

IOT 实验报告

2052134 刘治华

目录

| | |
|--------------------------------------|----|
| 实验一：IAR 实验开发环境配置 | 2 |
| 实验二：LED 灯控制实验 | 6 |
| 实验三：定时控制实验 | 9 |
| 实验四：片上温度 AD 实验 | 13 |
| 实验五：串口收发数据实验 | 19 |
| 实验六：声响/光敏传感器实验实验 | 21 |
| 实验七：磁场强度传感器实验 | 24 |
| 实验八：霍尔开关传感器实验 | 27 |
| 实验九：红外对射传感器实验 | 29 |
| 实验十：温湿度传感器实验 | 33 |
| 实验十一：TI-BLE4.0 协议栈入门实验 | 34 |
| 实验十二：基于 BLE4.0 的使能从机 notify 实验 | 37 |
| 实验十三：基于 BLE4.0 协议栈的点对点通信实验 | 49 |

实验一：IAR 实验开发环境配置

实验内容

本次实验主要实现 IAR 环境的安装和配置，并熟悉新工程的建立和代码调试。

物联网课程实验的蓝牙 4.0(CC2540)模块实验部分，使用的软件开发环境为 IAR Embedded Workbench for MCS-51。本次实验主要目的是学习如何使用该 IAR 环境搭建配套实验工程。后续实验的工程建立方法都是参照本节设置建立。

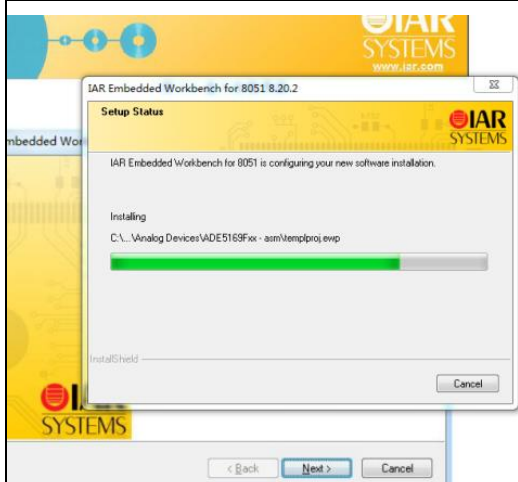
实验原理

本次实验主要是了 IAR 实验开发环境的安装和配置，并通过新建案例工程学习了建立新项目。

蓝牙 4.0(CC2540)模块实验部分，使用的软件开发环境为 IAR Embedded Workbench for MCS-51。关于 IAR 的详细说明文档请浏览 IAR 官方网站或软件安装文件夹下 8051\doc 里的支持文档。

IAR Systems 是全球领先的嵌入式系统开发工具和服务的供应商。公司成立于 1983 年，提供的产品和服务涉及到嵌入式系统的设计、开发和测试的每一个阶段，包括：带有 C/C++编译器和调试器的集成开发环境(IDE)、实时操作系统和中间件、开发套件、硬件仿真器以及状态机建模工具。

实验过程



① 安装软件

打开光盘目录\移动互联网无线传感网络部分\tools\BLE4.0\IAR-V820 软件，双击运行 EW8051-8202-Autorun.exe，进入安装界面，单击“Install IAR Embedded Workbench”进入安装界面。

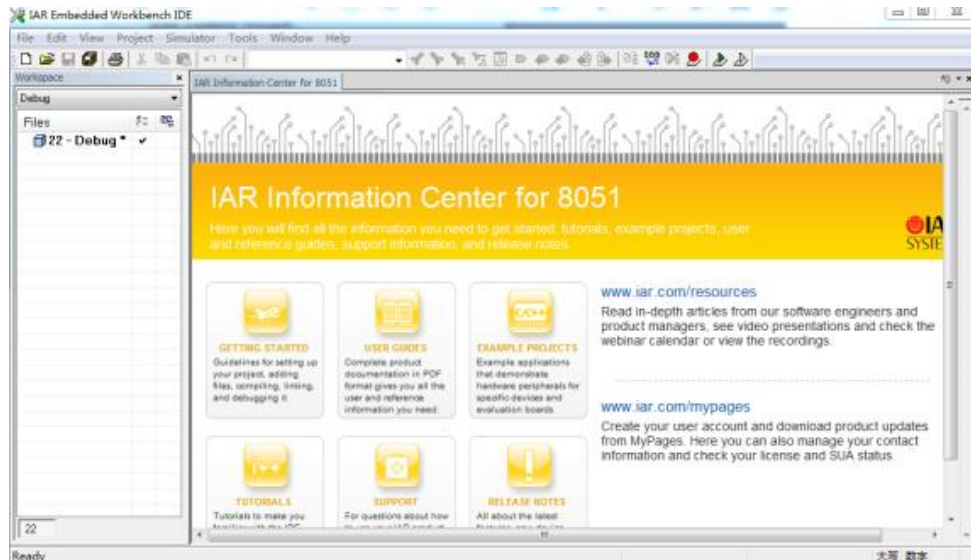
② 建立模板工程

1) 建立新工程

打开 IAR 软件，默认进入建立工作区菜单，我们先选择-取消，进入 IAR IDE 环境。点击 Project 菜单，选择 Create New Project

确认 Tool chain 栏已经选择 8051，在 Project templates: 栏选择 Empty project 点击下方 OK 按钮

点击右上角新建文件夹，创建新文件夹。在计算机相应目录下，创建工程目录，本例创建了 test_iar 目录，用来存放工程，进入到创建的 test_iar 文件夹中，更改工程名，如 test 点击 Save，这样便建立了一个空的工程。



2) 添加工程文件

选择菜单 Project\Add File 或在工作区窗口中，在工程名上点右键，在弹出的快捷菜单中选择 Add File，弹出文件打开对话框，选择需要的文件点击 打开 退出。

向文件里添加如下代码：

```
#include "ioCC2540.h"
void Delay(unsigned char n)
{
    unsigned char i;
    unsigned int j;
    for(i = 0; i < n; i++)
        for(j = 1; j < 1000; j++);
}
void main(void)
{
    P1SEL = 0x00; //P1 配置为普通 I/O 口
    P1DIR |= 0x2; //P1.1 输出
    P0SEL = 0x00; //P0 配置为普通 I/O 口
    P0DIR |= 0x1; //P0.0 输出
    while(1)
    {
        P1_1 = 1;
        Delay(10);
```

```

        P0_0 = 0;
        Delay(10);
        P1_1 = 0;
        Delay(10);
        P0_0 = 1;
        Delay(10);
    }
}

```

选择菜单 File\Save 弹出保存对话框,填写文件名为 test.c,点击保存。按照前面添加文件的方法将 test.c 添加到当前工程里,在工程中右键添加刚刚编写好的文件 test.c。

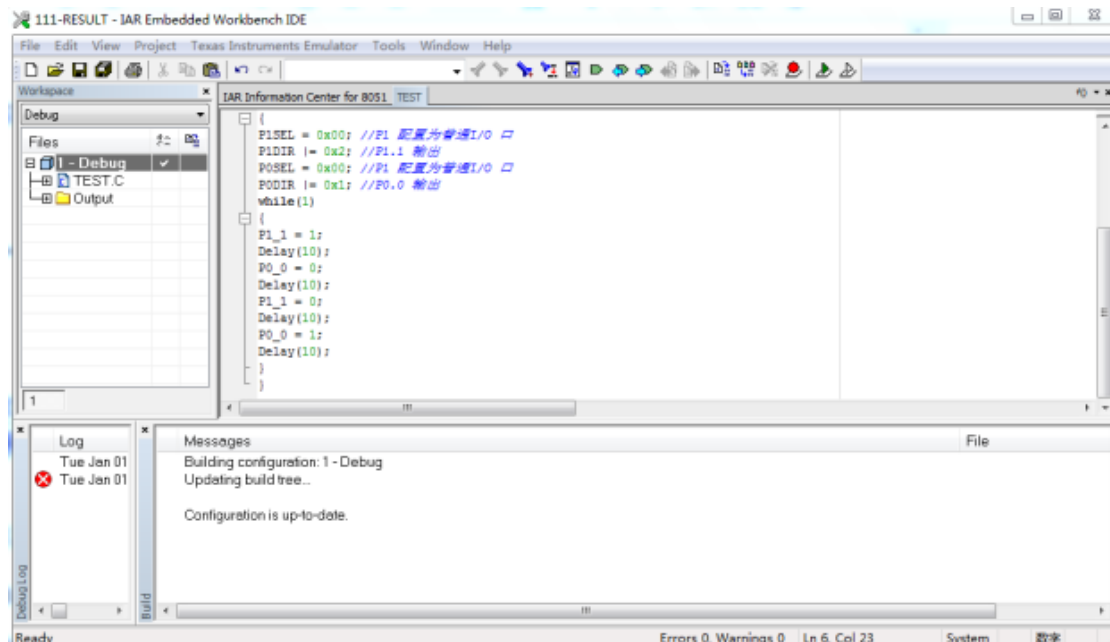
3) 配置工程选项

选择 Project 菜单下的 Options... 配置与 CC2540 相关的选项。 General Options--Target 标签: 按实验指导书配置 Target, 选择 Code model 为 Near 和 Data model 为 Large, Calling convention 为 XDATA stack reentrant 以及其它参数 点击 Derivative information 栏右边的按钮, 选择程序安装位置如这里是 IAR Systems\Embedded Workbench 6.4\8051\config\devices\Texas Instruments 下的文件 CC2540F256.i51。按照实验指导书进行配置。

在 Device Description file 选择 ioCC2540F256.ddf 文件, 其位置在程序安装文件夹下如 C:\Program Files\IAR Systems\Embedded Workbench 6.4\8051\config\devices\Texas Instruments

4) 编译和链接

选择 Project\Make 或 按 F7 键编译和连接工程。



实验现象

本次实验我们最终观察到案例项目正常编译运行，现象截图可参考“实验过程”部分。

思考总结

本次实验我们小组一起完成了实验环境的搭建，对实验环境和套件的熟悉程度对于接下来基本实验的完成是非常关键的。因此在实践过程中我们按照指导书一步步完成，最终样例程序得以成功运行。

实验二：LED 灯控制实验

实验内容

- 1、阅读移动互联网教学科研平台蓝牙 4.0 模块硬件部分文档，熟悉蓝牙 4.0 模块硬件接口；
- 2、使用 IAR 开发环境设计程序，利用 CC2540 的 IO 控制 LED 外设的闪烁。

本次实验需要完成程序通过配置 CC2540 IO 寄存器的高低电平来控制 LED 灯的状态，用循环语句来实现程序的不间断运行。

代码内容如下：

```
#include <ioCC2540.h>

#define uint unsigned int
#define uchar unsigned char

//定义控制 LED 灯的端口
#define LED0 P1_1 //定义 LED0 为 P11 口控制
#define LED1 P0_0 //定义 LED1 为 P00 口控制

//函数声明
void Delay(uint); //延时函数
void Initial(void); //初始化 P1、P0 口

/*****
//延时函数
*****/
void Delay(uint n)
{
    uint i,t;
    for(i = 0;i<5;i++)
        for(t = 0;t<n;t++);
}

/*****
//初始化程序
*****/
void Initial(void)
{
    P1DIR |= 0x2; //P1.1 输出
    P0DIR |= 0x1; //P0.0 输出
    LED0 = 0; //LED0 灯熄灭
```

```

    LED1 = 1; //LED1 灯熄灭
}

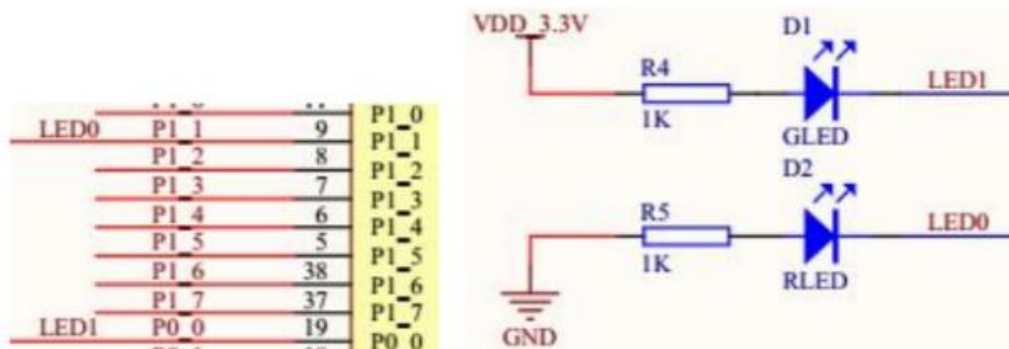
/*****
//主函数
*****/
void main(void)
{
    Initial(); //调用初始化函数
    LED0 = 1; //LED0 点亮
    LED1 = 0; //LED1 点亮
    while(1)
    {
        LED1 = !LED1; //LED2 闪烁
        Delay(50000);
    }
}

```

实验原理

①蓝牙 4.0(CC2540)模块 LED 硬件接口

蓝牙 4.0(CC2540)模块硬件上设计有 2 个 LED 灯，用来编程调试使用。分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时，LED1 灯点亮，P1_1 引脚为高电平时 LED0 灯亮。



②CC2540 IO 相关寄存器

以下图表列出了关于 CC2540 处理器的 P0 和 P1 IO 相关寄存器，P1 和 P0 寄存器为可读写的寄存器，P1DIR 和 P0DIR 为 IO 输入输出选择寄存器，其他 IO 寄存器的功能，使用默认配置。

| P0 (0x80) – Port 0 | | | | |
|--------------------|---------|-------|-----|--|
| Bit | Name | Reset | R/W | Description |
| 7:0 | P0[7:0] | 0xFF | R/W | Port 0. General-purpose I/O port. Bit-addressable from SFR. This CPU-internal register is readable, but not writable, from XDATA (0x7080). |

| P1 (0x90) – Port 1 | | | | |
|--------------------|---------|-------|-----|--|
| Bit | Name | Reset | R/W | Description |
| 7:0 | P1[7:0] | 0xFF | R/W | Port 1. General-purpose I/O port. Bit-addressable from SFR. This CPU-internal register is readable, but not writable, from XDATA (0x7090). |

表 3.1.1 P0和P1 寄存器

| P0DIR (0xFD) – Port 0 Direction | | | | |
|---------------------------------|-------------|-------|-----|---|
| Bit | Name | Reset | R/W | Description |
| 7:0 | DIRP0_[7:0] | 0x00 | R/W | P0.7 to P0.0 I/O direction 0: Input 1: Output |

| P1DIR (0xFE) – Port 1 Direction | | | | |
|---------------------------------|-------------|-------|-----|---|
| Bit | Name | Reset | R/W | Description |
| 7:0 | DIRP1_[7:0] | 0x00 | R/W | P1.7 to P1.0 I/O direction 0: Input 1: Output |

| |
|---|
| 实验过程 |
| <ol style="list-style-type: none"> 1. 使用 USB 线连接 PC 机和移动互联网教学科研平台的 CC-debug 接口 2. 启动 IAR 开发环境，新建工程，或直接使用 work\Basic\Exp1 实验工程。 3. 在 IAR 开发环境中编译、运行、调试程序。 |

| |
|---|
| 实验现象 |
| <p>本次实验我们观察到 LED 灯按照程序指定的逻辑闪烁。也就是 LED0 点亮，LED1 点亮，接着 LED2 闪烁。</p>  |

| |
|---|
| 思考总结 |
| <p>本次实验小组一起完成了 LED 灯的控制，通过阅读源代码我们对 IAR 蓝牙系列实验有了更底层的了解。本次实验内容比较简单，但在动手运行调试的过程中我们摸清了软件和硬件之间的联结，以便为接下来的实验铺平道路。</p> |

