

# 物联网实验套件 AY-IOT KIT 使用指南

## ——基于 Energia 和 CC3200 LaunchPad

杭州艾研信息技术有限公司

2017 年 10 月

## 申明

杭州艾研信息技术有限公司保留随时对其产品进行修正、改进和完善的权利，同时也保留在不作任何通告的情况下，终止其任何一款产品的供应的权利。用户在下订单前应及时获取相关信息的最新版本，并验证这些信息是当前的和完整的。

可通过如下方式获取最新信息、技术资料和技术支持：

技术支持电话：0571-86134572

技术支持邮箱：support@hpati.com

产品&资料下载中心：<http://www.hpati.com/products/>

互动论坛：<http://www.hpati.com/bbs/forum.php>

公司地址：浙江省杭州市西湖区留和路16号新峰商务楼B402

更多资讯请添加艾研信息官方微信（搜索公众号：艾研）或扫一扫下方二维码：



## 目录

1	实验套件.....	4
1.1	概要.....	4
1.2	实验套件模块简介.....	5
2	实验环境准备.....	8
2.1	简介.....	8
2.2	硬件环境.....	8
2.2.1	CC3200 LaunchPad BoosterPack .....	8
2.2.2	CC3200 LaunchPad 连接方式 .....	9
2.3	软件环境.....	9
2.3.1	Energia 软件下载及安装.....	9
2.3.2	CC3200 LaunchPad 支持库下载安装.....	10
2.3.3	CC3200 LaunchPad 驱动下载及安装 .....	13
3	Energia .....	16
3.1	开发环境.....	16
3.2	内置函数.....	19
3.3	最小代码.....	19
4	Energia 基础例程.....	20
4.1	GPIO .....	20
4.1.1	概要.....	20
4.1.2	Lab-01 LED 闪烁.....	21
4.1.3	Lab-02 不带延时的 LED 闪烁.....	24
4.1.4	Lab-03 通过按键控制 LED.....	28
4.2	外部中断.....	32
4.2.1	概要.....	32
4.2.2	Lab-04 按键中断.....	33
4.2.3	Lab-05 延时消抖.....	36
4.3	UART.....	38
4.3.1	概要.....	38
4.3.2	Serial 库.....	38
4.3.3	Lab-06 Hello World.....	41
4.3.4	Lab-07 Echo .....	43
4.4	ADC12.....	47
4.4.1	概要.....	47
4.4.2	Lab-08 悬空端口采样 .....	48
4.4.3	Lab-09 LMT84 模块采集温度 .....	53
4.5	PWM.....	58
4.5.1	概要.....	58
4.5.2	Lab-10 Fading .....	59
4.5.3	Lab-11 高亮 LED 模块 .....	61
4.6	I2C .....	66

4.6.1	概要.....	66
4.6.2	Wire 库(I2C) .....	67
4.6.3	Lab-12 数据发送基本过程.....	70
4.6.4	Lab-13 数据读取基本过程.....	72
4.6.5	Lab-14 读取 HDC1080 模块温湿度数据.....	75
4.7	SPI.....	83
4.7.1	概要.....	83
4.7.2	SPI 库.....	83
4.7.3	SPI 数据基本发送和读取过程.....	84
4.7.4	Lab-13 读取 LDC1000 模块数据.....	87
4.8	WiFi .....	97
4.8.1	概要.....	97
4.8.2	Lab-16 扫描当前环境下的可用网络 .....	97
4.8.3	Lab-17 网络连接.....	103
4.8.4	Lab-18 web 客户端.....	109
4.8.5	Lab-19 web 服务端.....	115
5	实验套件 AY-IOT Kit 实验例程.....	122
5.1	套件资源.....	122
5.1.1	套件源代码.....	122
5.1.2	套件使用视频.....	125
5.1.3	套件可完成的实验.....	125
5.2	套件模块.....	128
5.2.1	转接板.....	128
5.2.2	高亮 LED 驱动模块（MelodyLED） .....	128
5.2.3	步进电机模块（MelodyStepMotor） .....	134
5.2.4	模拟温度模块（MelodyLMT84） .....	140
5.2.5	温湿度传感器模块（MelodyHDC1080） .....	145
5.2.6	光感模块（MelodyOPT3001） .....	154
5.2.7	三轴加速度模块（MelodyADXL345） .....	160
5.2.8	LDC1000 模块（MelodyLED1000） .....	168
5.2.9	电池管理模块（MelodyLiBattery） .....	177
5.2.10	血压模块（MelodyBP） .....	181

# 物联网实验套件 AY-IOT KIT 使用指南

——基于 Energia 和 CC3200 LaunchPad

## 1 实验套件

### 1.1 概要

物联网实验套件使用 CC3200 为主控 MCU，使用 Energia 为软件开发平台，实现网络通信功能。套件为每个模块提供了不同的实验例程，通过各个实验例程用户可以了解到 Energia 平台下 CC3200 的 WiFi 操作过程，以及其他如 ADC、I2C、SPI、PWM 等外设的实现。

套件资料可在艾研信息官方网站上下载：[http://www.hpati.com/ay\\_wifi/product\\_57.html](http://www.hpati.com/ay_wifi/product_57.html) 或者 <http://www.hpati.com/resources/>。

套件整体模块见图 1-1。

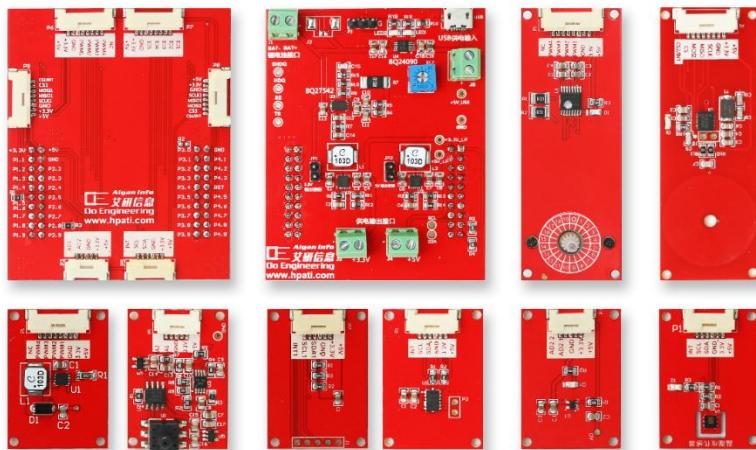
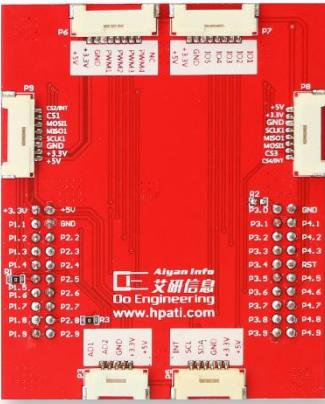
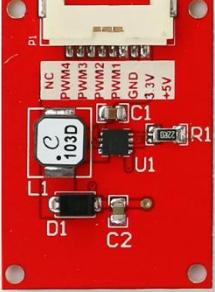
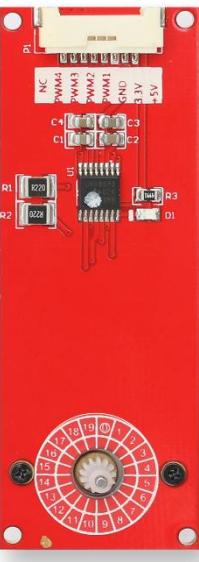
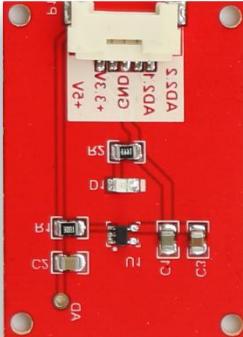
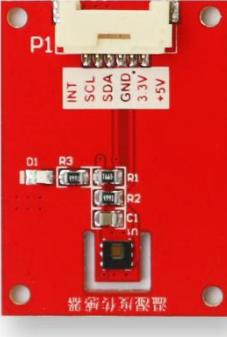
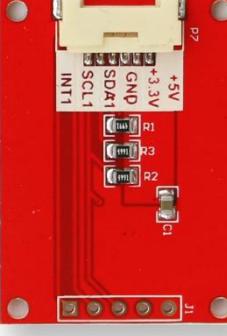
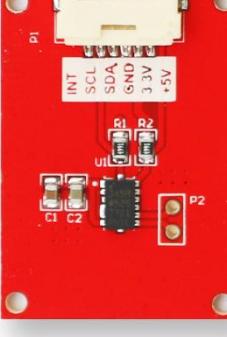


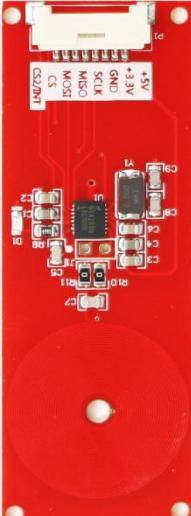
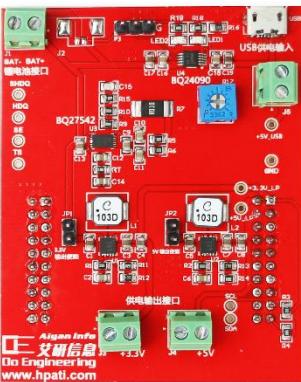
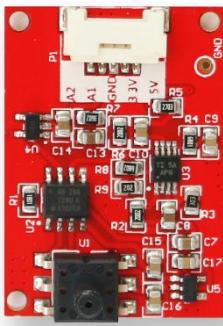
图 1-1 实验套件

物联网实验套件使用 TI 公司的 Energia 软件进行开发，下载地址为 <http://energia.nu/download/>，可选择最新版本下载。Energia 是一个开源软件，无代码限制（TI 的另一款开发软件——CCS 可永久免费使用代码限制版本），下载并解压后即可直接使用。对于 Energia 是使用可查看视频资源《Getting Started Guide》或者登录 Energia 官网了解，网址：[http://energia.nu/guide/guide\\_windows/](http://energia.nu/guide/guide_windows/)。Energia 本身提供很多库函数，可在 Energia 官网上了解库函数的使用：<http://energia.nu/reference/>。

## 1.2 实验套件模块简介

	模块	概况
转接板		<p>1、1 对 BoosterPack 接口，连接 CC3200；      2、1 组 GPIO 接口，包含 5 各 IO 口；      3、1 组 PWM 接口，包含 4 各 PWM 口；      4、2 组 SPI 通信接口；      5、1 组 I2C 通信接口；      6、1 组 AD 接口，包含 2 各 AD 采集通道。</p>
高亮 LED 驱动模块		<p>1、模块使用芯片 TPS61043 驱动；      2、使用 PWM 进行控制，可调节 LED 亮度。</p>
步进电机模块		<p>1、模块使用 DRV8833 驱动步进电机      2、使用 4 路 PWM 进行驱动，可通过软件改变 PWM 的输出频率和 4 路 PWM 的对应相位，从而改变步进电机的工作速度和方向。</p>

模拟温度采集模块		<p>1、模块使用芯片 LMT84 采集温度；      2、使用 ADC 接口获取 LMT84 温度传感器数据      3、测量精度为 <math>\pm 0.4^{\circ}C</math>      4、输出受到短路保护</p>
温湿度传感器模块		<p>1、模块使用芯片 HDC1080 采集温湿度数据；      2、使用 I2C 接口，读取 HDC1080 温湿度传感器数据；      3、相对湿度精度达到 <math>\pm 2\%</math>      4、温度精度达到 <math>\pm 0.2^{\circ}C</math>      5、14 位测量分辨率</p>
光感模块		<p>1、模块使用芯片 OPT3001 采集光感数据      2、使用 I2C 接口，读取 OPT3001 数据      3、测量范围：0.01Lux 至 83K Lux      4、23 位有效动态测量范围，具有自动增益范围设置功能</p>
三轴加速度模块		<p>1、模块使用 ADXL345，采集 X,Y,Z 三个方向上的加速度数据；      2、使用 I2C 接口，读取 ADXL345 采集到的数据；      3、可进行单振/双振，活动/非活动检测，自由落体检测</p>

LDC1000 模块		<p>1、模块使用芯片 LDC1000  2、使用 SPI 接口，读取 LDC1000 采集的数据  3、模块可对线性位置或者角度位置、位移、运动、压缩、振动以及金属成分进行测量。</p>
电池管理模块		<p>1、模块使用芯片 BQ24090 完成锂电池充电，BQ27542 完成锂电池参数监控；  2、模块实现锂电池充放电功能；  3、模块可选择充电电流大小；  4、模块对外输出电压可选择 3V 或者 5V  5、使用 I2C 接口，完成设置和读取 BQ27542 相关数据</p>
血压模块		<p>1、使用芯片 MPS3117 采集数据  2、使用 AD 接口，读取 MPS3117 数据  3、测量范围 5.8PSI  4、可实现静态压和血压波动数据采集</p>

## 2 实验环境准备

### 2.1 简介

本章节介绍实验环境的搭建，主要从硬件和软件环境两个方面展开。硬件环境：介绍 CC3200 LaunchPad 的连线方式和 BoosterPack 的端口说明；软件方面：介绍开 Energia 以及 CC3200 LaunchPad 支持库以及驱动的下载和安装。

### 2.2 硬件环境

- 本章所有实验均在 Windows 7 下实现
- 实验需准备 CC3200 LaunchPad
- 实验过程中涉及 AY-IOT Kit。资源：[http://www.hpati.com/ay\\_wifi/product\\_57.html](http://www.hpati.com/ay_wifi/product_57.html) 该套件自带例程，可直接下载使用，该部分内容将在第 5 章进行介绍。文档第 4 章在介绍 Energia 使用过程中会 AY-IOT Kit 中的部分模块，而该章节中的例程都是另外编写的。

#### 2.2.1 CC3200 LaunchPad BoosterPack

Energia 对 CC3200 LaunchPad BoosterPack 进行了重新定义，各端口定义可在 Energia 网站地址：[http://energia.nu/pin-maps/guide\\_cc3200launchpad/](http://energia.nu/pin-maps/guide_cc3200launchpad/) 上查看，也可将端口定义图下载以便查看。

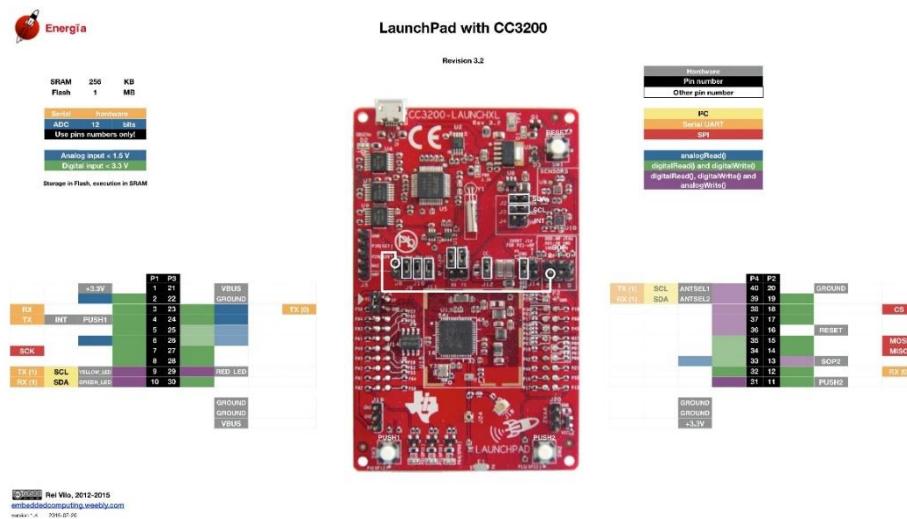


图 2-1 端口定义图

### 2.2.2 CC3200 LaunchPad 连接方式

CC3200 LaunchPad 通过板上的 USB 接口和 PC 机连接, 可实现程序烧写和串口通信功能。CC3200 LaunchPad 提供多种程序烧写方式: JTAG, SWD, FLASH。下载方式通过 SOP 三组跳线冒 (分别可称作 SOP0, SOP1, SOP2) 进行选择。板上 JTAG 管脚对应 J8, J9, J10, J11。Energia 环境下烧写程序选择连接 J8 和 SOP2, 连接方式可查考图 2-14 白线所示方式和图 2-15 实物连接方式。

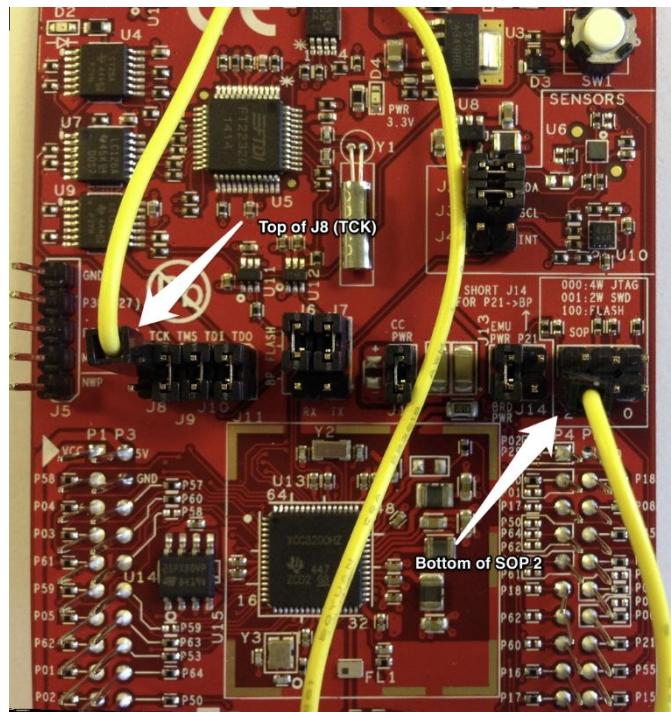


图 2-2 实物连接

## 2.3 软件环境

### 2.3.1 Energia 软件下载及安装

Energia 下载地址: <http://energia.nu/download/>, 可以选择 windows 平台下的最新版本。文档将使用当前最新版本 (版本: Energia 1.6.10E18) 进行介绍。文档中介绍的一些功能仅在当前最新版本上有效, 低于文档使用版本的其余版本 Energia, 其功能存在较大差别, 用户需自行熟悉这些差异。

Energia 1.6.10E18 (8/11/2016)

Mac OS X: Signed Binary release version 1.6.10E18 (8/11/2016)

Download here: [energia-1.6.10E18-macosx-signed.zip](#)

选择windows平台

Windows: Binary release version 1.6.10E18 (8/11/2016)

Download here: [energia-1.6.10E18-windows.zip](#)

Linux 64-bit: Binary release version 1.6.10E18 (8/11/2016) Built and tested on Ubuntu 14.04 LTS (Trusty Tahr).

Download here: [energia-1.6.10E18-linux64.tar.xz](#)

图 2-3 Energia 软件

下载 Energia-1.6.10E18.zip 后将其解压到任意目录即可直接使用，目录可选择【C:\Program Files (x86)】或者其它。双击对应目录下的 energia.exe 即可打开软件，软件界面如下：

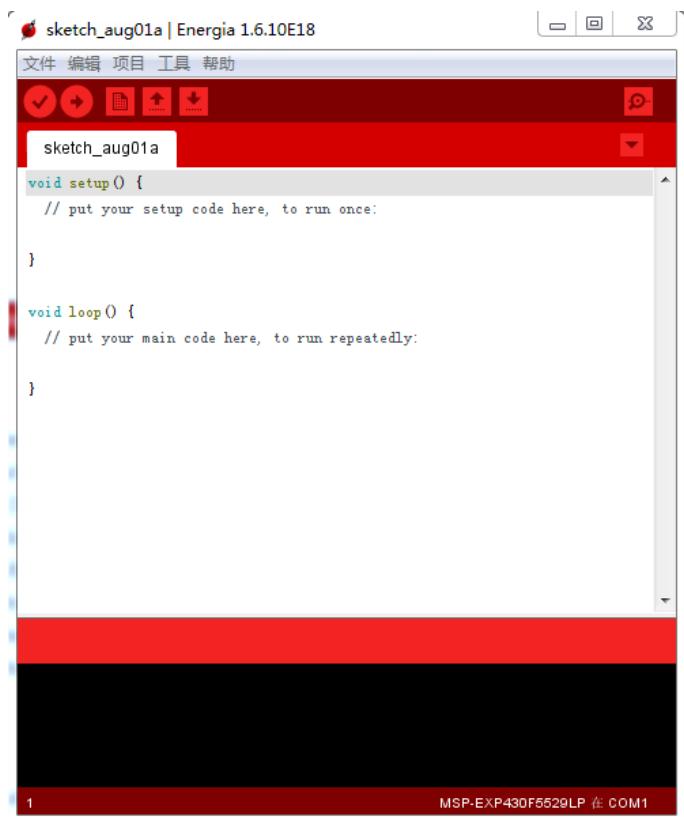


图 2-4 Energia 软件界面

### 2.3.2 CC3200 LaunchPad 支持库下载安装

Energia-1.6.10E18 为当前最新版本，该版本较以前版本发生了很多改变，其中之一就是软件增加了“开发板管理器”，其位于“工具->开发板”下。软件默认仅支持 MSP430 开发，之前版本默认支持所有的 LaunchPad 系列。故在进行实验前需要安装 CC3200 LaunchPad 的支持库，安装方式如下：

- 进入开发板管理器，界面如图 2-6 所示。开发板管理器中显示 Energia MSP430 boards 已经安装，其余开发板未安装；



图 2-5 默认支持的开发板

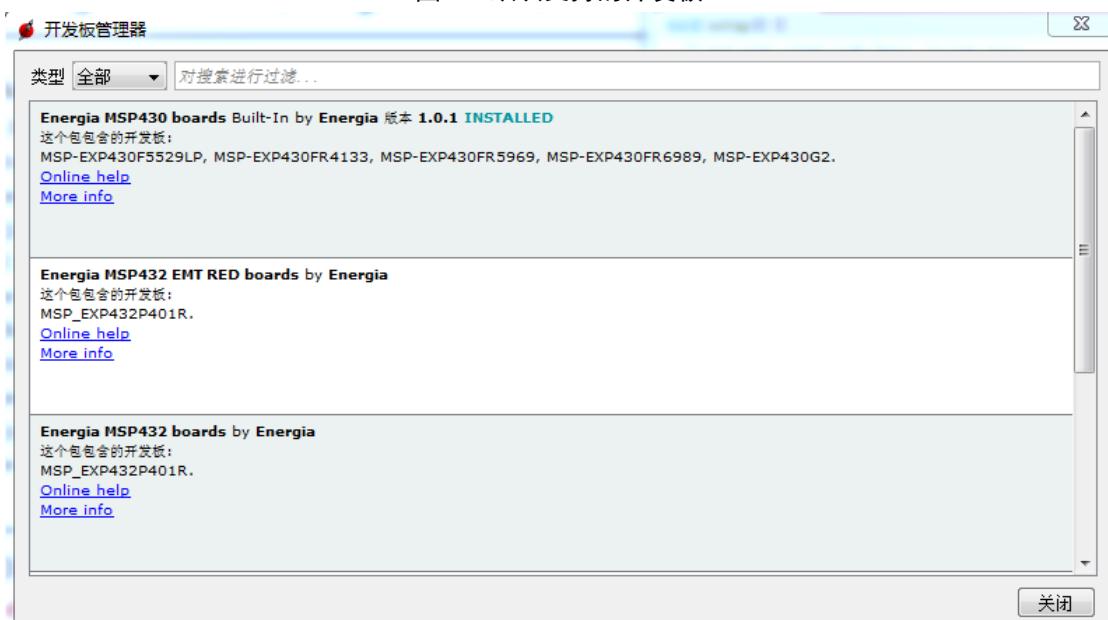


图 2-6 开发板管理器

- 选择 Energia CC3200 boards；

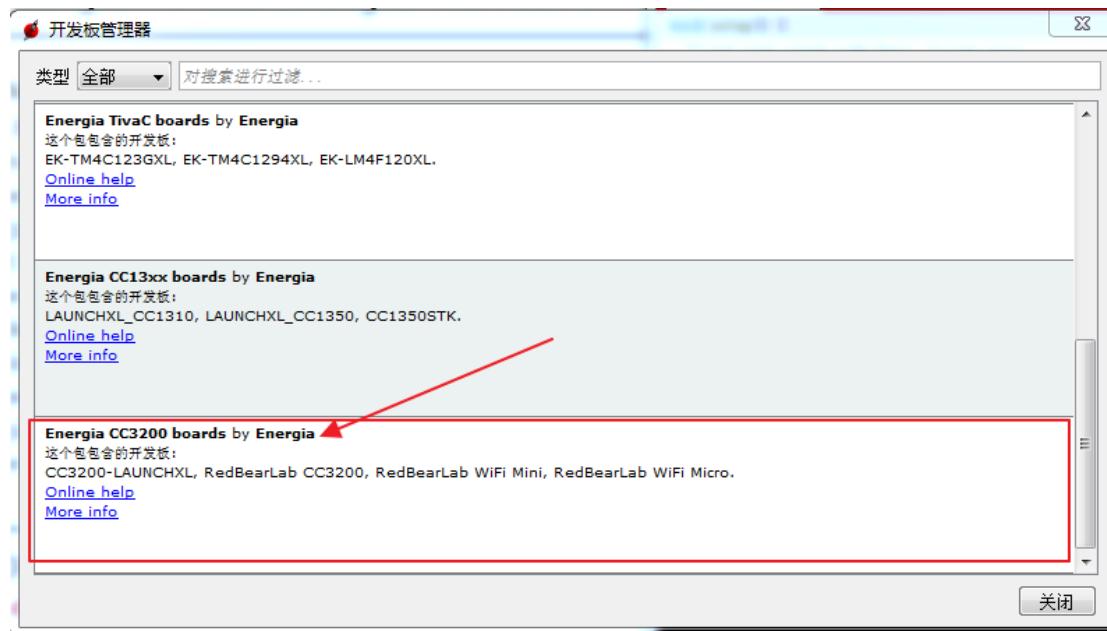


图 2-7 选择 Energia CC3200 boards

3. 选中后点击安装;

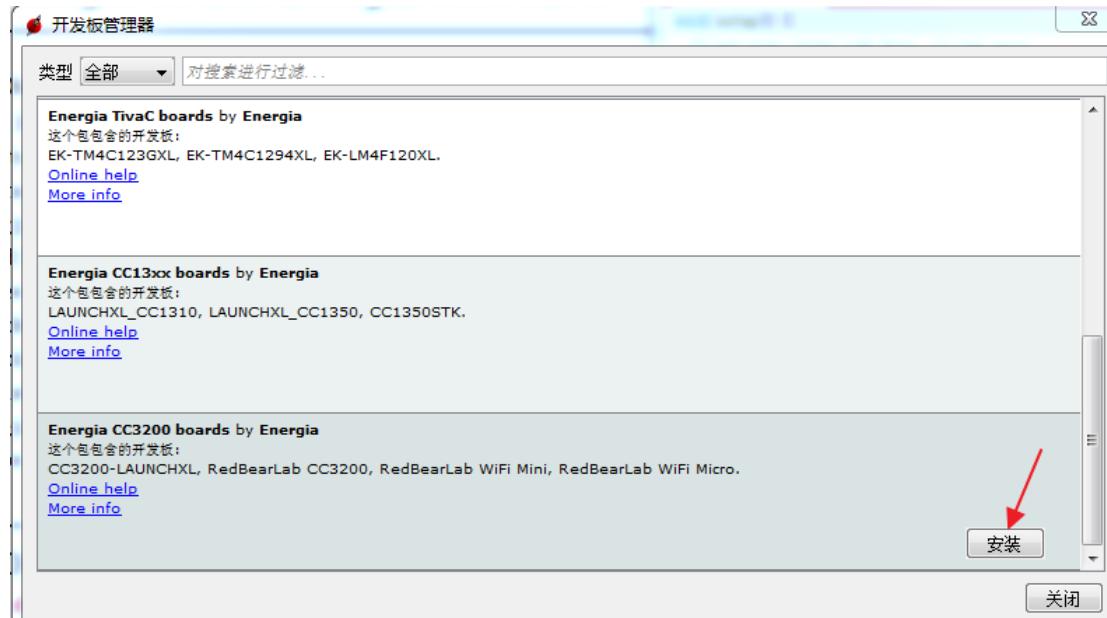


图 2-8 选择安装

4. 等待安装完成, 安装过程中会下载开发板所需的所有资源, 该过程需要联网;

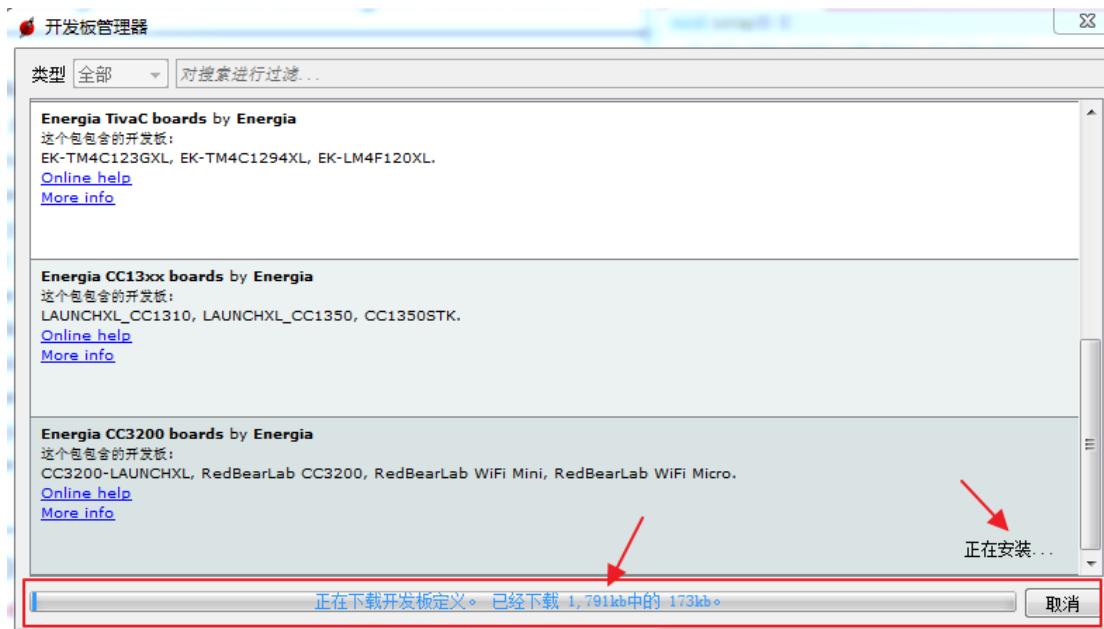


图 2-9 安装过程

5. 安装完成后可进入工具->开发板下查看已经安装的开发板，相比较图 2-5，图 2-10 多了 CC3200 系列开发板。

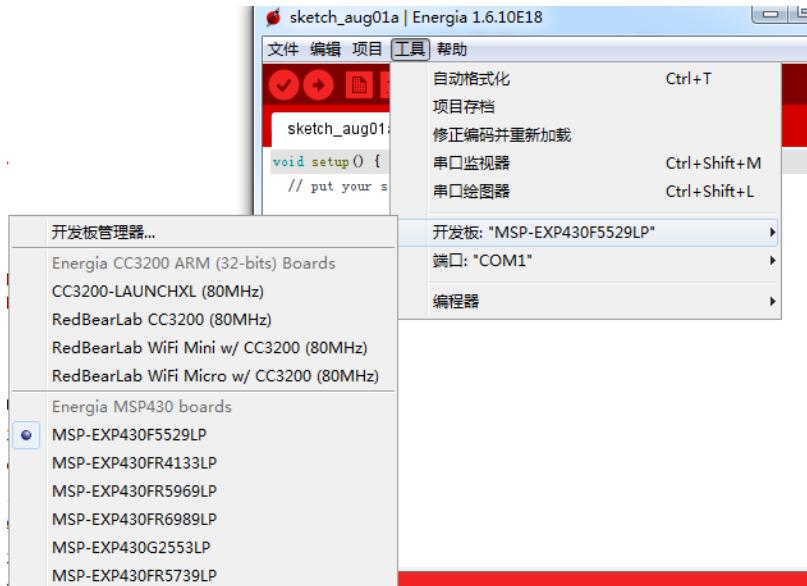


图 2-10 支持的开发板

### 2.3.3 CC3200 LaunchPad 驱动下载及安装

CC3200 驱动下载地址：[http://energia.nu/guide/guide\\_windows/](http://energia.nu/guide/guide_windows/)，选择 *Installing the LaunchPad drivers* 下的 *CC3200 LaunchPad*。进入下载页面时，若没有显示如图 2-11 所示信息，用户可以点击“CC3200 LaunchPad”前的“+”，图 2-11 信息将显示。选择 2 中的链接地址“*CC3200 LaunchPad CDC drivers zip file for Windows 32 and 64 bit*”进行驱动下载。

## Installing the LaunchPad drivers

To use Energia you will need to have the LaunchPad drivers installed. The drivers allow your PC to “see” the LaunchPad on a serial COM port when it is connected.

+ MSP-EXP432P401R LaunchPad

+ CC3200 LaunchPad

Follow the instructions on this page <http://energia.nu/cc320oguide/> for important jumper settings and firmware upgrade instructions.

1. Do **not** connect your CC3200 LaunchPad to your PC. If you already plugged it into your PC then unplug it before proceeding to step 2.
2. Download the CC3200 Drivers for Windows: [CC3200 LaunchPad CDC drivers zip file for Windows 32 and 64 bit](#)
3. Unzip and double click DPinst.exe for Windows 32bit or DPinst64.exe for Windows 64 bit.
4. Follow the installer instructions.
5. Connect your CC3200 LaunchPad to your PC. The CC3200 will be automatically recognized.

图 2-11 CC3200 驱动

下载的驱动压缩文件解压后，根据图 2-11 中 3 的说明选择合适的版本安装。安装过程如下：

1. 选择合适的版本进行安装，Windows 32bit 系统选择 DPInst.exe，Windows 64bit 系统选择 DPInst64.exe。双击对应版本 exe 文件后将出现驱动程序安装向导，如图 2-12 所示；

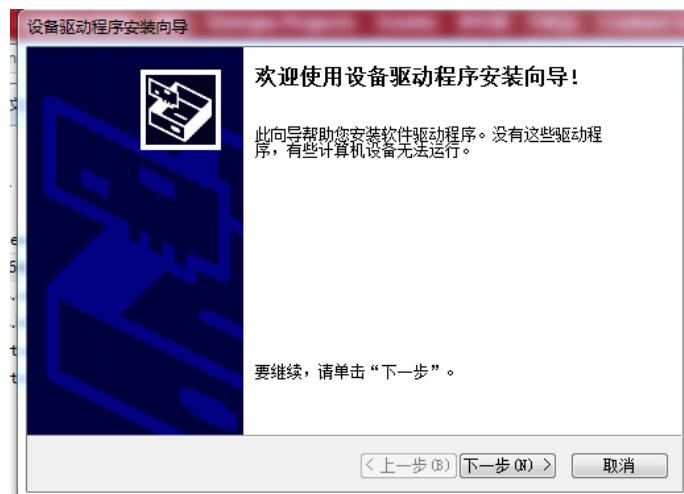


图 2-12 驱动安装向导

2. 选择下一步进行安装，等待安装完成；

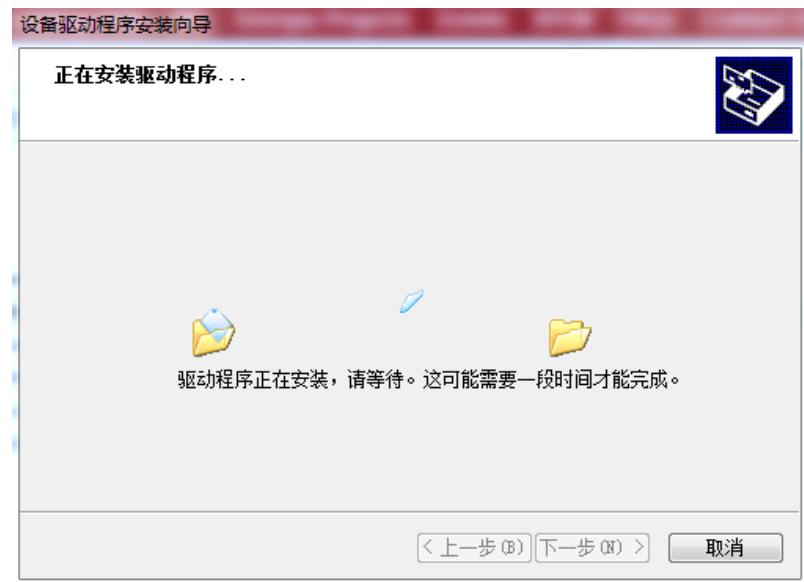


图 2-13 CC3200 LaunchPad 驱动安装过程

3. 安装完成,

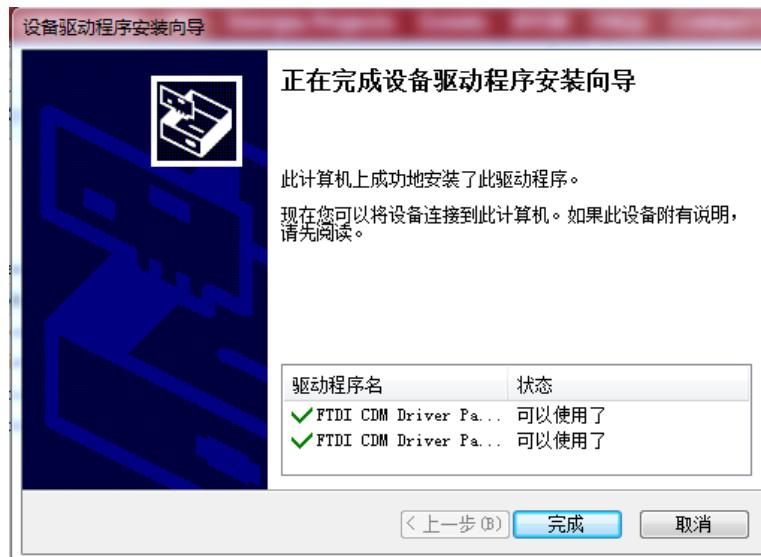


图 2-14 CC3200 LaunchPad 驱动安装完成

4. 插入 CC3200 LaunchPad 后, 系统能够自动识别到开发板。进入设备管理器后可看到 CC3200 LaunchPad 端口。



图 2-15 CC3200 LaunchPad 端口

### 3 Energia

#### 3.1 开发环境

Energia 开发环境包括一个代码编辑区（code editor）、控制台（console）、信息提示区（message area）、工具条（toolbar）以及菜单栏（menu）。

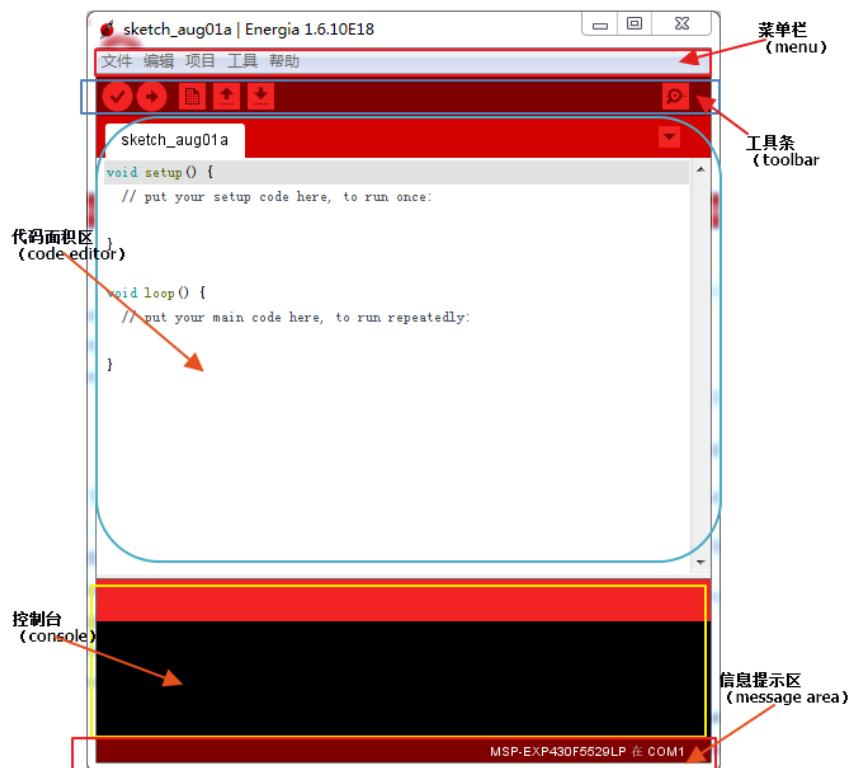


图 3-1 Energia 软件

1. 菜单栏（menu）：包括文件、编辑、项目、工具和帮助功能。下面介绍菜单栏中几个常用的功能：
  - “项目”下有加载库功能，其可在编辑代码时直接导入代码所需的库文件（即直接在代码首行插入 `include <xxx.h>`），比如加入 MQTT 库操作如下：

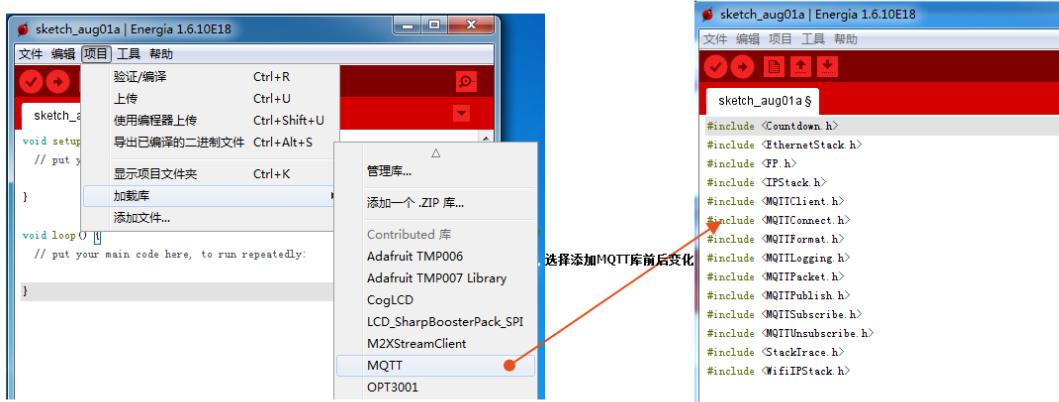


图 3-2 插入 MQTT 库

- “项目”下存在“库管理器”，可管理当前环境下所安装的库文件。其位于项目->加载库->管理库；

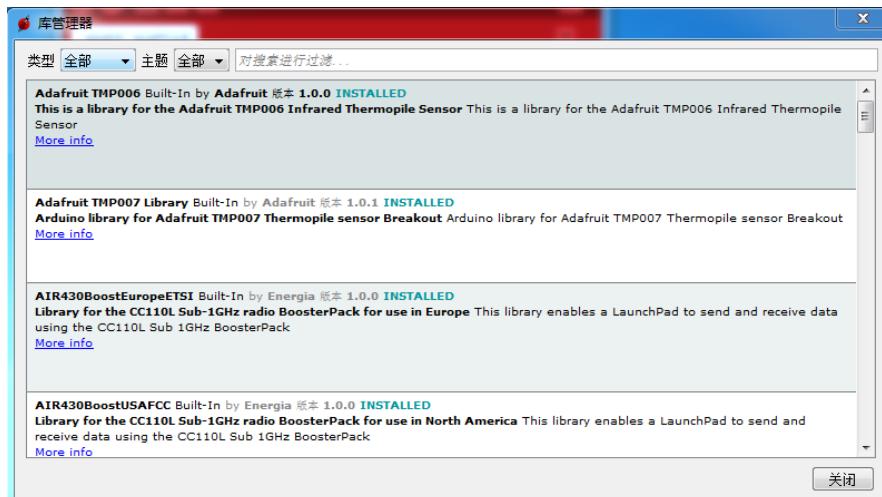


图 3-3 库管理器

- “工具”下可进行开发板选择，路径如下：工具->开发板->各类开发板

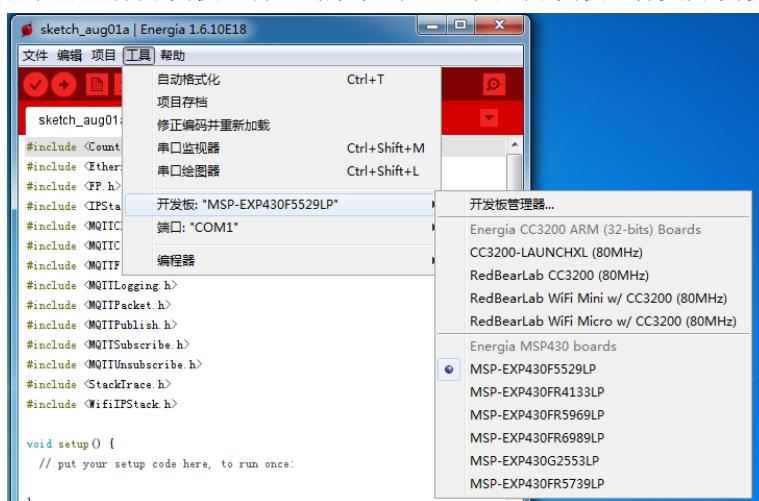


图 3-4 开发板

- “工具”下可进行端口选择，路径如下：工具->开发板->各个端口

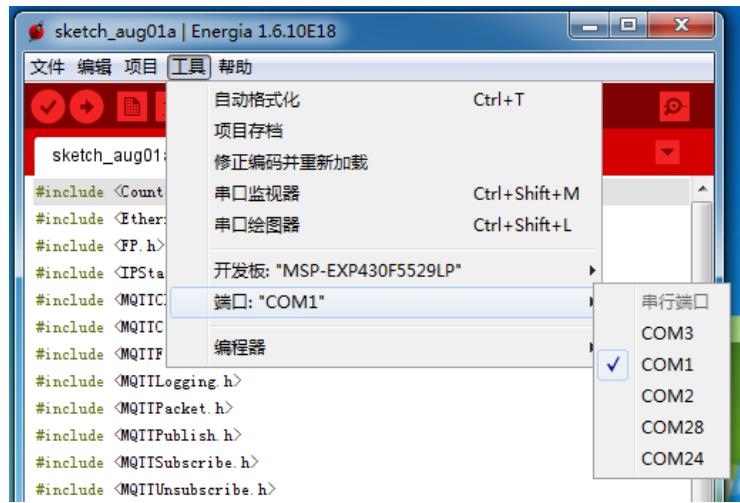


图 3-5 端口

2. 工具条 (toolbar): 包括验证 (Verify)、上传 (Upload)、新建 (New)、打开 (Open)、保存 (Save) 以及串口监视器 (Serial Monitor)。



: 编译 (Verify), 检查程序中的错误;



: 上传 (Upload), 编译程序并且将程序烧写至开发板;



: 新建 (New), 新建一个 sketch 文件, sketch 为 Energia 所描述的文件类型, 其后缀为.ino;



: 打开 (Open), 打开一个 sketch 文件;



: 保存 (Save), 保存当前 sketch 文件;



: 串口监视器 (Serial monitor), 打开串口监视器, 可查看开发板通过串口打印至 PC 的信息, 界面如下图:

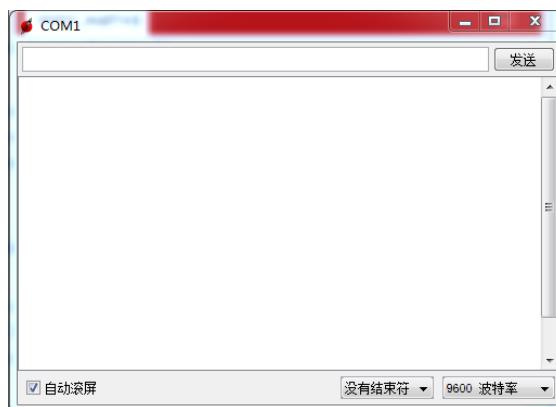


图 3-6 Serial monitor

3. 代码编辑区 (code editor): 代码编写区域;
4. 控制台 (Console): 程序的验证、编译和烧写的过程信息均显示在该区域;
5. 信息提示区 (message area): 显示当前环境下选择的开发板和端口。

### 3.2 内置函数

Energia 提供了一系列的内置变量和功能函数，如 `pinMode()`,`digitalWrite()`等，这些内容可在 Energia 官网上查看，地址：<http://energia.nu/reference/>。当然用户可以直接查看本地版本的 reference，保证在 PC 无联网情况下熟悉 Energia 功能。查看方式如下：

1. 选中需要查看的变量或者功能函数；
2. 点击右键，在快捷菜单中找到“在参考文件中查找”；

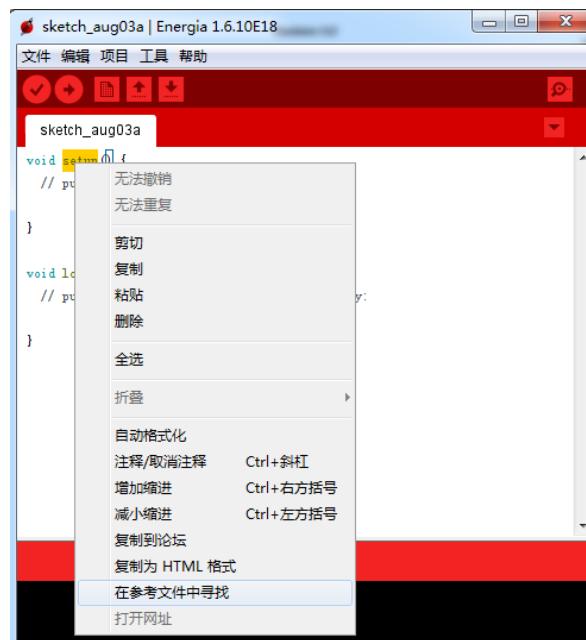


图 3-7 本地参考

3. 选中后单击可进入参考页面，提供的页面为本地网页模式。

### 3.3 最小代码

Energia 软件打开后所展示的界面就是 Energia 最小代码结构，可见图 2-4，其代码仅包括 `setup()` 和 `loop()` 函数。

**setup()函数：**当程序成功烧写至 MCU 后，该函数中的代码会首先执行并且仅执行一次，所以一般在 `setup()` 函数体内进行变量初始化，端口模式设置，初始化程序需要使用的外设资源等。

**loop()函数：** `setup()` 函数执行结束后，程序将进入 `loop()` 函数执行。该函数代码将持续执行，故所有对 MCU 的控制功能均可在该函数体内实现。

当程序中仅使用到 Energia 内置的函数一般无需使用“#include”进行引入对应“.h”文件；当使用其它库函数时，一般需要使用“#include”进行引入对应“.h”文件。导入方式可参考图 3-2。

对于属性 C/C++ 编程语言或者熟悉在 CCS 平台下开发 MCU 程序的用户可能会感到奇怪，因为程序的入口基本为 `main()` 函数，而 Energia 下最小代码不存在 `main()` 函数。根据 `setup()` 和 `loop()` 函数的功能，可以理解为程序模型如下：

```
void main(void)
```

```
{  
    setup();  
  
    while(1)  
    {  
        loop();  
    }  
}
```

其实 Energia 环境是存在 main()函数的，该函数可以在 CC3200 LaunchPad 开发包安装路径下找到，其路径为【C:\Users\用户名（PC 用户名，根据用户实而不同）\AppData\Local\Energia15\packages\energia\hardware\cc3200\1.0.2\cores\cc3200】，该路径下存在 main.cpp 文件，该文件中存在 main()函数，如下：

```
int main(void)  
{  
    setup();  
  
    for(;;){  
        loop();  
        if(serialEventRun) serialEventRun();  
    }  
}
```

其中 SerialEventRun()为串口事件函数，执行串口通信函数用于接收串口数据。该函数将在“4.3UART”具体介绍。

## 4 Energia 基础例程

本章节通过 GPIO、外部中断、UART、ADC12、PWM、I2C、SPI 和 WIFI 功能模块进行介绍，说明 Energia 平台下对 CC3200 LaunchPad 的开发使用。

### 4.1 GPIO

#### 4.1.1 概要

本章节具体介绍 Energia 环境下 GPIO 的数字输出和输入功能。输出功能具体通过控制 LED 闪烁进行说明，输入功能具体通过读取按键端口高低电平进行说明。

#### 4.1.2 Lab-01 LED 闪烁

##### 4.1.2.1 实验准备

实验完成 CC3200 LaunchPad 上 LED 闪烁功能。CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80MHz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。



图 4-1 开发板和端口选择结果显示

##### 4.1.2.2 端口映射

CC3200 LaunchPad 开发板拥有三个 LED，分别为连接至 P01、P02、P64 端口号，其直接对应 BoosterPack 中 P1\_9(YELLOW\_LED)、P1\_10(GREEN\_LED)、P3\_29(RED\_LED)，可查看图 2-1 或者图 4-2

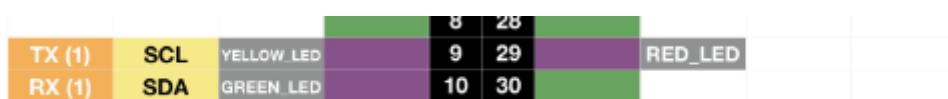


图 4-2 LED 对应端口

##### 4.1.2.3 代码解释

代码例程选择 Energia 提供的 Blink 实例，其可通过“文件->示例->01.Basics->Blink”打开。完整代码看 4.1.2.4。

程序使用了 LED 宏定义，方便修改实际使用的 LED 端口，如下：

```
// most launchpads have a red LED
#define LED RED_LED

//see pins_energia.h for more LED definitions
//#define LED GREEN_LED
```

在 setup() 函数中，首先对 LED 端口进行了初始化，初始化函数为 pinMode，函数原型如下：

表 4-1 函数 pinMode

语法	<b>pinMode(pin,mode)</b>
参数	<b>pin:</b> 端口号，该端口为 BoosterPack 定义的端口号，取值范围为 1~40。如例程中 RED_LED 为 P3_29（位于 P3 排针，29 号端口），即实际 RED_LED=29，可在 pin_energia.h 中查看定义。如图 4-3 所示； <b>mode:</b> pin 所指端口号配置的模式，取值为 INPUT, OUTPUT, INPUT_PULLUP, INPUT_PULLDOWN
返回值	无
说明	将参数 pin 所指的端口配置成 mode 所指代的功能
实例	pinMode(RED_LED,OUTPUT) (等同于 pinMode(29,OUTPUT))

```
staticconstuint8_t RED_LED =29;
staticconstuint8_t GREEN_LED =10;
staticconstuint8_t YELLOW_LED =9;
```

图 4-3 LED 于 pin\_energia.h 的定义

程序将 LED 端口设置成输出模式：

```
// initialize the digital pin as an output.
pinMode(LED, OUTPUT);
```

在 loop() 函数中实现了对 LED 的闪烁控制。程序是 digitalWrite 实现对 LED 的控制。函数原型如下：

表 4-2 函数 digitalWrite

语法	<b>digitalWrite(pin,value)</b>
参数	<b>pin:</b> 端口号，该端口为 BoosterPack 定义的端口号，取值范围为 1~40。如例程中 RED_LED 为 P3_29（位于 P3 排针，29 号端口），即实际 RED_LED=29，可在 pin_energia.h 中查看定义。如图 4-3 所示； <b>value:</b> pin 所指端口的输出电压值，取值为 HIGH 和 LOW，HIGH 表示输出高电平，LOW 表示输出低电平
返回值	无
说明	pin 所指的端口输出高低电平
实例	digitalWrite(RED_LED,HIGH) (等同于 digitalWrite(29,HIGH))

