

1. Course Description
2. Download supporting resources
3. Development Kit Introduction
 - 3.1. Introduction
 - 3.2. Hardware composition & technical parameters
 - 3.3. Circuit Schematic & PCB Diagram - Standard Board
 - 3.4. Circuit Schematic & PCB Diagram - MiNi Board
 - 3.5. CC2530F256 module specifications
 - 3.6. WiFi module specifications
 - 3.7. Communication distance test
4. Part 1: Preparation
 - 4.1. Introduction to ZigBee 3.0 that even a novice can understand
 - 4.2. IAR EW for 8051 Introduction and Installation
 - 4.3. TI Z-Stack 3.0 Introduction and Installation
 - 4.4. SmartRF Flash Programmer Download and Installation
 - 4.5. Serial Port Assistant Introduction and Installation
 - 4.6. SmartRF04EB Driver
 - 4.7. USB to Serial Port Driver
 - 4.8. Other software installation (optional)
 - 4.8.1. Xshell 7 Introduction and Installation Guide
 - 4.8.2. PuTTY Introduction and Installation
5. Part 2: Introduction to 51 MCU - Based on CC2530
 - 5.1. Chapter 1: CC2530 Development Basic Experiment
 - 5.1.1. Create a new workspace and project
 - 5.1.2. Source code writing and compilation
 - 5.1.3. Program download and simulation
 - 5.1.4. Firmware Burning
 - 5.2. Chapter 2: GPIO Experiment
 - 5.2.1. Multi-project management foundation
 - 5.2.2. GPIO output experiment - LED control
 - 5.2.3. GPIO input experiment - mechanical buttons
 - 5.2.4. GPIO input and output general configuration experiment
 - 5.2.5. GPIO external interrupt experiment
 - 5.3. Chapter 3: Timer Experiment
 - 5.3.1. Project Overview
 - 5.3.2. Timer T1 Experiment - Query Trigger
 - 5.3.3. Timer T3 Experiment - Interrupt Trigger
 - 5.3.4. Watchdog Timer Experiment
 - 5.3.5. Low Power Timer Experiment
 - 5.4. Chapter 4: Serial Communication Experiment
 - 5.5. Chapter 5: ADC Experiment - Using the Light Sensor
 - 5.6. Chapter 6: OLED Display Experiment
 - 5.7. Chapter 7: Peripheral Experiments
 - 5.7.1. DHT11 Temperature and Humidity Sensor
 - 5.7.2. NorFLASH reading and writing experiment
 - 5.7.3. Relay Control Experiment
6. Part 3: Z-Stack 3.0 Detailed Explanation
 - 6.1. Chapter 1: Detailed Explanation of Z-Stack 3.0 Architecture
 - 6.1.1. Z-Stack 3.0.1 File Organization
 - 6.1.2. Z-Stack 3.0.1 Project Framework
 - 6.2. Chapter 2: Task Scheduling Principles of Operating Systems
 - 6.3. Chapter 3: OSAL Detailed Explanation
 - 6.3.1. OSAL Task Scheduling Principle
 - 6.3.2. Task initialization and event processing
 - 6.3.3. Application of Z-Stack Events
 - 6.3.4. Using Dynamic Memory
 - 6.4. Chapter 4: Hardware Adaptation Layer Application - LED
 - 6.4.1. HAL file structure and project structure
 - 6.4.2. HAL Architecture Overview
 - 6.4.3. Introduction to LED API
 - 6.4.4. LED Experiment
 - 6.5. Chapter 5: Hardware Adaptation Layer Application - Buttons
 - 6.5.1 Keystroke Experiment
 - 6.5.2 Detailed Explanation of HAL Key Framework (Optional)
 - 6.6. Chapter 6: Hardware Adaptation Layer Application - Serial Port

6.7. Chapter 7: Hardware Adaptation Layer Application - Display Screen

6.8. Chapter 8: Hardware Adaptation Layer Application - ADC

7. Part 4: ZigBee 3.0 Network Programming

7.2. Chapter 1: ZigBee 3.0 Network Principles

7.1.1. Protocol Hierarchy

7.1.2. IEEE 802.15.4 Protocol

7.1.3. Network Layer

7.2. Chapter 2: ZigBee 3.0 BDB

7.2.1. Introduction to BDB

7.2.2. BDB Commissioning Modes

7.2.3. ZigBee 3.0 Networking Experiment

7.3. Chapter 3: Data Communication Based on AF

7.3.1. Simple Descriptor

7.3.2. Communication Principle

7.3.3. Introduction to Data Sending API

7.3.4. ZigBee 3.0 Communication Experiment

7.4. Chapter 4: ZCL Basic Principles

7.4.1. Introduction to ZCL

7.4.2. Detailed explanation of ZCL content

7.5. Chapter 5: Switch command transmission and reception based on ZCL

7.5.1. Application layer calls ZCL API

7.5.2. ZCL switch command sending and receiving API

7.5.3. ZCL switch command sending and receiving experiment

7.6. Chapter 6: Attribute reading and writing based on ZCL

7.6.1. ZCL attribute read and write API

7.6.2. ZCL attribute reading and writing experiment

7.7. Chapter 7: Attribute reporting experiment based on ZCL

7.7.1. Overview

7.7.2. Terminal Equipment Development

7.7.3. Coordinator Device Development

7.7.4. Simulation Debugging

8. Extracurricular: Project Practice

8.1. ZigBee 3.0 Environmental Information Collection (1)

8.1.1. System Introduction

8.1.2. Source code download

8.1.3. Hardware List

8.1.4. System Construction

8.1.5. Source code description

8.2. ZigBee 3.0 Environmental Information Collection (2)

8.2.1. System Introduction

8.2.2. Source code download

8.2.3. Hardware List

8.2.4. Using Tencent Cloud

8.2.5. Download and install MQTT.fx

8.2.6. MQTT.fx connects to Tencent Cloud

8.2.7. MQTT.fx publishes messages

8.2.8. Configuring the Development Board

8.2.9. Burn the coordinator firmware

8.2.10. Burning Router Firmware

8.2.11. Source code description

8.3. ZigBee Temperature and Humidity Monitoring & Sound and Light Alarm System (1)

8.3.1. System Introduction

8.3.2. Source code download

8.3.3. Hardware List

8.3.4. System Construction

8.3.5. Source code description

8.4. ZigBee automatic light on - based on light intensity & human body monitoring (1)

8.4.1. System Introduction

8.4.2. Source code download

8.4.3. Hardware List

8.4.4. System Construction

8.4.5. Source code description

8.5. File transmission system based on ZigBee

8.6. ZigBee-based automatic curtain opening and closing

8.7. ZigBee-based temperature, humidity & light intensity acquisition system

8.8. Agricultural environmental information collection based on ZigBee

8.8.1. System Introduction

8.8.2. Source code download

8.8.3. Hardware List

8.8.4. Using a private cloud server

8.8.5. Configuring the Development Board

8.8.6. Networking and WiFi Connection

8.8.7. Using Android APP

8.8.8. Source code description

8.9. Common problems in project construction

9. Extracurricular: Advanced Electives

9.1. Description of "Extracurricular: Advanced Elective"

9.2. Chapter 1: Serial Communication Protocol Design

9.2.1. Design basis

9.2.2. Protocol Format

9.3. Chapter 2: Optimizing the Coordinator Project Structure

9.3.1. Project Structure

9.3.2. Detailed explanation of application framework

9.3.2.1. Framework Description

9.3.2.2. zbmsg

9.3.2.3. zbcategory

9.4. Chapter 3: Coordinator host computer debugging

9.4.1. Host computer description

9.4.2. Debugging Instructions

9.5. Chapter 4: Dynamic Modification of Channels and PanId

9.5.1. Serial Port Protocol

9.5.2. Important interface description

9.5.2.1. NIB

9.5.2.2. NLME_UpdateNV

9.5.3. Architecture Adjustment

9.5.4. Application

9.5.4.1. zbnwk interface implementation

9.5.4.2. Serial communication analysis

9.5.4.3. Burning and debugging

9.6. Chapter 5: Obtaining Network Short Address and MAC Address

9.6.1. Interface Description

9.6.1.1. Description

9.6.1.2. Calling Process

9.6.1.3. Asynchronous Data

9.6.2. Debugging

9.7. Chapter 6: Network Access Control and Whitelist

9.7.1. Basic content

9.7.1.1. Network access control

9.7.1.2. Whitelist

9.7.2. Function Encapsulation

9.7.3. Program Debugging

9.8. Chapter 7: Coordinator Partition Storage Management

9.8.1. Software Framework

9.8.2. Application

9.8.3. Debugging

9.9. ZigBee 2 WiFi - Based on ESP8266

9.9.1. Use cloud server

9.9.2. Source code description and testing

9.10. Capturing and analyzing ZigBee wireless messages

9.11. Connect to Xiaomi Aqara smart socket and temperature and humidity sensor

9.12. NV Application of Z-Stack

9.12.1. Introduction to NV

9.12.2. NV reading and writing

9.13. HAL-based external FLASH application

9.14. TFT display experiment (optional)

9.15. Lighting project source code analysis

9.15.1. ZHA Lighting Project

9.15.2. ZHA Lighting source code analysis

9.15.3. Lighting brightness adjustment experiment

9.16. TemperatureSensor project source code analysis

9.16.1. ZHA TemperatureSensor Project

9.16.2. ZHA TemperatureSensor Source Code Analysis

9.17. Other extracurricular projects

9.17.1. Temperature and harmful gas SMS alarm system based on ZigBee

9.17.2. Multi-sensor detection and lighting alarm system based on ZigBee

9.17.3. Temperature, humidity, human infrared and sound and light alarm system based on ZigBee

9.17.4. ZigBee 3.0 multi-node networking practice

9.17.5. Temperature, humidity & signal strength detection system based on ZigBee

9.18. IAR EW for 8051 FAQ & Solutions

1. Course Description



Click to go to Bilibili to watch the video more clearly!

<https://www.bilibili.com/video/BV1k34y1D7Vz>

1. Introduction

This course will systematically and in-depth explain the principles and development technologies of ZigBee 3.0, and provide practical project explanations, so that students can not only learn the technical principles but also accumulate practical project experience.

2. Publication Statement

This course has been selected by **Tsinghua University Press**. Any similarity is considered plagiarism or piracy. Please refuse to use it!



3. Data Catalog

- [1. Course Description](#)
- [2. Download supporting resources](#)
- [3. Development Kit Introduction](#)
- [4. Part 1 - Preparation](#)
- [5. Part 2 - Introduction to 51 MCU - Based on CC2530](#)
- [6. Part 3 - Z-Stack 3.0 Detailed Explanation](#)
- [7. Part 4 - ZigBee 3.0 Network Programming](#)
- [8. Extracurricular - Project Practice](#)
- [9. Extracurricular - Advanced Electives](#)

4. For readers

- Beginners: Have a certain foundation in C/C++ language and simple MCU development
- Advanced users: those who want to learn more about ZigBee 3.0 technology principles or accumulate practical experience in ZigBee 3.0 projects

5. About the Author

- **The first Author: Michael**

Michael, former senior embedded software engineer at Xiaomi, has more than 7 years of development experience and has developed **Xiaomi** smart gateway, **Xiaomi** sensor, etc.

- (1) Expert in Linux/Android-based original BSP tailoring, kernel driver adaptation, software architecture design, and embedded C/C++ development
- (2) Familiar with mainstream IoT communication protocols such as ZigBee, WiFi, NB-IoT, 4G/5G, and related open source libraries

- **The second Author: Study**

is a former technical consultant of Emotibot and technical manager of JG. He has more than 5 years of experience in IoT industry solutions. He has provided technical solutions to well-known companies such as Kugou, SF Express, China Merchants Group and Skyworth TV. He is familiar with cutting-edge technologies such as AI, IoT and mobile development.



Official Information

Official website: www.sxf-iot.com
Official flagship store: <https://shop100413206.taobao.com>
WeChat public account: shanxuefang-iot



2. Download supporting resources

1. Resource Download

- **Development software** installation package:
<https://pan.baidu.com/s/1t0psD10befhcMbqvl1rt5g?pwd=2333>
Extraction code: 2333
- **Source code for Part 1 to Part 4 of this course:**
<https://pan.baidu.com/s/1MJ9jDmSsCNgLmaJJPfPzrA?pwd=2333>
Extraction code: 2333

2. Additional Information

- (1) After receiving the goods, the Taobao customer service robot will automatically send additional information;
(2) After receiving the goods, you can also click here to receive:



Note:

- (1) If you have already purchased the board, please be patient and wait. You cannot receive the card in advance for any reason.
(2) If you have not purchased the board, or if the school has purchased the board, do not ask the customer service robot for the information (the robot will not give it to you). You can ask friends, who have purchased the board or the school to get it. (3) You can get the card if you purchase any version of ZigBee module. You cannot receive the card if you have purchased other modules.

Additional information is as follows (continuously updated):

全部文件 > ... > ZigBee > 《ZigBee3.0开发指南》配套资源（附加部分）> 考外篇：项目实战

<input type="checkbox"/> 文件名		▲ 大小
<input type="checkbox"/> 其他项目 1	-	
<input type="checkbox"/> 其他项目 2	-	
<input type="checkbox"/>  ZigBee 3.0 环境信息采集 (1) .zip	64.58 MB	
<input type="checkbox"/>  ZigBee 3.0 环境信息采集 (1) (多节点版) .zip	281.51 MB	
<input type="checkbox"/>  ZigBee 3.0 环境信息采集 (2) .zip	56.09 MB	
<input type="checkbox"/>  ZigBee 温湿度监测 & 声光报警系统 (1) .zip	56.75 MB	
<input type="checkbox"/>  ZigBee 自动开灯——基...照度&人体监测 (1) .zip	56.40 MB	
<input type="checkbox"/>  基于ZigBee的光照自动开关窗帘.zip	38.80 MB	
<input type="checkbox"/>  基于ZigBee的农业环境信息采集.zip	69.76 MB	
<input type="checkbox"/>  基于Zigbee的温湿度 & 光照度采集系统.zip	33.58 MB	
<input type="checkbox"/>  基于ZigBee的温湿度 & 信号强度探测系统.zip	33.51 MB	
<input type="checkbox"/>  基于ZigBee的文件传送系统.zip	27.13 MB	

全部文件 > ... > ZigBee > 《ZigBee3.0开发指南》配套资源（附加部分）> 考外篇：进阶选修

<input type="checkbox"/> 文件名		▲ 大小
<input type="checkbox"/>  1. 串口通信协议设计.zip	423.00 B	
<input type="checkbox"/>  2. 优化协调器工程结构.zip	7.68 MB	
<input type="checkbox"/>  3. 协调器上位机调试.zip	9.56 MB	
<input type="checkbox"/>  4. 信道及PanId的动态修改.zip	12.92 MB	
<input type="checkbox"/>  5. ZigBee网络短地址及Mac地址的获取.zip	13.79 MB	
<input type="checkbox"/>  6. ZigBee网络层入网控制及白名单.zip	13.64 MB	
<input type="checkbox"/>  7. 协调器分区存储管理.zip	16.95 MB	
<input type="checkbox"/>  Lighting工程源码分析.zip	28.68 MB	
<input type="checkbox"/>  TemperatureSensor工程源码分析.zip	23.16 MB	
<input type="checkbox"/>  Z-Stack的NV应用.zip	16.07 MB	
<input type="checkbox"/>  ZigBee2Wifi_ESP8266.zip	27.21 MB	
<input type="checkbox"/>  必读说明.txt	533.00 B	
<input type="checkbox"/>  基于HAL的外部FLASH应用.zip	31.83 MB	
<input type="checkbox"/>  接入小米智能插座和温湿度传感器.zip	43.74 MB	

3. Join the exchange group

- (1) After receiving the goods, the Taobao customer service robot will automatically send an invitation to join the group;
(2) After receiving the goods, you can also click here to enter:



Join the group as soon as possible. There are many benefits in the group!

4. Evaluation and Encouragement

- Evaluation diagram



Your encouragement is the driving force for our **continuous updating and optimization**, thank you very much!

3. Development Kit Introduction

ZigBee 3.0

某米 | 工程师
设计



小白
也能学会

善学坊™

2023 新款

Official Store

<https://item.taobao.com/item.htm?id=683089996879>

Supporting Development Board Introduction

- 1. Introduction
- 2. Hardware composition & technical parameters
- 3. Circuit Schematic & PCB Diagram - Standard Board
- 4. Circuit Schematic & PCB Diagram - MiNi Board
- 5. CC2530F256 module specifications
- 6. WiFi module specifications
- 7. Communication distance test

3.1. Introduction

Official Store

<https://item.taobao.com/item.htm?id=683089996879>

Introduction

产品清单



ZigBee 标准板



WiFi 模块



USB线x2 (赠送)



ZigBee Mini板



仿真器

- 仿真线x1
- Mini5P线x1

— 善学坊 — IoT 低代码开发

*以上为推荐搭配清单，不代表实际清单

— ZigBee 开发套件 —

基于 ZigBee 3.0



课程设计

智能家居

工业物联

智慧农业

物联网开发套件系列 ... IoT development ... ZigBee 3.0 架构

更低功耗

更稳定

更高兼容

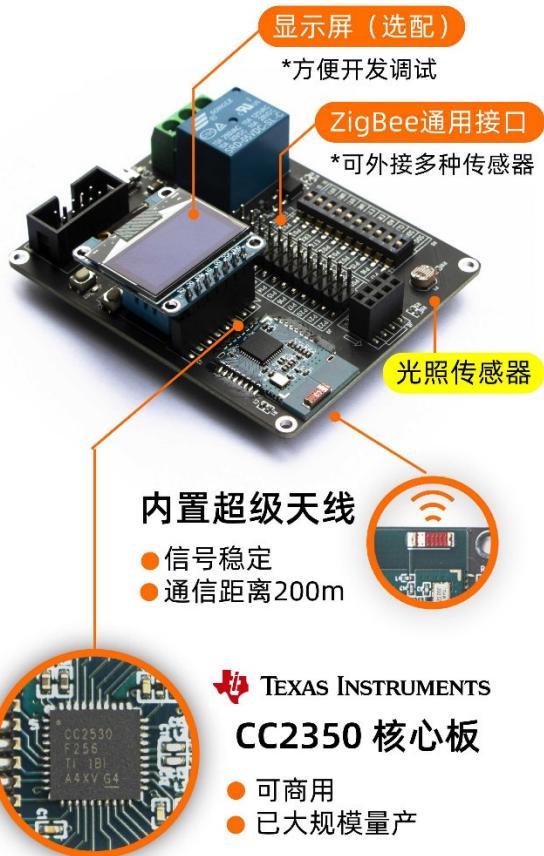
ZigBee 3.0 已被各大公司采用
别再学过时协议啦



善学坊™

物联网开发套件系列 ... IoT development ... 架构设计可商用

不止用于技术学习，
更适用于产品原型开发



TEXAS INSTRUMENTS

CC2350 核心板

- 可商用
- 已大规模量产

ZigBee 3.0 课程

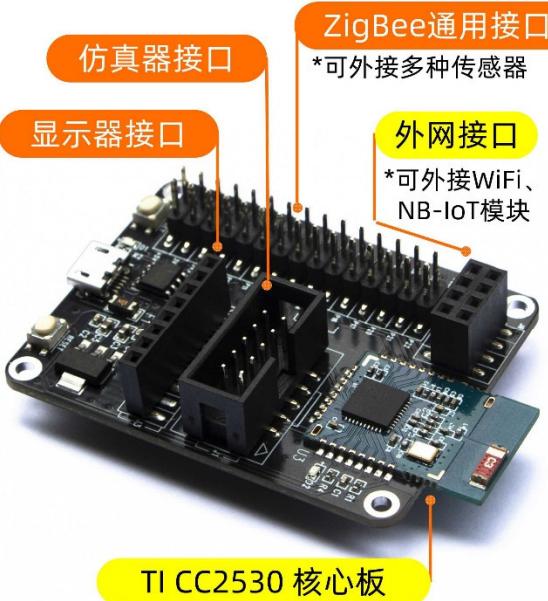
- 小米资深工程师 编著
- 小白也能学会 2021.10更新
- 免费听课：www.sxf-iot.com
- 教材已入选：



清华大学出版社

TSINGHUA UNIVERSITY PRESS

全新推出MINI板，低至 **59** 元/板



TI CC2530 核心板

- 可商用
- 已大规模量产

边长：61.5x41.5mm

- MINI板更适用作**网关**，标准板更适用作**终端**
- 超高性价比，满足开发需求！

第一部分：准备

- 1.1 小白也能读懂的 ZigBee 3.0 简介
- 1.2 IAR EW for 8051 简介与安装
- 1.3 SmartRF Flash Programmer 下载与安装
- 1.4 TI Z-Stack 3.0 简介与安装
- 1.5 串口助手简介与安装
- 1.6 必要的驱动安装
- 1.7 其他软件安装（非必须）

第二部分：51单片机入门——基于CC2530

- 第1章：搭建开发环境
- 第2章：GPIO实验
- 第3章：定时器实验
- 第4章：通信实验
- 第5章：ADC实验
- 第6章：显示器实验
- 第7章：外设实验

第三部分：Z-Stack 3.0 入门

- 第1章：《Z-Stack 3.0 入门》学习指导
- 第2章：操作系统的调度原理
- 第3章：Z-Stack架构说明
- 第4章：系统抽象层详解
- 第5章：硬件适配层应用——LED
- 第6章：硬件适配层应用——按键
- 第7章：硬件适配层应用——串口
- 第8章：硬件适配层应用——显示屏
- 第9章：硬件适配层应用——ADC
- 第10章：系统OSAL内容补充
- 第11章：NVI应用
- 第12章：PA应用（选修）
- 第13章：外部FLASH的应用（选修）

第四部分：ZigBee 3.0 网络编程

- 第1章：《ZigBee 3.0 网络编程》学习指导
- 第2章：ZigBee3.0 基础概念
- 第3章：ZigBee3.0 BDB相关内容
- 第4章：应用端点AF通信（广播、组播、P2P）
- 第5章：ZCL基础概念

- 开发工具：IAR
- 语 言：C语言

- ▶ 第6章：基于ZCL通信——开关控制
- ▶ 第7章：基于ZCL属性读写
- ▶ 第8章 基于ZCL属性上报
- ▶ 第9章：ZHA—Lighting
- ▶ 第10章：ZHA—TemperatureSensor

课外篇：项目实战

- 基于ZigBee的农业环境信息采集
- 基于ZigBee的文件传输系统

传感器探测与报警 系列案例

- 基于ZigBee的温度和有害气体短信报警系统
- 基于ZigBee的多传感器探测与亮灯报警系统
- 基于ZigBee的温湿度、人体红外与声光报警
- 基于ZigBee的温湿度 & 信号强度探测系统
- 基于ZigBee的温湿度 & 光照强度采集系统

课外篇：进阶选修

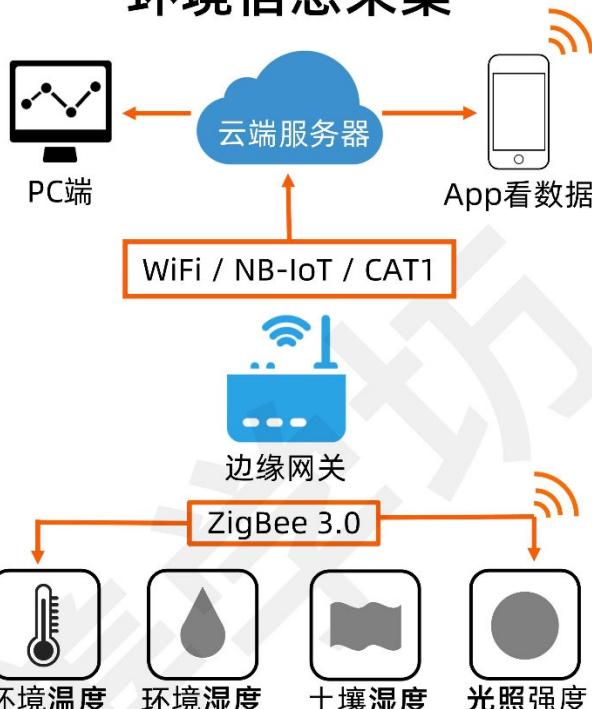
《课外篇：进阶选修》的说明

- ▶ 第1章：串口通信协议设计
- ▶ 第2章：优化协调器工程结构
- ▶ 第3章：协调器上位机调试
- ▶ 第4章：信道及PanId的动态修改
- ▶ 第5章：网络短地址及MAC地址的获取
- ▶ 第6章：入网控制及白名单
- ▶ 第7章：协调器分区存储管理
- ▶ 第8章：ZigBee2WiFi——基于ESP8266

版权声明与免责声明



环境信息采集



了解更多开发案例可联系客服

物联网开发套件系列
· · · IoT development · · ·

开放上位机源码



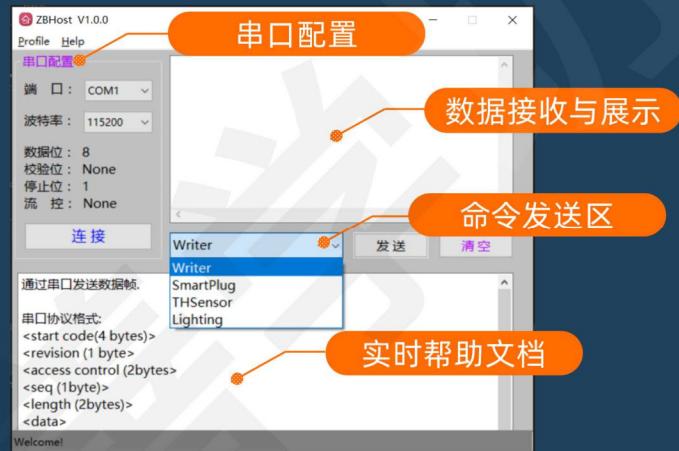
支持接入第三方云和App

阿里云 · IoT

Home Assistant

OneNET

App上看数据



* 开发套件的学习语言为C, Python 仅是上位机的开发语言

* 上位机可直接使用, 不会Python没关系



3.2. Hardware composition & technical parameters

Note: The following content contains professional knowledge. **Beginners must skip it** and start reading from "Part 1: Preparation" in the catalog.

1. ZigBee standard board

1. Main hardware components

- Zigbee wireless MCU: TI CC2530F256
- Ambient temperature and humidity two-in-one sensor: original authentic Osun DHT11
- Relay: Original Songle relay SRD-05VDC-SL
- Light sensor: SXF-GL5516
- External memory: M25PE80 1 MB NorFlash
- USB to serial port chip: Qinheng CH340
- Dip switch: original KE dip switch

2. Main technical parameters

- Model: zigbee-std-v3
- ZigBee application protocol version: ZigBee 3.0
- ZigBee core protocol version: ZigBee 2007 Pro
- ZigBee underlying protocol: IEEE 802.15.4
- ZigBee wireless communication distance: The communication distance between adjacent nodes is about 150 meters
- ZigBee wireless communication rate: 240kb/s (theoretical value)
- ZigBee electromagnetic wave frequency: 2.4GHz
- ZigBee wireless communication delay: < 1 second
- TI CC2530F256 Technical Parameters: <https://www.ti.com.cn/product/cn/CC2530>
- Aosong DHT11 technical parameters: <http://www-aosong.com/products-21.html>
- Songle relay SRD-05VDC-SL technical parameters: http://www.songle.com/Product_show_id_546.htm
- M25PE80-VMN6TP 1 MB NorFlash Specifications: <https://gitee.com/study-j/zigbee/tree/master/课外资料参考/M25PE80>

数据手册

- Qinheng CH340 Technical Parameters: <http://www.wch.cn/product/CH340.html>
- Light sensor GL5516 Technical parameters:

产品型号	5516
限大电压 (vdc)	150
限大功耗	100
环境温度	-30~+70
光谱峰值	540
亮电阻	5-10
暗电阻 (兆欧)	0.5
响应时间 (ms)	上升: 20 下降: 30

- Power supply method: Micro USB interface power supply, supports computer USB interface, mobile phone charger, official dry battery box, etc.
- Supply voltage: 4.5~5.5v
- Supply current: 0.5~2A

2. ZigBee Mini Board

1. Main hardware components

- Zigbee wireless MCU: TI CC2530F256
- USB to serial port chip: Qinheng CH340

2. Main technical parameters

- Model: zigbee-mini-v3

- ZigBee application protocol version: ZigBee 3.0
- ZigBee core protocol version: ZigBee 2007 Pro
- ZigBee underlying protocol: IEEE 802.15.4
- ZigBee wireless communication distance: The communication distance between adjacent nodes is about 150 meters
- ZigBee wireless communication rate: 240kb/s (theoretical value)
- ZigBee electromagnetic wave frequency: 2.4GHz
- ZigBee wireless communication delay: <= 1 second
- TI CC2530F256 Technical Parameters: <https://www.ti.com.cn/product/cn/CC2530>
- Qinheng CH340 Technical Parameters: <http://www.wch.cn/product/CH340.html>
- Power supply method: Micro USB interface power supply, supports computer USB interface, mobile phone charger, official dry battery box and most power banks.
- Supply voltage: 4.5~5.5v
- Supply current: 0.5~2A

3. Display

- Model: 0.96OLED12864
- Size: 0.96 inches
- Pixels: 128×64
- Interface: SPI
- Pins: GND, VCC, D0, D1, RES, DC, and CS

Detailed instructions: 0.96" OLED 128x64 Specifications

<https://zhuanlan.zhihu.com/p/628861772>

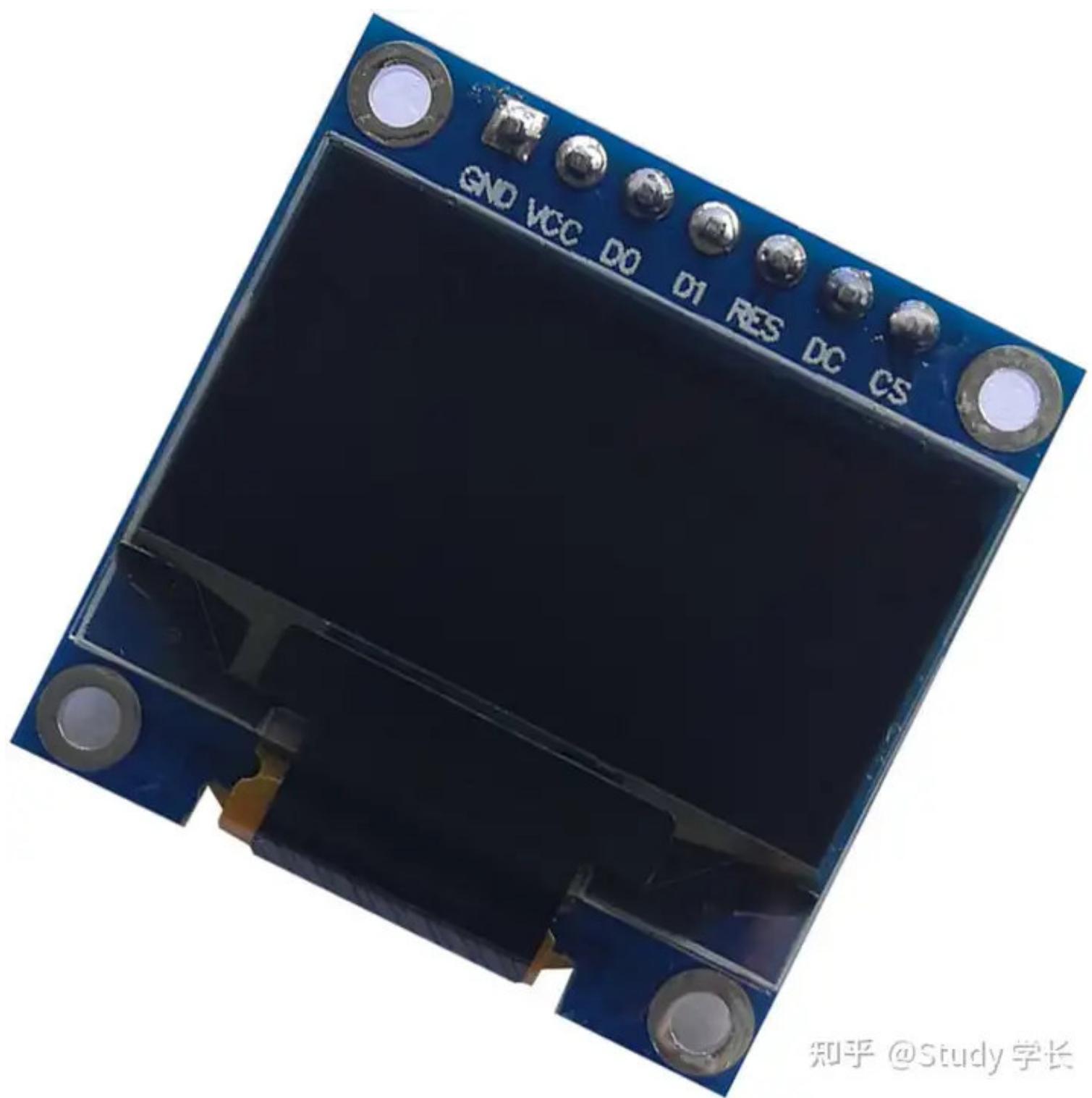
Table of contents

1. Introduction
2. Technical parameters
3. Pin Description
4. Circuit Schematic Diagram
5. Instructions for use

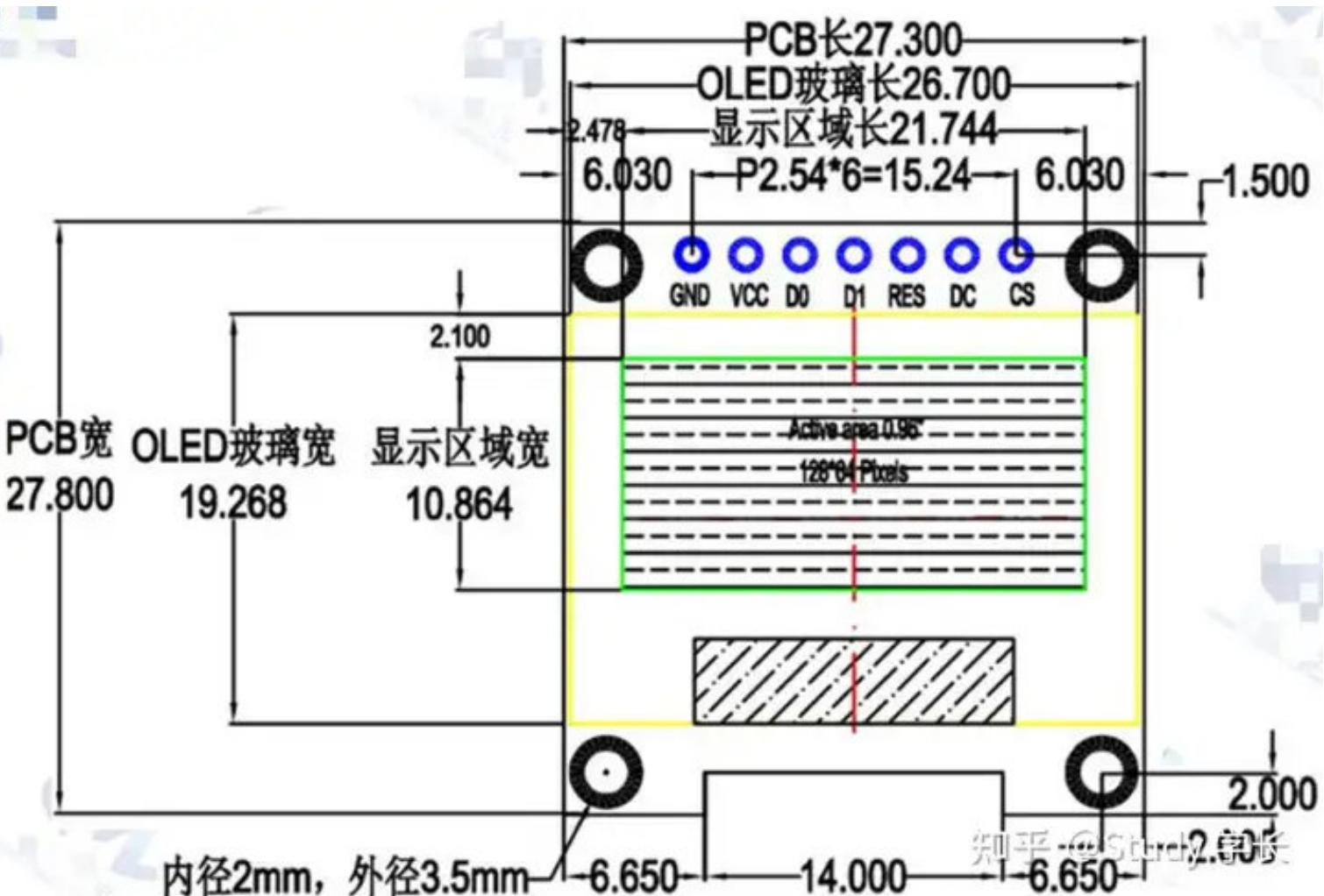
1. Introduction

(1) Sample store: See the comments section.

(2) The actual picture of this 0.96-inch OLED display is shown in the figure below. Its resolution is 64×128 pixels, which means there are 64 pixels vertically and 128 pixels horizontally.



知乎 @Study 学长



2. Technical parameters

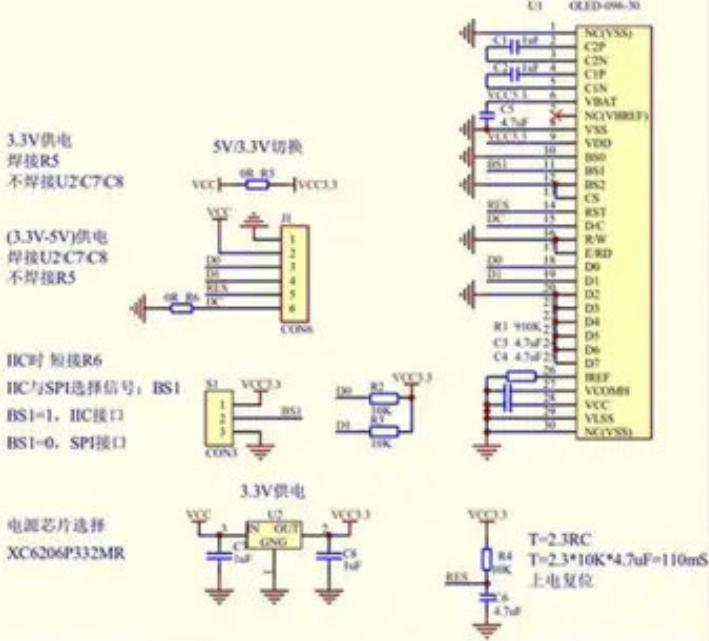
- (1) Model: 0.96OLED12864
- (2) Color: black and white
- (3) Size: 0.96 inches
- (4) Resolution: 128×64px
- (5) Driver chip: SSD1306
- (6) Dimensions: 27.3×27.8×2.7 (mm)
- (7) Display area: 21.74×11.20 (mm)
- (8) Spot size: 0.15×0.15 (mm)
- (9) Point spacing: 0.17×0.17 (mm)
- (10) Working voltage: 3.3v or 5v
- (11) Working current: about 28ma when working, about 25ua when sleeping
- (12) Operating temperature: -20~60°C
- (13) Storage temperature: -20~60°C
- (14) Number of pins: 7

3. Pin Description

- (1) GND: negative power supply interface
- (2) VCC: positive power supply interface, supports 3.3~5.0v power supply
- (3) D0: Clock interface
- (4) D1: Data interface
- (5) RES: reset interface
- (6) DC: Data/command selection interface
- (7) CS: Chip Select

4. Circuit Schematic Diagram

0.96寸OLED液晶模块



电源电路说明

- #### 1. U2焊接后支持3.3V–5V宽范围供电

- ## 2. R5 只有3.3V供电时短接

- ### 3. 切记VCC和GND不要接反

复位电路说明

- #### 1. R4\ C6未焊接, MCU通过IO口控制RES引脚

通信接口说明

SPI接口

1. S1接口短接到SPI(接地)
 2. D0=SCLK
 3. D1=SDIN

IIC接口

1. S1接口短接到IIC(接VCC3.3)
 2. R6接口短接(接地)
 3. D0=SCL
 4. D1=SDA

知乎 @Study 学长

5. Instructions for use

Refer to "STM32 Development Guide" → "OLED Display Experiment"

Link 1: <https://z7po9bxpe4.k.tothink.com/@stm32/11.OLEDxianshiqishiyan.html>

[Link 2: https://z7po9bxpe4.k.toptthink.com/@stm32](https://z7po9bxpe4.k.toptthink.com/@stm32) (Find "OLED Display Experiment" in the catalog)

4. Simulation Downloader

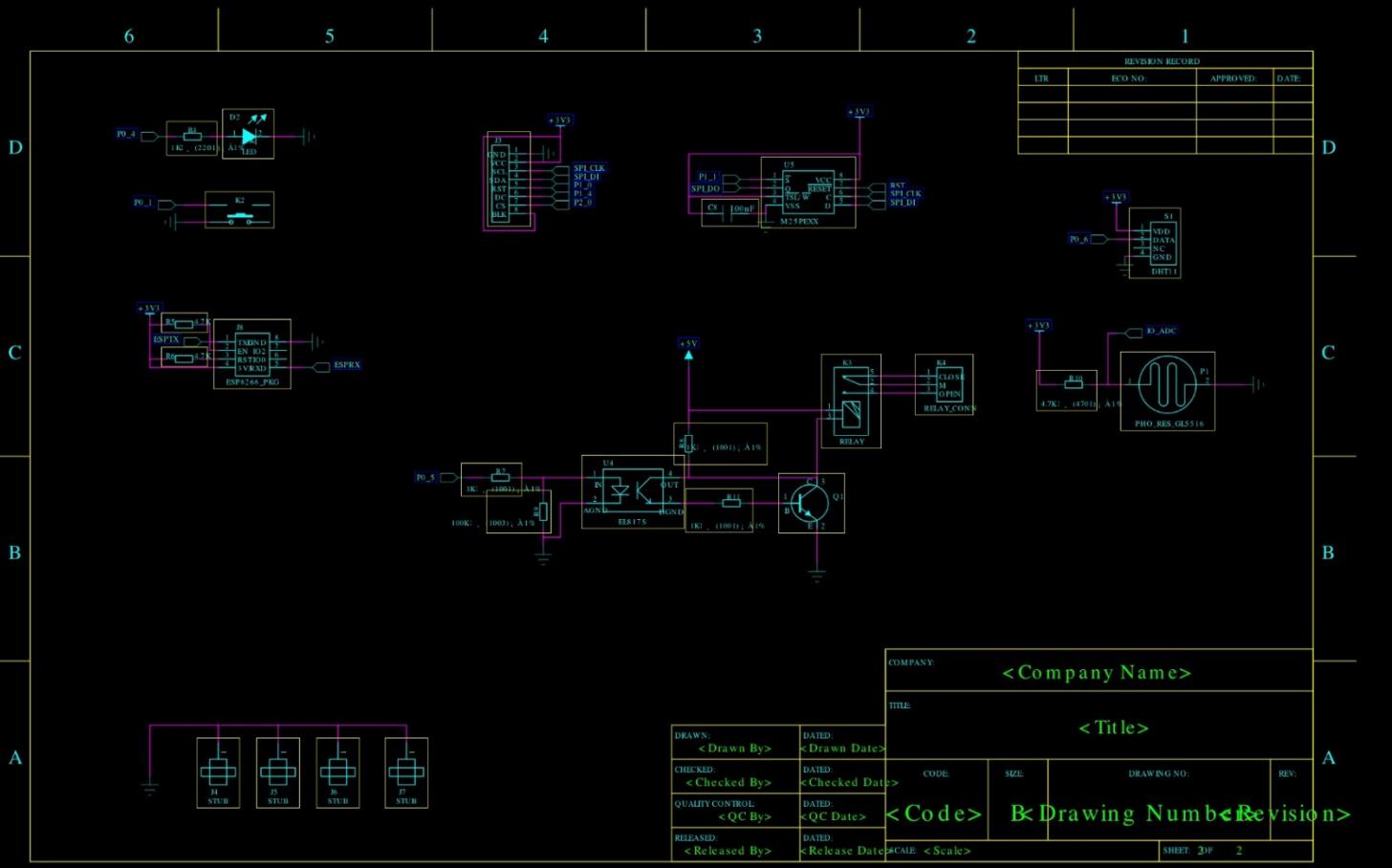
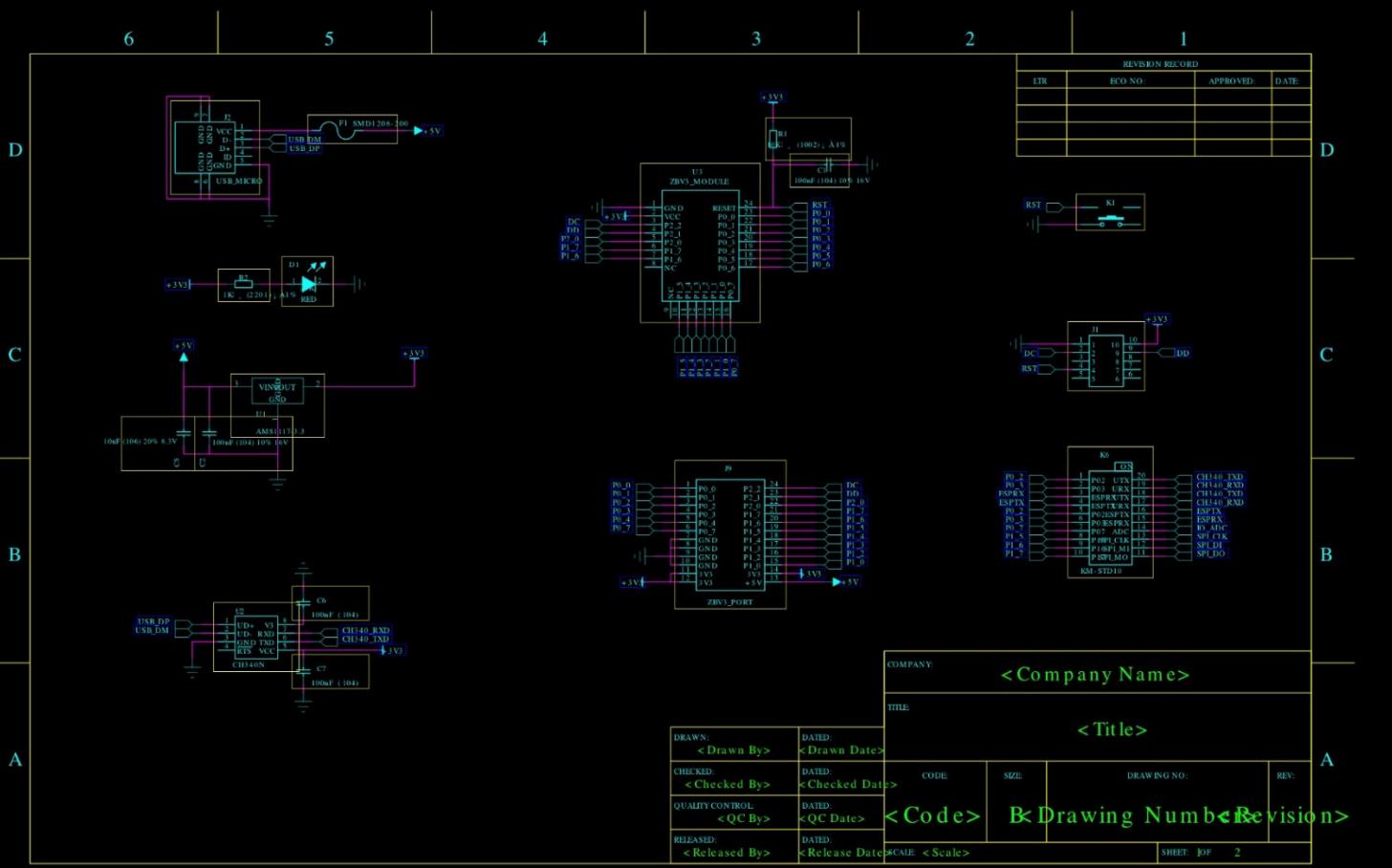
- Model: smart-rf04eb
 - Whether to support online simulation: support
 - Whether to support program download: support

3.3. Circuit Schematic & PCB Diagram - Standard Board

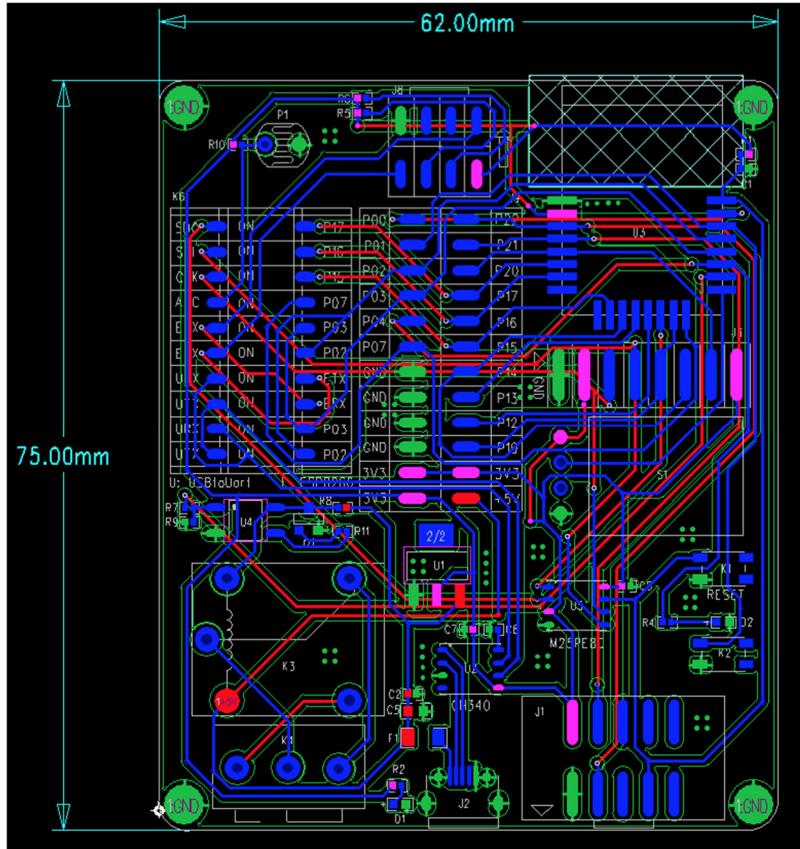
Note: The following content contains professional knowledge. **Beginners must skip it** and start reading from "Part 1: Preparation" in the catalog.

Circuit Schematic

1. The schematic diagram supports zooming in, [click the image] or directly [zoom in the webpage]
 2. Provide a clearer schematic diagram PDF download: <https://z7po9bxpe4.k.tothink.com/@zigbee-dev-guide/yuandaimaxiazi.html>



PCB Design

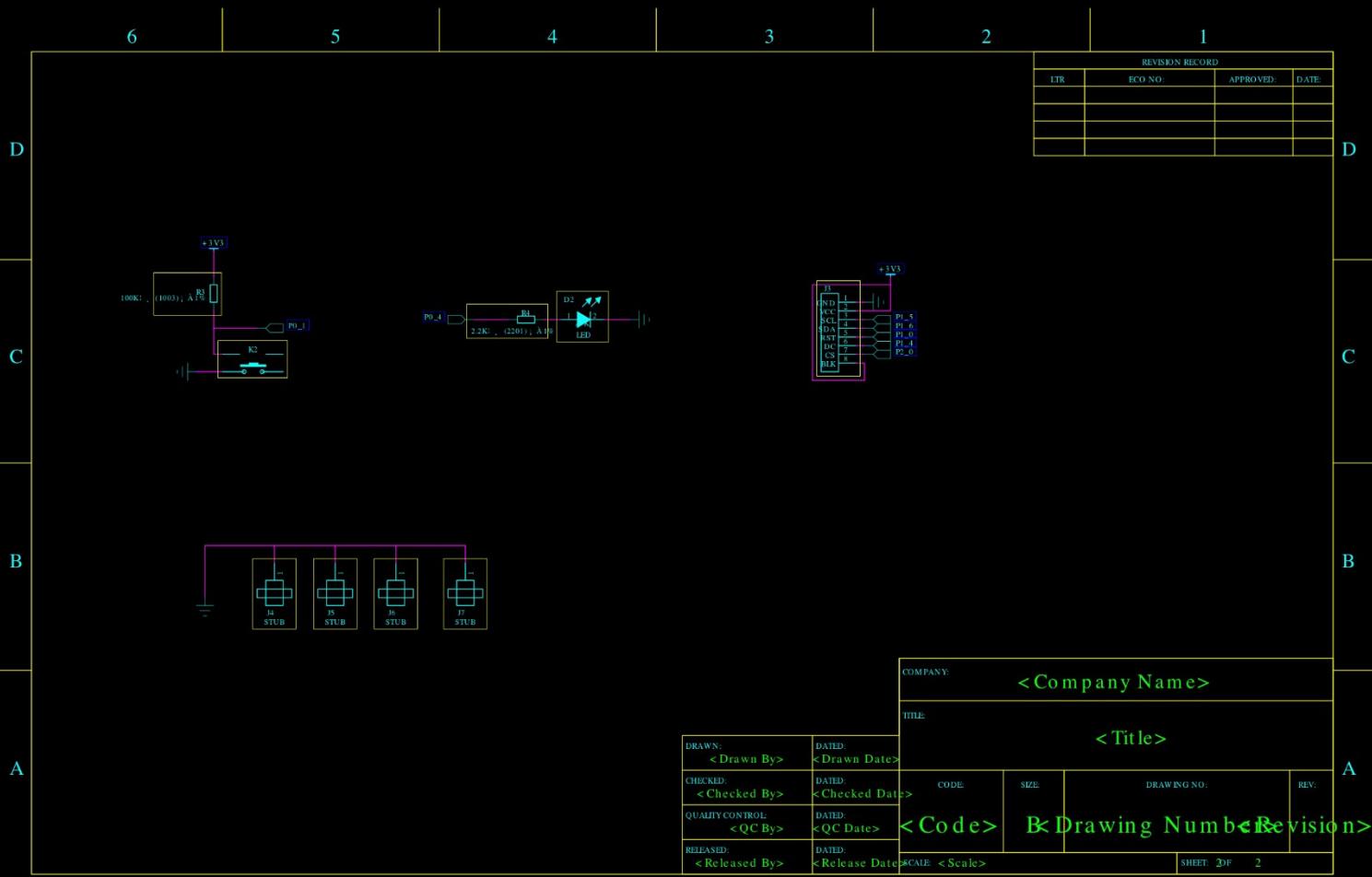
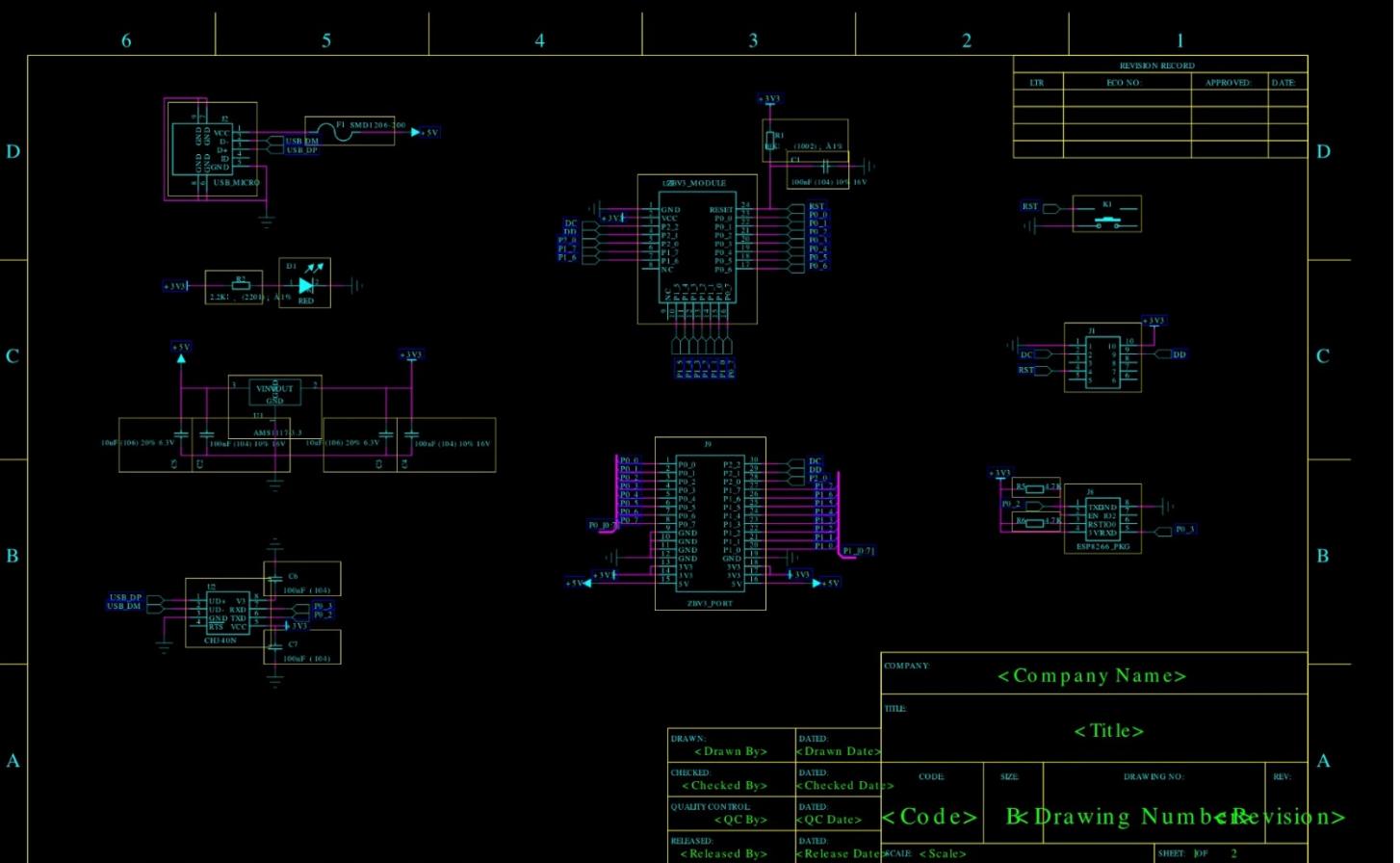


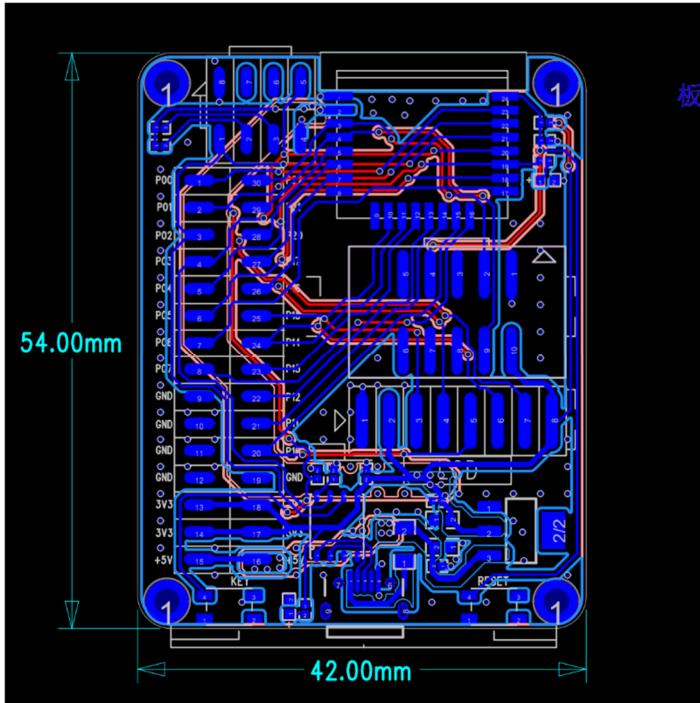
3.4. Circuit Schematic & PCB Diagram - MiNi Board

Note: The following content contains professional knowledge. **Beginners must skip it** and start reading from "Part 1: Preparation" in the catalog.

Circuit Schematic

1. The schematic diagram supports zooming in, [click the image] or directly [zoom in the webpage]
 2. Provide a clearer schematic diagram PDF download: <https://z7po9bxpe4.k.tothink.com/@zigbee-dev-guide/yuandaimaxiaza.html>





3.5. CC2530F256 module specifications

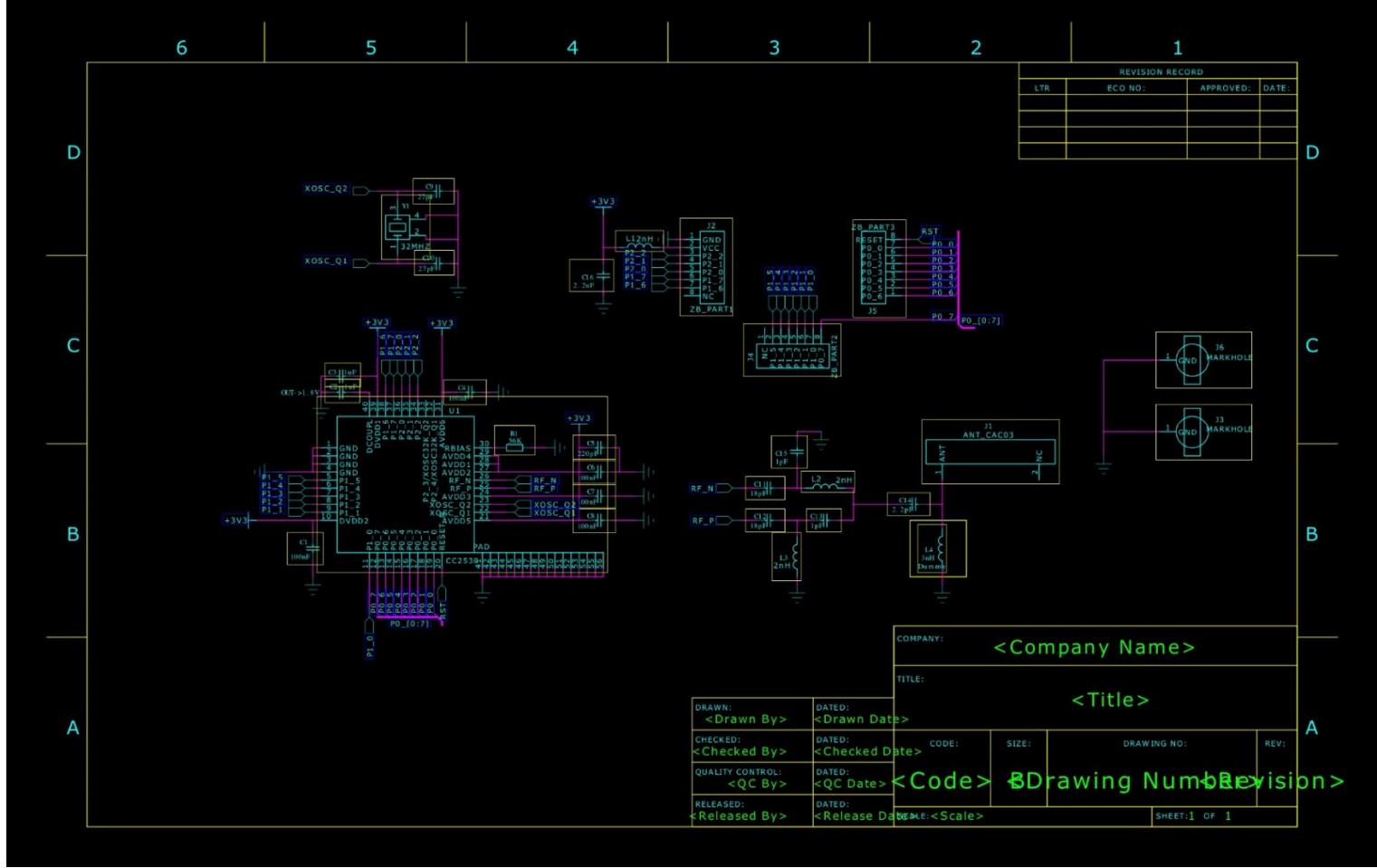
Note: The following content contains professional knowledge. **Beginners must skip it** and start reading from "Part 1: Preparation" in the catalog.

Specifications

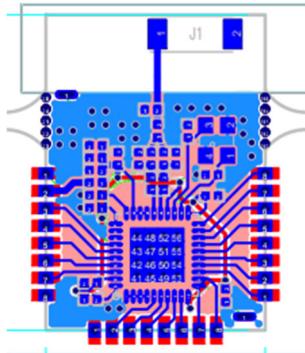
<https://z7po9bxpe4.k.tophink.com/@zb-module>

Circuit Schematic

1. The schematic diagram supports zooming in, [click the image](#) or directly [zoom in the webpage](#)
2. Provide a clearer schematic diagram PDF download: <https://z7po9bxpe4.k.tophink.com/@zigbee-dev-guide/yuandaimaxiazai.html>



PCB Design



3.6. WiFi module specifications

Note: The following content contains professional knowledge. **Beginners must skip it** and start reading from "Part 1: Preparation" in the catalog.

Module Description

This tutorial mainly talks about ZigBee, not WiFi. In addition, this WiFi module is provided by a third party to our company, and its supporting development materials may not be detailed enough, so please understand!

Downloads

The download address of ESP8266 module reference materials is as follows:

<https://gitee.com/study-j/wifi-esp8266>

StudyJ / WiFi开发指南—基于ESP8266

你当前开源项目尚未选择许可证 (LICENSE)，点此选择并创建开源许可

master 分支 2 标签 0 + Pull Request + Issue 文件 Web IDE 克隆/下载

StudyJ update README.md. 08bf7dc 6个月前
71413-ESP-01S-Relay-v4.0-master-1... add 5次提交 7个月前
ESP8266-01S add 7个月前
ESP8266模块官方指导文件 add 7个月前
FLASH_DOWNLOAD_TOOLS_V3.6.4 add 7个月前
ESP8266新手入门调试指南(补全).pdf add 7个月前
README.md update README.md. 6个月前

README.md

WiFi开发指南——基于ESP8266

简介

本仓库内容包含ESP8266 WiFi模块的相关学习资料

善学坊 唯一官网: <http://www.sxf-iot.com>
善学坊 唯一微信公众号: shanxuefang-iot

Module Introduction



Note: Since the supporting ESP8266 module is a development-free module, it is generally not necessary to re-burn the program, so the default **USB to serial port module** does not support burning the program.



可拆卸设计
方便灵活

支持USB直连PC
开发调试



物联网开发套件系列
... IoT development ...

可靠的硬件设计

原装正品ESP8266



性能稳定



高度集成

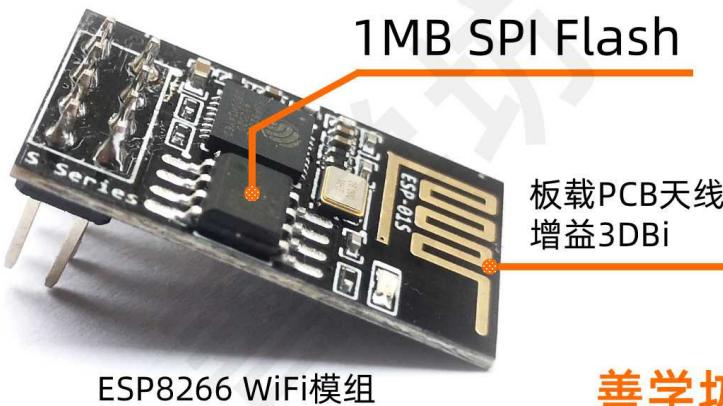


低功耗



32位处理器

善学坊™ 善学坊™



ESP8266 WiFi模组

善学坊™ 善学坊™

稳定的CH340



1000uF固态电容
保证电流供应

ESP8266 USB调试器

3.7. Communication distance test

Note: The following content contains professional knowledge. **Beginners must skip it** and start reading from "Part 1: Preparation" in the catalog.

Hint:

- The maximum communication distance is generally affected by two factors:
 - (1) module hardware design;
 - (2) the code written by the developer.
- The main purpose of this test is to verify that the module hardware itself supports long-distance communication. If readers find that the distance is short in actual projects, they should troubleshoot the problem from their own code.

Communication distance description

- Recommended communication distance: 1 ~ 50m
- Maximum communication distance: 150m +



Test Hardware

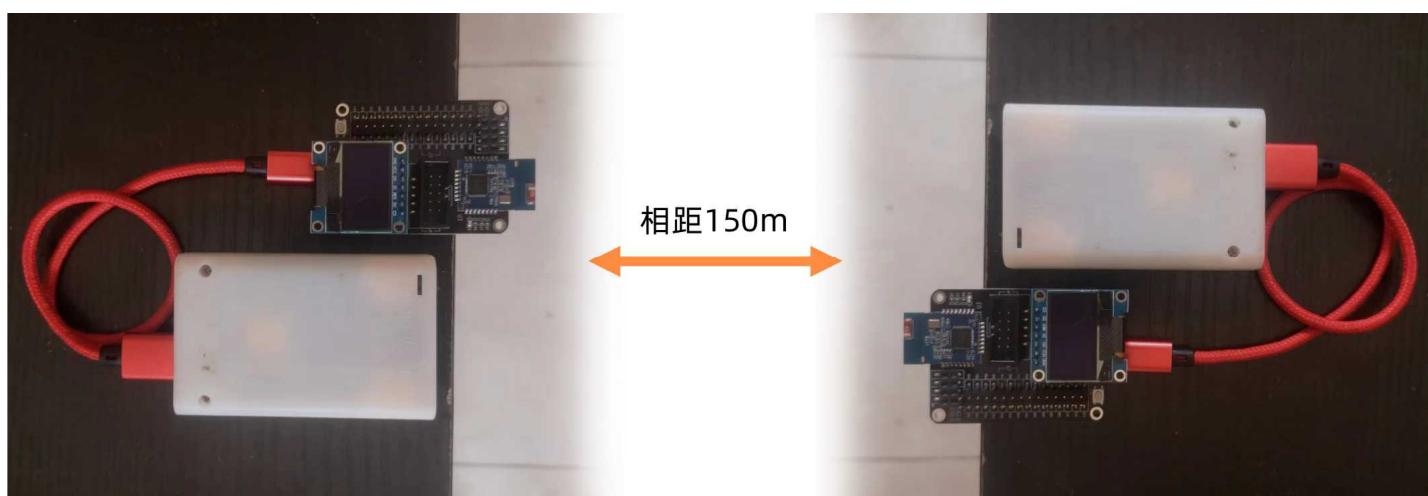
- ZigBee Mini板 × 2 : <https://item.taobao.com/item.htm?id=683089996879>
- 0.96寸 OLED × 2 : <https://item.taobao.com/item.htm?id=683089996879>
- 4.5v dry battery box × 2: <https://item.taobao.com/item.htm?id=683115420758>
- Nanfu No. 5 dry cell battery × 6

Note: Please use two ZigBee Mini boards first! If you use a ZigBee standard board + Mini board, please note:

- (1) The communication distance will be slightly shorter;
- (2) The DIP switch needs to be configured specifically to use the OLED screen. For detailed instructions, please refer to "Part 2" → "Chapter 6: OLED Display Experiment" in the catalog of this tutorial.

Test environment

- Open space, with no obstruction between the two development boards;
- The power supply voltage of the development board is 4.5~5.5v;
- The development board must not be placed on the ground and must be at least 1 meter above the ground;
- The modules must be oriented opposite each other, as shown in the diagram.



Test firmware download

- Link: <https://pan.baidu.com/s/1dLwd8ZHn5hUSgeaNGoBTDQ?pwd=2333>

- Password: 2333



Test steps

(1) Burn the "Distance Test Firmware A (Coordinator)" to the first Mini board.

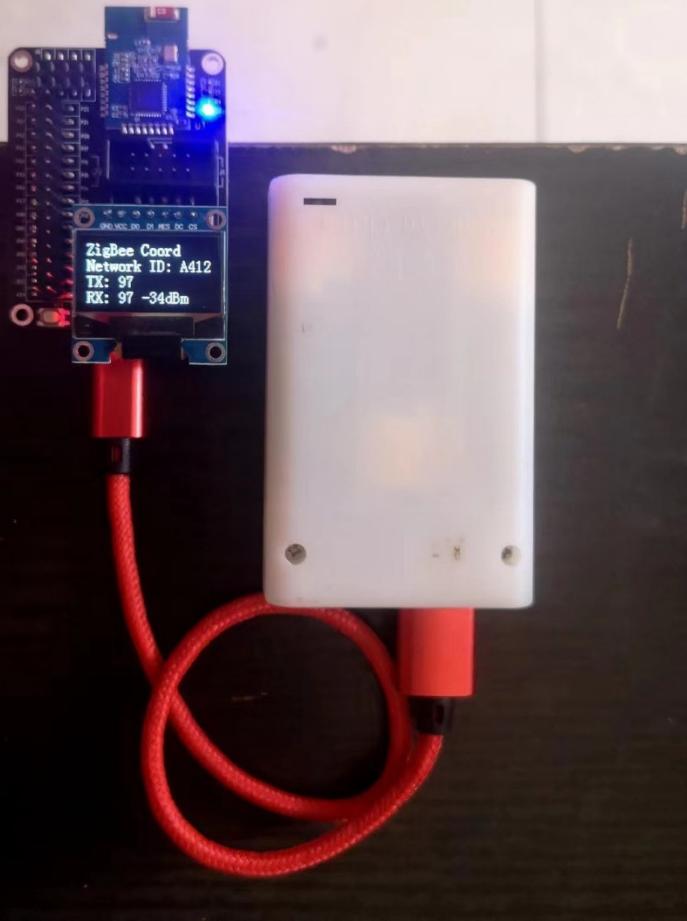
Tip: For the firmware burning method, please refer to "Part 2" → "Chapter 1: CC2530 Development Basics" → "Firmware Burning" in the catalog of this tutorial.

(2) Burn the "Distance Test Firmware B (Router)" to the second Mini board.

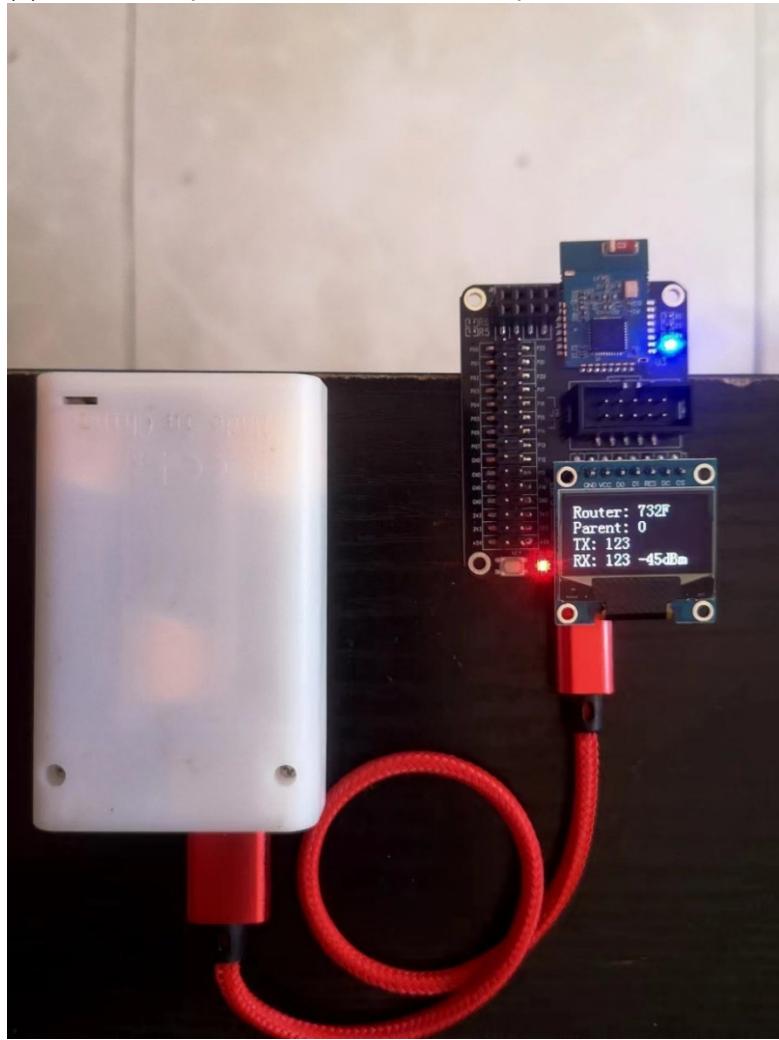
(3) Connect the hardware as shown in the figure.



(4) Turn on the power of the first development board, as shown in the figure.



(5) Turn on the power of the second development board, as shown in the figure.



(6) After turning on the power, the development boards will send data packets to each other at a frequency of 1 time/s;

(7) The screen displays the following:

- TX: the amount of data sent from the current development board to another development board;
- RX: the amount of data received from another development board;
- -XXdBm: signal strength. dBm is a unit of signal strength. Generally, the larger the dBm value, the better the signal. For example, -55dBm is better than -88dBm.

(8) When the blue LED on the development board flashes once, it means that a data packet has been received from another development board. Users can judge the current communication quality by the flashing frequency of this LED.

4. Part 1: Preparation

4.1. Introduction to ZigBee 3.0 that even a novice can understand

4.2. IAR EW for 8051 Introduction and Installation

4.3. TI Z-Stack 3.0 Introduction and Installation

4.4. SmartRF Flash Programmer Download and Installation

4.5. Serial Port Assistant Introduction and Installation

4.6. SmartRF04EB Driver

4.7. USB to Serial Port Driver

4.8. Other software installation (optional)

4.1. Introduction to ZigBee 3.0 that even a novice can understand

Technical support instructions

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers **will** answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Click <https://www.bilibili.com/video/BV1k34y1D7Vz/> to go to Bilibili to watch the video more clearly!

4.2. IAR EW for 8051 Introduction and Installation

Technical support instructions

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers **will** answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Software Download

- (1) Download address: See "Supporting Resource Download" in the catalog of this tutorial.
(2) After downloading the supporting resources, the location of the installation package is shown in the figure.

文件名	大小
CH34x USB转串口驱动.zip	229.23 KB
Flash-programmer-1.12.8.zip	18.24 MB
IAR EW for 8051 10.10.1.zip	327.78 MB
PuTTY.zip	2.36 MB
SmartRF04EB Driver.zip	1.97 MB
TI Z-Stack 3.0.2 (CSDN视频课程专用) .exe	22.64 MB
TI Z-Stack3.0.1.zip	23.34 MB
Ubiqua Protocol Analyzer.rar	3.95 MB
善学坊官方学习指南.html	212.00 B
图片取模软件.zip	512.11 KB
友善串口调试助手.zip	20.26 MB
字模软件2.2.zip	256.59 KB



IAR EW for 8051 Introduction and Installation

<https://zhuanlan.zhihu.com/p/535315364>

Table of contents

1. Introduction to IAR
2. Software Download
3. Installation steps
4. Software Registration

知乎

四、软件注册

本部分内容仅供参考，作者不为有需要的读者自行操作

请参考：zhuanlan.zhihu.com/p/535315364

IAR EW for 8051 是-
请务必支持正版软件！
30天免费使用版本。

编辑于 2022-06-29 12:23

正在从 IAR EW for 8051 10.10.1.zip 正在解压... IAR8051-10101-Autorun.exe 48% 已用时间：00:00:01 剩余时间：00:00:01 进度：45% 后台(B) 暂停(P) 取消 模式(M)... 帮助

22 个项目 选中 1 个项目 327 MB

51 单片机

▲ 赞同 ● 添加评论 分享 喜欢

还没有评论

写下你的评论...

22:09 2022/7/4

1. Introduction to IAR

IAR is a world-leading supplier of embedded system development tools and services. One of its most famous products is the C compiler IAR Embedded Workbench. IAR Embedded Workbench supports microprocessors from many well-known semiconductor companies. The currently supported microprocessors are shown in the figure (picture from IAR official website):



Since the ZigBee chip TI CC2530 is based on 8051, we choose IAR EW for 8051 as the development tool.

2. Software Download

(1) Course users can find a compressed file named "IAR EW for 8051 10.10.1.zip" in the supporting materials.

<input type="checkbox"/> 文件名		<input type="checkbox"/> 大小
<input type="checkbox"/> IAR EW for 8051 10.10.1.zip		327.78 MB
<input type="checkbox"/> Pads9.5.zip		1.55 GB
<input type="checkbox"/> PuTTY.zip		2.36 MB
<input type="checkbox"/> SourceInsight4.0.zip		21.19 MB
<input type="checkbox"/> SublimeText.zip		17.47 MB
<input type="checkbox"/> TI Z-Stack 3.0.2 (CSDN视频课程专用) .exe		22.64 MB
<input type="checkbox"/> TI Z-Stack3.0.1.zip		23.34 MB
<input type="checkbox"/> Ubiqua Protocol Analyzer.rar		3.95 MB
<input type="checkbox"/> Xshell-7.0.0041r_beta.exe	  	44.23 MB
<input type="checkbox"/> ZigBee仿真器驱动程序.zip		知乎@Study 学长 52.75 KB

You can also download it from the official website <https://www.iar.com/products/architectures/iar-embedded-workbench-for-8051/>

3. Installation steps

(1) After downloading the installation package, right-click the IAR installation package and select "Run as Administrator". After running, click Install IAR Embedded Workbench® for 8051, as shown in the figure.



IAR Embedded
Workbench

IAR
SYSTEMS

- Install IAR Embedded Workbench® for 8051
- Installation and licensing information
- Release notes
- Explore the installation media
- Exit

知乎 @Study 学长

(2) In the pop-up dialog box, click the Next button, as shown in the figure.



IAR Embedded
Workbench

IAR Embedded Workbench for 8051 10.10.1

IAR
EWCT
x EMS

- In
- In
- R
- E
- E

Welcome to the InstallShield Wizard for IAR Embedded Workbench for 8051

The InstallShield Wizard will install IAR Embedded Workbench for 8051 on your computer. To continue, click Next.

< Back

Next >

Cancel

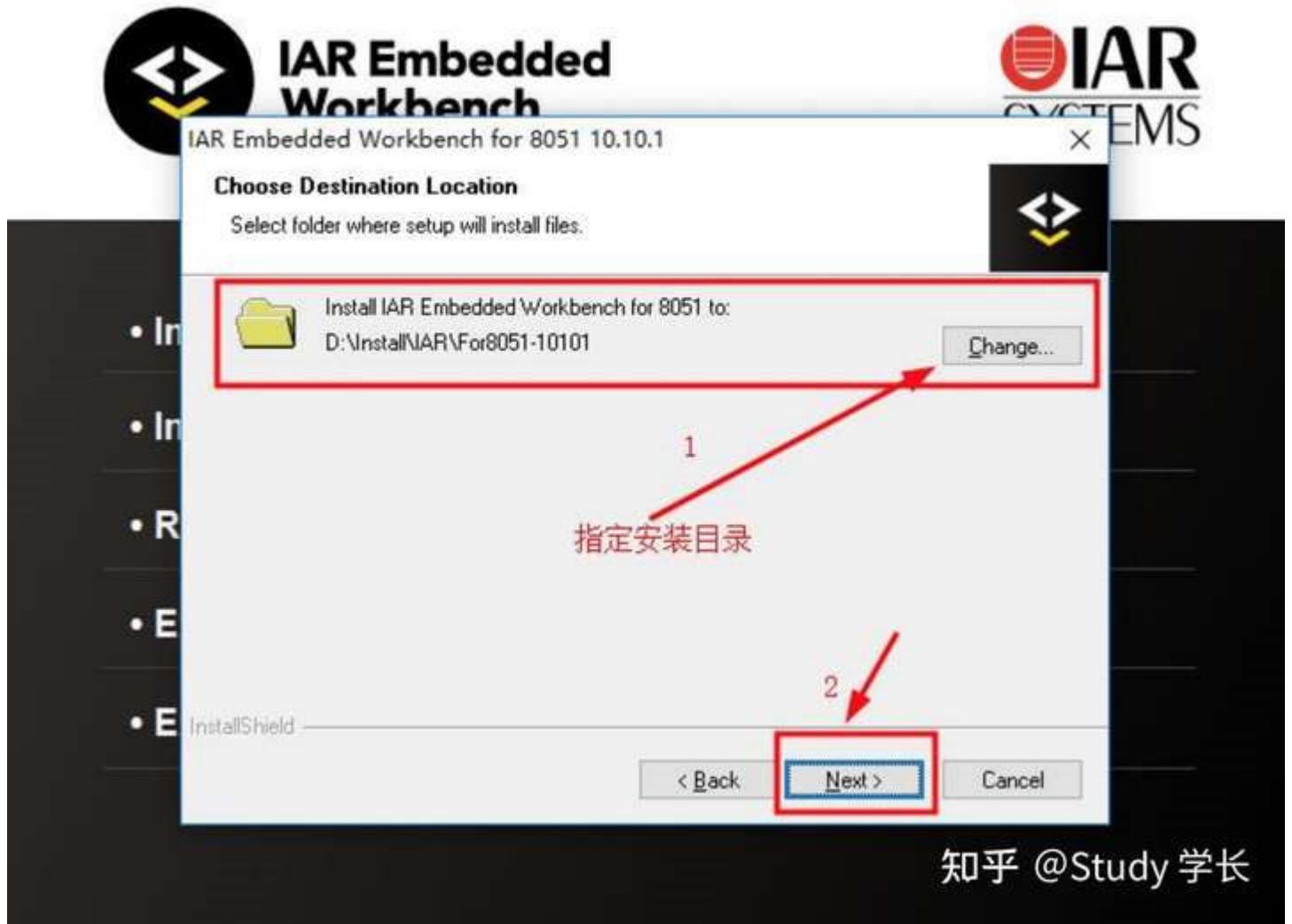
知乎 @Study 学长

(3) After reading and agreeing to the license agreement, select I accept the terms of the license agreement, and then click the Next button, as shown in the figure.



知乎 @Study 学长

(4) Click the Change button to select the software installation directory, and then click the Next button, as shown in the figure.



知乎 @Study 学长

(5) Select Complete to install all components, and then click the Next button, as shown in the figure.

IAR Embedded Workbench

IAR Embedded Workbench for 8051 10.10.1



Setup Type

Select the setup type to install.

Please select a setup type.

Complete



All program features will be installed. (Requires the most disk space.)

Custom



Select which program features you want installed. Recommended for advanced users.

- 1
- 2

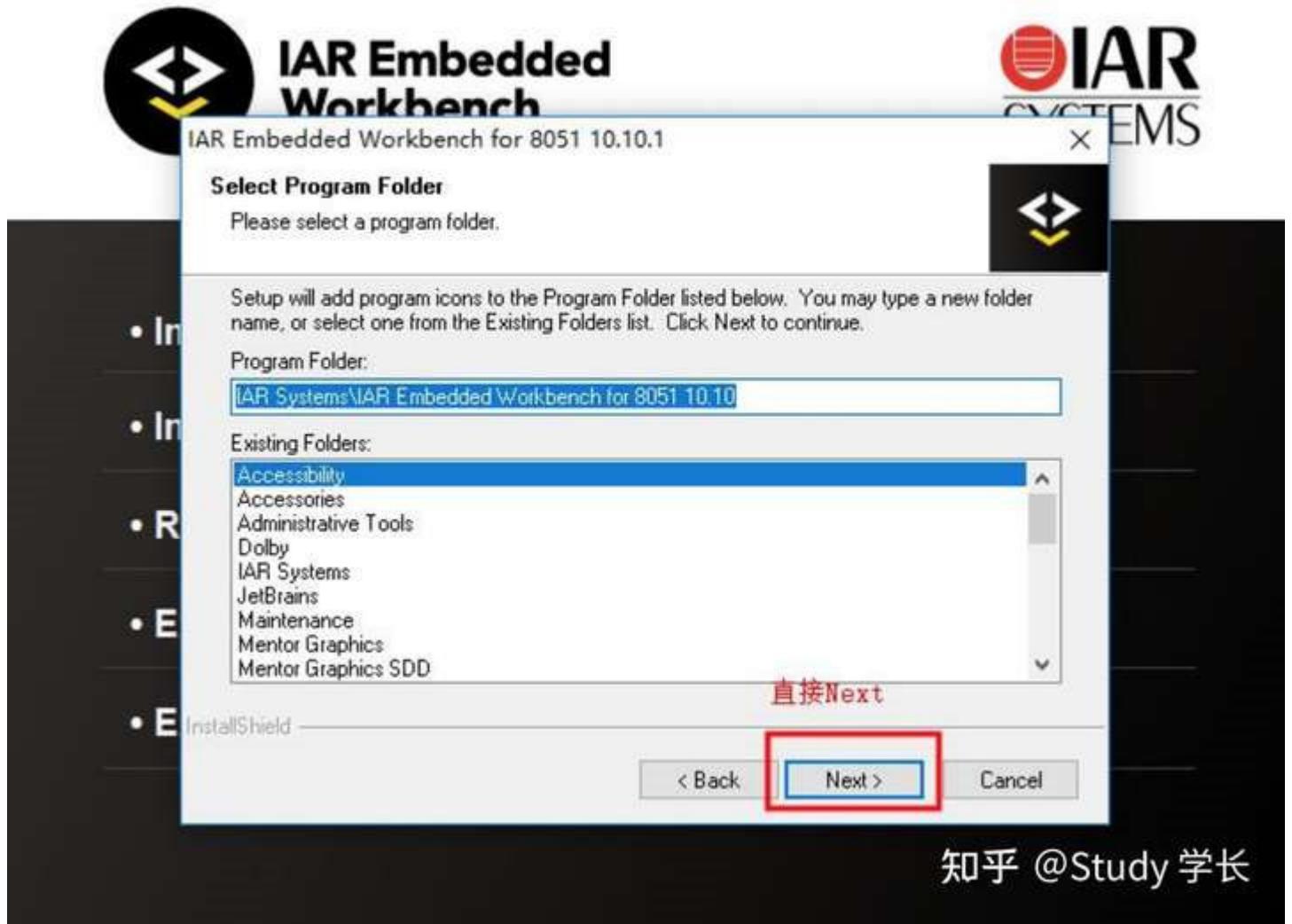
< Back

Next >

Cancel

知乎 @Study 学长

(6) Keep the default location of the program folder and click the Next button directly, as shown in the figure.



(7) Click the Install button, as shown in the figure.

知乎 @Study 学长



(8) Wait for the installation to complete, as shown in the figure.

知乎 @Study 学长



知乎 @Study 学长

(9) IAR will pop up the following dialog box asking whether to install the Dongle driver. Since it is not used for the time being, click the "No" button, as shown in the figure.



IAR Embedded Workbench

IAR Embedded Workbench for 8051 10.10.1

IAR
CYCLES
EMS

Setup Status

IAR Embedded Workbench for 8051 - InstallShield Wizard

• In

• In

• R

• E

• E



IAR Systems Setup is about to install dongle drivers on your system.

The driver installation for USB dongles might fail if a USB dongle is attached to your computer during the installation!

To continue with the dongle driver installation, remove all USB dongles and click Yes. To skip the dongle driver installation and continue with the rest of the installation, click No.

是(Y)

否(N)

Cancel

知乎 @Study 学长

(10) Continue waiting for the installation to complete, as shown in the figure.



**IAR Embedded
Workbench**

IAR
SYSTEMS

- Install IAR Embedded Workbench® for 8051

- Installation



Installing Visual C++ 2015 Redistributable Package (x86)

- Release

稍等片刻

- Explore the installation media

- Exit

知乎 @Study 学长

(11) Select Launch IAR Embedded Workbench, and then click the Finish button, as shown in the figure.



IAR Embedded Workbench

IAR Embedded Workbench for 8051 10.10.1

IAR
EWCT
EMS

InstallShield Wizard Complete

The InstallShield Wizard has successfully installed IAR Embedded Workbench for 8051. Click Finish to exit the wizard.

- View the release notes
- Launch IAR Embedded Workbench



< Back

Finish

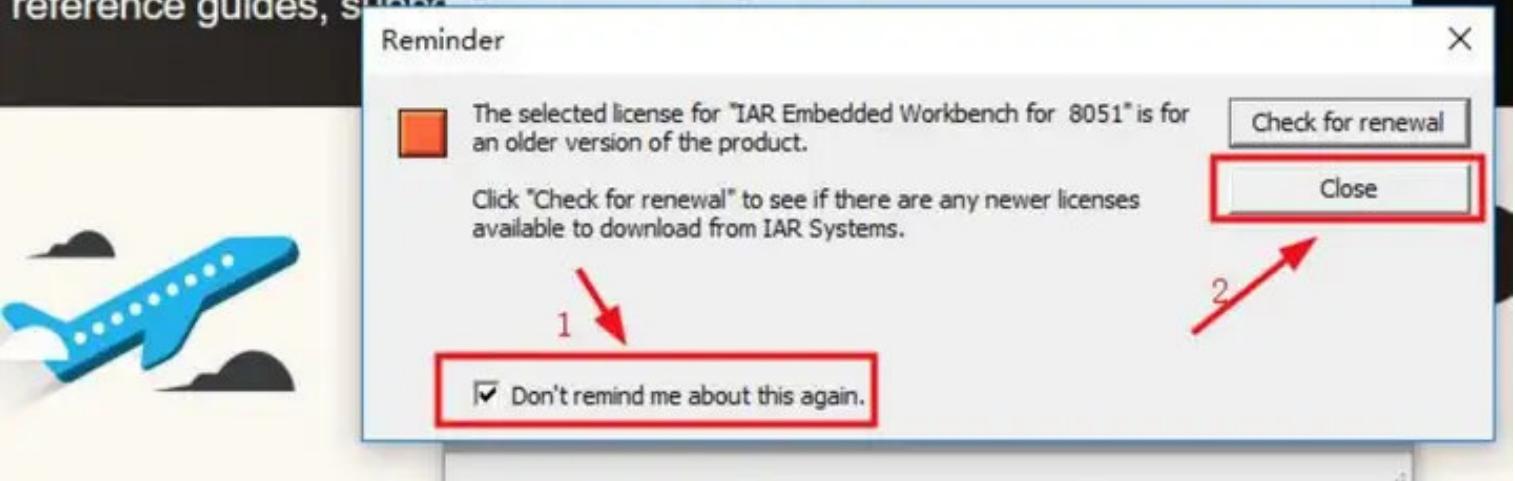
Cancel

知乎 @Study 学长

(12) If the following dialog box pops up, it means that another version of IAR has been installed. Check Don't remind me and click the Close button, as shown in the figure.

AR Information Center fo

e you will find all the information you need to get started: tutorials, example projects, reference guides, support forums, and more.



product explorer

over how to use basic and advanced product features.

User guides

Complete product documentation in PDF format gives you all the user and reference information you

Example projects

Example applications demonstrate hardware performance for specific devices and evaluation boards.

(13) If the following prompt box pops up, check Don't run the Wizard for this product at startup, and then click the "Cancel" button, as shown in the figure.

Welcome

This wizard will help you to activate your IAR Embedded Workbench for 8051 license.

- If you have a license number, enter it here:

- Use a network license

- Register with IAR Systems to get an evaluation license

Don't run the Wizard for this product at startup.



< 上一步

知E@Study

下一步 >

At this point, IAR EW for 8051 has been installed. If you need to register and activate, please continue reading below.
(14) Click Help, and then select Authorization Manager.

IAR Embedded Workbench IDE

File Edit View Project Tools Window



Workspace

Files

Help

Content... 1

Index...

Search...

Product updates

Release notes

IDE Project Management and Building Guide

C/C++ Compiler User Guide

Assembler User Guide

Linker and Library Tools Reference Guide

MISRA-C:1998 Reference Guide (Non C-STAT)

MISRA-C:2004 Reference Guide (Non C-STAT)

C-SPY Debugging Guide

C-STAT Static Analysis Guide

C-SPY RTOS ORTI Plug-in Reference Guide

Migration Guide (to version 8)

Migration Guide (to version 7)

Migrating from Keil to IAR Embedded Workbench

Add or modify support for devices

IAR C Library Functions Reference Guide

IAR on the Web

Information Center



License Manager...

About

(15) Select Offline Activation in the License Manager.

File View License Tools Windows Help

Product List
IAR Embedded C
Version 1.0
Standalone
The license

Activate License...
Use Network License...
Get Evaluation License...
License Details
Servers...
Check for License Renewal...
License Transfer...

Offline Activation...

Offline License Transfer...

(16) Enter the registration code in the following dialog box. You can purchase the registration code on the IAR official website, or continue reading below.

知乎 @Study 学长

Offline activation

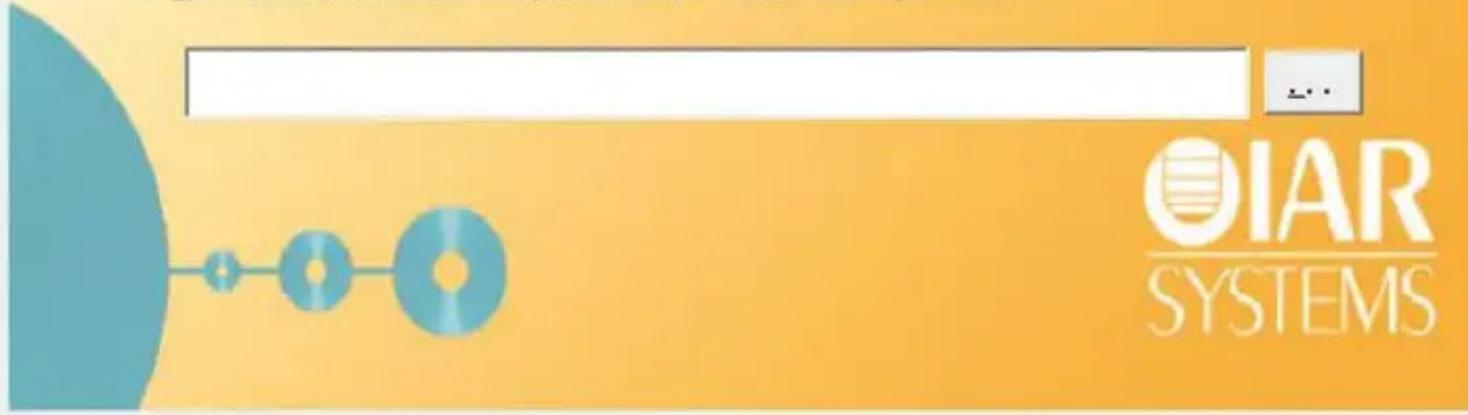
This wizard will help you to activate a license when the License Manager cannot access the internet.

- Generate an activation information file to send to IAR Systems.

License number:

The license number is usually in the format XXXX-XXXX-XXXX-XXXX.

- Use an activation response file from IAR Systems:



4. Software Registration

This section is for reference only. The author **does not provide technical support for this section**. Please operate it yourself if you need it.

IAR EW for 8051 is a paid development software. **If you use it for commercial purposes, please be sure to support genuine software!** If you use it for personal learning, you can use the official 30-day free trial version.



IAR EW for 8051 software registration

1. Introduction and Installation
2. Registration statement
3. Software Download and Registration
4. Frequently asked questions

1. Introduction and Installation

Please refer to: <https://www.kancloud.cn/aiot/zigbee/2486379>

2. Registration statement

IAR EW for 8051 is a paid development software. **If you use it for commercial purposes, please be sure to support genuine software!** If you use it for personal learning, you can use the official 30-day free trial version. This article is intended to provide non-commercial learning assistance for individual learning users. All the following information comes from the Internet. If it damages your interests or infringes on your rights, please contact the author to delete it. The content of this section is for reference only. The author does not provide technical support for the content of this section **in principle**. Please operate it yourself if necessary.

3. Software Download and Registration

After downloading the course resources, unzip IAR EW for 8051 10.10.1.zip to get a registration software named "IAR_UniversalKEY", as shown in the figure.

全部文件 > ... > 课程配套学习资料 (共享) > ZigBee > 《ZigBee3.0开发指南》配套软件工具

<input type="checkbox"/> 文件名	<input type="checkbox"/>	大小
 IAR EW for 8051 10.10.1.zip		327.78 MB
 Pads9.5.zip		1.55 GB
 PuTTY.zip		2.36 MB
 SourceInsight4.0.zip		21.19 MB
 SublimeText.zip		17.47 MB
 TI Z-Stack 3.0.2 (CSDN视频课程专用) .exe		22.64 MB
 TI Z-Stack3.0.1.zip		23.34 MB
 Ubiqua Protocol Analyzer.rar		3.95 MB
 Xshell-7.0.0041r_beta.exe		44.23 MB

Run the activation software and generate the registration code according to the steps in the picture. **Note: Do not close the software during this period!**



Offline License Activation

Coded By: unis e-mail: iarkgen@gmail.com

Special Thanks to: 雷锋, B*****P*****

[Product]

IAR Embedded Workbench for 8051, Standard

Note: Incomplete license (without library sources)!

Wanted password or someone with a real license or dongle!

1. 选择8051

[License Number]

7117-595-415-8531

Generate

[Activate license]

Use the **Browse** button below to navigate to the Activation Information file.
Then click **Activate license**.

2

Browse...