实验三：定时控制实验

实验内容

1. 阅读移动互联网教学科研平台蓝牙 4.0 模块硬件部分文档，熟悉蓝牙 4.0 模块硬件接口。

2. 使用 IAR 开发环境设计程序，利用 CC2540 的 Timer1 定时器控制 LED 外设的闪烁。

程序设计如下：

```
#include <ioCC2540.h>
```

```
#define uint unsigned int
```

```
#define uchar unsigned char
```

```
#define LED0 P1_1 //定义 LED0 为 P11 口控制
```

```
#define LED1 P0_0 //定义 LED1 为 P00 口控制
```

```
uint counter=0; //统计溢出次数
```

```
uint TempFlag; //用来标志是否要闪烁
```

```
void Initial(void);
```

```
void Delay(uint);
```

```
/*  
*****
```

```
//延时程序
```

```
*****/  
*/
```

```
void Delay(uint n)
```

```
{
```

```
    uint i,t;
```

```
    for(i = 0;i<5;i++)
```

```
        for(t = 0;t<n;t++);
```

```
}
```

```
/*  
*****
```

```
//初始化程序
```

```
*****/  
*/
```

```
void Initial(void)
```

```
{
```

```
    P1DIR |= 0x2; //P1.1 输出
```

```
    P0DIR |= 0x1; //P0.0 输出
```

```
    LED0 = 0; //LED0 灯熄灭
```

```
    LED1 = 1; //LED1 灯熄灭
```

```
    //初始化 T1 定时器
```

```
    T1CTL = 0x0d; //中断无效,128 分频;自动重装模式(0x0000->0xffff);
```

```
}
```

```
/*  
*****
```

```

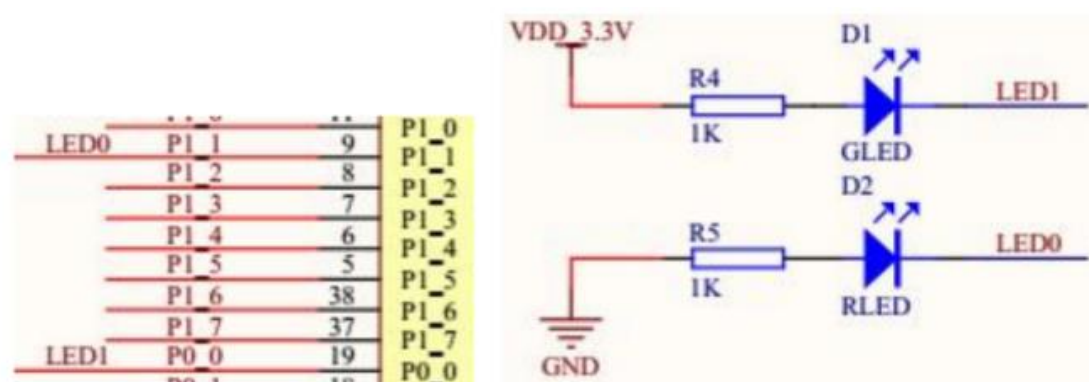
//主函数
*****/
void main()
{
    Initial(); //调用初始化函数
    LED1 = 0; //点亮 LED1
    while(1) //查询溢出
    {
        if(IRCON > 0)
        {
            IRCON = 0; //清溢出标志
            TempFlag = !TempFlag;
        }
        if(TempFlag)
        {
            LED0 = !LED1;
            LED1 = !LED1;
            Delay(6000);
        }
    }
}

```

实验原理

① 蓝牙 4.0(CC2540)模块 LED 硬件接口

蓝牙 4.0(CC2540)模块硬件上设计有 2 个 LED 灯，用来编程调试使用。分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时候，LED1 灯点亮，P1_1 引脚为高电平 时 LED0 灯亮。



③ CC2540 IO 相关寄存器

以下图表列出了关于 CC2540 处理器的 P0 和 P1 IO 相关寄存器，P1 和 P0 寄存器为可读写的寄存器，P1DIR 和 P0DIR 为 IO 输入输出选择寄存器，其他 IO 寄存器的功能，使用默认配置。详情请用户参考 CC2540 的用户指导手册。

P0 (0x80) – Port 0

| Bit | Name | Reset | R/W | Description |
|-----|---------|-------|-----|--|
| 7:0 | P0[7:0] | 0xFF | R/W | Port 0. General-purpose I/O port. Bit-addressable from SFR. This CPU-internal register is readable, but not writable, from XDATA (0x7080). |

P1 (0x90) – Port 1

| Bit | Name | Reset | R/W | Description |
|-----|---------|-------|-----|--|
| 7:0 | P1[7:0] | 0xFF | R/W | Port 1. General-purpose I/O port. Bit-addressable from SFR. This CPU-internal register is readable, but not writable, from XDATA (0x7090). |

表 3.2.1 P0和P1 寄存器

P0DIR (0xFD) – Port 0 Direction

| Bit | Name | Reset | R/W | Description |
|-----|-------------|-------|-----|---|
| 7:0 | DIRP0_[7:0] | 0x00 | R/W | P0.7 to P0.0 I/O direction 0: Input 1: Output |

P1DIR (0xFE) – Port 1 Direction

| Bit | Name | Reset | R/W | Description |
|-----|-------------|-------|-----|---|
| 7:0 | DIRP1_[7:0] | 0x00 | R/W | P1.7 to P1.0 I/O direction 0: Input 1: Output |

以下图表列举了和 CC2540 处理器 Timer1 定时器相关的寄存器，其中 T1CTL 为 Timer1 定时器 控制状态寄存器，通过该寄存器来设置定时器的模式和预分频系数。IRCON 寄存器为中断标志位寄存器，通过该寄存器可以判断相应控制器 Timer1 的中断状态。

T1CTL (0xE4) – Timer 1 Control

| Bit | Name | Reset | R/W | Description |
|-----|-----------|-------|-----|--|
| 7:4 | — | 0000 | R0 | Reserved |
| 3:2 | DIV[1:0] | 00 | R/W | Prescaler divider value. Generates the active clock edge used to update the counter as follows: 00: Tick frequency/1 01: Tick frequency/8 10: Tick frequency/32 11: Tick frequency/128 |
| 1:0 | MODE[1:0] | 00 | R/W | Timer 1 mode select. The timer operating mode is selected as follows: 00: Operation is suspended. 01: Free-running, repeatedly count from 0x0000 to 0xFFFF. 10: Modulo, repeatedly count from 0x0000 to T1CC0. 11: Up/down, repeatedly count from 0x0000 to T1CC0 and from T1CC0 down to 0x0000. |

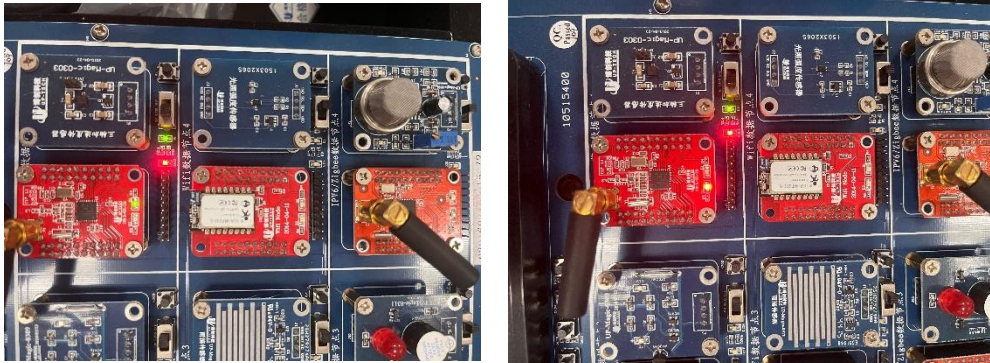
| Bit | Name | Reset | R/W | Description |
|-----|-------|-------|-----------|---|
| 7 | STIF | 0 | R/W | Sleep Timer interrupt flag 0: Interrupt not pending 1: Interrupt pending |
| 6 | — | 0 | R/W | Must be written 0. Writing a 1 always enables the interrupt source. |
| 5 | POIF | 0 | R/W | Port 0 interrupt flag 0: Interrupt not pending 1: Interrupt pending |
| 4 | T4IF | 0 | R/W H0 | Timer 4 interrupt flag. Set to 1 when Timer 4 interrupt occurs and cleared when CPU vectors to the interrupt service routine. 0: Interrupt not pending 1: Interrupt pending |
| 3 | T3IF | 0 | R/W H0 | Timer 3 interrupt flag. Set to 1 when Timer 3 interrupt occurs and cleared when CPU vectors to the interrupt service routine. 0: Interrupt not pending 1: Interrupt pending |
| 2 | T2IF | 0 | R/W H0 | Timer 2 interrupt flag. Set to 1 when Timer 2 interrupt occurs and cleared when CPU vectors to the interrupt service routine. 0: Interrupt not pending 1: Interrupt pending |
| 1 | T1IF | 0 | R/W H0 | Timer 1 interrupt flag. Set to 1 when Timer 1 interrupt occurs and cleared when CPU vectors to the interrupt service routine. 0: Interrupt not pending 1: Interrupt pending |
| 0 | DMAIF | 0 | R/W | DMA-complete interrupt flag 0: Interrupt not pending 1: Interrupt pending |

实验过程

1. 使用 USB 线连接 PC 机和移动互联网教学科研平台的 CC-debug 接口
2. 启动 IAR 开发环境，新建工程，或直接使用 work\Basic\Exp2 实验工程
3. 在 IAR 开发环境中编译、运行、调试程序。

实验现象

本次实验我们按预期观察到 LED 灯按照程序设置的逻辑顺序闪烁。



思考总结

本次实验小组一起完成了通过软件程序调用来控制 LED 灯的规律闪烁。在实验过程中我们通过利用 CC2540 的 Timer1 定时器，学习了如何控制 LED 灯外设，同时也更加熟悉了上节课配置的 IAR 开发环境程序。

实验四：片上温度 AD 实验

实验内容

① 阅读移动互联网教学科研平台蓝牙 4.0 模块硬件部分文档,熟悉蓝牙 4.0 模块硬件接口。

② 使用 IAR 开发环境设计程序,利用 CC2540 的内部温度传感器作为 A/D 输入源,将转换后的温度数值利用串口发送给 PC 机终端。

程序设计如下:

```
#include <iocc2540.h>
#include <stdio.h>
#include "../uart/hal_uart.h"
#define uchar unsigned char
#define uint unsigned int
#define uint8 uchar
#define uint16 uint
#define TRUE 1
#define FALSE 0
//定义控制 LED 灯的端口
#define LED0 P1_1 //定义 LED0 为 P11 口控制
#define LED1 P0_0 //定义 LED1 为 P00 口控制
//#define HAL_MCU_CC2530 1
// ADC definitions for CC2530/CC2530 from the hal_adc.c file
#define HAL_ADC_REF_125V 0x00 /* Internal 1.25V Reference */
#define HAL_ADC_DEC_064 0x00 /* Decimate by 64 : 8-bit resolution */
#define HAL_ADC_DEC_128 0x10 /* Decimate by 128 : 10-bit resolution */
#define HAL_ADC_DEC_512 0x30 /* Decimate by 512 : 14-bit resolution */
#define HAL_ADC_CHN_VDD3 0x0f /* Input channel: VDD/3 */
#define HAL_ADC_CHN_TEMP 0x0e /* Temperature sensor */
/*****
//延时函数
*****/
void Delay(uint n)
{
    uint i,t;
    for(i = 0;i<5;i++)
        for(t = 0;t<n;t++);
}
void InitLed(void)
{
    P1DIR |= 0x2; //P1.1 输出
    P0DIR |= 0x1; //P0.0 输出
```

```

LED0 = 0; //LED0 灯熄灭
LED1 = 1; //LED1 灯熄灭
}
/*****
*****
* @fn readTemp
* @brief read temperature from ADC
* @param none
* @return temperature
*/
static char readTemp(void)
{
static uint16 voltageAtTemp22;
static uint8 bCalibrate=TRUE; // Calibrate the first time the temp
sensor is read
uint16 value;
char temp;
ATEST = 0x01;
TR0 |= 0x01;
ADCIF = 0; //clear ADC interrupt flag
ADCCON3 = (HAL_ADC_REF_125V | HAL_ADC_DEC_512 | HAL_ADC_CHN_TEMP);
while ( !ADCIF ); //wait for the conversion to finish
value = ADCL; //get the result
value |= ((uint16) ADCH) << 8;
value >>= 4; // Use the 12 MSB of adcValue
/*
* These parameters are typical values and need to be calibrated
* See the datasheet for the appropriate chip for more details
* also, the math below may not be very accurate
*/
/* Assume ADC = 1480 at 25C and ADC = 4/C */
#define VOLTAGE_AT_TEMP_25 1480
#define TEMP_COEFFICIENT 4
// Calibrate for 22C the first time the temp sensor is read.
// This will assume that the demo is started up in temperature of 22C
if(bCalibrate)
{
voltageAtTemp22=value;
bCalibrate=FALSE;
}
temp = 22 + ( (value - voltageAtTemp22) / TEMP_COEFFICIENT );
// Set 0C as minimum temperature, and 100C as max
if( temp >= 100) return 100;
else if(temp <= 0) return 0;

```

```

else return temp;
}
/*****
*****
* @fn readVoltage
* @brief read voltage from ADC
* @param none
* @return voltage
*/

void main(void)
{
char temp_buf[20]; //, vol_buf[20];
uint8 temp; //, vol;
InitUart(); //baudrate: 57600
InitLed();
LED1 = 0;
while(1)
{
LED0 = !LED0; //LED2 blink 表示程序运行正常
temp = readTemp(); //read temperature value
//vol = readVoltage();
sprintf(temp_buf, (char*)"temperature:%d\r\n", temp);
//sprintf(vol_buf, (char*)"vol:%d\r\n", vol);
prints(temp_buf);
//prints(vol_buf);
Delay(50000); Delay(50000); Delay(50000);
} }

```

实验原理

1、蓝牙 4.0 (CC2540) 模块 LED 硬件接

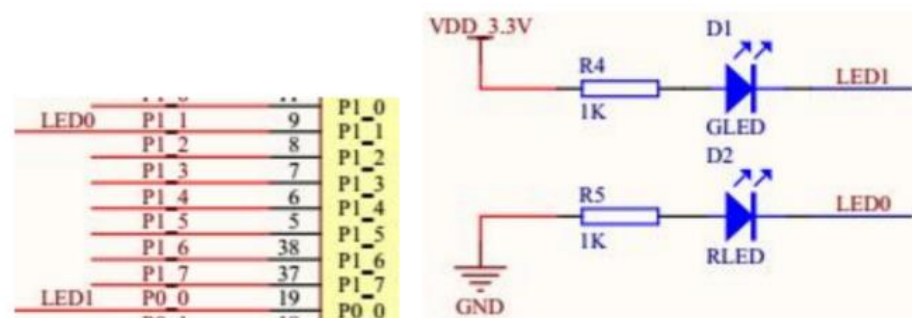
蓝牙 4.0 (CC2540) 模块硬件上设计有 2 个 LED 灯，用来编程调试使用。分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时，LED1

灯点亮，

P1_1 引脚为

高电平时

LED0 灯亮。



2、CC2540 IO 相关寄存器

以下图表列出了关于 CC2540 处理器的 P0 和 P1 IO 相关寄存器，P1 和 P0 寄存器为可读写的寄存器，P1DIR 和 P0DIR 为 IO 输入输出选择寄存器，其他 IO 寄存器的功能，使用默认配置。详情请用户参考 CC2540 的用户指导手册。

P0 (0x80) – Port 0

| Bit | Name | Reset | R/W | Description |
|-----|---------|-------|-----|--|
| 7:0 | P0[7:0] | 0xFF | R/W | Port 0. General-purpose I/O port. Bit-addressable from SFR. This CPU-internal register is readable, but not writable, from XDATA (0x7080). |

P1 (0x90) – Port 1

| Bit | Name | Reset | R/W | Description |
|-----|---------|-------|-----|--|
| 7:0 | P1[7:0] | 0xFF | R/W | Port 1. General-purpose I/O port. Bit-addressable from SFR. This CPU-internal register is readable, but not writable, from XDATA (0x7090). |

以下图表列举了部分和 CC2540 处理器、内部温度传感器操作相关的寄存器，其中包括 CLKCONCMD 和 CLKCONSTA 控制寄存器，用来控制系统时钟源和状态，SLEEP_CMD 和 SLEEP_STA 寄存器用来控制各种时钟源的开关和状态。另外还有未列出的 PERCFG 寄存器为外设功能控制寄存器，用来控制外设功能模式。而 UOCSR、UOGR、UOBUF、UOBAUD 等为串口相关寄存器。ADCCON1 和 ADCCON3 分别为 AD 转换控制器和 AD 转换设置寄存器。

| Bit | Name | Reset | R/W | Description |
|-----|--------------|-------|-----|---|
| 7 | OSC32K | 1 | R/W | 32 kHz clock-source select. Setting this bit initiates a clock-source change only. CLKCONSTA.OSC32K reflects the current setting. The 16 MHz RCOSC must be selected as system clock when this bit is to be changed. 0: 32 kHz XOSC 1: 32 kHz RCOSC |
| 6 | OSC | 1 | R/W | System clock-source select. Setting this bit initiates a clock-source change only. CLKCONSTA.OSC reflects the current setting. 0: 32 MHz XOSC 1: 16 MHz RCOSC |
| 5:3 | TICKSPD[2:0] | 001 | R/W | Timer ticks output setting. Cannot be higher than system clock setting given by OSC bit setting. 000: 32 MHz 001: 16 MHz 010: 8 MHz 011: 4 MHz 100: 2 MHz 101: 1 MHz 110: 500 kHz 111: 250 kHz Note that CLKCONCMD.TICKSPD can be set to any value, but the effect is limited by the CLKCONCMD.OSC setting; i.e., if CLKCONCMD.OSC = 1 and CLKCONCMD.TICKSPD = 000, CLKCONSTA.TICKSPD reads 001, and the real TICKSPD is 16 MHz. |
| 2:0 | CLKSPD | 001 | R/W | Clock speed. Cannot be higher than system clock setting given by the OSC bit setting. Indicates current system-clock frequency 000: 32 MHz 001: 16 MHz 010: 8 MHz 011: 4 MHz 100: 2 MHz 101: 1 MHz 110: 500 kHz 111: 250 kHz Note that CLKCONCMD.CLKSPD can be set to any value, but the effect is limited by the CLKCONCMD.OSC setting; i.e., if CLKCONCMD.OSC = 1 and CLKCONCMD.CLKSPD = 000, CLKCONSTA.CLKSPD reads 001, and the real CLKSPD is 16 MHz. Note also that the debugger cannot be used with a divided system clock. When running the debugger, the value of CLKCONCMD.CLKSPD should be set to 000 when CLKCONCMD.OSC = 0 or to 001 when CLKCONCMD.OSC = 1. |

| Bit | Name | Reset | R/W | Description |
|-----|--------------|-------|-----|---|
| 7 | OSC32K | 1 | R | Current 32-kHz clock source selected: 0: 32-kHz XOSC 1: 32-kHz RCOSC |
| 6 | OSC | 1 | R | Current system clock selected: 0: 32-MHz XOSC 1: 16-MHz RCOSC |
| 5:3 | TICKSPD[2:0] | 001 | R | Current timer ticks outputs setting 000: 32 MHz 001: 16 MHz 010: 8 MHz 011: 4 MHz 100: 2 MHz 101: 1 MHz 110: 500 kHz 111: 250 kHz |
| 2:0 | CLKSPD | 001 | R | Current clock speed 000: 32 MHz 001: 16 MHz 010: 8 MHz 011: 4 MHz 100: 2 MHz 101: 1 MHz 110: 500 kHz 111: 250 kHz |

