STC 车规 32位 8051 AEC-Q100 Grade1

超强抗干扰，超级加密

车规 AEC-Q100 Grade1 MCU 设计公司，高稳定，低功耗

官网/技术论坛：www.STCAI.com 分销电话：0513-5501 2928, 8989 6509

深圳国芯人工智能有限公司

车规 AEC-Q100 Grade1 MCU 设计公司

STC8H 系列

实验指导书



扫码去微信小商城

技术支持网站：www.STCAI.com/

官方技术论坛：www.STCAIMCU.com

《STC8H-实验指导书》实验内容与源代码：www.STCAIMCU.com/thread-20134-1-1.html

资料更新日期：**2026/6/30**

(本文档可直接添加备注和标记)

目 录

1 搭建实验环境 13

1.1 STC8H 单片机性能概述 13

1.2 STC8H 单片机产品线 14

1.3 STC8H8K64U 内部结构图 15

1.4 安装 AiCube-ISP 下载/编程/烧录/软件 16

1.5 STC8H8K64U 实验箱实物图及程序下载 17

1.5.1 STC8H8K64U 实验箱实物图及与电脑连接实物图 17

1.5.2 硬件 USB 下载用户程序步骤 18

1.5.3 下载用户程序文件并观察实验现象 20

1.6 添加型号和头文件到 Keil 23

1.7 使用【AiCube】工具新建一个 8 位 8051 项目，在 AiCube-ISP 中 25

1.7.1 准备工作，见前面几小节 25

1.7.2 使用【AiCube】创建项目 25

1.7.3 增加实验项目代码 30

1.7.4 使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果 31

1.8 新建项目和项目设置，在 Keil 中 34

1.8.1 准备工作，见前面几小节 34

1.8.2 新建项目 34

1.8.2.1 新建工程 -- 在上方的菜单栏 Project 标签中，选择“New uVision Project”菜单项，开始新建工程。 34

1.8.2.2 设置项目路径和名称 -- 将文件存入建好的文件夹中 35

1.8.2.3 选择单片机型号（以 STC8H8K64U Series 为例） 35

1.8.2.4 添加源代码文件到项目 -- 给工程项目增加一个主程序文件 36

1.8.3 项目的各种基础选项设置 38

1.8.3.1 Device 选项卡--勾选“Use Extended Linker（LX51）instead of BL51”可支持 REMOVEUNUSED 参数 38

1.8.3.2 Output 选项卡--勾选 Create HEX File，选择 HEX-80 38

1.8.3.3 LX51 Misc 选项卡--在 Misc controls 输入框输入‘REMOVEUNUSED’ 39

1.8.3.4 Debug 选项卡-- 使用 STCAI 工具进行硬件仿真设置 39

1.9 了解 File、Build、Debug 工具栏与仿真相关的常用操作，在 Keil 中 41

1.9.1 File Toolbar（File 工具栏） 42

1.9.1.1 Start/Stop Debug Session，File Toolbar 42

1.9.1.2 Breakpoint，File Toolbar 42

1.9.1.3 Insert/Remove Breakpoint，File Toolbar 43

1.9.1.4 Enable/Disable Breakpoint，File Toolbar 44

1.9.1.5 Disable All Breakpoints，File Toolbar 44

1.9.1.6 Kill All Breakpoints，File Toolbar 45

1.9.2 Build Toolbar（Build 工具栏） 45
1.9.2.1 Translate（Translate the currently active file），Build Toolbar 45
1.9.2.2 Build（Build target files），Build Toolbar 46
1.9.2.3 Rebuild（Rebuild all target files），Build Toolbar 46
1.9.2.4 Batch Build（Build selected project targets），Build Toolbar 46
1.9.3 Debug Toolbar（Debug 工具栏） 47
1.9.3.1 RST（Reset the CPU），Debug Toolbar 47
1.9.3.2 Run（Start code execution），Debug Toolbar 47
1.9.3.3 Stop（Stop code execution），Debug Toolbar 47
1.9.3.4 Step（Step one line），Debug Toolbar 48
1.9.3.5 Step Over（Step over the current line），Debug Toolbar 48
1.9.4 观察变量/寄存器/内存（数据存储器、程序存储器）窗口 49

2 实验 50

2.1 实验一：printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序 50
2.1.1 printf( )，printf_usb( )实验程序代码 50
2.1.2 准备工作，如已准备，跳过此节 51
2.1.4 下载用户程序到实验箱，观察实验现象 54
2.1.4.1 连接实验箱和电脑 54
2.1.4.2 硬件 USB 下载用户程序步骤 55
2.1.4.3 下载用户程序文件并查看实验现象 57
2.1.5 使用【AiCube】工具创建 -- printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序 60
2.1.5.1 安装 AiCube-ISP 下载/编程/烧录/软件 60
2.1.5.2 使用【AiCube】工具创建项目 61
2.1.5.3 增加实验项目代码 65
2.1.5.4 使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果 65
2.1.6 USB 不停电下载，ISP 软件如何设置 67
2.1.6.1 手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载 67
2.1.6.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载 68

2.2 2 实验二：查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他 69
2.2.1 printf( )，printf_usb( )实验程序代码 69
2.2.2 准备工作，如已准备，跳过此节 70
2.2.3 下载用户程序到 STC8H8K64U 实验箱 73
2.2.4 观察实验现象 74
2.2.5 使用【AiCube】工具创建“查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他” 75
2.2.5.1 安装 AiCube-ISP 下载/编程/烧录/软件 75
2.2.5.2 使用【AiCube】创建项目 75
2.2.5.3 增加实验项目代码 79
2.2.5.4 使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果 80
2.2.6 USB 不停电下载，ISP 软件如何设置 82
2.2.6.1 手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载 82
2.2.6.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载 83

2.3 实验三：中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其

他 84
2.3.1 printf( )，printf_usb( )实验程序代码 84
2.3.2 查看实验现象 85
2.3.3 使用【AiCube】工具创建“中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其他” 86
2.3.3.1 安装 AiCube-ISP 下载/编程/烧录/软件 86
2.3.3.2 使用【AiCube】创建项目 86
2.3.3.3 增加实验项目代码 90
2.3.3.4 使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果 91
2.3.4 USB 不停电下载，ISP 软件如何设置 93
2.3.4.1 手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载 93
2.3.4.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载 94
2.4 点灯实验一：点亮 LED 灯，继续完成前面的工程项目 95
2.4.1 原理介绍 95
2.4.2 代码部分 96
2.4.3 下载用户程序并查看实验现象 97
2.5 点灯实验二：让 LED 灯闪烁起来 99
2.5.1 原理介绍 99
2.5.2 代码部分 100
2.5.3 下载用户程序并查看实验现象 102
2.6 点灯实验三：通过 USB-CDC 方式控制 STC8H8K64U 进行 LED 点灯 104
2.6.1 原理介绍 104
2.6.2 准备工作 105
2.6.3 代码部分 107
2.6.4 下载用户程序到 STC8H8K64U 实验箱并查看实验现象 109
2.6.4.1 点击“多字符串发送”窗口中的【点亮 P60-P63 灯】按钮 111
2.6.4.2 点击“多字符串发送”窗口中的【点亮 P64-P67 灯】按钮： 112
2.6.5 使用【AiCube】项目创建助手工具创建通过 USB-CDC 方式控制 STC8H8K64U 进行 LED 点灯 113
2.6.5.1 安装 AiCube-ISP 下载/编程/烧录/软件 113
2.6.5.2 使用【AiCube】助手创建项目 113
2.6.5.3 增加实验项目代码 117
2.6.5.4 使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果 118
2.7 点灯实验四：使用 STC8H8K64U 内置 USB 直接仿真流水灯 122
2.7.1 原理介绍 122
2.7.2 实验程序 C 语言代码 123
2.7.3 仿真设置，在 Keil 中 127
2.7.3.1 Debug 选项卡： 127
2.7.3.2 再打开“Utilities”选项卡，进行如下勾选设置： 128
2.7.4 使用 STC8H8K64U 实验箱仿真用户程序 128
2.7.4.1 制作仿真芯片 128
2.7.5 Keil 仿真并观察实验现象 130
2.7.5.1 使用 Run 按钮，观察全速运行效果的实验现象 131
2.7.5.2 观察外设寄存器数据与实验现象 131

2.7.6 实验程序汇编语言代码 136
2.7.7 使用【AiCube】项目创建助手工具创建使用 STC8H8K64U 内置 USB 仿真流水灯 139
2.7.7.1 安装 AiCube-ISP 下载/编程/烧录/软件 139
2.7.7.2 使用【AiCube】助手创建项目 139
2.7.7.3 增加实验项目代码 142
2.7.7.4 使用 AiCube-ISP 软件设置仿真芯片，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果 143
2.8 I/O 口实验 -- I/O 口读取按键状态并点灯 145
2.8.1 实验要求 145
2.8.2 实验建议 145
2.8.3 原理图 146
2.8.3.1 STC8H8K64U 实验箱原理图 146
2.8.3.2 认识实验箱 147
2.8.4 I/O 口工作模式和内部上拉/下拉电阻介绍 148
2.8.4.1 内部上拉电阻和外部上拉/下拉电阻的部分区别 148
2.8.4.2 准双向口介绍 148
2.8.4.3 强推挽输出 148
2.8.4.4 高阻输入 149
2.8.4.5 开漏模式 149
2.8.5 实验 1：位操作，用 USB 直接仿真来调试 150
2.8.5.1 I/O 口设置 150
2.8.5.2 实验程序 C 语言代码 151
2.8.5.3 制作仿真芯片、仿真设置 153
2.8.5.4 观察实验现象 153
2.8.5.5 实验程序汇编语言代码 154
2.8.6 实验 1A--使用【AiCube】项目助手实现“位操作，用 USB 仿真来点灯” 158
2.8.6.1 安装 AiCube-ISP 下载/编程/烧录/软件 158
2.8.6.2 使用【AiCube】助手创建项目 159
2.8.6.3 增加实验项目代码 163
2.8.6.4 使用 AiCube-ISP 软件设置仿真芯片，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果 164
2.8.7 实验 2：整组 8 位端口操作，用 USB 直接仿真来调试 165
2.8.7.1 实验程序 C 代码 165
2.8.7.2 制作仿真芯片、仿真设置 166
2.8.7.3 观察实验现象 167
2.8.8 实验 2A--使用【AiCube】项目助手实现“整组 8 位端口操作，用 USB 仿真来点灯” 168
2.8.8.1 安装 AiCube-ISP 下载/编程/烧录/软件 168
2.8.8.2 使用【AiCube】助手创建项目 169
2.8.8.3 增加实验项目代码 173
2.8.8.4 使用 AiCube-ISP 软件设置仿真芯片，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果 174
2.8.9 实验 3：用 USB-CDC 读取按键状态在串口助手显示并点亮 LED 灯 175
2.8.9.1 实验介绍 175
2.8.9.2 准备工作--添加头文件和库函数 175
2.8.9.3 实验程序 C 代码 177

2.8.9.4 程序下载，观察实验现象 179
2.8.10 实验 3A--使用【AiCube】项目助手实现“用 USB-CDC 读取按键状态在串口助手显示并点亮 LED 灯” 181
2.8.10.1 安装 AiCube-ISP 下载/编程/烧录/软件 181
2.8.10.2 使用【AiCube】助手创建项目 182
2.8.10.3 增加实验项目代码 187
2.8.10.4 使用 AiCube-ISP 软件系统，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果 187
2.8.11 实验 4：用 USB-CDC 读取按键状态，在 ISP 软件的 【开天斧/屠龙刀-LED-DIP64(6)】 虚拟显示 190
2.8.11.1 通信协议 190
2.8.11.2 实验介绍 190
2.8.11.3 准备工作--添加头文件和库函数 190
2.8.11.4 实验程序 C 代码 193
2.8.11.5 程序下载 194
2.8.11.6 程序实现效果，观察实验现象 195
2.8.12 实验 4A--使用【AiCube】项目助手实现“用 USB-CDC 读取按键状态，并在【开天斧/屠龙刀-LED-DIP64(6)】 虚拟显示” 198
2.8.12.1 安装 AiCube-ISP 下载/编程/烧录/软件 198
2.8.12.2 使用【AiCube】助手创建项目 199
2.8.12.3 增加实验项目代码 204
2.8.12.4 使用 AiCube-ISP 软件系统，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果 204
2.8.13 实验 5：I/O 口行列按键扫描，蜂鸣器发声实战 208
2.8.13.1 实验介绍 208
2.8.13.2 原理图 208
2.8.13.3 准备工作--添加头文件和库函数 208
2.8.13.4 实验程序 C 代码 211
2.8.13.5 程序下载 214
2.8.13.6 程序实现效果，观察实验现象 215
2.9 传统外部中断实验 -- INT0，INT1，INT2，INT3 217
2.9.1 中断实验 1(外部中断 INT0/P3.2，INT1/P3.3 控制点灯并 USB 仿真) 217
2.9.1.1 实验介绍 217
2.9.1.2 原理图 217
2.9.1.3 实验程序 C 语言代码 218
2.9.1.4 制作仿真芯片 220
2.9.1.5 仿真设置，在 Keil 中 222
2.9.1.5.1 设置 Debug 选项卡 222
2.9.1.5.2 设置 Utilities 选项卡 223
2.9.1.6 程序实现效果，观察实验现象 224
2.9.1.7 实验程序汇编语言代码 227
2.9.2 中断实验 2(单片机主动触发外部中断 INT2/P3.6，INT3/P3.7 控制点灯并 USB 仿真) 230
2.9.2.1 实验介绍 230
2.9.2.2 原理图 230
2.9.2.3 实验程序 C 代码 232

2.9.2.4 制作仿真芯片 234
2.9.2.5 仿真设置，在 Keil 中 235
2.9.2.5.1 设置 Debug 选项卡 236
2.9.2.5.2 设置 Utilities 选项卡 237
2.9.2.6 程序实现效果，观察实验现象 237

2.10 定时器实验 -- 定时器 0/1，16 位定时器 243
2.10.1 定时器 0/1，16 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息） 243
2.10.1.1 实验介绍 243
2.10.1.2 原理图 243
2.10.1.3 准备工作--添加头文件和库函数 244
2.10.1.4 实验程序代码 247
2.10.1.5 程序下载 250
2.10.1.6 程序实现效果，观察实验现象 250
2.10.2 使用【AiCube】工具实现：定时器 0/1，16 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息） 252
2.10.2.1 安装 AiCube-ISP 下载/编程/烧录/软件 252
2.10.2.2 使用【AiCube】创建项目 253
2.10.2.3 增加实验项目代码 258
2.10.2.4 使用 STC8H8K64U 实验箱观察实验现象 259
2.10.3 定时器 0/1，INT0/INT1，中断组合应用实验，演示程序 261

2.11 大型综合实战 -- 课程设计：时钟+数码管+矩阵按键+蜂鸣器发声 265
2.11.1 实验介绍 265
2.11.1.1 按键扫描说明 265
2.11.1.2 实验现象和操作说明 265
2.11.1.3 Keil 环境下多文件项目管理说明 266
2.11.2 原理图 266
2.11.3 流程图 268
2.11.3.1 main -- 程序主函数 流程图 268
2.11.3.2 key -- 按键扫描以及按键事件处理 流程图 269
2.11.3.3 led -- 扫描 LED 数码管 流程图 270
2.11.3.4 timer -- 定时器初始化以及定时器中断服务程序 流程图 270
2.11.3.5 rtc -- 软件模拟 RTC 程序 流程图 271
2.11.3.6 beep -- 蜂鸣器发声控制程序 流程图 273
2.11.4 实验程序代码 273
2.11.4.1 main.c -- 程序主函数 273
2.11.4.2 key.c -- 按键扫描以及按键事件处理 274
2.11.4.3 led.c -- 扫描 LED 数码管 276
2.11.4.4 timer.c -- 定时器初始化以及定时器中断服务程序 278
2.11.4.5 rtc.c -- 软件模拟 RTC 程序 279
2.11.4.6 beep.c -- 蜂鸣器发声控制程序 282
2.11.4.7 config.h -- 项目配置的头文件 283
2.11.4.8 key.h -- 项目配置文件的头文件 283
2.11.4.9 led.h -- 项目配置文件的头文件 283
2.11.4.10 timer.h -- 项目配置文件的头文件 284
2.11.4.11 rtc.h -- 项目配置文件的头文件 284

2.11.4.12 beep.h -- 蜂鸣器发声控制程序的头文件 284
2.11.5 程序下载，观察实验现象 284

2.12 RTC 实验 287
2.12.1 RTC 实验(拓展学习之使用 RTC 功能设定闹钟) 287
2.12.1.1 实验介绍 287
2.12.1.2 原理图 287
2.12.1.3 准备工作--添加头文件、Keil 中断拓展插件、库函数 288
2.12.1.4 实验程序代码 291
2.12.1.5 程序下载，观察实验现象 294
2.12.2 使用【AiCube】助手实现 RTC 实验(测试 RTC，使用 USB-CDC 打印年/月/日/时/分/秒) 296
2.12.2.1 实验介绍 296
2.12.2.2 原理图 296
2.12.2.3 准备工作--添加 Keil 中断拓展插件 296
2.12.2.4 安装 AiCube-ISP 下载/编程/烧录/软件 298
2.12.2.5 使用【AiCube】助手创建项目 299
2.12.2.6 增加实验项目代码 303
2.12.2.7 使用 AiCube-ISP 软件系统，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果 305

2.13 串口实验 -- 串口通讯(硬件 UART1 与电脑收发通讯) 307
2.13.1 实验介绍 307
2.13.1.1 认识 STC8H8K64U 实验箱 307
2.13.1.2 专门定制的 USB 转串口/TTL - TypeC 接口线 307
2.13.1.3 USB Link1D 工具 308
2.13.1.4 STC8H8K64U 实验箱由 TTL - TypeC 接口线通过 USB Link1D 工具与电脑连接图 308
2.13.1.5 Keil 环境下多文件项目管理说明 308
2.13.2 原理图 309
2.13.3 实验程序代码 309
2.13.5.1. main.c -- 程序主函数 309
2.13.5.2. uart.c -- 初始化 UART 接口 310
2.13.5.3. config.h -- 项目配置的头文件 312
2.13.5.4. uart.h -- 项目配置文件的头文件 313
2.13.4 程序下载，观察实验现象，验证实验效果 313

2.14 串口实验 -- 串口通讯(硬件 UART2 与 UART3 收发通讯) 316
2.14.1 实验介绍 316
2.14.1.1 认识 STC8H8K64U 实验箱 316
2.14.1.2 2 根杜邦线（1 公头、1 母头） 316
2.14.1.3 USB 转串口/TTL 连接线 317
2.14.1.4 USB 转双串口工具@Ai8H2K12U：正面 317
2.14.1.5 本实验实验箱与电脑连接实物图 317
2.14.1.6 Keil 环境下多文件项目管理说明 318
2.14.2 原理图 318
2.14.3 实验程序代码 319
2.14.3.1 main.c -- 程序主函数 319
2.14.3.2 uart.c -- 初始化 UART 接口 320

