENDSTRING)-1);
```

UART.c 中：

```
/*************************************************************
**  函数名称: UART0_ISR
**  实现功能: UART0 接收中断处理函数
**  入口参数: None
**  返回结果: None
*************************************************************/
#pragma vector = URX0_VECTOR
 __interrupt void UART0_ISR(void)
 {
static char temp[1];
temp[0] = U0DBUF;
temp[0]++;//将字符+1
UART0_Send(temp, 1);//UART0 发送数据
URX0IF = 0;//清中断标志
 }
```

实验 4：休眠唤醒实验

一、实验目的

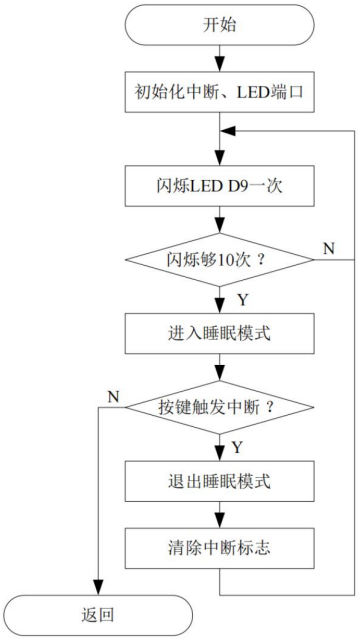
1. 理解 CC2530 睡眠工作模式
2. 掌握睡眠唤醒方法

二、实验环境

1. 装有 IAR 开发工具的 PC 机一台
2. 实验箱一台
3. CCDebugger（以及 USB A-B 延长线）一个
4. USB Mini 延长线一根

三、实验原理

程序流程图如下图所示



四、实验要求

代码运行后，电源状态指示灯 D8 亮，LED 灯 D9 闪烁十次，然后系统进入睡眠状态，D8 灭；按下 key1 按键后，系统被唤醒，重复上述过程。

五、实验现象

实验现象已上传到网络：https://www.bilibili.com/video/BV1264y197Jd/

按照题目要求提供了两种中断唤醒的方式，即定时器中断和按键中断。两种中断方式互相独立互不干扰。为了表示区分，定时器中断让灯闪烁三次，按键中断让灯闪烁五次。

可以看到，每隔固定的时间（时间的长短可以由 Delay 设置），小灯闪烁三次。当手动按动实验板按键，小灯会立刻闪烁五次。

六、代码分析

```
#include "ioCC2530.h"

#define LED1 P1_1
#define LED2 P2_0
#define LIGHT 0
#define DARK 1
#define uint unsigned int
#define uchar unsigned char
#define ulong unsigned long

void Init(){
    P1SEL = 0;
    P1DIR = 2;
    P2SEL = 0;
    P2DIR = 1;
    LED1 = LIGHT;
    LED2 = LIGHT;
    P0SEL &= ~0X30;
    P0DIR &= ~0X30;
    P0INP   &= ~0X30;//有上拉、下拉
    P2INP &= ~0X40; //选择上拉
    P0IEN |= 0X30;     //P12 P13
    PICTL |= 0X02;     //下降沿
    IEN1 |= 0X20;   //P1IE = 1;
    EA = 1;
    P0IFG |= 0x00;     //P12 P13
    //配置定时器工作在无阀值计数模式，单位时间为“128 系统时钟频率”
    ST2 = 0x00;
    ST1 = 0x0F;
    ST0 = 0x0F;
    EA = 1;
    STIE = 1;   //P1IE = 1;
    STIF = 0;

}
void set_control(uint sec){
    ulong sleepTimer = 0;
```

```
        sleepTimer |= ST0;
        sleepTimer|= (ulong)ST1 << 8;
        sleepTimer|= (ulong)ST2 << 16;
        sleepTimer +=((ulong)sec*(ulong)32768);
        ST2 = (uchar)(sleepTimer >> 16);
        ST1 = (uchar)(sleepTimer >> 8);
        ST0 = (uchar)sleepTimer;
}


/*************************************************************
*函数功能：初始化电源
*入口参数：para1,para2,para3,para4
*返回值    ：无
*说　明    ：para1,模式选择
*                                                           *
* para1    0  1    2    3                                   *
* mode       PM0     PM1     PM2     PM3
          *
*                                                           *
*************************************************************/
void PowerMode(uchar sel){
    if(sel<4){
        SLEEPCMD |= sel;

        PCON = 0x01;//睡眠
    }else{
         PCON = 0x00;//唤醒
    }
}

void Delay(int n){
    int i, j, k;
    for (i = 0; i < n; ++i){
        for (j = 0; j < 100; ++j){
            for (k = 0; k < 100; ++k){}
        }
    }
}

/****************************************
**定时器中断
****************************************/
```

```
#pragma vector = ST_VECTOR
      __interrupt void ST_ISR(void){
        STIF=0;
//闪烁 3 次
            PowerMode(4);
            for (int i = 0; i < 3; ++i){
                        LED1 = LIGHT; //点亮 LED1
                        Delay(12);
                        LED1 = DARK;
                        Delay(12);
            }
            Delay(30);
      }