SLEEP_CMD[0x8E] – Sleep-Mode Control Command

| Bit | Name | Reset | R/W | Description |
|-----|---------------|-------|-----|---|
| 7 | OSC32K_CALDIS | 0 | R/W | Disable 32-kHz RC oscillator calibration 0: 32-kHz RC oscillator calibration is enabled. 1: 32-kHz RC oscillator calibration is disabled. This setting can be written at any time, but does not take effect before the chip has been running on the 16-MHz high-frequency RC oscillator. |
| 6:3 | — | 0000 | RO | Reserved |
| 2 | — | 1 | R/W | Reserved. Always write as 1 |
| 1:0 | MODE[1:0] | 00 | R/W | Power-mode setting 00: Active/Idle mode 01: Power mode 1 (PM1) 10: Power mode 2 (PM2) 11: Power mode 3 (PM3) |

SLEEP_STA[0x9D] – Sleep-Mode Control Status

| Bit | Name | Reset | R/W | Description |
|-----|---------------|-------|-----|--|
| 7 | OSC32K_CALDIS | 0 | R | 32-kHz RC oscillator calibration status SLEEP_STA.OSC32K_CALDIS shows the current status of disabling of the 32-kHz RC calibration. The bit is not set to the same value as SLEEP_CMD.OSC32K_CALDIS before the chip has been run on the 32-kHz RC oscillator. |
| 6:5 | — | 00 | R | Reserved |
| 4:3 | RST[1:0] | XX | R | Status bit indicating the cause of the last reset. If there are multiple resets, the register only contains the last event. 00: Power-on reset and brownout detection 01: External reset 10: Watchdog Timer reset 11: Clock loss reset |
| 2:1 | — | 00 | R | Reserved |
| 0 | CLK32K | 0 | R | The 32-kHz clock signal (synchronized to the system clock) |

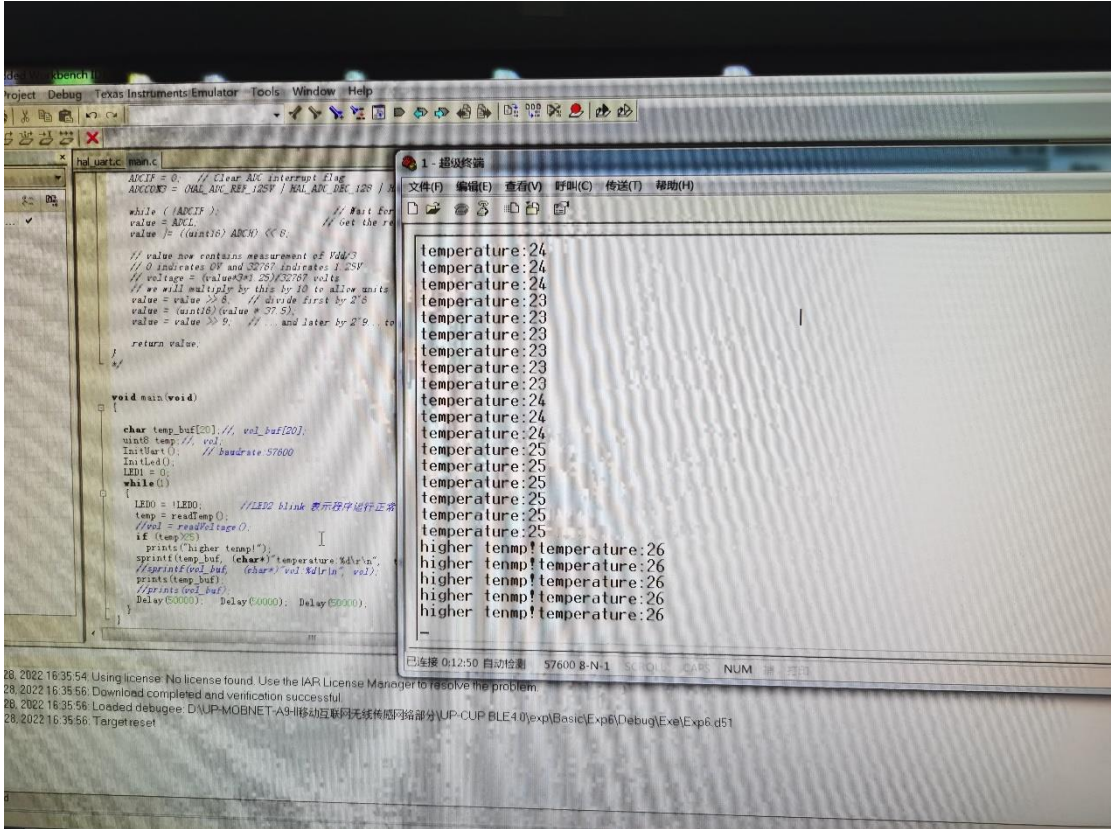
实验过程

1. 使用 USB 线连接 PC 机和移动互联网教学科研平台的 CC-debug 接口, 将系统配套串口线一端连接 PC 机, 一端连接移动互联网平台的串口 2 (com2) 上。
2. 启动 IAR 开发环境, 新建工程, 或直接使用 Exp6 实验工程。
3. 在 IAR 开发环境中编译、运行、调试程序。

4. 使用 PC 机自带的超级终端连接串口，将超级终端设置为串口波特率 57600、8 位、无奇偶校验、无硬件流模式，即可在终端收到模块传递过来的温度值。

实验现象

本次实验我们按预期观察到显示温度随我们手指的接触而进行变化。



总结思考

本次实验我们小组一起完成了通过软件程序调用与外部芯片检测来观察芯片温度的变化。在实验过程中我们通过利用 CC2540 内部温度传感器作为 A/D 输入源，将转换后的温度数值利用串口发送给 PC 机终端。学习了如何将外部变化与计算机内部检测相结合，同时也更加熟悉了上节课配置的 IAR 开发环境程序。是一次受益匪浅的实验经历。

实验五：串口收发数据实验

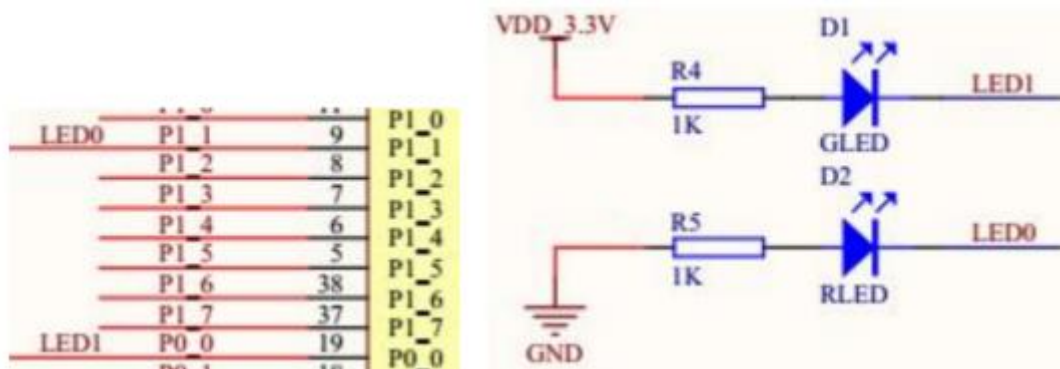
实验内容

1. 阅读移动互联网教学科研平台蓝牙 4.0 模块硬件部分文档，熟悉蓝牙 4.0 模块硬件接口。
2. 使用 IAR 开发环境设计程序，利用 CC2540 的内部温度传感器作为 A/D 输入源，将转换后的温度数值利用串口发送给 PC 机终端。
3. 使用 IAR 开发环境设计程序，利用 CC2540 的串口 0 进行数据收发通讯。

实验原理

蓝牙 4.0(CC2540)模块 LED 硬件接口：

蓝牙 4.0(CC2540)模块硬件上设计有 2 个 LED 灯，用来编程调试使用。分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时候，LED1 灯点亮，P1_1 引脚为高电平时 LED0 灯亮。



实验过程

一、串口收发数据实验步骤

- 1) 使用 USB 线连接 PC 机和移动互联网教学科研平台的 CC-debug 接口，将系统配套串口线一端连接 PC 机，一端连接移动互联网平台的串口 2(com2)上。
- 2) 启动 IAR 开发环境，新建工程，或直接使用 Exp6 实验工程。
- 3) 在 IAR 开发环境中编译、运行、调试程序。
- 4) 使用 PC 机自带的超级终端（注意：字符串必须以回车键结束或输入字符串长度超过 30 个字符，才会显示）连接串口，将超级终端设置为串口波特率 57600、8 位、无奇偶校验，无硬件流模式，当向串口终端输入数据输入回车结

束符时，将在超级终端看到串口输入的数据。

2、串口控制 LED 实验步骤

1) 使用 USB 线连接 PC 机和移动互联网教学科研平台的 CC-debug 接口，将系统配套串口线一端连接 PC 机，一端连接移动互联网平台的串口 2(com2)上。

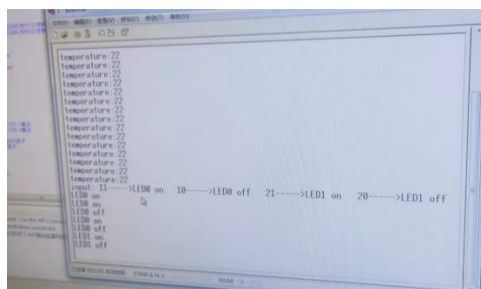
2) 启动 IAR 开发环境，新建工程，或直接使用 Exp6 实验工程。

3) 在 IAR 开发环境中编译、运行、调试程序。

4) 使用 PC 机自带的超级终端（注意：字符串必须以回车键结束或输入字符串长度超过 30 个字符，才会显示）连接串口，将超级终端设置为串口波特率 57600、8 位、无奇偶校验，无硬件流模式，当向串口终端输入数据输入回车结束符时，将在超级终端看到串口输入的数据。

实验现象

通过串口控制 LED



LED 灯变化



总结思考

本次实验我们小组一起完成了通过软件程序调用与串口来控制 LED 的变化。在实验过程中我们通过利用 CC2540 的串口 0 进行数据收发通讯并同时利用 CC2540 的串口 0 对板载 LED 灯进行控制。我们不仅学习了如何将外部变化与计算机内部检测相结合，同时也更加熟悉了上节课配置的 IAR 开发环境程序。

实验六：声响/光敏传感器实验实验

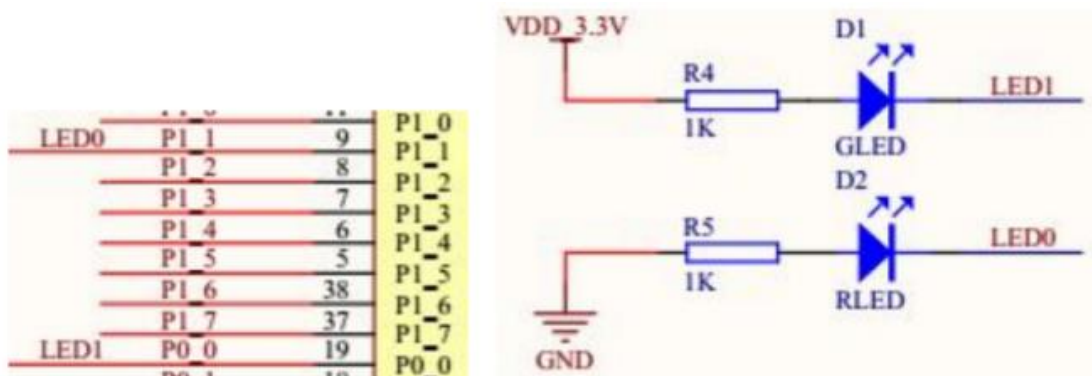
实验内容

1. 阅读移动互联网教学科研平台 蓝牙 4.0 模块硬件部分文档,熟悉蓝牙 4.0 模块硬件接口。
2. 使用 IAR 开发环境设计程序,利用 CC2540 的 IO 中断来监测声响/光敏传感器的状态。

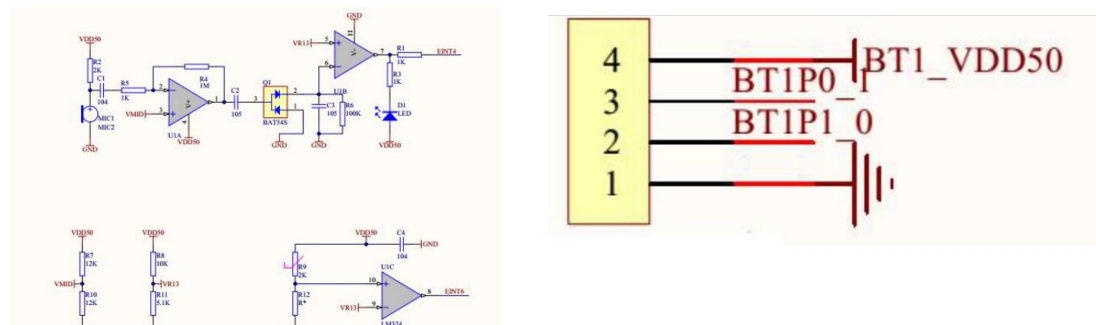
实验原理

1、蓝牙 4.0(CC2540)模块 LED 硬件接口

蓝牙 4.0(CC2540)模块硬件上设计有 2 个 LED 灯,用来编程调试使用分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时,LED1 灯点亮,P1_1 引脚为高电平时 LED0 灯亮。



2、声响/光敏传感器与 SENSOR IO/INT 接口



系统配套的声响/光敏传感器，与蓝牙 4.0 模块的 IO/INT 排针相连，这样我们可以知道，光敏传感器模块的信号线与蓝牙 4.0 模块的 P1_0 IO 引脚相连，声响传感器与 P0_1 IO 引脚相连。因此我们需要在代码中将该相应引脚配置成中断输入模式，来监测声响/光敏传感器状态。

3. CC2540 IO 相关寄存器

对于 CC2540 处理器的 P1 IO 端口相关寄存器，其中 P1DIR 为 IO 方向寄存器，P1INP 为 P1 端口输入配置寄存,P1CTL 用来控制 P1 端口中断使能和中断触发模式寄存器。P0 和 P1 端口中断控制和配置相关寄存器，其中 IEN0 为系统中断总开关控制寄存器，IEN1 用来控制 P0 端口中断使能，IEN2 用来控制 P1 端口中断使能，P0IFG 和 P1IFG 寄存器用来监测 P0 和 P1 端口中断状态的标志寄存器。

P0IEN (0xAB) – Port 0 Interrupt Mask

| Bit | Name | Reset | R/W | Description |
|-----|-------------|-------|-----|---|
| 7:0 | P0_[7:0]IEN | 0x00 | R/W | Port P0.7 to P0.0 interrupt enable 0: Interrupts are disabled. 1: Interrupts are enabled. |

P1IEN (0x8D) – Port 1 Interrupt Mask

| Bit | Name | Reset | R/W | Description |
|-----|-------------|-------|-----|---|
| 7:0 | P1_[7:0]IEN | 0x00 | R/W | Port P1.7 to P1.0 interrupt enable 0: Interrupts are disabled. 1: Interrupts are enabled. |

P0IFG (0x89) – Port 0 Interrupt Status Flag

| Bit | Name | Reset | R/W | Description |
|-----|-----------|-------|------|---|
| 7:0 | P0IF[7:0] | 0x00 | R/W0 | Port 0, inputs 7 to 0 interrupt status flags. When an input port pin has an interrupt request pending, the corresponding flag bit is set. |