点亮 LED：

```
digitalWrite(LED, HIGH);
```

关闭 LED：

```
digitalWrite(LED, LOW);
```

在点亮和关闭 LED 之间使用 `delay()` 函数进行延时操作。`loop()` 整体如下：

```
// the loop routine runs over and over again forever:
void loop() {
    digitalWrite(LED, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000); // wait for a second
    digitalWrite(LED, LOW); // turn the LED off by making the voltage LOW
    delay(1000); // wait for a second
}
```

`delay()` 函数如下：

表 4-3 函数 `delay()`

语法	<code>delay(ms)</code>
参数	<b>ms</b> : 延时毫秒数
返回值	无
说明	让程序执行进入延时等待
实例	<code>delay(1000)</code>

#### 4.1.2.4 完整代码和流程图

完整代码：

```
/*
Blink

The basic Energia example.

Turns on an LED on for one second, then off for one second, repeatedly.

Change the LED define to blink other LEDs.

Hardware Required:
* LaunchPad with an LED
*/
// most launchpads have a red LED
#define LED RED_LED

// see pins_energia.h for more LED definitions
// #define LED GREEN_LED
// the setup routine runs once when you press reset:
void setup() {
// initialize the digital pin as an output.
pinMode(LED, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
digitalWrite(LED, HIGH); // turn the LED on (HIGH is the voltage level)
delay(1000); // wait for a second
```

```

digitalWrite(LED, LOW); // turn the LED off by making the voltage LOW
delay(1000); // wait for a second
}

```

流程图：

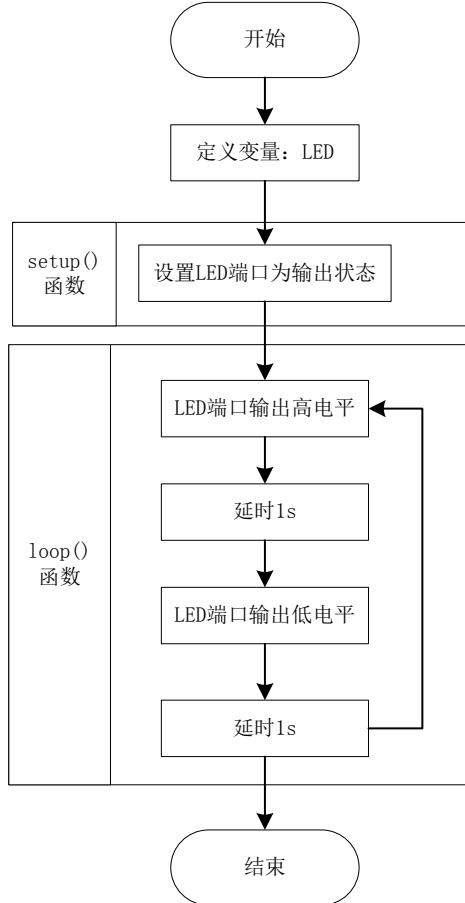


图 4-4 Blink 流程图

#### 4.1.2.5 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，LED 间隔 1s 进行闪烁。

#### 4.1.3 Lab-02 不带延时的 LED 闪烁

##### 4.1.3.1 实验准备

实验完成 CC3200 LaunchPad 上 LED 闪烁功能，本实验和 Lab-01 区别在于在程序实现过程中不使用延时功能。CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通

过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80Mhz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。选择结果显示可见图 4-1。

#### 4.1.3.2 端口映射

CC3200 LaunchPad 开发板拥有三个 LED，分别为连接至 P01、P02、P64 端口号，其直接对应 BoosterPack 中 P1\_9(YELLOW\_LED)、P1\_10(GREEN\_LED)、P3\_29(RED\_LED)，可查看图 4-2。程序中实际使用 P1\_10(GREEN\_LED)。

#### 4.1.3.3 代码解释

Lab-01 在功能实现上使用了延时，这样使代码非常直观，程序逻辑也非常简单。但是这样处理会使 MCU 长时间处于挂起状态，即一直在进行延时操作，这就使有限的 MCU 资源处于极大的浪费中。同时假设程序需要进行多个工作，比如识别按键和 LED 闪烁，如果在程序进行延时操作时用户按下了按键，那么该次按键事件将被忽略，因为 MCU 在进行长时间延时操作时无法判断按键对应端口的电平变化。

上述问题一般的处理方式是采用定时器中断，在每次中断发生时改变 LED 状态。中断处理服务以外的所有时间，MCU 均处于空闲状态。但是 Energia 顶层库文件未直接提供可用的定时器操作，用户可根据 Energia 底层库文件自行实现（即类似于在 CCS 环境下使用 CC3200SDK 进行开发）。然而 Energia 提供了几个与系统时钟相关的函数（系统时钟在 Energia 下默认开启工作，以中断方式实现），如 millis() 函数，该函数用于获取系统时钟已工作的毫秒数。所以在功能实现时可以使用 millis() 函数来实现“Blink Without Delay”功能，具体可看例程。

例程选择 Energia 提供的“Blink Without Delay”实例，可通过“文件->示例->02.Digital->BlinkWithoutDelay”打开，完整代码查看 4.1.3.4。

程序首先定义了 ledPin 变量，方便使用，亦方便更换其余 LED，如 P1\_9(YELLOW\_LED)、P1\_10(GREEN\_LED)。

```
constint ledPin = GREEN_LED;// the number of the LED pin
```

在 setup() 函数中，首先对 LED 端口进行了初始化，通过如下函数将 LED 端口设置成输出模式：pinMode() 函数介绍可见表 4-1。

```
// initialize the digital pin as an output.  
pinMode(ledPin, OUTPUT);
```

在 loop() 函数中，为了不使用 delay() 函数，程序在 loop() 开始执行时使用 millis() 读取当前时间（变量：currentMillis），然后该当前时间（变量：currentMillis）和之前时间（变量：previousMillis，初始化为 0）。当两个变量之差（变量：interval，初始化为 1000）大于 1s 时间（1000ms，millis() 返回 ms），LED 状态将被改变，然后将当前时间（currentMillis）保存至 previousMillis。代码实现如下：digitalWrite() 介绍可见表 4-2，millis() 可见表 4-4

```
// check to see if it's time to blink the LED; that is, if the  
// difference between the current time and last time you blinked  
// the LED is bigger than the interval at which you want to  
// blink the LED.
```

```

unsigned long currentMillis = millis();

if(currentMillis - previousMillis > interval){
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
    if(ledState == LOW)
        ledState = HIGH;
    else
        ledState = LOW;

    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
}

```

代码中使用到的 mills()如下：

表 4-4 函数 mills()

语法	<code>millis()</code>
参数	无
返回值	程序开始执行后工作的毫秒数
说明	函数返回 MCU 开始执行后工作的时间，以 ms 为单位计数，溢出后，计数值将从 0 重新开始计数。计数值为 unsigned long 类型，大约执行 50 天后发生溢出。
实例	<code>unsinged long currentMills = millis();</code>

#### 4.1.3.4 完整代码和流程图

完整代码：

```

// constants won't change. Used here to
// set pin numbers:
constint ledPin = GREEN_LED;// the number of the LED pin

// Variables will change:
int ledState = LOW;// ledState used to set the LED
long previousMillis =0;// will store last time LED was updated

// the follow variables is a long because the time, measured in
milliseconds,
// will quickly become a bigger number than can be stored in an int.
long interval =1000;// interval at which to blink (milliseconds)

```

```
void setup() {
// set the digital pin as output:
pinMode(ledPin, OUTPUT);
}

void loop()
{
// here is where you'd put code that needs to be running all the time.

// check to see if it's time to blink the LED; that is, if the
// difference between the current time and last time you blinked
// the LED is bigger than the interval at which you want to
// blink the LED.
unsignedlong currentMillis = millis();

if(currentMillis - previousMillis > interval){
// save the last time you blinked the LED
previousMillis = currentMillis;

// if the LED is off turn it on and vice-versa:
if(ledState == LOW)
    ledState = HIGH;
else
    ledState = LOW;

// set the LED with the ledState of the variable:
digitalWrite(ledPin, ledState);
}
}
```

流程图:

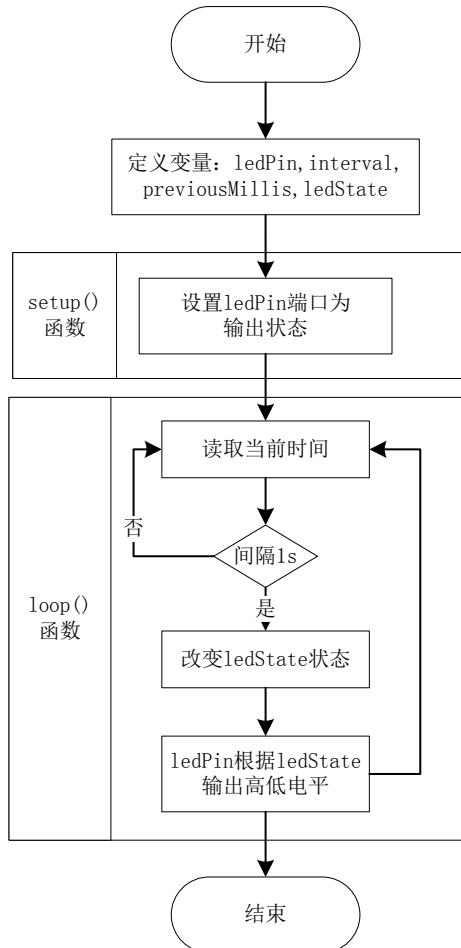


图 4-5 Blink Without Delay 流程图

#### 4.1.3.5 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，LED 间隔 1s 进行闪烁。

#### 4.1.4 Lab-03 通过按键控制 LED

##### 4.1.4.1 实验准备

实验完成按键控制 LED 亮灭功能，实现 GPIO 的输入和输出功能。CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80Mhz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。选择结果显示可见图 4-1。

#### 4.1.4.2 端口映射

CC3200 LaunchPad 开发板拥有三个 LED 和两个按键，LED 分别为连接至 P01、P02、P64 端口号，其直接对应 BoosterPack 中 P1\_9(YELLOW\_LED)、P1\_10(GREEN\_LED)、P3\_29(RED\_LED)，可查看图 4-6。程序中实际使用 P1\_10(GREEN\_LED)。按键分别连接至 P04(GP13)和 P15(GP22)，实际对应 BoosterPack 中的 P1\_3 和 P1\_11。在图 2-1 和图 4-6 中 PUSH1 被对应至 P1\_4，这其实是错误的，实际对应的是 P1\_3。具体可见 CC3200 LaunchPad 原理图，图 4-7 为 CC3200 LaunchPad 原理图部分截图。也可从代码中发现图 4-6 PUSH1 所示端口错误，代码需见 pin\_energia.h 文件，其位于【C:\Users\用户名（PC 用户名，根据用户实而不同）\AppData\Local\Energia15\packages\energia\hardware\cc3200\1.0.2\variants\CC3200-LAUNCHXL】。对应变量定义如下：

```
staticconstuint8_t RED_LED =29;
staticconstuint8_t GREEN_LED =10;
staticconstuint8_t YELLOW_LED =9;
staticconstuint8_t PUSH1 =3;
staticconstuint8_t PUSH2 =11;
```

从变量定义中可知 RED\_LED 连接 29 号端口即 P3\_29, GREEN\_LED 连接 10 号端口即 P1\_10, YELLOW\_LED 连接 9 号端口即 P1\_9, PUSH2 连接 11 号端口即 P1\_11, PUSH1 连接 3 号端口即 P1\_3，故由此可见图 4-6 所示 PUSH1 所连接端口错误。在程序使用中可直接使用 PUSH1，避免引起按下 PUSH1 而程序无响应问题。

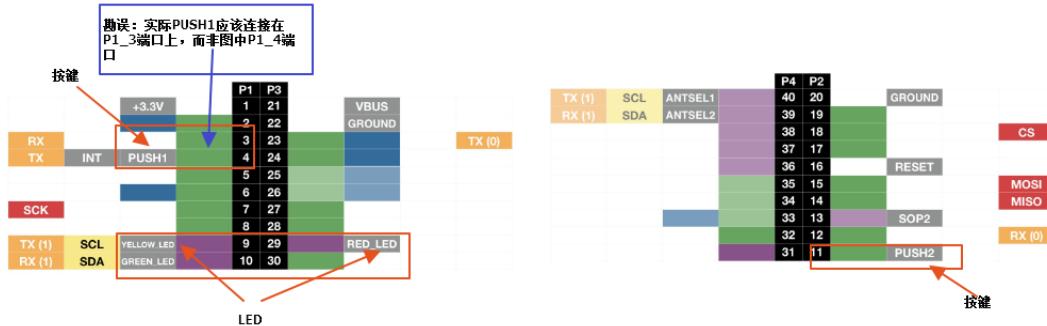


图 4-6 LED 和按键端口映射图

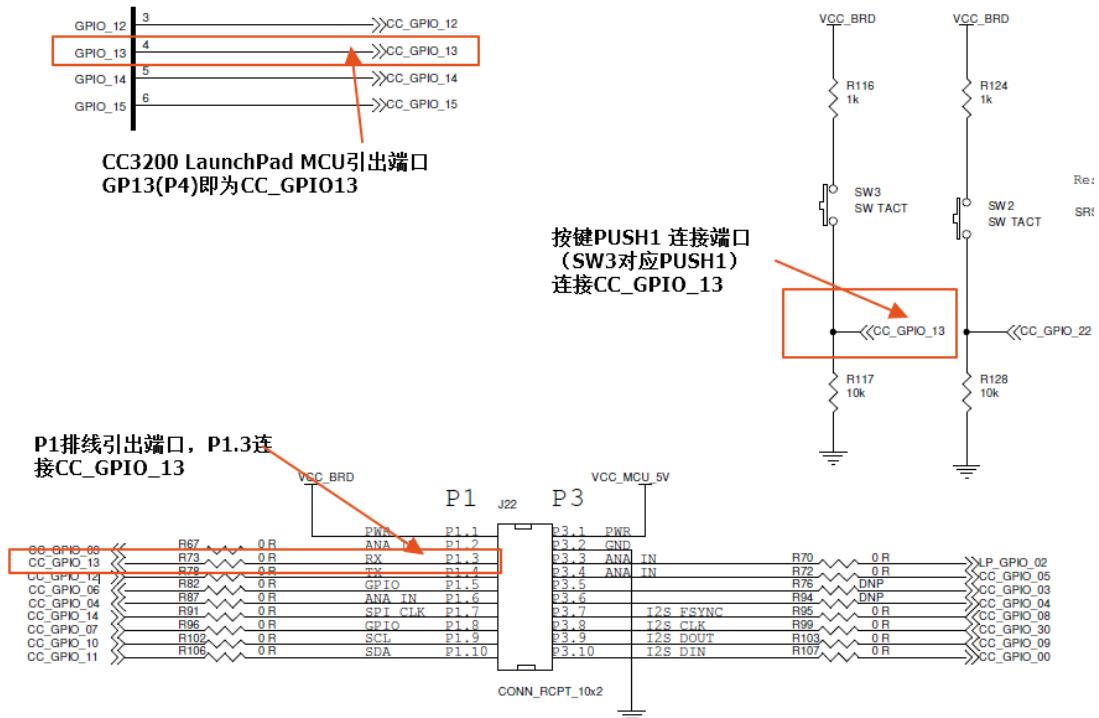


图 4-7 实际 PUSH1 所接端口

#### 4.1.4.3 代码解释

代码例程选择 Energia 提供的 Button 实例，其可通过“文件->示例->02.Digital->Button”打开。代码实现按键 PUSH2 控制 GREEN\_LED 亮灭状态。

程序首先定义 buttonPin 和 ledPin 两个变量，此可方便更换按键端口和 LED 端口。

```
const int buttonPin = PUSH2; // the number of the pushbutton pin
const int ledPin = GREEN_LED; // the number of the LED pin
```

在 setup() 函数中，首先对程序中使用到的端口进行了初始化，即设置 ledPin 为输出状态，buttonPin 为上拉输入状态，代码如下：pinMode() 函数介绍可见表 4-1。

```
// initialize the LED pin as an output:
pinMode(ledPin, OUTPUT);

// initialize the pushbutton pin as an input:
pinMode(buttonPin, INPUT_PULLUP);
```

在 loop() 函数中，首先读取按键端口当前电平，如果 buttonPin 端口为高电平则点亮 LED，如果 buttonPin 端口为低电平则熄灭 LED。代码实现如下：digitalWrite() 函数可见表 4-2，digitalRead() 函数见表 4-5。

```
// read the state of the pushbutton value:
buttonState = digitalRead(buttonPin);

// check if the pushbutton is pressed.
// if it is, the buttonState is HIGH:
if(buttonState == HIGH) {
```

```

// turn LED on:
digitalWrite(ledPin, HIGH);
}
else{
// turn LED off:
digitalWrite(ledPin, LOW);
}

```

digitalRead()函数如下：

表 4-5 函数 digitalRead()

语法	<b>digitalRead(pin)</b>
<b>参数</b>	<b>pin</b> : 端口号, 该端口为 BoosterPack 定义的端口号, 取值范围为 1~40.如例程中 PUSH2 为 P2_11 (位于 P2 排针, 11 号端口), 即实际 PUSH2=11, 可在 pin_energia.h 中查看定义.
<b>返回值</b>	返回 pin 端口电平状态, HIGH 为高电平, LOW 为低电平
<b>说明</b>	读取参数 pin 所指端口的电平状态 (高电平和低电平)
<b>实例</b>	int val = digitalRead(PUSH2)

#### 4.1.4.4 完整代码和流程图

完整代码:

```

// constants won't change. They're used here to
// set pin numbers:

constint buttonPin = PUSH2;// the number of the pushbutton pin
constint ledPin = GREEN_LED;// the number of the LED pin
// variables will change:
int buttonState =0;// variable for reading the pushbutton status

void setup(){
// initialize the LED pin as an output:
pinMode(ledPin, OUTPUT);
// initialize the pushbutton pin as an input:
pinMode(buttonPin, INPUT_PULLUP);
}

void loop(){
// read the state of the pushbutton value:
buttonState = digitalRead(buttonPin);

// check if the pushbutton is pressed.
// if it is, the buttonState is HIGH:
if(buttonState == HIGH){
// turn LED on:

```

```

        digitalWrite(ledPin, HIGH);
    }
else{
// turn LED off:
    digitalWrite(ledPin, LOW);
}
}

```

流程图：

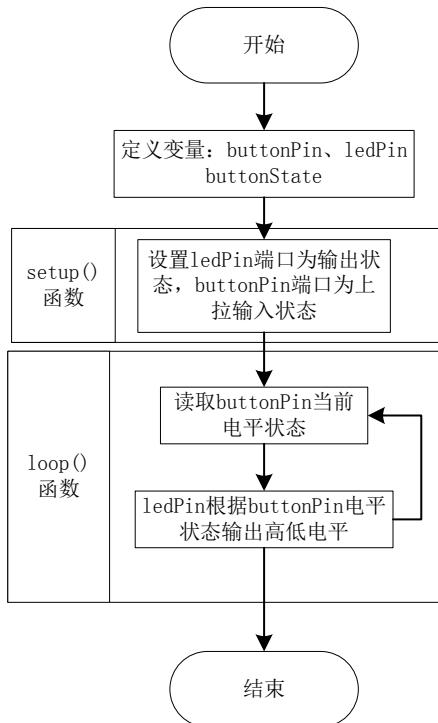


图 4-8 button 实例流程图

#### 4.1.4.5 实验现象

 点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，按下按键 PSUH2 (SW2) 即可改变 GREEN\_LED 状态。

## 4.2 外部中断

### 4.2.1 概要

CC3200 具有诸多中断源：GPIO 中断，Timer 中断，SPI 中断，I2C 中断等。但是 Energia 环境下实现的内置库函数直接暴露给用户使用的中断仅为 GPIO 中断，即外部中断，其可用函数 `attachInterrupt()` 实现，而其余中断实现均需要直接使用 CC3200SDK 进行编程。故本章

节仅介绍外部中断实现。

#### 4.2.2 Lab-04 按键中断

##### 4.2.2.1 实验准备

实验例程采用按键中断方式控制 LED 亮灭功能，本实验和“Lab-03 通过按键控制 LED”区别在于在程序使用外部中断方式实现。“Lab-03 通过按键控制 LED”实验例程中 loop()函数会持续读取 PUSH2 按键端口的电平状态，MCU 将一直处于活跃状态，一定程度上造成了 MCU 资源的过度消耗。如果将 Lab-03 中的代码改成中断方式实现，那么 LED 的亮灭状态改变可在中断服务程序中实现，MCU 可以进入休眠状态或者执行其余任务而无需将有限的资源消耗在读取 PUSH2 端口电平上。

CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，并通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80Mhz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。选择结果显示可见图 4-1。

##### 4.2.2.2 端口映射

例程所使用的端口和“Lab-03 通过按键控制 LED”一致。使用 PUSH2 按键和 GREEN\_LED，由 PUSH2 控制 GREEN\_LED 亮灭状态。端口映射图可见图 4-6 和图 4-7。方便查看给出按键的原理图，如下：

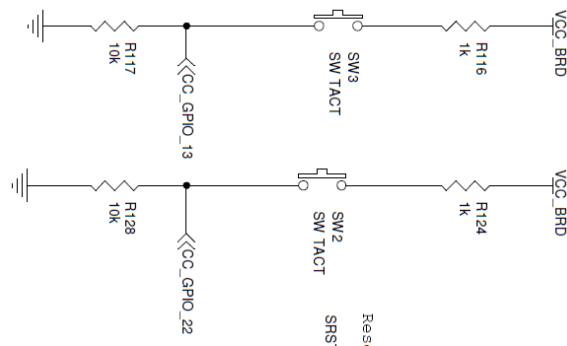


图 4-9 按键原理图

根据图 4-9 所示，按键端口在未按下状态时该端口为低电平，按下时该端口为高电平。即当使用中断方式时，RISING 中断模式发生在按键按下瞬间，而 FALLING 中断模式发生在按键松开瞬间。

#### 4.2.2.3 代码解释

代码例程对 Energia 官网上的 Example 略作修改，用户也可直接使用该 Example 进行实验，地址：<http://energia.nu/reference/attachinterrupt/>。

例程首先定义三个变量：ledPin, buttonPin, state。

```
constint ledPin = GREEN_LED;
constint buttonPin = PUSH2;
volatilebool state = HIGH;
```

在 setup() 函数中初始化 ledPin 和 buttonPin，即设置 ledPin 为输出状态，buttonPin 为上拉输入状态，代码如下：pinMode() 函数介绍可见表 4-1，同时初始化 ledPin 输出高电平，使用函数 digitalWrite() 实现，该函数介绍见表 4-2。最后开启 buttonPin 中断服务，使用函数 attachInterrupt() 实现，函数介绍如下：

表 4-6 函数 attachInterrupt()

语法	<b>attachInterrupt(interrupt, function, mode)</b>
参数	<b>interrupt:</b> 中断源 <b>function:</b> 中断发生时被回调的函数，该函数必须无参数无返回值 <b>mode:</b> 中断触发模式：具有 4 种可选模式，如下 LOW: 低电平触发 CHANGE: 电平状态发生改变时触发 RISING: 上升沿触发，即由低电平跳变成高电平时触发 FALLING: 下降沿触发，即由高电平跳变成低电平时触发
返回值	无
说明	开启外部中断服务，当中断发生时回调由参数 function 对应的中断服务函数
实例	attachInterrupt(PUSH2, blink, RISING)

blink() 函数仅实现 ledPin 端口电平翻转，实现如下：

```
void blink()
{
    state =! state;
    digitalWrite(ledPin, state);
}
```

实例使用按键中断方式控制 LED，并且在中断服务程序中实现 ledPin 电平控制，故 loop() 函数无需实现任何代码。

#### 4.2.2.4 完整代码和流程图

完整代码：

```
constint ledPin = GREEN_LED;
constint buttonPin = PUSH2;
volatilebool state = HIGH;
```

```

void setup() {
// put your setup code here, to run once:
pinMode(ledPin, OUTPUT);
digitalWrite(ledPin, state);
pinMode(buttonPin, INPUT_PULLUP);
attachInterrupt(buttonPin, blink, RISING);
}
void loop() {
// put your main code here, to run repeatedly:
}
void blink()
{
    state = !state;
    digitalWrite(ledPin, state);
}

```

流程图:

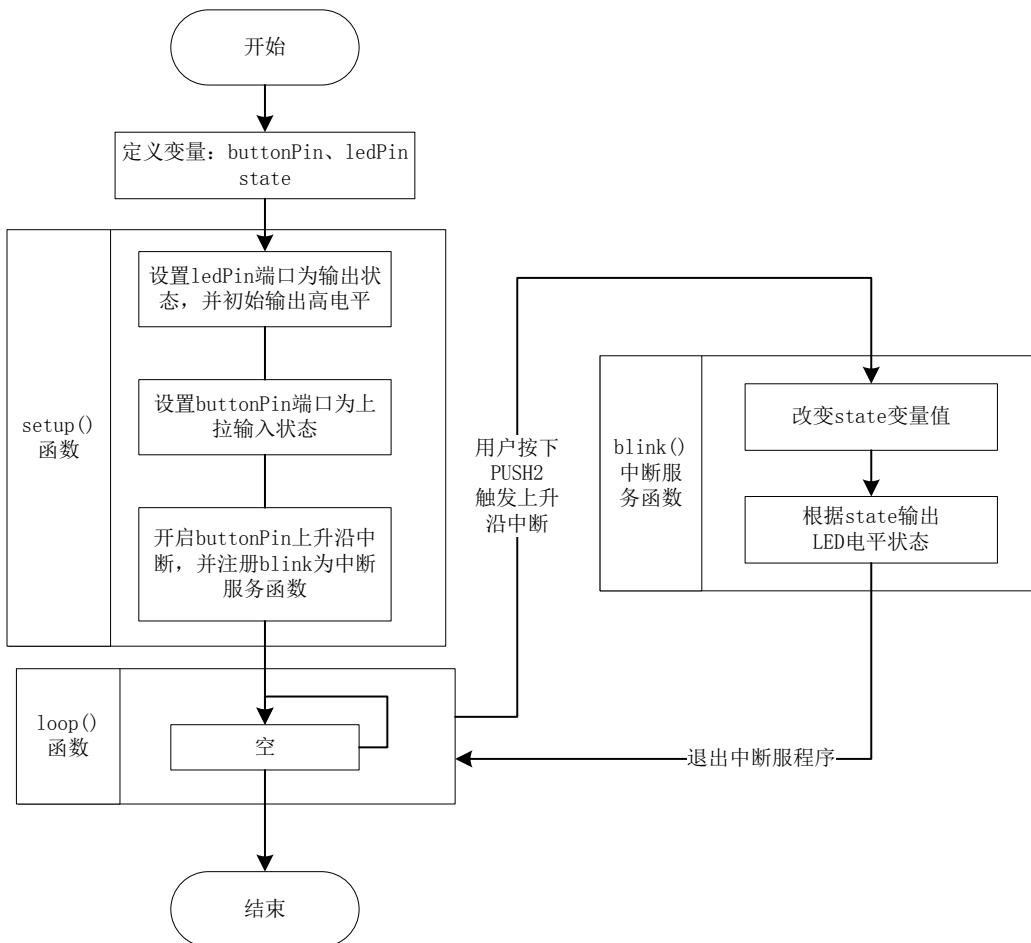


图 4-10 按键中断例程流程图

#### 4.2.2.5 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，按下按键 PSUH2（SW2）即可改变 GREEN\_LED 状态。在多次实验过程中可能会出现非理想状态现象，即 LED 快速闪烁。这些现象是由按键“抖动”引起的，按键“抖动”时程序会多次进入中断服务程序（blink()函数），而“抖动”时间非常短暂，所有就产生了 LED 快速闪烁现象。消除这种现象的其中一种方式为“延时消抖”，该方法将在 Lab-05 中介绍。

#### 4.2.3 Lab-05 延时消抖

未解决 4.2.2.5 所描述的现象，可以在程序中加入适当延时用以消除，当然也可以使用其他方式进行“消抖”，比如使用定时器方式，在本文档中仅介绍延时消抖方式。

##### 4.2.3.1 代码解释

本节代码仅修改了“4.2.2Lab-04 按键中断”中的 blink()函数。实现如下：

```
void blink()
{
    delay(20);
    if(digitalRead(buttonPin))
    {
        state = !state;
        digitalWrite(ledPin, state);
    }
}
```

在 blink()函数中添加了 delay()函数，延时适当时间后读取 buttonPin 端口是否处于高电平，因为 buttonPin 所指代的端口在按键按下时处于高电平，如果在延时后当前端口仍是高电平则表示当前按下操作为有效操作；如果延时后当前端口处于低电平则表示当前按下操作为按键抖动操作；同时还可以避免在一次有效按下操作过程中，按键产生的抖动情况。因为在 delay()过程中，按键的操作将被忽略。

##### 4.2.3.2 完整代码和流程图

###### 完整代码

```
constint ledPin = GREEN_LED;
constint buttonPin = PUSH2;
volatilebool state = HIGH;
void setup(){
// put your setup code here, to run once:
pinMode(ledPin, OUTPUT);
```

```

digitalWrite(ledPin, state);
pinMode(buttonPin, INPUT_PULLUP);
attachInterrupt(buttonPin, blink, RISING);
}

void loop() {
// put your main code here, to run repeatedly:
}

void blink()
{
    delay(20);
    if(digitalRead(buttonPin))
    {
        state = !state;
        digitalWrite(ledPin, state);
    }
}

```

流程图：

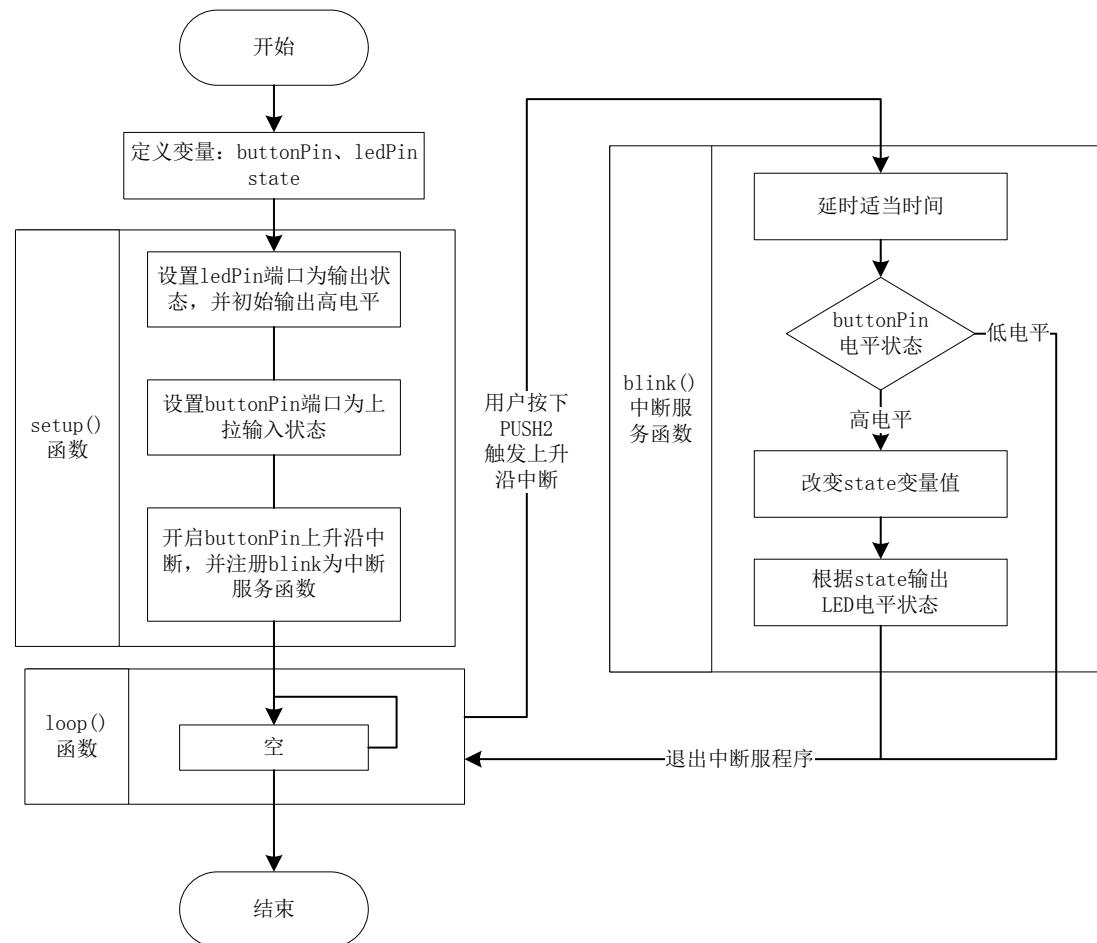


图 4-11 按键中断+延时消抖实例

#### 4.2.3.3 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，按下按键 PSUH2（SW2）即可改变 GREEN\_LED 状态。可适当加长延时时间以增加实验效果。

### 4.3 UART

#### 4.3.1 概要

UART 是一种简单的 PC 和 MCU 或者 MCU 和外设之间的异步通信方式，在硬件连接上其仅需要两根数据线（TX 和 RX）以及一根公共接地线。而 CC3200 LaunchPad 在使用 UART 与 PC 通信时，其实际通过板载 USB 接口进行数据传输，其内部的具体实现无需关心，用户在编程使用时只需按照 UART 编程方式进行编写代码即可。如果使用 UART 接口与其他 MCU 或者外设进行通信，则需要将 CC3200 LaunchPad 的 TX 线和其他 MCU 或者外设的 RX 线连接，CC3200 LaunchPad 的 RX 和其他 MCU 或者外设的 TX 线连接，同时共地连接即可。

在 Energia 下进行 UART 通信可以使用其提供的 Serial 库实现，库文件参考内容可见官网：<http://energia.nu/reference/serial/>。如果想要查看库文件源码可在 CC3200 LaunchPad 安装目录下找到：【C:\Users\用户名（PC 用户名，根据用户实而不同）\AppData\Local\Energia15\packages\energia\hardware\cc3200\1.0.2\cores\cc3200】该目录下的 HardwareSerial.h 和 HardwareSerial.cpp。

Energia 环境提供一个串口监视器（Serial monitor），可见图 3-6。串口监视器可实现数据方式和数据接收功能，用户可将其作为程序调试的辅助方式

#### 4.3.2 Serial 库

Serial 库使用 C++ 实现，具体实现方式可查看 HardwareSerial.h 和 HardwareSerial.cpp 文件。Serial 库存在多个成员函数，函数列表如下：

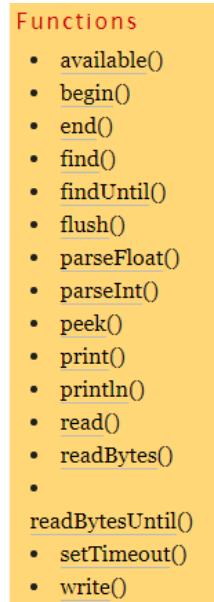


图 4-12 Serial 库成员函数列表

下面就在 UART 编程过程中经常使用的函数进行简单介绍。未说明的函数用户可在 Energia 官网上进行了解，地址：<http://energia.nu/reference/serial/> 在网页左侧 Functions 栏里找到对应函数点击进入即可。

表 4-7 函数 available()

语法	<code>Serial.available()</code>
参数	无
返回值	Serial 接收缓存区中有效字节数
说明	函数返回已经存储在 Serial 接收缓存区的有效字节数，缓存区大小由 SERIAL_BUFFER_SIZE 确定，该宏定义位于 HardwareSerial.h。该函数可在进行 read() 操作前使用。
实例	<pre>if(Serial.available()&gt; 0) {     //put your code in here, for example: read()     Serial.read() }</pre>

表 4-8 函数 begin()

语法	<code>Serial.begin(speed)</code>
参数	<b>speed</b> : 波特率，可选值：300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200。常用 9600 和 115200
返回值	无
说明	设置 UART 通信的波特率，并初始化 UART 功能。在程序使用 Serial 其余成员函数前未使用 begin() 函数，则程序无法正常运行。
实例	<code>Serial.begin(9600)</code>

表 4-9 函数 end()

<b>语法</b>	<code>Serial.end()</code>
<b>参数</b>	无
<b>返回值</b>	无
<b>说明</b>	UART 通信无效，使用该函数后可将 RX 和 TX 端口作为通用 GPIO 使用；重新使能 UART 功能需要使用 begin()函数
<b>实例</b>	<code>Serial.end()</code>

表 4-10 函数 print()

<b>语法</b>	<code>Serial.print(val)</code>
<b>参数</b>	<code>val</code> : 需要打印（传输）的值
<b>返回值</b>	返回当前传输的字节数
<b>说明</b>	打印的数值以 ASCII 码格式传输, 数值类型数据中的每一个数字都以 ASCII 码格式传输, 保留 float 类型 (float 类型默认保留两位小数); 字符和字符串类型则传输的即为写入的数据。传输的数据可使用串口监视器查看。
<b>实例</b>	<code>Serial.print(78)</code> : 传输“78”，串口监视器显示 78 <code>Serial.print(1.234)</code> : 传输“1.23”，串口监视器显示 1.23 <code>Serial.print('N')</code> : 传输“N”，串口监视器显示 N <code>Serial.print("Hello world.")</code> , 传输“Hello world.”，串口显示 Hello world. (对于实际 RX 和 TX 线上的数据可通过示波器等仪器查看)

表 4-11 函数 print()

<b>语法</b>	<code>Serial.print(val, format)</code>
<b>参数</b>	<code>val</code> : 需要打印（传输）的值 <code>format</code> : 数据传输格式
<b>返回值</b>	返回当前传输的字节数
<b>说明</b>	将数据以 <code>format</code> 指定格式转换，再以 ASCII 码传输显示，仅对数值类型有效。对于整数，可选择 BIN (二进制)、OCT (八进制)、DEC (十进制)、HEX (十六进制)；对于浮点数类型则表示小数点个数。
<b>实例</b>	<code>Serial.print(78, BIN)</code> : 传输“1001110”，串口监视器显示 1001110 <code>Serial.print(78, OCT)</code> : 传输“116”，串口监视器显示 116 <code>Serial.print(78, DEC)</code> : 传输“78”，串口监视器显示 78 <code>Serial.print(78, HEX)</code> : 传输“4E”，串口监视器显示 4E <code>Serial.print(1.23456, 0)</code> : 传输“1”，串口监视器显示 1 <code>Serial.print(1.23456, 2)</code> : 传输“2”，串口监视器显示 1.23

表 4-12 函数 println()

<b>语法</b>	<code>Serial.println(val) / Serial.println(val, format)</code>
<b>参数</b>	含义同 <code>print()</code> 函数
<b>返回值</b>	含义同 <code>print()</code> 函数
<b>说明</b>	除传输结果待 “\r\n” 外，其余等同于 <code>print()</code> 函数
<b>实例</b>	使用方式同 <code>print()</code> 函数

表 4-13 函数 read()

语法	<code>Serial.read()</code>
参数	无
返回值	串口接收缓存区中首个数据，取出后该数据在缓存区中删除。缓存区中没有数据时返回-1
说明	读取串口接收缓存区中的首个数据，取出后该数据在缓存区中删除，下一个数据程序缓存区的首个数据以便下一次读取。可配合 <code>available()</code> 函数使用
实例	<pre>if(Serial.available() &gt; 0) {     int value = Serial.read(); }</pre>

#### 4.3.3 Lab-06 Hello World

##### 4.3.3.1 实验准备

实验通过 UART 模块打印“Hello World”至 PC，通过 PC 上的串口监视器显示。CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，并通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80MHz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。选择结果显示可见图 4-1。

##### 4.3.3.2 代码解释

程序首先在 `setup()` 函数上初始化 UART 模块，初始化函数为 `Serial.begin(115200)`，波特率设置为 115200。函数简介见表 4-8。

`Serial.begin(115200);`

在 `loop()` 函数上每隔一定时间打印“Hello World”信息。

`Serial.println("Hello World");`  
`delay(1000);`

##### 4.3.3.3 完整代码和流程图

完整代码：

```
void setup() {
// put your setup code here, to run once:
  Serial.begin(115200);
}
```

```

void loop() {
// put your main code here, to run repeatedly:
Serial.println("Hello World");
delay(1000);
}

```

程序流程图：

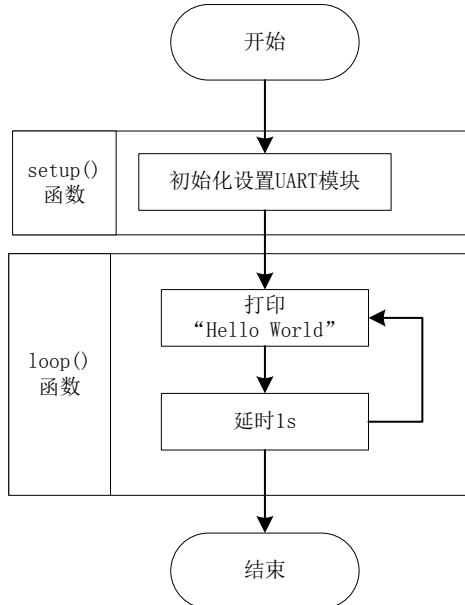


图 4-13 “Hello World”程序流程图

#### 4.3.3.4 实现现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，点击  打开串口监视器可观察到“Hello World”信息。



图 4-14 串口监视器试验现象

#### 4.3.4 Lab-07 Echo

##### 4.3.4.1 实验准备

PC 端通过串口监视器发送字符串至 CC3200 LaunchPad，LaunchPad 接收到完整数据后将回发数据至 PC 端，实现 echo 功能。

CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，并通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80Mhz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。选择结果显示可见图 4-1。

##### 4.3.4.2 代码解释

代码选择 Energia 提供的例程，其可以通过“文件->示例->04.Communication->SerialEvent”打开。

程序首先定义了两个变量：inputString 和 StringComplete。

inputString：保存接收的数据；

stringComplete：判断数据是否接收完整。

```
String inputString = "";// a string to hold incoming data
boolean stringComplete = false;// whether the string is complete
```

setup() 函数初始化 UART 模块并初始设置变量 inputString 存储空间，代码如下：

```
// initialize serial:
Serial.begin(9600);
// reserve 200 bytes for the inputString:
inputString.reserve(200);
```

loop() 函数中判断数据接收完成后就回发接受到的数据，发送完成后清除接收缓存并将 stringComplete 设置为 false，保证程序正常执行。代码如下：

```
// print the string when a newline arrives:
if(stringComplete){
    Serial.println(inputString);
// clear the string:
    inputString = "";
    stringComplete = false;
}
```

程序最关键的代码为函数 serialEvent()，该函数完成串口数据的接收。serialEvent() 函数会在两次 loop() 之间运行，即运行时序类似 loop()->serialEvent()->loop()，该时序的实现代码可见“3.3 最小代码”章节中 main() 函数介绍。

SerialEvent() 函数首先判断串口缓存区中是否存在有效数据，然后将有效数据读取出并存入 inputString 中，当接收到换行符（'\n'）表示此处数据接收完成。代码实现如下：

```
while(Serial.available()) {
// get the new byte:
```

```
char inChar = (char)Serial.read();
// add it to the inputString:
    inputString += inChar;
// if the incoming character is a newline, set a flag
// so the main loop can do something about it:
if(inChar == '\n'){
    stringComplete = true;
}
}
```

#### 4.3.4.3 完整代码和流程图

完整代码:

```
/*
  Serial Event example
  When new serial data arrives, this sketch adds it to a String.
  When a newline is received, the loop prints the string and
  clears it.

String inputString = "";// a string to hold incoming data
boolean stringComplete = false;// whether the string is complete
void setup(){
// initialize serial:
  Serial.begin(9600);
// reserve 200 bytes for the inputString:
  inputString.reserve(200);
}
void loop(){
// print the string when a newline arrives:
if(stringComplete){
  Serial.println(inputString);
// clear the string:
  inputString = "";
  stringComplete = false;
}
}
/*
  SerialEvent occurs whenever a new data comes in the
  hardware serial RX. This routine is run between each
  time loop() runs, so using delay inside loop can delay
  response. Multiple bytes of data may be available.
*/
```

```
void serialEvent() {
  while(Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    inputString += inChar;
    // if the incoming character is a newline, set a flag
    // so the main loop can do something about it:
    if(inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

程序流程图

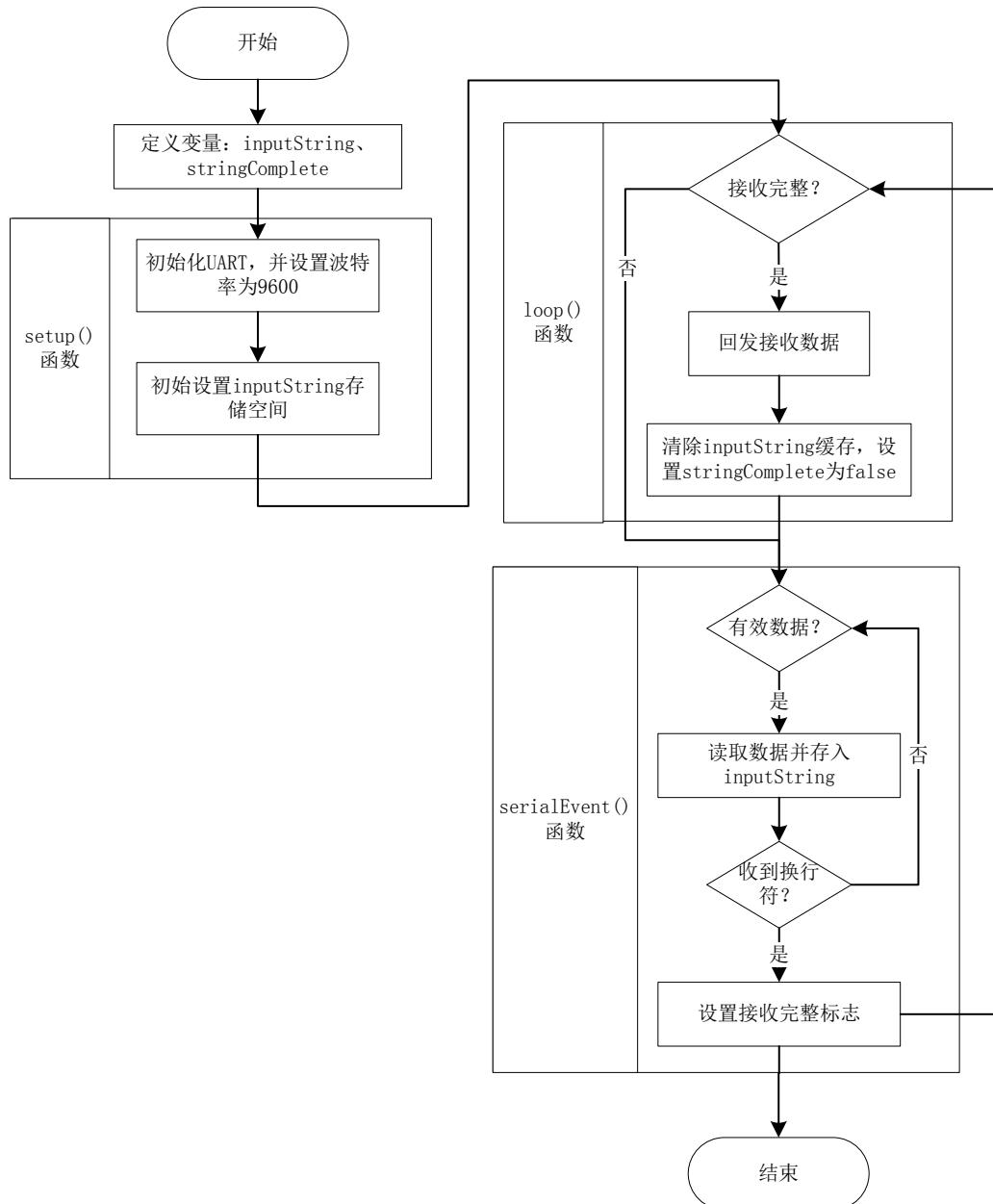


图 4-15 程序流程图

#### 4.3.4.4 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，点击  打开串口监视器，设置波特率为 9600。实验现象如下：

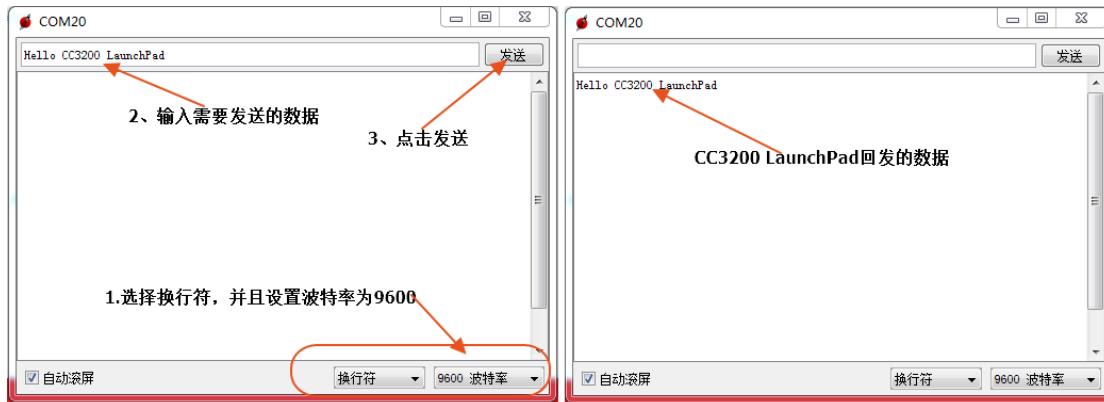


图 4-16 echo 例程实验现象

## 4.4 ADC12

### 4.4.1 概要

本章节具体介绍 Energia 平台下 ADC12 模块的使用。CC3200 ADC 的输入电压范围为 0~1.46V（如图 4-17 所示，也可在 <http://energia.nu/reference/analogwrite/> 找到“CAUTION”段落查看），在使用过程中需要特别注意。

**CAUTION – Analog input is limited to 1.46V on the CC3200. Higher voltages may damage the MCU.** — Reference: §3.2 Drive Strength and Reset States for Analog-Digital Multiplexed Pins, CC3200 SimpleLink™ Wi-Fi® and Internet-of-Things Solution, a Single-Chip Wireless MCU (swaso32f).

**ADC模块最多输入电压限制为1.46V**  
 下划线所示的文档可以查看**ADC**模块电压限制范围

图 4-17 ADC 电压范围限制

打开图 4-17 中提示的文档，找到 3.2 节 “Drive Strength and Reset States for Analog Digital Multiplexed pins”，在“Table 3-4”中可以找到提示信息，如图 4-18 所示。

Table 3-4. Drive Strength and Reset States for Analog-Digital Multiplexed Pins

Pin	Board Level Configuration and Use	Default State at First Power Up or Forced Reset	State after Configuration of Analog Switches (ACTIVE, LPDS, and HIB Power Modes)	Maximum Effective Drive Strength (mA)
29	Connected to the enable pin of the RF switch (ANTSEL1). Other use not recommended.	Analog is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4
30	Connected to the enable pin of the RF switch (ANTSEL2). Other use not recommended.	Analog is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4
45	VDD_ANA2 (pin 47) must be shorted to the input supply rail. Otherwise, the pin is driven by the ANA2 DC-DC.	Analog is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4
50	Generic I/O	Analog is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4
52	The pin must have an external pullup of 100 K to the supply rail and must be used in output signals only.	Analog is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4
53	Generic I/O	Analog is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4
57	Analog signal (1.8 V absolute, 1.46 V full scale)	ADC is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4
58	Analog signal (1.8 V absolute, 1.46 V full scale)	ADC is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4
59	Analog signal (1.8 V absolute, 1.46 V full scale)	ADC is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4
60	Analog signal (1.8 V absolute, 1.46 V full scale)	ADC is isolated. The digital I/O cell is also isolated.	Determined by the I/O state, as are other digital I/Os.	4

图 4-18 ADC full scale

在 Energia 官网上提供的 LaunchPad with CC3200 的 Pin Map 图中也可以发现，CC3200 ADC 模块的输入电压范围较小。如图 4-19 所示

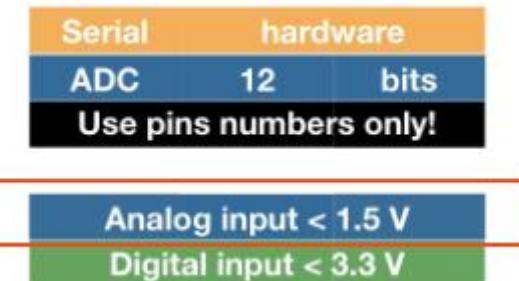


图 4-19 CC3200 模拟输入电压范围

除需要注意输入电压范围之外，ADC12 在 Energia 平台下进行开发使用非常方便。下面将通过几个实验例程来介绍在 Energia 平台下进行 ADC12 功能模块的开发使用。实验例程将涉及 AY-IOT Kit，以到达更加直观的实验效果。

#### 4.4.2 Lab-08 悬空端口采样

##### 4.4.2.1 实验准备

实验完成悬空端口数据采集，并通过串口打印采样值。该例程无任何实际意义，因为悬空的端口所采集的数据为无效数据。提供该试验例程主要是为了说明 ADC12 功能模块的编程方式。

CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接

至 PC。选择开发板为“CC3200-LAUNCHXL(80MHz)”,端口根据图 2-15 进行选择,即选择“COM20”(实际端口需用户查看设备管理器确定)。

#### 4.4.2.2 端口映射

CC3200 具有 4 个 12-bit 的 ADC 通道, Energia 对外开放 4 个端口作为 ADC 功能。如图



图 4-20 AD 端口

同时也可以在源代码中查看 AD 端口的定义。源代码位于:【C:\Users\用户名 (PC 用户名, 根据用户实而不同)\AppData\Local\Energia15\packages\energia\hardware\cc3200\1.0.2\variants\CC3200-LAUNCHXL】. 定义如下:

```
staticconstint8_t A0 =23;
staticconstint8_t A1 =2;
staticconstint8_t A2 =6;
staticconstint8_t A3 =24;
```

#### 4.4.2.3 代码解释

程序使用 A1 端口采集数据, 同时将采集到的数据通过串口打印出来。程序首先定义采集端口 analogInPin 变量, 并赋值为 A1。如果后续需要修改其它 AD 端口则只需修改该变量值即可。变量定义如下:

```
int analogInPin = A1;
```

setup()函数中仅需要对串口进行初始化设置即可, ADC 模块无需提前进行初始化设置。  
setup()函数实现如下:

```
void setup () {
// put your setup code here, to run once:
Serial.begin(9600);
}
```

loop()函数中首先读取 analogInPin 端口电压, 然后将采集到的数据发送至 PC 串口监视器进行显示。适当延时后重复上述操作。loop()内部实现如下:

```
int val = analogRead(analogInPin);
```

```

Serial.print("inputValue = ");
Serial.println(val);
delay(1000);

```

程序使用到函数 `analogRead()`, 函数原型如下:

表 4-14 函数 `analogRead()`

语法	<code>analogRead(pin)</code>
参数	<code>pin</code> : 模拟输入端口
返回值	电压值转变后的数据, 返回 0~4095, 对应实际电压为 0~1.46V, 实际电压范围查看图 4-17。
说明	采集 <code>pin</code> 所指向端口的电压数据, 采集到数据后需要将该值转变为实际可用数据
实例	<code>int value = analogRead(A1)</code>

#### 4.4.2.4 完整代码和流程图

##### 完整代码