Activate license

No file chosen

3. 复制该号码

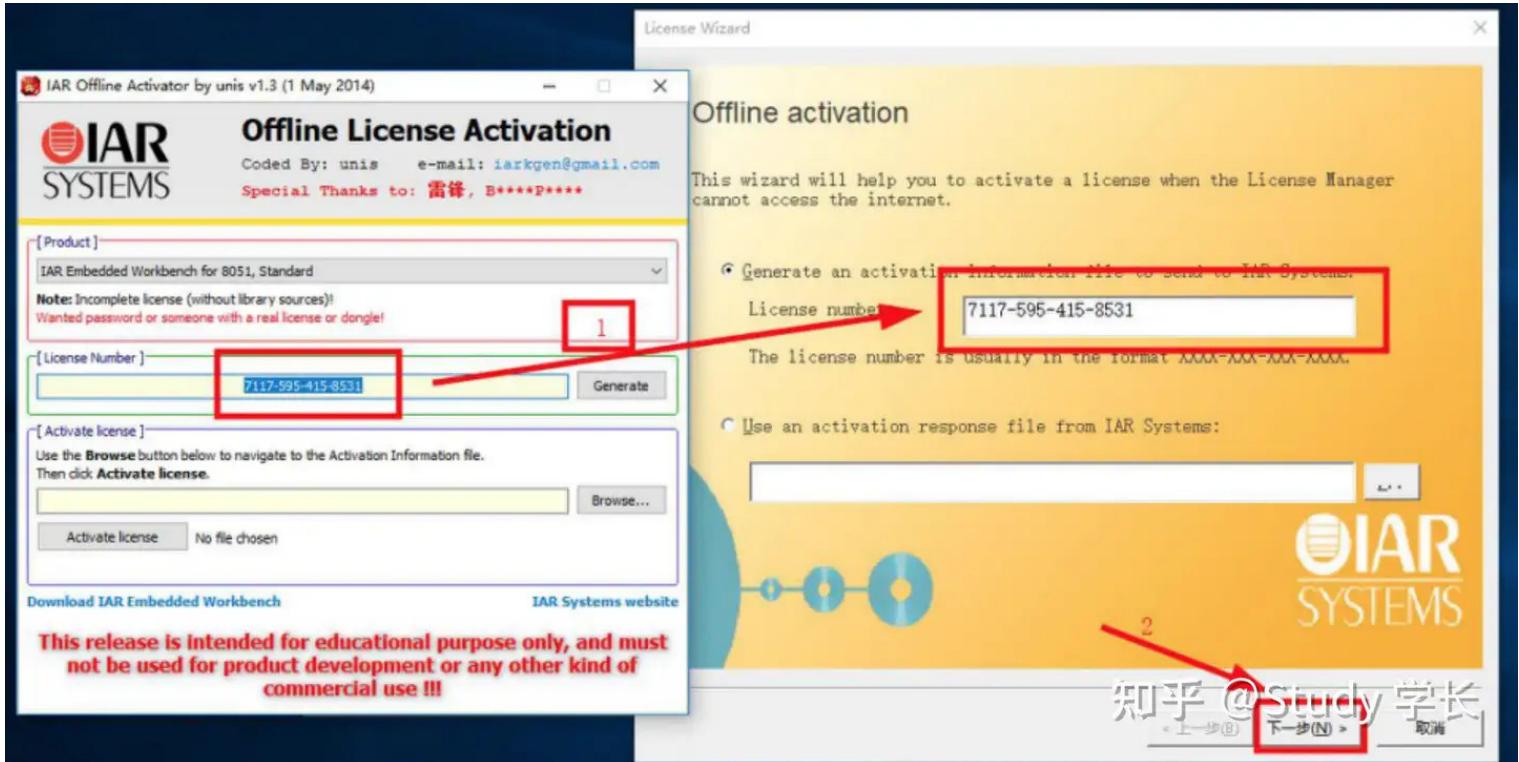
[Download IAR Embedded Workbench](#)

[IAR Systems website](#)

This release is intended for educational purpose only, and must
not be used for product development or any other kind of
commercial use !!!

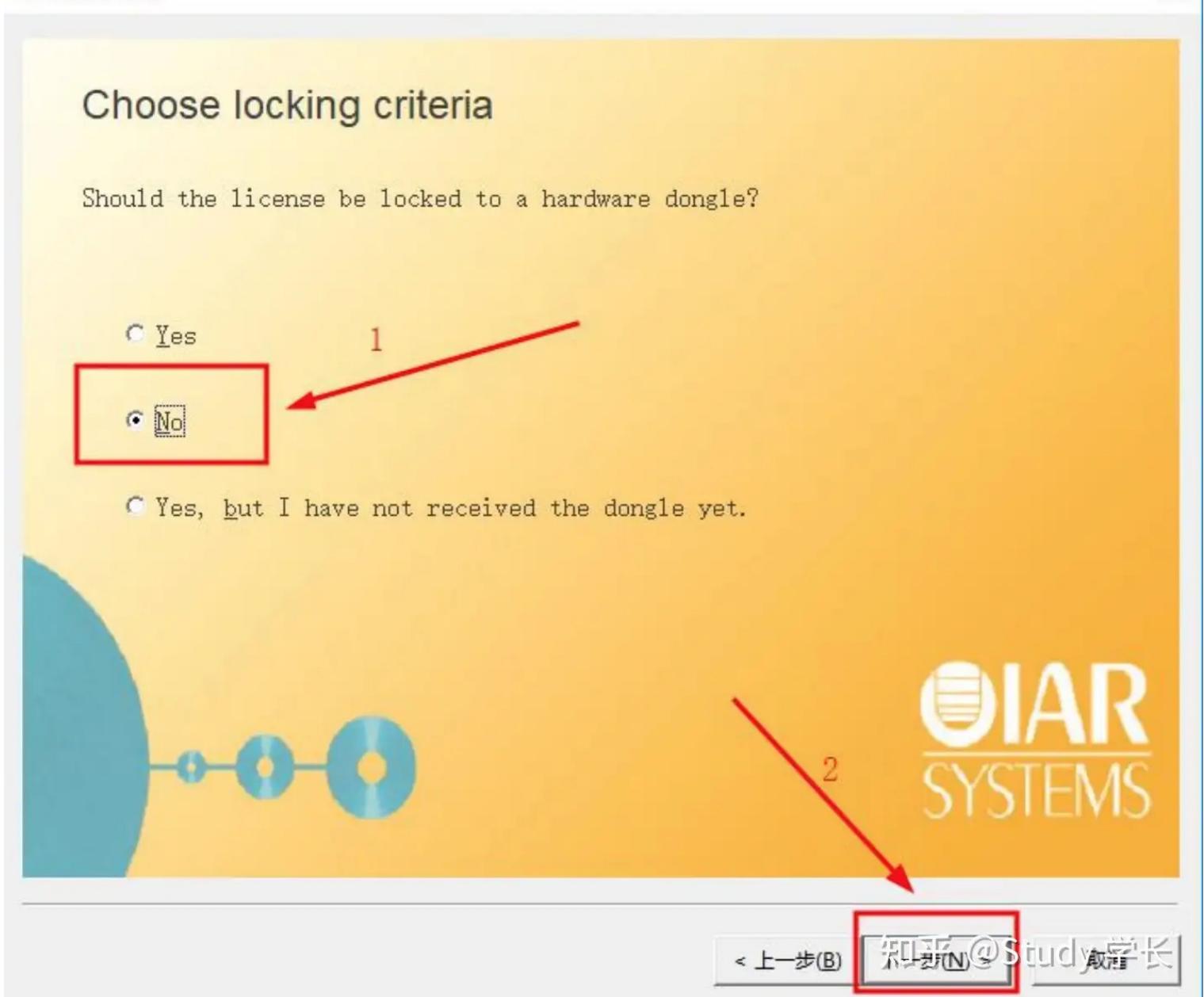
知乎 @Study 学长

Copy the registration code into the dialog box that pops up during the installation step, as shown in the figure.



Select No to not associate a Dongle.

License Wizard



Select the installation directory of IAR EW for 8051 to save the files to be generated

Save activation information

Choose where to save the activation information file.

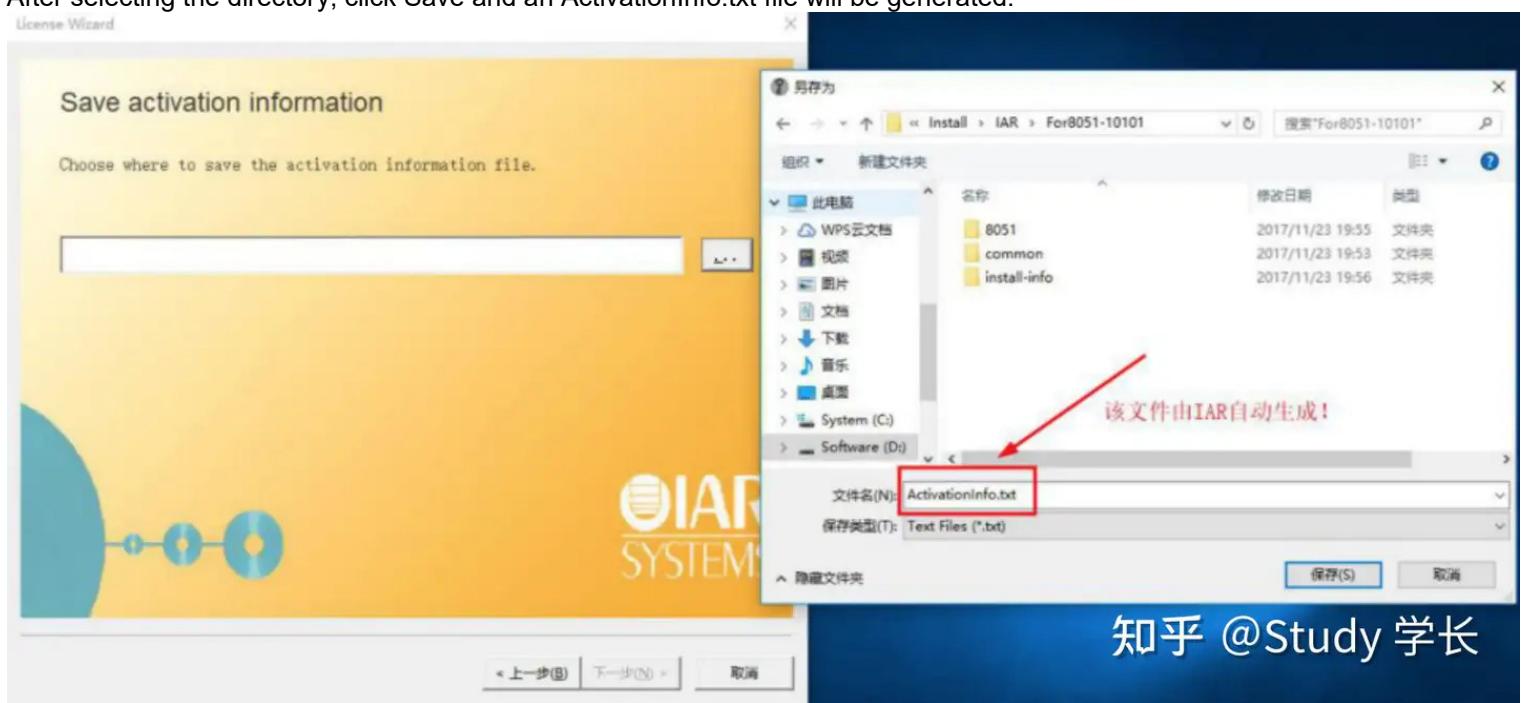


选择安装目录，用来保存注册
请求文件！！！

IAR
SYSTEMS

< 上一步(B) 知乎 @Study 学长 取消

After selecting the directory, click Save and an ActivationInfo.txt file will be generated.



Click Next.

知乎 @Study 学长

Save activation information

Choose where to save the activation information file.

...

点击下一步



< 上一步(B)

下一步(N) >

完成(F)

Click Next.

Request an activation response file

On a computer with internet access, go to the following web page:

<https://register.iar.com/activate>

Follow the instructions on the page. When asked for the activation information file, specify your generated activation information file:

D:\Install\IAR\For8051-10101\ActivationInfo.txt

[Open file location](#)

Click Next when you have obtained the activation response file.

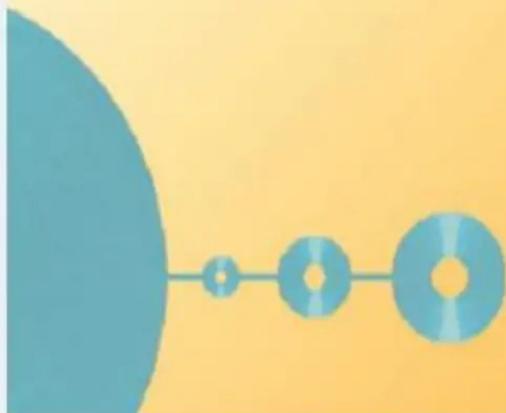


[◀ 上一步\(B\)](#) [下一步\(N\) >](#) [\\$Study 学长](#)

The dialog box indicates that you need to input the registration file (generated by [the registration software](#)) and return to the registration software.

Use the response file to activate the license

Browse to the activation response file from the previous step and click Next.

 ...

◀ 上一步(B) 知乎@Study 完成 取消

Select the ActivationInfo.txt file generated in step 4 as shown in the figure.



Offline License Activation

Coded By: unis e-mail: iarkgen@gmail.com

Special Thanks to: 雷锋, B*****P*****

[Product]

IAR Embedded Workbench for 8051, Standard

Note: Incomplete license (without library sources)!

Wanted password or someone with a real license or dongle!

[License Number]

7117-595-415-8531

Generate

[Activate license]

Use the **Browse** button below to navigate to the Activation Information file.

Then click **Activate license**.

D:\Install\IAR\For8051-101\1\ActivationInfo.txt

Browse...

Activate license

选中IAR自动生成的注册请求文件！

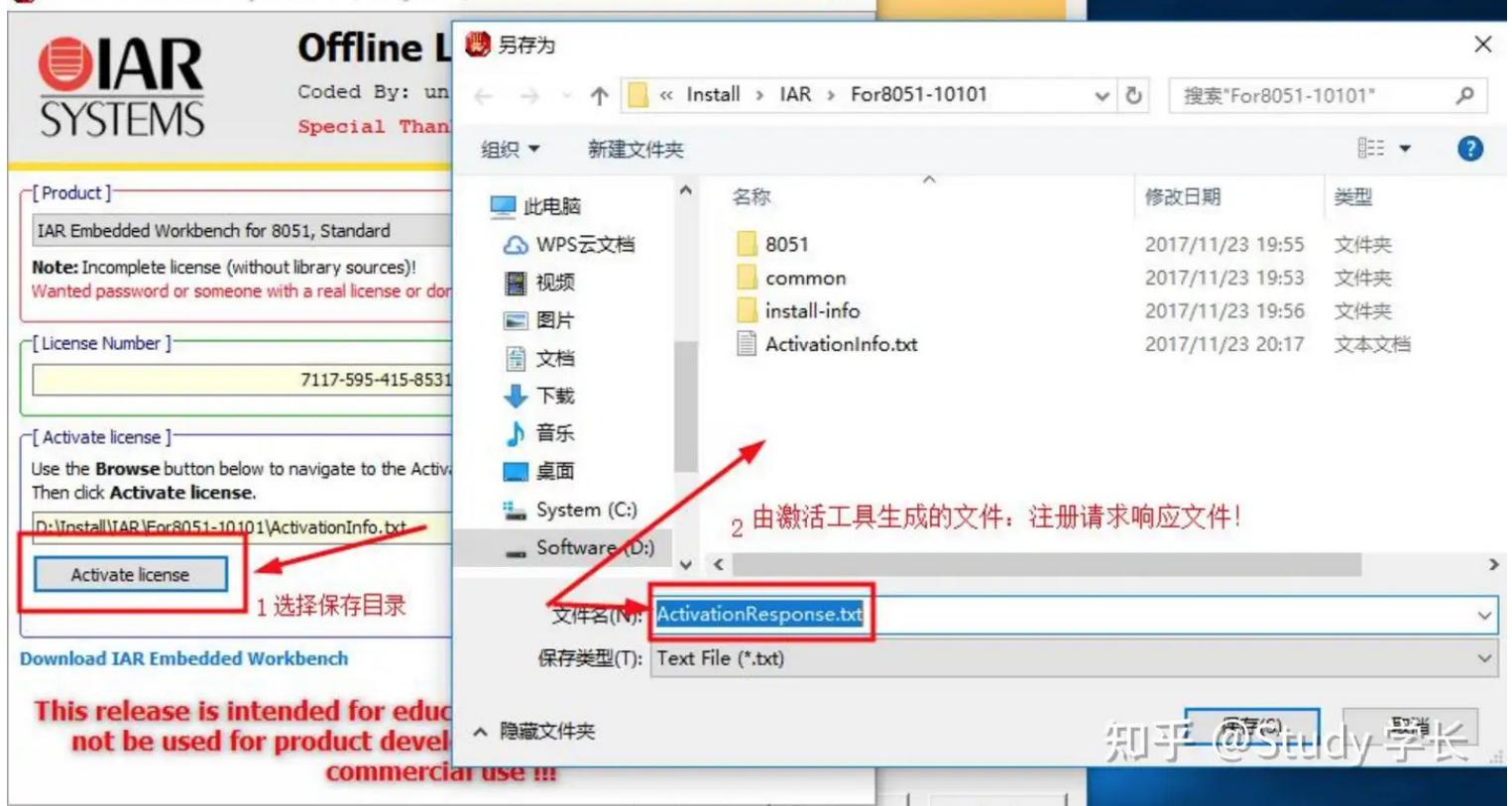
[Download IAR Embedded Workbench](#)

[IAR Systems website](#)

**This release is intended for educational purpose only, and must
not be used for product development or any other kind of
commercial use !!!**

知乎 @Study 学长

Click "Activate license" and select the installation directory to save the generated registration file.



After saving the registration file: ActivationResponse.txt, the following dialog box pops up to indicate successful registration.



Offline License Activation

Coded By: unis e-mail: iarkgen@gmail.com

Special Thanks to: 雷锋, B*****P*****

[Product]

IAR Embedded Workbench for 8051, Standard

Note: Incomplete license (without library sources)!

Wanted password or someone with a real license or dongle!

[License Number]

7117-595-415-8531

Generate

[Activate license]

Use the **Browse** button below to navigate to the Activation Information file.

Then click **Activate license**.

D:\Install\IAR\For8051-10101\ActivationInfo.txt

Browse...

Activate license

Done!

响应文件生成成功！！

Move the Activation Response file to the installation computer and finish the activation process.

[Download IAR Embedded Workbench](#)

[IAR Systems website](#)

**This release is intended for educational purpose only, and must
not be used for product development or any other kind of
commercial use !!!**

知乎 @Study 学长

Import the registration file into the dialog box that pops up in step 7.

Use the response file to activate the license

Browse to the activation response file from the previous step and click Next.



The following dialog box pops up to indicate that activation is complete.

Finished



知乎 @Study 学长

Select No to not export the file.

Extract files

Your license for "IAR Embedded Workbench for 8051" entitles you to extract the following package:

IAR Library Source for 8051

Note: Extracting the files requires administrator privileges.

Would you like to extract the files now?

1

Yes

2

No

Don't notify me about this again.

知乎 @Study 学长

Check whether the registration is complete, then click the Exit button



File View License Tools Windows Help

Product List X

IAR Embedded Workbench for 8051

Version 10.10

Standalone license - IAR Embedded Workbench for 8051

(Double-click a product to see more details)

注册完成!

知乎 @Study 学长

Click Exit to exit the IAR installer.

**IAR Embedded
Workbench**IAR
SYSTEMS

- Install IAR Embedded Workbench® for 8051
- Installation and licensing information
- Release notes
- Explore the installation media
- **Exit**

知乎 @Study 学长

After closing the registration tool, close IAR and then restart it.

4. Frequently asked questions

When registering software, what should I do if the message "Could not get imformation for the given license number" appears after entering the license number?

Carefully check every step of the tutorial and follow it exactly.

Unplug the network cable from the computer, turn off the WiFi completely, and turn off any other networks, then try again;

If you have installed other versions of IAR, you can try to crack the code with other versions;

Try again on another computer;

The registered software may not be applicable to the new IAR version. You can use the IAR version provided by the author.

If the above methods still cannot solve other problems, you can try other methods.

4.3. TI Z-Stack 3.0 Introduction and Installation

Technical support instructions

1. Generally, self-study is the main method.

2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>

3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Software Download

(1) Download address: See "Supporting Resource Download" in the catalog of this tutorial.

(2) After downloading the supporting resources, the location of the installation package is shown in the figure.

全部文件 > ... > 课程配套学习资料（共享）> ZigBee > 《ZigBee3.0开发指南》配套软件工具		
<input type="checkbox"/> 文件名	大小	
<input type="checkbox"/> CH34x USB转串口驱动.zip	229.23 KB	
<input type="checkbox"/> Flash-programmer-1.12.8.zip	18.24 MB	
<input type="checkbox"/> IAR EW for 8051 10.10.1.zip	327.78 MB	
<input type="checkbox"/> PuTTY.zip	2.36 MB	
<input type="checkbox"/> SmartRF04EB Driver.zip	1.97 MB	
<input type="checkbox"/> TI Z-Stack 3.0.2 (CSDN视频课程专用).exe	22.64 MB	
<input checked="" type="checkbox"/> TI Z-Stack3.0.1.zip	23.34 MB	
<input type="checkbox"/> Ubiqua Protocol Analyzer.rar	3.95 MB	
<input type="checkbox"/> 善学坊官方学习指南.html	212.00 B	
<input type="checkbox"/> 图片取模软件.zip	512.11 KB	
<input type="checkbox"/> 友善串口调试助手.zip	20.26 MB	
<input type="checkbox"/> 字模软件2.2.zip	256.59 KB	

TI Z-Stack 3.0 Introduction and Installation

- Zhihu boutique tutorial: <https://zhuanlan.zhihu.com/p/535318714>

4.4. SmartRF Flash Programmer Download and Installation

Technical support instructions

1. Generally, self-study is the main method.

2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>

3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Software Download

- (1) Download address: See "Supporting Resource Download" in the catalog of this tutorial.
(2) After downloading the supporting resources, the location of the installation package is shown in the figure.

全部文件 > ... > 课程配套学习资料（共享）> ZigBee > 《ZigBee3.0开发指南》配套软件工具	文件名	大小
CH34x USB转串口驱动.zip	229.23 KB	
Flash-programmer-1.12.8.zip	18.24 MB	
IAR EW for 8051 10.10.1.zip	327.78 MB	
PuTTY.zip	2.36 MB	
SmartRF04EB Driver.zip	1.97 MB	
TI Z-Stack 3.0.2 (CSDN视频课程专用).exe	22.64 MB	
TI Z-Stack3.0.1.zip	23.34 MB	
Ubiqua Protocol Analyzer.rar	3.95 MB	
善学坊官方学习指南.html	212.00 B	
图片取模软件.zip	512.11 KB	
友善串口调试助手.zip	20.26 MB	
字模软件2.2.zip	256.59 KB	

SmartRF Flash Programmer Download and Installation Tutorial

- Zhihu boutique tutorial: <https://zhuanlan.zhihu.com/p/370210691>

4.5. Serial Port Assistant Introduction and Installation

Technical support instructions

- Generally, self-study is the main method.
- You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
- Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Software Download

(1) Download address: See "Supporting Resource Download" in the catalog of this tutorial.

(2) After downloading the supporting resources, the location of the installation package is shown in the figure.

全部文件 > ... > 课程配套学习资料（共享）> ZigBee > 《ZigBee3.0开发指南》配套软件工具	文件名	大小
<input type="checkbox"/> CH34x USB转串口驱动.zip		229.23 KB
<input type="checkbox"/> Flash-programmer-1.12.8.zip		18.24 MB
<input type="checkbox"/> IAR EW for 8051 10.10.1.zip		327.78 MB
<input type="checkbox"/> PuTTY.zip		2.36 MB
<input type="checkbox"/> SmartRF04EB Driver.zip		1.97 MB
<input type="checkbox"/> TI Z-Stack 3.0.2 (CSDN视频课程专用) .exe		22.64 MB
<input type="checkbox"/> TI Z-Stack3.0.1.zip		23.34 MB
<input type="checkbox"/> Ubiqua Protocol Analyzer.rar		3.95 MB
<input type="checkbox"/> 善学坊官方学习指南.html		212.00 B
<input type="checkbox"/> 图片取模软件.zip		512.11 KB
<input type="checkbox"/> 友善串口调试助手.zip		20.26 MB
<input type="checkbox"/> 字模软件2.2.zip		256.59 KB

Download and install the Friendly Serial Debug Assistant

- Zhihu boutique tutorial: <https://zhuanlan.zhihu.com/p/371061428>

4.6. SmartRF04EB Driver

Technical support instructions

- Generally, self-study is the main method.
- You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
- Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Software Download

- (1) Download address: See "Supporting Resource Download" in the catalog of this tutorial.
(2) After downloading the supporting resources, the location of the installation package is shown in the figure.

全部文件 > ... > 课程配套学习资料（共享）> ZigBee > 《ZigBee3.0开发指南》配套软件工具	
文件名	大小
CH34x USB转串口驱动.zip	229.23 KB
Flash-programmer-1.12.8.zip	18.24 MB
IAR EW for 8051 10.10.1.zip	327.78 MB
PuTTY.zip	2.36 MB
SmartRF04EB Driver.zip	1.97 MB
TI Z-Stack 3.0.2 (CSDN视频课程专用).exe	22.64 MB
TI Z-Stack3.0.1.zip	23.34 MB
Ubiqua Protocol Analyzer.rar	3.95 MB
善学坊官方学习指南.html	212.00 B
图片取模软件.zip	512.11 KB
友善串口调试助手.zip	20.26 MB
字模软件2.2.zip	256.59 KB

SmartRF04EB Driver

- Zhihu boutique tutorial: <https://zhuanlan.zhihu.com/p/370241426>

4.7. USB to Serial Port Driver

Technical support instructions

- Generally, self-study is the main method.
- You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
- Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Software Download

- (1) Download address: See "Supporting Resource Download" in the catalog of this tutorial.
(2) After downloading the supporting resources, the location of the installation package is shown in the figure.

<input type="checkbox"/> 文件名	<input type="checkbox"/> 大小
CH34x USB转串口驱动.zip	229.23 KB
Flash-programmer-1.12.8.zip	18.24 MB
IAR EW for 8051 10.10.1.zip	327.78 MB
PuTTY.zip	2.36 MB
SmartRF04EB Driver.zip	1.97 MB
TI Z-Stack 3.0.2 (CSDN视频课程专用) .exe	22.64 MB
TI Z-Stack3.0.1.zip	23.34 MB
Ubiqua Protocol Analyzer.rar	3.95 MB
善学坊官方学习指南.html	212.00 B
图片取模软件.zip	512.11 KB
友善串口调试助手.zip	20.26 MB
字模软件2.2.zip	256.59 KB

USB to Serial Driver

- Zhihu boutique tutorial «CH340 USB to serial port driver installation» (<https://zhuanlan.zhihu.com/p/382351660>)

4.8. Other software installation (optional)

4.8.1. Xshell 7 Introduction and Installation Guide

4.8.1.1. Xshell 7 Introduction and Installation Guide

Technical support instructions

- Generally, self-study is the main method.
- You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
- Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Xshell 7 Introduction and Installation Guide

- Zhihu boutique tutorial: <https://zhuanlan.zhihu.com/p/486650064>

4.8.2. PuTTY Introduction and Installation

4.8.2.1. PuTTY Introduction and Installation

4.8.2.1.1. PuTTY Introduction and Installation

Technical support instructions

- Generally, self-study is the main method.
- You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
- Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Software Download

(1) Download address: See "Supporting Resource Download" in the catalog of this tutorial.

(2) After downloading the supporting resources, the location of the installation package is shown in the figure.

全部文件 > ... > 课程配套学习资料（共享）> ZigBee > 《ZigBee3.0开发指南》配套软件工具

<input type="checkbox"/> 文件名	大小
<input type="checkbox"/> CH34x USB转串口驱动.zip	229.23 KB
<input type="checkbox"/> Flash-programmer-1.12.8.zip	18.24 MB
<input type="checkbox"/> IAR EW for 8051 10.10.1.zip	327.78 MB
<input checked="" type="checkbox"/> PuTTY.zip	2.36 MB
<input type="checkbox"/> SmartRF04EB Driver.zip	1.97 MB
<input type="checkbox"/> TI Z-Stack 3.0.2 (CSDN视频课程专用) .exe	22.64 MB
<input type="checkbox"/> TI Z-Stack3.0.1.zip	23.34 MB
<input type="checkbox"/> Ubiqua Protocol Analyzer.rar	3.95 MB
<input type="checkbox"/> 善学坊官方学习指南.html	212.00 B
<input type="checkbox"/> 图片取模软件.zip	512.11 KB
<input type="checkbox"/> 友善串口调试助手.zip	20.26 MB
<input type="checkbox"/> 字模软件2.2.zip	256.59 KB

PuTTY Introduction and Installation

- Zhihu boutique tutorial: <https://zhuanlan.zhihu.com/p/535091460>

5. Part 2: Introduction to 51 MCU - Based on CC2530

5.1. Chapter 1: CC2530 Development Basic Experiment

5.2. Chapter 2: GPIO Experiment

5.3. Chapter 3: Timer Experiment

5.4. Chapter 4: Serial Communication Experiment

5.5. Chapter 5: ADC Experiment - Using the Light Sensor

5.6. Chapter 6: OLED Display Experiment

5.7. Chapter 7: Peripheral Experiments

5.1. Chapter 1: CC2530 Development Basic Experiment

5.1.1. Create a new workspace and project

5.1.2 Source code writing and compilation

5.1.3 TI Z-Stack 3.0 Introduction and Installation

5.1.4 Firmware Burning

5.1.1. Create a new workspace and project

Technical support instructions:

- Generally, self-study is the main method.

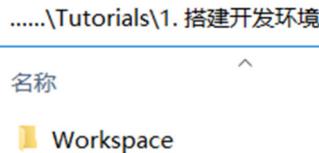
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Video Explanation

<https://www.bilibili.com/video/BV1k34y1D7Vz?p=2>

Create a new workspace

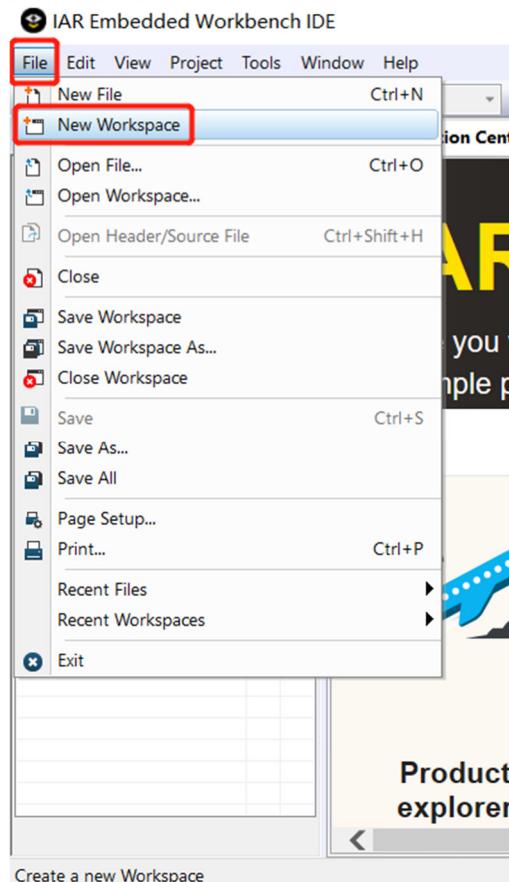
- (1) Create a folder named Workspace, as shown in the figure.



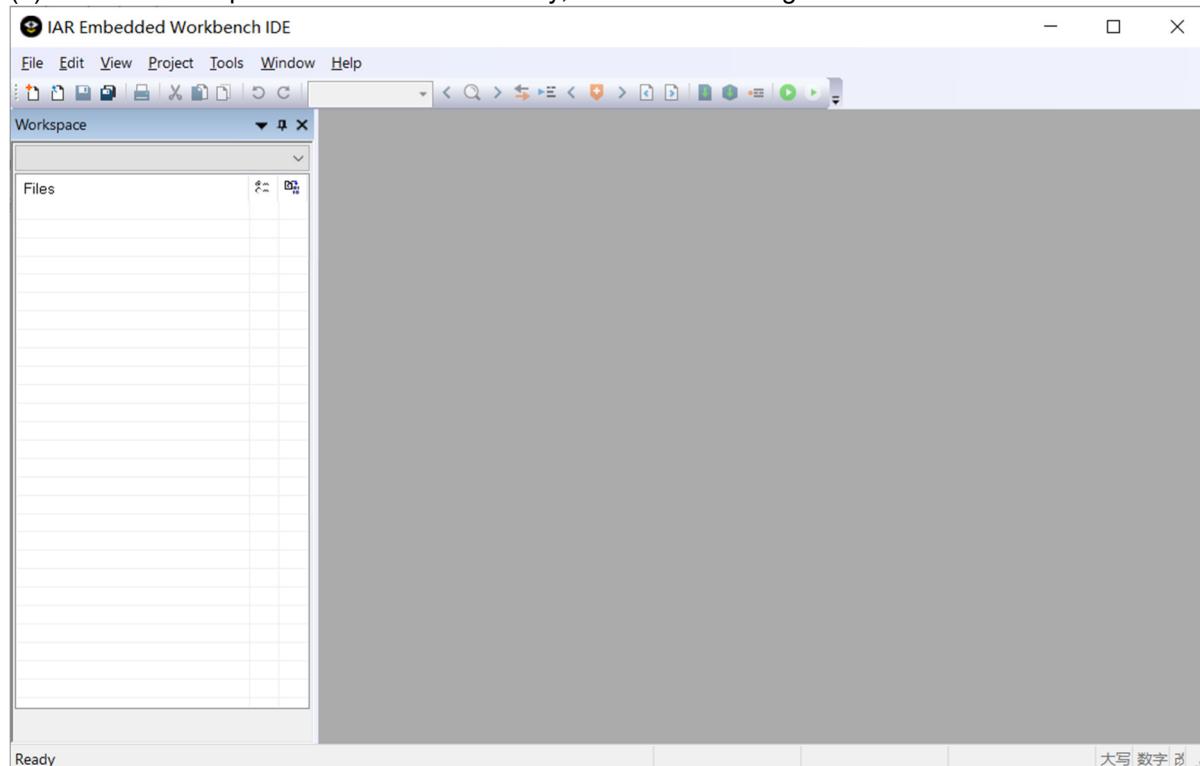
- (2) Run IAR EW for 8051 10.10.1, as shown in the figure.



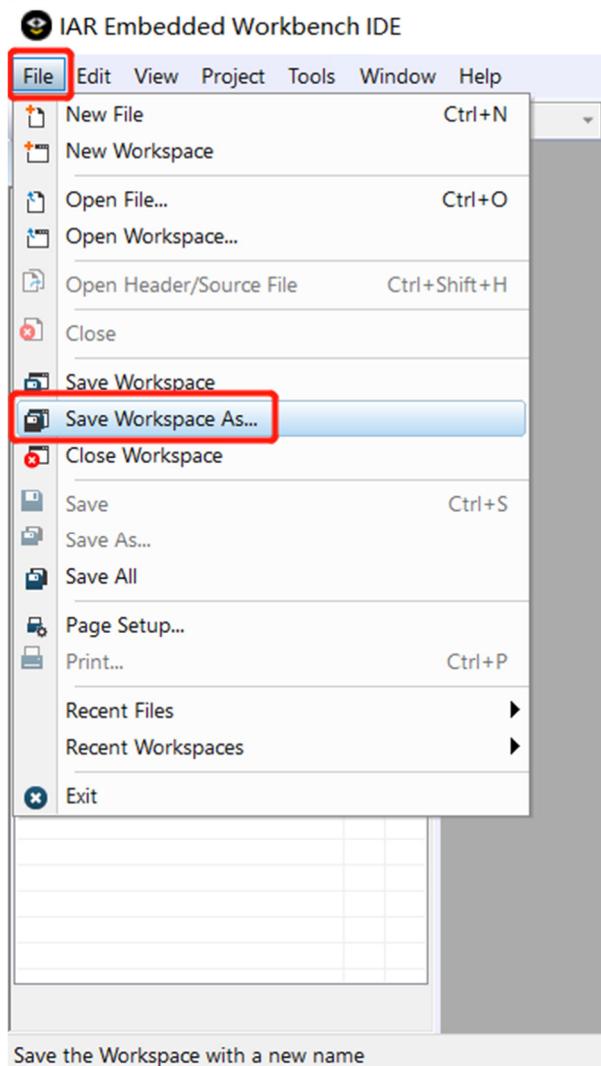
(3) Select File and New Workspace in turn to create a new workspace, as shown in the figure.



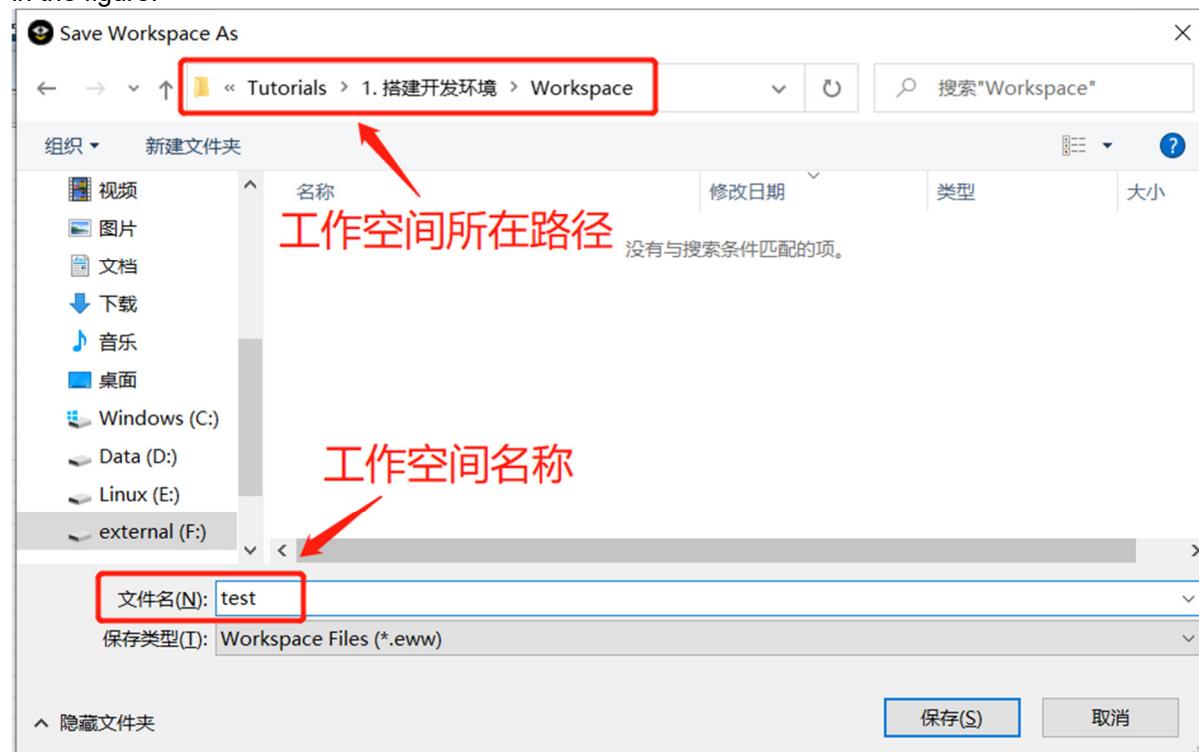
(4) The new workspace is created successfully, as shown in the figure.



(5) Select File and Save Workspace As... to save the workspace, as shown in the figure.



(6) Select the directory where you want to save the workspace, then set the workspace name, and finally click Save, as shown in the figure.



(7) After the workspace is saved successfully, it will be as shown in the figure.

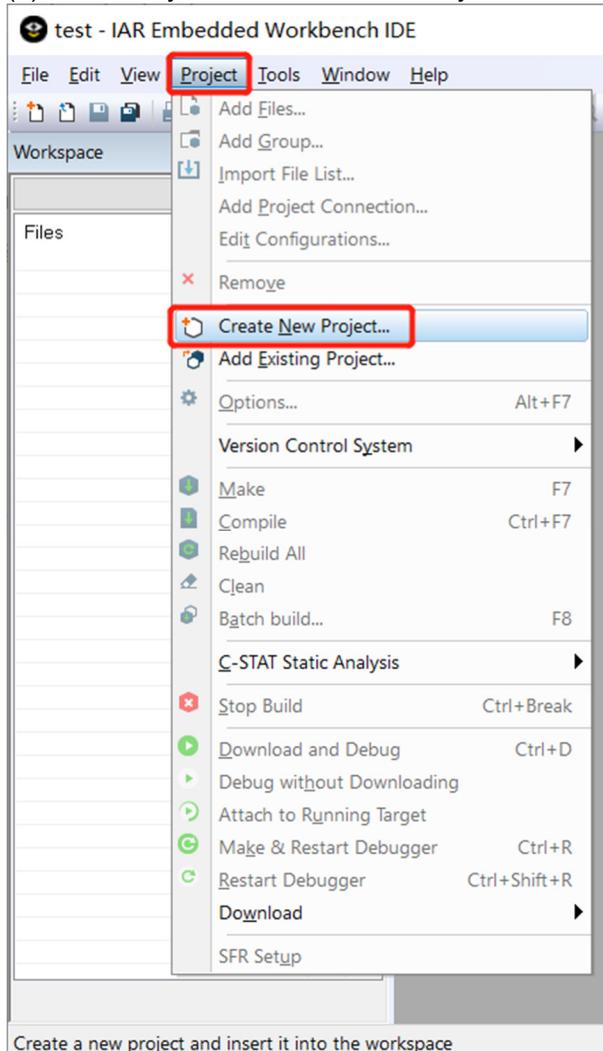
.....\Tutorials\1. 搭建开发环境\Workspace

名称

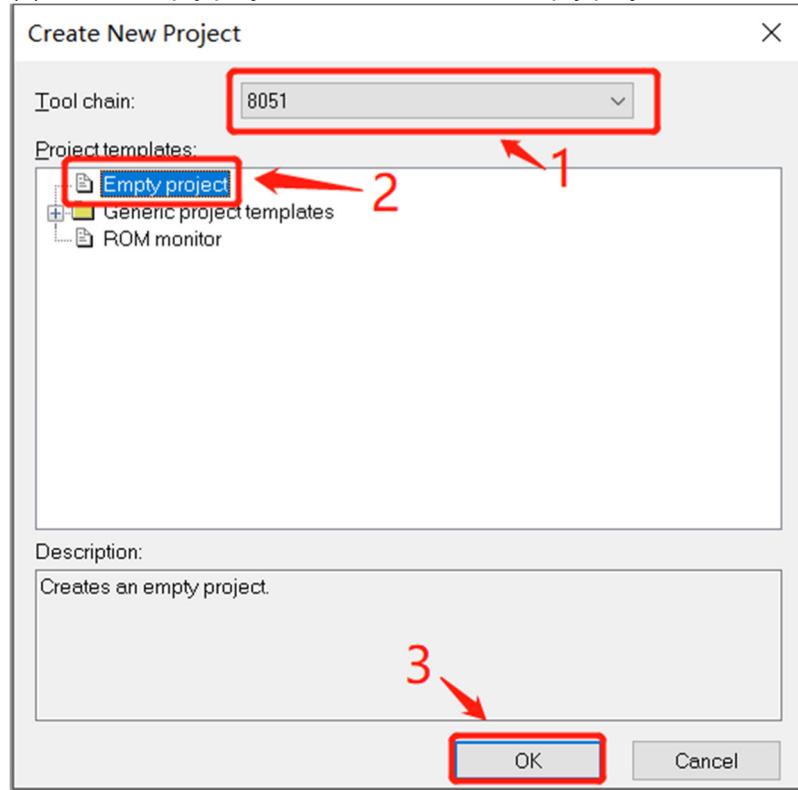
settings
test.eww

New Construction

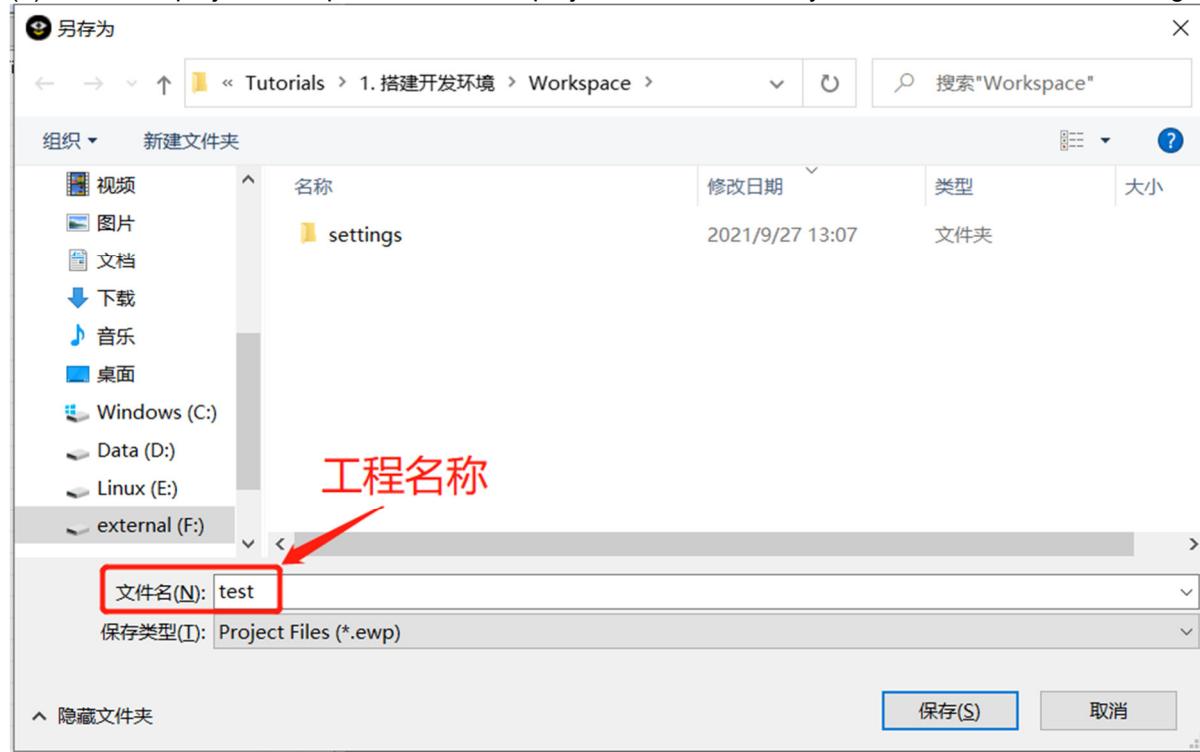
(1) Select Project and Create New Project... in turn to create a new project in the workspace, as shown in the figure.



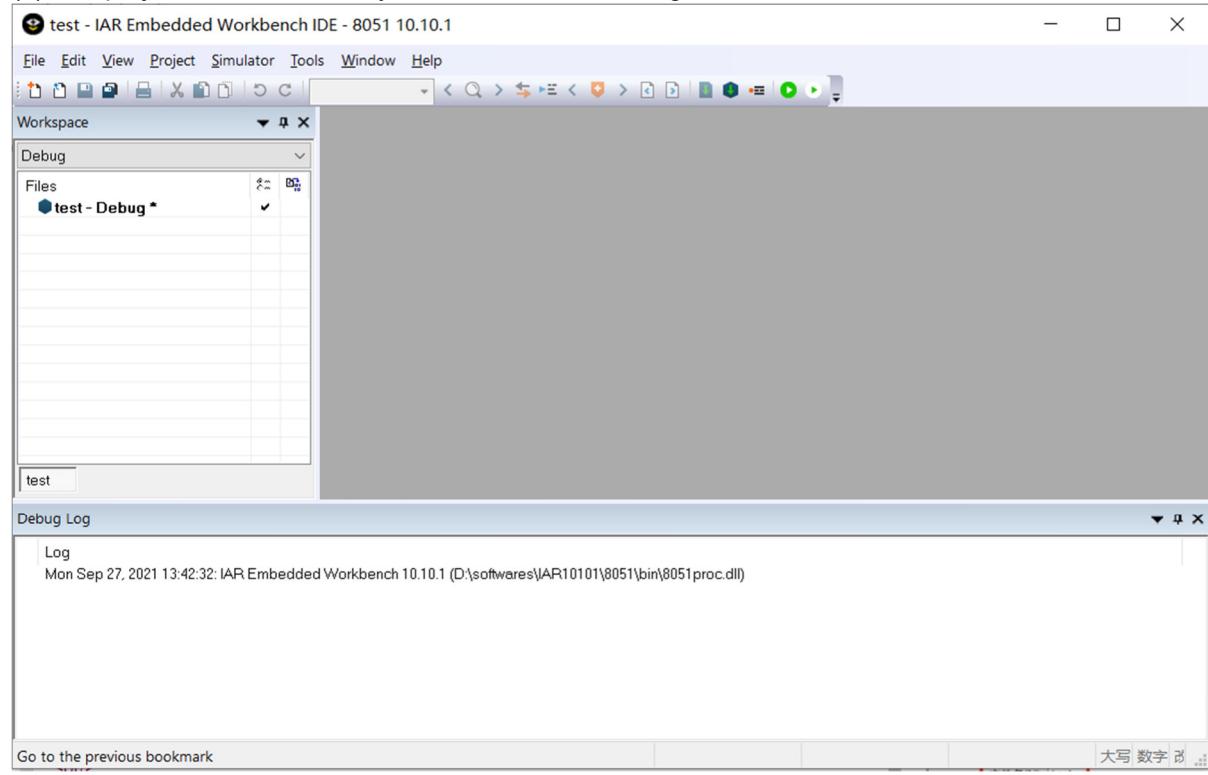
(2) Select Empty project and create a new empty project, as shown in the figure.



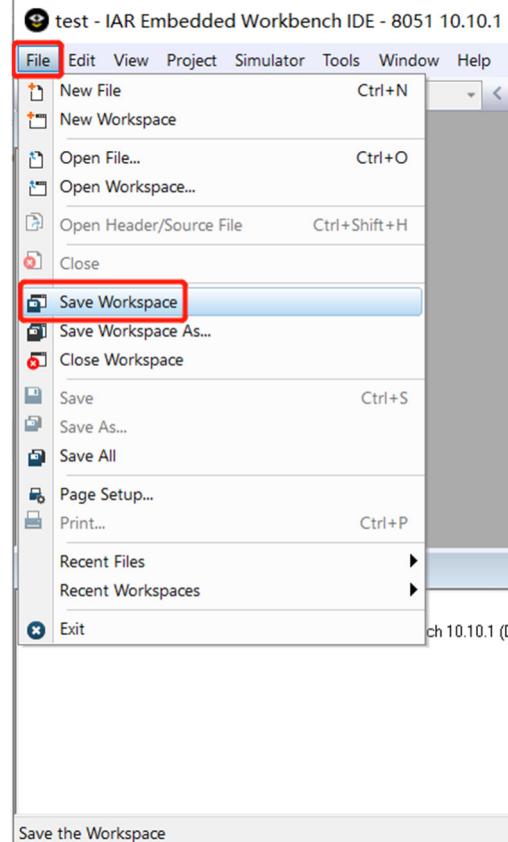
(3) Select the project save path and edit the project name, and finally click Save, as shown in the figure.



(4) The project was successfully built, as shown in the figure.



(5) Select File and Save Workspace in turn to save the workspace, as shown in the figure.



5.1.2. Source code writing and compilation

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

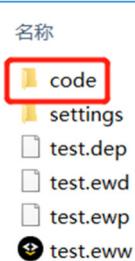
Video Explanation

<https://www.bilibili.com/video/BV1k34y1D7Vz?p=3>

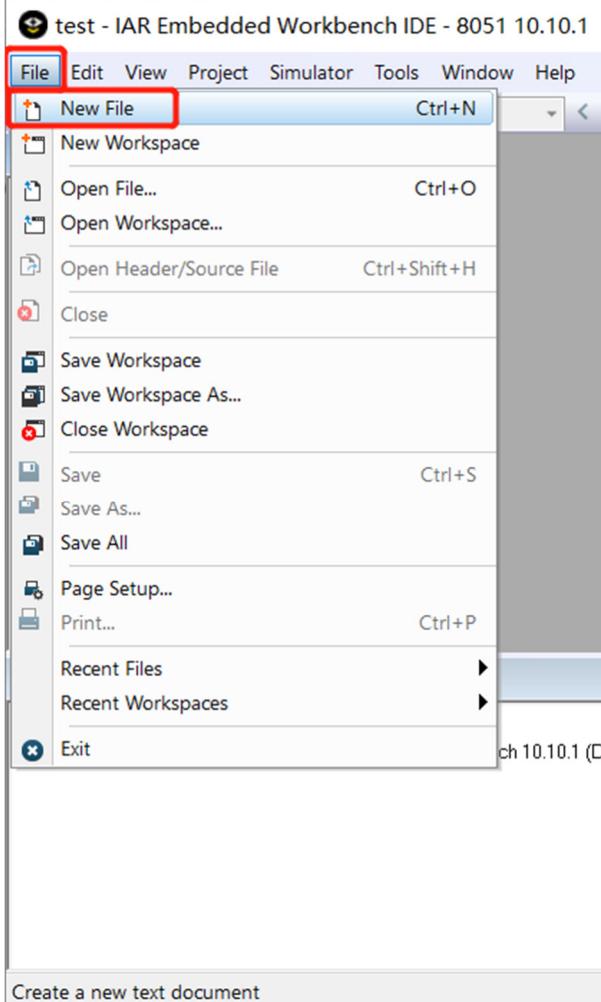
1. Create a new source code folder

(1) In the directory where the workspace and project are saved, create a new code folder to save the source code, as shown in the figure.

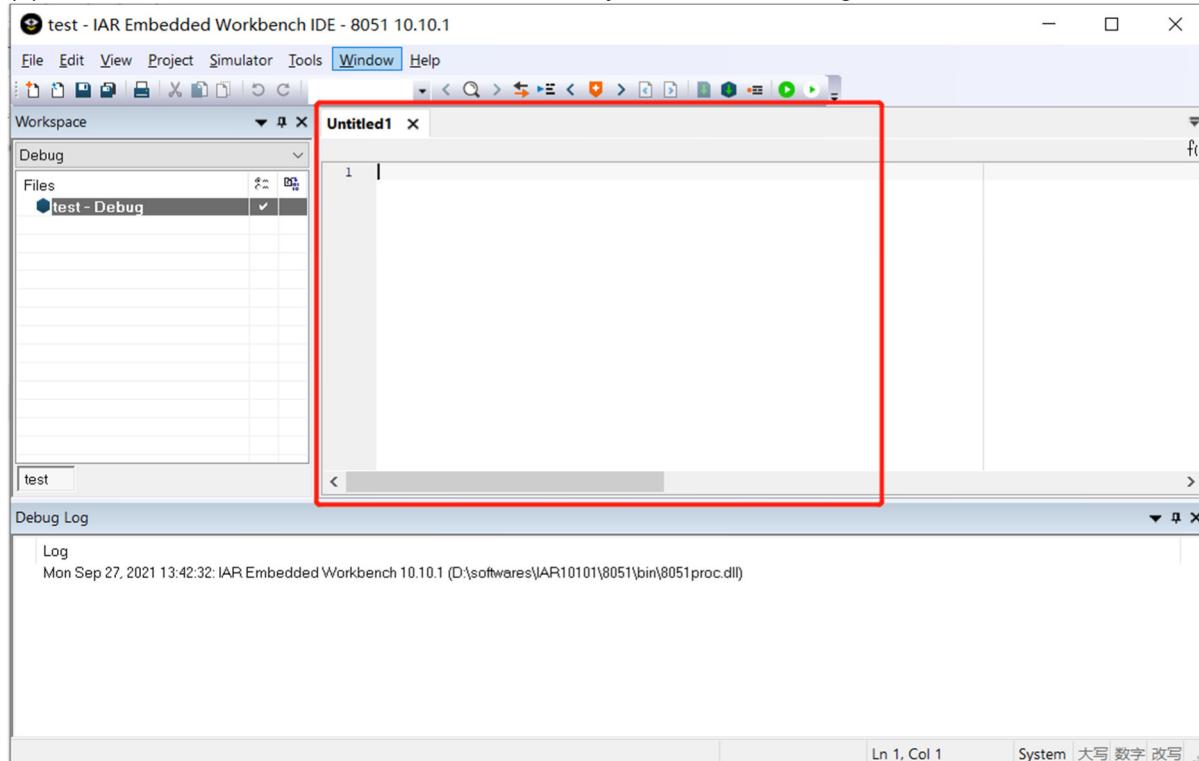
.....\Tutorials\1. 搭建开发环境\Workspace



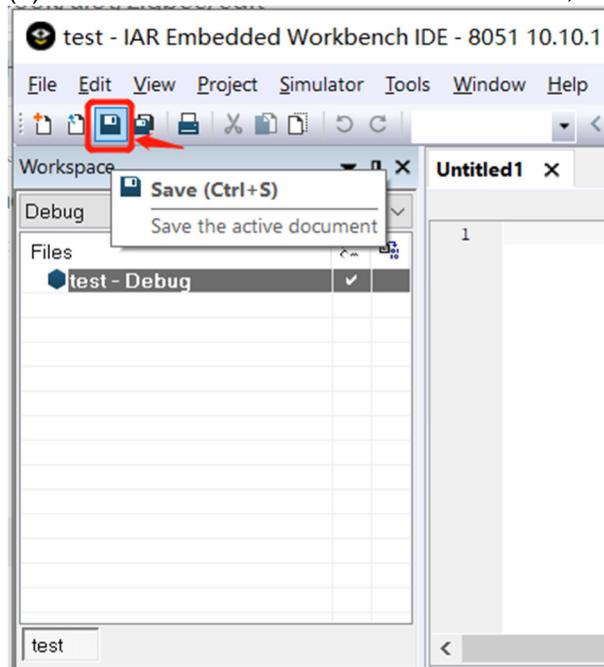
(2) Select File and New File in turn to create a new source code file, as shown in the figure.



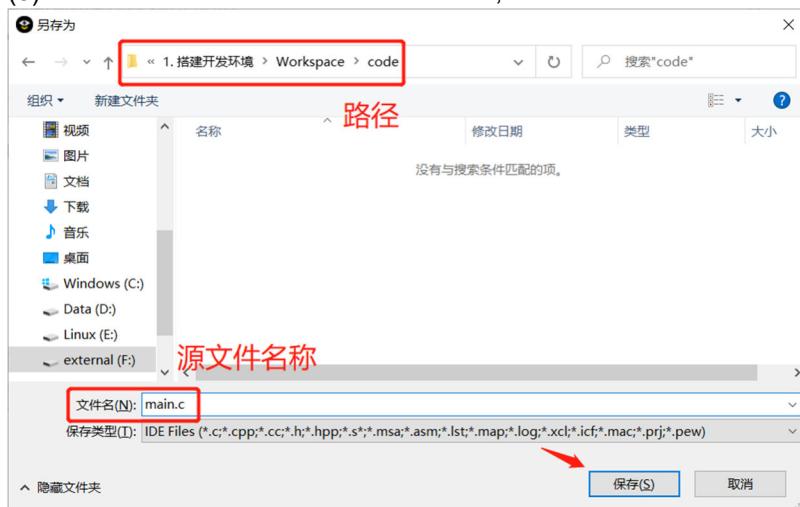
(3) The new source code file is created successfully, as shown in the figure.



(4) Click the Save button to save the source file, as shown in the figure.

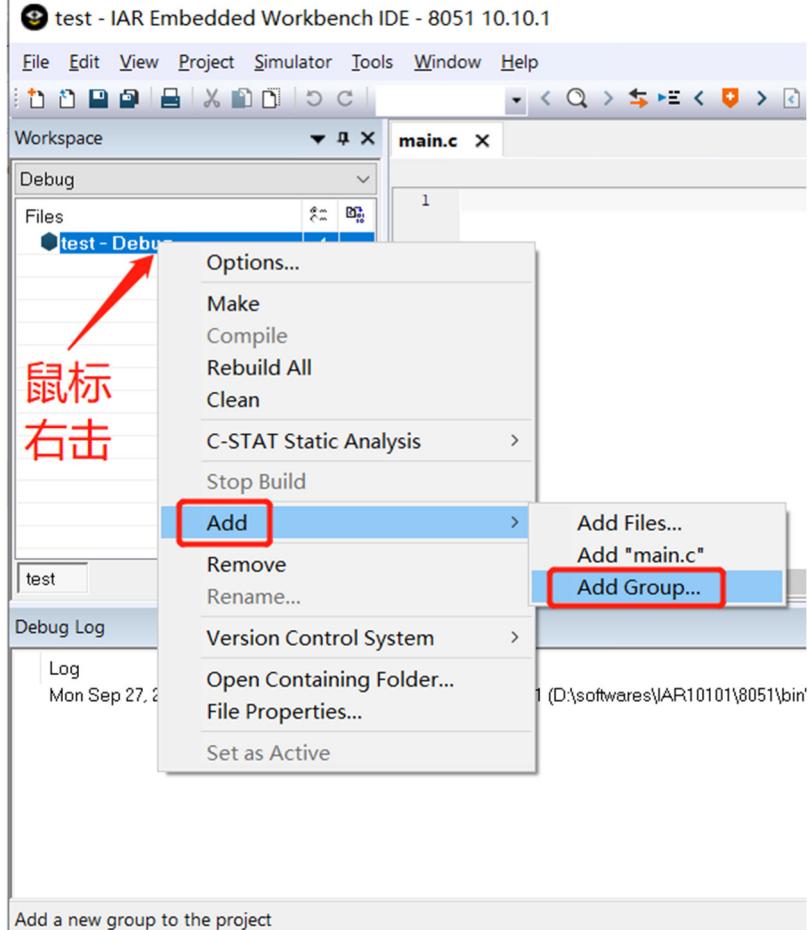


(5) Select the source file save destination, name the source file main.c, and then click Save, as shown in the figure.

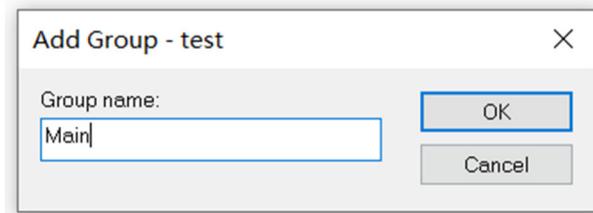


2. Add source files to the project

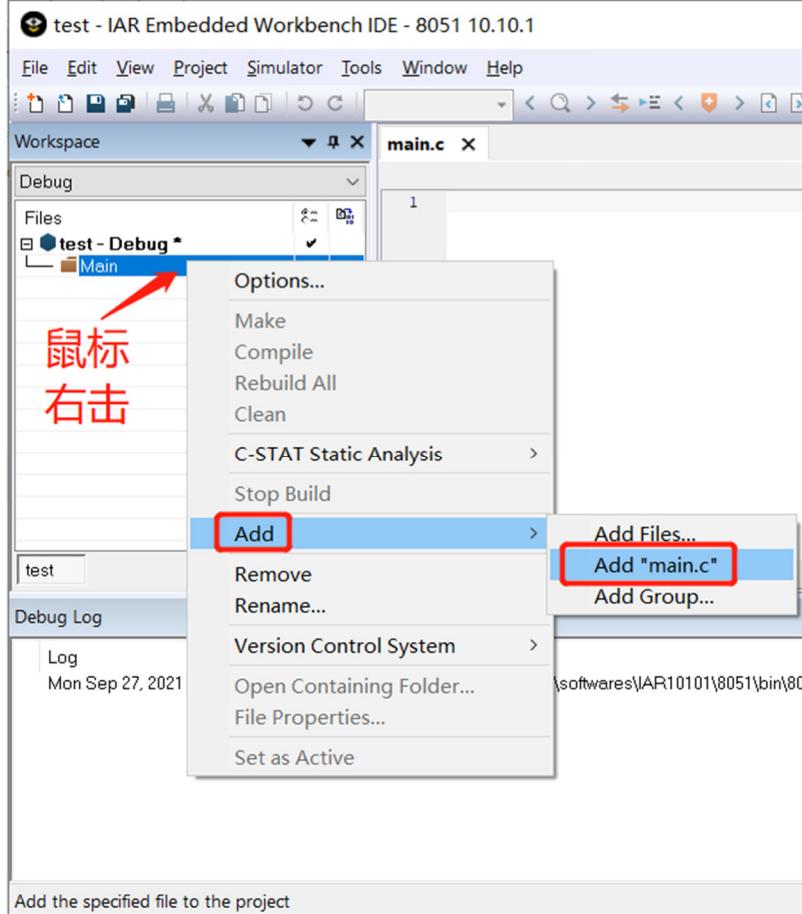
(1) Right-click the project name, then select Add and AddGroup to create a new group, as shown in the figure.



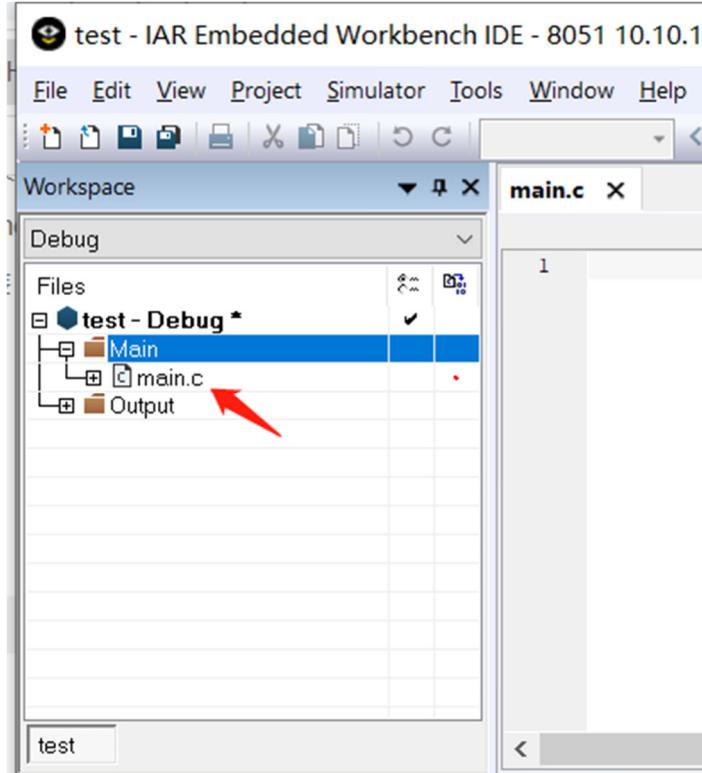
(2) Set the group name to Main, and then click OK, as shown in the figure.



(3) Right-click the newly created group, select Add and Add "main.c" in turn, and add the newly created source file to the group, as shown in the figure.



(4) After the source file is added successfully, as shown in the figure.



3. Write a simple test program

After adding main.c, you can add the following code in main.c:

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

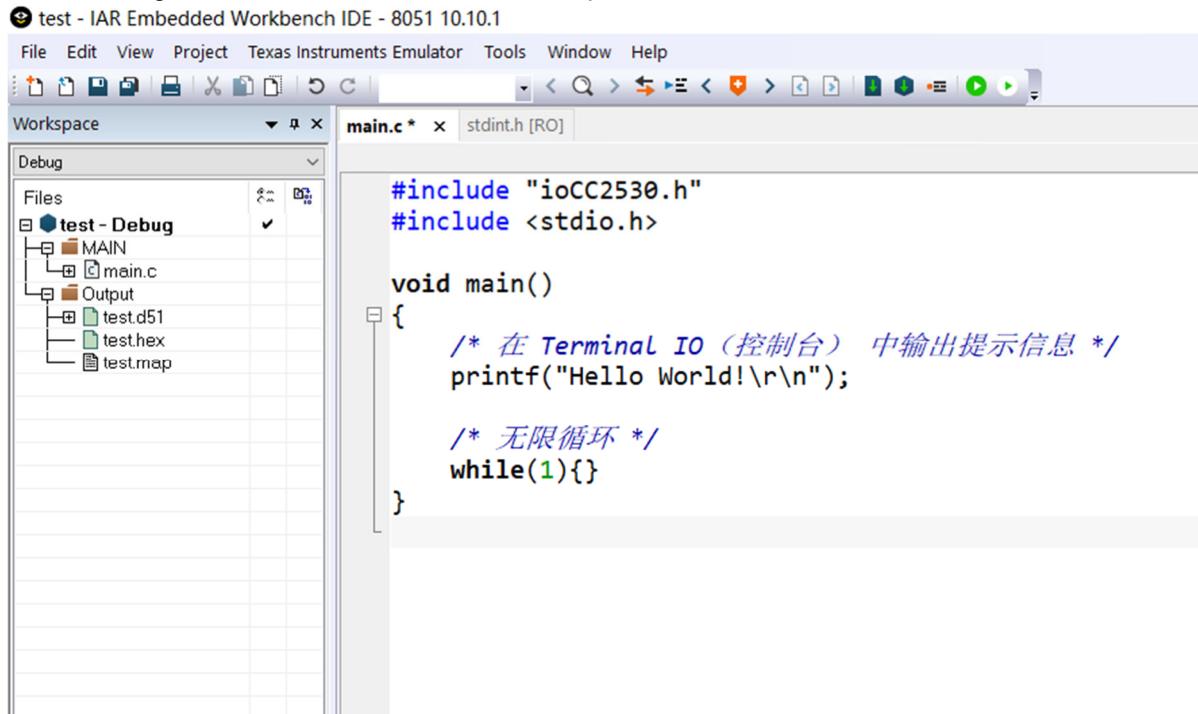
void main()
```

```
{
/* Output prompt information in Terminal IO (console) */
printf("Hello World!\r\n");

/* infinite loop */
while(1) {}

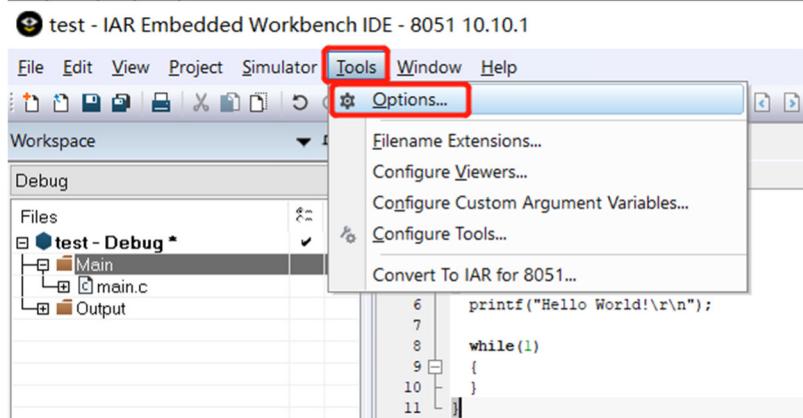
}
```

After entering the code, it will be as shown in the picture.

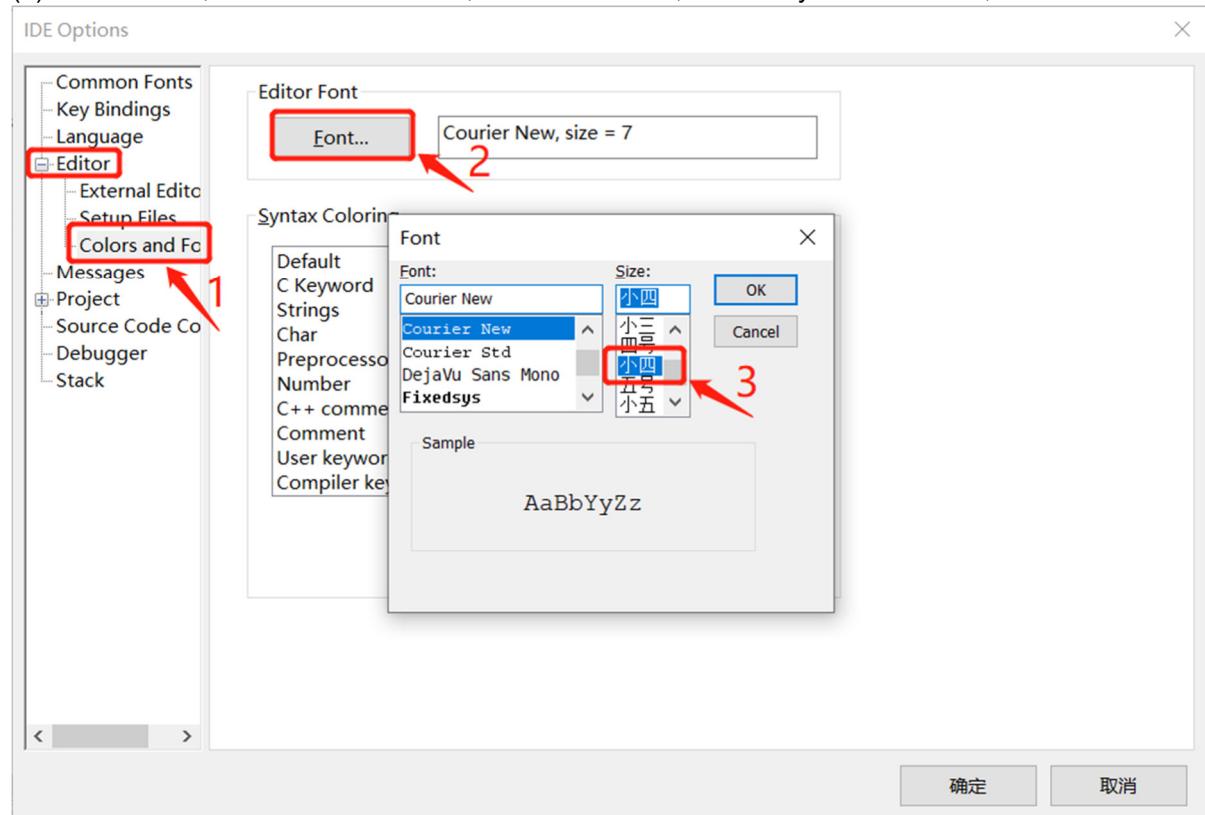


4. Set font and line number

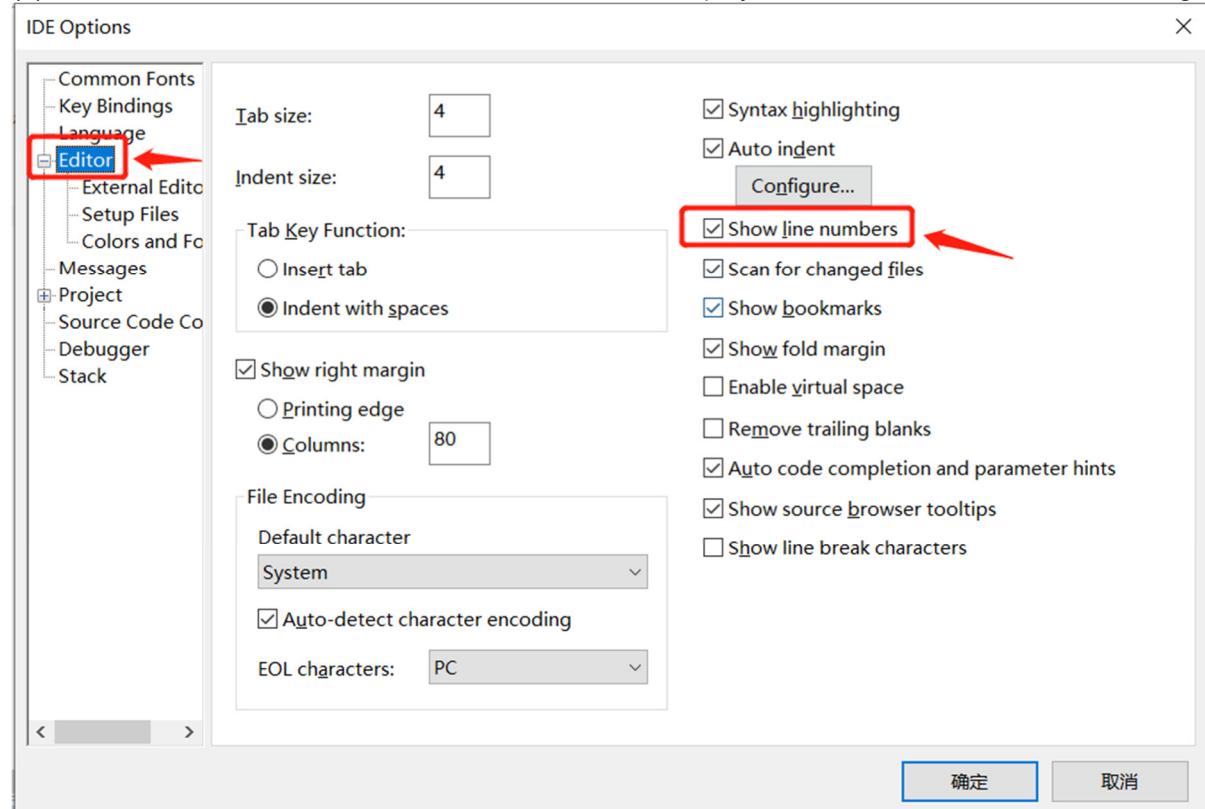
(1) You can set the code font and line number according to your personal preferences. Select Tools and Options in turn to open the settings, as shown in the figure below.



(2) Select Editor, then Colors and Font, then click Font..., and finally select the font, as shown in the figure.

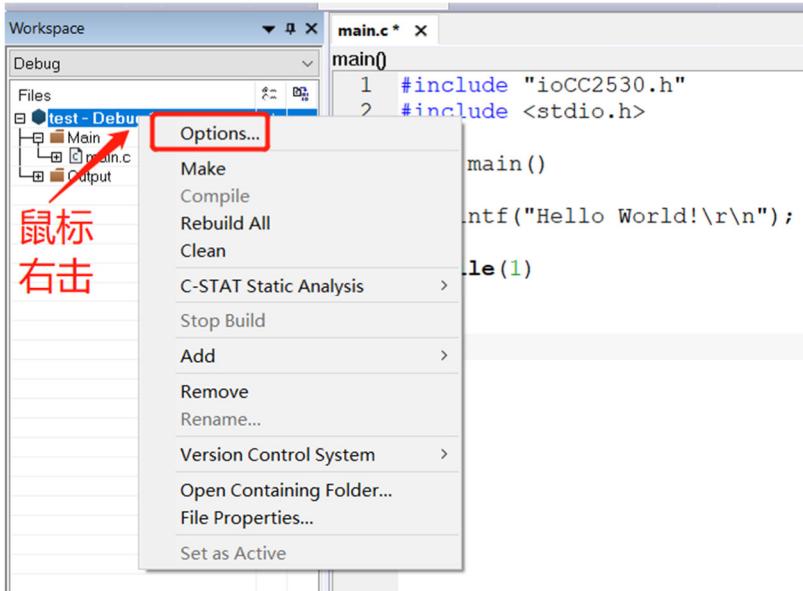


(3) Select Editor, and then check Show line numbers to display the line numbers, as shown in the figure.



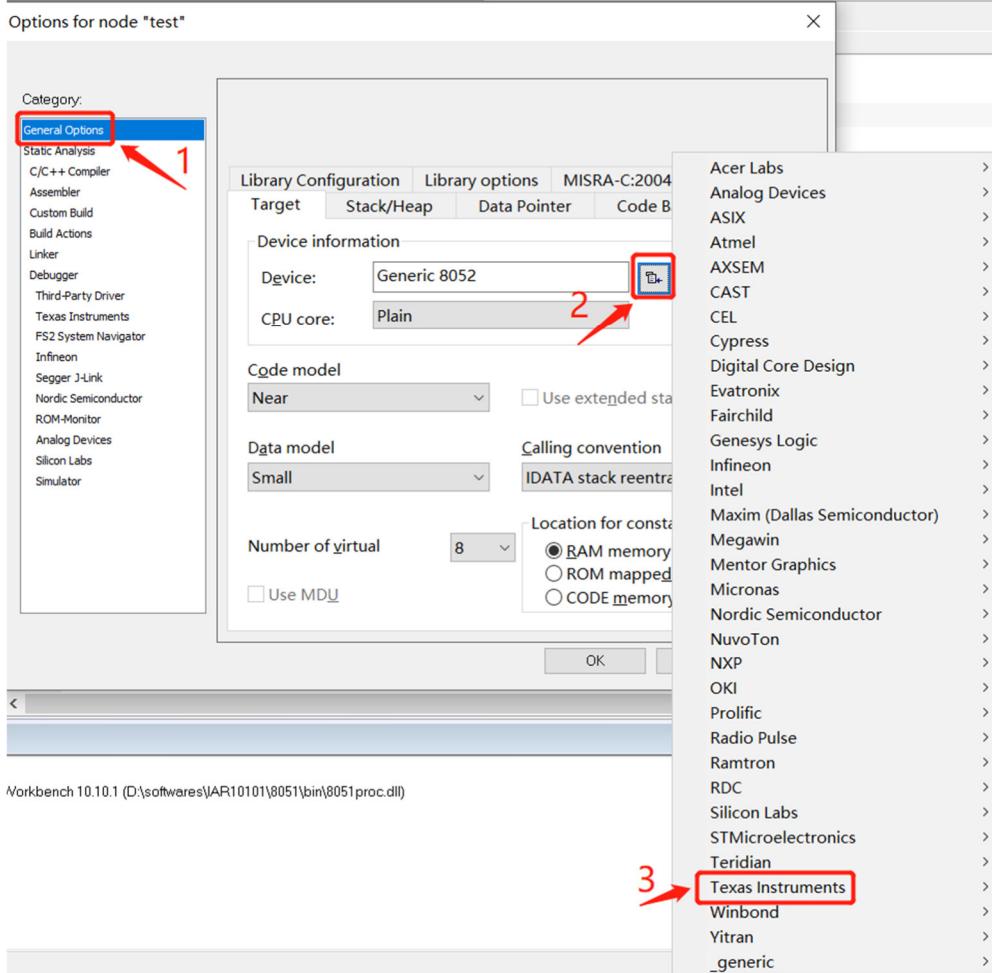
5. Project Configuration

After adding the source code, you need to configure the project before compiling. Right-click the project name and select Options..., as shown in the figure.

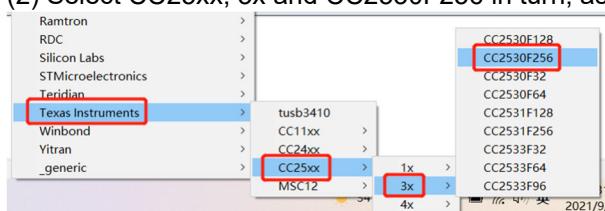


Basic configuration

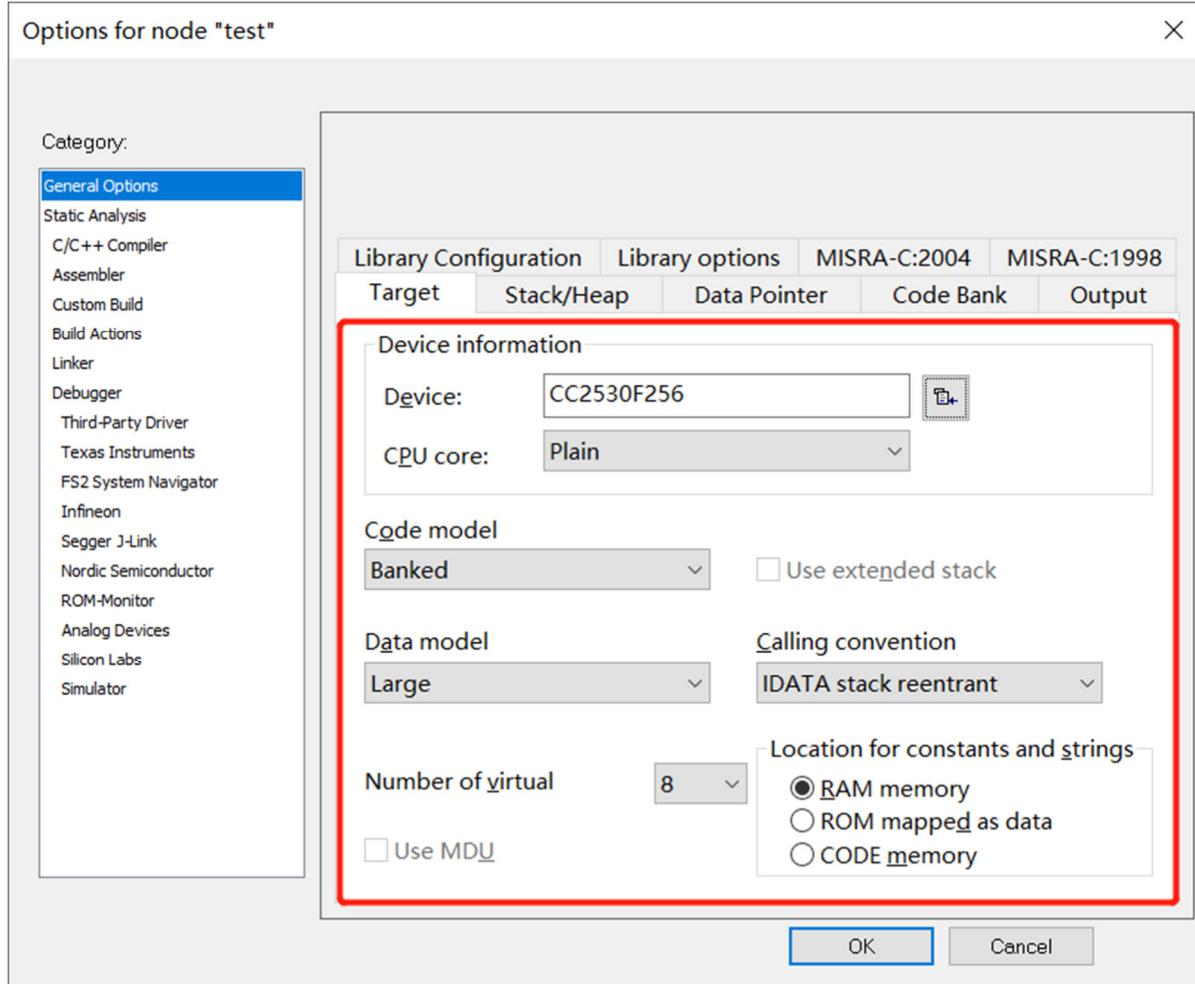
(1) can be configured according to the ZigBee chip model used. Here we take CC2530F256 as an example to explain the configuration process. Select General Options, Target, Device and Texas Instruments in sequence, as shown in the figure.



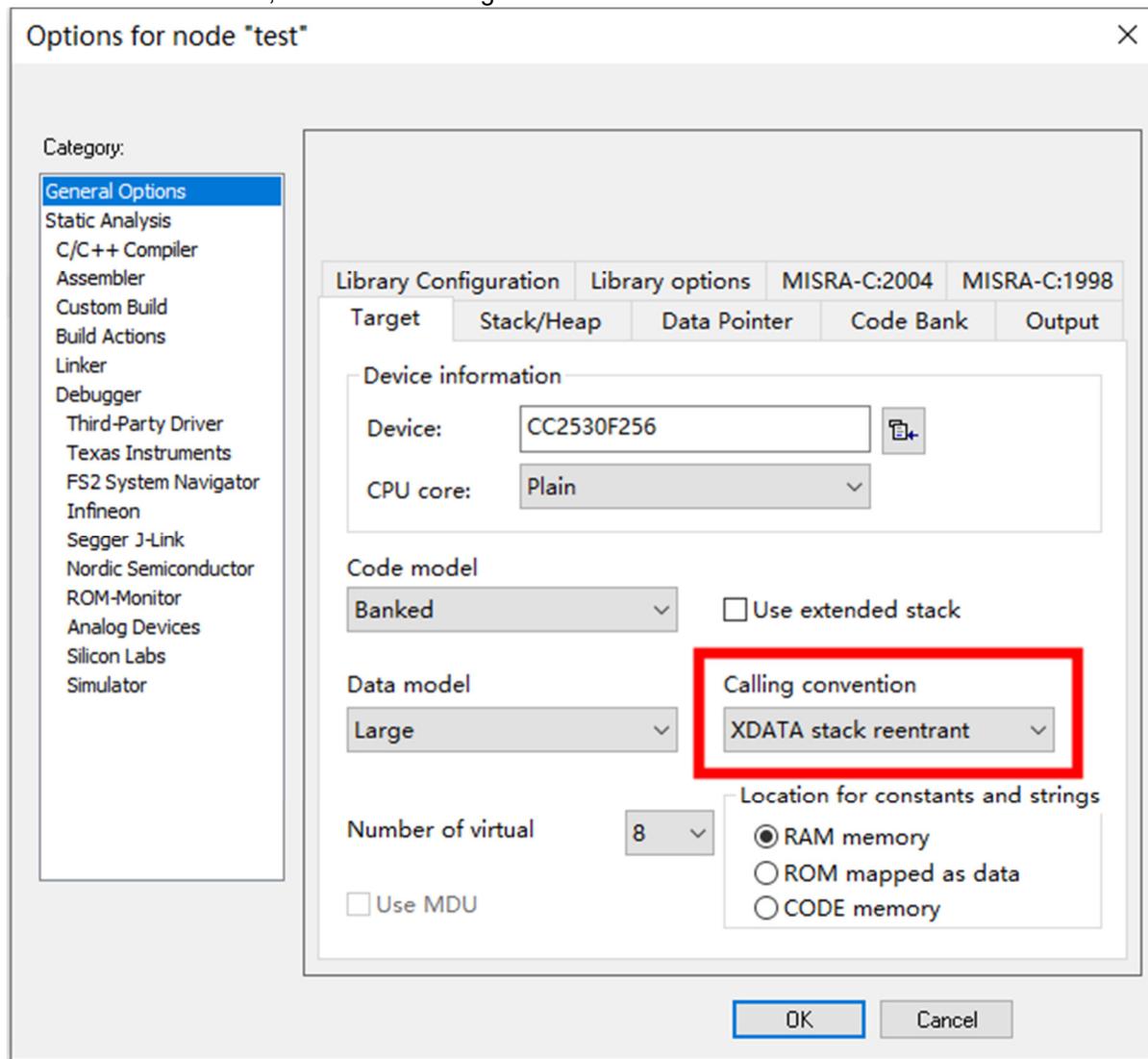
(2) Select CC25xx, 3x and CC2530F256 in turn, as shown in the figure.



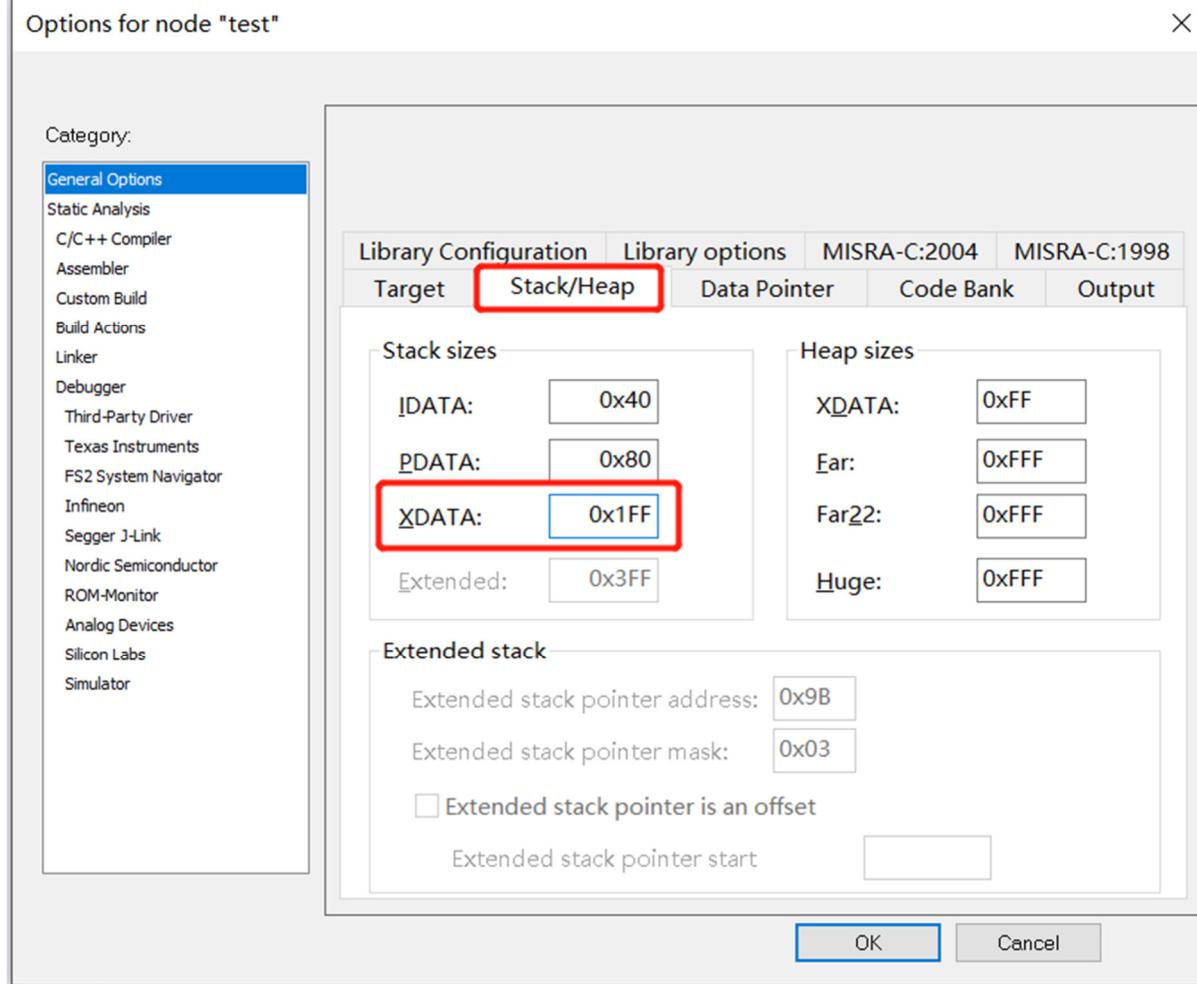
(3) After confirming that there are no problems with the configuration, click OK, as shown in the figure.



(4) In the Calling convention selection box, there may not be an option for IDATA stack reentrant. In this case, just change it to XDATA stack reentrant, as shown in the figure.

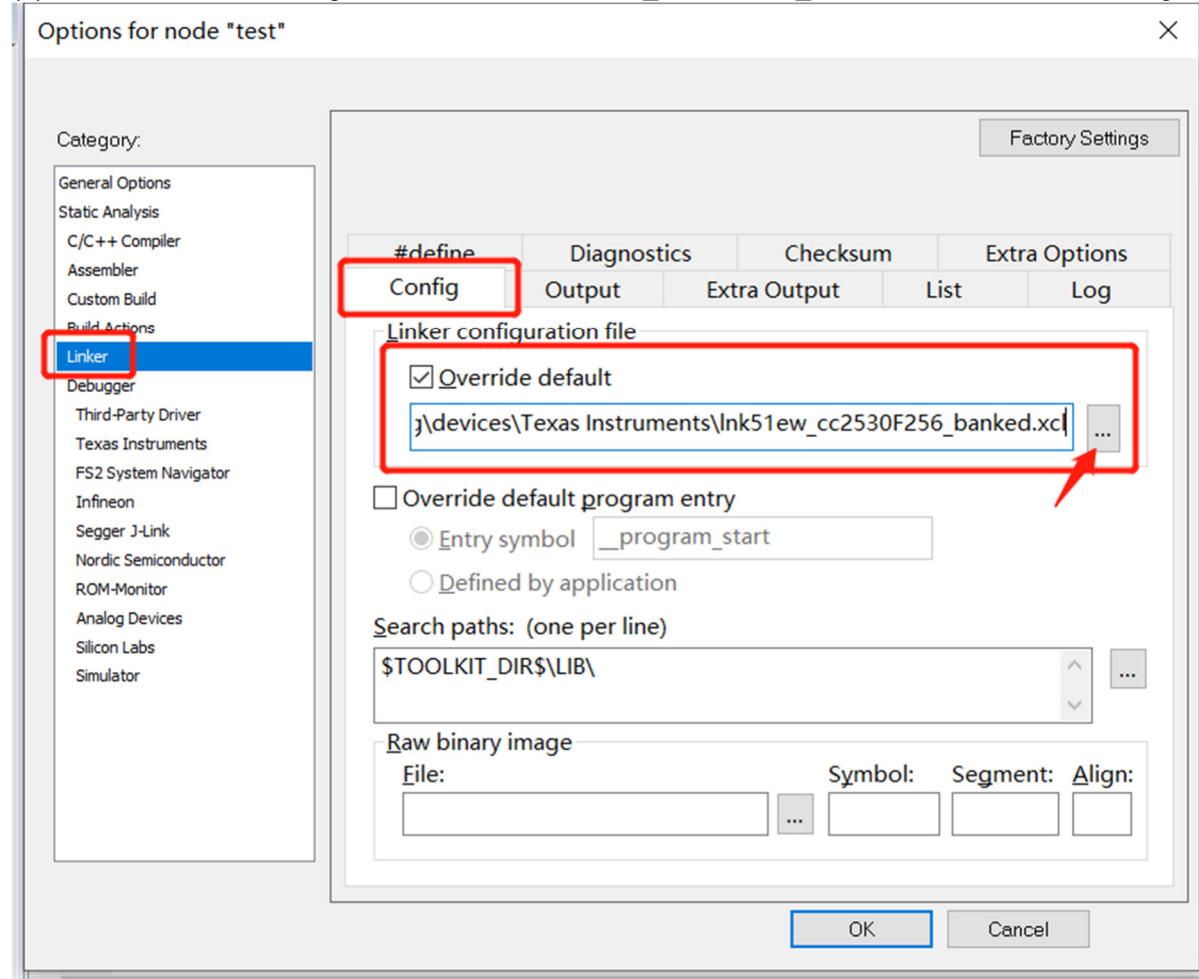


(5) Select Stack/Heap and configure the size of the data segment XDATA to 1FF, as shown in the figure.

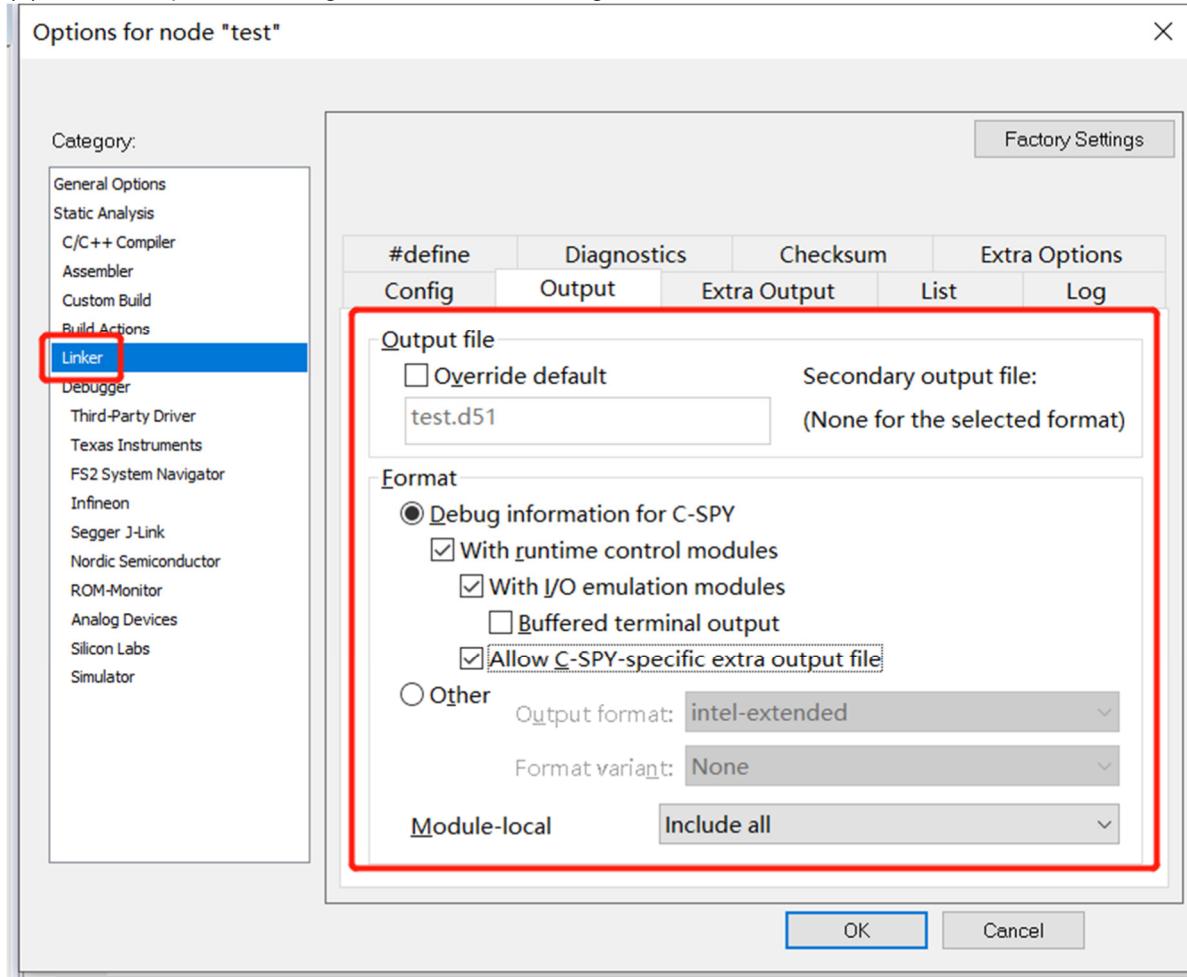


Link and Output Configuration

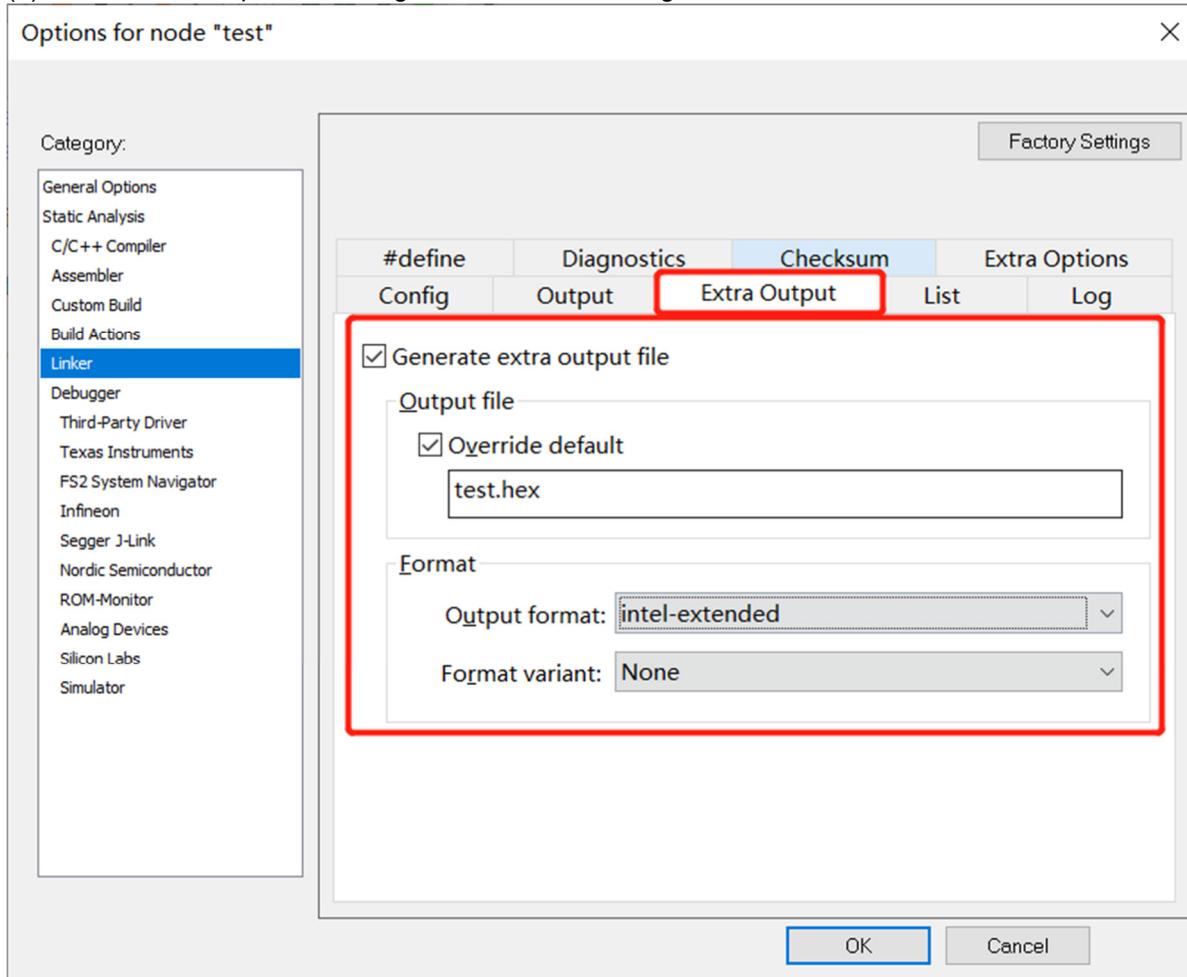
(1) Select Linker and Config, and then select `lnk51ew_cc2530f256_banked.xcl`, as shown in the figure.



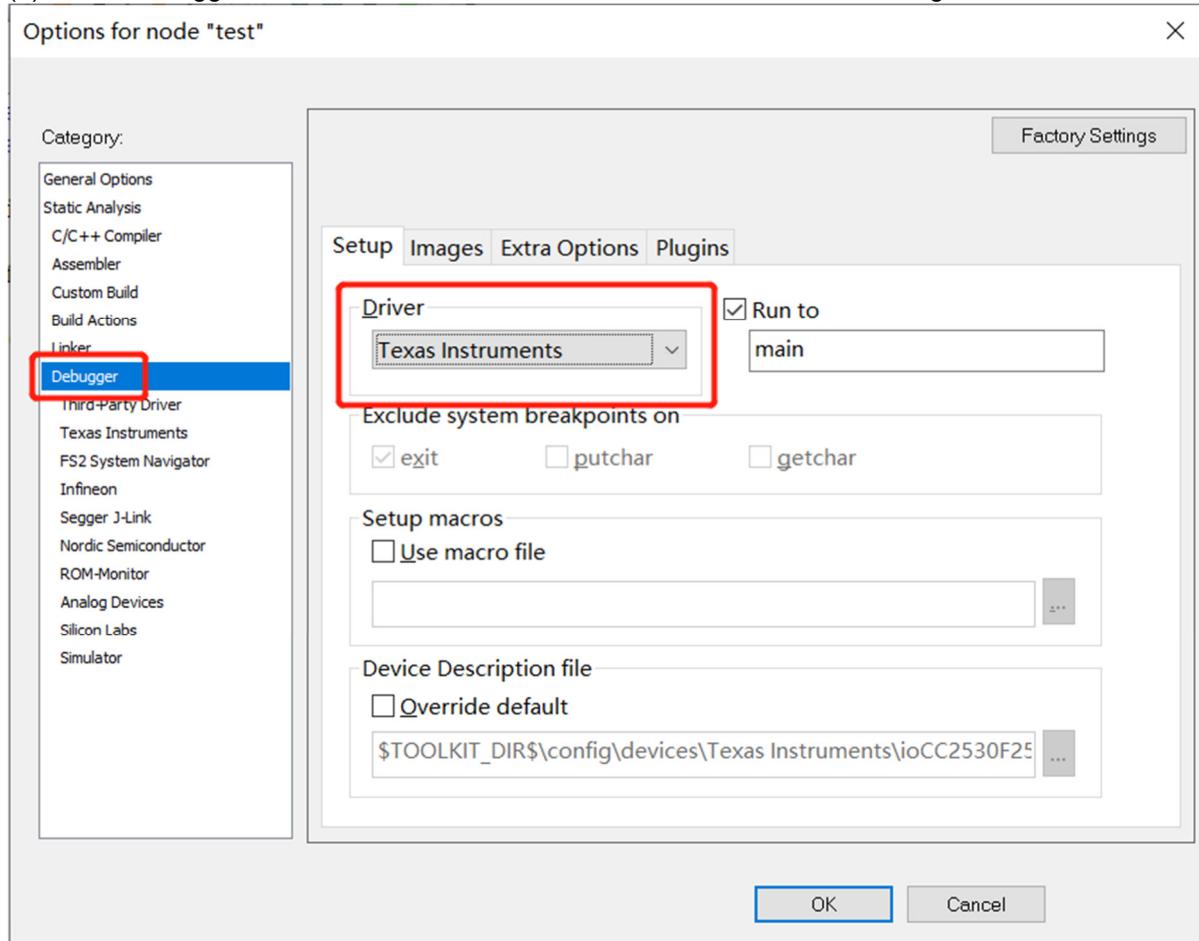
(2) Select Output and configure as shown in the figure.



(3) Select Extra Output and configure as shown in the figure.

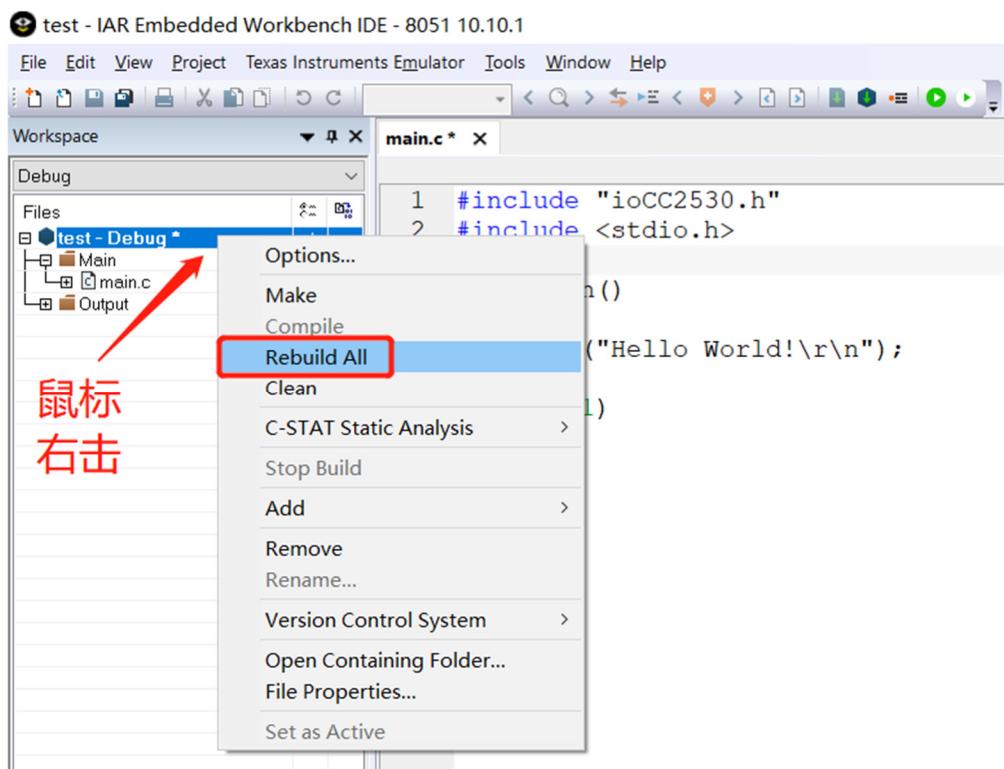


(4) Select Debugger and select Texas Instruments in Driver, as shown in the figure.



6. Compile and link the project

(1) After the configuration is complete, you can compile and link the project. Right-click the project name and select Rebuild All to recompile and link the entire project, as shown in the figure.



(2) You can view the compilation and linking results at the location shown in the figure.

The screenshot shows the IAR Embedded Workbench IDE interface. The top menu bar includes File, Edit, View, Project, Texas Instruments Emulator, Tools, Window, and Help. The workspace shows a project named "test - Debug" with files Main.c and main.c. The code editor window displays the following C code:

```
1 #include "ioCC2530.h"
2 #include <stdio.h>
3
4 void main()
5 {
6     printf("Hello World!\r\n");
7
8     while(1)
9     {
10    }
11 }
```

The bottom pane shows the build log:

```
Messages
Updating build tree...
0 file(s) deleted.
Updating build tree...
main.c
Linking
Total number of errors: 0
Total number of warnings: 0
```

A red box highlights the message "Total number of errors: 0".

5.1.3. Program download and simulation

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=4>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Hardware Configuration

(1) Connect the ZigBee emulator to the development board and computer respectively, and then you can start program downloading and simulation.

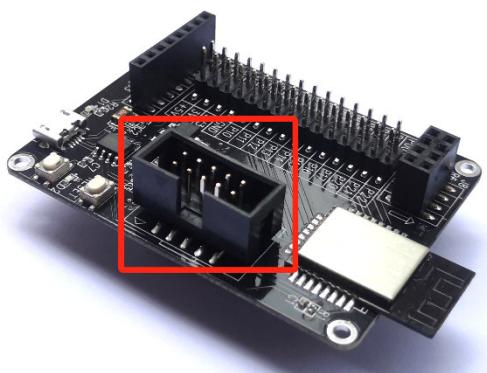
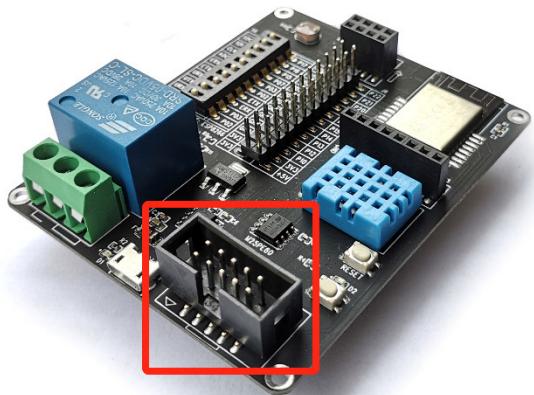
(2) The ZigBee simulation kit includes a ZigBee simulator, a USB cable, and a serial port cable, as shown in the figure below.



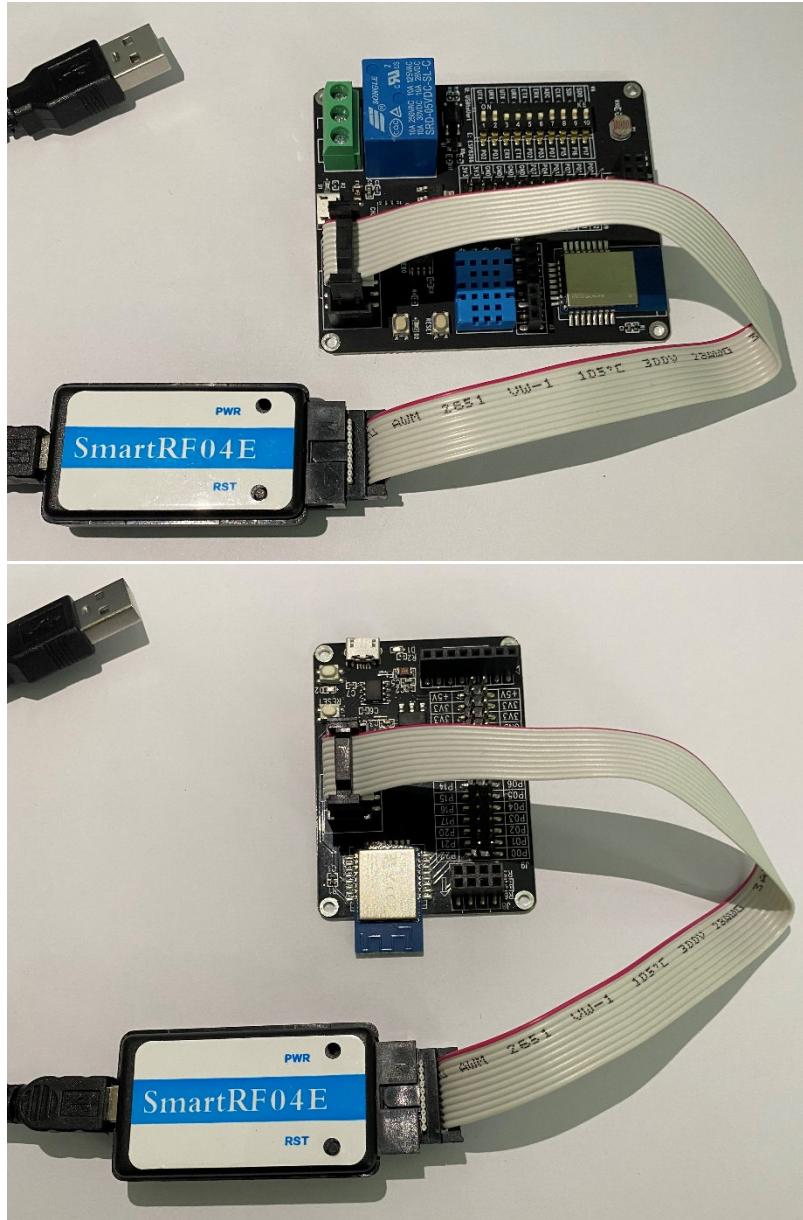
(3) The wiring method is shown in the figure below.



(4) ZigBee development boards are equipped with an emulator interface, as shown in the figure.



(5) Connect the emulator to the development board as shown in the figure.



(6) Then, connect the emulator's USB port to the computer's USB port.

Note: Every time you connect the USB cable to the computer, you need to press the reset button on the emulator.

Program download and simulation

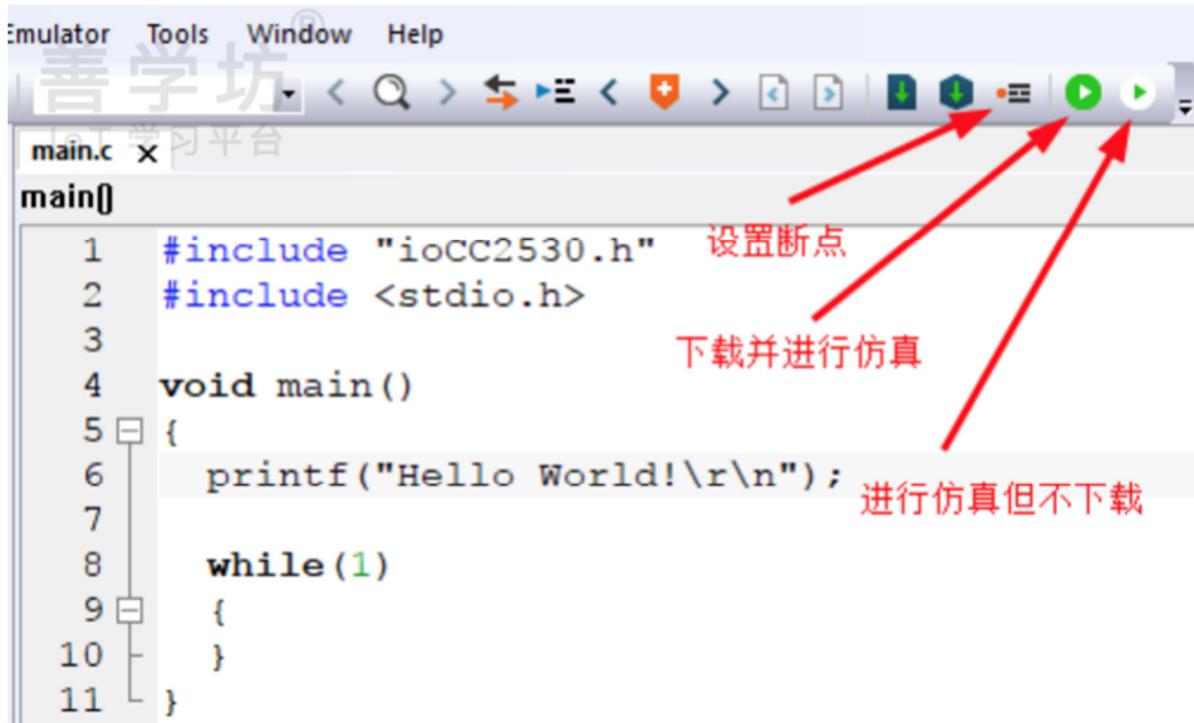
Note: Before studying this lesson, you need to install the supporting software tools, especially the emulator driver.

Basic Operations

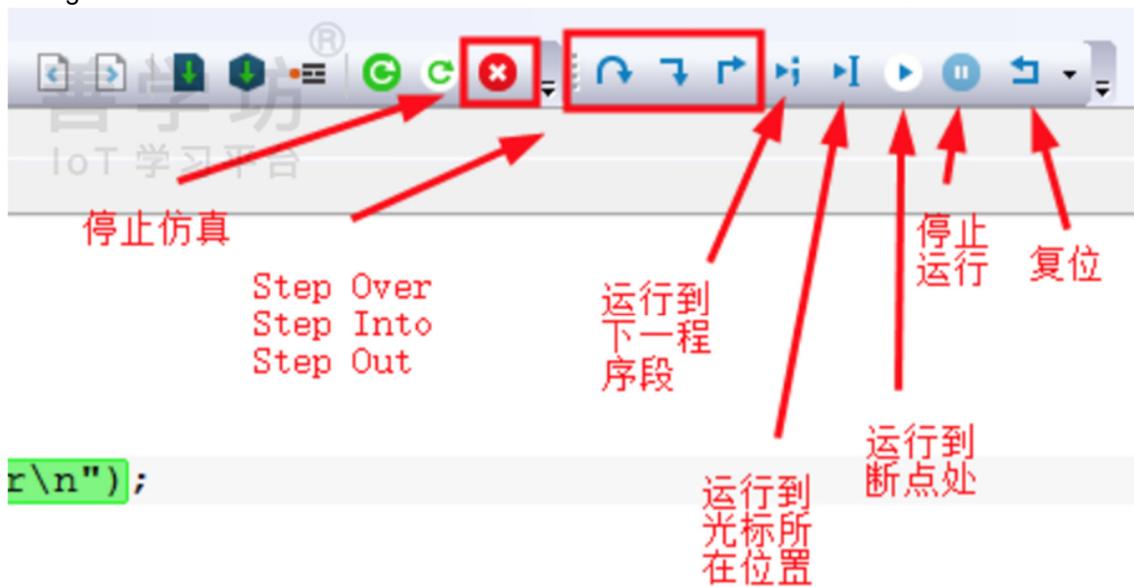
(1) As shown in the figure, we can find two simulation modes in IAR:

- One is to download the program to the chip

- The other type does not download the program to the chip, but only performs online debugging



(2) Click the "Download and Simulate" button, IAR will start downloading and simulating, and switch to the interface shown in the figure.

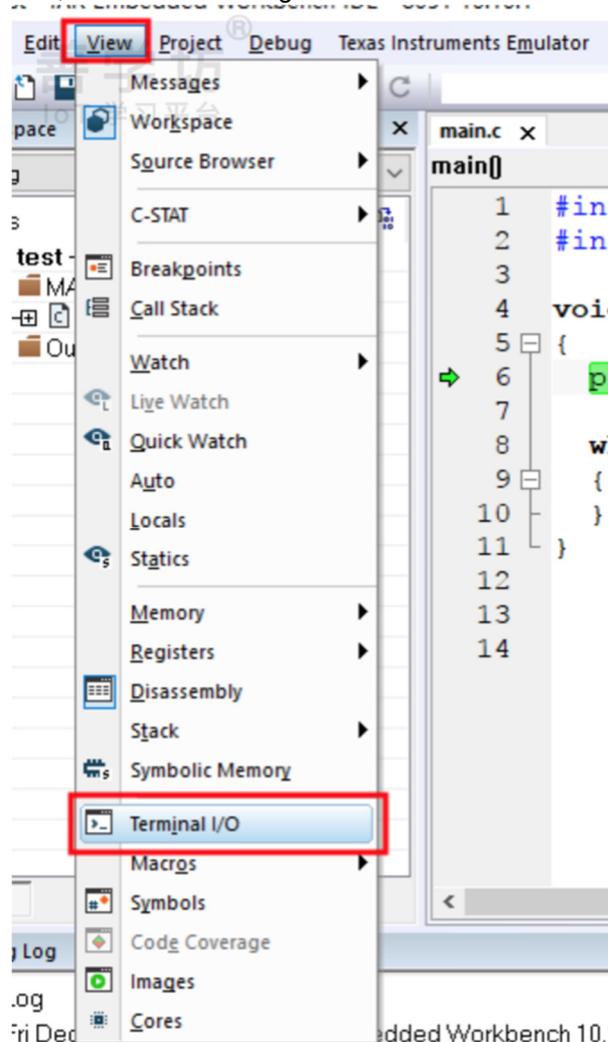


(3) During single-step code execution, in addition to being able to run directly to the breakpoint, the cursor position, or the next program segment, there are also the following modes to choose from:

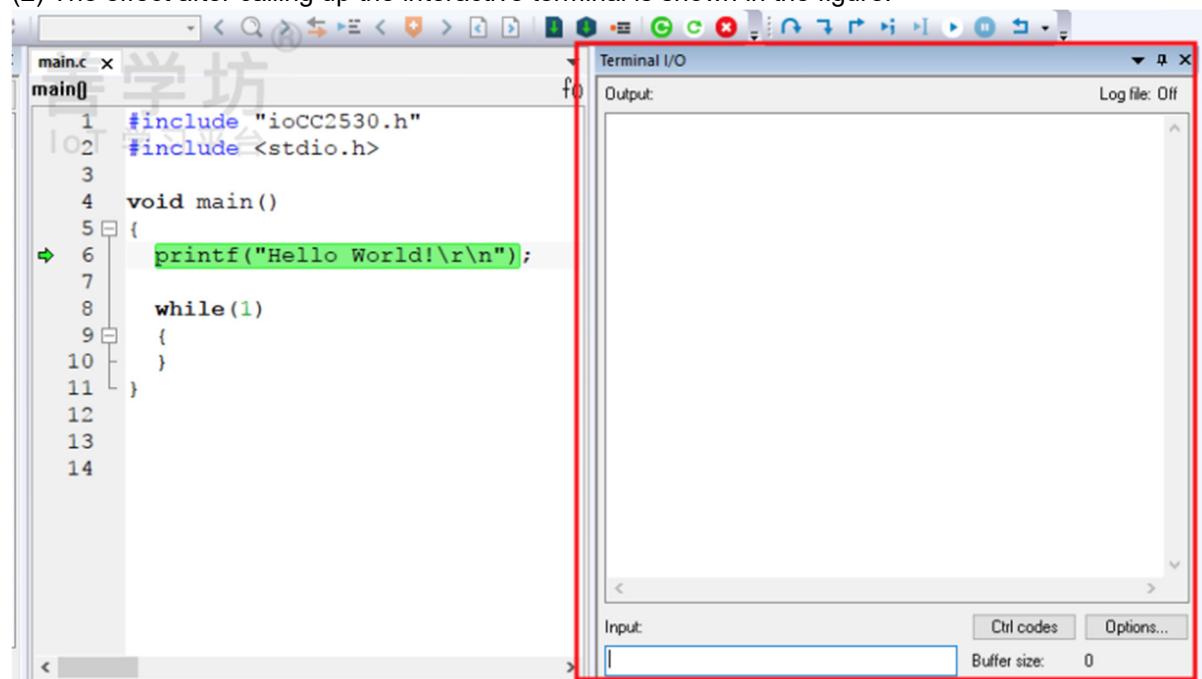
- step over**: When the execution reaches the function call, it will not enter the function, but stop and wait after the entire function is executed, that is, the entire function is executed as one step;
- step into**: When the execution reaches the function call, it will enter the function and continue to execute single-step;
- step out**: When single-stepping into a function, use step out to directly execute the current function and return to the previous level.

Using the interactive terminal

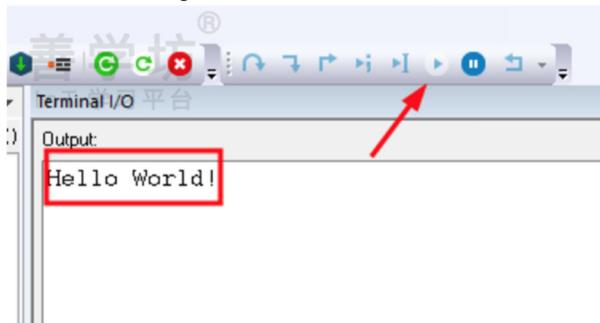
(1) Next, you can call up the interactive terminal to view the function information output by printf. Click View and Terminal I/O in turn, as shown in the figure.



(2) The effect after calling up the interactive terminal is shown in the figure.



(3) Click the "GO (F5)" button to run the program code. You can see that the interactive terminal outputs "Hello World!", as shown in the figure.



(4) At this point, you have learned the entire simulation process. However, the simulation program is very powerful and has many functions. You need to learn and use it in subsequent courses to finally achieve the goal of mastering it.

5.1.4. Firmware Burning

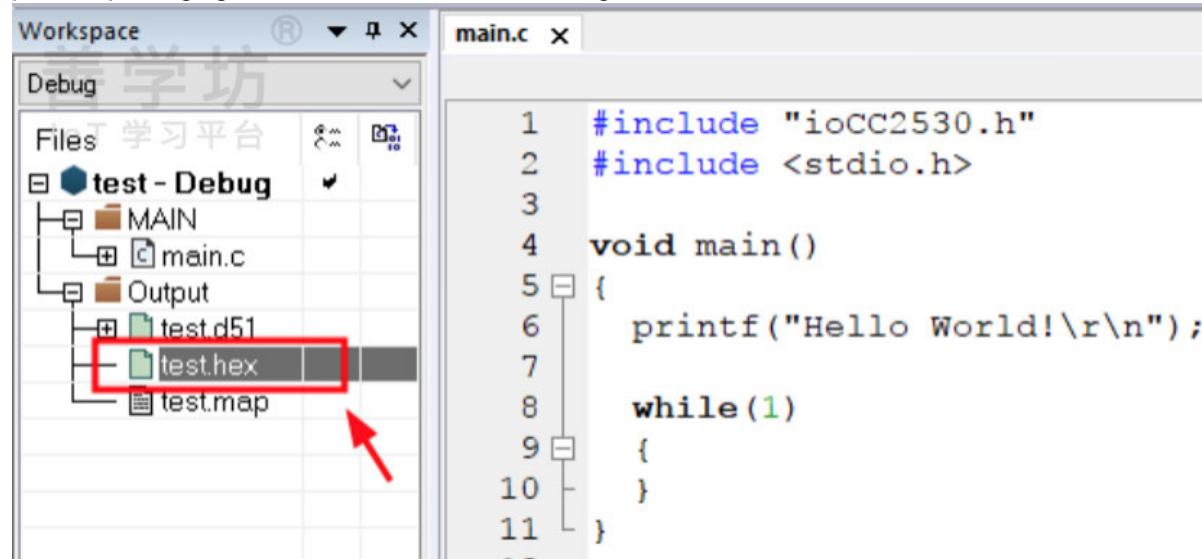
Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=5>

Technical support instructions:

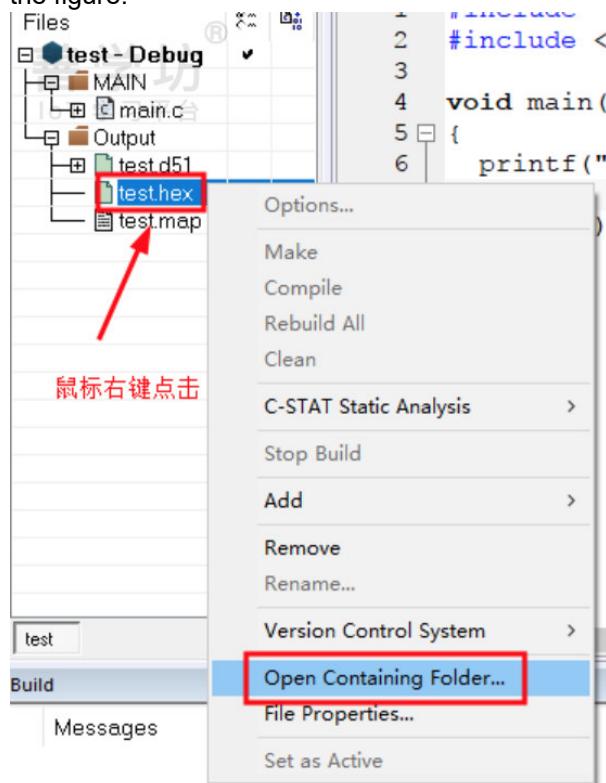
1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Firmware Introduction

(1) After configuring and compiling the link project, IAR will generate a hex file. This hex file can be used as the firmware for product packaging and release, as shown in the figure.



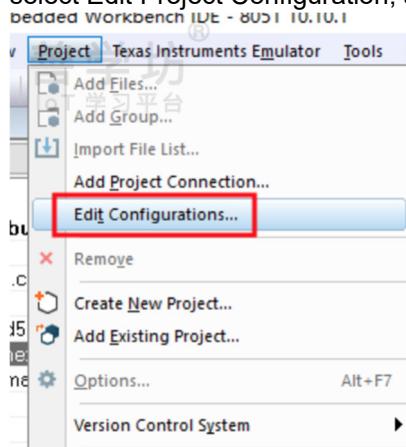
(2) Right-click the test.hex file and select Open Containing Folder... to view the directory where the file is located, as shown in the figure.



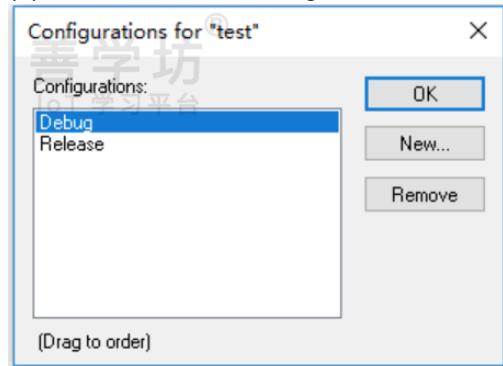
(3) In the actual development process, it is generally necessary to generate two versions of firmware:

- One is the Debug version for debugging
- One is the Release version of the product officially released

When creating a project, IAR also defaults to the relevant configuration. Click Project and Edit Configurations... in turn, and select Edit Project Configuration, as shown in the figure.



(4) You can see the configuration window as shown in the figure.



Firmware burning

You can burn the firmware to the development board and observe the running results of the firmware on the development board. The steps for firmware burning are as follows:

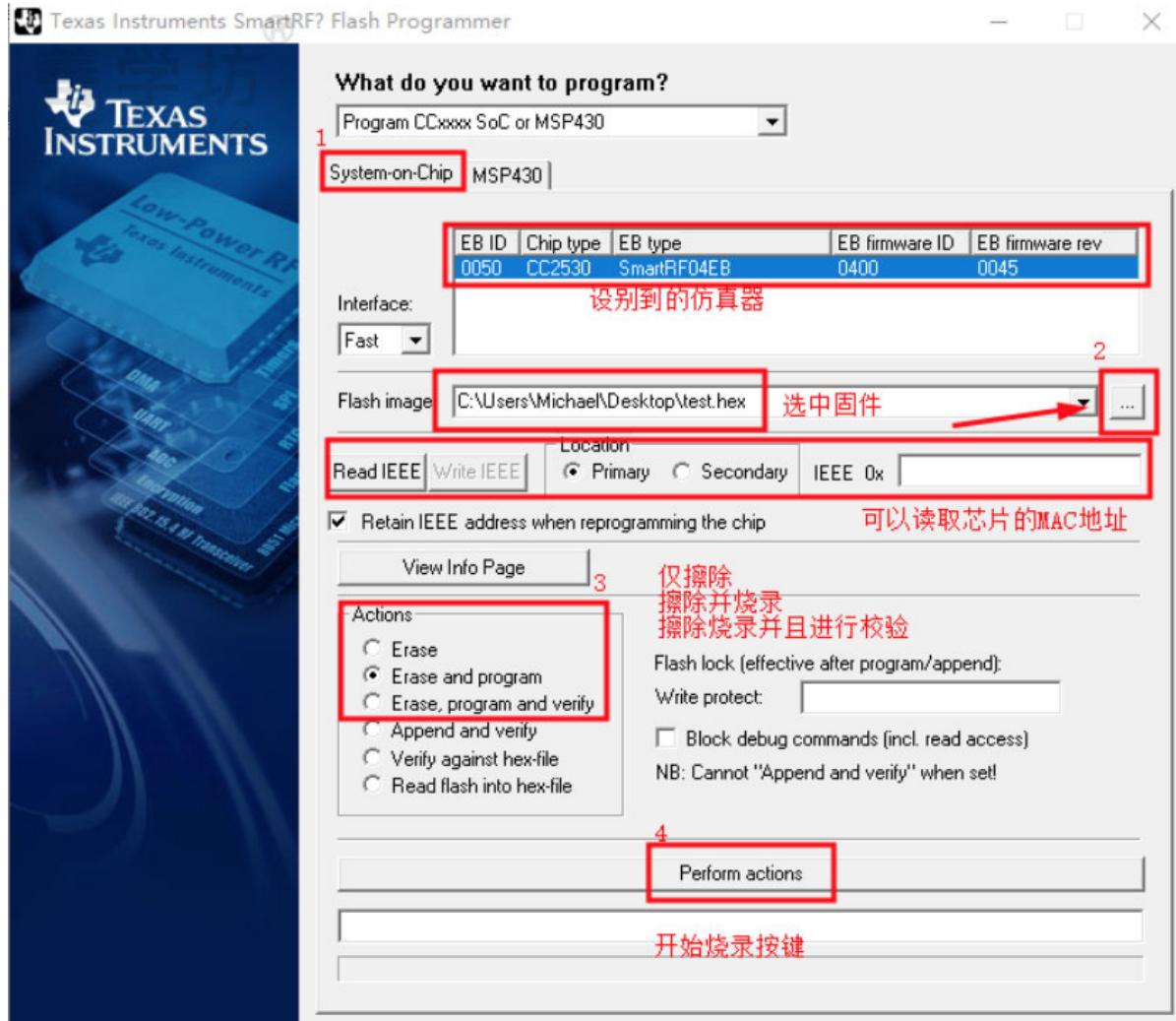
(1) Make sure that SmartRF Flash Programmer has been installed. For installation instructions, please refer to "Part 1: Preparation" → "SmartRF Flash Programmer Download and Installation"

(2) Connect the development board to the computer through the emulator. Refer to the previous lesson for the connection method.

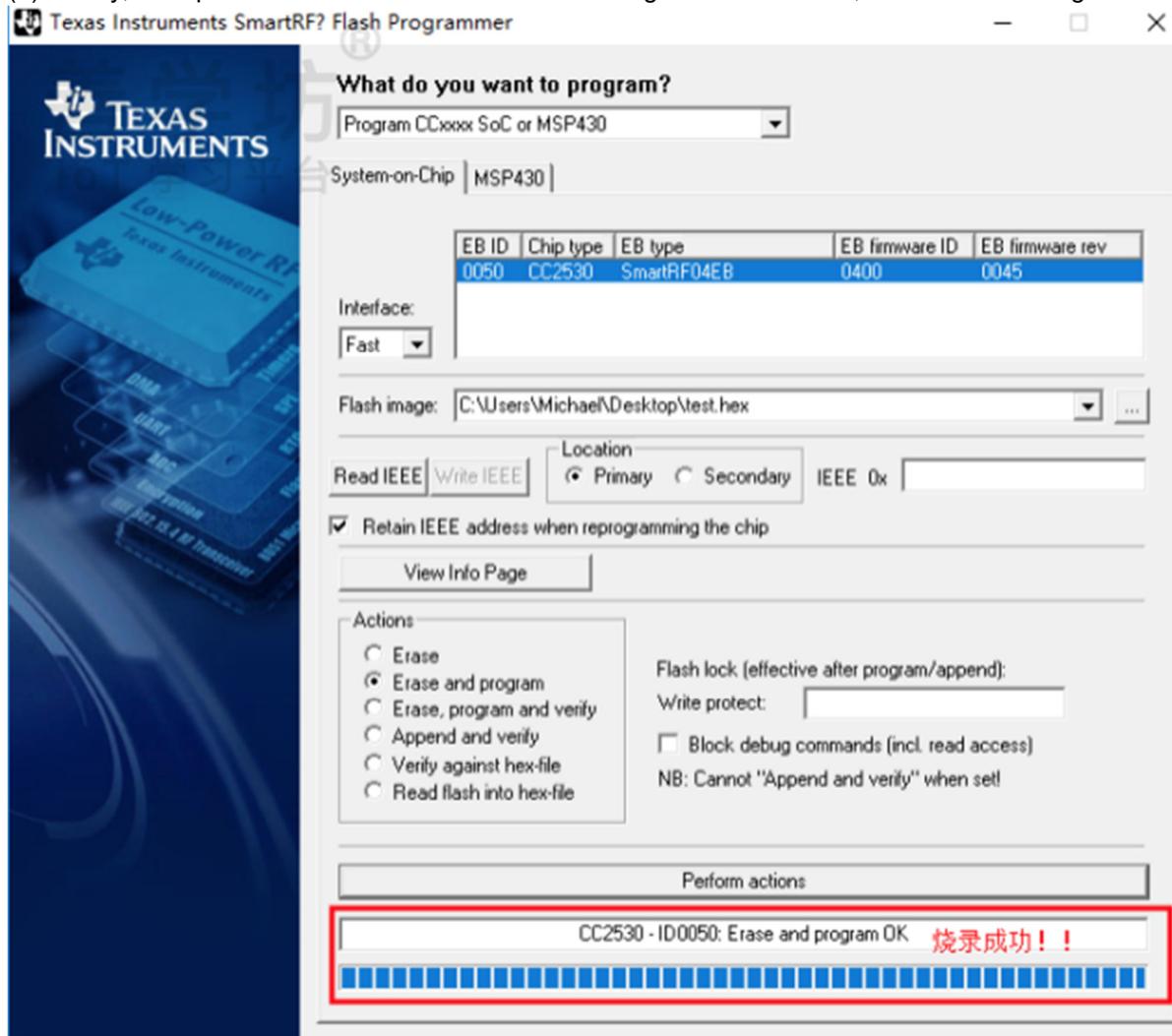
(3) Press the reset button of the emulator.



(4) Open SmartRF Flash Programmer, click System-on-Chip, and you can see the recognized emulator. Then select the firmware to be burned and select Erase and program, as shown in the figure.



(5) Finally, click perform actions and wait for the burning to be successful, as shown in the figure.



5.2. Chapter 2: GPIO Experiment

5.2.1 Multi-project management foundation

5.2.2 GPIO output experiment - LED control

5.2.3 GPIO input experiment - mechanical buttons

5.2.4 GPIO input and output general configuration experiment

5.2.5 GPIO external interrupt experiment

5.2.1. Multi-project management foundation

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=6>

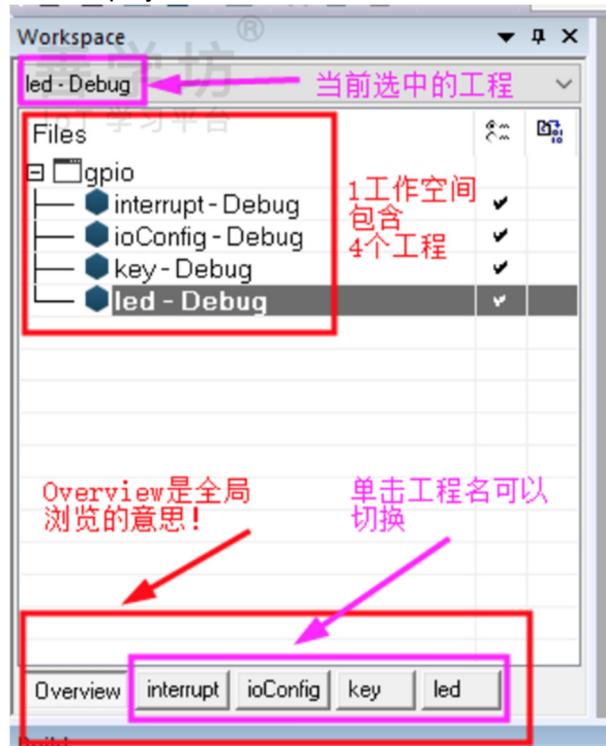
Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

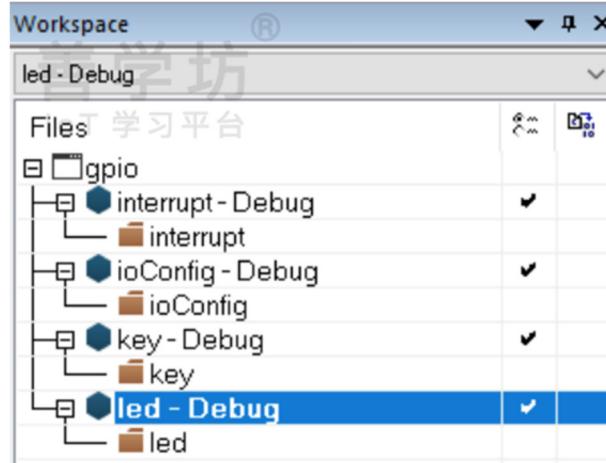
In the last lesson, we explained how to create a new workspace and project. In fact, a workspace can contain multiple projects.

Multi-project management foundation

(1) Create a new workspace according to the method explained in the previous lesson, and then create 4 new projects in the workspace, namely led, key, ioConfig and interrupt. After the creation is completed, it will be shown in the figure. Click the Overview tab to browse the entire workspace, and you can see that one space contains 4 projects. Right-click led-Debug to select the project, and the currently selected project will also be displayed in the upper left corner. You can also click the tab with the project name at the bottom to switch to the corresponding project.



(2) Create a corresponding group for each project. The group will look like the following figure after creation.

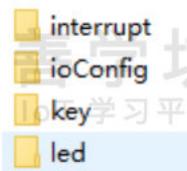


(3) Create a new code folder in the workspace directory to store the source code, as shown in the figure. The workspace directory contains a workspace file ending with.eww.

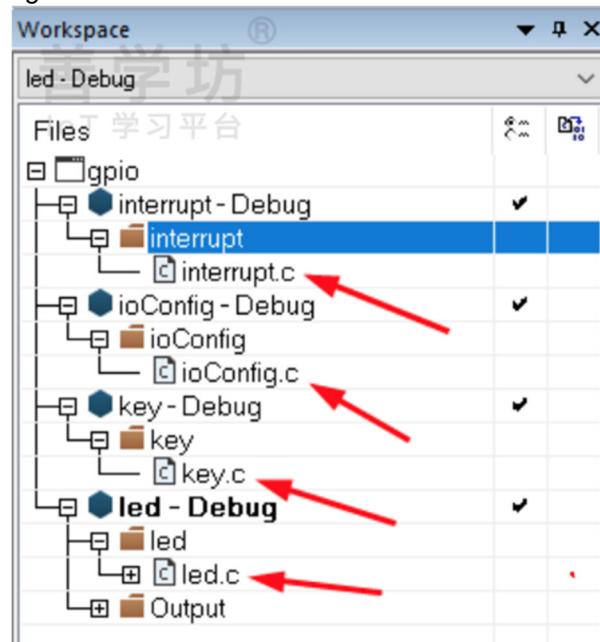
名称	修改日期	类型
code	2017/12/11 10:48	文件夹
Debug	2017/12/11 10:57	文件夹
settings	2017/12/11 10:47	文件夹
gpio.eww	2017/12/11 10:48	IAR IDE Worksp...
interrupt.dep	2017/12/11 10:48	DEP 文件
interrupt.ewd	2017/12/11 10:42	EWD 文件
interrupt.ewp	2017/12/11 10:42	EWP 文件
ioConfig.dep	2017/12/11 10:48	DEP 文件
ioConfig.ewd	2017/12/11 10:48	EWD 文件
ioConfig.ewp	2017/12/11 10:48	EWP 文件
key.dep	2017/12/11 10:48	DEP 文件
key.ewd	2017/12/11 10:40	EWD 文件
key.ewp	2017/12/11 10:40	EWP 文件
led.dep	2017/12/11 10:48	DEP 文件
led.ewd	2017/12/11 10:39	EWD 文件
led.ewp	2017/12/11 10:39	EWP 文件

工作空间文件

(4) Create four subfolders corresponding to the project names in the code folder to store the corresponding source code, as shown in the figure.



(5) Create a new source file ending with.c in each folder and add it to the project. After adding, it will be shown as shown in the figure below.



(6) Next, the reader needs to configure these projects according to the methods explained in the previous lesson. After the configuration is completed, development can begin.

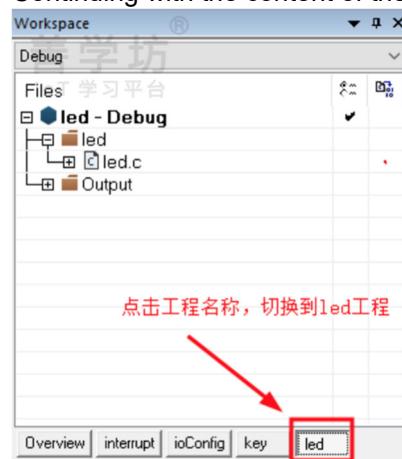
5.2.2. GPIO output experiment - LED control

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=7>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

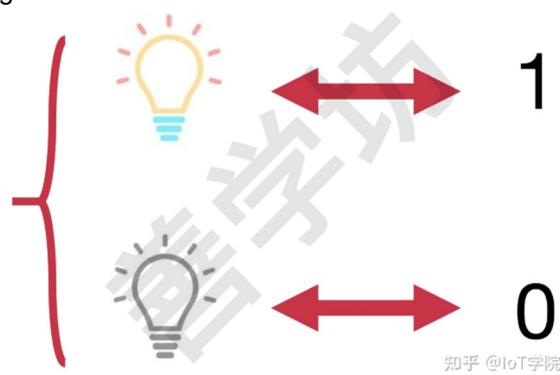
Continuing with the content of the previous lesson, we switch to the LED project, as shown in the figure.



The logic layer and circuit layer of a computer

Logical Layer

In the computer logic layer, binary numbers (0 or 1) are generally used to control objects or represent the state of objects. For example, 1 can be used to represent turning on the light, and 0 can be used to represent turning off the light, as shown in the figure.



Circuit Layer

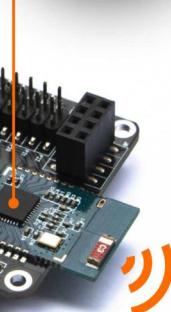
In the computer circuit layer, a high level (3.3 V) is generally used to represent a logical 1, and a low level (0 V) is used to represent a logical 0. The principle of the logic and circuit layers may be difficult for beginners to understand, but you can continue to study and gradually understand the meaning.

Chip pins

The pins of a chip can be understood as wires leading out from the chip and used to connect external components.

善学坊™
IoT 低代码开发

ZigBee Mini板



小白
也能学会

2023 新款

The pins of a chip can generally be divided into power pins, control pins, and GPIO. Power pins are generally used to power the chip; control pins are generally related to hardware design, which we can temporarily ignore.

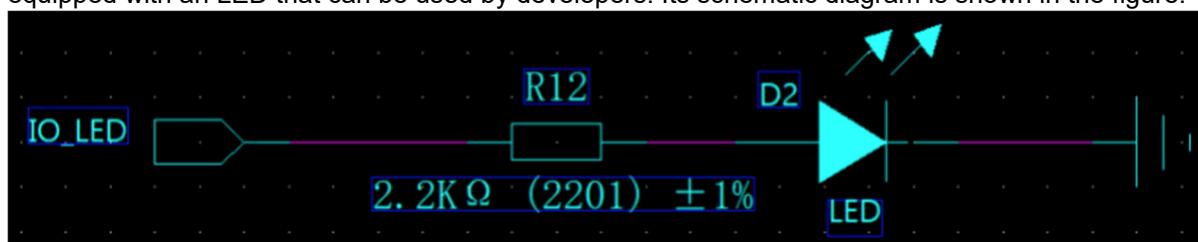
GPIO

GPIO stands for General-Purpose Input/Output, which means general-purpose input/output in Chinese. It is a pin we often use. Generally speaking, a chip will have multiple GPIOs, and each GPIO has two working modes, namely output signal mode and receiving signal mode.

In output signal mode, we can control the GPIO to output high level (3.3 v) or low level (0 v) through code. In input signal mode, we can detect whether the GPIO is in high level or low level state through code.

Introduction to LED

LED is the abbreviation of Light Emitting Diode, which means "light emitting diode" in Chinese. It is a component that can convert electricity into light. Generally speaking, LED is a kind of lamp we often say. The ZigBee development board is equipped with an LED that can be used by developers. Its schematic diagram is shown in the figure.



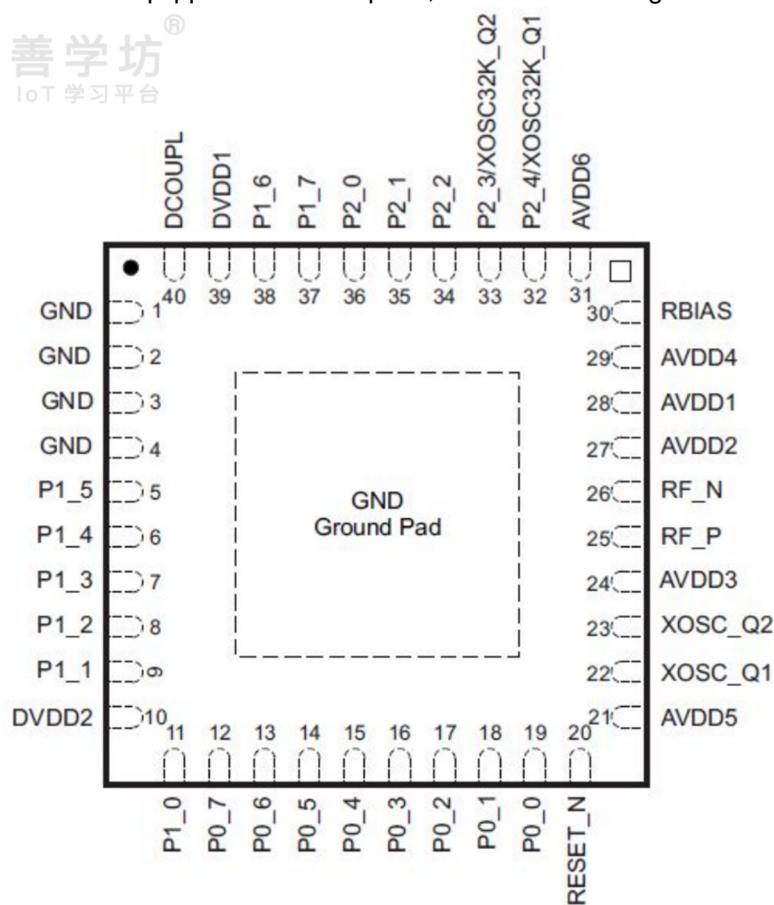
D2 represents an LED, and its right end is grounded (GND), so the voltage at the right end is 0v. The left end of the LED is connected to R12 and IO_LED in sequence. IO_LED is an IO port of CC2530, which can output high level (3.3v) or low level (0v). R12 is a voltage-stabilizing resistor, which is used to prevent the circuit from burning out the LED due to excessive voltage.

- When IO_LED outputs a high level, the voltage at the left end of the LED is 3.3v, and the voltage at the right end is 0v, forming a voltage difference of 3.3v between the left and right ends, so the LED is lit;
- On the contrary, when IO_LED outputs a low level, the voltages at the left and right ends of the LED are both 0v, so the LED is turned off.

Note: If you cannot understand this circuit diagram, it may indicate that you lack knowledge about hardware principles and need to supplement it first.

CC2530 Pin Introduction

CC2530 is equipped with 40 IO ports, as shown in the figure.



Among them, P0_0~P0_7 belong to P0 port, P1_0~P1_7 belong to P1 port, and P2_0~P2_5 belong to P2 port. These IO ports can be used programmatically. The following will explain their usage with examples.

Check out TI's official data sheet

(1) The CC2530 data sheet can be found in the course resources, as shown in the figure.

A screenshot of a course resource page. The URL in the address bar is: « 学习资料 > ZigBee > 《Zigbee3.0开发指南》配套代码（公开部分） > 考外资料参考 > CC2530数据手册. The page displays a table of files:

名称	修改日期	类型	大小
cc2530-datasheet.pdf	2023/3/17 11:04	Microsoft Edg...	2,345 KB
CC2530中文数据手册完全版.pdf	2023/3/17 11:04	Microsoft Edg...	9,207 KB

A red arrow points to the file "CC2530中文数据手册完全版.pdf".

(2) The table of contents of this data sheet is as follows.

- ▶ 1 简介
- ▶ 2 8051CPU
- ▶ 3 调试接口
- ▶ 4 电源管理和时钟
- ▶ 5 复位
- ▶ 6 闪存控制器
- ▶ 7 I/O端口
- ▶ 8 DMA控制器
- ▶ 9 定时器1 (16位定时器)
- ▶ 10 定时器3和定时器4 (8位定时器)
- ▶ 11 睡眠定时器
- ▶ 12 ADC
- ▶ 13 随机数发生器
- ▶ 14 AES协处理器
- ▶ 15 看门狗定时器
- ▶ 16 USART
- ▶ 17 USB控制器
- ▶ 18 定时器2 (MAC定时器1)
- ▶ 19 无线电
- 20 稳压器
- ▶ 21 可用的软件

(3) Expand the "I/O Port" chapter, and the table of contents is as follows.

- ▼ 7 I/O端口
 - 7.1 未使用的I/O引脚
 - 7.2 低I/O电压
 - 7.3 通用I/O
 - 7.4 通用I/O中断
 - 7.5 通用I/ODMA
 - ▶ 7.6 外设I/O
 - 7.7 调试接口
 - 7.8 32kHzXOSC输入
 - 7.9 无线测试输出信号
 - 7.10 掉电信号MUX(PMUX)
 - 7.11 I/O引脚

(4) This data sheet can be used as a reference book, allowing us to check the details of CC2530 at any time. For example, when using GPIO, you can find the "I/O Port" section, and when using the timer, you can find the timer section.

How to use the IO port

Brief introduction of related registers

You can use the IO port by configuring the relevant registers, for example, configuring the specified IO port to output signal mode and controlling its output high/low level. The registers related to IO port configuration in CC2530 are as follows.

- P0: Port 0 Configuration Register
- P1: Port 1 Configuration Register
- P2: Port 2 Configuration Register
- PERCFG: Peripheral Control Register
- APCFG: Analog Peripheral I/O Configuration Register
- P0SEL: Port 0 function selection register
- P1SEL: Port 1 function selection register
- P2SEL: Port 2 function selection register
- P0DIR: Port 0 Direction Register
- P1DIR: Port 1 Direction Register
- P2DIR: Port 2 Direction Register
- P0INP: Port 0 Input Mode Register
- P1INP: Port 1 input mode register
- P2INP: Port 2 input mode register
- P0IFG: Port 0 interrupt status flag register
- P1IFG: Port 1 interrupt status flag register
- P2IFG: Port 2 interrupt status flag register
- PICTL: Interrupt Edge Register
- P0IEN: Port 0 interrupt mask register
- P1IEN: Port 1 interrupt mask register
- P2IEN: Port 2 interrupt mask register
- PMUX: Power-down signal Mux register
- OBSSEL0: Observation Output Control Register 0
- OBSSEL1: Observation Output Control Register 1
- OBSSEL2: Observation Output Control Register 2
- OBSSEL3: Observation Output Control Register 3
- OBSSEL4: Observation Output Control Register 4
- OBSSEL5: Observation Output Control Register 5

Use P0_4 IO port

In the ZigBee development board, P0_4 of CC2530 is connected to the LED, so the LED can be turned on and off through P0_4 of CC2530. The status and behavior of P0_4 can be controlled by configuring the registers related to P0_4. The registers related to P0_4 and the corresponding descriptions are shown in the table.

register	illustrate
P0	8-bit register, 8 bits correspond to P0_0~P0_7 respectively, and are used to set or read the level status of these 8 IO respectively.
P0SEL	8-bit register, 8 bits correspond to P0_0~P0_7 respectively, and configure the functions of these 8 IO ports respectively. If the bit corresponding to the IO port is 0, it means that the IO port is used for general input/output; if it is 1, it means that the IO port is used for a specific function
P0DIR	8-bit register, 8 bits correspond to P0_0~P0_7 respectively, and configure the communication direction of these 8 IO respectively. If the bit corresponding to the IO port is 0, it means that the IO port is in input signal mode; if it is 1, it means that the IO port is in output signal mode

According to the table description, the relevant registers of P0_4 can be configured and its output level can be controlled as follows. The code is as follows:

```
// 把P0_4配置为通用输出IO口  
P0SEL &= ~(1<<4); // 把P0SEL寄存器的第4位设置为0, 表示把P0_4配置为通用IO口  
P0DIR |= (1<<4); // 把P0DIR寄存器的第4位设置为1, 表示把P0_4配置为输出信号模式  
  
P0_4=0 ; //输出低电平  
P0_4=1 ; //输出高电平
```

P0_4 is defined in the header file ioCC2530.h. After configuring the relevant registers, assigning 0 or 1 to it can control its output level state. Here we briefly explain the left shift operator `<<`, the negation operator `~`, the bitwise AND operator `&`, and the bitwise OR operator `|` to deepen the reader's understanding of the code.

(1) The left shift operator `<<`

shifts all binary digits to the left by the specified number of bits, removes the high-order digits (discards), and fills the low-order digits with 0. For example, "1<<4" means that all the digits in 0000 0001 are shifted to the left by 4 bits, and the high-order 0s are removed and discarded, and the low-order digits are filled with 0, so the result of the operation is 0001 0000.

(2) The inversion operator `~`

inverts all binary digits. For example, the result of inverting 0001 0000 is 1110 1111.

(3) Bitwise AND operator `&`

The bitwise AND operation rule is to convert the numbers on both sides into binary numbers, and then perform the AND operation on the corresponding bits of the two numbers bit by bit. The AND operation rule is 1&1=1, 1&0=0, 0&1=0 and 0&0=0. For example, the result of the bitwise AND operation of 1101 1101 and 1110 1111 is 1100 1101.

(4) Bitwise OR operator `|`

The bitwise OR operation rule is to convert the numbers on both sides into binary numbers, and then perform the OR operation on the corresponding bits of the two numbers bit by bit. The OR operation rule is 1&1=1, 1&0=1, 0&1=1 and 0&0=0. For example, the result of the bitwise AND operation of 1101 1101 and 0001 0000 is 1101 1101.

Writing LED control code

After learning the relevant principles, you can write code to control the LED. The sample code is as follows:

```
//51单片机入门/2.GPIO实验/Workspace/code/led/led.c
```

```
#include "ioCC2530.h"
```

```
#include <stdio.h>
```

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

```
//P0_4由头文件ioCC2530.h定义
```

```
#define LED      P0_4
```

```
//定义LED的开关状态和对应的值
```

```
#define LED_ON     1
```

```
#define LED_OFF    0
```

```
static void delayMs(uint16_t nMs);
```

```
static void initLed(void);
```

```
void main()
```

```
{
```

```
    initLed(); //初始化LED
```

```
    while(1) {
```

```
        printf("Set led to on!\r\n");
```

```
        LED = LED_ON; //开启LED
```

```
        delayMs(500); //延时0.5s后才继续往下执行程序
```

```
        printf("Set led to off!\r\n");
```

```
        LED = LED_OFF; //关闭LED
```

```
        delayMs(500); //延时0.5s后才继续往下执行程序
```

```
    } /* while */
```

```
}
```

```
/**
```

```
* @fn    delayMs
```

```
*
```

```
* @brief   让程序延后指定的时间才接着运行
```

```
*
```

```
* @param  nMs - 时间长度，以毫秒为单位，值范围：165535
```

```
*
```

```
* @return  none
```

```
*/
```

```
static void delayMs(uint16_t nMs)
```

```
{
```

```
    uint16_t i,j;
```

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

```
        //经由实际测试可以得出执行535次循环耗时最接近1ms
```

```
        for (j = 0; j < 535; j++);
```

```
}
```

```
/**
```

```
* @fn    initLed
```

```
*
```

```
* @brief 初始化LED, 完成P0_4相关寄存器的配置
```

```
*/
```

```
static void initLed()
```

```
{
```

```
P0SEL &= ~(1<<4);
```

```
P0DIR |= (1<<4);
```

```
}
```

The above code implements the function of blinking LED. The program first initializes registers P0SEL and P0DIR through initLed, and then inverts the LED value at a fixed time, thus achieving the effect of blinking LED.

Simulation debugging

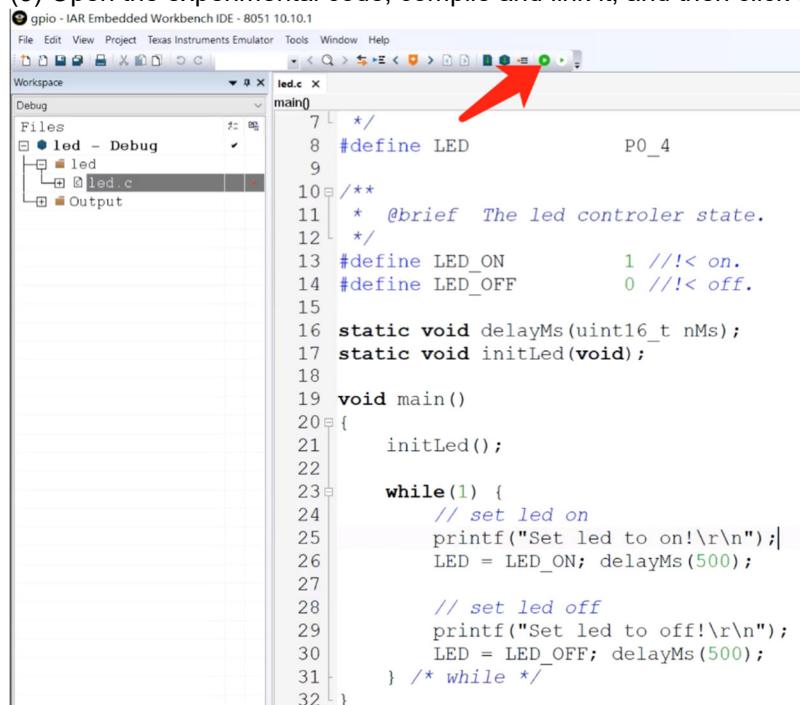
Note: Before studying this lesson, you need to master the basic program downloading and simulation operations, refer to "Part 2: Introduction to 51 MCU - Based on CC2530" → "Chapter 1: CC2530 Development Basic Experiment" → "Program Download and Simulation".

(1) Connect the development board to the computer through the emulator.

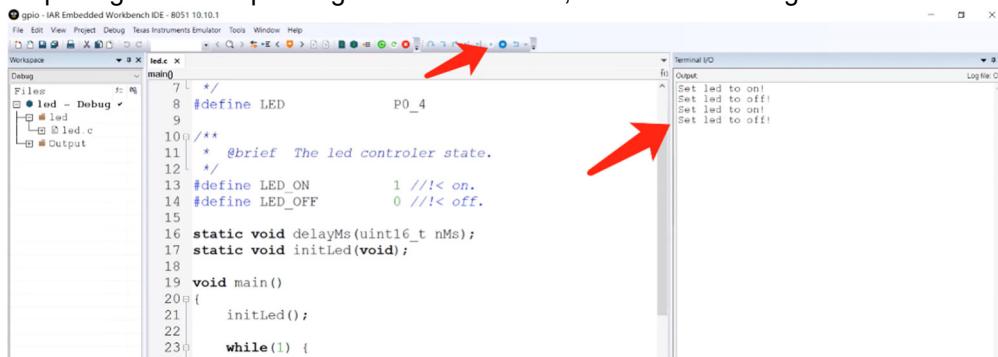
(2) Press the reset button of the emulator, as shown in the figure.



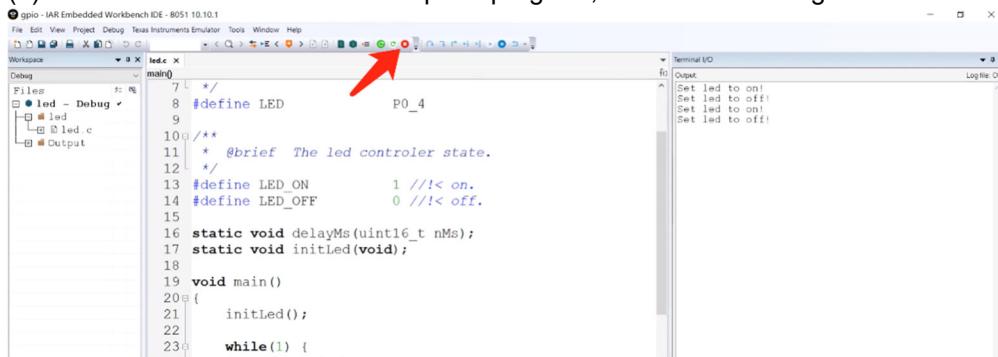
(3) Open the experimental code, compile and link it, and then click the "Download Simulation" button, as shown in the figure.



(4) Click the Go button to run the program at full speed. You can observe the LED flashing and the Terminal I/O continuously outputting the corresponding status information, as shown in the figure.



(5) Click the button as shown to stop the program, as shown in the figure.



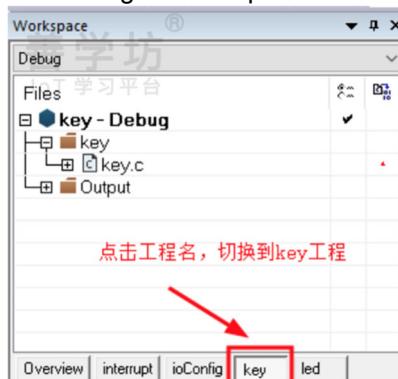
5.2.3. GPIO input experiment - mechanical buttons

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=8>

Technical support instructions:

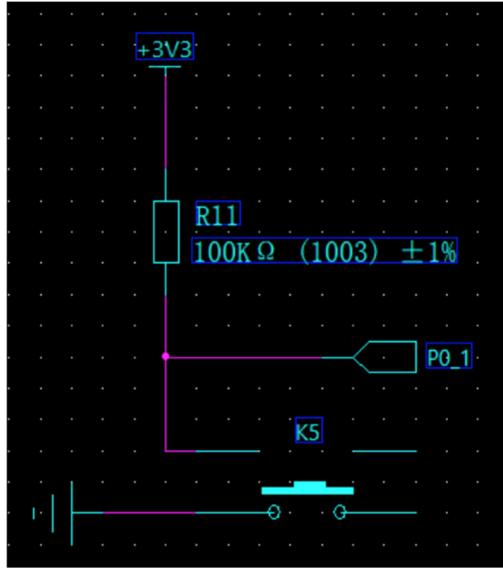
1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Continuing from the previous lesson, we switch to the button project, as shown in the figure.



Introduction to Button Principle

The matching ZigBee development boards are equipped with buttons, and their schematic diagram is shown in the figure.



If you lack knowledge of hardware principles and fail to understand this diagram, you need to supplement your knowledge first.

In the figure, K5 is a button, R11 is a 100KΩ pull-up resistor, and P0_1 is an IO port of CC2530. The relevant principles are as follows:

- (1) When the button is not pressed, P0_1 is connected to 3v3 through the pull-up resistor R11, so the input level of P0_1 is high.
- (2) When the button is pressed, P0_1 is grounded, so the input level of P0_1 is low.

From the above analysis, it can be seen that the input level state of P0_1 can be detected to detect whether the button is pressed.

P0_1 related registers

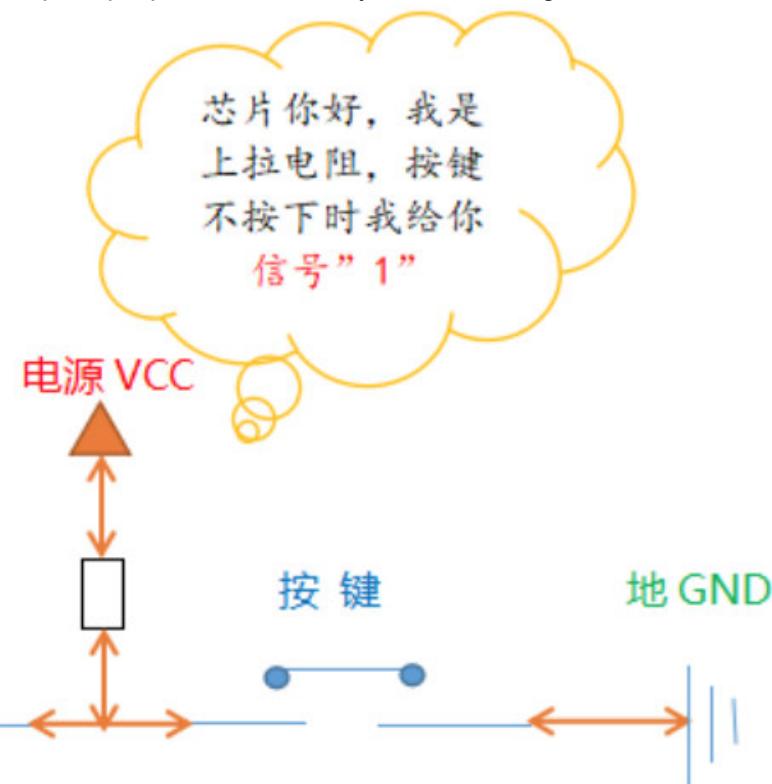
Similar to the LED experiment, to detect the input level status of P0_1, you must first configure the relevant registers. The relevant registers of P0_1 are shown in the table below.

register	illustrate
P0	8-bit register, 8 bits correspond to P0_0~P0_7 respectively, and are used to set or read the level status of these 8 ports respectively.
P0SEL	8-bit register, 8 bits correspond to P0_0~P0_7 respectively, and configure the functions of these 8 IO ports respectively. If the bit corresponding to the IO port is 0, it means that the IO port is used for general input/output; if it is 1, it means that it is used for a specific function
P0DIR	8-bit register, 8 bits correspond to P0_0~P0_7 respectively, and configure the communication direction of these 8 IO ports respectively. If the bit corresponding to the IO port is 0, it means that the IO port is in input signal mode; if it is 1, it means that it is in output signal mode
P0INP	8-bit register, 8 bits correspond to P0_0~P0_7 respectively, and configure the input mode of these 8 IO ports respectively. If the bit corresponding to the IO port is 0, it means that the IO port is in pull-up/pull-down input mode; if it is 1, it means tri-state mode
P2INP[7:5]	The 5th, 6th and 7th bits of the P2INP register are used to configure the pull-up or pull-down mode of terminals 0, 1 and 2 respectively. If it is 0, it indicates the pull-up mode; if it is 1, it indicates the pull-down mode.

Pull-up and pull-down input

The relevant registers involve pull-up and pull-down inputs, which are briefly explained. In layman's terms, pull-up means inputting a high level to the IO port in the default state, and on the contrary, pull-down means a low level. According to the

above key schematic diagram, this is a pull-up input mode. The key schematic diagram is shown in the figure.



When the button is not pressed, CC2530 is connected to the 3.3v power supply and the input level of its pin is high (corresponding to signal 1). When the button is pressed, CC2530 is connected to GND and the input level of its pin is low (corresponding to signal 0).

Register Configuration

Through the above analysis, the configuration code of the relevant registers is as follows:

```
P0SEL &= ~(1<<1); //把P0SEL寄存器的第1位设置为0, 即让P0_1用作通用IO口  
P0DIR &= ~(1<<1); //把P0DIR寄存器的第1位设置为0, 即让P0_1处于输入信号模式  
P0INP &= ~(1<<1); //把P0INP寄存器的第1位设置为0, 即让P0_1处于上拉/下拉输入模式  
P2INP &= ~(1<<5); //把P2INP寄存器的第5位设置为0, 即让端口0处于上拉输入模式
```

Writing key codes

Write code to flip the LED on and off each time you press the button. Open the key.c file in the Key folder and you can see the following sample code:

```
//2. 51单片机入门/2. GPIO实验/Workspace/code/key/key.c
```

```
#include "ioCC2530.h"  
#include <stdio.h>  
#include <stdint.h>  
  
#define DEBUG  
//#define xDEBUG  
  
#ifdef DEBUG  
#define DEBUG_LOG(...) printf(__VA_ARGS__)
```

```
#else
#define DEBUG_LOG(...)
#endif

#define LED    P0_4
#define LED_ON   1
#define LED_OFF  0

/***
 * @brief 按钮及其状态的定义。其中 · P0_1是在头文件ioCC2530.h, 可以检测其值来判断P0_1引脚的电平状态
 */
#define BUTTON  P0_1

#define BUTTON_NORMAL 1//按钮的默认状态
#define BUTTON_DOWN   0//按钮被按下

static void delayMs(uint16_t nMs);
static void initLed(void);
static void initButton(void);

void main() {
    initLed(); // 初始化LED
    initButton(); // 初始化按键
    while(1) {
        if (BUTTON != BUTTON_DOWN) // 如果按键没有被按下
            continue; // 结束本次while循环
        else {
            /* 以下为对按键的机械抖动的处理处理代码 */
            delayMs(10); // 延后10毫秒
            if (BUTTON != BUTTON_DOWN) // 再次检测按钮的状态，如果按键没有被按下
                continue; // 结束本次while循环
        }
        while (BUTTON == BUTTON_DOWN); // 等待BUTTON != BUTTON_DOWN, 即等待按键松开
        DEBUG_LOG("Key Pressed!\r\n");
        LED = (LED == LED_ON)? LED_OFF: LED_ON; // 翻转LED的状态
    }
}

static void delayMs(uint16_t nMs)
{
    uint16_t i,j;
    for (i = 0; i < nMs; i++) for (j = 0; j < 535; j++);
}
```

```

}

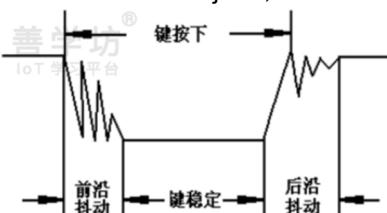
static void initLed()
{
    P0SEL &= ~(1<<4);
    P0DIR |= (1<<4);
}

/*
 * @fn    initButton
 *
 * @brief  初始化Button，完成P0_1相关寄存器的配置
 */
static void initButton()
{
    P0SEL &= ~(1<<1);//把P0SEL寄存器的第1位设置为0，即让P0_1用作通用IO口
    P0DIR &= ~(1<<1);//把P0DIR寄存器的第1位设置为0，即让P0_1处于输入信号模式
    P0INP &= ~(1<<1);//把P0INP寄存器的第1位设置为0，即让P0_1处于上拉/下拉输入模式
    P2INP &= ~(1<<5);//把P2INP寄存器的第5位设置为0，即让端口0处于上拉输入模式
}

```

Dealing with mechanical button jitter

The above code includes the processing of key jitter, which is briefly explained. Since the key uses a spring inside, when the key is pressed or released, it will produce a certain vibration, which can be called mechanical jitter. This mechanical jitter will cause the level to jitter, as shown in the figure.



The horizontal axis in the figure is the level, and the vertical axis is the time, showing the level change from the button being pressed to the button being released. The jitter generated when the button is pressed is called the leading edge jitter, and the jitter generated when the button is released is the trailing edge jitter. This jitter time generally lasts 5~10ms. Therefore, in the code, when it is detected that the button is pressed, it is necessary to delay 10ms before checking again whether the button is really pressed.

Using debug mode

During the process of program development and debugging, you can set a debugging mode. In the debugging mode, you can output relevant debugging information to understand the program running status. However, after the program development is completed, you can turn off the debugging mode and stop outputting debugging information.

DEBUG_LOG is a macro definition, which is defined at the beginning of the above code.

- When the code defines the DEBUG macro, it means that the current mode is debug mode, and DEBUG_LOG and printf are consistent.

- If the DEBUG macro is not defined, it means that the debug mode is turned off, and DEBUG_LOG does nothing.

Therefore, when you need to turn off the debug mode, you can change DEBUG to xDEBUG, which means turning off the debug mode.

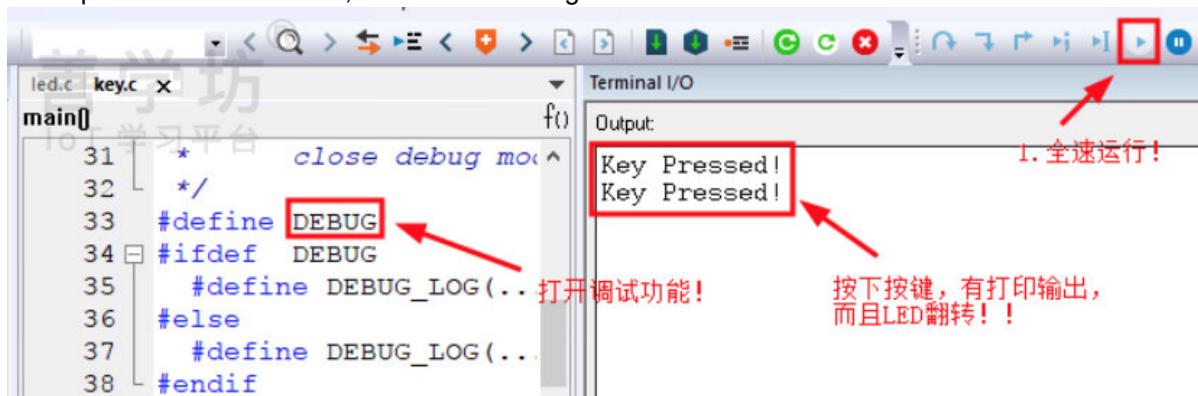
Simulation debugging

Note: Before studying this lesson, you need to master the basic program downloading and simulation operations, refer to "Part 2: Introduction to 51 MCU - Based on CC2530" → "Chapter 1: CC2530 Development Basic Experiment" → "Program Download and Simulation".

- (1) Connect the development board to the computer through the emulator.
- (2) Press the reset button on the emulator.



- (3) Open the experimental code, turn on the debugging mode, and after the compilation and linking are successful, click the "Download Simulation" button to run the program at full speed. Whenever a key is pressed, the corresponding information will be output in the Terminal I/O, as shown in the figure.



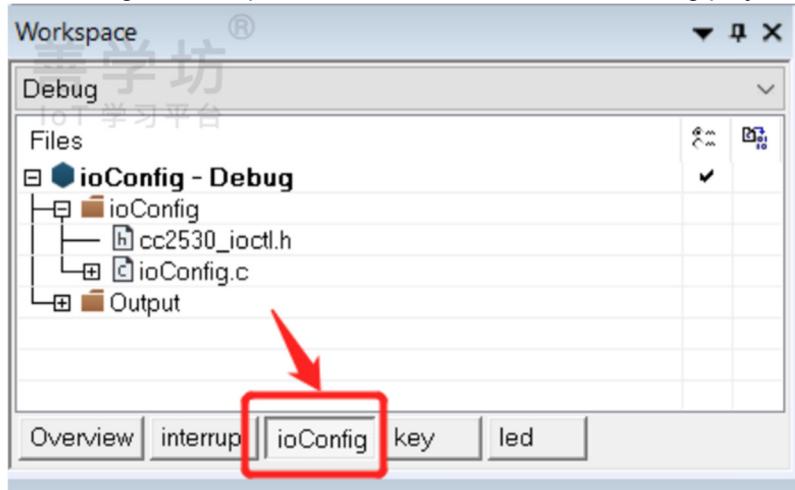
5.2.4. GPIO input and output general configuration experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=9>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Continuing from the previous lesson, switch to the ioConfig project, as shown in the figure.



GPIO input and output general configuration API

The effect to be achieved in this experiment is the same as that in the previous lesson, that is, controlling the LED flipping by pressing a button. The difference is that this lesson uses the GPIO input and output general configuration API CC2530_IOCTL, and developers no longer need to find and configure the relevant registers of GPIO by themselves.

Open the header file cc2530_ioctl.h and find the CC2530_IOCTL definition. The code is as follows:

```
//2.51单片机入门/4.串口通信实验/Workspace/code/common/cc2530_ioctl.h
```

```
/**  
 * @brief 配置GPIO模式  
 *  
 * @param port - CC2530引脚port号  
 * @param pin - CC2530引脚pin号  
 * @param mode - 该GPIO的模式  
 *  
 * @warning P1_0和P1_1不能配置为输入模式  
 */  
  
#define CC2530_IOCTL(port, pin, mode) do {  
    if (port > 2 || pin > 7) break;  
  
    if (mode == CC2530_OUTPUT) CC2530_IO_OUTPUT(port, pin);  
    else CC2530_IO_INPUT(port, pin, mode);  
} while(0)
```

The mode parameter can pass in the following parameters to indicate different modes:

```
//2.51单片机入门/4.串口通信实验/Workspace/code/common/cc2530_ioctl.h
```

```
/** @brief CC2530 GPIO mode. */  
  
#define CC2530_OUTPUT      0 //输出模式  
  
#define CC2530_INPUT_PULLUP 1 //上拉输入模式  
  
#define CC2530_INPUT_PULLDOWN 2 //下拉输入模式
```

```
#define CC2530_INPUT_TRISTATE 3 //3态模式
```

If you need to configure the P0_4 pin of CC2530 as output signal mode, just call it as follows:

```
CC2530_IOCTL(0, 4, CC2530_OUTPUT);
```

It can be seen that after using CC2530_IOCTL, the configuration of the GPIO port becomes very simple, and there is no need to find the relevant registers for configuring P0_4!

Using CC2530_IOCTL

Before using CC2530_IOCTL, you need to define the connection between LED and button and CC2530 pins respectively. In the header file cc2530_ioctl.h, you can find the connection definition between LED and button and CC2530 pins. The code is as follows:

```
//2.51单片机入门/4.串口通信实验/Workspace/code/common/cc2530_ioctl.h
```

// LED与GPIO的连接定义

```
#define LED_PORT    0    //Led port.  
#define LED_PIN     4    //Led pin.  
#define LED        P0_4  //Led GPIO.
```

// Button与GPIO的连接定义

```
#define BUTTON_PORT 0    //Button port.  
#define BUTTON_PIN   1    //Button pin.  
#define BUTTON      P0_1  //Button GPIO.
```

The above code defines the connection between the LED and the P0_4 pin of CC2530, and the connection between the button and the P0_1 pin of CC2530.

The advantage of doing this is that if the hardware circuit changes, for example, the LED is not connected to P0_4 or the button is not connected to P0_1, then the developer only needs to modify the configuration here without modifying the code.

The effects of the following two initialization functions are the same as the LED and button initialization functions in the previous chapters.

```
//2.51单片机入门/2.GPIO实验/Workspace/code/ioConfig/ioConfig.c
```

```
/*  
 * 初始化LED  
 */  
static void initLed()  
{  
    CC2530_IOCTL(LED_PORT, //LED的port  
                 LED_PIN, //LED的pin  
                 CC2530_OUTPUT); //配置为输出  
  
    LED = LED_OFF;  
}  
  
/*  
 * 初始化按键  
 */
```

```

*/
static void initButton()
{
    CC2530_IOCTL(BUTTON_PORT, //按键的port
    BUTTON_PIN, //按键的pin
    CC2530_INPUT_PULLUP); //配置为上拉输入
}

```

Simulation debugging

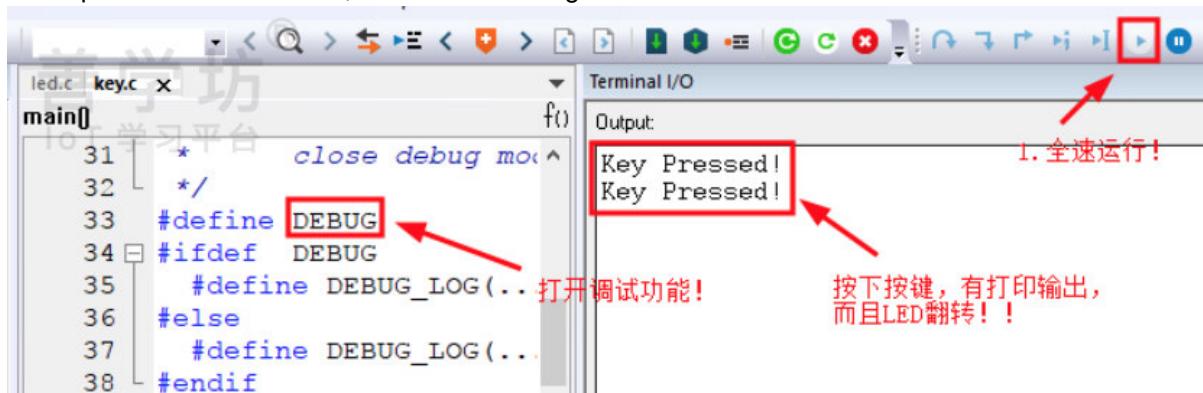
Note: Before studying this lesson, you need to master the basic program downloading and simulation operations, refer to "Part 2: Introduction to 51 MCU - Based on CC2530" → "Chapter 1: CC2530 Development Basic Experiment" → "Program Download and Simulation".

(1) Connect the development board to the computer through the emulator.

(2) Press the reset button of the emulator, as shown in the figure.



(3) Open the experimental code, turn on the debugging mode, and after the compilation and linking are successful, click the "Download Simulation" button to run the program at full speed. Whenever a key is pressed, the corresponding information will be output in the Terminal I/O, as shown in the figure.



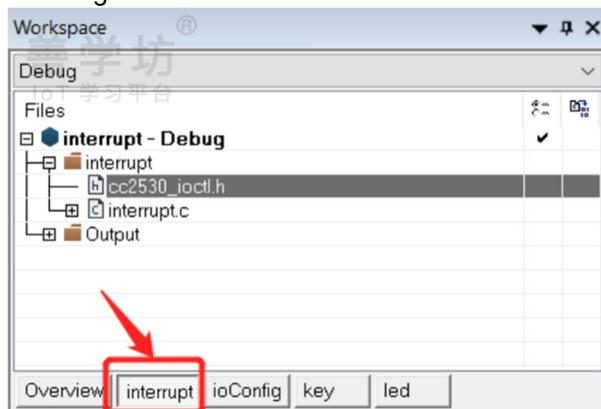
5.2.5. GPIO external interrupt experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=10>

Technical support instructions:

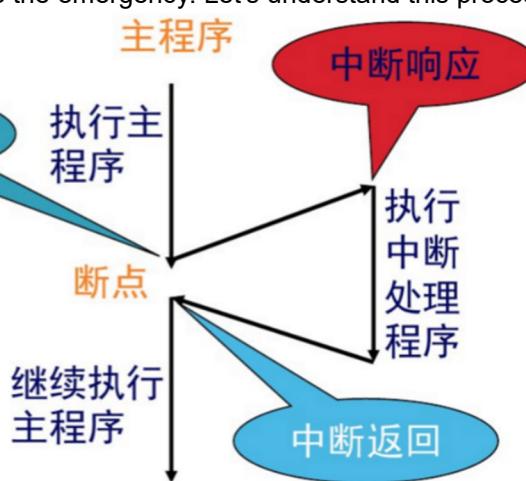
1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Continuing with the content of the previous lesson, switch to the project corresponding to the interruption experiment, as shown in the figure.



What is an interrupt?

In layman's terms, an interruption means that the system suddenly encounters an emergency during normal operation, such as a user suddenly pressing a button, a fire sensor detecting a fire, or the timer expires, and the system needs to put down the current work to handle the emergency. Let's understand this process through a logic diagram.



As shown in the figure, when the system is executing the main program, it encounters an emergency at a certain point in time (also called a breakpoint). At this time, the following three things usually happen:

(1) Interrupt request

This emergency event requests the system to pause the main program and handle the emergency.

(2) Interrupt response.

The system will execute a program called interrupt processing to handle the emergency.

(3) Interrupt return.

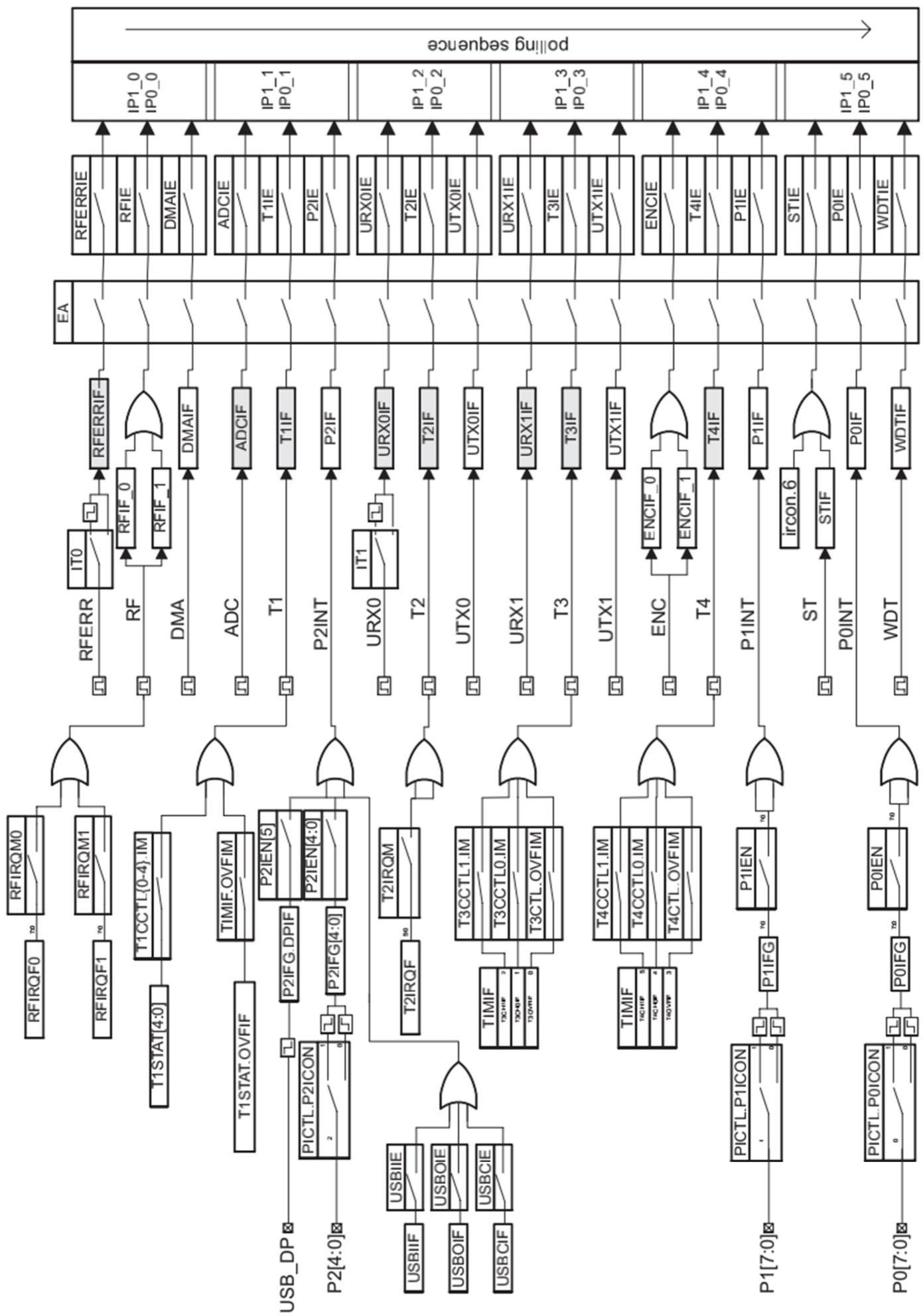
After handling the emergency, the system will return to the main program and continue to execute the main program from the paused position.

Interrupt Priority

When multiple interrupts occur at the same time, which interrupt should be processed first? We can define a priority for each interrupt, telling the system to process interrupts with higher priorities first and delay processing interrupts with lower priorities.

The interrupt system principle of CC2530

The interrupt system principle of CC2530 is shown in the figure.



The interrupt system principle diagram is quite complicated. Here we take the interrupt principle of port P0 as an example to explain. The rightmost part of the figure above is the interrupt process of port P0. Its related registers are P0, PICTL.P0ICON, P0IFG, P0IEN, P0IF, EA and IEN1.P0IE. See the table for related descriptions.

register	illustrate
PICTL.P0ICON	PICTL is an 8-bit port interrupt control register. P0ICON is the 0th bit of PICTL, which is used to configure the interrupt triggering mode of port P0. If the value of P0ICON is: 0, the rising edge of the input causes an interrupt; if the falling edge of the input causes an interrupt
P0IFG	8-bit register, 8 bits correspond to P0_0~P0_7 respectively, and are used to indicate the input interrupt status of P0_7 to P0_0 pins respectively. If an interrupt occurs, the corresponding flag bit will be set to 1
P0IEN	8-bit register, 8 bits correspond to P0_0~P0_7, respectively, and are used to configure whether the P0_7 to P0_0 pins allow interrupts. If the value of the corresponding bit is: 0, disable interrupts; 1, enable interrupts.
P0IF	If an interrupt occurs on port 0, the value is set to 1
EA	Interrupt master switch, if the value is: 0, disable all interrupts; 1: enable all interrupts
IEN1.P0IE	IEN1 is an 8-bit interrupt enable 1 register, which is used to configure whether to allow port interrupts, timer interrupts or DMA interrupts. P0IE is the 5th bit of IEN1, which is used to configure whether to allow port 0 to cause an interrupt. If the value is: 0, interrupts are disabled; 1, interrupts are enabled

Combined with the relevant register description, if you need to implement the CPU interrupt caused by the P0 port, the configuration steps are as follows:

- (1) Specify the interrupt triggering mode of port 0 through the P0ICON bit of PICTL.
- (2) Clear the P0IFG register to indicate that all IO ports in port 0 do not trigger interrupts in the default state.
- (3) Set the corresponding position of P0IEN to 1, indicating that the corresponding IO port is allowed to trigger interrupts. For example, if you need to allow P0_0 to trigger interrupts, then set the 0th position of P0IEN to 1.
- (4) Clear the P0IF register to indicate that port 0 does not trigger interrupts in the default state.
- (5) Set the EA register to 1, indicating that all interrupts are allowed
- (6) Set the P0IE position of the IEN1 register to 1, indicating that port 0 is allowed to cause interrupts.

When port 0 triggers an interrupt, the interrupt is distributed to the specified interrupt processing function by the interrupt vector IP0_5 or IP1_5. The following will explain it with specific examples.

Configuration of key interrupt

The button of the ZigBee development board is connected to P0_1 and uses the pull-up input method. If you need to display an interrupt after pressing the button, the corresponding configuration code is as follows:

```
PICTL |= 0x01;//P0ICON=1, 指定端口0在输入下降沿信号时触发中断 · 即输入电平由高电平转为低电平时触发中断
```

```
P0IFG = 0x00;//清零P0IFG寄存器
```

```
P0IEN |= (1<<1);// 把P0_1设置为允许引发中断
```

```
P0IF = 0x00;//清零P0IF寄存器
```

```
EA = 1;//允许所有中断
```

```
IEN1 |= 0x20;//P0IE=1, 允许端口0引起中断
```

Write key interrupt code

Open the code of this experiment, the code of the main function is as follows:

```
//2. 51单片机入门/2. GPIO实验/Workspace/code/interrupt/interrupt.c
```

```
void main()
```

```
{
```

```
    initLed(); //初始化LED
```

```
    initButton(); //初始化按键
```

```

while(1) {//每隔100ms计数一次
    counter_g++; //计数1次
    delayMs(100); //延迟100ms
}
}

```

In the above code, counter_g is a counter that counts every 100ms. In the button initialization function initButton, configure P0_1 as an interrupt input. The code is as follows:

```

//2. 51单片机入门/2. GPIO实验/Workspace/code/interrupt/interrupt.c

static void initButton(void)
{
    PICTL |= 0x01; //P0ICON=1, 指定端口0在输入下降沿信号时触发中断，即输入电平由高电平转为低电平时触发中断
    P0IFG = 0x00; //清零P0IFG寄存器
    P0IEN |= (1<<1); //把P0_1设置为允许引发中断
    P0IF = 0x00; //清零P0IF寄存器
    EA = 1; //允许所有中断
    IEN1 |= 0x20; //P0IE=1, 允许端口0引起中断
}

```

The interrupt handling function is defined as follows:

```

//2. 51单片机入门/2. GPIO实验/Workspace/code/interrupt/interrupt.c

#pragma vector = P0INT_VECTOR
__interrupt void buttonISR(void)
{
    delayMs(10); //处理机械按键抖动

    if (BUTTON == BUTTON_DOWN) {
        DEBUG_LOG("counter_g: %d\r\n", counter_g); //输出计数器的值
        LED = (LED == LED_ON)? LED_OFF: LED_ON; //翻转LED灯的状态
    }