/****************************************
**按键中断
****************************************/
#pragma vector = P0INT_VECTOR
  __interrupt void P0_ISR(void) {
      if(P0IFG>0){
            P0IFG = 0;
      }
      P0IF = 0;

      PowerMode(4);
      for (int i = 0; i < 5; ++i){
                  LED2 = LIGHT; //点亮 LED2
                  Delay(12);
                  LED2 = DARK;
                  Delay(12);
      }
  }

void main(void){
    Init();
            Delay(10);
            LED1 = DARK;
            LED2 = DARK;
    for(;;){
              Delay(20);
              set_control(3);
        PowerMode(2);
            }
}
```

实验 5：传感与执行实验

实验 5.1 雨滴传感器实验

一、实验目的

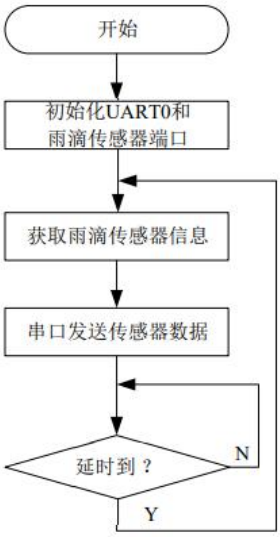
1. 理解雨滴传感器的工作原理
2. 掌握单片机驱动雨滴外传感器的方法

二、实验环境

1. 装有 IAR 开发工具的 PC 机一台
2. 实验箱一台
3. CCDebugger（以及 USB A-B 延长线）一个
4. USB Mini 延长线一根

三、实验原理

程序流程图如图



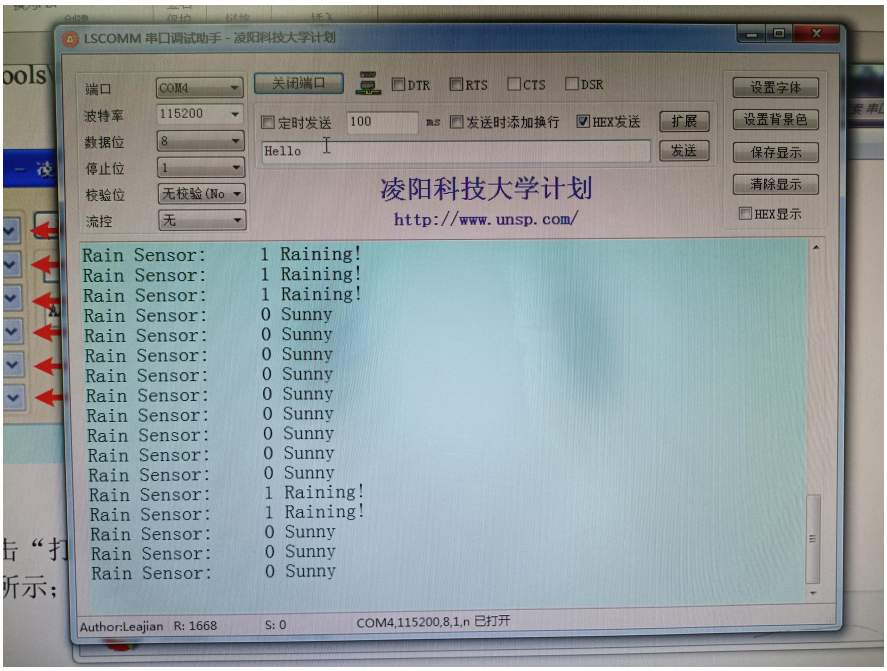
四、实验要求

将检测到的数据通过串口调试助手显示，观察有无水滴时的数据变化。

五、实验现象

实验现象已上传至网络：https://www.bilibili.com/video/BV1Fv411H7TU/