P1IFG (0x8A) – Port 1 Interrupt Status Flag

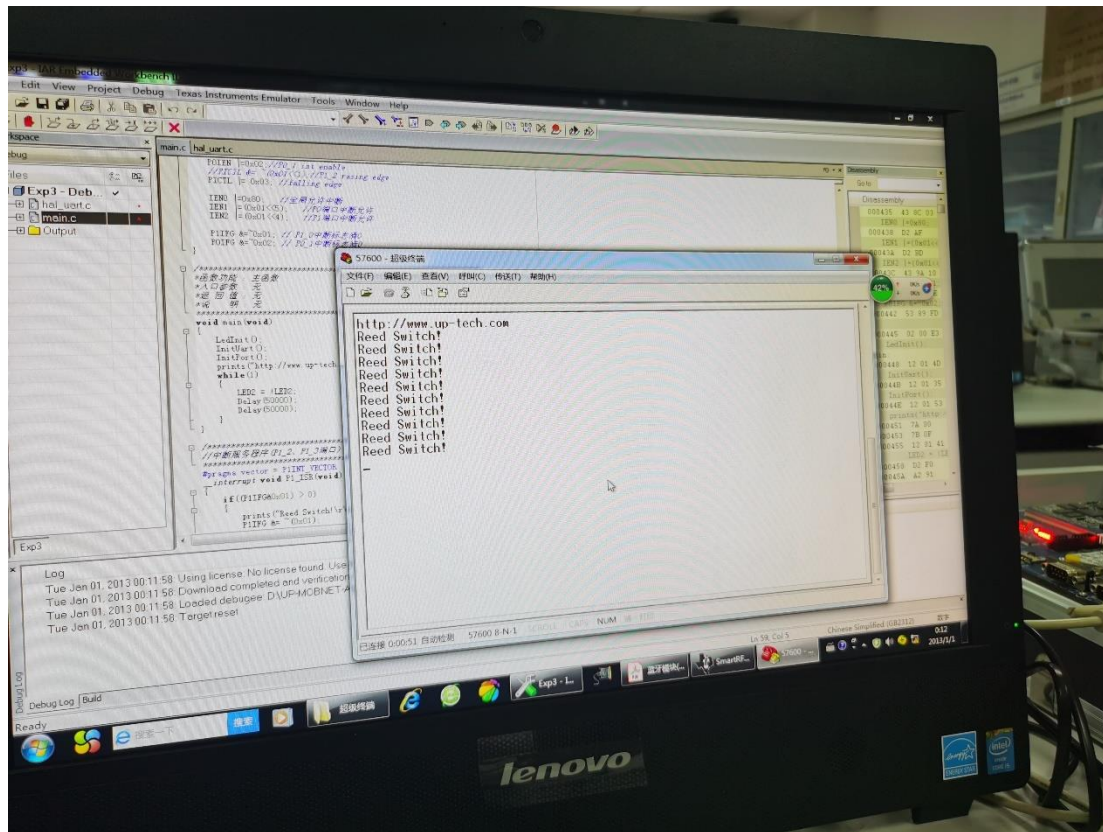
| Bit | Name | Reset | R/W | Description |
|-----|-----------|-------|------|---|
| 7:0 | P1IF[7:0] | 0x00 | R/W0 | Port 1, inputs 7 to 0 interrupt status flags. When an input port pin has an interrupt request pending, the corresponding flag bit is set. |

实验过程

1. 使用 USB 线连接 PC 机和移动互联网教学科研平台的 CC-debug 接口,将系统配套串口线一端连接 PC 机，一端连接移动互联网平台的串口 2(com2)上。
2. 启动 IAR 开发环境，新建工程，将 Exp4 实验工程中代码拷贝到新建工程中。
3. 在 IAR 开发环境中编译、运行、调试程序。
4. 使用串口调试助手连接串口，将串口调试助手设置为串口波特率 57600、8 位、无奇偶校验，无硬件流模式，当有声音时,串口输出 Sound Switch,当光敏的光由强变暗时串口打印 Light Switch。

实验现象

隐藏节点实验主要是揭示无线节点传输过程中可能发生的冲突以及解决的过程。实验中需要控制的是节点之间的距离可以自由调整，RTS/CTS 控制可以启用或不启用，然后观察数据包丢失现象来获得实验结论。实验场景如下：

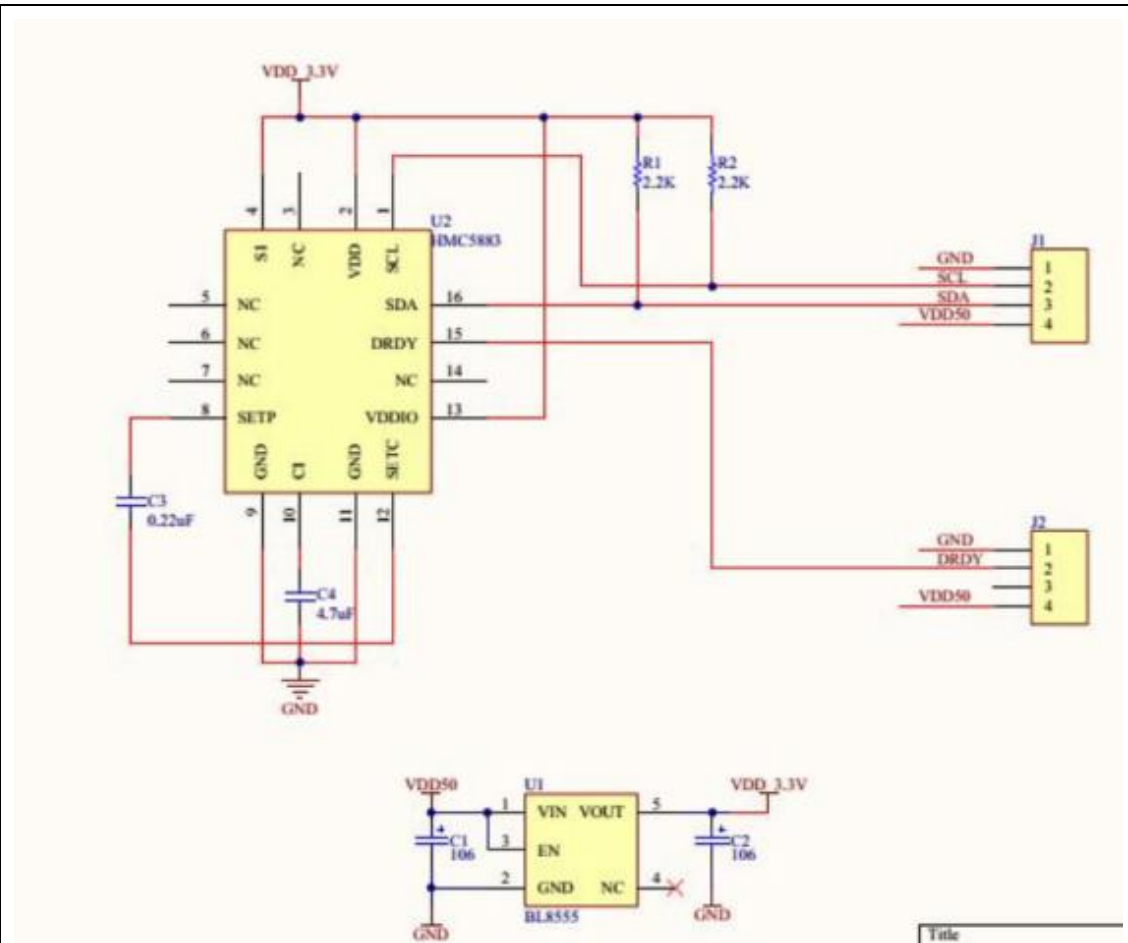


思考总结

本次实验小组一起完成了通过软件程序来监测声响/光敏传感器的状态。在实验过程中我们通过利用 CC2540 的 IO 中断来监测声响/光敏传感器的状态，最后通过电脑的输出验证了声响/光敏传感器的状态的切换，我们同时也更加熟悉了该蓝牙板块。

实验七：磁场强度传感器实验

| |
|---|
| 实验内容 |
| <div>1. 阅读移动互联网教学科研平台 蓝牙 4.0 模块硬件部分文档，熟悉蓝牙 4.0 模块硬件接口。</div> <div>2. 使用 IAR 开发环境设计程序，利用 CC2540 的 IO 口来模拟 I2C 读取三轴加速度传感器的状态。</div> <div>3. 阅读磁场强度传感器芯片文档，熟悉该传感器的使用及时序操作。</div> |
| 实验原理 |
| <div>1、蓝牙 4.0(CC2540)模块 LED 硬件接口</div> <div><p>蓝牙 4.0(CC2540)模块硬件上设计有 2 个 LED 灯，用来编程调试使用分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时，LED1 灯点亮，P1_1 引脚为高电平时 LED0 灯亮。</p><div></div></div> <div>2、磁场强度传感器模块硬件接口</div> <div><p>系统配套的三轴加速度传感器，与蓝牙 4.0 模块的 P0_1、P1_0 管脚相连，这样我们可以知道，三轴 加速度传感器模块的时钟线与蓝牙 4.0 模块的 P0_1 IO 引脚相连，数据线与 P1_0 引脚相连。因此我们需要 在代码中将相应引脚进行输入输出控制模拟该传感器时序，来读取磁场强度传感器状态。</p></div> |

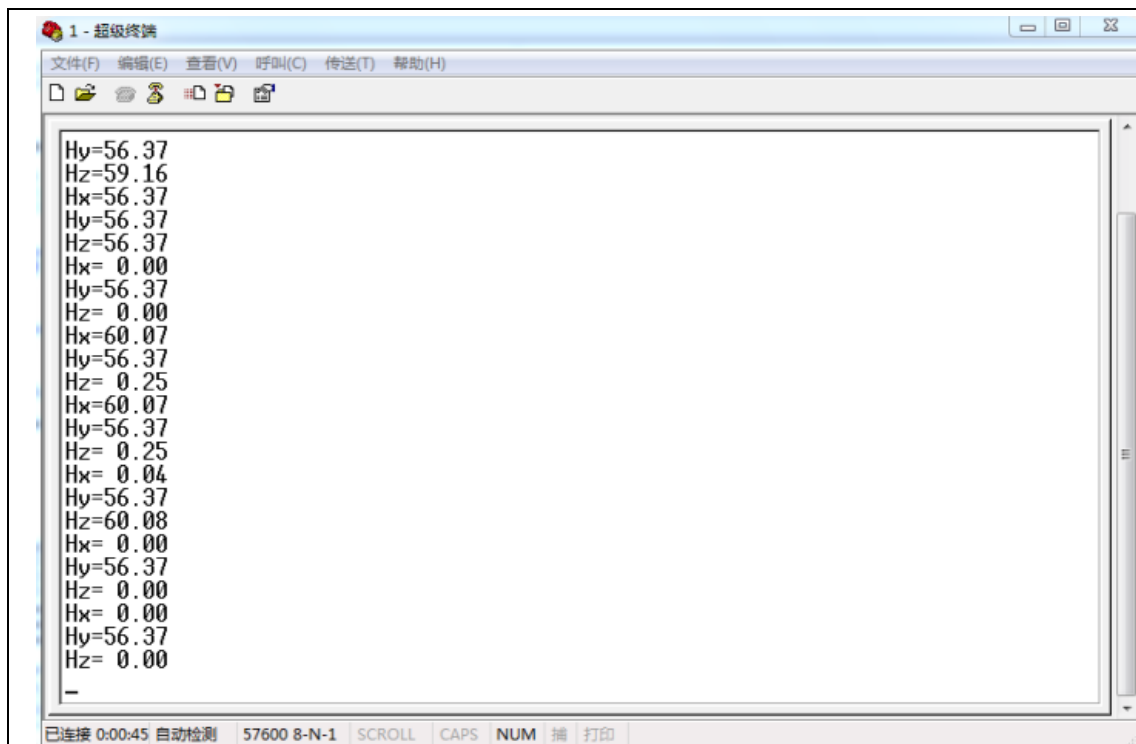


实验过程

- 1、使用 USB 线连接 PC 机和移动互联网教学科研平台的 CC-debug 接口,将系统配套串口线一端连接 PC 机，一端连接移动互联网平台的串口 2(com2)上
- 2、启动 IAR 开发环境，新建工程，打开 Sensor/Exp2 实验工程
- 3、在 IAR 开发环境中编译、运行、调试程序
- 4、使用串口调试助手连接串口，将串口调试助手设置为串口波特率 57600、8 位、无奇偶校验，无硬件流模式，当周围磁场强度发生变化时，超级终端则会打印相关数据。

实验现象

将磁铁从不同方向靠近传感器，超级终端会对应输出三轴上加速度的变化



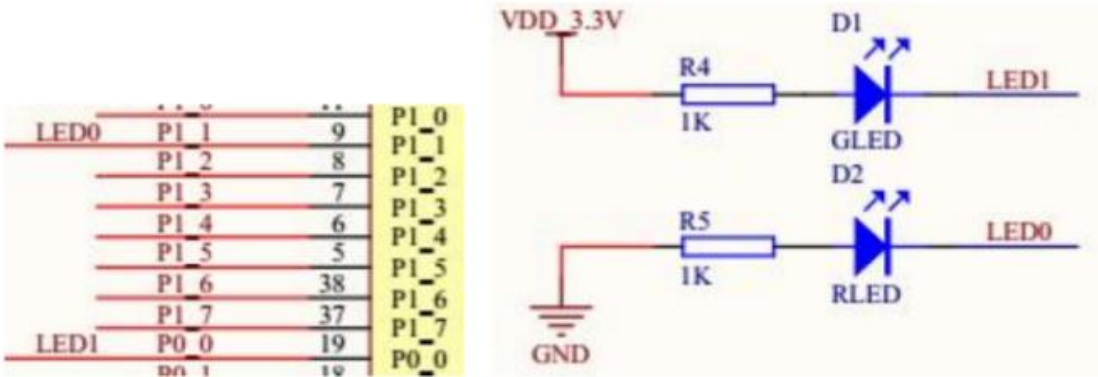
```
Hy=56.37
Hz=59.16
Hx=56.37
Hy=56.37
Hz=56.37
Hx= 0.00
Hy=56.37
Hz= 0.00
Hx=60.07
Hy=56.37
Hz= 0.25
Hx=60.07
Hy=56.37
Hz= 0.25
Hx= 0.04
Hy=56.37
Hz=60.08
Hx= 0.00
Hy=56.37
Hz= 0.00
Hx= 0.00
Hy=56.37
Hz= 0.00
-
```

已连接 0:00:45 自动检测 57600 8-N-1 SCROLL CAPS NUM 摘 打印

思考总结

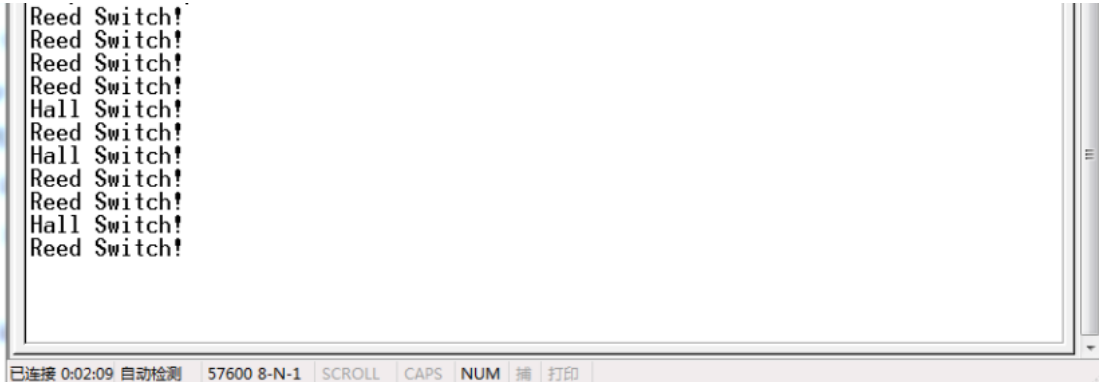
本次实验,小组完成了磁场强度传感器的实验。和以往的实验原理基本类似,本次实验也是通过利用 CC2540 的 IO 中断来检测磁场强度传感器的状态,然后通过三轴加速度的数据体现。在完成了基本试验后,我们也尝试不让传感器立即上报检测的变化数据,而是有一定延时性的输出,但是在调试代码的过程中并没有成功,不过进一步加深了对中断的理解。