    //处理完中断后，需要清零相关寄存器
    P0IFG = 0;
    P0IF = 0;
}

```

#pragma is a compiler preprocessing directive, which tells the compiler that vector is a variable that is only valid at the compile stage. vector represents an interrupt vector, which is assigned the value of P0INT_VECTOR, and is followed by the function buttonISR with the keyword __interrupt, indicating that this function is the interrupt processing function caused by P0, that is, after P0 causes an interrupt, this function handles the interrupt. In the interrupt processing function, the interrupt flag must be cleared to 0, otherwise when the interrupt is encountered again, the CPU will not execute the interrupt processing function again.

Simulation debugging

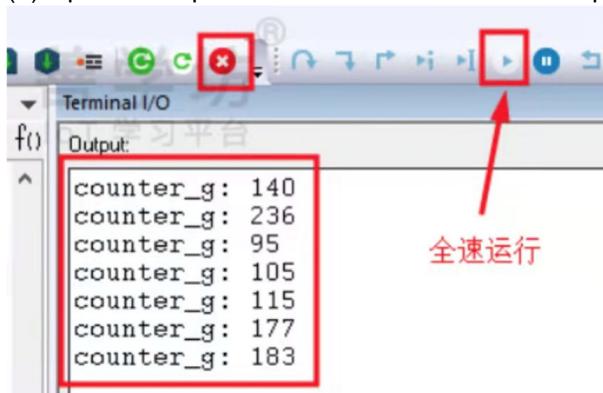
Note: Before studying this lesson, you need to master the basic program downloading and simulation operations, refer to "Part 2: Introduction to 51 MCU - Based on CC2530" → "Chapter 1: CC2530 Development Basic Experiment" → "Program Download and Simulation".

Using the emulator and ZigBee development board, you can observe the running results on the ZigBee development board. The simulation debugging steps are as follows:

- (1) Connect the development board to the computer through the emulator.
- (2) Press the reset button of the emulator, as shown in the figure.



- (3) Open the experimental code and run it at full speed after it is compiled and linked, as shown in the figure.



When the button is pressed, the counter value will be displayed in Terminal I/O and the state of the LED light will be flipped. If you need to stop simulation debugging, you can click the red cross button in the figure.

5.3. Chapter 3: Timer Experiment

5.3.1. Project Overview

- 5.3.2. Timer T1 Experiment - Query Trigger
- 5.3.3. Timer T3 Experiment - Interrupt Trigger
- 5.3.4. Watchdog Timer Experiment
- 5.3.5. Low Power Timer Experiment

5.3.1. Project Overview

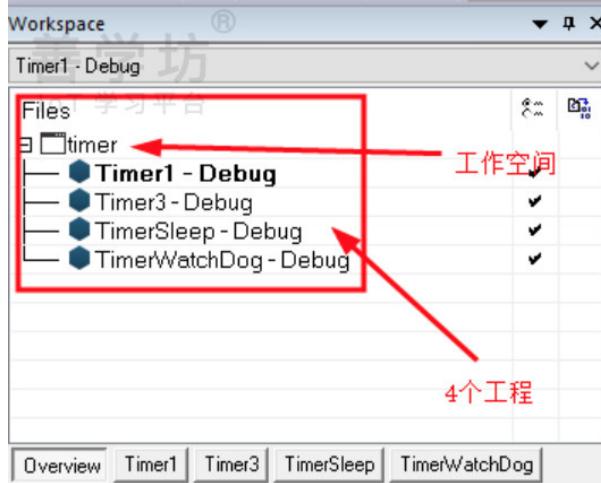
Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=11>

Technical support instructions:

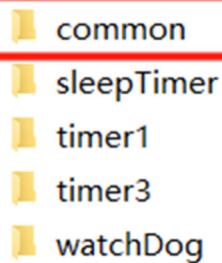
1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Create a project

Similar to the previous chapter, we create a project for each lesson in a workspace:



Create relevant directories and files in the code directory, where the common folder is used to store shared files:



Files in common (we separate the delay function and pin common configuration into public files):

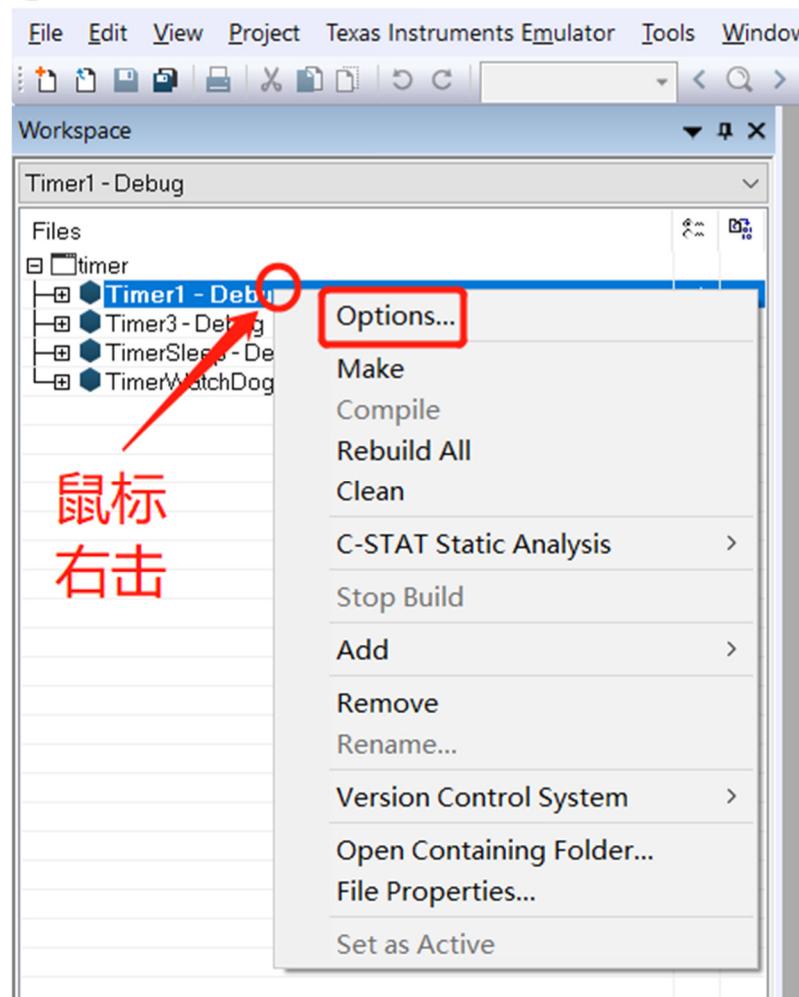


Project Configuration

However, to use the public directory for this project, you must first set it up in the project so that the compiler can find the directory where the file is located.

(1) Open the project options and make settings.

timer - IAR Embedded Workbench IDE - 8051 10.10.1



(2) Go to the C/C++ Compiler tab, select the Preprocessor option, and then click the following button.

Options for node "Timer1"

善学坊
IOT 学习平台

Category:

General Options
Static Analysis
C/C++ Compiler (highlighted)
Assembler
Custom Build
Build Actions
Linker
Debugger
Third-Party Driver
Texas Instruments
FS2 System Navigator
Infineon
Segger J-Link
Nordic Semiconductor
ROM-Monitor
Analog Devices
Silicon Labs
Simulator

Factory Settings

Multi-file Compilation
 Discard Unused Publics

Extra Options MISRA-C:2004 MISRA-C:1998

Language 1 Language 2 Code Optimizations Output

List Preprocessor Diagnostics

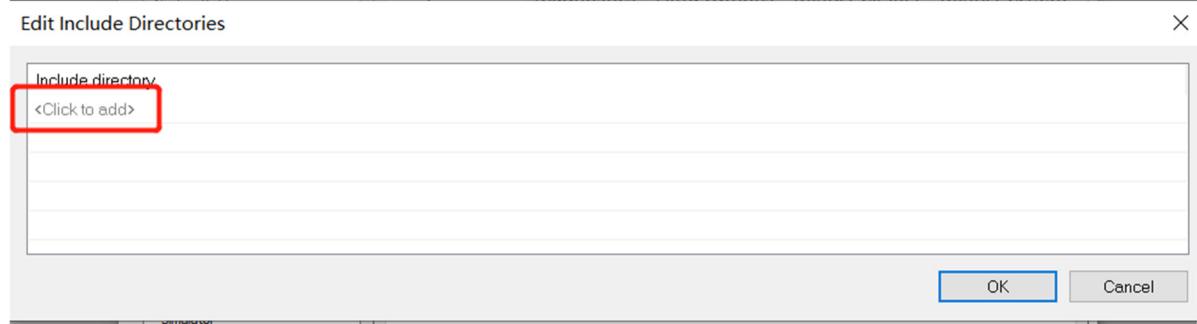
Ignore standard include directories

Additional include directories: (one per line)

Preinclude

Defined symbols: (one per line)

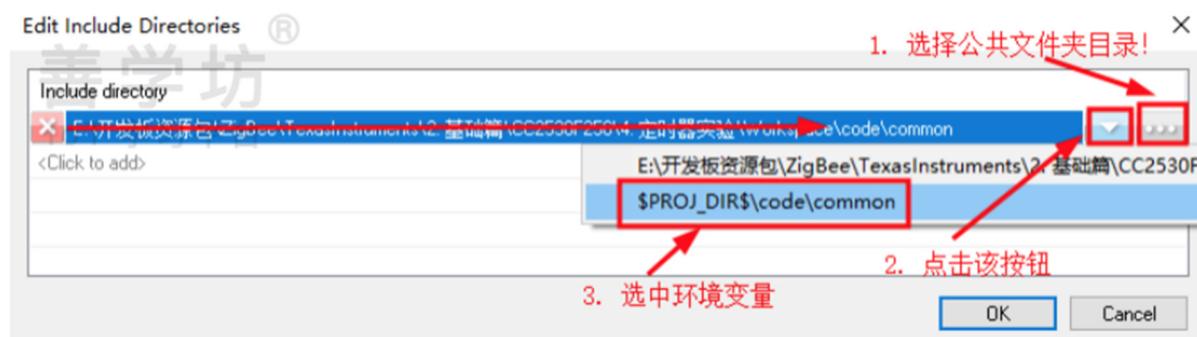
(3) Click to add.



(4) In the pop-up dialog box, find the directory where the common folder is located:



(5) You can also convert the absolute path to a relative path:



Tip: This has already been set up in the project that comes with this course. Here we will mainly explain how to set the folder path.

5.3.2. Timer T1 Experiment - Query Trigger

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=12>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

In this section, we will use the CC2530's internal timer 1 to perform timed on/off control on the LED, etc., to achieve the effect of blinking the LED.

Basic theory of timers

(1) System clock frequency

The clock generator generates pulses at constant time intervals. This intermittent pulse can be figuratively understood as the heartbeat of the chip, and the clock frequency is used to describe the rate of this heartbeat. People usually use the number of pulses generated by the clock generator within 1 second to describe the clock frequency. For example, "clock frequency 10 MHz" means the number of heartbeats within 1 second is 10,000,000 times. CC2530 has two clock frequencies available for developers: 32MHz and 16MHz.

(2) Frequency division

coefficient Frequency division means reducing the clock frequency to 1/N of the original frequency, also known as N division. For example, when the clock frequency is 16MHz, then 2 division is 8MHz. The frequency division coefficient is expressed as 1/N, for example, the frequency division coefficient of 2 division is: 1/2.

(3) The relationship between the system clock cycle

and frequency can be expressed by the formula: $T = 1 / f$, where T is the clock cycle and f is the clock frequency. The clock cycle and clock frequency are inversely proportional. For example, when the clock frequency is 16MHz, it means that the clock generator can generate 16,000,000 pulses within 1s, and the clock cycle can represent the time required to generate a pulse, that is, $1 / 160,000 \text{ } 00\text{s}$.

(4) Counter

The counter is the core of the timer and is used to record the number of pulses generated by the clock generator. Since the clock period of the pulse is constant, the formula for calculating the timing time is: $t = nT$, where t is the timing time, n is the number of counts, and T is the clock period.

(5) Overflow

Since the range of the counter is limited, an overflow will occur when the number of counts exceeds the maximum value. For example, when the size of the counter is 16 bits, the counting range is 0~65535, so the counter will overflow when the number of counts exceeds 65535. After the overflow occurs, the counter value will change from the maximum value to 0.

Timing principle

The relationship between frequency and period is further explained using the formula. We use f to represent the clock frequency and T to represent the clock period, and then we can use this formula to represent their relationship:

- $T = 1 / f \quad (1)$

After we count t seconds, assume that the counter has counted N times from 0 (assuming that the counter has not overflowed). As explained above, the clock period T represents the time required for one heartbeat, so the relationship between t and N is as follows:

- $t = N \times T \quad (2)$

Then, we can deduce:

- $N = t / T \quad (3)$

The default system clock frequency of CC2530 is 16MHz (16000000Hz), and its Timer 1 uses 128 division, so the clock frequency of the timer is $16000000 / 128 \text{ Hz}$.

- According to formula (1) $T = 1 / f$, we can calculate that the clock period of timer 1 is $T = 128/16000000 \text{ seconds}$.
- In the case of a 5-second timing (i.e. $t=5 \text{ seconds}$), according to formula (2) $N = t / T$, the count value of the counter $N = 5 / (128/16000000) = 625000$.

Handling Overflow

When the timer overflows, an interrupt will occur. At this time, Bit 1 of the IRCON register will be set from 0 to 1. Therefore, we only need to check this flag to determine whether an overflow has occurred.

For specific related register descriptions, please read the following descriptions

Timer 1 is a 16-bit timer that counts 65536 times each time it overflows, so after 5 seconds, it will overflow: $625000 / 65536 = 9.54$, which is an integer, 9 times. Conversely, if it overflows 9 times, we can roughly assume that 5 seconds have passed.

Related registers

寄存器	作用	说明
T1CTL	定时器 1 控制器	<p>T1CTL (Bit 3:2) 分频器划分值，如下：</p> <ul style="list-style-type: none"> 00: 标记频率/1 01: 标记频率/8 10: 标记频率/32 11: 标记频率/128 <p>T1CTL (Bit 1:0) 选择定时器 1 模式</p> <ul style="list-style-type: none"> 00: 暂停运行。 01: 自由运行，从 0x0000 到 0xFFFF 反复计数。 10: 模，从 0x0000 到 T1CC0 反复计数。 11: 正计数/倒计数，从 0x0000 到 T1CC0 反复计数并且从 T1CC0 倒计数到 0x0000。

T1STAT	定时器 1 状态	<p>Bit5: 定时器 计数器溢出中断标志</p> <p>Bit4: 定时器 1 通道 4 中断标志</p> <p>Bit3: 定时器 1 通道 3 中断标志</p> <p>Bit2: 定时器 1 通道 2 中断标志</p> <p>Bit1: 定时器 1 通道 1 中断标志</p> <p>Bit0: 定时器 1 通道 0 中断标志</p> <p>Bit[5:0]没有清零的说法，写 1 会自动清零！</p>
IRCON	中断标志	<p>Bit1: 定时器 1 中断标志。中断发生时设为 1 并且当 CPU 向量指向中断服务例程时清除。</p> <ul style="list-style-type: none"> 0: 无中断未决 1: 中断未决

Register Configuration

- Timer 1 is a 16-bit timer, which means the counter can count from 0 to 65535.
- Timer 1 supports 5 channels

1.T1CTL = 0x0D; // 0000 1101 : 128分频 · 自由计数(从0~65535)

2.T1STAT= 0x21; // bit0写1, 清空通道0中断状态位

3. // bit5写1, 清空计数器中断状态位

Program Description

The main function code is as follows:

```
void main()
{
    uint8_t Counter = 0;

    initLed();
    initTimer1();

    while(1) {
        if (!(IRCON & 0x02)) continue; // Timer1 interrupt not pending

        IRCON &= ~(0x02); // Clear timer1 interrupt flag

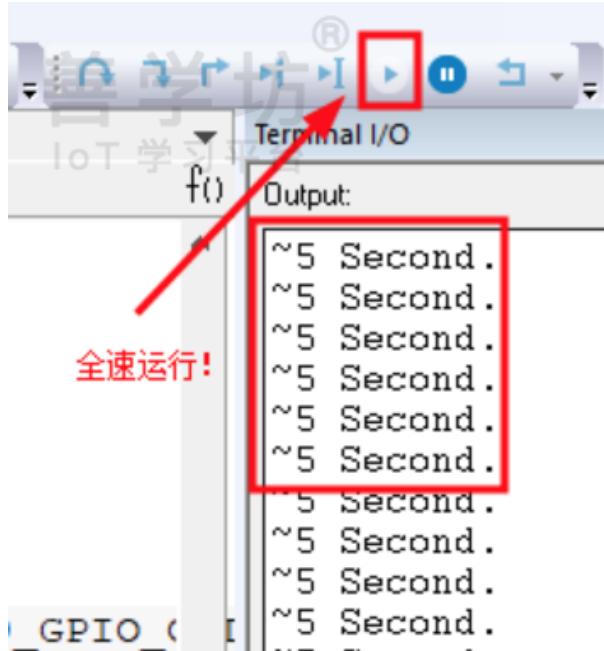
        if (++Counter < 9) continue; // ~5 second
        else Counter = 0;

        DEBUG_LOG("~5 Second.\r\n");
        LED = (LED == LED_ON)?LED_OFF:LED_ON;
    } /* while */
}
```

The main function defines a counter to record the number of timer overflows. We have calculated that the timer will overflow 9 times after 5 seconds. Every time the timer overflows, the overflow flag needs to be cleared. After overflowing 9 times, we need to reset the counter to 0 and count again. In this way, we can print out the corresponding prompt every 5 seconds and flip the LED light on and off.

Simulation debugging

(1) Connect the development board to the emulator and enter the simulation mode:



(2) You can see that information is printed out approximately every 5 seconds and the LED light flips.

5.3.3. Timer T3 Experiment - Interrupt Trigger

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=13>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Timer 3 Introduction

Timer 3 is an 8-bit timer with a counting range of 0 to 255. In the last lesson, we used the query method to set the time. This lesson will guide readers to use the interrupt method.

Related registers

寄存器	作用	说明													
T3CTL	定时器 3 控制器	<p>Bit[7:5] : 定时器时钟分频倍数选择:</p> <table style="margin-left: 40px;"> <tr> <td>000: 不分频</td> <td>001: 2 分频</td> <td>010: 4 分频</td> </tr> <tr> <td>011: 8 分频</td> <td>100: 16 分频</td> <td>101: 32 分频</td> </tr> <tr> <td>110: 64 分频</td> <td>111: 128 分频</td> <td></td> </tr> </table> <p>Bit4 : T3 起止控制位</p> <p>Bit3 : 溢出中断掩码</p> <p>0: 关溢出中断 1: 开溢出中断</p> <p>Bit2 : 清计数值, 高电平有效</p> <p>Bit[1:0] T3 模式选择</p> <table style="margin-left: 40px;"> <tr> <td>00: 自动重装 0x00-0xFF</td> </tr> <tr> <td>01: DOWN (从 T3CC0 到 0X00 计数一次)</td> </tr> <tr> <td>10: 模计数 (反复从 0X00 到 T3CC0 计数)</td> </tr> <tr> <td>11: UP/DOWN(反复从 0 到 T3CC0 计数再到 0)</td> </tr> </table>	000: 不分频	001: 2 分频	010: 4 分频	011: 8 分频	100: 16 分频	101: 32 分频	110: 64 分频	111: 128 分频		00: 自动重装 0x00-0xFF	01: DOWN (从 T3CC0 到 0X00 计数一次)	10: 模计数 (反复从 0X00 到 T3CC0 计数)	11: UP/DOWN(反复从 0 到 T3CC0 计数再到 0)
000: 不分频	001: 2 分频	010: 4 分频													
011: 8 分频	100: 16 分频	101: 32 分频													
110: 64 分频	111: 128 分频														
00: 自动重装 0x00-0xFF															
01: DOWN (从 T3CC0 到 0X00 计数一次)															
10: 模计数 (反复从 0X00 到 T3CC0 计数)															
11: UP/DOWN(反复从 0 到 T3CC0 计数再到 0)															
T3CC0	定时器 3 通道 0 捕获/比较值	T3CCTL0.MODE=1 时 写 该 寄 存 器 会 导 致 T3CC0.VAL[7:0] 更 新 到 写 入 值 延 迟 到 T3CNT.CNT[7:0]=0													

T3CCTL0	T3 通道 0 捕获 / 比较控制寄存器	<p>Bit6: 通道 0 中断屏蔽 0: 中断禁止 1: 中断使能</p> <p>Bit[5: 3] T3 通道 0 比较输出模式选择</p> <p>Bit2: T3 通道 0 模式选择: 0: 捕获 1: 比较</p> <p>Bit[1:0] T3 通道 0 捕获模式选择 00 没有捕获 01 上升沿捕获 10 下降沿捕获 11 边沿捕获</p>
T3CCTL1	T3 通道 1 捕获 / 比较控制寄存器	<p>Bit6: 通道 1 中断屏蔽 0: 中断禁止 1: 中断使能</p> <p>Bit[5: 3] T3 通道 1 比较输出模式选择</p> <p>Bit2: T3 通道 1 模式选择: 0: 捕获 1: 比较</p> <p>Bit[1:0] T3 通道 1 捕获模式选择 00 没有捕获 01 上升沿捕获 10 下降沿捕获 11 边沿捕获</p>
T3CC1	定时器 3 通道 1 捕获/比较值	定时器捕获/比较值通道 1。当 T3CCTL1.MODE=1 (比较模式) 时写该寄存器会导致 T3CC1.VAL[7:0] 更新写入值延迟到 T3CNT.CNT[7:0]=0x00
T3IE	定时器 3 中断使能开关	0: 关闭 1: 允许中断

Register Configuration

```

1.T3CTL = 0xE8; // Bit[7:5]: 111 -> 128分频 ;
2.          // Bit3 : 1 -> 打开溢出中断
3.          // Bit[1:0] : 00 -> 自由计数，反复从0到255
4.T3IE = 1; // 使能定时器3中断
5.EA = 1; // 开启中断总开关
6.T3CTL |= 0x10; // 启动定时器

```

Timing principle

The system clock frequency of timer 3 is: 16 MHz divided by 128, that is, 16000000/128Hz

- From formula 1 in the previous lesson, we can conclude that the clock cycle T=128/16000000

- When the timing is 5 seconds, we can deduce from Formula 2 in the previous lesson that the count value

$$N=5/(128/16000000) = 625000.$$

It can be concluded that the timer 3 takes 5 seconds to count 625,000 times when the system clock frequency is 16 MHz divided by 128.

Handling Overflow

Timer 3 is an 8-bit counter that counts 256 times for each overflow. Therefore, the number of overflows after 5 seconds of timing is: $625000 / 256 = 2441.4$, or 2441 times.

Source code analysis

```
void main()
{
    initLed();
    initTimer3();

    while(1) {}
}
```

Working contents of timer 3 initialization function

- 128-way
- Enable overflow interrupt
- Free counting
- Interrupt Enable
- Last start timer

```
static void initTimer3(void)
{
    T3CTL |= 0xE8; // Tick frequency/128
        // Overflow interrupt is enabled
        // Free running, repeatedly count from 0x00 to 0xFF

    T3IE |= 1; // Enable timer3 interrupt
    EA |= 1; // Enable Interrupts

    T3CTL |= 0x10; // Start timer
}
```

Timer 3 interrupt service function

- Declare interrupt vector T3_VECTOR
- Overflow count 2441 times, overflow 2441 times duration is approximately equal to 5 seconds
- Debug output and invert LED every 5 seconds

```
/*
 * Timer3 interrupt service function
*/
```

```

*/
#pragma vector = T3_VECTOR
__interrupt void Timer3_ISR(void)
{
    // ~5s
    if (++counter_g == 2441) {
        counter_g = 0;

        DEBUG_LOG("Timer3 timeout -> 5-seconds!\r\n");
        LED = (LED == LED_ON)?LED_OFF: LED_ON;
    }
}

```

Simulation debugging

After opening the project provided with this lesson and compiling it, perform simulation debugging. The running results are shown in the figure below:



5.3.4. Watchdog Timer Experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=14>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Theoretical basis

(1) Watchdog timer

The watchdog timer is also a timer. Compared with timers T1 and T3, its special feature is that it resets the chip when the time is up.

(2) Feeding the watchdog

To prevent the watchdog timer from resetting the chip, we can restart the watchdog timer before it expires (i.e., clear the timer). We call this action of clearing the timer "feeding the watchdog".

(3) Program runaway

During the normal operation of the program, it may suddenly encounter an unexpected event (such as static interference) and fail to run normally. We call this phenomenon "program runaway". When the program runs away, the watchdog is not "fed" in time, and the chip will be reset when the time is up. After resetting the chip, we can use technical means to restore the program to normal. (This is also the significance of the watchdog timer)

(4) Application scenarios

The watchdog timer can be used in harsh environments such as those with high electrical noise, high power failure rate, or electrostatic discharge, or in situations where higher reliability is required. If the system does not need to use a watchdog, it can be configured as an interval timer to interrupt at a specified time interval.

Related registers

寄存器	作用	说明
WDCTL	看门狗控制寄器	<p>Bit[7:4]</p> <p>清除定时器。在这几位依次写入 0xA 和 0x5，定时器被清除。</p> <p>注意定时器仅写入 0xA 后，在 1 个看门狗时钟周期内写入 0x5 时被清除。当看门狗定时器是 IDLE 时写这些位没有影响。当运行在定时器模式，定时器可以通过写 1 到 CLR[0]（不管其他 3 位）被清除为 0x0000（但是不停止）。</p> <p>Bit[3:2]</p> <p>模式选择。看门狗模式还是定时器模式。当处于定时器模式，设置这些位为 IDLE 将停止定时器。注意：当运行在定时器模式时要转换到看门狗模式，首先停止 WDT，然后启动 WDT 处于看门狗模式。当运行在看门狗模式，写这些位没有影响。</p> <p>00: IDLE 01: IDLE 10: 看门狗模式 11: 定时器模式</p> <p>Bit[1:0]</p> <p>定时器间隔选择。这些位选择定时器间隔定义为 32kHz 振荡器周期的规定数。注意间隔只能在 WDT 处于 IDLE 时改变，这样间隔必须在定时器启动的同时设置。</p> <p>00: 定时周期×32,768 (~1 s) 当运行在 32 kHz XOSC 01: 定时周期×8192 (~0.25 s) 10: 定时周期×512 (~15.625 ms) 11: 定时周期×64 (~1.9 ms)</p>

The register initialization configuration code is as follows:

1.WDCTL = 0x00; // 打开 IDLE模式才能设置看门狗

2.WDCTL = 0x08; // 看门狗模式、定时1秒

The dog feeding configuration code is as follows:

```
1.WDCTL |= (0xA << 4); // 在Bit[7:4]依次写入0xA和0x5, 定时器被清除
```

```
2.WDCTL |= (0x5 << 4);
```

Program Analysis

The main function code is as follows:

```
void main()
{
    initLed();
    initWatchDogTimer();

    while(1) {
        #if 0 //0 or 1

        delayMs(SYSCLK_16MHZ, 1500);

        #else

        delayMs(SYSCLK_16MHZ, 500);
        watchDogFeed();

        delayMs(SYSCLK_16MHZ, 500);
        watchDogFeed();

        delayMs(SYSCLK_16MHZ, 500);
        watchDogFeed();

        #endif

        LED = (LED == LED_ON)?LED_OFF: LED_ON;
    }
}
```

The watchdog timer is set to 1 second in the code, so the watchdog must be fed within 1 second.

The first program delays 1.5 seconds and then flips the LED light. Since the dog is not fed in time, the system keeps resetting and the LED light flashing process cannot be seen.

Although the second program also delays for 1.5 seconds, the delay is divided into three times and the watchdog is fed every 500ms, so the watchdog will not cause a reset, the program runs normally, and the LED light flashes.

Debugging Simulation

(1) Open the first section of the program pre-compilation: #if 1, compile and download to the development board, the LED light is always off!

(2) Open the second program pre-compilation: #if 0, compile and download to the development board, the LED light flashes!

5.3.5. Low Power Timer Experiment

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Theoretical basis

(1) Sleep and wake-up

There are many modules inside CC2530, such as timer module, transceiver module and 51 core, etc. These digital modules do not consume power when they are not working. We call this state of the module " **sleep** ". When these modules enter the working state from the "sleep" state, the process is called " **wake-up** ".

(2) Low power consumption

Low power consumption means that the chip consumes very little power. Generally speaking, the amount of power consumption depends on how many modules are in sleep mode and how many modules are in working mode inside the chip. We can also classify low power consumption: the more modules inside the chip are in sleep mode, the lower its power consumption.

(3) The low-power timer

is also called the sleep timer. It can set a certain period of time, during which the relevant module will enter the sleep state and wake it up when the time is up.

The sleep timer in CC2530 is a 24-bit timer with a clock frequency of 32768kHz.

Power Management Modes

CC2530 has the following centralized power management modes:

(1) Full-function mode:

The high-frequency crystal oscillator (16M or 32M) and the low-frequency crystal oscillator (32.768K RCOSC/XOSC) are fully operational, and the digital processor module operates normally.

(2) Idle mode

is the same as full-function mode except that the CPU core stops running.

(3) PM1

high-frequency crystal oscillator is turned off, low-frequency crystal oscillator is working normally, and digital core module is working normally

(4) PM2

low-frequency crystal oscillator works, the digital core module is turned off, and the system wakes up through RESET, external interrupt or sleep counter overflow.

(5)

All PM3 crystal oscillators are turned off, the digital processor core module is turned off, and the system can only be awakened by RESET or external interrupt. This mode has the lowest power consumption.

Related registers

寄存器	作用	说明
PCON	供电模式控制	Bit[0] 供电模式控制。 写 1 到该位强制设备进入 SLEEP.MODE (注意 MODE=0x00 且 IDLE = 1 将停止 CPU 内核活动) 设置的供电模式, 这位读出来一直是 0。当活动时, 所有的使能中断将清除这个位, 设备将重新进入主动模式。
SLEEPDM D	睡眠模式控制	Bit[1:0] 供电模式设置 00 : 主动/空闲模式 01 : 供电模式 1 10 : 供电模式 2 11 : 供电模式 3
ST0		睡眠计数器数据 Bit[7:0]
ST1	24 位睡眠计数器	睡眠计数器数据 Bit[15:8]
ST2		睡眠计数器数据 Bit[23:16]
STIE	使能休眠定时器 中断开关	0: 关闭 1: 打开
STIF	休眠定时器中断 标志位	0: 无中断 1: 有中断

Register Configuration

The sleep timer initialization configuration code is as follows:

```

1.ST2 = 0; ST1 = 0; ST0 = 0; // 清零休眠定时器计数器
2.STIE = 1; // 开启休眠定时器中断
3.STIF = 0; // 清零休眠定时器中断标志
4.EA = 1; // 打开中断总开关

```

To configure the sleep time, follow these steps:

(1) Since the sleep timer operates at a clock frequency of 32768 Hz, it counts 32768 times in 1 second!

(2) When configuring the sleep time, you only need to read the current counter value, add the time count value we need to set, and then reset it into the register.

^

Read the current sleep timer count value:

```

1.uint32_t sleepTimer = 0;
2.sleepTimer = (uint32_t)ST0;
3.sleepTimer |= (uint32_t)ST1 << 8;
4.sleepTimer |= (uint32_t)ST2 << 16;

```

Add the number of counts corresponding to the time we need to time, because counting 32768 times is 1 second, assuming we need to time sec seconds, that is counting sec * 32768 times:

```
1.// 更新睡眠时间  
2.sleepTimer += (uint32_t)sec * 32768;  
把计数值重新配置到寄存器中  
1.ST2 = (uint8_t)(sleepTimer >> 16);  
2.ST1 = (uint8_t)(sleepTimer >> 8);  
3.ST0 = (uint8_t)(sleepTimer);
```

To configure hibernation mode:

```
1.SLEEPCMD |= mode; // 设置该定时器Bit[1:0], mode: 0~3  
2.PCON = 0x01; // 进入睡眠模式
```

To exit Hibernation mode:

```
1.// 退出休眠  
2.PCON = 0x00;
```

Source code analysis

The main function code is as follows:

```
void main()  
{  
    initLed();  
    initSleepTimer();  
  
    while(1)  
    {  
        uint8_t i;  
        for (i = 0; i < 6; i++) {  
            LED = (LED == LED_ON)?LED_OFF: LED_ON;  
            delayMs(SYSLCK_32MHZ,250);  
        }  
  
        DEBUG_LOG("Sleeping...\r\n");  
  
        setSleepPeriod(5);  
        setPowerMode(POWER_MODE_PM2);  
    }  
}
```

After the main function is initialized, it does two things: flashes the LED light three times, then sets the sleep timer to 5 seconds, then sets the power management mode to PM2, and then starts to enter the sleep state (CPU does not work).

Sleep timer interrupt service function (the sleep timer will be interrupted when the time is up, and the CPU will start working again in the interrupt handler):

```
/*
```

```
* Sleep timer ISR
*/
#pragma vector = ST_VECTOR
__interrupt void SleepTimer_ISR(void)
{
    STIF = 0;           // Clear interrupt flag
    setPowerMode(POWER_MODE_ACTIVE); // Entry active power mode

    DEBUG_LOG("Activating...\r\n");
}
```

Initialize the sleep timer function:

```
static void initSleepTimer(void)
{
    ST2 = 0;
    ST1 = 0;
    ST0 = 0;

    STIE = 1;
    STIF = 0;
    EA = 1;
}
```

Set the sleep timer timing, that is, read the current count value, add a new value, and reset the counter:

```
static void setSleepPeriod(uint8_t nS)
{
    uint32_t sleepTimer = 0;
    sleepTimer |= (uint32_t)ST0;
    sleepTimer |= (uint32_t)ST1 << 8;
    sleepTimer |= (uint32_t)ST2 << 16;

    sleepTimer += (uint32_t)nS * 32768;

    ST2 = (uint8_t)(sleepTimer >> 16);
    ST1 = (uint8_t)(sleepTimer >> 8);
    ST0 = (uint8_t)(sleepTimer);
}
```

To set the power mode (low power mode):

```
static void setPowerMode(PowerMode_t mode)
{
    if (mode > _POWER_MODE_MAX) {
        DEBUG_LOG("Power mode not found: %d\r\n", (int)mode);
        return;
}
```

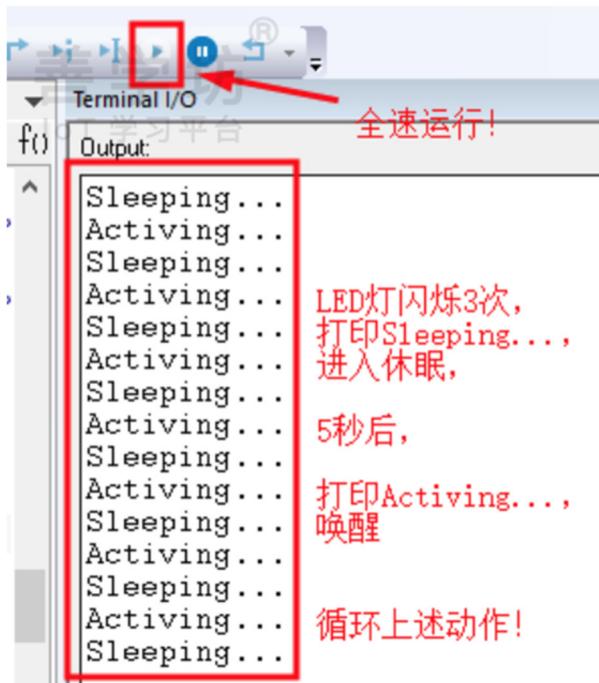
```
if (mode == POWER_MODE_ACTIVE) {  
    PCON = 0x00;  
  
    return; // Don't sleep  
}  
  
SLEEPCMD |= mode; // Set power mode  
  
PCON = 0x01; // Entering sleep mode
```

The sleep timer has four power modes, which we define as type PowerMode t:

```
/*
 * Power modes: Active mode, PM1, PM2, PM3
 */
typedef enum
{
    POWER_MODE_IDLE = 0,
    POWER_MODE_PM1 = 1,
    POWER_MODE_PM2 = 2,
    POWER_MODE_PM3 = 3,
    POWER_MODE_ACTIVE = 4,
    _POWER_MODE_MAX = POWER_MODE_ACTIVE
}PowerMode_t;
```

Debugging Simulation

After the compilation is complete, connect the development board and the computer through the emulator and enter the simulation mode:



As you can see, the development board prints Sleep... after the LED flashes 3 times, and CC2530 enters the sleep state. The sleep timer will trigger an interrupt after the timing time is up, and the interrupt service function will make the chip re-enter the normal working state.

5.4. Chapter 4: Serial Communication Experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=16>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

This chapter mainly explains how to use the serial port 0 of CC2530 to send and receive strings with the host computer. Readers will learn how to use the serial port assistant to send strings to the development board, and use the development board to send strings to the serial port assistant and display them.

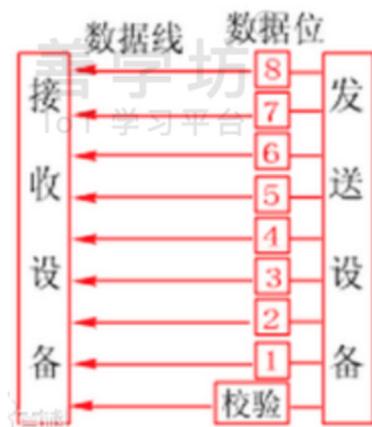
UART Principle Description

USART

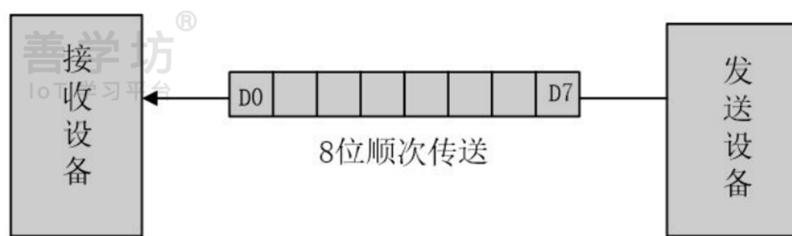
stands for Universal Synchronous/Asynchronous Receiver/Transmitter. CC2530 has two built-in USARTs, USART0 and USART1. USART includes the concepts of serial communication, synchronous communication and asynchronous communication, which will be introduced below.

Parallel and parallel communication

- Parallel communication refers to sending each data bit at the same time. The schematic diagram of sending 8 bits using parallel communication is shown in the figure.



- Serial communication refers to sending individual bits one by one. The schematic diagram of sending 8 bits through serial communication is shown in the figure.



Generally speaking, parallel communication is faster than serial communication, but requires more data pins.

Asynchronous communication and synchronous communication

Let's take a simple example to illustrate the main difference between asynchronous communication and synchronous communication:

- (1) Asynchronous communication is similar to sending text messages on a mobile phone. Its characteristics are that it can be sent at any time and only one message can be sent at a time.

(2) Synchronous communication is similar to making a phone call. Its characteristics are that you can only talk after the other party answers the call, and you can chat (communicate) as long as you want after the other party answers the call.

1. Characteristics of asynchronous communication

(1) The receiving device must always be ready to receive data.

(2) The sending device can send data at any time.

(3) The sending device can only send one data frame at a time, that is, a long piece of data needs to be divided into multiple data frames, and the composition format of the data frame is specified as follows:

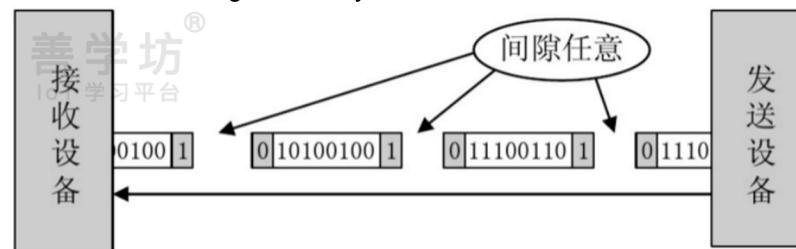
A. 1 start bit.

B. 5, 7 or 8 data bits, that is, the information to be transmitted.

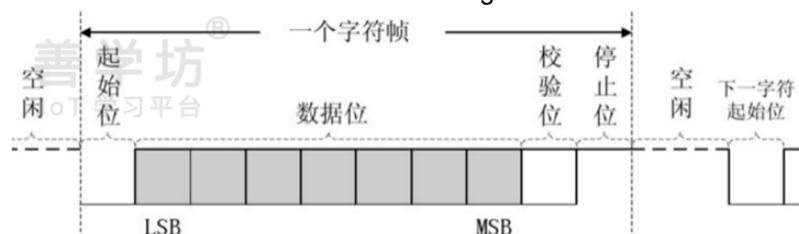
C. 1 parity bit.

D. 1 to 2 stop bits, which is specified as 1.

The schematic diagram of asynchronous communication is shown in the figure.



The data frame format is shown in the figure.



2. Characteristics of synchronous communication

(1) The sending device must synchronize its clock frequency with the receiving device before sending a message.

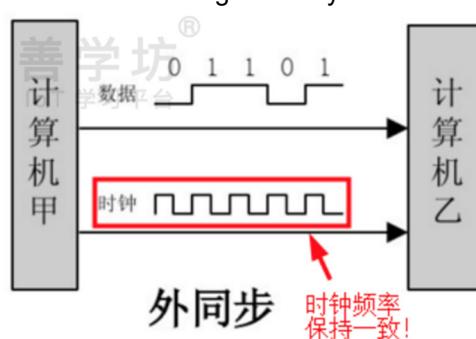
(2) The sending device sends a data block (which can be understood as multiple bytes) each time, and the message format is as follows:

A. 2 synchronization characters as the start mark of a data block.

B. Multiple continuously transmitted data bytes.

C. 2-byte CRC code. CRC stands for Cyclical Redundancy Check, which is a commonly used error checking code in the field of data communication. The data receiver can use this code to determine whether the received data is garbled.

The schematic diagram of synchronous communication is shown in the figure.



The format of the data block is shown in the figure.



3. UART

UART stands for Universal Asynchronous Receiver/Transmitter, also known as serial port. It can be understood as USART without the support for synchronous communication. It is usually used for communication between host and embedded devices. For example, the supporting ZigBee development board can use UART to communicate with the PC host.

4. SPI

SPI stands for Serial Peripheral Interface, a type of USART. It is a communication bus with the characteristics of fast speed, full-duplex and synchronous communication.

Related Registers

CC2530 has two serial ports, serial port 0 and serial port 1. This section uses serial port 0 as an example to explain how to use serial port communication. P0_2 and P0_3 can be used as RX (receiver) and TX (transmitter) of serial port 0, respectively. In addition, P1_5 and P1_4 can also be used as TX and RX of serial port 0, respectively. Since the ZigBee development board uses P0_2 and P0_3 by default, it is necessary to configure the relevant registers of P0_2 and P0_3.

U0CSR

U0CSR is the USART0 control and status register. The functions of its various bits are as follows:

- (1) Bit[7] indicates USART mode. A value of 0 indicates SPI mode and a value of 1 indicates UART mode.
- (2) Bit[6] indicates whether the UART receiver is enabled. A value of 0 indicates disabling the receiver and a value of 1 indicates enabling the receiver.
- (3) Bit[5] indicates SPI master or slave mode. A value of 0 indicates SPI master mode and a value of 1 indicates SPI slave mode.
- (4) Bit[4] indicates the stop bit error status of the UART frame. A value of 0 indicates no error and a value of 1 indicates an error.
- (5) Bit[3] indicates the UART parity error status. A value of 0 indicates no error and a value of 1 indicates an error.
- (6) Bit[2] indicates the transmit byte status. A value of 0 indicates no byte is received and a value of 1 indicates ready to receive a byte.
- (7) Bit[1] indicates the transmit byte status. A value of 0 indicates no byte is transmitted and a value of 1 indicates the last byte written to the data buffer register is transmitted.
- (8) Bit[0], USART transmit/receive active status. In SPI slave mode, this bit is equal to the slave mode selection. 0: USART idle, 1: USART busy.

U0GCR

U0GCR is the USART0 general control register. The functions of its various bits are as follows:

- (1) Bit[7] indicates the SPI clock polarity. A value of 0 indicates a negative clock polarity and a value of 1 indicates a positive clock polarity.
- (2) Bit[6] indicates the SPI clock phase. A value of 0 indicates that when SCK inverts from CPOL to CPOL, data is output to MOSI, and when SCK inverts from CPOL to CPOL, data is sampled to MISO; a value of 1 indicates that when SCK inverts from CPOL to CPOL, data is output to MOSI, and when SCK inverts from CPOL to CPOL, data is sampled to MISO.
- (3) Bit[5] indicates the transmission bit order. A value of 0 indicates that LSB is transmitted first, and a value of 1 indicates that MSB is transmitted first.
- (4) Bit[4:0] indicates the baud rate index value BAUD_E.

U0BAUD

represents the value of the fractional part of the baud rate, BAUD_M. BAUD_E and BAUD_M determine the baud rate of the UART and the main SCK clock frequency of the SPI.

U0DBUF

U0DBUF is the buffer register of USART0, which is used for buffering when receiving or sending data.

UTX0IF

UTX0IF is the TX interrupt flag register of USART0. A value of 0 indicates no interrupt is pending, and a value of 1 indicates an interrupt is pending.

URX0IE

URX0IE is the RX interrupt enable register of USART0. A value of 0 indicates interrupt disabled, and a value of 1 indicates interrupt enabled.

CLKCONCMD

CLKCONCMD is the clock control command register. The functions of its bits are as follows:

- (1) Bit [7] is used to configure the 32 kHz clock oscillator. A value of 0 indicates 32 kHz XOSC and a value of 1 indicates 32 kHz RCOSC.
- (2) Bit [6] is used to configure the system clock source. A value of 0 indicates 32 MHz XOSC and a value of 1 indicates 16 MHz RCOSC.
- (3) Bit [5:3] is used to configure the timer mark output. The configuration method is shown in the table.

value	significance
000	32 MHz
001	16 MHz
010	8 MHz
011	4 MHz
100	2 MHz
101	1 MHz
110	500 kHz
111	250 kHz

(4) Bit [2:0] is used to configure the clock speed. The configuration method is shown in the table.

value	significance
000	32 MHz
001	16 MHz
010	8 MHz
011	4 MHz
100	2 MHz
101	1 MHz
110	500 kHz
111	250 kHz

CLKCONSTA

The CLKCONSTA register is a read-only register used to obtain the current clock status.

PERCFG

PERCFG is a peripheral controller, in which bit 0 is used to configure the position of USART0. A value of 0 means using position 1, that is, using P0_2 and P0_3; a value of 1 means using position 2, that is, using P1_4 and P1_5.

P2DIR

P2DIR is used to control the direction of port 2 and the priority of the peripheral function of port 0. Bit[7:6] is used to control the priority of the peripheral function of port 0. When the value is 00, it means that port 0 is used as USART0 first.

URX0IF

URX0IF is the serial port 0 RX interrupt flag register, in which Bit[1] is used for USART0 receive interrupt flag.

The accompanying video course will explain the above registers in detail

Configuring Serial Port 0

According to the above-mentioned related register descriptions, the configuration steps of serial port 0 are as follows:

(1) Configure the position of serial port 0 through the peripheral control register PERCFG, that is, configure P0_2 and P0_3 to be used as peripheral interfaces instead of ordinary IO ports, and then configure P0 to be used as USART0 first. The code is as follows:

```
PERCFG = 0x00;//配置P0_2和P0_3用作串口0的TX（发送端）和RX（接收端）
```

```
P0SEL = 0x3c;//配置P0_2和P0_3用作外设功能·而不是GPIO
```

```
P2DIR &= ~0xc0;//配置端口0优先用作USART0
```

(2) Set the USART0 working mode to UART0. The code is as follows:

```
U0CSR |= 0x80;
```

(3) Set the baud rate of the serial communication to 115200. The configuration code is as follows:

```
U0GCR |= 11;//BAUD_E
```

```
U0BAUD |= 216;//BAUD_M
```

The commonly used baud rate configurations are shown in the table.

Baud rate (bps)	UxBAUD.BAUD_M	UxGCR_BAUD_E	error(%)
2400	59	6	0.14
4800	59	7	0.14
9600	59	8	0.14
14400	216	8	0.03
19200	59	9	0.14
28800	216	9	0.03
38400	59	10	0.14
57600	216	10	0.03
76800	59	11	0.14
115200	216	11	0.03
230400	216	12	0.03

(4) Configure interrupt related registers. The code is as follows:

```
UTX0IF = 0;// 清零UART0发送中断标志位
```

```
URX0IF = 0;// 清零UART0接收中断标志位
```

```
URX0IE = 1;//使能串口0接收中断
```

```
EA = 1;//打开中断总开关
```

(5) Enable the data receiving function of serial port 0. The code is as follows:

```
U0CSR |= 0x40;//启用数据接收功能
```

Implementation of writing serial port data transmission and reception

Open the uart0.c file in this experiment code and find the main function. The code is as follows:

```
//2.51单片机入门/4.串口通信实验/Workspace/code/uart/uart0.c
```

```
void main()
{
    setSystemClk32MHZ();//设置系统时钟频率为32MHz
    initUart0();//初始化串口0

    while(1){}
}
```

setSystemClk32MHZ function

By default, as mentioned in the previous section, the clock frequency of CC2530 is 16MHz. Developers can use a 32MHz external crystal oscillator as the system clock source to increase the clock frequency of CC2530 to 32MHz, thereby increasing the processing speed of CC2530. The configuration steps are as follows:

- (1) Configure an external 32MHz crystal oscillator as the system clock source through CLKCONCMD.
- (2) Wait for the clock source to stabilize.
- (3) Set the system clock frequency to 32MHz through CLKCONCMD.

The corresponding configuration code is as follows:

```
/** @brief 设置系统时钟为32MHz
*/
#define setSystemClk32MHZ() do {
    CLKCONCMD &= ~0x40;
    while(CLKCONSTA & 0x40);
    CLKCONCMD &= ~0x47;
} while(0)
```

Initialization of Serial Port 0

Through the above analysis, the definition code of the Serial Port 0 initialization function initUart0 is as follows:

```
/*
 * 初始化串口0
*/
static void initUart0(void)
{
    PERCFG = 0x00;//配置P0_2和P0_3用作串口0的TX（发送端）和RX（接收端）

    P0SEL = 0x3c;//配置P0_2和P0_3用作外设功能，而不是GPIO

    P2DIR &= ~0xc0;//配置端口0优先用作USART0

    U0CSR |= 0x80;//设置USART0工作模式为UART0

    /* 设置波特率为115200 */
    U0GCR |= 11;//BAUD_E
```

```
U0BAUD |= 216;//BAUD_M
```

```
UTX0IF = 0;// 清零UART0发送中断标志位  
URX0IF = 0;// 清零UART0接收中断标志位  
URX0IE = 1;//使能串口0接收中断  
EA = 1;//打开中断总开关
```

```
U0CSR |= 0x40;//启用数据接收功能
```

```
}
```

Data reception of serial port 0

After initializing serial port 0, CC2530 will generate an interrupt after receiving data from serial port 0. Developers only need to process the received data in the corresponding interrupt handling function. The code is as follows:

```
#pragma vector = URX0_VECTOR  
__interrupt void URX0_ISR(void)  
{  
    uint8_t rxChar;  
  
    URX0IF = 0;//清零中断标志位  
  
    rxChar = U0DBUF;//U0DBUF存放了从串口0接收到的数据  
    uart0Send(&rxChar, 1); //通过串口0发送数据，即把从串口0接收到的数据又从串口0发送回去  
}
```

Data transmission of serial port 0

You can use serial port 0 to send data to the host computer (such as the serial port assistant). The steps to send a byte are as follows:

- (1) Store the data in the serial port 0 data buffer register U0DBUF. The data in U0DBUF will be automatically sent out through serial port 0.
- (2) Wait for the transmission to be completed. After the data is sent, the interrupt flag is set to 1.
- (3) Clear the interrupt flag.

In the previous serial port data receiving interrupt processing function, uart0Send is called. uart0Send is a serial port 0 data sending function written by the author. Its function definition code is as follows:

```
/*  
 * @fn    uart0Send  
 *  
 * @brief 通过串口0发送数据  
 *  
 * @param pMsg - 待发送数据的地址  
 * @param msgLen - 待发送数据的长度，以字节为单位  
 *  
 * @return  none  
 */  
  
static void uart0Send(uint8_t *pMsg, uint8_t msgLen)  
{
```

```

uint8_t i;

for (i = 0; i < msgLen; i++) {
    U0DBUF = pMsg[i];//把数据存放到串口0数据缓存寄存器U0DBUF中

    while (UTX0IF == 0);//等待发送完成

    UTX0IF = 0;//清零中断标志位
}

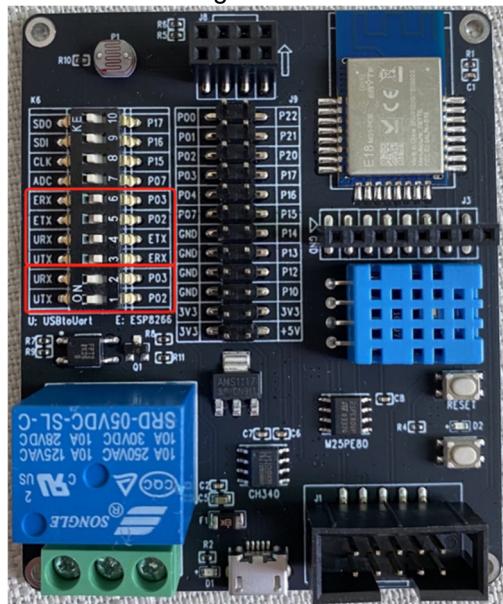
}

```

Debugging Simulation

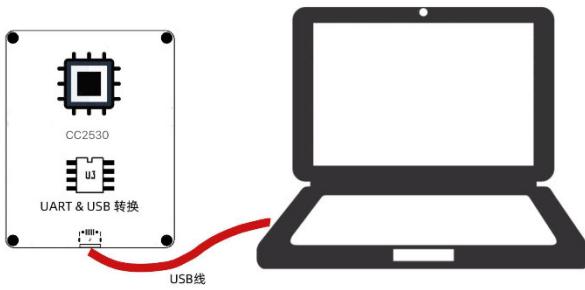
You can run this experimental code to observe the running results. The operation steps are as follows:

- (1) After compiling and linking this experimental code, burn the program into the ZigBee development board.
- (2) Since the program needs to use serial port 0, if you use the ZigBee standard board for testing, you need to set the 1st and 2nd positions of the dip switch to URX and UTX respectively, and the 3rd, 4th, 5th and 6th positions to ERX, ETX, P02 and P03, as shown in the figure.



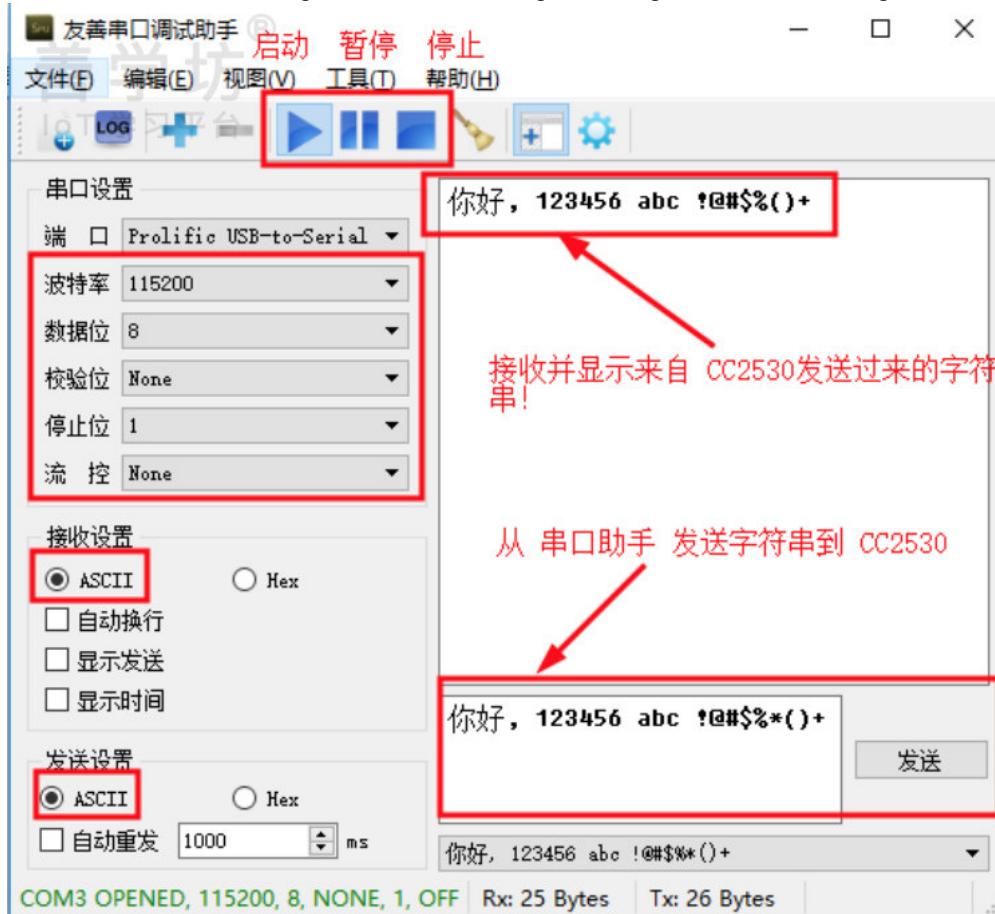
Tip: If you use the ZigBee Mini board for testing, no additional configuration is required.

- (3) Use a Micro USB cable to connect the development board to the computer.



Note: Use the " Mirco USB cable " to connect to the computer instead of using an emulator.

(4) Open the friendly serial port debugging assistant and set the baud rate, data bit, parity bit, stop bit and flow control, as well as the character encoding method for sending/receiving as shown in the figure, and then click the "Start" button.



(5) Enter a string in the input box and click Send. You can see that the data sent from the serial port assistant to the development board will be sent back from the development board.

If you have not installed the serial port debugging assistant, please go to the "Part 1" section and follow the prompts to download and install it.

5.5. Chapter 5: ADC Experiment - Using the Light Sensor

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=17>

Technical support instructions:

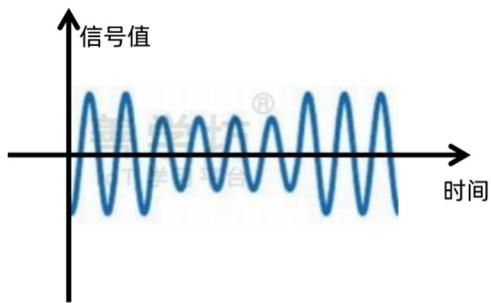
1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Introduction to ADC Principle

- Analog Signal

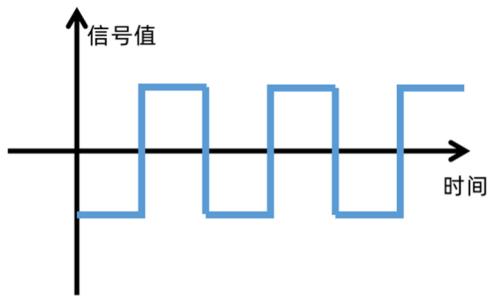
Analog signal refers to information expressed by **continuously changing physical quantities**. Its characteristic is

that the change of signal value is continuous and uninterrupted, as shown in the figure.



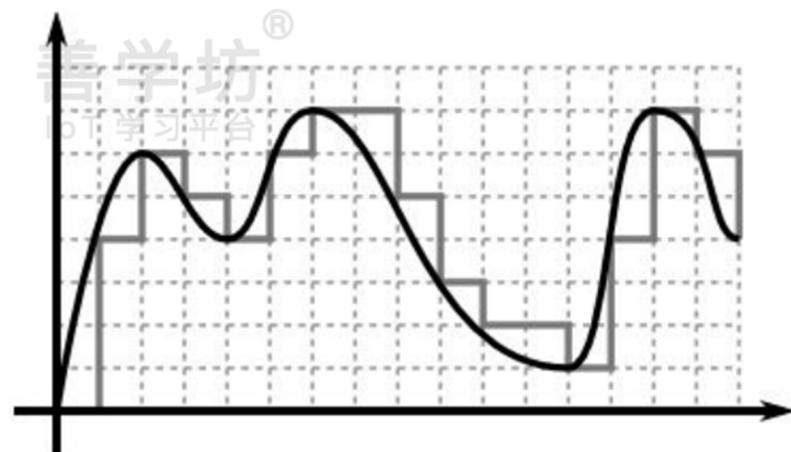
- **Digital**

Signal A digital signal is information expressed by a series of intermittently changing quantities. Its characteristic is that the signal value changes suddenly, as shown in the figure below.



- **ADC**

ADC stands for Analog to Digital Converter, which is used to convert analog signals into digital signals. For example, the black line in the figure is an analog signal, and ADC can convert it into a step-shaped digital signal through specific rules or algorithms.



ADC Application Introduction

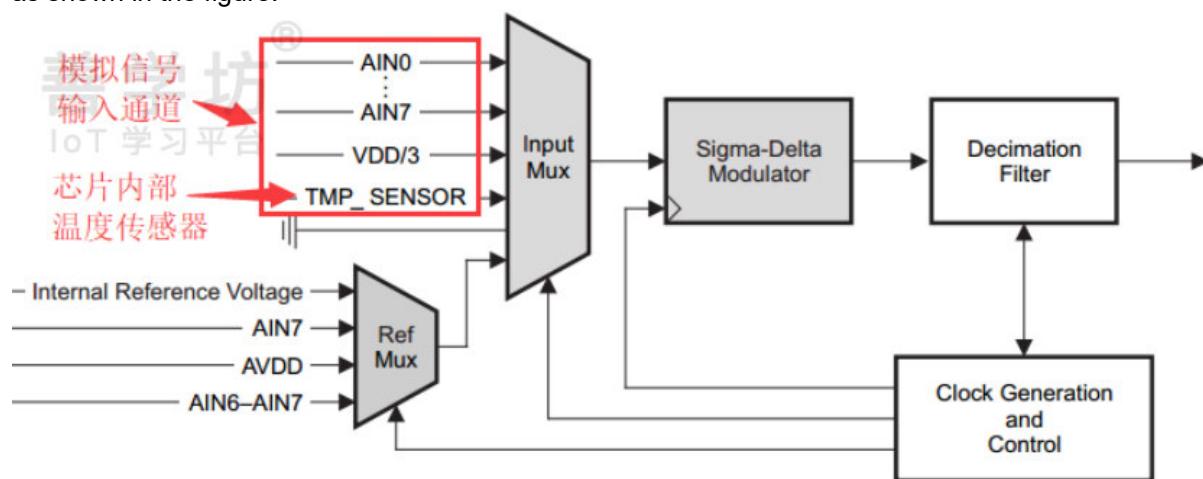
ADC is widely used. For example, the signals collected by some commonly used simple sensors are all analog signals, as shown in the figure.



- Sensor introduction: https://www.zhihu.com/column/c_1488457014951493632
- Sensor store: <https://item.taobao.com/item.htm?id=683757286045>

CC2530 ADC sampling channel

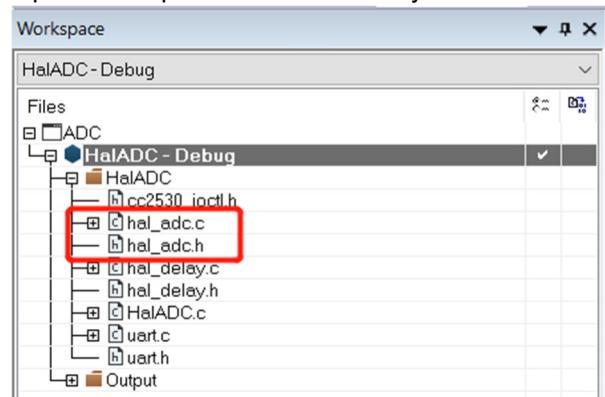
CC2530 has multiple built-in ADC sampling channels, such as AIN0~AIN7, VDD/3, and the chip's built-in temperature sensor, as shown in the figure.



The ADC sampling channels AIN0 - AIN7 correspond to the eight GPIOs P0_0 to P0_7, that is, the analog signal can be input to CC2530 through these eight GPIOs. After CC2530 receives the analog signal from these GPIOs, it can use ADC to obtain the digital signal.

ADC API Description

Open this experimental code and you can find the hal_adc.h and hal_adc.c files, as shown in the figure.



These two files are the ADC driver API provided by Z-Stack 3.0 for developers, which are very convenient to use.

Tip: To make it easier for beginners to learn, the author made some minor modifications to the API.

Open the hal_adc.h file and you can find the following API definition:

```
//2. 51单片机入门/5. ADC实验/Workspace/code/HalADC/hal_adc.h

/*
 * 使用默认的配置初始化ADC服务，使用ADC服务前必须先调用此API
 */
extern void HalAdcInit( void );

/*
 * A/D转换，即使用指定的采样精度从指定的ADC通道中获取采样值
 *
 * @param channel - ADC通道
 * @param resolution - 采用分辨率
 */
extern uint16 HalAdcRead( uint8 channel, uint8 resolution );
```

The HalAdcRead function needs to pass in a channel parameter, which represents the ADC sampling channel. The sampling channel definition code is as follows:

```
//2. 51单片机入门/5. ADC实验/Workspace/code/HalADC/hal_adc.h

#define HAL_ADC_CHN_AIN0 0x00 /*AIN0*/
#define HAL_ADC_CHN_AIN1 0x01 /*AIN1*/
#define HAL_ADC_CHN_AIN2 0x02 /*AIN2*/
#define HAL_ADC_CHN_AIN3 0x03 /*AIN3*/
#define HAL_ADC_CHN_AIN4 0x04 /*AIN4*/
#define HAL_ADC_CHN_AIN5 0x05 /*AIN5*/
#define HAL_ADC_CHN_AIN6 0x06 /*AIN6*/
#define HAL_ADC_CHN_AIN7 0x07 /*AIN7*/
```

The HalAdcRead function also needs to pass in a parameter resolution to represent the sampling resolution. The definition code is as follows:

```
//2. 51单片机入门/5. ADC实验/Workspace/code/HalADC/hal_adc.h

#define HAL_ADC_RESOLUTION_8 0x01
#define HAL_ADC_RESOLUTION_10 0x02
#define HAL_ADC_RESOLUTION_12 0x03
#define HAL_ADC_RESOLUTION_14 0x04
```

CC2530 provides 8-bit, 10-bit, 12-bit and 14-bit analog-to-digital conversion sampling resolutions for developers to use. Let's take an example to explain the meaning of sampling resolution. For example, the photoresistor (light sensor) on the ZigBee standard board can output different voltages according to different light intensities. The higher the light intensity, the greater the output voltage, and vice versa. Its output voltage value range is 0~1.65v, which is an analog signal value. If the sampling resolution is 8 bits at this time, CC2530 will use a 1-byte (8-bit) integer variable to store the sampling value. CC2530 will map the voltage value of 0~3.3v to [0,255] and 0~1.65v to [0,127] to obtain the sampling value. For example, when the photoresistor

is affected by light, if:

- (1) the voltage value captured by the ADC is about 1.65v, then the sampling value is about 127.
- (2) the voltage value captured by the ADC is about 0v, then the sampling value is about 0.
- (3) The voltage value captured by ADC is about 0.825V, so the sampling value is about 63.

If you read the sample value of P0_7 with an 8-bit sampling resolution, you can write the code as follows:

```
P1SEL |= 0x80;//把P0_7配置为使用外设功能模式
```

```
uint8 Val;//采样信号值
```

```
Val = HalAdcRead(
```

```
    HAL_ADC_CHN_AIN7,//采样通道
```

```
    HAL_ADC_RESOLUTION_8);//采样分辨率
```

ADC Sampling Experiment

Since the light sensor is connected to the P0_7 pin of CC2530 in the matching ZigBee standard board, this experiment configures P0_7 as input mode and then reads the sample value of ADC channel AIN7 with 8-bit resolution. The code is as follows:

```
//2.51单片机入门/5. ADC实验/Workspace/code/HalADC/HalADC.c
```

```
void main(void)
```

```
{
```

```
    uint8_t adcVal;//采样值
```

```
    char adcStr[10] = {0};
```

```
    setSystemClk32MHZ();
```

```
    initUart0(USART_BAUDRATE_115200);
```

```
    CC2530_IOCTL(
```

```
        0, 7,//采样通道AIN7对应P0_7
```

```
        CC2530_INPUT_PULLUP);//配置为上拉输入
```

```
    HalAdcInit();//初始化ADC服务
```

```
    while(1) {
```

```
        /*ADC采样*/
```

```
        adcVal = HalAdcRead(
```

```
            HAL_ADC_CHN_AIN7,//采样通道AIN7对应P0_7
```

```
            HAL_ADC_RESOLUTION_8);//采样分辨率为8位
```

```
        /*通过串口0输出采样值*/
```

```
        sprintf(adcStr, "%d\r\n", adcVal);
```

```
        uart0Send((unsigned char *)adcStr, strlen(adcStr));
```

```

delayMs(SYSCLK_32MHZ, 2000); //延时2s
}
}

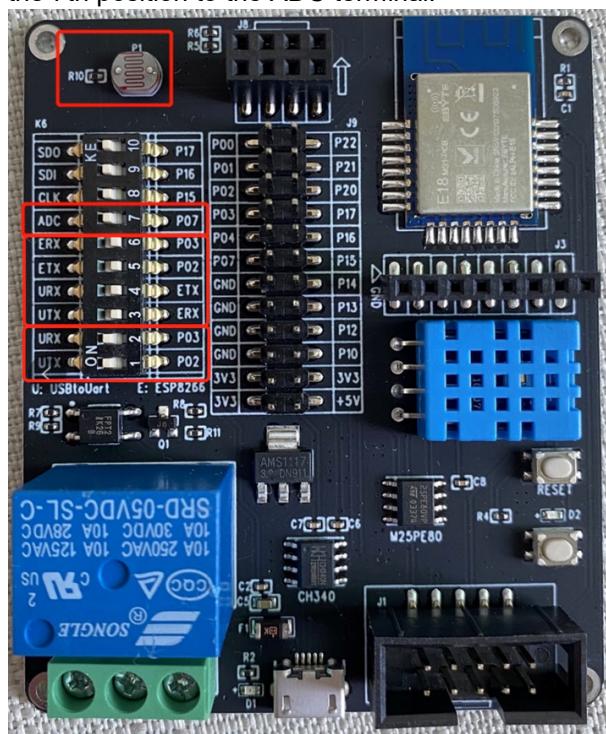
```

The above code collects analog signals from AIN7 every 2 seconds and converts them into digital signals. ADC is implemented simply like this, and readers can also modify the sampling time interval according to actual needs.

Code Testing

You can run this experimental code to observe the running results. The steps are as follows:

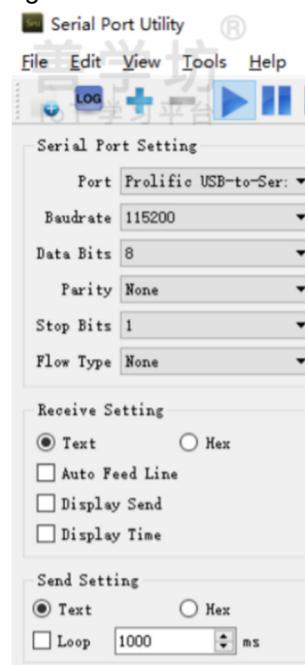
- (1) After compiling and linking this experimental code, burn the program into the matching ZigBee development board.
- (2) Since this experiment requires the use of a light sensor and serial port 0, therefore:
 (2A) If you use the ZigBee standard board for debugging, you need to set the 1st and 2nd positions of the dip switch to the URX and UTX terminals respectively, the 3rd, 4th, 5th and 6th positions to the ERX, ETX, P02 and P03 terminals respectively, and the 7th position to the ADC terminal.



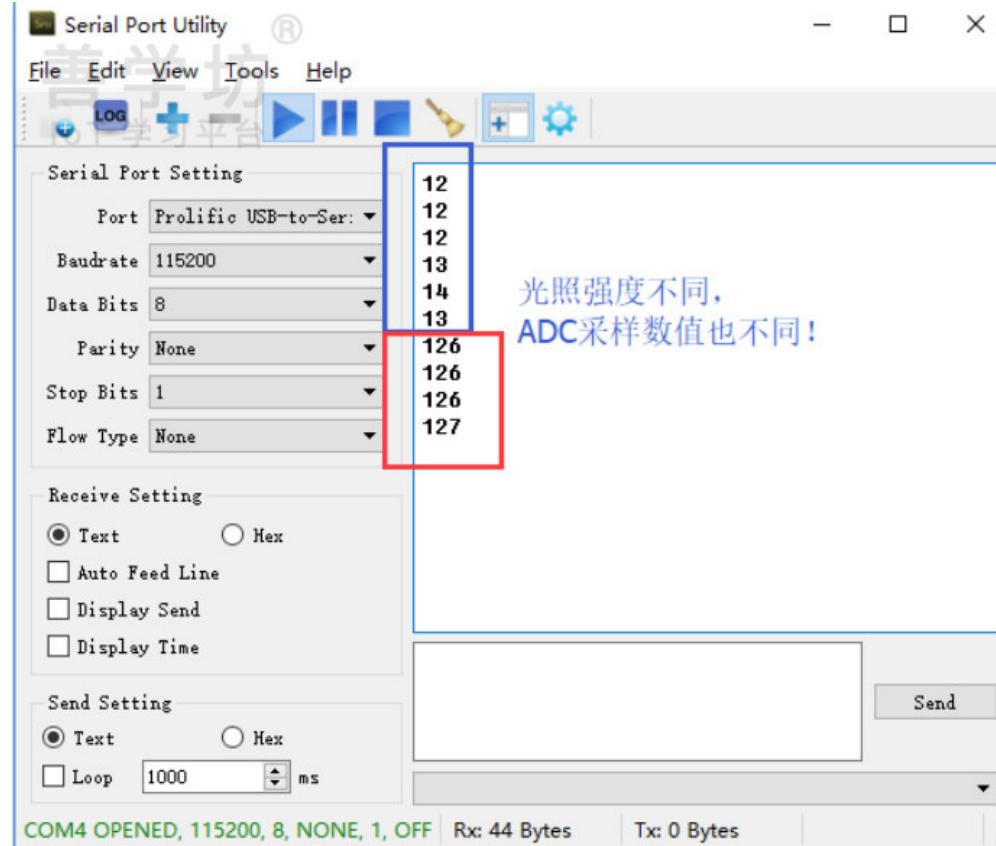
- (2B) If you use the ZigBee Mini board for debugging, you need to connect the analog signal output pin of the light sensor to the P0_7 pin. The output signal of this light sensor will be used as the analog signal of the system input.

- (3) Use a Micro USB cable to connect the ZigBee development board to the computer.

(4) Open the serial port debugging assistant and make relevant settings for the serial port debugging assistant as shown in the figure.



(5) After the configuration is complete, the ADC value can be observed in the serial port assistant.



(6) This value can be understood as the ambient light intensity perceived by the light sensor, which changes with the light intensity.

5.6. Chapter 6: OLED Display Experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=18>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

This lesson will explain how to use the OLED display to display characters and pictures based on CC2530.

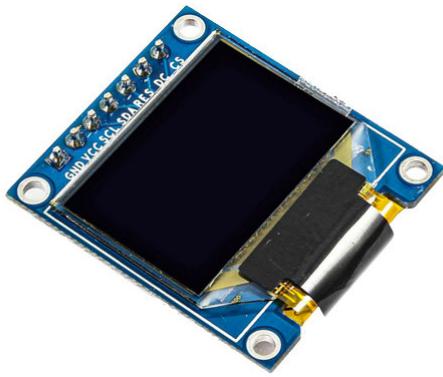
Video Explanation

<https://www.bilibili.com/video/BV1k34y1D7Vz?p=18>

Introduction to OLED Display Technology

(1) OLED stands for Organic Light-Emitting Diode, which has the advantages of self- **luminescence**, wide viewing angle, high contrast and low power consumption.

(2) The actual picture of the 0.96-inch OLED display in the development kit is shown in the figure below. Its resolution is 64×128 pixels, that is, there are 64 pixels vertically and 128 pixels horizontally.



(3) The technical parameters of the display are as follows:

- Model: 0.96OLED12864
- Size: 0.96 inches
- Pixels: 128×64
- Interface: SPI
- Pins: GND, VCC, D0, D1, RES, DC, and CS

Tip: For detailed instructions, please refer to: <https://zhuanlan.zhihu.com/p/628861772>

Introduction to SPI Communication Protocol

SPI is a communication protocol similar to UART. Its full name is Serial Peripheral Interface, which **has** the characteristics of fast transmission speed, simultaneous data transmission and reception, and synchronous communication between the transmitter and the receiver. The matching OLED display uses the SPI communication protocol.

Fonts and images

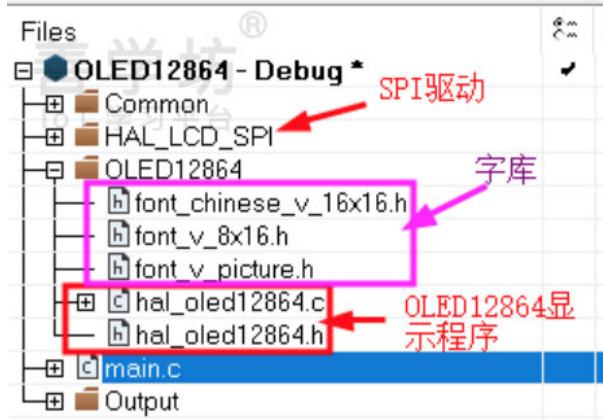
In order for the OLED screen to correctly display various characters or pictures, you also need to prepare a set of fonts and image libraries, such as standard ASCII fonts, Chinese character fonts, and self-made image libraries.

OLED Driver API

In order to drive the OLED screen, the author designed a corresponding driver API for readers to learn and use.

The OLED driver API can be divided into two levels, namely **the upper OLED display API** and **the lower SPI driver API**. The driver API also requires the cooperation of font and image library files. The positions of these three in the experimental code are

shown in the figure.



Introduction to OLED Display API

After configuring the display pins and font/image library files, call the OLED Display API to display the specified content on the screen. Open the hal_oled12864.h file in the OLED122864 folder, and you can find the definition code of the OLED Display API:

```
//2.51单片机入门/6.显示器实验/Workspace/code/HW_LCD/HAL_OLED/hal_oled12864.h

/** @fn    halOLED12864Init
 *
 * @brief  初始化OLED12864显示器，在使用屏幕前必须先调用
 */
void halOLED12864Init(void);

/** @fn    halOLED12864ClearScreen
 *
 * @brief  清除屏幕上显示内容
 */
void halOLED12864ClearScreen(void);

/** @fn    halOLED12864Show
 *
 * @brief  在屏幕上显示字符串，支持的字符格式：1. 8×16 ASCII码；2. 16×16 汉字
 *
 * @param line - 参数值范围：0 ~ 3
 * @param column - 参数值范围：0 ~ 127
 * @param str - 待显示的字符串
 *
 * @warning 16×16汉字
 */

的字库需要先在此文件中定义FONT_TABLE_CHINESE_16×16
*/
void halOLED12864ShowX16(unsigned char line, unsigned char column, const unsigned char *str);
```

```

/**
 * @fn    halOLED12864ShowPicture
 *
 * @brief 在屏幕上显示图片
 *
 * @param x - 指定在横向从左边数起第x个像素开始显示图像，参数值范围：0 ~ 127
 * @param y - 指定在纵向从上边数起的第y个像素开始显示图像，参数值范围：0 ~ 64
 * @param picWidth - 图片的宽度，参数值范围：1~128
 * @param picHeight - 图片的高度，参数值范围：1~64
 * @param pic - 待显示的图片
 */

```

```
void halOLED12864ShowPicture(unsigned char x, unsigned char y, unsigned char picWidth, unsigned char picHeight, const unsigned char *pic);
```

Detailed explanation of halOLED12864ShowX16 usage

The resolution of the OLED12864 display is 64x128 pixels, which can be understood as a two-dimensional table with **64 rows and 128 columns**. The halOLED12864ShowX16 function supports the display of 8×16 standard ASCII characters and 16×16 Chinese characters, where 8×16 refers to the font that occupies 8 rows and 16 columns on the screen, and 16×16 refers to the font that occupies 16 rows and 16 columns on the screen. Therefore, it can be understood that the halOLED12864ShowX16 function supports the display of 4 lines of ASCII characters or Chinese characters on the screen, so the value range of the parameter line is 0~3, but the parameter column still uses pixels to indicate the horizontal position of the pixel at which the character starts to be displayed, so its value range is 0~127.

Before calling the above display API, you need to configure the display pins first, that is, configure the display pins with the GPIO of CC2530. The matching 0.96-inch OLED display contains the following 7 pins:

- 1) GND - ground
- (2) VCC - power supply (2.8~5.5v)
- (3) SCL (D0) - SPI clock
- (4) SDA (D1) - SPI data
- (5) CS - SPI chip select
- (6) RES - screen reset pin
- (7) DC - data or command selection

Open the hal_lcd_spi.h file and you can see the following pin configuration code.

```
//2.51单片机入门/6.显示器实验/Workspace/code/HW_LCD/HAL_LCD_SPI/hal_lcd_spi.h
```

```
#ifndef HAL_LCD_SPI_SW
/* SCL -> CC2530 P1_5 引脚*/
#define HAL_LCD_SPI_SCK_PORT      1
#define HAL_LCD_SPI_SCK_PIN      5

/* SDA -> CC2530 P1_6 引脚*/
#define HAL_LCD_SPI_SDA_PORT      1
#define HAL_LCD_SPI_SDA_PIN      6

#endif
```

```
/* CS -> CC2530 P2_0 引脚*/
#define HAL_LCD_SPI_CS_PORT      2
#define HAL_LCD_SPI_CS_PIN      0
```

```

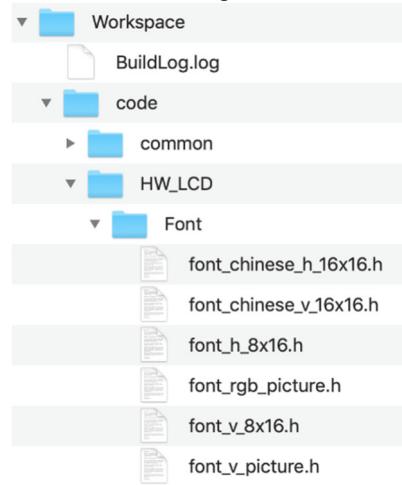
/* DC -> CC2530 P1_4 引脚*/
#define HAL_LCD_SPI_DC_PORT 1
#define HAL_LCD_SPI_DC_PIN 4

/* RES -> CC2530 P1_0 引脚*/
#define HAL_LCD_SPI_RST_PORT 1
#define HAL_LCD_SPI_RST_PIN 0

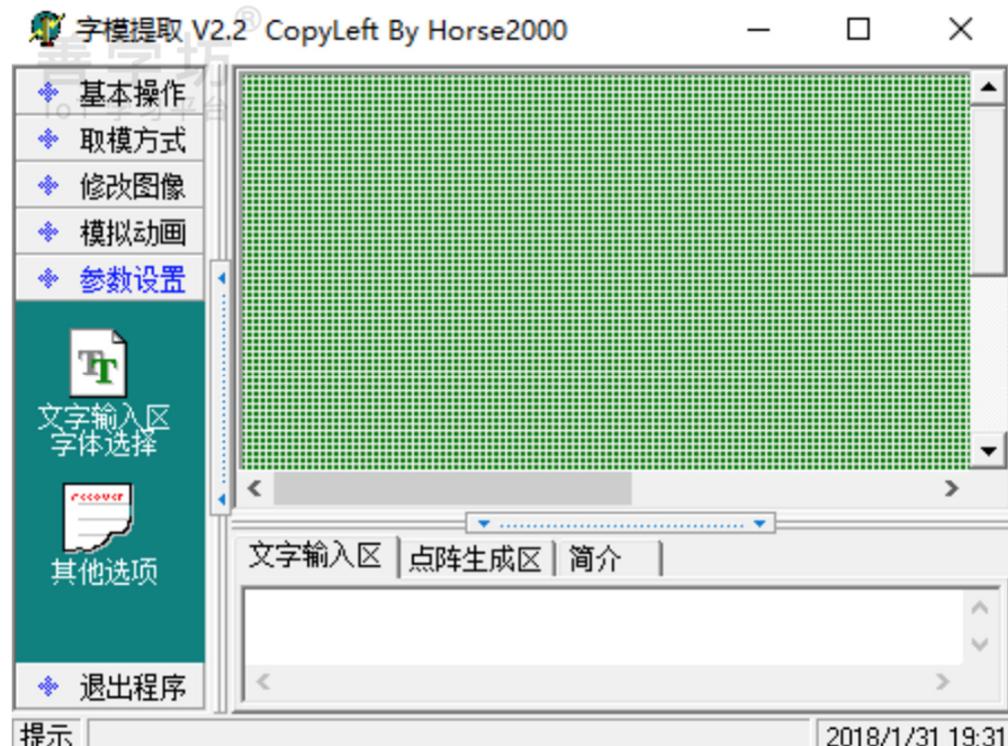
```

The above code completes the configuration of 5 pins, and the VCC and GND pins do not need to be configured. If you need to modify the pin configuration, just modify the number on the right side of the above code.

After configuring the pins, you also need to configure the font library and image library. There are two types of font libraries, namely 8×16 standard ASCII font library and 16×16 Chinese font library. The font library file is located in the experimental code as shown in the figure.



In the software tools that come with this course, you can find the software "Font Model Software 2.2.zip", which is used to generate fonts. Run the font model software, as shown in the figure.

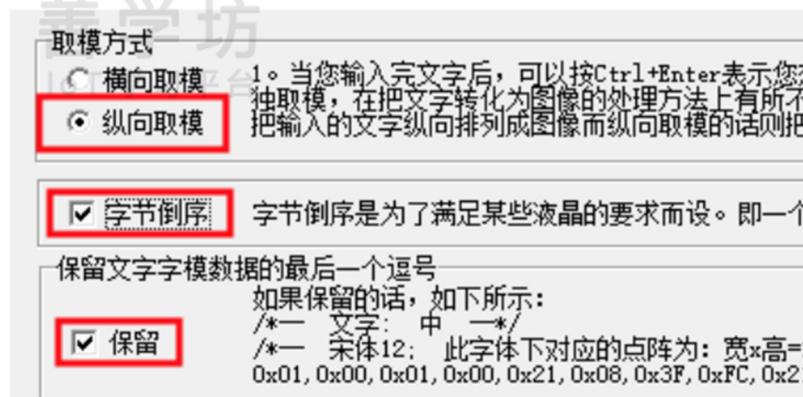


If you have not downloaded the software tools for this course, you can download them from the download page :

8×16 standard ASCII font modulo The modulo steps are as follows:

(1) Click "Parameter Settings" → "Other Options", and select "Vertical Modulo", "Byte Reverse Order" and "Retain" in turn, as shown in the figure.

选项

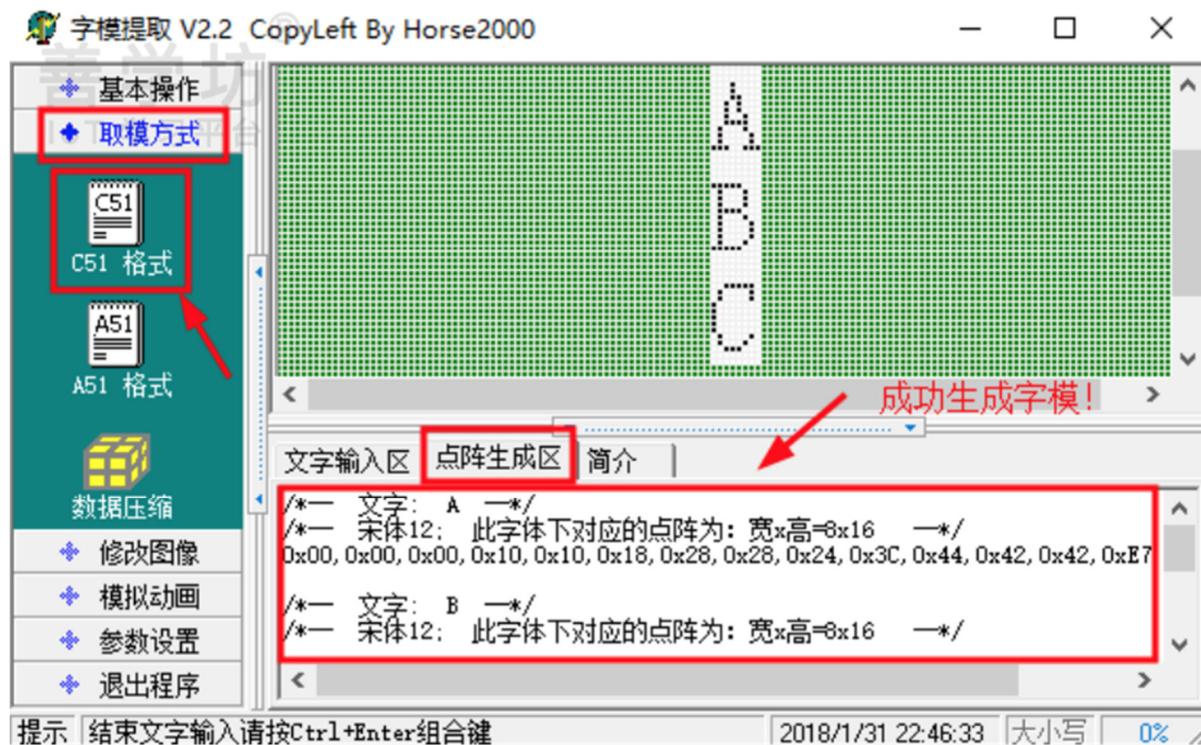


Different screen displays have different requirements for fonts. The matching OLED display requires "vertical modulus" and "byte reverse order".

(2) Enter the text you want to modify in the text input area and press Ctrl+Enter, as shown in the figure.



(3) Click "Modulation Method" → "C51 Format", select "Dot Matrix Generation Area", and you can see the successfully generated font, as shown in the figure.



In this way, the font library corresponding to the 8×16 font is ready. The method of taking the 16×16 Chinese characters is basically the same as that of the 8×16 font. You only need to follow the same method and enter the Chinese characters in the input area.

Configuring the image library

is similar to taking font modulo. Import the image in the basic operation, and then refer to the font modulo process to complete the image modulo, as shown in the figure.



Using the OLED Display API

After configuring the display pins and font files, call the OLED display API to display the content on the screen. Open the supporting experimental code, open main.c in the OLED12864 folder, and you can see the sample code using the OLED display API:

```
//2. 51单片机入门/6. 显示器实验/Workspace/code/OLED12864/main.c
```

```
void main(void)
{
    setSystemClk32MHZ(); //把系统频率设置为32MHz

    halOLED12864Init(); //初始化

    while(1)
    {
        /* Test1 - 显示 8x16 的字符 */

        halOLED12864ShowX16(0, 0, "0123456789"); //在第1行显示

        halOLED12864ShowX16(1, 0, "abcdefghijklABCDE"); //在第2行显示

        halOLED12864ShowX16(2, 0, "{}[]!@#$%"); //在第3行显示

        halOLED12864ShowX16(3, 0, "=====>"); //在第3行显示

        delayMs(SYSLCK_32MHZ, 4000); //延迟

        halOLED12864ClearScreen(); //清空

        /* Test2 - 显示 8x16 字符 和 16x16 汉字 */

        halOLED12864ShowX16(0, 0, "今天气温 : ");

        halOLED12864ShowX16(1, 30, "温度 : 22 °C");

        halOLED12864ShowX16(2, 30, "湿度 : 30 %");

        /* 注意：对于汉字，必须先取字模后存放到汉字字库文件font_chinese_v_16x16.h中 */

        delayMs(SYSLCK_32MHZ, 4000); //延迟

        halOLED12864ClearScreen(); //清空

        /* Test3 - 在坐标 (30像素, 30像素) 处显示分辨率为 32x32 像素的图片 */

        halOLED12864ShowPicture(30, 30, 32, 32, Picture_32x32_Appleco);

        /* 注意：对于图片，系需要先取模后存放在图库文件font_v_picture.h中 */

        delayMs(SYSLCK_32MHZ, 4000); //延迟

        halOLED12864ClearScreen(); //清空

        /* Test4 - 全屏显示 128x64 图片，即在坐标 (30像素, 30像素) 处显示一张分辨率为128x64像素的图片 */

        halOLED12864ShowPicture(0, 0, 128, 64, Picture_128x128_SuccessPic);

        delayMs(SYSLCK_32MHZ, 4000); //延迟
```

```
halOLED12864ClearScreen(); //清空
```

```
}
```

In the actual development process, developers generally only need to apply the above code to use the OLED screen, which is very convenient.

Design principles of SPI driver API

The design principle of the SPI driver API is relatively complex, and readers only need to have a simple understanding of its principles.

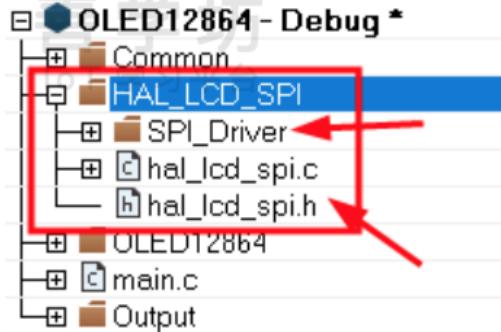
As mentioned above, the driver API of the OLED screen designed by the author can be divided into two levels, namely **the upper OLED display API** and **the lower SPI driver API**. Here is a brief explanation of the design principle of this SPI driver API.

Design idea

The author divides the SPI driver API into two parts, namely **the general SPI driver API** and **the dedicated SPI driver API**.

- (1) **The general SPI driver API** is suitable for a variety of devices that use SPI communication. The SPI_Driver folder in the figure stores this part of the API.
(2) **The dedicated SPI driver API** is based on **the general SPI driver API** and is specifically adapted to specific devices, such as the matching OLED display. The hal_lcd_spi.h and hal_lcd_spi.c in the figure are this part of the API

Files



The benefits of doing this are quite obvious. The universal SPI driver API can be used in various SPI devices. You only need to add corresponding adaptation codes for these devices.

Support for hardware and software SPI modes

In the general SPI driver API, hardware SPI and software SPI modes are supported. Hardware mode SPI refers to the implementation of the SPI protocol using the hardware processing unit, and software SPI mode refers to the implementation of SPI by writing program code. As shown in the figure, hw_spi.h/c is a SPI driver API based on hardware implementation, and sw_spi.h/c is a SPI driver API implemented by software simulation, as shown in the figure.



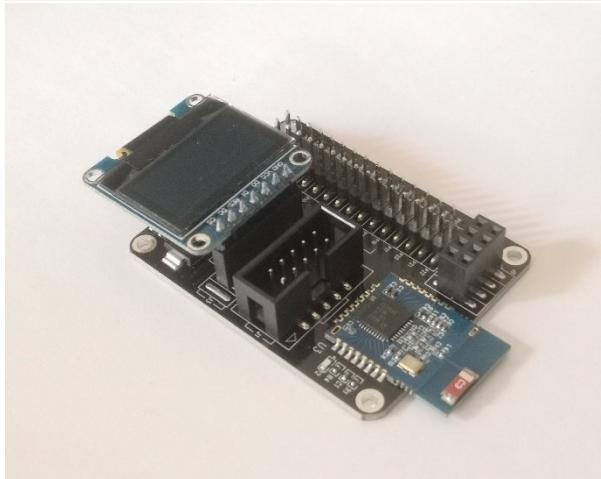
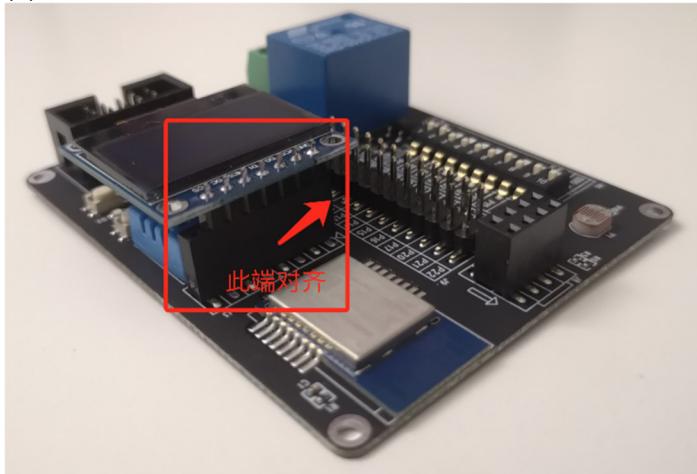
You can choose which method to use by configuring in the adapter programs hal_lcd_spi.h and hal_lcd_spi.c, as shown in the figure.