雨滴传感器可用于检测是否有降雨，可用手指轻轻按压雨滴传感器的表面来测试。当传感器表面比较干燥，检测结果显示“Sunny”，当传 感器表面比较潮湿，检测结果显示“Raining”。



六、代码分析

因这次实验并未修改代码，故不贴代码

实验 5.2 执行器实验

一、实验目的

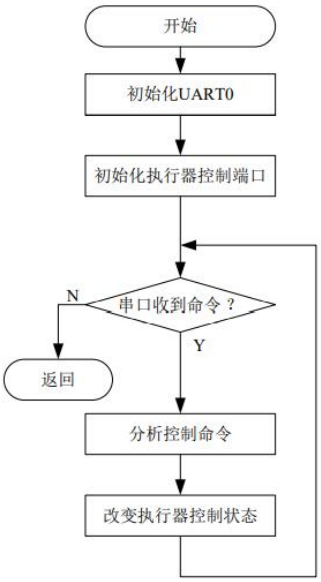
1. 理解执行节点的工作原理
2. 掌握单片机驱动执行节点的方法

二、实验环境

1. 装有 IAR 开发工具的 PC 机一台
2. 实验箱一台
3. CCDebugger（以及 USB A-B 延长线）一个
4. USB Mini 延长线一根

三、实验原理

执行器（控制器）节点程序流程图如图

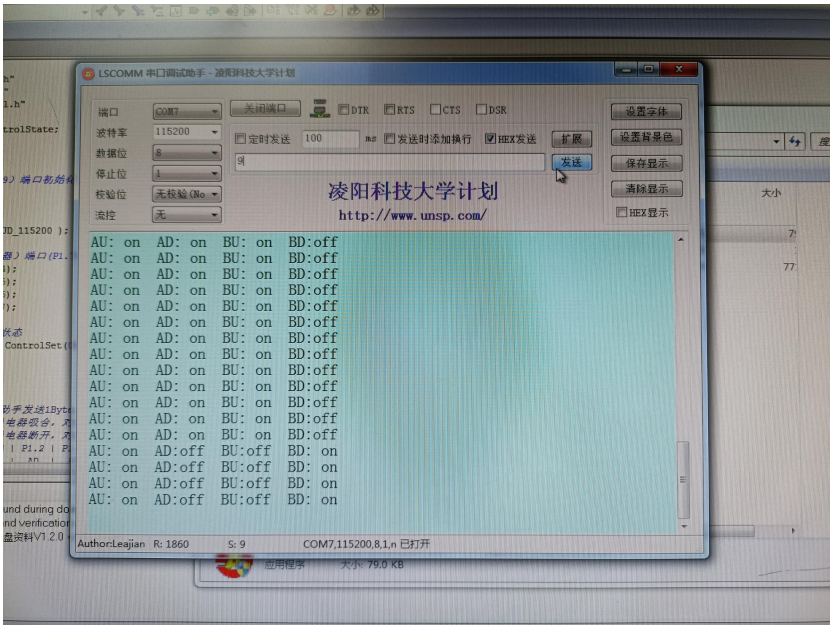


四、实验要求

CC2530 通过串口向串口调试助手不断输出当前的四个继电器的状态，并且通过串口调试助手发送十六进制的值时，可以控制四个继电器的开关状态。

五、实验现象

在“发送”按钮左侧的文本框内输入十六进制形式的字节数据，并点击“发送”按钮，可以控 制四个继电器的开关状态，四个继电器分别对应于这个字节中的bit0~bit3。



六、代码分析

因此次实验并未修改代码，故补贴代码

实验 6：Zstack 使用任务实现两个灯交替闪烁

一、实验目的

1.了解 ZigBee 2007 协议栈操作系统的工作机制
2.熟悉系统中任务的基本格式

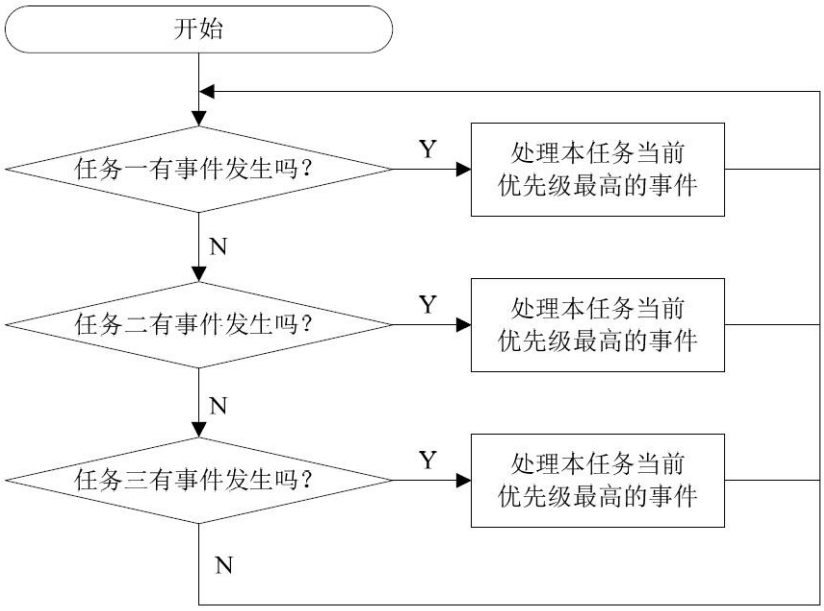
二、实验环境

1.装有 IAR 开发工具的 PC 机一台
2.实验箱一台
3.CCDebugger（以及 USB A-B 延长线）一个

三、实验原理

ZigBee 2007 协议枝操作系统简介

操作系统的使命就是对几项不同的任务进行调度，使其协调有序的在 CPU 上运行。ZigBee 协议栈中的操作系统采用轮转查询的方式来协调调度各项任务。程序流程图如图所示。



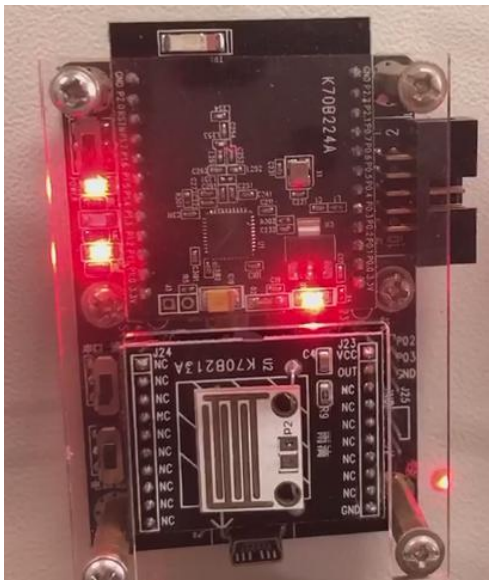
四、实验要求

要求能通过任务建立使得两个灯交替闪烁。

五、实验现象

实验现象已上传至网络：https://www.bilibili.com/video/BV1Dq4y1L7HU/

上面的 LED 灯亮起



下面的 LED 灯亮起

六、代码分析