```

int analogInPin = A1;
void setup() {
// put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
// put your main code here, to run repeatedly:
  int val = analogRead(analogInPin);
  Serial.print("inputValue = ");
  Serial.println(val);
  delay(1000);
}

```

流程图:

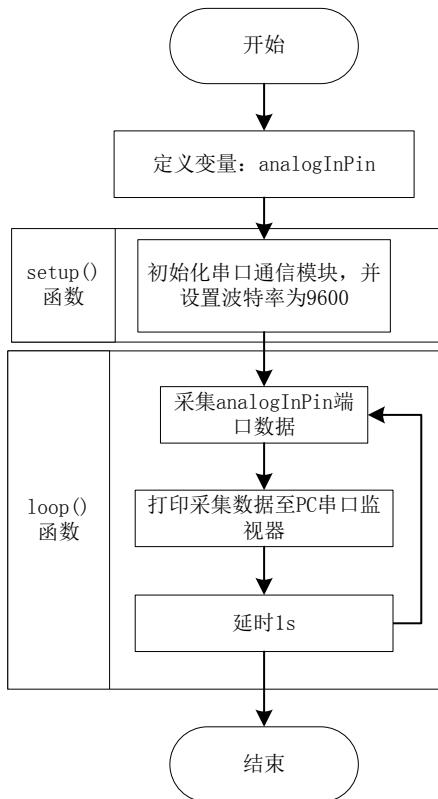


图 4-21 lab-08 程序流程图

#### 4.4.2.5 实验现象

点击 ，程序没有错误后即可点击 进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，点击 打开串口监视器，设置波特率为 9600。实验现象如下：



图 4-22 lab-08 实验现象

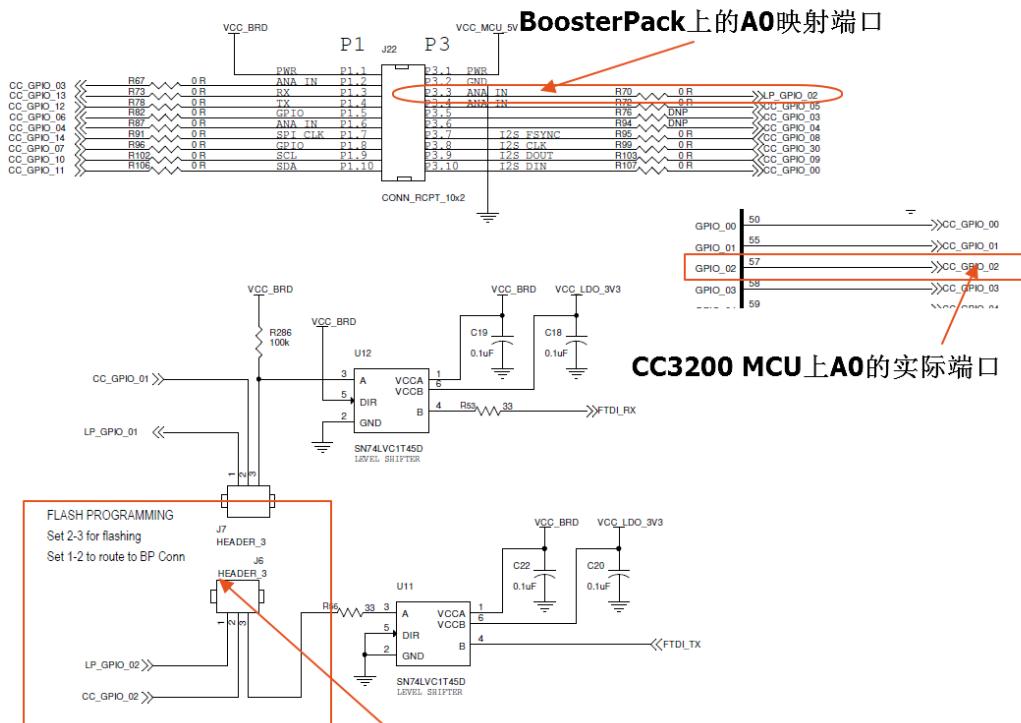
如果用户将程序中的变量 `analogInPin` 的值改成其余三个端口：A0、A2、A3，可以发现 A2 和 A3 的实验结果和 A1 相同，而 A0 的实验结果与其它三个端口的结果不相同。选择 A0

端口后的实验现象如下：



图 4-23 选择 A0 的实验结果

需要知道如图 4-23 所展示现象的原因就得看 CC3200 LaunchPad 的原理图。原理图部分内容如下(根据图 4-20 可知 A0 端口为 P3\_23 端口, 即 P3 排针的第三个端口, 观察 LaunchPad 可知该端口为 P57):



通过跳线帽选择 **GPIO\_02** (A0 的实际 MCU 端口) 功能  
**LaunchPad** 实际连接 2-3 跳线帽, 即 **CC\_GPIO\_02** 未  
 连接至 **LP\_GPIO\_02** (**BoosterPack** 对应端口) 上

图 4-24 A0 端口所在原理图

根据图 4-24 可知, A0 端口所对应的 MCU 上的 CC\_GPIO\_02 端口实际连接至 U11 (SN74LVC1T45D) A 端口 (J6 处, 图 4-25), 该处原理图与 USB 板载仿真器以及虚拟串口有

关。对应此部分的原理介绍不在本手册范围内。建议在实际使用中减少对 A0 端口的操作。

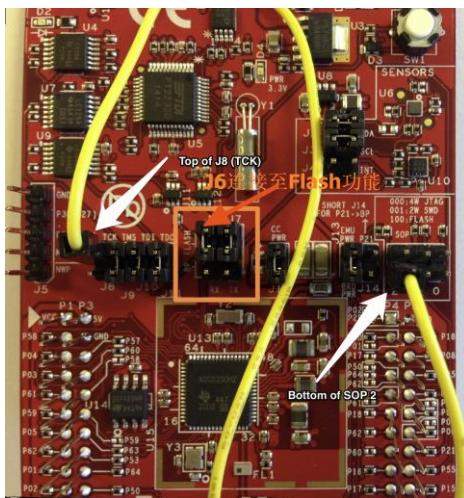


图 4-25 J6 连接 Flash 端

#### 4.4.3 Lab-09 LMT84 模块采集温度

##### 4.4.3.1 实验准备

实验实现 LMT84 模块的温度采集功能。LMT84 是一个温度采集传感器，其输出值为当前温度对应的电压值。实验完成对该电压值的采集然后根据芯片手册给出的计算公式实现对电压数据和温度数据的转换。LMT84 模块为 AY-IOT Kit 配套模块。该实验可加深对 CC3200 的 ADC 功能模块的应用，因为“4.4.2Lab-08 悬空端口采样”例程采集到的数据没有实际应用价值。

LMT84 为 AY-IOT Kit 配套模块，其连接方式可以下载艾研官方网站提供的文档《AY-IOT KIT 连接使用须知》，网址：[http://www.hpati.com/ay\\_doc\\_download/](http://www.hpati.com/ay_doc_download/)。

物联网实验套件AY-IOT KIT使用指南	3 MB	2017-03-20	公开	<a href="#">下载</a>
AY-IOT KIT连接使用须知	2 MB	2017-02-21	公开	<a href="#">下载</a>
AY-IOT KIT原理图	231 KB	2017-02-21	公开	<a href="#">下载</a>

图 4-26 AY-IOT KIT 连接使用须知

实物连接方式如下：

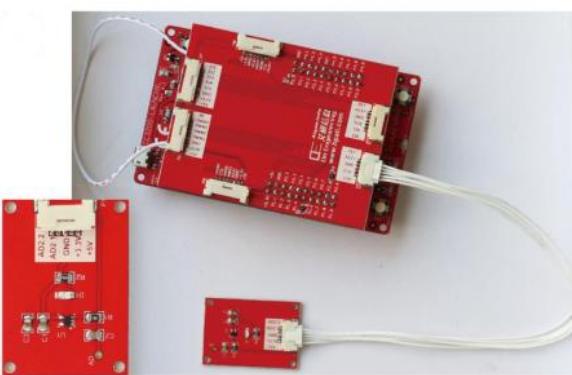


图 4-27 LMT84 连接方式

实物连接完成后，CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80Mhz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。

#### 4.4.3.2 端口映射

图 4-28 所示为 LMT84 的原理图，其使用 A1 作为 ADC 采样端口，实现温度数据采集。原理图中的 A1 即实际对应图 4-20 所指的 A1 端口。

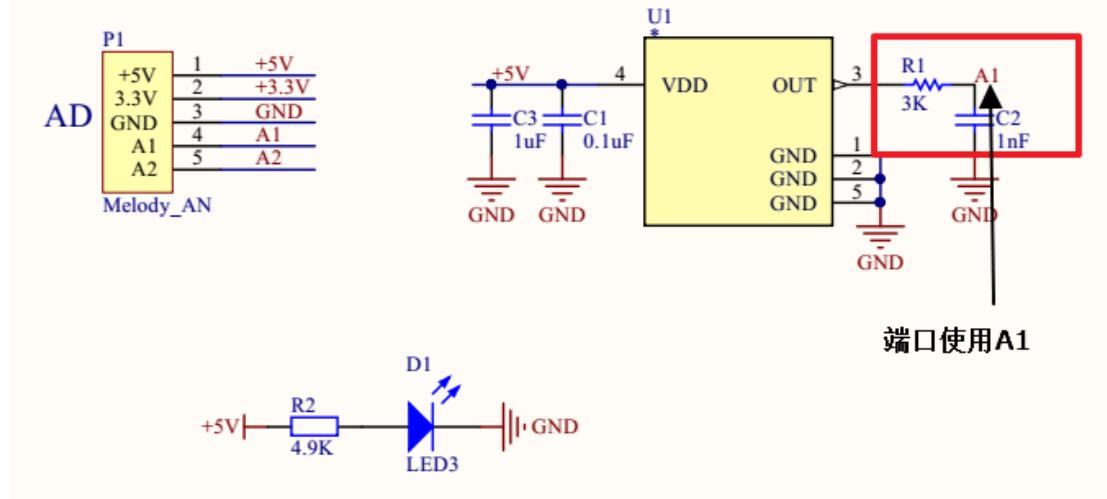


图 4-28 LMT84 原理图

#### 4.4.3.3 代码解释

LMT84 存在温度测量范围，故程序首先定义了两个常量：minTemperature、maxTemperature，温度测量范围即为[minTemperature, maxTemperature]。同时定义采集端口 temperaturePin 为常量，并且初始化为 A1（LMT84 采用 A1 作为采集端口）。常量定义如下：

```
const int16_t minTemperature = -50;
const int16_t maxTemperature = 150;
const int temperaturePin = A1;
```

setup() 函数中仅需要对串口进行初始化设置即可，LMT84 模块无需提前进行初始化设置，因为其使用的是 ADC 功能，ADC 功能在 Energia 平台下无需提前初始化，只需要在使用时直接调用 analogRead() 函数即可。setup() 函数实现如下：

```
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
}
```

loop() 函数中循环读取 temperaturePin 指向端口的电压值，并通过 calcTemperature() 函数

进行电压和温度之间的换算，同时将温度值打印至 PC 串口监视器。loop()函数内部实现如下：

```
uint16_t value = analogRead(temperaturePin);
Serial.println(calcTemperature(value));
delay(1000);
```

其中 calcTemperature()函数具体实现可见 4.4.3.4 章节中的完整代码，函数简介如下。

表 4-15 函数 calcTemperature()

语法	<b>calcTemperature(sample)</b>
参数	<b>sample:</b> AD 采样值，由函数 analogRead()返回
返回值	sample 采样值对应的实际温度值，计算公式需要查看 LMT84 的 datasheet
说明	计算输入参数 sample 对应的温度值
实例	<code>float temperature = calcTemperature(analogRead(temperaturePin))</code>

在函数实现中最关键部分为电压数据转换成温度数据，在 LMT84 的 datasheet 中给出了多种转换意见：查表和线性拟合公式。查表：LMT84 手册给出了从  $-50^{\circ}\text{C} \sim 150^{\circ}\text{C}$  之间，分度为  $1^{\circ}\text{C}$  的输出电压值。线性拟合公式：LMT84 手册给出了两个公式：

公式一：

$$T = \frac{5.506 - \sqrt{(-5.506)^2 + 4 \times 0.00176 \times (870.6 - V_{\text{TEMP}}(\text{mV}))}}{2 \times (-0.00176)} + 30$$

公式二：

$$V - V_1 = \left( \frac{V_2 - V_1}{T_2 - T_1} \right) \times (T - T_1)$$

因公式二在实现时需要根据查表数据获取  $(V_1, T_1)$  和  $(V_2, T_2)$ ，而公式一在实现上简单方便，所以例程选择了公式一计算温度值。如果用户需要使用公式二实现，可根据以下示例实现，该示例可在 LMT84 手册中找到。

示例：假设例程检测的环境温度范围为  $20^{\circ}\text{C} \sim 50^{\circ}\text{C}$ ，根据查表可知此时对应的电压输出值为  $925\text{mV} \sim 760\text{mV}$ ，即此时  $(V_2, T_2) = (760\text{mV}, 50^{\circ}\text{C})$ ， $(V_1, T_1) = (925\text{mV}, 20^{\circ}\text{C})$ ，根据“公式二”可得

$$V - 925\text{mV} = \left( \frac{760\text{mV} - 925\text{mV}}{50^{\circ}\text{C} - 20^{\circ}\text{C}} \right) \times (T - 20^{\circ}\text{C})$$

$$V - 925\text{mV} = (-5.5\text{mV} / ^{\circ}\text{C}) \times (T - 20^{\circ}\text{C})$$

$$V = (-5.5\text{mV} / ^{\circ}\text{C}) \times T + 1035\text{mV}$$

故温度为

$$T = \frac{V - 1035\text{mV}}{(-5.5\text{mV} / ^{\circ}\text{C})}$$

当然该公式仅适合温度范围为  $20^{\circ}\text{C} \sim 50^{\circ}\text{C}$ 。其余温度范围按照该示例推导即可。

#### 4.4.3.4 完整代码和流程

完整代码：

```
const int16_t minTemperature = -50;
```

```

constint16_t maxTemperature =150;
constint temperaturePin = A1;
float calcTemperature(uint16_t sample);

void setup()
{
// put your setup code here, to run once:
    Serial.begin(115200);
}

void loop()
{
// put your main code here, to run repeatedly:
uint16_t value = analogRead(temperaturePin);
    Serial.print("temperature = ");
    Serial.println(calcTemperature(value));
    delay(1000);
}

float calcTemperature(uint16_t sample)
{
uint16_t voltage = map(sample,0,4095,0,1460);
float temperature =0.0;
    temperature =(5.506- sqrt(pow(-5.506,2)+4*0.00176*(870.6-
voltage)))/(2*(-0.00176))+30.0;

    if(temperature > maxTemperature)
    {
        return maxTemperature;
    }
    elseif(temperature < minTemperature)
    {
        return minTemperature;
    }
    else
    {
        return temperature;
    }
}

```

在 calcTemperature() 函数实现中使用了 map() 函数，将采集到的 sample 值根据相关比例转换到实际电压值。转换比例由该函数第 2~第 5 参数确定：即处于 0~4095 之间是 sample 值映射到 0~1460 之间的 voltage 值。函数简介如下：

表 4-16 函数 map()

原型	long map(long x, long in_min, long in_max, long out_min, long out_max)
----	--

参数	x: 需要被映射到[out_min, out_max]的数据, 当前所在范围[in_min, in_max]; in_min: 参数 x 所在当前范围的最小值; in_max: 参数 x 所在当前范围的最大值; out_min: 参数 x 被映射范围的最小值; out_max: 参数 x 被映射范围的最大值;
返回值	参数 x 被映射到[out_min, out_max]的值
说明	将参数 x 从[in_min, in_max]范围映射到[out_min, out_max]范围内
实例	voltage = map(sample, 0, 4095, 0, 1460)

流程图:

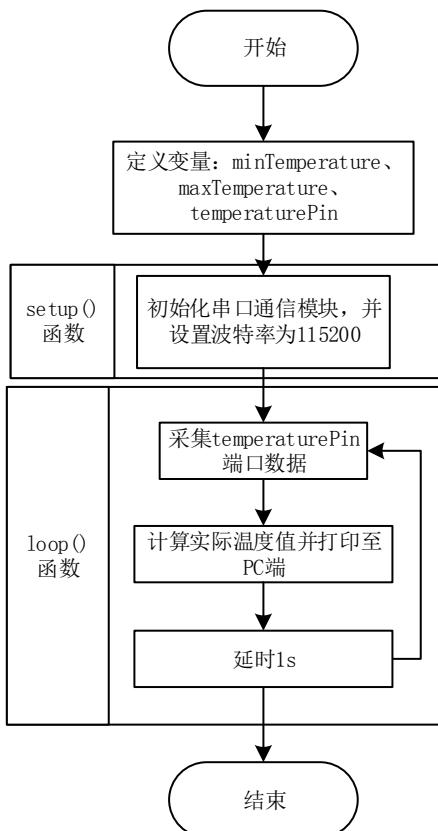


图 4-29 lab-09 流程图

#### 4.4.3.5 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，点击  打开串口监视器，设置波特率为 115200。实验现象如下

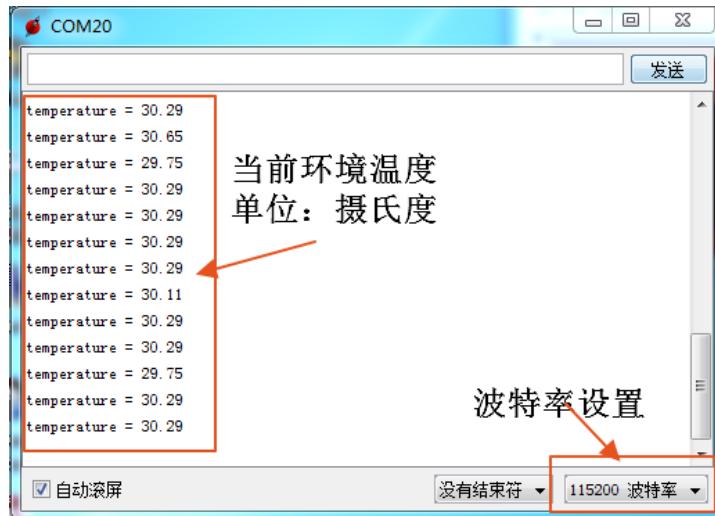


图 4-30 LMT84 模块采集温度实验现象

根据实验可知当前环境温度在  $20^{\circ}\text{C} \sim 50^{\circ}\text{C}$  之间，可以使用示例中“公式二”推导出的公式进行计算。替换例程 `calcTemperature()` 函数中的计算公式。

公式替换为：

```
temperature = 1.0 * (voltage - 1035) / (-5.5);
```

实验结果为：



图 4-31 使用“公式二”实验现象

对比图 4-30 和图 4-31 实验现象，两个公式的计算结果相差不大。对于使用“公式二”的实验，用户可以根据字节兴趣选择其他温度区间进行公式推导和计算，但需要保证实际温度需要在选择的温度区间范围内。

## 4.5 PWM

### 4.5.1 概要

本章节具体介绍 Energia 环境下 PWM 功能的使用。首先根据 Energia 提供的例程 Fading“”

说明 PWM 功能的编程实现，最后通过 AY-IOT Kit 中的高亮 LED 模块更加直观展示 PWM 的应用。

#### 4.5.2 lab-10 Fading

##### 4.5.2.1 实验准备

实验完成定时修改 PWM 的输出占空比。CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80Mhz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。

##### 4.5.2.2 端口映射

CC3200 LaunchPad 上存在多个具有模拟输出功能的 GPIO 口，观察图 2-1 端口定义图可知，图中“紫色”标记的端口具有模拟功能，所以 CC3200 LaunchPad 上的三个 LED 都可作为 PWM 输出端口，三个 LED 分别为连接至 P01、P02、P64 端口号，其直接对应 BoosterPack 中 P1\_9(YELLOW\_LED)、P1\_10(GREEN\_LED)、P3\_29(RED\_LED)。



图 4-32 LED 对应端口

##### 4.5.2.3 代码解释

代码例程选择 Energia 提供的 Fading 实例，其可通过“文件->示例->03.Analog->Fading”打开。完整代码看 4.5.2.4。

程序首先定义了 ledPin 作为 PWM 的输出端口，方便修改实际使用的 PWM 输出端口，如下

```
int ledPin = 9; // LED connected to digital pin 9
```

在 setup() 函数中，无需对 PWM 输出端口进行初始化设置。PWM 输出功能仅使用 analogWrite() 函数即可实现，故在 loop() 函数中调用该函数即可。analogWrite() 函数定义如表 4-17.

表 4-17 函数 analogWrite()

语法	analogWrite(pin, value)
参数	pin: PWM 输出端口 value: PWM 输出占空比对应值，取值范围 0 (0%) ~255 (100%)
返回值	无
说明	pin 端口输出占空比为 value/255 的 PWM
实例	analogWrite(9, 128);

loop()函数中实现了 ledPin 端口的逐步变量和变暗过程，代码实现可见 4.5.2.4。

#### 4.5.2.4 完整代码和流程图

完整代码：

```
/*
Fading
This example shows how to fade an LED using the analogWrite() function.

*/
int ledPin =9;// LED connected to digital pin 9

void setup(){
// nothing happens in setup
}

void loop(){
// fade in from min to max in increments of 5 points:
for(int fadeValue =0; fadeValue <=255; fadeValue +=5){
// sets the value (range from 0 to 255):
analogWrite(ledPin, fadeValue);
// wait for 30 milliseconds to see the dimming effect
delay(30);
}
// fade out from max to min in increments of 5 points:
for(int fadeValue =255; fadeValue >=0; fadeValue -=5){
// sets the value (range from 0 to 255):
analogWrite(ledPin, fadeValue);
// wait for 30 milliseconds to see the dimming effect
delay(30);
}
}
```

流程图：

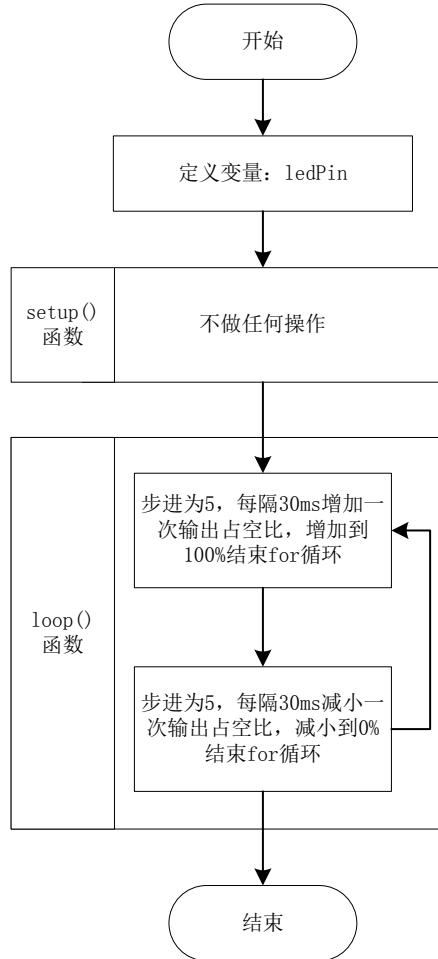


图 4-33 Lab-10 流程图

#### 4.5.2.5 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，LED 将实现逐步变亮再变暗的过程。

### 4.5.3 Lab-11 高亮 LED 模块

#### 4.5.3.1 实验准备

实验实现高亮 LED 模块亮度调节功能。高亮 LED 模块使用 PWM 驱动，改变 PWM 的输出占空比即可实现亮度调节。

高亮 LED 模块为 AY-IOT Kit 配套模块，其连接方式可以下载艾研官方网站提供的文档《AY-IOT KIT 连接使用须知》，网址：[http://www.hpati.com/ay\\_doc\\_download/](http://www.hpati.com/ay_doc_download/)

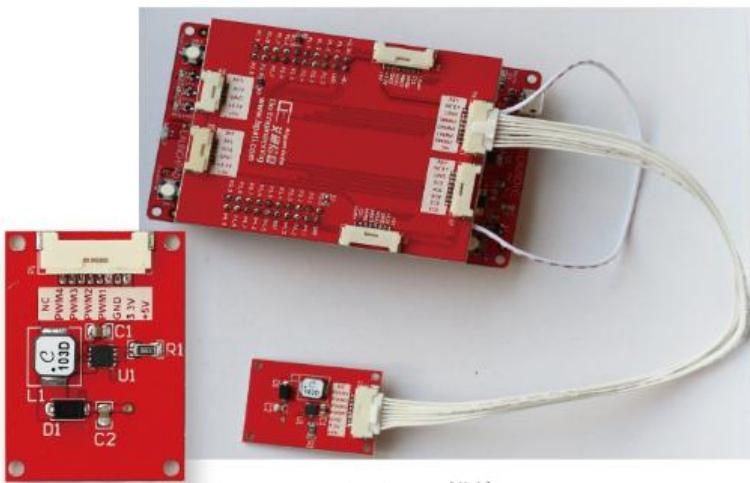


图 4-34 高亮 LED 模块连接方式

模块实物连接完成后，CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80MHz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。

#### 4.5.3.2 端口映射

根据高亮 LED 模块原理图可知，模块使用 PWM1 作为控制端口实现 LED 亮度调节。PWM 连接 P1 中 PWM1，查看 AY-IOT Kit 中的 Melody\_Hub 原理图可知 PWM1 连接至 CC3200 的 P4\_40 端口即 CC3200 BoosterPack 40 号端口。

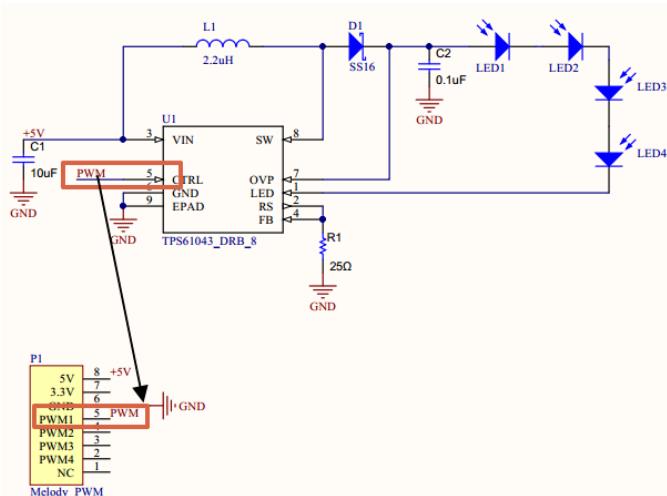


图 4-35 高亮 LED 原理图

查看 energia 下对 CC3200 BoosterPack 的定义可知，40 号端口实际程序库上对应为 P02 端口而非 Launchpad 硬件连接上的 P29 端口，所以需要将 CC3200 Launchpad 上 40 号端口的连接至 P29 处的电阻去除，然后将 P02 和 40 好端口短接或者使用 0 欧姆电阻连接（如果身边没有合适的工具进行焊接操作，实现现象可选择观察 GREEN\_LED 代替，因为 P02 还连接了该 LED）。程序定义如下：

```

PIN_17,      /* 31 - GPIO_24 */
PIN_16,      /* 32 - GPIO_23 */
PIN_60,      /* 33 - GPIO_05 */
PIN_62,      /* 34 - GPIO_07 */
PIN_18,      /* 35 - GPIO_28 */
PIN_21,      /* 36 - GPIO_25 */
PIN_64,      /* 37 - GPIO_09 */
PIN_17,      /* 38 - GPIO_24 */
PIN_01,      /* 39 - GPIO_10 */
PIN_02      /* 40 - GPIO_11 */

```

图 4-36 BoosterPack 40 端口定义

源代码如下：

```

const uint16_t digital_pin_to_pin_num[] = {
    NOT_A_PIN,/* dummy */
    NOT_A_PIN,/* 1 - 3.3V */
    PIN_58,/* 2 - GPIO_03 */
    PIN_04,/* 3 - GPIO_13 */
    PIN_03,/* 4 - GPIO_12 */
    PIN_61,/* 5 - GPIO_06 */
    PIN_59,/* 6 - GPIO_04 */
    PIN_05,/* 7 - GPIO_14 */
    PIN_62,/* 8 - GPIO_07 */
    PIN_01,/* 9 - GPIO_10 */
    PIN_02,/* 10 - GPIO_11 */
    PIN_15,/* 11 - GPIO_22 */
    PIN_55,/* 12 - GPIO_01 */
    PIN_21,/* 13 - GPIO_25 */
    PIN_06,/* 14 - GPIO_15 */
    PIN_07,/* 15 - GPIO_16 */
    NOT_A_PIN,/* 16 - RESET */
    PIN_45,/* 17 - GPIO_31 */
    PIN_08,/* 18 - GPIO_17 */
    PIN_18,/* 19 - GPIO_28 */
    NOT_A_PIN,/* 20 - GND */
    NOT_A_PIN,/* 21 - 5V */
    NOT_A_PIN,/* 22 - GND */
    PIN_57,/* 23 - GPIO_02 */
    PIN_60,/* 24 - GPIO_05 */
    PIN_58,/* 25 - GPIO_03 */
    PIN_59,/* 26 - GPIO_04 */
    PIN_63,/* 27 - GPIO_08 */
    PIN_53,/* 28 - GPIO_30 */
    PIN_64,/* 29 - GPIO_09 */
    PIN_50,/* 30 - GPIO_00 */
    PIN_17,/* 31 - GPIO_24 */

```

```
PIN_16, /* 32 - GPIO_23 */
PIN_60, /* 33 - GPIO_05 */
PIN_62, /* 34 - GPIO_07 */
PIN_18, /* 35 - GPIO_28 */
PIN_21, /* 36 - GPIO_25 */
PIN_64, /* 37 - GPIO_09 */
PIN_17, /* 38 - GPIO_24 */
PIN_01, /* 39 - GPIO_10 */
PIN_02      /* 40 - GPIO_11 */

};
```

#### 4.5.3.3 代码解释

实例通过按键控制高亮 LED 模块亮度，在 `setup()` 函数内设置按键对应端口为上拉输入，并且初始设置 PWM 输出为 0，同时开启串口通讯模块，以便用于观察当前输出值。实现如下：

```
Serial.begin(115200);
pinMode(buttonPin, INPUT_PULLUP);

analogWrite(ledPin,bright);
```

在 `loop()` 函数中首先读取按键端口的电平状态，识别到按键按下后进行延时消抖；之前采用 `while()` 循环识别按键是否已经释放；按键释放后改变 PWM 输出值，改变一个 `brightStep` 大小，同时通过串口打印至 PC 进行显示。实现见 4.5.3.4。

#### 4.5.3.4 完整代码和流程图

##### 完整代码

```
const uint8_t ledPin =40;
const uint8_t buttonPin = PUSH1;
const uint8_t brightStep =32;

uint8_t bright =0;
void setup()
{
// put your setup code here, to run once:
Serial.begin(115200);
pinMode(buttonPin, INPUT_PULLUP);

analogWrite(ledPin,bright);
}
```

```

void loop()
{
// put your main code here, to run repeatedly:

if(digitalRead(buttonPin))
{
    delay(20);
//waiting for push up
while(digitalRead(buttonPin));
    bright += brightStep;
    analogWrite(ledPin,bright);
    Serial.print("bright: ");
    Serial.println(bright);
}
}

```

流程图

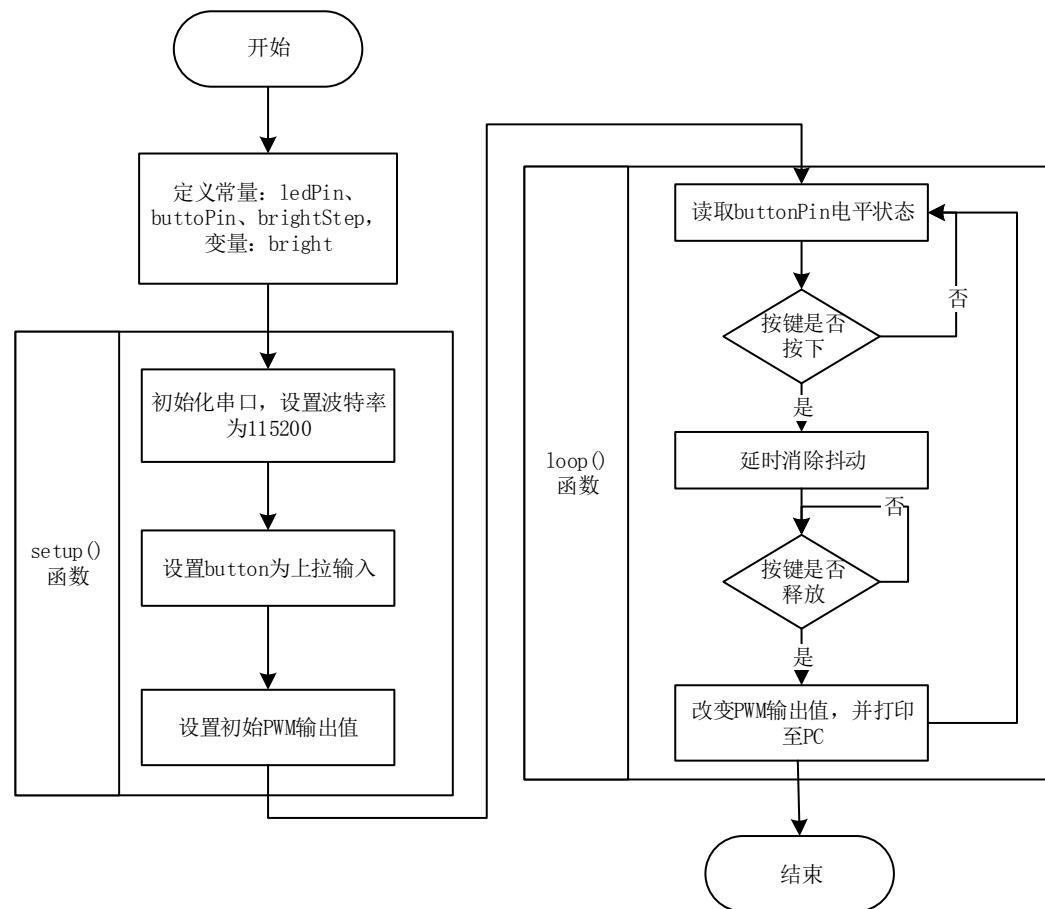


图 4-37 lab-11 流程图

#### 4.5.3.5 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，按下 PUSH1 (SW3) LED 亮度将不断变亮，达到最亮时继续按下按键 LED 将熄灭，因为程序中 “bright” 为 “uin8\_t” 类型，步进值 “brightStep” 为 32，恰好为 256 的约数，故当 “bright” 增大溢出时该值将变成 0。



图 4-38 Lab-09 高亮 LED 模块实验现象

例程使用 `analogWrite()` 函数输出 PWM，根据表 4-17 介绍可知，该函数的输出值为 0~255，并不是 0%~100% 的占空比数据。如果用户需要根据百分比数据进行输出则可根据“表 4-16 函数 `map()`”进行映射。映射代码实现如下

```
int output = map(duty, 0, 100, 0, 255);
```

`duty` 为 PWM 占空比，`output` 为 PWM 实际输出值，该值可作为 `analogWrite()` 函数的参数使用。

## 4.6 I2C

### 4.6.1 概要

I2C 设备之间通信仅需要两根数据线：SDA(Serial Data Line) 和 SCL(Serial Clock Line)。SDA 线作为数据通信线，可实现数据的发送和接收；SCL 线作为时钟信号线，实现 I2C 设备之间的时钟同步。

在 Energia 下进行 I2C 通信可以使用其提供的 `Wire` 库实现，库文件参考内容可见官网：<http://energia.nu/reference/wire/>。如果想要查看库文件源码可在 CC3200 LaunchPad 安装目录下找到：【C:\Users\用户名 (PC 用户名，根据用户实而不同)\AppData\Local\Energia15\packages\energia\hardware\cc3200\1.0.2\libraries\Wire】该目录下的 `Wire.h` 和 `Wire.cpp`。`Wire(I2C)` 库常用函数将在 4.6.2 章节介绍。

本章将通过三个实验例程来介绍 I2C 的应用，例程：数据发送基本过程、数据读取基本过程、读取 HDC1080 模块温湿度数据。

#### 4.6.2 Wire 库(I2C)

Wire 库（I2C）使用 C++ 实现，具体实现方式可查看 Wire.h 和 Wire.cpp 文件。Wire(I2C) 库存在多个成员函数，函数列表如下：

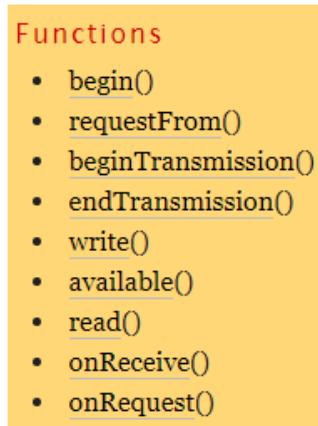


图 4-39 Wire (I2C) 库成员函数

下面就对 I2C 编程过程中经常使用的函数进行简单介绍。未说明的函数用户可在 Energia 官网上进行了解，地址：<http://energia.nu/reference/Wire/> 在网页左侧 Functions 栏里找到对应函数点击进入即可。

表 4-18 函数 begin()

语法	<code>Wire.begin()</code>
参数	无
返回值	无
说明	初始化 I2C 设备，以主设备或者从设备模式加入总线。
实例	<code>Wire.begin() / Wire.begin(0x07)</code>

表 4-19 函数 requestFrom()

语法	<code>Wire.requestFrom(address, quantity)</code>
参数	<b>address</b> : 7-bit 从设备地址; <b>quantity</b> : 请求的数据个数
返回值	从设备返回的数据个数
说明	主设备向从设备请求数据，请求数据完成后可以使用 <code>available()</code> 函数和 <code>read()</code> 函数取出数据。请求完成后产生一个“stop”信号，释放 I2C 总线
实例	<code>Wire.requestFrom(0x07, 2)</code> , 向设备地址为 0x06 的从设备上请求两个数据

表 4-20 函数 requestFrom()

语法	<code>Wire.requestFrom(address, quantity, stop)</code>
参数	<b>address</b> : 7-bit 从设备地址; <b>quantity</b> : 请求的数据个数 <b>stop</b> : boolean 类型, <b>true</b> : 请求完成后产生一个“stop”信号，释放 I2C 设备总线; <b>false</b> : 请求完成后产生一个“restart”信号，实现数据连续读取（实现 <code>continue</code> 传输模式）。

返回值	从设备返回的数据个数
说明	主设备向从设备请求数据，请求数据完成后可以使用 available() 函数和 read() 函数取出数据。
实例	Wire.requestFrom(0x07, 2, true), 向设备地址为 0x06 的从设备上请求两个数据，请求完成后释放 I2C 总线

表 4-21 函数 beginTransmission()

语法	Wire.beginTransmission( <b>address</b> )
参数	address: 7-bit 从设备地址
返回值	无
说明	主设备向从设备发起数据传输，从设备地址由参数 address 确定。完整传输时序为：beginTransmission()->write()->endTransmission()
实例	Wire.beginTransmission(0x07);

表 4-22 函数 endTransmission()

语法	Wire.endTransmission()
参数	无
返回值	返回传输状态: 0: success 1: 数据长度大于传输缓存 2: 主设备发送从设备地址时返回 NACK 信号 3: 主设备发送数据时返回 NACK 信号 4: 其他错误
说明	结束由主设备通过 beginTransmission() 发起的数据传输。完整数据传输时序：beginTransmission()->write()->endTransmission()。同时产生一个“stop”信号，释放 I2C 总线。
实例	Wire.endTransmission(): 结束传输同时释放 I2C 总线

表 4-23 函数 endTransmission()

语法	Wire.endTransmission( <b>stop</b> )
参数	stop: boolean 类型。true: 完成一次传输后，主设备产生一个“stop”信号，释放 I2C 总线；false: 完成一次传输后，主设备产生一个“restart”信号，保持 I2C 总线忙状态，实现数据连续发送（实现 continue 传输模式）。
返回值	返回传输状态: 0: success 1: 数据长度大于传输缓存 2: 主设备发送从设备地址时返回 NACK 信号 3: 主设备发送数据时返回 NACK 信号 4: 其他错误
说明	结束由主设备通过 beginTransmission() 发起的数据传输。完整数据传输时序：beginTransmission()->write()->endTransmission()。
实例	Wire.endTransmission(true): 结束传输同时释放 I2C 总线

表 4-24 函数 write()

<b>语法</b>	<b>Wire.write(value)</b>
<b>参数</b>	value: 发送单个字节数据
<b>返回值</b>	返回发送的字节个数
<b>说明</b>	主设备写数据至从设备，该函数需要在 beginTransmission()之后使用，写数据完整时序：beginTransmission()->write()->endTransmission()。
<b>实例</b>	Wire.write(0x5A)

表 4-25 函数 write()

<b>语法</b>	<b>Wire.write(string)</b>
<b>参数</b>	value: 发送字符串至从设备（将字符串看作是多个字节数据即可）
<b>返回值</b>	返回发送的字节个数
<b>说明</b>	主设备写数据至从设备，该函数需要在 beginTransmission()之后使用，写数据完整时序：beginTransmission()->write()->endTransmission()。
<b>实例</b>	Wire.write("1234")

表 4-26 函数 write()

<b>语法</b>	<b>Wire.write(data, length)</b>
<b>参数</b>	data: 待发送的数据缓存（数组） length: 缓存长度
<b>返回值</b>	返回发送的字节个数
<b>说明</b>	主设备写数据至从设备，该函数需要在 beginTransmission()之后使用，写数据完整时序：beginTransmission()->write()->endTransmission()。
<b>实例</b>	int data[2] = {1,2}; Wire.write(data, 2)

表 4-27 函数 available()

<b>语法</b>	<b>Wire.available()</b>
<b>参数</b>	无
<b>返回值</b>	返回有效可读字节数
<b>说明</b>	函数返回已经存储在 I2C 接收缓存中的数据个数，该函数可在进行 read() 操作前使用。
<b>实例</b>	if(Wire.available() > 0) { //put your code in here, for example: read() Wire.read() }

表 4-28 函数 read()

<b>语法</b>	<b>Wire.read()</b>
<b>参数</b>	无
<b>返回值</b>	I2C 接收缓存区中的首个数据

说明	读取 I2C 接收缓存区中的首个数据，取出后该数据在缓存区中删除，下一个数据程序缓存区的首个数据以便下一次读取。可配合 available()函数使用
实例	<pre>if(Wire.available()&gt; 0) { //put your code in here, for example: read()     int data = Wire.read() }</pre>

#### 4.6.3 Lab-12 数据发送基本过程

本章节使用 Energia 平台下的例程：master\_writer。该例程可通过“文件->示例->Wire->master\_writer”打开。该例程没有合适的 slave 设备进行通信，所以此处仅介绍数据发送基本过程的代码实现，无需进行实验验证。

##### 4.6.3.1 代码解释

I2C 数据发送时序逻辑是程序实现过程中最关键的部分，完成该时序的编程就实现 I2C 通信中数据发送环节。I2C 数据发送时序逻辑如错误！未找到引用源。。

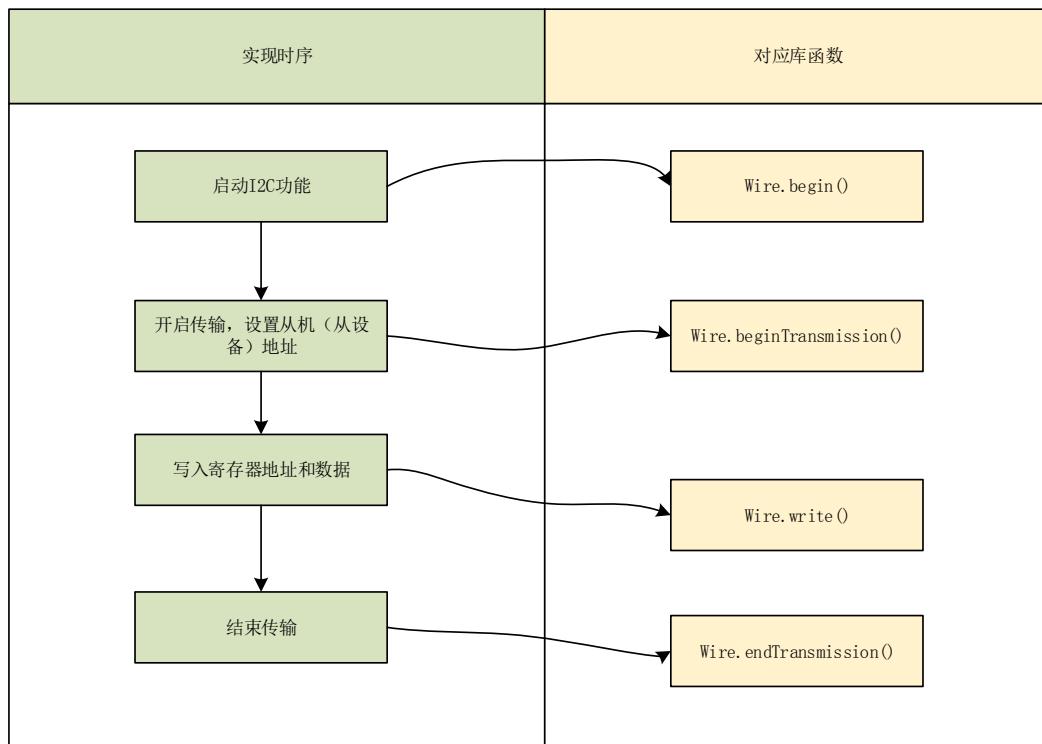


图 4-40 I2C 库函数使用——数据发送基本过程时序展示

其中 Wire.begin() 函数在 setup() 中调用，完成初始化 I2C 功能模块，而 loop() 函数内需要完成 I2C 通信时序，根据和对应库函数所示，代码实现如下。完整例程请查看下一节“4.6.3.2 完整代码和流程图”。

```
Wire.beginTransmission(4); // transmit to device #4
Wire.write("x is "); // sends five bytes
Wire.write(x); // sends one byte
Wire.endTransmission(); // stop transmitting
```

#### 4.6.3.2 完整代码和流程图

完整代码:

```
#include <Wire.h>

void setup()
{
    Wire.begin(); // join i2c bus (address optional for master)
}

byte x = 0;

void loop()
{
    Wire.beginTransmission(4); // transmit to device #4
    Wire.write("x is "); // sends five bytes
    Wire.write(x); // sends one byte
    Wire.endTransmission(); // stop transmitting

    x++;
    delay(500);
}
```

流程图

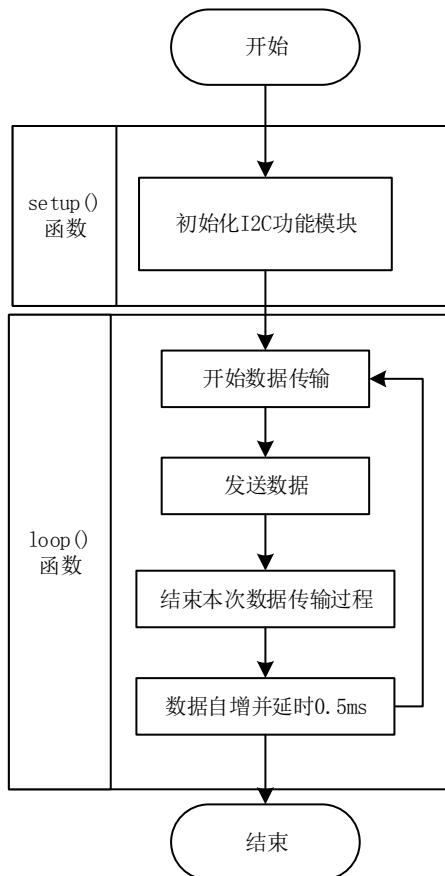


图 4-41 Lab-12 流程图

#### 4.6.4 Lab-13 数据读取基本过程

本章节使用 Energia 平台下的例程：master\_reader。该例程可通过“文件->示例->Wire->master\_reader”打开。该例程没有合适的 slave 设备进行通信，所以此处仅介绍数据读取基本过程的代码实现，无需进行实验验证。

##### 4.6.4.1 代码解释

I2C 数据读取时序逻辑是程序实现过程中最关键的部分，完成该时序的编程就实现 I2C 通信中数据读取环节。I2C 数据读取时序逻辑如图 4-42。

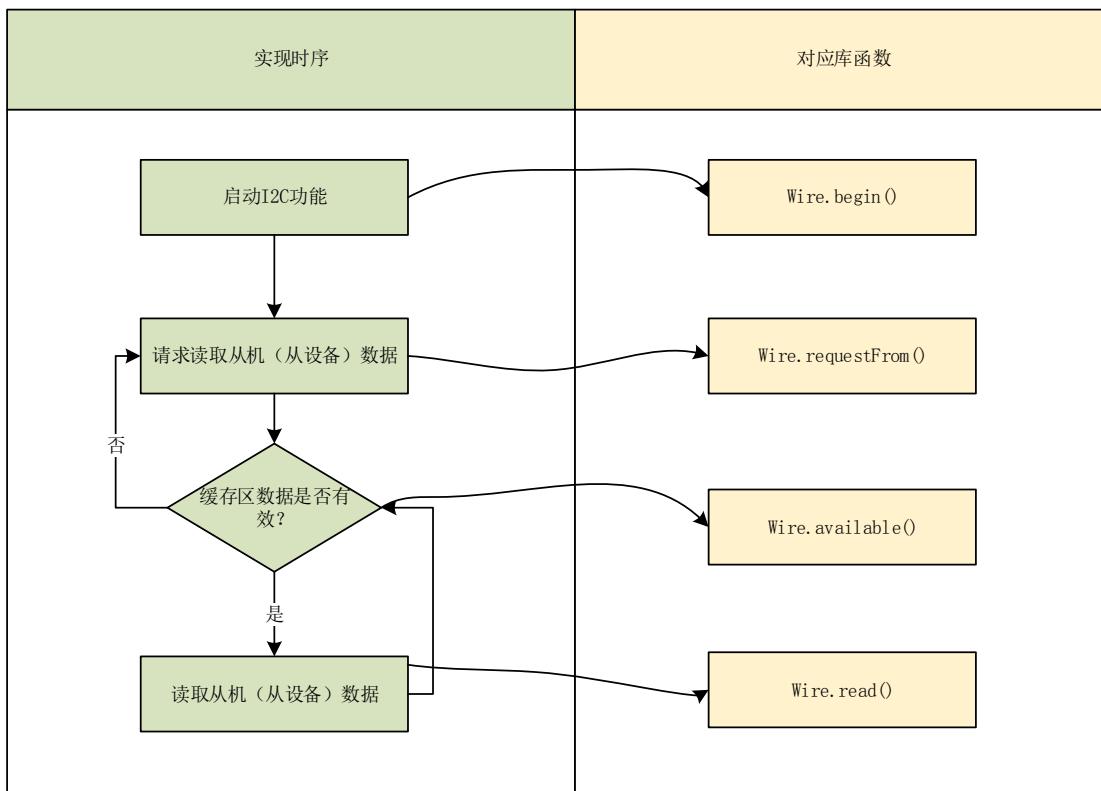


图 4-42 I2C 库函数使用——数据读取基本过程时序展示

其中 `Wire.begin()` 函数在 `setup()` 中调用，完成初始化 I2C 功能模块，而 `loop()` 函数内需要完成 I2C 通信时序，根据图 4-42 对应库函数所示，代码实现如下。完整例程请查看下一节“4.6.4.2 完整代码和流程图”。

```

Wire.requestFrom(2, 6); // request 6 bytes from slave device #2

while(Wire.available()) // slave may send less than requested
{
    char c = Wire.read(); // receive a byte as character
}

```

#### 4.6.4.2 完整代码和流程图

完整代码：

```

#include <Wire.h>

void setup()
{
    Wire.begin(); // join i2c bus (address optional for master)
    Serial.begin(9600); // start serial for output
}

void loop()

```

```

{
    Wire.requestFrom(2, 6); // request 6 bytes from slave device #2
    while(Wire.available()) // slave may send less than requested
    {
        char c = Wire.read(); // receive a byte as character
        Serial.print(c); // print the character
    }
    delay(500);
}

```

流程图：

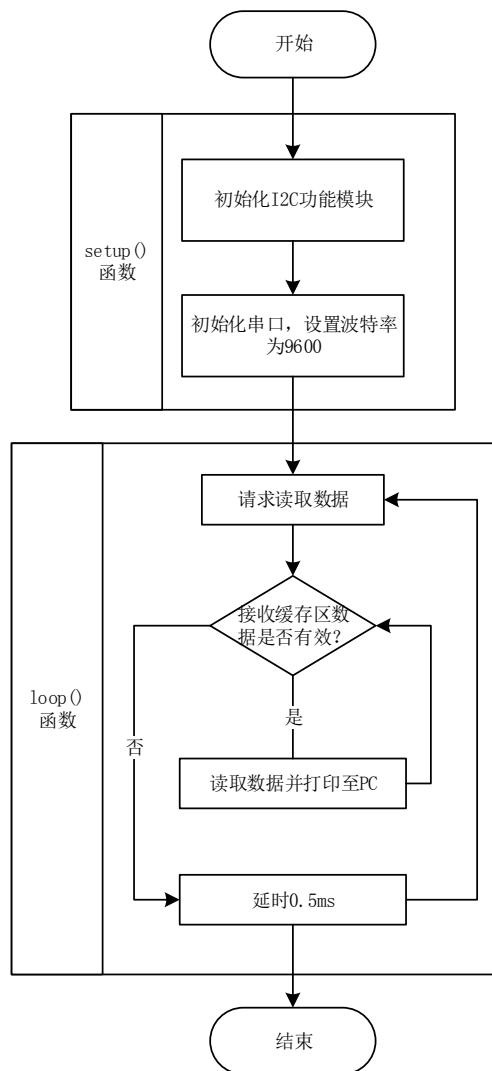


图 4-43 Lab-13 流程图

## 4.6.5 Lab-14 读取 HDC1080 模块温湿度数据

### 4.6.5.1 实验准备

实验实现 HDC1080 模块的温湿度采集功能。HDC1080 是一个温湿度传感器，其通过 I2C 接口输出温湿度数据。实验读取温湿度数据后根据芯片手册给出的计算公式实现对温度和湿度数据的计算。HDC1080 模块为 AY-IOT Kit 配套模块。该实验可加深对 CC3200 的 I2C 功能模块的应用，因为“4.6.3Lab-12 数据发送基本过程”和“4.6.4Lab-13 数据读取基本过程”例程没有进行实验验证。

HDC1080 模块为 AY-IOT Kit 配套模块，其连接方式可以下载艾研官方网站提供的文档《AY-IOT KIT 连接使用须知》，网址：[http://www.hpati.com/ay\\_doc\\_download/](http://www.hpati.com/ay_doc_download/)。连接方式如。

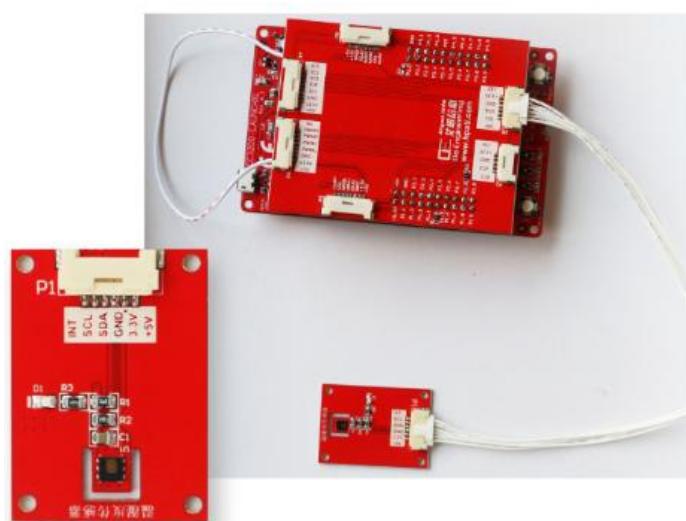


图 4-44 HDC1080 模块连接方式

实物连接完成后，CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80Mhz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。

### 4.6.5.2 端口映射

图 4-45 所示为 Energia 下对 I2C 通信端口的定义，有 Wire.begin() 开启的 I2C 通信模块使用的 SDA 和 SCL 通信端口即为图 4-45 所指示端口。



图 4-45 I2C 通信接口

Wire 库中 begin() 对 SCL 和 SDA 端口进行了初始化，其初始化的端口即为 P1\_9 和 P1\_10，函数初始化如下：