```
35 /** @brief Config:
36 * HAL_OLED12864_SPI_SW: SW-SPI Bus 硬件实现SPI
37 * HAL_OLED12864_SPI_HW: HW-SPI Bus
38 */
39 #if !defined(HAL_LCD_SPI_SW) && !defined(HAL_LCD_SPI_HW)
40     #define HAL_LCD_SPI_SW ← 软件模拟实现SPI
41 #endif
```

Debugging Simulation

You can run this experimental code to observe the running results. The steps are as follows:

(1) Insert the OLED screen into the standard board or Mini board, as shown in the figure.



(2) Since this experiment requires SPI communication, if you use the ZigBee standard board for testing, you need to set the 8th, 9th and 10th positions of the dip switch to the CLK, SDI and SDO terminals respectively, as shown in the figure.



(3) After compiling and linking this experimental code, burn the program into the matching ZigBee development board, and you can see the screen displaying the corresponding content in a loop.

5.7. Chapter 7: Peripheral Experiments

5.7.1. DHT11 Temperature and Humidity Sensor

5.7.2. NorFLASH reading and writing experiment

5.7.3. Relay Control Experiment

5.7.1. DHT11 Temperature and Humidity Sensor

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=19>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

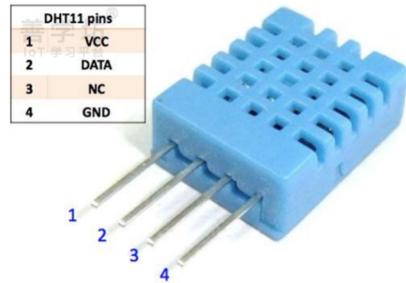
This section will explain how to use the DHT11 temperature and humidity sensor to sense the ambient temperature and humidity, and display the data on the screen. The usage of many sensors is similar. By learning to use DHT11, I believe that readers can learn from it and use more types of sensors.

Introduction to DHT11

The DHT11 digital temperature and humidity sensor is a composite sensor that can detect temperature and humidity. It has a built-in temperature measuring element, a resistive humidity sensing element, and a single-chip microcomputer. Taking the DHT11 temperature and humidity sensor produced by Aosong as an example, its effective measurement range is:

- Temperature: 0~50°C
- Humidity: 20~95%

The actual DHT11 is shown in the figure.



As you can see, DHT11 has 4 pins, their functions are described in the table below.

PIN	name	illustrate
1	VCC	Power supply pin, 3~5.5VDC
2	DATA	Temperature and humidity data output
3	NC	Vacant pins
4	GND	Ground pin, connected to the negative pole of the power supply

The temperature and humidity data can be obtained from the DATA pin. The following is a brief explanation of its communication protocol.

Communication protocol analysis of DATA pin

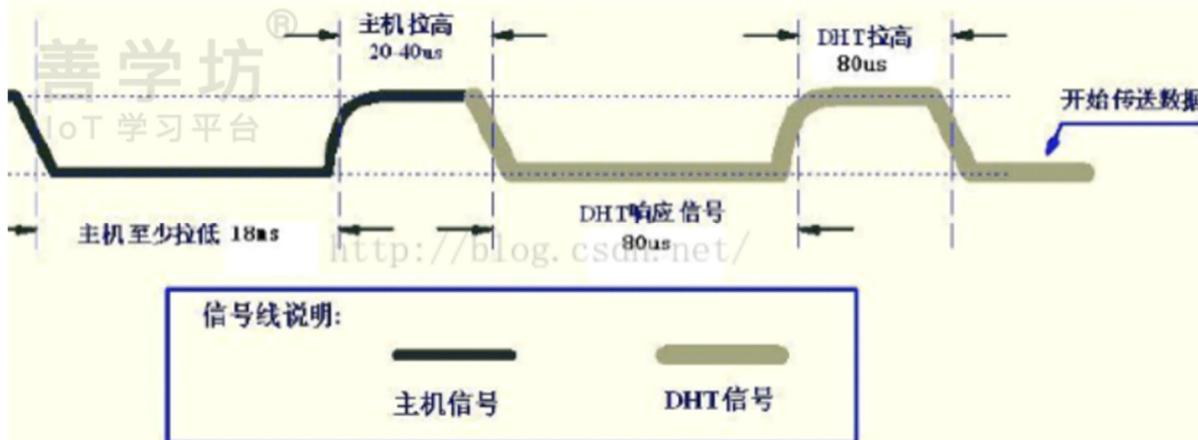
The method of obtaining temperature and humidity data from DHT11 is relatively simple. First, CC2530 is paired with DHT11 (handshake), and then data is received from the DATA pin according to a specific protocol.

Pairing (handshake)

Before sending data, DHT11 needs to pair with CC2530. The pairing protocol is as follows:

- (1) The DATA pin is at a high level (3.3v) in the initial default state.
- (2) CC2530 pulls the DATA pin down for more than 18ms, then pulls it up for 20~40us, and DHT11 is activated. (3) DHT11 will actively pull the DATA pin down for 80us, indicating that it has received the CC2530 command and the pairing is successful.
- (4) Then DHT11 will pull up again, and after 80us, it will start sending temperature and humidity data to CC2530.

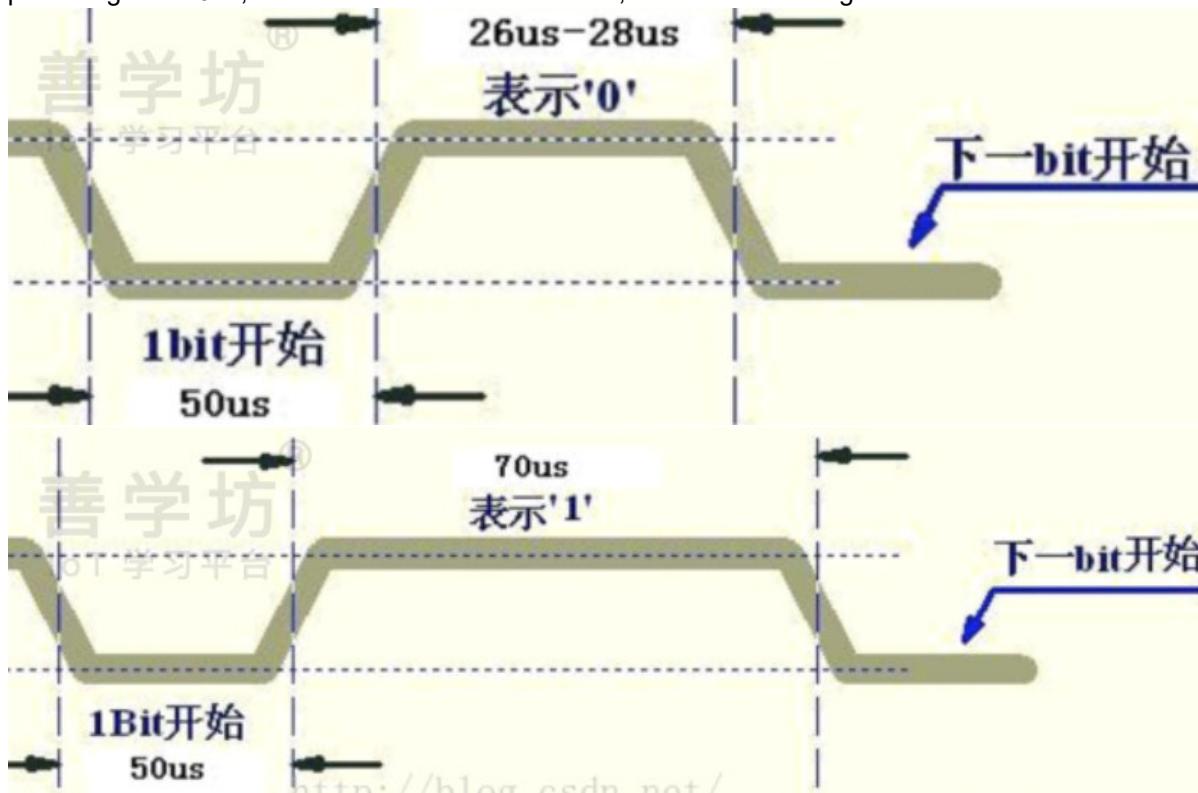
The pairing process is shown in the figure.



The temperature and humidity data received

by DHT11 are represented by binary data. These binary data are sent to CC2530 in a bit-by-bit order. The specific principle is as follows:

- (1) Before sending each bit, DHT11 will pull the level of the DATA pin down for 50us to tell CC2530: "I will send a bit next."
- (2) Then, DHT11 pulls the level of the DATA pin high. If it is pulled high for 26~28us, it means that the data sent is 0; if it is pulled high for 70us, it means that the data sent is 1, as shown in the figure.

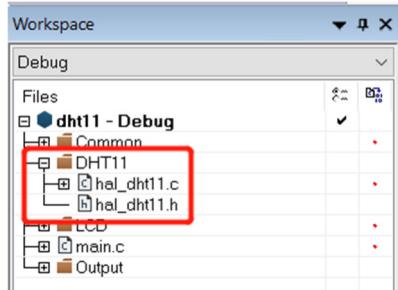


In this way, the temperature and humidity data are sent to CC2530. The communication protocol of DHT11 is roughly introduced, but there are still many details that have not been explained. Interested readers can refer to more relevant information or carefully study the source code of the API introduced next.

DHT11 Driver API Introduction

Based on the above principles, the author has designed a set of DHT11 driver APIs, which are very convenient to use. Open the

DHT11 folder in the supporting project, and you can find the DHT11 driver API, as shown in the figure.



Open the hal_dht11.h file and you can find the API definition code:

```
//2.51单片机入门/7.外设实验/7.1温湿度传感器DHT11/Workspace/code/DHT11_MAIN/hal_dht11.h

/**
 * @fn    halDHT11Init
 *
 * @brief  初始化函数，使用DHT11前必须先调用此函数
 */
void halDHT11Init(void);

/**
 * @fn    halDHT11GetData
 *
 * @brief  获取DHT11的温湿度数据
 *
 * @return 温湿度数据值
 */
halDHT11Data_t halDHT11GetData(void);
```

The halDHT11Data_t is a structure used to store temperature and humidity data. Its definition code is as follows:

```
1./** @brief 用于表示DHT11温湿度数据 */
2.typedef struct {
3.    unsigned char ok; //ok的值非0时温湿度数据才有效
4.    unsigned char temp; //温度值，取值范围：0~50
5.    unsigned char humi; //湿度值，取值范围：20~95
6.} halDHT11Data_t;
```

Using DHT11 Driver API

Before calling

the API, you need to pair the DATA pin with the CC2530 IO port. The ZigBee standard board uses the CC2530 P0_6 pin to connect with the DHT11 DATA pin. The following configuration code can be found in the hal_dht11.h file:

```
1.#define HAL_DHT11_PORT 0 //Port0.
2.#define HAL_DHT11_PIN 6 //Pin6.
```

If the DATA pin needs to be connected to other pins of CC2530 in hardware, just change the pin number here.

API call example

Open the main.c file and you can see the API call example code:

//2. 51单片机入门/7. 外设实验/7.1 温湿度传感器DHT11/Workspace/code/DHT11_MAIN/main.c

```
void main(void)
{
    halDHT11Data_t dht11Dat;//定义温湿度数据结构体
    uint8 tempStr[50], humiStr[50];

    setSystemClk32MHZ();//初始化系统时钟频率

    //初始化显示器
    #ifdef LCD_OLED12864
        //初始化OLED12864屏幕
        halOLED12864Init()
    #else
        //初始化TFT屏幕
        halTFTInit(HAL_TFT_PIXEL_WHITE);
    #endif

    halDHT11Init();//初始化DHT11温湿度传感器

    while(1)
    {
        dht11Dat = halDHT11GetData();//获取温湿度数据

        if (dht11Dat.ok) {//如果数据正确获取

            sprintf((char *)tempStr, "Temp: %d", dht11Dat.temp);//显示温度
            sprintf((char *)humiStr, "Humi: %d", dht11Dat.humi);//显示湿度

            //把数据显示到屏幕中
            #ifdef LCD_OLED12864
                //显示到OLED12863屏幕
                halOLED12864ShowX16(0,0, tempStr);
                halOLED12864ShowX16(1,0, humiStr);
            #else
                //显示到TFT屏幕
                halTFTShowX16(0,0, HAL_TFT_PIXEL_RED, HAL_TFT_PIXEL_WHITE, tempStr);
                halTFTShowX16(0,16, HAL_TFT_PIXEL_RED, HAL_TFT_PIXEL_WHITE, humiStr);
            #endif
        }
    }
}
```

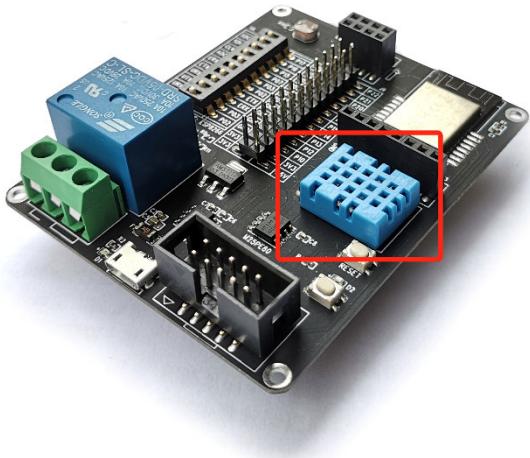
```
delayMs(SYSCLK_32MHZ, 4000); //延时  
} /* while */  
}
```

Debugging Simulation

You can run this experimental code to observe the running results. The steps are as follows:

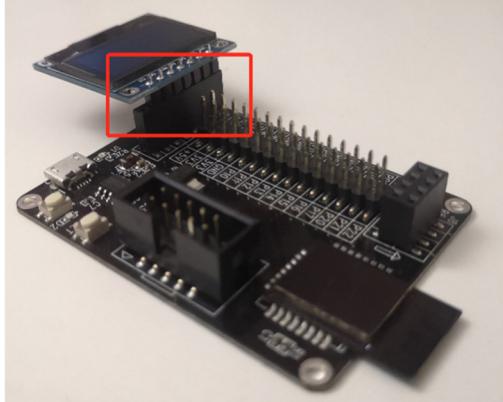
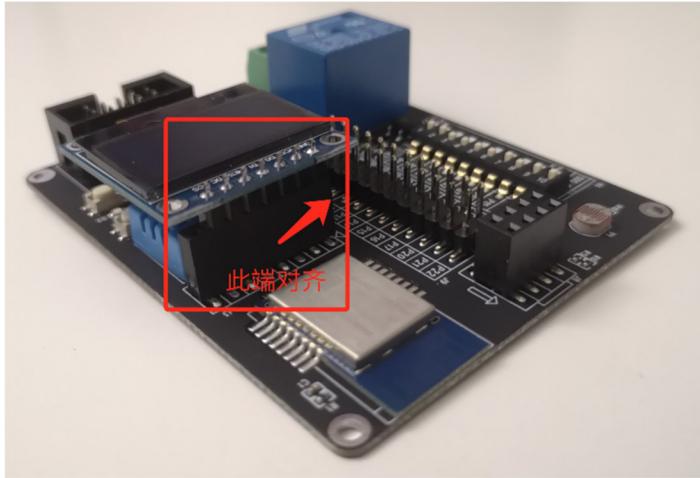
(1) After compiling and linking the project code, burn the program into the matching ZigBee development board.

(2) Since this experiment requires the use of the DHT11 temperature and humidity sensor, the ZigBee standard board has already integrated the DHT11 temperature and humidity sensor, and the DATA pin is connected to the P0_6 pin by default, as shown in the figure.



If you use ZigBee Mini for testing, you need to connect the DATA pin of DHT11 to the P0_6 pin, and connect the VCC and GND pins of DHT11 to the 3.3v~5.5v power supply and ground wire respectively.

(3) Insert the OLED screen into the standard board or Mini board, as shown in the figure.



(4) Power on the development board and the ambient temperature and humidity data will be displayed on the screen.

5.7.2. NorFLASH reading and writing experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=20>

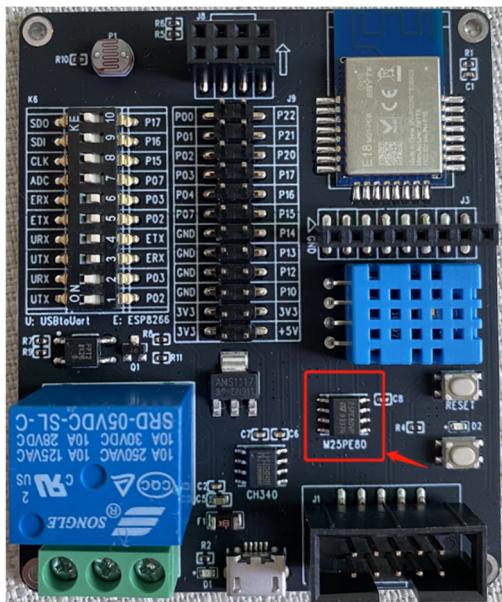
Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Hardware Preparation

This lesson requires the use of NOR Flash memory

- The ZigBee standard board has NOR Flash memory, as shown in the figure.



- Since the ZigBee MiNi board does not have NOR Flash memory by default, readers who do not have a standard board can skip this lesson

Flash Memory Introduction

Flash memory can be used to save information, such as system configuration information, data documents, etc., and has a wide range of uses.

According to different internal storage structures, Flash memory can be divided into two types: NOR Flash and NAND Flash.

NOR Flash

NOR Flash memory has fast reading speed, high storage reliability, and supports the use of random address access to storage space, but it has small storage capacity and is expensive, and is mostly used to store programs for electronic products.

NAND Flash

Compared with NOR Flash, **NAND Flash** memory has large capacity, high number of repeated read and write times and low price, but it has slow data reading speed and does not support random address access to storage space. It is somewhat similar to a CD or hard disk and is mostly used in memory cards and USB flash drives.

M25PE80 Introduction

The ZigBee standard board has an M25PE80 chip, which is a NOR Flash with a capacity of 1024KB (8M bit). In addition, the CC2530F256 also has a Flash memory with a capacity of 256KB. In other words, the standard board has a total of 1024KB+256KB Flash capacity.

- M25PE80 physical picture:



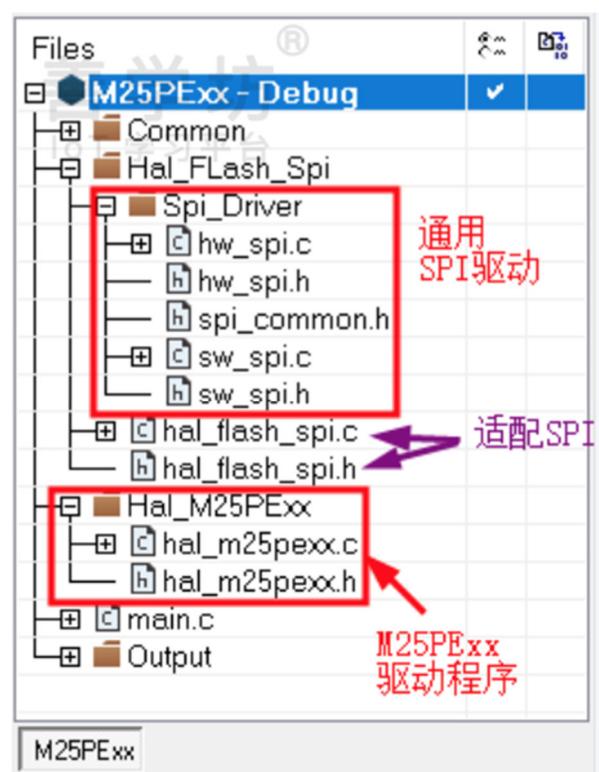
- M25PE80 pin diagram:



Understanding the M25PE80 API

The communication protocol of M25PE80 is SPI. When learning the display experiment, we have already explained the design method of SPI driver API. We **only need to adapt the special SPI driver API of M25PE80 based on the general SPI driver API**.

Open the supporting project code and expand Hal_Flash_Spi, you can see the general SPI driver API designed by the author and the SPI driver API dedicated to the M25Exx driver, as shown in the figure.



With this API, the use of M25PE80 is very simple. You only need to learn the three APIs in the hal_m25pexx.h/c file, which are defined as follows:

```
/**  
 * @fn    halM25PExxInit  
 *  
 * @brief  初始化M25PExx  
 *  
 * @return none  
 */  
  
void halM25PExxInit(void);  
  
/**  
 * @fn    halM25PExxRead  
 *  
 * @brief  从M25PExx中读取数据  
 *  
 * @param  addr - 将要读取的数据所在的存储器地址.  
 * @param  pBuf - 变量指针，用于保存存储器中读出来的数据.  
 * @param  len - 指定从存储器中读取多少个字节的数据.  
 *  
 * @return 如果读取成功，则返回0  
 */  
  
int halM25PExxRead(uint32 addr, uint8 *pBuf, uint16 len);  
  
/**  
 * @fn    halM25PExxWrite  
 *  
 * @brief  把数据写入到存储器中  
 *  
 * @param  addr - 说明把数据写入到存储器的哪个地址  
 * @param  pBuf - 变量指针，指向将要写入到存储器的数据  
 * @param  len - 指定把多少个字节的数据写入到存储器中  
 *  
 * @return 如果写入成功，则返回0  
 */  
  
int halM25PExxWrite(uint32 addr, uint8 *pBuf, uint16 len);
```

Using the M25PE80 API

Open the main.c file, and you can see the sample code for reading and writing data using the M25PE80 API:

```
void main(void)  
{
```

```
uint8 writeVal = 0;//此变量的值将会被写入到存储器中

uint8 readVal = 0;//从存储器读取到的值将会存入此变量中

char str[50];

setSystemClk32MHZ();//初始化系统时钟为32MHz

initUart0(USART_BAUDRATE_115200);//初始化串口0

halM25PExxInit();//初始化M25PE80存储器

//进入到循环中

while(1) {

/* 1.写数据到M25PE80存储器中 */

    //串口通信

    sprintf(str, "Write: %d\r\n", writeVal);

    uart0Send((unsigned char *)str, strlen(str));

    //把writeVal的值写入到存储器中地址为0x12345的存储空间中；由于writeVal的类型为uint8，也就1个字节·所以传入的数据长度为1

    if (halM25PExxWrite(0x12345, &writeVal, 1) != 0) {

        uart0Send("Write Error\r\n", 13);//如果函数返回值不等于0，表示写入错误

        continue;

    }

    writeVal++;//把写入值增加1

    delayMs(SYSLCK_32MHZ, 1000);//延迟

    /* 2.从M25PE80存储器中读取数据 */

    //从halM25PExxRead的0x12345处读取1个字节的数据·并将其保存到readVal中

    if (halM25PExxRead(0x12345, &readVal, 1) != 0) {

        uart0Send("Read Error\r\n", 12);//如果函数返回值不等于0，表示读取错误

        continue;

    }

    //串口通信

    sprintf(str, "Read: %d\r\n", readVal);

    uart0Send((uint8 *)str, strlen(str));

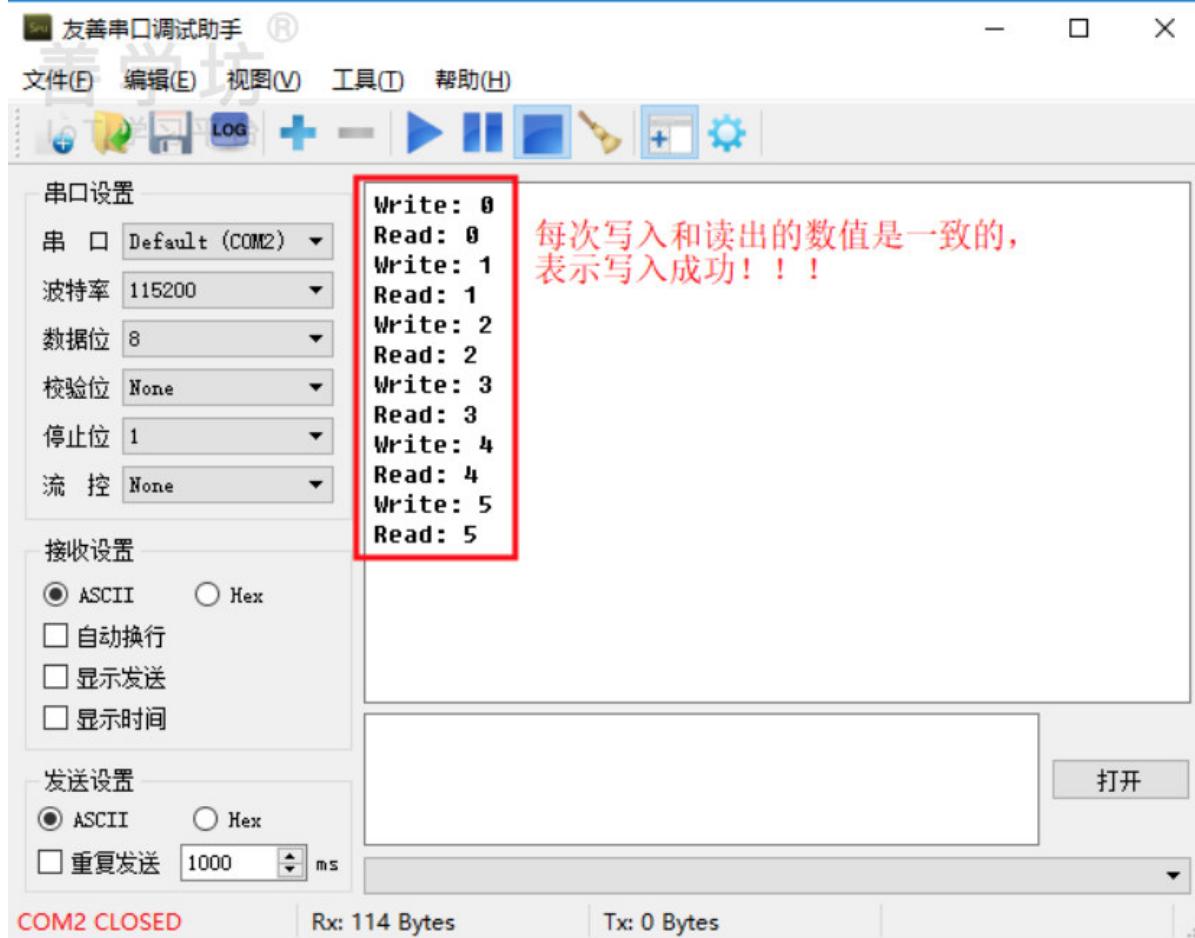
    delayMs(SYSLCK_32MHZ, 1000);//延迟

}
```

Debugging Simulation

Before learning this lesson, you need to master the basic program download and simulation operations. Please refer to: «Program Download and Simulation» (<https://z7po9bxpe4.k.topthink.com/@zigbee-dev-guide/7.2%20NorFLASHduxieshiyan.html#!>)

1. Use the emulator to connect the ZigBee standard board to the computer.
2. Press the reset button in the emulator
3. Open the supporting project, compile the source code and burn the program to the ZigBee standard board.
4. Disconnect the emulator, connect the development board to the computer with a Micro USB cable, and open the serial port debugging assistant to see the writing and reading process:



5.7.3. Relay Control Experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=21>

Technical support instructions:

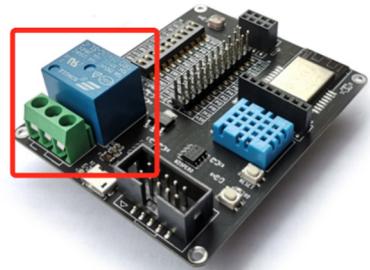
1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

This section will explain how to use relays to control high-voltage equipment.

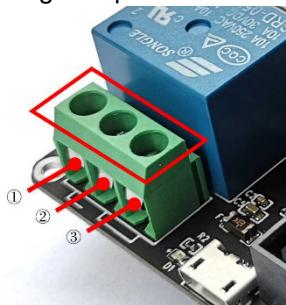
Relay Introduction

The operating voltage of common LED lights, small motors or small sensors is around 3.3V, so the microcontroller can directly control their on and off. However, the operating voltage of household light bulbs, household electric fans or electromagnetic locks is 220V, so the microcontroller cannot directly control their on and off. At this time, the microcontroller can use relays to

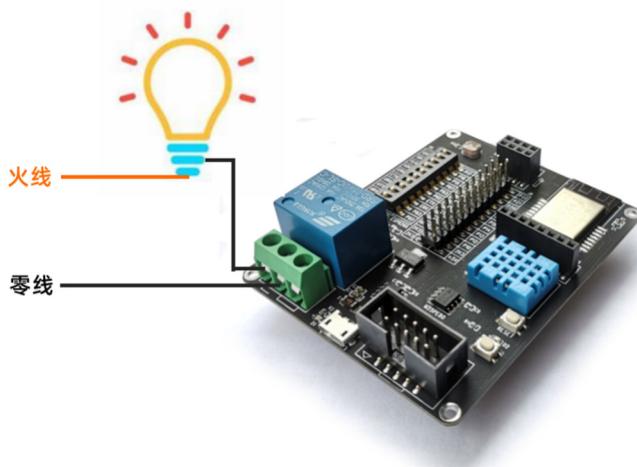
control the on and off of high-voltage electrical appliances. The ZigBee standard board integrates relays, as shown in the figure.



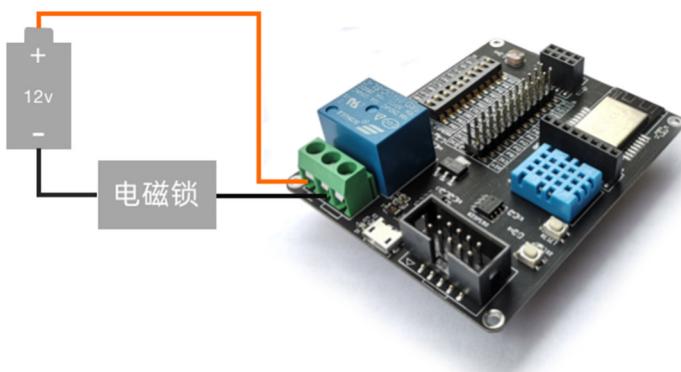
The green part is the terminal block, which has 3 wiring ports in total and each port has a screw hole above it.



Taking a 220v household light bulb as an example, let's briefly explain the wiring method of the relay. Use a screwdriver to unscrew the screws of ports 1 and 2, insert the neutral wires respectively, and then tighten the screws, as shown in the figure. At this time, **the relay acts as a switch**, which can control the disconnection or closing of the neutral wire, thereby controlling the switch of the light bulb. You can also insert the neutral wire into ports 2 and 3 respectively. The difference is that if it is inserted into ports 1 and 2, it is disconnected by default; if it is connected to ports 2 and 3, it is closed by default.



Similarly, the schematic diagram of connecting a 12v electromagnetic lock using a relay is shown in the figure. The 12v battery in the figure is used to power the electromagnetic lock.

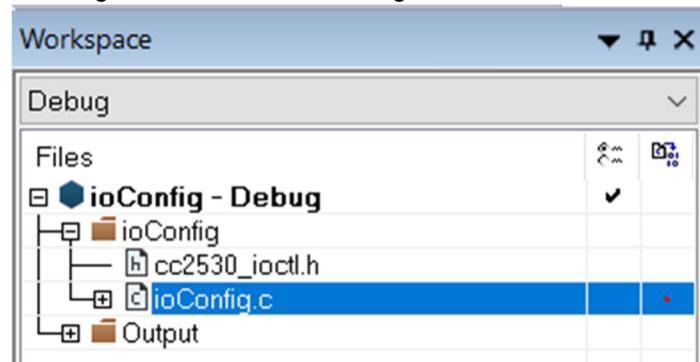


Using Relays

The use of relays is very simple. You can control the opening and closing of the relay by controlling the voltage level of the relay's control pin.

Pin Configuration

Before using the relay, you need to pair the control pin of the relay with the IO port of CC2530. The ZigBee standard board uses the P0_5 pin of CC2530 to connect with the control pin of the relay. Open the code of this experiment and you can find the ioConfig.c file, as shown in the figure.



The following configuration code can be found in the ioConfig.c file of this experimental code:

```
#define RELAY_PORT    0  
#define RELAY_PIN     5  
#define RELAY        P0_5
```

P0_5 is defined by the header file ioCC2530.h and is used to represent the P0_5 pin, so RELAY actually represents the P0_5 pin. If the relay control pin needs to be connected to other pins of CC2530, you only need to modify the pin mapping here. For example, if you need to connect an external relay control pin to the P1_2 pin, the code is as follows:

```
#define RELAY_PORT    1  
#define RELAY_PIN     2  
#define RELAY        P1_2
```

Controlling the relay

To control the opening and closing of the relay is essentially to control the level of the P0_5 pin. The sample code can be found in ioConfig.c:

//2. 51单片机入门/7. 外设实验/7.3 继电器开关控制/Workspace/code/ioConfig/ioConfig.c

```
#include "cc2530_ioctl.h"  
#include <stdio.h>  
  
/** @brief GPIO映射定义 */  
#define RELAY_PORT    0  
#define RELAY_PIN     5  
#define RELAY        P0_5  
  
/** @brief 继电器开关状态定义 */  
#define RELAY_ON      1  
#define RELAY_OFF     0  
  
static void delayMs(uint16_t nMs);  
static void initRelay(void);  
  
void main()
```

```

{
    initRelay();

    while(1) {
        delayMs(1000); // 延迟

        // 反转 RELAY 引脚的电平状态
        RELAY = (RELAY == RELAY_ON)? RELAY_OFF: RELAY_ON;

    } /* while */
}

/*
 * 延迟指定的时间
 *
 * @param nMs - 时间长度，单位为微秒
 */
static void delayMs(uint16_t nMs)
{
    uint16_t i,j;
    for (i = 0; i < nMs; i++)
        for (j = 0; j < 535; j++);
}

/*
 * 初始化继电器
 */
static void initRelay()
{
    CC2530_IOCTL(
        RELAY_PORT,
        RELAY_PIN,
        CC2530_OUTPUT);
    RELAY = RELAY_OFF;
}

```

The above code implements the function of continuously switching the relay. It can be seen that the use of relays is actually very simple.

Debugging Simulation

You can run this experimental code to observe the running results. The operation steps are as follows:

- (1) After compiling and linking the supporting engineering code, burn the program to the supporting ZigBee development board.
- (2) Since this experiment requires the use of relays, the ZigBee standard board has integrated relays, and its control pin is connected to the P0_5 pin by default. If you use the ZigBee Mini board for testing, you need to connect an external relay control pin at the P0_5 pin, and connect the VCC and GND pins of the relay to the 5v power supply and ground wire respectively.
- (3) Power on the development board through the MicroUSB cable, and you can observe the relay opening and closing continuously.

You must use a Micro USB cable to power the development board in order to use the relay!

6. Part 3: Z-Stack 3.0 Detailed Explanation

- 6.1. Chapter 1: Detailed Explanation of Z-Stack 3.0 Architecture
- 6.2. Chapter 2: Task Scheduling Principles of Operating Systems
- 6.3. Chapter 3: OSAL Detailed Explanation
- 6.4. Chapter 4: Hardware Adaptation Layer Application - LED
- 6.5. Chapter 5: Hardware Adaptation Layer Application - Buttons
- 6.6. Chapter 6: Hardware Adaptation Layer Application - Serial Port
- 6.7. Chapter 7: Hardware Adaptation Layer Application - Display Screen
- 6.8. Chapter 8: Hardware Adaptation Layer Application - ADC

6.1. Chapter 1: Detailed Explanation of Z-Stack 3.0 Architecture

- 6.1.1. Z-Stack 3.0.1 File Organization

- 6.1.2. Z-Stack 3.0.1 Project Framework

6.1.1. Z-Stack 3.0.1 File Organization

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=22>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Protocol stack introduction and installation

Z-Stack 3.0 is a technical solution developed by TI (Texas Instruments) for the ZigBee 3.0 protocol, also known as the TI ZigBee 3.0 protocol stack. In layman's terms, Z-Stack 3.0 is a program (library) written by TI based on the ZigBee 3.0 protocol specification. Developers can easily call various APIs in Z-Stack 3.0 to perform data communication based on the ZigBee protocol. This lesson requires the use of the TI Z-Stack 3.0 protocol stack, and readers can go to "Part 1" and follow the prompts to install it.

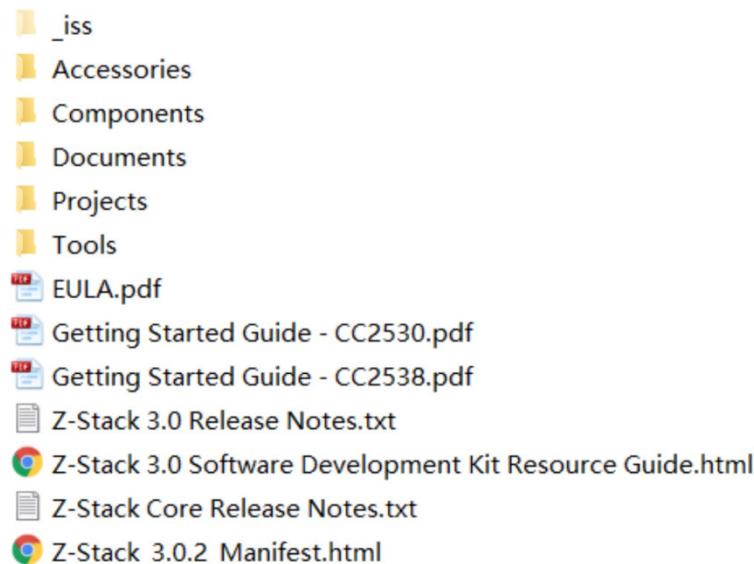
File Organization

After the default installation of Z-Stack 3.0 is completed, you can find the corresponding folder in the C drive of your computer, as shown in the figure.



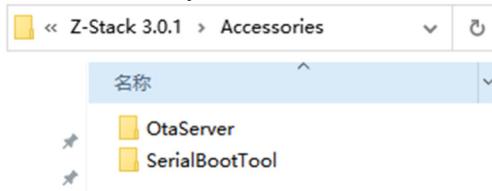
Open the Z-Stack 3.0.1 folder as shown.

名称



Accessories Catalog

Various auxiliary tools are stored in the Accessories directory, as shown in the figure.

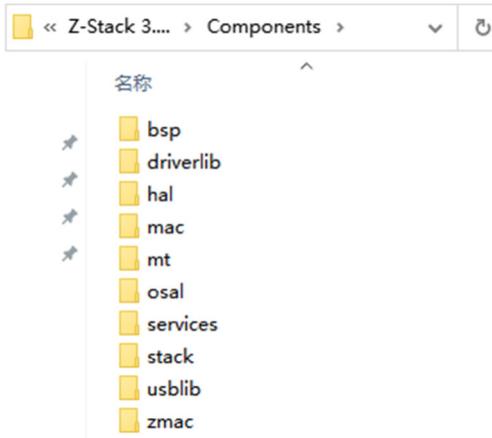


The folders in the Accessories directory are described as follows:

- (1) OtaServer: stores the OTA (Over The Air) test tool
- (2) SerialBootTool: stores the serial port upgrade test tool.

Components directory

The Z-Stack core source code and link library are stored. Enter the Components directory, as shown in the figure.



The folders in the Components directory are described as follows:

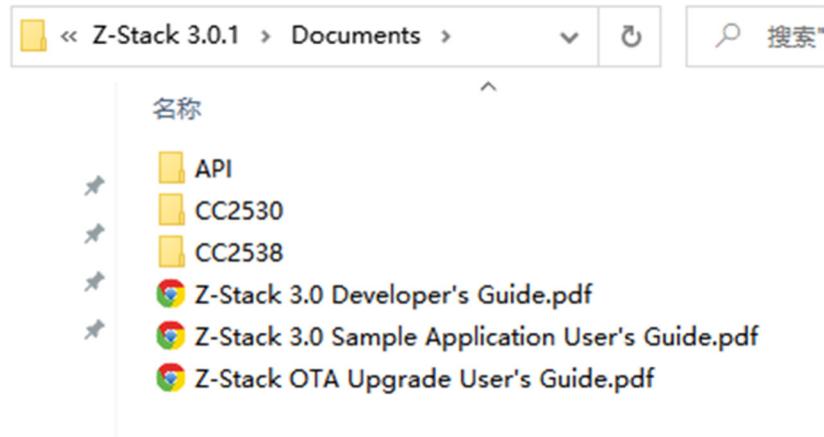
- (1) bsp: board support package, used to adapt the hardware resources of TI's official development board.
- (2) driverlib: driver library, which stores the library of TI's official non-open source drivers.
- (3) hal: hardware abstraction layer, which stores various drivers.
- (4) mac: media access control, which implements physical layer communication and IEEE 802.15.4 protocol.
- (5) mt: monitoring layer, which provides support for monitoring the operating status of each layer of the protocol stack.
- (6) osal: operating system abstraction layer, which is the operating system of Z-Stack 3.0.
- (7) services: provides some common and commonly used functions.
- (8) stack: implementation of ZigBee protocol.

(9) uslplib: USB library, which is required when the chip supports USB (such as CC2538).

(10) zmac: content belonging to the mac layer.

Documents directory

The relevant documents for Z-Stack development assistance are stored. Enter the Documents directory, as shown in the figure.

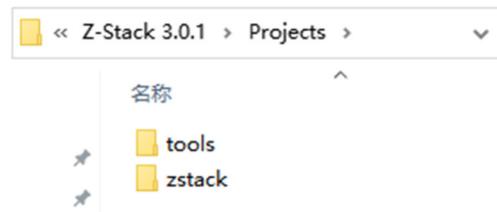


The folders and files in the Documents directory are described as follows:

- (1) API: Stores the documentation of the Z-Stack 3.0 API.
- (2) CC2530: Stores the documentation applicable to the CC2530 MCU.
- (3) CC2538: Stores the documentation applicable to the CC2538 MCU.
- (4) Z-Stack 3.0 Developer's Guide.pdf: Z-Stack 3.0 development guide.
- (5) Z-Stack 3.0 Sample Application User's Guide.pdf: Z-Stack 3.0 application sample guide.
- (6) Z-Stack OTA Upgrade User's Guide.pdf: OTA (Over The Air) air upgrade instructions and guide.

Projects directory

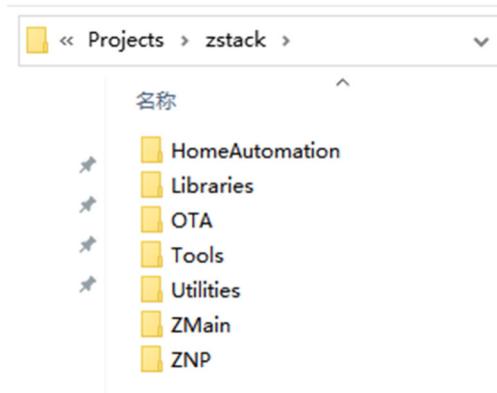
The project files and source code files of the ZigBee application routines are stored. Enter the Projects directory, as shown in the figure.



The folders in the Projects directory are described as follows:

- (1) tools: stores tools related to ZigBee application examples.
- (2) zstack: stores ZigBee related examples.

Enter the zstack folder as shown.

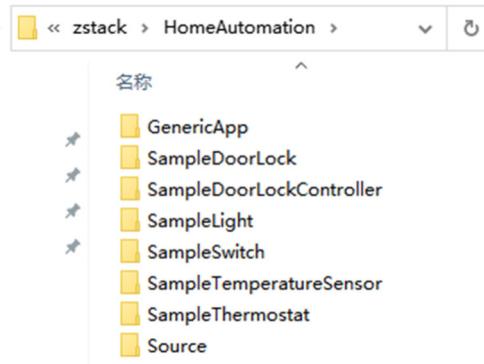


The folders in the zstack directory are described as follows:

- (1) HomeAutomation: related routines for the field of home automation.
- (2) Libraries: store link library files. TI's closed source code will be compiled into link libraries for developers to use.
- (3) OTA: OTA (Over The Air) air upgrade routines.

- (4) Tools: store files related to project configuration.
- (5) Utilities: public folder.
- (6) ZMain: store source code files where the main function is located and source code files related to system hardware startup.
- (7) ZNP: ZNP (ZigBee And Processor) routines.

Go into the HomeAutomation folder as shown.



The folders in the HomeAutomation directory are described as follows:

- (1) GenericApp: Generic application scenario examples.
- (2) SampleDoorLock: ZigBee 3.0 door lock example.
- (3) SampleDoorLockController: ZigBee 3.0 door lock controller example.
- (4) SampleLight: ZigBee 3.0 light example.
- (5) SampleSwitch: ZigBee 3.0 socket example.
- (6) SampleTemperatureSensor: ZigBee 3.0 temperature and humidity sensor example.
- (7) SampleThermostat: ZigBee 3.0 temperature controller example.
- (8) Source: Folder for storing public code.

Other Directories

- (1) Tools: Stores tools related to development and debugging.
- (2) EULA.pdf: Copyright statement file.
- (3) Getting Started Guide - CC2530.pdf: Getting Started Guide file for CC2530 MCU.
- (4) Getting Started Guide - CC2538.pdf: Getting Started Guide file for CC2538 MCU.
- (5) Z-Stack 3.0 Release Notes.txt: Z-Stack 3.0 release description file.
- (6) Z-Stack 3.0 Software Development Kit Resource Guide.html: Z-Stack 3.0 development resource guide link file.
- (7) Z-Stack Core Release Notes.txt: Z-Stack Core release description file.
- (8) Z-Stack_3.0.1_Manifest.html: Z-Stack 3.0 key information description list.
- (9) _iss: This folder is a hidden folder and stores related files used to uninstall Z-Stack 3.0.

6.1.2. Z-Stack 3.0.1 Project Framework

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=23>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

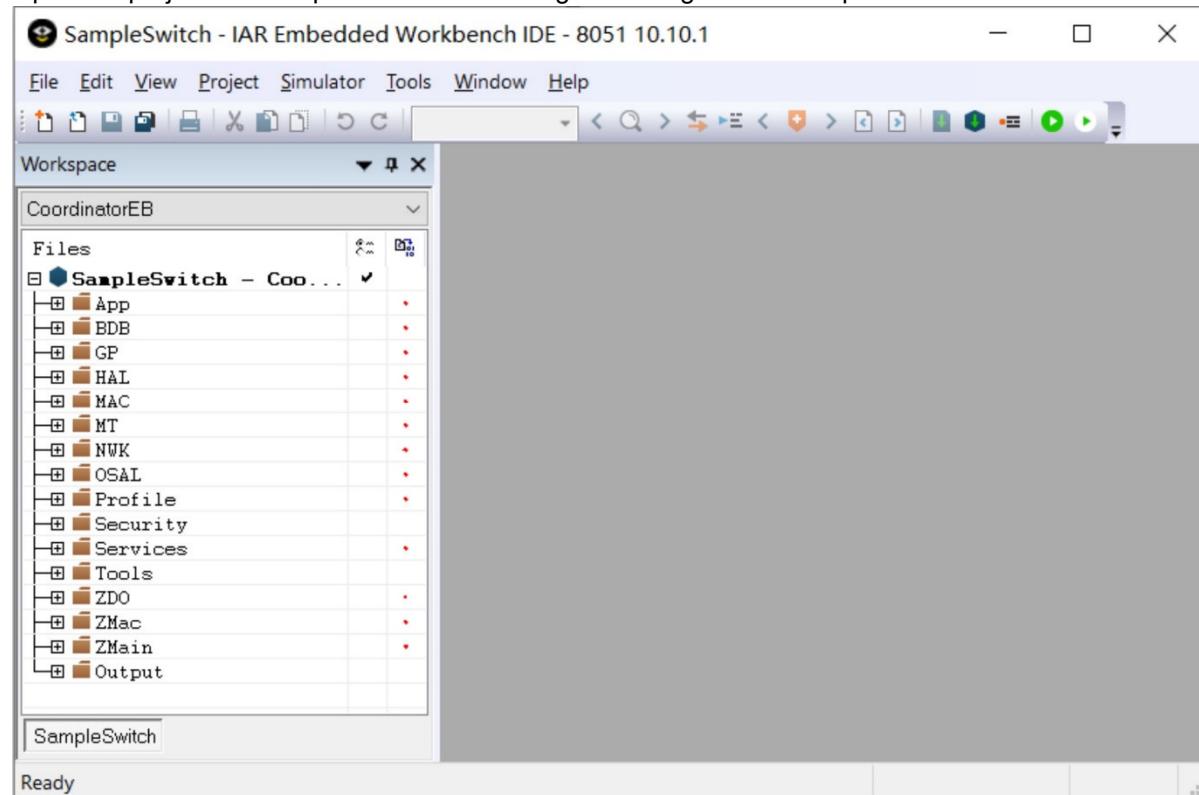
Engineering Organization Structure

This section will use the SampleSwitch example to explain the project file structure of the protocol stack. Enter the folder where the SampleSwitch project is located, as shown in the figure.

The screenshot shows a file explorer window with the following details:

名称	修改日期	类型	大小
CoordinatorEB	2022/4/8 21:10	文件夹	
settings	2022/4/8 21:11	文件夹	
BuildLog.log	2022/4/8 21:10	文本文档	1 KB
SampleSwitch.dep	2022/4/19 21:19	DEP 文件	139 KB
SampleSwitch.ewd	2022/4/8 21:10	EWD 文件	122 KB
SampleSwitch.epw	2022/4/8 21:10	EWP 文件	243 KB
SampleSwitch.eww	2022/4/8 21:10	IAR IDE Worksp...	1 KB

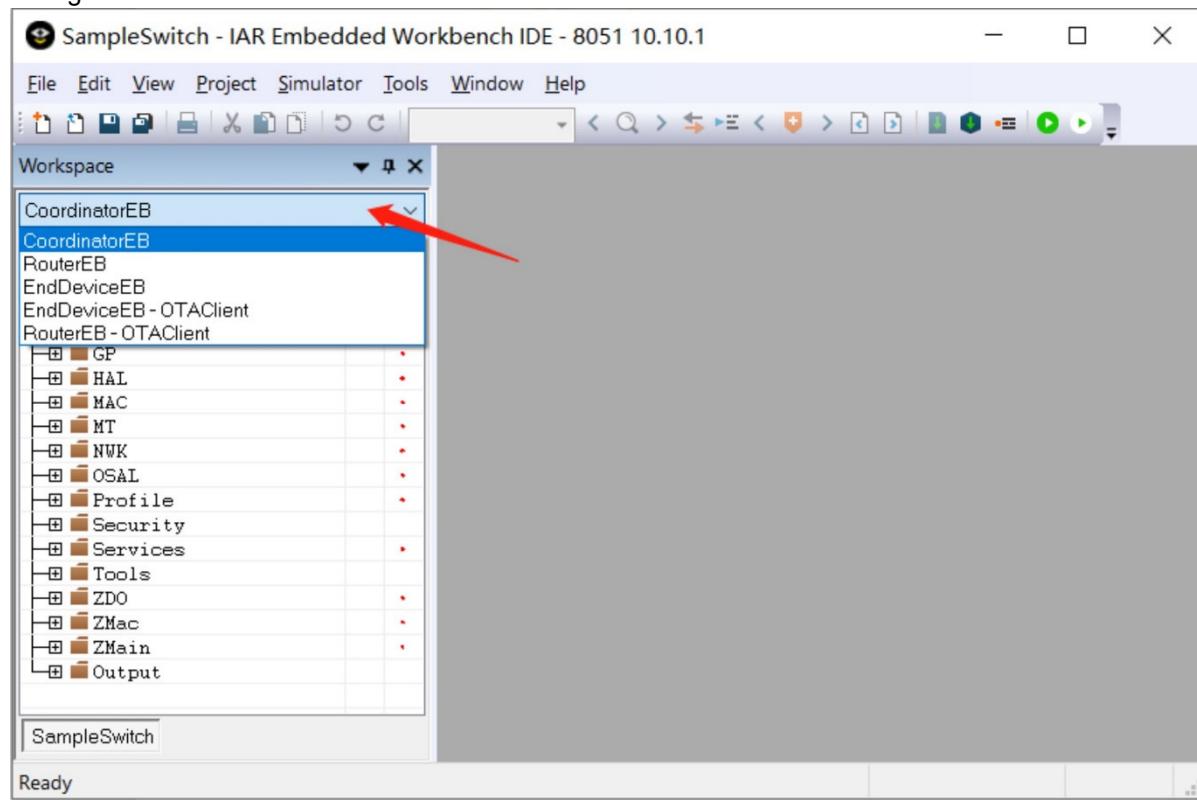
Open the project file SampleSwitch.eww through the integrated development environment IAR 10.10.1, as shown in the figure.



The descriptions of each group in the project are as follows:

- (1) App: Stores source code files related to the application.
- (2) BDB: Implements ZigBee BDB (Base Device Behavior) functions.
- (3) GP: Implements ZigBee GP (Green Power) functions.
- (4) HAL: Hardware abstraction layer, stores various drivers.
- (5) MAC: Media access control, implements physical layer communication and IEEE 802.15.4 protocol.
- (6) MT: Monitoring layer, provides support for monitoring the operating status of each layer of the protocol stack.
- (7) NWK: ZigBee network layer.
- (8) OSAL: Operating system abstraction layer.
- (9) Profile: Stores source code files for ZigBee standardization definitions and related function implementations.
- (10) Security: Implements security-related services.
- (11) Services: Provides some common and commonly used functions.
- (12) Tools: Stores files related to project configuration.
- (13) ZDO: Stores source code files related to ZDO (ZigBee Device Object).
- (14) ZMac: Contents belonging to the mac layer.
- (15) ZMain: Stores source code files where the main function is located and source code files related to system hardware startup.
- (16) Output: Stores files output when the project is compiled/linked.

You can select different ZigBee network device types in the project. Click the tab → Select Network Device Type, as shown in the figure.



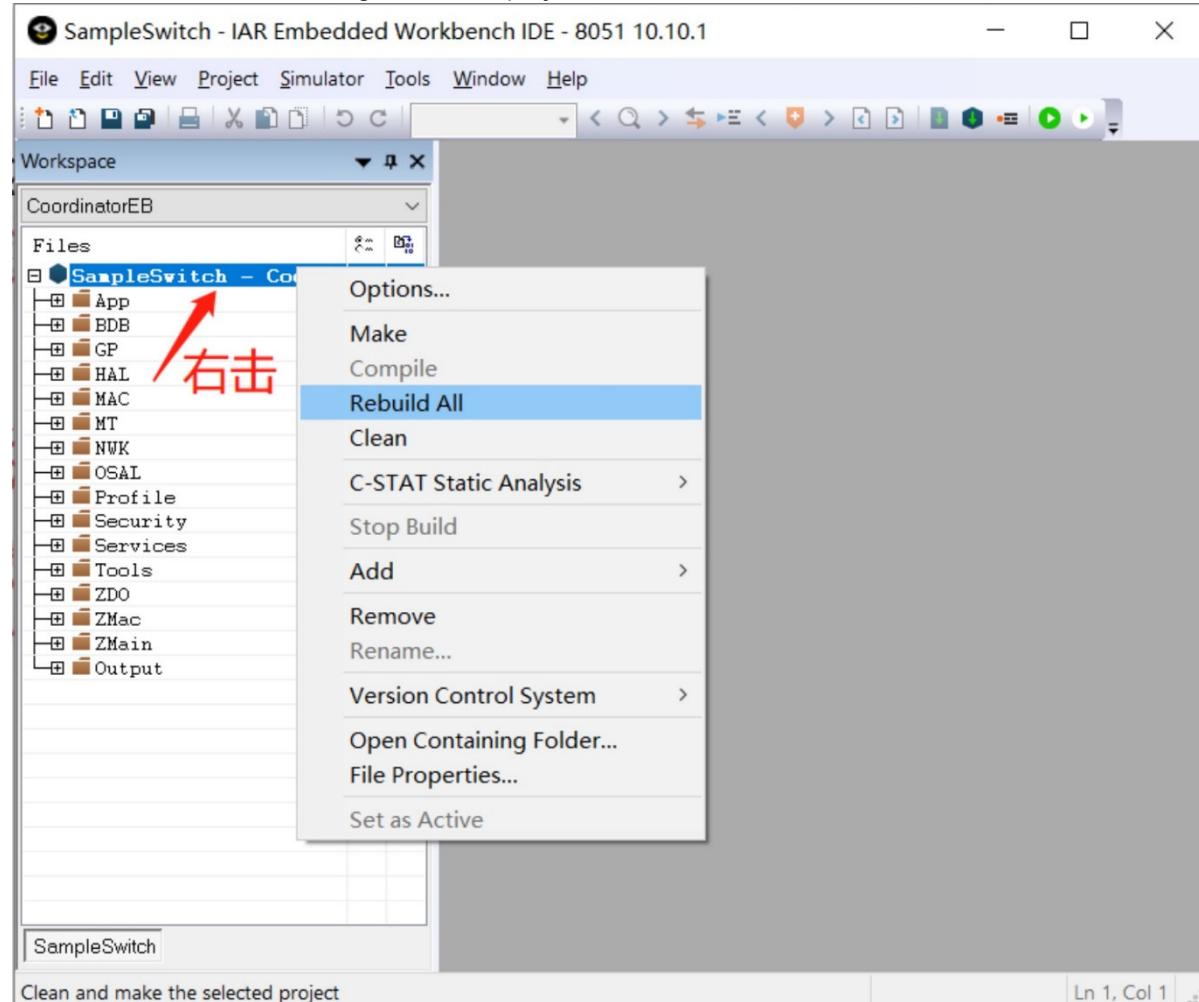
There are three types of ZigBee network devices, namely Coordinator, Router and EndDevice. The meanings of the options in the figure are described as follows:

- (1) CoordinatorEB: ZigBee coordinator.
- (2) RouterEB: ZigBee router.
- (3) EndDeviceEB: ZigBee end device.
- (4) EndDeviceEB-OTAClient: ZigBee end device that supports OTA (Over The Air) air upgrade.
- (5) RouterEB-OTAClient: ZigBee router that supports OTA (Over The Air) air upgrade.

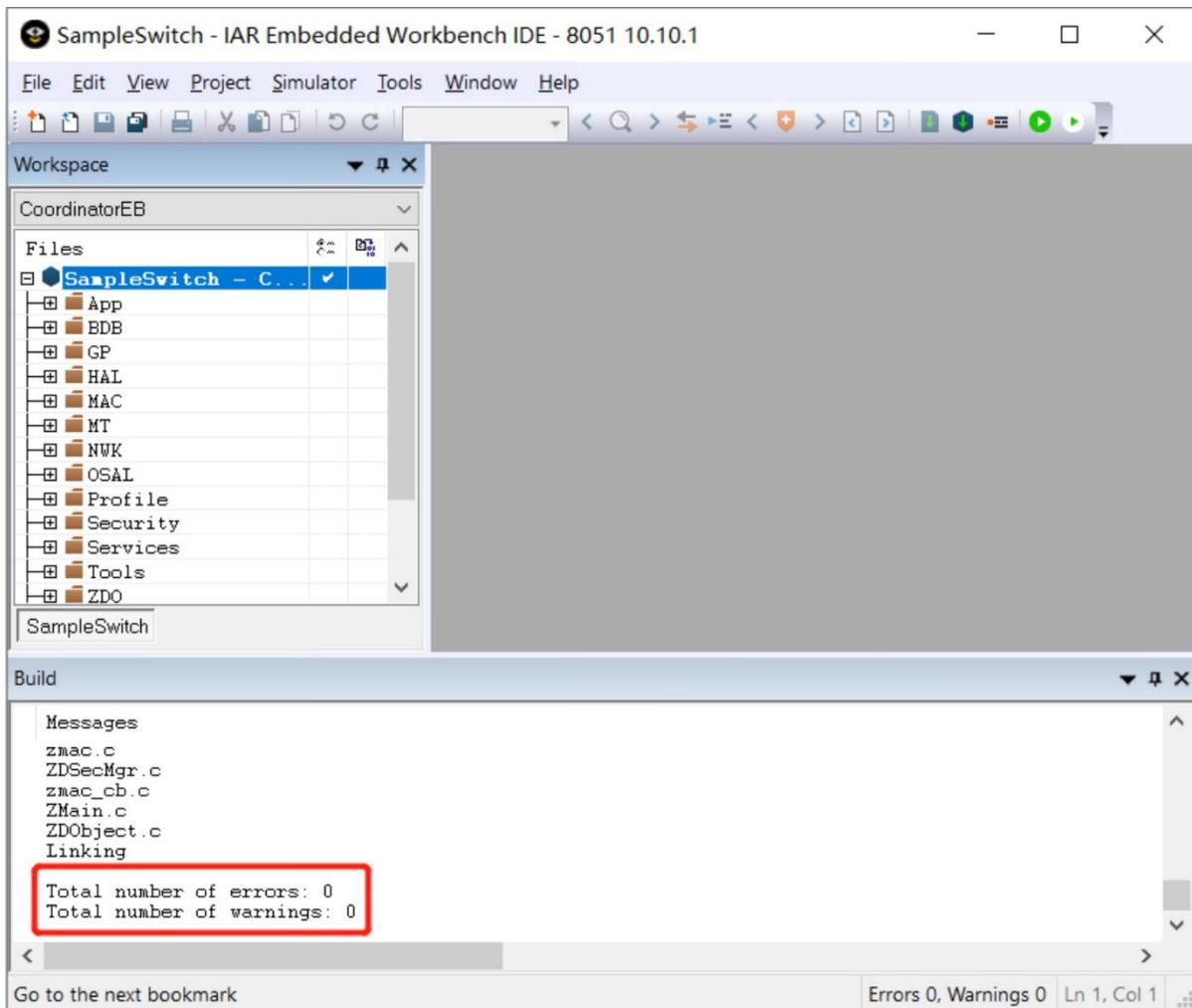
In the following chapters, the meaning of these device types will be explained in detail.

Project compilation and linking

Select CoordinatorEB, then right-click the project name, and then select Rebuild All, as shown in the figure.



There are no errors or warnings during the compilation and linking process, as shown in the figure.



6.2. Chapter 2: Task Scheduling Principles of Operating Systems

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=24>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

This section will use a simple example to illustrate the task scheduling mechanism.

Basic theory

The system's task scheduling process involves several concepts including tasks, task pools, priorities, polling, and system scheduling cycles. These concepts are introduced below.

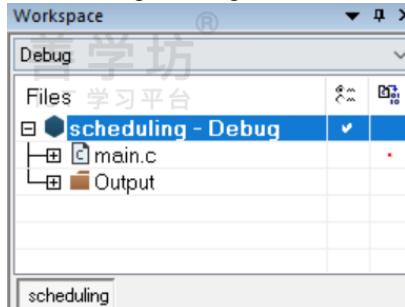
- **Task:** It can be understood as a specific task that needs to be processed by the processor, such as "turn on the light after 1 second" or "turn off the light", etc.
- **Task pool:** It is a buffer that can store multiple tasks. For example, a task pool can store the tasks of "turn on the light after 1 second", "turn off the light after 2 seconds", "turn on the light after 3 seconds" and "turn off the light after 1 minute". The system will execute each task in the task pool at the specified time.

- **Priority:** Since there may be multiple tasks that need to be executed at the same time, it is necessary to distinguish which tasks should be processed first and which tasks should be postponed at this time. Priority is used to mark the priority level of each task. Under the same conditions, the system will give priority to high-priority tasks, while low-priority tasks need to wait for processing. In addition, the system may also interrupt tasks with lower priorities and process higher-priority tasks instead.
- **Polling:** The system will check the task pool **at regular intervals** to see if there are any tasks that need to be processed now. This process is called polling.
- **Operating system scheduling cycle:** The scheduling cycle refers to the specific length of time in the concept of polling "every period of time". The system scheduling cycle is also the minimum time cycle of a task. For example, if the system scheduling cycle is 1 second, but there is a task called "turn off the light after 0.1 seconds", although the task requires turning off the light after 0.1 seconds, since 0.1 seconds is less than the system scheduling cycle, the task will not be executed until 1 second later.

Hands-on system scheduling

The overall process

and the engineering code for this lesson are shown in the figure.



Open the main.c file, the main function code is as follows:

```
void main()
{
    initLed(); // 初始化LED灯
    taskListInit(); // 初始化任务池

    addTask(2000, TASK_LED_ON); // 往任务池中添加一个任务，即2s后打开LED
    addTask(3000, TASK_LED_OFF); // 往任务池中添加一个任务，即3s后关闭LED

    // 每隔1ms轮询1次
    while(1) {
        delayMs(1); // 暂停1ms
        polling(); // 轮询
    }
}
```

The above code simply simulates the system scheduling process. Its purpose is to make it easier for readers to understand. It is inevitable that there are some places that are not rigorous.

In the main function, the task pool is first initialized, and then two tasks are added to the task pool, namely "turn on the light after 2 seconds" and "turn off the light after 3 seconds", and then the while loop is entered. The delayMs function allows the program to wait for 1 millisecond before continuing to run, and the polling function allows the program to check if there are any tasks that need to be processed. In this way, this code implements checking every 1 millisecond to see if there are any tasks that need to be processed, thus simulating a system polling with a system scheduling period of 1 millisecond.

Implementation of task pool

There are many data structures suitable for task pools, such as queues (first-in, first-out), stacks (first-in, last-out), and tree structures (traversal). These data structures can be implemented as static arrays or dynamic linked lists. For the convenience of readers, a static array is used here as the implementation of the task pool data structure.

Define a structure to represent a task. The code is as follows:

```
/** @brief 任务结构体的定义 */
typedef struct task_t {
    bool occupy; //是否已占用 : true表示当前有任务 ; false表示当前没有任务
    uint16_t task; //任务内容
    uint16_t expire; //等待时间
} task_t;
```

So we can define a structure array to represent the task pool. The code is as follows:

```
/** @brief 任务池的定义 */
#define TASK_LIST_SIZE 5
static task_t taskList_g[TASK_LIST_SIZE];
```

Next, we define four APIs to operate the task pool.

1. taskListInit(void)

The task pool initialization function taskListInit(void) sets the occupy value of each element in the array to false, indicating that the element does not store tasks. The code is as follows:

```
/*
 * 任务池初始化
 */
static void taskListInit()
{
    //把所有任务都设置为未使用
    for (uint16_t i = 0; i < TASK_LIST_SIZE; i++)
        taskList_g[i].occupy = false;
}
```

2. addTask(uint16_t expire, uint16_t event)

The addTask function first searches the entire array to see if there is an element that is not occupied (occupy=false). If found, save the task information to the array element and set the flag bit occupy to "occupied" (occupy=true). The code is as follows:

```
/*
 * 往任务池中添加任务
*/
```

```

* @param expire 延迟多久执行
* @param task 任务内容
*/
static void addTask(uint16_t expire, uint16_t task)
{
    for (uint16_t i = 0; i < TASK_LIST_SIZE; i++) {
        if(taskList_g[i].occupy) continue;

        taskList_g[i].task = task;
        taskList_g[i].expire = expire;
        taskList_g[i].occupy = true;

        break;
    }
}

```

3. The polling()

polling function searches the entire array to see if there are any array elements that are "occupied" and the corresponding tasks are due. If so, the task is processed and the array element control is released after processing, that is, occupy is set to false. The code is as follows:

```

/*
* 轮询
*/
static void polling()
{
    for (uint16_t i = 0; i < (sizeof(taskList_g)/sizeof(taskList_g[0])); i++) {
        if (taskList_g[i].occupy && (~taskList_g[i].expire == 0)) {
            //执行任务
            taskHandler(taskList_g[i].task);
            //释放任务池中的空间
            taskList_g[i].occupy = false;
        }
    }
}

```

Each time the for loop is executed, the expire value will be reduced by 1. If the expire value is reduced to 0, it means that the task has reached the time to be executed, and the task processing function will be called to execute the task. In addition, since the tasks at the front of the task pool will be traversed and executed first, the priority of the tasks at the front is higher.

4. taskHandler(uint16_t task)

The task processing function taskHandler(uint16_t task) performs corresponding processing according to the content of each task. It should be noted that after processing the corresponding task, we added a new task to the task pool, such as TASK_LED_ON, which is to turn on the LED, and then we set the TASK_LED_OFF task to achieve the effect of periodic tasks. The code is as follows:

```

/*
* 执行指定的任务
*
* @param task 任务
*/

```

```

*/
static void taskHandler(uint16_t task)
{
    /* 根据指定的任务 · 执行对应的操作 */

    if (task == TASK_LED_ON) {
        LED = LED_ON;//开灯

        printf("Set Led On!\n");

        //执行完任务后 · 重新往任务池中添加一个开灯任务

        addTask(2000, TASK_LED_ON);
    }

    else if(task == TASK_LED_OFF) {
        LED = LED_OFF;//关灯

        printf("Set Led Off!\n");

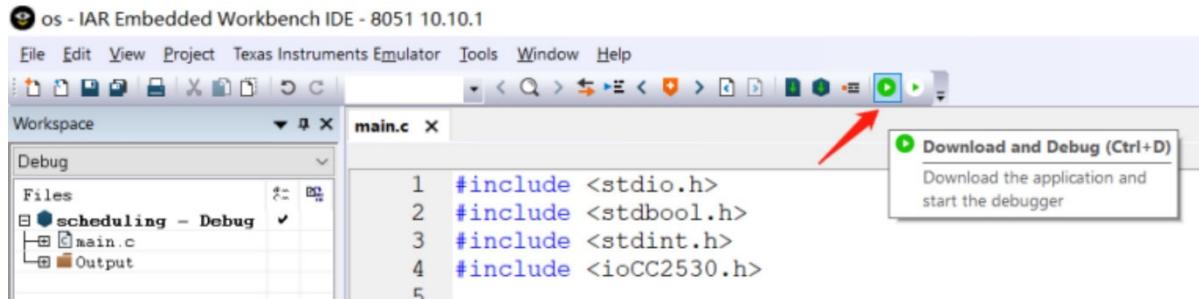
        //执行完任务后 · 重新往任务池中添加一个关灯任务

        addTask(2000, TASK_LED_OFF);
    }
}

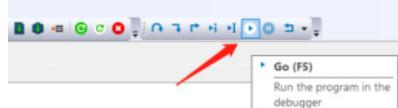
```

Debugging Simulation

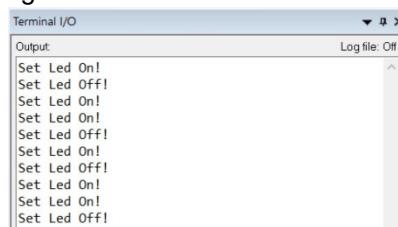
Click the Download and Debug button in the integrated development environment IAR 10.10.1 to compile, link, download the program and enter the simulation mode, as shown in the figure.



After entering the simulation mode, click the Go button to run the program, as shown in the figure.



After the program runs, you can see Set Led On and Set Led Off output alternately in the Terminal I/O window, as shown in the figure.



6.3. Chapter 3: OSAL Detailed Explanation

6.3.1. OSAL Task Scheduling Principle

6.3.2. Task initialization and event processing

6.3.3. Application of Z-Stack Events

6.3.4. Using Dynamic Memory

6.3.1. OSAL Task Scheduling Principle

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=25>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

This lesson will explain the task scheduling principle of OSAL by analyzing the source code of the Sample Switch routine. It involves complex source code, and readers only need to have **a general understanding of the entire task scheduling process** for now.

About OSAL

OSAL (Operating System Abstraction Layer) can be generally understood as a simplified operating system, which provides basic functions such as memory management, interrupt management and task scheduling for the correct operation of Z-Stack.

Understanding the task scheduling process

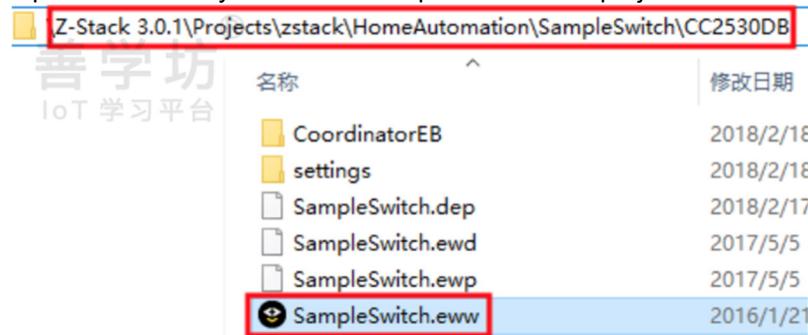
The task scheduling of OSAL is actually similar to the task scheduling implemented by the author in the previous class, that is, initializing the task pool and polling the task pool.

Open the engineering code that comes with this lesson, as shown in the figure.

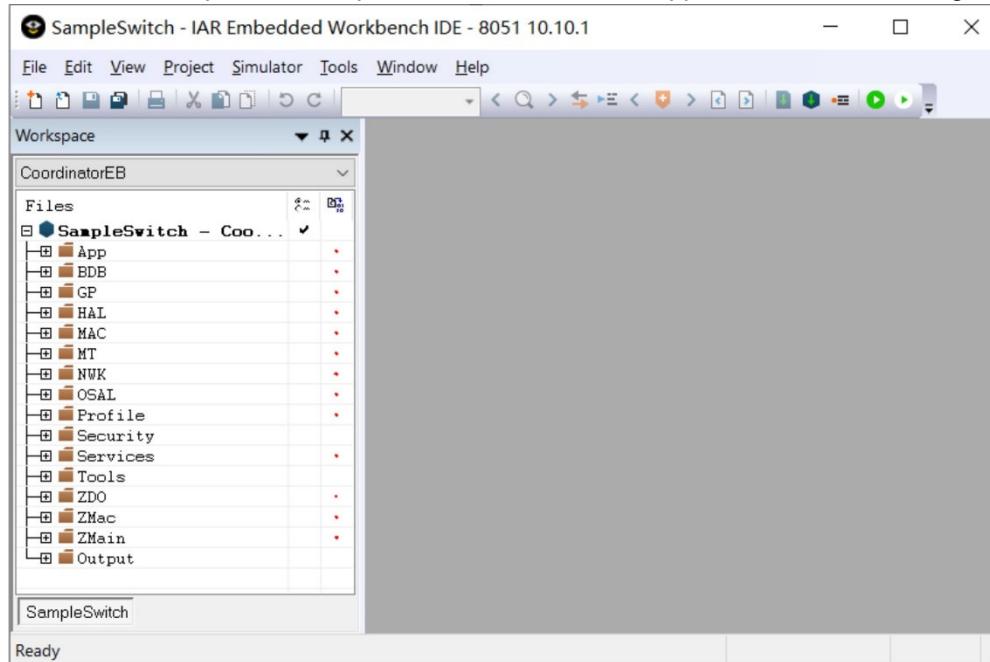


Readers will find that the accompanying engineering code is different from the previous Z-Stack. This is because the author has cut out some unused files to facilitate readers' learning.

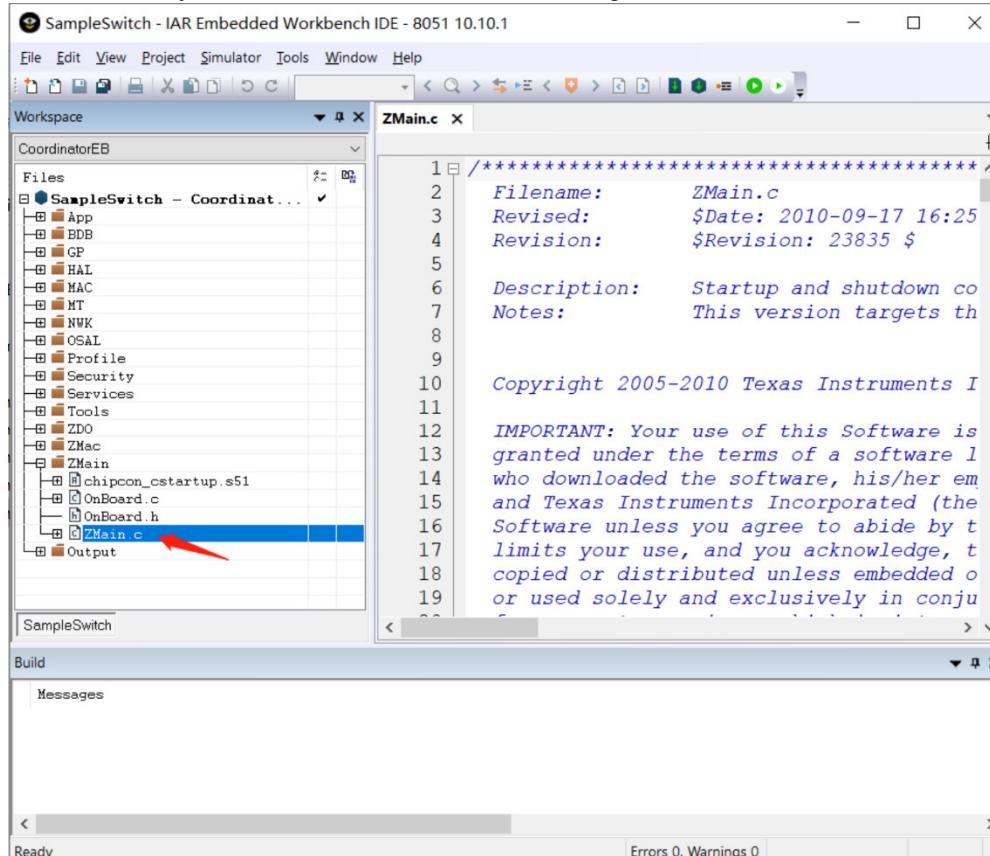
Open the directory where the SampleSwitch.eww project file is located, as shown in the figure.



Double-click to open the SampleSwitch.eww file. It will appear as shown in the figure below.



A program usually starts with the main() function, and Z-Stack 3.0 is no exception. Its main() function is in the ZMain.c file in the ZMain directory, which is located as shown in the figure.



Open the ZMain.c file and find the main() function. Its code is as follows:

```
1.int main( void )  
2.{  
3. // Turn off interrupts  
4. osal_int_disable( INTS_ALL );      // 关闭所有中断  
5.  
6. // Initialization for board related stuff such as LEDs  
7. HAL_BOARD_INIT(); // 初始化板载资源 · 比如PA、时钟源等  
8.
```

```
9. // Make sure supply voltage is high enough to run
10. zmain_vdd_check(); // 检测供电电压是否可以支撑芯片正常运行
11.
12. // Initialize board I/O
13. InitBoard( OB_COLD ); // 初始化板载I/O, 比如按键配置为输入
14.
15. // Initialize HAL drivers
16. HalDriverInit(); // 初始化硬件适配层 · 比如串口、显示器等
17.
18. // Initialize NV System
19. osal_nv_init( NULL ); // 初始化NV (芯片内部FLASH的一块空间 )
20.
21. // Initialize the MAC
22. ZMacInit(); // 初始化MAC层 ( 数据链路层 )
23.
24. // Determine the extended address
25. zmain_ext_addr(); // 确定芯片的物理地址
26.
27.#if defined ZCL_KEY_ESTABLISH
28. // Initialize the Certicom certificate information.
29. zmain_cert_init(); // 初始化认证信息
30.#endif
31.
32. // Initialize basic NV items
33. zgInit(); // 初始化存储在NV中的协议栈全局信息 · 如网络启动方式等
34.
35.#ifndef NONWK
36.// Since the AF isn't a task, call it's initialization routine
37. afInit(); // 初始化AF(射频)
38.#endif
39.
40. // Initialize the operating system
41. osal_init_system(); // 初始化OSAL(操作系统抽象层)
42.
43. // Allow interrupts
44. osal_int_enable( INTS_ALL ); // 使能所有中断
45.
46. // Final board initialization
47. InitBoard( OB_READY ); // 初始化板载IO资源 · 比如按键
48.
49. // Display information about this device
50. zmain_dev_info(); // 在显示器上显示设备物理地址
```

```
51.  
52. /* Display the device info on the LCD */  
53.#ifdef LCD_SUPPORTED  
54. zmain_lcd_init(); // 在显示器上显示设备信息 · 比如制造商等  
55.#endif  
56.  
57.  
58.  
59.#ifdef WDT_IN_PM1  
60. /* If WDT is used, this is a good place to enable it. */  
61. WatchDogEnable( WDTIMX ); // 启动看门狗功能  
62.#endif  
63.  
64. /* 进入系统轮询 */  
65. osal_start_system(); // No Return from here  
66.  
67.  
68. return 0; // Shouldn't get here.  
69.} // main()
```

There are two key function calls in this function, the code is as follows:

```
//初始化OSAL, 包括初始化任务池  
osal_init_system();  
  
//轮询任务池  
osal_start_system();
```

It can be seen that the task scheduling process of OSAL is similar to that explained in the previous class, that is, initializing the task pool and polling the task pool.

The definitions of the `osal_init_system()` function and the `osal_start_system()` function can be found in the `OSAL.c` file in the `OSAL` directory. The location of `OSAL.c` is shown in the figure.

```

SampleSwitch - IAR Embedded Workbench IDE - 8051 10.10.1
File Edit View Project Simulator Tools Window Help
Workspace ZMain.c OSAL.c
CoordinatorEB
Files SampleSwitch - CoordinatorEB
App
BBB
GP
HAL
MAC
MT
NUK
OSAL
coadef.h
OSAL.c
OSAL.h
OSAL_Clock.c
OSAL_Clock.h
OSAL_Math.s51
OSAL_Memory.c
OSAL_Memory.h
OSAL_Nv.c
OSAL_Nv.h
OSAL_PwrMgr.c
OSAL_PwrMgr.h
OSAL_Tasks.h
OSAL_Timers.c
OSAL_Timers.h
ZcomDef.h
Profile
Security
Services
Tools
ZDO
ZMac
ZMain
chipcon_cstartup.s51
OnBoard.c
OnBoard.h

```

```

1364
1365 /**
1366 * @fn osal_start_system
1367 *
1368 * @brief
1369 *
1370 * This function is the main loop function of the task system
1371 * ZBIT and UBIT are not defined). This Function doesn't return
1372 *
1373 * @param void
1374 *
1375 * @return none
1376 */
1377 void osal_start_system( void )
1378 {
1379 #ifdef USE_ICALL
1380 /* Kick off timer service in order to allocate resources up
1381 * The first timeout is required to schedule next OSAL timer
1382 * as well. */
1383 ICall_Errorno errno = ICall_setTimer(1, osal_msec_timer_cb,
1384 (void *) osal_msec_timer,
1385 &osal_timerid_msec_timer)
1386 if (errno != ICALL_ERRNO_SUCCESS)
1387 {
1388 ICall_abort();
1389 }
1390#endif /* USE_ICALL */
1391

```

Ready Errors 0, Warnings 0 Ln 1377, Col 31 System 大写 数字 改写

The osal_init_system() function code is as follows:

```

uint8 osal_init_system( void )

{
#ifndef !defined USE_ICALL && !defined OSAL_PORT2TIRTOS

// 初始化内存分配系统

osal_mem_init();

#endif /* !defined USE_ICALL && !defined OSAL_PORT2TIRTOS */

// 初始化消息队列

osal_qHead = NULL;

// 初始化OSAL定时器

osalTimerInit();

// 初始化电源管理系统

osal_pwrmgr_init();

#endif USE_ICALL

osal_prepare_svc_enroll();

#endif /* USE_ICALL */

// 初始化任务池

osalInitTasks();

#endif !defined USE_ICALL && !defined OSAL_PORT2TIRTOS

// Setup efficient search for the first free block of heap.

osal_mem_kick();

#endif /* !defined USE_ICALL && !defined OSAL_PORT2TIRTOS */

```

```

#define USE_ICALL

// Initialize variables used to track timing and provide OSAL timer service

osal_last_timestamp = (uint_least32_t) ICall_getTicks();

osal_tickperiod = (uint_least32_t) ICall_getTickPeriod();

osal_max_msecs = (uint_least32_t) ICall_getMaxMSecs();

/* Reduce ceiling considering potential latency */

osal_max_msecs -= 2;

#endif /* USE_ICALL */

return ( SUCCESS );

}

```

In the above code, you can find a task pool initialization function osalInitTasks(). As the name suggests, its job is to initialize the task pool.

The osal_start_system() function code is as follows:

```

void osal_start_system( void )

{

#endif USE_ICALL

/* Kick off timer service in order to allocate resources upfront.

 * The first timeout is required to schedule next OSAL timer event

 * as well. */

ICall_Errorno errno = ICall_setTimer(1, osal_msec_timer_cback,

                                    (void *) osal_msec_timer_seq,

                                    &osal_timerid_msec_timer);

if (errno != ICALL_ERRNO_SUCCESS)

{

    ICall_abort();

}

#endif /* USE_ICALL */

#if !defined ( ZBIT ) && !defined ( UBIT )

//主循环

for(;;)

#endif

{

    //系统轮询调度

    osal_run_system();

}

#endif USE_ICALL

ICall_wait(ICALL_TIMEOUT_FOREVER);

#endif /* USE_ICALL */

}
}

```

In the main loop of the osal_start_system() function, the osal_run_system() function is called cyclically. The main work of this function is to poll the task pool. The definition of the osal_run_system() function in the OSAL.c file is as follows:

```
1. void osal_run_system( void )
2. {
3.     uint8 idx = 0;
4.
5.     /* 更新时间，并整理出到期的任务。系统的时钟周期是：320us */
6.     osalTimeUpdate();
7.     Hal_ProcessPoll(); // 硬件适配层中断查询
8.
9.     do {
10.         if (tasksEvents[idx]) // 查看是否有任务需要处理
11.         {
12.             break;
13.         }
14.     } while (++idx < tasksCnt); // 轮询整个任务池
15.
16.     if (idx < tasksCnt) // 循环结束后，如果idx < tasksCnt表示任务池有任务需要处理
17.     {
18.         uint16 events;
19.         halIntState_t intState;
20.         HAL_ENTER_CRITICAL_SECTION(intState); // 关闭中断
21.         events = tasksEvents[idx]; // events中保存了该任务中的待处理事件
22.         tasksEvents[idx] = 0; // 清空此任务中的所有待处理事件
23.         HAL_EXIT_CRITICAL_SECTION(intState); // 恢复中断
24.
25.         activeTaskID = idx;
26.         events = (tasksArr[idx])(idx, events); // 处理任务中的事件
27.         activeTaskID = TASK_NO_TASK;
28.
29.         HAL_ENTER_CRITICAL_SECTION(intState); // 关闭中断
30.         tasksEvents[idx] |= events; // 保存还没被处理的事件到任务中
31.         HAL_EXIT_CRITICAL_SECTION(intState); // 恢复中断
32.     }
33. #if defined(POWER_SAVING) && !defined(USE_ICALL)
34. else // Complete pass through all task events with no activity? {
35.     osal_pwrmgr_powerconserve(); // 如果没有任务需要处理则进入低功耗
36. }
37. #endif
38.
39. /* Yield in case cooperative scheduling is being used. */
```

```
40.#if defined (configUSE_PREEMPTION)&&(configUSE_PREEMPTION == 0) {  
41.    osal_task_yield();  
42.}  
43.#endif
```

In order to better reflect the polling logic of this function, the original code has been simplified.

In the above code, let's focus on the do-while loop. The code is as follows:

```
9. do {  
10.   if (tasksEvents[idx])// 查看是否有任务需要处理  
11.   {  
12.     break;  
13.   }  
14. } while (++idx < tasksCnt);// 轮询整个任务池
```

The main function of this loop is to poll the entire task pool, that is, to see if there are any tasks to be processed. There is only one condition in the loop, and if the condition is met, the loop ends.

tasksEvents is an array of uint16 type, each element of which represents a type of task. That is to say, tasksEvents is a task pool, and tasksCnt is the size of this task pool.

The operating logic of this loop is:

- First, the initial value of idx is 0;
- When the value of tasksEvents[idx] is 0, it means that there is nothing to be processed in the task. At this time, the condition is not met and the next loop is entered;
- Before each loop is executed, idx is incremented by 1, and then a check is made to see if it is less than tasksCnt.
- When the value of tasksEvents[idx] is not equal to 0, it means that there is something to be processed in the task. At this time, the condition is met, so the loop ends through break;
- When the loop ends, if there are no tasks to be processed in the entire task pool, then idx must be \geq tasksCnt.

Therefore, if $idx < tasksCnt$, it means that there are tasks to be processed in the task pool, and tasksEvents[idx] is the task that needs to be processed. Therefore, after the loop ends, Z-Stack first uses the if ($idx < tasksCnt$) statement to determine whether there are any tasks to be processed.

Missions and Events

Each task may contain a series of pending things, which can be commonly referred to as "events". For example, a task may include three events (pending things): turning on the LED light, closing the window, and turning on the air conditioner.

Each element in tasksEvents is a variable of type uint16. Each element represents a task and stores a series of events contained in the task. So how does a variable of type uint16 store a series of events? I will explain this in detail in the following chapters.

6.3.2. Task initialization and event processing

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=26>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

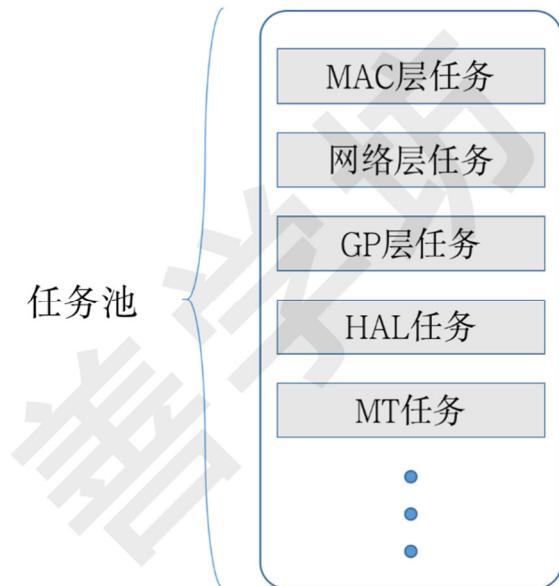
Hierarchy

From the previous chapters, we can see that Z-Stack can be divided into multiple levels, such as:

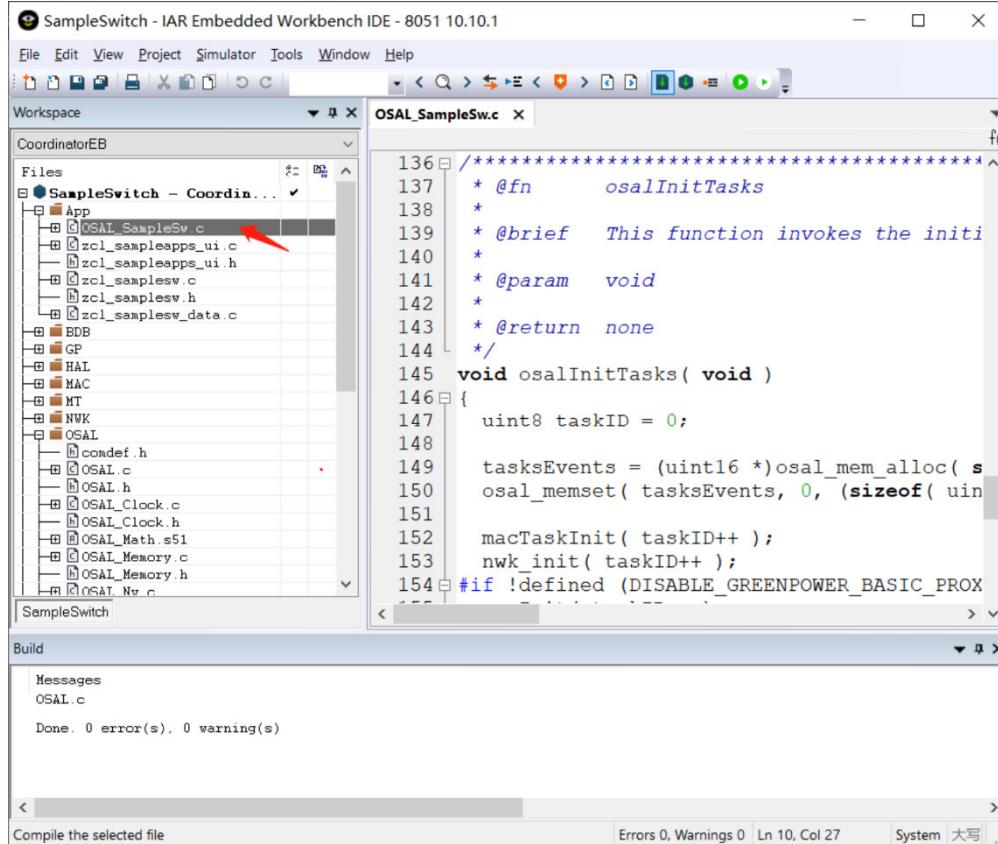
- MAC layer
- NWK (Network Layer)
- HAL (Hardware Adaptation Layer)
- APP (Application Layer)

Task pool initialization

Each level has a corresponding task to handle the affairs of this level. For example, the MAC layer corresponds to a MAC layer task, the network layer corresponds to a network layer task, the HAL corresponds to a HAL task, and the application layer corresponds to an application layer task, etc. These tasks at each level constitute a task pool, which is the tasksEvents array mentioned in the previous lesson, as shown in the figure.



The task pool initialization function osalInitTasks() mentioned in the previous lesson can be found in the OSAL_SampleSw.c file in the App directory, as shown in the figure.



The osalInitTasks() function definition code is as follows:

```
*****  
* @fn    osalInitTasks  
*  
* @brief This function invokes the initialization function for each task.  
*  
* @param void  
*  
* @return none  
*/  
  
void osalInitTasks( void )  
{  
    uint8 taskID = 0;  
  
    tasksEvents = (uint16 *)osal_mem_alloc( sizeof( uint16 ) * tasksCnt );  
    osal_memset( tasksEvents, 0, (sizeof( uint16 ) * tasksCnt) );  
  
    macTaskInit( taskID++ );  
    nwk_init( taskID++ );  
  
    #if !defined (DISABLE_GREENPOWER_BASIC_PROXY) && (ZG_BUILD_RTR_TYPE)  
        gp_Init( taskID++ );  
    #endif  
    Hal_Init( taskID++ );  
    #if defined( MT_TASK )  
        MT_TaskInit( taskID++ );  
    #endif
```

```

#endif

APS_Init( taskID++ );

#ifndef defined ( ZIGBEE_FRAGMENTATION )
APSF_Init( taskID++ );
#endif

ZDApp_Init( taskID++ );

#if defined ( ZIGBEE_FREQ_AGILITY ) || defined ( ZIGBEE_PANID_CONFLICT )
ZDNwkMgr_Init( taskID++ );
#endif

// Added to include TouchLink functionality

#if defined ( INTER_PAN )
StubAPS_Init( taskID++ );
#endif

// Added to include TouchLink initiator functionality

#if defined( BDB_TL_INITIATOR )
touchLinkInitiator_Init( taskID++ );
#endif

// Added to include TouchLink target functionality

#if defined ( BDB_TL_TARGET )
touchLinkTarget_Init( taskID++ );
#endif

zcl_Init( taskID++ );
bdb_Init( taskID++ );
zclSampleSw_Init( taskID++ );

#if (defined OTA_CLIENT) && (OTA_CLIENT == TRUE)
zclOTA_Init( taskID );
#endif

#endif
}

```

The code of this function is relatively complicated. For the time being, readers can simply follow the author's explanation to understand the principles behind it.

This function first applies for a task pool storage space, which is the tasksEvents array. Then it calls many functions with the word "init" to initialize tasks at various levels, for example:

- Call macTaskInit() function to initialize the MAC layer task.
- Call the nwk_init() function to initialize the network layer task.
- Call zclSampleSw_Init() function to initialize the application layer tasks.

This process is similar to creating and initializing a task pool in the "OSAL Task Scheduling Principles" section. The main differences are:

- First, the data structure stored in the task pool is different. Here, tasksEvents is an array of type uint16.
- Secondly, tasksEvents supports the storage of multiple types of tasks, such as MAC layer tasks, network layer tasks, and application layer tasks.
- Finally, each task here may contain multiple pending events.

Event Handler

An array is also defined in the OSAL_SampleSw.c file. The code is as follows:

```
//创建一个元素类型为pTaskEventHandlerFn的数组
1.const pTaskEventHandlerFn tasksArr[] = {
2.    macEventLoop, //第1个数组元素
3.    nwk_event_loop, //第2个数组元素
//  
//第3个数组元素
4.#if !defined (DISABLE_GREENPOWER_BASIC_PROXY) && (ZG_BUILD_RTR_TYPE)
5.    gp_event_loop,
6.#endif
//  
//第4个数组元素
7.    Hal_ProcessEvent,
//第5个数组元素
8.#if defined( MT_TASK )
9.    MT_ProcessEvent,
10.#endif
//  
//第6个数组元素
11.    APS_event_loop,
//  
//第7个数组元素
12.#if defined ( ZIGBEE_FRAGMENTATION )
13.    APSF_ProcessEvent,
14.#endif
//  
//第8个数组元素
15.    ZDApp_event_loop,
//  
//第9个数组元素
16.#if defined ( ZIGBEE_FREQ_AGILITY ) || defined ( ZIGBEE_PANID_CONFLICT )
17.    ZDNwkMgr_event_loop,
18.#endif
//  
//第10个数组元素
19.    //Added to include TouchLink functionality
20. #if defined ( INTER_PAN )
21.    StubAPS_ProcessEvent,
22. #endif
//
```

```

//第11个数组元素

23. // Added to include TouchLink initiator functionality
24. #if defined ( BDB_TL_INITIATOR )
25. touchLinkInitiator_event_loop,
26. #endif
//
//第12个数组元素

27. // Added to include TouchLink target functionality
28. #if defined ( BDB_TL_TARGET )
29. touchLinkTarget_event_loop,
30. #endif
//
31. zcl_event_loop, //第13个数组元素
32. bdb_event_loop. //第14个数组元素
33. zclSampleSw_event_loop, //第15个数组元素
//
//第16个数组元素

34.#if (defined OTA_CLIENT) && (OTA_CLIENT == TRUE)
35. zclOTA_event_loop
36.#endif

37.};

```

As mentioned in the previous section, each task in tasksEvents can contain 0 or more events to be processed, and the array type variable pTaskEventHandlerFn is a function pointer type variable used to point to the event corresponding to the processing function. Therefore, this code defines an event processing function array. Each element in this array represents the event processing function of a certain level of task, for example:

- The event processing function corresponding to the MAC layer task is macEventLoop(), which specifically handles events in the MAC layer task.
- The event processing function corresponding to the network layer task is nwk_event_loop(), which specifically handles events in the network layer task.
- The event processing function corresponding to the application layer task is zclSampleSw_event_loop(), which specifically handles events in the application layer task.

Taking the application layer event processing function zclSampleSw_event_loop() as an example, the declaration of the function can be found in the zcl_samplesw.h file. The code is as follows:

```

/*
 * Event Process for the task
*/
extern UINT16 zclSampleSw_event_loop( byte task_id, UINT16 events );

```

6.3.3. Application of Z-Stack Events

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

The content of this lesson is the foundation for subsequent courses. I hope everyone can study hard and lay a good foundation for subsequent courses.

Event type and code:

Readers can find that the parameters of each level of event processing function contain 1 task id and 1 events parameter, for example:

- MAC layer event processing function, as shown in the figure.

```
/* - - - - - Internal Functions - - - - - */
/* These functions are used when creating the OSAL MAC task. They must not be used for any other purpose.
 */
extern void macTaskInit(uint8 taskId);
extern uint16 macEventLoop(uint8 taskId, uint16 events);

/* - - - - - Functions - - - - - */
***** @fn MAC_Init
***** @brief This function initializes the MAC subsystem. It must be called once when the software system is started and before any other function in the MAC API is called.
```

- The network layer event processing function is shown in the figure.

```
/* FUNCTIONS */
/*
 * NWK Task Initialization
 */
extern void nwk_init( byte task_id );

/*
 * Calls mac_data_req then handles the buffering
 */
extern ZStatus_t nwk_data_req_send( nwkDB_t* db );

/*
 * NWK Event Loop
 */
extern UINT16 nwk_event_loop( byte task_id, UINT16 events );

/*
 * Process incoming command packet
 */
extern void CommandPktProcessing( NLDE_FrameFormat_t *ff );
//extern void CommandPktProcessing( NLDE_FrameFormat_t *ff );
```

- The application layer event processing function is shown in the figure.

```
extern uint8 zclSampleSw_OnOffSwitchType;
extern uint8 zclSampleSw_OnOffSwitchActions;
extern CONST uint8 zclSampleSw_NumAttributes;

/* - - - - - FUNCTIONS - - - - - */
/*
 * Initialization for the task
 */
extern void zclSampleSw_Init( byte task_id );

/*
 * Event Process for the task
 */
extern UINT16 zclSampleSw_event_loop( byte task_id, UINT16 events );

/*
 * Reset all writable attributes to their default values.
 */
extern void zclSampleSw_ResetAttributesToDefaultValues(void); //implemented in zcl_sampleSw.c
```

Taking the application layer event processing function as an example, its second parameter `UINT16 events` represents an event set, which contains 0 or more events to be processed. However, events is a 16-bit variable, how does it represent an event set?

The answer is that Z-Stack 3.0 uses a one-hot code approach to encode event types.

Classification of events

Before explaining the one-hot code, let's first understand the classification of events.

- When the highest bit of events is 1, it means that this is a system event set, that is, all events in events are system events.
- When the highest bit of events is 0, it means that this is a user event set, that is, all events in events are user events.

Developers can define the meaning of user events and handle them accordingly.

Use one-hot code to list

all user event codes and analyze the rules of one-hot code, as shown in the table below.

Binary encoding	Hexadecimal encoding	Event Name
0000 0000 0000 0001	0x00 01	User Event A
0000 0000 0000 0010	0x00 02	User Event B
0000 0000 0000 0100	0x00 04	User Event C
0000 0000 0000 1000	0x00 08	User Event D
0000 0000 0001 0000	0x00 10	User Event E
0000 0000 0010 0000	0x00 20	User Event F
0000 0000 0100 0000	0x00 40	User Event G
0000 0000 1000 0000	0x00 80	User Event H
0000 0001 0000 0000	0x01 00	User Event I
0000 0010 0000 0000	0x02 00	User Event J
0000 0100 0000 0000	0x04 00	User Event K
0000 1000 0000 0000	0x08 00	User Event L
0001 0000 0000 0000	0x10 00	User Event M
0010 0000 0000 0000	0x20 00	User Event N
0100 0000 0000 0000	0x40 00	User Event O

The event name can be named according to actual needs, such as a light-on event, a light-off event, or a warning event.

From these codes, we can draw two rules:

1. Except for the highest bit used to indicate system events or user events, among the other 15 bits, only one bit is 1, and the other bits are all 0.
2. 15 bits are used to represent 15 types of user events.

These two rules are also the rules of one-hot codes.

Using rule 1, we can easily understand why events can represent an event set. Now assume that the value of events is 0000 0101 0101, where the 1st, 3rd, 5th, and 7th bits from the right are 1, so we can understand that the event set events contains user events A, C, E, and G.

Using rule 2, we can get that events can contain up to 15 types of user events.

Defining User Events

You can use the following method to define a user event in the zcl_samplesw.h file:

1. Define the event name and the corresponding code.

```
#define SAMPLEAPP_TEST_EVT 0x0040
```

2. Copy it to the location shown in the figure in the zcl_samplesw.h file.