实验八：霍尔开关传感器实验

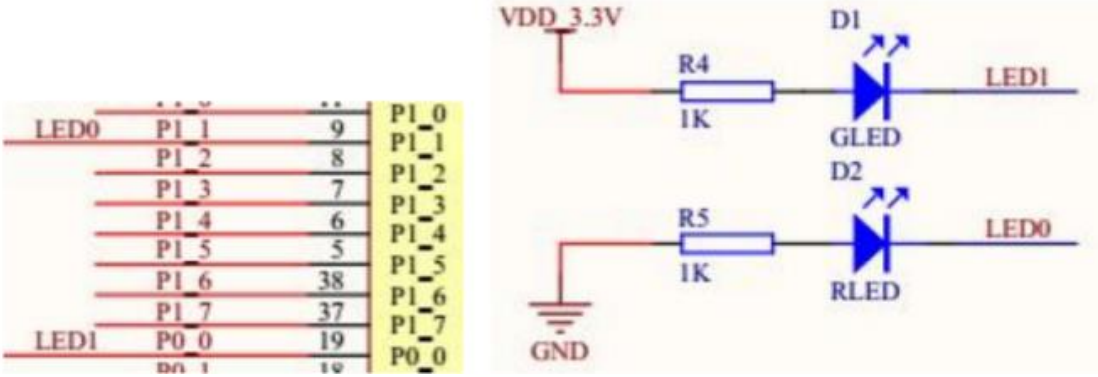
| |
|---|
| 实验内容 |
| <div>1、阅读移动互联网教学科研平台 蓝牙 4.0 模块硬件部分文档，熟悉蓝牙 4.0 模块硬件接口</div> <div>2、使用 IAR 开发环境设计程序，利用 CC2540 的 IO 中断来检测干簧门磁/霍尔开关传感器的状态</div> |
| 实验原理 |
| <div>1、蓝牙 4.0(CC2540)模块 LED 硬件接口</div> <div>蓝牙 4.0(CC2540)模块硬件上设计有 2 个 LED 灯，用来编程调试使用分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时时候，LED1 灯点亮，P1_1 引脚为高电平时 LED0 灯亮。</div> <div></div> |

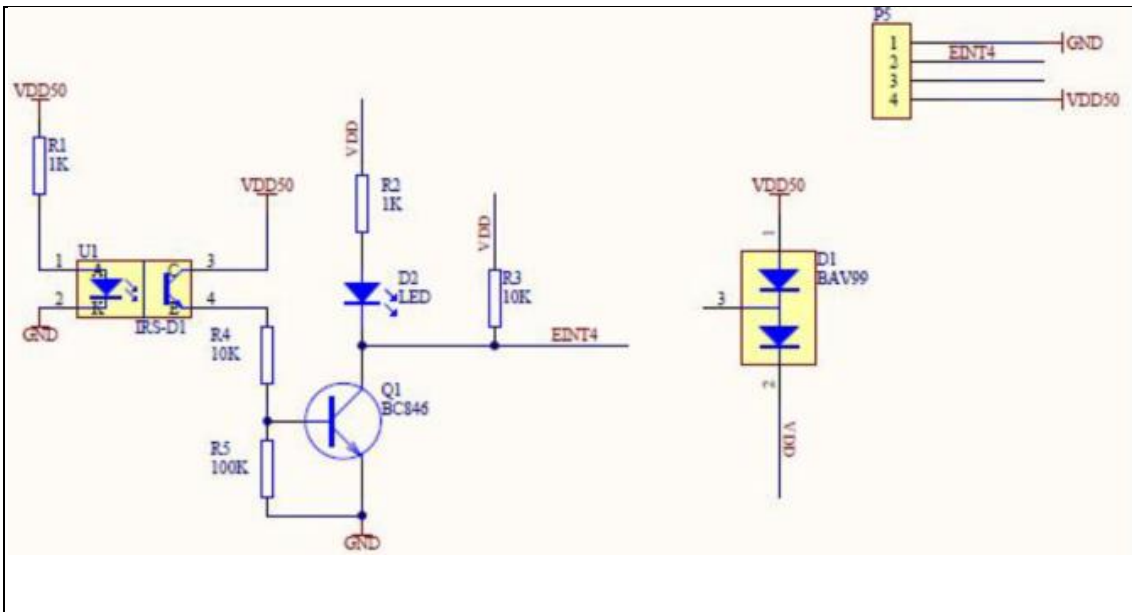
2、磁场强度传感器模块硬件接口



| |
|---|
| <p>系统配套的干簧门磁/霍尔开关传感器，与蓝牙 4.0 模块的 IO/INT 排针相连，这样我们可以知道，干簧门磁传感器模块的信号线与蓝牙 4.0 模块的 P1_0 IO 引脚相连，霍尔开关传感器与 P0_1 IO 引脚相连。因此我们需要在代码中将该相应引脚配置成中断输入模式，来监测干簧门磁/霍尔开关传感器状态。</p> |
| <p>实验过程</p> |
| <ol style="list-style-type: none"> 1. 使用 USB 线连接 PC 机和移动互联网教学科研平台的 CC-debug 接口,将系统配套串口线一端连接 PC 机，一端连接移动互联网平台的串口 2(com2)上。 2. 启动 IAR 开发环境，新建工程，打开 Sensor/Exp3 实验工程。 3. 在 IAR 开发环境中编译、运行、调试程序。 4. 使用串口调试助手连接串口，将串口调试助手设置为串口波特率 57600、8 位、无奇偶校验，无硬件流模式，当干簧门磁开关监测到磁力，向串口输出“Reed Switch”，当霍尔开关监测到磁力,向串口输出 “Hall Switch” 字符串。 |
| <p>实验现象</p> |
| <p>将磁铁靠近传感器，当干簧门磁和霍尔开关分别检测到磁力时，超级终端输出对应的变化。</p>  |
| <p>思考总结</p> |
| <p>本次实验，小组共同完成了干簧门磁/霍尔开关传感器实验。此次实验并不像其他实验一样顺利，在建立好实验工程，确定了设备连通性和代码正确性后，用磁铁靠近传感器时并不会出现预料的输出结果。后来发现我们犯了一个低级错误——选错了传感器，我们选择的传感器上根本没有干簧门/霍尔开关传感器，因此也就不可能产生对应的实验现象。在更换了传感器重新实验后，得到了预期中的实验现象。</p> |

实验九：红外对射传感器实验

| |
|--|
| 实验目的 |
| <div>1、阅读 UP-MobNet-II 型系统 ZIGBEE 模块硬件部分文档，熟悉 ZIGBEE 模块硬件接口</div> <div>2、学习红外对射传感器检测原理和使用方法</div> |
| 实验原理 |
| <div>1、蓝牙 4.0(CC2540)模块 LED 硬件接口</div> <div><p>蓝牙 4.0(CC2540)模块硬件上设计有 2 个 LED 灯，用来编程调试使用分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时，LED1 灯点亮，P1_1 引脚为高电平时 LED0 灯亮。</p><div></div></div> <div>2、红外对射传感器模块硬件接口</div> <div><p>系统配套的红外对射传感器，与 SENSOR A/D 排针相连，这样我们可以知道，红外对射传感器模块的信号线与 ZigBee 模块的 P0_0 IO 引脚相连。因此我们需要在代码中将该引脚配置成中断输入模式，来监测红外对射传感器状态。</p></div> |



代码分析

主函数源码

```
#include <ioCC2530.h>
#include "../uart/hal_uart.h"
#define uint unsigned int
#define uchar unsigned char
//定义控制灯的端口
#define LED1 P1_0
#define LED2 P1_1
/*****/
void Delay(uint n)
{
    uint i,t;
    for(i = 0;i<5;i++)
        for(t = 0;t<n;t++);
}
void LedInit(void)
{
    P1DIR |= 0x03; //设置 LED
    LED1 = 1;
    LED2 = 1;
}
/*****
//IO P1_2 中断模式初始化
*****/
void InitIrda(void)
{
    // P0_0 interrupt io initialized.
    P0DIR &= ~(0x01<<0); //P0_0 input mode
```

```

P0INP &= ~(0x01<<0); //P0_0 Pull up
P0IEN |= (0x01<<0); //P0_0 int enable
PICTL &= ~(0x01<<0); //P0 rasing edge

//PICTL |= (0x01<<0); //falling edge
IEN0 |= 0x80; //全局允许中断
IEN1 |= (0x01<<5); //P1 端口中断允许
P0IFG &= ~(0x01<<0); // P0_0 中断标志清 0
}
/*****
*函数功能：主函数
*入口参数：无
*返回值：无
*说明：无
*****/
void main(void)
{
    LedInit();
    InitUart();
    InitIrda();
    prints("http://www.up-tech.com\r\n");
    while(1)
    {
        LED2 = !LED2;
        Delay(50000);
        Delay(50000);
    }
}
/*****
//中断服务程序(P1_2 端口)
*****/
#pragma vector = P0INT_VECTOR
__interrupt void P0_ISR(void)
{
    prints("IRDS Warning!\r\n");
    if((P0IFG&0X01) == 0x01) //中断
    {
        P0IFG &= ~(0x01);
        LED1 = !LED1;
        Delay(1000);
    }
    P0IF = 0; //清中断标志
}

```

思考总结

本次实验受限于疫情通过线上的方式开展，并不能接触到实际的实验设备进行实验操作，受限较大，故而实验以原理和代码为主。源码中的核心代码就是中断服务程序，和之前实验的代码进行了对比后，发现基本的中断处理流程也是基本类似的。总之，此次实验主要是接触一种新的传感器，对中断有了进一步深入的理解。

实验十：温湿度传感器实验

实验内容

阅读 SHTX0 温湿度传感器芯片文档，熟悉该传感器的使用及时序操作。使用 IAR 开发环境设计程序，利用 CC2530 的 IO 口来监测温湿度传感器的状态。

实验原理

主要的温湿度采集由 sht11.c 的模拟时序代码完成，下面列出了相关的函数

```
void s_connectionreset(void);

char s_measure(unsigned char *p_value, unsigned char *p_checksum, unsigned
char
mode);

void calc_sht11(float *p_humidity ,float *p_temperature);
float calc_dewpoint(float h,float t);
char s_read_byte(unsigned char ack);
char s_write_byte(unsigned char value);
extern void Sht11Delay(uint n);
extern void Sht11Init(void);
extern char GetHumiAndTemp(float *humi, float *temp);
```

思考总结

本次实验受限于疫情通过线上的方式开展，并不能接触到实际的实验设备进行实验操作，受限较大。此外，给出的样例主函数代码中，不停地读取数据，如果不做 Sht11Delay 延时，会导致 IIC 设备很快被挂起，在硬件上产生故障，这是因为主 CPU 的频率远高于 IIC，故而示例的代码如果直接应用于实际会产生很大的隐患，这是需要特别注意的地方。

实验十一：TI-BLE4.0 协议栈入门实验

实验内容

学习 TI 蓝牙 4.0 协议栈软件架构，掌握 TI 蓝牙 4.0 协议栈软件开发流程。

实验原理

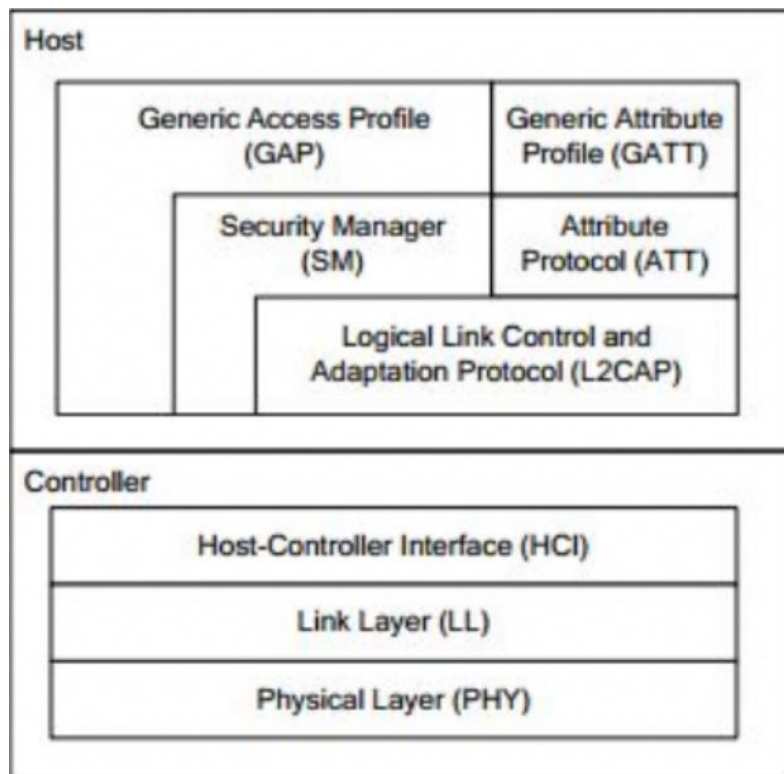
1、BLE4.0 协议栈概述

协议定义的是一系列通信标准，通信双方需要共同按照这一标准进行正常的数据收发；协议栈是协议的具体实现形式，通俗的理解为代码实现的函数库，以便开发人员调用。

BLE4.0 协议栈就是将各个层定义的协议都集合在一起，以函数的形式实现，并提供一些应用层 API，供用户调用。

2、BLE4.0 协议栈基础

蓝牙 4.0 协议栈结构图如图所示：



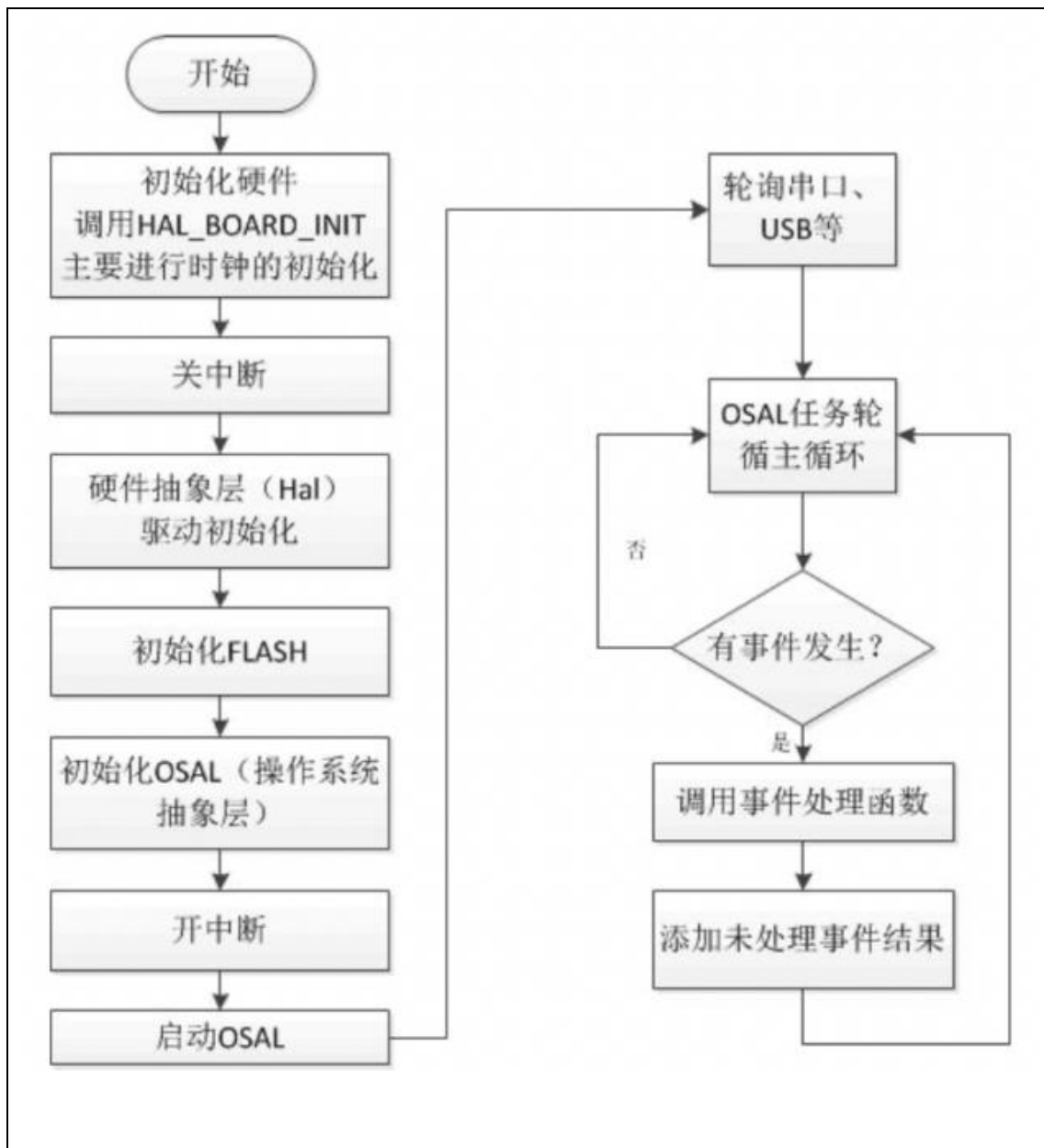
各层的作用和设置如下表所示：

| 层名 | 备注 |
|---|---|
| Physical Layer(物理层) | 1Mbps 自适应跳频的 GFSK 射频，工作于免许可证的 2.4GHz ISM(工业、科学和医疗)频段。 |
| Linker Layer(链路层) | 用于控制设备的射频状态。 |
| Host Controller Interface(主机控制接口层) | 为主机和控制器之间提供标准通信接口。可以提供软件 API 或硬件接口，如 UART、SPI、USB。 |
| Logical Link Control and Adaptation Protocol(逻辑链路控制及自适应协议层) | 为上层提供数据封装服务，允许逻辑上的点对点数据通信。 |
| Security Manager(安全管理层) | 定义了配对和密钥分配方式，并为协议栈其他层和另一个设备之间的安全连接和数据交换提供服务。 |
| Attribute Protocol(属性协议层) | 允许设备向另一个设备展示一块特定的数据，称之为“属性”。 |
| Generic Attribute profile(通用属性配置文件层) | 定义了使用 ATT 的服务框架。 |