```
MAP_PinTypeI2C(PIN_01, PIN_MODE_1);
MAP_PinTypeI2C(PIN_02, PIN_MODE_1);
```

查看 CC3200 LaunchPad P1\_9 和 P1\_10 端口可以发现其对应的即为 P01 (PIN\_01) 和 P02 (PIN\_02)，也就是图 4-45 中的 SCL 和 SDA。

图 4-46 所示为 HDC1080 的原理图，其中 SCL 和 SDA 为 I2C 通信接口，实现温度和湿度数据传输。原理图中的 SDA 和 SCL 即实际对应图 4-45 所指的 I2C 通信端口。

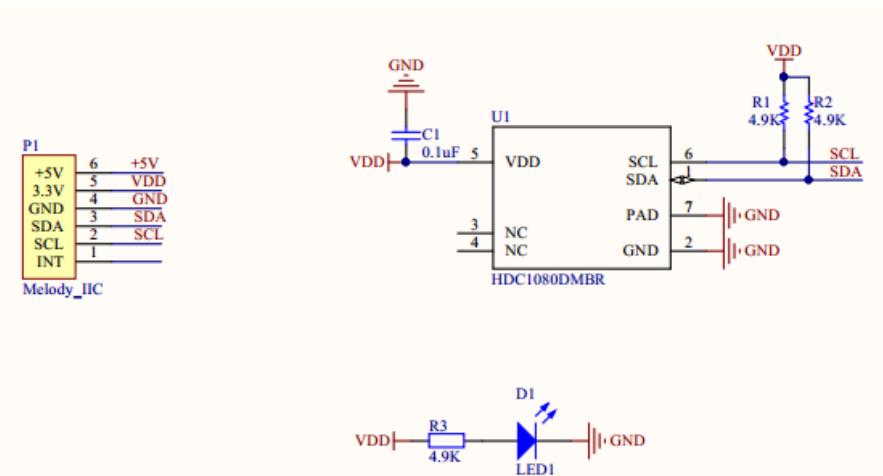


图 4-46 HDC1080 模块原理图

#### 4.6.5.3 代码解释

##### 4.6.5.3.1 写时序

HDC1080 写时序如下：

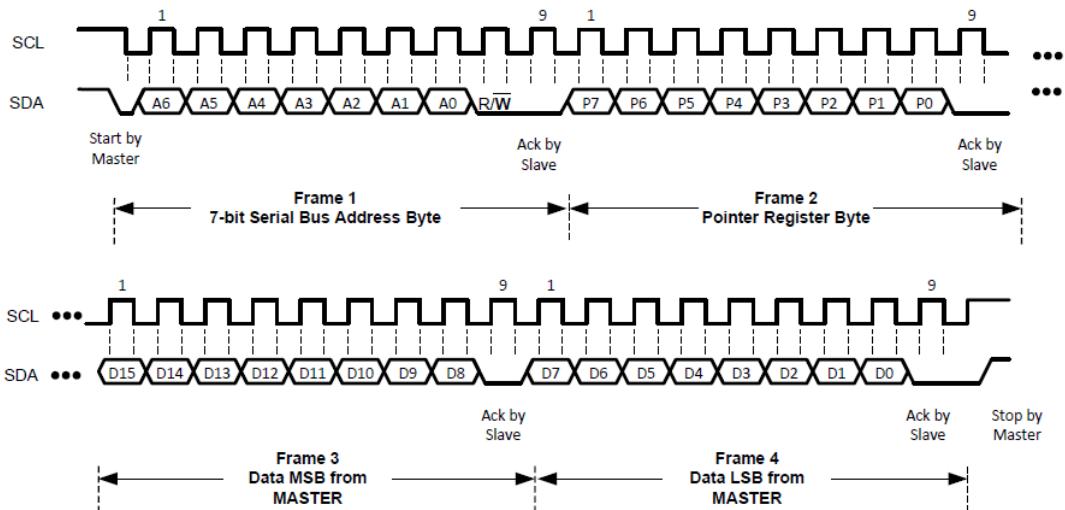


图 4-47 写时序

在实现时 cc3200 做为 Master, HDC1080 做为 slave。数据写入过程: 第一步: cc3200 发送 HDC1080 的地址, 根据芯片手册可知 HDC1080 的地址为 0b01000000 (0x40); 第二步: 发送需要写入的寄存器地址; 第三步: 发送两个字节长度的数据, 先发送高字节数据, 在发送低字节数据。

结合图 4-40 和图 4-47, 写数据程序实现如下 (不包括启动 I2C 功能, I2C 启动仅需调用 `Wire.begin()` 即可, 并且在使用过程中仅可被调用一次。)

```
void write(uint8_t regAddress,uint8_t* pdata,uint8_t length)
{
    Wire.beginTransmission(ADDRESS);
    Wire.write(regAddress);
    Wire.write(pdata,length);
    Wire.endTransmission();
}
```

使用方式:

```
uint8_t data[2]={0};
data[0]=(NORMAL_OPERATION >>8)&0xFF;
data[1]= NORMAL_OPERATION &0xFF;

write(REG_CONFIGURATION, data,2);
```

#### 4.6.5.3.2 读时序

HDC1080 读时序如下:

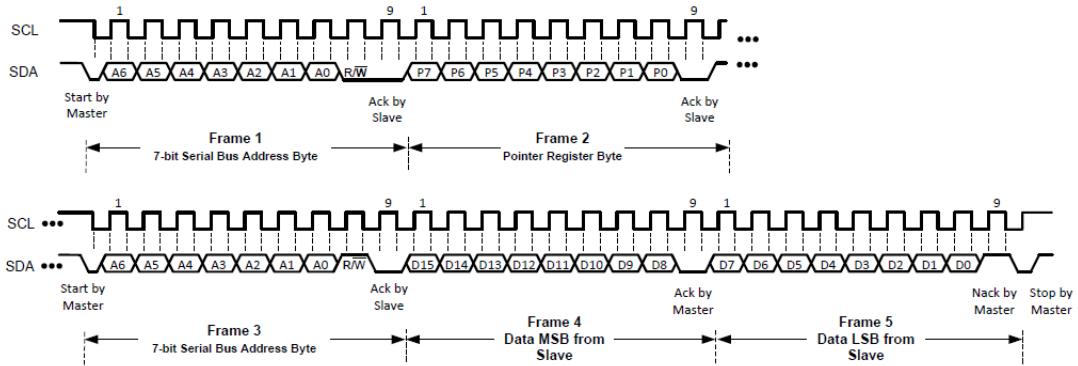


图 4-48 读时序

在实现时 cc3200 做为 Master, HDC1080 做为 slave。数据读取过程: 第一步: cc3200 发送 HDC1080 的地址, 根据芯片手册可知 HDC1080 的地址为 0b01000000 (0x40); 第二步: 发送需要读取数据的寄存器地址; 第三步: 再次发送 HDC1080 的地址, 第四步: 读取两个字节长度的数据, 高字节在前, 低字节在后。

结合图 4-40、图 4-42 和图 4-48, 读数据程序实现如下 (不包括启动 I2C 功能, I2C 启动仅需调用 Wire.begin() 即可, 并且在使用过程中仅可被调用一次。)

```
uint16_t read(uint8_t regAddress)
{
    uint16_t data = 0;
    Wire.beginTransmission(ADDRESS);
    Wire.write(regAddress);
    Wire.endTransmission();

    delay(40);

    Wire.requestFrom(ADDRESS, 2);
    if(Wire.available() >= 2)
    {
        data = Wire.read() << 8;
        data |= Wire.read();
    }
    return data;
}
```

使用时直接调用该函数即可, 在实例程序中函数 hdc1080\_getTemperature() 和 hdc1080\_getHumidity() 使用了 read() 来读取当前环境下的温度和湿度数据。

#### 4.6.5.3.3 例程函数

表 4-29 函数 read()

原型	uint16_t read(uint8_t regAddress)
参数	regAddress: HDC1080 寄存器地址

返回值	regAddress 对应寄存器的数据
说明	读取 regAddress 指定寄存器的数据
实例	uint16_t temperature = read(REG_TEMPERATURE);

表 4-30 函数 write()

原型	<b>void write(uint8_t regAddress, uint8_t *pdata, uint8_t length)</b>
参数	<b>regAddress:</b> HDC1080 寄存器地址; <b>*pdata:</b> 指向需要发送的数据缓存区指针; <b>length:</b> 待发送数据长度
返回值	无
说明	发送数据至 regAddress 指向的寄存器
实例	<code>uint8_t data[2]={0}; data[0]=(NORMAL_OPERATION &gt;&gt;8)&amp;0xFF; data[1]= NORMAL_OPERATION &amp;0xFF; write(REG_CONFIGURATION, data,2);</code>

表 4-31 函数 hdc1080\_init()

原型	<b>void hdc1080_init(void)</b>
参数	无
返回值	无
说明	开启 I2C 功能模块，初始化设置 HDC1080 工作模式
实例	Hdc1080_init()

表 4-32 函数 hdc1080\_getTemperature()

原型	<b>float hdc1080_getTemperature(void)</b>
参数	无
返回值	计算后的当前温度值
说明	读取 HDC1080 对应温度寄存器上的数据，并转换成温度数据。函数内部调用表 4-29 函数 read()
实例	float temperature = hdc1080_getTemperature();

表 4-33 函数 hdc1080\_getHumidity()

原型	<b>float hdc1080_getHumidity(void)</b>
参数	无
返回值	计算后的当前湿度值
说明	读取 HDC1080 对应的湿度寄存器上的数据，并转换成湿度数据。函数内部调用表 4-29 函数 read()
实例	float humidity = hdc1080_getHumidity();

read() 和 write() 函数作为基础函数被使用，其被 hdc1080\_init()、hdc1080\_getTemperature() 和 hdc1080\_getHumidity() 使用。在例程实现中，setup() 函数调用 init()，loop() 函数调用 hdc1080\_getTemperature() 和 hdc1080\_getHumidity() 函数即可。

#### 4.6.5.3.4 例程实现

例程直接调用 4.6.5.3.3 章节所示的函数实现。在 `setup()` 函数内，例程直接使用 `hdc1080_init()` 函数完成 I2C 通信模块和 HDC1080 模块的初始化设置，并开启串口实现温湿度数据传输至 PC 端进行展示。在 `loop()` 函数内，例程调用 `hdc1080_getTemperature()` 和 `hdc1080_getHumidity()` 函数获取当前环境下的温湿度数据，并通过串口传输至 PC 端。例程中使用到的 HDC1080 的地址和寄存器地址都通过宏定义实现。例程代码详见 4.6.5.4。

通过 I2C 从 HDC1080 中读取的数据并非直接是温度和湿度数据，所以程序需要通过相应的转换才能得到真正的温湿度数据。转换公式如下所示，用户也可以通过 HDC1080 的 datasheet 获知。

温度数据的转换公式：

$$Temperature(^{\circ}C) = \left( \frac{TEMPERATURE[15:00]}{2^{16}} \right) \times 165^{\circ}C - 40^{\circ}C$$

其中 `TEMPERATURE[15:00]` 是通过 I2C 读取的 HDC1080 温度寄存器中的数据。该转换公式在函数 `hdc1080_getTemperature()` 中实现。

湿度数据转换公式：

$$Humidity(\%RH) = \left( \frac{HUMIDITY[15:00]}{2^{16}} \right) \times 100\% RH$$

其中 `HUMIDITY[15:00]` 是通过 I2C 读取的 HDC1080 湿度寄存器中的数据。该转换公式在函数 `hdc1080_getHumidity()` 中实现。

#### 4.6.5.4 完整代码和流程图

##### 完整代码

```
#include <Wire.h>

//define hdc1080's address and registers' address
#define ADDRESS          0x40      //hdc1080's address

#define REG_TEMPERATURE   0x00
#define REG_HUMIDITY      0x01
#define REG_CONFIGURATION 0x02
#define NORMAL_OPERATION  0x0000

//read data from register
uint16_t read(uint8_t regAddress)
{
    uint16_t data = 0;
    Wire.beginTransmission(ADDRESS);
    Wire.write(regAddress);
    Wire.endTransmission();
    delay(40);
}
```

```
Wire.requestFrom(ADDRESS,2);
if(Wire.available()>=2)
{
    data = Wire.read()<<8;
    data |= Wire.read();
}
return data;
}

//write data to register
void write(uint8_t regAddress,uint8_t*pdata,uint8_t length)
{
    Wire.beginTransmission(ADDRESS);
    Wire.write(regAddress);
    Wire.write(pdata,length);
    Wire.endTransmission();
}

void hdc1080_init(void)
{
    Wire.begin();
    uint8_t data[2]={0};
    data[0]=(NORMAL_OPERATION >>8)&0xFF;
    data[1]= NORMAL_OPERATION &0xFF;
    write(REG_CONFIGURATION, data,2);
}

float hdc1080_getTemperature(void)
{
    uint16_t temperature =0;
    temperature = read(REG_TEMPERATURE);
    return(temperature /65536.0*165.0-40.0);
}

float hdc1080_getHumidity(void)
{
    uint16_t humidity =0;
    humidity = read(REG_HUMIDITY);

    return(humidity /65536.0*100.0);
}

void setup()
{
// put your setup code here, to run once:
    hdc1080_init();
    Serial.begin(115200);
```

```

}

void loop()
{
// put your main code here, to run repeatedly:
Serial.print("temperature:");
Serial.println(hdc1080_getTemperature());
Serial.print("humidity:");
Serial.println(hdc1080_getHumidity());
delay(1000);
}

```

流程图

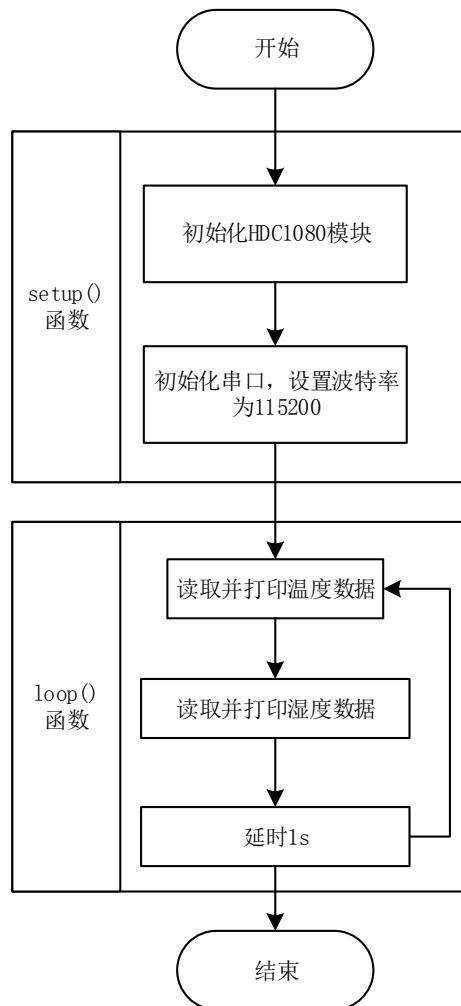


图 4-49 Lab-14 流程图

#### 4.6.5.5 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写

完成后程序将自动运行，点击  打开串口监视器，设置波特率为 115200。实验现象如下

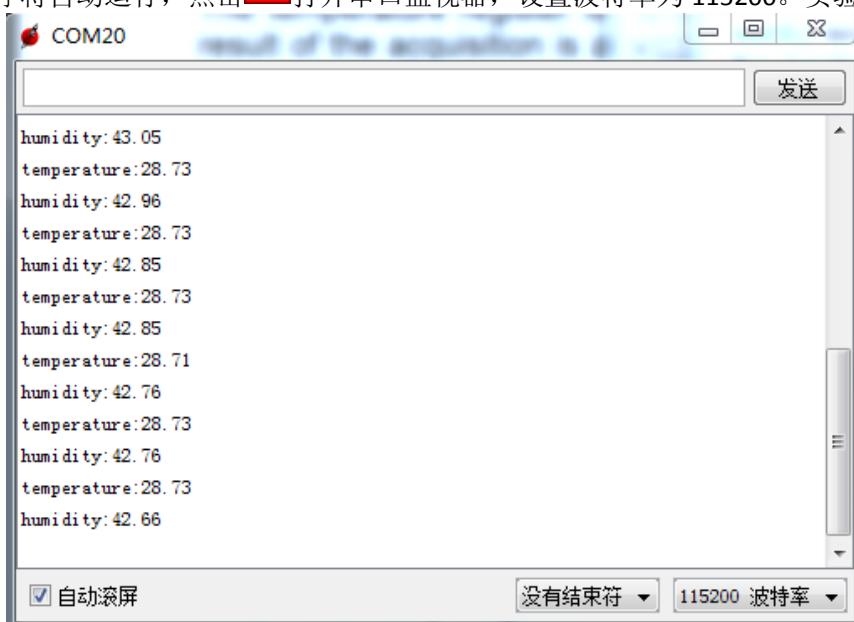


图 4-50 HDC1080 模块实验现象

## 4.7 SPI

### 4.7.1 概要

SPI 设备之间通信需要四根数据线: MISO(Master In Slave Out)、MOSI(Master Out Slave In)、SCK(Serial ClockD)、SS(Slave Select)。

MISO: 从设备数据发送线，主设备使用该数据线实现数据读取；

MOSI: 主设备数据发送线，从设备通过该数据线获取数据；

SCK: 时钟信号线，同步主从设备之间的数据发送和读取；

SS: 片选线，低电平有效，该数据线处于低电平时。主从设备之间传输数据有效，即主设备发出的数据从设备能够读取成功。

在 Energia 下进行 SPI 通信可以使用其提供的 SPI 库实现，库文件参考内容可见官网：  
<http://energia.nu/reference/spi/>。如果想要查看库文件源码可在 CC3200 LaunchPad 安装目录下找到：【C:\Users\用户名（PC 用户名，根据用户实而不同）\AppData\Local\Energia15\packages\energia\hardware\cc3200\1.0.2\libraries\SPI】该目录下的 SPI.h 和 SPI.cpp。

### 4.7.2 SPI 库

SPI 库使用 C++ 实现，具体实现方式可查看 SPI.saaaassssh 和 Wire.cpp 文件。Wire(I2C)库存在多个成员函数，函数列表如下

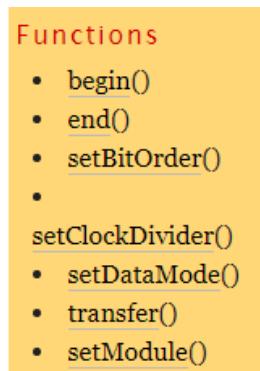


图 4-51 SPI 库成员函数列表

下面就对 SPI 编程过程中经常使用的函数进行简单介绍。未说明的函数用户可在 Energia 官网上进行了解，地址：<http://energia.nu/reference/spi/> 在网页左侧 Functions 栏里找到对应函数点击进入即可。

表 4-34 函数 begin()

语法	<b>SPI.begin()</b>
参数	无
返回值	无
说明	初始化 SPI 功能
实例	SPI.begin()

表 4-35 函数 end()

语法	<b>SPI.end()</b>
参数	无
返回值	无
说明	释放 SPI 总线
实例	SPI.begin()

表 4-36 函数 transfer()

语法	<b>SPI.transfer(val)</b>
参数	<b>val</b> : 需要发送的数据（一个字节长度）
返回值	返回读取出的数据
说明	发送一个字节数据至 SPI 总线，数据发送状态下， <b>val</b> 数据会发送至 SPI 从设备，读取模式下，该操作实现一次“虚写”过程，实现读取从设备发送的数据。
实例	SPI.transfer(0x5A)

#### 4.7.3 SPI 数据基本发送和读取过程

Energia 平台下 SPI 库在实现过程中将 SS(Slave Select)也可以称为 CS 线留给用户自行实现。SPI 库 begin() 函数内部实现如下。可以发现设置 SS (CS) 线的代码注释了，并且在注释的代

码上还进行了说明：使用软件控制 CS 线即留给用户使用 digitalWrite() 函数控制。

```
void SPIClass::begin() {
    unsigned long initialData = 0;

    if(SSIModule == NOT_ACTIVE) {
        SSIModule = BOOST_PACK_SPI;
    }

    MAP_PRCMPeripheralClkEnable(PRCM_GSPI, PRCM_RUN_MODE_CLK);

    /* Configure pins as SPI */
    MAP_PinTypeSPI(g_ulSSIPins[SSIModule][0],
    g_ulSSIPinModes[SSIModule][0]);
    /* Leave the CS pin under software control */
    // MAP_PinTypeSPI(g_ulSSIPins[SSIModule][1],
    // g_ulSSIPinModes[SSIModule][1]);
    MAP_PinTypeSPI(g_ulSSIPins[SSIModule][2],
    g_ulSSIPinModes[SSIModule][2]);
    MAP_PinTypeSPI(g_ulSSIPins[SSIModule][3],
    g_ulSSIPinModes[SSIModule][3]);

    MAP_PRCMPeripheralReset(PRCM_GSPI);
    MAP_SPIReset(SSIBASE);

    /* 4 MHz clock, MODE 0, 3 PIN with CS under S/W control */
    MAP_SPIConfigSetExpClk(SSIBASE, MAP_PRCMPeripheralClockGet(PRCM_GSPI),
    4000000, SPI_MODE_MASTER, SPI_MODE0,
    (SPI_HW_CTRL_CS |
        SPI_3PIN_MODE |
        SPI_TURBO_OFF |
        SPI_CS_ACTIVELOW |
        SPI_WL_8));
}

MAP_SPIEnable(SSIBASE);
}
```

SPI 在数据通信过程基本时序为：1、SS(CS)置低电平，使数据通信有效；2、进行数据读取和发送；3、SS(CS)置高电平，结束本次数据通信过程。具体数据读取和发送时序实现可见 4.7.3.1 发送过程和 4.7.3.2 读取过程

#### 4.7.3.1 发送过程

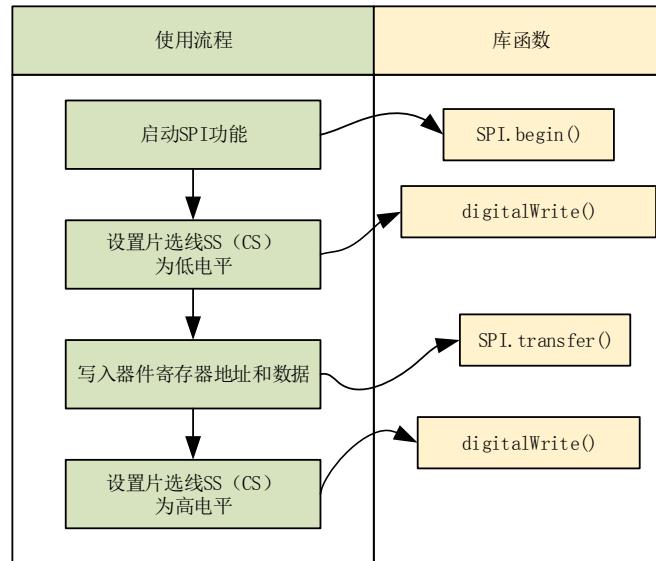


图 4-52 SPI 数据发送

#### 4.7.3.2 读取过程

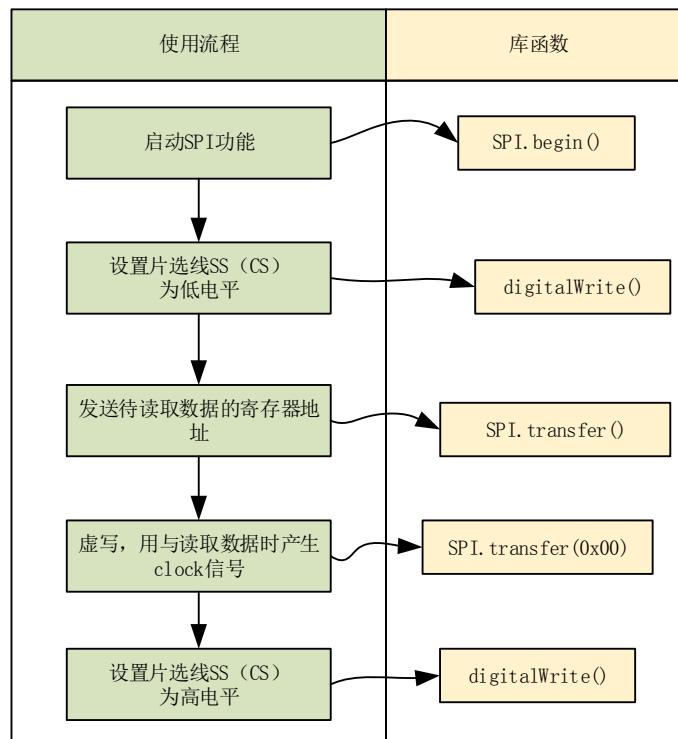


图 4-53 读取过程

#### 4.7.4 Lab-15 读取 LDC1000 模块数据

##### 4.7.4.1 实验准备

实验实现 LDC1000 模块的数据采集功能。LDC1000 是电感传感器，其通过 SPI 接口输出接近系数和当前频率数据。实验获取这些数据后根据芯片手册给出的计算公式计算出互感系数。LDC1000 模块为 AY-IOT Kit 配套模块。该实验可加深对 CC3200 的 SPI 功能模块的应用，LDC1000 模块为 AY-IOT Kit 配套模块，其连接方式可以下载艾研官方网站提供的文档《AY-IOT KIT 连接使用须知》，网址：[http://www.hpati.com/ay\\_doc\\_download/](http://www.hpati.com/ay_doc_download/)。连接方式如。

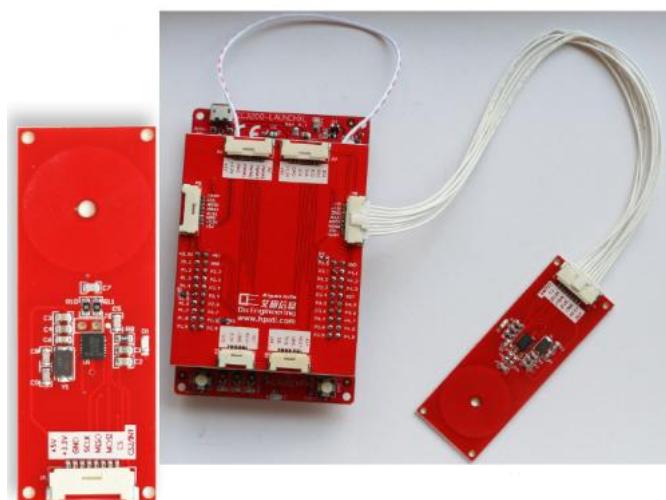


图 4-54 LDC1000 模块连接方式

实物连接完成后，CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80Mhz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。

##### 4.7.4.2 端口映射

根据 4.7.3 节可知，SPI 通信中 SCK，MISO，MOI 有 CC3200 的 SPI 模块实现，CS(SS:Slave Select)留给用户自行控制。SPI 模块端口映射如图 4-56。SCK 对应 P1\_7，MOSI 对应 P2\_15，MISO 对应 P2\_14。观察 CC3200 LaunchPad 实物可以发现 P1\_5 对应 P05 (PIN\_05)、P2\_15 对应 P07 (PIN\_07)、P2\_14 对应 P06 (PIN\_06)，而 CS 线选择 P1\_8 (P62)，可见 AY-IOT Kit 套件原理图：[http://www.hpati.com/ay\\_doc\\_download/](http://www.hpati.com/ay_doc_download/)。



图 4-55 AY-IOT KIT 原理图下载图示

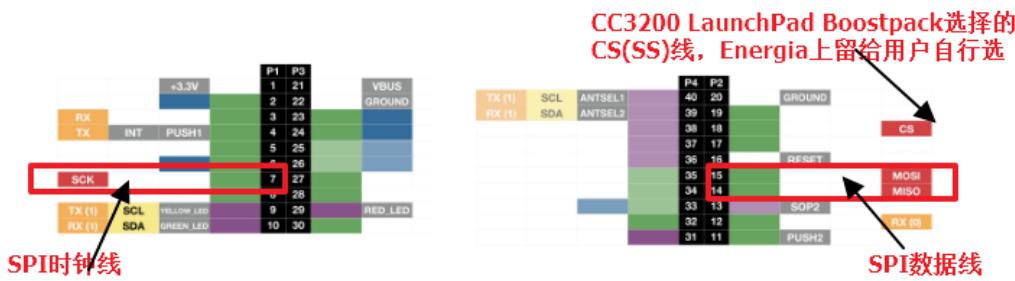


图 4-56 SPI 端口映射

图 4-57 所示为 LDC1000 的原理图, 其中 SCLK、SDI、SDO 和 CSB 为 SPI 通信接口, 数据传输。原理图中的 SCLK、SDI、SDO 即实际对应图 4-56 所指的 SCK、MOSI、MISO 通信端口, CSB 即为 P1\_8 (P62), 作为模块的片选线使用。

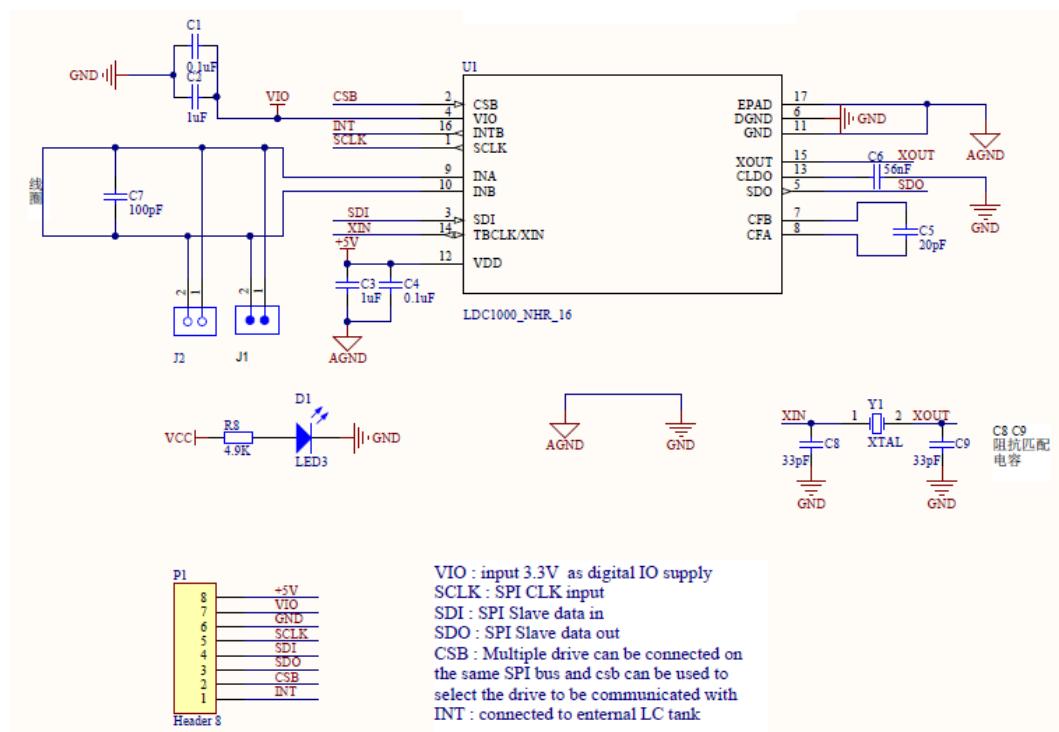


图 4-57 LDC1000 模块原理图

SPI 端口定义如下:

```
static const unsigned long g_ulSSIIPins[] [4]={  
{PIN_05 /* SCLK */, PIN_08 /* SS */, PIN_06 /* MISO */, PIN_07 /* MOSI */}  
};
```

而实际使用过程中开启 SPI 功能仅需要使用 SPI.begin()即可, 该函数功能即为初始化设计这些端口的状态。SPI.begin()函数可以查看 4.7.3 节。

#### 4.7.4.3 代码解释

##### 4.7.4.3.1 LDC1000 时序实现

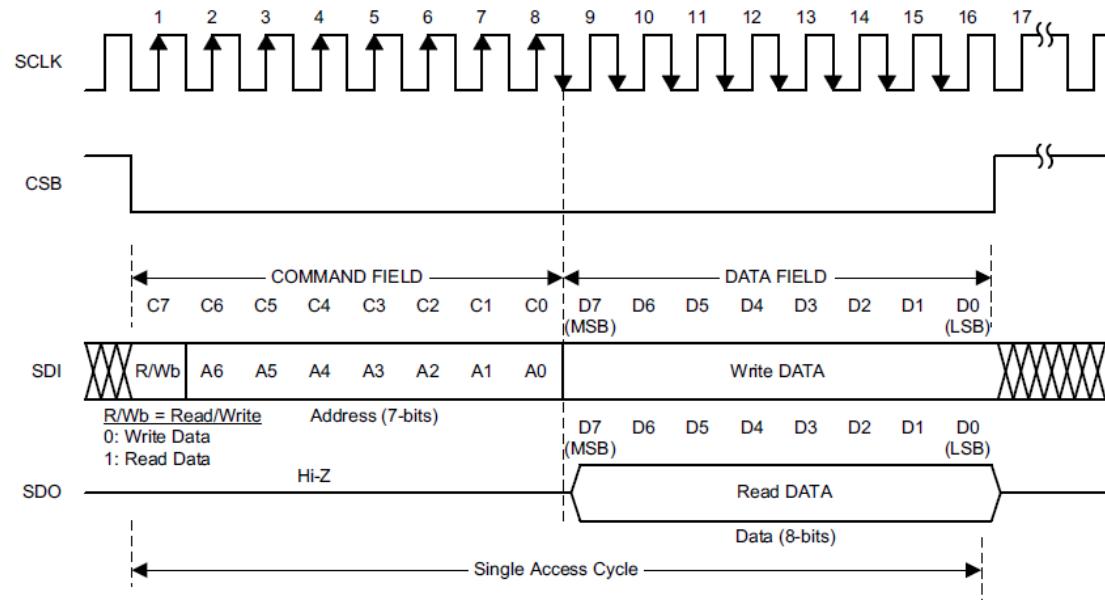


图 4-58 LDC1000 时序图

根据时序图可知，LDC1000 数据通过由两个字段构成：COMMAND 和 DATA，字段长度均为 8-bits。

**COMMAND:** 包括一个读写位，7 个地址位。读写位用来区别当前数据通信过程中 DATA 字段

传输方向；地址位用来识别当前设备。

**DATA:** 传输 d 数据。当读写位为 1 时，当前传输过程为读数据，SPI 主设备读取数据；当读写位为 0 时，当前传输过程为写数据，SPI 主设备发送数据。

代码实现如下：

```

void write(uint8_t regAddress,uint8_t data)
{
    digitalWrite(LDC_CS, LOW);

    SPI.transfer(regAddress | LDC_WRITEBIT); //COMMAND:write bit + 7-bits address
    SPI.transfer(data); //DATA

    digitalWrite(LDC_CS, HIGH);
}

uint8_t read(uint8_t regAddress)
{

```

```

uint8_t result = 0;

digitalWrite(LDC_CS, LOW);

SPI.transfer(regAddress | LDC_WRITEBIT); //COMMAND:read bit + 7-bits
result = SPI.transfer(0x00); //virtual write for read DATA

digitalWrite(LDC_CS, HIGH);
return result;
}

```

其中 LDC\_CS、LDC\_WRITEBIT、LDC\_READBIT 定义如下：

#define LDC_CS	8	//cs boosterpack P1_8(PIN62)
#define LDC_WRITEBIT	0x00	//write bit
#define LDC_READBIT	0x80	//read bit

#### 4.7.4.3.2 例程函数

表 4-37 函数 write()

原型	<b>void write(uint8_t regAddress, uint8_t data)</b>
参数	regAddress: 寄存器地址 data: 需要发送的数据
返回值	无
说明	发送数据至指定的寄存器
实例	write(0x01, 0x13)

表 4-38 函数 read()

原型	<b>uint8_t read(uint8_t regAddress)</b>
参数	regAddress: 寄存器地址
返回值	参数指定寄存器的数据
说明	读取参数 regAddress 所指定的寄存器中的数据
实例	uint8_t value = read(0x13)

表 4-39 函数 ldc1000\_init()

原型	<b>void ldc1000_init(void)</b>
参数	无
返回值	无
说明	使能 SPI 功能模块，初始化 LDC1000 工作模式
实例	ldc1000_init()

表 4-40 函数 ldc1000\_getProximity()

原型	<code>uint16_t ldc1000_getProximity (void)</code>
参数	无
返回值	LDC1000 采集到的接近系数
说明	函数读取 LDC1000 接近系数寄存器中的数据
实例	<code>uint16_t proximity = ldc1000_getProximity()</code>

表 4-41 函数 ldc1000\_getFrequency()

原型	<code>uint32_t ldc1000_getFrequency (void)</code>
参数	无
返回值	LDC1000 采集到的频率值
说明	函数读取 LDC1000 频率寄存器中的数据
实例	<code>uint16_t proximity = ldc1000_getFrequency()</code>

表 4-42 函数 ldc1000\_calcInductance()

原型	<code>float ldc1000_calcInductance(uint32_t frequency)</code>
参数	frequency: 频率值, 由函数 ldc1000_getFrequency() 读取到的频率值
返回值	互感值
说明	计算 LDC1000 的互感值
实例	<code>float inductance = ldc1000_calcInductance(ldc1000_getFrequency ())</code>

read() 和 write() 函数作为基础函数被使用, 其被 ldc1000\_init()、ldc1000\_getProximity () 和 ldc1000\_getFrequency () 使用。在例程实现中, setup() 函数调用 ldc1000\_init(), loop() 函数调用 ldc1000\_getProximity () 和 ldc1000\_getFrequency () 函数即可, 在获取频率值后使用 ldc1000\_calcInductance() 函数计算互感值。

#### 4.7.4.3.3 例程实现

例程直接调用 4.7.4.3.2 章节所示的函数实现。在 setup() 函数内, 例程直接使用 ldc1000\_init() 函数完成 SPI 通信模块和 LDC1000 模块的初始化设置, 并开启串口实现数据传输至 PC 端进行展示。在 loop() 函数内, 例程调用 ldc1000\_getProximity () 和 ldc1000\_getFrequency () 函数获取当前状态下的接近系数和频率数据; 调用 ldc1000\_calcInductance() 函数计算互感值, 并通过串口传输至 PC 端。例程中使用到的 LDC1000 的地址和寄存器地址都通过宏定义实现。例程代码详见 4.7.4.4。

例程通过 SPI 接口能够获取实验过程中的接近系数和频率数据, 而互感值需要通过频率数据计算得出。根据 LDC1000 的 datasheet 可知计算过程分成两步: 第一步根据 SPI 读取的芯片“Frequency Counter Data”寄存器的数据(即为上述提到的频率数据)计算出 LC 谐振频率; 第二步根据计算出的谐振频率计算互感值。

第一步计算公式:

$$f_{sensor} = \frac{f_{ext} \times responseTime}{3 \times f_{count}}$$

$f_{ext}$  是基准时钟周期，有外部晶振提供，可见图 4-57，其大小为 8MHz；  $responseTime$  由 LDC1000 Configuration（地址为 0x04）寄存器低 3 位确定，其大小选择 6144；  $f_{count}$  为寄存器“Frequency Counter Data”的输出值，由例程通过 SPI 接口直接读取。

第二步计算公式：

$$L = \frac{1}{C \times (2\pi * f_{sensor})^2}$$

$C$  为 LDC1000 线圈的并联电阻，其值可见图 4-57，其大小为  $100\text{pF}$ ；  $f_{sensor}$  为第一步计算公式的结果。

例程在函数 `ldc1000_calcInductance()` 实现上述两个步骤的计算过程，而  $f_{count}$  由函数 `ldc1000_getFrequency()` 获取。

#### 4.7.4.4 完整代码和流程图

##### 完整代码

```
#include <SPI.h>

#define LDC_CS 8 //cs, boosterpack P1_8(PIN62)

#define LDC_WRITEBIT 0x00 //write bit
#define LDC_READBIT 0x80 //read bit

#define REG_RPMAX 0x01
#define REG_RPMIN 0x02
#define REG_SENSORFREQUENCY 0x03
#define REG_LDCCONFIG 0x04
#define REG_CLKCONFIG 0x05
#define REG_THRESHOLDHIGH_LSB 0x06
#define REG_THRESHOLDHIGH_MSB 0x07
#define REG_THRESHOLDLOW_LSB 0x08
#define REG_THRESHOLDLOW_MSB 0x09
#define REG_INTCONFIG 0x0A
#define REG_PWRCONFIG 0x0B
#define REG_PROXIMITY_LSB 0x21
#define REG_PROXIMITY_MSB 0x22
#define REG_FREQUENCY_LSB 0x23
#define REG_FREQUENCY_MID 0x24
#define REG_FREQUENCY_MSB 0x25

#define VALUE_SENSORFREQUENCY 0xD1
```

```
#define VALUE_LDCCONFIG          0x17
#define VALUE_CLKCONFIG           0x02
#define VALUE_INICONFIG          0x04
#define VALUE_THRESHOLDHIGH_LSB   0x50
#define VALUE_THRESHOLDHIGH_MSB   0x14
#define VALUE_THRESHOLDLOW_LSB    0xC0
#define VALUE_THRESHOLDLOW_MSB    0x12
#define VALUE_PWRCONFIG            0x01
#define VALUE_RPMAX                0x13
#define VALUE_RPMIN                0x3A

#define LDC_FREQUENCY_EXT          8
#define LDC_RESPONSE_TIME          6144
#define LDC_PARALLEI_CAP            100

const float EPSINON = 0.00001;
void write(uint8_t regAddress,uint8_t data)
{
    digitalWrite(LDC_CS, LOW);

    SPI.transfer(regAddress | LDC_WRITEBIT);
    SPI.transfer(data);

    digitalWrite(LDC_CS, HIGH);
}

uint8_t read(uint8_t regAddress)
{
    uint8_t result = 0;

    digitalWrite(LDC_CS, LOW);

    SPI.transfer(regAddress | LDC_READBIT);
    result = SPI.transfer(0x00);

    digitalWrite(LDC_CS, HIGH);
    return result;
}

void ldc1000_init(void)
{
    pinMode(LDC_CS, OUTPUT);
    SPI.begin();
}
```

```
//configure LDC1000
    write(REG_RPMAX, VALUE_RPMAX);
    write(REG_RPMIN, VALUE_RPMIN);
    write(REG_SENSORFREQUENCY, VALUE_SENSORFREQUENCY);
    write(REG_LDCCONFIG, VALUE_LDCCONFIG);
    write(REG_CLKCONFIG, VALUE_CLKCONFIG);
    write(REG_THRESHOLDHIGH_LSB, VALUE_THRESHOLDHIGH_LSB);
    write(REG_THRESHOLDHIGH_MSB, VALUE_THRESHOLDHIGH_MSB);
    write(REG_THRESHOLDLOW_LSB, VALUE_THRESHOLDLOW_LSB);
    write(REG_THRESHOLDLOW_MSB, VALUE_THRESHOLDLOW_MSB);
    write(REG_PWRCONFIG, VALUE_PWRCONFIG);
}

uint16_t ldc1000_getProximity(void)
{
    uint16_t temp[2]={0};
    uint16_t proximity =0;
    temp[0]= read(REG_PROXIMITY_LSB)&0x00FF;
    temp[1]= read(REG_PROXIMITY_MSB)&0x00FF;
    proximity =(temp[1]<<8) | temp[0];
    return proximity;
}
uint32_t ldc1000_getFrequency(void)
{
    uint32_t count[3]={0};
    uint32_t frequency =0;
    count[0]= read(REG_FREQUENCY_LSB)&0x000000FF;
    count[1]= read(REG_FREQUENCY_MID)&0x000000FF;
    count[2]= read(REG_FREQUENCY_MSB)&0x000000FF;
    frequency =(count[2]<<16) | (count[1]<<8) | count[0];
    return frequency;
}
float ldc1000_calcInductance(uint32_t frequency)
{
    if(!frequency)
    {
        return 0.0;
    }
    float sensorFreq =(1.0/3)*(1.0* LDC_FREQUENCY_EXT / frequency)*
LDC_RESPONSE_TIME;
    if((sensorFreq >=EPSINON)&&(sensorFreq <= EPSINON))
    {
        return 0.0;
```

```
}

float inductance = 1.0 * 1000000 / (LDC_PARALLEL_CAP * pow(TWO_PI *
sensorFreq, 3));
return inductance;
}

void setup()
{
// put your setup code here, to run once:
Serial.begin(11500);
ldc1000_init();
}

void loop()
{
// put your main code here, to run repeatedly:
uint16_t proximity = ldc1000_getProximity();
Serial.print("Proximity:");
Serial.print(proximity);
uint32_t frequency = ldc1000_getFrequency();
Serial.print("\t\tFrequency:");
Serial.print(frequency);

Serial.print("\t\tInductance:");
Serial.println(ldc1000_calcInductance(frequency));
delay(1000);
}
```

流程图:

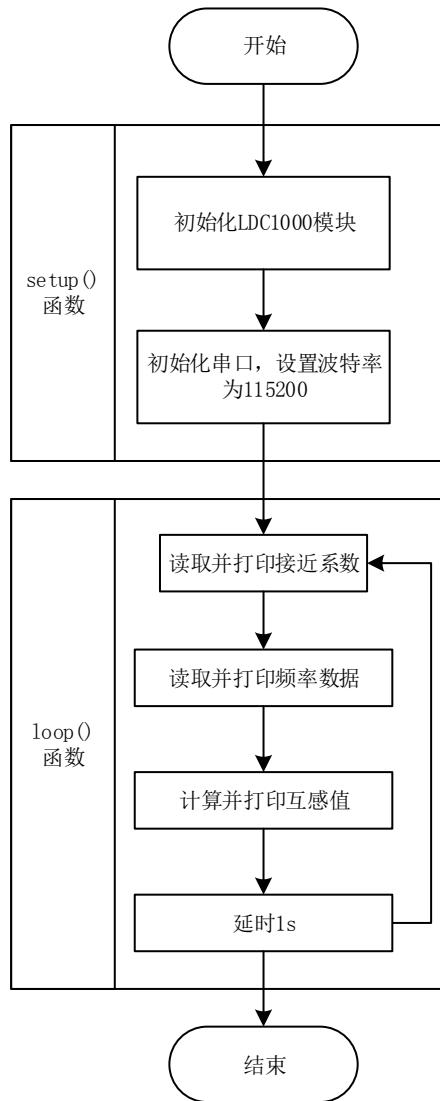


图 4-59 lab-15 流程图

#### 4.7.4.5 实验现象

点击 ，Energia 上可能会提示如下错误。

```
C:\Users\hai\Desktop\lab-13\lab-13.ino: In function 'void ldc1000_init()':
lab-13:70: error: 'SPI' was not declared in this scope
    SPI.begin();
    ^
exit status 1
'SPI' was not declared in this scope
```

图 4-60 SPI 未定义错误

SPI 库实际为 Energia 提供，其源文件位于 **[C:\Users\用户名 (PC 用户名, 根据用户实而不**

同) \AppData\Local\Energia15\packages\energia\hardware\cc3200\1.0.2\libraries\SPI】文件夹内, 编译出错问题未找到原因(使用 Energia 提供的 SPI 例程也出现同样错误)。现提供另外一种解决方式: 将 SPI 源文件复制到 lab-13.ino 例程代码同一目录下: 如图 4-61 方式放置。



图 4-61 SPI 源文件方式目录

重新编译成将不再提示图 4-60 所示错误, 点击  进行编译烧写程序至 CC3200 LaunchPad, 烧写完成后程序将自动运行, 点击  打开串口监视器, 设置波特率为 115200。将 LDC1000 模块接近硬币或者其它金属物质, 其互感器会有相应变化。

## 4.8 WiFi

参考 WiFi 库中的例程

### 4.8.1 概要

在 Energia 下 CC3200LaunchPad 实现物联网功能可以使用其提供的 WiFi 库实现, 库文件参考内容可见官网: <http://energia.nu/reference/wifi/>。如果想要查看库文件源码可在 CC3200 LaunchPad 安装目录下找到: 【C:\Users\用户名 (PC 用户名, 根据用户实而不同)\AppData\Local\Energia15\packages\energia\hardware\cc3200\1.0.2\libraries\WiFi】。

WiFi 库使用 C++ 实现, 整个 WiFi 包括 WiFi 类、IPAddress 类、Server 类、Client 类、UDP 类。WiFi 类初始化网络设置; IPAddress 类提供网络的配置信息, 如 IP 地址等; Server 类实现服务端应用实现与客户端(其他设备)数据的发送和接收, 如 webServer; Client 类可创建客户端应用程序, 实现与服务器(其他设备)直接的数据发送和接收; UDP 类实现 UDP 格式类型的数据发送和接收。

本章将通过 4 个实验例程来介绍 WiFi 库的使用。

### 4.8.2 Lab-16 扫描当前环境下的可用网络

#### 4.8.2.1 实验准备

实验例程实现 CC3200 LaunchPad 自动扫描当前环境下可用的网络(热点: AP), 扫描完成后通过串口打印至 PC 端进行显示。

C3200 LaunchPad 根据图 2-2 进行连接, 保证程序能够正常烧写, 通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80MHz)”, 端口根据图 2-15 进行选择, 即选择“COM20”(实际端口需用户查看设备管理器确定)。

#### 4.8.2.2 代码解释

实验例程选择 Energia 提供的 Example: ScanNetworks，其可通过“文件->示例->WiFi->ScanNetworks”打开。完整代码看 4.8.2.3

例程使用串口功能打印程序运行过程中的状态数据，在 `setup()` 函数内部使用 `WiFi.init()` 函数初始化 WiFi 功能模块；之后读取当前 CC3200 的固件信息，MAC 地址；最后扫描当前环境下可用的网络。在 `loop()` 函数内，例程定时扫描当前环境下的可用网络。

例程使用到的 WiFi 库函数介绍如下：WiFi 为 WiFi 类（`WiFiClass`）的对象

表 4-43 函数 `init()`

原型	<b>static bool init()</b>
参数	无
返回值	WiFi 功能初始化状态，true：初始化成功；false：初始化失败
说明	初始化 WiFi 功能模块
实例	<code>WiFi.init() / WiFiClass::init()</code> 。例程中使用 <code>WiFi.init()</code>

表 4-44 函数 `firmwareVersion()`

原型	<b>const char* firmwareVersion()</b>
参数	无
返回值	固件版本
说明	返回 CC3200 的固件版本
实例	<code>Serial.println(WiFi.firmwareVersion())</code>

表 4-45 函数 `macAddress()`

原型	<b>uint8_t* macAddress(uint8_t* mac)</b>
参数	<code>mac</code> ：指向存放 mac 缓存的指针
返回值	返回指向 mac 缓存的指针
说明	读取 CC3200 的 MAC 地址
实例	<code>byte mac[6];</code> <code>WiFi.macAddress(mac);</code>

表 4-46 函数 `scanNetworks()`

原型	<b>int8_t scanNetworks()</b>
参数	无
返回值	扫描到的可用网络个数
说明	扫描设备 CC3200 当前环境下可用网络
实例	<code>int8_t count = WiFi.scanNetworks();</code>

表 4-47 函数 `SSID()`

原型	<b>char* SSID(uint8_t networkItem)</b>
参数	<code>networkItem</code> ：网络编号，取值范围由 <code>scanNetworks()</code> 函数返回值确定
返回值	对应网络编号的 SSID
说明	在扫描可用网络时，可根据扫描到的网络编号获取对应网络的 SSID

实例	Serial.println(WiFi.SSID(1))
----	------------------------------

表 4-48 函数 RSSI()

原型	<b>int32_t RSSI(uint8_t networkItem)</b>
参数	<b>networkItem:</b> 网络编号, 取值范围由 scanNetworks() 函数返回值确定
返回值	对应网络编号的信号强度
说明	在扫描可用网络时, 可根据扫描到的网络编号获取对应网络的信号强度
实例	Serial.println(WiFi.RSSI(1))

表 4-49 函数 encryptionType()

原型	<b>uint8_t encryptionType(uint8_t networkItem)</b>
参数	<b>networkItem:</b> 网络编号, 取值范围由 scanNetworks() 函数返回值确定
返回值	对应网络编号的加密方式, 分为: ENC_TYPE_WEP, ENC_TYPE_TKIP, ENC_TYPE_CCMP, ENC_TYPE_NONE, ENC_TYPE_AUTO
说明	在扫描可用网络时, 可根据扫描到的网络编号获取对应网络的加密方式
实例	Serial.println(WiFi.encryptionType())

#### 4.8.2.3 完整代码和流程图

完整代码:

```
/*
This example prints the Wifi MAC address, and
scans for available Wifi networks.

Every ten seconds, it scans again. It doesn't actually
connect to any network, so no encryption scheme is specified.