2.14.3.3 timer.c -- 定时器初始化以及定时器中断服务程序 322
2.14.3.4 config.h -- 项目配置的头文件 323
2.14.3.5 uart.h -- 项目配置文件的头文件 323
2.14.3.6 timer.h -- 项目配置文件的头文件 324
2.14.4 程序下载，观察实验现象，验证实验效果 324

2.15 串口实验 -- 串口通讯(硬件 UART2 与 UART3 数据透传) 327
2.15.1 实验介绍 327
2.15.1.1 认识 STC8H8K64U 实验箱 327
2.15.1.2 5 根杜邦线（1 公头、1 母头） 328
2.15.1.3 USB 转串口/TTL 连接线 328
2.15.1.4 USB 转双串口工具@Ai8H2K12U：正面 328
2.15.1.5 USB 转双串口工具@Ai8H2K12U：背面 328
2.15.1.6 本实验实验箱与电脑连接实物图 329
2.15.1.7 Keil 环境下多文件项目管理说明 330
2.15.2 原理图 330
2.15.3 实验程序代码 331
2.15.3.1 main.c -- 程序主函数 331
2.15.3.2 uart.c -- 初始化 UART 接口 332
2.15.3.3 config.h -- 项目配置的头文件 334
2.15.3.4 uart.h -- 项目配置文件的头文件 334
2.15.4 程序下载、观察实验现象，验证实验效果 335

2.16 串口实验 -- 串口 1_2_3_4(中断设置标志位，主循环查询)，演示实验 339

2.17 串口实验 -- 串口 1_2_3_4 + 定时器 0_1_3_4 + 外部中断 INT0_1_2_3 (中断设置标志位，主循环查询)，演示实验 345

2.18 串口实验 -- 串口通讯(串口通讯，串口绘图，ADC 检测按键，ADC 热敏电阻测温) 356
2.18.1 实验介绍 356
2.18.1.1 认识 STC8H8K64U 实验箱 356
2.18.1.2 2 根杜邦线（1 公头、1 母头） 357
2.18.1.3 USB 转串口/TTL 连接线 357
2.18.1.4 USB 转双串口工具@Ai8H2K12U：正面 357
2.18.1.5 USB 转双串口工具@Ai8H2K12U：背面 357
2.18.1.6 本实验实验箱与电脑连接实物图 358
2.18.1.7 Keil 环境下多文件项目管理说明 358
2.18.2 原理图 359
2.18.3 实验程序代码 360
2.18.3.1 main.c -- 程序主函数 360
2.18.3.2 uart.c -- 初始化 UART 接口 361
2.18.3.3 adc.c -- ADC 初始化和 ADC 数据采集 361
2.18.3.4 timer.c -- 定时器初始化以及定时器中断服务程序 362
2.18.3.5 config.h -- 项目配置的头文件 363
2.18.3.6 uart.h -- 项目配置文件的头文件 363
2.18.3.7 adc.h -- 项目配置文件的头文件 364
2.18.3.8 timer.h -- 项目配置文件的头文件 364
2.18.4 程序下载，串口绘图相关设置 364

2.18.5 观察实验现象，验证实验效果 367

2.19 大型综合实战 -- 模拟 RTC 时钟+数码管+ADC 检测按键(查询) 368
2.19.1 实验介绍 368
2.19.1.1 ADC 键盘计算键码说明 368
2.19.1.2 ADC 按键实验现象和操作说明 368
2.19.1.3 按键扫描说明 369
2.19.1.4 Keil 环境下多文件项目管理说明 369
2.19.2 原理图 370
2.19.2.1 ADC 按键原理图 370
2.19.2.2 P6、P7 口控制数码管原理图 370
2.19.3 实验程序代码 371
2.19.3.1 main.c -- 程序主函数 371
2.19.3.2 key.c -- 按键扫描以及按键事件处理 372
2.19.3.3 led.c -- 扫描 LED 数码管 374
2.19.3.4 timer.c -- 定时器初始化以及定时器中断服务程序 376
2.19.3.5 rtc.c -- 软件模拟 RTC 程序 377
2.19.3.6 adc.c -- ADC 初始化和 ADC 查询方式检测按键 381
2.19.3.7 config.h -- 项目配置的头文件 382
2.19.3.8 key.h -- 项目配置文件的头文件 382
2.19.3.9 led.h -- 项目配置文件的头文件 382
2.19.3.10 timer.h -- 项目配置文件的头文件 383
2.19.3.11 rtc.h -- 项目配置文件的头文件 383
2.19.3.12 adc.h -- 项目配置文件的头文件 383
2.19.4 程序下载，观察实验现象，验证实现效果 384

2.20 ADC 实验 -- 数码管+ NTC 热敏电阻+ADC 测温(中断方式) 386
2.20.1 实验介绍 386
2.20.1.1 ADC 通过采集 NTC 热敏电阻的电压值进行测温 386
2.20.1.2 操作说明和实验现象 387
2.20.1.3 Keil 环境下多文件项目管理说明 388
2.20.2 原理图 389
2.20.2.1 NTC 温度检测原理图 389
2.20.2.2 P6、P7 口控制数码管原理图 389
2.20.3 实验程序代码 390
2.20.3.1 main.c -- 程序主函数 390
2.20.3.2 led.c -- 扫描 LED 数码管 391
2.20.3.3 timer.c -- 定时器初始化以及定时器中断服务程序 393
2.20.3.4 adc.c -- ADC 初始化和 ADC 中断方式数据采集程序 393
2.20.3.5 ntc.c -- ADC 中断方式检测 NTC 热敏电阻测温程序 394
2.20.3.6 config.h -- 项目配置的头文件 400
2.20.3.7 led.h -- 项目配置文件的头文件 400
2.20.3.8 timer.h -- 项目配置文件的头文件 401
2.20.3.9 adc.h -- 项目配置文件的头文件 401
2.20.3.10 ntc.h -- 项目配置文件的头文件 401
2.20.4 程序下载，观察实验现象 402

2.21 大型综合实战 -- 课程设计：硬件 SPI 读写串行 Flash 404
2.21.1 实验介绍 404
2.21.1.1 认识 STC8H8K64U 的实验箱 404
2.21.1.2 专门定制的 USB 转串口/TTL - TypeC 接口线 404
2.21.1.3 USB Link1D 工具 405
2.21.1.4 STC8H8K64U 实验箱通过 USB Link1D 工具与电脑连接图 405
2.21.1.5 按键去抖动说明 405
2.21.1.6 Keil 环境下多文件项目管理说明 406
2.21.2 原理图 406
2.21.3 实验程序代码 407
2.21.3.1 main.c -- 程序主函数 407
2.21.3.2 spi.c -- 初始化 SPI 接口、串行 Flash 操作函数 408
2.21.3.3 uart.c -- 初始化 UART 接口 413
2.21.3.4 key.c -- 按键扫描以及按键事件处理 413
2.21.3.5 timer.c -- 定时器初始化以及定时器中断服务程序 416
2.21.3.6 config.h -- 项目配置的头文件 416
2.21.3.7 spi.h -- 项目配置文件的头文件 417
2.21.3.8 uart.h -- 项目配置文件的头文件 418
2.21.3.9 key.h -- 项目配置文件的头文件 418
2.21.3.10 timer.h -- 项目配置文件的头文件 418
2.21.4 程序下载，CDC 串口助手相关设置 419
2.21.5 观察实验现象，验证实验效果 421

2.22 SPI 实验 -- 硬件 SPI 驱动 OLED 显示图片、图形、字符 422
2.22.1 实验介绍 422
2.22.1.1 AiCube-ISP 系统软件图片取模工具的使用 422
2.22.1.2 AiCube-ISP 系统软件字库生成工具的使用 423
2.22.1.3 使用 AiCube-ISP 系统软件图片取模工具生成 ASCII 的点阵字符 C 代码 424
2.22.1.4 认识 STC8H8K64U 实验箱和 OLED 屏 426
2.22.1.5 Keil 环境下多文件项目管理说明 426
2.22.2 原理图 427
2.22.3 实验程序代码 427
2.22.3.1 main.c -- 程序主函数 427
2.22.3.2 oled.c -- SPI 驱动 OLED 显示图片、图形、字符 429
2.22.3.3 config.h -- 项目配置的头文件 441
2.22.3.4 oled.h -- 项目配置文件的头文件 441
2.22.3.5 font.h -- 项目配置文件的头文件 442
2.22.3.6 pic.h -- 项目配置文件的头文件 443
2.22.3.7 ascii.h -- 项目配置文件的头文件 446
2.22.4 程序下载，观察实验现象，验证实验效果 451

3 仿真 -- STC8H8K64U 实验箱仿真 453

3.1 USB 直接通信仿真 -- STC8H8K64U 实验箱仿真 453
3.1.1 安装 AiCube-ISP 软件，制作仿真芯片 453
3.1.1.1 安装 AiCube-ISP 下载/编程/烧录 工具软件 453

3.1.1.2 STC8H8K64U 实验箱实物图及与电脑连接实物图 454
3.1.1.3 硬件 USB 下载用户程序步骤 456
3.1.1.4 制作仿真芯片 457
3.1.2 Debug 和 Utilities 选项设置--在 Keil 集成环境中 459
3.1.2.1 Debug 选项卡--使用 STCAI 工具进行硬件仿真设置 460
3.1.2.2 Utilities 选项卡--勾选“Use Debug Driver” 461
3.1.3 进入 Debug 界面，观察实验现象 461
3.2 串口通信仿真 -- STC8H8K64U 实验箱仿真 466
3.2.1 安装 AiCube-ISP 软件，制作仿真芯片 466
3.2.1.1 安装 AiCube-ISP 下载/编程/烧录 工具软件 466
3.2.1.2 STC8H8K64U 实验箱完整实物图（含外壳）：正面 467
3.2.1.3 USB 转双串口工具@Ai8H2K12U：正面 468
3.2.1.4 USB 转双串口/TTL 连接线：正面 468
3.2.1.5 STC8H8K64U 实验箱通过 USB 转双串口工具与电脑连接实物图 468
3.2.1.6 制作仿真芯片 469
3.2.2 Debug 和 Utilities 选项设置--在 Keil 集成环境中 471
3.2.2.1 Debug 选项卡--使用 STCAI 工具进行硬件仿真设置 471
3.2.2.2 Utilities 选项卡--勾选“Use Debug Driver” 472
3.2.3 进入 Debug 界面，观察实验现象 472

更新记录 477

1 搭建实验环境

1.1 STC8H 单片机性能概述

STC8H 系列单片机是不需要外部晶振和外部复位的单片机，是以超强抗干扰/超低价/高速/低功耗为目标的 8051 单片机，在相同的工作频率下，STC8H 系列单片机比传统的 8051 约快 12 倍（速度快 11.2~13.2 倍），依次按顺序执行完全部的 111 条指令，STC8H 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8H 系列单片机是 STC 生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/强抗静电/较强抗干扰的新一代 8051 单片机，超级加密。指令代码完全兼容传统 8051。

MCU 内部集成高精度 R/C 时钟(±0.3%，常温下+25℃)，−0.88%~+1.05%温漂(-20℃~+65℃)，−1.38%~+1.42%温漂(-40℃~+85℃)，−3%~+3%温漂(-40℃~+125℃)。ISP 编程时工作频率可设置，可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP 编程时 4 级复位门槛电压可选)。

MCU 内部有 3 个可选时钟源：内部高精度 IRC 时钟（可适当调高或调低）、内部 32KHz 的低速 IRC、外部 4M~33M 晶振或外部时钟信号。用户代码中可自由选择时钟源，时钟源选定后可再经过 8-bit 的分频器分频后再将时钟信号提供给 CPU 和各个外设（如定时器、串口、SPI 等）。

MCU 提供两种低功耗模式：IDLE 模式和 STOP 模式。IDLE 模式下，MCU 停止给 CPU 提供时钟，CPU 无时钟，CPU 停止执行指令，但所有的外设仍处于工作状态，此时功耗约为 1.3mA（6MHz 工作频率）。STOP 模式即为主时钟停振模式，即传统的掉电模式/停电模式/停机模式，此时 CPU 和全部外设都停止工作，功耗可降低到 0.6uA@Vcc=5.0V，0.4uA@Vcc=3.3V。

掉电模式可以使用 INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、T3(P0.4)、T4(P0.6)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.6)、RXD3(P0.0/P5.0)、RXD4(P0.2/P5.2)、I2C_SDA(P1.4/P2.4/P3.3)以及比较器中断、低压检测中断、掉电唤醒定时器唤醒。

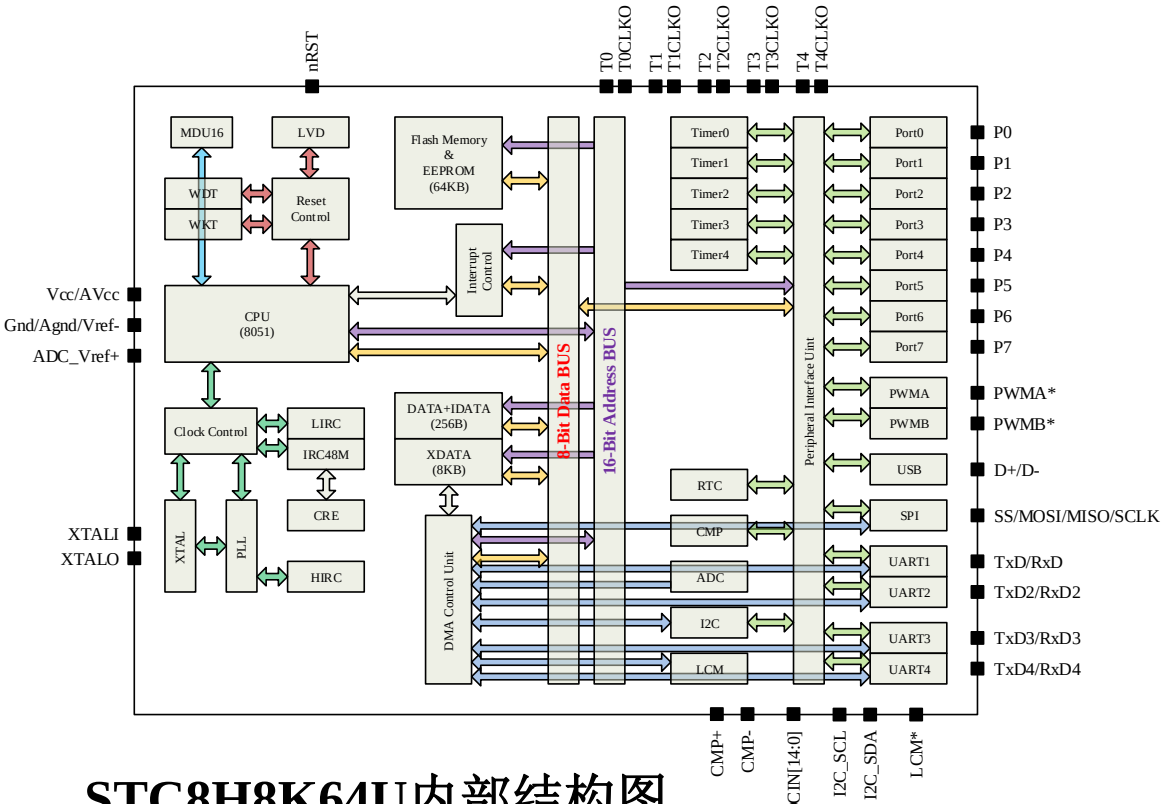
MCU 提供了丰富的数字外设（串口、定时器、高级 PWM 以及 I^2C、SPI、USB）接口与模拟外设（超高速 ADC、比较器），可满足广大用户的设计需求。

STC8H 系列单片机内部集成了增强型的双数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能。

1.2 STC8H 单片机产品线

产品线	I/O	UART	定时器	ADC	高级 PWM	PCA	CMP	SPI I2C	USB	MDU16	MDU32C	MDU32D	CRC16	LED 驱动	Touch Key	RTC	I/O 中断	彩屏 LCM	LCD 驱动	DMA
STC8H1K08 系列	17	2	3	$9_{CH}*10_{B}$	●		●	●												
STC8H1K28 系列	29	2	5	$12_{CH}*10_{B}$	●		●	●												
STC8H3K64S4 系列	45	4	5	$12_{CH}*12_{B}$	●		●	●		●							●			
STC8H3K64S2 系列	45	2	5	$12_{CH}*12_{B}$	●		●	●		●							●			
STC8H8K64U-A 版本	60	4	5	$15_{CH}*12_{B}$	●		●	●	●	●										
STC8H8K64U-B/C/D 版本	60	4	5	$15_{CH}*12_{B}$	●		●	●	●	●						●	●	●		●
STC8H4K64TL 系列	44	4	5	$15_{CH}*12_{B}$	●		●	●		●				●	●	●	●	●		●
STC8H4K64TLCD 系列	60	4	5	$15_{CH}*12_{B}$	●		●	●		●					●	●	●	●	●	●
STC8H1K08T 系列	16	2	3	$15_{CH}*12_{B}$	●		●	●		●					●	●	●			●
STC8H2K12U 系列	16	2	4	$15_{CH}*12_{B}$	●		●	●	●	●						●	●			
STC8H2K12U-A 系列	16	2	4	$15_{CH}*12_{B}$	●		●	●	●	●						●	●			
STC8H2K12U-B 系列	16	2	4	$15_{CH}*12_{B}$	●		●	●	●	●		●	●			●	●			
STC8H2K32U 系列	28	2	6	$15_{CH}*12_{B}$	●		●	●	●		●		●			●	●			
STC8G1K08-SOP8 系列	6	1	2					●		●										
STC8G1K08A-SOP8 系列	6	1	2	$6_{CH}*10_{B}$		●		●		●										