```
#include "APP_Base.h"

#if defined(SAPP_ZSTACK_DEMO)
#include "hal_led.h"
// 任务建立实验范例代码
// 任务处理函数
uint16 Hello_ProcessEvent(uint8 task_id, uint16 events);
uint16 Hello_ProcessEvent(uint8 task_id, uint16 events)
{
    if(events & 0x0001)
    {
        // 控制 LED 闪烁
        HalLedBlink(HAL_LED_1, 1, 50, 250);
        // 启动定时器，设置 1 秒钟后再次触发该任务
        osal_start_timerEx(task_id+1, 0x0001, 1000);
    }
    // 清除定时器事件标志
    return (events ^ 0x0001);
}
uint16 Flash_ProcessEvent(uint8 task_id, uint16 events);
uint16 Flash_ProcessEvent(uint8 task_id, uint16 events)
{

    if(events & 0x0001)
    {
        // 控制 LED 闪烁
```

```
            HalLedBlink(HAL_LED_2, 1, 50, 250);
            // 启动定时器，设置 1 秒钟后再次触发该任务
            osal_start_timerEx(task_id-1, 0x0001, 1000);
        }
        // 清除定时器事件标志
        return (events ^ 0x0001);
    }
    #endif

    // 任务列表
    const pTaskEventHandlerFn tasksArr[] = {
        macEventLoop,
        nwk_event_loop,
        Hal_ProcessEvent,
    #if defined( MT_TASK )
        MT_ProcessEvent,
    #endif
        APS_event_loop,
    #if defined ( ZIGBEE_FRAGMENTATION )
        APSF_ProcessEvent,
    #endif
        ZDApp_event_loop,
    #if defined ( ZIGBEE_FREQ_AGILITY ) || defined ( ZIGBEE_PANID_CONFLICT )
        ZDNwkMgr_event_loop,
    #endif
    #if defined(SAPP_ZSTACK)
        sapp_controlEpProcess,
        sapp_functionEpProcess,
    #endif
    #if defined(SAPP_ZSTACK_DEMO)
        // 任务建立实验范例代码
        // 任务列表
        Hello_ProcessEvent,
        Flash_ProcessEvent,
    #endif
    };
    const uint8 tasksCnt = sizeof(tasksArr)/sizeof(tasksArr[0]);