*/
#ifndef __CC3200R1M1RGC__
// Do not include SPI for CC3200 LaunchPad
#include <SPI.h>
#endif
#include <WiFi.h>

void setup() {
//Initialize serial and wait for port to open:
Serial.begin(115200);

WiFi.init();
Serial.println(WiFi.firmwareVersion());

// Print WiFi MAC address:

```

```
printMacAddress();

// scan for existing networks:
Serial.println("Scanning available networks...");
listNetworks();
}

void loop(){
delay(10000);

// scan for existing networks:
Serial.println("Scanning available networks...");
listNetworks();
}

void printMacAddress(){
// the MAC address of your Wifi
byte mac[6];

// print your MAC address:
WiFi.macAddress(mac);
Serial.print("MAC: ");
Serial.print(mac[5], HEX);
Serial.print(":");
Serial.print(mac[4], HEX);
Serial.print(":");
Serial.print(mac[3], HEX);
Serial.print(":");
Serial.print(mac[2], HEX);
Serial.print(":");
Serial.print(mac[1], HEX);
Serial.print(":");
Serial.println(mac[0], HEX);
}

void listNetworks(){
// scan for nearby networks:
Serial.println("** Scan Networks **");
int numSsid = WiFi.scanNetworks();
if(numSsid == -1)
{
    Serial.println("Couldn't get a wifi connection");
    while(true);
}
```

```
}

// print the list of networks seen:
Serial.print("number of available networks:");
Serial.println(numSsid);

// print the network number and name for each network found:
for(int thisNet =0; thisNet < numSsid; thisNet++){
    Serial.print(thisNet);
    Serial.print(" ");
    Serial.print(WiFi.SSID(thisNet));
    Serial.print("\tSignal: ");
    Serial.print(WiFi.RSSI(thisNet));
    Serial.print(" dBm");
    Serial.print("\tEncryption: ");
    printEncryptionType(WiFi.encryptionType(thisNet));
}

void printEncryptionType(int thisType){
// read the encryption type and print out the name:
switch(thisType){
case ENC_TYPE_WEP:
    Serial.println("WEP");
break;
case ENC_TYPE_TKIP:
    Serial.println("WPA");
break;
case ENC_TYPE_CCMP:
    Serial.println("WPA2");
break;
case ENC_TYPE_NONE:
    Serial.println("None");
break;
case ENC_TYPE_AUTO:
    Serial.println("Auto");
break;
}
}
```

流程图

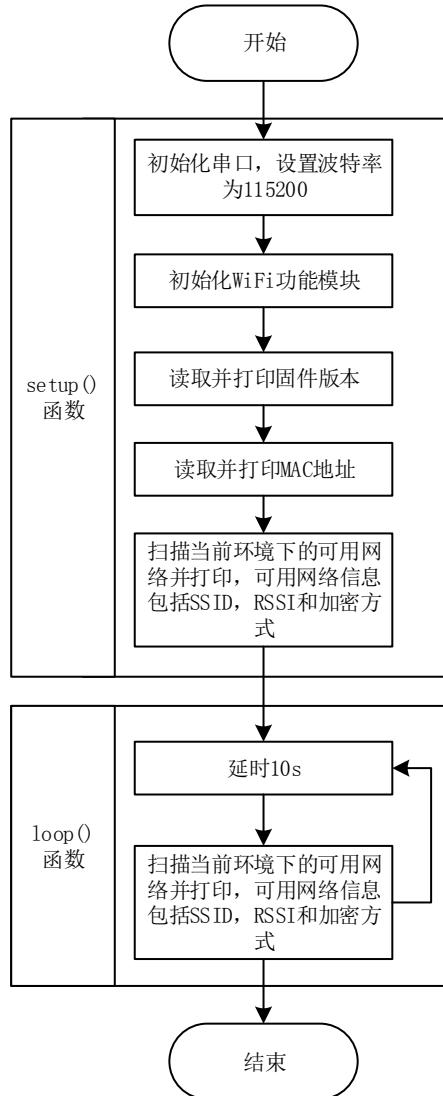


图 4-62 Lab-16 流程图

#### 4.8.2.4 实验现象

点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，点击  打开串口监视器，设置波特率为 115200。程序将每隔 10s 中打印当前环境下扫描到的可用网络信息：SSID、RSSI、加密方式。实验现象如下



图 4-63 Lab-16 实验现象

#### 4.8.3 Lab-17 网络连接

##### 4.8.3.1 实验准备

实验例程实现 CC3200 LaunchPad 连接至当前环境下可用的网络（热点：AP），并通过串口监视器显示当前连接的网络信息和获取到的 IP 地址。

CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80MHz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。

##### 4.8.3.2 代码解释

实验例程选择 Energia 提供的 Example: ConnectWithWPA，其可通过“文件->示例->WiFi->ConnectWithWPA”打开。完整代码看 4.8.3.3。

例程使用串口功能打印程序运行过程中的状态数据，在 setup() 函数内部使用 WiFi.begin() 函数初始化 WiFi 功能模块，并完成网络连接。在 WiFi.begin() 函数内部调用了 WiFi.init() 函数，所以在 setup() 函数上无需显示调用该函数。用户可以查看安装目录下的 WiFi.cpp 文件中 begin() 函数的定义。WiFi.begin() 函数调用完成后，程序将判断 CC3200 是否成功连接至网络（AP），如果连接失败程序将进入“死循环”：通过串口打印等待标记“.”，然后延时 300ms，直至连接成功退出“死循环”。网络连接成功后程序获取 IP 地址和当前网络信息并打印至 PC 串口监视器。

在 loop() 函数内，例程定时打印当前连接的网络信息至 PC 串口监视器。

例程使用到的 WiFi 库函数介绍如下：WiFi 为 WiFi 类（WiFiClass）的对象。例程中使用的函数如：WiFi.macAddress()、WiFi.RSSI()、WiFi.encryptionType() 可在 4.8.2.2 节查看。

函数 WiFi.begin() 有多个重载函数，本节仅介绍例程使用到的 WiFi.begin() 函数中的其中一个重载函数，如表 4-50 所示。

表 4-50 函数 begin()

原型	<b>int begin(char* ssid, char *passphrase)</b>
参数	ssid：指向 ssid（网络名字）缓存的指针 passphrase：指向网络密码缓存的指针
返回值	连接状态信息
说明	连接 ssid 指定的网络，密码由 passphrase 指定
实例	WiFi.begin("energia","launchpad")

表 4-51 函数 status()

原型	<b>uint8_t status()</b>
参数	无
返回值	网络连接的状态
说明	函数返回网络连接的状态信息
实例	uint8_t status = WiFi.status()

表 4-52 函数 localIP()

原型	<b>IPAddress localIP()</b>
参数	无
返回值	获取的 IP 地址
说明	网络连接后，可使用该函数获取当前分配的 IP 地址
实例	IPAddress ip = WiFi.localIP()

表 4-53 函数 BSSID()

原型	<b>uint8_t* BSSID(uint8_t* bssid)</b>
参数	bssid：指向 bssid（所连网络（AP）的 MAC 地址）缓存的指针
返回值	指向 bssid 的指针
说明	函数返回当前所连网络（AP）的 MAC 地址
实例	byte bssid[6]; WiFi.BSSID(bssid);

实验例程在使用时需要将程序中的 ssid 和 password 数组修改成用户当前环境下可用的网络名字和网络密码，此处网络可以理解为路由器或热点。实验例程为：

```
// your network name also called SSID
char ssid[]="energia";
// your network password
char password[]="supersecret";
```

#### 4.8.3.3 完整代码和流程图

##### 完整代码

```
/*
This example connects to an unencrypted Wifi network.
Then it prints the MAC address of the Wifi BoosterPack / LaunchPad,
the IP address obtained, and other network details.

*/
#ifndef __CC3200R1M1RGC__
// Do not include SPI for CC3200 LaunchPad
#include <SPI.h>
#endif
#include <WiFi.h>

// your network name also called SSID
char ssid[]="energia";
// your network password
char password[]="supersecret";

void setup(){
//Initialize serial and wait for port to open:
Serial.begin(115200);

// attempt to connect to Wifi network:
Serial.print("Attempting to connect to Network named: ");
// print the network name (SSID);
Serial.println(ssid);
// Connect to WPA/WPA2 network. Change this line if using open or WEP
network:
WiFi.begin(ssid, password);
while( WiFi.status()!= WL_CONNECTED){
// print dots while we wait to connect
Serial.print(".");
delay(300);
}

Serial.println("\nYou're connected to the network");
Serial.println("Waiting for an ip address");

while(WiFi.localIP()== INADDR_NONE){
// print dots while we wait for an ip addresss
Serial.print(".");
delay(300);
}
```

```
}

Serial.println("\nIP Address obtained");

// you're connected now, so print out the status
printCurrentNet();
printWifiData();

}

void loop() {
// check the network connection once every 10 seconds:
delay(10000);
printCurrentNet();
}

void printWifiData() {
// print your WiFi IP address:
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);
Serial.println(ip);

// print your MAC address:
byte mac[6];
WiFi.macAddress(mac);
Serial.print("MAC address: ");
Serial.print(mac[5], HEX);
Serial.print(":");
Serial.print(mac[4], HEX);
Serial.print(":");
Serial.print(mac[3], HEX);
Serial.print(":");
Serial.print(mac[2], HEX);
Serial.print(":");
Serial.print(mac[1], HEX);
Serial.print(":");
Serial.println(mac[0], HEX);
}

void printCurrentNet() {
// print the SSID of the network you're attached to:
Serial.print("SSID: ");
Serial.println(WiFi.SSID());
```

```
// print the MAC address of the router you're attached to:  
byte bssid[6];  
WiFi.BSSID(bssid);  
Serial.print("BSSID: ");  
Serial.print(bssid[5], HEX);  
Serial.print(":");  
Serial.print(bssid[4], HEX);  
Serial.print(":");  
Serial.print(bssid[3], HEX);  
Serial.print(":");  
Serial.print(bssid[2], HEX);  
Serial.print(":");  
Serial.print(bssid[1], HEX);  
Serial.print(":");  
Serial.println(bssid[0], HEX);  
  
// print the received signal strength:  
long rssi = WiFi.RSSI();  
Serial.print("signal strength (RSSI):");  
Serial.println(rssi);  
  
// print the encryption type:  
byte encryption = WiFi.encryptionType();  
Serial.print("Encryption Type:");  
Serial.println(encryption, HEX);  
Serial.println();  
}
```

流程图:

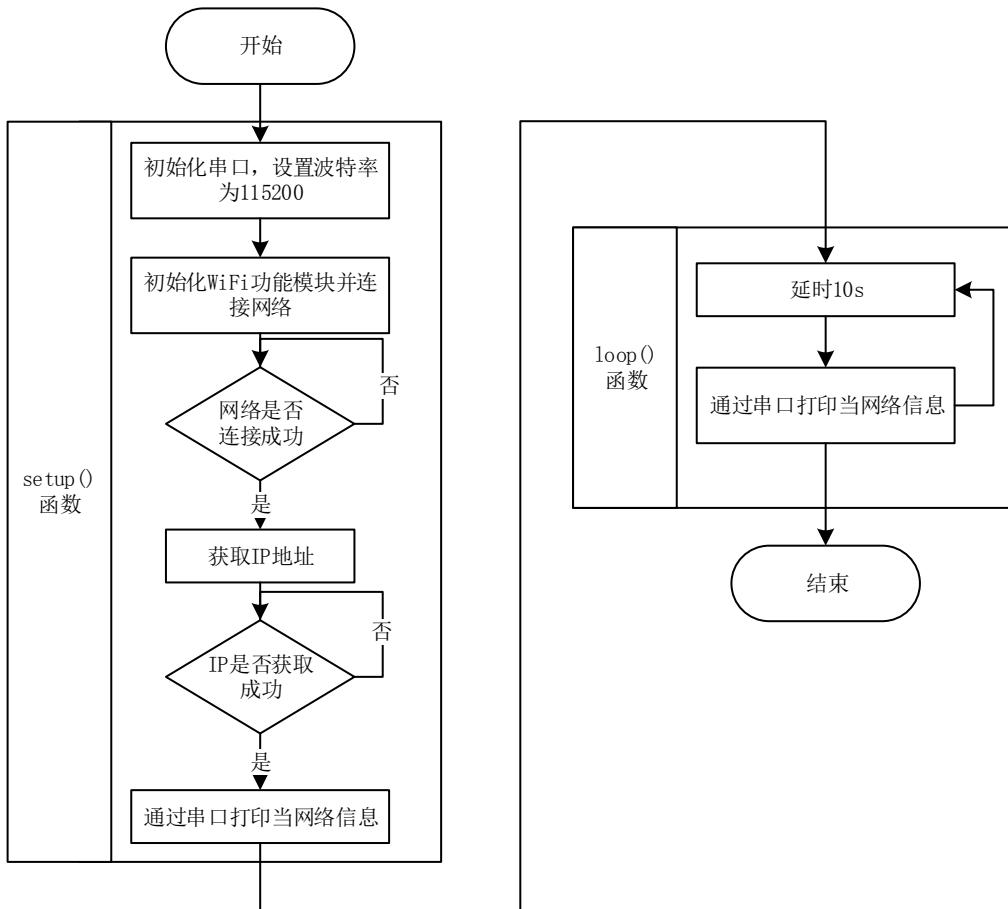


图 4-64 Lab-17 流程图

#### 4.8.3.4 实验现象

根据 4.8.3.2 中所述，将 ssid 和 password 改成可用网络后，点击 ，程序没有错误后即可点击 进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，点击 打开串口监视器，设置波特率为 115200。程序将每隔 10s 打印当前所联网络信息。实验现象如下。

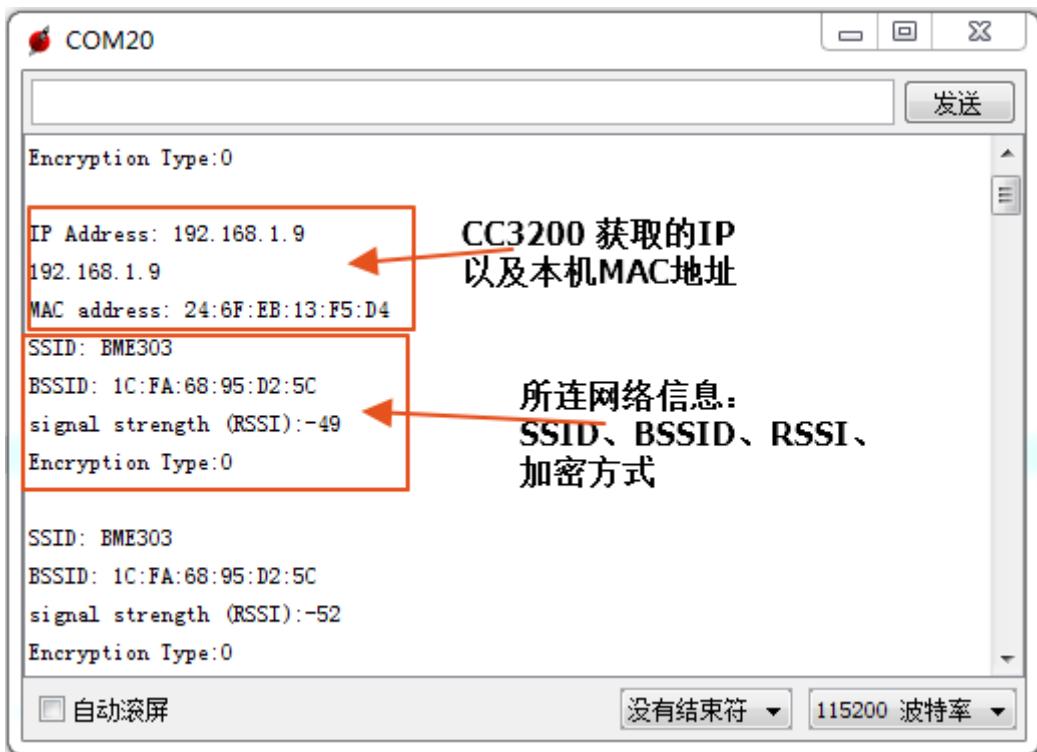


图 4-65 Lab17 实验现象

#### 4.8.4 Lab-18 web 客户端

##### 4.8.4.1 实验准备

实验例程实现 CC3200 LaunchPad 连接至当前环境下可用的网络（热点：AP），并且通过串口监视器显示当前连接的网络信息和获取到的 IP 地址；联网成功后 CC3200 LaunchPad 作为 webClient 连接 webServer: energia.nu，并发送 GET 请求；请求发送后 CC3200 Launchpad 读取 server 的响应信息并打印至串口监视器。

CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80MHz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。

##### 4.8.4.2 代码解释

实验例程选择 Energia 提供的 Example: WiFiWebClient，其可通过“文件->示例->WiFi->WiFiWebClient”打开。完整代码看 4.8.4.3。

例程使用串口功能打印程序运行过程中的状态数据，在 setup() 函数内部使用 WiFi.begin()

函数初始化 WiFi 功能模块，并完成网络连接。在 WiFi.begin() 函数内部调用了 WiFi.init() 函数，所以在 setup() 函数上无需显示调用该函数。用户可以查看安装目录下的 WiFi.cpp 文件中 begin() 函数的定义。WiFi.begin() 函数调用完成后，程序将判断 CC3200 是否成功连接至网络（AP），如果连接失败程序将进入“死循环”：通过串口打印等待标记“.”，然后延时 300ms，直至连接成功退出“死循环”。网络连接成功后程序获取 IP 地址和当前网络信息并打印至 PC 串口监视器。最后 CC3200 LaunchPad 作为 webClient 连接 webServer 并发送 GET 请求。

在 loop() 函数内，例程等待 webServer 的响应信息，接收到响应信息后例程将数据传输至 PC 端并通过串口监视器展示；如果 webServer 端断开了与 CC3200 LaunchPad 的连接，例程将关闭网络连接。

例程使用到的 WiFi 库函数介绍如下：WiFi 为 WiFi 类（WiFiClass）的对象。例程中使用的函数如：WiFi.macAddress()、WiFi.RSSI()、WiFi.encryptionType() 可在 4.8.2.2 节查看；WiFi.begin()、WiFi.status()、WiFi.localIP() 可在 4.8.3.2 节查看。例程实现时还使用到了 WiFiClient 类，该类包含在 WiFi 库中，其中例程使用的 client 即为 WiFiClient 类的对象。

WiFiClient 类中成员函数 connect() 具有多个重载函数，本节仅简述例程总使用到的函数。

表 4-54 函数 connect()

原型	<b>virtual int connect(const char *host, uint16_t port)</b>
参数	host: 指向 server hostname 的指针 port: server 的端口号
返回值	连接状态: false, 连接失败; true, 连接成功
说明	连接由参数 host 和 port 指定的 server
实例	int status = client.connect("energia.nu", 80)

表 4-55 函数 available()

原型	<b>virtual int available()</b>
参数	无
返回值	接收缓存区中的数据个数
说明	函数返回接收缓存区中的数据个数，该函数可与 read() 函数一起使用，实现数据读取功能。
实例	while(client.available()) { Serial.print(client.read()); }

WiFiClient 类中成员函数 read() 有多个重载函数，本节仅介绍例程中使用的 read() 函数

表 4-56 函数 read()

原型	<b>virtual int read()</b>
参数	无
返回值	读取到的数据
说明	函数返回接收缓存区中的数据
实例	while(client.available()) { Serial.print(client.read()); }

表 4-57 函数 connected()

<b>原型</b>	<code>virtual uint8_t connected()</code>
<b>参数</b>	无
<b>返回值</b>	连接状态: true: 已连接, false: 连接已断开
<b>说明</b>	函数返回当初 client 的网络连接状态,该函数可以与 connect()函数一起使用实现网络重连, 也可以与 stop()函数一起使用实现 client 关闭网络操作。
<b>实例</b>	<pre>if(!client.connected()) {     client.stop(); }</pre>

表 4-58 函数 stop()

<b>原型</b>	<code>virtual stop()</code>
<b>参数</b>	无
<b>返回值</b>	无
<b>说明</b>	关闭网络连接
<b>实例</b>	<pre>if(!client.connected()) {     client.stop(); }</pre>

#### 4.8.4.3 完整代码和流程图

完整代码:

```
#include <WiFi.h>

// your network name also called SSID
char ssid[]="energia";
// your network password
char password[]="lanuchpad";

// if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
//IPAddress server(50,62,217,1); // numeric IP for Google (no DNS)
char server[]="energia.nu";// name address for Google (using DNS)

// Initialize the Ethernet client library
// with the IP address and port of the server
// that you want to connect to (port 80 is default for HTTP):
WiFiClient client;
```

```
void setup() {
//Initialize serial and wait for port to open:
Serial.begin(115200);

// attempt to connect to Wifi network:
Serial.print("Attempting to connect to Network named: ");
// print the network name (SSID);
Serial.println(ssid);
// Connect to WPA/WPA2 network. Change this line if using open or WEP
network:
WiFi.begin(ssid, password);
while( WiFi.status()!= WL_CONNECTED) {
// print dots while we wait to connect
Serial.print(".");
delay(300);
}

Serial.println("\nYou're connected to the network");
Serial.println("Waiting for an ip address");

while(WiFi.localIP()== INADDR_NONE) {
// print dots while we wait for an ip addresss
Serial.print(".");
delay(300);
}
Serial.println("\nIP Address obtained");
printWifiStatus();

Serial.println("\nStarting connection to server...");
// if you get a connection, report back via serial:
if(client.connect(server,80)){
    Serial.println("connected to server");
// Make a HTTP request:
client.println("GET /hello.html HTTP/1.1");
client.println("Host: energia.nu");
client.println("Connection: close");
client.println();
}
}

void loop() {
// if there are incoming bytes available
```

```
// from the server, read them and print them:  
while(client.available()) {  
    char c = client.read();  
    Serial.write(c);  
}  
  
// if the server's disconnected, stop the client:  
if(!client.connected()) {  
    Serial.println();  
    Serial.println("disconnecting from server.");  
    client.stop();  
  
// do nothing forevermore:  
while(true);  
}  
}  
  
void printWifiStatus() {  
// print the SSID of the network you're attached to:  
    Serial.print("SSID: ");  
    Serial.println(WiFi.SSID());  
  
// print your WiFi shield's IP address:  
    IPAddress ip = WiFi.localIP();  
    Serial.print("IP Address: ");  
    Serial.println(ip);  
  
// print the received signal strength:  
    long rssi = WiFi.RSSI();  
    Serial.print("signal strength (RSSI):");  
    Serial.print(rssi);  
    Serial.println(" dBm");  
}
```

流程图:

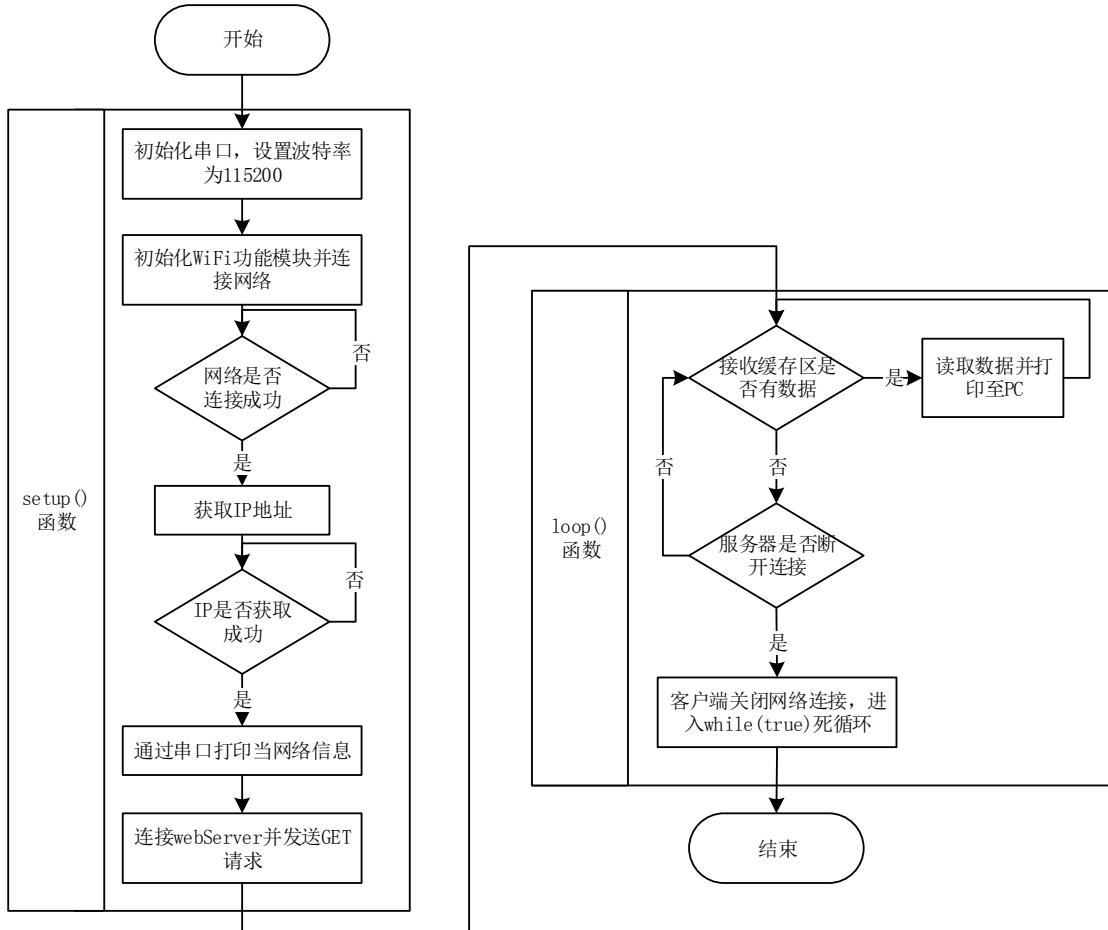


图 4-66 Lab18 流程图

#### 4.8.4.4 实验现象

根据 4.8.3.2 中所述，将 ssid 和 password 改成可用网络后，点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，点击  打开串口监视器，设置波特率为 115200。例程将打印网络连接过程信息以及 GET 请求的响应数据。实验现象如下

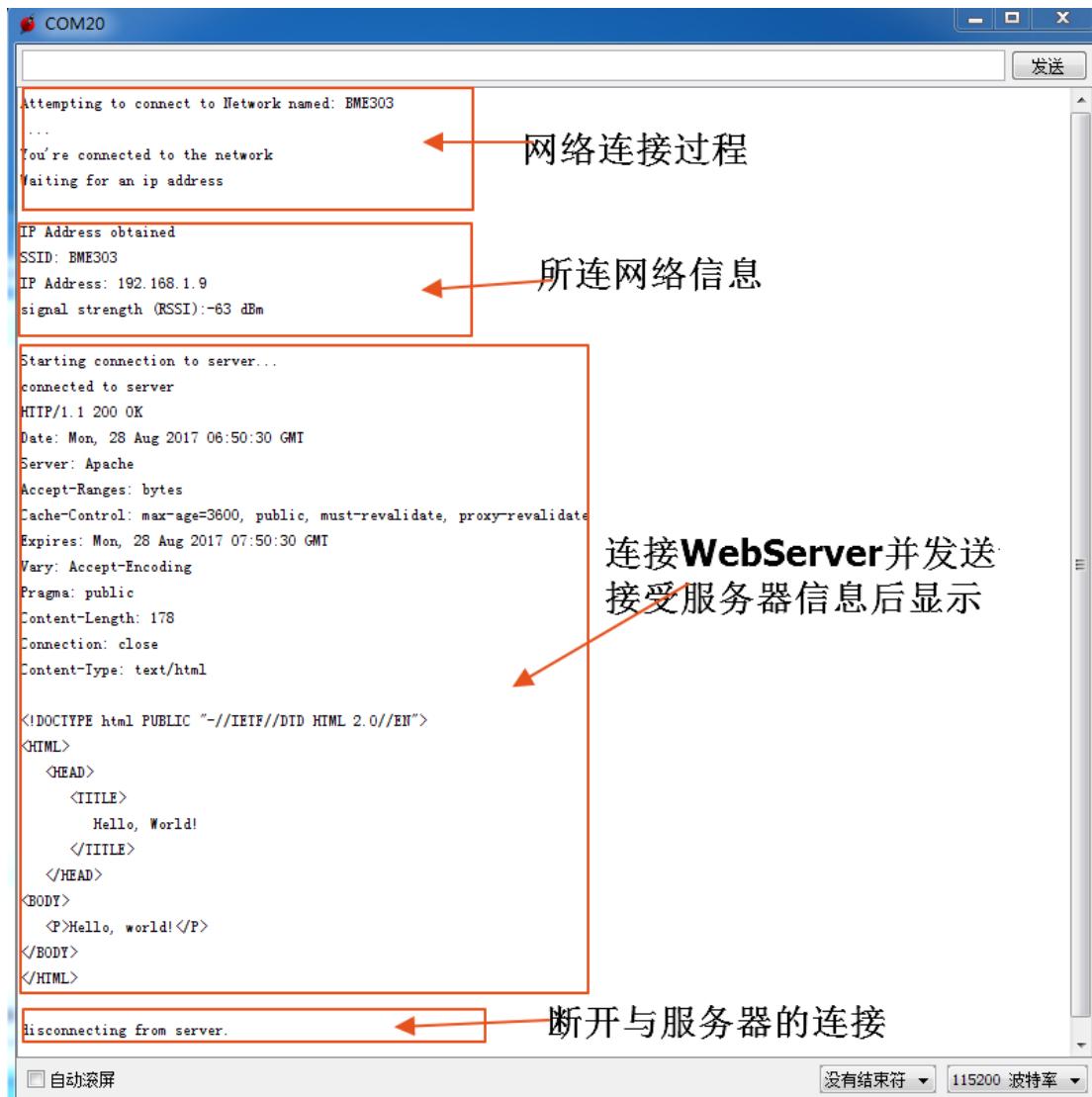


图 4-67 Lab-18 实验现象

#### 4.8.5 Lab-19 web 服务端

##### 4.8.5.1 实验准备

实验例程实现 CC3200 LaunchPad 连接至当前环境下可用的网络（热点：AP），并且通过串口监视器显示当前连接的网络信息和获取到的 IP 地址；联网成功后 CC3200 LaunchPad 开启 webServer 服务，接收客户端（一般为浏览器）发送的 HTTP 请求；CC3200 Launchpad 接收到请求后进行解析然后回发响应信息至客户端。

CC3200 LaunchPad 根据图 2-2 进行连接，保证程序能够正常烧写，通过板载 USB 口连接至 PC。选择开发板为“CC3200-LAUNCHXL(80MHz)”，端口根据图 2-15 进行选择，即选择“COM20”（实际端口需用户查看设备管理器确定）。

#### 4.8.5.2 代码解释

实验例程选择 Energia 提供的 Example: WiFiWebServer，其可通过“文件->示例->WiFi->WiFiWebClient”打开。完整代码看 4.8.4.3。

例程使用串口功能打印程序运行过程中的状态数据，在 `setup()` 函数内部使用 `WiFi.begin()` 函数初始化 WiFi 功能模块，并完成网络连接。在 `WiFi.begin()` 函数内部调用了 `WiFi.init()` 函数，所以在 `setup()` 函数上无需显示调用该函数。用户可以查看安装目录下的 `WiFi.cpp` 文件中 `begin()` 函数的定义。`WiFi.begin()` 函数调用完成后，程序将判断 CC3200 是否成功连接至网络（AP），如果连接失败程序将进入“死循环”：通过串口打印等待标记“.”，然后延时 300ms，直至连接成功退出“死循环”。网络连接成功后程序获取 IP 地址和当前网络信息并打印至 PC 串口监视器。最后 CC3200 LaunchPad 开启 `webServer` 服务。

在 `loop()` 函数内，例程监听客户端连接。当有客户端连接时，一般为浏览器通过 IP 地址访问 CC3200 LaunchPad 中的 `webServer`，服务器接收到新客户端连接请求后，将数据封装成 HTTP 报文回发至客户端，完成一次请求响应过程。

例程使用到的 WiFi 库函数介绍如下：WiFi 为 WiFi 类（`WiFiClass`）的对象。例程中使用的函数如：`WiFi.macAddress()`、`WiFi.RSSI()`、`WiFi.encryptionType()` 可在 4.8.2.2 节查看；`WiFi.begin()`、`WiFi.status()`、`WiFi.localIP()` 可在 4.8.3.2 节查看。例程实现时使用了 WiFiClient 类：`client` 为 WiFiClient 对象，函数 `client.connected()`、`client.available()`、`client.read()`、`client.stop()` 可在 4.8.4.3 节查看。例程实现 `webServer` 功能，故需使用 WiFiServer 类：例程中的 `server` 为 WiFiServer 的对象，对应函数介绍如下：

表 4-59 函数 `begin()`

<b>原型</b>	<code>void begin()</code>
<b>参数</b>	无
<b>返回值</b>	无
<b>说明</b>	开启 <code>webServer</code> 服务
<b>实例</b>	<code>server.begin()</code>

<b>原型</b>	<code>WiFiClient available(uint8_t* status = NULL)</code>
<b>参数</b>	<code>status</code> : 默认值为 NULL，在函数实现中也未使用
<b>返回值</b>	WiFiClient 对象
<b>说明</b>	函数返回 WiFiClient 对象
<b>实例</b>	<code>WiFiClient client = server.available()</code>

#### 4.8.5.3 完整代码和流程图

完整代码：

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiServer.h>
```

```
// your network name also called SSID
char ssid[]="networkname";
// your network password
char password[]="password";
// your network key Index number (needed only for WEP)
int keyIndex =0;

WiFiServer server(80);

void setup(){
    Serial.begin(115200); // initialize serial communication
    pinMode(RED_LED, OUTPUT); // set the LED pin mode

    // attempt to connect to Wifi network:
    Serial.print("Attempting to connect to Network named: ");
    // print the network name (SSID):
    Serial.println(ssid);
    // Connect to WPA/WPA2 network. Change this line if using open or WEP
    // network:
    WiFi.begin(ssid, password);
    while( WiFi.status() != WL_CONNECTED){
        // print dots while we wait to connect
        Serial.print(".");
        delay(300);
    }

    Serial.println("\nYou're connected to the network");
    Serial.println("Waiting for an ip address");

    while(WiFi.localIP()== INADDR_NONE){
        // print dots while we wait for an ip addresss
        Serial.print(".");
        delay(300);
    }

    // you're connected now, so print out the status
    printWifiStatus();

    Serial.println("Starting webserver on port 80");
    server.begin(); // start the web server on port 80
    Serial.println("Webserver started!");
}
```

```
void loop() {
// listen for incoming clients
    WiFiClient client = server.available();
    if(client){
        Serial.println("new client");
        // an http request ends with a blank line
        boolean currentLineIsBlank = true;
        while(client.connected()){
            if(client.available()){
                char c = client.read();
                Serial.write(c);
                // if you've gotten to the end of the line (received a newline
                // character) and the line is blank, the http request has ended,
                // so you can send a reply
                if(c =='\n' && currentLineIsBlank){
                    // send a standard http response header
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-Type: text/html");
                    client.println("Connection: close");// the connection will
                    be closed after completion of the response
                    client.println("Refresh: 5");// refresh the page
                    automatically every 5 sec
                    client.println();
                    client.println("<!DOCTYPE HTML>");
                    client.println("<html>");
                    // output the value of each analog input pin
                    for(int analogChannel =0; analogChannel <6; analogChannel++){
                        int sensorReading = analogChannel;//analogRead(analogChannel);
                        client.print("analog input ");
                        client.print(analogChannel);
                        client.print(" is ");
                        client.print(sensorReading);
                        client.println("<br />");
                    }
                    client.println("</html>");
                    break;
                }
                if(c =='\n'){
                    // you're starting a new line
                    currentLineIsBlank = true;
                }
                elseif(c !='\r'){

```

```
// you've gotten a character on the current line
    currentLineIsBlank = false;
}
}
}

// give the web browser time to receive the data
delay(1);
// close the connection:
client.stop();
Serial.println("client disconnected");
}

}

void printWifiStatus(){
// print the SSID of the network you're attached to:
Serial.print("Network Name: ");
Serial.println(WiFi.SSID());
// print your WiFi shield's IP address:
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);
// print the received signal strength:
long rssi = WiFi.RSSI();
Serial.print("signal strength (RSSI):");
Serial.print(rssi);
Serial.println(" dBm");
}
```

流程图:

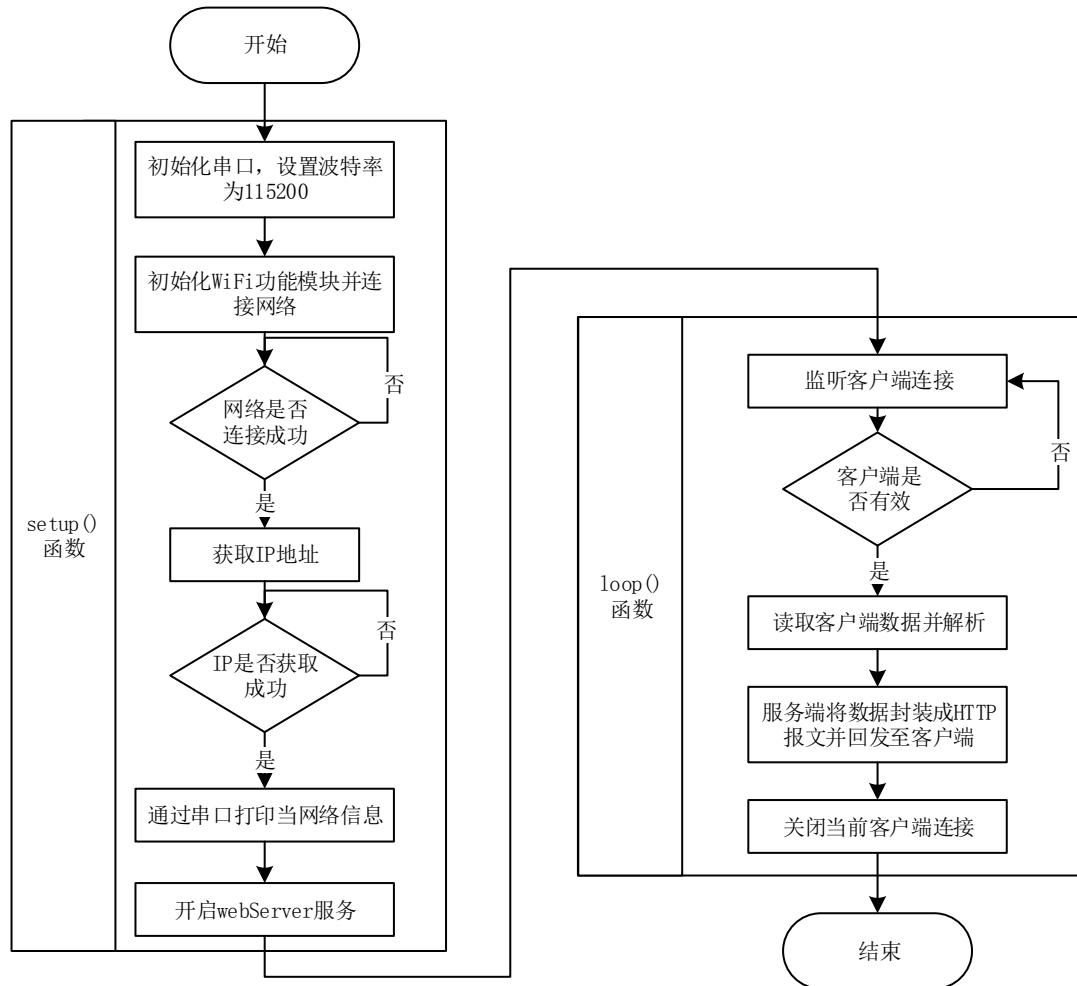
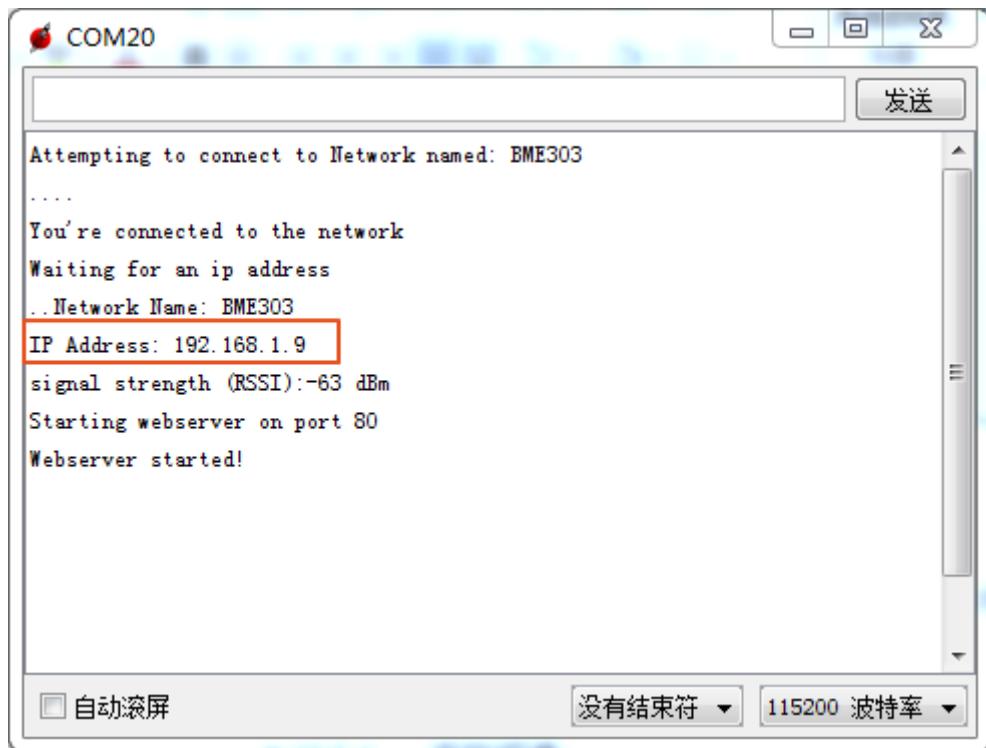


图 4-68 Lab-19 流程图

#### 4.8.5.4 实验现象

根据 4.8.3.2 中所述，将 ssid 和 password 改成可用网络后，点击 ，程序没有错误后即可点击  进行编译烧写程序至 CC3200 LaunchPad，烧写完成后程序将自动运行，点击  打开串口监视器，设置波特率为 115200。打开任意浏览器，输入串口监视器中的 IP Address，浏览器将接收到例程发送的数据。实验现象如下



```

Attempting to connect to Network named: BME303
.....
You're connected to the network
Waiting for an ip address
.. Network Name: BME303
IP Address: 192.168.1.9
signal strength (RSSI):-63 dBm
Starting webserver on port 80
Webserver started!

```

图 4-69 Lab-19 开始 Webserver

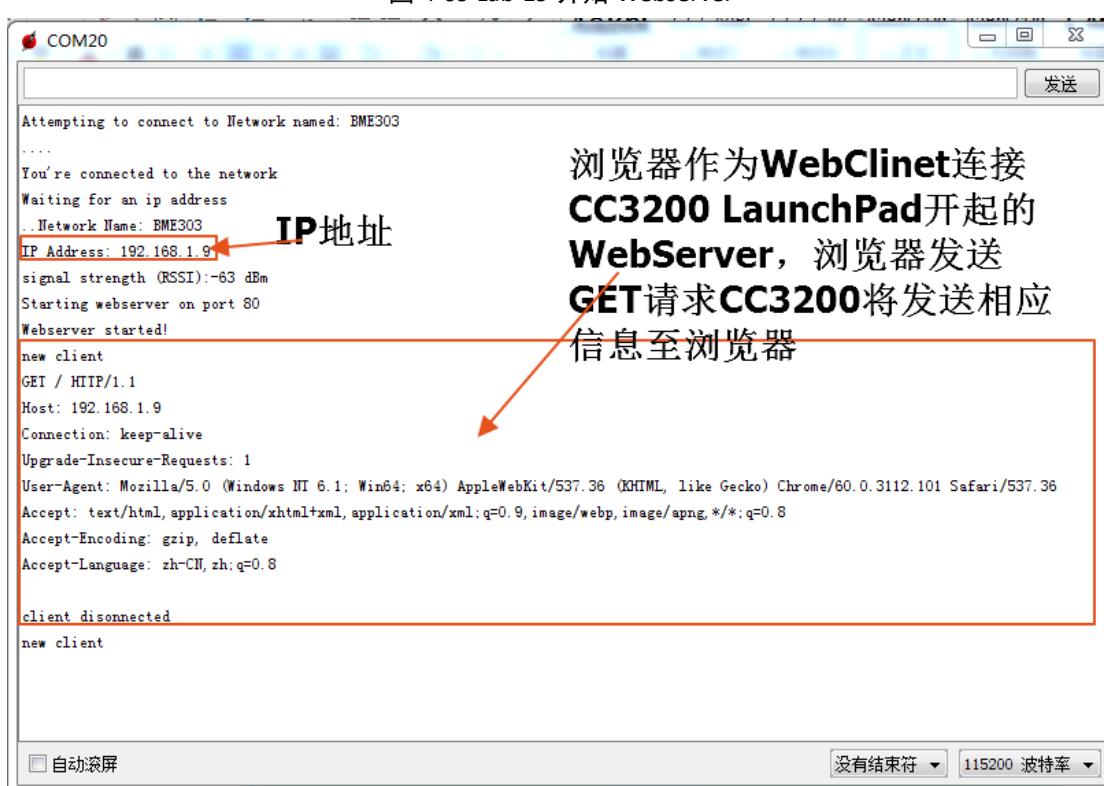


图 4-70 Lab-19 WebClient 连接过程信息

IP地址

浏览器作为WebClinet连接  
CC3200 LaunchPad开起的  
WebServer, 浏览器发送  
GET请求CC3200将发送相应  
信息至浏览器

```

Attempting to connect to Network named: BME303
.....
You're connected to the network
Waiting for an ip address
.. Network Name: BME303
IP Address: 192.168.1.9
signal strength (RSSI):-63 dBm
Starting webserver on port 80
Webserver started!
new client
GET / HTTP/1.1
Host: 192.168.1.9
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8

client disconnected
new client

```

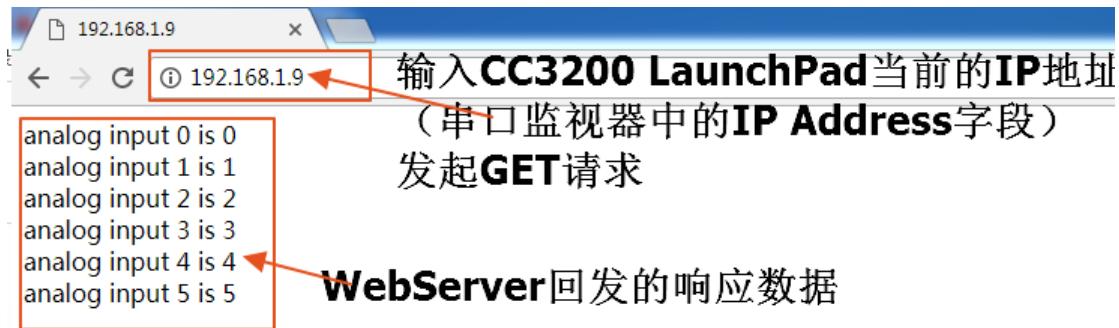


图 4-71 Lab-19 WebClient 信息

## 5 实验套件 AY-IOT Kit 实验例程

本章围绕实验套件 AY-IOT Kit 展开，介绍套件所有模块的实验例程。

### 5.1 套件资源

#### 5.1.1 套件源代码

从艾研信息官方网站 [www.hpati.com](http://www.hpati.com) 上下载例程代码，将代码放进 Energia 相应库文件目录下后可直接使用。操作可查看视频资源《Getting Started Guide》，下面也简单介绍操作过程：

1、进入网站下载例程代码 ([http://www.hpati.com/ay\\_source\\_download/](http://www.hpati.com/ay_source_download/))；

当前位置：首页 > 资源中心 > 代码类					
名称	大小	时间	权限	下 载	
AY-IOT KIT Datasheet	12 MB	2016-12-21	公开		
物联网实验开发套件AY-IOT KIT程序	35 KB	2017-01-10	公开		
AY_SEB kit for CCS6.5 例程.rar	6 MB	2016-12-21	公开		
AY_SEB分模块程序	7 MB	2015-09-28	公开		

图 5-1 例程

2、下载例程解压，每个例程包含对应模块的源代码已经 examples：

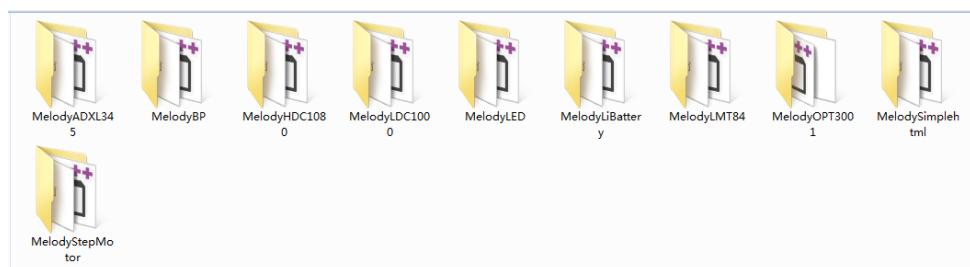


图 5-2 例程文件

3、将解压后所有的文件（即图 5-2 中的所有文件）复制进 CC3200 LaunchPad 开发包安装路径，其路径为【 C:\Users\ 用户名 (PC 用户名，根据用户实而不同)】

\AppData\Local\Energia15\packages\energia\hardware\cc3200\1.0.2\cores\cc3200】，

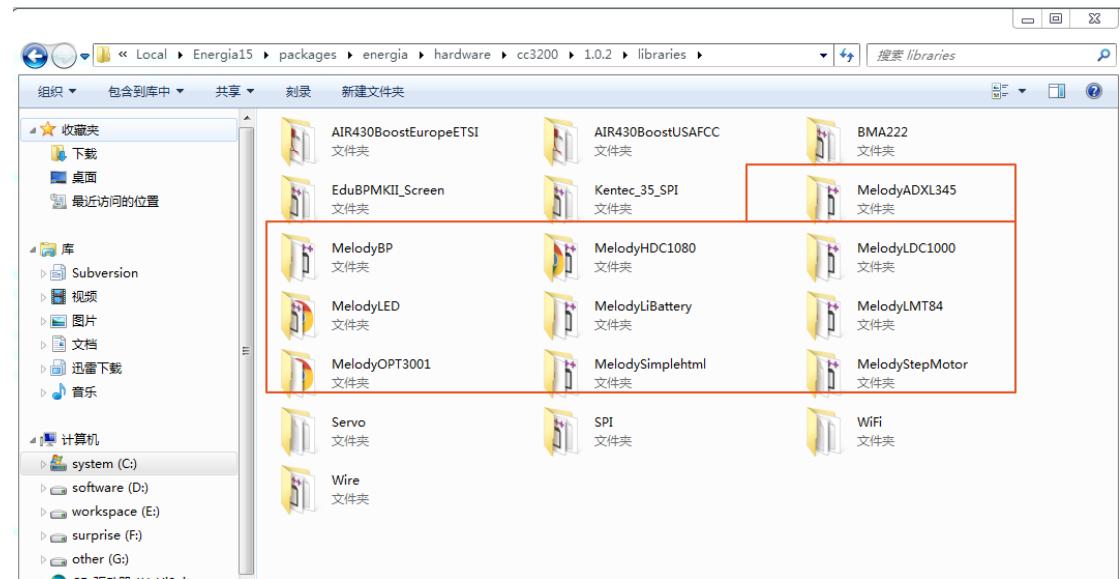


图 5-3 库文件目录

4、打开 Energia，在菜单栏 Tools->Board 下选择 Launchpad CC3200 w/cc3200 (80MHz)

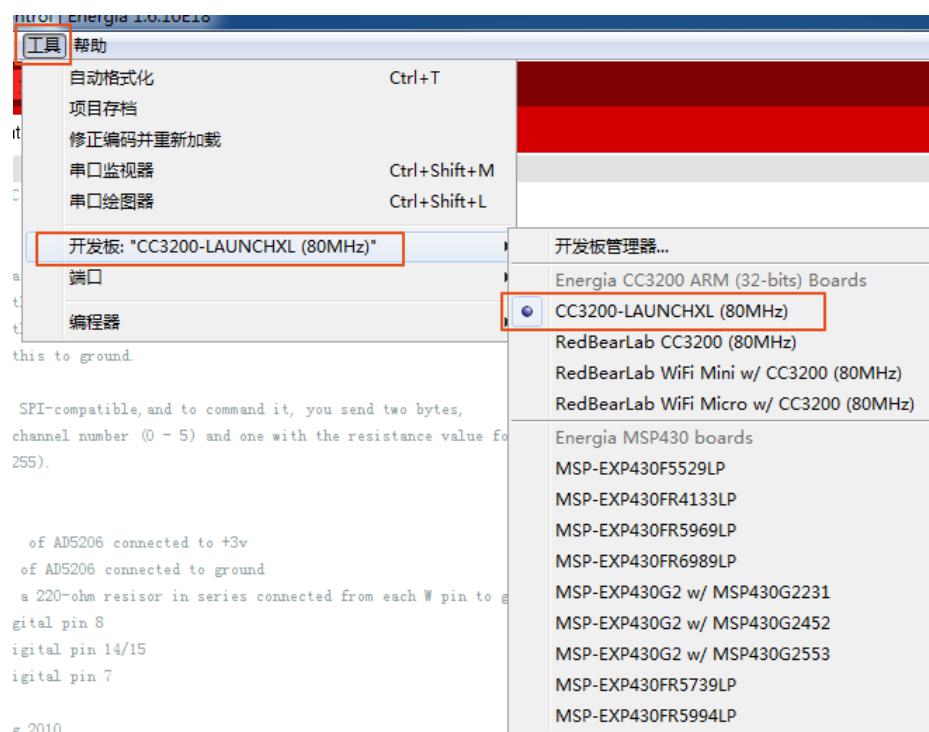


图 5-4 Tools 下选择 Board

5、查看链接进入的库文件文件

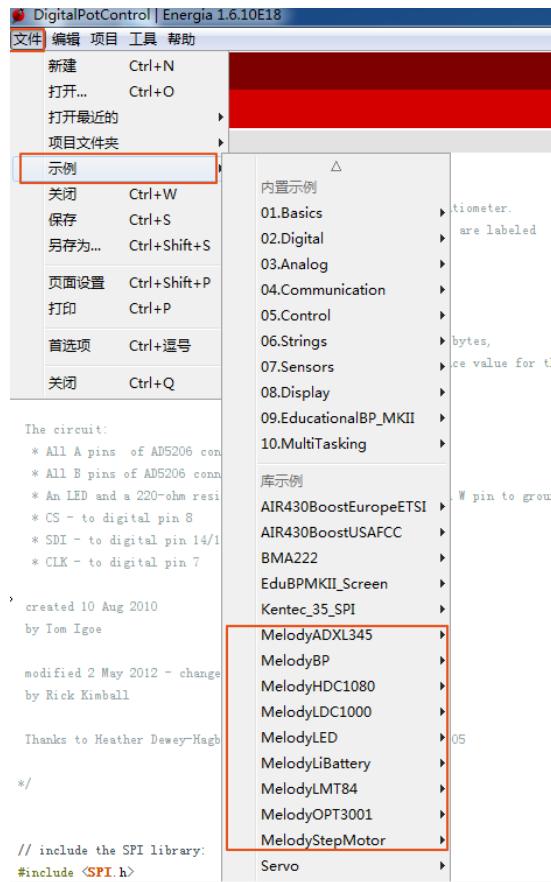


图 5-5 sketch 下查看库文件

## 6、查看 examples



图 5-6 File 下查看 Examples

7、选择一个例程编译后即可下载使用。

### 5.1.2 套件使用视频

在艾研官网上可直接观看，地址：<http://www.hpati.com/IOTKIT/>

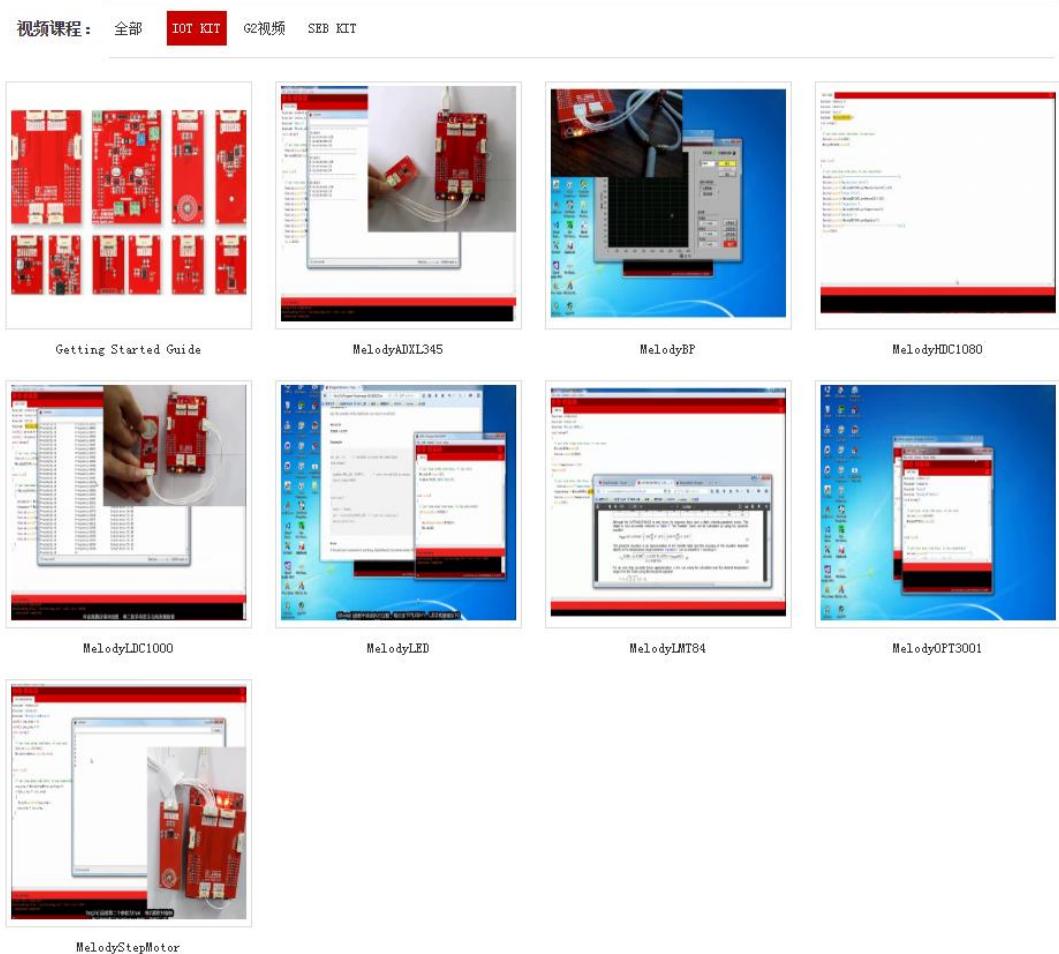


图 5-7 视频资源

### 5.1.3 套件可完成的实验

模块名称	器件	例程	功能
三轴加速度 模块	ADXL345	ADXL345	使用 I2C 通信完成 ADXL345 寄存器数据的读写 读取 XYZ 三轴数据
		SimpleWebADXL345	使用 I2C 通信完成 ADXL345 寄存器数据的读写 读取 XYZ 三轴数据 自动获取 IP 地址，开启 Web 服务，实现网络通信