1.3 STC8H8K64U 内部结构图



STC8H8K64U内部结构图

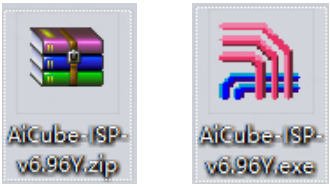
单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	idata，内部传统 8051 RAM 字节	xdata，内部大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	所有的 I/O 口均支持中断并可掉电唤醒	INT0 / INT1 / INT2 / INT3 / INT4 上升沿 / 下降沿中断 可掉电唤	DMA 8080/6800 接口 LCM 模块驱动(8 位和 16 位)	RTC 实时时钟	DMA 串口并可掉电唤醒	全速 USB	DMA SPI 并可掉电唤醒	I²C 并可掉电唤醒	MDU16 硬件 16 位乘除法器	定时器/计数器（T0/T1/T2/T3/T4 外部管脚也可掉电唤醒）	16 位高级 PWM 定时器 互补对称死区控制	掉电唤醒专用定时器	DMA 15 路高速 ADC（8 路 PWM 可当 8 路 D/A 使用）	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（45MHz 以下可调）追频	可对外输出时钟及复位	可设置下次更新程序需口令	支持 RS485 下载	硬件 USB－HID 免驱下载 和 USB 直接仿真	价格及封装								供货信息
																															程序加密后传输（防拦截），结合发布用户项目下载程序，超级绝配	LQFP64 /QFN64	LQFP48/QFN48	LQFP44	LQFP32/QFN32	SOP28<18mm*7.5mm>	TSSOP28 <9.7mm*4.4mm>	TSSOP20 <6.5mm*6.5mm>	SOP16	
STC8H8K32U	1.9-5.5	32K	256	8K	2	32K	60	有	有	有	有	4	有	有	有	有	5	8	有	12 位	有	有	有	4 级	有	是	是	是	是	¥2.0	¥1.6	¥1.7	¥1.6	暂无		¥1.4	¥1.4	现货
STC8H8K64U	1.9-5.5	64K	256	8K	2	IAP	60	有	有	有	有	4	有	有	有	有	5	8	有	12 位	有	有	有	4 级	有	是	是	是	是	¥2.5	¥2.2	¥2.3	¥1.9	暂无	¥1.8	¥1.4	¥1.4	

1.4　安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

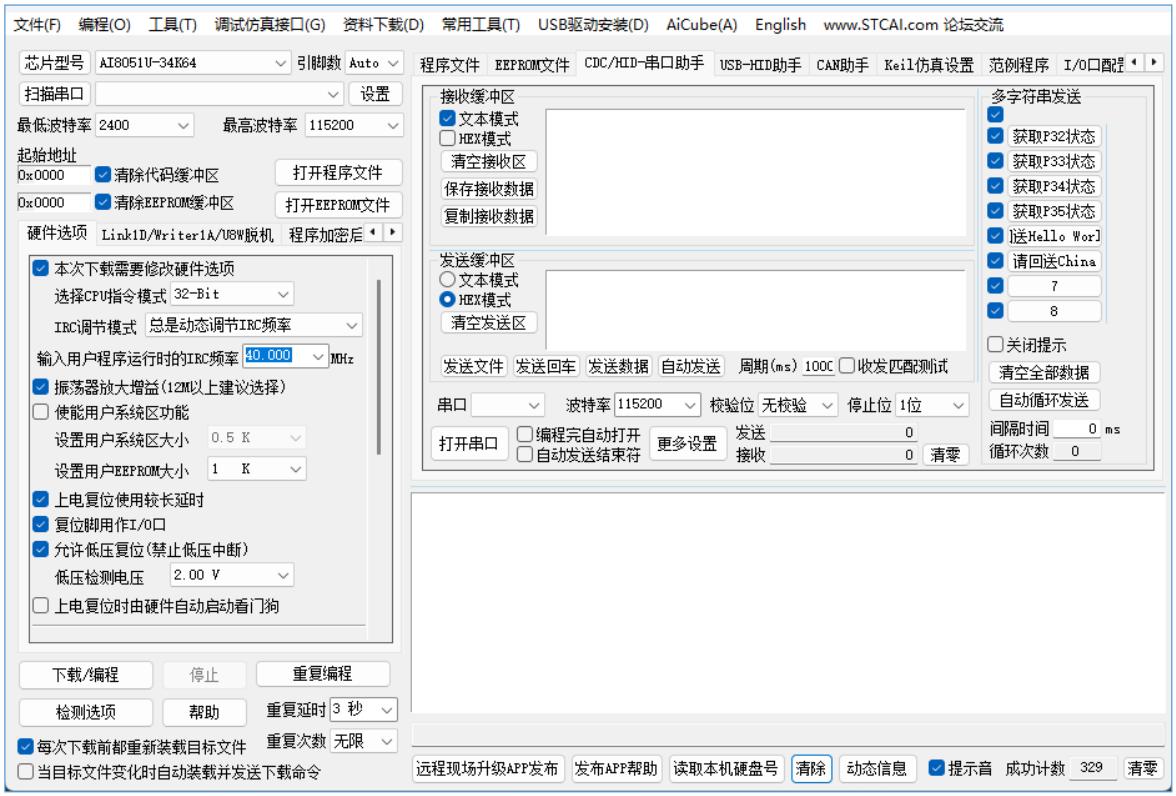


下载下来的是压缩包文件图标和解压缩后的 AiCube-ISP 文件图标，如下图：



如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

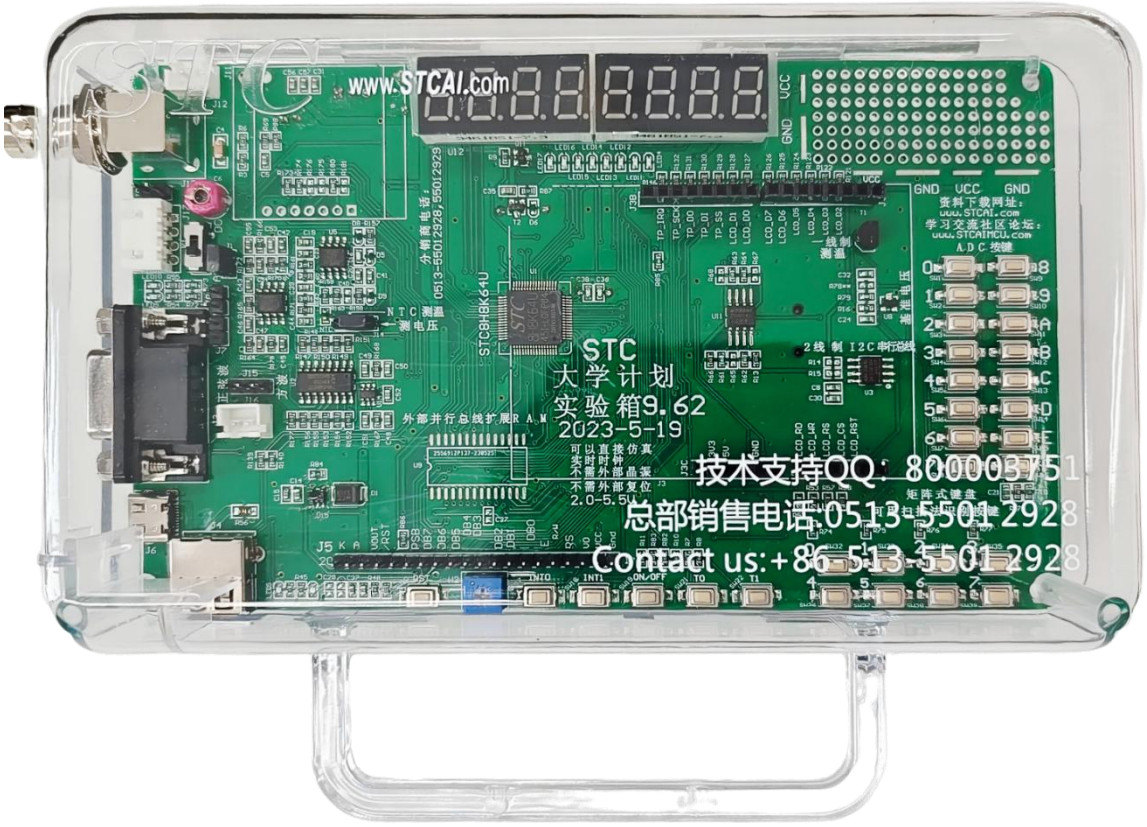
双击 AiCube-ISP 系统软件图标，打开 AiCube-ISP 软件界面，即可直接使用，如下图：



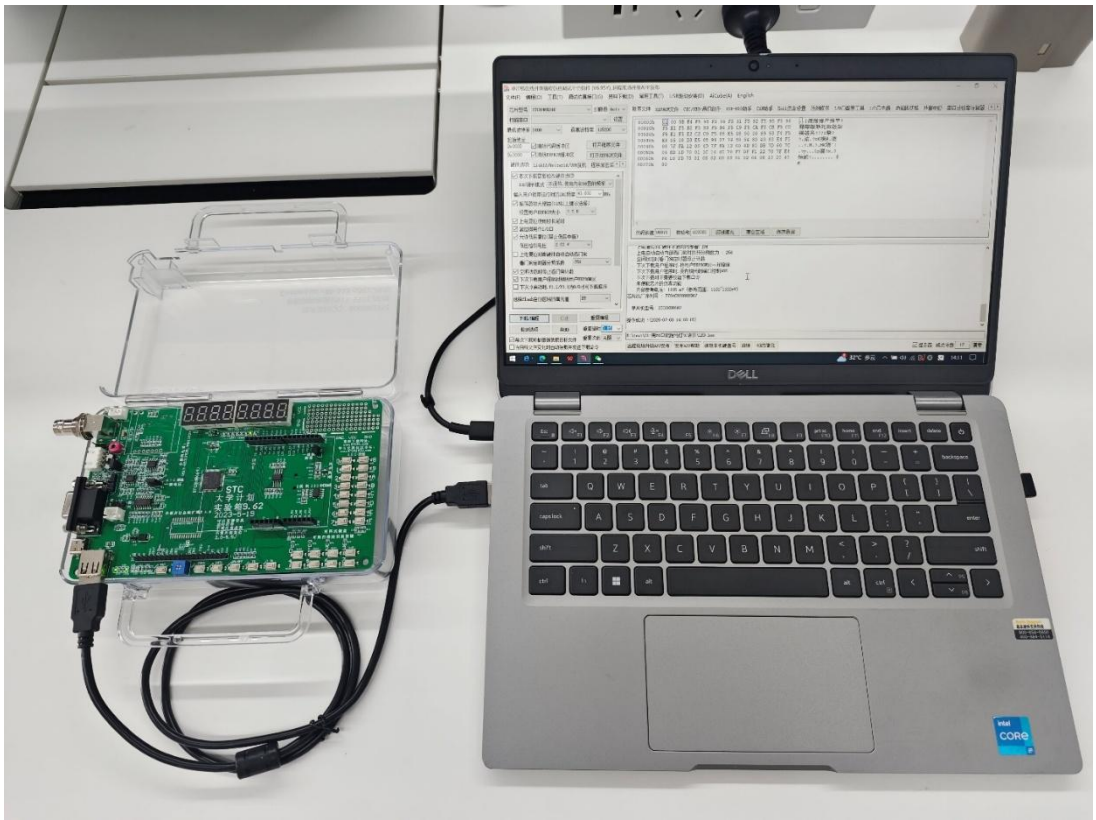
1.5 STC8H8K64U 实验箱实物图及程序下载

1.5.1 STC8H8K64U 实验箱实物图及与电脑连接实物图

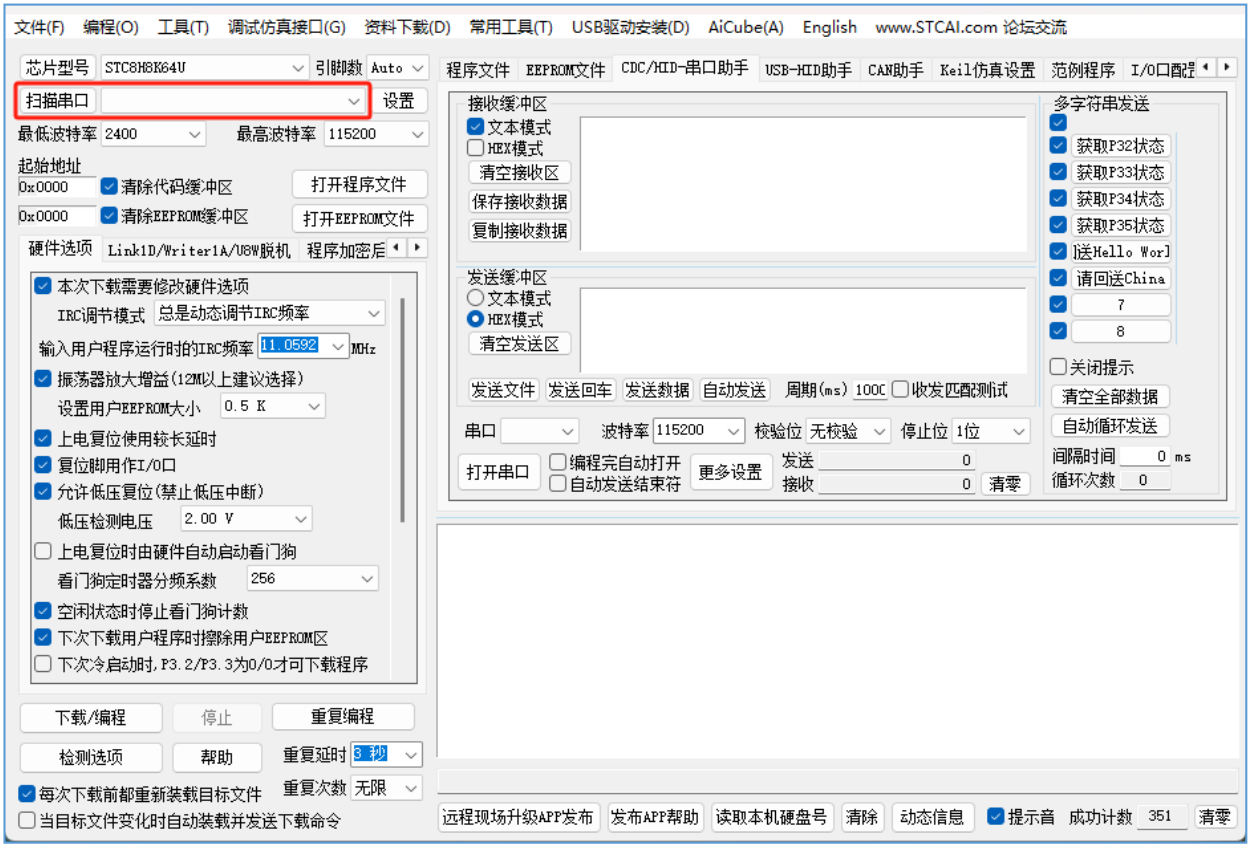
➲ STC8H8K64U 实验箱完整实物图（含外壳）：正面



➲ STC8H8K64U 实验箱与电脑连线实物图：USB-TypeA 到 USB-TypeA 线将实验箱与电脑正确连接。

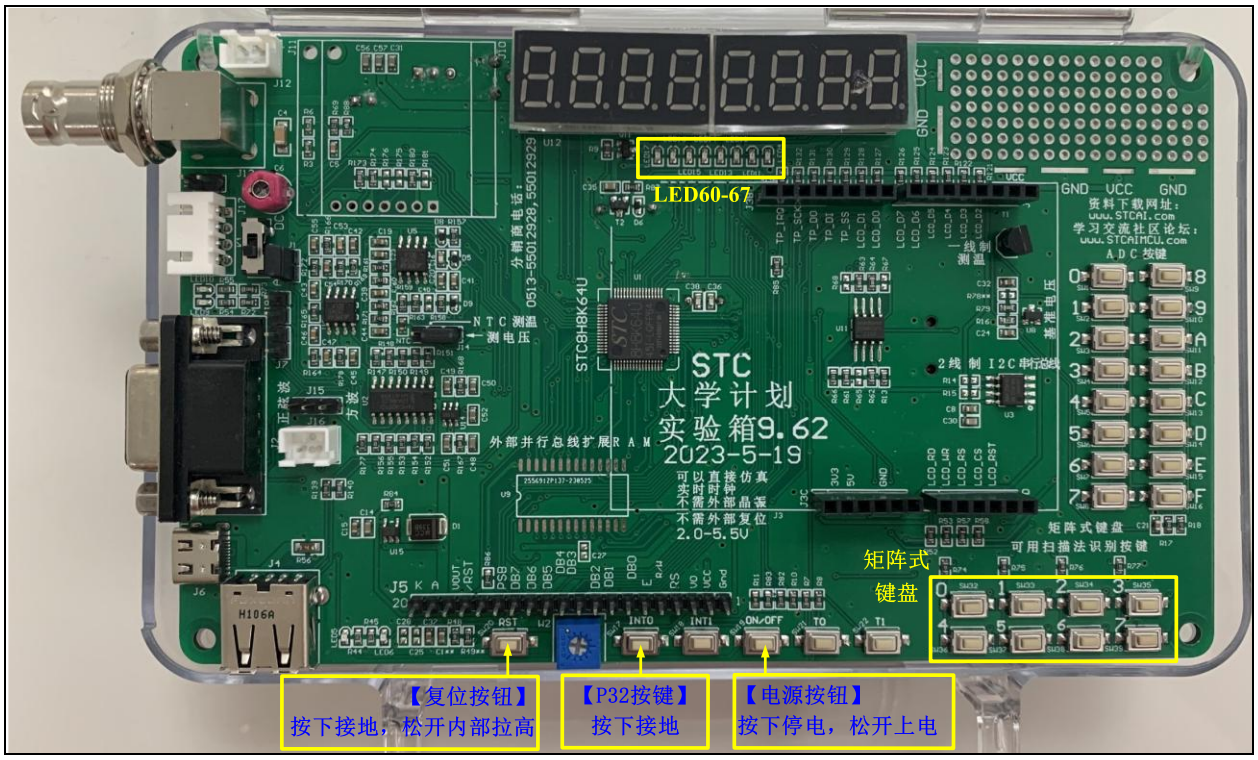


现在运行 AiCube-ISP-v6.95Z 或以上版本。选择单片机型号：STC8H8K64U，如下图，我们看到“扫描串口”后面的端口列表框不是“（HID1）USB-Writer”，说明还不能下载程序到实验箱。

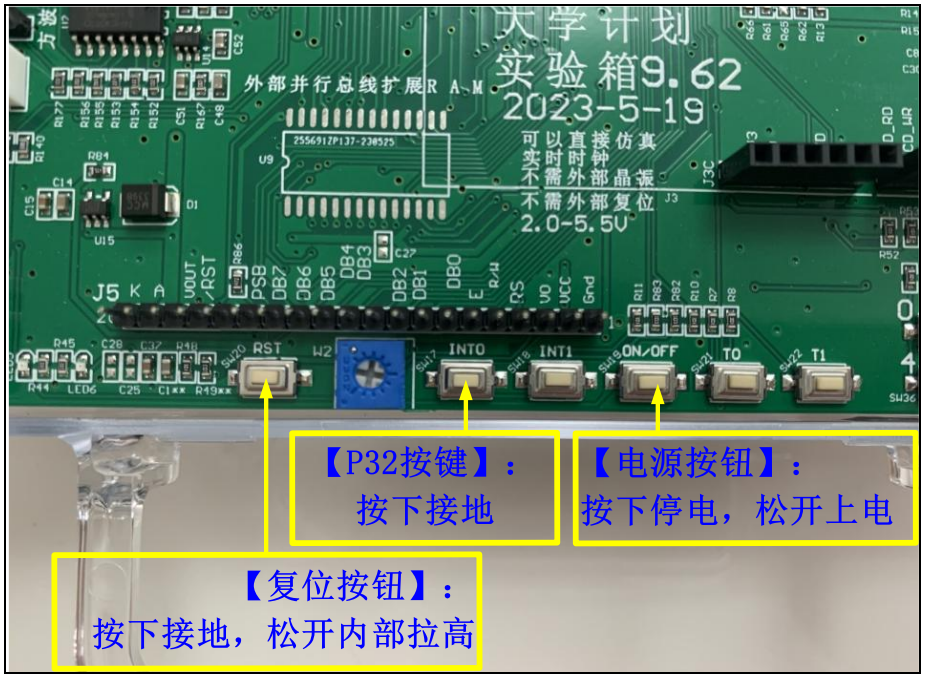


1.5.2　硬件 USB 下载用户程序步骤

下图是 STC8H8K64U 实验箱打开外盒的正面高清实物图：



局部放大实验箱图片，其正下方有两排按键，找到 P3.2 按键和 Power_SW 按键，如下图：



【方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===本 STC8H8K64U 实验箱是：按下电源按钮停电，再松开电源按钮上电，