    // 初始化任务
    void osalInitTasks( void )
    {
        uint8 taskID = 0;
```

```
        macTaskInit( taskID++ );
        nwk_init( taskID++ );
        Hal_Init( taskID++ );
    #if defined( MT_TASK )
        MT_TaskInit( taskID++ );
    #endif
        APS_Init( taskID++ );
    #if defined ( ZIGBEE_FRAGMENTATION )
        APSF_Init( taskID++ );
    #endif
        ZDApp_Init( taskID++ );
    #if defined ( ZIGBEE_FREQ_AGILITY ) || defined
( ZIGBEE_PANID_CONFLICT )
        ZDNwkMgr_Init( taskID++ );
    #endif
    #if defined(SAPP_ZSTACK)
        sapp_taskInitProcess();
    #endif
    #if defined(SAPP_ZSTACK_DEMO)
        // 任务建立实验范例代码
        // 启动定时器
        osal_start_timerEx(taskID, 0x0001, 1000);
    #endif
    }
```

实验 7：创建 Zstack 功能节点周期性发送数据

一、实验目的

1. 了解基于 Z-Stack 协议栈的 SappWsn 应用程序框架的工作机制
2. 认识协议栈中将节点按照“功能端点”区分的思想

二、实验环境

1. 装有 IAR 开发工具的 PC 机一台
2. 实验箱一台
3. CCDebugger（以及 USB A-B 延长线）一个
4. USB Mini 延长线一根

三、实验原理

Z-Stack for SunplusAPP 简介

Z-Stack for SunplusAPP（以下简称 Z-Stack APP）是基于 TI 提供的 ZStack 中的 SampleAPP，修改了其应用层代码而得到一个完整的基于 Zigbee 2007 协议栈的 Zigbee 节点开发框架。

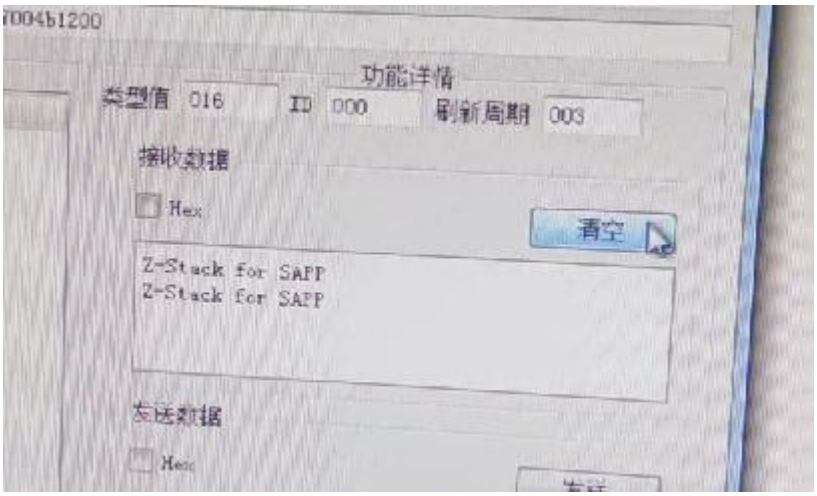
四、实验要求

在 SappWsn 应用程序框架下，编写程序，使得节点具有一个测试功能端点，该端点每隔 3 秒钟向协调器发送一个“Z-Stack for SunplusAPP”的字符串。

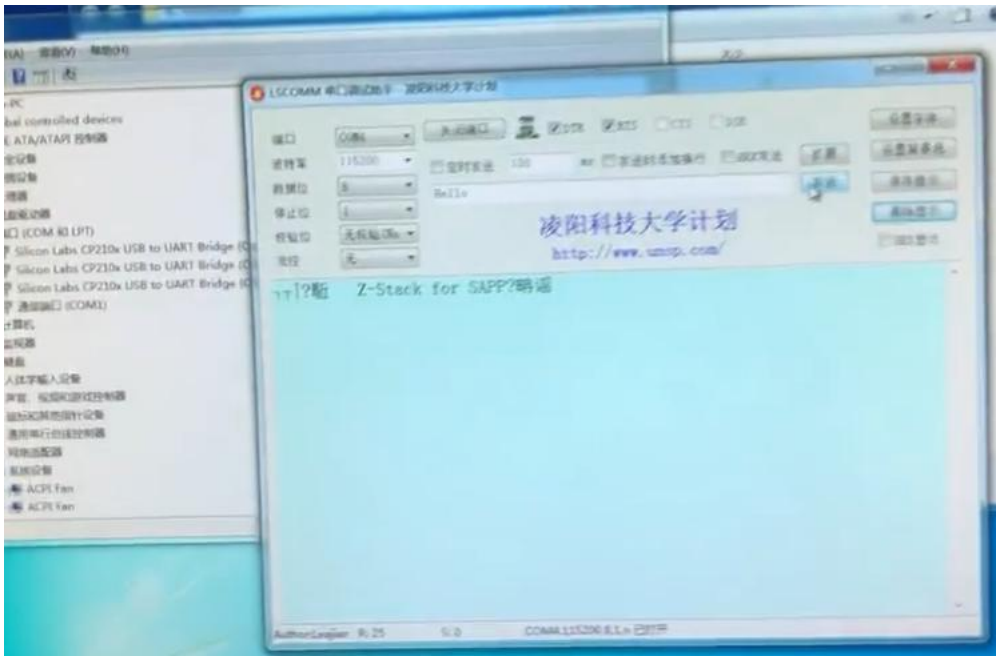
五、实验现象

实验现象已上传到网络：https://www.bilibili.com/video/BV1PM4y1g7ub/

周期性的文字信息成功发送给了协调器节点。协调器节点与计算机通过串口连接，可以直接发送信息。



雨滴节点将信息通过串口直接发送到计算机



六、代码分析

在 SAPP_Device.h 文件中添加功能控制的 define 定义。

```
//#define HAS_EXECUTEA          // 模拟执行器(预留扩展)
//#define HAS_REMOTER           // 红外遥控(预留扩展)
#define HAS_TESTFUNCTION        // 虚拟功能
//#define HAS_125KREADER        // 125K电子标签阅读器
//#define HAS_SPEAKER           // 语音报警器
```

在 EndDeviceEB 中的虚拟设备的代码中添加使用 UART 向计算机发送数据的代码。

```
/*********************************/
#if defined(HAS_TESTFUNCTION)
#define TEST_STRING     "Z-Stack for SAPP"
static uint8 lastData[119] = TEST_STRING;
static uint8 lastLen = 0;
void testFunc_RecvData(struct ep_info_t *ep, uint16 addr, uint8 endPoint, afMSGCommandF
void testFunc_RecvData(struct ep_info_t *ep, uint16 addr, uint8 endPoint, afMSGCommandF
{
    lastLen = msg->DataLength;
    memcpy(&lastData[sizeof(TEST_STRING) - 1], msg->Data, lastLen);
    SendData(ep->ep, lastData, 0x0000, TRANSFER_ENDPOINT,
                    lastLen + sizeof(TEST_STRING) - 1);
}
void testFunc_TimeOut(struct ep_info_t *ep);
void testFunc_TimeOut(struct ep_info_t *ep)
{
    SendData(ep->ep, lastData, 0x0000, TRANSFER_ENDPOINT,
                    lastLen + sizeof(TEST_STRING) - 1);
    UART0_Send( TEST_STRING,sizeof(TEST_STRING) - 1);
}
#endif
```

```
/*********************************/
/*  虚拟功能                     */
/*********************************/
#if defined(HAS_TESTFUNCTION)
#define TEST_STRING         "Z-Stack for SAPP"
```