			实现简单 HTTP 报文的发送和分析 可通过 Web 客服端（浏览器）读取三轴数据
温湿度传感器模块	HDC1080	HDC1080	使用 I2C 通信完成 HDC1080 寄存器数据的读写 读取温湿度数据
		SimpleWebHDC1080	使用 I2C 通信完成 HDC1080 寄存器数据的读写 读取并转换温湿度数据 自动获取 IP 地址，开启 Web 服务，实现网络通信 实现简单 HTTP 报文的发送和分析 可通过 Web 客服端（浏览器）读取温湿度数据
模拟温度模块	LMT84	LMT84	读取并转换温度数据
		SimpleWebLMT84	读取并转换温度数据 自动获取 IP 地址，开启 Web 服务，实现网络通信 实现简单 HTTP 报文的发送和分析 可通过 Web 客服端（浏览器）读取温度数据
光感模块	OPT3001	OPT3001	使用 I2C 通信完成 OPT3001 寄存器数据的读写 读取并转换光强数据
		SimpleWebOPT3001	使用 I2C 通信完成 OPT3001 寄存器数据的读写 读取并转换光强数据 自动获取 IP 地址，开启 Web 服务，实现网络通信 实现简单 HTTP 报文的发送和分析 可通过 Web 客服端（浏览器）读取光强数据
高亮 LED 驱动模块	TPS61043 LED	LED	使用 PWM 控制 LED 亮度
		LEDWithButton	使用按钮调节 PWM，改变 LED 亮度
		SimpleWeb LED	使用 PWM 控制 LED 亮度 自动获取 IP 地址，开启 Web 服务，实现网络通信 实现简单 HTTP 报文的发送和分析

			<p>析 可通过 Web 客服端（浏览器） 读取 LED 亮度当前数据 可通过 Web 客服端（浏览器） 调节 PWM，改变 LED 亮度</p>
步进电机模块	DRV8833 步进电机	SimpleStepMotor	<p>使用 PWM 控制步进电机转速和方向</p>
		SimpleWebStepMotor	<p>使用 PWM 控制步进电机转速和方向 自动获取 IP 地址，开启 Web 服务，实现网络通信 实现简单 HTTP 报文的发送和分析 可通过 Web 客服端（浏览器） 读取步进电机当前状态 可通过 Web 客服端（浏览器） 调节 PWM，改变步进电机转速和方向</p>
LDC1000 模块	LDC1000	LDC1000	<p>通过 SPI 完成 LDC1000 寄存器读写 读取接近数据 Proximity 和频率 Frequency Counter 计算电感</p>
		SimpleWebLDC1000	<p>通过 SPI 完成 LDC1000 寄存器读写 读取接近数据 Proximity 和频率 Frequency Counter 计算电感 自动获取 IP 地址，开启 Web 服务，实现网络通信 实现简单 HTTP 报文的发送和分析 可通过 Web 客服端（浏览器） 读取接近数据 Proximity 和频率 Frequency Counter 以及电感</p>
血压模块	MPS-3117	BloodPressureSample	<p>使用 AD 完成静态压和血压波动数据采集 完成与上位机程序通信</p>
电池管理模块	BQ24090 BQ27542	无	实现锂电池充电
		LiBattery	<p>使用 I2C 完成 BQ27542 寄存器读写 监控锂电池状态</p>

## 5.2 套件模块

本章将介绍各个实验例程中的知识点和注意事项：包括套件模块接口说明，原理介绍以及软件实现。

套件使用之前需要查看《AY-IOT 连接使用须知》

对应网址：[http://www.hpati.com/ay\\_doc\\_download/](http://www.hpati.com/ay_doc_download/)。

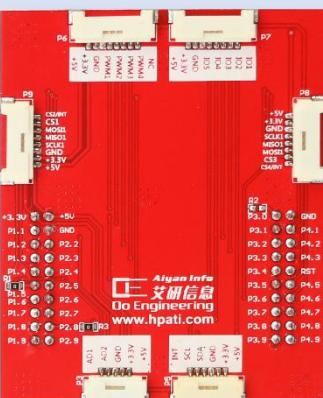
若未仔细了解《套件连接须知》，在之后的使用过程中将会产生异常状态：如 cc3200 出厂时排针处有部分端口未使用 0 欧姆电阻连接，而实验套件部分模块需要使用这些端口。

在下文介绍各个模块时假设用户已经了解《AY-IOT 连接使用须知》中的各个事项。

### 5.2.1 转接板

物联网使用套件各个模块和 cc3200 之间的连接需要使用到转接板。转接板将 cc3200 的 boostpack 端口映射成各传感器模块接口。转接板和对外端口如下：

表 5-1 转接板

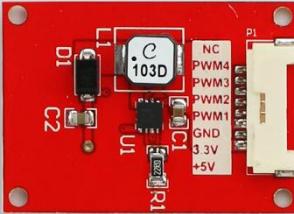
转接板	接口
	<p>1、1 对 BoosterPack 接口，连接 CC3200；      2、1 组 GPIO 接口，包含 5 各 IO 口；      3、1 组 PWM 接口，包含 4 各 PWM 口；      4、2 组 SPI 通信接口；      5、1 组 I2C 通信接口；      6、1 组 AD 接口，包含 2 各 AD 采集通道。</p>

### 5.2.2 高亮 LED 驱动模块 (MelodyLED)

#### 5.2.2.1 模块

高亮 LED 驱动模块使用 PWM 接口，模块如下

表 5-2 高亮 LED 驱动模块

高亮 LED 驱动模块	概况
	模块使用芯片 TPS61043 驱动 LED 使用 PWM 进行控制，可调节 LED 亮度

### 5.2.2.2 实现原理及原理图简介

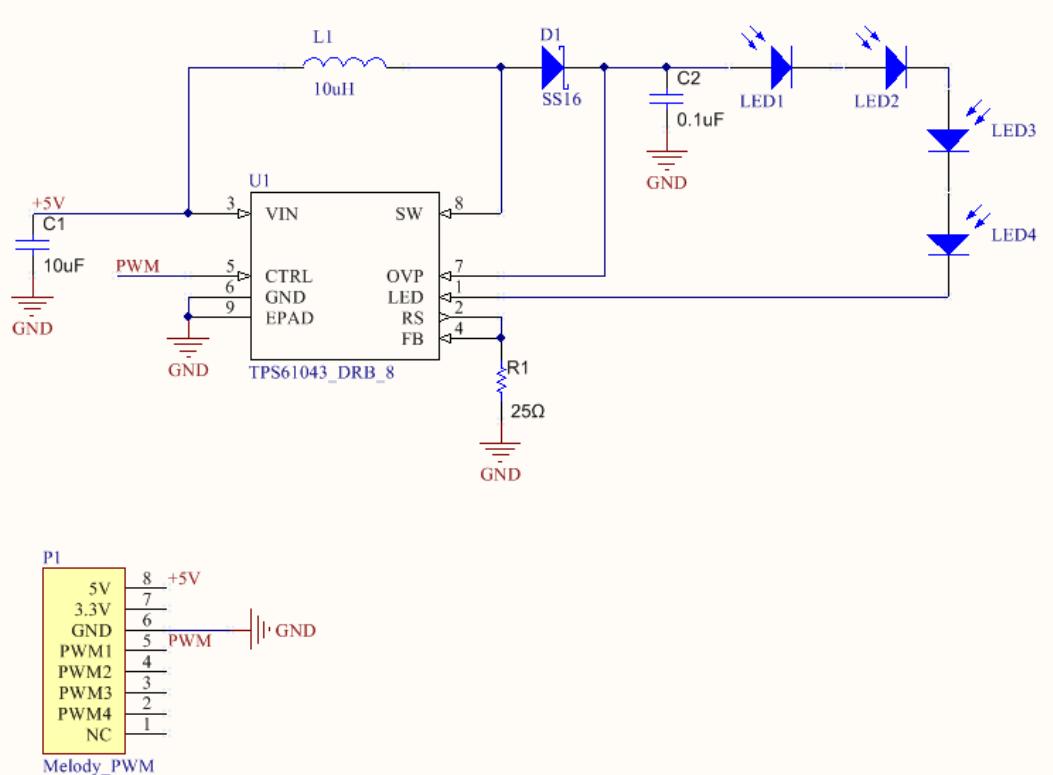


图 5-8 模块原理图

TPS61043 是 TI 的一款具有恒定电流输出的高频升压转换器，本设计中用于驱动 4 颗 LED 灯，LED 上的电流，通过外部检测电阻 R1 来设置，并由反馈引脚（FB）直接调节，FB 将传感电阻 RS 上的电压调节为 252 mV（典型值）。得  $I_{out}=252mV/R1$ 。

为了控制 LED 亮度，可以通过向控制引脚（CTRL）施加 100Hz 至 50kHz 的频率范围的 PWM（脉宽调制）信号来脉冲化 LED 电流。

### 5.2.2.3 软件实现

高亮 LED 驱动模块使用 PWM 驱动 LED, Energia 已经实现了 PWM 库函数, 用户直接使用即可。PWM 库函数可在 <http://energia.nu/reference/analogwrite/> 查看。模块留有 4 个 PWM 接口, 根据原理图可以发现, 模块实际仅使用了 PWM1 接口, 驱动库也是根据 PWM1 实现的。

表 5-3 analogWrite() 函数

Energia 库函数	void analogWrite(uint8_t pin,int val)
说明	指定 pin 端口输出 val/255 占空比, 频率固定为 490Hz 的 PWM
使用范例	<b>analogWrite(4,128)</b>

函数源代码可在【Energia 安装目录】->hardware->cc3200->cores->cc3200->wiring\_analog.c 中查看。实现如图 5-9:

```
void analogWrite(uint8_t pin,int val){
/* duty cycle(%) = val / 255;
 * Frequency of 490Hz specified by Arduino API */
uint8_t timer = digitalPinToTimer(pin);
if(timer == NOT_ON_TIMER)
return;
if(val ==0){
    pinMode(pin, OUTPUT);
    digitalWrite(pin, LOW);
return;
}
if(val >=255){
    pinMode(pin, OUTPUT);
    digitalWrite(pin, HIGH);
return;
}
PWMWrite(pin,255, val,490);
}
```

图 5-9 analogWrite() 库函数

### 5.2.2.3.1 LED 驱动库实现

驱动库使用 C++ 实现, 源代码可直接查看 MelodyLED 文件下的 Melody\_LED.h 和 Melody\_LED.cpp 文件。下面简单介绍库函数。

表 5-4 begin() 函数

驱动库函数	void begin()
说明	初始化 LED, 设置 LED 输出亮度为默认值 0
使用范例	MelodyLED.begin();

表 5-5 begin()函数

驱动库函数	<b>void begin(uint8_t bright)</b>
说明	初始化 LED，设置 LED 输出亮度为 bright，范围为 0~255
使用范例	MelodyLED.begin(10);

表 5-6 setBright()函数

驱动库函数	<b>void setBright(uint8_t bright)</b>
说明	设置 LED 亮度
使用范例	MelodyLED.setBright(20);

表 5-7 getBright 函数

驱动库函数	<b>uint8_t getBright()</b>
说明	获取 LED 亮度
使用范例	uint8_t bright = MelodyLED.getBright();

### 5.2.2.3.2 实验例程

实验例程操作及结果展示可浏览视频资料：[http://www.hpati.com/product\\_videos/v79.html](http://www.hpati.com/product_videos/v79.html)

例程 1，LED.ino：点亮 LED。例程代码如图 5-10，仅需要在 setup 函数中使用 MelodyLED.begin()函数即可。其中 MelodyLED 定义在 Melody\_LED.cpp 文件中。

```
#include <stdbool.h>
#include <stdint.h>
#include <Melody_LED.h>

void setup()
{
    // put your setup code here, to run once:
    MelodyLED.begin(10);
}

void loop()
{
    // put your main code here, to run repeatedly:
}
```

图 5-10 点亮 LED 例程

例程 2，LEDWithButton.ino：通过按键调节 LED 亮度。程序流程图如图 5-11 所示，程序代码如图 5-12 所示。

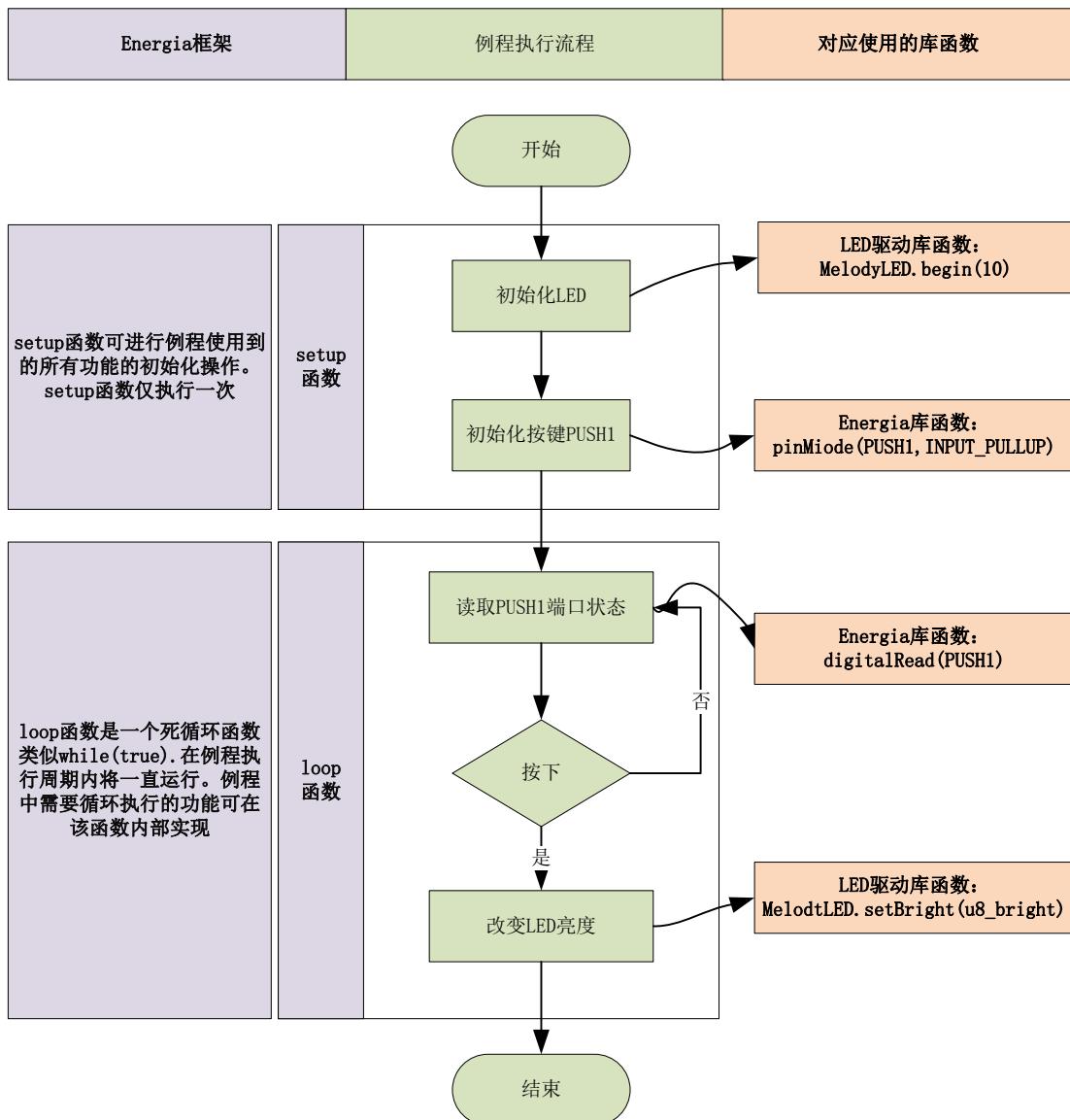


图 5-11LEDWithButton 例程实现流程图

```

#include <stdbool.h>
#include <stdint.h>
#include <Melody_LED.h>
uint8_t u8_bright =10;
void setup()
{
// put your setup code here, to run once:
    MelodyLED.begin(10);
    pinMode(PUSH1, INPUT_PULLUP);
}

void loop()
{

```

```
// put your main code here, to run repeatedly:
if(digitalRead(PUSH1))
{
while(digitalRead(PUSH1));
    u8_bright +=10;
    MelodyLED.setBright(u8_bright);
}
}
```

图 5-12 LEDWithButton 例程

例程 3,SimpleWebLED.ino:通过网络改变 LED 亮度。cc3200 设置为 webserver 模式，用户可是有浏览器访问 cc3200，实验过程中 cc3200 的 IP 地址通过串口打印至 PC 端。Energia 开发工具提供了串口监视工具。程序例程图如图 5-13 所示，网络驱动库函数可见 MelodySimplehtml 文件夹下的 Melody\_Simplehtml.h 和 Melody\_Simplehtml.cpp 文件。例程代码如图 5-14 所示。

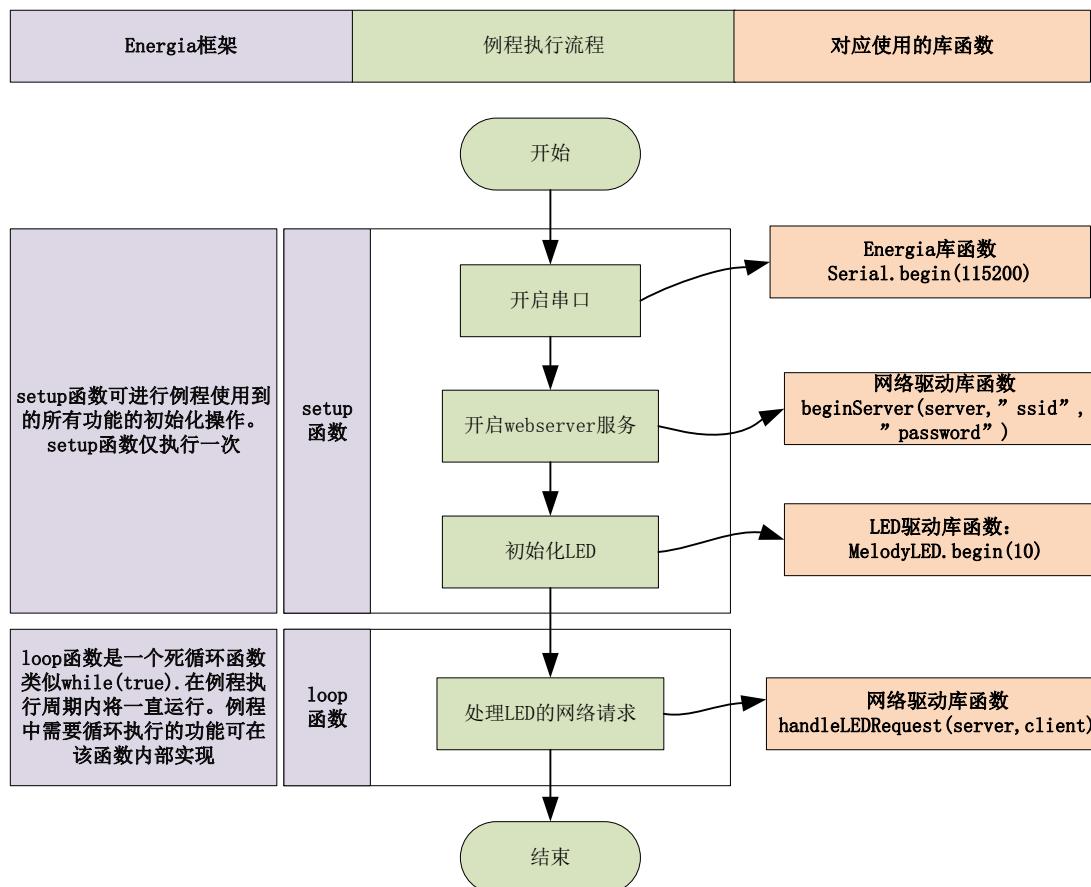


图 5-13 SimpleWebLED 例程流程图

```
#include <stdbool.h>
#include <stdint.h>
#include <WiFi.h>
#include <Melody_LED.h>
```

```
#include "Melody_Simplehtml.h"

WiFiServer server(80);
WiFiClient client;

void setup()
{
// put your setup code here, to run once:
Serial.begin(115200);
beginServer(server,"ssid","password");
MelodyLED.begin(10);
}

void loop()
{
handleLEDRequest(server,client);
}
```

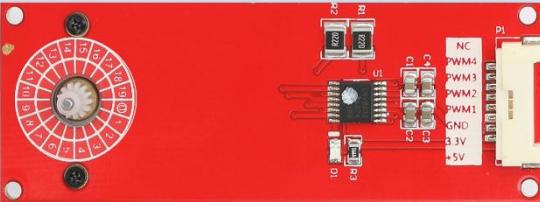
图 5-14SimpleWebLED 例程

### 5.2.3 步进电机模块 (*MelodyStepMotor*)

#### 5.2.3.1 模块

步进电机模块使用 PWM 接口进行驱动。模块如下

表 5-8 步进电机模块

步进电机模块	概况
	<p>1 模块使用 DRV8833 驱动步进电机      2 使用 4 路 PWM 进行驱动，可通过软件改变 PWM 的输出频率和 4 路 PWM 的对应相位，从而改变步进电机的工作速度和方向。</p>

### 5.2.3.2 实现原理及原理图简介

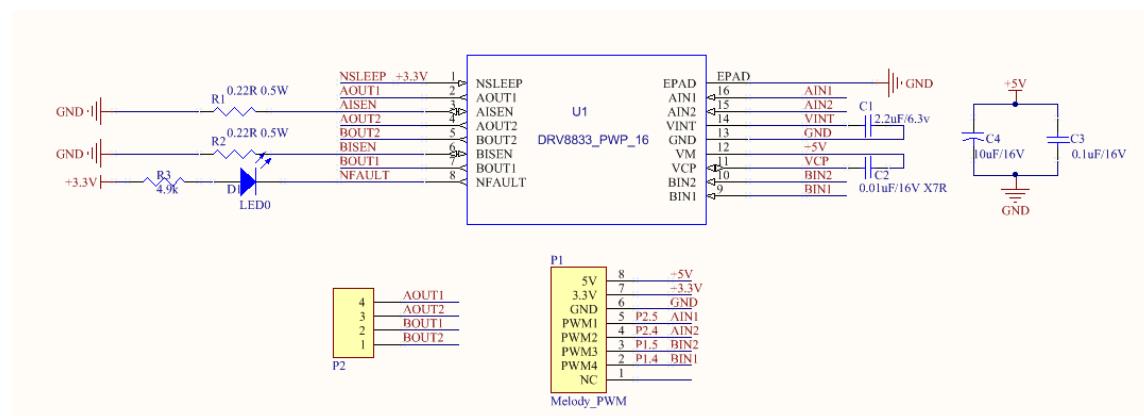


图 5-15 步进电机模块原理图

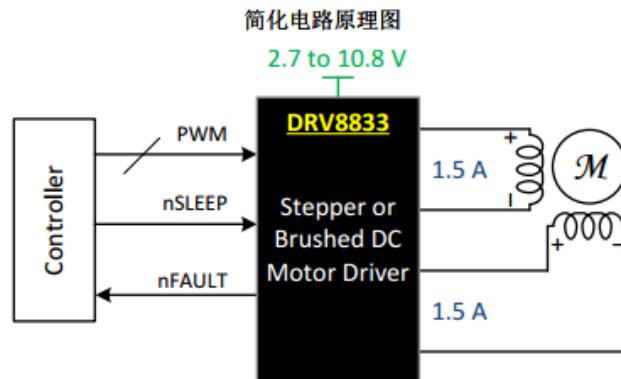


图 5-16 简化原理图

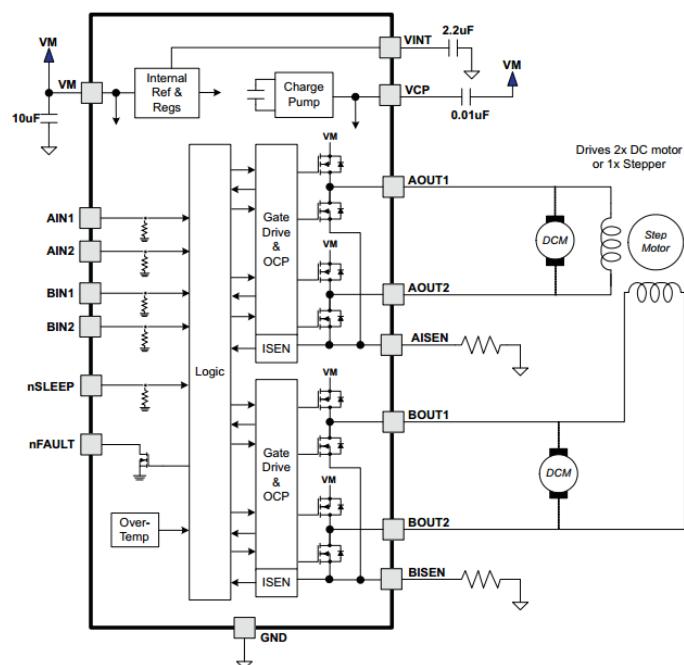


图 5-17 芯片内部原理

本模块采用 TI 的双通道 H 桥电流控制电机驱动器 DRV8833，用于驱动步进电机。每个 H 桥的输出驱动器模块由配置为 H 桥的 N 沟道功率 MOSFET 组成，用于驱动电机绕组。每个 H 桥均具备调节或限制绕组电流的电路。步进电机的步进量以及转速可通过 4 路 PWM 来调节。

### 5.2.3.3 软件实现

根据原理图可知步进电机使用 4 路 PWM 驱动控制，该模块中的 PWM 实现与 5.2.2 节高亮 LED 驱动模块（MelodyLED）中的 PWM 实现不同，此处 PWM 实现使用 cc3200SDK 中的 timer\_if.c 和 timer\_if.h 实现（这两个文件位于 MelodyStepMotor 下的 utility 文件夹中），因为 Energia 实现的库无法同时输出 4 路 PWM 波，具体可看 Wiring\_analog.c 文件中库函数 PWMWrite()的实现。

#### 5.2.3.3.1 步进电机驱动库实现

驱动库使用 C++ 实现，源代码可直接查看 MelodyStepMotor 文件下的 Melody\_StepMotor.h 和 Melody\_StepMotor.cpp 文件。下面简单介绍库函数。MelodyStepMotor 定义在 Melody\_StepMotor.cpp 中。

表 5-9 begin()函数

驱动库函数	<b>void begin()</b>
说明	初始化步进电机，设置默认最小速度，顺时针转动,起始步进位置为 0
使用范例	MelodyStepMotor.begin();

表 5-10 begin()函数

驱动库函数	<b>void begin(uint8_t speed)</b>
说明	初始化步进电机，设置速度为 speed，速度范围 1~10，顺时针转动,起始步进位置为 0
使用范例	MelodyStepMotor.begin(1);

表 5-11 begin()函数

驱动库函数	<b>void begin(uint8_t speed, bool direction)</b>
说明	初始化步进电机，设置速度为 speed，速度范围 1~10，转动方向为 direction: true 为顺时针，false 为逆时针,起始步进位置为 0
使用范例	MelodyStepMotor.begin(1,true);

表 5-12 begin()函数

驱动库函数	<b>void begin(uint8_t speed, bool direction, uint8_t step)</b>
说明	初始化步进电机，设置速度为 speed: 速度范围 1~10，转动方向为 direction: true 为顺时针，false 为逆时针,起始步进位置为 step: 范围 0~19

使用范例	MelodyStepMotor.begin(1,true,0);
------	----------------------------------

表 5-13 changeSpeed()函数

驱动库函数	<b>void changeSpeed(uint8_t speed)</b>
说明	改变步进电机速度，速度范围为 1~10
使用范例	changeSpeed(5);

表 5-14 getSpeed()函数

驱动库函数	<b>uint8_t getSpeed()</b>
说明	获取步进电机当前运转速度，速度范围为 1~10
使用范例	uint8_t speed = MelodyStepMotor.getSpeed();

表 5-15 changeDirection()函数

驱动库函数	<b>void changeDirection (bool direction)</b>
说明	改变步进电机转动方向，true: 顺时针方向，false: 逆时针方向
使用范例	MelodyStepMotor.changeDirection(true);

表 5-16 getDirection()函数

驱动库函数	<b>bool getDirection ()</b>
说明	获取步进电机转动方向，true: 顺时针方向，false: 逆时针方向
使用范例	bool direction = MelodyStepMotor.getDirection();

表 5-17 getSteps()函数

驱动库函数	<b>uint8_t getSteps()</b>
说明	获取步进电机步进值，范围 0~19
使用范例	uint8_t step = MelodyStepMotor.getSteps();

### 5.2.3.3.2 实验例程

实验例程操作及结果展示可浏览视频资料：[http://www.hpati.com/product\\_videos/v81.html](http://www.hpati.com/product_videos/v81.html)

例程 1, SimpleStepMotor.ino：启动步进电机顺时针转动，通过串口上传当前步进位置。  
例程流程图如图 5-18，例程源代码如图 5-19。

Energia框架	例程执行流程	对应使用的库函数
-----------	--------	----------

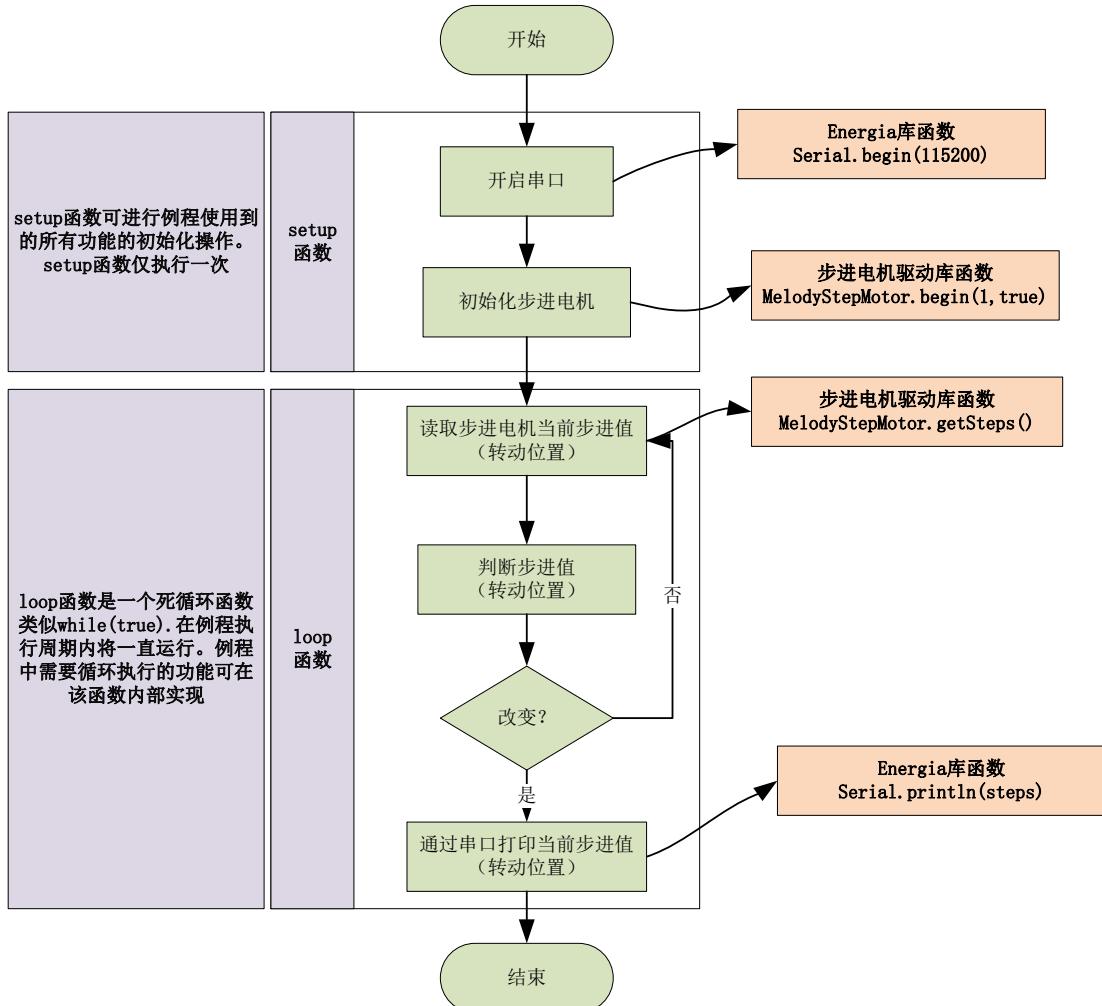


图 5-18 SimpleStepMotor 例程流程图

```

#include <stdbool.h>
#include <stdint.h>
#include "Melody_StepMotor.h"
uint8_t cur_step = 0;
uint8_t per_step = 0;
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    MelodyStepMotor.begin(1, true);
}

void loop()
{
    // put your main code here, to run repeatedly:
}

```

```

cur_step = MelodyStepMotor.getSteps();
if(per_step != cur_step)
{
    Serial.println(cur_step);
    per_step = cur_step;
}
}

```

图 5-19 SimpleStepMotor 例程源代码

例程 2, SimpleWebStepMotor.ino:通过网络改变步进电机转动方向和速度。cc3200 设置为 webserver 模式，用户可是有浏览器访问 cc3200，实验过程中 cc3200 的 IP 地址通过串口打印至 PC 端。Energia 开发工具提供了串口监视工具。网络驱动库函数可见 MelodySimplehtml 文件夹下的 Melody\_Simplehtml.h 和 Melody\_Simplehtml.cpp 文件。程序例程图如图 5-20 所示，例程代码如图 5-21 所示。

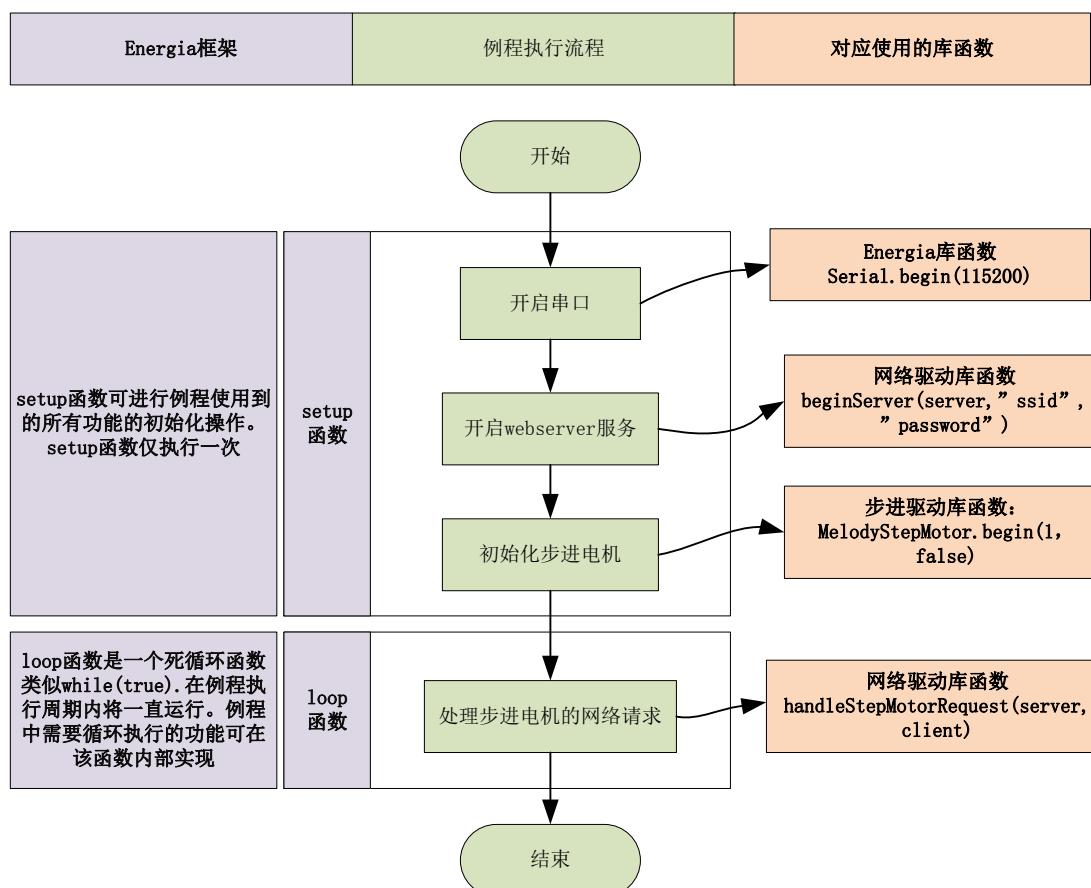


图 5-20 SimpleWebStepMotor 例程流程图

```

#include <stdbool.h>
#include <stdint.h>
#include <WiFi.h>
#include "Melody_StepMotor.h"
#include "Melody_Simplehtml.h"

WiFiServer server(80);

```

```
WiFiClient client;
void setup()
{
// put your setup code here, to run once:
Serial.begin(115200);
beginServer(server,"ssid","password");
MelodyStepMotor.begin(1,false);
}

void loop()
{
// put your main code here, to run repeatedly:
handleStepMotorRequest(server,client);
}
```

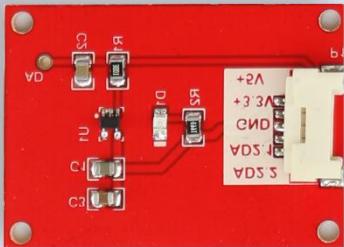
图 5-21 SimpleWebStepMotor 例程源代码

## 5.2.4 模拟温度模块 (*MelodyLMT84*)

### 5.2.4.1 模块

模拟温度模块使用 AD 接口进行温度采集。模块如下

表 5-18 模拟温度采集模块

模拟温度采集模块	概况
	模块使用芯片 LMT84 使用 ADC 接口采集温度数据 测量精度 $\pm 0.4^{\circ}\text{C}$ 输出受到短路保护

#### 5.2.4.2 实现原理及原理图简介

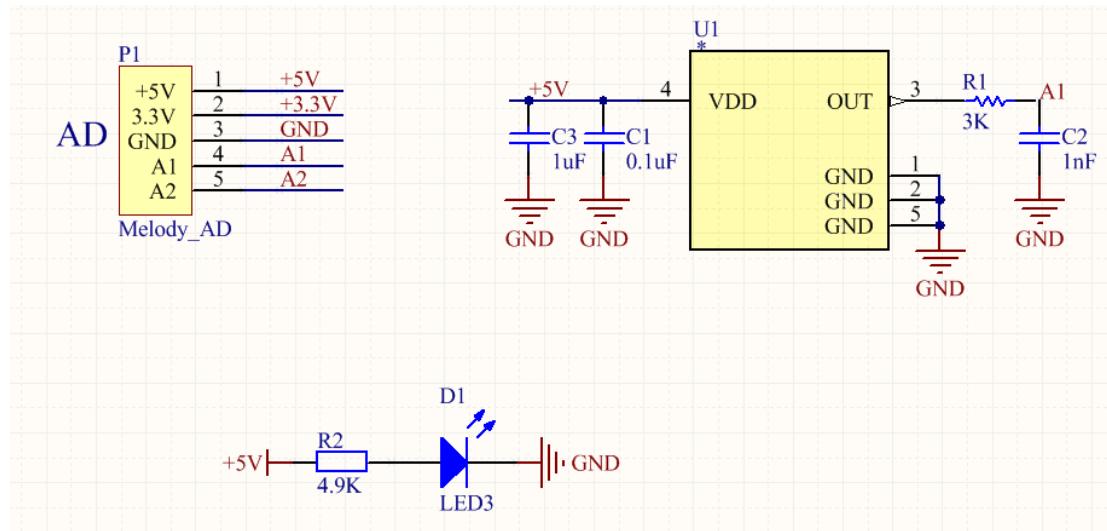


图 5-22 模拟温度模块原理图

LMT84 是高精度 CMOS 集成电路温度传感器，此传感器具有与温度成线性反比例关系的模拟电压输出，再用 CC3200 的内部 ADC，即可获得 LMT84 的模拟电压输出，进而转换成温度。测量范围为-50°C~150°C。

测量温度与模拟电压关系为： $T = (V_{out} - 1035mV) / (-5.5mV/^\circ C)$

详细的对应关系，可参考 LMT84 的手册 <http://www.ti.com.cn/cn/lit/ds/symlink/lmt84.pdf>

#### 5.2.4.3 软件实现

模拟温度模块使用 AD 进行数据采集，Energia 已经实现了 ADC 库函数，用户直接使用即可。ADC 库函数可在 <http://energia.nu/reference/analogread/> 查看，函数名为 analogRead(pin)。

表 5-19 analogRead() 函数

Energia 库函数	int analogRead (uint8_t pin)
说明	读取指定端口电压值。
使用范例	analogRead(A1); A1 为 cc3200 ADC 功能端口，定义可在 【Energia 安装目录】->hardware->cc3200->variants ->Launchpad->pins_energia.h

函数源代码可在 【Energia 安装目录】->hardware->cc3200->cores->cc3200->wiring\_analog.c 中查看，库函数实现如

```
uint16_t analogRead(uint8_t pin)
{
    uint16_t channel, val;
    uint16_t pinNum = digitalPinToPinNum(pin);

    switch(pinNum) {
    case PIN_57:{channel = ADC_CH_0;}break;
```

```

case PIN_58:{channel = ADC_CH_1;}break;
case PIN_59:{channel = ADC_CH_2;}break;
case PIN_60:{channel = ADC_CH_3;}break;
default:return0;
}

while(ADCFIFOLvlGet(ADC_BASE, channel)){
    // flush the channel's FIFO if not empty
    ADCFIFORead(ADC_BASE, channel);
}

PinTypeADC(pinNum,0xFF);
ADCChannelEnable(ADC_BASE, channel);
ADCTimerConfig(ADC_BASE,0x1ffff);
ADCTimerEnable(ADC_BASE);
ADCEnable(ADC_BASE);

while(!ADCFIFOLvlGet(ADC_BASE, channel));
val = ADCFIFORead(ADC_BASE, channel)&0x3FFF;

ADCDisable(ADC_BASE);
ADCChannelDisable(ADC_BASE, channel);
ADCTimerDisable(ADC_BASE);

val = val >>2;
return mapResolution(val,12, _readResolution);
}

```

图 5-23 Energia 库函数 analogRead()函数

#### 5.2.4.3.1 模拟温度驱动库实现

驱动库使用 C++ 实现，源代码可直接查看 MelodyLMT84 文件下的 Melody\_LMT84.h 和 Melody\_LMT84.cpp 文件。下面简单介绍库函数。MelodyLMT84 定义在 Melody\_LMT84.cpp

表 5-20 begin()函数

驱动库函数	<b>void begin()</b>
说明	初始化温度采集端口设置，默认使用 A1 端口
使用范例	MelodyLMT84.begin();

表 5-21 begin()函数

驱动库函数	<b>void begin(uint8_t port)</b>
说明	初始化指定端口进行温度采集
使用范例	MelodyLMT84.begin(A1);

表 5-22 getTemperature()函数

驱动库函数	float getTemperature()
说明	获取温度值
使用范例	float temperature = MelodyLMT84.getTemperature();

#### 5.2.4.3.2 实验例程

实验例程操作及结果展示可浏览视频资料：[http://www.hpati.com/product\\_videos/v78.html](http://www.hpati.com/product_videos/v78.html)

例程 1, LMT84.ino:读取模拟温度模块温度值并且通过串口上传。例程流程图如图 5-24，例程源程序如图 5-25：

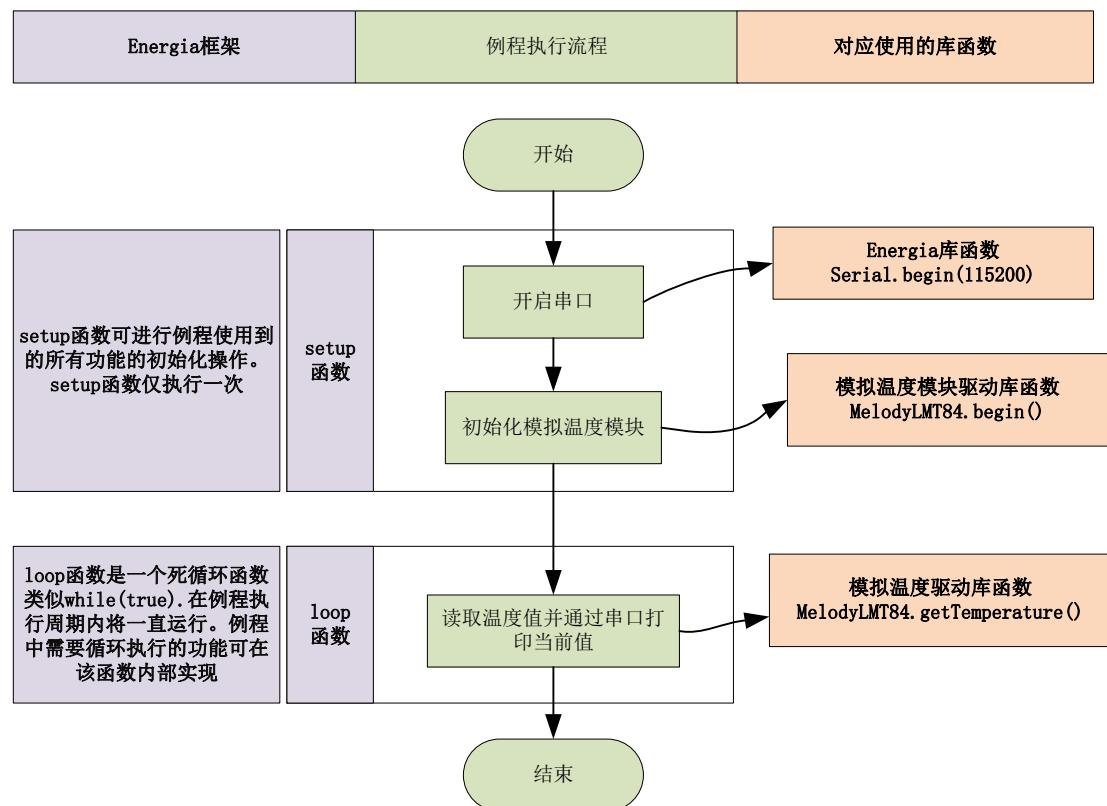


图 5-24 LMT84 例程流程图

```

#include <stdbool.h>
#include <stdint.h>
#include "Melody_LMT84.h"
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    MelodyLMT84.begin();
}

```

```

}

float temperature = 0.0;
void loop()
{
    // put your main code here, to run repeatedly:
    Serial.print("temperature:");
    temperature = MelodyLMT84.getTemperature();
    Serial.println(temperature);
    delay(200);
}

```

图 5-25 LMT84 例程源代码

例程 2, SimpleWebLMT84.ino: 通过网络读取温度值。cc3200 设置为 webserver 模式，用户可是有浏览器访问 cc3200，实验过程中 cc3200 的 IP 地址通过串口打印至 PC 端。Energia 开发工具提供了串口监视工具。程序例程图如所示，网络驱动库函数可见 MelodySimplehtml 文件夹下的 Melody\_Simplehtml.h 和 Melody\_Simplehtml.cpp 文件。例程流程图如图 5-26，例程代码如图 5-27 所示。

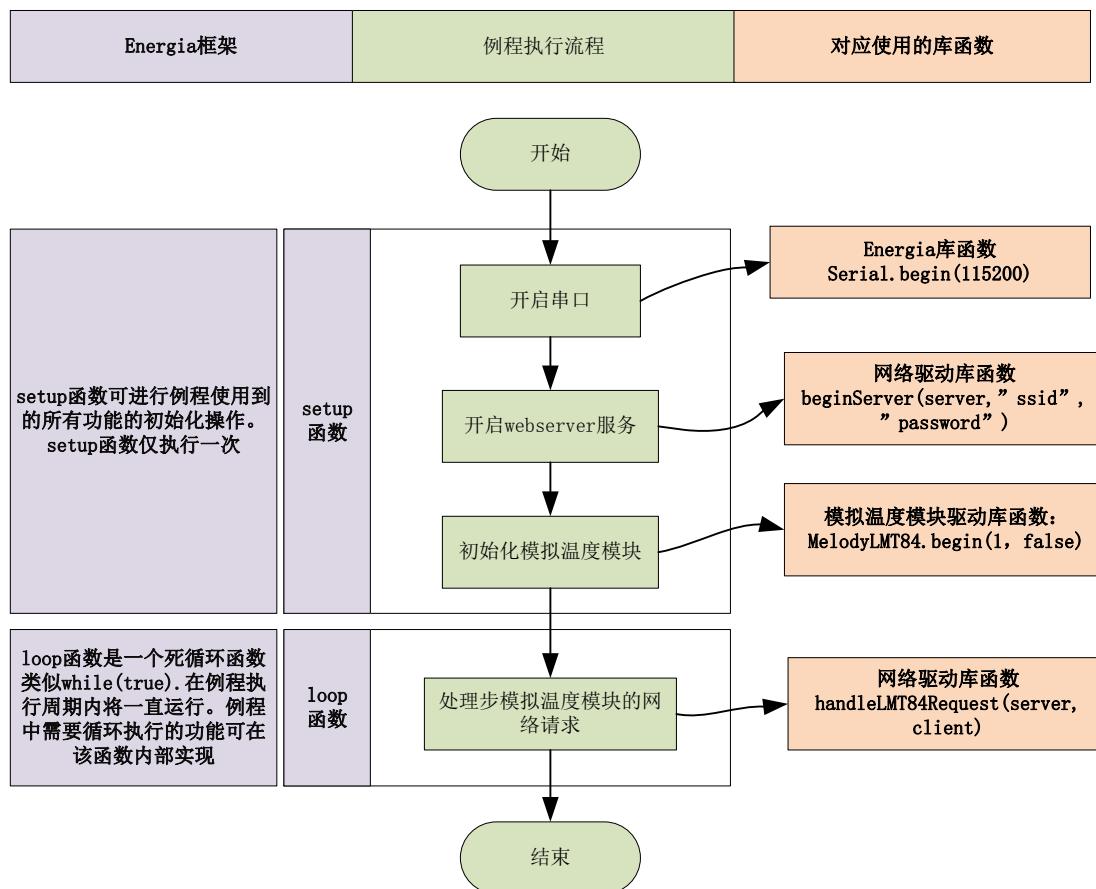


图 5-26 SimpleWebLMT84 例程流程图

```

#include <stdbool.h>
#include <stdint.h>
#include <WiFi.h>

```

```

#include "Melody_LMT84.h"
#include "Melody_Simplehtml.h"

WiFiServer server(80);
WiFiClient client;

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    beginServer(server,"ssid","password");
    MelodyLMT84.begin();
}

void loop()
{
    // put your main code here, to run repeatedly:
    handleLMT84Request(server,client);
}

```

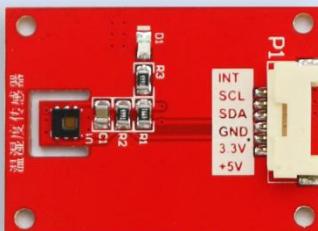
图 5-27 SimpleWebLMT84 例程源代码

## 5.2.5 温湿度传感器模块 (*MelodyHDC1080*)

### 5.2.5.1 模块

温湿度传感器模块使用 I<sub>2</sub>C 接口与 cc3200 进行通信，可读取模块的温度和湿度数据。模块如下：

表 5-23 温湿度传感器模块

温湿度传感器模块	概况
	1 模块使用芯片 HDC1080 2 使用 I <sub>2</sub> C 接口，读取模块的温湿度数据 3 相对湿度精度达到±2% 4 温度精度达到±0.2°C 5 14 位测量分辨率