```
SampleSwitch - IAR Embedded Workbench IDE - 8051 10.10.1
File Edit View Project Texas Instruments Emulator Tools Window Help
Workspace CoordinatorEB
Files
SampleSwitch - CoordinatorEB
App
OSAL_SampleSw.c
zcl_sampleapps_ui.c
zcl_sampleapps_ui.h
zcl_samplesw.c
zcl_samplesw.h
zcl_samplesw_data.c
BDB
GP
HAL
MAC
MT
NNWK
OSAL
comdefh
OSAL.c
OSAL.h
OSAL_Clock.c
OSAL_Clock.h
OSAL_Math.s51
OSAL_Memory.c
OSAL_Memory.h
OSAL_Nv.c
OSAL_Nv.h
OSAL_PwrMgr.c
OSAL_PwrMgr.h
OSAL_Tasks.h
OSAL_Timers.c
OSAL_Timers.h
ZComDef.h
Profile
Security
Services
Tools
ZDO
ZMac
ZMain
chipcon_cstartup.s51
OnBoard.c
OnBoard.h
ZMain.c
Output
zcl_samplesw.h
44
45 #ifdef __cplusplus
46 extern "C"
47 {
48 #endif
49
50 ****
51 * INCLUDES
52 */
53 #include "zcl.h"
54
55 ****
56 * CONSTANTS
57 */
58 #define SAMPLESW_ENDPOINT 8
59
60 #define LIGHT_OFF 0x00
61 #define LIGHT_ON 0x01
62
63 // Events for the sample app
64 #define SAMPLEAPP_END_DEVICE_REJOIN_EVT 0x0001
65
66 // UI Events
67 #define SAMPLEAPP_LCD_AUTO_UPDATE_EVT 0x0010
68 #define SAMPLEAPP_KEY_AUTO_REPEAT_EVT 0x0020
69
70 // Test
71 #define SAMPLEAPP_TEST_EVT 0x0040
72
73
74 #define SAMPLEAPP_END_DEVICE_REJOIN_DELAY 10000
75
76 ****
77 * MACROS
78 */
79 ****
80 * TYPEDEFS
81 */
82
```

Handling User Events

You can add relevant processing in the application layer event processing function in the zcl_samplesw.c file. The code is as follows:

```
uint16 zclSampleSw_event_loop( uint8 task_id, uint16 events )
{
    afIncomingMSGPacket_t *MSGpkt;
    (void)task_id; // Intentionally unreferenced parameter
```

```
//用户事件 : SAMPLESW_TOGGLE_TEST_EVT
if( events & SAMPLESW_TOGGLE_TEST_EVT )
{
    osal_start_timerEx(zclSampleSw_TaskID,SAMPLESW_TOGGLE_TEST_EVT,500);
    zclGeneral_SendOnOff_CmdToggle( SAMPLESW_ENDPOINT, &zclSampleSw_DstAddr, FALSE, 0 );
}

//消除已经处理的事件然后返回未处理的事件
return (events ^ SAMPLESW_TOGGLE_TEST_EVT);
}

//SYS_EVENT_MSG : 0x8000表示系统事件 . 也就是说检测uint16最高位
if ( events & SYS_EVENT_MSG )
{
    //省略系统事件的处理代码
    .....
}

// 消除系统事件标识然后返回未处理的事件
return (events ^ SYS_EVENT_MSG);
}

#ifndef ZG_BUILD_ENDDEVICE_TYPE
//用户事件 : SAMPLEAPP_END_DEVICE_REJOIN_EVT
if ( events & SAMPLEAPP_END_DEVICE_REJOIN_EVT )
{
    bdb_ZedAttemptRecoverNwk();
    return ( events ^ SAMPLEAPP_END_DEVICE_REJOIN_EVT );
}
#endif

//用户事件 : SAMPLEAPP_LCD_AUTO_UPDATE_EVT
if ( events & SAMPLEAPP_LCD_AUTO_UPDATE_EVT )
{
    UI_UpdateLcd();
    return ( events ^ SAMPLEAPP_LCD_AUTO_UPDATE_EVT );
}

//用户事件 : SAMPLEAPP_KEY_AUTO_REPEAT_EVT
if ( events & SAMPLEAPP_KEY_AUTO_REPEAT_EVT )
{
    UI_MainStateMachine(UI_KEY_AUTO_PRESSED);
    return ( events ^ SAMPLEAPP_KEY_AUTO_REPEAT_EVT );
}
```

```
// 处理刚才自己定义的用户事件 : SAMPLEAPP_TEST_EVT
```

```
if ( events & SAMPLEAPP_TEST_EVT )  
{  
    printf("Hello World!\r\n");  
  
    // 消除已经处理的事件然后返回未处理的事件  
    return ( events ^ SAMPLEAPP_TEST_EVT );  
}  
  
// Discard unknown events  
return 0;  
}
```

From the previous explanation, we can know that in each user event type code, only one bit is 1, and the other bits are 0. The event type code of SAMPLEAPP_TEST_EVT is 0x0040, and its binary number is: 0000 0000 0100 0000. The 7th bit from the right of this code is 1, and the rest are 0.

Therefore, the above code uses events & SAMPLEAPP_TEST_EVT to perform an AND operation on the event collection parameter events and the predefined event type SAMPLEAPP_TEST_EVT to determine whether the third bit from the right in events is 1. If it is 1, the value of events & SAMPLEAPP_TEST_EVT is 1, which means that the event collection parameter events contains the event SAMPLEAPP_TEST_EVT, so the program executes the corresponding processing code, that is, executes:

```
printf("Hello World!\r\n");
```

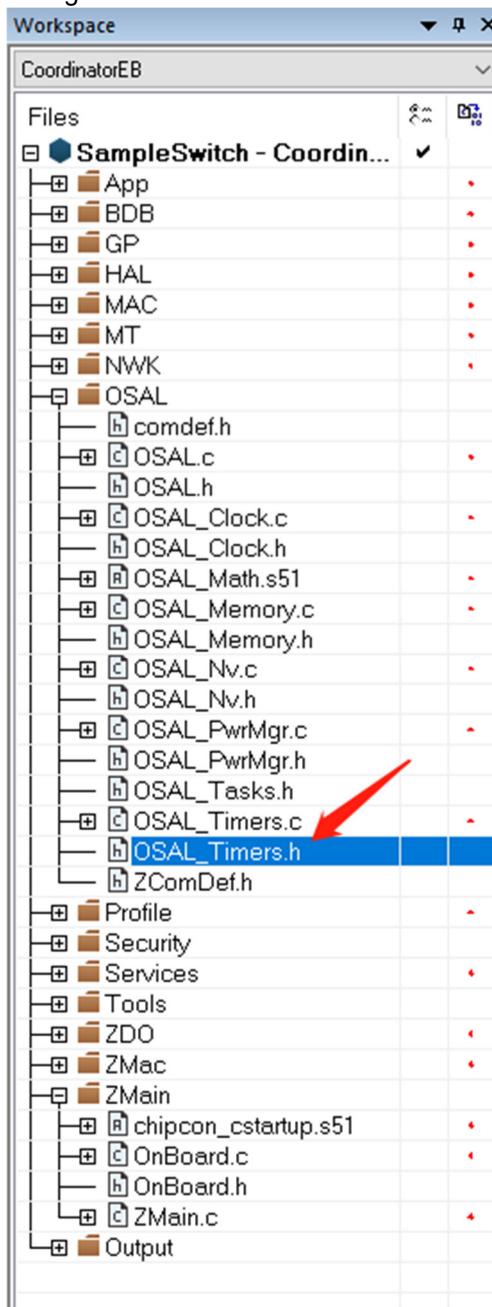
Next, the code uses events ^ SAMPLEAPP_TEST_EVT to clear the third bit in events to 0, and then uses this value as the return value of the function, indicating that the event in events has been processed.

This code uses the & and ^ operations. If you are not familiar with these two operations, you need to review them first.

Triggering user events

The event type and the corresponding processing method have been defined before, but OSAL will execute the corresponding processing code only after the event is triggered in OSAL.

OSAL provides a special API to trigger events. Expand the OSAL layer and you can find the OSAL_Timers.h file, as shown in the figure.



In the OSAL_Timers.h file, you can find the API for triggering events. The function declaration is as follows:

```
uint8 osal_start_timerEx(uint8 task_id,uint16 event_id,uint32 timeout_value);
```

This function has three parameters, which are described as follows:

- task_id: Task ID, used to mark the level of task to which this event belongs.
- event_id: event ID, used to mark the type of this event.
- timeout_value: indicates how many milliseconds it will take to process this event.

If you want to process the customized event 3 seconds after the event is triggered, you can add the following code at the end of the application layer initialization function zclSampleSw_Init():

```
osal_start_timerEx(  
    zclSampleSw_TaskID,//标记本事件属于应用层任务  
    SAMPLEAPP_TEST_EVT,//标记本事件的类型  
    3000);//表示3000ms后才处理这个事件
```

Among them, zclSampleSw_TaskID is a global variable used to mark that this event belongs to the application layer task.

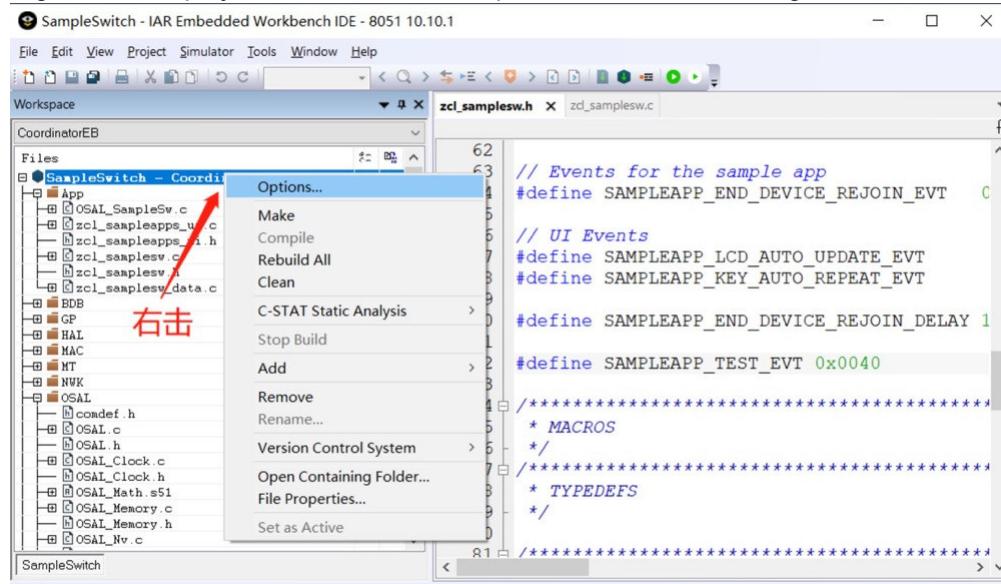
The effect after adding the code is shown in the figure.

The screenshot shows a code editor window with a file named `zd_samplesw.c`. A red arrow points to the function name `zclSampleSw_Init` at the top. Another red arrow points to the line of code `osal_start_timerEx(zclSampleSw_TaskID, SAMPLEAPP_TEST_EVT, 3000);` which has been highlighted with a purple box. A third red arrow points to the closing brace of the function definition at line 297, with the text "函数结尾大括号!" (Function end brace!) written below it.

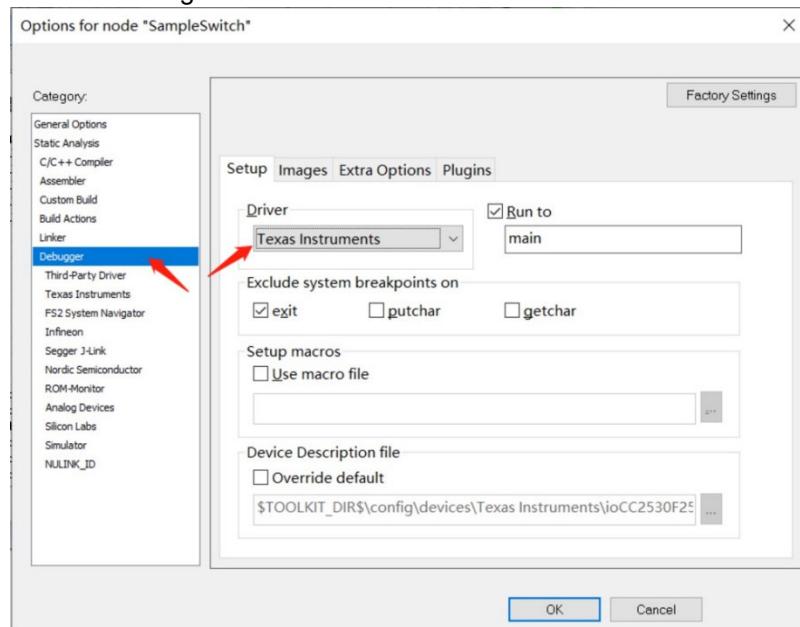
```
zd_samplesw.c x 文件: zd_samplesw.c
zclSampleSw_Init(byte) 函数名: zclSampleSw_Init
1292
1293     // Set test event!!!!!
1294     osal_start_timerEx(zclSampleSw_TaskID,
1295                         SAMPLEAPP_TEST_EVT,
1296                         3000);
1297 }
1298
```

Debugging Simulation

Right-click the project name and select Options, as shown in the figure.

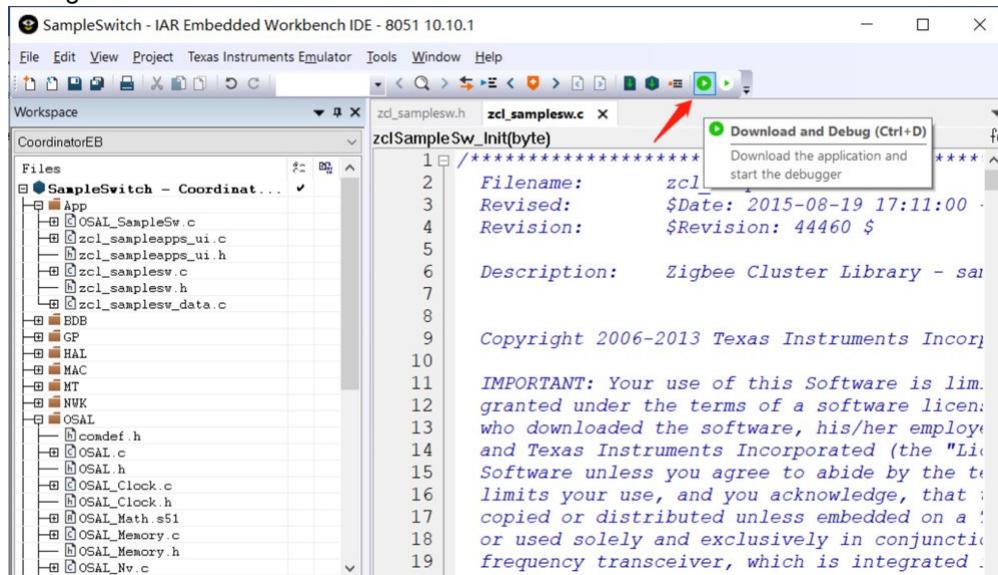


Select Debugger, then select Texas Instruments in the Driver tab, and finally click the OK button to complete the setup, as shown in the figure.



Use the emulator to connect the matching ZigBee development board to the computer.

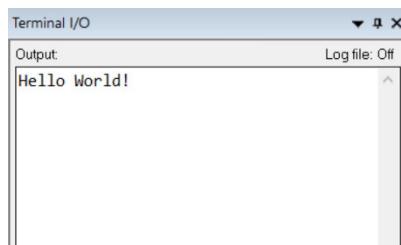
Click the Download and Debug button to compile, link, and download the program and enter the simulation mode, as shown in the figure.



After entering the simulation mode, click the Go button to run the program, as shown in the figure.



After 3 seconds of running, the program will output "Hello World!" in the Terminal I/O window, as shown in the figure.



6.3.4. Using Dynamic Memory

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=28>

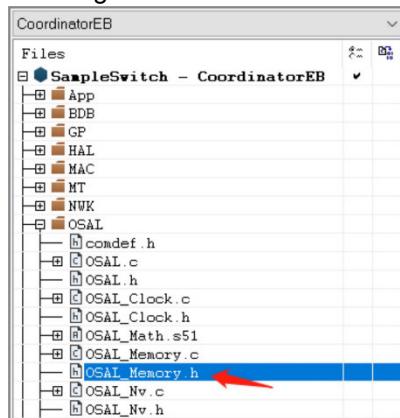
Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers **will** answer community questions as soon as possible, but they are front-line developers and [**cannot guarantee**] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Introduction to Dynamic Memory

When writing a program, the developer may not be sure how much memory is needed to store data, and hopes to use the corresponding memory space according to actual needs **during the program running**. This kind of memory space that can be applied and released during the program running is called **dynamic memory**.

The API for dynamic memory allocation in Z-Stack 3.0 is in the OSAL_Memory.h file, which is in the OSAL directory, as shown in the figure.



Allocating and releasing dynamic memory

In the OSAL_Memory.h file, you can find two important APIs, defined as follows:

```
/**  
 * @fn osal_mem_alloc  
 *  
 * @brief 动态申请内存空间  
 *  
 * @param size - 申请多少个字节的内存空间  
 *  
 * @return 返回该内存空间的指针  
 */  
void *osal_mem_alloc( uint16 size );
```

```
/**  
 * @fn osal_mem_free  
 *  
 * @brief 动态释放内存空间  
 *  
 * @param ptr - 待释放的内存空间指针  
 *  
 */  
void osal_mem_free( void *ptr );
```

If the memory space is not used, be sure to call this API to release the memory space.

Dynamic memory operations

After applying for dynamic memory, you can call the memory operation API to use the memory. The memory operation API is in the OSAL.h file. The following two APIs are commonly used and are defined as follows:

```
/**  
 * @fn osal_memcpy  
 *  
 * @brief 把内存空间的内容复制到另一个内存空间中
```

```

*
* @param void* - 目标内存空间
*
* @param const void GENERIC * - 源内存空间
*
* @param unsigned int - 复制多少个字节
*
* @return
*/
void *osal_memcpy(void*, const void GENERIC *,unsigned int);

/**
* @fn osal_memset
*
* @brief 把内存空间的值设置为指定的值
*
* @param dest - 内存空间
* @param value - 指定的值
* @param len - 把从dest起的len个字节的存储空间的值设置为value
*
* @return
*/
extern void *osal_memset( void *dest, uint8 value, int len );

```

Using dynamic memory

Make some changes to the event handling function in the previous lesson. The modified code is as follows:

```

//事件 : SAMPLEAPP_TEST_EVT
if ( events & SAMPLEAPP_TEST_EVT )
{
    //字符串 : "Hello World!\n"
    char *str = "Hello World!\n";

    //从堆空间中申请32个字节的内存空间
    char *mem = osal_mem_alloc(32);

    //如果申请成功
    if (mem != NULL) {
        //清零内存空间
        osal_memset(mem, 0, 32);

        //将字符串拷贝到内存空间中
        osal_memcpy(mem, str, osal_strlen(str));
    }
}

```

```

//打印内存空间内存
printf(mem);

//释放内存空间
osal_mem_free(mem);

}

//重新触发事件，3000毫秒后执行
osal_start_timerEx(zclSampleSw_TaskID, SAMPLEAPP_TEST_EVT, 3000);

//消除已经处理的事件，然后返回未处理的事件
return ( events ^ SAMPLEAPP_TEST_EVT );
}

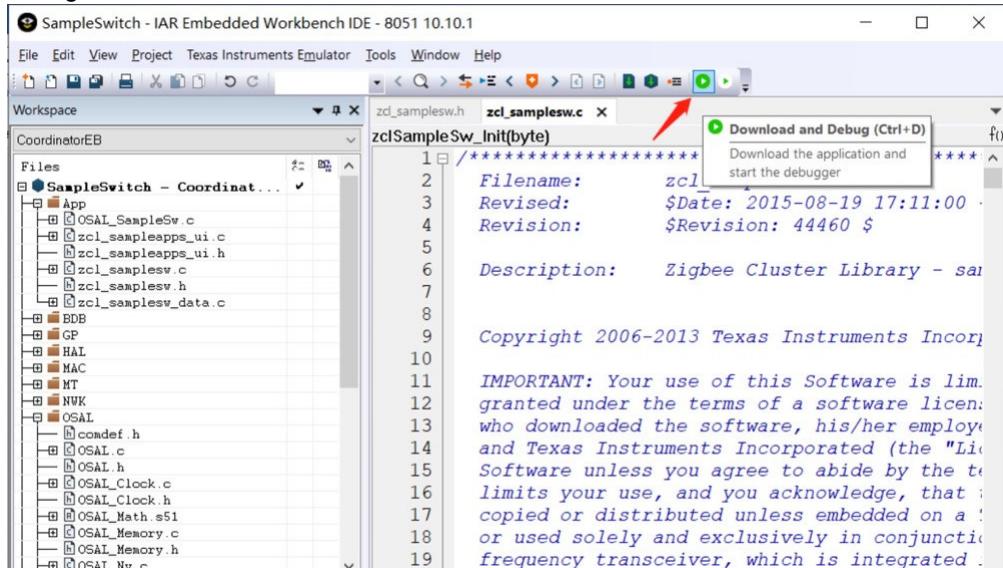
```

In this code, after the event processing is completed, the event is re-triggered, so that the event is continuously triggered at a constant time interval.

Debugging Simulation

Use the emulator to connect the matching ZigBee development board to the computer.

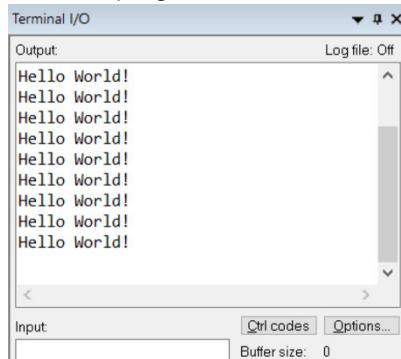
Click the Download and Debug button to compile, link, and download the program and enter the simulation mode, as shown in the figure.



After entering the simulation mode, click the Go button to run the program, as shown in the figure.



After the program runs, "Hello World!" will be output in the Terminal I/O window every 3 seconds, as shown in the figure.



6.4. Chapter 4: Hardware Adaptation Layer Application - LED

6.4.1. HAL file structure and project structure

6.4.2. HAL Architecture Overview

6.4.3. Introduction to LED API

6.4.4. LED Experiment

6.4.1. HAL file structure and project structure

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=29>

Technical support instructions:

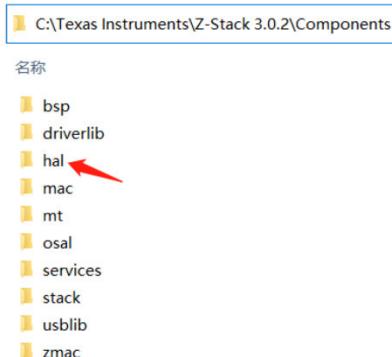
1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

The Hardware Adaptation Layer (HAL) is used to adapt to various hardware and provide a unified interface for the upper layer, making it easier for developers to use various hardware. This lesson briefly explains the structure of HAL from the two aspects of file structure and project structure.

File structure of HAL

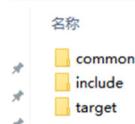
The file structure of HAL is relatively complex, and readers only need to follow the author's explanation to understand it briefly.

After installing Z-Stack 3.0, the HAL-related source code can be found in the Components folder of Z-Stack 3.0, as shown in the figure.



Enter the hal directory, as shown in the figure.

3. OSAL 详解 > Z-Stack 3.0.1 > Components > hal



The folders are as follows:

- (1) common: stores common files.
- (2) include: stores various driver interface files.
- (3) target: stores driver source files for various types of motherboards.

Enter the common folder, as shown in the figure.

C:\Texas Instruments\Z-Stack 3.0.2\Components\hal\common

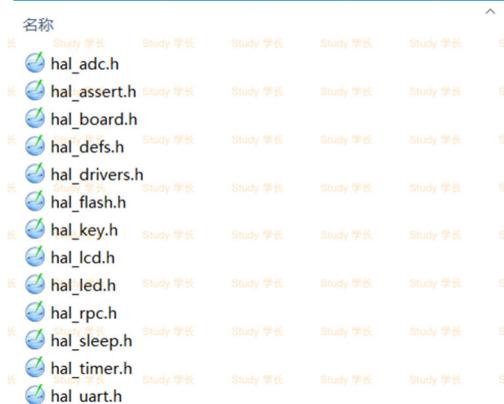


The files in the common directory are described as follows:

- (1) hal_assert.c: source code file that implements the conditional validity judgment function.
- (2) hal_drivers.c: the file where the hardware abstraction layer HAL task initialization and event processing functions are located, which is the entry source file of HAL.

Return to the previous level and enter the include directory, as shown in the figure.

C:\Texas Instruments\Z-Stack 3.0.2\Components\hal\include



The files in the include directory are described as follows:

- (1) hal_adc.h: ADC (analog-to-digital conversion) driver header file.
- (2) hal_assert.h: Header file for conditional validity judgment function.
- (3) hal_board.h: Hardware resource configuration header file for various types of motherboards.
- (4) hal_defs.h: General definition header file.
- (5) hal_drivers.h: Header file for HAL task initialization and event processing functions.
- (6) hal_flash.h: FLASH (memory) driver header file.
- (7) hal_key.h: Key driver header file.
- (8) hal_lcd.h: Display driver header file.
- (9) hal_led.h: LED driver header file.
- (10) hal_rpc.h: RPC (Remote Procedure Call) driver header file.
- (11) hal_sleep.h: Sleep function driver header file.
- (12) hal_timer.h: Timer driver header file.
- (13) hal_uart.h: serial port driver header file.

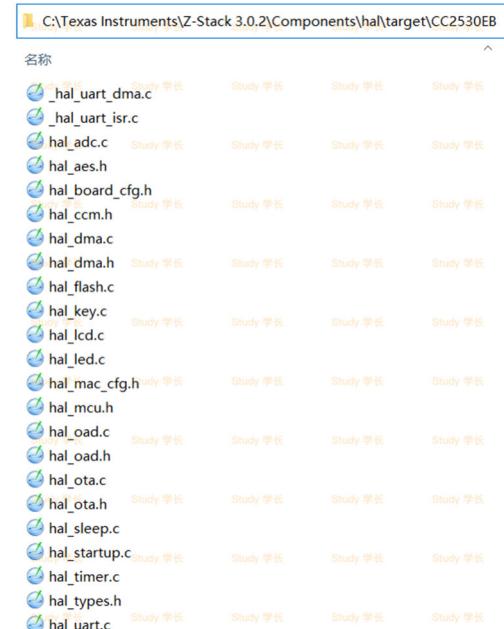
Go back one level and enter the target folder, as shown in the figure.



The folders in the target directory are described as follows:

- (1) CC2530EB: Drivers for the evaluation board with CC2530 as the chip controller.
- (2) CC2530USB: Drivers for the evaluation board with USB to serial port and CC2530 as the chip controller.
- (3) CC2530ZNP: Drivers for the ZNP (ZigBee And Processor) evaluation board with CC2530 as the chip controller.
- (4) CC2538: Drivers for the evaluation board with CC2538 as the chip controller.
- (5) CC2538ZNP: Drivers for the ZNP (ZigBee And Processor) evaluation board with CC2538 as the chip controller.

Taking CC2530EB as an example, enter this directory, as shown in the figure.



There are two types of hardware driver source files stored in this directory:

- One is the source code file that implements the interface in the include directory just introduced, such as hal_led, hal_lcd.c, hal_adc.c, etc.
- The other is the driver source files related to the current motherboard type, such as DMA (Direct Memory Access), OTA (Over The Air), etc.

Let's briefly explain the various files in this directory:

- (1) _hal_uart_dma.c: Serial port DMA (Direct Memory Access) driver source file.
- (2) _hal_uart_isr.c: Serial port ISR (Interrupt Service Routines) driver source file.
- (3) hal_adc.c: ADC (Analog-to-Digital Converter) driver source file
- (4) hal_aes.h: AES (Advanced Encryption Standard) driver header file.
- (5) hal_board_cfg.h: Hardware resource configuration header file for the current motherboard.
- (6) hal_ccm.h: CCM (Counter with CBC-MAC) driver header file.
- (7) hal_dma.c: DMA (Direct Memory Access) driver source file.
- (8) hal_dma.h: DMA (Direct Memory Access) driver header file.
- (9) hal_flash.c: FLASH (memory) driver source file.
- (10) hal_key.c: Key driver source file.
- (11) hal_lcd.c: Display driver interface file.
- (12) hal_led.h: LED (light-emitting diode) driver header file.
- (13) hal_mac_cfg.h: MAC (Medium Access Control) related configuration header file.
- (14) hal_mcu.h: Driver header file related to the main control chip.
- (15) hal_oad.c: OAD (On Air Download) driver source file.

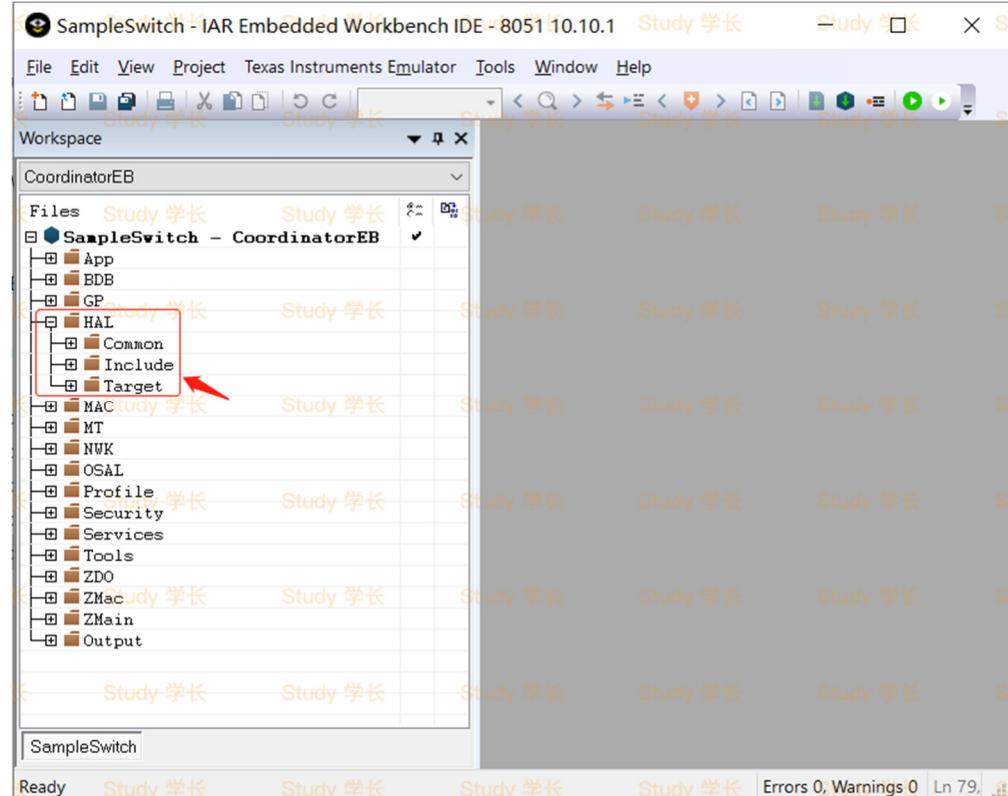
- (16) hal_oad.h: OAD (On Air Download) driver header file.
- (17) hal_ota.c: OTA (Over The Air) driver source file.
- (18) hal_ota.h: OTA (Over The Air) driver header file.
- (19) hal_sleep.c: Sleep function driver source file.
- (20) hal_startup.c: System startup related driver source file.
- (21) hal_timer.c: Timer driver source file.
- (22) hal_types.h: Type definition header file.
- (23) hal_uart.c: Serial port driver source file.

HAL Project Structure

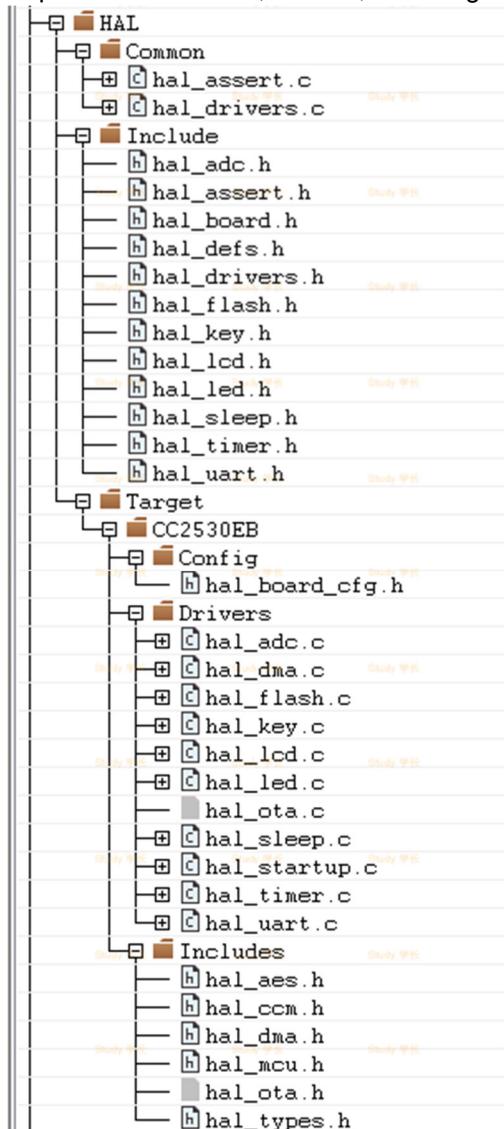
This section will take the SampleSwitch routine as an example to explain the HAL project file structure.

Project Organization Structure

Open the SampleSwitch project and find the directory where the HAL is located, as shown in the figure.



Expand the Common, Include, and Target directories respectively, as shown in the figure.



There is a CC2530EB directory in the Target directory, which means that this project uses the driver files related to the CC2530EB motherboard.

The CC2530EB directory contains three directories: Config, Drivers, and Includes. The Config directory stores the motherboard's hardware resource configuration header files, the Drivers directory stores various driver source files, and the Includes directory stores various driver header files.

6.4.2. HAL Architecture Overview

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=30>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

This lesson continues to use the SampleSwitch routine as an example to explain the HAL architecture. Similarly, readers only need to follow the author's explanation ideas to roughly understand the content of this lesson.

HAL Architecture Overview

Initialize

the HAL initialization function Hal_Init() in the hal_drivers.c file, the code is as follows:

```
/*
 * @fn    Hal_Init
 *
 * @brief Hal Initialization function.
 *
 * @param task_id - Hal TaskId
 *
 * @return None
 */
void Hal_Init( uint8 task_id )
{
    /* Register task ID */
    Hal_TaskID = task_id;

#ifndef CC2591_COMPRESSION_WORKAROUND
    osal_start_reload_timer( Hal_TaskID, PERIOD_RSSI_RESET_EVT, PERIOD_RSSI_RESET_TIMEOUT );
#endif
}
```

Driver initialization

In the hal_drivers.c file, you can find the driver initialization function HalDriverInit(). The main function of this function is to initialize various hardware peripherals. The code is as follows:

```
/*
 * @fn    Hal_DriverInit
 *
 * @brief Initialize HW - These need to be initialized before anyone.
 *
 * @return None
 */
void HalDriverInit (void)
{
    //定时器 · 通过设置宏定义HAL_TIMER为TRUE来使能该功能
    #if (defined HAL_TIMER) && (HAL_TIMER == TRUE)
    #endif

    //模数转换功能 · 通过设置宏定义HAL_ADC为TRUE来使能该功能
    #if (defined HAL_ADC) && (HAL_ADC == TRUE)
        HalAdcInit();
    #endif

    //DMA (直接存储器访问) · 通过设置宏定义HAL_DMA为TRUE来使能该功能
    #if (defined HAL_DMA) && (HAL_DMA == TRUE)
        HalDmaInit();
    #endif
}
```

```
//AES (高级加密标准) · 通过设置宏定义HAL_AES为TRUE来使能该功能
```

```
#if (defined HAL_AES) && (HAL_AES == TRUE)
```

```
    HalAesInit();
```

```
#endif
```

```
//显示器 · 通过设置宏定义HAL_LCD为TRUE来使能该功能
```

```
#if (defined HAL_LCD) && (HAL_LCD == TRUE)
```

```
    HalLcdInit();
```

```
#endif
```

```
//LED, 通过设置宏定义HAL_LED为TRUE来使能该功能
```

```
#if (defined HAL_LED) && (HAL_LED == TRUE)
```

```
    HalLedInit();
```

```
#endif
```

```
//串口, 通过设置宏定义HAL_UART为TRUE来使能该功能
```

```
#if (defined HAL_UART) && (HAL_UART == TRUE)
```

```
    HalUARTInit();
```

```
#endif
```

```
//按键 · 通过设置宏定义HAL_KEY为TRUE来使能该功能
```

```
#if (defined HAL_KEY) && (HAL_KEY == TRUE)
```

```
    HalKeyInit();
```

```
#endif
```

```
//SPI, 通过设置宏定义HAL_SPI为TRUE来使能该功能
```

```
#if (defined HAL_SPI) && (HAL_SPI == TRUE)
```

```
    HalSpiInit();
```

```
#endif
```

```
//HID (Human Interface Device), 通过设置宏定义HAL_HID为TRUE来使能该功能
```

```
#if (defined HAL_HID) && (HAL_HID == TRUE)
```

```
    usbHidInit();
```

```
#endif
```

```
}
```

In the actual development process, developers can use specified peripherals according to their needs. In this code, it can be seen that if you need to use a specified peripheral, you only need to define the corresponding macro. For example, if you want to use LED, then define the HAL_LED macro as TRUE. If you don't need it, then don't define it. After defining the corresponding macro, the corresponding initialization will be executed, such as the LED initialization function HalLedInit(). The specific macro definition method will be described in the subsequent chapters.

If developers need to add other peripherals, they can add peripherals according to this architecture of the protocol stack.

Event processing

HAL's event processing function Hal_ProcessEvent() is in the hal_drivers.c file. Its main function is to process events at the HAL layer. The code is as follows:

```

/*****
 * @fn    Hal_ProcessEvent
 *
 * @brief Hal Process Event
 *
 * @param task_id - Hal TaskId
 *        events - events
 *
 * @return None
 ****/

```

uint16 Hal_ProcessEvent(uint8 task_id, uint16 events)

```

{
    uint8 *msgPtr;

    (void)task_id; // Intentionally unreferenced parameter

    if ( events & SYS_EVENT_MSG )
    {
        //省略系统事件的处理代码
        .....
        return events ^ SYS_EVENT_MSG;
    }

    //蜂鸣器事件，需要宏定义HAL_BUZZER为TRUE才使能该事件
    #if (defined HAL_BUZZER) && (HAL_BUZZER == TRUE)
        if (events & HAL_BUZZER_EVENT)
        {
            HalBuzzerStop();
            return events ^ HAL_BUZZER_EVENT;
        }
    #endif

    //RSSI重置事件，需要宏定义PERIOD_RSSI_RESET_EVT为TRUE才使能该事件
    #ifdef CC2591_COMPRESSION_WORKAROUND
        if (events & PERIOD_RSSI_RESET_EVT)
        {
            macRxResetRssi();
            return (events ^ PERIOD_RSSI_RESET_EVT);
        }
    #endif

    //LED闪烁事件
    if (events & HAL_LED_BLINK_EVENT)

```

```

{
#ifndef (defined (BLINK_LEDS)) && (HAL_LED == TRUE)
    HalLedUpdate();
#endif /* BLINK_LEDS && HAL_LED */

    return events ^ HAL_LED_BLINK_EVENT;
}

//按键事件

if (events & HAL_KEY_EVENT)

{
#ifndef (defined HAL_KEY) && (HAL_KEY == TRUE)
/* Check for keys */

    HalKeyPoll();

/* if interrupt disabled, do next polling */

    if (!Hal_KeyIntEnable)

    {
        osal_start_timerEx( Hal_TaskID, HAL_KEY_EVENT, 100);
    }
#endif

    return events ^ HAL_KEY_EVENT;
}

//低功耗事件，需要宏定义HAL_SLEEP_TIMER_EVENT为TRUE才使能该事件

#ifndef defined POWER_SAVING
if (events & HAL_SLEEP_TIMER_EVENT)

{
    halRestoreSleepLevel();

    return events ^ HAL_SLEEP_TIMER_EVENT;
}

if (events & HAL_PWRMGR_HOLD_EVENT)

{
    (void)osal_pwrmgr_task_state(Hal_TaskID, PWRMGR_HOLD);

    (void)osal_stop_timerEx(Hal_TaskID, HAL_PWRMGR_CONSERVE_EVENT);
    (void)osal_clear_event(Hal_TaskID, HAL_PWRMGR_CONSERVE_EVENT);

    return (events & ~(HAL_PWRMGR_HOLD_EVENT | HAL_PWRMGR_CONSERVE_EVENT));
}

if (events & HAL_PWRMGR_CONSERVE_EVENT)

{
    (void)osal_pwrmgr_task_state(Hal_TaskID, PWRMGR_CONSERVE);

    return events ^ HAL_PWRMGR_CONSERVE_EVENT;
}

```

```
}
```

```
#endif
```

```
return 0;
```

```
}
```

HAL polling

In the hal_drivers.c file, you can find the Hal_ProcessPoll() function. The main function of this function is to poll those functional modules that need to be processed quickly. The code is as follows:

```
/*****************************************************************************  
 * @fn    Hal_ProcessPoll  
 *  
 * @brief This routine will be called by OSAL to poll UART, TIMER...  
 *  
 * @return None  
******/  
  
void Hal_ProcessPoll ()  
{  
    //是否进入低功耗模式  
    #if defined( POWER_SAVING )  
        ALLOW_SLEEP_MODE();  
    #endif  
  
    //串口  
    #if (defined HAL_UART) && (HAL_UART == TRUE)  
        HalUARTPoll();  
    #endif  
  
    //SPI  
    #if (defined HAL_SPI) && (HAL_SPI == TRUE)  
        HalSpiPoll();  
    #endif  
  
    //HID  
    #if (defined HAL_HID) && (HAL_HID == TRUE)  
        usbHidProcessEvents();  
    #endif  
  
}
```

So far, we have briefly introduced the overall structure of HAL. Next, we will explain how to use HAL to use basic hardware resources such as LEDs, buttons, serial ports, displays, and ADC.

6.4.3. Introduction to LED API

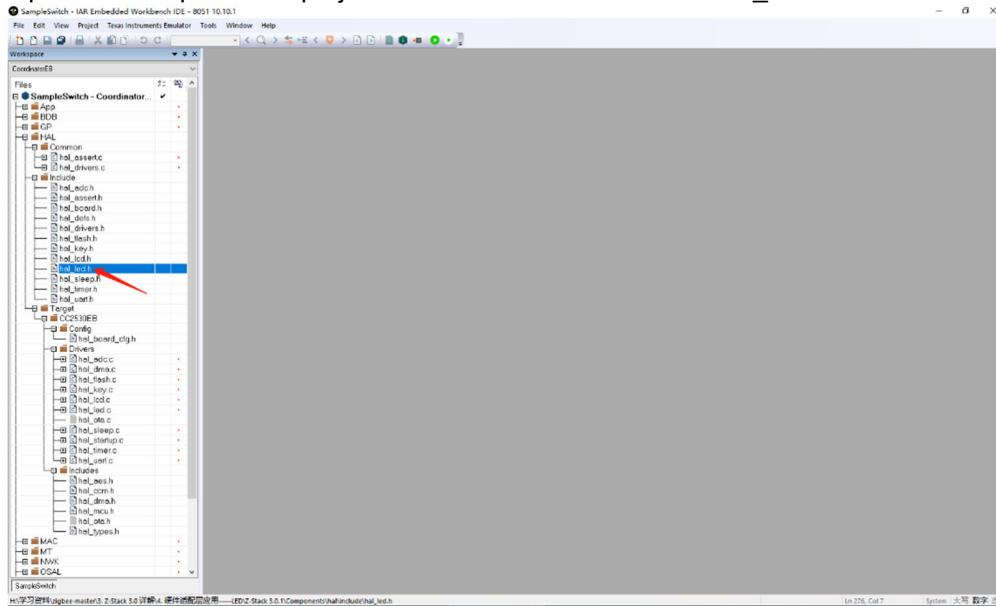
Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=31>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Related API Description

Open the SampleSwitch project and find the LED API in the hal_led.h file. The file location is shown in the figure.



In this header file, there are 3 commonly used APIs, the code is as follows:

```
/*
 * 初始化LED
 */

extern void HalLedInit( void );

/*
 * 开关LED
 */

extern uint8 HalLedSet( uint8 led, uint8 mode );

/*
 * 闪烁LED
 */

extern void HalLedBlink( uint8 leds, uint8 cnt, uint8 duty, uint16 time );
```

HalLedInit() is used to initialize the LED. In the previous chapter, we explained that this function is called in the driver initialization function HalDriverInit() in the hal_drivers.c file.

In Z-Stack 3.0, HAL supports 4 LEDs by default. The logic definition of LEDs is as follows:

```
/* LEDS - The LED number is the same as the bit position */

#define HAL_LED_1 0x01
#define HAL_LED_2 0x02
#define HAL_LED_3 0x04
#define HAL_LED_4 0x08
```

```
#define HAL_LED_ALL (HAL_LED_1 | HAL_LED_2 | HAL_LED_3 | HAL_LED_4)
```

HalLedSet(uint8 led, uint8 mode) is used to set the working mode of the LED. The first parameter led is used to specify the LED to be set, and the second parameter mode is used to specify the working mode. There are three working modes:

- (1) Turn on, the corresponding macro definition is HAL_LED_MODE_ON.
- (2) Turn off, the corresponding macro definition is HAL_LED_MODE_OFF.
- (3) Toggle, the corresponding macro definition is HAL_LED_MODE_TOGGLE.

For example, if you want to turn on the first LED, you can call it in this way:

```
HalLedSet(  
    HAL_LED_1,//指定第1盏LED  
    HAL_LED_MODE_ON);//开启
```

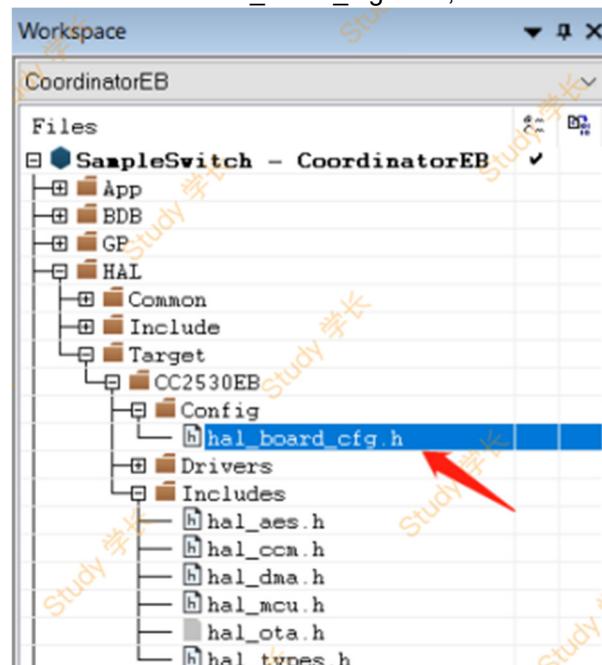
HalLedBlink(uint8 leds, uint8 cnt, uint8 duty, uint16 time) is used to make the LED blink. The first parameter leds is used to specify the LED, the second parameter cnt is used to specify the number of blinks, the third parameter duty is used to specify the duty cycle when the LED is turned on, and the fourth parameter time is used to specify the time period of each blink (unit: ms).

Let's take an example to illustrate how to use it. For example, if you need to make the first LED flash 10 times, each flashing period is 2000ms, and 60% of the time in this 2000ms the LED should be on, then you can call it as follows:

```
HalLedBlink(  
    HAL_LED_1,//指定第1盏LED  
    10,//指定闪烁次数是10次  
    60,//指定60%的时间LED是处于开启状态  
    2000);//指定1次闪烁的时间周期是2000ms, 也就是在1次闪烁周期中·LED的开启时长是1200ms, 关闭时长是800ms
```

Physical mapping of LEDs

Before calling the LED API, you need to configure the physical mapping definition of the LED. The physical mapping definition of the LED is in the hal_board_cfg.h file, and the location of the file is shown in the figure.



The default physical mapping of LEDs is defined as follows:

```
//LED1  
#define LED1_BV      BV(0)  
#define LED1_SBIT    P1_0  
#define LED1_DDR     P1DIR
```

```

#define LED1_POLARITY ACTIVE_HIGH

#ifndef defined (HAL_BOARD_CC2530EB_REV17)
//LED2

#define LED2_BV      BV(1)
#define LED2_SBIT    P1_1
#define LED2_DDR     P1DIR
#define LED2_POLARITY ACTIVE_HIGH

//LED3

#define LED3_BV      BV(4)
#define LED3_SBIT    P1_4
#define LED3_DDR     P1DIR
#define LED3_POLARITY ACTIVE_HIGH

#endif ENABLE_LED4_DISABLE_S1

//LED4

#define LED4_BV      BV(1)
#define LED4_SBIT    P0_1
#define LED4_DDR     P0DIR
#define LED4_POLARITY ACTIVE_HIGH
#define LED4_SET_DIR() do {LED4_DDR |= LED4_BV;} while (0)

#else
#define LED4_SET_DIR()
#endif

#endif

```

Here we take LED1 as an example to explain the meaning of the physical mapping definition.

(1) BV(N) means that the value 1 is shifted N bits to the left, which is equivalent to $(1 \ll N)$. Therefore, the value of BV(0) is $1 \ll 0$, which is equal to 1.

\ll is the left shift operator. $1 \ll 0$ means that the value 1 is shifted to the left by 0 bits. The value after the shift remains unchanged and is still equal to 1. For example, the binary number of the value 1 is 0000 0001. $1 \ll 2$ means that all the bits in 0000 0001 are shifted to the left by two bits, and 0 is added to the low bits and the high bits are discarded. Therefore, the calculation result is 0000 0100, and the corresponding decimal number is 3. If the reader does not understand this operator, it is necessary to supplement the relevant knowledge.

- (2) #define LED1_SBIT P1_0 indicates that LED1 is connected to P1_0 of CC2530.
- (3) #define LED1_DDR P1DIR indicates that the direction register corresponding to LED1 is P1DIR. P1DIR has been explained in the previous chapter and will not be explained here.
- (4) #define LED1_POLARITY ACTIVE_HIGH indicates that LED1 is driven at a high level, that is, when a high level is input to P1_0, LED1 will be lit.

These are the default configurations of the protocol stack, and developers need to modify them according to the actual hardware connection. For example, the matching ZigBee development board connects the LED to P0_4, so the pin of LED1 can be modified to P0_4. The modified code is as follows:

```

//LED1

#define LED1_BV      BV(4)
#define LED1_SBIT    P0_4
#define LED1_DDR     P0DIR

```

```
#define LED1_POLARITY ACTIVE_HIGH
```

The remaining LED2, LED3 and LED4 are not used and can be ignored for now.

6.4.4. LED Experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=32>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

The previous chapters explained the application of events. This lesson uses the event mechanism to achieve the effect of flashing LEDs.

Redefining the physical mapping of LEDs

In the hal_led.h file, redefine the physical mapping definition of LED1. The code is as follows:

```
//LED1
#define LED1_BV      BV(4)
#define LED1_SBIT    P0_4
#define LED1_DDR     P0DIR
#define LED1_POLARITY ACTIVE_HIGH
```

Because the matching ZigBee development board connects the LED to P0_4, the pin of LED1 is changed to P0_4 here.

Application of LED API

In the chapter explaining OSAL, a SAMPLEAPP_TEST_EVT event was customized and the corresponding event handling code was written. In this lesson, the event handling code is modified to flash the LED.

Back to the application layer, modify the processing logic of the custom event SAMPLEAPP_TEST_EVT in the event processing function zclSampleSw_event_loop() in the zcl_samplesw.c file. The code is as follows:

```
// 处理自定义的用户事件 : SAMPLEAPP_TEST_EVT
```

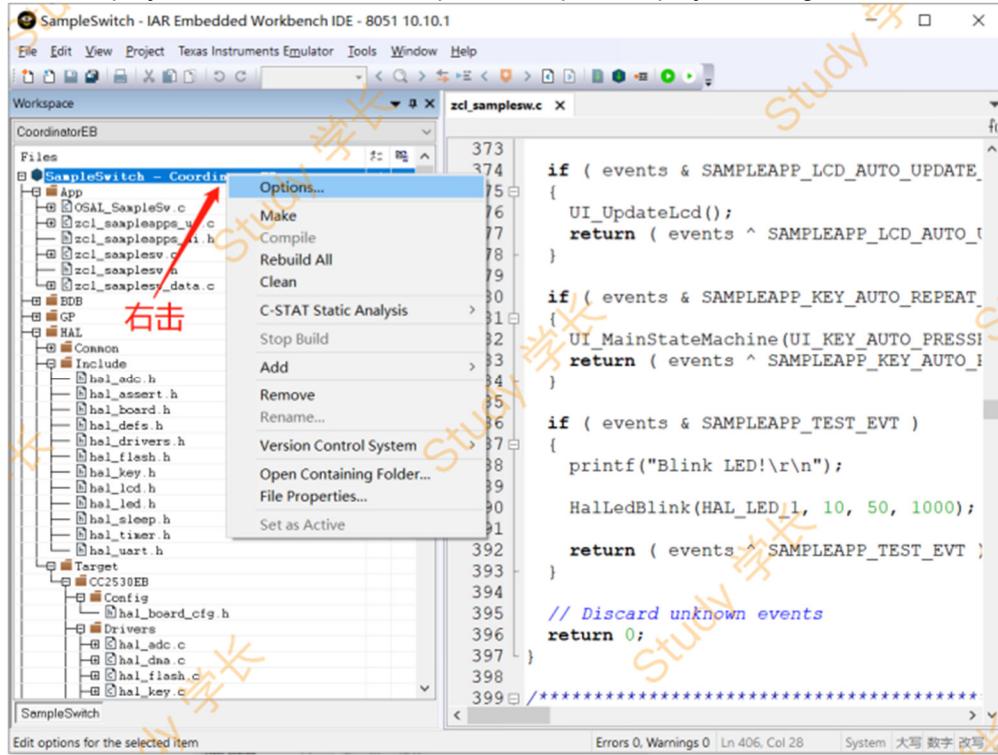
```
if ( events & SAMPLEAPP_TEST_EVT )
{
    printf("Blink LED!\r\n");

    HalLedBlink(
        HAL_LED_1 // 指定第1盏LED
        10 // 指定闪烁次数是10次
        50 // 指定50%的时间LED是处于开启状态
        1000); // 指定1次闪烁的时间周期是1000ms

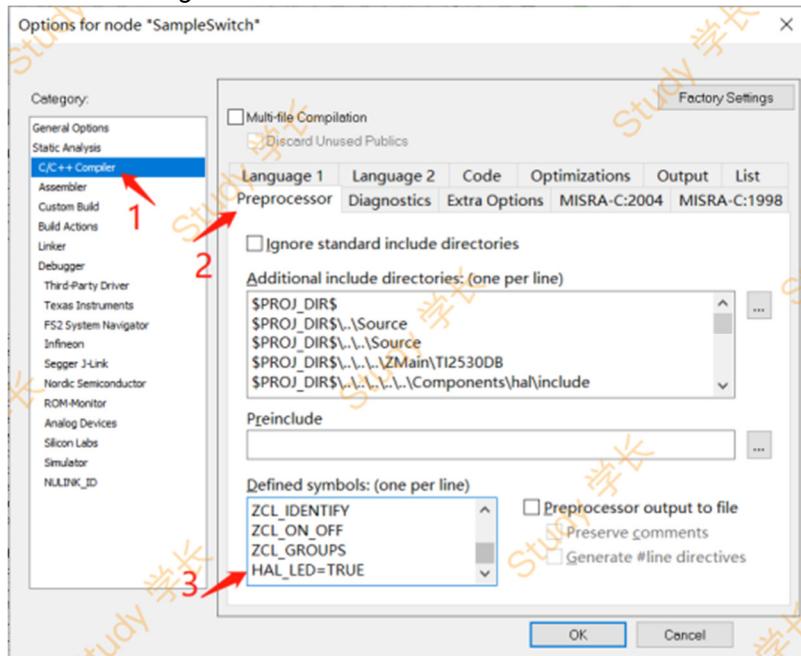
    // 消除已经处理的事件 · 然后返回未处理的事件
    return ( events ^ SAMPLEAPP_TEST_EVT );
}
```

Enable LED macro definition

As mentioned in the previous chapter, you need to define the corresponding macros for the peripheral functions you need to use. Now, if you need to use the LED function, you just need to define the macros corresponding to the LED function. Right-click the project name, then select Options to open the project configuration, as shown in the figure.



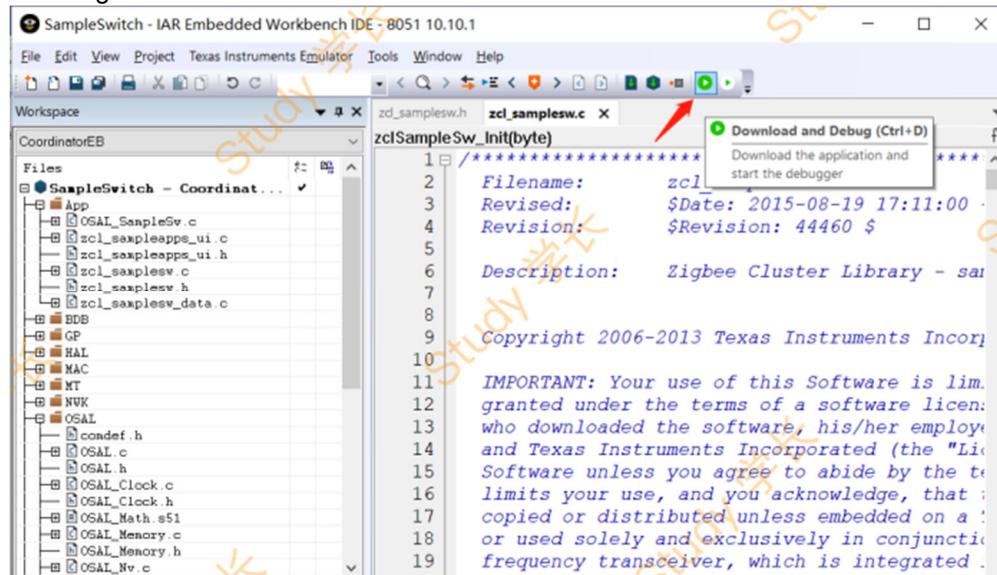
In the project settings, select C/C++ Compiler→Preprocessor, add HAL_LED=TRUE in Defined symbols, and then click OK, as shown in the figure.



Code Testing

(1) Use the provided emulator to connect the ZigBee development board to the computer.

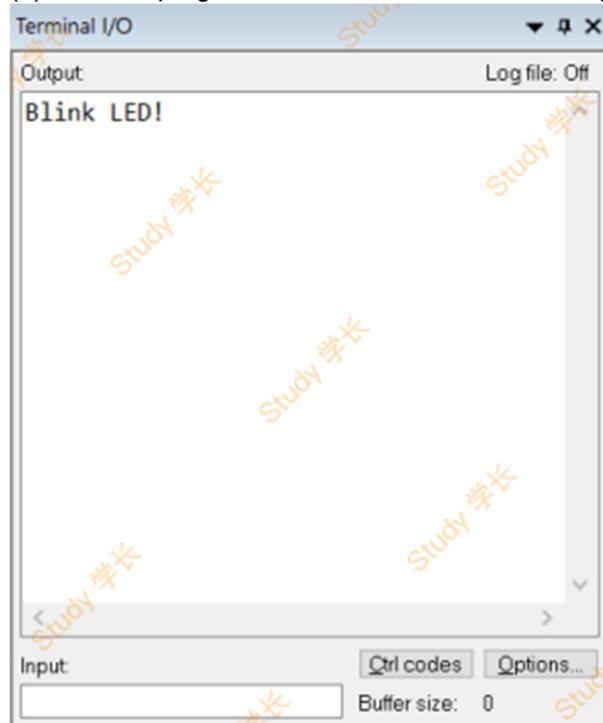
(2) Click the Download and Debug button to compile, link, and download the program and enter the simulation mode, as shown in the figure.



(3) After entering the simulation mode, click the Go button to run the program, as shown in the figure.



(4) After the program runs, "Blink LED!" will be output in the Terminal I/O window, as shown in the figure.



(5) At the same time, it can be observed that the LED connected to P0_4 on the ZigBee development board flashes 10 times.

6.5. Chapter 5: Hardware Adaptation Layer Application - Buttons

6.5.1. Keystroke Experiment

6.5.2. Detailed Explanation of HAL Key Framework (Optional)

6.5.1. Keystroke Experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

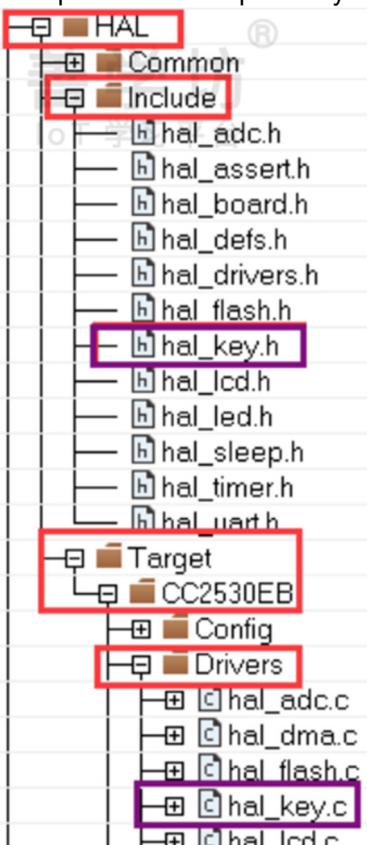
Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

This lesson will explain how to use the HAL button driver API directly in an experimental way. The experimental content of this lesson is to flash the LED after pressing the button.

Button API Documentation

HAL provides a complete key driver API, which is defined in hal_key.h and hal_key.c, as shown in the following figure.



Key definition and mapping

(1) Open the header file hal_led.h and find the following code:

```
hal_key.h x
68
69 #define HAL_KEY_SW_1 0x01 // Joystick up
70 #define HAL_KEY_SW_2 0x02 // Joystick right
71 #define HAL_KEY_SW_5 0x04 // Joystick center
72 #define HAL_KEY_SW_4 0x08 // Joystick left
73 #define HAL_KEY_SW_3 0x10 // Joystick down
74
75 #define HAL_KEY_SW_6 0x20 // Button S1 if available
76 #define HAL_KEY_SW_7 0x40 // Button S2 if available
```

(2) SW_1 to SW_5 are direction buttons, which are rarely used, so they are not explained in detail. SW_6 and SW_7 are regular buttons.

(3) Similar to LED, the connection between the button and the CC2530 pin can be configured in the header file hal_board_cfg.h. The following is the default configuration of SW_6, the code is as follows:

```
/* S1 按键 */  
  
#define PUSH1_BV      BV(1)  
#define PUSH1_SBIT    P0_1  
  
//按键按下时的电平  
  
#if defined (HAL_BOARD_CC2530EB_REV17)  
    #define PUSH1_POLARITY ACTIVE_HIGH  
#elif defined (HAL_BOARD_CC2530EB_REV13)  
    #define PUSH1_POLARITY ACTIVE_LOW  
#else  
    #error Unknown Board Identifier  
#endif
```

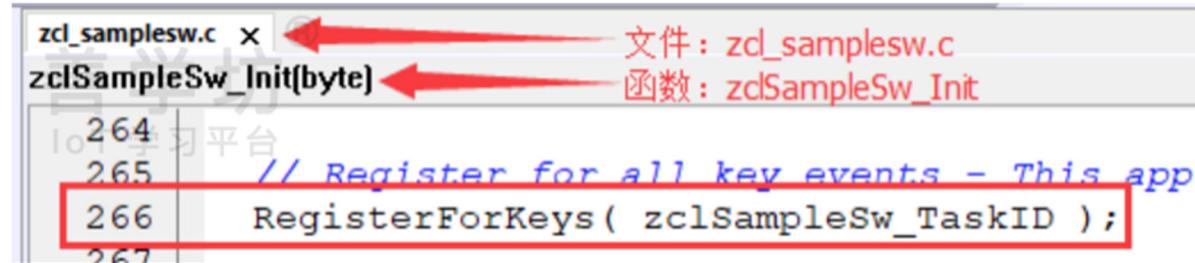
(4) Usually, this button can meet the needs of most devices. The pin connected to the button in the matching ZigBee development board is P0_1. However, the circuit of our button is low level when pressed and high level when not pressed, so we need to modify the configuration. The modified configuration code is as follows:

```
/* S1 按键 */  
  
#define PUSH1_BV      BV(1)  
#define PUSH1_SBIT    P0_1  
  
//表示低电平时驱动按键  
  
#define PUSH1_POLARITY ACTIVE_LOW
```

Tip: The button circuit has been explained in the previous chapter. Readers who need it can look it up.

Handling key events

(1) Since the buttons are scheduled by OSAL, they must be registered in OSAL before being used in Z-Stack. You can register in the application layer. The initialization function zclSampleSw_Init of the application layer has completed this registration by default, as shown in the figure:



zcl_samplesw.c x 文件: zcl_samplesw.c
zclSampleSw_Init(byte) 函数: zclSampleSw_Init
264
265 // Register for all key events - This app
266 RegisterForKeys(zclSampleSw_TaskID);
267

(2) After registration, if the key is pressed, an application layer system event KEY_CHANGE will be generated. Open the zclSampleSw_event_loop function in the zcl_samplesw.c file and find the KEY_CHANGE event processing function

zclSampleSw_HandleKeys, as shown in the figure.

zcl_samplesw.c x 文件: zcl_samplesw.c
zclSampleSw_event_loop[uint8, uint16] 函数: zclSampleSw_event_loop

```
325     if ( events & SYS_EVENT_MSG ) ← 系统事件
326     {
327         while ( (MSGpkt = (afIncomingMSGPacket.
328             {
329                 switch ( MSGpkt->hdr.event )
330                 {
331                     case ZCL_INCOMING_MSG:
332                         // Incoming ZCL Foundation comma.
333                         zclSampleSw_ProcessIncomingMsg (
334                             break; ← 按键事件
335
336                     case KEY_CHANGE:
337                         zclSampleSw_HandleKeys( ((keyCha: ← 事件处理
338                             break;
339
340             }
341         }
342     }
```

(3) The code definition of the event processing function zclSampleSw_HandleKeys is as follows:

```
*****@fn zclSampleSw_HandleKeys
*
* @brief Handles all key events for this device.
*
* @param shift - true if in shift/alt.
* @param keys - bit field for key events. Valid entries:
*               HAL_KEY_SW_5
*               HAL_KEY_SW_4
*               HAL_KEY_SW_2
*               HAL_KEY_SW_1
*
* @return none
*/
static void zclSampleSw_HandleKeys( byte shift, byte keys )
{
    UI_MainStateMachine(keys); ← 在显示屏上进行显示！！
```

(4) The default working content of the zclSampleSw_HandleKeys function is to display the key pressed information on the LCD display. Developers can add key event processing code here, such as the flashing LED explained in the previous chapter. The reference code is as follows:

```
static void zclSampleSw_HandleKeys( byte shift, byte keys )
{
    UI_MainStateMachine(keys); ← 按键信息中有HAL_KEY_SW_6
    if(keys & HAL_KEY_SW_6) ← 闪烁RGB
    {
        HalLedBlink(HAL_LED_1, 10, 50, 400); // Red
        HalLedBlink(HAL_LED_2, 10, 50, 1000); // Green
        HalLedBlink(HAL_LED_3, 10, 50, 2000); // Blue
    }
}
```

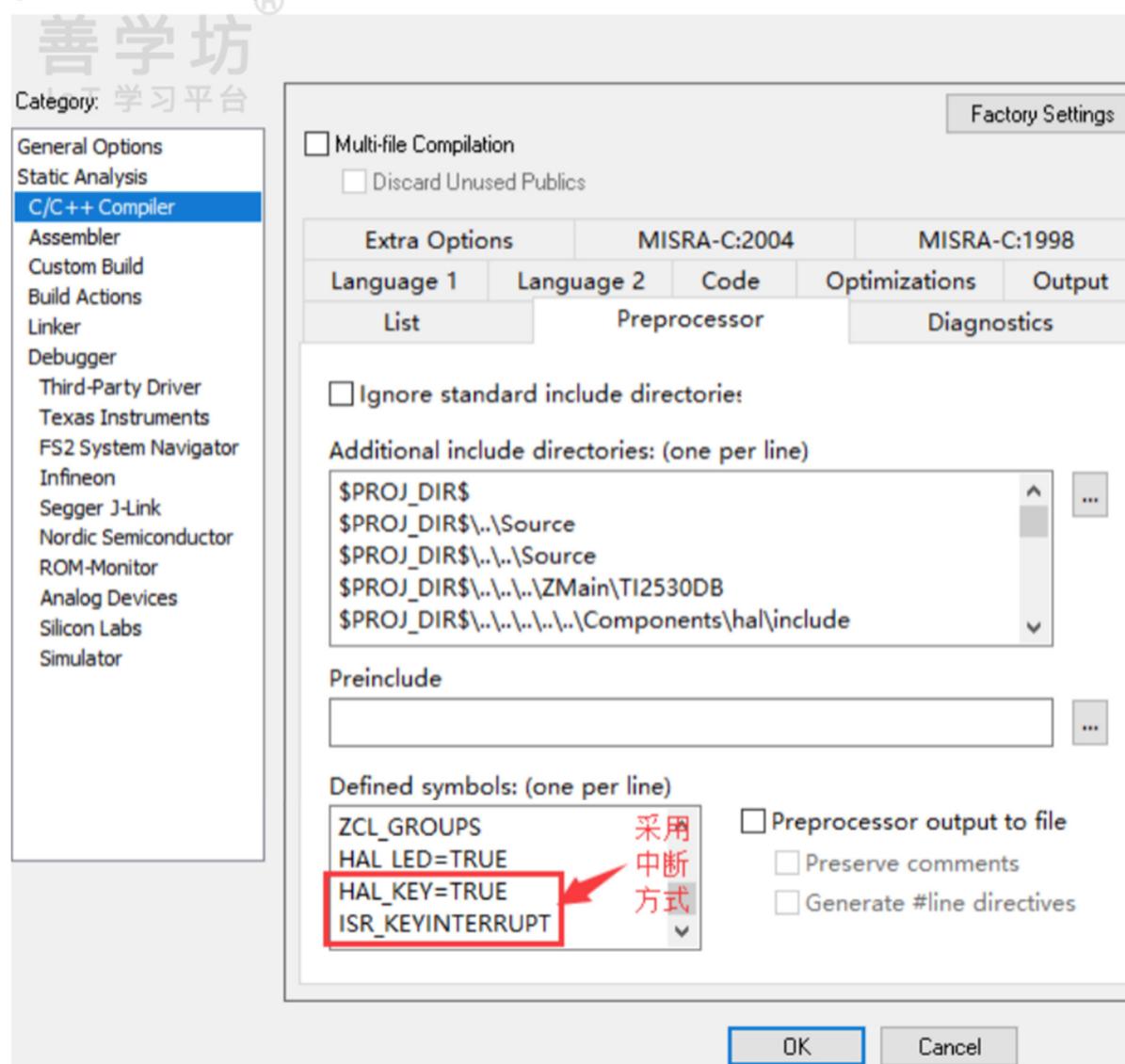
Enable key macros

(1) Similar to LED, before using the key, you need to enable the macro definition HAL_KEY corresponding to the key. Enter the following code in the Defined symbols in the figure below.

HAL_KEY=TRUE

(2) After adding, as shown in the figure.

Options for node "SampleSwitch"



(3) The macro ISR_KEYINTERRUPT is defined because the system uses interrupts to detect key presses. For now, you only need to have a general understanding of it.

Debugging Simulation

After compiling the project, burning the program to the development board and running it, you can see the LED flashing when the button is pressed.

6.5.2. Detailed Explanation of HAL Key Framework (Optional)

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

- Generally, self-study is the main method.
- You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
- Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Overview

The last lesson explained the use of the HAL button API. This lesson will analyze the principles in depth. **If you do not need to learn the principles for the time being, you can skip this lesson.**

hal_key.c file

The main code of the HAL key framework is in the hal_key.c file. Open the file to find the following important functions:

```
HalKeyInit  
HalKeyConfig  
HalKeyPoll  
HAL_ISR_FUNCTION( halKeyPort0lsr, P0INT_VECTOR )
```

Initialization function HalKeyInit

The code for the initialization function HalKeyInit is as follows:

```
1. void HalKeyInit( void )  
2.{  
3. /* Initialize previous key to 0 */  
4. halKeySavedKeys = 0;  
5.  
6. HAL_KEY_SW_6_SEL &= ~(HAL_KEY_SW_6_BIT); // I/O为普通GPIO  
7.  
8.#if ! defined ENABLE_LED4_DISABLE_S1  
9. HAL_KEY_SW_6_DIR &= ~(HAL_KEY_SW_6_BIT); // I/O配置为输入  
10.#endif  
11.  
12. HAL_KEY_JOY_MOVE_SEL &= ~(HAL_KEY_JOY_MOVE_BIT);  
13. HAL_KEY_JOY_MOVE_DIR &= ~(HAL_KEY_JOY_MOVE_BIT);  
14.  
15. /* Initialize callback function */  
16. pHalKeyProcessFunction = NULL;  
17.  
18. /* Start with key is not configured */  
19. HalKeyConfigured = FALSE;  
20.}
```

The main work of the initialization function is to configure the GPIO as an input function. The definitions of HAL_KEY_SW_6_SEL and other definitions are actually registers related to the key GPIO configuration. These definitions can be found in hal_key.c. If we need to add other keys, we can also follow this method.

The definitions of configuring HAL_KEY_6 related registers are as follows:

hal_key.c x

HalKeyInit()

```
110
111 /* SW_6 is at P0.1 */
112 #define HAL_KEY_SW_6_PORT      P0
113 #define HAL_KEY_SW_6_BIT        BV(1)
114 #define HAL_KEY_SW_6_SEL        P0SEL
115 #define HAL_KEY_SW_6_DIR        P0DIR
116
117 /* edge interrupt */
118 #define HAL_KEY_SW_6_EDGEBIT   BV(0)
119 #define HAL_KEY_SW_6_EDGE       HAL_KEY_FAL
120
121
122 /* SW_6 interrupts */
123 #define HAL_KEY_SW_6_IEN        IEN1 /* CE
124 #define HAL_KEY_SW_6_IENBIT    BV(5) /* Ma
125 #define HAL_KEY_SW_6_ICTL      POIEN /* Po
126 #define HAL_KEY_SW_6_ICTLBIT   BV(1) /* PC
127 #define HAL_KEY_SW_6_PXIFG    POIFG /* In
128
```

There is a place in the initialization function HalKeyInit that needs to be explained. There is a piece of code in this function (using the quick configuration HAL_KEY_SW_6 key pin as input function):

```
1.#if ! defined ENABLE_LED4_DISABLE_S1
2. HAL_KEY_SW_6_DIR &= ~(HAL_KEY_SW_6_BIT); // I/O配置为输入
3.#endif
```

ZStack's original intention is that the S1 button pin P0_1 can be used to drive the LED, that is, it is reused with the button. Whether it is used as an LED depends on whether the macro ENABLE_LED4_DISABLE_S1 is defined. However, this macro is defined in the part of the code that is not open source in the protocol stack, which is also a bug in ZStack. Therefore, we need to modify this code and remove the pre-compilation (comment it out), otherwise the button pin will not be configured as input:

```
1.// #if ! defined ENABLE_LED4_DISABLE_S1
2. HAL_KEY_SW_6_DIR &= ~(HAL_KEY_SW_6_BIT); // I/O配置为输入
3.// #endif
```

HalKeyConfig

This function is mainly used to configure interrupt-related content:

```
1.void HalKeyConfig (bool interruptEnable, halKeyCBack_t cback)
2.{}
3. Hal_KeyIntEnable = interruptEnable; // 标志：是否使能中断
4.
5. pHalKeyProcessFunction = cback; // 回调函数，按键按下时调用
6.
7. /* Determine if interrupt is enable or not */
```

```

8. if (Hal_KeyIntEnable) // 如果使能了中断，必须配置中断相关内容
9. {
10. PICTL &= ~(HAL_KEY_SW_6_EDGEBIT); /* Clear the edge bit */
11. /* For falling edge, the bit must be set. */
12. #if (HAL_KEY_SW_6_EDGE == HAL_KEY_FALLING_EDGE)
13. PICTL |= HAL_KEY_SW_6_EDGEBIT;
14. #endif
15.
16. /* Interrupt configuration:
17. * - Enable interrupt generation at the port
18. * - Enable CPU interrupt
19. * - Clear any pending interrupt
20. */
21. HAL_KEY_SW_6_ICTL |= HAL_KEY_SW_6_ICTLBIT;
22. HAL_KEY_SW_6_IEN |= HAL_KEY_SW_6_IENBIT;
23. HAL_KEY_SW_6_PXIFG = ~(HAL_KEY_SW_6_BIT);
24.
25. ....... // 屏蔽无关紧要的代码
26.
27. if (HalKeyConfigured == TRUE)
28. {
29. osal_stop_timerEx(Hal_TaskID, HAL_KEY_EVENT); // 不需要轮询
30. }
31. }
32. else // 如果没有使用中断，必须把中断相关内容关闭掉
33. {
34. HAL_KEY_SW_6_ICTL &= ~(HAL_KEY_SW_6_ICTLBIT);
35. HAL_KEY_SW_6_IEN &= ~(HAL_KEY_SW_6_IENBIT);
36. osal_set_event(Hal_TaskID, HAL_KEY_EVENT); // 按键轮询事件
37. }
38.
39. HalKeyConfigured = TRUE;
40.}

```

HalKeyPoll

```

1.void HalKeyPoll (void)
2.{ 
3. uint8 keys = 0;
4. if ((HAL_KEY_JOY_MOVE_PORT & HAL_KEY_JOY_MOVE_BIT))
5. {
6. keys = halGetJoyKeyInput();
7. }
8. /*
9. *

```

```

10. */
11. if (!Hal_KeyIntEnable)
12. {
13.     if (keys == halKeySavedKeys)
14.     {
15.         return;
16.     }
17.     halKeySavedKeys = keys;
18. }
19. else
20. {
21.     /* Key interrupt handled here */
22. }
23. if (HAL_PUSH_BUTTON1()) // 检测HAL_KEY_SW_6是否按下, 添加其他
24. {
    // 按键时, 同样也需要在这里做检测!
25.     keys |= HAL_KEY_SW_6;
26. }
27.
28. if (pHalKeyProcessFunction
29.#ifdef HAL_LEGACY_KEYS
30.     && keys
31.#endif
32. )
33. {
34.     (pHalKeyProcessFunction) (keys, HAL_KEY_STATE_NORMAL);
35. }
36.}

```

This function is used to check whether a key is pressed. If it is in interrupt mode, debounce when an interrupt is detected and then enter this function; if it is not in interrupt mode, periodic polling is used to check whether the key is pressed.

Function HAL_ISR_FUNCTION(halKeyPort0Isr, P0INT_VECTOR):

This is an interrupt processing function. The interrupt vector is P0INT_VECTOR, which is the interrupt of port P0. The key is connected to P0_1. Pressing the key to generate an interrupt will lead to this function:

```

1.HAL_ISR_FUNCTION( halKeyPort0Isr, P0INT_VECTOR )
2.{ 
3.     HAL_ENTER_ISR();
4.     if (HAL_KEY_SW_6_PXIFG & HAL_KEY_SW_6_BIT) // P0_1中断
5.     {
6.         halProcessKeyInterrupt(); // 调用这个函数来处理中断
7.         // 最终会调用 HalKeyPoll
8.     }
9. /*
10.     Clear the CPU interrupt flag for Port_0
11.     PxIFG has to be cleared before PxIF
12. */

```

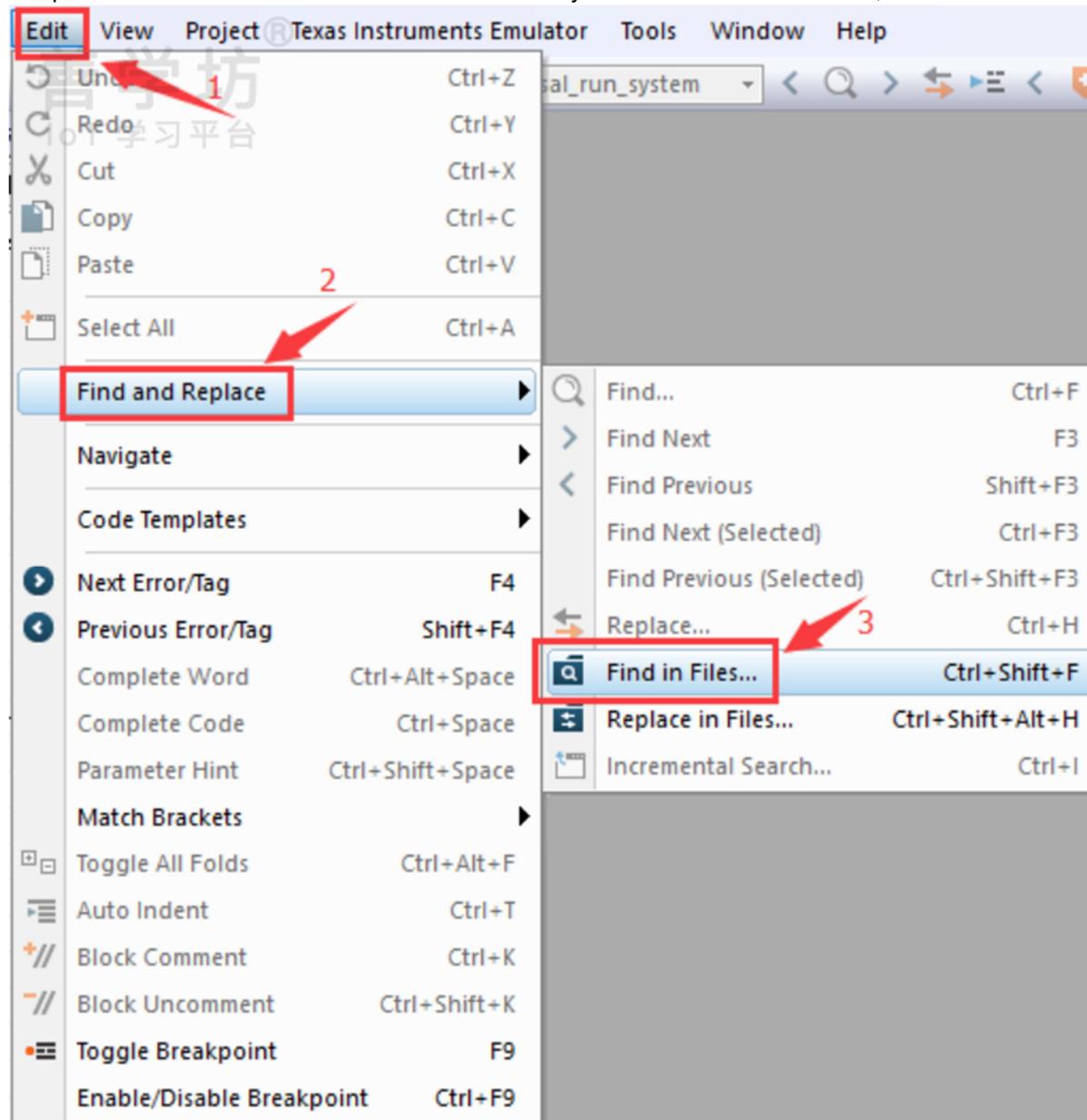
```
13. HAL_KEY_SW_6_PXIFG = 0; // 清除中断标志  
14. HAL_KEY_CPU_PORT_0_IF = 0;  
15. CLEAR_SLEEP_MODE();  
16. HAL_EXIT_ISR();  
17.}
```

The final interrupt will be processed in the function halProcessKeyInterrupt():

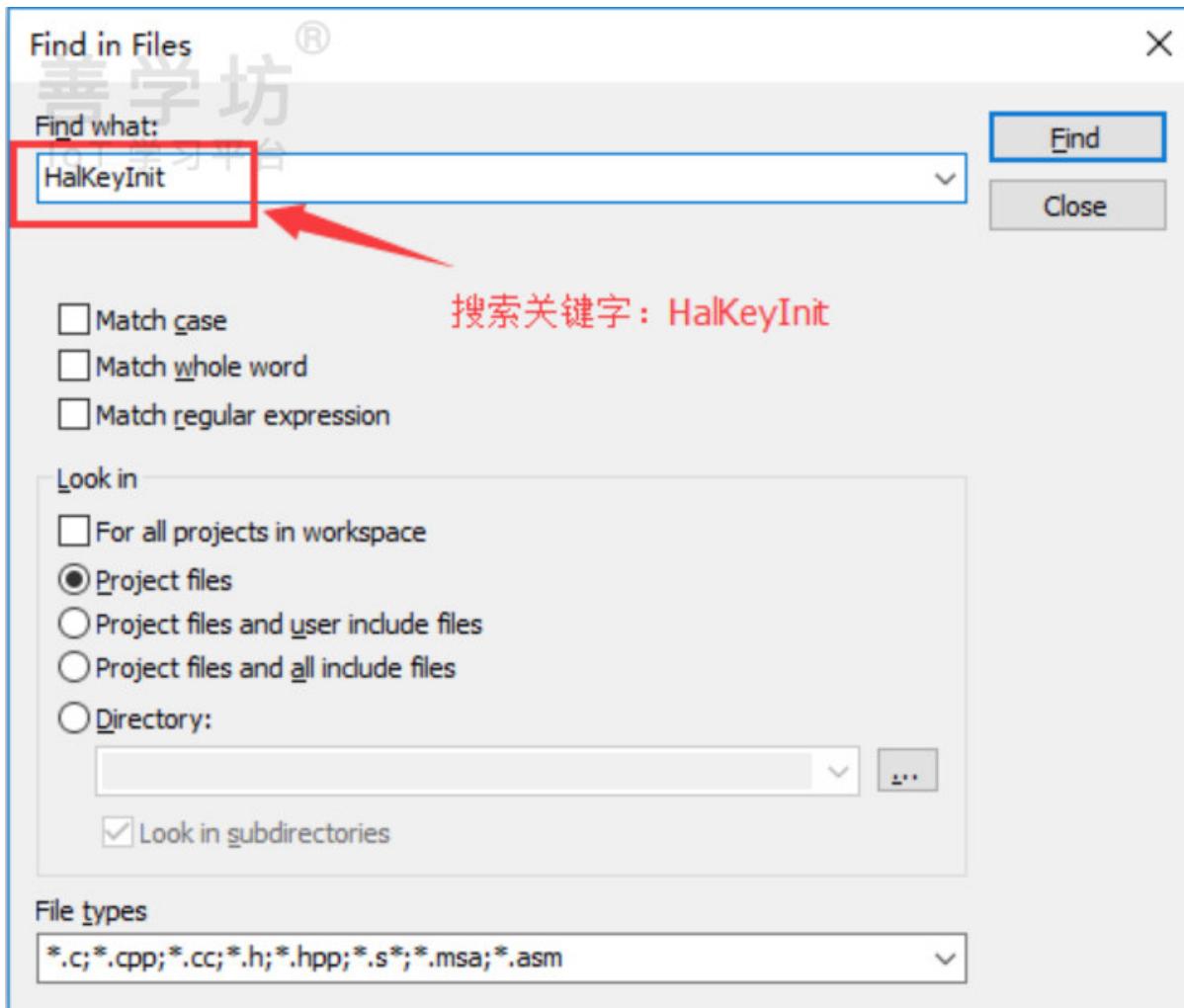
```
1. void halProcessKeyInterrupt (void)  
2. {  
3.     bool valid=FALSE;  
4.     // 清除HAL_KEY_SW_6引脚对应的中断标志位  
5.     if (HAL_KEY_SW_6_PXIFG & HAL_KEY_SW_6_BIT)  
6.     {  
7.         HAL_KEY_SW_6_PXIFG = ~(HAL_KEY_SW_6_BIT);  
8.         valid = TRUE;  
9.     }  
10.  
11.     if (valid)  
12.     { // 启动一个事件 · 25ms后到期 (去抖) , 事件会在hal_drivers.c  
13.         // 中被处理 ( Hal_ProcessEvent) , 最终会调用HalKeyPoll  
osal_start_timerEx (Hal_TaskID, HAL_KEY_EVENT,  
HAL_KEY_DEBOUNCE_VALUE);  
14.     }  
15. }
```

Question: Where will the function explained above be called in the end?

We provide a method that can be used to view all keyword-related content in IAR, and then find where the function is called:



This search function is a global search, which means that IAR will search all files in the project, match the keywords we want to search, and then display the results; for example, if we want to search for all content related to the keyword HalKeyInit, we can enter this keyword in the pop-up dialog box and click "Find". Then we can double-click the corresponding result to view related content according to the search results:



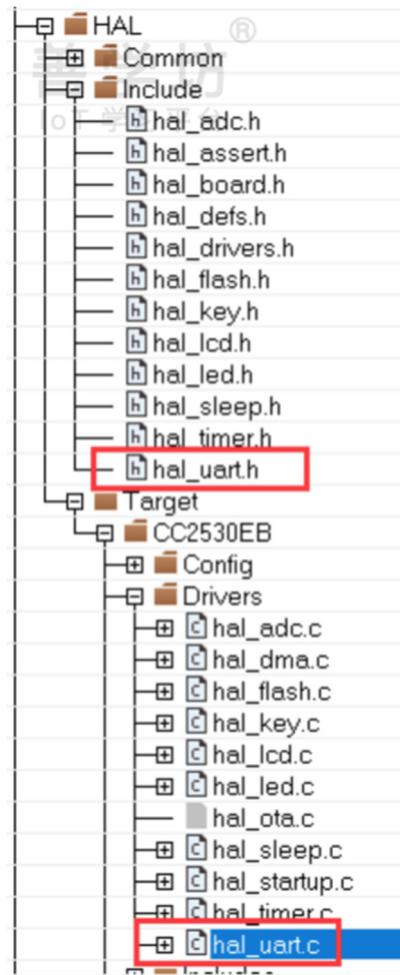
6.6. Chapter 6: Hardware Adaptation Layer Application - Serial Port

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

HAL provides a complete serial port driver API, which is defined in hal_uart.h and hal_uart.c, as shown in the following figure.



Serial port configuration

Before using the serial port, you need to configure it first. For this purpose, the author has written a serial port configuration function `zclSampleSw_Init`, which can be called in the application layer initialization function `zclSampleSw_Init` to complete the serial port configuration, as shown in the figure below.

文件: `zcl_samplesw.c`

函数: `zclSampleSw_Init`

```

zcl_samplesw.c × ←
zclSampleSw_Init(byte) ←
305
306 // Init Uart
307 zclSampleSw_InitUart(); ↗ 用户代码:
308 }                                配置串口

```

Open the supporting project code and find the definition of the `zclSampleSw_InitUart` function at the end of the `zcl_samplesw.c` file. The code is as follows:

```

1. static void zclSampleSw_InitUart(void)
2. {
3.     halUARTCfg_t uartConfig;
4.     /* UART Configuration */
5.     uartConfig.configured = TRUE; // 允许配置
6.     uartConfig.baudRate = HAL_UART_BR_115200; // 波特率
7.     uartConfig.flowControl = FALSE; // 关闭硬件流控
8.     uartConfig.flowControlThreshold = 0; // 和流控相关
9.     uartConfig.rx.maxBufSize = ZCLSAMPLESW_UART_BUF_LEN; // 接收缓冲区大小

```

```

10. uartConfig.tx.maxBufSize = 0;//发送缓冲区大小
11. uartConfig.idleTimeout = 6;//默认超时时间
12. uartConfig.intEnable = TRUE;//使能中断
13. uartConfig.callBackFunc = zclSampleSw_UartCB;//设置回调函数
14. /* Start UART */
15. HalUARTOpen(HAL_UART_PORT_0, &uartConfig);//根据配置打开串口
16.

```

The following is a detailed explanation of the above code.

- uartConfig.rx.maxBufSize = ZCLSAMPLESW_UART_BUF_LEN is used to set the size of the serial port 0 receive buffer. The macro ZCLSAMPLESW_UART_BUF_LEN is defined by the author in the zcl_samplesw.c file and is defined as follows:

```
#define ZCLSAMPLESW_UART_BUF_LEN 128
```

- uartConfig.tx.maxBufSize = 0 is used to configure the size of the send buffer. We do not need a send buffer, so we set the length to 0.
- HalUARTOpen(HAL_UART_PORT_0, &uartConfig) is used to open serial port 0 according to the configuration. HalUARTOpen is an API provided by HAL.
- The function of uartConfig.callBackFunc = zclSampleSw_UartCB is to set a callback function.

Receive serial port data

When the serial port receives data, the callback function zclSampleSw_UartCB will be called. Developers can receive and process serial port data in this function. The definition of the callback function zclSampleSw_UartCB can be found at the end of the zcl_samplesw.c file. The code is as follows:

```
/*
 * @fn    zclSampleSw_UartCB
 *
 * @brief Uart Callback
 */
static void zclSampleSw_UartCB(uint8 port, uint8 event)
{
    //获取当前串口接收缓冲区有多少字节的数据
    uint8 rxLen = Hal_UART_RxBufLen(HAL_UART_PORT_0);

    if(rxLen != 0)//如果字节数数量不等于0
    {
        //从串口缓冲区中读取数据
        HalUARTRead(HAL_UART_PORT_0, zclSampleSw_UartBuf, rxLen);

        //通过串口发送数据 · 例如把数据发送到串口助手
        HalUARTWrite(HAL_UART_PORT_0, zclSampleSw_UartBuf, rxLen);
    }
}
```

The above code calls the API specifically used to operate the serial port, as follows:

Hal_UART_RxBufLen : 查看当前串口接收缓冲区有多少字节的数据。

HalUARTRead : 从串口缓冲区中读取数据。

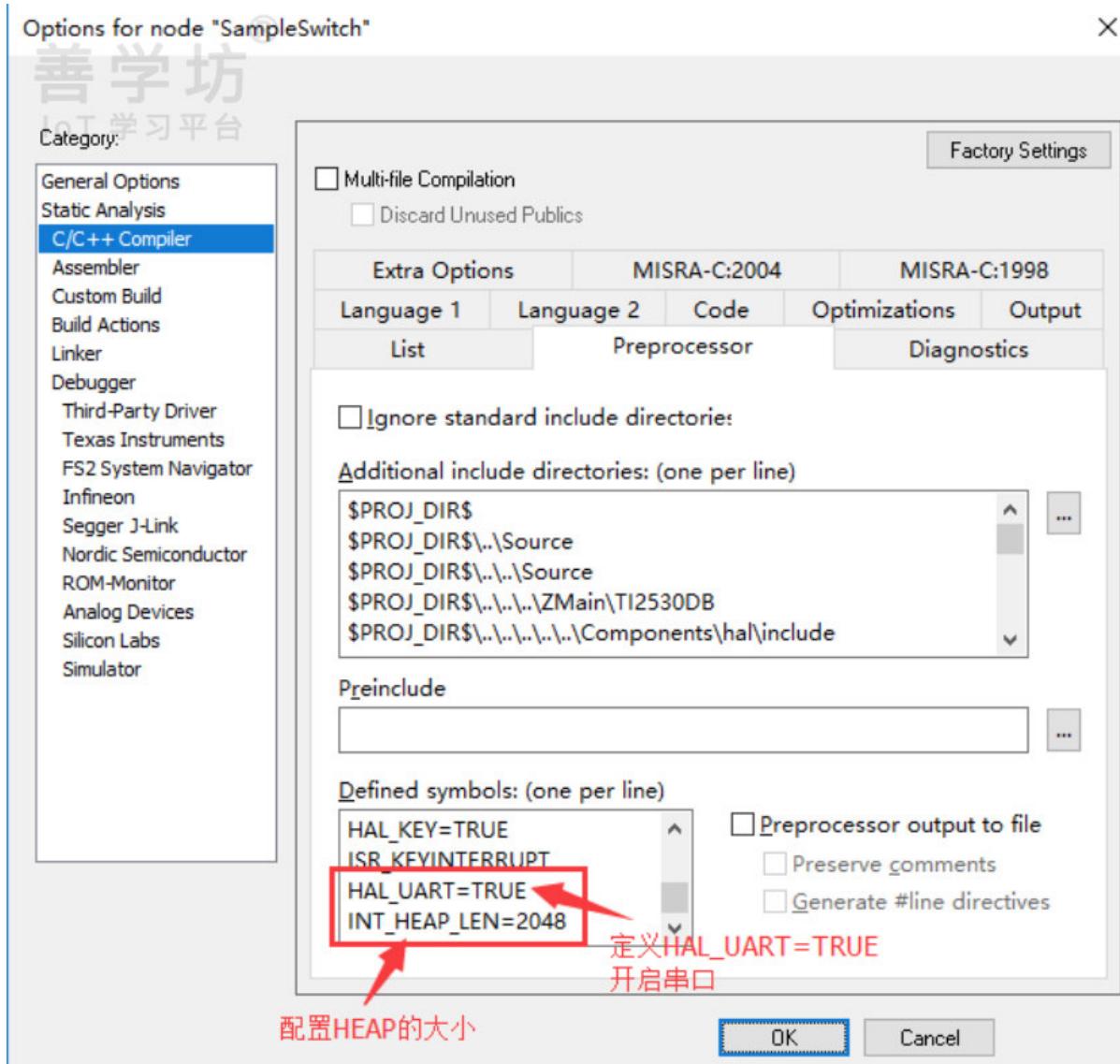
HalUARTWrite : 通过串口发送数据，比如发送到串口助手。

Enable the macro definition of the serial port

When using the serial port function, you need to enable the macro definition HAL_UART corresponding to the serial port. Enter the following code in the Defined symbols in the figure below.

HAL_UART=TRUE

INT_HEAP_LEN=2048



INT_HEAP_LEN

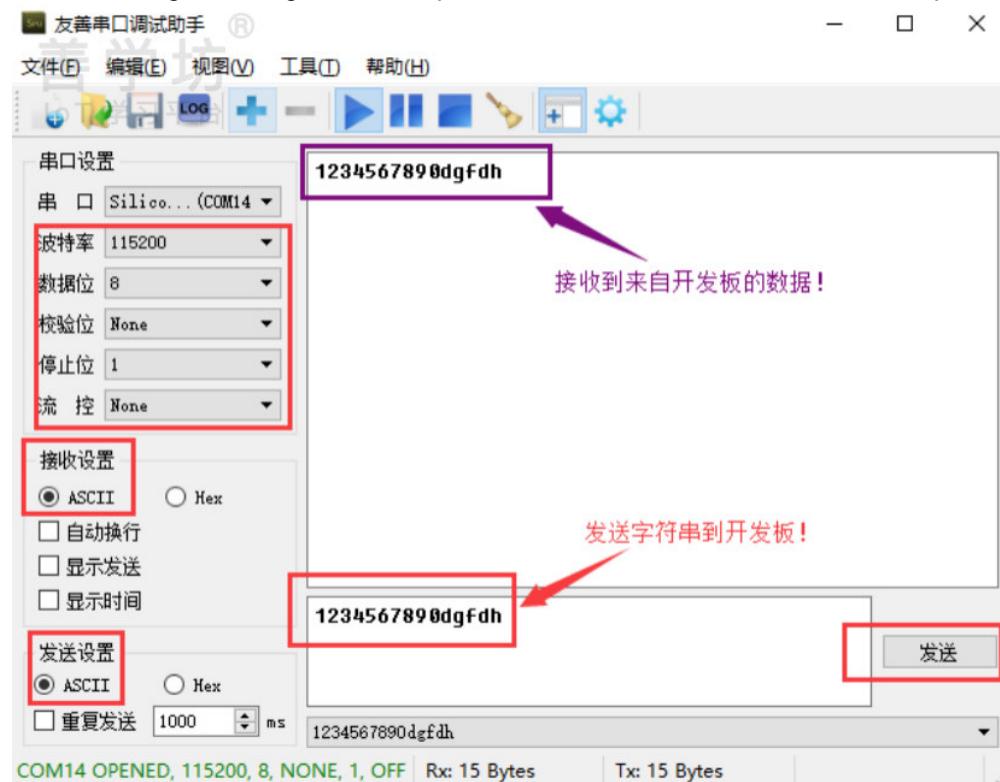
HEAP means heap, which is a piece of memory space. When the program dynamically applies for a small piece of memory space, this small piece of memory space comes from the heap. The default heap size of the project is 3071 bytes. After defining the macro INT_HEAP_LEN=2048, the heap size is reduced to 2048 bytes. The reason for reducing the heap size is that the serial communication function will use more stack memory space, and the stack memory space may not be enough.

Debugging Simulation

You can debug the simulation by following the steps below:

- After compiling the project, use the simulation downloader to burn the program into the development board
- Disconnect the simulator and connect the development board to the computer using a Micro USB cable.
- Open the serial port assistant and send the string to the development board through the serial port assistant

After receiving the string, the development board will send it back to the serial port assistant, as shown in the figure below.



6.7. Chapter 7: Hardware Adaptation Layer Application - Display Screen

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

In the previous C51 single-chip development chapter, it has been explained that if a display is used, the developer only needs to install the relevant driver into HAL and then use the display through HAL.

Display driver transplantation

In the project code accompanying this lesson, there is a "HAL Update" folder, as shown in the figure below.



The author has ported the driver programs of OLED12864 and TFT12864 color screens used in the previous chapters to HAL. The ported codes are in the hal folder, as shown in the figure below.



The usage is:

1. Delete the hal folder in the protocol stack, which is in Z-Stack 3.0.1\Components
2. Copy the hal folder in the HAL update folder to Z-Stack 3.0.1\Components to replace the hal folder in the protocol stack, as shown in the figure below.



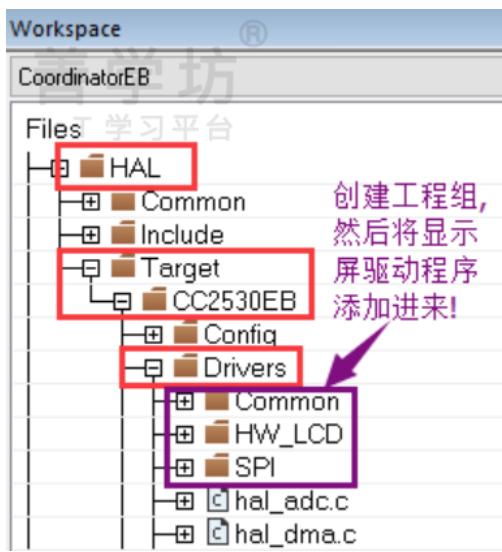
3. After the replacement is successful, the display driver can be found in the directory Z-Stack 3.0.1\Components\hal\target\CC2530EB, as shown in the figure.



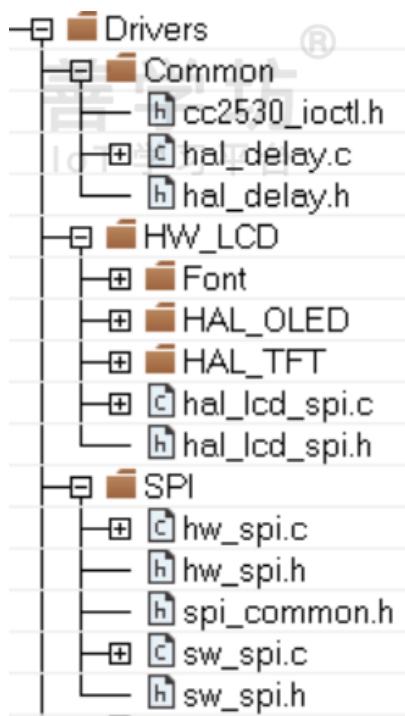
These drivers are the same display drivers used in the previous chapters.

4. Add the driver to the project

- A. Create three project groups under HAL's Drivers, namely Common, HW_LCD and SPI, as shown in the figure.



B. Add the relevant drivers as shown in the figure.



5. Add a route to the project, as described below.

Driver Description

The SPI in the figure above is the general SPI driver explained in the previous chapter. Common stores the delay function and the general IO configuration program, and HW_LCD stores the font library, LCD SPI adapter program and display driver. When I transplanted it, I made some variable type compatibility, mainly uint8 and uint8_t, uint16 and uint_16, etc.

The details of HW_LCD are shown in the figure below.



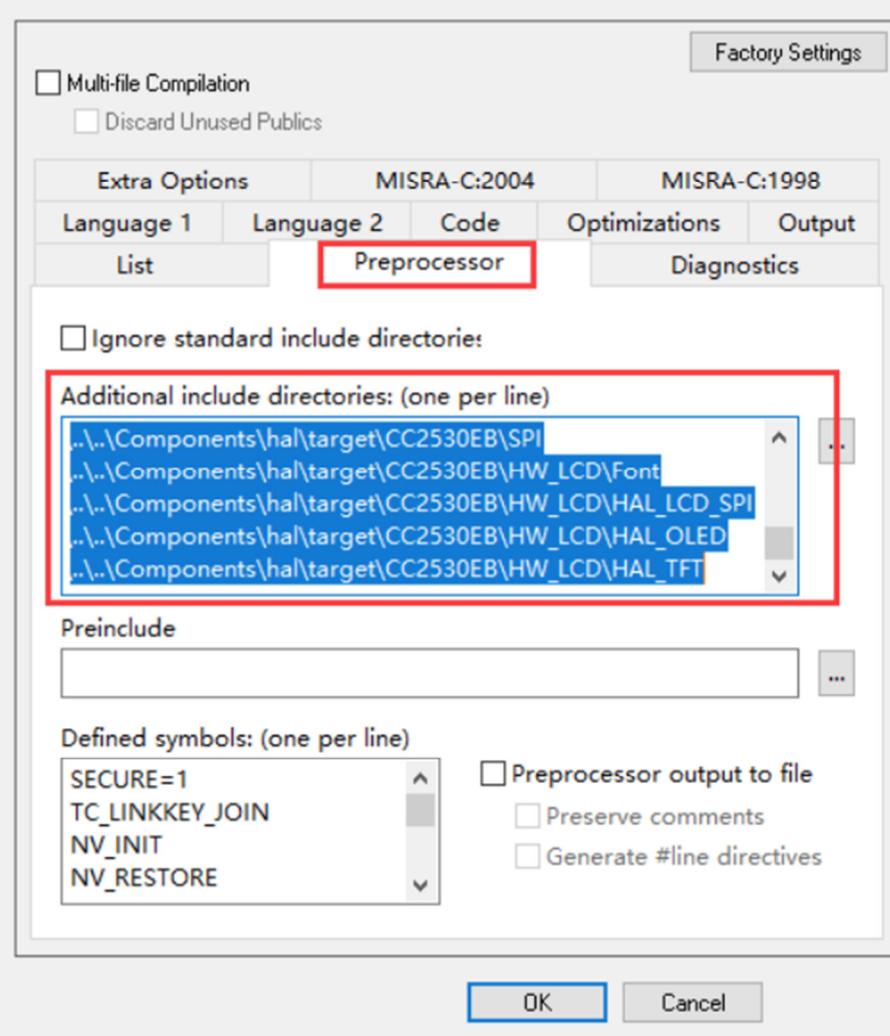
Adding routes to the project

Finally, you need to add the driver path to the project. Enter the following code in the Preprocessor in the figure below:

```
$PROJ_DIR$..\..\..\..\Components\hal\target\CC2530EB\Common  
$PROJ_DIR$..\..\..\..\Components\hal\target\CC2530EB\SPI  
$PROJ_DIR$..\..\..\..\Components\hal\target\CC2530EB\HW_LCD\Font  
$PROJ_DIR$..\..\..\..\Components\hal\target\CC2530EB\HW_LCD\HAL_LCD_SPI  
$PROJ_DIR$..\..\..\..\Components\hal\target\CC2530EB\HW_LCD\HAL_OLED  
$PROJ_DIR$..\..\..\..\Components\hal\target\CC2530EB\HW_LCD\HAL_TFT
```

Category:

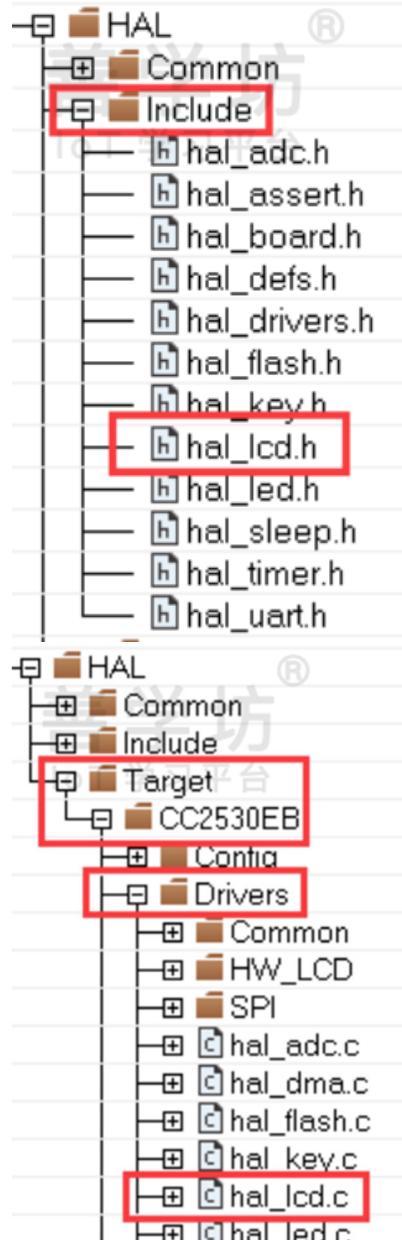
- General Options
- Static Analysis
- C/C++ Compiler**
- Assembler
- Custom Build
- Build Actions
- Linker
- Debugger
- Third-Party Driver
- Texas Instruments
- FS2 System Navigator
- Infineon
- Segger J-Link
- Nordic Semiconductor
- ROM-Monitor
- Analog Devices
- Silicon Labs
- Simulator



At this point, the display driver has been ported to HAL, and then you can use the LCD API that HAL prepares for developers.

HAL LCD API

HAL provides a complete display driver API, which is defined in hal_lcd.h and hal_lcd.c, as shown in the figure.



Open the hal_lcd.h file and you can find the following display API:

```
/**  
 * 在指定的行中显示字符串  
 *  
 * @param str - 待显示的字符串  
 *  
 * @param option - 在哪一行显示数据  
 */
```

```
void HalLcdWriteString ( char *str, uint8 option);
```

```
/**  
 * 在指定行用指定进制显示数值  
 *  
 * @param value - 待显示的数值  
 * @param radix - 指定的进制
```

```
* @param option - 在哪一行显示数据
```

```
*
```

```
*/
```

```
void HalLcdWriteValue ( uint32 value, const uint8 radix, uint8 option);
```

If you need to display "Hello World!" on the first line of the screen, you can call HalLcdWriteString as follows:

```
HalLcdWriteString( "Hello World!", HAL_LCD_LINE_1 );
```

If you need the first line to display the value of a variable in decimal, you can call HalLcdWriteValue as follows:

```
1.uint8 val = 100;
```

```
2.HalLcdWriteValue ( val, 10, HAL_LCD_LINE_1 );
```

Enable LCD function

Before using the display function, you need to enable the macro, HAL_LCD.

1. If you are using the matching 0.96-inch OLED screen, enter the following code in the Defined symbols in the figure below.