```
    static uint8 lastData[119] = TEST_STRING;
    static uint8 lastLen = 0;
    void testFunc_RecvData(struct ep_info_t *ep, uint16 addr, uint8 endPoint,
afMSGCommandFormat_t *msg);
    void testFunc_RecvData(struct ep_info_t *ep, uint16 addr, uint8 endPoint,
afMSGCommandFormat_t *msg)
    {
        lastLen = msg->DataLength;
        memcpy(&lastData[sizeof(TEST_STRING) - 1], msg->Data, lastLen);
        SendData(ep->ep, lastData, 0x0000, TRANSFER_ENDPOINT,
                                lastLen + sizeof(TEST_STRING) - 1);
    }
    void testFunc_TimeOut(struct ep_info_t *ep);
    void testFunc_TimeOut(struct ep_info_t *ep)
    {
        SendData(ep->ep, lastData, 0x0000, TRANSFER_ENDPOINT,
                                lastLen + sizeof(TEST_STRING) - 1);
         UART0_Send( TEST_STRING, sizeof(TEST_STRING)-1);
    }
    #endif
```

实验 8：Zstack 的传感和与执行实验

实验 8.1：Zstack 的雨滴传感器实验

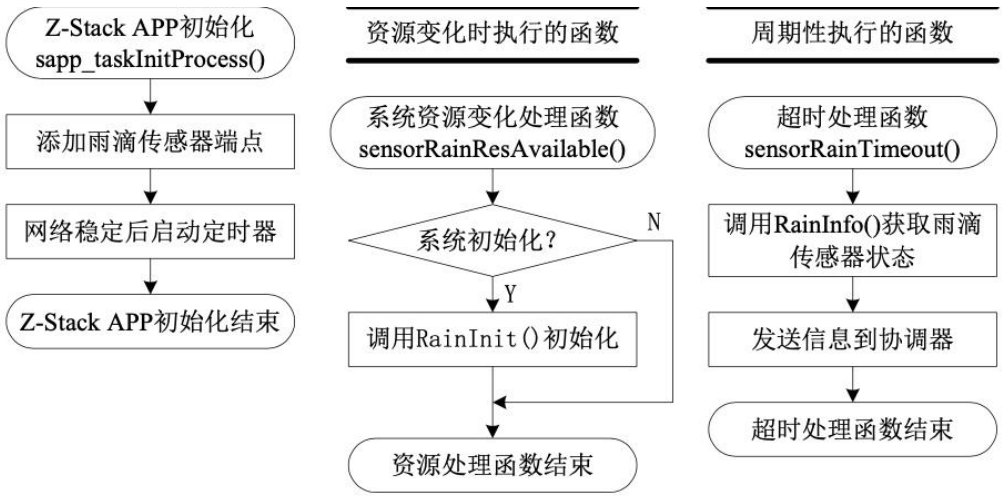
一、实验目的

了解基于 Z-Stack 协议栈的 SappWsn 应用程序框架的工作机制
掌握在 SappWsn 应用程序框架下添加雨滴传感器驱动的方法

二、实验环境

装有 IAR 开发工具的 PC 机一台
实验箱一台
CCDebugger(以及 USB A-B 延长线)一个
USB Mini 延长线一根

三、实验原理



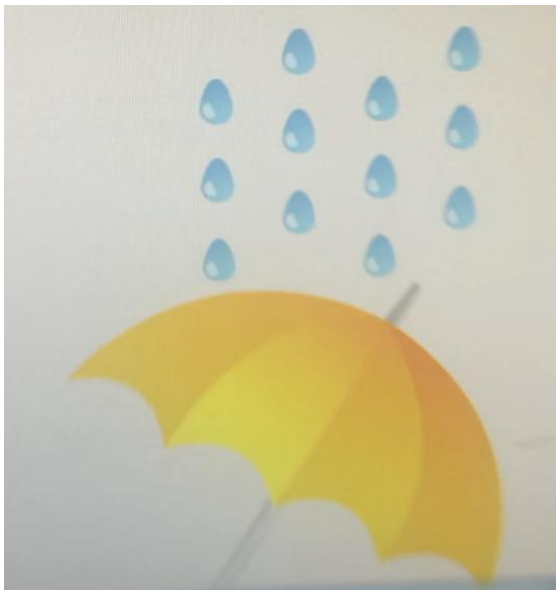
四、实验要求

利用 Z-Stack APP 应用程序框架，添加雨滴传感器的驱动程序，使得节点可以周期性发送雨滴传 感器的状态数据给协调器:当没有检测到有雨时发送 0，否则发送 1。

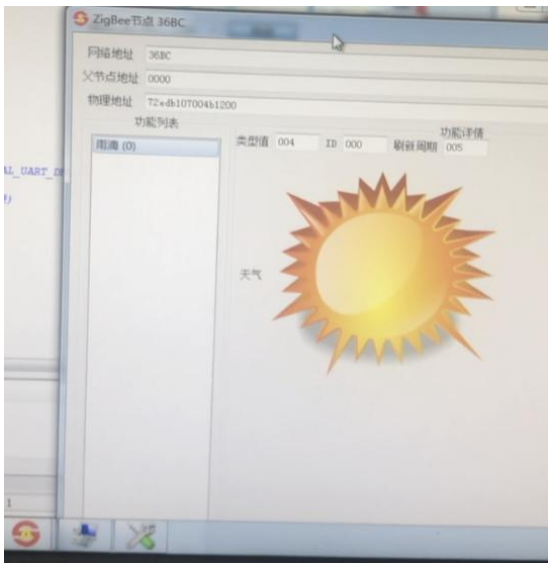
五、实验现象

实验现象已上传至网络：https://www.bilibili.com/video/BV1Fv411H7TU/
没有进行操作时，为晴天状态

进行操作（用手按雨滴传感器）后，是雨天状态



操作后，将手拿开，我们就又可以得到晴天状态了。



这样一来，我们就成功完成了雨滴的检测。

六、代码分析