这时电脑端 ISP 下载软件会自动识别出“(HID1) USB Writer”，此时就与 P3.2 状态无关了，可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

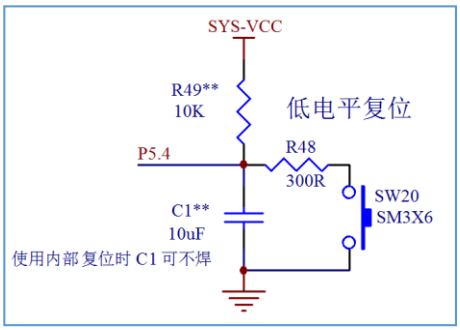
3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

电脑端 ISP 下载软件提示：下载进行中…，几秒钟后，提示下载成功！

【方法二：复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P5.4-nRST 出厂时默认是 P5.4-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。



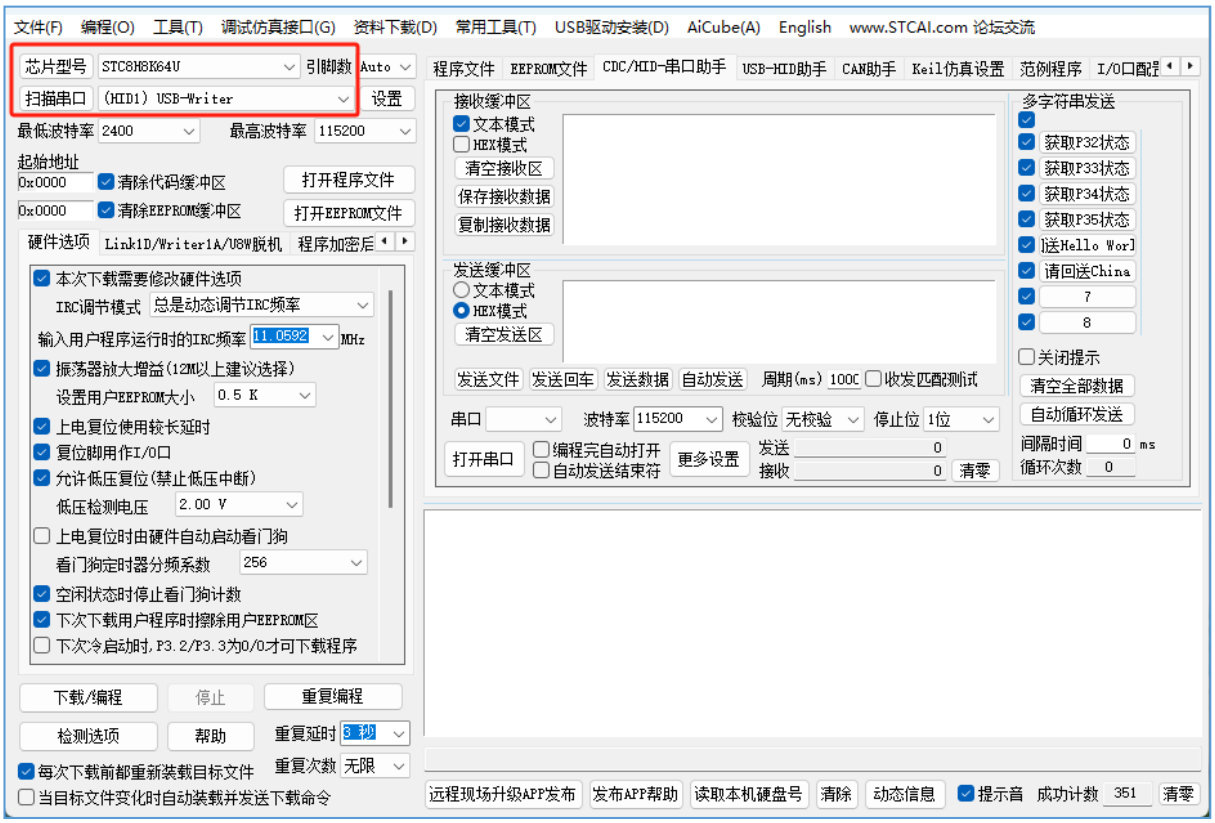
1、按下复位管脚/P5.4-nRST 外接的低电平复位按键，复位 MCU，再松开复位键，MCU 首先从系统程序区启动运行系统程序，判断是否要下载用户程序，

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮

电脑端 ISP 下载软件提示：下载进行中…，下载成功！

备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

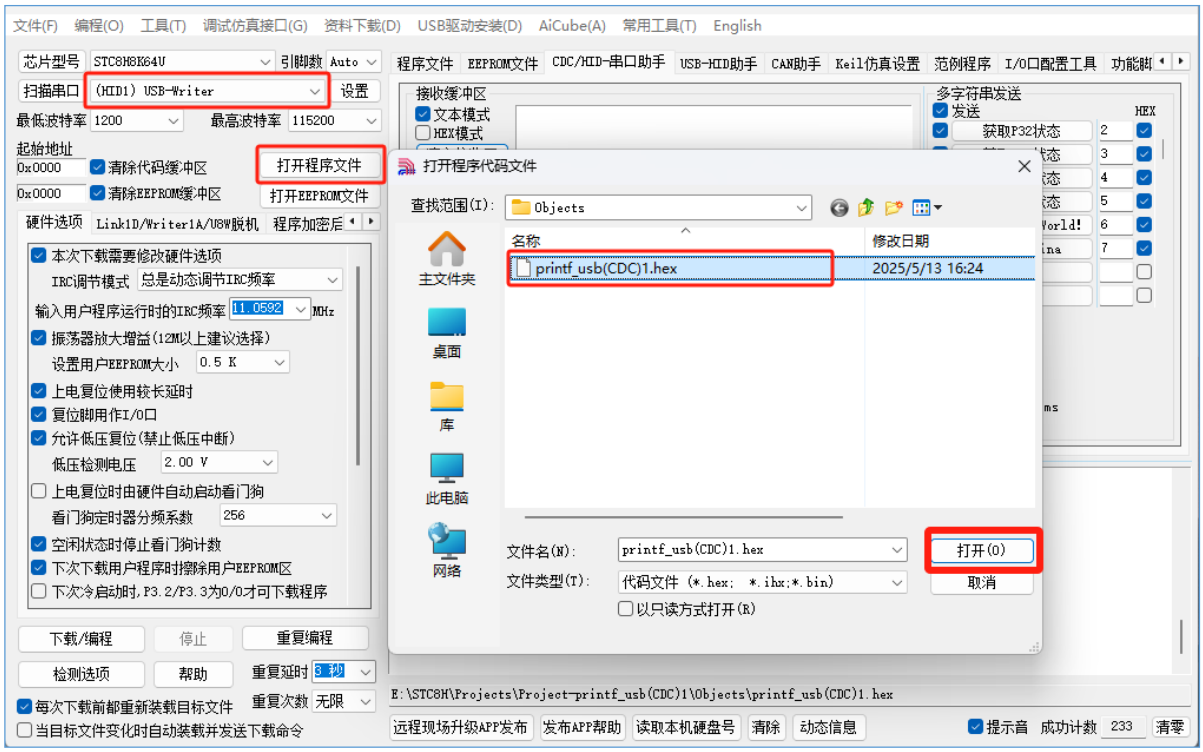


通过上述方法一或者方法二的操作步骤，我们看到 AiCube-ISP 系统软件界面的“扫描串口”的端口列表框显示“（HID1）USB-Writer”，表明 STC8H8K64U 实验箱已准备好，可以下载用户程序了。

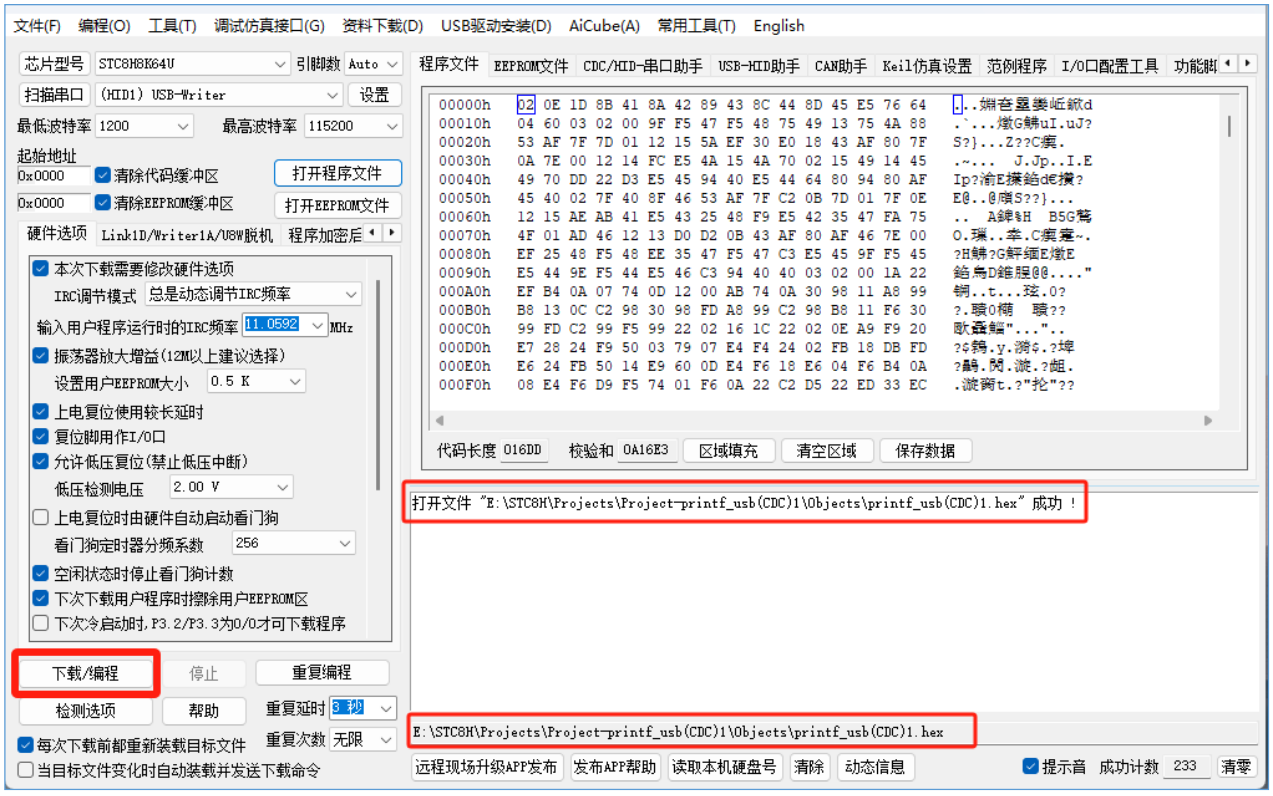
1.5.3　下载用户程序文件并观察实验现象

以“实验一：printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序”为例：

点击 AiCube-ISP 软件界面上的“打开程序文件”：print_usb(CDC)1.hex（“打开程序文件”打开的是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中）：

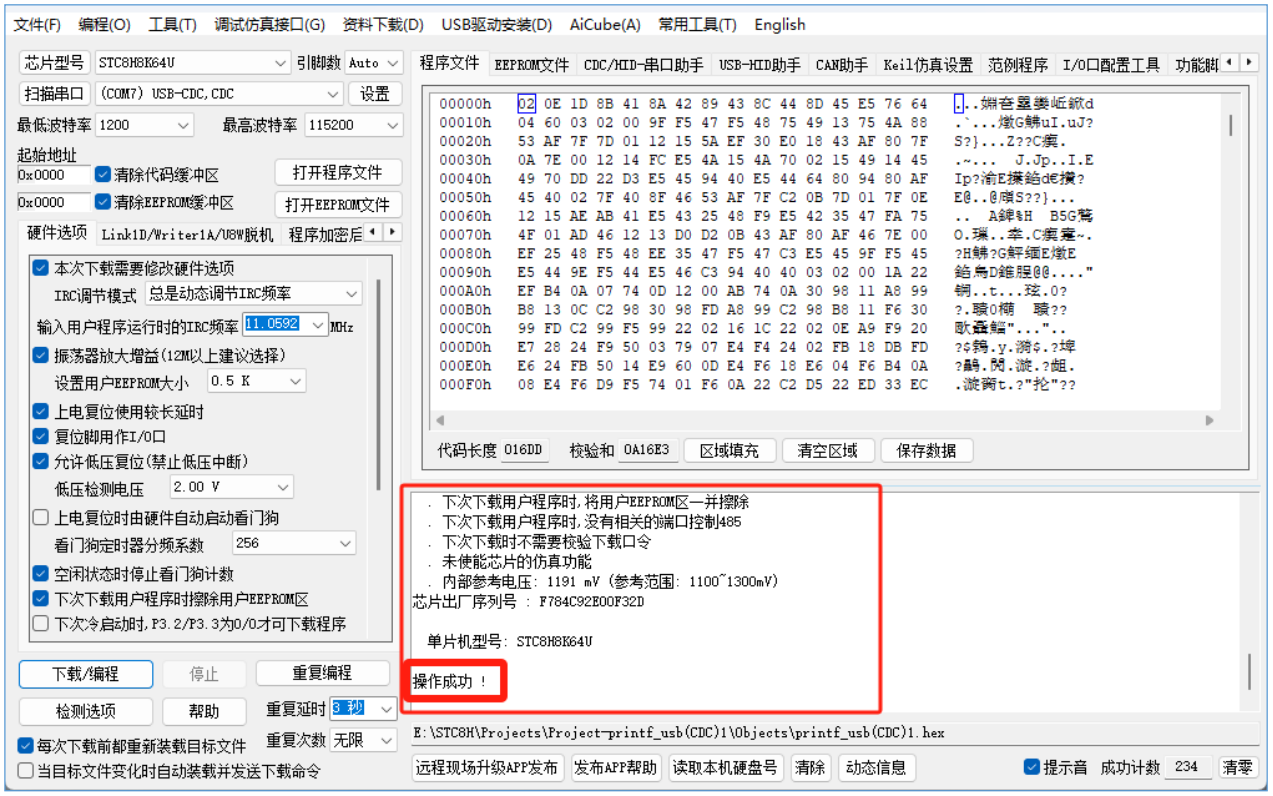


然后，AiCube-ISP 软件界面如下图所示，出现“‘打开文件***.hex’成功!”