通过协议栈，每一层的实体能够根据预定的格式准确的提取需要在本层处理的数据信息，最终用户应用程序得到最终的数据信息并进行处理。

3、BLE4.0 软件初始化流程

BLE4.0 初始化的主要工作流程，大致分为系统启动，驱动初始化，OSAL 初始化和启动，进入任务轮循几个阶段。



思考总结

通过本实验主要了解了了 TI-BLE4.0 协议栈的基础知识，加深了对蓝牙 4.0 协议栈结构、各层的作用和设置、BLE4.0 软件初始化流程这三方面内容的理解。

实验十二：基于 BLE4.0 的使能从机 notify 实验

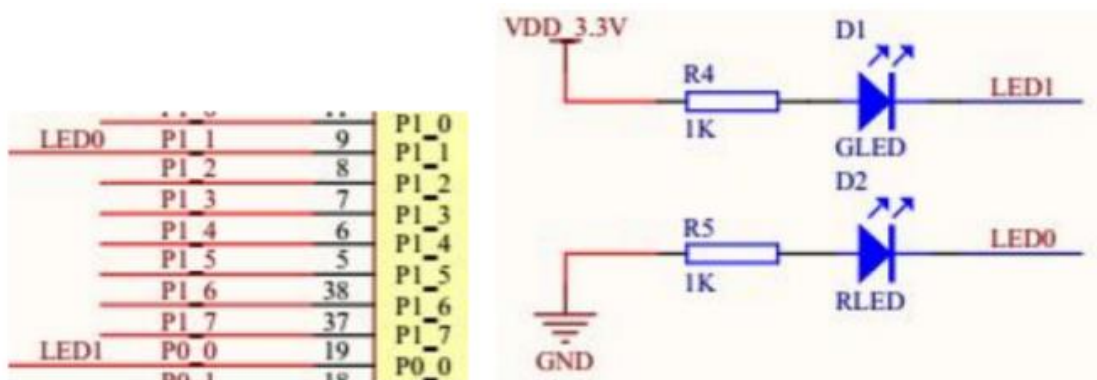
实验内容

学习 TI BLE4.0 协议栈内容，掌握 CC2400 模块无线组网原理及过程。

实验原理

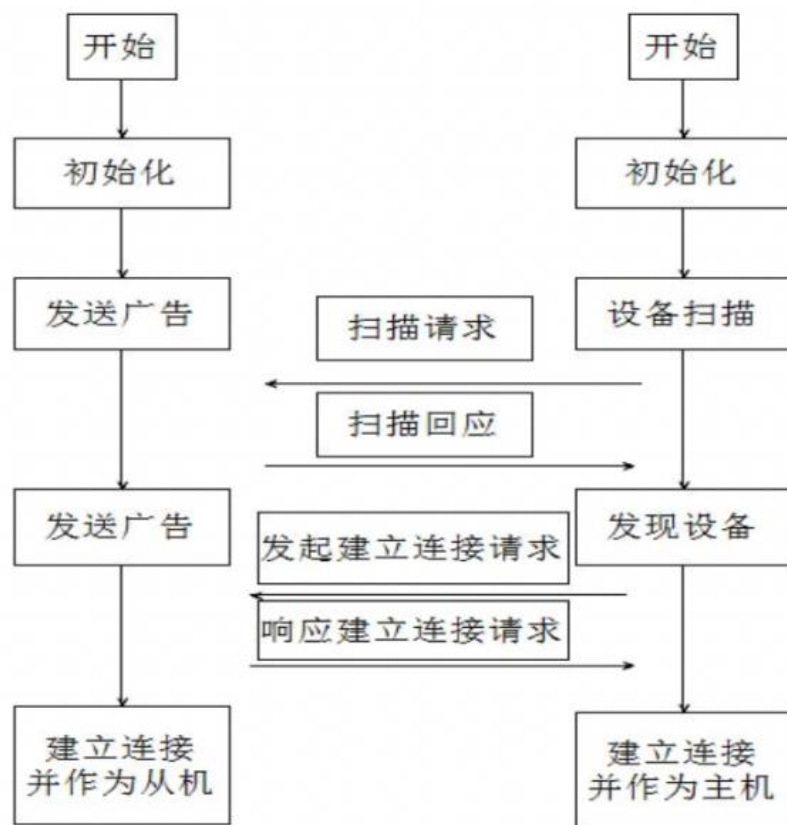
1、蓝牙 4.0(CC2540)模块 LED 硬件接口

蓝牙 4.0(CC2540)模块硬件上设计有 2 个 LED 灯，用来编程调试使用分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时，LED1 灯点亮，P1_1 引脚为高电平时 LED0 灯亮。

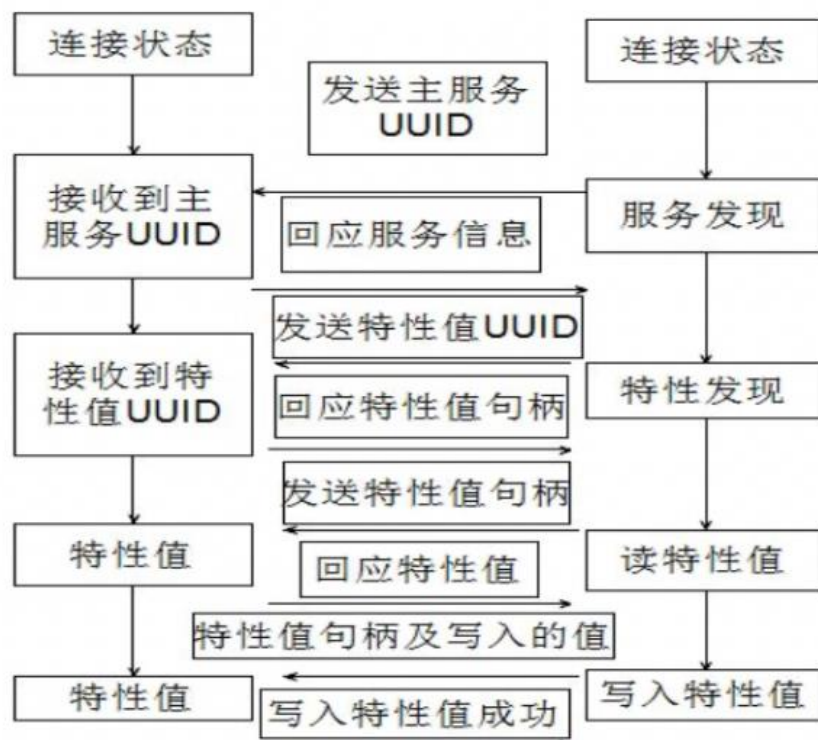


2、流程图：集中器发起建立连接请求，外部设备响应后，两设备进入连接状态；集中器通过特定的 UUID 进行 GATT 数据服务的发现；发现 GATT 数据服务后，集中器发送要进行数据操作的“特性”的 UUID，节点设备将这个“特性”的句柄返回给集中器，句柄特性值在属性表中的地址；集中器通过句柄并进行应用数据的读取和写入操作。

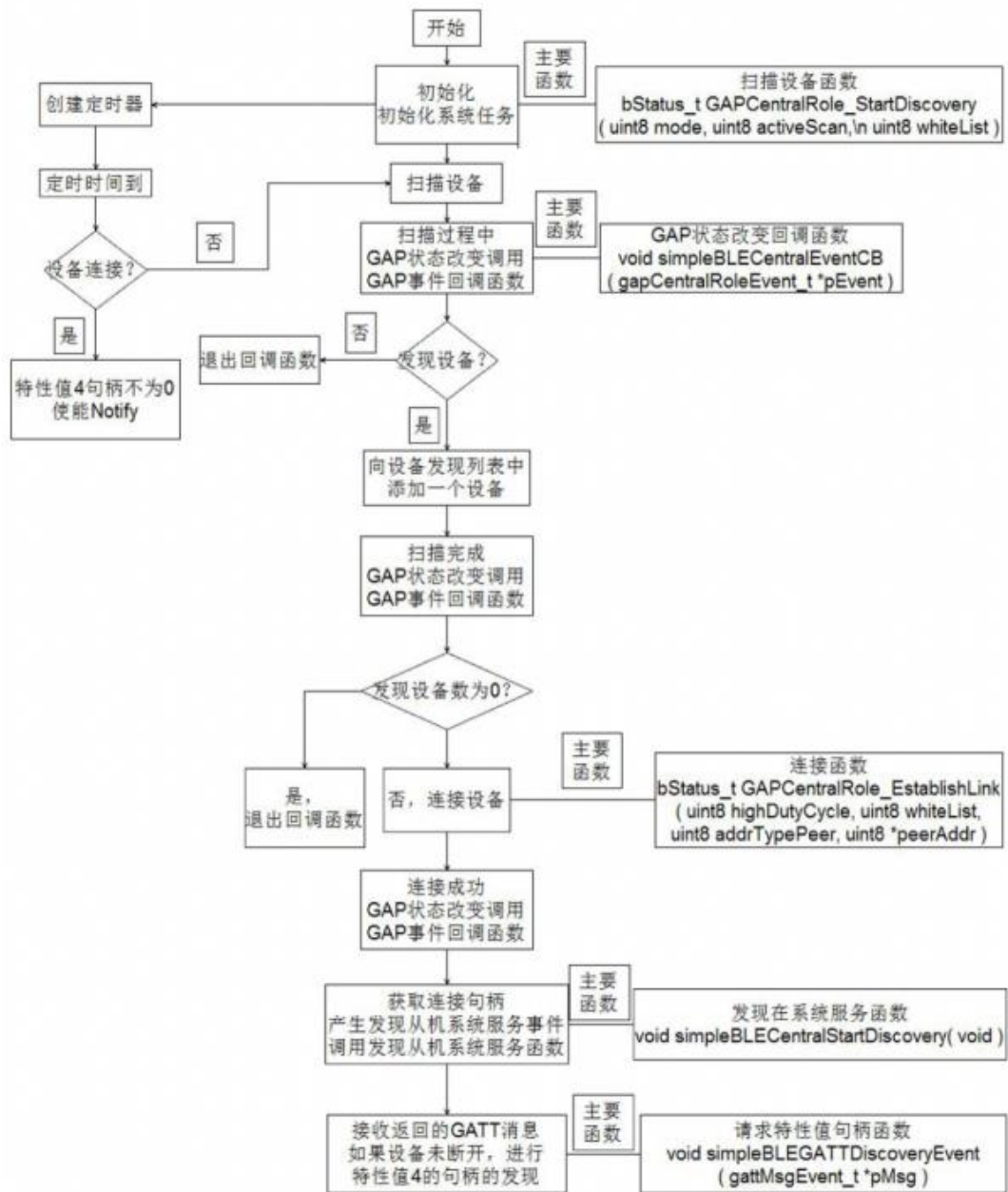
1) 建立连接流程图



2) 传输数据流程图



3) 主要函数说明图:



代码分析

1. Central 关键代码:

```

static void simpleBLECentralEventCB( gapCentralRoleEvent_t *pEvent )
{
    /* @kzkuan */
    uint8  addrType;
    uint8  *peerAddr;
    switch ( pEvent->gap.opcode )
    {
        case GAP_DEVICE_INIT_DONE_EVENT:
    
```

```

{
}
break;
case GAP_DEVICE_INFO_EVENT:
{
    /* if filtering device discovery results based on service UUID
    */
    /* 扫描时指定的 GAP 服务 UUID 来过滤 UUID 不符合的广告设备 if
    ( DEFAULT_DEV_DISC_BY_SVC_UUID == TRUE ) */
    {
        if ( simpleBLEFindSvcUuid( SIMPLEPROFILE_SERV_UUID,
                                   pEvent->deviceInfo.pEvtData,
                                   pEvent->deviceInfo.dataLen ) )
        {
            simpleBLEAddDeviceInfo( pEvent->deviceInfo.addr,
                                   pEvent->deviceInfo.addrType ); /* 向设备发现
列表添加一个设备 */
        }
    }
    break;
case GAP_DEVICE_DISCOVERY_EVENT:
{
    /* 设备扫描完成 GAP 状态变为
    此项 */
    /* discovery complete */
    simpleBLEScanning = FALSE;
    /* if not filtering device discovery results based on service UUID if
    ( DEFAULT_DEV_DISC_BY_SVC_UUID == FALSE ) */
    {
        /* Copy results */
        simpleBLEScanRes = pEvent->discCmpl.numDevs;
        osal_memcpy( simpleBLEDevList, pEvent->discCmpl.pDevList,
                     (sizeof(gapDevRec_t) * pEvent->discCmpl.numDevs) );
    }
    if ( simpleBLEScanRes >
0 )
        /* 发现设备 */
        {
            for ( simpleBLEScanIdx =
0; simpleBLEScanIdx < simpleBLEScanRes;
simpleBLEScanIdx++ )
            {
                addrType =
simpleBLEDevList[simpleBLEScanIdx].addrType;
                peerAddr = simpleBLEDevList[simpleBLEScanIdx].addr;

```



```

        GAPCentralRole_EstablishLink( DEFAULT_LINK_HIGH_DUTY_CYCLE,
        DEFAULT_LINK_WHITE_LIST, addrType, peerAddr );    /* 发起连接请求 */
    }
    simpleBLEScanRes = 0;
}
/* initialize scan index to last device */
simpleBLEScanIdx = simpleBLEScanRes;
} break;
case GAP_LINK_ESTABLISHED_EVENT:
{
    /* 设备已连接成功 */
    connecting = 0;
    if ( pEvent->gap.hdr.status == SUCCESS )
    {
        simpleBLEState =
        BLE_STATE_CONNECTED;

        /* 连接状态 */
        柄
        simpleBLEConnHandle =
        pEvent->linkCmpl.connectionHandle;
        /* 连接句 simpleBLEProcedureInProgress = TRUE; */
        /* If service discovery not performed initiate service discovery */
        if ( simpleBLECharHdl == 0 )
        {
            osal_start_timerEx( simpleBLETaskId,
            START_DISCOVERY_EVT,
            DEFAULT_SVC_DISCOVERY_DELAY );    /* 启动发现 GATT 服务事件 */
        }
    }
    }else {
        simpleBLEState = BLE_STATE_IDLE;
        simpleBLEConnHandle = GAP_CONNHANDLE_INIT;
        simpleBLERssi = FALSE;
        simpleBLEDiscState = BLE_DISC_STATE_IDLE;
    }
} break;
case GAP_LINK_TERMINATED_EVENT: { /* 设备断开连接进入该选项 */
    connecting = 1;
    simpleBLEState = BLE_STATE_IDLE;
    simpleBLEConnHandle = GAP_CONNHANDLE_INIT;
    simpleBLERssi = FALSE;
    simpleBLEDiscState = BLE_DISC_STATE_IDLE;
    simpleBLECharHdl = 0;
}

```

```

        simpleBLEProcedureInProgress    = FALSE;
    } break;
    case GAP_LINK_PARAM_UPDATE_EVENT:
    {
    } break;
    default:
        break;
    }
}

uint16 SimpleBLECentral_ProcessEvent( uint8 task_id, uint16 events )
{
    VOID task_id; /* OSAL required parameter that isn't used in this
function */
    if ( events & SYS_EVENT_MSG )
    {
        uint8 *pMsg;
        if ( (pMsg = osal_msg_receive( simpleBLETaskId ) ) != NULL )
        {
            simpleBLECentral_ProcessOSALMsg( (osal_event_hdr_t *)
pMsg );
            /* Release the OSAL message */
            VOID osal_msg_deallocate( pMsg );
        }
        /* return unprocessed events */
        return(events ^ SYS_EVENT_MSG);
    }
    if ( events & START_DEVICE_EVT )
    {
        /* Start the Device */
        VOID GAPCentralRole_StartDevice( (gapCentralRoleCB_t *)
&simpleBLERoleCB );

        /* Register with bond manager after starting device */
        GAPBondMgr_Register( (gapBondCBs_t *) &simpleBLEBondCB );
        /*下面的代码是注册一个定时器 */
        osal_start_timerEx( simpleBLETaskId, SBP_PERIODIC_EVT,
SBP_PERIODIC_EVT_PERIOD );
        return(events ^ START_DEVICE_EVT);
    }
    if ( events & SBP_PERIODIC_EVT )
    {
        /* 定时时间到重置定时器 */
        if ( SBP_PERIODIC_EVT_PERIOD )
        {