```
#if defined(HAS_RAIN)
#define RAIN_IO_GROUP0
  #define RAIN_IO_BIT0
void sensorRainResAvailable(struct ep_info_t *ep, RES_TYPE type, void *res);
void sensorRainResAvailable(struct ep_info_t *ep, RES_TYPE type, void *res)
  {
if(type == ResInit) //雨滴传感器初始化
{
HalIOSetInput(RAIN_IO_GROUP, RAIN_IO_BIT, Pull_None);
HalIOIntSet(ep->ep, RAIN_IO_GROUP, RAIN_IO_BIT, IOInt_Rising, 0);
}
if(type == ResIOInt)//雨滴传感器收到雨的数据并发送
{
uint8 RainValue = 1;
SendData(ep->ep, &RainValue, 0x0000, TRANSFER_ENDPOINT,
sizeof(RainValue));
}
  }
void sensorRainTimeout(struct ep_info_t *ep);
void sensorRainTimeout(struct ep_info_t *ep)
  {
uint8 value = HalIOGetLevel(RAIN_IO_GROUP, RAIN_IO_BIT);
SendData(ep->ep, &value, 0x0000, TRANSFER_ENDPOINT, sizeof(value));
  }
#endif
```

实验 8.2：Zstack 执行器实验

一、实验目的

1.了解基于 Z-Stack 协议栈的 SappWsn 应用程序框架的工作机制
2.掌握在 SappWsn 应用程序框架下添加执行节点驱动的方法

二、 实验环境

装有 IAR 开发工具的 PC 机一台
实验箱一台
CCDebugger（以及 USB A-B 延长线）一个
USB Mini 延长线一根

三、实验原理

在 Z-Stack APP 中的 HAL\Target\CC2530EB\Includes 组中，提供了一个 hal_io.h 的文件。其中，提供了名为 HalIOSetOutput 的函数，可以将执行器端口（P1[4.5.6.7]）设置为输入，并通过 HalIOSetLevel 函数可以设置执行器端口输出的电平来控制四个继电器的导通和关闭状态。

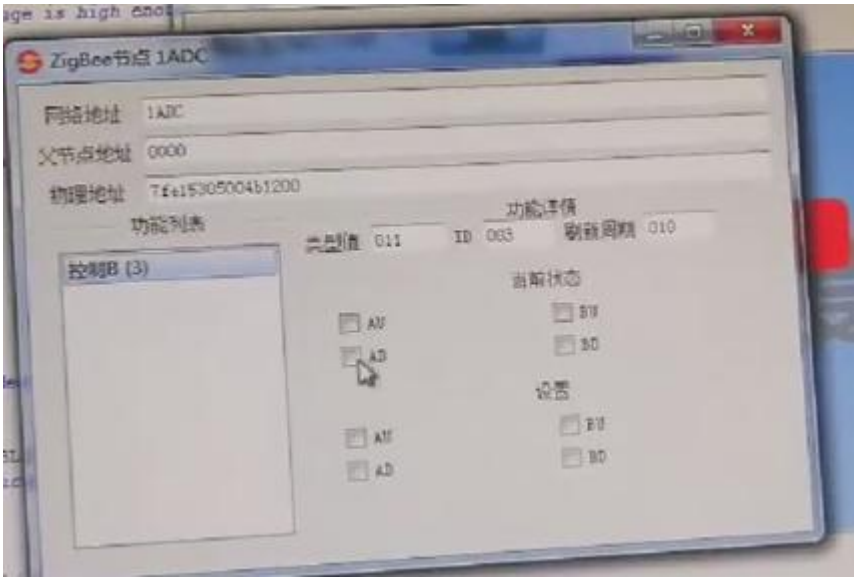
四、实验要求

利用 Z-Stack APP 应用程序框架，添加执行节点的驱动程序，使得节点可以周期性发送当前继电器的状态给协调器，同时也可以接收协调器的控制指令，控制继电器的状态发生变化。

五、实验步骤与现象

网络拓扑结构

在“当前状态”栏，显示的是四个继电器的当前状态；通过控制执行器节点，修改“设置状态”中的四个复选框的状态，并点击“应用”按钮，来控制四个继电器。



六、代码分析