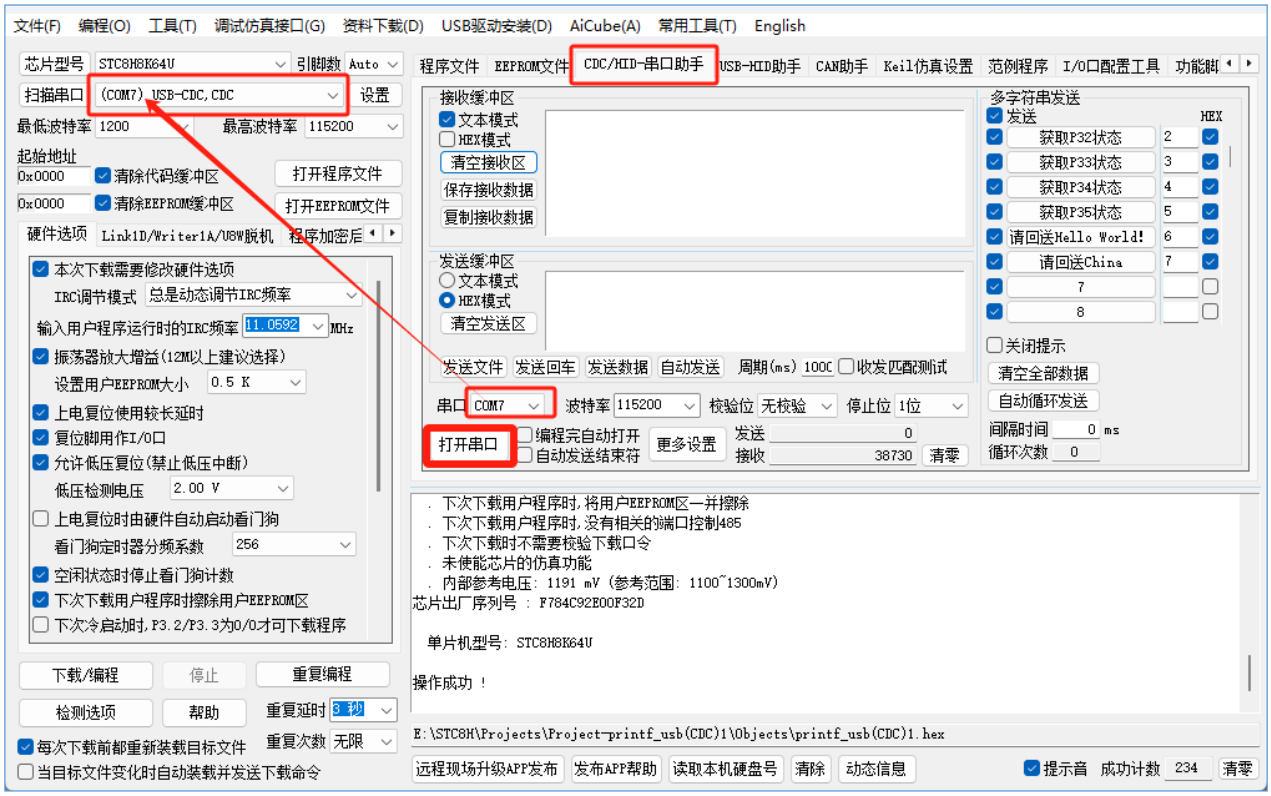


点击“下载/编程”按钮，把 print_usb(CDC)1.hex 烧录到实验箱，AiCube-ISP 软件界面出现：操作成功！

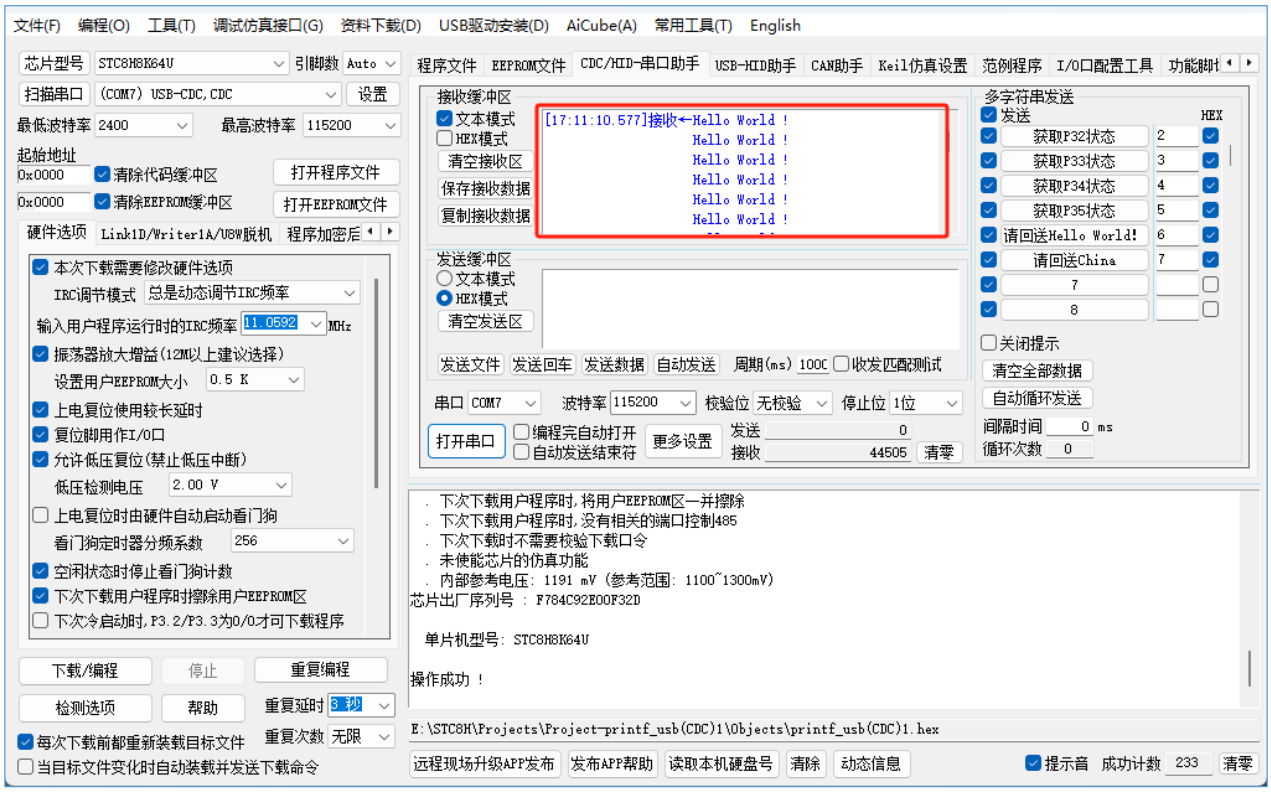
恭喜你！



我们看到此时串口显示:（COM7）USB-CDC，CDC，选择 AiCube-ISP 系统软件的“CDC/HID-串口助手”选项卡，找到“打开串口”按钮，如下图：



选择好对应的串口号，点击“打开串口”按钮，CDC/HID-串口助手的“接收缓冲区”，就会不停的显示“Hello World!”，如下图：

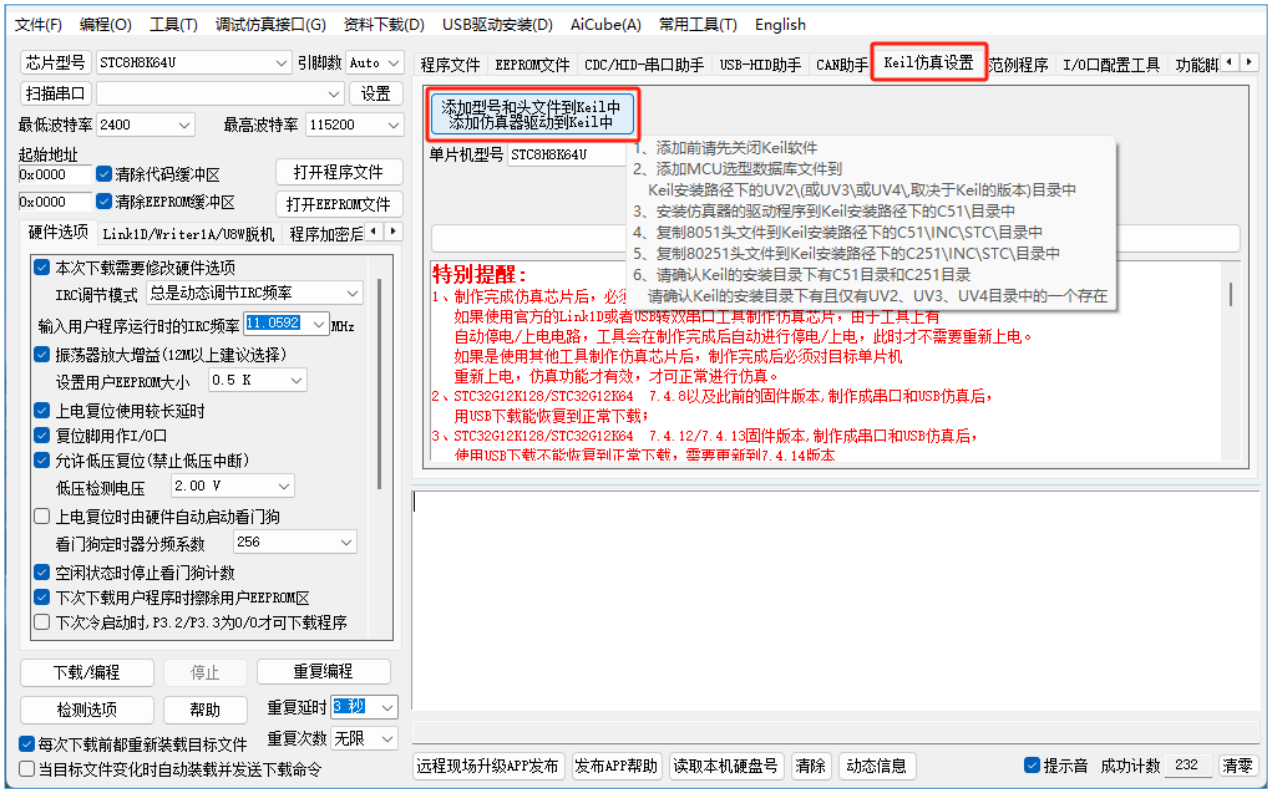


至此，printf_usb()显示“Hello World!”实验成功！

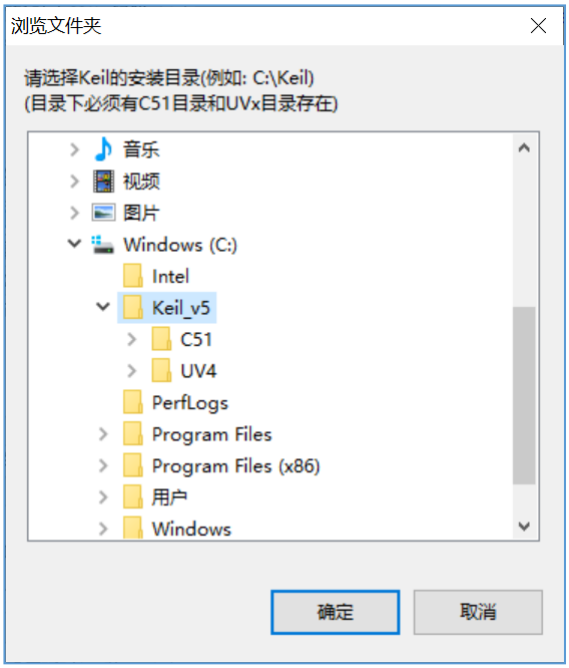
1.6 添加型号和头文件到 Keil

使用 Keil 之前需要先安装 STC 的仿真驱动。STC 的仿真驱动的安装步骤如下：

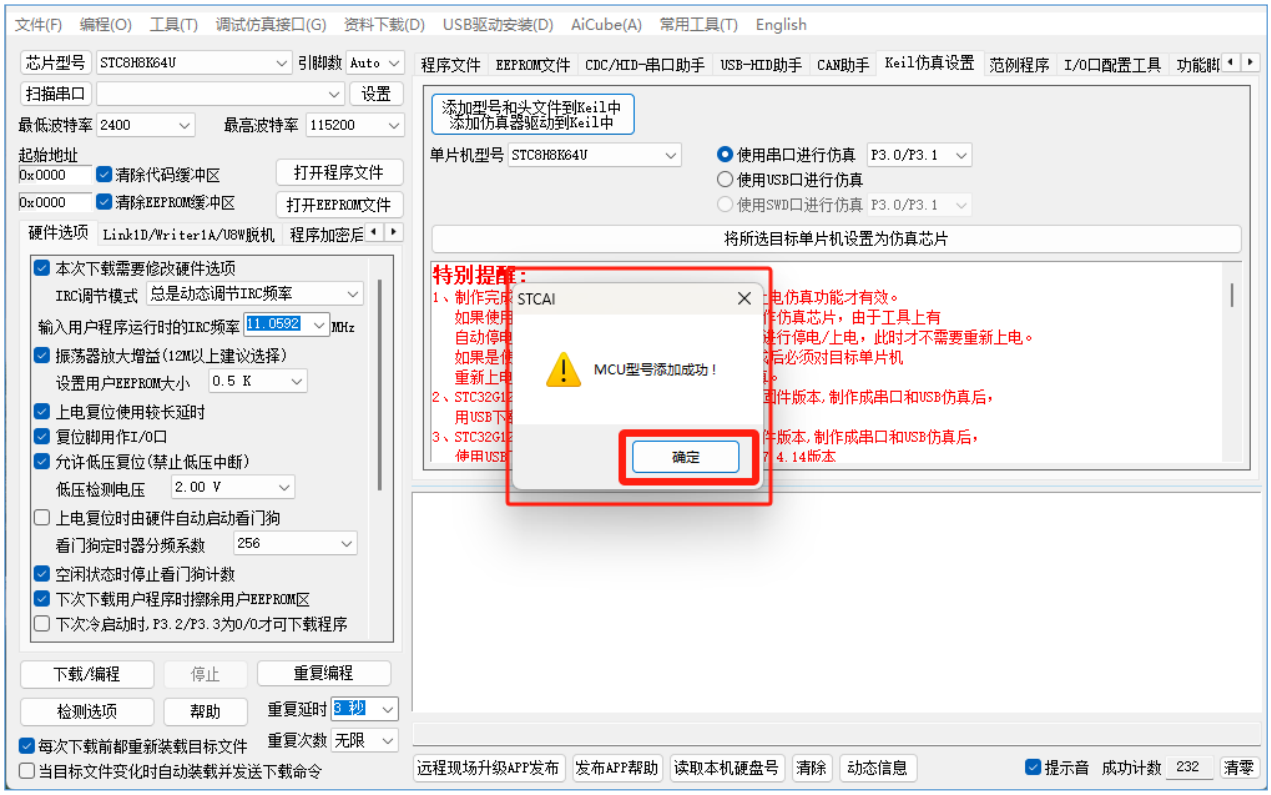
首先开 AiCube-ISP 下载软件，然后在软件右边功能区的“Keil 仿真设置”选项卡，在“单片机型号”中选择：STC8H8K64U（以下皆以此系列型号为例），点击“添加型号和头文件到 Keil 中 添加 STC 仿真器驱动到 Keil 中”按钮：



按下后会出现如下画面：



将目录定位到 Keil 软件的安装目录，然后确定。安装成功后会弹出如下的提示框：



即表示驱动正确安装了

头文件默认复制到 Keil 安装目录下的“C251\INC\STC”目录中
在 C 代码中使用“#include <STC8H.H>”或者“#include "STC8H.H"”进行包含均可正确使用

1.7　使用【AiCube】工具新建一个 8 位 8051 项目，在 AiCube-ISP 中

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】工具，可视化、图形化项目创建（配置），目前支持 **Ai8051U-8Bit**、**Ai8051U-32Bit**、**STC32G144K246**、**STC32G12K128**、**STC8H8K64U**、**STC15F2K60S2**、**STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的图形化、智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用【AiCube】工具来实现项目新建、配置及代码生成。

本次使用【AiCube】工具新建一个 8 位 8051 项目，使用 STC8H8K64U-8Bit 作为目标 MCU，应用 USB-CDC 功能，实现最简单的：printf_usb("Hello World !\r\n")。

1.7.1　准备工作，见前面几小节

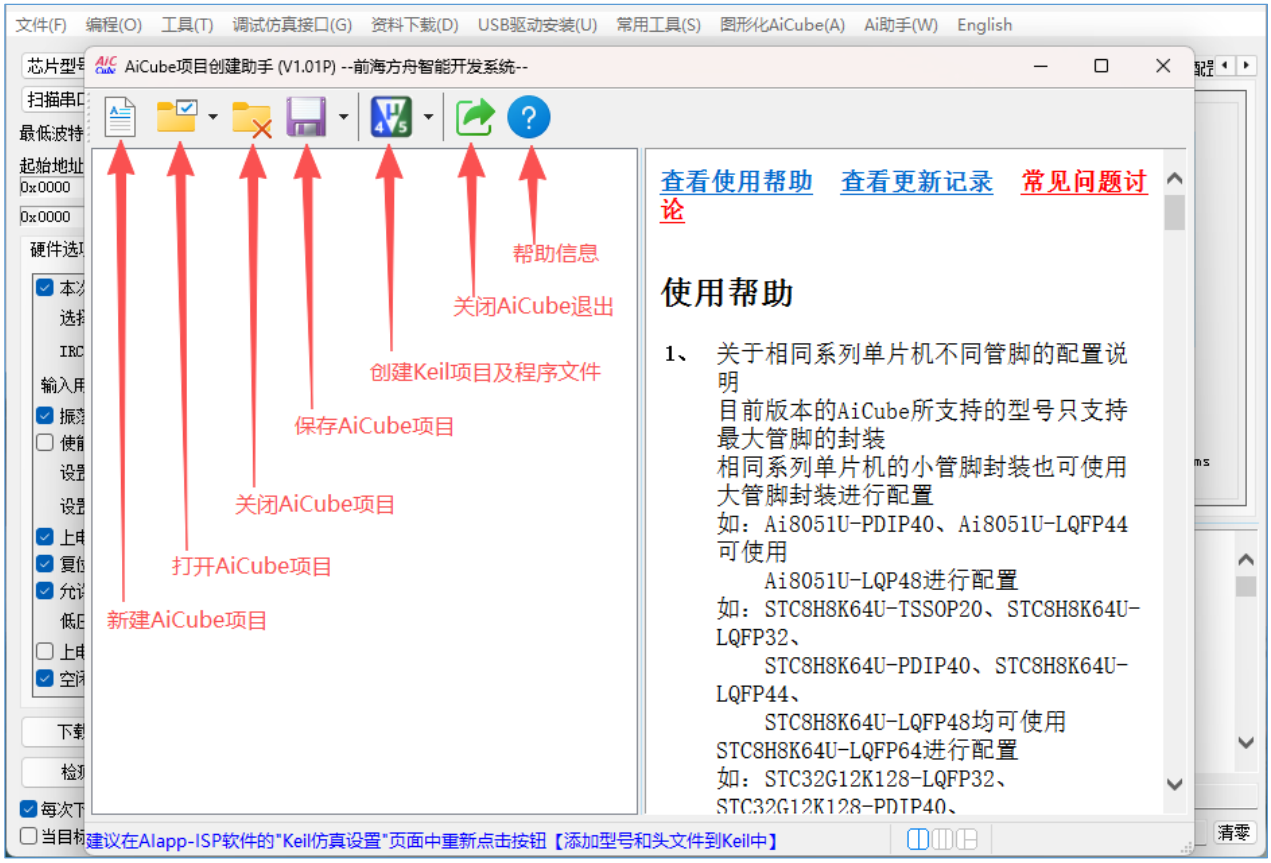
如何能建立自己的 Project，需要做的准备工作：
一、下载、安装 Keil C251/C51，见前面几小节；
二、下载并解压缩 AiCube-ISP，见前面几小节；
三、添加型号和头文件，见前面几小节；
四、建立一个文件夹存放你的 Projects，本例新建文件夹：E:\Projects-AiCube(V1.01P)。

1.7.2　使用【AiCube】创建项目

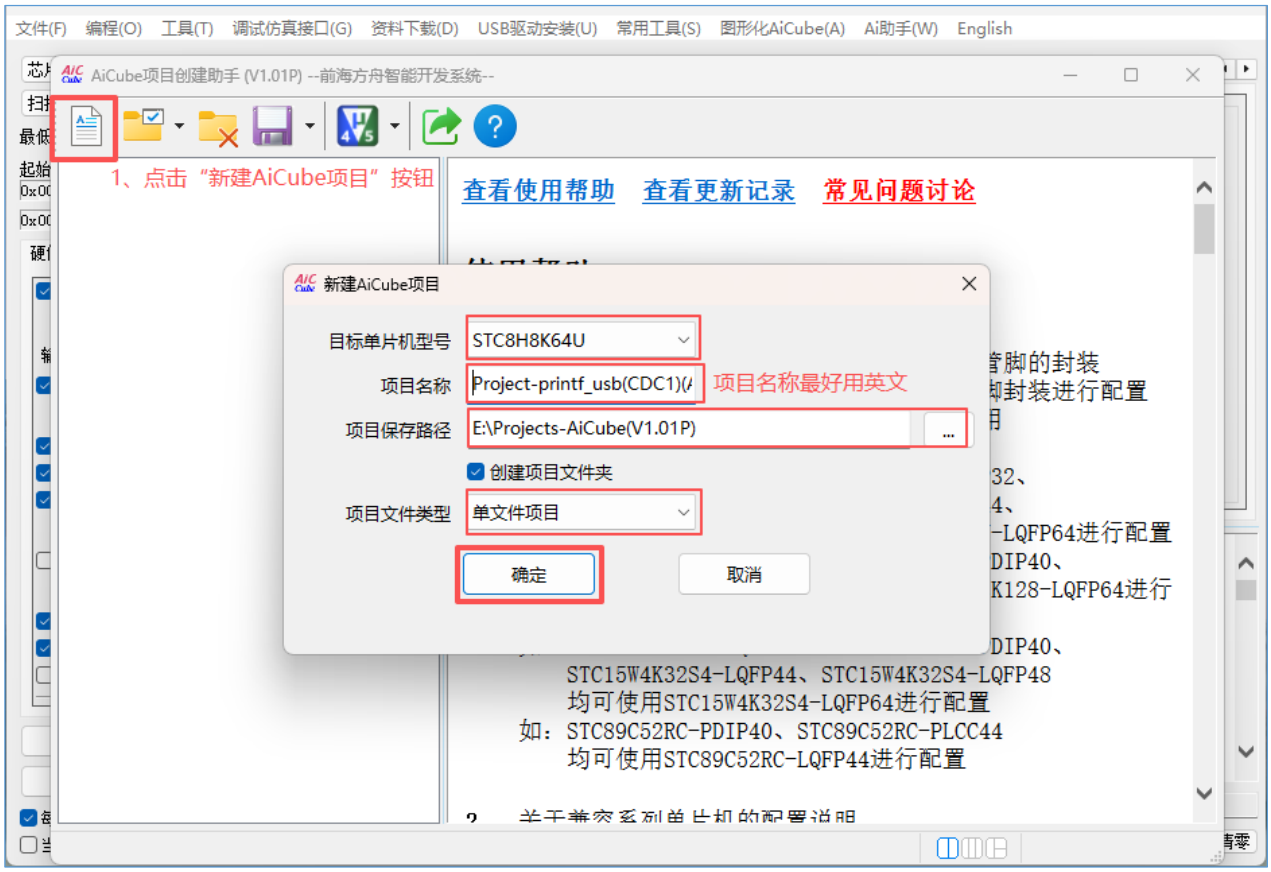
➲ 双击“AiCube-ISP”图标，打开 AiCube-ISP 软件界面，如下图，在菜单栏上找到【图形化 AiCube(A)】菜单项：