```
HAL_LCD=TRUE
```

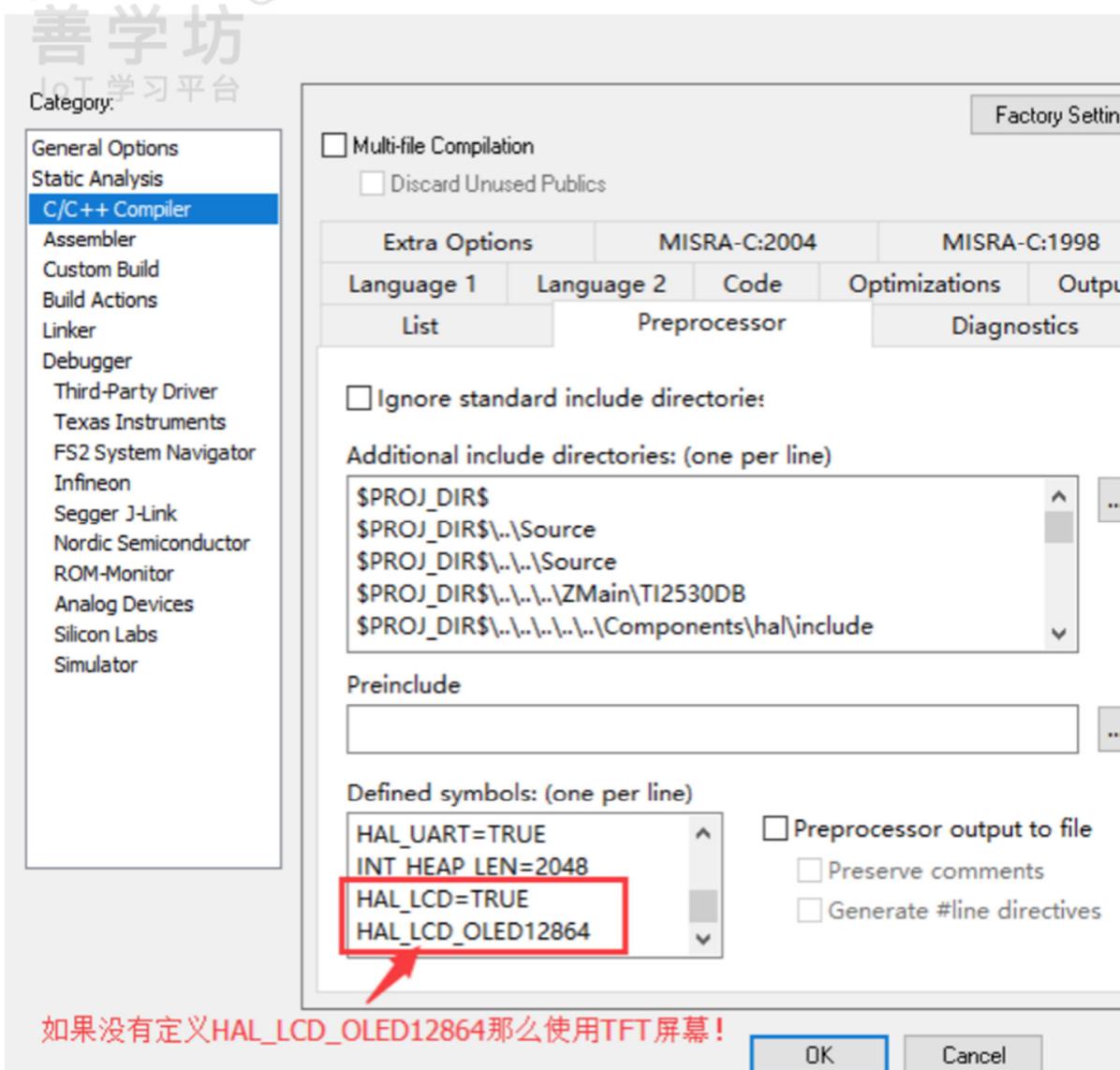
```
HAL_LCD_OLED12864
```

2. If you are using the matching 0.96-inch TFT screen, enter

```
HAL_LCD=TRUE
```

As shown in the figure below.

Options for node "SampleSwitch"



How to identify the screen type? Please refer to: [Display experiment](#)

Debugging Simulation

Compile the entire project, then burn the program to the development board and run it at full speed. You can see that the screen displays Z-Stack related debugging information.

6.8. Chapter 8: Hardware Adaptation Layer Application - ADC

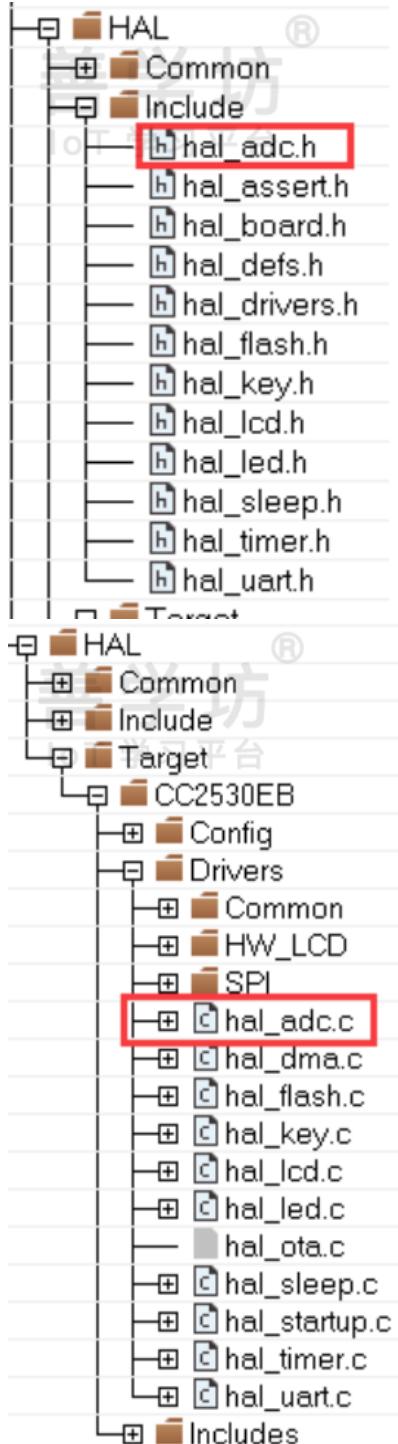
Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Introduction to HAL ADC API

HAL provides a complete ADC driver API, which is defined in hal_adc.h and hal_adc.c, as shown in the following figure.



The principle of ADC has been explained in detail in the previous chapters. This lesson mainly introduces the HAL ADC API. Open the hal_adc.h file and you can find the following API:

```
/*
 * Read value from a specified ADC Channel at the given resolution
 *
 * @param channel - 取值范围是0 ~ 7, 分别对应CC2530的P0_0 ~ P0_7引脚
 * @param resolution - 采样精度
 */
extern uint16 HalAdcRead ( uint8 channel, uint8 resolution );
```

The value range of the resolution parameter is shown in the following figure:

```
59  /* Resolution */
60  #define HAL_ADC_RESOLUTION_8      0x01
61  #define HAL_ADC_RESOLUTION_10     0x02
62  #define HAL_ADC_RESOLUTION_12     0x03
63  #define HAL_ADC_RESOLUTION_14     0x04
```

If you need to use 8-bit sampling accuracy to sample the ADC value of P0_0, you can call HalAdcRead as follows:

```
1.uint8 adcVal;
2.adcVal = HalAdcRead(HAL_ADC_CHANNEL_0, HAL_ADC_RESOLUTION_8);
```

HAL_ADC_CHANNEL_0 represents channel 0, which corresponds to the P0_0 pin of CC2530. 8 channels are defined in the hal_adc.h file, as shown in the figure below.

```
65  /* Channels */
66  #define HAL_ADC_CHANNEL_0      0x00
67  #define HAL_ADC_CHANNEL_1      0x01
68  #define HAL_ADC_CHANNEL_2      0x02
69  #define HAL_ADC_CHANNEL_3      0x03
70  #define HAL_ADC_CHANNEL_4      0x04
71  #define HAL_ADC_CHANNEL_5      0x05
72  #define HAL_ADC_CHANNEL_6      0x06
73  #define HAL_ADC_CHANNEL_7      0x07
```

HAL ADC Experiment

This experiment combines HAL's LED, button, serial port, display and ADC APIs to flash the LED and start ADC sampling after pressing the button, and display the sampled value on the display and send it to the serial port.

In this chapter, we combine buttons, LEDs, displays, ADC, and serial ports to implement a function: when a button is pressed, the ADC samples and flashes the RGB light, then displays the sampled value on the display and sends the sampled value to the serial port! To use ADC, you need to include the corresponding header file: hal_adc.h.

Add the following code in the key processing function zclSampleSw_HandleKeys of the application layer:

```
static void zclSampleSw_HandleKeys( byte shift, byte keys )
{
    UI_MainStateMachine(keys);

    if(keys & HAL_KEY_SW_6)
    {
        uint8 adcVal;
        char adcStr[30];

        /* 读取通道7的ADC数值 · 在配套的ZigBee开发板中 · P0_7是链接着光照传感器 */
        adcVal = HalAdcRead( HAL_ADC_CHANNEL_7, HAL_ADC_RESOLUTION_8 );

        /* 闪烁LED灯 */
        HalLedBlink(HAL_LED_1, 10, 50, 400); // Red
        HalLedBlink(HAL_LED_2, 10, 50, 1000); // Green
        HalLedBlink(HAL_LED_3, 10, 50, 2000); // Blue

        /* 格式化数据到字符串 */
    }
}
```

```

sprintf(adcStr, "ADC: %d\n", adcVal);

/* 在显示屏上将ADC值显示在第四行 */

HalLcdWriteString(adcStr, HAL_LCD_LINE_4);

/* 将ADC数值通过串口发送到串口调试助手 */

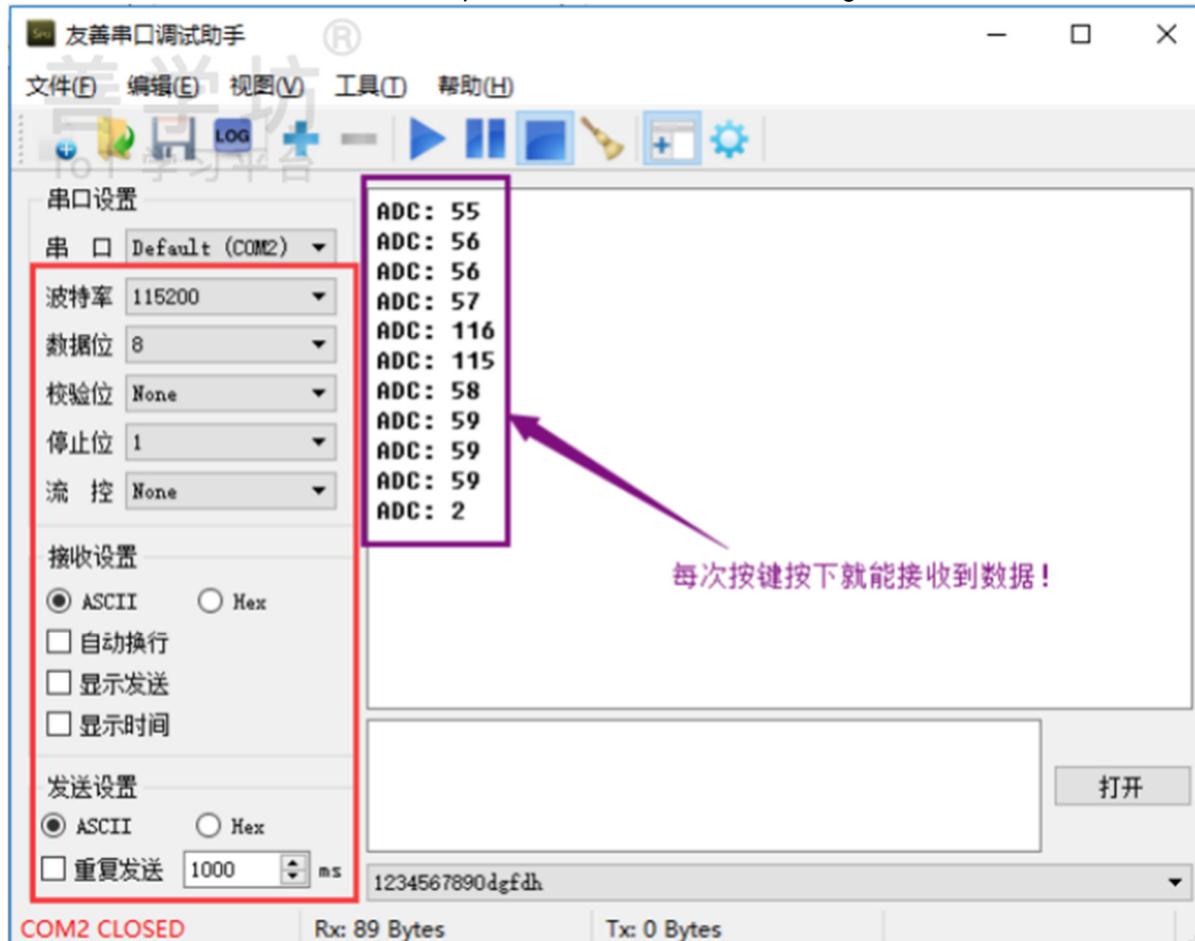
HalUARTWrite(HAL_UART_PORT_0, (uint8 *)adcStr, osal_strlen(adcStr));

}
}

```

Simulation debugging

Compile the entire project, then burn the program to the development board. Connect the ZigBee development board and the computer with a Micro USB cable. When you press the button, you can see that the fourth line of the screen displays the ADC value. You can also see it in the serial port assistant, as shown in the figure below.



7. Part 4: ZigBee 3.0 Network Programming

7.1. Chapter 1: ZigBee 3.0 Network Principles

7.2. Chapter 2: ZigBee 3.0 BDB

7.3. Chapter 3: Data Communication Based on AF

7.4. Chapter 4: ZCL Basic Principles Chapter 5: Switch Command Transmission and Reception Based on ZCL

7.5. Chapter 5 - Switch Command Transmission and Reception Based on ZCL

7.6. Chapter 6: Attribute reading and writing based on ZCL

7.1. Chapter 1: ZigBee 3.0 Network Principles

7.1.1. Protocol Hierarchy

7.1.2. IEEE 802.15.4 Protocol

7.1.3. Network Layer

7.1.1. Protocol Hierarchy

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

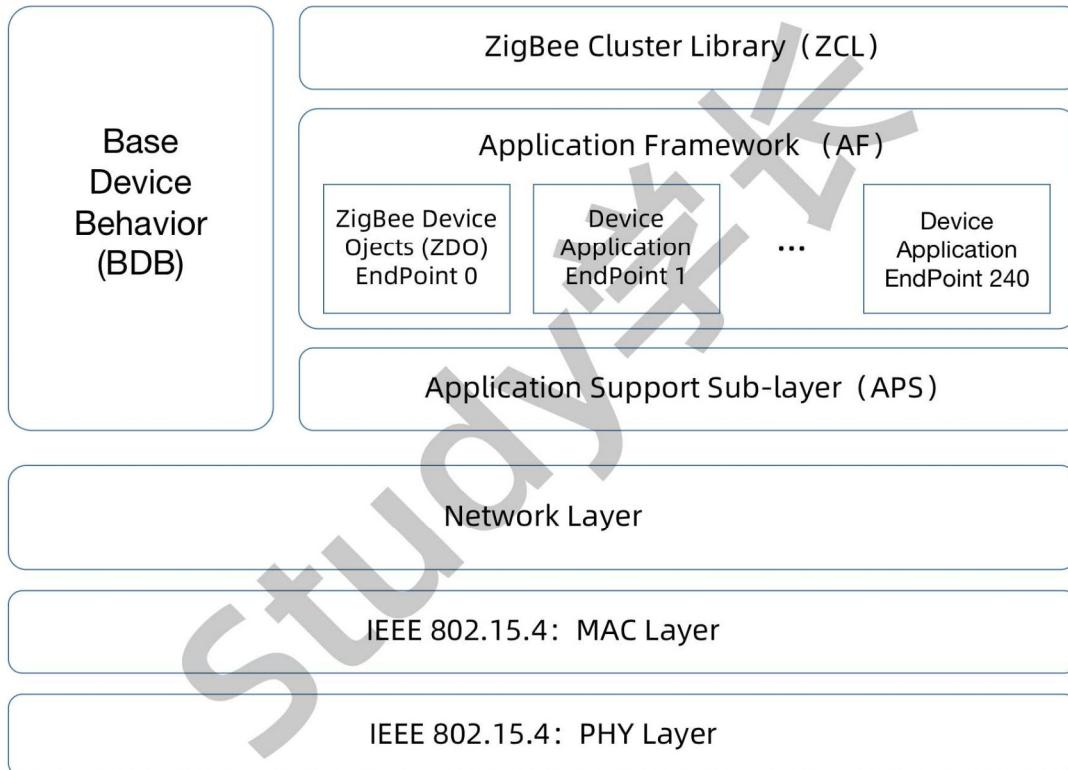
Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers, so it is difficult to guarantee the timeliness of the answers. It is hard work to answer them. Thank you for your understanding! Solution!

This lesson will explain several ZigBee network concepts. If you find it difficult, don't worry! With subsequent learning, I believe that readers will gradually understand the ZigBee protocol!

Protocol Hierarchy

The hierarchical structure of the ZigBee protocol is shown in the figure.



IEEE 802.15.4 PHY layer: Physical layer, whose working content is defined by IEEE 802.15.4. Its main function is to convert the data of one device into electromagnetic wave signals and then send them to another device, which then interprets the electromagnetic wave signals to obtain data.

IEEE 802.15.4 MAC layer: MAC layer, whose work content is defined by IEEE 802.15.4, its main function is to control multiple network devices to communicate in an orderly manner using physical communication resources electromagnetic waves.

Network (NWK) layer: Network layer, responsible for networking, data transmission, and network security management between multiple devices.

Application layer: The application layer can be divided into the following two levels:

- (1) Application Support (APS) Sub-Layer: The application support sub-layer is the transition from the network layer to the application framework layer, providing functions such as data transmission and reception, security encryption, and device address management.
- (2) Application Framework: The application framework layer consists of one or more application endpoints (EndPoint). The application endpoint is the entry and exit for communication between different devices, and is also the basis for describing what functions the device has.

For example, there is an endpoint 1 in the ZigBee smart light, which is used to describe the switch and brightness of the light, as shown in the figure.



For example, there is an endpoint 1 in the ZigBee smart socket, which is used to describe the switch function of the socket, as shown in the figure.



When ZigBee devices communicate over the network, it is ultimately communication between application endpoints. For example, endpoint 1 of the ZigBee coordinator sends an open socket command to endpoint 1 of the smart socket.

If you have learned about the port number of the remote server, you will find that the endpoint mentioned here is similar to the port number of the remote server. For example, the 80 in the URL <http://baidu.com:80> is the port number, indicating the HTTP service. Entering <http://baidu.com:80> in the browser means that the browser will communicate with the HTTP service in the baidu.com URL.

The endpoint is represented by an unsigned 8-bit integer type, with a value range of [0~255, each ZigBee device supports up to 240 application endpoints \(1240\)](#), where endpoint 0 is a special endpoint, endpoint 255 is used for broadcasting to other endpoints, and endpoints 241~254 are reserved endpoints.

ZigBee Cluster Library (ZCL): ZigBee Cluster Library, defined by the ZigBee Alliance, is the basis of ZigBee 3.0 and the basis for the interconnection of devices from different manufacturers.

ZigBee Device Objects (ZDO): ZigBee device object, also known as application endpoint 0. It is the middleware for other application layer endpoints to interact with the application sublayer management entity. The main functions it provides are as follows:

- (1) Manage ZigBee devices.
- (2) Create, scan, and join a network.
- (3) Bind and unbind application endpoints.
- (4) Security management.

Base Device Behavior (BDB): ZigBee basic device behavior, which defines the device behavior specifications to ensure interoperability between devices from different manufacturers. The scope of the basic device behavior specifications is defined as follows:

- (1) The environment required by the basic device.
- (2) The initialization process of the basic device.
- (3) The commissioning process of the basic device.
- (4) The reset process of the basic device.
- (5) The security process of the basic device.

7.1.2. IEEE 802.15.4 Protocol

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers, so it is difficult to guarantee the timeliness of the answers. It is hard work to answer them. Thank you for your understanding! Solution!

When discussing ZigBee technology, we cannot avoid the IEEE 802.15.4 protocol, because the physical layer and MAC layer of ZigBee are based on the IEEE 802.15.4 protocol.

IEEE is an organization called the Institute of Electrical and Electronics Engineers. 802.15 is a department in this association. The 4th working group in the department developed a protocol called IEEE 802.15.4. This protocol is specially designed for low-rate wireless personal area networks (WPANs) and has the characteristics of ultra-low complexity, ultra-low power consumption, and low data transmission rate.



Physical Layer

The physical layer of IEEE 802.15.4 is the lowest layer of the Zigbee protocol structure and provides the most basic services. Generally, as an application developer of ZigBee technology, you only need to understand that the main function of the physical layer is to convert the data of one device into an electromagnetic wave signal and then send it to another device, and then the other device interprets the electromagnetic wave signal to obtain the data.



IEEE 802.15.4 provides three physical layer protocols based on 2.4GHz, 868MHz and 915MHz electromagnetic wave bands. There are some differences between these three:

- **Regional differences:**

2.4 GHz can be used all over the world, but 868 MHz and 915 MHz can only be used in Europe and the United States, respectively. Therefore, in China, the ZigBee protocol is based on 2.4 GHz.

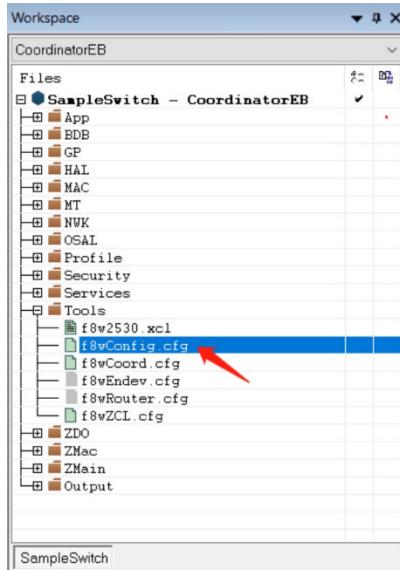
- **The communication rate difference is that**

the 2.4GHz physical layer supports a data rate of 240kb/s, while the data rates of the 868MHz and 915MHz physical layers are 20kb/s and 40kb/s respectively.

- **The difference in the number of channels is that**

the 868MHz band defines one channel, the 915MHz band defines 10 channels, and the 2.4GHz band defines 16 channels.

The channels of the 2.4GHz frequency band are defined in the ZStack 3.0 project file Tools/f8wConfig.cfg. The location of the project file is shown in the figure.



The default channel in the configuration file f8wConfig.cfg is 2.4G band channel 11, and the channel definition is as follows:

```
/* Default channel is Channel 11 - 0x0B */

// Channels are defined in the following:

//      0 : 868 MHz  0x00000001
//      1 - 10: 915 MHz  0x000007FE
//      11 - 26: 2.4 GHz  0x07FFF800
//
//--DMAX_CHANNELS_868MHZ  0x00000001
//--DMAX_CHANNELS_915MHZ  0x000007FE
//--DMAX_CHANNELS_24GHZ  0x07FFF800
//--DDEFAULT_CHANLIST=0x04000000 // 26 - 0x1A
//--DDEFAULT_CHANLIST=0x02000000 // 25 - 0x19
//--DDEFAULT_CHANLIST=0x01000000 // 24 - 0x18
//--DDEFAULT_CHANLIST=0x00800000 // 23 - 0x17
//--DDEFAULT_CHANLIST=0x00400000 // 22 - 0x16
//--DDEFAULT_CHANLIST=0x00200000 // 21 - 0x15
//--DDEFAULT_CHANLIST=0x00100000 // 20 - 0x14
//--DDEFAULT_CHANLIST=0x00080000 // 19 - 0x13
```

```
//-DDEFAULT_CHANLIST=0x00040000 // 18 - 0x12  
//-DDEFAULT_CHANLIST=0x00020000 // 17 - 0x11  
//-DDEFAULT_CHANLIST=0x00010000 // 16 - 0x10  
//-DDEFAULT_CHANLIST=0x00008000 // 15 - 0x0F  
//-DDEFAULT_CHANLIST=0x00004000 // 14 - 0x0E  
//-DDEFAULT_CHANLIST=0x00002000 // 13 - 0x0D  
//-DDEFAULT_CHANLIST=0x00001000 // 12 - 0x0C  
-DDEFAULT_CHANLIST=0x00000800 // 11 - 0x0B
```

ZigBee networks created in different channels do not interfere with each other, but multiple independent ZigBee networks can also be established on the same channel. So how to distinguish between multiple ZigBee networks built on the same channel?

The answer is that each ZigBee network is assigned a unique ID number, called "PanID", which can be used to distinguish different ZigBee networks in the same channel.

MAC layer

If multiple network devices need to send data, how can we control them to send data in an orderly manner?



In order to solve this problem, the Media Access Control (MAC) layer came into being.

The media access control layer is built on top of the physical layer. It does not care how the data is converted into electromagnetic wave signals or what the frequency of the electromagnetic waves is. It only cares about the part it is responsible for, that is:

- First, divide the devices into coordinators and ordinary devices;
- Second, the coordinator generates and sends beacon frames, and ordinary devices synchronize with the coordinator based on the coordinator's beacon frames;
- 3. Association and disassociation of personal area networks;
- 4. Ensure the communication security of wireless channels;
- 5. Support carrier sense multiple access with collision avoidance (CSMA/CA);
- 6. Provide Guaranteed Time Slot (GTS) service;
- 7. Provide reliable transmission services at the MAC layer between different devices.

Similarly, for the time being, we only need to understand in a simple way that the main function of the MAC layer is to control multiple network devices to communicate reliably in an orderly manner.

MAC Address

Each device in the ZigBee network has a fixed MAC address, also known as a physical address or IEEE address, which is used to identify the address of the MAC layer device. MAC is a 64-bit binary address, which is usually fixed into the chip by the chip manufacturer during the chip production process.

7.1.3. Network Layer

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

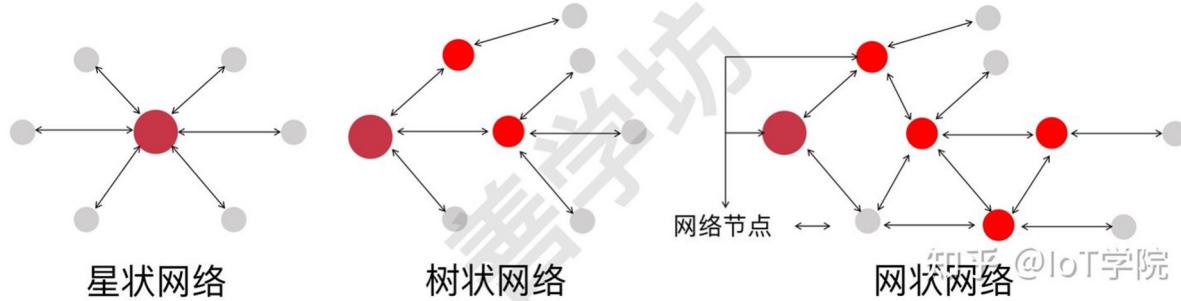
The ZigBee network layer is based on the IEEE 802.15.4 protocol and is the core part of the ZigBee protocol, so it is also commonly called the "core protocol". It is mainly responsible for the following three aspects:

- Multi-device networking
- Data Transfer
- Network Security Management

Detailed explanation of multi-device networking

The first aspect of network topology

is responsible for the networking between multiple devices, that is, the construction and maintenance of star networks, tree networks and mesh networks.



ZigBee device roles

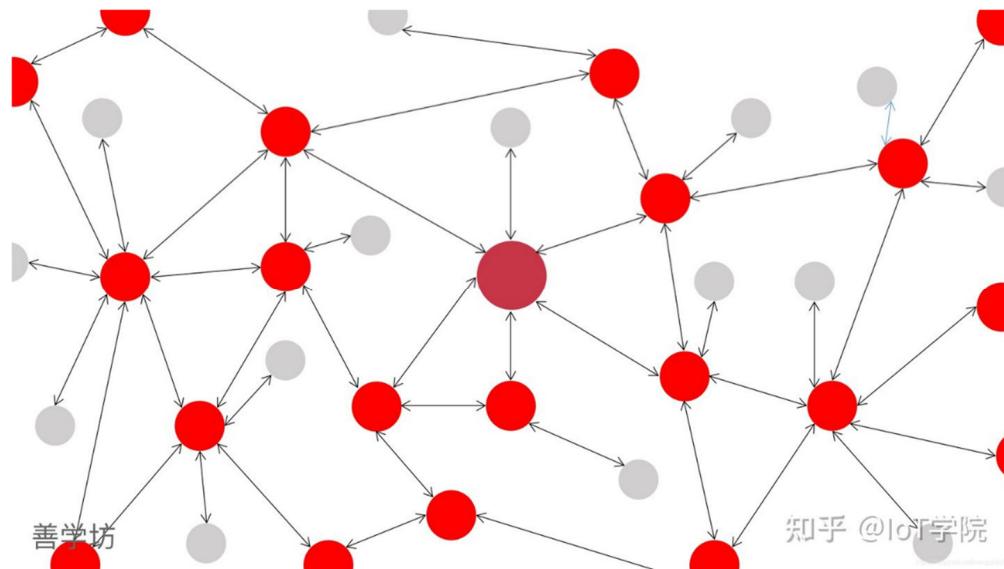
Devices in the network can be commonly referred to as "network nodes". There are three types of ZigBee network nodes:

- **Coordinator:** Acts as the gateway (central node) of the ZigBee network. It is usually responsible for the conversion between the ZigBee protocol and other protocols such as NB-IoT and WiFi, establishing a network on a specific channel, etc. It also has the function of a router.
- **Router:** Also known as a repeater, responsible for data routing. All terminal devices need to be added to the network through a coordinator or router.
- **End Device:** Also known as a leaf node, it must be connected to the ZigBee network through a coordinator or router. For example, in a smart home scenario, the end device is usually a temperature and humidity sensor, a wireless switch button, or various household appliances.

Networking Features

ZigBee networking has three main features. First, it supports the construction and maintenance of mesh networks with more than 10,000 network nodes, far exceeding Bluetooth's 8 and WiFi's 32. Its benefits are:

- On the one hand, more devices can be added to the automation control and remote control, and the limitation on the number of devices no longer exists;
- On the other hand, in a mesh network, there are one or more communication links between two network nodes, which can provide multi-channel communication services. In complex industrial scenarios, it is often impossible to ensure that every wireless network communication link is always unobstructed. Multi-channel communication enables communication using other links when a link is blocked, ensuring the stability of communication.



Second, **it supports dynamic routing**, that is, dynamically calculating the optimal communication path between any two nodes in the network based on the real-time status of each network node. For example, in a mesh network scenario, there may be multiple communication paths between any two nodes, and the optimal communication path can be dynamically selected by calculating the real-time quality of each path.

Third, **it supports self-organizing networks**, that is, when network nodes are separated and cannot communicate because they are out of communication range, when they come back to the communication range and gather together again, they can automatically rebuild the network to achieve data communication.

Data transmission details

Data transmission refers to the transmission of control instructions and device status information between devices. For example, taking air conditioners as an example, the control instructions here refer to the air conditioner's switch, cooling temperature setting, working mode setting and other instructions; status information refers to the state of the air conditioner at a certain

moment, such as the set temperature, indoor temperature, working mode, etc.

善学坊



Safety Management

Network security management refers to the encryption and decryption of data, etc.

Network Address

Each ZigBee device in the network is assigned a network address for identification, and the corresponding device can be found through this network address. The ZigBee network address is a 16-bit address (0x0000 - 0xFFFF). In ZStack 3.0, there are several special network addresses that need to be understood:

- (1) The network address of the coordinator is fixed at 0x0000.
- (2) 0xFFFF - This is a broadcast address that broadcasts to the entire ZigBee.
- (3) 0xFFFD - An address that is only broadcast to devices that have enabled reception.
- (4) 0xFFFC - An address that is only broadcast to coordinators and routing devices.
- (5) 0xFFFE - Used as an invalid address.
- (6) 0xFFF8 ~ 0xFFFF - Reserved addresses.
- (7) 0x0001 ~ 0xFFFF are assigned to devices in the ZigBee network and used as network addresses.

7.2. Chapter 2: ZigBee 3.0 BDB

7.2.1. Introduction to BDB

7.2.2. BDB Commissioning Modes

7.2.3. ZigBee 3.0 Networking Experiment

7.2.1. Introduction to BDB

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=33>

Technical support instructions:

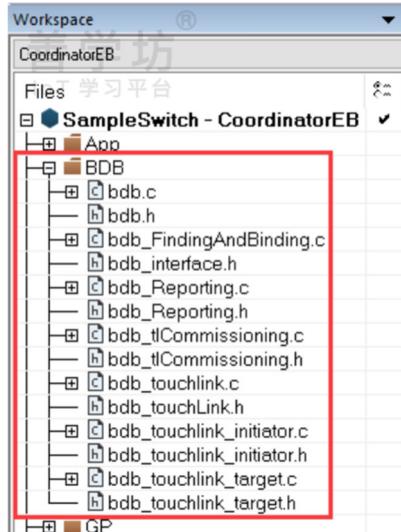
1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

The many theoretical knowledge explained in this lesson are not easy to understand. It doesn't matter if you don't understand them for the time being. I believe that with subsequent learning, your understanding will gradually deepen!

Before ZigBee devices can send data to each other, they need to build a network. BDB (Base Device Behavior) is a new feature of ZigBee 3.0, which provides a unified mechanism for each ZigBee device **to correctly build a ZigBee network**. BDB mainly includes the following three aspects:

- **Commissioning Modes:** Commissioning mode defines the basic specifications for networking between ZigBee devices
- **BDB Security:** defines some network security specifications
- **Reset Methods:** Developers can use multiple reset methods

BDB-related code files can be found in the BDB folder of the protocol stack, as shown in the figure.



Since developers are generally less familiar with BDB Security and Reset Methods, we will not explain them in detail for now. **Commissioning Modes is the core content of ZigBee networking** and will be the focus of the next lesson.

7.2.2. BDB Commissioning Modes

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=34>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers **will** answer community questions as soon as possible, but they are front-line developers and [**cannot guarantee**] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

The multiple theoretical knowledge explained in this lesson is not easy to understand. It doesn't matter if you don't understand it for the time being. I believe that with subsequent learning, your understanding will gradually deepen!

Commissioning can be directly translated as "debugging" or "trying to operate", which is a bit strange, so this article will not translate it for the time being. Readers can generally understand it as "building a network", so "Commissioning Modes" can be generally understood as "networking mode", but for the sake of rigor, this article still uses Commissioning.

BDB provides four main commissioning modes for developers to use, namely Network Steering, Network Formation, Finding and Binding (F & B) and Touchlink.

Network Steering

Defines how devices join the ZigBee network. The specific methods are as follows:

- If the devices are not already in a ZigBee network, they will search for a suitable ZigBee network and join it.
- In particular, for router-type devices, after successfully joining the network, other devices are allowed to join the ZigBee network through this device.

All devices that need to join the ZigBee network must support Network Steering.

Network Formation

Network Formation stipulates that coordinator-type devices need to establish a centrally trusted secure network. The characteristic of this network is that all devices that need to join the network must be approved by the trust center before joining, and the coordinator itself is the trust center.

Similarly, for router-type devices, if conditions permit, a distributed security network will be created. This type of network will not be explained here for the time being.

All coordinator type devices must support Network Formation, while for router type devices, this is an optional mode.

Finding and Binding (F & B)

As the name suggests, Finding and Binding means finding and binding, so what do we find and bind?

ZigBee 3.0 uses Cluster to describe the functions of devices. Each device has its own functions and its own series of Clusters. The discovery and binding here refers to the mutual discovery and binding between Clusters of ZigBee devices. As the Cluster is explained in depth in the following chapters, readers will have a deeper understanding of the principles.

All ZigBee devices must support Finding and Binding (F & B).

Touchlink

It is generally used for direct communication between two ZigBee devices. Here is an example to illustrate its principle.

Assume that there are two ZigBee devices that support Touchlink:

- One is a wireless button that supports sending a factory reset command via Touchlink;
- The other is the light, which supports receiving factory reset instructions sent via Touchlink and performing corresponding processing.

The user can hold the button close to the light, **keeping the two about 2 cm apart**, and then press the button to send a command. The light will receive the command and perform corresponding processing.

If the user **moves the button farther away to test**, the light will not receive the command. If the user moves the button closer to **a newly purchased identical light** to test, the light will also receive the command.

From this example, we can conclude that Touchlink has the following features:

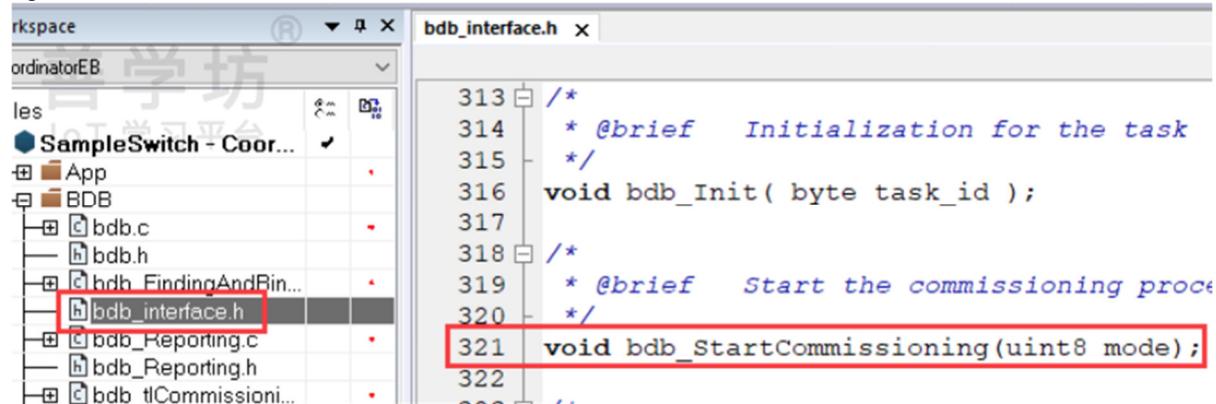
- The communication distance is short, about 2cm. When the button sends a command to the light, the light will determine the distance of the button by detecting the network signal strength of the button, and then decide whether to process the command.
- Devices can communicate directly with each other. You can use this button to communicate directly with a newly purchased identical lamp.

Touchlink is relatively rarely used, and not all ZigBee devices necessarily need to support Touchlink. Developers can make their own devices support or not support this function.

Introduction to BDB Commissioning Modes API

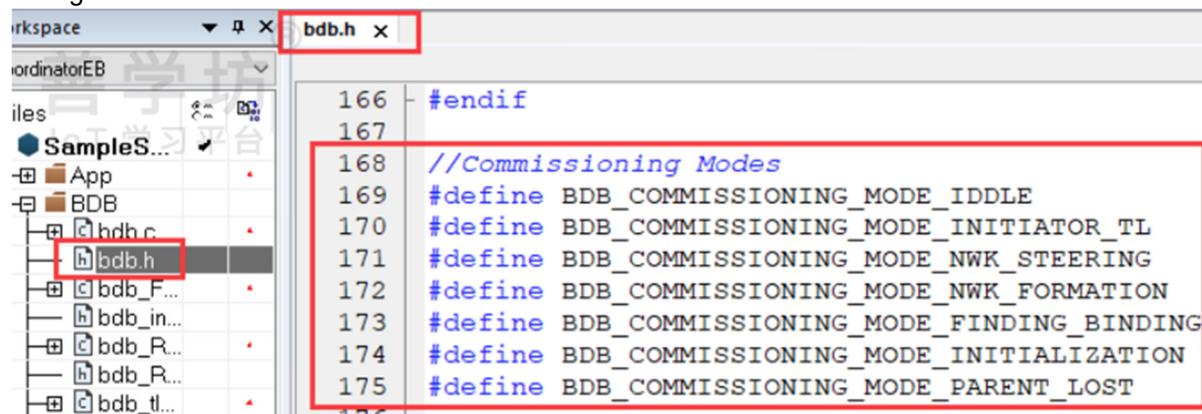
Although the above Commissioning mode is relatively complex, thanks to Z-Stack 3.0, the networking code of ZigBee devices is very simple and only requires calling one API.

Open the supporting project code and find the BDB Commissioning Modes API in the bdb_interface.h file, as shown in the figure.



```
313 /*  
314  * @brief  Initialization for the task  
315  */  
316 void bdb_Init( byte task_id );  
317  
318 /*  
319  * @brief  Start the commissioning process  
320  */  
321 void bdb_StartCommissioning(uint8 mode);  
322 ..
```

This function needs to pass in a mode parameter. The mode defined by the protocol stack can be found in bdb.h, as shown in the figure.



```
166 #endif  
167  
168 //Commissioning Modes  
169 #define BDB_COMMISSIONING_MODE_IDLE  
170 #define BDB_COMMISSIONING_MODE_INITIATOR_TL  
171 #define BDB_COMMISSIONING_MODE_NWK_STEERING  
172 #define BDB_COMMISSIONING_MODE_NWK_FORMATION  
173 #define BDB_COMMISSIONING_MODE_FINDING_BINDING  
174 #define BDB_COMMISSIONING_MODE_INITIALIZATION  
175 #define BDB_COMMISSIONING_MODE_PARENT_LOST
```

Finally, we can build a ZigBee network. First, let the coordinator create a network. The code is as follows:

```
bdb_StartCommissioning()//组建网络  
  
BDB_COMMISSIONING_MODE_NWK_FORMATION //支持Network Formation  
  
BDB_COMMISSIONING_MODE_FINDING_BINDING //支持Finding and Binding (F & B)  
);
```

Then, let the router or terminal device join the network. The code is as follows:

```
bdb_StartCommissioning()//设备入网  
  
BDB_COMMISSIONING_MODE_NWK_STEERING //支持Network Steering  
  
BDB_COMMISSIONING_MODE_FINDING_BINDING //支持Finding and Binding (F & B)  
);
```

Generally speaking, in the actual development process, developers only need to use this API to basically meet the functional requirements of network creation and device access. This is a major advantage of BDB, it is easy to use!

7.2.3. ZigBee 3.0 Networking Experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=35>

Technical support instructions:

1. Generally, self-study is the main method.

2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>

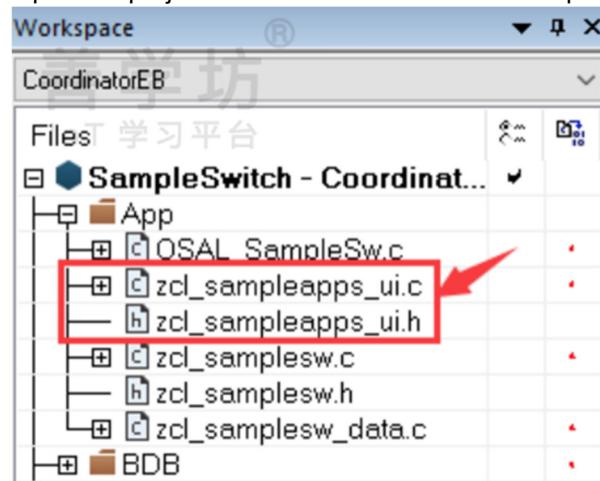
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

After learning the basic principles of BDB Commissioning Modes in the last class, this class explains how to implement network establishment between devices.

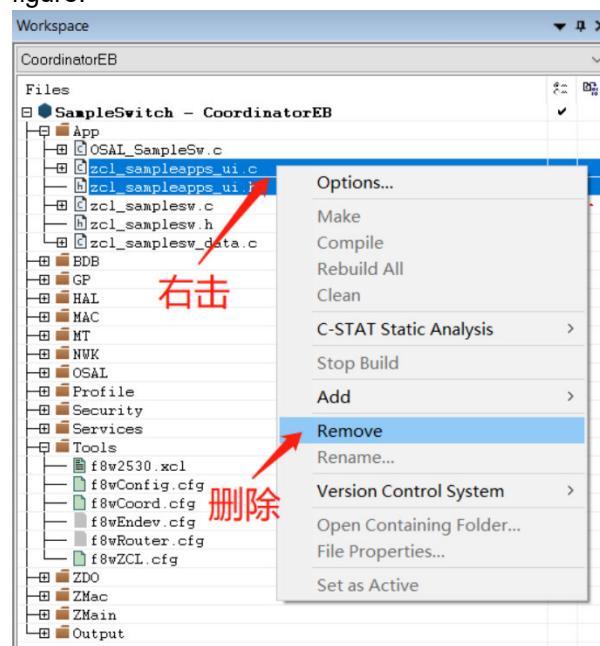
Cutting application layer UI

Protocol stack tailoring is a technology that developers need to master, because Z-Stack 3.0 includes rich functions, but in the actual development process, some functions may be redundant and even affect the development process.

Open the project code that comes with this chapter and expand the App layer code, as shown in the figure.



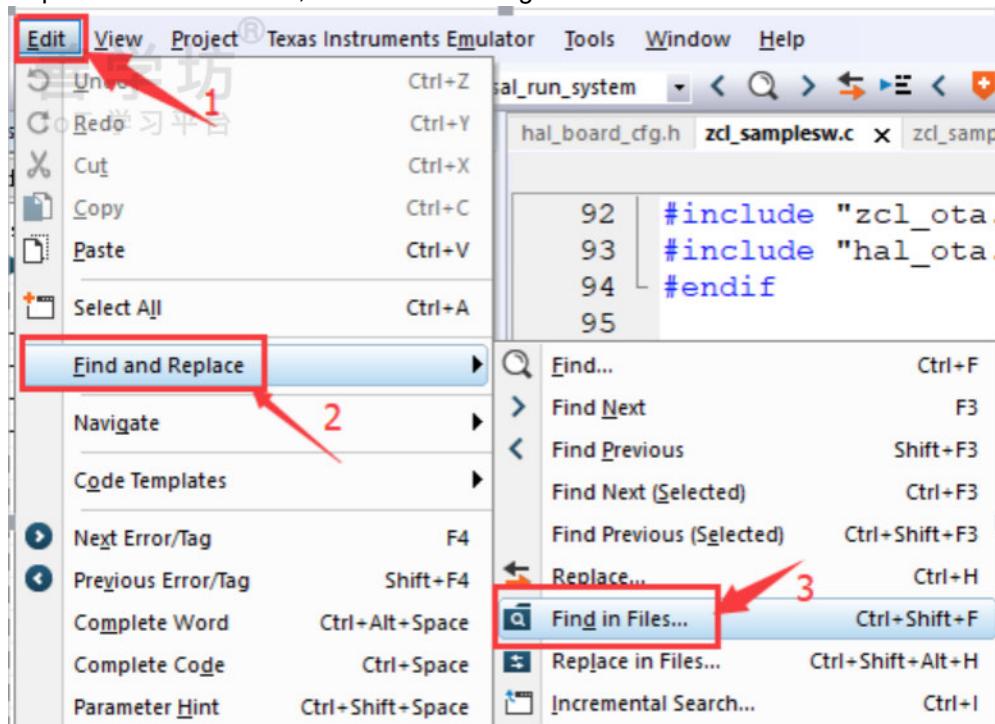
The two files in the red box are UIs designed for TI's evaluation board. They are not needed in the product development process and will affect other development work, so they need to be cut out. Right-click the file and select Remove, as shown in the figure.



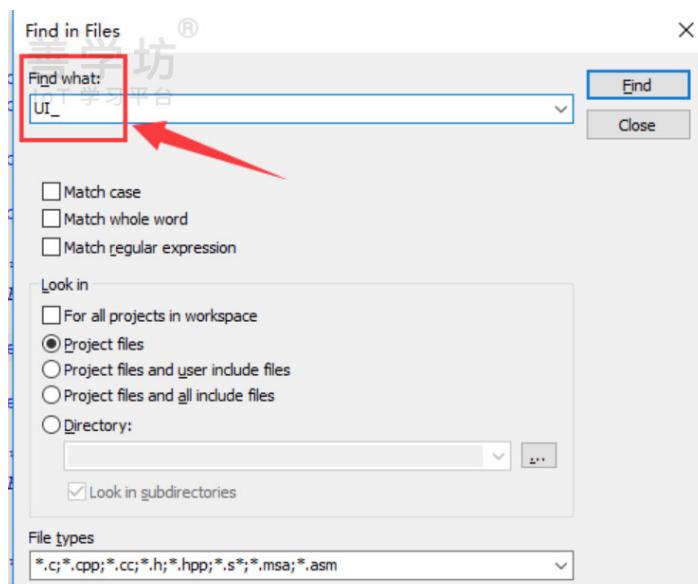
After cutting, it will look like the picture.



These UIs are called in the zcl_samplesw.c file, so you also need to delete them all. Click Edit, then select Find and Replace→Find in Files..., as shown in the figure.



Enter "UI_" in the pop-up window, and then click the Find button, as shown in the figure.



Finally, in the file zcl_samplesw.c, delete the places with "UI_" one by one according to the search results, as shown in the figure.

```
zcl_samplesw.c x
104 * MACROS
105 */
106 #define UI_STATE_TOGGLE_LIGHT 1 //UI_
107
108 #define APP_TITLE "TI Sample Switch"
109
110 ****
111 * TYPEDEFS
```

After the deletion is completed correctly, the project code can be compiled successfully.

Write device networking code

As mentioned in the previous chapter, the device networking code is very simple:

- The first step is to call `bdb_StartCommissioning()` to let the coordinator create the network.
- The second step is to call `bdb_StartCommissioning()` again to allow other devices to join the network.

When compiling

In a project, you can choose to have the compiled program work in a coordinator, router, or terminal, that is, to select the role of the device in the ZigBee network. In the code, you can use the macro `ZDO_COORDINATOR` to determine the current working mode. The judgment method is that if the macro `ZDO_COORDINATOR` is defined, it means that the program works in a coordinator environment, otherwise it works in a router or terminal environment.

Write code

Add the following code to the initialization function `zclSampleSw_Init()` in `zcl_samplesw.c`:

```
#ifdef ZDO_COORDINATOR
    bdb_StartCommissioning( BDB_COMMISSIONING_MODE_NWK_FORMATION |
        BDB_COMMISSIONING_MODE_FINDING_BINDING );

    NLME_PermitJoiningRequest(255);

#else
    bdb_StartCommissioning( BDB_COMMISSIONING_MODE_NWK_STEERING |
        BDB_COMMISSIONING_MODE_FINDING_BINDING );
#endif
```

The code is relatively simple. If it is a coordinator, then let the coordinator create a network. If it is a router or terminal, then join the network.

`NLME_PermitJoiningRequest(255)` allows other devices to join the network created by this coordinator. This is also mentioned in the introduction of Network Formation. That is, the device must obtain the approval of the trust center before joining the network. Parameter 255 means always allowed. If it is changed to 0, it means never allowed. If it is changed to 1, it means [means in 1254 seconds allowed](#).

Handling device network access failure

However, the process of device accessing the network may be affected by many factors, resulting in network access failure, such as signal interference during the access process. Developers need to learn how to handle device network access failure. A simple and effective way is to let the device automatically retry network access after the device fails to access the network.

In the `zcl_samplesw.c` file, you can find a `zclSampleSw_ProcessCommissioningStatus()` function, which is used to process the Commissioning results, such as whether the coordinator successfully creates the network and whether the device successfully joins the network.

In this function, you can take appropriate actions based on the result of Commissioning. For example, if the device fails to join the network, call `bdb_StartCommissioning()` again to join the network again.

This process can be implemented using the event mechanism. When the device fails to access the network, an event is started, and the program tries to access the network again after 1 second. The event usage of Z-Stack 3.0 has been explained in the previous chapter, so I will not explain it here, but use it directly.

1. Define a user event in the header file `zcl_samplesw.h` and define the time interval for re-entering the network. The code is as follows:

```
#ifdef ZDO_COORDINATOR

/* 路由器或者终端 */

#ifndef
// 重新加入事件

#define SAMPLEAPP_REJOIN_EVT      0x0100

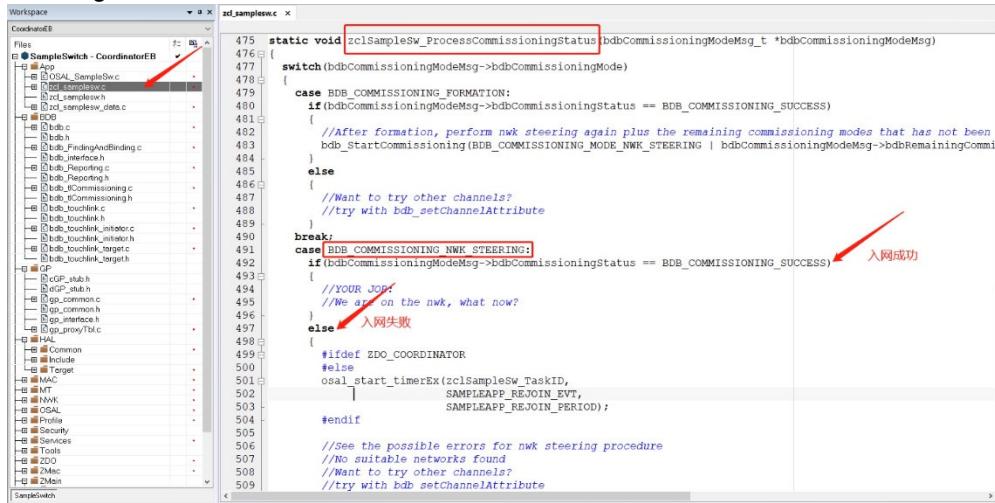
// 时间间隔：1000ms(1秒)

#define SAMPLEAPP_REJOIN_PERIOD   1000

#endif
```

Since the device access is for ZigBee devices of router or terminal type, macros are also used to determine the program role.

2. Find the location of network access failure processing in the function zclSampleSw_ProcessCommissioningStatus, as shown in the figure.



3. Add the following code at the location where network access fails:

```
/* 协调器 */

#ifndef ZDO_COORDINATOR

/* 路由器或者终端 */

#ifndef
// 启动重新加入事件

osal_start_timerEx(zclSampleSw_TaskID,
                    SAMPLEAPP_REJOIN_EVT,
                    SAMPLEAPP_REJOIN_PERIOD);

#endif
```

In the code, the device role is first determined, and then the event just defined is started.

Developers can also add some code at the location where the network access is successful, such as displaying the successful network access information on the display.

3. Process the event in the application layer event processing function zclSampleSw_event_loop(), that is, restart the network access. The code is as follows.

```
#ifdef ZDO_COORDINATOR

#ifndef
if (events & SAMPLEAPP_REJOIN_EVT) //如果事件类型为重新加入网络事件
{
    /* 重新加入网络 */
```

```

bdb_StartCommissioning(BDB_COMMISSIONING_MODE_NWK_STEERING |
    BDB_COMMISSIONING_MODE_FINDING_BINDING );

return ( events ^ SAMPLEAPP_REJOIN_EVT );
}

#endif

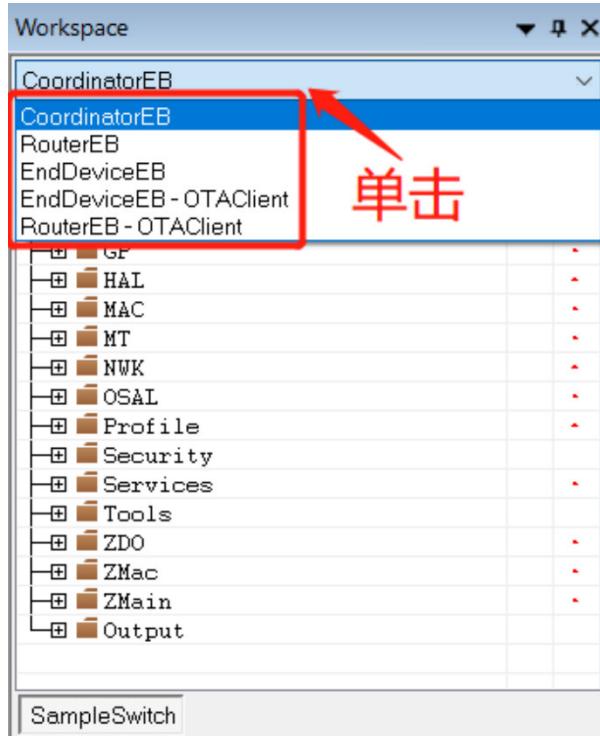
```

Compile and download

Role selection

The network role of the device has been mentioned many times in the previous article. In fact, you can choose different network roles when compiling the code, so as to write programs for different roles.

Click the tab in the project, and you can see the selection box as shown in the figure, where you can select the corresponding working mode of the device to be developed.



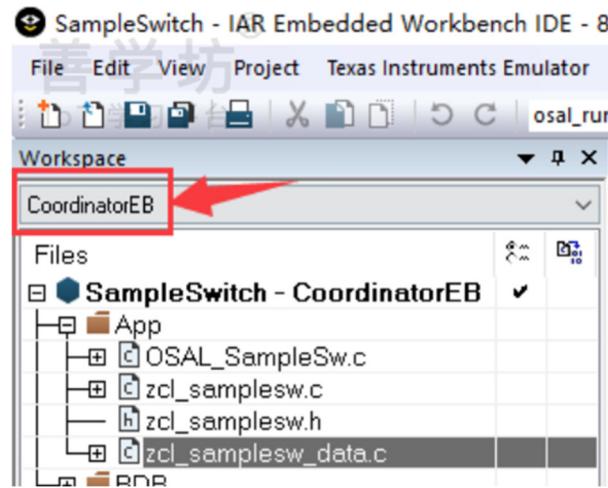
The meanings of CoordinatorEB, RouterEB, and EndDeviceEB are:

- (1) CoordinatorEB: Coordinator role device.
- (2) RouterEB: Router role device.
- (3) EndDeviceEB: Terminal role device.

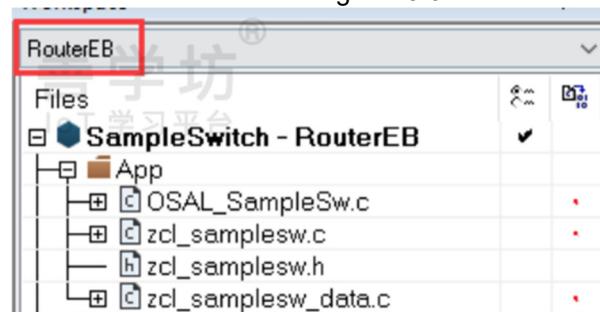
In addition to this, there are EndDeviceEB-OTAClient and RouterEB-OTAClient. In fact, these two refer to terminal devices with OTA functions and routing devices with OTA functions.

First, select CoordinatorEB as the role to use, as shown in the figure. After compiling, download it to one of the development boards. It is recommended to use the ZigBee 3.0 Mini board. This development board will act as a coordinator role device in the

ZigBee network.



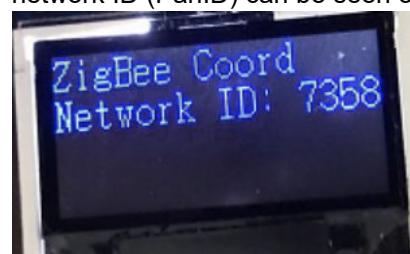
Then select RouterEB, as shown in the figure. After the compilation is completed, download it to another board. It is recommended to use the ZigBee 3.0 standard board. This development board will act as a router in the ZigBee network.



Note: After switching the project, you also need to enable the LED, serial port, display and other functions in the pre-compilation of the project, and add the screen path to the project.

Simulation debugging

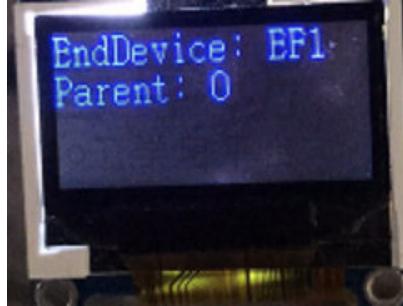
(1) After the coordinator is powered on, it will automatically create a network. After the network is successfully created, the network ID (PanID) can be seen on the screen, as shown in the figure.



(2) Then, after powering on the router or terminal, it will automatically join the network created by the coordinator. After joining, you can see the assigned network address and the parent node's network address on the screen. The router or terminal device will be shown in the figure after joining.

- router





7.3. Chapter 3: Data Communication Based on AF

7.3.1. Simple Descriptor

7.3.2. Communication Principle

7.3.3. Introduction to Data Sending API

7.3.4. ZigBee 3.0 Communication Experiment

7.3.1. Simple Descriptor

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=36>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

In Z-Stack 3.0, simple descriptors are used to describe a certain service of a device, which can also be called a function or application. For example, a ZigBee temperature and humidity sensor has a temperature and humidity monitoring service, and a simple description can be used to describe the specific content of this temperature and humidity monitoring service.

Structural analysis

A simple descriptor is essentially a structure, which is defined in Z-Stack 3.0 as follows:

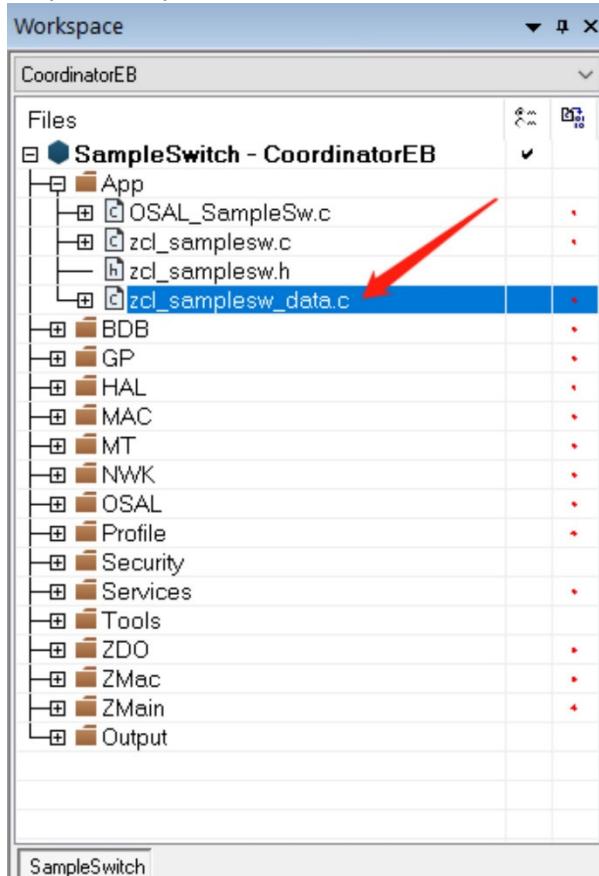
```
typedef struct{
    uint8 EndPoint;//端点号
    uint16 AppProId;//描述所在应用场景，例如家居自动化
    uint16 AppDeviceId;//设备ID
    uint8 AppDevVer:4;//由开发者自定义，表示版本号
    uint8 Reserved:4;//保留字段
    uint8 AppNumInClusters;//端点支持的输入簇个数
    cld_t *pAppInClusterList;//指向输入簇列表的指针
    uint8 AppNumOutClusters;//端点支持的输出簇个数
    cld_t *pAppOutClusterList;//指向输出簇列表的指针
}SimpleDescriptorFormat_t;
```

The following is a brief explanation of the meaning of each element.

- **Endpoint number:** EndPoint can be understood as the number of the simple descriptor, and its value range is 0 to 255. In the same ZigBee device, each simple descriptor has a different endpoint number, and the endpoint number can be used to find the corresponding simple descriptor.
- **AppProfileId:** Profile ID, which indicates the application scenario to which this simple descriptor belongs. This application scenario can be home automation, smart lighting, smart retail, etc. The ZigBee Alliance defines corresponding ID values for different scenarios, called Profile IDs.
- **AppDeviceId:** Device ID, indicating the type of device to which this simple descriptor belongs. This device type can be a socket, a lamp, or a sensor. Similarly, the ZigBee Alliance defines corresponding ID values for different types of devices, called Device IDs.
- **AppDevVer:** This value can be customized by the developer to indicate the version number.
- **Reserved:** Reserved field, which can be ignored for now.
- **Cluster:** Cluster, or cluster, can be divided into input cluster (In Cluster) and output cluster (Out Cluster), used to describe the specific content of this service. A simple descriptor can contain multiple Clusters, which together describe the specific content of this service, which will be explained in detail in subsequent chapters.

Application Examples

We have just explained the definition of a simple description, so the next step is to create a simple descriptor for the device. The simple descriptor of the device is created in the zcl_samplesw_data.c file, and the location of this file is shown in the figure.



A simple descriptor of the device can be found in this file. The code is as follows:

```
SimpleDescriptionFormat_t zclSampleSw_SimpleDesc =
```

```
{
```

```

SAMPLESW_ENDPOINT,          // int Endpoint;
ZCL_HA_PROFILE_ID,         // uint16 AppProflId[2];
ZCL_HA_DEVICEID_ON_OFF_LIGHT_SWITCH, // uint16 AppDeviceId[2];
SAMPLESW_DEVICE_VERSION,    // int AppDevVer:4;
SAMPLESW_FLAGS,            // int AppFlags:4;
ZCLSAMPLESW_MAX_INCLUSTERS, // byte AppNumInClusters;
(cld_t *)zclSampleSw_InClusterList, // byte *pAppInClusterList;
ZCLSAMPLESW_MAX_OUTCLUSTERS, // byte AppNumInClusters;
(cld_t *)zclSampleSw_OutClusterList // byte *pAppInClusterList;
};

```

Explain in detail the meaning of each element.

- SAMPLESW_ENDPOINT is the endpoint number defined in the zcl_samplesw.h file and is defined as follows:

```
#define SAMPLESW_ENDPOINT 8
```

- ZCL_HA_PROFILE_ID is defined by the ZigBee Alliance and represents the Profile ID in the smart home field.
- ZCL_HA_DEVICEID_ON_OFF_LIGHT_SWITCH is a device type ID, indicating that this is a smart socket, and its value is defined by the ZigBee Alliance. Different companies must use this device type ID when developing this type of smart socket, which is the basis for interconnection.
- SAMPLESW_DEVICE_VERSION is the version number defined in the zcl_samplesw_data.c file and is defined as follows:

```
#define SAMPLESW_DEVICE_VERSION 1
```

- SAMPLESW_FLAGS is a reserved field defined in the zcl_samplesw_data.c file and can be temporarily ignored. It is defined as follows:

```
#define SAMPLESW_FLAGS 0
```

- ZCLSAMPLESW_MAX_INCLUSTERS is defined in the zcl_samplesw_data.c file and indicates the maximum number of input clusters supported.
- (cld_t *)zclSampleSw_InClusterList represents the input cluster list.
- ZCLSAMPLESW_MAX_OUTCLUSTERS is defined in the zcl_samplesw_data.c file and indicates the maximum number of output clusters supported.
- (cld_t *)zclSampleSw_OutClusterList represents the output cluster list.

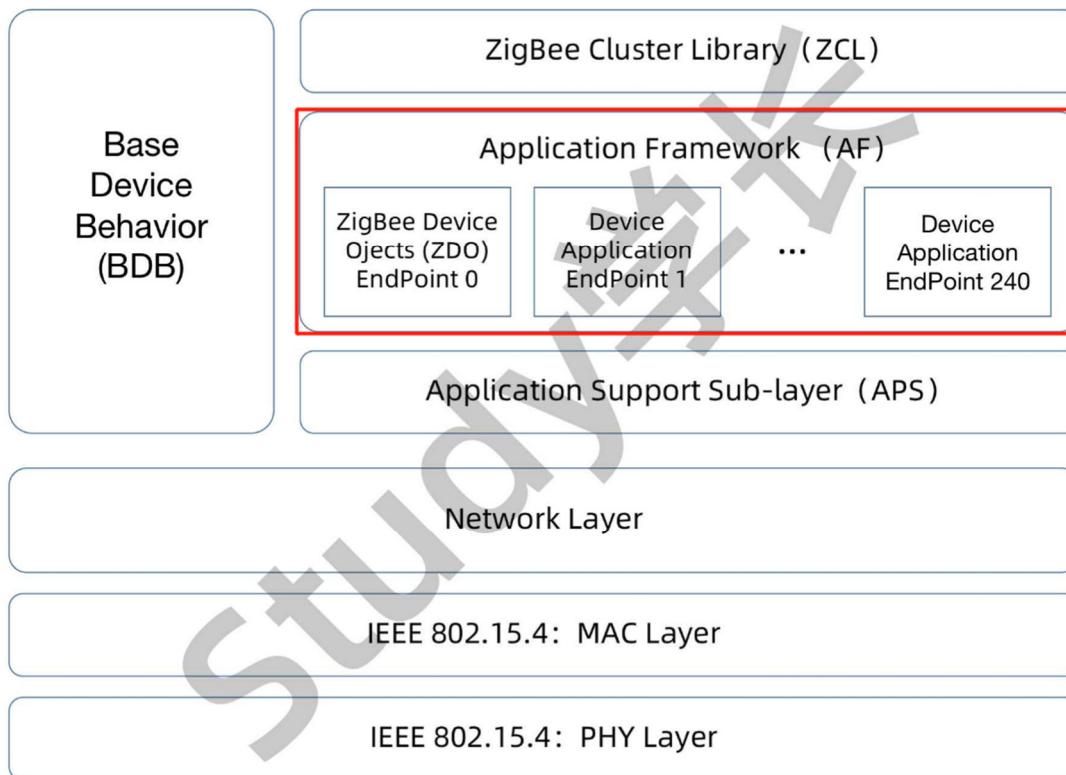
7.3.2. Communication Principle

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=37>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

The full name of AF is Application Framework, which means application framework in Chinese. The AF layer is also the application framework layer. Its position in the ZigBee protocol hierarchy is shown in the figure.



Developers can write code based on this level to achieve data communication between ZigBee devices.

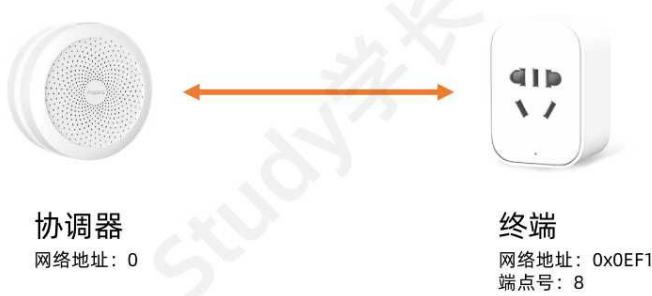
Communication principle of AF layer

Network address

The previous chapter has explained the network address. All ZigBee devices in the network will be assigned a network address to identify the location of the device. The corresponding device can be found through this network address.

Endpoint Number

Readers already know that ZigBee 3.0 uses simple descriptors to describe specific services, and each simple descriptor on a device is equipped with a unique endpoint number. In fact, ZigBee 3.0 communication is ultimately **sent to a specific service of the device, that is, the endpoint number**. Let's take an example to illustrate the principle.



Assume that there is a coordinator and a smart socket terminal in the ZigBee network. The network address of the smart socket is 0xEF1, and it is equipped with a simple descriptor to describe some functions of the socket. The endpoint number of this simple descriptor is 8.

If the coordinator wants to send an open command to this socket, it cannot simply use the network address 0xEF1 to indicate that the command is sent to this socket. It also needs to use endpoint number 8 to indicate that the command is sent to the service corresponding to endpoint number 8.

If you are familiar with the port number of the remote server, you will find that they are similar. For example, this URL «<http://www.sxf-iot.com:80>» contains both the domain name and a port number 80.

The value range of endpoint

number is 0~255, where:

- Endpoint number 0 is assigned to ZDO (ZigBee Device Object).
- Endpoint number 255 is broadcast, that is, when data is sent to this endpoint number, all services (simple descriptors) of the device will receive the data.
- Endpoint numbers 241 to 254 are temporarily reserved and cannot be used for the time being.
- The remaining endpoint numbers 1 to 240 can be used freely by developers. For example, the endpoint numbers used by the SampleSwitch project that readers have been studying can be found in the project, as shown in the figure.

The screenshot shows a software development environment with a 'Workspace' window on the left and a code editor window on the right. The workspace shows a project named 'CoordinatorEB' with a 'SampleSwitch - Coor...' folder containing an 'App' folder and files like 'OSAL_SampleSw.c', 'zcl_samplesw.c', and 'zcl_samplesw.h'. A red arrow points from the 'zcl_samplesw.h' file in the workspace to its content in the code editor. The code editor window title is 'zcl_samplesw.h'. The code itself is as follows:

```
55 //*****
56 * CONSTANTS
57 */
58 #define SAMPLESW_ENDPOINT 8
59
60 #define LIGHT_OFF
61 #define LIGHT_ON
62
```

Communication Mode

ZigBee supports three communication modes: point-to-point, broadcast and multicast.

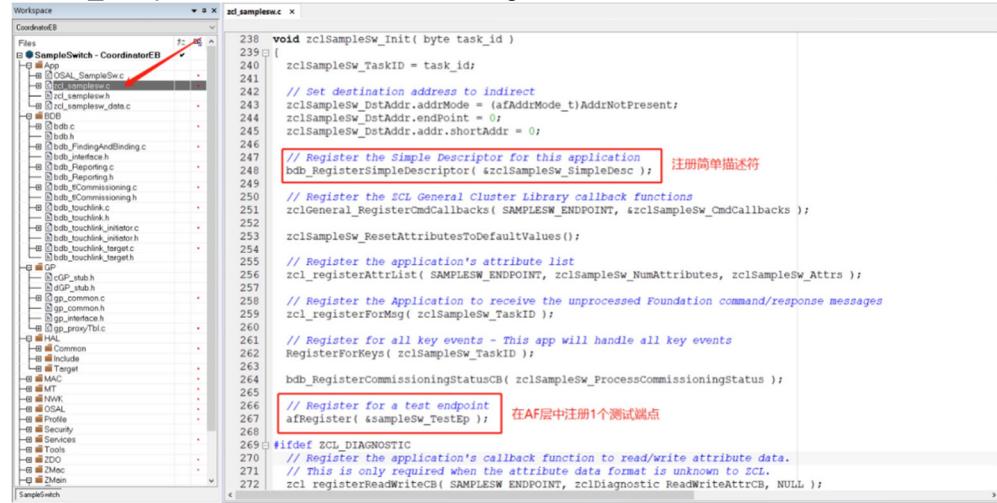
- Peer to Peer (P2P): As the name suggests, it is one-to-one communication between two devices.
- Broadcast: Send data to all other devices.
- Multicast: Send data to a part of the devices. The "part" here refers to a specific group.

Registering a simple descriptor

Before using AF communication, you need to register a simple descriptor and make the corresponding endpoint number effective, or make the service corresponding to this simple descriptor effective. The process is:

(1) Create a simple descriptor. We have already seen the relevant definition in the chapter on simple descriptors.

(2) Register this simple descriptor. Open the supporting project and find the following code in the zclSampleSw_Init() function in the zcl_samplesw.c file, as shown in the figure.



```
238 void zclSampleSw_Init( byte task_id )
239 {
240     zclSampleSw_TaskID = task_id;
241
242     // Set destination address to indirect
243     zclSampleSw_DstAddr.addrMode = (afAddrMode_t)AddrNotPresent;
244     zclSampleSw_DstAddr.endPoint = 0;
245     zclSampleSw_DstAddr.shortAddr = 0;
246
247     // Register the Simple Descriptor for this application
248     bdb_RegisterSimpleDescriptor( &zclSampleSw_SimpleDesc ); // 注册简单描述符
249
250     // Register the ZCL General Cluster Library callback functions
251     zclGeneral_RegisterCmdCallbacks( SAMPLESW_ENDPOINT, &zclSampleSw_CmdCallbacks );
252
253     zclSampleSw_ResetAttributesToDefaultValues();
254
255     // Register the application's attribute list
256     zcl_registerAttrList( SAMPLESW_ENDPOINT, zclSampleSw_NumAttributes, zclSampleSw_Attrs );
257
258     // Register the Application to receive the unprocessed Foundation command/response messages
259     zcl_registerForMsg( zclSampleSw_TaskID );
260
261     // Register for all key events - This app will handle all key events
262     RegisterForKeys( zclSampleSw_TaskID );
263
264     bdb_RegisterCommissioningStatusCB( zclSampleSw_ProcessCommissioningStatus );
265
266     // Register for a test endpoint
267     afRegister( &sampleSw_TestEp ); // 在AF层中注册1个测试端点
268
269 #ifdef ZCL_DIAGNOSTIC
270     // Register the application's callback function to read/write attribute data.
271     // This is only required when the attribute data format is unknown to ZCL.
272     zcl_registerReadWriteCB( SAMPLESW_ENDPOINT, zclDiagnostic_ReadWriteAttrCB, NULL );
273 
```

In the application layer initialization function zclSampleSw_Init(), there are two ways to register simple descriptors:

- (1) You can call bdb_RegisterSimpleDescriptor() to register simple descriptors in BDB.
- (2) You can also directly call afRegister() to register simple descriptors in AF.

After registering the simple descriptor, you can use the AF layer's communication API to send and receive data.

7.3.3. Introduction to Data Sending API

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=38>

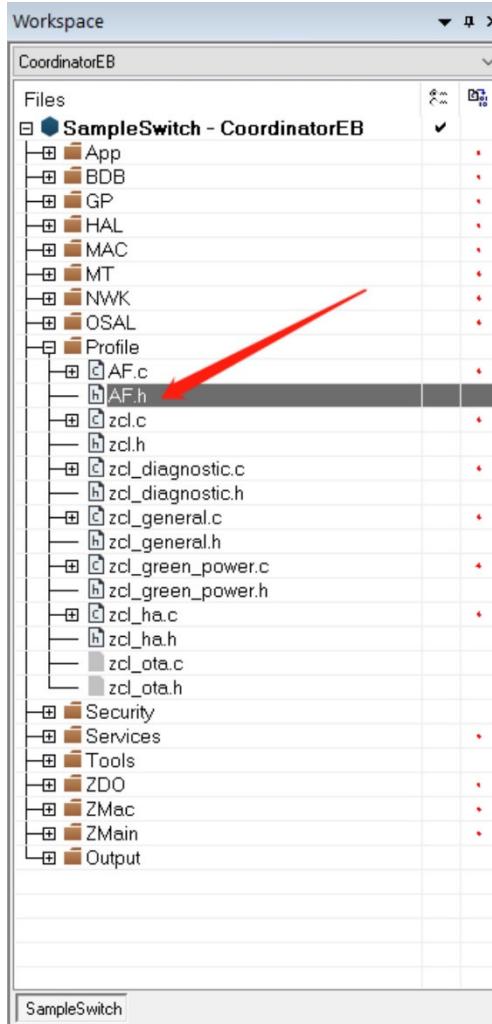
Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

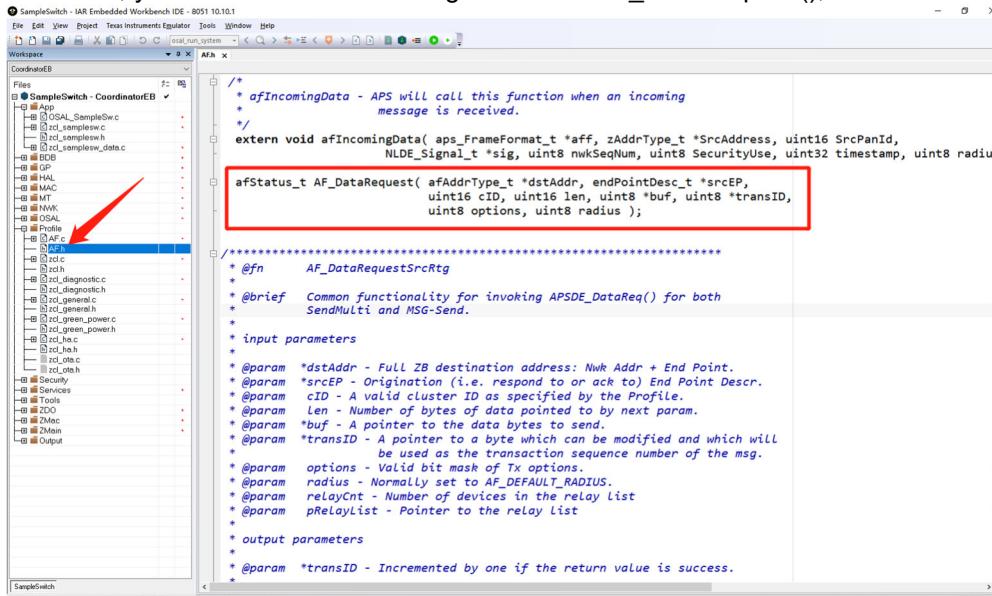
After learning the communication principles, this lesson explains the data sending API of the AF layer.

Data sending API

Open the supporting project and you can find an AF.h file in Profile, as shown in the figure.



In this file, you can find a data sending API called AF_DataRequest(), as shown in the figure



This is the data sending API of the AF layer, and its parameters are described as follows.

```
/*
 * @param dstAddr 目标设备地址，包含网络地址和端点号
 *
 * @param srcEP 发送设备的简单描述符
 *
 * @param cID Cluster ID，后续章节将会详细讲解
 *
 * @param len 待发送数据的长度
 *
 * @param buf 待发送的数据
```

```

* @param transID 传输ID，可以用来给每一次发送的数据包编一个号

* @param options 附加选项，可以用来给这次数据发送添加一些说明

* @param radius 最大的路由跳转级数

*/
afStatus_t AF_DataRequest(
    afAddrType_t *dstAddr,
    endPointDesc_t *srcEP,
    uint16 cID,
    uint16 len,
    uint8 *buf,
    uint8 *transID,
    uint8 options,
    uint8 radius );

```

The SampleSwitch routine in the protocol stack encapsulates the point-to-point communication API, broadcast communication API, and multicast communication API based on this API.

Peer-to-peer communication API

A point-to-point communication API can be found in the zcl_samplesw.c file. The code is as follows:

```

/*
* @param destNwkAddr 目标设备的网络地址
* @param cid Cluster ID，后续课程将会详细讲解
* @param len 数据长度
* @param data 数据内容
*/
static void zclSampleSw_AF_P2P(
    uint16 destNwkAddr,
    uint16 cid,
    uint8 len,
    uint8 *data)
{
    afAddrType_t dstAddr; //寻址信息配置

    static uint8 transferId = 0; //传输ID，是数据包的标识符

    /* Destination */
    dstAddr.addrMode = afAddr16Bit; //设置目标地址模式为16位网络地址，表示使用P2P的通信方式
    dstAddr.addr.shortAddr = destNwkAddr; //目标设备的网络地址
    dstAddr.endPoint = SAMPLESW_ENDPOINT; //目标设备的端点号

    transferId++;
    AF_DataRequest(&dstAddr,
        &sampleSw_TestEp); //已经创建好的简单描述符
}

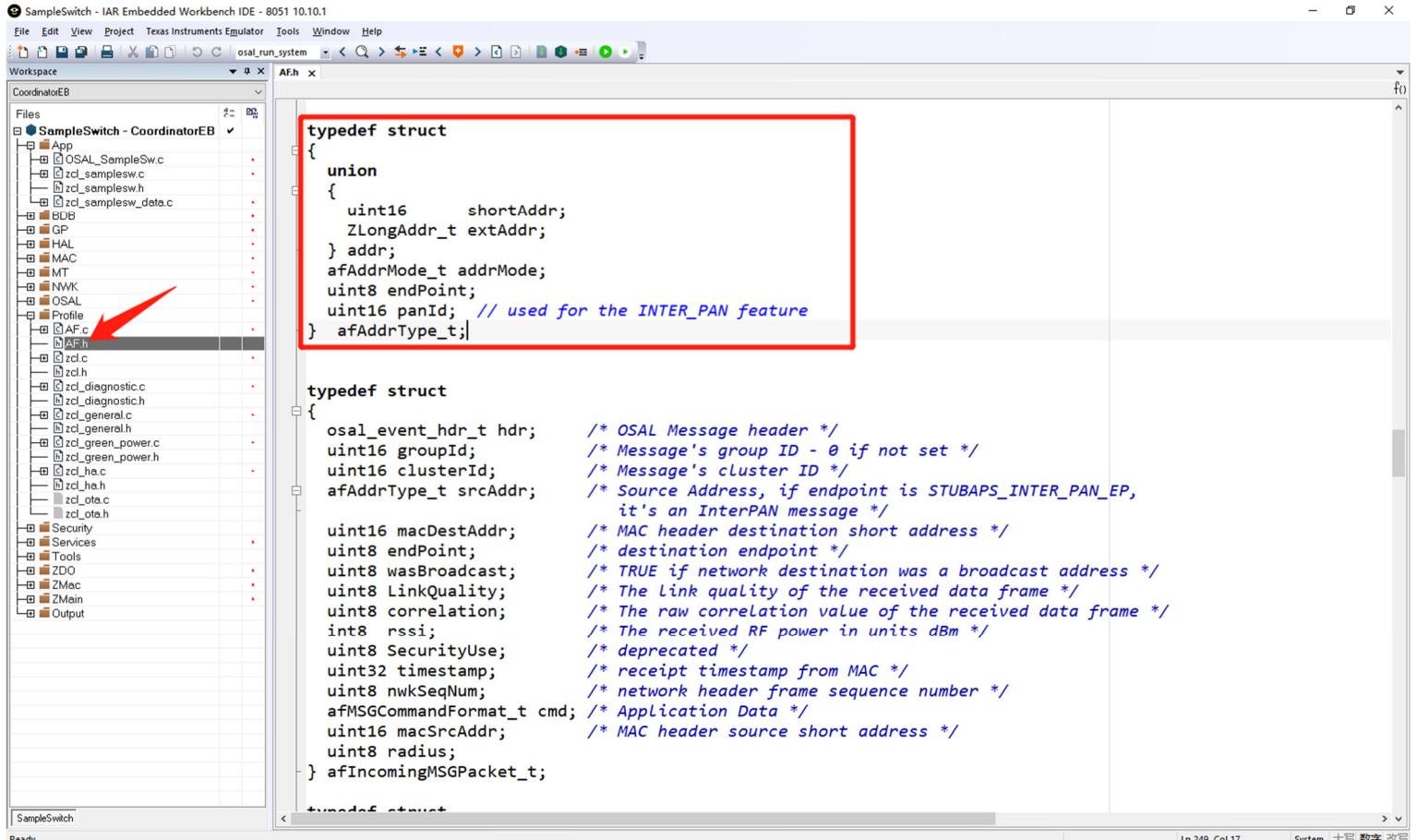
```

```

        cid,
        len,
        data,
        &transferId,
    AF_DISCV_ROUTE, //进行路由扫描操作，用于建立发送数据报文的通信路径。关于这个参数，暂时保持例程默认的代码就可以了
    AF_DEFAULT_RADIUS); //指定最大的路由跳转级数
}

```

This API uses a variable of type afAddrType_t to configure addressing information. This variable is defined as follows:



```

SampleSwitch - IAR Embedded Workbench IDE - 8051 10.10.1
File Edit View Project Texas Instruments Emulator Tools Window Help
osal_run_system Af.h
Workspace
CoordinatorEB
Files
SampleSwitch - CoordinatorEB
App
OSAL_SampleSw.c
zcl_samplesw.c
zcl_samplesw.h
zcl_samplesw_data.c
BDB
GP
HAL
MAC
MT
NWK
OSAL
Profile
AF.c
AF.h
zcl.c
zcl.h
zcl_diagnostic.c
zcl_diagnostic.h
zcl_general.c
zcl_general.h
zcl_green_power.c
zcl_green_power.h
zcl_ha.c
zcl_ha.h
zcl_ota.c
zcl_ota.h
Security
Services
Tools
ZDO
ZMac
ZMain
Output
SampleSwitch
Ready
Ln 249, Col 17 System 大写 数字 改用

```

```

typedef struct
{
    union
    {
        uint16     shortAddr;
        ZLongAddr_t extAddr;
    } addr;
    afAddrMode_t addrMode;
    uint8 endPoint;
    uint16 panId; // used for the INTER_PAN feature
} afAddrType_t;

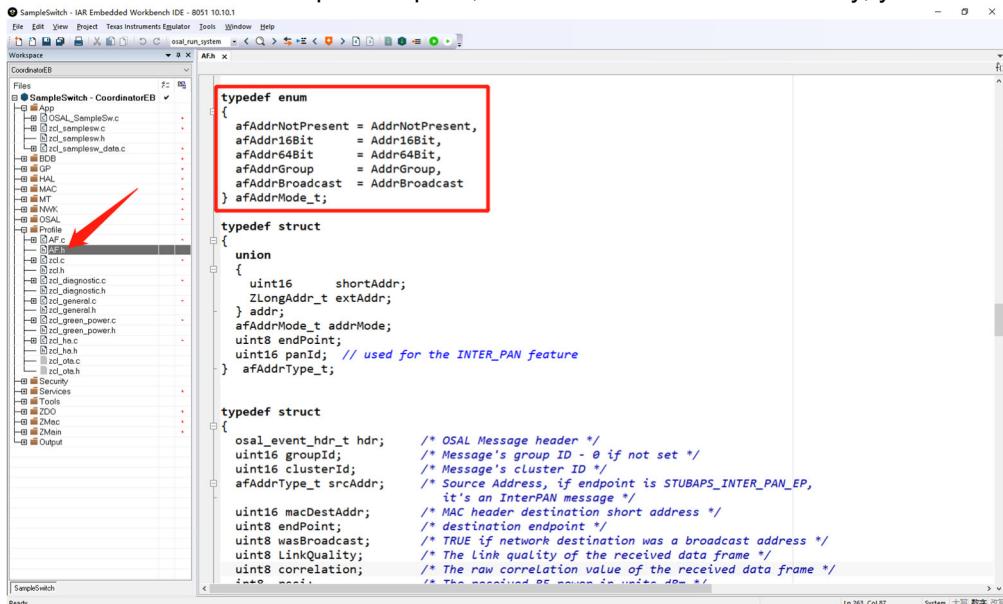
typedef struct
{
    osal_event_hdr_t hdr; /* OSAL Message header */
    uint16 groupId; /* Message's group ID - 0 if not set */
    uint16 clusterId; /* Message's cluster ID */
    afAddrType_t srcAddr; /* Source Address, if endpoint is STUBAPS_INTER_PAN_EP,
                           it's an InterPAN message */

    uint16 macDestAddr; /* MAC header destination short address */
    uint8 endPoint; /* destination endpoint */
    uint8 wasBroadcast; /* TRUE if network destination was a broadcast address */
    uint8 LinkQuality; /* The Link quality of the received data frame */
    uint8 correlation; /* The raw correlation value of the received data frame */
    int8 rssi; /* The received RF power in units dBm */
    uint8 SecurityUse; /* deprecated */
    uint32 timestamp; /* receipt timestamp from MAC */
    uint8 nwkSeqNum; /* network header frame sequence number */
    afMSGCommandFormat_t cmd; /* Application Data */
    uint16 macSrcAddr; /* MAC header source short address */
    uint8 radius;
} afIncomingMSGPacket_t;

typedef struct

```

This API also uses an afAddrMode_t variable to represent the address mode (type), which is used to indicate whether the communication method is point-to-point, broadcast or multicast. Similarly, you can view its definition, the code is as follows:



```

SampleSwitch - IAR Embedded Workbench IDE - 8051 10.10.1
File Edit View Project Texas Instruments Emulator Tools Window Help
osal_run_system Af.h
Workspace
CoordinatorEB
Files
SampleSwitch - CoordinatorEB
App
OSAL_SampleSw.c
zcl_samplesw.c
zcl_samplesw.h
zcl_samplesw_data.c
BDB
GP
HAL
MAC
MT
NWK
OSAL
Profile
AF.c
AF.h
zcl.c
zcl.h
zcl_diagnostic.c
zcl_diagnostic.h
zcl_general.c
zcl_general.h
zcl_green_power.c
zcl_green_power.h
zcl_ha.c
zcl_ha.h
zcl_ota.c
zcl_ota.h
Security
Services
Tools
ZDO
ZMac
ZMain
Output
SampleSwitch
Ready
Ln 263, Col 87 System 大写 数字 改用

```

```

typedef enum
{
    afAddrNotPresent = AddrNotPresent,
    afAddr16Bit = Addr16Bit,
    afAddr64Bit = Addr64Bit,
    afAddrGroup = AddrGroup,
    afAddrBroadcast = AddrBroadcast
} afAddrMode_t;

typedef struct
{
    union
    {
        uint16     shortAddr;
        ZLongAddr_t extAddr;
    } addr;
    afAddrMode_t addrMode;
    uint8 endPoint;
    uint16 panId; // used for the INTER_PAN feature
} afAddrType_t;

typedef struct
{
    osal_event_hdr_t hdr; /* OSAL Message header */
    uint16 groupId; /* Message's group ID - 0 if not set */
    uint16 clusterId; /* Message's cluster ID */
    afAddrType_t srcAddr; /* Source Address, if endpoint is STUBAPS_INTER_PAN_EP,
                           it's an InterPAN message */

    uint16 macDestAddr; /* MAC header destination short address */
    uint8 endPoint; /* destination endpoint */
    uint8 wasBroadcast; /* TRUE if network destination was a broadcast address */
    uint8 LinkQuality; /* The Link quality of the received data frame */
    uint8 correlation; /* The raw correlation value of the received data frame */
    int8 rssi; /* The received RF power in units dBm */
} afIncomingMSGPacket_t;

typedef struct

```

As you can see, afAddrMode_t is an enumeration type variable.

At the end of the zclSampleSw_AF_P2P() function, you can see that AF_DataRequest() is called to send data. Developers can also write their own peer-to-peer communication API based on AF_DataRequest() instead of using the default zclSampleSw_AF_P2P() in this project.

Broadcast Communication API

The broadcast communication API can be found in the zcl_samplesw.c file. The code is as follows:

```
/*
 * @param cid Cluster ID
 * @param len 待发送数据的长度
 * @param *data 待发送数据的内容
 */
static void zclSampleSw_AF_Broadcast(
    uint16 cid,
    uint8 len,
    uint8 *data)
{
    afAddrType_t dstAddr;
    static uint8 transferId = 0;

    /* Destination */
    dstAddr.addrMode = afAddrBroadcast; // 使用广播模式
    dstAddr.addr.shortAddr = 0xFFFF; // 广播地址
    dstAddr.endPoint = SAMPLESW_ENDPOINT; // 目标设备的端点号

    /* Transfer id */
    transferId++;

    /* Send */
    AF_DataRequest(
        &dstAddr,
        &sampleSw_TestEp, // 已经创建好的简单描述符
        cid,
        len,
        data,
        &transferId,
        AF_TX_OPTIONS_NONE,
        AF_DEFAULT_RADIUS ); // 指定了最大的路由跳转级数
}
```

From the definition of the broadcast communication API, we can find that it is similar to the point-to-point communication API. The network address 0xFFFF is used in the broadcast communication API, which means that the data is sent to all devices on the network. But in fact, ZigBee has 3 broadcast addresses, as shown below:

0xFFFF : 广播给该网络中的所有设备

0xFFFD : 只广播给该网络中打开收听功能的设备

Similarly, developers can also write their own broadcast communication API based on AF_DataRequest() instead of using the default zclSampleSw_AF_Broadcast() in this project.

Multicast Communication API

Using the multicast communication API, you can send data only to a specified group of devices. The multicast communication API can be found in the zcl_samplesw.c file. The code is as follows:

```
/*
 * @param groupId 组ID
 * @param cid ClusterID, 后续章节将会详细讲解
 * @param len 待发送数据的长度
 * @param data 待发送数据的内容
 */

static void zclSampleSw_AF_Groupcast(
    uint16 groupId,
    uint16 cid,
    uint8 len,
    uint8 *data)
{
    afAddrType_t dstAddr;
    static uint8 transferId = 0;

    /* Destination */
    dstAddr.addrMode = afAddrGroup;//使用组播通信模式
    dstAddr.addr.shortAddr = groupId;
    dstAddr.endPoint = SAMPLESW_ENDPOINT;//组中的设备的端点号

    /* Transfer id */
    transferId++;

    /* Send */
    AF_DataRequest(&dstAddr,
        &sampleSw_TestEp,//已经创建好的简单描述符
        cid,
        len,
        data,
        &transferId,
        AF_TX_OPTIONS_NONE,
        AF_DEFAULT_RADIUS );//指定最大的路由跳转级数，暂时可忽略
}
```

The above code is relatively simple, just sending data to the specified group ID. The following chapters will explain how to add a device to a specified group through experiments.

7.3.4. ZigBee 3.0 Communication Experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz?p=39>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Experiment Introduction

After learning the AF layer communication API, this chapter will explain how to use these APIs in an experimental way. The experimental equipment includes a coordinator and a router (or terminal). The content is:

1. The router (or terminal) sends data to the coordinator in a point-to-point manner at regular intervals.
2. Add the router (or terminal) to a group.
3. The coordinator sends broadcast data and multicast data periodically.

Device initialization

Open the SampleSwitch project code, and add the following code to the end of the application layer initialization function zclSampleSw_Init() in the zcl_samplesw.c file:

```
#ifdef ZDO_COORDINATOR

bdb_StartCommissioning( BDB_COMMISSIONING_MODE_NWK_FORMATION |
                        BDB_COMMISSIONING_MODE_FINDING_BINDING );

NLME_PermitJoiningRequest(255);

// Broadcast
osal_start_timerEx(zclSampleSw_TaskID,
                    SAMPLEAPP_BROADCAST_EVT,
                    SAMPLEAPP_BROADCAST_PERIOD);

// groupcast
osal_start_timerEx(zclSampleSw_TaskID,
                    SAMPLEAPP_GROUPCAST_EVT,
                    SAMPLEAPP_GROUPCAST_PERIOD);

#else

bdb_StartCommissioning( BDB_COMMISSIONING_MODE_NWK_STEERING |
                        BDB_COMMISSIONING_MODE_FINDING_BINDING );

// Add group
aps_Group_t group = {
    .ID = GROUP_ID,
    .name = "",
};

aps_AddGroup(SAMPLESW_ENDPOINT, &group);
```

```
// P2P
```

```
osal_start_timerEx(zclSampleSw_TaskID,  
    SAMPLEAPP_P2P_EVT,  
    SAMPLEAPP_P2P_PERIOD);
```

```
#endif
```

After adding, it will appear as shown in the picture.