```
/*******************************/
/* 二进制执行器传感器*/
/*******************************/
#if defined(HAS_EXECUTEB)
#define ControlInit()do { HalIOSetOutput(1,4);HalIOSetOutput(1,5);HalIOSetOutput(1,6);HalIOSetOutput(1,7);Control(0); } while(0)
#define Control(mask)do { HalIOSetLevel(1,4,mask&0x01);HalIOSetLevel(1,5,mask&0x02);HalIOSetLevel(1,6,mask&0x04);HalIOSetLevel(1,7,mask&0x08); } while(0)
void OutputExecuteBResAvailable(struct ep_info_t *ep, RES_TYPE type, void *res);
void OutputExecuteBResAvailable(struct ep_info_t *ep, RES_TYPE type, void *res)
{
if(type == ResInit)
ControlInit();
}
void outputExecuteB(struct ep_info_t *ep, uint16 addr, uint8 endPoint, afMSGCommandFormat_t *msg);
void outputExecuteB(struct ep_info_t *ep, uint16 addr, uint8 endPoint, afMSGCommandFormat_t *msg)
{
//msg->Data[], msg->DataLength, msg->TransSeqNumber
Control(msg->Data[0]);
SendData(ep->ep, &msg->Data[0], 0x0000, TRANSFER_ENDPOINT, 1);
}
void outputExecuteBTimeout(struct ep_info_t *ep);
void outputExecuteBTimeout(struct ep_info_t *ep)
{
uint8 value = P1 >> 4;
SendData(ep->ep, &value, 0x0000, TRANSFER_ENDPOINT, sizeof(value));
}
#endif
```

实验 9：Zstack 的雨滴和执行器控制闭环实验

一、 实验目的

1.了解基于 Z-Stack 协议栈的 SappWsn 应用程序框架的工作机制
2.掌握在 SappWsn 应用程序框架下将执行节点与雨滴传感器组网的方法

二、 实验环境

装有 IAR 开发工具的 PC 机一台
实验箱一台
CCDebugger（以及 USB A-B 延长线）一个
USB Mini 延长线一根

三、实验原理

在 雨滴传感器发送状态数据给协调器的同时，发送数据给执行器，以改变继电器的状态

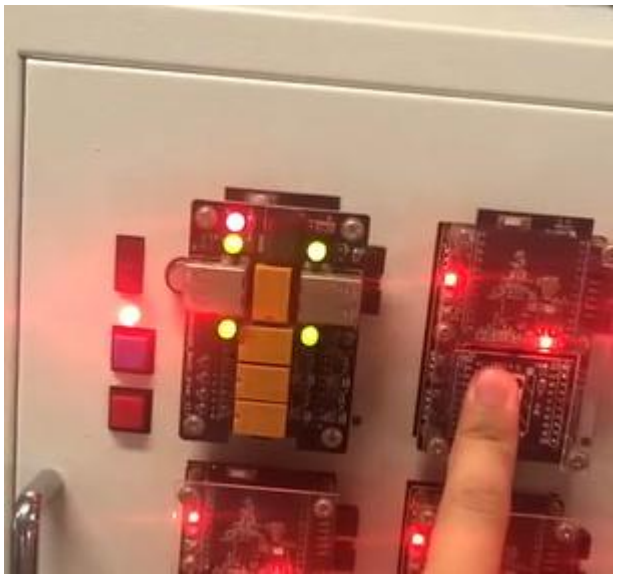
四、 实验要求

利用 Z-Stack APP 应用程序框架，同时添加执行节点和雨滴传感器的驱动程序，使得节点可以周期性发送雨滴传感器的状态数据给协调器，同时根据余地传感器的状态数据发送控制指令给执行节点，控制继电器的状态发生变化。

五、 实验现象

实验现象已上传至网络：https://www.bilibili.com/video/BV1VU4y1V7Sg/
用湿润的手按雨滴传感器后
雨滴传感器的状态发生改变

同时继电器的状态发生了改变



六、代码分析

```
#if defined(HAS_RAIN)
#define RAIN_IO_GROUP            0
#define RAIN_IO_BIT              0
void sensorRainResAvailable(struct ep_info_t *ep, RES_TYPE type, void *res);
void sensorRainResAvailable(struct ep_info_t *ep, RES_TYPE type, void *res)
{
    if(type == ResInit)
    {
        HalIOSetInput(RAIN_IO_GROUP, RAIN_IO_BIT, Pull_None);
        HalIOIntSet(ep->ep, RAIN_IO_GROUP, RAIN_IO_BIT, IOInt_Rising, 0);
    }//IO 端口中断触发，中断源检测
    if(type == ResIOInt)
    {
        uint8 RainValue = 1;
        SendData(ep->ep, &RainValue, 0x0000, TRANSFER_ENDPOINT,
sizeof(RainValue));
    }
}
void sensorRainTimeout(struct ep_info_t *ep);
void sensorRainTimeout(struct ep_info_t *ep)
{
    uint8 value = HalIOGetLevel(RAIN_IO_GROUP, RAIN_IO_BIT);
SendData(ep->ep, &value, 0x0000, TRANSFER_ENDPOINT, sizeof(value));
SendData(ep->ep, &value, 0xcc1B, TRANSFER_ENDPOINT, sizeof(value));
}
#endif
```