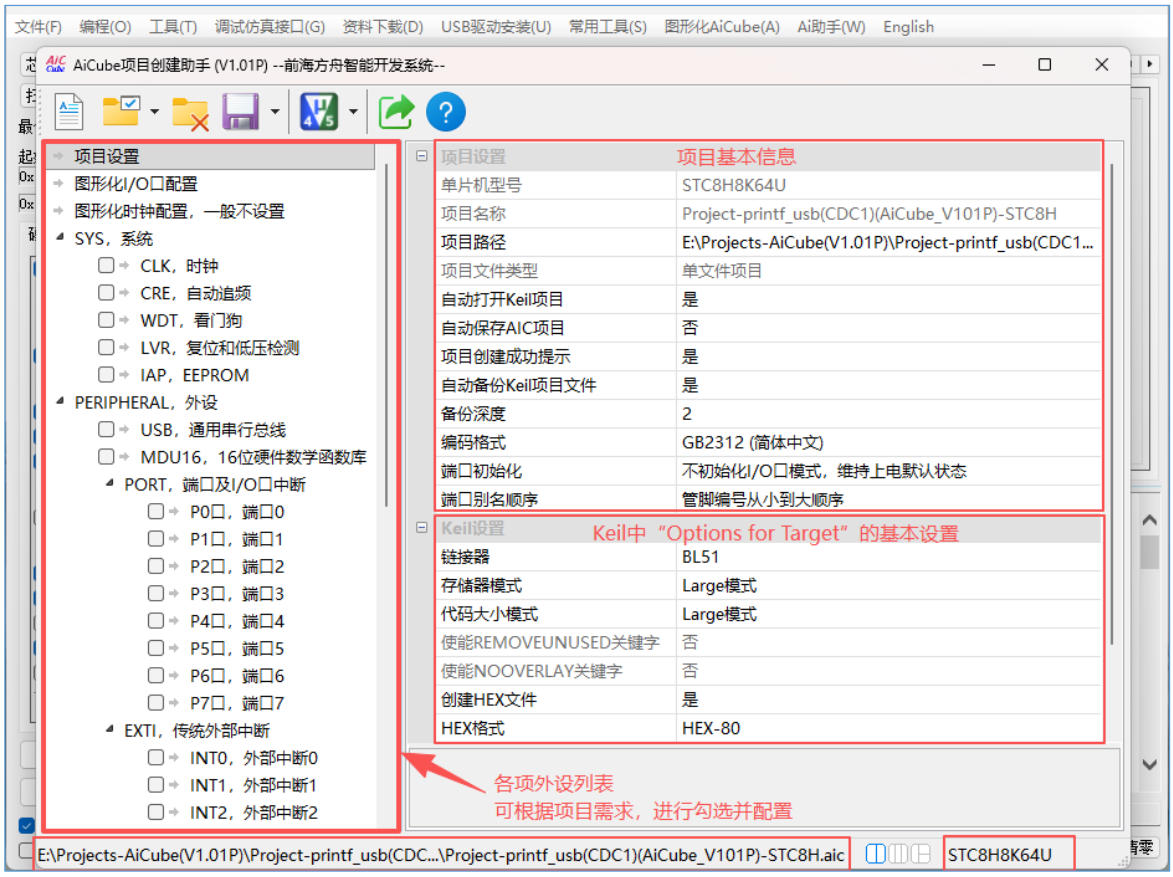
➲ 点击【图形化 AiCube(A)】菜单，弹出如下窗口：



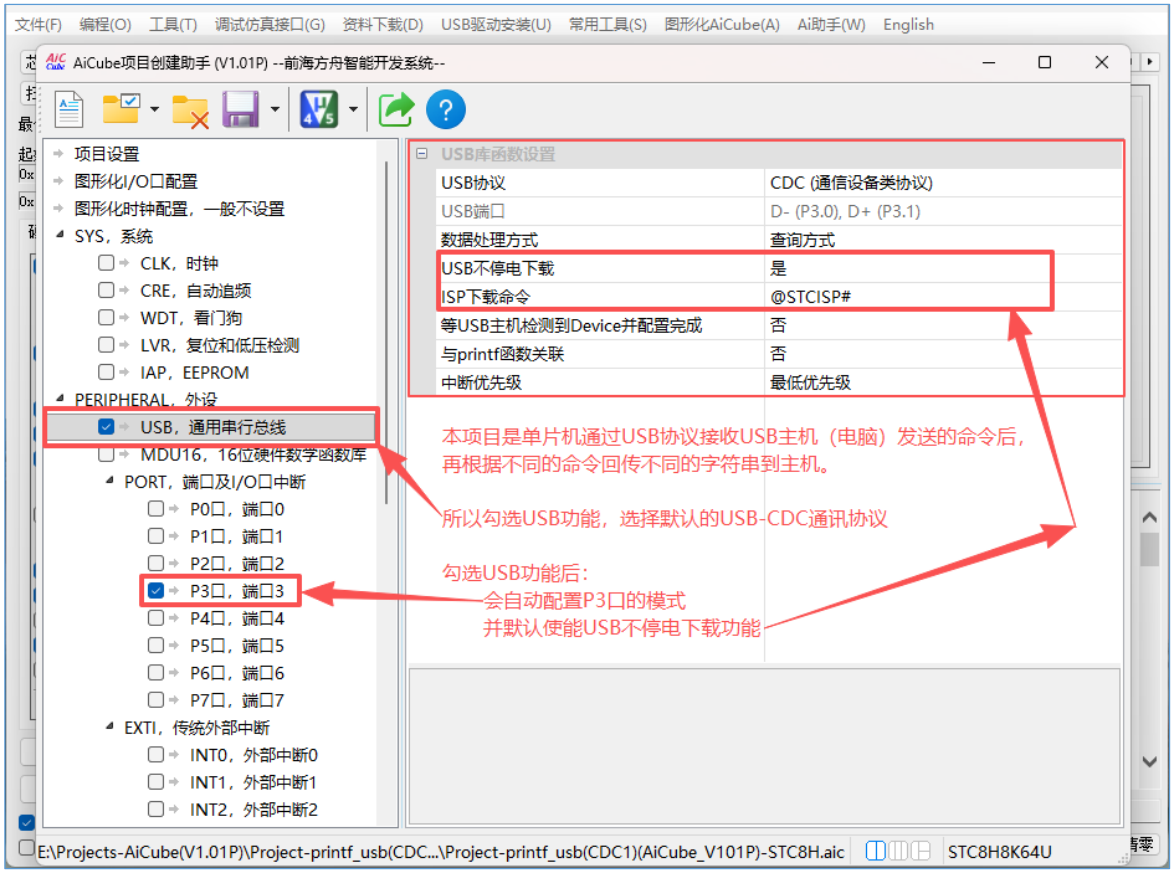
➲ 点击按钮“新建 AiCube 项目”，出现如下浮窗，选择并填写项目基本信息，建议项目名称用英文！



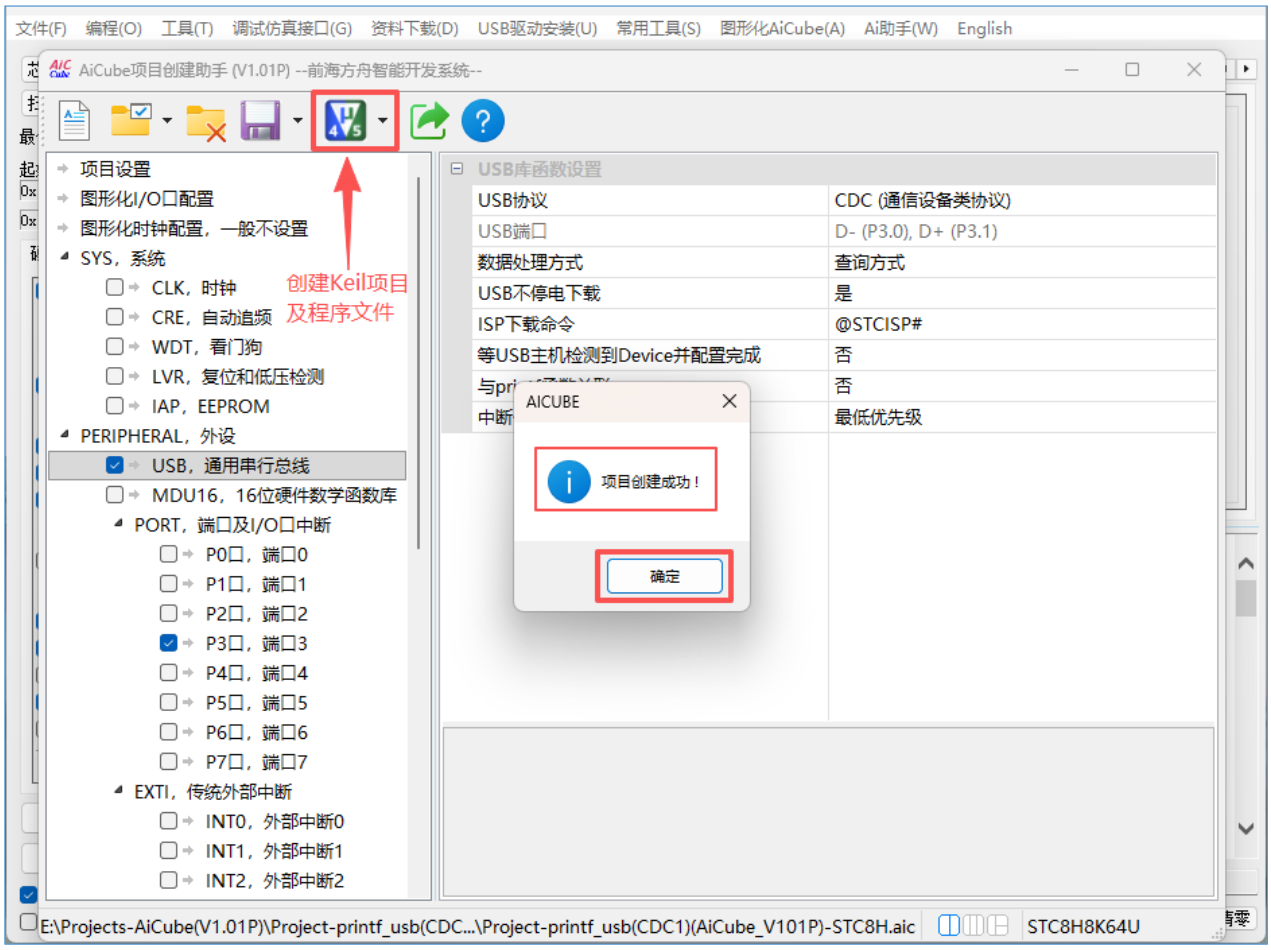
➲ 点击“确定”按钮，出现如下界面：



➲ 因本项目用到 USB-CDC 功能，要加载 USB-CDC 库函数，对 USB 初始化等，使用【AiCube】助手根据本项目的实际需求，对项目进行如下配置：



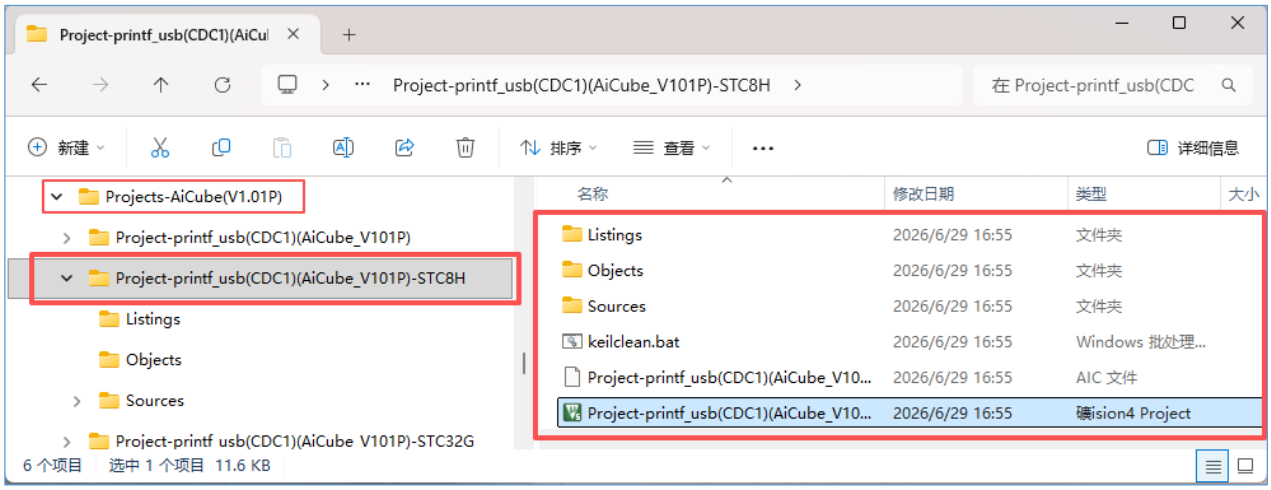
➲ 点击“创建 Keil 项目及程序文件”按钮，【AiCube】提示：“项目创建成功！”，如下图：



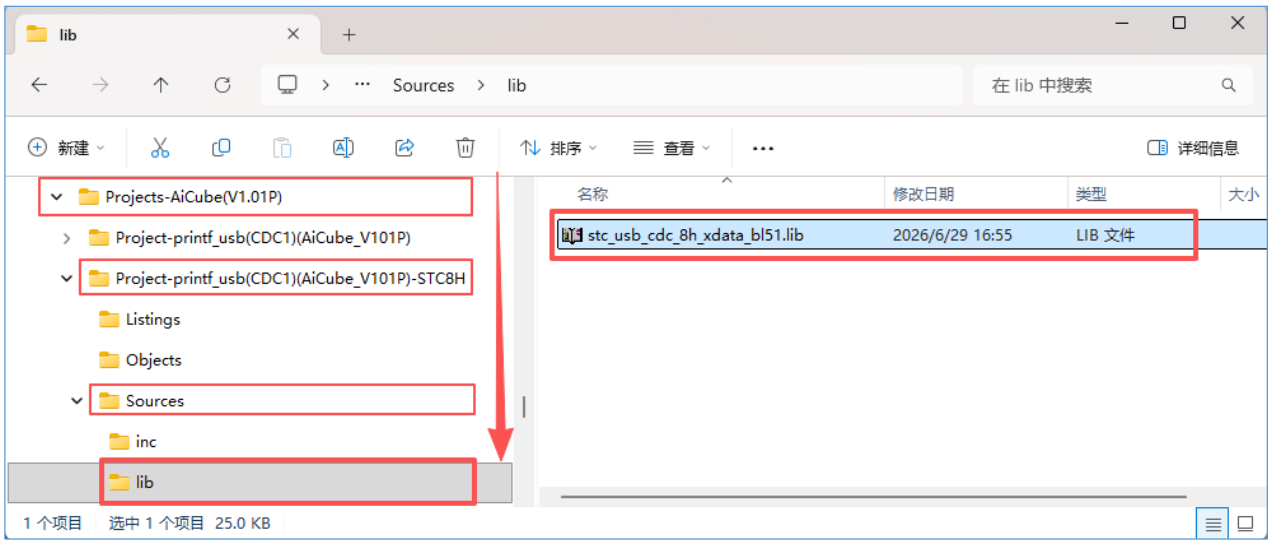
➲ 点击“确定”按钮，系统会自动打开 Keil 软件，调入新创建的 Keil 工程项目，本实验创建的项目是“Project-printf_usb(CDC1)(AiCube_V101P)-STC8H”。现在，可以关闭【AiCube】工具，当然，也可以不关闭。

➔ 观察 1：项目文件夹，本例项目文件夹：E:\Projects-AiCube(V1.01P)

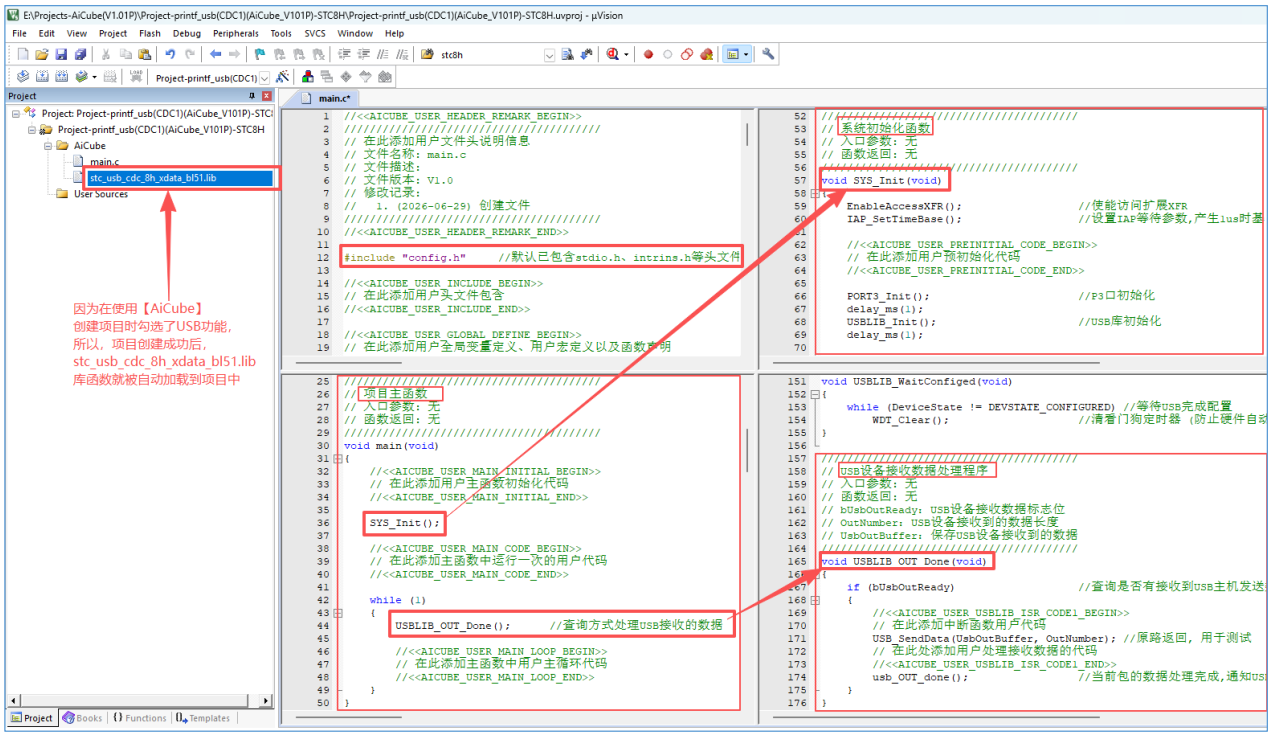
➲ 我们发现，经过上述操作，【AiCube】工具已经帮我们建立了项目目录及 Project-printf_usb(CDC1)(AiCube_V101P)-STC8H.uvproj 文件，如下图：