```

```

       osal_start_timerEx( simpleBLETaskId, SBP_PERIODIC_EVT,
                           SBP_PERIODIC_EVT_PERIOD );
    }
    countTime++;
    if ( connecting )
    {
        LED_RED = ~LED_RED;
    }else{
        LED_RED = 1;
    }
    if ( Sending )
    {
        LED_GREEN  = 0;
        Sending     = 0;
    }else{
        LED_GREEN = 1;
    }
    if ( countTime == 30 )
    {
        countTime = 0;
    }
/*
 * 定时 3S 钟调用定时器处理函数 performPeriodicTask();
 * osal_set_event( simpleBLETaskId, START_RESEARCH_EVT );
 */
    }
    return(events ^ SBP_PERIODIC_EVT);
}
if ( events & START_DISCOVERY_EVT )
{
/* 处理发现 GATT 服务事件，调用如下函数 simpleBLECentralStartDiscovery( );
*/
    return(events ^ START_DISCOVERY_EVT);
}
if ( events & START_RESEARCH_EVT )
{
/* 扫描设备 */
    GAPCentralRole_StartDiscovery( DEFAULT_DISCOVERY_MODE,
                                   DEFAULT_DISCOVERY_ACTIVE_SCAN,
                                   DEFAULT_DISCOVERY_WHITE_LIST );
    return(events ^ START_RESEARCH_EVT);
}
/* Discard unknown events */
return(0);
}

```

```

static void simpleBLECentralStartDiscovery( void )
{
    uint8 uuid[ATT_BT_UUID_SIZE] =
{ LO_UINT16( SIMPLEPROFILE_SERV_UUID ),
    HI_UINT16( SIMPLEPROFILE_SERV_UUID ) };
    /* Initialize cached handles */
    simpleBLESvcStartHdl = simpleBLESvcEndHdl = simpleBLECharHdl = 0;
    simpleBLEDiscState = BLE_DISC_STATE_SVC;          /* 设置查找状态标
志为系统服务 */
    /* Discovery simple BLE service */
    GATT_DiscPrimaryServiceByUUID( simpleBLEConnHandle,
                                    uuid,
                                    ATT_BT_UUID_SIZE,
                                    )
    simpleBLETaskId );          /* 查找系统服务 */

static void simpleBLECentral_ProcessOSALMsg( osal_event_hdr_t *pMsg )
{
    switch ( pMsg->event )
    {
        case KEY_CHANGE:
            simpleBLECentral_HandleKeys( ( (keyChange_t *) pMsg)->state,
            ( (keyChange_t
                                    *) pMsg)->keys );

            break;
        case GATT_MSG_EVENT:
            simpleBLECentralProcessGATTMsg( (gattMsgEvent_t *) pMsg ); /*
GATT 消息处理函数 break; */
    }
}

static void simpleBLECentralProcessGATTMsg( gattMsgEvent_t *pMsg )
{
    if ( simpleBLEState != BLE_STATE_CONNECTED )
    {
        /*
        * In case a GATT message came after a connection has dropped,
        * ignore the message
        */
        return;
    }
    if ( (pMsg->method == ATT_READ_RSP) ||
        ( (pMsg->method == ATT_ERROR_RSP) &&

```

```

        (pMsg->msg.errorRsp.reqOpcode == ATT_READ_REQ) ) )
    {
        if ( pMsg->method == ATT_ERROR_RSP )
        {
            uint8 status = pMsg->msg.errorRsp.errCode;
            LCD_WRITE_STRING_VALUE( "Read Error", status, 10,
HAL_LCD_LINE_1 );
        }else {
/* After a successful read, display the read value uint8 valueRead =
pMsg->msg.readRsp.value[0]; */
            LCD_WRITE_STRING_VALUE( "Read rsp:", valueRead, 10,
HAL_LCD_LINE_1 );
        }
        simpleBLEProcedureInProgress = FALSE;
    }else if ( (pMsg->method == ATT_WRITE_RSP) ||
        ( (pMsg->method == ATT_ERROR_RSP) &&
        (pMsg->msg.errorRsp.reqOpcode == ATT_WRITE_REQ) ) ) )
    {
        if ( pMsg->method == ATT_ERROR_RSP == ATT_ERROR_RSP )
        {
            uint8 status = pMsg->msg.errorRsp.errCode;
            LCD_WRITE_STRING_VALUE( "Write Error", status, 10,
HAL_LCD_LINE_1 );
        }else {
/*
 * After a succesful write, display the value that was written and
increment value
 * LCD_WRITE_STRING_VALUE( "Write sent:", simpleBLECharVal++, 10,
HAL_LCD_LINE_1 );
 */
        }
        simpleBLEProcedureInProgress = FALSE;
    }else if ( pMsg->method == ATT_HANDLE_VALUE_NOTI ) /* 接
收通知发送过来的数据 { */
        uint8 value = pMsg->msg.handleValueNoti.value[0]; /* 将特性
值 4 的值赋值给 value */
        value += 0x30;
        HalUARTWrite( HAL_UART_PORT_0, "Notify Value:", 13 ); /*
串口打印特性值 4 的值 HalUARTWrite(HAL_UART_PORT_0,&value,1); */
        HalUARTWrite( HAL_UART_PORT_0, "\n", 1 );
    }else if ( simpleBLEDiscState != BLE_DISC_STATE_IDLE )
    {
        simpleBLEGATTDiscoveryEvent( pMsg ); /* 调
用 GATT 发现事件函数，用于发现特性值句柄 */
    }

```

```

}
}

static void simpleBLEGATTDiscoveryEvent( gattMsgEvent_t *pMsg )
{
    attReadByTypeReq_t req;
    if ( simpleBLEDiscState == BLE_DISC_STATE_SVC )
    {
        /*
        * Service found, store handles
        * 务发现存储消息句柄
        */
        if ( pMsg->method == ATT_FIND_BY_TYPE_VALUE_RSP &&
            pMsg->msg.findByTypeValueRsp.numInfo > 0 )
        {
            /* 存储起始和结束句柄 */
            simpleBLESvcStartHdl =
pMsg->msg.findByTypeValueRsp.handlesInfo[0].handle;
            simpleBLESvcEndHdl =
pMsg->msg.findByTypeValueRsp.handlesInfo[0].grpEndHandle;
        }
        /* If procedure complete */
        if ( (pMsg->method == ATT_FIND_BY_TYPE_VALUE_RSP &&
            pMsg->hdr.status == bleProcedureComplete) ||
            (pMsg->method == ATT_ERROR_RSP) )
        {
            if ( simpleBLESvcStartHdl != 0 )
            {
                /* Discover characteristic */
                simpleBLEDiscState = BLE_DISC_STATE_CHAR;
                req.startHandle = simpleBLESvcStartHdl;
                req.endHandle = simpleBLESvcEndHdl;
                req.type.len = ATT_BT_UUID_SIZE;
                req.type.uuid[0] =
LO_UINT16( SIMPLEPROFILE_CHAR4_UUID ); /* 特性值 4 的 UUID
req.type.uuid[1] = HI_UINT16(SIMPLEPROFILE_CHAR4_UUID); */
                /* 查找特性值 4 的消息句柄 */
                GATT_ReadUsingCharUUID( simpleBLEConnHandle, &req,
simpleBLETaskId );
            }
        }
        }else if ( simpleBLEDiscState == BLE_DISC_STATE_CHAR )
        {
            /* Characteristic found, store handle */

```

```

        if ( pMsg->method == ATT_READ_BY_TYPE_RSP &&
            pMsg->msg.readByTypeRsp.numPairs > 0 )
        {
/* 找到特性值 4 的句柄 */
            simpleBLECharHdl =
BUILD_UINT16( pMsg->msg.readByTypeRsp.dataList[0],
                pMsg->msg.readByTypeRsp.dataList[1] );
            LCD_WRITE_STRING( "Simple Svc Found", HAL_LCD_LINE_1 );
            simpleBLEProcedureInProgress = FALSE;
        }
        simpleBLEDiscState = BLE_DISC_STATE_IDLE;
    }
}

void performPeriodicTask( void )
{
    if ( simpleBLEState != BLE_STATE_CONNECTED )
    {
        if ( !simpleBLEScanning )
        {
            /* 如果没有连接设备且没有扫描则进行设备扫描 */
            simpleBLEScanning = TRUE;
            simpleBLEScanRes = 0;

#ifdef DEBUG
            HalUARTWrite( HAL_UART_PORT_0, "Discovering\n", 12 );
            Sending = 1;
#endif

            LCD_WRITE_STRING( "Discovering...", HAL_LCD_LINE_1 );
            LCD_WRITE_STRING( "", HAL_LCD_LINE_2 );
            GAPCentralRole_StartDiscovery( DEFAULT_DISCOVERY_MODE,
                                           DEFAULT_DISCOVERY_ACTIVE_SCAN,
                                           DEFAULT_DISCOVERY_WHITE_LIST );
        }
        }else if ( simpleBLEState == BLE_STATE_CONNECTED &&
            simpleBLECharHdl != 0 &&
            simpleBLEProcedureInProgress == FALSE )
        {
/* 如果已连接设备且设备的特性值 4 的句柄不为 0,则使能特性值 4 的 Notify uint8 status; */
            if ( simpleBLECharVal == 9 )
                simpleBLECharVal = 0;
/* Do a write */
            attWriteReq_t req;
            req.handle = simpleBLECharHdl + 1; /* 特性值 4 的句柄+1 req.len
= 2; */
            req.value[0] = 0x01;

```



```

        req.value[1]    = 0x00;
        req.sig         = 0;
        req.cmd         = 0;
/* 特性值写函数 */
        status = GATT_WriteCharValue( simpleBLEConnHandle, &req,
simpleBLETaskId );
        if ( status == SUCCESS )
        {
            simpleBLEProcedureInProgress = TRUE;
        }
    }
}

```

2. Peripheral 关键代码

```

static void performPeriodicTask( void )
{
    uint8  valueToCopy;
    uint8  stat;
/*
 * 以下代码是将特性值 3 的值复制到特性值 4
 * Call to retrieve the value of the third characteristic in the
profile
 */
    stat = SimpleProfile_GetParameter( SIMPLEPROFILE_CHAR3,
&valueToCopy );
    if ( stat == SUCCESS )
    {
/*
 * Call to set that value of the fourth characteristic in the profile.
Note
 * that if notifications of the fourth characteristic have been enabled
by
 * a GATT client device, then a notification will be sent every time
this
 * function is called.
 */
        SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR4, sizeof(uint8),
&valueToCopy );
    }
}

```

总结思考

从本次实验中学学习到了 TI BLE4.0 协议栈的具体内容，掌握了 CC2400 模块进行无线组网的原理及过程。

实验十三：基于 BLE4.0 协议栈的点对点通信实验

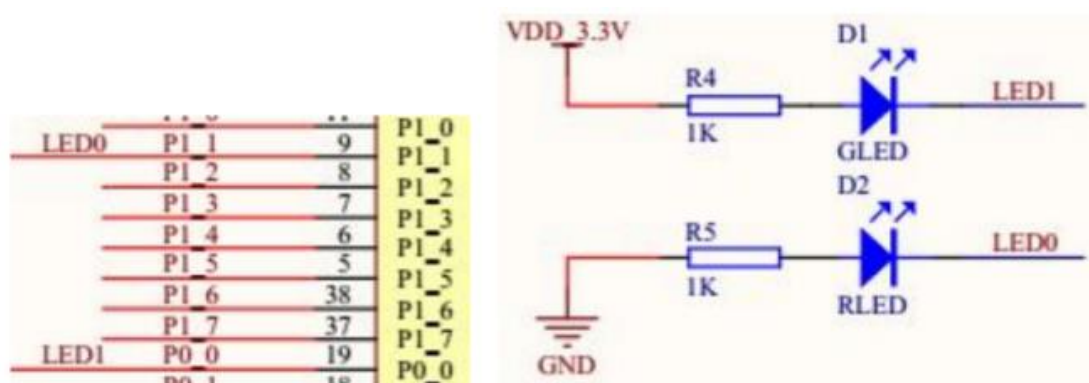
实验内容

学习 TI BLE4.0 协议栈内容，掌握 CC2540 模块无线组网原理及过程

实验原理

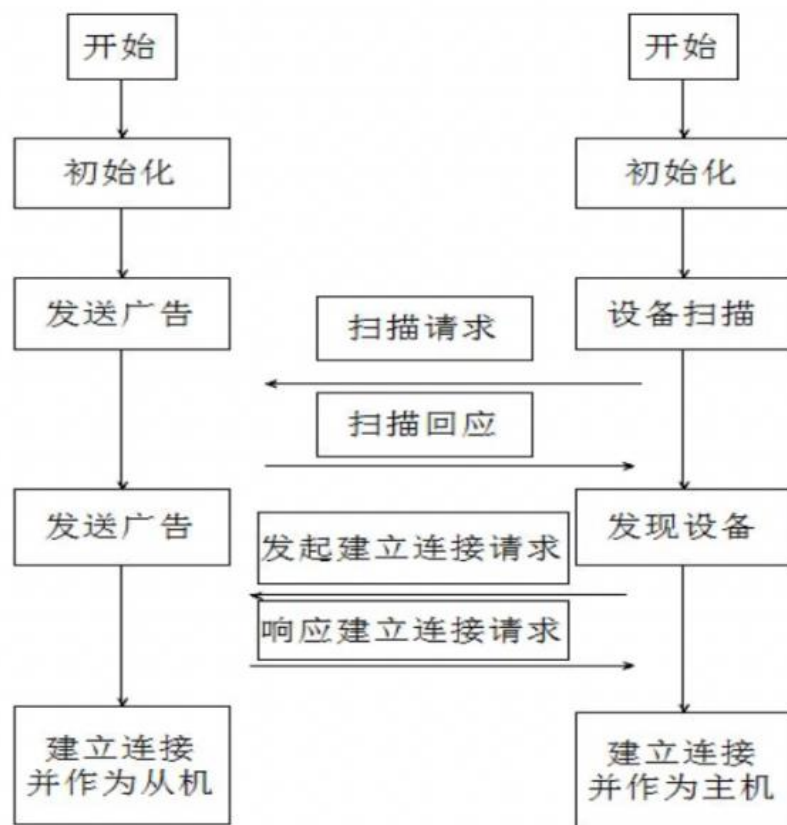
1. 蓝牙 4.0(CC2540)模块 LED 硬件接口

蓝牙 4.0(CC2540)模块硬件上设计有 2 个 LED 灯，用来编程调试使用分别连接 CC2540 的 P0_0、P1_1 两个 IO 引脚。从原理图上可以看出当 P0_0 引脚为低电平时，LED1 灯点亮，P1_1 引脚为高电平时 LED0 灯亮。

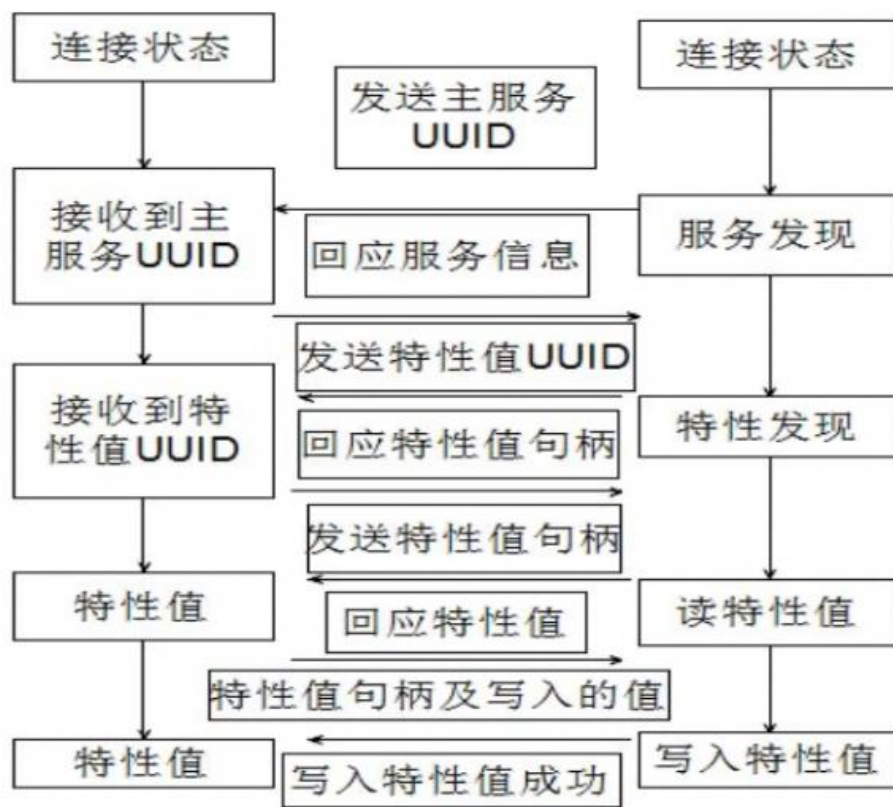


2. 流程图：集中器发起建立连接请求，外部设备响应后，两设备进入连接状态；集中器通过特定的 UUID 进行 GATT 数据服务的发现；发现 GATT 数据服务后，集中器发送要进行数据操作的“特性”的 UUID，节点设备将这个“特性”的句柄返回给集中器，句柄特性值在属性表中的地址；集中器通过句柄进行应用数据的读取和写入操作。