### 5.2.5.2 实现原理及原理图简介

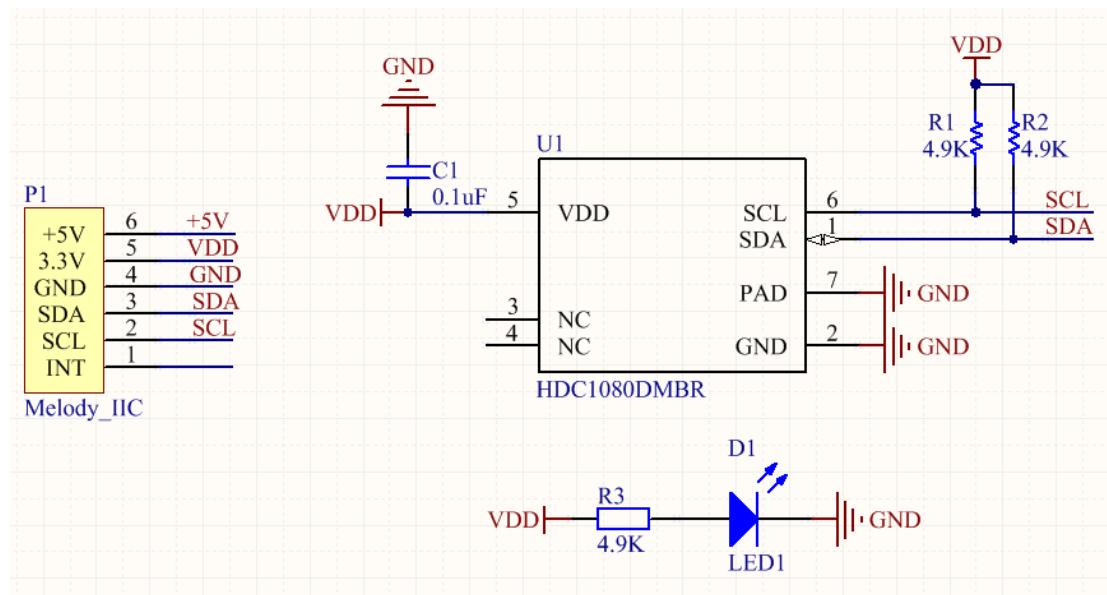


图 5-28 温湿度传感器模块原理图

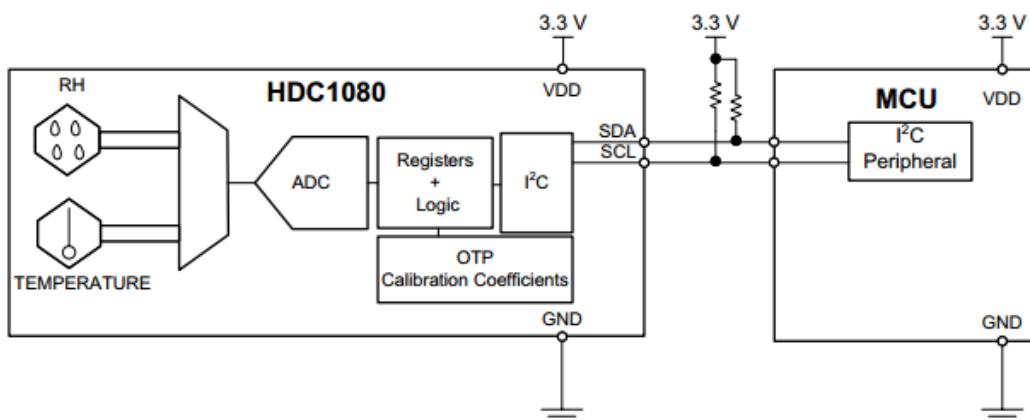


图 5-29 原理简化图

HDC1080 集成了温度传感器和数字湿度传感器，并能够以超低功耗提供出色的测量精度，相对湿度精度为±2%（典型值），温度精度为±0.2°C（典型值）。具有 14 位测量分辨率。通过 I2C 接口，与 MCU 连接即可，使用便利。

### 5.2.5.3 软件实现

温湿度传感器模块使用 I2C 接口，Energia 已经实现了 I2C 库函数，用户直接使用即可。I2C 库函数，可在 <http://energia.nu/reference/wire/> 查看。I2C 库函数在使用上异于 5.2.2 章节“高亮 LED 驱动模块(MelodyLED)”中 PWM 库函数和 5.2.4 章节“模拟温度模块(MelodyLMT84)”中 AD 库函数的使用；上述两个章节中库函数使用仅需直接调用一个函数即可（如 `analogWrite()` 函数和 `analogRead()` 函数）。本章节模块使用 I2C 库函数，下面将简单介绍 I2C 库函数使用。

### 5.2.5.3.1 I2C 库函数

根据 Energia 官网上的介绍，I2C 库函数使用 C++ 实现，由多个成员函数构成，用户可直接使用库文件已经定义的“Wire”对象调用其成员函数，如 Wire.begin()。各成员函数如图 5-30 所示。



The screenshot shows the 'Functions' section of the Wire Library (I2C) Overview page. The listed functions are:

- begin()
- requestFrom()
- beginTransmission()
- endTransmission()
- write()
- available()
- read()
- onReceive()
- onRequest()

图 5-30 I2C 库函数

数据发送基本过程：

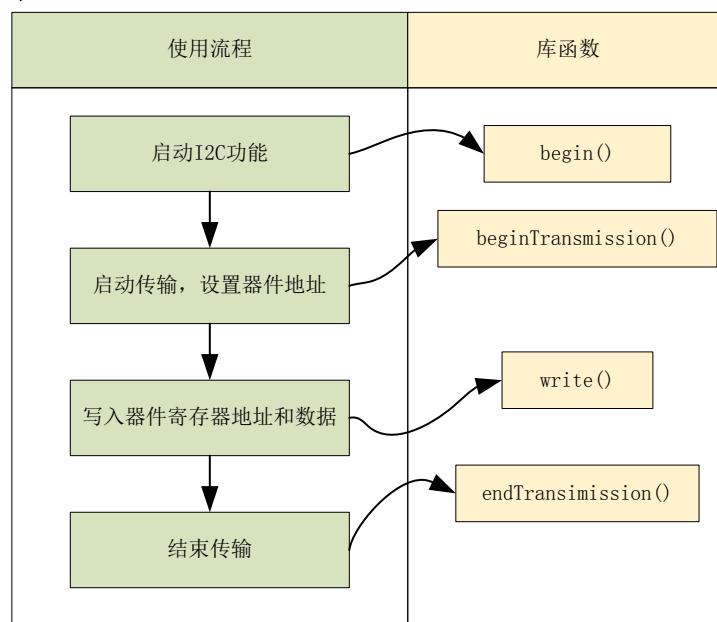


图 5-31 I2C 库函数使用——数据发送基本过程流程展示

程序实现可参考 Energia 提供的例程，可在 Energia 软件 File->Examples->Wire->master\_writer 找到。代码如下：

```
#include <Wire.h>

void setup()
{
}
```

```

Wire.begin(); // join i2c bus (address optional for master)
}

byte x = 0;

void loop()
{
    Wire.beginTransmission(4); // transmit to device #4
    Wire.write("x is "); // sends five bytes
    Wire.write(x); // sends one byte
    Wire.endTransmission(); // stop transmitting

    x++;
    delay(500);
}

```

图 5-32 I2C 库函数使用——数据发送基本过程代码实现

数据读取基本过程：

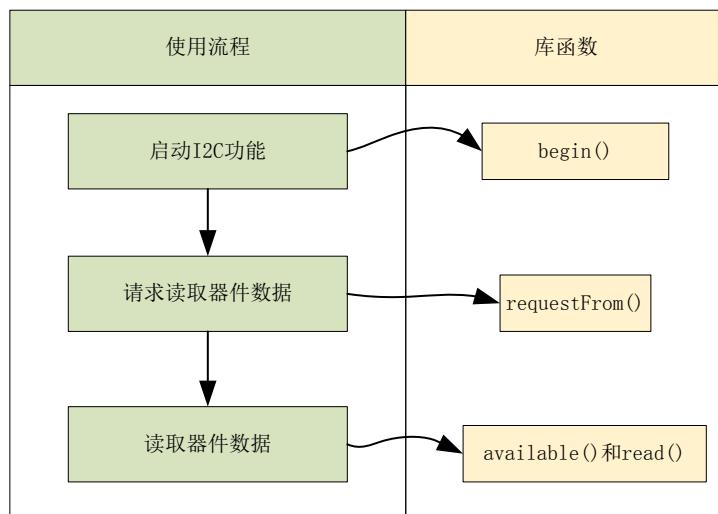


图 5-33 I2C 库函数使用——数据读取基本过程流程展示

程序实现可参考 Energia 提供的例程，可在 Energia 软件 File->Examples->Wire->master\_writer 找到。代码如下：

```

#include <Wire.h>

void setup()
{
    Wire.begin(); // join i2c bus (address optional for master)
    Serial.begin(9600); // start serial for output
}

void loop()
{
}

```

```

Wire.requestFrom(2, 6); // request 6 bytes from slave device #2

while(Wire.available()) // slave may send less than requested
{
    char c = Wire.read(); // receive a byte as character
    Serial.print(c); // print the character
}
delay(500);
}

```

图 5-34 I2C 库函数使用——数据读取基本过程代码实现

**注意：I2C 库函数使用时，数据发送和数据读取过程需根据实际器件时序实现。**

### 5.2.5.3.2 温湿度传感器驱动库实现

#### 写时序

温湿度传感器模块使用 HDC1080，该芯片的写时序如下：

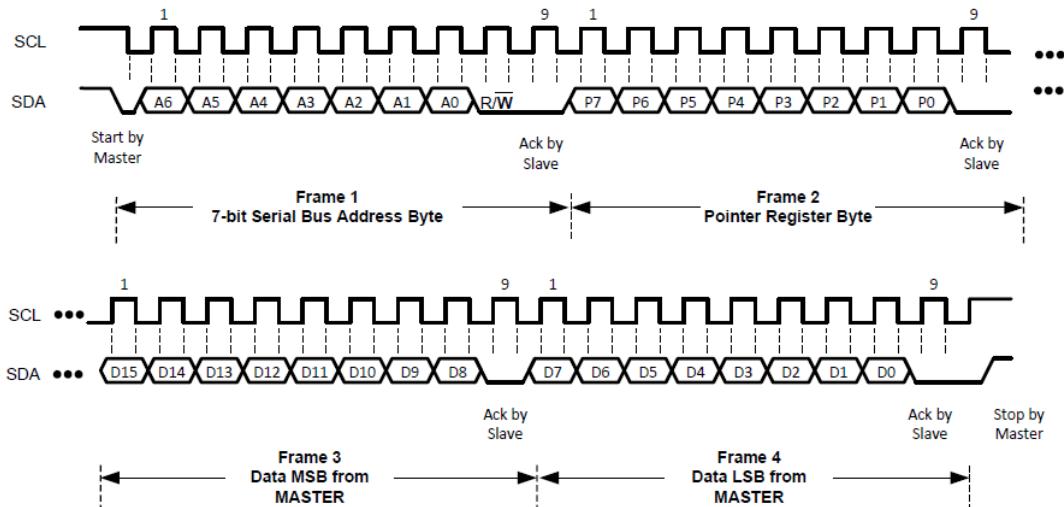


图 5-35 写时序

在实现时 cc3200 做为 Master，HDC1080 做为 slave。数据写入过程：第一步：cc3200 发送 HDC1080 的地址，根据芯片手册可知 HDC1080 的地址为 0b01000000 (0x40)；第二步：发送需要写入的寄存器地址；第三步：发送两个字节长度的数据，先发送高字节数据，在发送低字节数据。

结合图 5-31 和图 5-35，写数据程序实现如下（不包括启动 I2C 功能，I2C 启动仅需调用 `Wire.begin()` 即可，并且在使用过程中仅可被调用一次。）

```

*****
* @brief: 写数据
* @param: regaddress: 寄存器地址
*          pdata, 指向需要写入数据的指针
*          length, 数据长度
*****

```

```

* @return: none
*****
void CMelodyHDC1080::write(uint8_t regaddress,uint8_t*pdata,uint8_t
                           length)
{
    Wire.beginTransmission(HDC_ADDRESS);
    Wire.write(regaddress);
    Wire.write(pdata,length);
    Wire.endTransmission();
}

```

图 5-36 写数据代码实现

### 读时序

HDC1080 读时序如下：

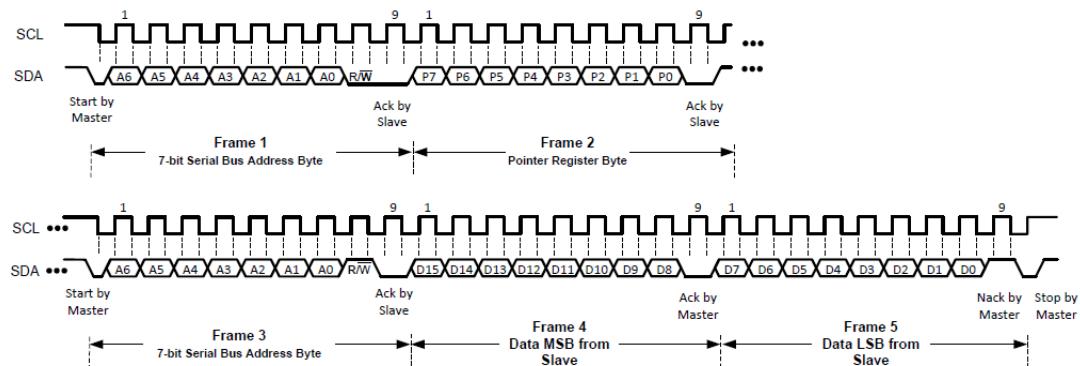


图 5-37 读时序

在实现时 cc3200 做为 Master, HDC1080 做为 slave。数据读取过程：第一步：cc3200 发送 HDC1080 的地址，根据芯片手册可知 HDC1080 的地址为 0b01000000 (0x40)；第二步：发送需要读取数据的寄存器地址；第三步：再次发送 HDC1080 的地址，第四步：读取两个字节长度的数据，高字节在前，低字节在后。

结合图 5-33 图 5-31 和图 5-37 读时序，读数据程序实现如下（不包括启动 I2C 功能，I2C 启动仅需调用 `Wire.begin()` 即可，并且在使用过程中仅可被调用一次。）

```

*****
* @brief: 读取数据
* @param: regaddress:寄存器地址
* @return: 读取的数据
*****
uint16_t CMelodyHDC1080::read(uint8_t regaddress)
{
    uint16_t data =0;
    Wire.beginTransmission(HDC_ADDRESS);
    Wire.write(regaddress);
    Wire.endTransmission();

```

```

delay(40);

Wire.requestFrom(HDC_ADDRESS, 2);
if(Wire.available()>=2)
{
    data = Wire.read()<<8;
    data |= Wire.read();
}
return data;
}

```

图 5-38 读数据代码实现

驱动库部分函数如下：类对象“MelodyHDC1080”定义在 Melody\_HDC1080.cpp 文件中。

表 5-24 begin()函数

驱动库函数	<b>void begin()</b>
说明	启动 I2C 功能，初始化 HDC1080
使用范例	MelodyHDC1080.begin();

表 5-25 getTemperature()函数

驱动库函数	<b>float getTemperature()</b>
说明	获取温度值
使用范例	float temperature = MelodyHDC1080.getTemperature();

表 5-26 getHumidity()函数

驱动库函数	<b>float getHumidity()</b>
说明	获取湿度值
使用范例	float humidity = MelodyHDC1080.getHumidity();

表 5-27 getManufacturerID()函数

驱动库函数	<b>uint16_t getManufacturerID ()</b>
说明	获取 Manufacturer ID
使用范例	uint16_t id = MelodyHDC1080.getManufacturerID();

表 5-28 getDeviceID()函数

驱动库函数	<b>uint16_t getDeviceID()</b>
说明	获取 Device ID
使用范例	uint16_t id = MelodyHDC1080.getDeviceID();

### 5.2.5.3.3 实验例程

实验例程操作及结果展示可浏览视频资料：[http://www.hpati.com/product\\_videos/v75.html](http://www.hpati.com/product_videos/v75.html)

例程 1, HDC1080.ino: 读取当前温度数据, 使用串口打印。实例流程图如图 5-39, 实例代码如图 5-40,

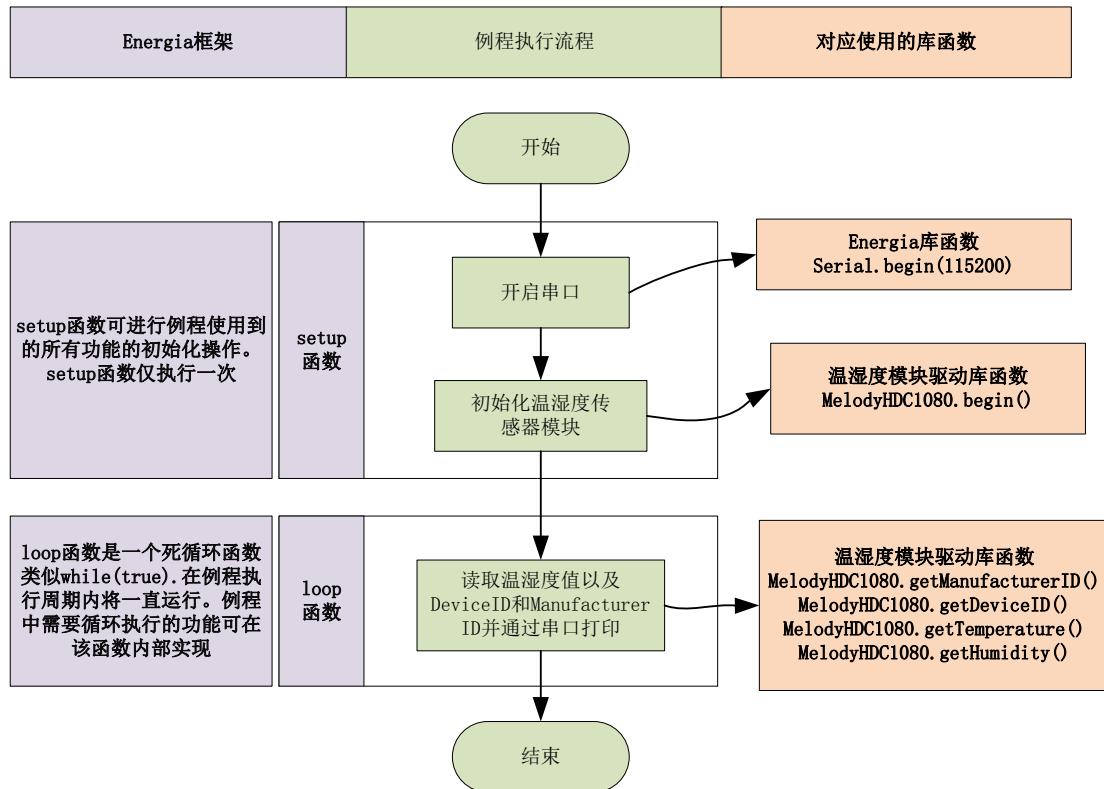


图 5-39 HDC1080 例程流程图

```
#include <stdbool.h>
#include <stdint.h>
#include "Wire.h"
#include "Melody_HDC1080.h"

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    MelodyHDC1080.begin();
}

void loop()
{
    // put your main code here, to run repeatedly:
    Serial.println("-----");
    Serial.print("Manufacturer ID:0x");
    Serial.println(MelodyHDC1080.getManufacturerID(), HEX);
    Serial.print("Device ID:0x");
    Serial.println(MelodyHDC1080.getDeviceID(), HEX);
    Serial.print("Temperature:");
}
```

```

Serial.println(MelodyHDC1080.getTemperature());
Serial.print("Humidity:");
Serial.println(MelodyHDC1080.getHumidity());
Serial.println("-----\n\n");
delay(1000);
}

```

图 5-40 HDC1080 例程源程序

例程 2, SimpleWebHDC1080.ino: 通过网络读取温湿度值。cc3200 设置为 webserver 模式，用户可是有浏览器访问 cc3200，实验过程中 cc3200 的 IP 地址通过串口打印至 PC 端。Energia 开发工具提供了串口监视工具。程序例程图如所示，网络驱动库函数可见 MelodySimplehtml 文件夹下的 Melody\_Simplehtml.h 和 Melody\_Simplehtml.cpp 文件。例程流程图如图 5-41，例程代码如所示图 5-42。

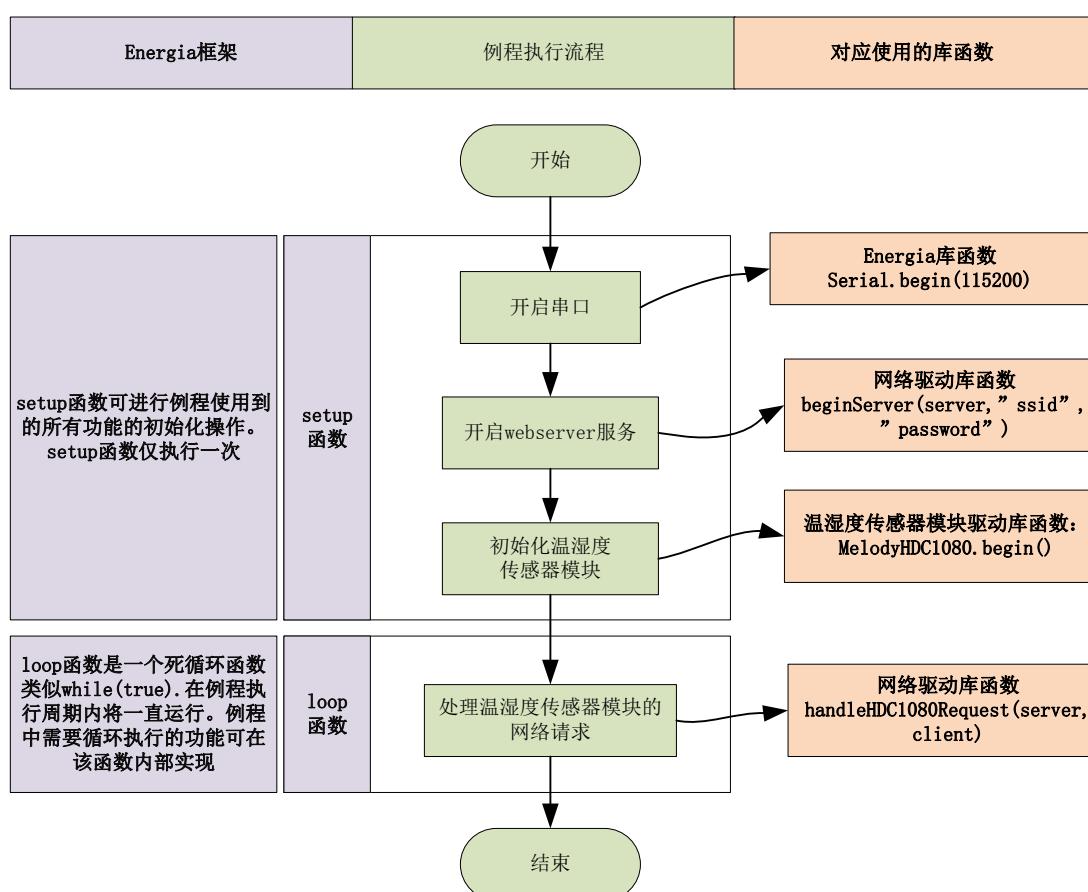


图 5-41 SimpleWebHDC1080 例程流程图

```

#include <stdbool.h>
#include <stdint.h>
#include <WiFi.h>
#include <Wire.h>
#include <Melody_HDC1080.h>
#include <Melody_Simplehtml.h>

```

```
WiFiServer server(80);
WiFiClient client;
void setup()
{
// put your setup code here, to run once:
Serial.begin(115200);
beginServer(server,"ssid","password");
MelodyHDC1080.begin();

}

void loop()
{
// put your main code here, to run repeatedly:
handleHDC1080Request(server,client);
}
```

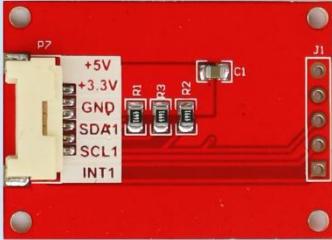
图 5-42 SimpleWebHDC1080 例程源程序

## 5.2.6 光感模块 (*MelodyOPT3001*)

### 5.2.6.1 模块

光感模块使用 I2C 接口与 cc3200 进行通信，可读取模块的光照强度数据。模块如下：

表 5-29 光感模块

光感模块	概况
	<ul style="list-style-type: none"><li>1 模块使用芯片 OPT3001</li><li>2 使用 I2C 接口通信</li><li>3 测量范围：0.01lux 至 83k lux</li><li>4 23 位有效动态范围，具有自动增益范围设置功能</li></ul>

### 5.2.6.2 实现原理及原理图简介

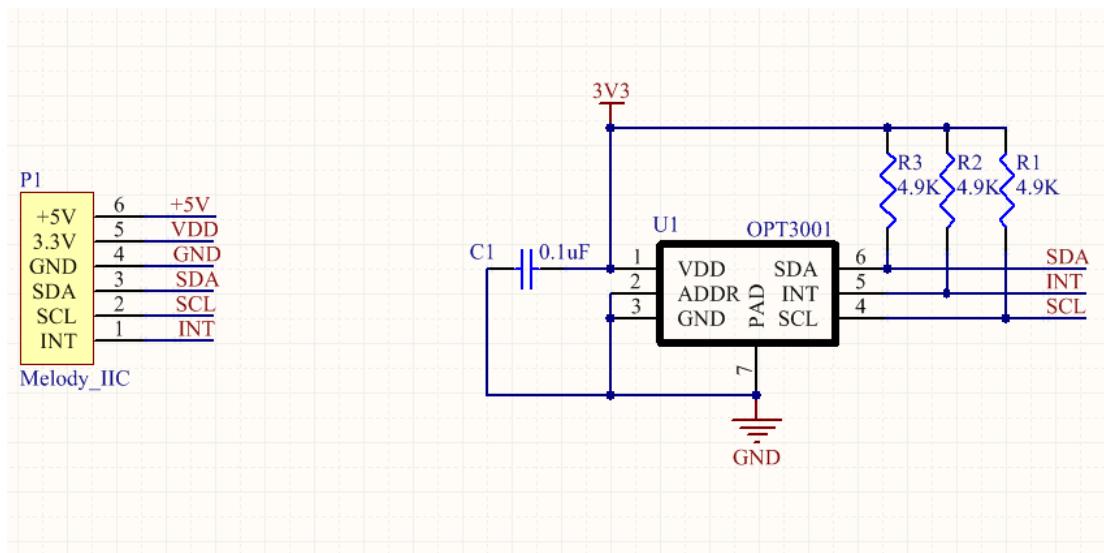


图 5-43 光感模块原理图

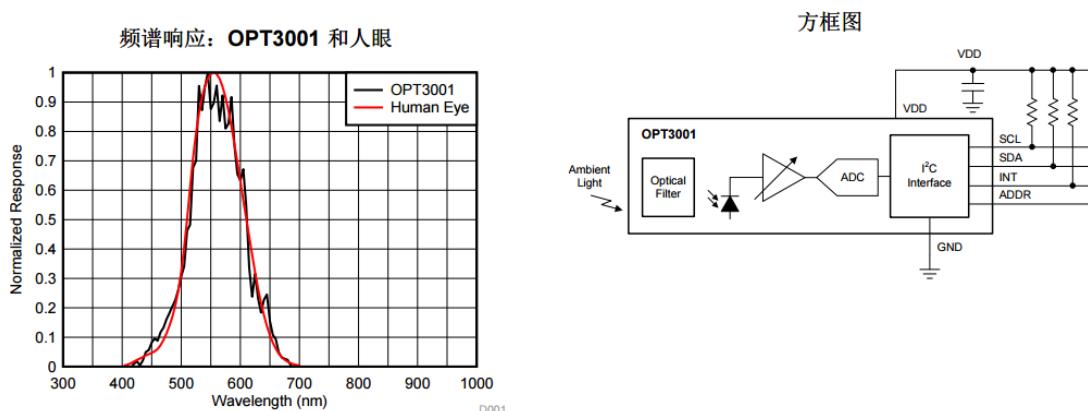


图 5-44 频谱相应和简化原理图

OPT3001 传感器用于测量可见光的密度。OPT3001 器件兼具精密的频谱响应和较强的 IR 阻隔功能，因此能够如人眼般准确测量光强且不受光源影响。OPT3001 专门针对构建基于光线的人眼般体验的系统而设计，是人眼匹配度低且红外阻隔能力差的光电二极管、光敏电阻或其它环境光传感器的首选理想替代产品。

测量范围可达 0.01lux 至 83k lux，且内置有满量程设置功能，无需手动选择满量程范围。数字输出通过 I2C 接口于 MCU 连接即可

### 5.2.6.3 软件实现

光感模块使用 I2C 接口，Energia 已经实现了 I2C 库函数，用户直接使用即可。I2C 库函数，可在 <http://energia.nu/reference/wire/> 查看，可以直接查看“5.2.5.3.1 I2C 库函数”对 I2C 库函数的使用介绍。

### 5.2.6.3.1 光感模块驱动程序实现

#### 写时序

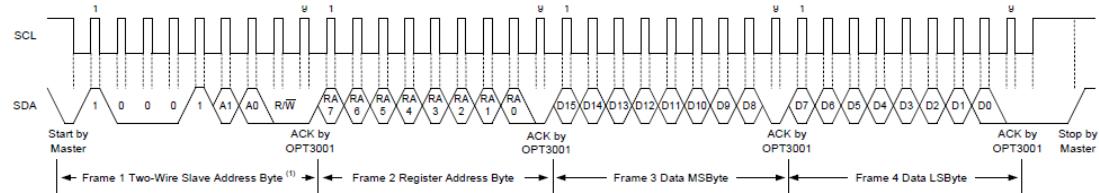


图 5-45 写时序

在实现时 cc3200 做为 Master, OPT3001 做为 slave。数据写入过程：第一步：cc3200 发送 OPT3001 的地址；第二步：发送需要写入数据的寄存器地址；第三步：发送两个字节长度的数据，先发送高字节数据，在发送低字节数据。

结合图 5-31 和图 5-45，写数据程序实现如下（不包括启动 I2C 功能，I2C 启动仅需调用 Wire.begin() 即可，并且在使用过程中仅可被调用一次。）

```
/*
 * @brief: 写寄存器数据
 * @param: regAddress, 寄存器地址
 *          data, 待写入数据
 * @return: none
 */

void CMelodyOPT3001::write(uint8_t regAddress, uint16_t data)
{
    Wire.beginTransmission(OPTADDRESS);
    Wire.write(regAddress);
    Wire.write(data >> 8);
    Wire.write(data);
    Wire.endTransmission();
}
```

图 5-46 写时序程序实现

#### 读时序

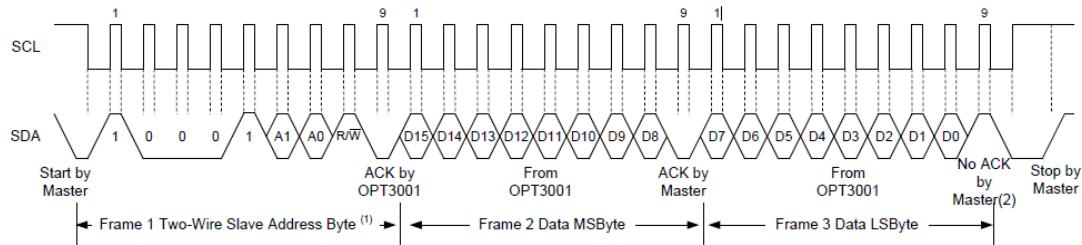


图 5-47 读时序

在实现时 cc3200 做为 Master, OPT3001 做为 slave。数据读取过程：第一步：cc3200 发送 OPT3001 的地址；第二步：读取两个字节长度的数据，高字节在前，低字节在后。

结合图 5-31 和图 5-47，写数据程序实现如下（不包括启动 I2C 功能，I2C 启动仅需调用

Wire.begin()即可，并且在使用过程中仅可被调用一次。)

```

/*
 * @brief: 读取寄存器数据
 * @param: regAddress, 寄存器地址
 * @return: none
 */
uint16_t CMelodyOPT3001::read(uint8_t regAddress)
{
    Wire.beginTransmission(OPTADDRESS);
    Wire.write(regAddress);
    Wire.endTransmission();

    uint16_t data = 0;
    Wire.requestFrom(OPTADDRESS, 2);
    if(Wire.available() >= 2)
    {
        data = Wire.read();
        data = (data << 8) | Wire.read();
    }
    return data;
}

```

图 5-48 读时序程序实现

驱动库部分函数如下：类对象“MelodyOPT3001”定义在 Melody\_OPT3001.cpp 文件中。

表 5-30 begin() 函数

驱动库函数	void begin(void)
说明	开启 I2C 功能，初始化配置 OPT3001
使用范例	MelodyOPT3001.begin();

表 5-31 getManufacturerID() 函数

驱动库函数	uint16_t getManufacturerID (void)
说明	获取 OPT3001 的 Manufacturer ID
使用范例	uint16_t id = MelodyOPT3001.getManufacturerID();

表 5-32 getDeviceID() 函数

驱动库函数	uint16_t getDeviceID (void)
说明	获取 OPT3001 的 Device ID
使用范例	uint16_t id = MelodyOPT3001.getDeviceID();

表 5-33 getLux() 函数

驱动库函数	float getLux(void)
说明	获取当前光照强度
使用范例	uint16_t id = MelodyOPT3001.getDeviceID();

### 5.2.6.3.2 实验例程

实验例程操作及结果展示可浏览视频资料：[http://www.hpati.com/product\\_videos/v80.html](http://www.hpati.com/product_videos/v80.html)

例程 1, OPT3001.ino: 读取当前光照强度，使用串口打印。实例流程图如图 5-49，实例代码如图 5-50。

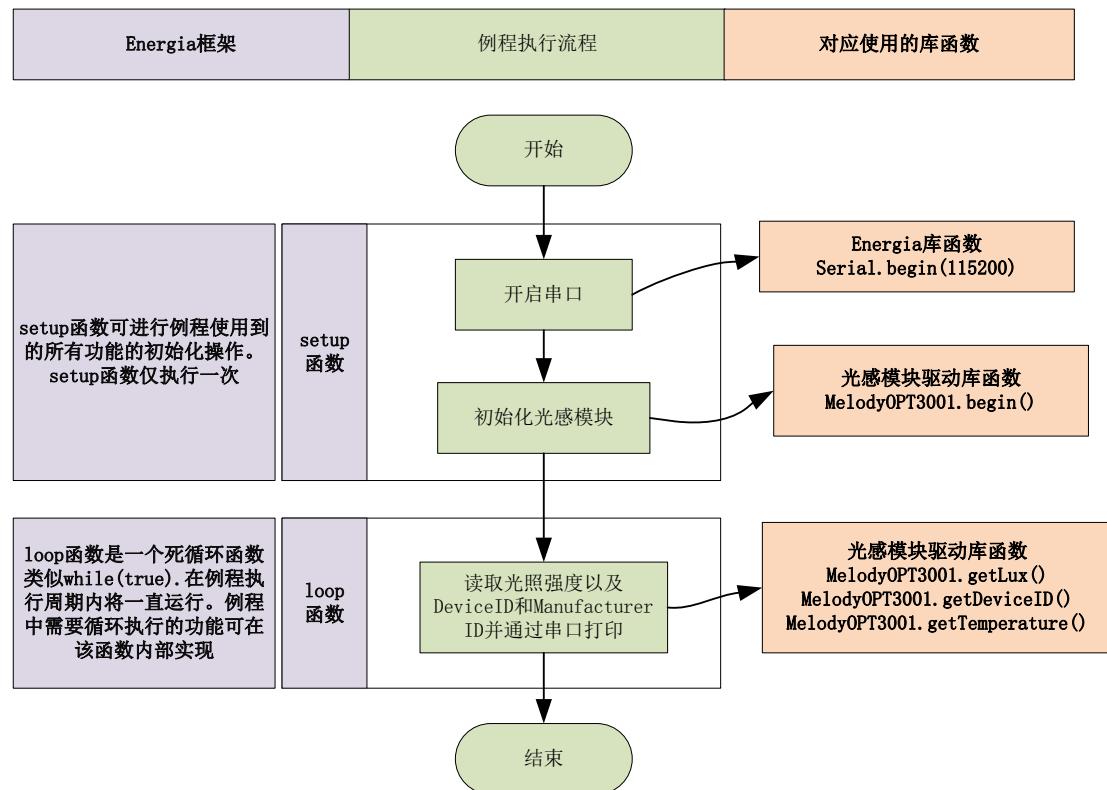


图 5-49 OPT3001 实例流程图

```

#include <stdbool.h>
#include <stdint.h>
#include "Wire.h"
#include "Melody_OPT3001.h"

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    MelodyOPT3001.begin();
}

void loop()
{
    // put your main code here, to run repeatedly:
    Serial.println("-----");
    Serial.print("Manufacturer ID:0x");
}

```

```

Serial.println(MelodyOPT3001.getManufacturerID(), HEX);
Serial.print("Device ID:0x");
Serial.println(MelodyOPT3001.getDeviceID(), HEX);
Serial.print("Lux:");
Serial.println(MelodyOPT3001.getLux());
Serial.println("-----");
delay(1000);
}

```

图 5-50 OPT300 例程源代码

例程 2, SimpleWebOPT3001.ino: 通过网络读取光照强度。cc3200 设置为 webserver 模式, 用户可是有浏览器访问 cc3200, 实验过程中 cc3200 的 IP 地址通过串口打印至 PC 端。Energia 开发工具提供了串口监视工具。程序例程图如所示, 网络驱动库函数可见 MelodySimplehtml 文件夹下的 Melody\_Simplehtml.h 和 Melody\_Simplehtml.cpp 文件。例程流程图如图 5-51, 例程代码如所示图 5-52。

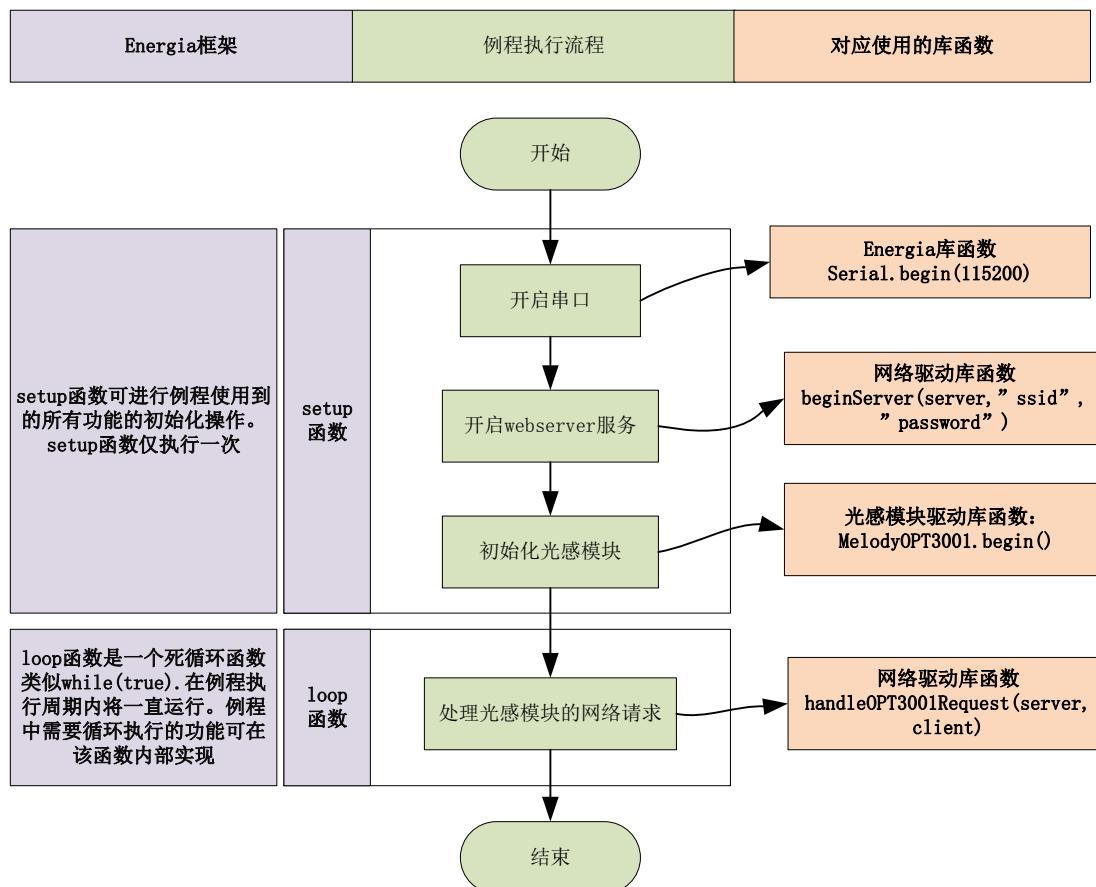


图 5-51 SimpleWebOPT3001 例程流程图

```

#include <stdbool.h>
#include <stdint.h>
#include <WiFi.h>
#include "Wire.h"
#include "Melody_OPT3001.h"

```

```
#include "Melody_Simplehtml.h"

WiFiServer server(80);
WiFiClient client;
void setup()
{
// put your setup code here, to run once:
Serial.begin(115200);
beginServer(server,"ssid","password");
MelodyOPT3001.begin();
}

void loop()
{
// put your main code here, to run repeatedly:
handleOPT3001Request(server,client);
}
```

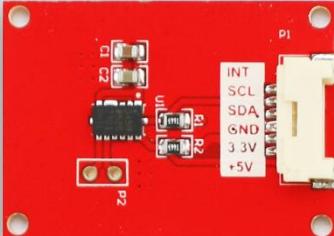
图 5-52 SimpleWebOPT3001 例程源程序

### 5.2.7 三轴加速度模块 (*MelodyADXL345*)

#### 5.2.7.1 模块

三轴加速度使用 I2C 通信，模块如下：

表 5-34 三轴加速度模块

三轴加速度模块	概况
	<p>1 模块使用 ADXL345      2 使用 I2C 通信接口      3 可进行单振/双振检测，活动/非活动检测，自由落体检测</p>

### 5.2.7.2 实现原理及原理图简介

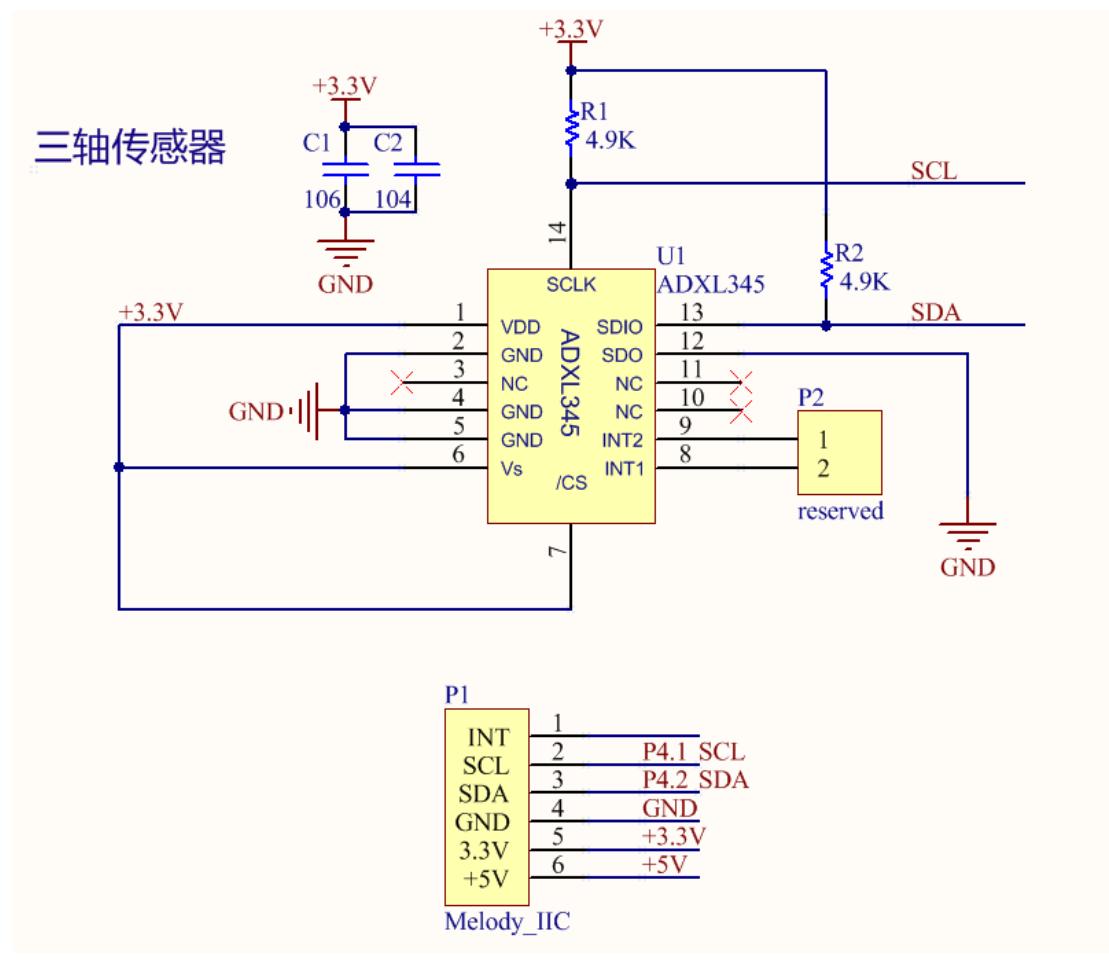


图 5-53 三轴加速度原理图

### 功能框图

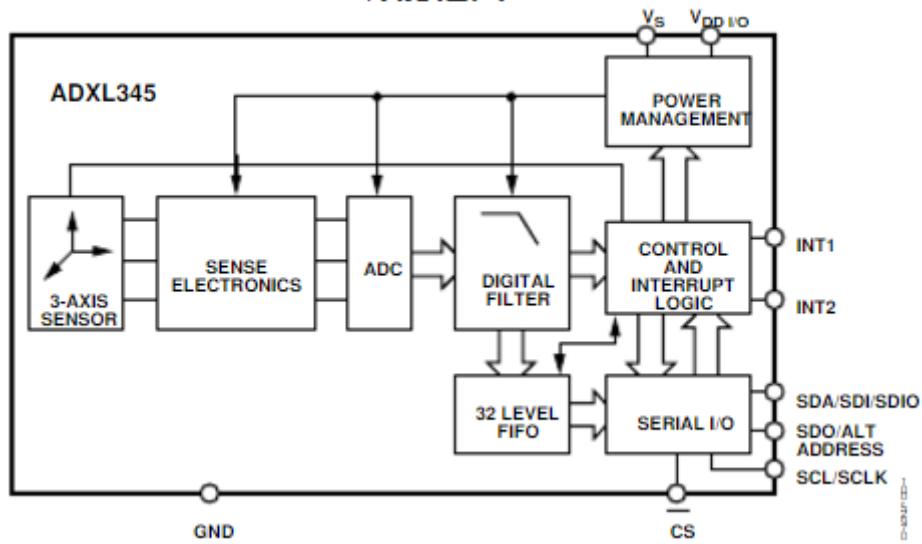


图 5-54 ADXL345 内部结构

ADXL 是一款小而薄的超低功耗 3 轴加速度传感器，分辨率高（13 位），测量范围±16g，

数字输出数据为 16 位二进制补码格式，可通过 I2C 接口进行访问。

### 5.2.7.3 软件实现

三轴加速度模块使用 I2C 接口，Energia 已经实现了 I2C 库函数，用户直接使用即可。I2C 库函数，可在 <http://energia.nu/reference/wire/> 查看，可以直接查看“5.2.5.3.1 I2C 库函数”对 I2C 库函数的使用介绍

#### 5.2.7.3.1 三轴加速度模块驱动库实现

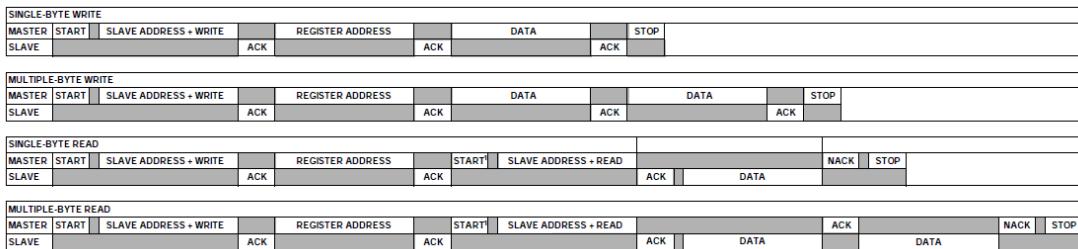


图 5-55 I2C 器件寻址

图 5-55 给出了芯片——ADXL345 的 I2C 数据通信格式，其中前两个是“写时序”格式，后两个为“读时序”模式。

**写时序：**分成“单字节写入”和“多字节写入”。

“单字节写入”操作顺序：cc3200 作为 Master，ADXL345 作为 Slave。第一步：写入器件地址；第二步：写入待写入数据的寄存器地址；第三步：写入待写入数据。

“多字节写入”操作顺序：cc3200 作为 Master，ADXL345 作为 Slave。第一步：写入器件地址；第二步：写入待写入数据的寄存器地址；第三步：依次写入待写入数据。

程序实现如下：

```
/*
 * @brief: 写单字节数据
 * @param: regAddress, 寄存器地址
 *          data, 待写入数据
 * @return: none
 */

void CMelodyADXL345::write(uint8_t regAddress, uint8_t data)
{
    write(regAddress, &data, 1);
}

/*
 * @brief: 写入多个字节数据
 * @param: regAddress, 寄存器地址
 *          pdata, 指向待写入数据的指针
 *          length, 待写入数据长度
 */
```

```

* @return: none
*****
void CMelodyADXL345::write(uint8_t regAddress,uint8_t*pdata,uint8_t
                           length)
{
    Wire.beginTransmission(ADXLADDRESS);
    Wire.write(regAddress);
    Wire.write(pdata,length);
    Wire.endTransmission();
}

```

图 5-56 写时序程序实现

**读时序：**分成“单字节读取”和“多字节读取”。

“单字节读取”操作顺序：cc3200 作为 Master，ADXL345 作为 Slave。第一步：写入器件地址；第二步：写入待读取数据的寄存器地址；第三步：再次写入器件地址；第四步：读取待读取寄存器的数据。

“多字节读取”操作顺序：cc3200 作为 Master，ADXL345 作为 Slave。第一步：写入器件地址；第二步：写入待读取数据的寄存器地址；第三步：再次写入器件地址；第四步：依次读取数据。

程序实现如下：

```

*****
* @brief: 读取单字节数据
* @param: regAddress, 寄存器地址
* @return: 读取的数据
*****
uint8_t CMelodyADXL345::read(uint8_t regAddress)
{
    uint8_t data =0;
    Wire.beginTransmission(ADXLADDRESS);
    Wire.write(regAddress);
    Wire.endTransmission();

    Wire.requestFrom(ADXLADDRESS,1);
    if(Wire.available()>=1)
    {
        data = Wire.read();
    }
    return data;
}
*****
* @brief: 读取多字节数据
* @param: regAddress, 寄存器地址
*         pdata, 指向存放读取数据缓存指针
*         length, 待读取数据长度

```

```

* @return: 实际读取数据长度
*****
uint8_t CMelodyADXL345::read(uint8_t
                                regAddress,uint8_t*pdata,uint
                                t8_t length)
{
    uint8_t readlength =0;
    Wire.beginTransmission(ADXLADDRESS);
    Wire.write(regAddress);
    Wire.endTransmission();

    Wire.requestFrom(ADXLADDRESS,length);
    while(Wire.available())
    {
        pdata[readlength++]=Wire.read();
    }

    return readlength;
}

```

图 5-57 读时序程序实现

以上为 I2C 通信的“写时序”和“读时序”实现，其余驱动库函数均需要调用此两处函数。其余库函数如下：类对象“MelodyADXL345”定义在 Melody\_ADXL345.cpp 中。

表 5-35 begin() 函数

驱动库函数	void begin(void)
说明	初始化设置 ADXL345
使用范例	MelodyADXL345.begin();

表 5-36 getXAccelerate() 函数

驱动库函数	int16_t getXAccelerate (void)
说明	获取 X 轴数据
使用范例	int16_t x = MelodyADXL345.getXAccelerate();

表 5-37 getYAccelerate() 函数

驱动库函数	int16_t getYAccelerate (void)
说明	获取 Y 轴数据
使用范例	int16_t y = MelodyADXL345.getYAccelerate();

表 5-38 getZAccelerate() 函数

驱动库函数	int16_t getZAccelerate (void)
说明	获取 Z 轴数据
使用范例	int16_t z = MelodyADXL345.getZAccelerate();

表 5-39 getDeviceID()函数

驱动库函数	<code>uint8_t getDeviceID (void)</code>
说明	获取 ID 值
使用范例	<code>uint8_t id = MelodyADXL345.getDeviceID();</code>

### 5.2.7.3.2 实验例程

实验例程操作及结果展示可浏览视频资料：[http://www.hpati.com/product\\_videos/v74.html](http://www.hpati.com/product_videos/v74.html)

例程 1, ADXL345.ino:读取每个轴的数据，使用串口打印。实例流程图如图 5-58，实例代码如图 5-59。

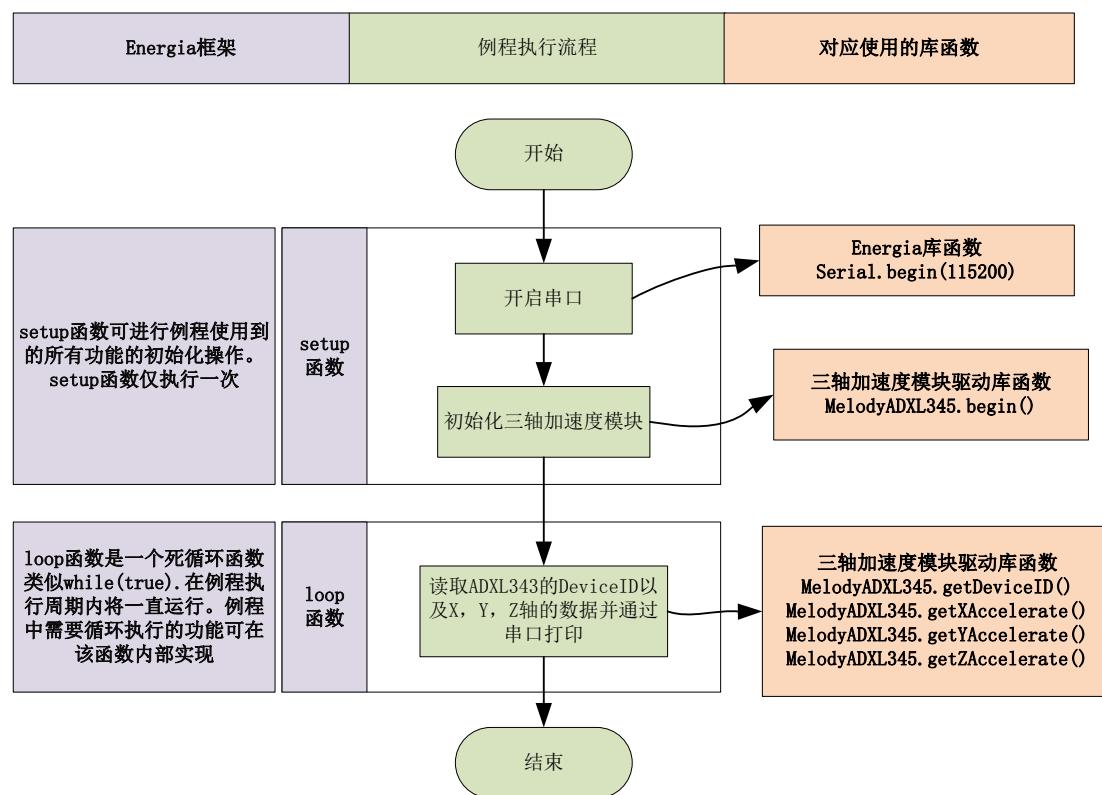


图 5-58 ADXL345 例程流程图

```
#include <stdbool.h>
#include <stdint.h>
#include "Wire.h"
#include "Melody_ADXL345.h"

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    MelodyADXL345.begin();
}
```

```
void loop()
{
// put your main code here, to run repeatedly:
Serial.println("-----");
Serial.print("ID:0x");
Serial.println(MelodyADXL345.getDeviceID(),HEX);
Serial.print("X Accelerate:");
Serial.println(MelodyADXL345.getXAccelerate());
Serial.print("Y Accelerate:");
Serial.println(MelodyADXL345.getYAccelerate());
Serial.print("Z Accelerate:");
Serial.println(MelodyADXL345.getZAccelerate());
Serial.println("-----");
delay(1000);
}
```

图 5-59 ADXL345 例程源程序

例程 2, SimpleWebADXL345.ino:通过网络读取光照强度。cc3200 设置为 webserver 模式，用户可是有浏览器访问 cc3200，实验过程中 cc3200 的 IP 地址通过串口打印至 PC 端。Energia 开发工具提供了串口监视工具。程序例程图如所示，网络驱动库函数可见 MelodySimplehtml 文件夹下的 Melody\_Simplehtml.h 和 Melody\_Simplehtml.cpp 文件。例程流程图如图 5-60，例程代码如所示图 5-61。