➲ 再打开 Sources>lib 子目录，【AiCube】也已经将相应的 USB 库函数加载进项目目录：



➔ 观察 2：已打开的 Keil 项目文件 Project-printf_usb(CDC1)(AiCubeV101P)-STC8H.uvproj



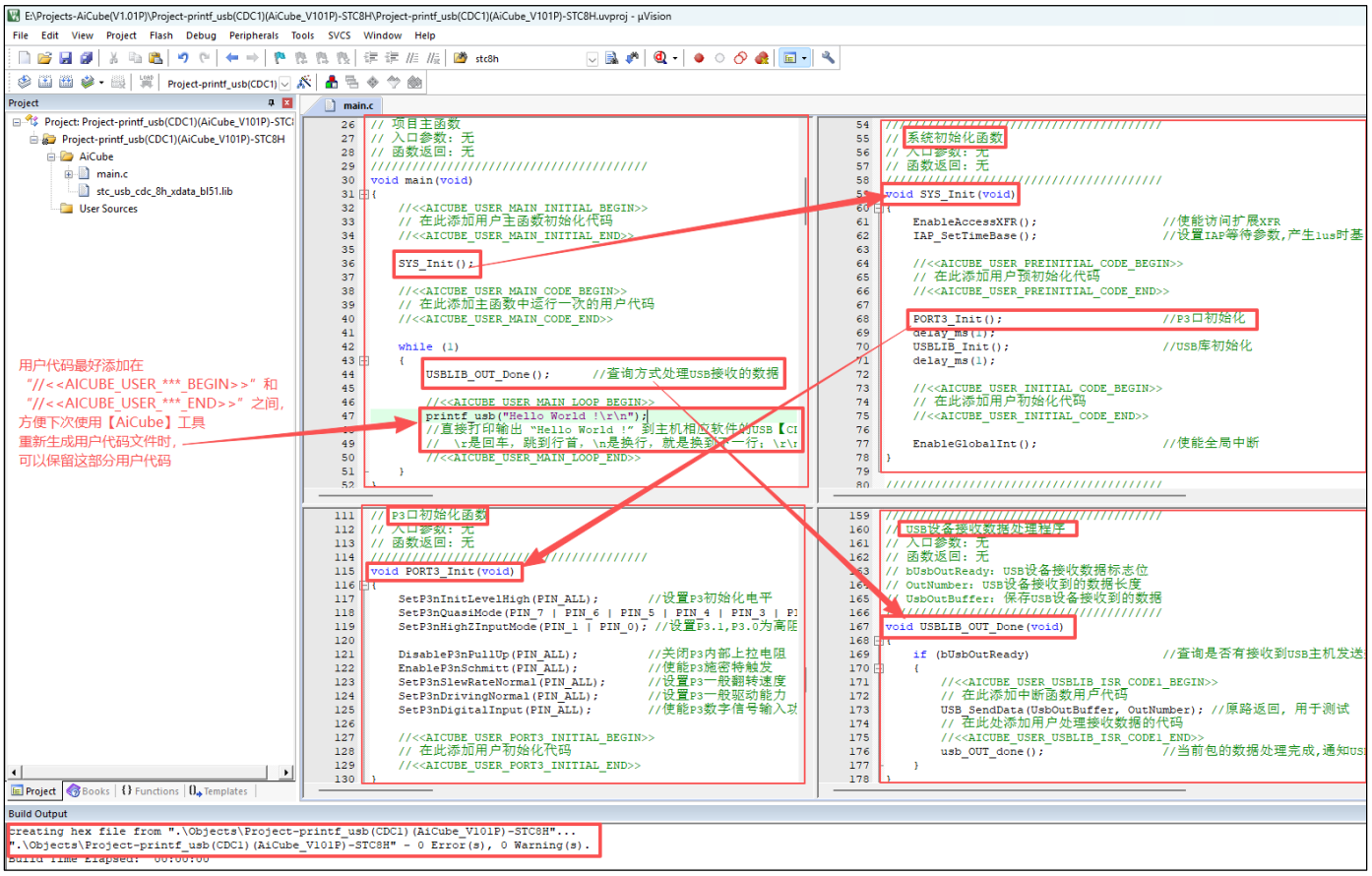
根据在【AiCube】中的配置，从上图中我们看到【AiCube】创建的本项目中不仅有了 main.c，还已经自动加载了 USB-CDC 库函数（stc_usb_cdc_8h_xdata_bl51.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、微秒和毫秒延时函数、P3 口初始化函数、USB 相关的配置及处理函数。

现在，本项目只需要在 main.c 的主函数的 while 循环中加入用户代码即可。

1.7.3　　增加实验项目代码

为了实现单片机通过 USB 协议接收 USB 主机（电脑）发送字符串"Hello World !"，我们现在只需要在主函数的 while 循环中加入“printf_usb("Hello World !\r\n");”的代码即可。保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口分裂出 4 个窗口，对比观察）：

```
printf_usb("Hello World !\r\n");
//直接打印输出 “Hello World !” 到主机相应软件的 USB【CDC/HID-串口助手】
//   \r 是回车，跳到行首，\n 是换行，就是换到下一行；\r\n 不要交换次序
```

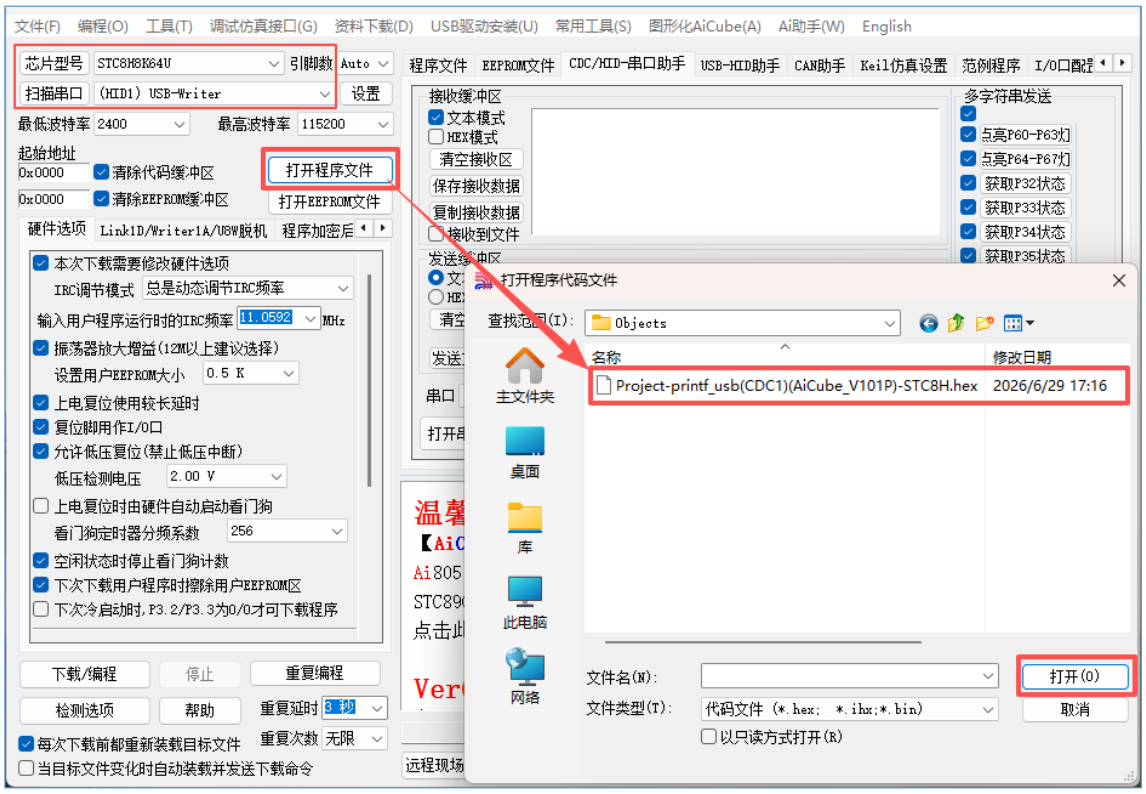


0 Error(s)，0 Warning(s)，在\Objects\目录下创建了 hex 文件。完美！

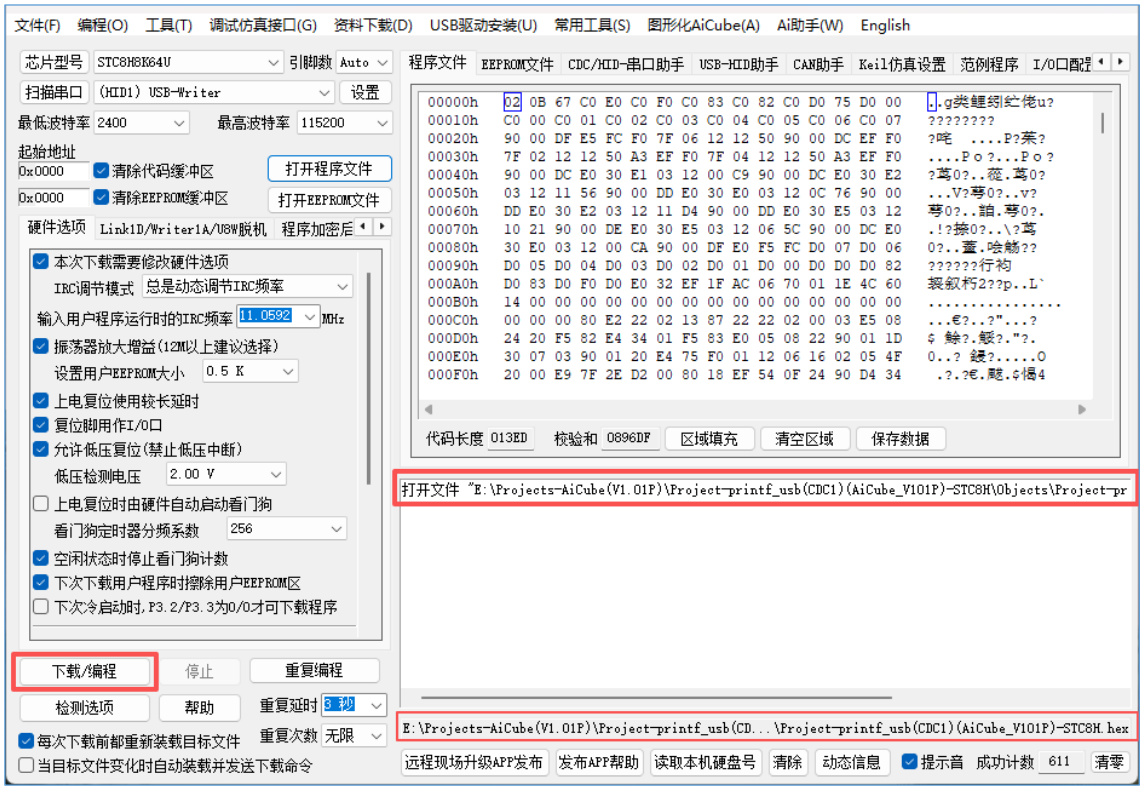
下面就到了用 STC8H8K64U 实验箱查看项目现象的时候。

1.7.4 使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果

连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“Project-printf_usb(CDC1)(AiCubeV101P)-STC8H.hex”（HEX 文件的位置除非特殊指定，一般位于该项目目录下的子目录 Objects 中），如下图：



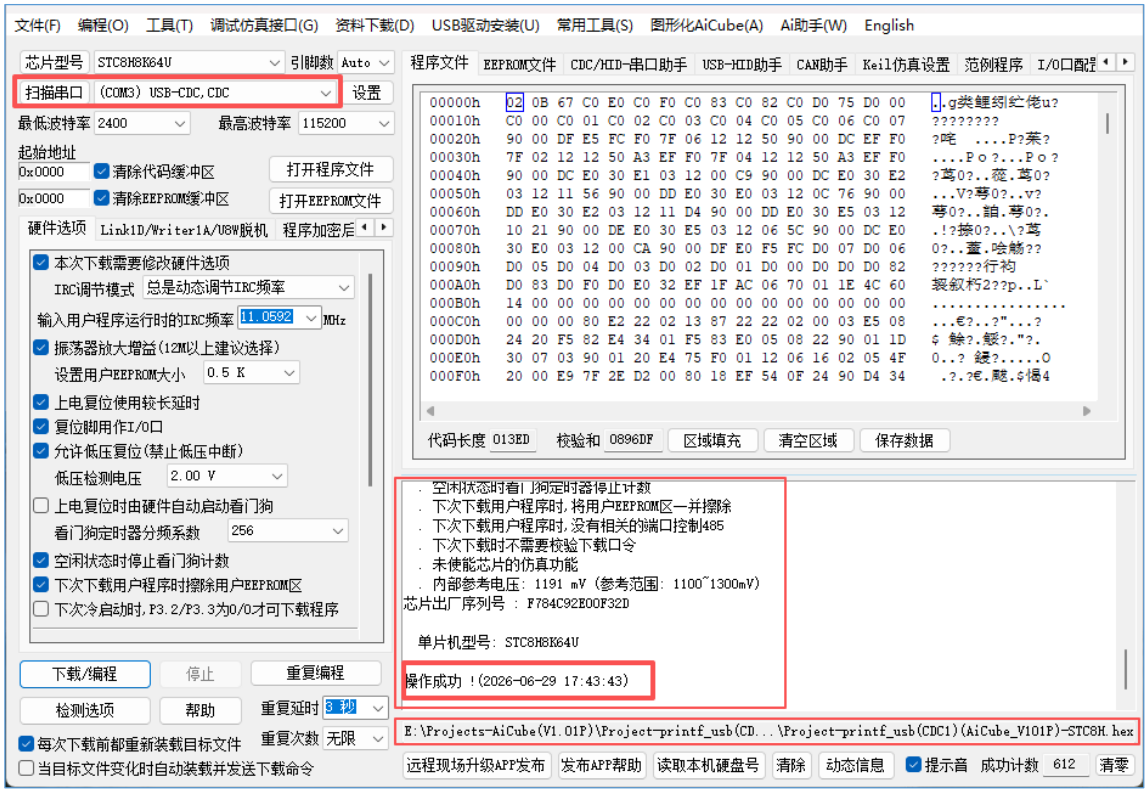
➲ 点击“打开”按钮，AiCube-ISP 软件界面如下图所示，出现“‘打开文件***.hex’成功！”



➲ 找到并点击“下载/编程”按钮

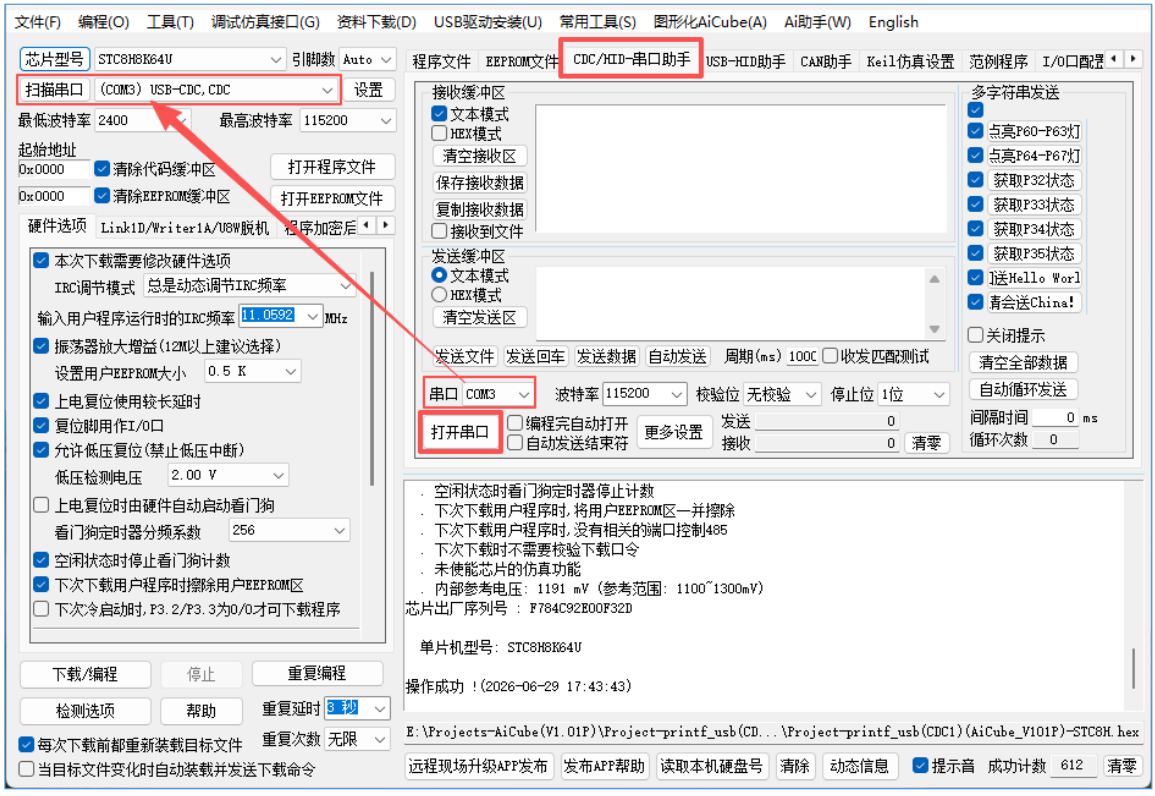
把 Project-printf_usb(CDC1)(AiCubeV101P)-STC8H.hex 文件烧录进 STC8H8K64U 实验箱，AiCube-ISP 软件界面出现：操作成功！