```
// Register for a test endpoint  
afRegister( &sampleSw_TestEp );  
  
#ifdef ZDO_COORDINATOR  
bdb_StartCommissioning( BDB_COMMISSIONING_MODE_NWK_FORMATION |  
    BDB_COMMISSIONING_MODE_FINDING_BINDING );  
  
NLME_PermitJoiningRequest(255);  
  
// Broadcast  
osal_start_timerEx(zclSampleSw_TaskID,  
    SAMPLEAPP_BROADCAST_EVT,  
    SAMPLEAPP_BROADCAST_PERIOD);  
  
// groupcast  
osal_start_timerEx(zclSampleSw_TaskID,  
    SAMPLEAPP_GROUPCAST_EVT,  
    SAMPLEAPP_GROUPCAST_PERIOD);  
  
#else  
  
bdb_StartCommissioning( BDB_COMMISSIONING_MODE_NWK_STEERING |  
    BDB_COMMISSIONING_MODE_FINDING_BINDING );  
  
// Add group  
aps_Group_t group = {  
    .ID = GROUP_ID,  
    .name = "",  
};  
aps_AddGroup(SAMPLESW_ENDPOINT, &group);  
  
// P2P  
osal_start_timerEx(zclSampleSw_TaskID,  
    SAMPLEAPP_P2P_EVT,  
    SAMPLEAPP_P2P_PERIOD);  
#endif  
}
```

Initialization of the coordinator role

In the coordinator role, `bdb_StartCommissioning()` is first called to create a ZigBee network, and then `NLME_PermitJoiningRequest(255)` is called to allow other devices to join the network. Then `osal_start_timerEx()` is called to generate a broadcast event and a multicast event. The events and the corresponding event periods are defined in the `zcl_samplesw.h` file. The code is as follows:

```
//广播事件  
  
#define SAMPLEAPP_BROADCAST_EVT      0x0080  
  
#define SAMPLEAPP_BROADCAST_PERIOD   1000  
  
  
//组播事件  
  
#define SAMPLEAPP_GROUPCAST_EVT      0x0100  
  
#define SAMPLEAPP_GROUPCAST_PERIOD   1000
```

Terminal (or router role) initialization

In the terminal (or router) role, first call `bdb_StartCommissioning()` to join the ZigBee network, and then create a group. The code to create a group is as follows:

```
// Add group  
aps_Group_t group = {  
    .ID = GROUP_ID,  
    .name = "",  
};
```

The definition of `GROUP_ID` is also in the `zcl_samplesw.c` file. The code is as follows:

```
// GroupId  
#define GROUP_ID      21
```

Then call `aps_AddGroup()` to join this group. The code is as follows:

```
aps_AddGroup(  
    SAMPLESW_ENDPOINT,//端点号  
    &group);//待加入的组
```

Finally, call `osal_start_timerEx()` to start a point-to-point communication event. The definition of this event and event cycle is in the `zcl_samplesw.h` file. The code is as follows:

```
// P2P  
  
#define SAMPLEAPP_P2P_EVT      0x0100  
  
#define SAMPLEAPP_P2P_PERIOD    1000
```

Event handling in the coordinator

In the application layer initialization function `zclSampleSw_Init()`, the coordinator role device generates a broadcast event `SAMPLEAPP_BROADCAST_EVT` and a multicast event `SAMPLEAPP_GROUPCAST_EVT`, so the corresponding event processing code needs to be written.

Add the corresponding event processing code in the application layer event processing function `zclSampleSw_event_loop()` in the `zcl_samplesw.c` file:

```
// Broadcast event  
  
if ( events & SAMPLEAPP_BROADCAST_EVT )  
{  
    zclSampleSw_AF_Broadcast(CLUSTER_BROADCAST,  
        10, "Broadcast");  
  
    osal_start_timerEx(zclSampleSw_TaskID,  
        SAMPLEAPP_BROADCAST_EVT,  
        SAMPLEAPP_BROADCAST_PERIOD);  
  
    return ( events ^ SAMPLEAPP_BROADCAST_EVT );  
}  
  
// Groupcast event  
  
if ( events & SAMPLEAPP_GROUPCAST_EVT )  
{  
    zclSampleSw_AF_Groupcast(GROUP_ID,  
        CLUSTER_GROUPCAST,  
        10, "Groupcast");  
  
    osal_start_timerEx(zclSampleSw_TaskID,  
        SAMPLEAPP_GROUPCAST_EVT,  
        SAMPLEAPP_GROUPCAST_PERIOD);  
  
    return ( events ^ SAMPLEAPP_GROUPCAST_EVT );  
}
```

After adding, it will appear as shown in the picture

```
#ifndef ZDO_COORDINATOR /* Coordinator */
// Broadcast event
if ( events & SAMPLEAPP_BROADCAST_EVT )
{
    zclSampleSw_AF_Broadcast(CLUSTER_BROADCAST,
        10, "Broadcast");

    osal_start_timerEx(zclSampleSw_TaskID,
        SAMPLEAPP_BROADCAST_EVT,
        SAMPLEAPP_BROADCAST_PERIOD);

    return ( events ^ SAMPLEAPP_BROADCAST_EVT );
}

// Groupcast event
if ( events & SAMPLEAPP_GROUPCAST_EVT )
{
    zclSampleSw_AF_Groupcast(GROUP_ID,
        CLUSTER_GROUPCAST,
        10, "Groupcast");

    osal_start_timerEx(zclSampleSw_TaskID,
        SAMPLEAPP_GROUPCAST_EVT,
        SAMPLEAPP_GROUPCAST_PERIOD);

    return ( events ^ SAMPLEAPP_GROUPCAST_EVT );
}

else
// Rejoin Event
if ( events & SAMPLEAPP_REJOIN_EVT )
{
    bdb_StartCommissioning(BDB_COMMISSIONING_MODE_NWK_STEERING |
        BDB_COMMISSIONING_MODE_FINDING_BINDING );
}

```

In the broadcast event processing, `zclSampleSw_AF_Broadcast()` is first called to send a broadcast message. The code is as follows:

```
zclSampleSw_AF_Broadcast(
    CLUSTER_BROADCAST,//Cluster ID
    10,//待发送数据的长度
    "Broadcast");//待发送数据的内容
```

Then call `osal_start_timerEx()` again to regenerate a broadcast event.

The processing of multicast events is similar. It also calls `zclSampleSw_AF_Broadcast()` to send a multicast, and then calls `osal_start_timerEx()` to regenerate a multicast event.

`CLUSTER_BROADCAST` and `CLUSTER_GROUPCAST`, plus `CLUSTER_P2P` which will be discussed below, are all Cluster IDs, which are customized by the author. The code is as follows:

```
#define CLUSTER_P2P      0
#define CLUSTER_BROADCAST  1
#define CLUSTER_GROUPCAST  2
```

It can be seen that these three Cluster IDs are essentially a macro definition, and its values are 0, 1, and 2.

Event processing at the terminal (or router)

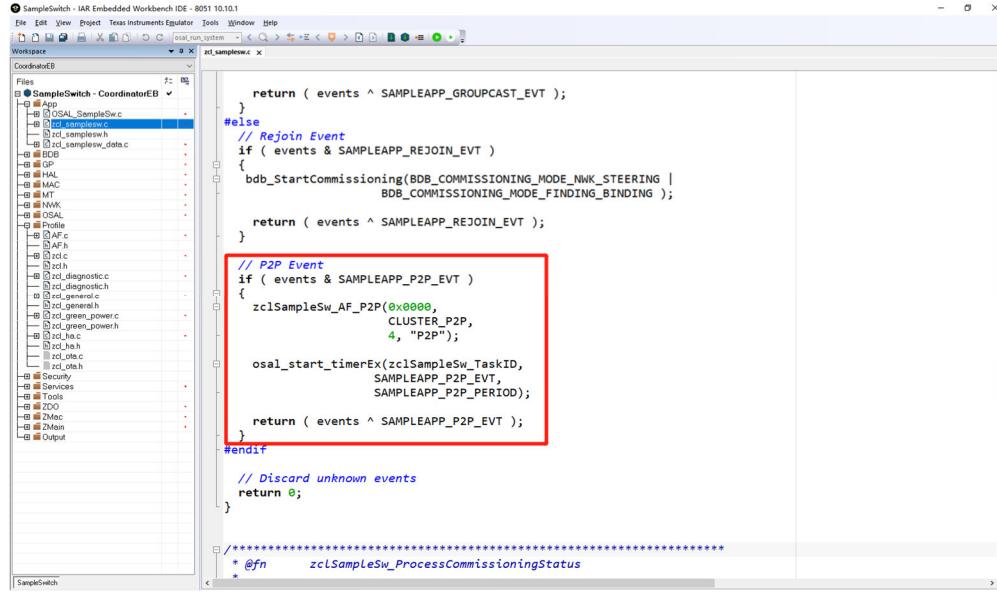
In the application layer initialization function `zclSampleSw_Init()`, the terminal (or router) role device generates a point-to-point communication event, so the corresponding event processing code needs to be written.

Add the corresponding event processing code in the application layer event processing function `zclSampleSw_event_loop()` in the `zcl_samplesw.c` file.

```
// P2P Event
if ( events & SAMPLEAPP_P2P_EVT )
{
    zclSampleSw_AF_P2P(0x0000,
        CLUSTER_P2P,
        4, "P2P");
```

```
osal_start_timerEx(zclSampleSw_TaskID,  
                    SAMPLEAPP_P2P_EVT,  
                    SAMPLEAPP_P2P_PERIOD);  
  
return ( events ^ SAMPLEAPP_P2P_EVT );  
}
```

After adding, it will appear as shown in the picture.



In the point-to-point communication event processing, `zclSampleSw_AF_Broadcast()` is first called to send a point-to-point message. The code is as follows:

zclSampleSw_AF_P2P(

0x0000,//目标设备的网络地址

CLUSTER_P2P,//Cluster ID

4,//待发送数据的长度

"P2P");//待发送数据的内容

The network address 0x0000 is the network address of the coordinator. Similar to the broadcast and multicast event processing, after sending the data, call `osal_start_timerEx()` to regenerate a point-to-point communication event.

Receiving and processing data

We have explained how to send data before. Now let's explain how to receive data. After the ZigBee device is successfully networked, when it receives data, it will generate a system event AF_INCOMING_MSG_CMD to indicate that data has been received.

Open the zcl_samplesw.c file and you can find an application layer event processing function zclSampleSw_event_loop(), as shown in the figure.

```
zclSampleSw_event_loop(uint8, uint16)
```

```
uint16 zclSampleSw_event_loop( uint8 task_id, uint16 events )
```

```
{
```

```
    afIncomingMSGPacket_t *MSGpkt;
```

```
    (void)task_id; // Intentionally unreferenced parameter
```

```
    if ( events & SYS_EVENT_MSG )
```

```
    {
```

```
        while ( (MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive( zclSampleSw_TaskID )) )
```

```
        {
```

```
            switch ( MSGpkt->hdr.event )
```

```
            {
```

```
                case AF_INCOMING_MSG_CMD:
```

```
                    zclSampleSw_AF_RxProc( MSGpkt );
```

```
                    break;
```

```
                default:
```

```
                    break;
```

```
            }
```

```
            // Release the memory
```

```
            osal_msg_deallocate( (uint8 *)MSGpkt );
```

```
        }
```

```
        // return unprocessed events
```

```
        return (events & SYS_EVENT_MSG);
```

It can be seen that the system event processing code already includes the recognition of the system event AF_INCOMING_MSG_CMD. The developer only needs to perform corresponding processing after receiving this event.

The author has defined a data processing function zclSampleSw_AF_RxProc() to process the received data. The code is as follows:

```
/*
 * @param MSGpkt 接收到数据
 */

static void zclSampleSw_AF_RxProc(afIncomingMSGPacket_t *MSGpkt)
{

    /*计数器 · 记录接收到的点对点通信数据包个数*/

    static uint8 p2pCnt = 0;

    /*计数器 · 记录接收到的广播通信数据包个数*/

    static uint8 bcCnt = 0;

    /*计数器 · 记录接收到的组播通信数据包个数*/

    static uint8 gcCnt = 0;

    switch( MSGpkt->clusterId ) // 判断接收到的数据包的Cluster ID, 后续章节将会详细讲解Cluster ID
    {

        case CLUSTER_P2P:

            p2pCnt++; // 接收到P2P数据包, 进行计数

            // 把接收到的数据和计数器的值显示在屏幕上
            HallLCDWriteStringValue((char *)MSGpkt->cmd.Data.p2pCnt, 10, 3);
            break;

        case CLUSTER_BROADCAST:
    }
}
```

```

bcCnt++; // 接收到广播数据包，进行计数

HalLcdWriteStringValue((char *)MSGpkt->cmd.Data,bcCnt,10,3);

break;

case CLUSTER_GROUCAST:

gcCnt++; // 接收到组播数据包，进行计数

HalLcdWriteStringValue((char *)MSGpkt->cmd.Data,gcCnt,10,4);

break;

default:

break;

}
}

```

The code uses the three previously defined Cluster IDs, which can be used to identify data packets sent using different communication methods. The meanings are as follows:

- CLUSTER_P2P: sent in P2P communication mode
- CLUSTER_BROADCAST: sent when broadcasting communication mode
- CLUSTER_GROUCAST: sent in multicast mode

Simulation debugging

1. Select the RouterEB (or EndDeviceEB) role to compile the project and download the firmware to the standard board.
2. Select the CoordinatorEB role to compile the project and download the firmware to the Mini board.
3. After powering on the Mini board, the Mini board will automatically create a network.
4. After powering on the standard board, it will automatically join the network created by the Mini board.
5. Both the Mini board and the standard board will receive the data packet, as shown in the figure.

- Mini Board



Standard plate



7.4. Chapter 4: ZCL Basic Principles

7.4.1. Introduction to ZCL

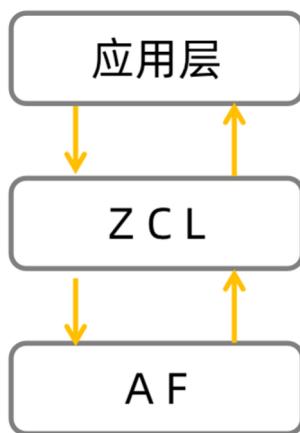
7.4.2. Detailed explanation of ZCL content

7.4.1. Introduction to ZCL

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

The previous chapters explained how to use the AF layer communication API, but now more commonly used is the data communication method based on ZCL. The full name of ZCL is ZigBee Cluster Library, which means ZigBee cluster library in Chinese. The relationship between ZCL and AF is shown in the figure:



There are three layers in the figure, namely AF, ZCL and application layer, which are explained in detail below.

1. AF layer

The AF layer has been explained in detail in the previous chapters. The data communication API of the AF layer can be called to send and receive data between ZigBee devices. However, the biggest problem with directly using the AF layer communication API is that it is difficult for devices developed by different companies to interconnect.

2. Application Layer

Developers focus on developing various functions of the device at this level. In the chapter on AF communication, we directly call the communication API of the AF layer in the application layer to send and receive data.

3. Introduction to ZCL layer

The ZigBee Alliance built the ZCL layer between the AF layer and the application layer. Its biggest role is to achieve interoperability among various ZigBee devices. ZCL defines various application fields (Profile), device types (Device), clusters (Cluster), device attributes and commands of ZigBee devices. These definitions are uniformly customized by the ZigBee Alliance. When developing ZigBee devices, all manufacturers follow these definitions to achieve interoperability.

4. ZCL layer architecture

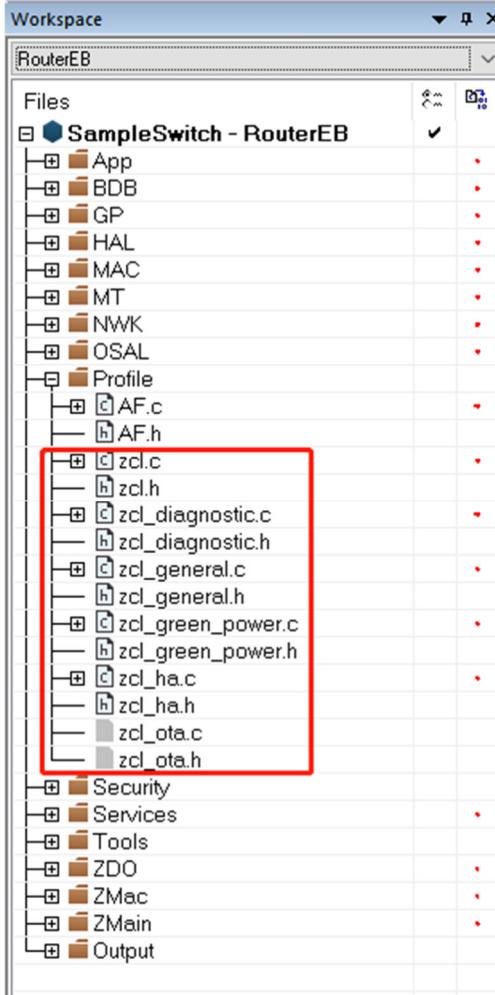
(1) Most of the ZCL content is in the following directories in Z-Stack 3.0:

(2) Part of its contents is shown in the figure

此电脑 > 文件 (H) > 学习资料 > zigbee-master > 4. ZigBee 3.0 网络编程 > 3. 基于AF的数据通信 > Z-Stack 3.0.1 > Components > stack > zcl

名称	修改日期	类型	大小
zcl.c	2022/4/8 21:13	C Source file	156 KB
zcl.h	2022/4/8 21:13	C Header file	51 KB
zcl_appliance_control.c	2022/4/8 21:13	C Source file	30 KB
zcl_appliance_control.h	2022/4/8 21:13	C Header file	19 KB
zcl_appliance_events_alerts.c	2022/4/8 21:13	C Source file	22 KB
zcl_appliance_events_alerts.h	2022/4/8 21:13	C Header file	10 KB
zcl_appliance_identification.h	2022/4/8 21:13	C Header file	7 KB
zcl_appliance_statistics.c	2022/4/8 21:13	C Source file	31 KB
zcl_appliance_statistics.h	2022/4/8 21:13	C Header file	13 KB
zcl_cc.c	2022/4/8 21:13	C Source file	21 KB
zcl_cc.h	2022/4/8 21:13	C Header file	17 KB
zcl_cert_data.c	2022/4/8 21:13	C Source file	7 KB
zcl_closures.c	2022/4/8 21:13	C Source file	130 KB
zcl_closures.h	2022/4/8 21:13	C Header file	78 KB
zcl_diagnostic.c	2022/4/8 21:13	C Source file	15 KB
zcl_diagnostic.h	2022/4/8 21:13	C Header file	10 KB
zcl_electrical_measurement.c	2022/4/8 21:13	C Source file	25 KB
zcl_electrical_measurement.h	2022/4/8 21:13	C Header file	38 KB
zcl_general.c	2022/4/8 21:13	C Source file	127 KB
zcl_general.h	2022/4/8 21:13	C Header file	101 KB
zcl_gp.h	2022/4/8 21:13	C Header file	57 KB
zcl_green_power.c	2022/4/8 21:13	C Source file	31 KB
zcl_green_power.h	2022/4/8 21:13	C Header file	46 KB
zcl_hvac.c	2022/4/8 21:13	C Source file	38 KB
zcl_hvac.h	2022/4/8 21:13	C Header file	35 KB
zcl_key_establish.c	2022/4/8 21:13	C Source file	119 KB
zcl_key_establish.h	2022/4/8 21:13	C Header file	9 KB
zcl_lighting.c	2022/4/8 21:13	C Source file	51 KB
zcl_lighting.h	2022/4/8 21:13	C Header file	35 KB
zcl_ll.c	2022/4/8 21:13	C Source file	58 KB
zcl_ll.h	2022/4/8 21:13	C Header file	31 KB
zcl_meter_identification.h	2022/4/8 21:13	C Header file	6 KB
zcl_ms.c	2022/4/8 21:13	C Source file	14 KB
zcl_ms.h	2022/4/8 21:13	C Header file	11 KB
zcl_ota.c	2022/4/8 21:13	C Source file	88 KB

(3) A device generally uses only part of the content in ZCL. For example, the SampleSwitch project that this book has been explaining uses these contents, as shown in the figure.



5. Code Analysis

(1) Let's briefly analyze the code in ZCL. Open the zcl.h file and find the ZCL layer data sending function zcl_SendCommand(). The code is as follows:

```
*****  
* @fn    zcl_SendCommand  
*  
* @brief Used to send Profile and Cluster Specific Command messages.  
*  
*      NOTE: The calling application is responsible for incrementing  
*            the Sequence Number.  
*  
* @param srcEp - source endpoint  
* @param destAddr - destination address  
* @param clusterID - cluster ID  
* @param cmd - command ID  
* @param specific - whether the command is Cluster Specific  
* @param direction - client/server direction of the command  
* @param disableDefaultRsp - disable Default Response command  
* @param manuCode - manufacturer code for proprietary extensions to a profile  
* @param seqNumber - identification number for the transaction  
* @param cmdFormatLen - length of the command to be sent  
* @param cmdFormat - command to be sent  
*  
* @return ZSuccess if OK  
*/  
  
ZStatus_t zcl_SendCommand( uint8 srcEP, afAddrType_t *destAddr,  
                           uint16 clusterID, uint8 cmd, uint8 specific, uint8 direction,  
                           uint8 disableDefaultRsp, uint16 manuCode, uint8 seqNum,  
                           uint16 cmdFormatLen, uint8 *cmdFormat )  
{  
    endPointDesc_t *epDesc;  
    zclFrameHdr_t hdr;  
    uint8 *msgBuf;  
    uint16 msgLen;  
    uint8 *pBuf;  
    uint8 options;  
    ZStatus_t status;  
  
    epDesc = afFindEndPointDesc( srcEP );  
    if ( epDesc == NULL )  
    {  
        return ( ZInvalidParameter ); // EMBEDDED RETURN  
    }  
  
    #if defined ( INTER_PAN )  
    if ( StubAPS_InterPan( destAddr->panId, destAddr->endPoint ) )
```

```

{
    options = AF_TX_OPTIONS_NONE;
}
else
#endif
{
    options = zclGetClusterOption( srcEP, clusterID );

    // The cluster might not have been defined to use security but if this message
    // is in response to another message that was using APS security this message
    // will be sent with APS security

    if ( !( options & AF_EN_SECURITY ) )
    {
        afIncomingMSGPacket_t *origPkt = zcl_getRawAFMsg();

        if ( ( origPkt != NULL ) && ( origPkt->SecurityUse == TRUE ) )
        {
            options |= AF_EN_SECURITY;
        }
    }
}

zcl_memset( &hdr, 0, sizeof( zclFrameHdr_t ) );

// Not Profile wide command (like READ, WRITE)
if ( specific )
{
    hdr.fc.type = ZCL_FRAME_TYPE_SPECIFIC_CMD;
}
else
{
    hdr.fc.type = ZCL_FRAME_TYPE_PROFILE_CMD;
}

if ( ( epDesc->simpleDesc == NULL ) ||
     ( zcl_DeviceOperational( srcEP, clusterID, hdr.fc.type,
                               cmd, epDesc->simpleDesc->AppProfId ) == FALSE ) )
{
    return ( ZFailure ); // EMBEDDED RETURN
}

// Fill in the Manufacturer Code
if ( manuCode != 0 )
{
    hdr.fc.manuSpecific = 1;
}

```

```

    hdr.manuCode = manuCode;
}

// Set the Command Direction

if ( direction )
{
    hdr.fc.direction = ZCL_FRAME_SERVER_CLIENT_DIR;
}
else
{
    hdr.fc.direction = ZCL_FRAME_CLIENT_SERVER_DIR;
}

// Set the Disable Default Response field

if ( disableDefaultRsp )
{
    hdr.fc.disableDefaultRsp = 1;
}
else
{
    hdr.fc.disableDefaultRsp = 0;
}

// Fill in the Transaction Sequence Number

hdr.transSeqNum = seqNum;

// Fill in the command

hdr.commandID = cmd;

// calculate the needed buffer size

msgLen = zclCalcHdrSize( &hdr );
msgLen += cmdFormatLen;

// Allocate the buffer needed

msgBuf = zcl_mem_alloc( msgLen );
if ( msgBuf != NULL )
{
    // Fill in the ZCL Header

    pBuf = zclBuildHdr( &hdr, msgBuf );

    // Fill in the command frame

    zcl_memcpy( pBuf, cmdFormat, cmdFormatLen );

    status = AF_DataRequest( destAddr, epDesc, clusterID, msgLen, msgBuf,
        &zcl_TransID, options, AF_DEFAULT_RADIUS );
}

```

```

zcl_mem_free( msgBuf );
}

else
{
    status = ZMemError;
}

return ( status );
}

```

It can be seen that in the zcl_SendCommand() function, the AF layer's data sending function AF_DataRequest() is actually called to send data.

(2) There is also a function called zcl_event_loop() in the zcl.c file. In this function, we can see the reception and processing of AF layer data. The code is shown in the figure.

```

File Edit User Object Texas Instruments Equator Tools Windows Help
File Edit User Object Texas Instruments Equator Tools Windows Help
Workspace
RouteEB
File
SampleSwitch - RouterEB
|-> App
|-> BDB
|-> GP
|-> HAN
|-> MAC
|-> MT
|-> PAN
|-> ODSAL
|-> Profile
|   |-> AF.c
|   |-> BDB.h
|   |-> HAN.h
|   |-> MAC.h
|   |-> MT.h
|   |-> PAN.h
|   |-> ODSAL.h
|-> zcl
|-> zcl_diagnostic.c
|-> zcl_diagnostic.h
|-> zcl_general.c
|-> zcl_general.h
|-> zcl_greengpio_power.c
|-> zcl_greengpio_power.h
|-> zcl_ha.c
|-> zcl_ha.h
|-> zcl_oob.h
|-> Security
|-> Utilities
|-> Tools
|-> ZDO
|-> ZMac
|-> ZMain
|-> Output

```

```

zcl_event_looplnt( uint16 )
*****
* @fn      zcl_eventLoop
* @brief   Event Loop Processor for zcl.
* @param   task_id - task id
* @param   events - event bitmap
* @return  unprocessed events
*/
uint16 zcl_event_loop( uint8 task_id, uint16 events )
{
    uint8 *msgPtr;
    (void)task_id; // Intentionally unreferenced parameter
    if ( events & SYS_EVENT_MSG )
    {
        msgPtr = osal_msg_receive( zcl_TaskID );
        while ( msgPtr != NULL )
        {
            uint8 dealloc = TRUE;
            if ( *msgPtr == AF_INCOMING_MSG_CMD )
            {
                zcl_ProcessMessageMSG( (afIncomingMSGPacket_t *)msgPtr );
            }
            else
            {
                uint8 taskID;
                taskID = zcl_getExternalFoundationHandler( (afIncomingMSGPacket_t *)msgPtr );
                if ( taskID != TASK_NO_TASK )
                {
                    // send it to another task to process.
                    osal_msg_send( taskID, msgPtr );
                }
            }
            osal_msg_free( msgPtr );
            msgPtr = osal_msg_receive( zcl_TaskID );
        }
    }
}

```

The above analysis also confirms that the bottom layer of the ZCL layer is the AF layer.

7.4.2. Detailed explanation of ZCL content

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

- Generally, self-study is the main method.
- You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
- Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

ZCL mainly includes application field (Profile), device type (Device), cluster (Cluster), attribute (Attribute) and command (Command).

1. Application Areas (Profile)

(1) Profile has been mentioned many times in the previous chapters, so here is a formal introduction. In the ZigBee protocol, Profile refers to the application field. The ZigBee protocol defines multiple profiles, such as ZHA (ZigBee Home Automation) for home automation and ZLL (ZigBee Light Link) for lighting. The smart socket used in this article is a product belonging to the ZHA profile.

(2) The ZigBee protocol **assigns a fixed ID to each type of Profile**. Open the Profile/zcl_ha.h file to find the ID value of the ZHA Profile. The code is as follows:

#define ZCL_HA_PROFILE_ID 0x0104

(3) Open the Components\stack\zcl\zcl_ll.h file in the protocol stack and find the ID value of the ZLL Profile. The code is as follows:

#define ZLL_PROFILE_ID 0xc05e

(4) You can also view other Profile IDs in this way. In the chapter on simple descriptors, we mentioned that the AppProfil field represents the application scenario to which this simple descriptor belongs, and its value is actually these Profile IDs.

2. Device Type

(1) Each Profile can contain multiple types of devices. For example, the ZHA Profile contains devices such as smart sockets, temperature and humidity sensors, and curtain controllers. Each type of device is assigned an ID, called a Device ID, and in the same Profile, each Device ID is different.

(3) In ZigBee 3.0, the device types in the ZHA Profile are divided into five categories: Generic, Lighting, Closures, HVAC (Heating Ventilation and Air Conditioning) and IAS (Intruder Alarm Systems).

Device	Device ID
On/Off Switch	0x0000
Level Control Switch	0x0001
On/Off Output	0x0002
Level Controllable Output	0x0003
Scene Selector	0x0004
Configuration Tool	0x0005
Remote Control	0x0006
Combined Interface	0x0007
Range Extender	0x0008
Mains Power Outlet	0x0009
Door Lock	0x000A
Door Lock Controller	0x000B
Simple Sensor	0x000C
Consumption Awareness Device	0x000D
Home Gateway	0x0050
Smart plug	0x0051
White Goods	0x0052
Meter Interface	0x0053
Reserved	0x0060–0x00FF

Lighting	On/Off Light	0x0100
	Dimmable Light	0x0101
	Color Dimmable Light	0x0102
	On/Off Light Switch	0x0103
	Dimmer Switch	0x0104
	Color Dimmer Switch	0x0105
	Light Sensor	0x0106
	Occupancy Sensor	0x0107
	Reserved	0x0108 – 0x1FF
Closures	Shade ®	0x0200
	Shade Controller	0x0201
	Window Covering Device	0x0202
	Window Covering Controller	0x0203
	Reserved	0x0204 – 0x2FF
HVAC	Heating/Cooling Unit	0x0300
	Thermostat	0x0301
	Temperature Sensor	0x0302
	Pump	0x0303
	Pump Controller	0x0304
	Pressure Sensor	0x0305
	Flow Sensor	0x0306
	Mini Split AC	0x0307
	Reserved	0x0308 - 0x3FF
Intruder Alarm Systems	IAS Control and Indicating Equipment	0x0400
	IAS Ancillary Control Equipment	0x0401
	IAS Zone	0x0402
	IAS Warning Device	0x0403
	Reserved	0x0404-0xFFFF

(3) The ZHA-related source files are zcl_ha.h and zcl_ha.c in the Profile project directory. Open the zcl_ha.h file to find the Device IDs of all the above device types. The code is as follows:

(3A) Generic

```
// Generic Device IDs

#define ZCL_HA_DEVICEID_ON_OFF_SWITCH          0x0000
#define ZCL_HA_DEVICEID_LEVEL_CONTROL_SWITCH    0x0001
#define ZCL_HA_DEVICEID_ON_OFF_OUTPUT           0x0002
#define ZCL_HA_DEVICEID_LEVEL_CONTROLLABLE_OUTPUT 0x0003
#define ZCL_HA_DEVICEID_SCENE_SELECTOR          0x0004
#define ZCL_HA_DEVICEID_CONFIGURATION_TOOL      0x0005
#define ZCL_HA_DEVICEID_REMOTE_CONTROL          0x0006
#define ZCL_HA_DEVICEID_COMBINED_INTERFACE      0x0007
```

```

#define ZCL_HA_DEVICEID_RANGE_EXTENDER          0x0008
#define ZCL_HA_DEVICEID_MAINS_POWER_OUTLET      0x0009
#define ZCL_HA_DEVICEID_DOOR_LOCK               0x000A
#define ZCL_HA_DEVICEID_DOOR_LOCK_CONTROLLER    0x000B
#define ZCL_HA_DEVICEID_SIMPLE_SENSOR           0x000C
#define ZCL_HA_DEVICEID_CONSUMPTION_AWARENESS_DEVICE 0x000D
#define ZCL_HA_DEVICEID_HOME_GATEWAY            0x0050
#define ZCL_HA_DEVICEID_SMART_PLUG              0x0051
#define ZCL_HA_DEVICEID_WHITE_GOODS             0x0052
#define ZCL_HA_DEVICEID_METER_INTERFACE         0x0053

```

(3B) Lighting

```

// Lighting Device IDs

#define ZCL_HA_DEVICEID_ON_OFF_LIGHT           0x0100
#define ZCL_HA_DEVICEID_DIMMABLE_LIGHT         0x0101
#define ZCL_HA_DEVICEID_COLORED_DIMMABLE_LIGHT 0x0102
#define ZCL_HA_DEVICEID_ON_OFF_LIGHT_SWITCH    0x0103
#define ZCL_HA_DEVICEID_DIMMER_SWITCH          0x0104
#define ZCL_HA_DEVICEID_COLOR_DIMMER_SWITCH    0x0105
#define ZCL_HA_DEVICEID_LIGHT_SENSOR           0x0106
#define ZCL_HA_DEVICEID_OCCUPANCY_SENSOR       0x0107

```

(3C) Closures

```

// Closures Device IDs

#define ZCL_HA_DEVICEID SHADE                0x0200
#define ZCL_HA_DEVICEID SHADE_CONTROLLER      0x0201
#define ZCL_HA_DEVICEID WINDOW_COVERING_DEVICE 0x0202
#define ZCL_HA_DEVICEID WINDOW_COVERING_CONTROLLER 0x0203

```

(3D) HVAC

```

// HVAC Device IDs

#define ZCL_HA_DEVICEID HEATING_COOLING_UNIT 0x0300
#define ZCL_HA_DEVICEID THERMOSTAT            0x0301
#define ZCL_HA_DEVICEID TEMPERATURE_SENSOR    0x0302
#define ZCL_HA_DEVICEID PUMP                 0x0303
#define ZCL_HA_DEVICEID PUMP_CONTROLLER      0x0304
#define ZCL_HA_DEVICEID PRESSURE_SENSOR      0x0305
#define ZCL_HA_DEVICEID FLOW_SENSOR          0x0306
#define ZCL_HA_DEVICEID MINI_SPLIT_AC        0x0307

```

(3E) IAS

```

// Intruder Alarm Systems (IAS) Device IDs

#define ZCL_HA_DEVICEID IAS_CONTROL_INDICATING_EQUIPMENT 0x0400
#define ZCL_HA_DEVICEID IAS_ANCILLARY_CONTROL_EQUIPMENT   0x0401
#define ZCL_HA_DEVICEID IAS_ZONE                          0x0402
#define ZCL_HA_DEVICEID IAS_WARNING_DEVICE               0x0403

```

(4) The ZLL-related source files are zcl_ll.h and zcl_ll.c in the project directory Profile. Open the zcl_ll.h file and you can find the Device ID defined by the ZLL Profile (domain). The code is as follows:

```
// ZLL Basic Lighting Device IDs
#define ZLL_DEVICEID_ON_OFF_LIGHT          0x0000
#define ZLL_DEVICEID_ON_OFF_PLUG_IN_UNIT    0x0010
#define ZLL_DEVICEID_DIMMABLE_LIGHT         0x0100
#define ZLL_DEVICEID_DIMMABLE_PLUG_IN_UNIT  0x0110

// ZLL Color Lighting Device IDs
#define ZLL_DEVICEID_COLOR_LIGHT           0x0200
#define ZLL_DEVICEID_EXTENDED_COLOR_LIGHT   0x0210
#define ZLL_DEVICEID_COLOR_TEMPERATURE_LIGHT 0x0220

// ZLL Lighting Remotes Device IDs
#define ZLL_DEVICEID_COLOR_CONTROLLER       0x0800
#define ZLL_DEVICEID_COLOR_SCENE_CONTROLLER 0x0810
#define ZLL_DEVICEID_NON_COLOR_CONTROLLER   0x0820
#define ZLL_DEVICEID_NON_COLOR_SCENE_CONTROLLER 0x0830
#define ZLL_DEVICEID_CONTROL_BRIDGE        0x0840
#define ZLL_DEVICEID_ON_OFF_SENSOR         0x0850
```

(5) Using this method, you can also view the Device IDs that can be used in other Profiles. In the chapter on simple descriptors, we mentioned that the AppDeviceId field represents the type of the device, and its value is actually the Device ID value.

3. Cluster

(1) What is the function of Cluster? Readers can keep this question in mind and continue reading.

(2) In the chapter on AF communication, the author customized three Cluster IDs, each of which represents a Cluster, used to indicate different communication types. These three Clusters are customized by the author and are not defined by the ZigBee Alliance. Therefore, only the code written by the author can understand their meaning, and other developers do not know them. Therefore, these three Clusters can be understood as private.

(3) However, the ZigBee Alliance has defined many standard clusters that can be used by all developers. These clusters can be understood as public. Each cluster is assigned a unique ID, called a cluster ID. These clusters can be divided into two categories: clusters belonging to a specific profile, and general clusters that are shared by all profiles and do not belong to a specific profile.

(4) The definition of Cluster ID is in the file zcl.h. This article lists some Clusters. The code is as follows:

(4A) General Clusters

```
// General Clusters
#define ZCL_CLUSTER_ID_GEN_BASIC           0x0000
#define ZCL_CLUSTER_ID_GEN_POWER_CFG        0x0001
#define ZCL_CLUSTER_ID_GEN_DEVICE_TEMP_CONFIG 0x0002
#define ZCL_CLUSTER_ID_GEN_IDENTIFY         0x0003
#define ZCL_CLUSTER_ID_GEN_GROUPS          0x0004
#define ZCL_CLUSTER_ID_GEN_SCENES          0x0005
#define ZCL_CLUSTER_ID_GEN_ON_OFF          0x0006
#define ZCL_CLUSTER_ID_GEN_ON_OFF_SWITCH_CONFIG 0x0007
#define ZCL_CLUSTER_ID_GEN_LEVEL_CONTROL    0x0008
```

```

#define ZCL_CLUSTER_ID_GEN_ALARMS          0x0009
#define ZCL_CLUSTER_ID_GEN_TIME            0x000A
#define ZCL_CLUSTER_ID_GEN_LOCATION        0x000B
#define ZCL_CLUSTER_ID_GEN_ANALOG_INPUT_BASIC 0x000C
#define ZCL_CLUSTER_ID_GEN_ANALOG_OUTPUT_BASIC 0x000D
#define ZCL_CLUSTER_ID_GEN_ANALOG_VALUE_BASIC 0x000E
#define ZCL_CLUSTER_ID_GEN_BINARY_INPUT_BASIC 0x000F
#define ZCL_CLUSTER_ID_GEN_BINARY_OUTPUT_BASIC 0x0010
#define ZCL_CLUSTER_ID_GEN_BINARY_VALUE_BASIC 0x0011
#define ZCL_CLUSTER_ID_GEN_MULTISTATE_INPUT_BASIC 0x0012
#define ZCL_CLUSTER_ID_GEN_MULTISTATE_OUTPUT_BASIC 0x0013
#define ZCL_CLUSTER_ID_GEN_MULTISTATE_VALUE_BASIC 0x0014
#define ZCL_CLUSTER_ID_GEN_COMMISSIONING     0x0015
#define ZCL_CLUSTER_ID_GEN_PARTITION         0x0016

#define ZCL_CLUSTER_ID_GEN_POWER_PROFILE      0x001A
#define ZCL_CLUSTER_ID_GEN_APPLIANCE_CONTROL   0x001B

#define ZCL_CLUSTER_ID_GEN_POLL_CONTROL       0x0020

```

(4B) Retail Clusters

```

// Retail Clusters

#define ZCL_CLUSTER_ID_MOBILE_DEVICE_CONFIGURATION 0x0022
#define ZCL_CLUSTER_ID_NEIGHBOR_CLEANING           0x0023
#define ZCL_CLUSTER_ID_NEAREST_GATEWAY             0x0024

```

(4C) Closures Clusters

```

// Closures Clusters

#define ZCL_CLUSTER_ID_CLOSURES_SHADE_CONFIG      0x0100
#define ZCL_CLUSTER_ID_CLOSURES_DOOR_LOCK          0x0101
#define ZCL_CLUSTER_ID_CLOSURES_WINDOW_COVERING    0x0102

```

(4D) HVAC Clusters

```

// HVAC Clusters

#define ZCL_CLUSTER_ID_HVAC_PUMP_CONFIG_CONTROL   0x0200
#define ZCL_CLUSTER_ID_HVAC_THERMOSTAT             0x0201
#define ZCL_CLUSTER_ID_HVAC_FAN_CONTROL            0x0202
#define ZCL_CLUSTER_ID_HVAC_DIHUMIDIFICATION_CONTROL 0x0203
#define ZCL_CLUSTER_ID_HVAC_USER_INTERFACE_CONFIG   0x0204

```

(4E) Lighting Clusters

```

// Lighting Clusters

#define ZCL_CLUSTER_ID_LIGHTING_COLOR_CONTROL     0x0300
#define ZCL_CLUSTER_ID_LIGHTING_BALAST_CONFIG      0x0301

```

(4F) Measurement and Sensing Clusters

```
// Measurement and Sensing Clusters
```

```
#define ZCL_CLUSTER_ID_MS_ILLUMINANCE_MEASUREMENT      0x0400
#define ZCL_CLUSTER_ID_MS_ILLUMINANCE_LEVEL_SENSING_CONFIG 0x0401
#define ZCL_CLUSTER_ID_MS_TEMPERATURE_MEASUREMENT        0x0402
#define ZCL_CLUSTER_ID_MS_PRESSURE_MEASUREMENT          0x0403
#define ZCL_CLUSTER_ID_MS_FLOW_MEASUREMENT              0x0404
#define ZCL_CLUSTER_ID_MS_RELATIVE_HUMIDITY            0x0405
#define ZCL_CLUSTER_ID_MS_OCCUPANCY_SENSING           0x0406
```

(4G) Security and Safety (SS) Clusters

```
// Security and Safety (SS) Clusters
```

```
#define ZCL_CLUSTER_ID_SS_IAS_ZONE                  0x0500
#define ZCL_CLUSTER_ID_SS_IAS_ACE                   0x0501
#define ZCL_CLUSTER_ID_SS_IAS_WD                  0x0502
```

(4H) Protocol Interfaces

```
// Protocol Interfaces
```

```
#define ZCL_CLUSTER_ID_PI_GENERIC_TUNNEL          0x0600
#define ZCL_CLUSTER_ID_PI_BACNET_PROTOCOL_TUNNEL    0x0601
#define ZCL_CLUSTER_ID_PI_ANALOG_INPUT_BACNET_REG   0x0602
#define ZCL_CLUSTER_ID_PI_ANALOG_INPUT_BACNET_EXT   0x0603
#define ZCL_CLUSTER_ID_PI_ANALOG_OUTPUT_BACNET_REG  0x0604
#define ZCL_CLUSTER_ID_PI_ANALOG_OUTPUT_BACNET_EXT  0x0605
#define ZCL_CLUSTER_ID_PI_ANALOG_VALUE_BACNET_REG   0x0606
#define ZCL_CLUSTER_ID_PI_ANALOG_VALUE_BACNET_EXT   0x0607
#define ZCL_CLUSTER_ID_PI_BINARY_INPUT_BACNET_REG   0x0608
#define ZCL_CLUSTER_ID_PI_BINARY_INPUT_BACNET_EXT   0x0609
#define ZCL_CLUSTER_ID_PI_BINARY_OUTPUT_BACNET_REG  0x060A
#define ZCL_CLUSTER_ID_PI_BINARY_OUTPUT_BACNET_EXT  0x060B
#define ZCL_CLUSTER_ID_PI_BINARY_VALUE_BACNET_REG   0x060C
#define ZCL_CLUSTER_ID_PI_BINARY_VALUE_BACNET_EXT   0x060D
#define ZCL_CLUSTER_ID_PI_MULTISTATE_INPUT_BACNET_REG 0x060E
#define ZCL_CLUSTER_ID_PI_MULTISTATE_INPUT_BACNET_EXT 0x060F
#define ZCL_CLUSTER_ID_PI_MULTISTATE_OUTPUT_BACNET_REG 0x0610
#define ZCL_CLUSTER_ID_PI_MULTISTATE_OUTPUT_BACNET_EXT 0x0611
#define ZCL_CLUSTER_ID_PI_MULTISTATE_VALUE_BACNET_REG 0x0612
#define ZCL_CLUSTER_ID_PI_MULTISTATE_VALUE_BACNET_EXT 0x0613
#define ZCL_CLUSTER_ID_PI_11073_PROTOCOL_TUNNEL     0x0614
#define ZCL_CLUSTER_ID_PI_ISO7818_PROTOCOL_TUNNEL    0x0615
#define ZCL_CLUSTER_ID_PI_RETAIL_TUNNEL             0x0617
```

(4I) Smart Energy (SE) Clusters

```
// Smart Energy (SE) Clusters
```

```
#define ZCL_CLUSTER_ID_SE_PRICE                 0x0700
#define ZCL_CLUSTER_ID_SE_DRLC                  0x0701
#define ZCL_CLUSTER_ID_SE_METERING            0x0702
```

```

#define ZCL_CLUSTER_ID_SE_MESSAGING          0x0703
#define ZCL_CLUSTER_ID_SE_TUNNELING          0x0704
#define ZCL_CLUSTER_ID_SE_PREPAYMENT          0x0705
#define ZCL_CLUSTER_ID_SE_ENERGY_MGMT         0x0706
#define ZCL_CLUSTER_ID_SE_CALENDAR            0x0707
#define ZCL_CLUSTER_ID_SE_DEVICE_MGMT          0x0708
#define ZCL_CLUSTER_ID_SE_EVENTS               0x0709
#define ZCL_CLUSTER_ID_SE_MDU_PAIRING          0x070A

#define ZCL_CLUSTER_ID_SE_KEY_ESTABLISHMENT    0x0800

#define ZCL_CLUSTER_ID_TELECOMMUNICATIONS_INFORMATION 0x0900
#define ZCL_CLUSTER_ID_TELECOMMUNICATIONS_CHATTING     0x0904
#define ZCL_CLUSTER_ID_TELECOMMUNICATIONS_VOICE_OVER_ZIGBEE 0x0905

#define ZCL_CLUSTER_ID_HA_APPLIANCE_IDENTIFICATION 0x0B00
#define ZCL_CLUSTER_ID_HA_METER_IDENTIFICATION      0x0B01
#define ZCL_CLUSTER_ID_HA_APPLIANCE_EVENTS_ALERTS    0x0B02
#define ZCL_CLUSTER_ID_HA_APPLIANCE_STATISTICS       0x0B03
#define ZCL_CLUSTER_ID_HA_ELECTRICAL_MEASUREMENT    0x0B04
#define ZCL_CLUSTER_ID_HA_DIAGNOSTIC                 0x0B05

```

(4J) Light Link cluster

```

// Light Link cluster
#define ZCL_CLUSTER_ID_TOUCHLINK              0x1000

```

(5) Each Cluster can contain multiple specific attributes and commands.

4. Attribute

(1) Similar to the attributes in object-oriented programming, the attributes here are used to describe the characteristics of a certain type of thing. For example, the attributes of a tiger include gender, age, and weight. In addition to pre-defining multiple clusters, the ZigBee Alliance also pre-defines a set of corresponding attributes for each cluster for developers to use. The attributes that a cluster should contain can be defined by the developer.

(2) Open the zcl_samplesw_data.c file and find the code as shown in the figure.

The screenshot shows a code editor with the title "zcl_samplesw_data.c" and a tab labeled "属性列表!". The code defines an array of attribute records for a cluster. Three specific elements are highlighted with red boxes and arrows pointing to annotations:

- Line 116:** `CONST zclAttrRec_t zclSampleSw_Attrs[] =`
- Line 120:** `ZCL_CLUSTER_ID_GEN_BASIC, Cluster: Basic`
 { // Attribute record
 ATTRID_BASIC_ZCL_VERSION,
 ZCL_DATATYPE_UINT8,
 ACCESS_CONTROL_READ,
 (void *)&zclSampleSw_ZCLVersion
 Annotations: "ZCL版本号" (ZCL version number), "只读" (read-only)
 A red arrow points from the first highlighted line to this block.
- Line 129:** `ZCL_CLUSTER_ID_GEN_BASIC, Cluster: Basic`
 { // Attribute record
 ATTRID_BASIC_HW_VERSION,
 ZCL_DATATYPE_UINT8,
 ACCESS_CONTROL_READ,
 (void *)&zclSampleSw_HWRevision //
 Annotations: "硬件版本号" (Hardware version number), "只读" (read-only)
 A red arrow points from the second highlighted line to this block.
- Line 138:** `ZCL_CLUSTER_ID_GEN_BASIC, Cluster: Basic`
 { // Attribute record
 ATTRID_BASIC_MANUFACTURER_NAME,
 ZCL_DATATYPE_CHAR_STR, 制造商名称
 ACCESS_CONTROL_READ, 只读
 (void *)&zclSampleSw_ManufacturerName
 Annotations: "制造商名称" (Manufacturer name), "只读" (read-only)
 A red arrow points from the third highlighted line to this block.

The above code creates an attribute array, which is a structure array. Let's analyze the first element:

```
{  
    ZCL_CLUSTER_ID_GEN_BASIC,//Basic Cluster, 由ZigBee联盟预定义  
    { //Attribute record  
        ATTRID_BASIC_ZCL_VERSION,//ZCL版本号属性, 由ZigBee联盟自定义  
        ZCL_DATATYPE_UINT8,//说明这个属性的数据类型是8个比特位的变量 ( 无符号整型变量 )  
        ACCESS_CONTROL_READ,//说明这个属性只能被读取不能被修改  
        (void*)&zclSampleSw_ZCLVersion//函数引用, 暂不做讲解  
    }  
}
```

(3) The design of the above code is a bit special. Its main work content is to add the attribute ATTRID_BASIC_ZCL_VERSION to the Cluster ZCL_CLUSTER_ID_GEN_BASIC, and specify the data type of this attribute and that it can only be read.

(4) If you look at the following two elements, you will find that the other two attributes are actually added to the Cluster ZCL_CLUSTER_ID_GEN_BASIC. In this way, this Cluster contains three attributes and the operation commands corresponding to these three attributes.

5. Five Commands

(1) Developers can let the source device send a command to a cluster of the target device. When the target device receives this command, it needs to perform processing related to the cluster, such as modifying the attributes of the cluster.

(2) Each Cluster contains a specific set of commands, that is, each Cluster can only receive a specific set of commands. Commands can be divided into two types: basic commands and attribute-related commands.

- (2A) Basic Commands

Each Cluster contains basic commands, that is, basic commands that can be received by all Clusters, such as read commands, write commands, and report commands. The definitions of these commands are in the zcl.h file, as shown in the figure.

```

212
213 //**** Foundation Command IDs ****/
214 #define ZCL_CMD_READ      ← 读
215 #define ZCL_CMD_READ_RSP
216 #define ZCL_CMD_WRITE     ← 写
217 #define ZCL_CMD_WRITE_UNDIVIDED
218 #define ZCL_CMD_WRITE_RSP
219 #define ZCL_CMD_WRITE_NO_RSP
220 #define ZCL_CMD_CONFIG_REPORT
221 #define ZCL_CMD_CONFIG_REPORT_RSP
222 #define ZCL_CMD_READ_REPORT_CFG
223 #define ZCL_CMD_READ_REPORT_CFG_RSP
224 #define ZCL_CMD_REPORT    ← 上报
225 #define ZCL_CMD_DEFAULT_RSP
226 #define ZCL_CMD_DISCOVER_ATTRS

```

- (2B) Cluster-qualified commands

Cluster-qualified commands only exist in certain specific Clusters, that is, Cluster-qualified commands can only be received by certain specific Clusters. Open zcl_general.h and you can find the code as shown in the figure:

```

336 //***** *****
337 //**** On/Off Cluster Attributes ****/
338 //***** *****
339 #define ATTRID_ON_OFF
340
341 #define ATTRID_ON_OFF_GLBAL_SCENE_CTRL
342 #define ATTRID_ON_OFF_ON_TIME
343 #define ATTRID_ON_OFF_OFF_WAIT_TIME
344
345 //***** *****
346 //**** On/Off Cluster Commands ****/
347 //***** *****
348 #define COMMAND_OFF
349 #define COMMAND_ON
350 #define COMMAND_TOGGLE
351 #define COMMAND_OFF_WITH_EFFECT
352 #define COMMAND_ON_WITH_RECALL_GLOBAL_SCENE
353 #define COMMAND_ON_WITH_TIMED_OFF
...

```

(3) COMMAND_OFF and COMMAND_ON are commands specifically for ON/OFF Cluster.

6. Cluster Application Examples

(1) Now we can finally give an example to illustrate the use of Clusters. Now we need to design a ZigBee lamp that supports brightness adjustment. We can use two Clusters to describe its services: the switch Cluster and the brightness Cluster.

- **The switch Cluster** is used to represent and control the switch state of the light, so the switch Cluster should **contain a switch attribute** and can be controlled by on and off commands.
- **The brightness Cluster** is used to represent and control the brightness of the light, so the brightness Cluster should **contain a brightness attribute** and can be controlled by the brightness adjustment command.

(2) From this example, we can see that Cluster is very powerful and can be used to logically implement various services of the device. In addition, its uses are far more than that. Subsequent chapters will explain more uses of Cluster.

7.5. Chapter 5: Switch command transmission and reception based on ZCL

7.5.1. Application layer calls ZCL API

7.5.2. ZCL switch command sending and receiving API

7.5.3. ZCL switch command sending and receiving experiment

7.5.1. Application layer calls ZCL API

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

1. Basic calling mode

(1) The previous chapter explained that the ZCL API can be called at the application layer to perform data communication. This lesson explains the basic mode of calling the ZCL API at the application layer.

(2) Open the application layer initialization function zclSampleSw_Init in the zcl_samplesw.c file and find the following code:

```
// Register the ZCL General Cluster Library callback functions
zclGeneral_RegisterCmdCallbacks(SAMPLESW_ENDPOINT,
&zclSampleSw_CmdCallbacks);

zclSampleSw_ResetAttributesToDefaultValues();

// Register the application's attribute list
zcl_registerAttrList(SAMPLESW_ENDPOINT,zclSampleSw_NumAttributes,zclSampleSw_Attrs);

// Register the Application to receive the unprocessed Foundation command/response messages
zcl_registerForMsg( zclSampleSw_TaskID );
```

(3) Briefly explain the working content of these three or four functions.

- **zclGeneral_RegisterCmdCallbacks()**

registers a command execution callback (Call Back). This callback contains a list of command processing functions.

When the device receives a command, it finds the corresponding command processing function in this list and processes the command.

- **zclSampleSw_ResetAttributesToDefaultValues()**
sets some attributes to their default values.
- **zcl_registerAttrList()**
registers an attribute list for the device. As mentioned in the previous section, the function parameter zclSampleSw_Attrs is an attribute list, and its definition can be found in the zcl_samplesw_data.c file.
- **zcl_registerForMsg()**
tells the device that it needs to receive unprocessed Foundation commands or response messages. After calling this function, when receiving a basic command such as a read command, write command, or report command, a system event ZCL_INCOMING_MSG will occur, and the developer can do relevant processing in the event handling function.

2. zclSampleSw_CmdCallbacks()

(1) The definition of the command execution callback can be found in the zcl_samplesw.c file. The code is as follows:

```

193  ****
194  * ZCL General Profile Callback table
195  */
196  static zclGeneral_AppCallbacks_t zclSampleSw_CmdCallbacks =
197  {
198      zclSampleSw_BasicResetCB,    // Basic Cluster Reset command
199      NULL,                     // Identify Trigger Effect command
200      NULL,                     // On/Off cluster commands
201      NULL,                     // 处理On/Off Cluster的命令!          // On/Off cluster
202      NULL,                     // On/Off cluster
203      NULL,                     // On/Off cluster
204  #ifdef ZCL_LEVEL_CTRL
205      NULL,                     // Level Control 1
206      NULL,                     // Level Control 1
207      NULL,                     // Level Control 2
208      NULL,                     // Level Control 2
209  #endif
210  #ifdef ZCL_GROUPS
211      NULL,                     // Group Response
212  #endif
213  #ifndef ZCL_GROUPS

```

(2) zclSampleSw_CmdCallbacks is a command processing function list (array), each element of which represents a processing function for a specific command. You can view which command processing function each element represents by commenting or viewing the type definition of the function list. If the element value is NULL, it means that there is no processing function for the command, that is, the command is not processed. If you need to on/off the Cluster command, you can set the processing function at the corresponding element position.

3. ZCL_INCOMING_MSG()

(1) In the AF communication section, when the device receives data from the AF layer, a system event AF_INCOMING_MSG_CMD is generated. The developer only needs to handle it in the corresponding event processing function. The meaning of ZCL_INCOMING_MSG is similar. If the system event ZCL_INCOMING_MSG occurs, it means that there are basic commands or messages that need to be processed.

(2) Open the zdSampleSw_event_loop() function in the zcl_samplesw.c file, and you can find the event processing function zclSamplesw_ProcessIncomingMsg() of ZCL_INCOMING_MSG:

zd_samplesw.c x
文件: zd_samplesw.c
函数: zdSampleSw_event_loop

```
330
331     if ( events & SYS_EVENT_MSG )
332     {
333         while ( (MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive( zclSam
334         {
335             switch ( MSGpkt->hdr.event ) ZCL Foundation Command/Response 消息
336             {
337                 case ZCL_INCOMING_MSG:
338                     // Incoming ZCL Foundation command/response messages
339                     zclSampleSw_ProcessIncomingMsg( (zclIncomingMsg_t *)MSGpkt );
340                     break;
341             }
342         }
343     }
```

(3) The definition of zclSampleSw_ProcessIncomingMsg() function is as follows:

```
1. static void zclSampleSw_ProcessIncomingMsg(zclIncomingMsg_t *pInMsg)
2. {
3.     switch ( pInMsg->zclHdr.commandID )//根据不同的commandID做相应的处理
4.     {
5. #ifdef ZCL_READ
6.     case ZCL_CMD_READ_RSP://读命令响应
7.         zclSampleSw_ProcessInReadRspCmd( pInMsg );
8.         break;
9. #endif
10. #ifdef ZCL_WRITE
11.    case ZCL_CMD_WRITE_RSP://写命令响应
12.        zclSampleSw_ProcessInWriteRspCmd( pInMsg );
13.        break;
14. #endif
15. #ifdef ZCL_REPORT // 上报命令
16.    case ZCL_CMD_CONFIG_REPORT:
17.        //zclSampleSw_ProcessInConfigReportCmd( pInMsg );
18.        break;
19.    case ZCL_CMD_CONFIG_REPORT_RSP:
20.        //zclSampleSw_ProcessInConfigReportRspCmd( pInMsg );
21.        break;
22.    case ZCL_CMD_READ_REPORT_CFG:
23.        //zclSampleSw_ProcessInReadReportCfgCmd( pInMsg );
24.        break;
25.    case ZCL_CMD_READ_REPORT_CFG_RSP:
26.        //zclSampleSw_ProcessInReadReportCfgRspCmd( pInMsg );
27.        break;
28.    case ZCL_CMD_REPORT:
29.        //zclSampleSw_ProcessInReportCmd( pInMsg );
30.        break;
```

31.#endif

```
32.     case ZCL_CMD_DEFAULT_RSP: //默认响应
33.         zclSampleSw_ProcessInDefaultRspCmd( pInMsg );
34.         break;
```

```
35.#ifdef ZCL_DISCOVER // 扫描命令
```

```
36.     ..... // 不展开
```

```
37.#endif
```

```
38.     default:
```

```
39.         break;
```

```
40. }
```

```
41. if ( pInMsg->attrCmd )
```

```
42.     osal_mem_free( pInMsg->attrCmd );
```

```
43. }
```

The content of this function is quite easy to understand. It basically performs corresponding processing according to different commandIDs.

Tip: For now, you only need to have a general understanding of the contents of this function without having to understand every line of code in depth.

(4) It should be noted that before using the basic commands of ZCL, the corresponding macros need to be enabled. For example, to use the "read" command, ZCL_READ needs to be enabled. The enabling method is the same as that of the HAL macro, as shown in the figure.

Category:

- General Options
- Static Analysis
- C/C++ Compiler**
- Assembler
- Custom Build
- Build Actions
- Linker
- Debugger
- Third-Party Driver
- Texas Instruments
- FS2 System Navigator
- Infineon
- Segger J-Link
- Nordic Semiconductor
- ROM-Monitor
- Analog Devices
- Silicon Labs
- Simulator

Multi-file Compilation
 Discard Unused Publics

Extra Options	MISRA-C:2004		
Language 1	Language 2	Code	Opt
List	Preprocessor		

Ignore standard include directories

Additional include directories: (one per line)

```
$PROJ_DIR$  
$PROJ_DIR$\..\Source  
$PROJ_DIR$\..\..\Source  
$PROJ_DIR$\..\..\..\ZMain\TI2530DB  
$PROJ_DIR$\..\..\..\..\Components\hal\include
```

Preinclude

Defined symbols: (one per line)

ZCL_READ
 ZCL_DISCOVER
 ZCL_WRITE
 ZCL_BASIC

Preproce
 Preset
 Gener

7.5.2. ZCL switch command sending and receiving API

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

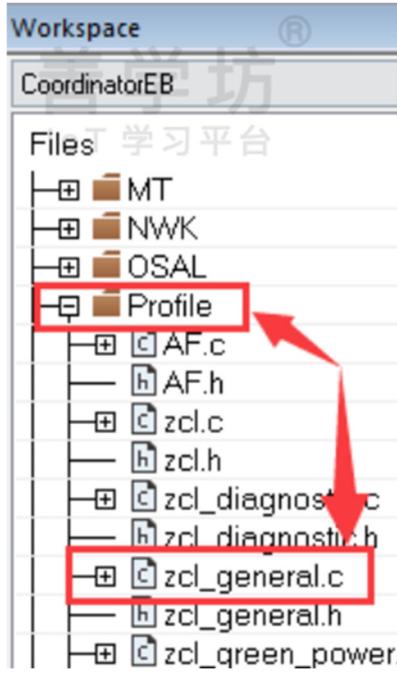
Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

In the ZCL communication process section, the author gave an example of a gateway (coordinator) and a smart socket (terminal or router). The following introduces the ZCL command sending and receiving API based on this example.

Send Command API

The On/Off command mentioned above is a Cluster-limited command in the On/Off Cluster. The On/Off Cluster also contains other commands, such as Toggle (reverse state). You can call a special API to use the command. Open the Profile folder and you can find the zcl_general.h file, as shown in the figure below.



The following three APIs can be found in the zcl_general.h file:

```
#ifdef ZCL_ON_OFF
/*
 * Send an On Off Command - COMMAND_ONOFF_OFF
 * Use like:
 *   ZStatus_t zclGeneral_SendOnOff_CmdOff( uint16 srcEP, afAddrType_t *dstAddr, uint8 disableDefaultRsp, uint8 seqNum );
*/
#define zclGeneral_SendOnOff_CmdOff(a,b,c,d) zcl_SendCommand( (a), (b), ZCL_CLUSTER_ID_GEN_ON_OFF, COMMAND_OFF, TRUE, ZCL_FRAME_CLIENT_SERVER_DIR, (c), 0, (d), 0, NULL )

/*
 * Send an On Off Command - COMMAND_ONOFF_ON
 * Use like:
 *   ZStatus_t zclGeneral_SendOnOff_CmdOn( uint16 srcEP, afAddrType_t *dstAddr, uint8 disableDefaultRsp, uint8 seqNum );
*/
#define zclGeneral_SendOnOff_CmdOn(a,b,c,d) zcl_SendCommand( (a), (b), ZCL_CLUSTER_ID_GEN_ON_OFF, COMMAND_ON, TRUE, ZCL_FRAME_CLIENT_SERVER_DIR, (c), 0, (d), 0, NULL )

/*
 * Send an On Off Command - COMMAND_ONOFF_TOGGLE
 * Use like:
 *   ZStatus_t zclGeneral_SendOnOff_CmdToggle( uint16 srcEP, afAddrType_t *dstAddr, uint8 disableDefaultRsp, uint8 seqNum );
*/
#define zclGeneral_SendOnOff_CmdToggle(a,b,c,d) zcl_SendCommand( (a), (b), ZCL_CLUSTER_ID_GEN_ON_OFF, COMMAND_TOGGLE, TRUE, ZCL_FRAME_CLIENT_SERVER_DIR, (c), 0, (d), 0, NULL )
```

These three APIs implement sending close, open and reverse state commands respectively. They are all defined using #define, and finally call the zcl_SendCommand function to send commands. The definition of the zcl_SendCommand function is as follows:

```
extern ZStatus_t zcl_SendCommand(
    uint8 srcEP, //源端点, 发送者的端点号
```

```

afAddrType_t *dstAddr,//目标设备地址

uint16 clusterID, uint8 cmd,//Cluster ID和命令

uint8 specific,//是否为属性关联命令

uint8 direction,//通信方向

uint8 disableDefaultRsp;//是否关闭默认响应 ( 目标设备的响应 )

uint16 manuCode //manu code

uint8 seqNum,//数据包标识号 · 由开发者自定义

uint16 cmdFormatLen//命令格式长度

uint8 *cmdFormat//命令格式

);

```

Taking zclGeneral_SendOnOff_CmdOff as an example, let's briefly introduce its call to zcl_SendCommand. The code is as follows:

```

zcl_SendCommand(
    (a),
    (b),
    ZCL_CLUSTER_ID_GEN_ON_OFF,//Cluster ID
    COMMAND_OFF,//待发送命令为关闭命令
    TRUE,//TRUE表示属性关联命令
    ZCL_FRAME_CLIENT_SERVER_DIR,//表示通信方向为从Client到Server
    (c),
    0,//manu code为0
    (d),
    0,//命令格式长度为0
    NULL//命令格式为空
)

```

Usually, developers can call various command sending APIs encapsulated based on zcl_SendCommand, such as the three command sending APIs mentioned above.

Receiving Commands

The method of receiving On/Off commands is relatively simple. You only need to add a command processing function to the corresponding position of the ZCL command processing function list zclSampleSw_CmdCallbacks to realize the reception and processing of commands. The next lesson will explain it with specific cases.

7.5.3. ZCL switch command sending and receiving experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

This lesson will explain how to use the ZCL communication API to send On/Off commands in an experimental way. The experimental equipment includes a gateway (coordinator) and a smart socket (terminal or router). The content is that when the smart socket is added to the network, the gateway automatically and regularly sends On and Off commands to the smart socket to control the switch of the smart socket.

Smart socket development

Open the zcl_samplesw.c file and find the following code in the application initialization function zdSampleSw_Init:

```
zcl_samplesw.c × 文件: zcl_samplesw.c  
zclSampleSw_Init(byte) 函数: zclSampleSw_Init  
303 ifndef ZDO_COORDINATOR  
304 // Init Uart  
305 zclSampleSw_InitUart();  
306  
307 ZDO_RegisterForZDOMsg ( zclSampleSw_TaskID, Device_ance );  
308  
309 else bdb_StartCommissioning( BDB_COMMISSIONING_MODE_NWK_FORMATION |  
310 BDB_COMMISSIONING_MODE_FINDING_BINDING );  
311  
312 NLME_PermitJoiningRequest(255); 路由器/终端设备  
313  
314 else bdb_StartCommissioning( BDB_COMMISSIONING_MODE_NWK_STEERING |  
315 BDB_COMMISSIONING_MODE_FINDING_BINDING );  
316  
317 zclSampleSw_DeviceAnnce(); 发送通知信息!  
318  
319 endif  
320 }
```

The codes in #else and #endif are executed when the device is in the terminal (or router) role. The bdb_StartCommissioning function is used to build a network, which has been explained in the BDB chapter.

The zclSampleSw_DeviceAnnce function is customized by the author and is used to broadcast a data packet to the entire network. The data packet contains the address of the device. When the coordinator receives this data packet, it knows the address information of the device. Its definition code is as follows:

```
static void zclSampleSw_DeviceAnnce( void )  
{  
    ZDP_DeviceAnnce(  
        NLME_GetShortAddr(),//获取本设备的网络地址 ( 短地址 )  
        NLME_GetExtAddr(),//获取本设备的物理地址 ( 通常就是MAC地址 )  
        ZDO_Config_Node_Descriptor.CapabilityFlags,//暂不展开简介 · 可忽略  
        0//暂不展开讲解 · 可忽略  
    );  
}
```

This function calls the ZDP_DeviceAnnce function, which is a protocol stack API. The above code can be applied to broadcast the address information of the device to the network.

Processing Instructions

Before processing the On/Off command, you need to register a callback function in the ZCL command callback function list, that is, add the zclSampleSw_OnOffCB function in the zcl_samplesw.c file, as shown in the following figure.

```
zcl_samplesw.c * x
zclSampleSw_CmdCallbacks

203 // ****
204 * ZCL General Profile Callback table
205 */
206 static zclGeneral_AppCallbacks_t zclSampleSw_CmdCallbacks =
207 {
208     zclSampleSw_BasicResetCB,           // Basic Cluster
209     NULL,                            // Identify Trig
210
211 #ifdef ZDO_COORDINATOR           // On/Off Cluster命令
212     NULL,
213 #else
214     zclSampleSw_OnOffCB,            // 处理函数!
215 #endif                           // On/Off cluster commands
216 }
```

The definition code of the zclSampleSw_OnOffCB function is as follows:

```
1. static void zclSampleSw_OnOffCB( uint8 cmd )
2. {
3.     if(cmd == COMMAND_ON) // 命令为ON时
4.     {
5.         HalLcdWriteString("Set ON", 4); // 打印信息到屏幕
6.
7.         HalLedSet(HAL_LED_ALL, HAL_LED_MODE_ON); // 开启所有LED
8.     }
9.     else if(cmd == COMMAND_OFF) // 命令为OFF时
10.    {
11.        HalLcdWriteString("Set OFF", 4); // 打印信息到屏幕
12.
13.        HalLedSet(HAL_LED_ALL, HAL_LED_MODE_OFF); // 关闭所有LED
14.    }
15.}
```

Gateway Development

The gateway mainly completes two things:

- Receive and process address information broadcast by smart sockets
- Start a periodic event to periodically send On/Off commands to the smart socket

Receiving address information

When the coordinator receives the address message broadcast by the smart socket (Annce), a system event ZDO_CB_MSG is generated. The event can be processed in the application layer event processing function. The code is shown in the figure

below.



```
zcl_sampleSw.c x
zclSampleSw_event_loop(uint8, uint16) ← 文件: zcl_sampleSw.c
函数: zclSampleSw_event_loop

370
371 #ifdef ZDO_COORDINATOR
372     case ZDO_CB_MSG:
373         zclSampleSw_processZDOMgs( (zdoIncomingMsg_t *)MSGpkt );
374         break;
375     #endif
```

The code definition of the event processing function zclSampleSw_ProcessZDOMgs is as follows:

```
1. static void zclSampleSw_processZDOMgs(zdoIncomingMsg_t *pMsg)
2. {
3.     switch ( pMsg->clusterID )//判断消息中的Cluster ID
4.     {
5.         case Device_anne://如果是Device_anne
6.         {
7.             // 把目标设备的网络地址保存到全局变量中
8.             zclSampleSw_OnOffTestAddr = pMsg->srcAddr.addr.shortAddr;
9.             // 在屏幕上显示目标设备网络地址和提示信息
10.            HalLcdWriteStringValue("Node:", pMsg->srcAddr.addr.shortAddr, 16, 3);
11.            HalLcdWriteString("On/Off Test...", 4);
12.
13.            //周期地产生SAMPLEAPP_ONOFF_TEST_EVT事件，即发送On/Off指令
14.            osal_start_timerEx(zclSampleSw_TaskID,
15.                                SAMPLEAPP_ONOFF_TEST_EVT,//事件类型，在zcl\sampleSw.h文件中定义
16.                                SAMPLEAPP_ONOFF_TEST_PERIOD);//时间间隔，在zcl\sampleSw.h文件中定义
17.        }
18.        break;
19.    default:
20.        break;
21.    }
22.}
```

Handle SAMPLEAPP_ONOFF_TEST_EVT event

Find the zclSampleSw_eventloop function in the zcl_samplesw.c file and add the event handling code, as shown in the figure below.

```
zcl_samplesw.c x
zclSampleSw_event_loop(uint8, uint16)

396
397 #ifdef ZDO_COORDINATOR
398     if ( events & SAMPLEAPP_ONOFF_TEST_EVT )
399     {
400         zclSampleSw_OnOffTest(); // 发送On/Off命令给目标设备!
401         osal_start_timerEx(zclSampleSw_TaskID,
402                             SAMPLEAPP_ONOFF_TEST_EVT,
403                             SAMPLEAPP_ONOFF_TEST_PERIOD);
404
405         return ( events ^ SAMPLEAPP_ONOFF_TEST_EVT );
406     }
407 #endif
```

Send On/Off command

In the SAMPLEAPP_ONOFF_TEST_EVT event processing code, zclSampleSw_OnOffTest is called to send instructions. The function definition is as follows:

```
static void zclSampleSw_OnOffTest(void)
{
    afAddrType_t destAddr;//用于保存目标设备的地址信息
    static uint8 txID = 0;
    static bool on = true;//静态变量，指示智能插座的开关状态

    destAddr.endPoint = SAMPLESW_ENDPOINT;//端点号
    destAddr.addrMode = Addr16Bit;//地址模式（类型）为16位的地址，使用P2P的通信方式
    destAddr.addr.shortAddr = zclSampleSw_OnOffTestAddr;//网络地址

    if(on) {//如果智能插座正在开启
        HalLcdWriteString("Command: ON", 4); // 屏幕打印提示信息
        zclGeneral_SendOnOff_CmdOn();//发送打开命令
        SAMPLESW_ENDPOINT;//端点号
        &destAddr;//地址信息
        TRUE;//TRUE表示属性关联命令
        txID++);
    }
    else//如果智能插座已关闭
    {
        HalLcdWriteString("Command: OFF", 4); // 屏幕打印提示信息
        zclGeneral_SendOnOff_CmdOff();//发送关闭命令
        SAMPLESW_ENDPOINT;//端点号
        &destAddr;//地址信息
    }
}
```

```
TRUE,//TRUE表示属性关联命令
```

```
txID++);
```

```
}
```

```
on = !on;//反转开关状态
```

```
}
```

Simulation debugging

- Compile the coordinator project and then burn the firmware into one of the development boards, which will act as the gateway;
- Compile the terminal (or router) project and then burn it into another development board, which acts as a smart socket;
- Power the gateway and smart socket respectively;

The smart socket will automatically join the network created by the gateway, and then you can see the commands sent by the gateway on the display of the smart socket, as shown in the figure.



7.6. Chapter 6: Attribute reading and writing based on ZCL

7.6.1. ZCL attribute read and write API

7.6.2. ZCL attribute reading and writing experiment

7.6.1. ZCL attribute read and write API

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Adding properties

All the properties required by the device to be developed need to be defined in the property list. As described in the previous section, open the zcl_samplesw_data.c file to find the code definition of the property list:

```
zcl_samplesw_data.c x      文件: zcl_samplesw_data.c  
zclSampleSw_Attrs ← 属性列表: zclSampleSw_Attrs  
115  
116 CONST zclAttrRec_t zclSampleSw_Attrs[] =  
117 {  
118     // *** General Basic Cluster Attributes ***  
119     {  
120     }  
137     {  
146     }  
155     {  
164     }  
173     {  
182     }  
191     {  
200     }  
209  
210     // *** Identify Cluster Attribute ***  
211     {  
220     }  
229  
230  
231     // *** On / Off Switch Configuration Cluster *** ,  
232     {  
241     {  
242         ZCL_CLUSTER_ID_GEN_ON_OFF_SWITCH_CONFIG, Cluster  
243         { // Attribute record  
244             ATTRID_ON_OFF_SWITCH_ACTIONS, 属性 可读可写  
245             ZCL_DATATYPE_ENUM8,  
246             ACCESS_CONTROL_READ | ACCESS_CONTROL_WRITE,  
247             (void *)&zclSampleSw_OnOffSwitchActions  
248         }  
249     },
```

This array has been explained in the ZCL content section, and here we will explain it in more detail. Take the attribute ATTRID_ON_OFF_SWITCH_ACTIONS as an example. This attribute:

- Is an 8-bit attribute (ZCL_DATATYPE_ENUM8)
- You can see that this property can be read and written (ACCESS_CONTROL_READ | ACCESS_CONTROL_WRITE).
- This property is added to Cluster ZCL_CLUSTER_ID_GEN_ON_OFF_SWITCH_CONFIG

Developers can apply the above code to add specified attributes to a specified Cluster.

Attribute read and write command API

You can call the attribute read and write command API to read or write the specified attribute. Open the zcl.h file and find the definition of the attribute read and write command API. The code is as follows:

```
#ifdef ZCL_READ  
/*  
 * 发送一个读取属性命令  
 */  
extern ZStatus_t zcl_SendRead(  
    uint8 srcEP, // 源端点号  
    afAddrType_t *dstAddr, // 目标设备地址信息  
    uint16 realClusterID, // Cluster ID  
    zclReadCmd_t *readCmd, // “读”信息  
    uint8 direction, // 通信方向
```

```

uint8 disableDefaultRsp, //是否关闭默认响应 ( 目标设备的响应 )
uint8 seqNum); // 数据包标号 · 由开发者自定义

/*
 * 省略部分代码
 */
#endif // ZCL_READ

#ifndef ZCL_WRITE
/*
 * 发送一个写入属性命令 ZCL_CMD_WRITE
 * 按以下方式调用:
 * ZStatus_t zcl_SendWrite(
 *     uint8 srcEP, // 源应用端点
 *     afAddrType_t *dstAddr, // 目标设备地址信息
 *     uint16 realClusterID, // Cluster ID
 *     zclWriteCmd_t *writeCmd, // “写”信息
 *     uint8 direction, // 通信方向
 *     uint8 disableDefaultRsp, //是否关闭默认响应 ( 目标设备的响应 )
 *     uint8 seqNum); // 数据包标号 · 由开发者自定义
 */
#define zcl_SendWrite(a,b,c,d,e,f,g) (zcl_SendWriteRequest( (a), (b), (c), (d), ZCL_CMD_WRITE, (e), (f), (g) ))

/*
 * 省略部分代码
 */
#endif // ZCL_WRITE

```

Command Processing

After the client device sends the above command to the server device, it will receive the response information returned by the server and generate the system event ZCL_INCOMING_MSG. Therefore, this event needs to be processed. Open the zclSampleSw_event_loop function in the zcl_samplesw.c file and add the corresponding event processing function

zclSampleSw_ProcessIncomingMsg. The code is as follows:

```
zd_samplesw.c × 文件: zd_samplesw.c  
zdSampleSw_event_loop(uint8, uint16) 函数: zdSampleSw_event_loop  
346     if ( events & SYS_EVENT_MSG )  
347     {  
348         while ( (MSGpkt = (afIncomingMSGPacket*)  
349             {  
350                 switch ( MSGpkt->hdr.event )  
351                 {  
352                     case ZCL_INCOMING_MSG:  
353                         // Incoming ZCL Foundation comm.  
354                         zclSampleSw_ProcessIncomingMsg()  
355                         break;  
356     }
```

zclSampleSw_ProcessIncomingMsg is a function defined by the developer. The code is as follows:

```
1. static void zclSampleSw_ProcessIncomingMsg(zclIncomingMsg_t *pInMsg)  
2.{  
3.     switch ( pInMsg->zclHdr.commandID )  
4.     {  
5. #ifdef ZCL_READ  
6.         case ZCL_CMD_READ_RSP: // 读命令响应信息  
7.             zclSampleSw_ProcessInReadRspCmd( pInMsg );//读响应信息处理函数  
8.             break;  
9. #endif  
10. #ifdef ZCL_WRITE  
11.         case ZCL_CMD_WRITE_RSP: // 写命令响应信息  
12.             zclSampleSw_ProcessInWriteRspCmd( pInMsg ); //写命令响应信息处理函数  
13.             break;  
14. #endif  
15. #ifdef ZCL_REPORT  
16.         ..... // 暂时不展开  
17. #endif  
18.  
19.         case ZCL_CMD_DEFAULT_RSP: // 默认响应信息  
20.             zclSampleSw_ProcessInDefaultRspCmd( pInMsg );//默认响应信息处理函数  
21.             break;  
22.  
23. #ifdef ZCL_DISCOVER  
24.         ..... // 暂时不展开  
25. #endif  
26.  
27.     default:  
28.         break;  
29.     }  
30.
```

```
31. if ( pInMsg->attrCmd )  
32.     osal_mem_free( pInMsg->attrCmd );  
33. }
```

The above code uses the default response (ZCL_CMD_DEFAULT_RSP), read response (ZCL_CMD_READ_RSP) and write response (ZCL_CMD_WRITE_RSP), which are explained in detail below.

- **When the client**

sends a command to the server, it can specify whether the server needs to return a default response message. The client can use this to determine whether the server has actually received the command.

- **Read Response**

After the client sends a read attribute command to the server, the server returns a read response, which contains information such as whether the attribute was read successfully and the attribute value.

- **Write Response**

After the client sends a write command to the server, the server returns a write response, which contains information such as whether the attribute value is written successfully.

7.6.2. ZCL attribute reading and writing experiment

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

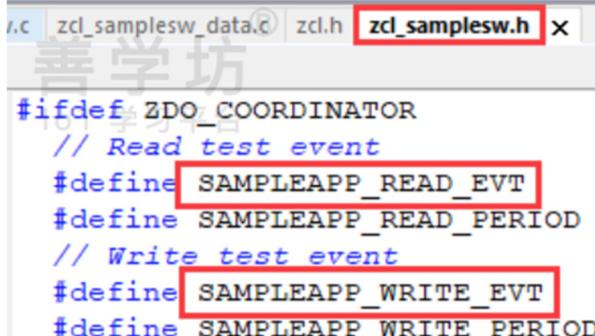
Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

This lesson will explain how to use the ZCL read and write command API in an experimental way. The experimental equipment includes a coordinator and a terminal. The content is that the coordinator sends a write command to the terminal, and then sends a read command, and finally compares whether the written information is consistent with the read information.

Defining read and write events

Define a read command event and a write command event in the zcl_samplesw.h file. The code is as follows:



```
#ifdef ZDO_COORDINATOR  
    // Read test event  
    #define SAMPLEAPP_READ_EVT  
    #define SAMPLEAPP_READ_PERIOD  
    // Write test event  
    #define SAMPLEAPP_WRITE_EVT  
    #define SAMPLEAPP_WRITE_PERIOD
```

After the coordinator receives the information broadcast by the terminal device (Annce), it starts a read command event. The code is as follows:

```
zcl_samplesw.c x zcl_samplesw_data.c zcl.h  
善学坊  
IoT学习平台  
875     * @brief      Process ZDO Message  
876     */  
877     static void zclSampleSw_processZDOMgs(zdoIncomingMsg_t *pMsg)  
878     {  
879         switch ( pMsg->clusterID )  
880         {  
881             case Device_annce:  
882             {  
883                 zclSampleSw_TestAddr = pMsg->srcAddr.addr.shortAddr;  
884  
885                 HalLcdWriteStringValue("Node:", pMsg->srcAddr.addr.shor  
886                 HalLcdWriteString("Test...", 4);  
887  
888  
889             osal_start_timerEx(zclSampleSw_TaskID,  
890                         SAMPLEAPP_READ_EVT,  
891                         SAMPLEAPP_READ_PERIOD);  
892  
893             break;  
894  
895             default:  
896             break;  
897         }  
898     }
```

In the zclSample_event_loop event processing function in the zcl_samplews.c file, you can find the processing code for the read command event as follows:

```
//  
if ( events & SAMPLEAPP_READ_EVT )//如是读命令事件  
{  
    zclSampleSw_ReadTest();//读命令事件处理函数  
  
    //启动一个写命令事件  
    osal_start_timerEx(zclSampleSw_TaskID,  
                        SAMPLEAPP_WRITE_EVT,  
                        SAMPLEAPP_WRITE_PERIOD);  
  
    return ( events ^ SAMPLEAPP_READ_EVT );  
}
```

The code definition of the read command event processing function zclSampleSw_ReadTest is as follows:

```
1. static void zclSampleSw_ReadTest(void)  
2.{  
3.     afAddrType_t destAddr;  
4.     zclReadCmd_t *readCmd;  
5.     static uint8 txID = 0;  
6.
```

```

7. destAddr endPoint = SAMPLESW_ENDPOINT;
8. destAddr addrMode = afAddr16Bit;
9. destAddr addr shortAddr = zclSampleSw_TestAddr;
10.

//申请一个动态内存

11. readCmd = (zclReadCmd_t *)osal_mem_alloc(sizeof(zclReadCmd_t) +
12.                                         sizeof(uint16));
13. if(readCmd == NULL)//判断是否成功申请到内存
14. {
15.     return;
16.

17.     readCmd->numAttr = 1;//待读取的属性数量为1
18.     readCmd->attrID[0] = ATTRID_ON_OFF_SWITCH_ACTIONS;//待读取的属性ID
19.
20.     zcl_SendRead(SAMPLESW_ENDPOINT,
21.                 &destAddr,
22.                 ZCL_CLUSTER_ID_GEN_ON_OFF_SWITCH_CONFIG//Cluster ID
23.                 readCmd,
24.                 ZCL_FRAME_CLIENT_SERVER_DIR);//通信方向是由客户端到服务器端
25.     TRUE,
26.     txID++);
27.
28.     osal_mem_free(readCmd);//释放内存
29.
30. }

```

A write command event is also started in the code during the read command processing. Below the read command event processing code, you can find the write command event processing code as follows:

```

if ( events & SAMPLEAPP_WRITE_EVT )//如果是写命令事件
{
    zclSampleSw_WriteTest();//写命令处理函数

    osal_start_timerEx(zclSampleSw_TaskID,//启动一个读命令事件
                        SAMPLEAPP_READ_EVT,
                        SAMPLEAPP_READ_PERIOD);

    return ( events ^ SAMPLEAPP_WRITE_EVT );
}

```

The code definition of the write command event processing function zclSampleSw_WriteTest is as follows:

```

static void zclSampleSw_WriteTest(void)
{
    afAddrType_t destAddr;
    zclWriteCmd_t *writeCmd;
    static uint8 txID = 0;

    destAddr.endPoint = SAMPLESW_ENDPOINT;

```

```

destAddr.addrMode = afAddr16Bit;
destAddr.addr.shortAddr = zclSampleSw_TestAddr;

writeCmd=(zclWriteCmd_t *)osal_mem_alloc(sizeof(zclWriteCmd_t) +
                                         sizeof(zclWriteRec_t));//申请一个动态内存

if(writeCmd == NULL)//判断动态内存是否申请成功
    return;

writeCmd->attrList[0].attrData=(uint8*)osal_mem_alloc(sizeof(uint8));//申请一个动态内存

if(writeCmd->attrList[0].attrData == NULL)//判断动态内存是否申请成功
    return;

writeCmd->numAttr = 1;//待写入的属性数量

writeCmd->attrList[0].attrID = ATTRID_ON_OFF_SWITCH_ACTIONS;待写入的属性的ID

writeCmd->attrList[0].dataType = ZCL_DATATYPE_ENUM8;//属性值的类型

*(writeCmd->attrList[0].attrData) = txID;//属性值

HalLcdWriteStringValue("Write:", txID, 10, 4);

zcl_SendWrite(SAMPLESW_ENDPOINT,
              &destAddr,
              ZCL_CLUSTER_ID_GEN_ON_OFF_SWITCH_CONFIG,//Cluster ID
              writeCmd,
              ZCL_FRAME_CLIENT_SERVER_DIR;//通信方向是由客户端到服务器端
              TRUE,
              txID++);

osal_mem_free(writeCmd->attrList[0].attrData); // 释放内存

osal_mem_free(writeCmd); // 释放内存
}

```

Read response processing

Process the information read from the server in the client, that is, write a read command response information processing function. The code is as follows:

```

#ifndef ZCL_READ
/****************************************
* @fn    zclSampleSw_ProcessInReadRspCmd
*
* @brief 读响应处理函数
*
* @param pInMsg - 待处理的消息
*
* @return

```

```

*/
static uint8 zclSampleSw_ProcessInReadRspCmd( zclIncomingMsg_t *pInMsg )
{
    zclReadRspCmd_t *readRspCmd;
    uint8 i;

    readRspCmd = (zclReadRspCmd_t *)pInMsg->attrCmd;

    for (i = 0; i < readRspCmd->numAttr; i++)//readRspCmd->numAttr为属性的数量
    {
        if( pInMsg->clusterId == ZCL_CLUSTER_ID_GEN_ON_OFF_SWITCH_CONFIG && //如果该消息是关于指定的Cluster
            readRspCmd->attrList[i].attrID == ATTRID_ON_OFF_SWITCH_ACTIONS )//如果该属性的ID是指定的属性ID
        {
            uint8 val;

            val = *(readRspCmd->attrList[i].data);//读取属性值

            HalLcdWriteStringValue("Read:", val, 10, 4); //显示信息到屏幕中
        }
    }

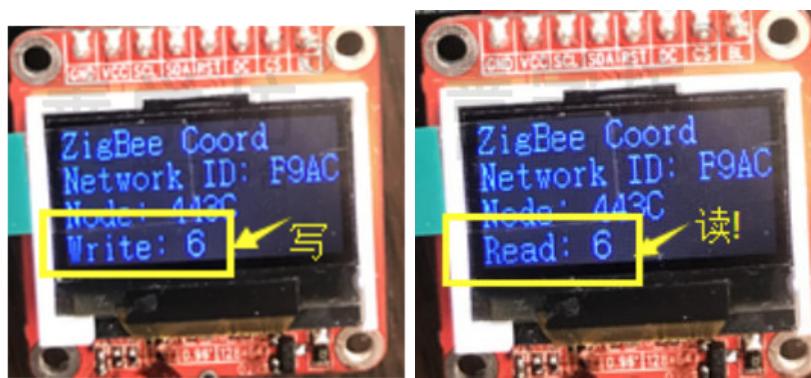
    return TRUE;
}

#endif // ZCL_READ

```

Simulation debugging

- Compile the coordinator and terminal (router) projects separately, and then download them to the two development boards respectively.
- After the terminal (router) device joins the ZigBee network, you can see the following prompt information displayed on the coordinator screen.



It can be observed that the read and written data are the same.

7.7. Chapter 7: Attribute reporting experiment based on ZCL

7.7.1. Overview

7.7.2. Terminal Equipment Development

7.7.3. Coordinator Device Development

7.7.4. Simulation Debugging

7.7.1. Overview

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Sensor-type terminal devices can use the attribute reporting API to report data to the coordinator. For example, the temperature and humidity sensor reports the collected temperature and humidity data, and the human infrared sensor reports this information when it detects a person.

This lesson will explain the attribute reporting technology based on ZCL in an experimental way. The main experimental equipment includes a coordinator device and a terminal (or router) device. The experimental content is:

- The terminal device periodically reports data to the coordinator.
- The coordinator receives the data and displays it on the screen.

7.7.2. Terminal Equipment Development

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

The terminal equipment development content mainly includes the following parts:

- Define and start attribute reporting events.
- Define event processing functions to report attributes.
- Enable the corresponding macro definition.

Define and start attribute reporting events

Add event definition and event occurrence time definition in the zcl_samplesw.h file. The code is as follows:

```
#ifdef ZDO_COORDINATOR
```

```
#else
```

```
/*
```

* 此处省略了部分代码

```
*/  
  
#define SAMPLEAPP_REPORT_EVT      0x0040  
#define SAMPLEAPP_REPORT_PERIOD    3000  
  
#endif
```

Add the startup attribute reporting event code in the application layer initialization function zclSample_Init in the zcl_samplesw.c file:

```
osal_start_timerEx(zclSampleSw_TaskID,  
                    SAMPLEAPP_REPORT_EVT,//事件  
                    SAMPLEAPP_REPORT_PERIOD);//延迟处理事件的时间长度
```

Defining event handling functions

After defining the event, you need to add an event processing function. Open the zcl_samplesw.c file and find the event processing function zclSampleSw_event_loop, and add the following code:

```
#ifdef ZDO_COORDINATOR  
#else  
/*  
* 此处省略了部分代码  
*/  
  
if ( events & SAMPLEAPP_REPORT_EVT )  
{  
    zclSampleSw_ReportTest(); //属性上报事件的处理函数  
  
    //启动下一个属性上报事件  
    osal_start_timerEx(zclSampleSw_TaskID,  
                        SAMPLEAPP_REPORT_EVT,  
                        SAMPLEAPP_REPORT_PERIOD);  
  
    return ( events ^ SAMPLEAPP_REPORT_EVT );  
}  
#endif
```

Introduction to the Attribute Reporting API

You only need to call the attribute reporting API to complete the attribute reporting function. In the zcl.c file, you can find the attribute reporting API. The code is as follows:

```
/*  
* 上报一个或多个属性值  
*/  
  
extern ZStatus_t zcl_SendReportCmd(  
    uint8 srcEP, //源端点号
```

```

afAddrType_t *dstAddr, //目标设备地址信息

uint16 realClusterID, //属性所属Cluster ID

zclReportCmd_t *reportCmd, //描述待上报的属性值

uint8 direction, //通信方向

uint8 disableDefaultRsp, //是否关闭默认响应 ( 目标设备的响应 )

uint8 seqNum); //数据包标号 · 由开发者自定义

```

Attribute reporting function

After learning the API, you can write the attribute reporting event processing function zclSampleSw_ReportTest, whose code is defined as follows:

```

/*
 * 数据上报事件的处理函数 · 用于上报数据
 */

static void zclSampleSw_ReportTest(void)
{
    static uint8 seqNum = 0;

    zclReportCmd_t *reportCmd;

    //目标设备的地址信息

    afAddrType_t destAddr;
    destAddr.addrMode = afAddr16Bit;
    destAddr.endPoint = SAMPLESW_ENDPOINT;
    destAddr.addr.shortAddr = 0x0000; //0x0000表示协调器的网络地址

    reportCmd = (zclReportCmd_t *)osal_mem_alloc(sizeof(zclReportCmd_t)+sizeof(zclReport_t)); //申请内存空间

    if(reportCmd == NULL) //判断内存空间是否申请成功
        return;

    reportCmd->attrList[0].attrData = (uint8 *)osal_mem_alloc(sizeof(uint8)); //申请内存空间

    if(reportCmd->attrList[0].attrData == NULL) //判断内存空间是否申请成功
        return;

    reportCmd->numAttr = 1; //属性数量为1

    reportCmd->attrList[0].attrID = ATTRID_ON_OFF_SWITCH_TYPE; //属性ID

    reportCmd->attrList[0].dataType = ZCL_DATATYPE_ENUM8; //数据类型

    *((uint8 *) (reportCmd->attrList[0].attrData)) = seqNum; //属性值

    //上报数据

    zcl_SendReportCmd(SAMPLESW_ENDPOINT, //源端点号

```

```

&destAddr, //地址信息

ZCL_CLUSTER_ID_GEN_ON_OFF_SWITCH_CONFIG //Cluster ID
reportCmd.

ZCL_FRAME_CLIENT_SERVER_DIR //通信方向为从客户端到服务端

TRUE //关闭默认响应 ( 目标设备的响应 )

seqNum++ ); //数据包标号，每上报一次数据 seqNum 的值就会增加1

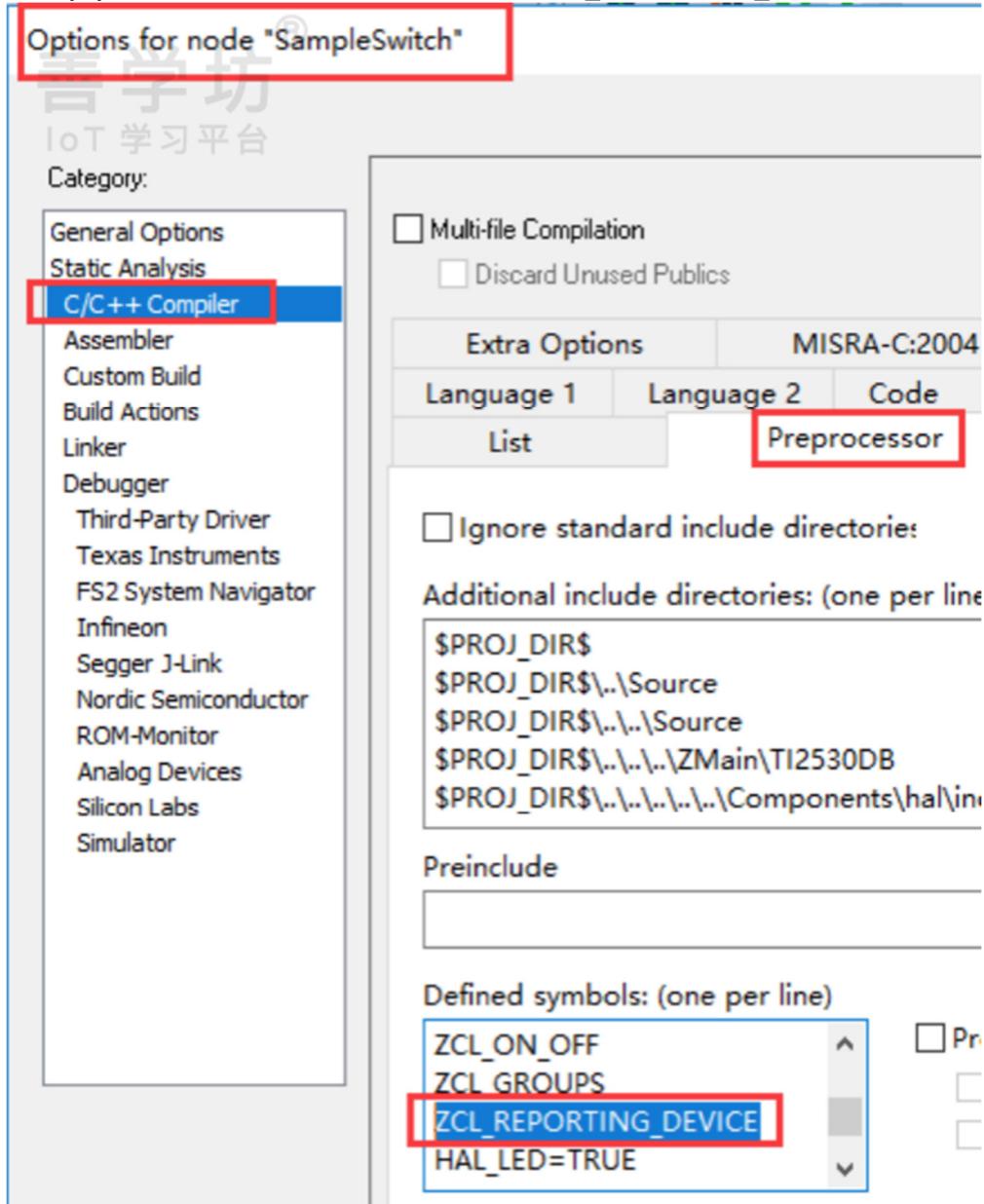
HalLcdWriteStringValue("Report: ", (seqNum-1), 10, 4); //显示

// 释放内存空间！
osal_mem_free(reportCmd->attrList[0].attrData);
osal_mem_free(reportCmd);
}

```

Enable the corresponding macro definition

Finally, you need to enable a macro definition ZCL_REPORTING_DEVICE, as shown in the figure below.



7.7.3. Coordinator Device Development

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

The coordinator device development content mainly includes the following parts:

- Receive and process attribute data.
- Enable the corresponding macro definition.

Receiving and processing attribute data

Similar to the read and write commands, after the coordinator receives the report information, the system event ZCL_INCOMING_MSG will occur. Open the zdSampleSw_event_loop function in the zcl_samplesw.c file, and you can find the event processing function zclSamplesw_ProcessIncomingMsg of ZCL_INCOMING_MSG. Some of the code is as follows:

```
1. static void zclSampleSw_ProcessIncomingMsg( zclIncomingMsg_t *pInMsg )  
2.{  
3.     switch ( pInMsg->zclHdr.commandID )  
4.     {  
5.         ..... // 不展开  
6.     }  
7. #ifdef ZCL_REPORT  
8.     ..... // 不展开  
9.     case ZCL_CMD_REPORT://属性上报  
10.    zclSampleSw_ProcessInReportCmd( pInMsg ); //属性上报处理函数  
11.    break;  
12. #endif  
13.    ..... // 不展开  
14. }  
15. if ( pInMsg->attrCmd )  
16.     osal_mem_free( pInMsg->attrCmd );  
17. }
```

The attribute reporting processing function zclSampleSw_ProcessInReportCmd is customized by the author, and the code is as follows:

```
#ifdef ZCL_REPORT  
static uint8 zclSampleSw_ProcessInReportCmd( zclIncomingMsg_t *pInMsg )  
{  
    zclReportCmd_t *reportCmd;  
    uint8 i;  
  
    reportCmd = (zclReportCmd_t *)pInMsg->attrCmd;
```

```

for ( i = 0; i < reportCmd->numAttr; i++ )//reportCmd->numAttr为属性数量
{
    if( pInMsg->clusterId == ZCL_CLUSTER_ID_GEN_ON_OFF_SWITCH_CONFIG//Cluster ID
&& reportCmd->attrList[i].attrID == ATTRID_ON_OFF_SWITCH_TYPE)//属性ID
    {
        int8 attrDat = *(reportCmd->attrList[i].attrData);//读取属性值
        HalLcdWriteStringValue("Rx Value:", attrDat, 10, 4); //显示属性值
    }
}
return ( TRUE );
}

#endif // ZCL_REPORT

```

Enable the corresponding macro definition

Finally, you need to enable two macros: ZCL_REPORT_DESTINATION_DEVICE and ZCL_REPORT, as shown in the figure below.

Options for node "SampleSwitch"

吉子功

IoT 学习平台

Category:

General Options

Static Analysis

C/C++ Compiler

Assembler

Custom Build

Build Actions

Linker

Debugger

Third-Party Driver

Texas Instruments

FS2 System Navigator

Infineon

Segger J-Link

Nordic Semiconductor

ROM-Monitor

Analog Devices

Silicon Labs

Simulator

Multi-file Compilation

Discard Unused Publics

Extra Options

MISRA-C:2004

Language 1

Language 2

Code

Opt

List

Preprocessor

Ignore standard include directories

Additional include directories: (one per line)

\$PROJ_DIR\$

\$PROJ_DIR\$\..\Source

\$PROJ_DIR\$\..\..\Source

\$PROJ_DIR\$\..\..\..\ZMain\TI2530DB

\$PROJ_DIR\$\..\..\..\..\Components\hal\include

Preinclude

(empty line)

Defined symbols: (one per line)

ZCL_ON_OFF

ZCL GROUPS

ZCL_REPORT_DESTINATION_ID

ZCL_REPORT

Preproce

Presel

Gener

7.7.4. Simulation Debugging

Tutor video explanation: <https://www.bilibili.com/video/BV1k34y1D7Vz/>

Technical support instructions:

1. Generally, self-study is the main method.
2. You can ask questions in the official Q&A community: <https://bbs.csdn.net/forums/zigbee>
3. Engineers will answer community questions as soon as possible, but they are front-line developers and [cannot guarantee] the timeliness of the answer. It is hard work to answer them, thank you for your understanding!

Compile the coordinator and terminal (router) projects separately, and then download them to the two development boards respectively.

After the terminal (router) device joins the ZigBee network, you can see the following prompt information displayed on the coordinator screen.