Energia框架	例程执行流程	对应使用的库函数
-----------	--------	----------

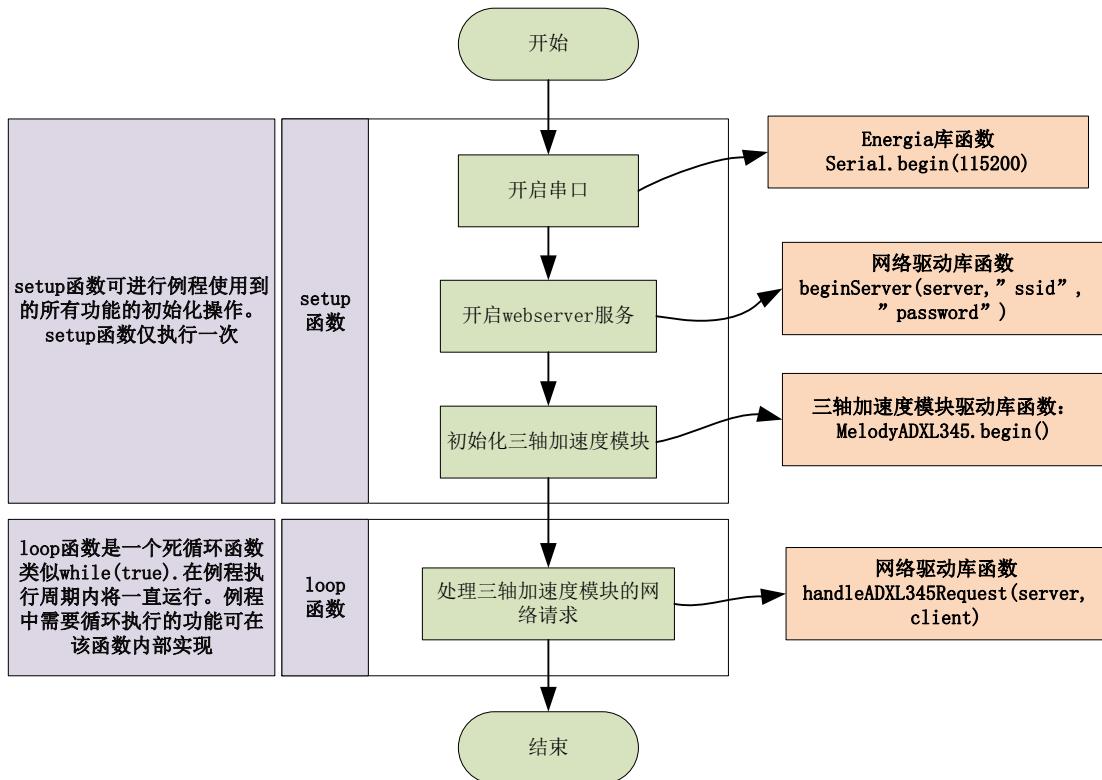


图 5-60 SimpleWebADXL345 例程流程图

```

#include <stdbool.h>
#include <stdint.h>
#include <WiFi.h>
#include <Wire.h>
#include "Melody_ADXL345.h"
#include "Melody_Simplehtml.h"

WiFiServer server(80);
WiFiClient client;

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    beginServer(server,"ssid","password");
    MelodyADXL345.begin();
}

void loop()
{
}

```

```
// put your main code here, to run repeatedly:  
handleADXL345Request(server,client);  
}
```

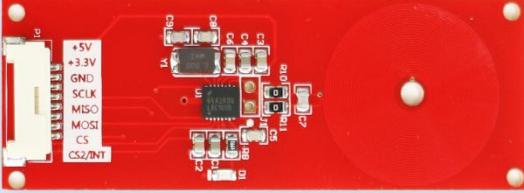
图 5-61 SimpleWebADXL345 例程源程序

## 5.2.8 LDC1000 模块 (*MelodyLDC1000*)

### 5.2.8.1 模块

LDC1000 使用 SPI 通信，模块如下

表 5-40 LDC1000 模块

LDC1000 模块	概况
	<p>1、SPI 接口通信 2、可对线性位置或者角位置、位移、运动、压缩、振动以及金属成分进行测量</p>

### 5.2.8.2 实现原理及原理图简介

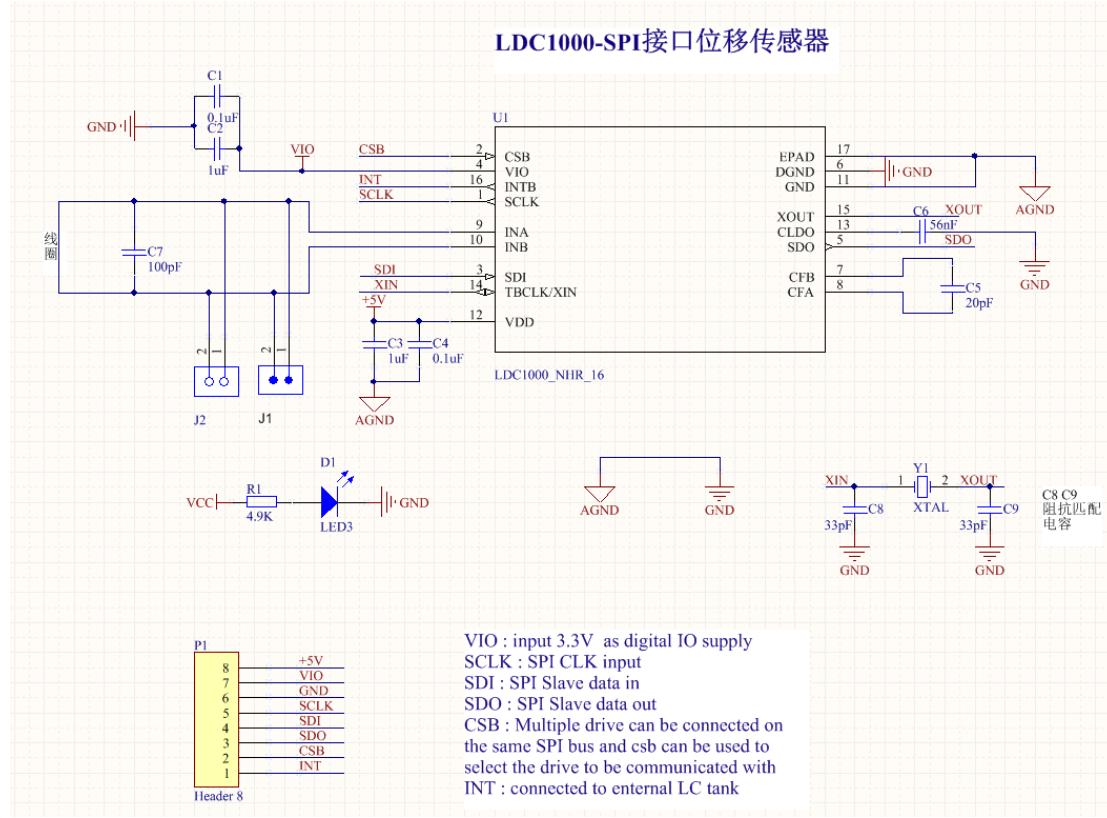


图 5-62 LDC1000 模块原理图

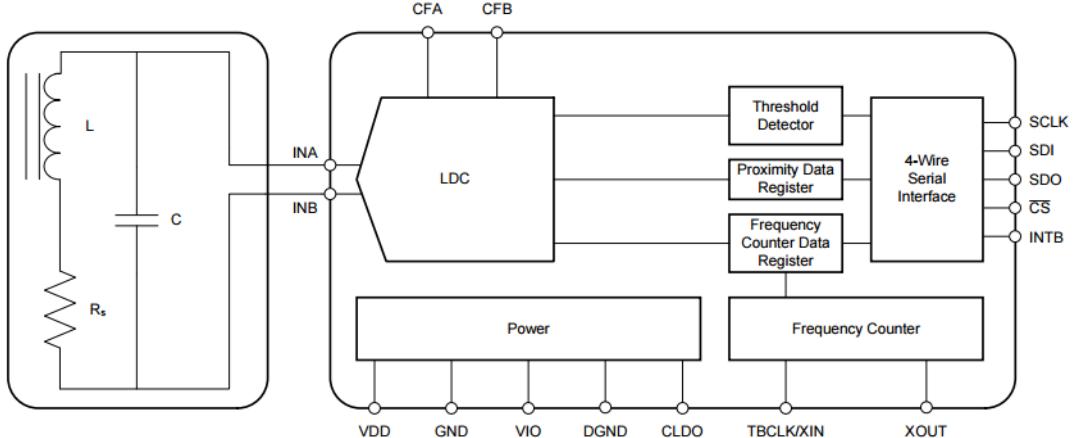


图 5-63 LDC1000 功能结构图

LDC1000 是世界首款电感到数字转换器。提供低功耗，小封装，低成本解决方案。SPI 接口很方便连接 MCU。LDC1000 只需要外接一个 PCB 线圈或者自制线圈就可以实现非接触式电感检测。LDC1000 的电感检测并不是指像 Q 表那样测试线圈的电感量，而是可以测试外部金属物体和 LDC1000 的测试线圈的空间位置关系。

利用 LDC1000 这个特性配以外部设计的金属物体即可很方便实现：水平或垂直距离检

测；角度检测；位移监测；运动检测；振动检测；金属成分检测（合金检测）。可以广泛的应用在汽车、消费电子、计算机、工业、通信和医疗领域。

LDC1000 测量电感频率是用测试 LC 谐振频率的方法。LDC1000 有外部的基准时钟，也是使用计数方法来做频率计。看一下手册中给出的公式就能清晰的看出计数原理。

$$f_{sensor} = (1/3) \times (F_{ext} / F_{count}) \times (responseTime)$$

$f_{sensor}$  是 LC 谐振频率， $F_{ext}$  是外部基准时钟频率， $F_{count}$  是 LDC1000 内部计数器值（0x23,0x24,0x25），Response time 是寄存器设定的一个值（0x04）。

将上面公式左右分别求倒数

$$1/f_{sensor} = 3 \times (F_{ext} / F_{count}) \times (responseTime)$$

将 response time 移动到等式左面

$$responseTime \times (1/f_{sensor}) = 3 \times F_{count} \times (1/F_{ext})$$

这样看就很清晰了， $1/f_{sensor}$  是 LC 谐振周期， $1/F_{ext}$  是基准时钟周期，也就是说在 response time 个 LC 谐振周期内，使用 LDC1000 的  $F_{count}$  计数器记录基准时钟的个数用来推算 LC 的谐振频率。

根据  $f_{sensor} = 1/\sqrt{2 * \pi * L * C}$  计算出  $L$ 。（ $C$  是电路设计中的已知量）

$$L = 1/[C \times (2 \times \pi \times f_{sensor})^2]$$

### 5.2.8.3 软件实现

LDC1000 模块使用 SPI 接口，同 I2C 功能一样，Energia 已经实现了 SPI 库函数，用户直接使用即可。SPI 库函数，可在 <http://energia.nu/reference/spi/> 查看。

#### 5.2.8.3.1 SPI 库函数

根据 Energia 官网上的介绍，SPI 库函数使用 C++ 实现，由多个成员函数构成，用户可直接使用库文件已经定义的“SPI”对象调用其成员函数，如 SPI.begin()。各成员函数如图 5-30 所示。

### Functions

- begin()
- end()
- setBitOrder()
- 
- setClockDivider()
- setDataMode()
- transfer()
- setModule()

## SPI Library

This library allows you to communicate with SPI devices, with the Energia target board as the master device.

### A Brief Introduction to the Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by microcontrollers for communicating with one or more peripheral devices quickly over short distances. It can also be used for communication between two microcontrollers.

图 5-64 SPI 库函数

SPI 通信分成写数据和读数据，下面先简单介绍 SPI 库函数的使用，然后根据 LDC1000 的时序图介绍 LDC1000 驱动库的实现。

**SPI 基本写过程**，流程图如下：

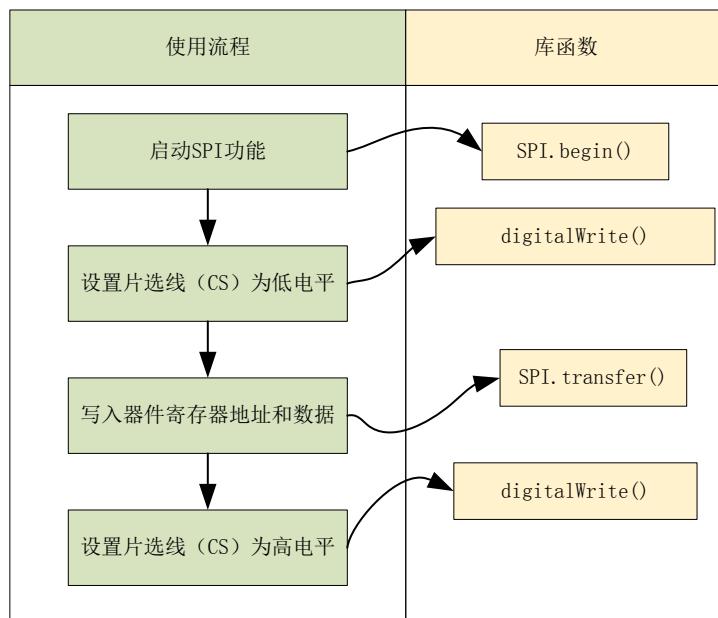


图 5-65 SPI 基本写过程流程图

**SPI 基本读过程**，流程图如下：

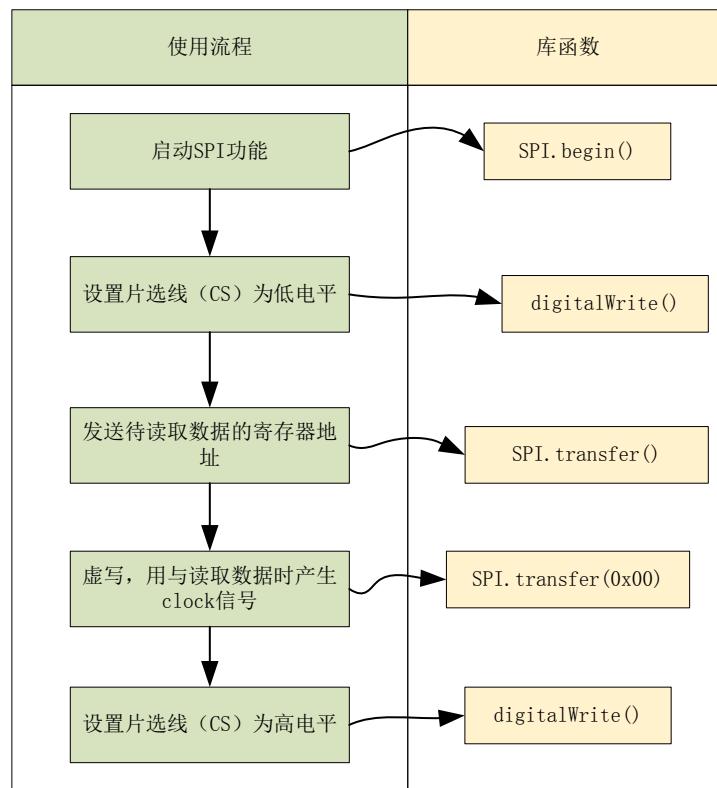


图 5-66 SPI 基本读过程流程图

如需查看 Energia 例程代码，可在 Energia 开发工具的菜单栏->File->Examples->SPI 下查看。

### 5.2.8.3.2 LDC1000 模块驱动库实现

LDC1000 时序如下：

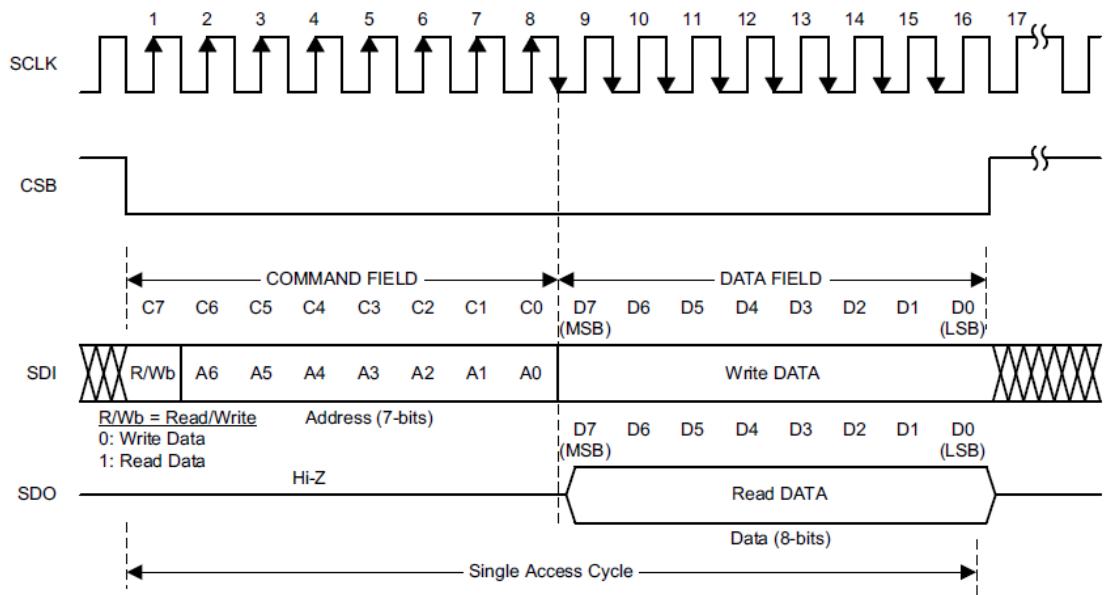


图 5-67 LDC1000 时序

结合图 5-65 和图 5-67 实现写操作程序如下：(不包括启动 SPI 功能，SPI 启动仅需调用

SPI.begin()即可，并且在使用过程中仅可被调用一次。)

```
***** @brief: 写数据
* @param: regAddr, 寄存器地址
*          data, 数据
* @return: none
*****
void CMelodyLDC1000::write(uint8_t regAddr,uint8_t data)
{
    digitalWrite(LDCSPI_CS,LOW);

    SPI.transfer(regAddr | LDCSPI_WBIT);
    SPI.transfer(data);

    digitalWrite(LDCSPI_CS,HIGH);
}
```

图 5-68 LDC1000 写操作实现

结合图 5-66 和图 5-67 实现写操作程序如下：(不包括启动 SPI 功能，SPI 启动仅需调用 SPI.begin()即可，并且在使用过程中仅可被调用一次。)

```
***** @brief: 读取数据
* @param: regAddr, 寄存器地址
* @return: none
*****
uint8_t CMelodyLDC1000::read(uint8_t regAddr)
{
    uint8_t result =0;

    digitalWrite(LDCSPI_CS,LOW);

    SPI.transfer(regAddr | LDCSPI_RBIT);
    result = SPI.transfer(0x00);

    digitalWrite(LDCSPI_CS,HIGH);

    return result;
}
```

图 5-69 LDC1000 读操作实现

以上为 SPI 通信的“写操作”和“读操作”实现，其余驱动库函数均需要调用此两处函数。

部分库函数如下：类对象“MelodyLDC1000”定义在 Melody\_LDC1000.cpp 中。

表 5-41 begin() 函数

驱动库函数	<b>void begin()</b>
说明	开启 SPI 功能，初始化设置 LDC1000
使用范例	MelodyLDC1000.begin();

表 5-42 available() 函数

驱动库函数	bool available()
说明	判断是否可有数据可读
使用范例	MelodyLDC1000.available();

表 5-43 getProximity()函数

驱动库函数	<code>uint16_t getProximity()</code>
说明	获取 Proximity 数据
使用范例	<code>uint16_t proximity = MelodyLDC1000.getProximity();</code>

驱动库函数	uint32_t getFrequency()
说明	获取 Frequency 数据
使用范例	uint16_t proximity = MelodyLDC1000.getFrequency();

表 5-44 calcInductance() 函数

驱动库函数	float calcInductance(uint32_t frequency)
说明	计算互感值、 frequency: 频率值, 该值可由函数 getFrequency() 获取
使用范例	uint16_t proximity = MelodyLDC1000.getProximity(); float inductance = calcInductance(proximity);

### 5.2.8.3.3 实验例程

实验例程操作及结果展示可浏览视频资料：[http://www.hpati.com/product\\_videos/v77.html](http://www.hpati.com/product_videos/v77.html)

例程 1, LDC1000.ino: 获取 LDC1000 的 Proximity 和 Frequency, 并且计算互感值。例程流程图如图 5-70, 例程程序实现如图 5-71。

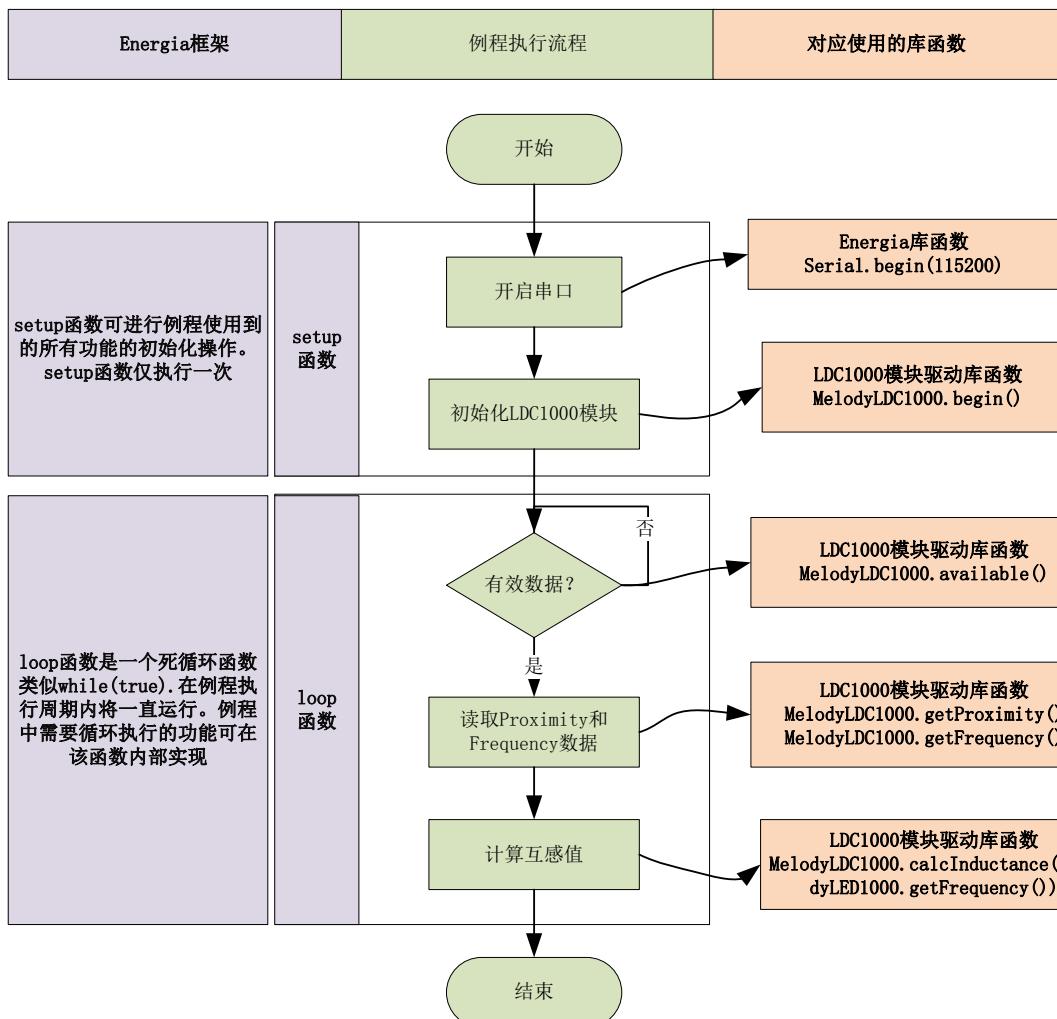


图 5-70 LDC1000 例程流程图

```
#include <stdbool.h>
#include <stdint.h>
#include <SPI.h>
#include "Melody_LDC1000.h"
uint16_t proximity =0;
uint32_t frequency =0;
void setup()
{
// put your setup code here, to run once:
  Serial.begin(115200);
  MelodyLDC1000.begin();
}
void loop()
{
// put your main code here, to run repeatedly:
if(MelodyLDC1000.available())
{

```

```

proximity = MelodyLDC1000.getProximity();
frequency = MelodyLDC1000.getFrequency();
Serial.print("Proximity:");
Serial.print(proximity);
Serial.print("\t\tFrequency:");
Serial.print(frequency);
Serial.print("\t\tInductance:");
Serial.println(MelodyLDC1000.calcInductance(frequency));
}
}

```

图 5-71 LDC1000 例程源程序

例程 2, SimpleWebLDC1000.ino:通过网络读取光照强度。cc3200 设置为 webserver 模式，用户可是有浏览器访问 cc3200，实验过程中 cc3200 的 IP 地址通过串口打印至 PC 端。Energia 开发工具提供了串口监视工具。程序例程图如所示，网络驱动库函数可见 MelodySimplehtml 文件夹下的 Melody\_Simplehtml.h 和 Melody\_Simplehtml.cpp 文件。例程流程图如图 5-72，例程代码如所示图 5-73。

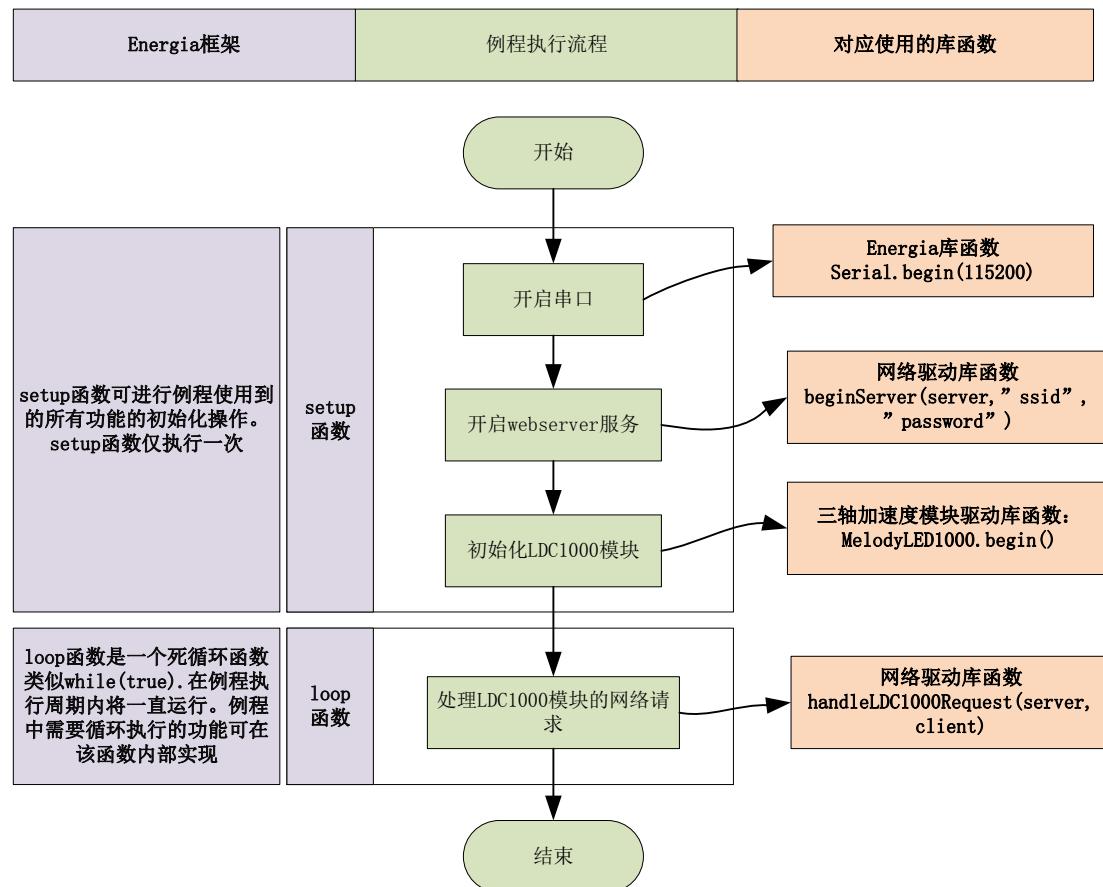


图 5-72 SimpleWebLDC1000 例程流程图

```

#include <stdbool.h>
#include <stdint.h>

```

```

#include <WiFi.h>
#include <SPI.h>
#include "Melody_LDC1000.h"
#include "Melody_Simplehtml.h"

WiFiServer server(80);
WiFiClient client;

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);
    beginServer(server,"ssid","password");
    MelodyLDC1000.begin();
}

void loop()
{
    // put your main code here, to run repeatedly:
    handleLDC1000Request(server,client);
}

```

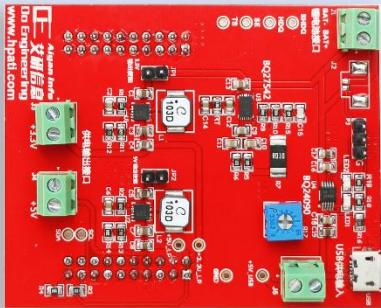
图 5-73 SimpleWebLDC1000 例程源程序

## 5.2.9 电池管理模块 (*MelodyLiBattery*)

### 5.2.9.1 模块

电池管理模块如下：模块的连接请查看《AY-IOT 连接使用须知》

表 5-45 电池管理模块

模块	概况
	<p>1、模块使用芯片 BQ24090 完成锂电池充电， BQ27542 完成锂电池参数监控； 2、BQ27542 使用 I2C 接口通信； 3、模块实现锂电池充放电功能； 4、可选择充电电流大小； 5、对外输出电压可选择 3V 和 5V</p>

### 5.2.9.2 实现原理及原理图简介

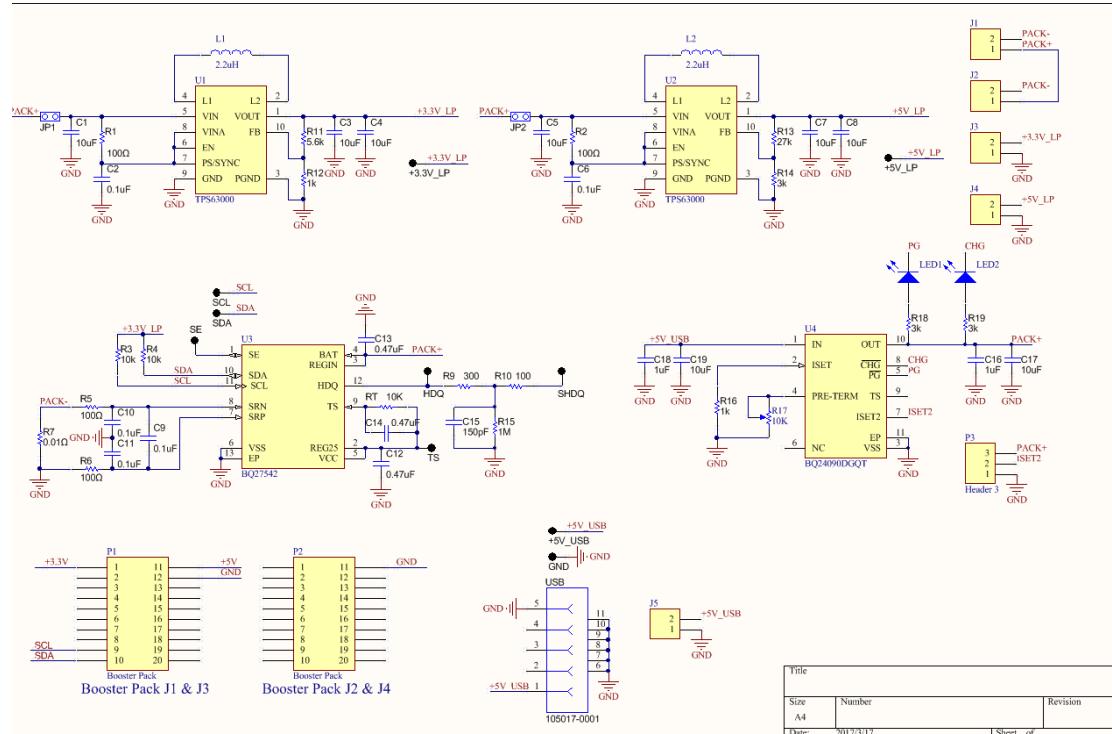


图 5-74 电池管理模块原理图

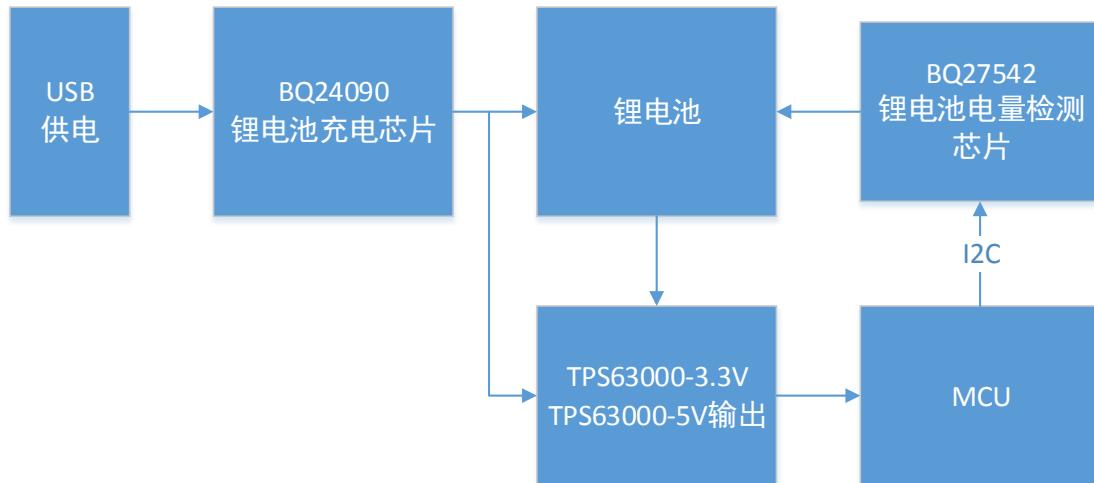


图 5-75 锂电池功能框图

本模块使用 USB 接口对整体进行供电，通过 BQ24090 对锂电池进行充电，BQ27542 对锂电池的电量进行检测，电量信息需通过 MCU 的 I2C 去读取 BQ27510 的内部信息。输出部分采用了两片 TPS63000，一片输出 3.3V，另一片输出 5V，用户可通过跳线帽 JP1 和 JP2 来选择使用

### 5.2.9.3 软件实现

#### 5.2.9.3.1 BQ27542 驱动库实现

部分库函数如下：类对象“MelodyLiBattery”定义在 Melody\_L.cpp 中。

表 5-46 begin()函数

驱动库函数	<b>void begin(void)</b>
说明	启动 I2C 功能
使用范例	MelodyLiBattery.begin();

表 5-47 getInternalTemperature()函数

驱动库函数	<b>float getInternalTemperature(void)</b>
说明	获取 BQ27542 内部温度
使用范例	float temperature = MelodyLiBattery.getInternalTemperature();

表 5-48 getVoltage()函数

驱动库函数	<b>uint16_t getVoltage(void)</b>
说明	获取当前电压值
使用范例	uint16_t voltage = MelodyLiBattery.getVoltage();

表 5-49 getAverageCurrent()函数

驱动库函数	<b>int16_t getAverageCurrent(void)</b>
说明	获取平均电流
使用范例	int16_t current = MelodyLiBattery.getAverageCurrent();

表 5-50 getRemainingCapacity()函数

驱动库函数	<b>uint16_t getRemainingCapacity(void)</b>
说明	获取当前剩余电量
使用范例	uint16_t capacity = MelodyLiBattery.getRemainingCapacity();

表 5-51 getStateOfCharge()函数

驱动库函数	<b>uint8_t getStateOfCharge(void)</b>
说明	获取当前剩余电量百分比
使用范例	uint8_t stateOfCharge = MelodyLiBattery.getStateOfCharge();

#### 5.2.9.3.2 实验例程

例程 1，LiBattery.ino: 获取当前模块的电压值，平均电流值，BQ27542 的内部温度值以及使用的锂电池的剩余电量和剩余电量百分比，这些数据都通过串口打印值 PC 串口助手。例程流程图如图 5-76，例程代码实现如图 5-77。

Energia框架	例程执行流程	对应使用的库函数
-----------	--------	----------

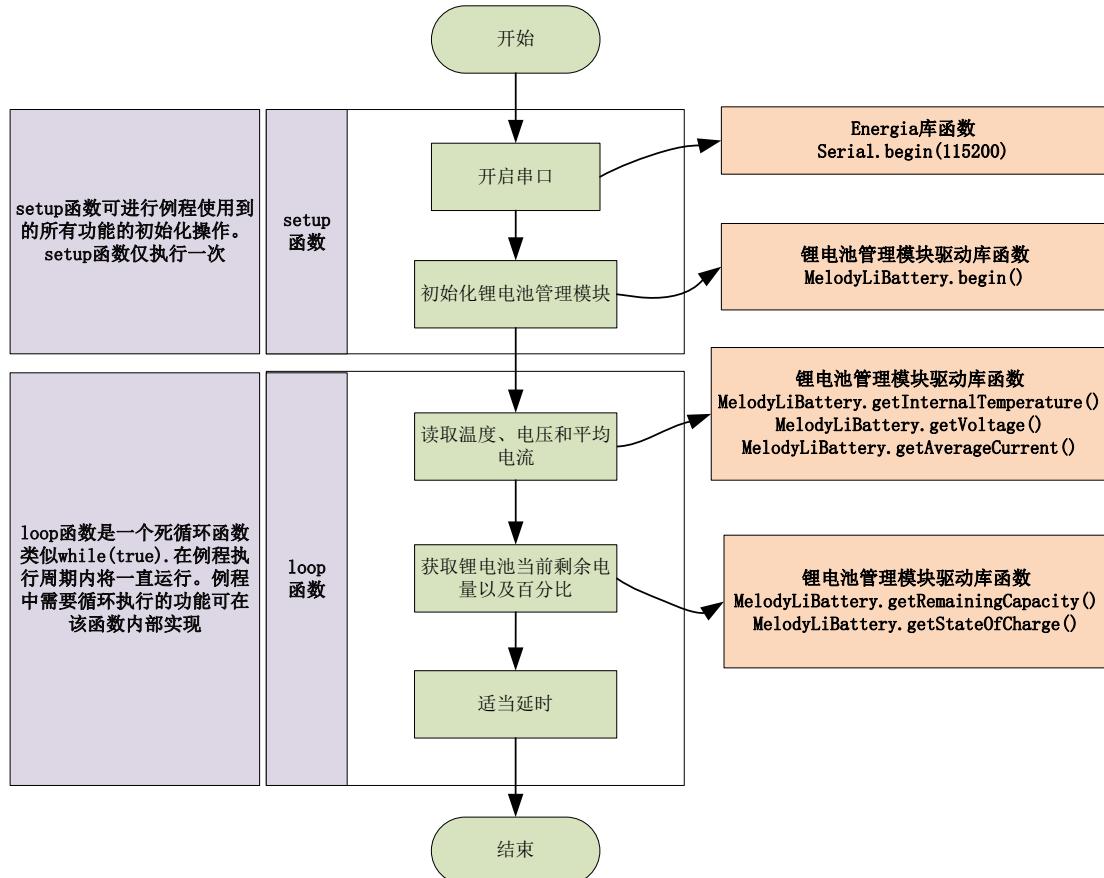


图 5-76 LiBattery 例程流程图

```

#include <stdbool.h>
#include <stdint.h>
#include "Wire.h"
#include "Melody_LiBattery.h"

void setup()
{
    // put your setup code here, to run once:
    MelodyLiBattery.begin();
    Serial.begin(115200);
}

void loop()
{
    // put your main code here, to run repeatedly:
    Serial.println("-----");
    Serial.print("Temperature:");
    Serial.print(MelodyLiBattery.getInternalTemperature());
    Serial.println(" 'C'");
}

```

```
Serial.print("Voltage:");
Serial.print(MelodyLiBattery.getVoltage());
Serial.println(" mV");
Serial.print("Current:");
Serial.print(MelodyLiBattery.getAverageCurrent());
Serial.println(" mA");
Serial.print("Remaining Capacity:");
Serial.print(MelodyLiBattery.getRemainingCapacity());
Serial.println(" mAH");
Serial.print("State of charge:");
Serial.print(MelodyLiBattery.getStateOfCharge());
Serial.println(" %");
Serial.println("-----");
delay(2000);
}
```

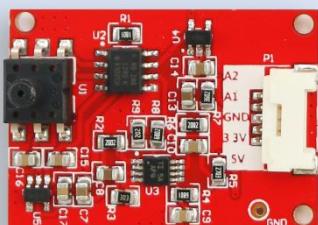
图 5-77 LiBattery 例程源程序

### 5.2.10 血压模块 (*MelodyBP*)

#### 5.2.10.1 模块

血压模块如表 5-52

表 5-52 血压模块

模块	概况
	<ul style="list-style-type: none"><li>1 使用芯片 MPS-3117</li><li>2 测量范围 5.8PSI</li><li>3 使用 AD 接口采集数据</li><li>3 可实现静态压和血压波动数据采集</li></ul>

### 5.2.10.2 实现原理及原理图简介

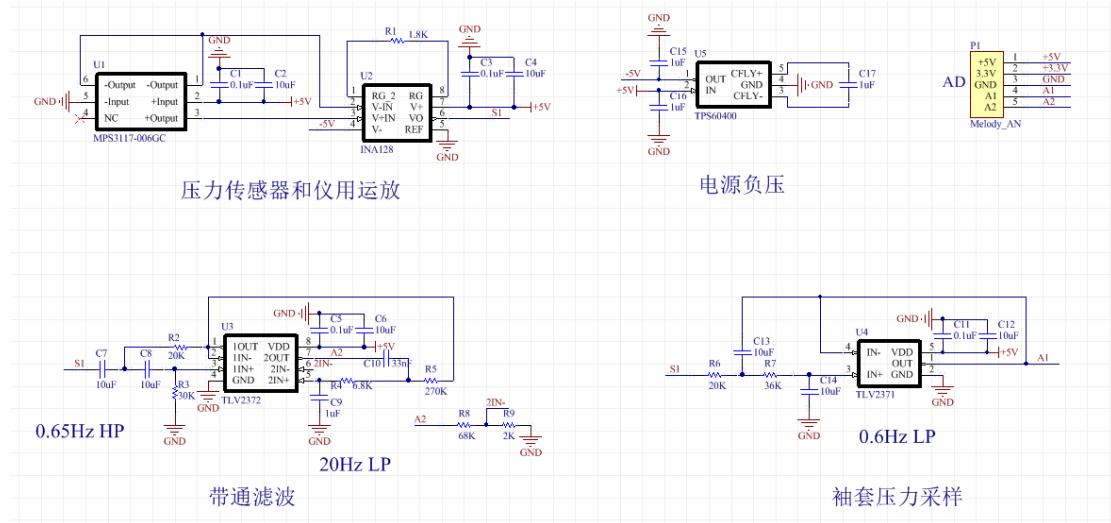


图 5-78 血压模块原理图

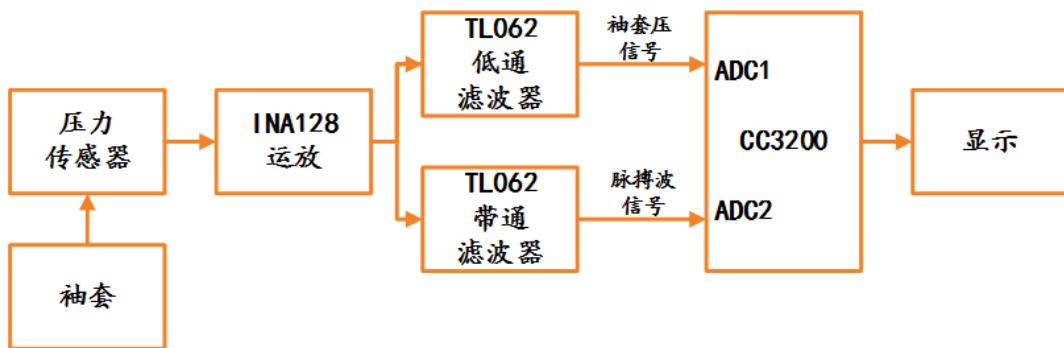


图 5-79 血压模块功能框图

硬件模块主要包括：CC3200、MPS-3117-006GC 压力传感器、INA128 精密低功耗仪器放大电路、TLV2371 低通、TLV2372 带通滤波器。

压力传感器是电子血压计的核心部分之一，关系到整个系统的精度。本文采用的是台湾 METRODYN 全磊微机电公司的 MPS3117-006GC 压阻式压力传感器，可经由定电压(5 V)或定电流(1, 1.5 mA)驱动产生正比于输入压力之毫伏等级电压输出讯号，具有优异的性能与长时间稳定性。MPS3117-006GC 具有以下特点：

- (1) 测量范围为 0~5.8PSI, 即 0~300mmHg;
- (2) 操作温度范围-40~85°C;
- (3) 供电电压: 5V;
- (4) 满量程输出: 75mV;
- (5) 灵敏度: 0.25mV/mmHg

由于压力传感器输出电压较小，使用 INA128 仪器放大器对输出的压力信号进行放大，放大倍数  $G=1+50K\Omega/R_g$ ,  $R_g$  选择  $1800\Omega$ ，即放大倍数  $G=28.78$ ，血压计的使用范围一般为 0~180mmHg，即 0~45mV，经放大后得到的压力信号范围为 0~1.295V。

### 5.2.10.3 软件实现

血压模块需要配套上位机软件一起使用，软件下载链接：  
[http://www.hpati.com/ay\\_tools\\_download/](http://www.hpati.com/ay_tools_download/)，软件界面如下：

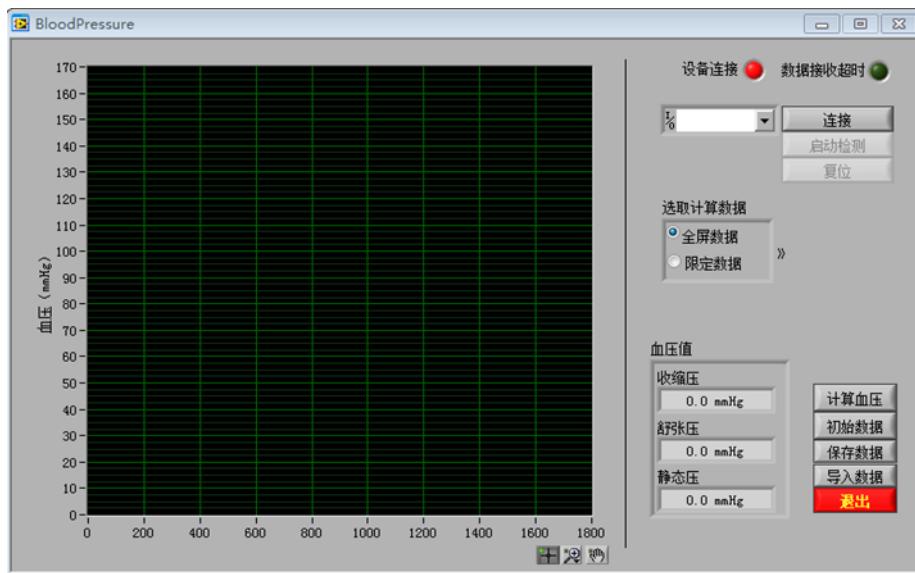


图 5-80 血压模块上位机软件

软件实现如下功能：

- 1 完成与血压模块之间的通信；
- 2 对血压模块采集的数据进行分析，波峰检测；
- 3 计算静态压、收缩压、舒张压；
- 4 保存和导入固定格式数据；
- 5 可将采集数据导出成 excel 格式数据；
- 6 可导出图像数据；

对于软件操作可查看视频：<http://www.hpati.com/IOTKIT/v82.html>

### 5.2.10.3.1 血压模块驱动库实现

模块部分库函数如下：类对象“MelodyBP”定于在 Melody\_BP.cpp 文件中。

表 5-53 begin() 函数

驱动库函数	<b>void begin(void)</b>
说明	初始化设置模块
使用范例	MelodyBP.begin();

表 5-54 measurement() 函数

驱动库函数	<b>bool measurement(void)</b>
说明	完成检测过程，返回值 bool 类型： true， 检测完成； false， 检测未完成。
使用范例	bool state = MelodyBP.measurement();

表 5-55 transmitBoolPressure()函数

驱动库函数	<b>void transmitBoolPressure(HardwareSerial &amp;h_serial)</b>
说明	发送读取的血压数据至上位机。 参数: h_serial:使用的串口引用
使用范例	MelodyBP.transmitBoolPressure(Serial);

表 5-56 receiveComPacket()函数

驱动库函数	<b>char* receiveComPacket(HardwareSerial &amp;h_serial,char* p_Combuffer,uint8_t Comlength)</b>
说明	接收串口数据 参数: h_serial, 使用的串口引用 p_Combuffer, 接收数据缓存 Comlength, 接受数据长度
使用范例	MelodyBP.receiveComPacket(Serial,buffer,10);

表 5-57 available()函数

驱动库函数	<b>bool available(void)</b>
说明	判断数据是否接受完成 返回值: true, 接受完成 false, 未接受完成
使用范例	bool complete = MelodyBP.available();

表 5-58 handleComPacket()函数

驱动库函数	<b>eComState handleComPacket(HardwareSerial &amp;h_serial, char* p_Combuffer)</b>
说明	处理接受的数据包 参数: h_serial, 使用的串口引用 p_Combuffer, 数据缓存 返回值: eComState, 枚举类型, 定义在 Melody_BP.h 文件 值为: IDLE, ONLINE, MEASURE, RESET。用来表示数据包 类型
使用范例	eComState state = MelodyBP.handleComPacket(Serial,buffer);

表 5-59 clear()函数

驱动库函数	<b>void clear(void)</b>
说明	清除数据和状态
使用范例	MelodyBP.clear();

### 5.2.10.3.2 实验例程

实验例程操作及结果展示可浏览视频资料: [http://www.hpati.com/product\\_videos/v82.html](http://www.hpati.com/product_videos/v82.html)

例程 1, BloodPressureSample.ino: 与上位机程序配套使用, 完成血压数据的采集和上传。例程流程图如图 5-81, 程序实现如图 5-82。

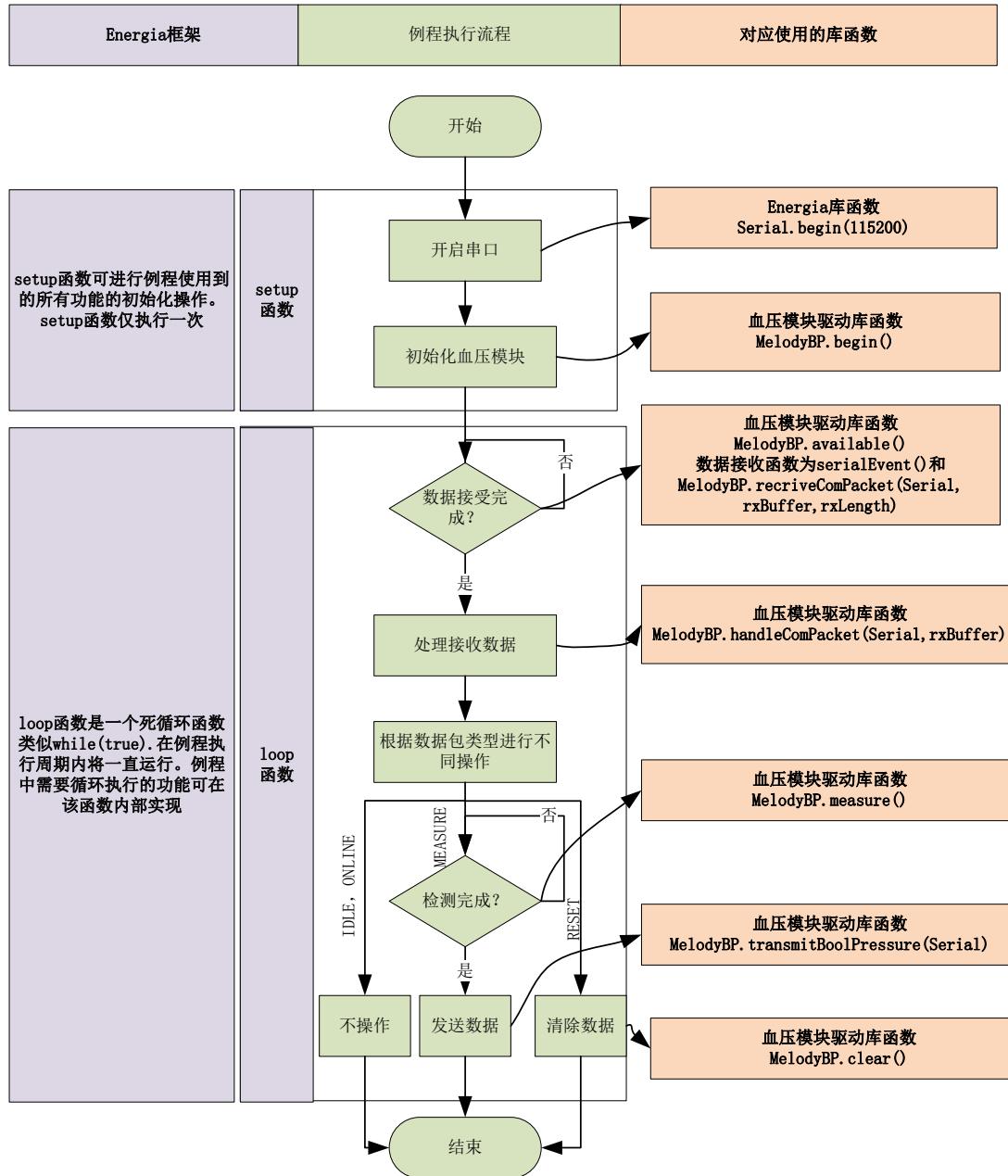


图 5-81 BloodPressureSample 例程流程图

```
#include <stdint.h>
#include <stdbool.h>
#include "Melody_BP.h"

const uint8_t rxLength = 10;
char rxBuffer[rxLength] = {0};

eComState runState = IDLE;
```

```
void setup()
{
// put your setup code here, to run once:
Serial.begin(115200);
MelodyBP.begin();
}

void loop()
{
// put your main code here, to run repeatedly:
//if(MelodyBP.available() && (runState != MEASURE))
if(MelodyBP.available())
{
    runState = MelodyBP.handleComPacket(Serial,rxBuffer);
    memset(rxBuffer,0,rxLength);
}
switch(runState)
{
case IDLE:
break;
case ONLINE:
//do something

    runState = IDLE;
break;
case MEASURE:
if(MelodyBP.measurement())
{
    MelodyBP.transmitBoolPressure(Serial);
    runState = IDLE;
}
break;
case RESET:
    MelodyBP.clear();
    runState = IDLE;
break;
default:
break;
}
}

void serialEvent()
{
    MelodyBP.receiveComPacket(Serial,rxBuffer,rxLength);
}
```

}

图 5-82 BloodPressureSample 例程源程序