1) 建立连接流程图



4) 传输数据流程图



代码分析

1. Central 关键代码:

```
static void simpleBLECentralEventCB( gapCentralRoleEvent_t *pEvent )
{
    /* @kzkuan */
    uint8  addrType;
    uint8  *peerAddr;
    switch ( pEvent->gap.opcode )
    {
        case GAP_DEVICE_INIT_DONE_EVENT:
        {
        }
        break;
        case GAP_DEVICE_INFO_EVENT:
        {
            /* if filtering device discovery results based on service UUID
            */
            /* 扫描时指定的 GAP 服务 UUID 来过滤 UUID 不符合的广告设备 if
            ( DEFAULT_DEV_DISC_BY_SVC_UUID == TRUE ) */
            {
                if ( simpleBLEFindSvcUuid( SIMPLEPROFILE_SERV_UUID,
                                           pEvent->deviceInfo.pEvtData,
                                           pEvent->deviceInfo.dataLen ) )
                {
                    simpleBLEAddDeviceInfo( pEvent->deviceInfo.addr,
                                           pEvent->deviceInfo.addrType ); /* 向设备发现
列表添加一个设备 */
                }
            }
            break;
            case GAP_DEVICE_DISCOVERY_EVENT:
            {
                /* 设备扫描完成 GAP 状态变为
                此项 */
                /* discovery complete */
                simpleBLEScanning = FALSE;
                /* if not filtering device discovery results based on service UUID if
                ( DEFAULT_DEV_DISC_BY_SVC_UUID == FALSE ) */
                {
                    /* Copy results */
                    simpleBLEScanRes = pEvent->discCmpl.numDevs;
                    osal_memcpy( simpleBLEDevList, pEvent->discCmpl.pDevList,
                                (sizeof(gapDevRec_t) * pEvent->discCmpl.numDevs) );
                }
            }
        }
    }
}
```

```

        if ( simpleBLEScanRes >
0 )

                /* 发现设备 */

        {
                for ( simpleBLEScanIdx =
                        0; simpleBLEScanIdx < simpleBLEScanRes;
simpleBLEScanIdx++ )
                {
                        addrType      =
simpleBLEDevList[simpleBLEScanIdx].addrType;
                        peerAddr      = simpleBLEDevList[simpleBLEScanIdx].addr;
                        GAPCentralRole_EstablishLink( DEFAULT_LINK_HIGH_DUTY_CYC
LE, DEFAULT_LINK_WHITE_LIST, addrType, peerAddr );      /* 发起连接请求
*/

                }
                simpleBLEScanRes = 0;
        }
        /* initialize scan index to last device */
        simpleBLEScanIdx = simpleBLEScanRes;
    } break;
    case GAP_LINK_ESTABLISHED_EVENT:
    {
                /* 设备已连接成功 */

                connecting = 0;
                if ( pEvent->gap.hdr.status == SUCCESS )
                {
                        simpleBLEState =
BLE_STATE_CONNECTED;

                                /* 连接状态 */

                        柄
                                simpleBLEConnHandle =
pEvent->linkCmpl.connectionHandle;
                                /* 连接句 simpleBLEProcedureInProgress = TRUE; */
/* If service discovery not performed initiate service discovery */
                                if ( simpleBLECharHdl == 0 )
                                {
                                        osal_start_timerEx( simpleBLETaskId,
START_DISCOVERY_EVT,
DEFAULT_SVC_DISCOVERY_DELAY );      /* 启动发现 GATT 服
务事件 */
                                }
                }
        } else {
                simpleBLEState      = BLE_STATE_IDLE;
                simpleBLEConnHandle = GAP_CONNHANDLE_INIT;

```

```

        simpleBLERssi      = FALSE;
        simpleBLEDiscState = BLE_DISC_STATE_IDLE;
    }
} break;
case GAP_LINK_TERMINATED_EVENT: { /* 设备断开连接进入该选项 */
    connecting      = 1;
    simpleBLEState   = BLE_STATE_IDLE;
    simpleBLEConnHandle = GAP_CONNHANDLE_INIT;
    simpleBLERssi     = FALSE;
    simpleBLEDiscState = BLE_DISC_STATE_IDLE;
    simpleBLECharHdl   = 0;
    simpleBLEProcedureInProgress = FALSE;
} break;
case GAP_LINK_PARAM_UPDATE_EVENT:
{
} break;
default:
    break;
}
}

uint16 SimpleBLECentral_ProcessEvent( uint8 task_id, uint16 events )
{
    VOID task_id; /* OSAL required parameter that isn't used in this
function */
    if ( events & SYS_EVENT_MSG )
    {
        uint8 *pMsg;
        if ( (pMsg = osal_msg_receive( simpleBLETaskId ) ) != NULL )
        {
            simpleBLECentral_ProcessOSALMsg( (osal_event_hdr_t *)
pMsg );
            /* Release the OSAL message */
            VOID osal_msg_deallocate( pMsg );
        }
        /* return unprocessed events */
        return( events ^ SYS_EVENT_MSG );
    }
    if ( events & START_DEVICE_EVT )
    {
        /* Start the Device */
        VOID GAPCentralRole_StartDevice( (gapCentralRoleCB_t *)
&simpleBLERoleCB );
    }
}

```

```

/* Register with bond manager after starting device */
    GAPBondMgr_Register( (gapBondCBs_t *) &simpleBLEBondCB );
/*下面的代码是注册一个定时器 */
    osal_start_timerEx( simpleBLETaskId, SBP_PERIODIC_EVT,
SBP_PERIODIC_EVT_PERIOD );
    return(events ^ START_DEVICE_EVT);
}
if ( events & SBP_PERIODIC_EVT )
{
/* 定时时间到重置定时器 */
    if ( SBP_PERIODIC_EVT_PERIOD )
    {
        osal_start_timerEx( simpleBLETaskId, SBP_PERIODIC_EVT,
            SBP_PERIODIC_EVT_PERIOD );
    }
    countTime++;
    if ( connecting )
    {
        LED_RED = ~LED_RED;
    }else{
        LED_RED = 1;
    }
    if ( Sending )
    {
        LED_GREEN = 0;
        Sending = 0;
    }else{
        LED_GREEN = 1;
    }
    if ( countTime == 30 )
    {
        countTime = 0;
/*
* 定时 3S 钟调用定时器处理函数 performPeriodicTask();
* osal_set_event( simpleBLETaskId, START_RESEARCH_EVT );
*/
    }
    return(events ^ SBP_PERIODIC_EVT);
}
if ( events & START_DISCOVERY_EVT )
{
/* 处理发现 GATT 服务事件，调用如下函数 simpleBLECentralStartDiscovery( );
*/
    return(events ^ START_DISCOVERY_EVT);
}

```



```

    }
    if ( events & START_RESEARCH_EVT )
    {
/* 扫描设备 */
        GAPCentralRole_StartDiscovery( DEFAULT_DISCOVERY_MODE,
                                        DEFAULT_DISCOVERY_ACTIVE_SCAN,
                                        DEFAULT_DISCOVERY_WHITE_LIST );
        return(events ^ START_RESEARCH_EVT);
    }
/* Discard unknown events */
    return(0);
}

static void simpleBLECentralStartDiscovery( void )
{
    uint8 uuid[ATT_BT_UUID_SIZE] =
    { LO_UINT16( SIMPLEPROFILE_SERV_UUID ),
      HI_UINT16( SIMPLEPROFILE_SERV_UUID ) };
/* Initialize cached handles */
    simpleBLESvcStartHdl = simpleBLESvcEndHdl = simpleBLECharHdl = 0;
    simpleBLEDiscState = BLE_DISC_STATE_SVC; /* 设置查找状态标志为系统服务 */
/* Discovery simple BLE service */
    GATT_DiscPrimaryServiceByUUID( simpleBLEConnHandle,
                                   uuid,
                                   ATT_BT_UUID_SIZE,
                                   simpleBLETaskId ); /* 查找系统服务 */

static void simpleBLECentral_ProcessOSALMsg(osal_event_hdr_t *pMsg )
{
    switch ( pMsg->event )
    {
    case KEY_CHANGE:
        simpleBLECentral_HandleKeys( ( (keyChange_t *) pMsg)->state,
                                     ( (keyChange_t *) pMsg)->keys );

        break;
    case GATT_MSG_EVENT:
        simpleBLECentralProcessGATTMsg( (gattMsgEvent_t *) pMsg ); /* GATT 消息处理函数 break; */
    }
}
}

```

```

static void simpleBLECentralProcessGATTMsg( gattMsgEvent_t *pMsg )
{
    if ( simpleBLEState != BLE_STATE_CONNECTED )
    {
        /*
        * In case a GATT message came after a connection has dropped,
        * ignore the message
        */
        return;
    }
    if ( (pMsg->method == ATT_READ_RSP) ||
        ( (pMsg->method == ATT_ERROR_RSP) &&
          (pMsg->msg.errorRsp.reqOpcode == ATT_READ_REQ) ) )
    {
        if ( pMsg->method == ATT_ERROR_RSP )
        {
            uint8 status = pMsg->msg.errorRsp.errCode;
            LCD_WRITE_STRING_VALUE( "Read Error", status, 10,
HAL_LCD_LINE_1 );
        }else {
            /* After a successful read, display the read value uint8 valueRead =
            pMsg->msg.readRsp.value[0]; */
            LCD_WRITE_STRING_VALUE( "Read rsp:", valueRead, 10,
HAL_LCD_LINE_1 );
        }
        simpleBLEProcedureInProgress = FALSE;
    }else if ( (pMsg->method == ATT_WRITE_RSP) ||
        ( (pMsg->method == ATT_ERROR_RSP) &&
          (pMsg->msg.errorRsp.reqOpcode == ATT_WRITE_REQ) ) )
    {
        if ( pMsg->method == ATT_ERROR_RSP == ATT_ERROR_RSP )
        {
            uint8 status = pMsg->msg.errorRsp.errCode;
            LCD_WRITE_STRING_VALUE( "Write Error", status, 10,
HAL_LCD_LINE_1 );
        }else {
            /*
            * After a succesful write, display the value that was written and
            increment value
            * LCD_WRITE_STRING_VALUE( "Write sent:", simpleBLECharVal++, 10,
            HAL_LCD_LINE_1 );
            */
        }
        simpleBLEProcedureInProgress = FALSE;
    }
}

```

```

    }else if ( pMsg->method == ATT_HANDLE_VALUE_NOTI )           /* 接
收通知发送过来的数据 { */
        uint8 value = pMsg->msg.handleValueNoti.value[0];       /* 将特性
值 4 的值赋值给 value */
        value += 0x30;
        HalUARTWrite( HAL_UART_PORT_0, "Notify Value:", 13 );   /*
串口打印特性值 4 的值 HalUARTWrite(HAL_UART_PORT_0,&value,1); */
        HalUARTWrite( HAL_UART_PORT_0, "\n", 1 );
    }else if ( simpleBLEDiscState != BLE_DISC_STATE_IDLE )
    {
        simpleBLEGATTDiscoveryEvent( pMsg );                     /* 调
用 GATT 发现事件函数，用于发现特性值句柄 */
    }
}

static void simpleBLEGATTDiscoveryEvent( gattMsgEvent_t *pMsg )
{
    attReadByTypeReq_t req;
    if ( simpleBLEDiscState == BLE_DISC_STATE_SVC )
    {
        /*
        * Service found, store handles
        * 务发现存储消息句柄
        */
        if ( pMsg->method == ATT_FIND_BY_TYPE_VALUE_RSP &&
            pMsg->msg.findByTypeValueRsp.numInfo > 0 )
        {
            /* 存储起始和结束句柄 */
            simpleBLESvcStartHdl =
pMsg->msg.findByTypeValueRsp.handlesInfo[0].handle;
            simpleBLESvcEndHdl =
pMsg->msg.findByTypeValueRsp.handlesInfo[0].grpEndHandle;
        }
        /* If procedure complete */
        if ( (pMsg->method == ATT_FIND_BY_TYPE_VALUE_RSP &&
            pMsg->hdr.status == bleProcedureComplete) ||
            (pMsg->method == ATT_ERROR_RSP) )
        {
            if ( simpleBLESvcStartHdl != 0 )
            {
                /* Discover characteristic */
                simpleBLEDiscState = BLE_DISC_STATE_CHAR;
                req.startHandle = simpleBLESvcStartHdl;
                req.endHandle = simpleBLESvcEndHdl;
            }
        }
    }
}

```

```

        req.type.len      = ATT_BT_UUID_SIZE;
        req.type.uuid[0]  =
LO_UINT16( SIMPLEPROFILE_CHAR4_UUID ); /* 特性值 4 的 UUID
req.type.uuid[1] = HI_UINT16(SIMPLEPROFILE_CHAR4_UUID); */
/* 查找特性值 4 的消息句柄 */
        GATT_ReadUsingCharUUID( simpleBLEConnHandle, &req,
simpleBLETaskId );
    }
}
} else if ( simpleBLEDiscState == BLE_DISC_STATE_CHAR )
{
/* Characteristic found, store handle */
    if ( pMsg->method == ATT_READ_BY_TYPE_RSP &&
        pMsg->msg.readByTypeRsp.numPairs > 0 )
    {
/* 找到特性值 4 的句柄 */
        simpleBLECharHdl =
BUILD_UINT16( pMsg->msg.readByTypeRsp.dataList[0],
                pMsg->msg.readByTypeRsp.dataList[1] );
        LCD_WRITE_STRING( "Simple Svc Found", HAL_LCD_LINE_1 );
        simpleBLEProcedureInProgress = FALSE;
    }
    simpleBLEDiscState = BLE_DISC_STATE_IDLE;
}
}

void performPeriodicTask( void )
{
    if ( simpleBLEState != BLE_STATE_CONNECTED )
    {
        if ( !simpleBLEScanning )
        {
            /* 如果没有连接设备且没有扫描则进行设备扫描 */
            simpleBLEScanning = TRUE;
            simpleBLEScanRes = 0;
#ifdef DEBUG
            HalUARTWrite( HAL_UART_PORT_0, "Discovering\n", 12 );
            Sending = 1;
#endif
            LCD_WRITE_STRING( "Discovering...", HAL_LCD_LINE_1 );
            LCD_WRITE_STRING( "", HAL_LCD_LINE_2 );
            GAPCentralRole_StartDiscovery( DEFAULT_DISCOVERY_MODE,
                DEFAULT_DISCOVERY_ACTIVE_SCAN,
                DEFAULT_DISCOVERY_WHITE_LIST );
        }
    }
}

```

```

    }else if ( simpleBLEState == BLE_STATE_CONNECTED &&
               simpleBLECharHdl != 0 &&
               simpleBLEProcedureInProgress == FALSE )
    {
        /* 如果已连接设备且设备的
        特性值 4 的句柄不为 0,则使能特性值 4 的 Notify uint8 status; */
        if ( simpleBLECharVal == 9 )
            simpleBLECharVal = 0;
        /* Do a write */
        attWriteReq_t req;
        req.handle = simpleBLECharHdl + 1; /* 特性值 4 的句柄+1 req.len
        = 2; */
        req.value[0] = 0x01;
        req.value[1] = 0x00;
        req.sig = 0;
        req.cmd = 0;
        /* 特性值写函数 */
        status = GATT_WriteCharValue( simpleBLEConnHandle, &req,
        simpleBLETaskId );
        if ( status == SUCCESS )
        {
            simpleBLEProcedureInProgress = TRUE;
        }
    }
}

```

2. Peripheral 关键代码:

```

static void performPeriodicTask( void )
{
    uint8  valueToCopy;
    uint8  stat;
    /*
    * 以下代码是将特性值 3 的值复制到特性值 4
    * Call to retrieve the value of the third characteristic in the
    profile
    */
    stat = SimpleProfile_GetParameter( SIMPLEPROFILE_CHAR3,
    &valueToCopy );
    if ( stat == SUCCESS )
    {
        /*
        * Call to set that value of the fourth characteristic in the profile.
        Note
        * that if notifications of the fourth characteristic have been enabled
        by

```

```
* a GATT client device, then a notification will be sent every time
this
* function is called.
*/
    SimpleProfile_SetParameter( SIMPLEPROFILE_CHAR4, sizeof(uint8),
&valueToCopy );
    }
}
```

总结思考

通过本次实验，学习到了 TI BLE4.0 协议栈的具体内容，掌握了 CC2400 模块进行无线组网的原理及过程，在上一个实验的基础上，又学习到了如何进行点对点通信。