恭喜你！



在 AiCube-ISP 系统软件界面，我们看到：此时的串口显示“（COM3）USB-CDC，CDC”。

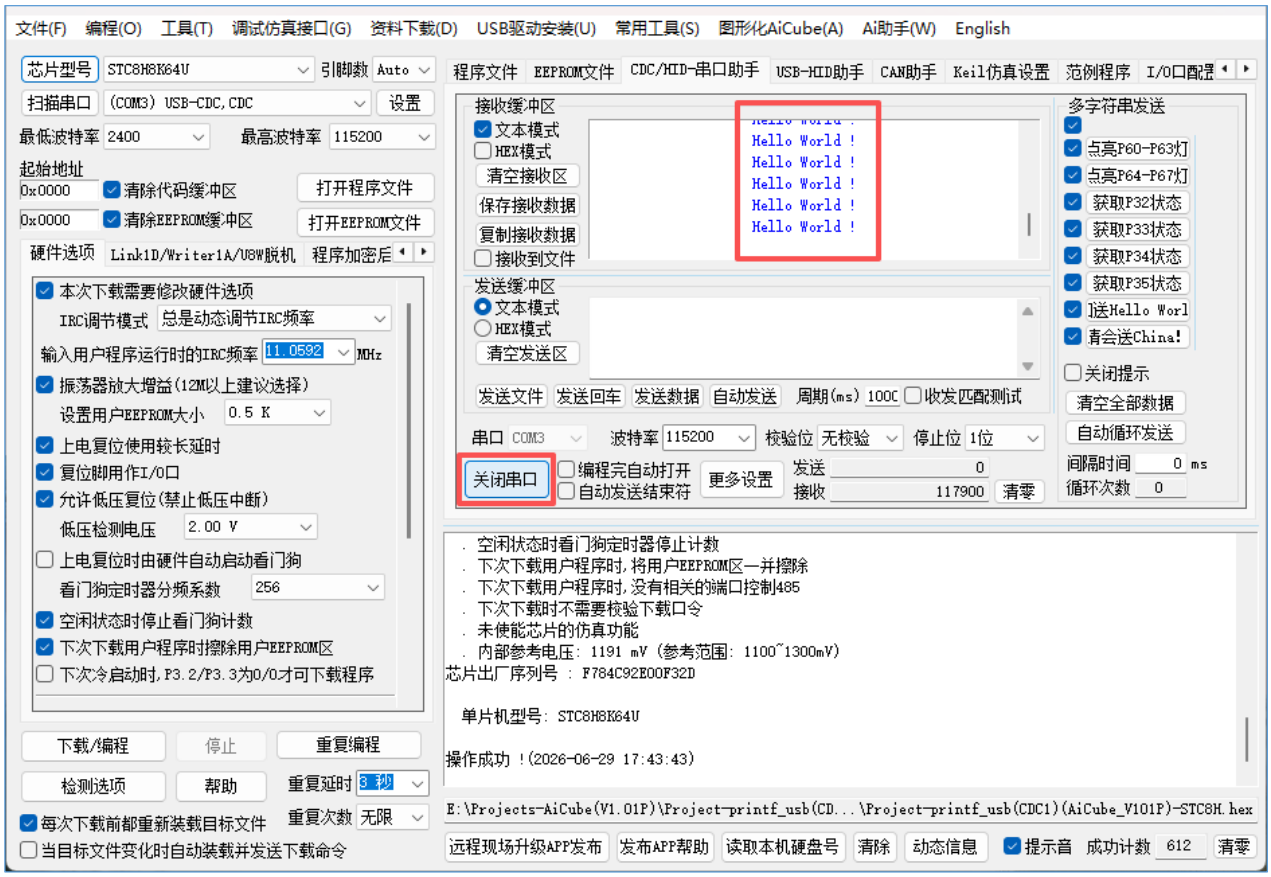
➲ 打开“CDC/HID-串口助手”选项卡，找到“打开串口”按钮，如下图：



➲ 点击“打开串口”按钮！观察实验现象：

✓ 在 CDC/HID-串口助手的“接收缓冲区”，就会不停的显示“Hello World！”；

✓ 同时，“打开串口”按钮，显示为“关闭串口”



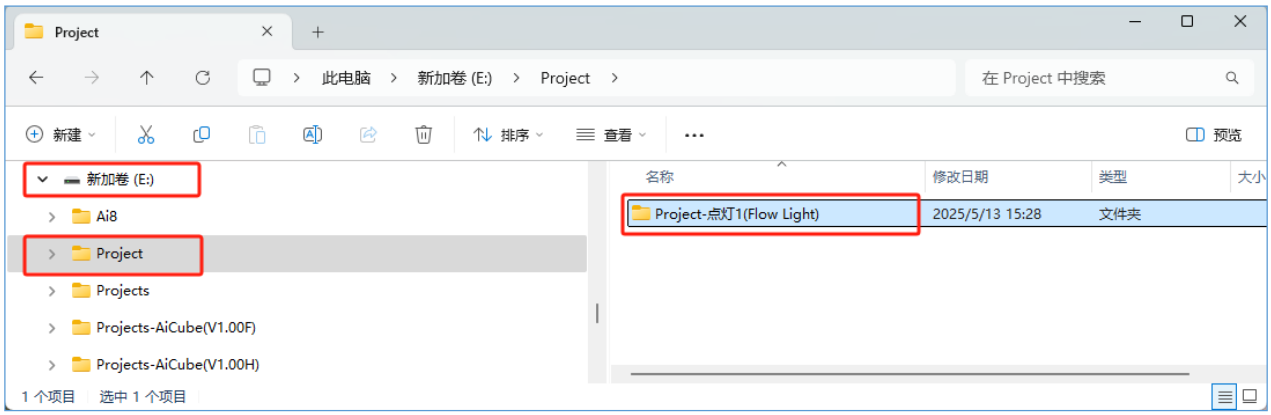
完成实验验证！

有了 STC 的【AiCube】助手，单片机编程就是这么简单！快用起来吧！

1.8 新建项目和项目设置，在 Keil 中

1.8.1 准备工作，见前面几小节

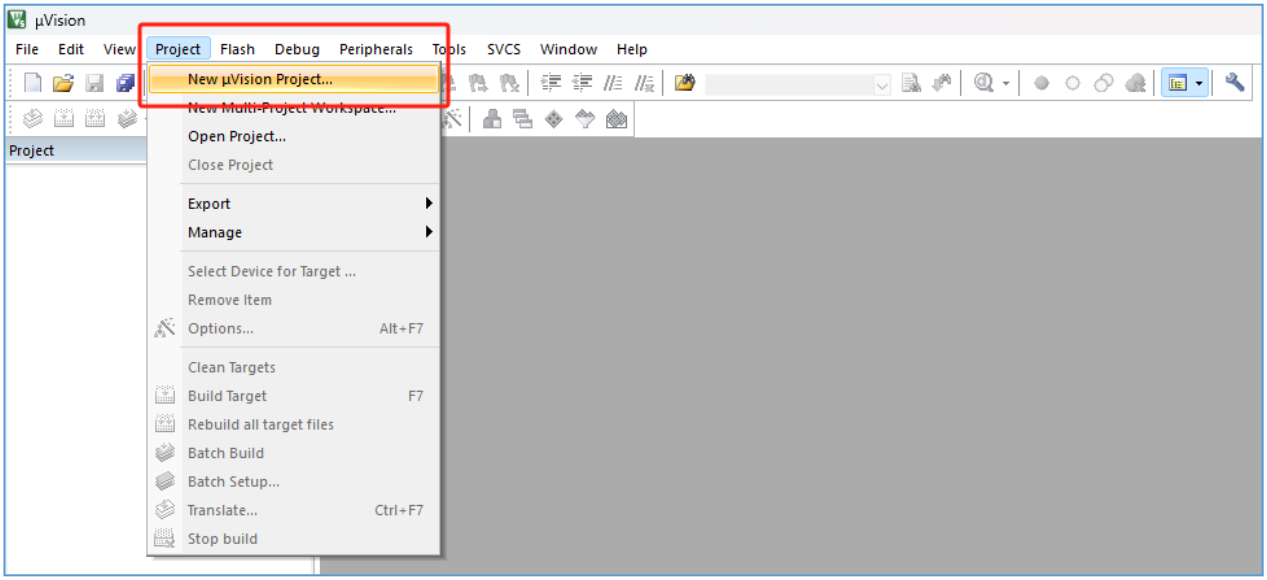
如何能建立自己的 Project，需要做的准备工作：
一、下载、安装 Keil C251/C51，见前面几小节；
二、下载、安装 AiCube-ISP，见前面几小节；
三、添加型号和头文件，见前面几小节；
四、建立一个文件夹存放你的 Project。
在 E 盘创建一个文件夹 Project，用于存放自己的项目文件



1.8.2 新建项目

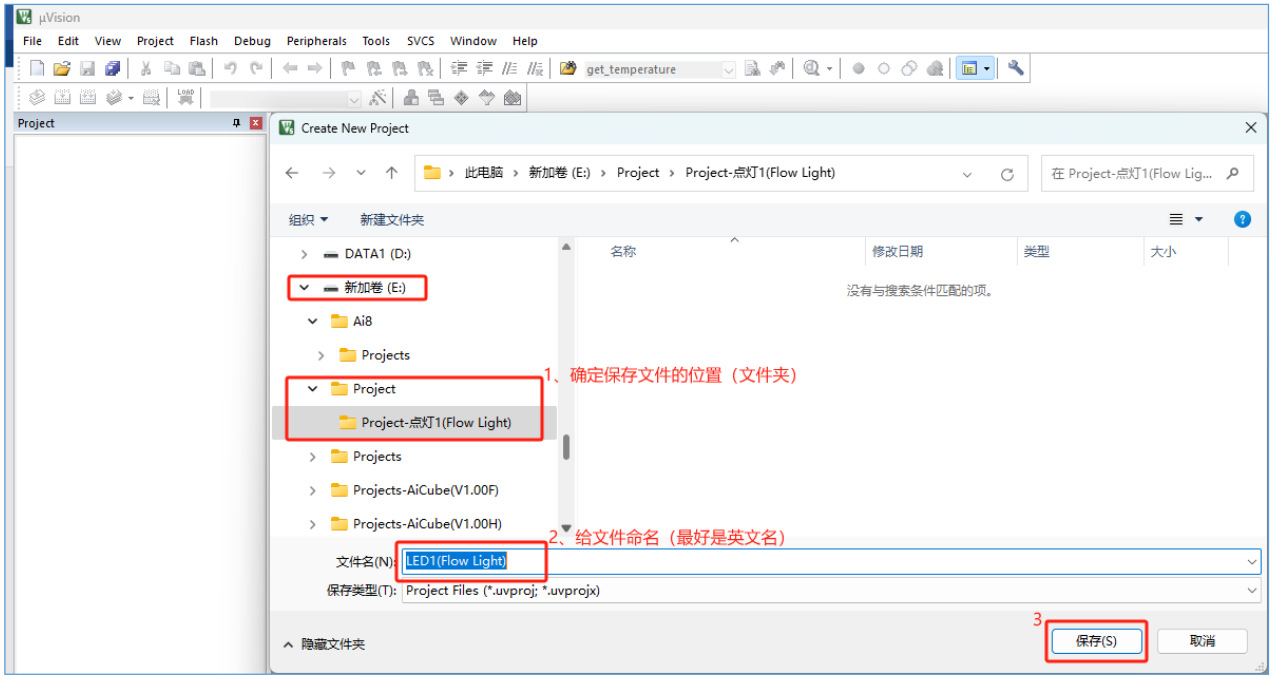
打开 Keil 软件

1.8.2.1 新建工程 -- 在上方的菜单栏 Project 标签中，选择“New uVision Project”菜单项，开始新建工程。



1.8.2.2 设置项目路径和名称 -- 将文件存入建好的文件夹中

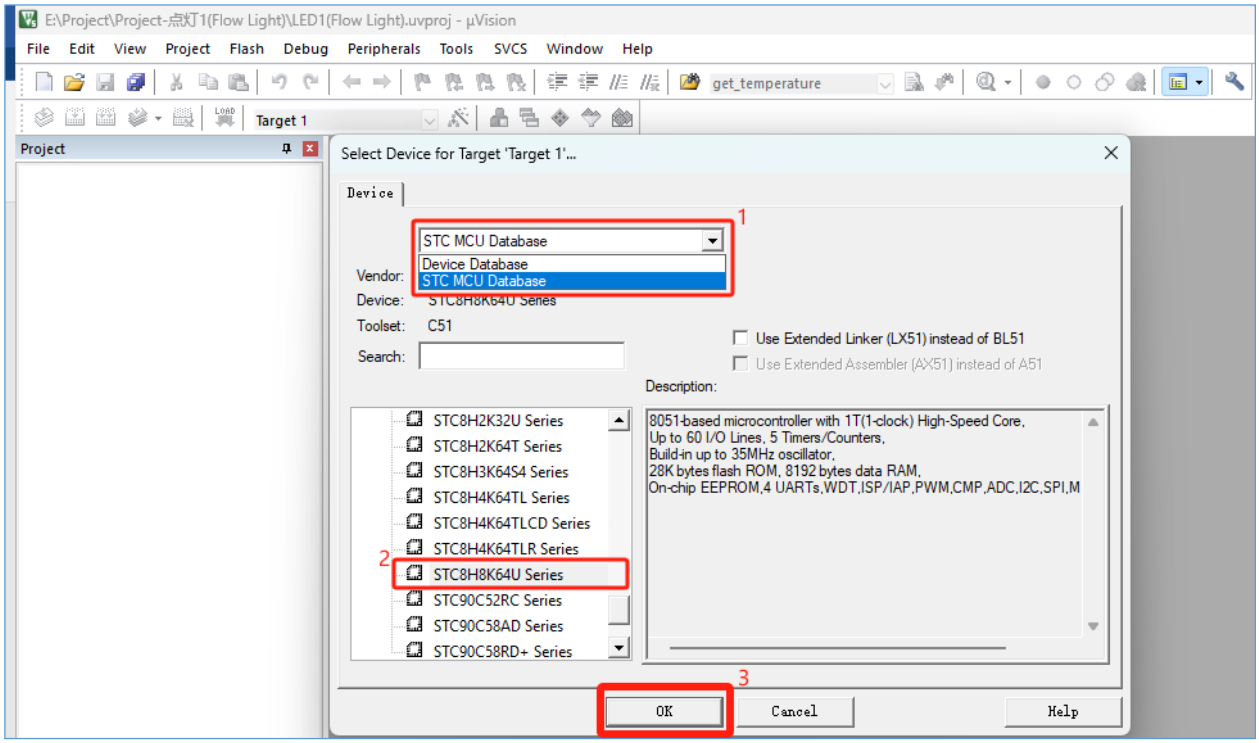
再给项目起个名称（方便查找）（尽量是英文的，如 Test…），点击“保存”。如下图：



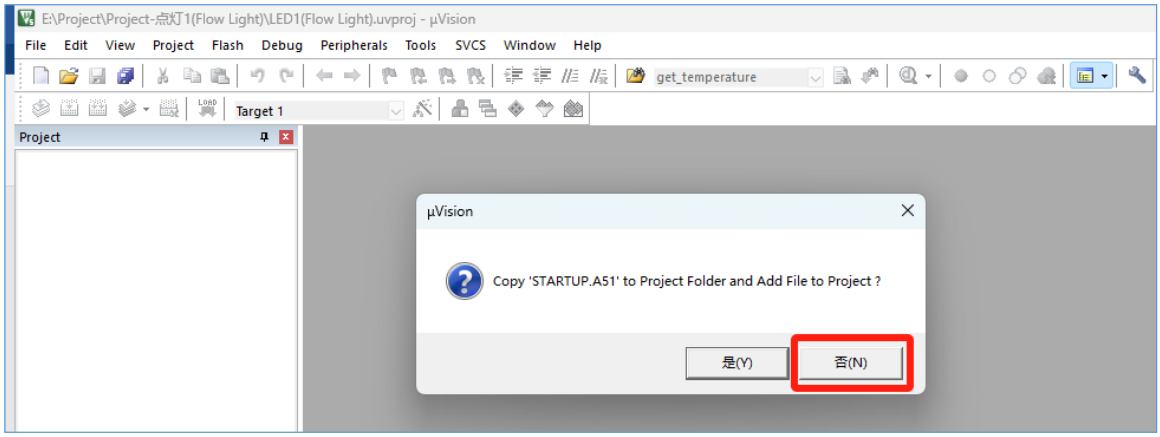
1.8.2.3 选择单片机型号（以 STC8H8K64U Series 为例）

在“Select Device for Target’Target1’”选项卡中：

- 选择 STC MCU Database
- 然后展开 STC 选项，选择“STC8H8K64U Series”选项（以下皆以此系列为例）
- 最后点击‘OK’以完成单片机选择。

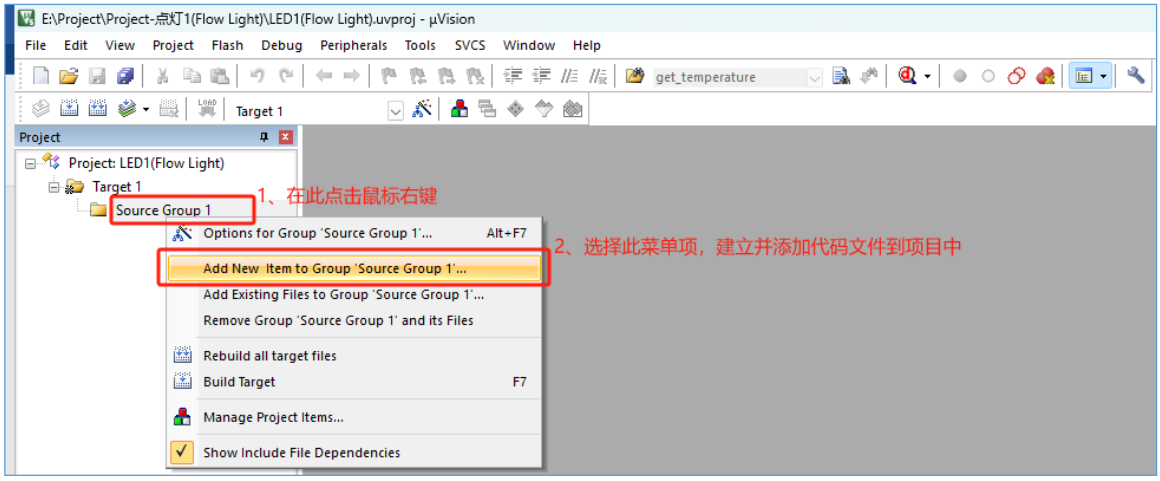


出现如下界面，点击“否”：

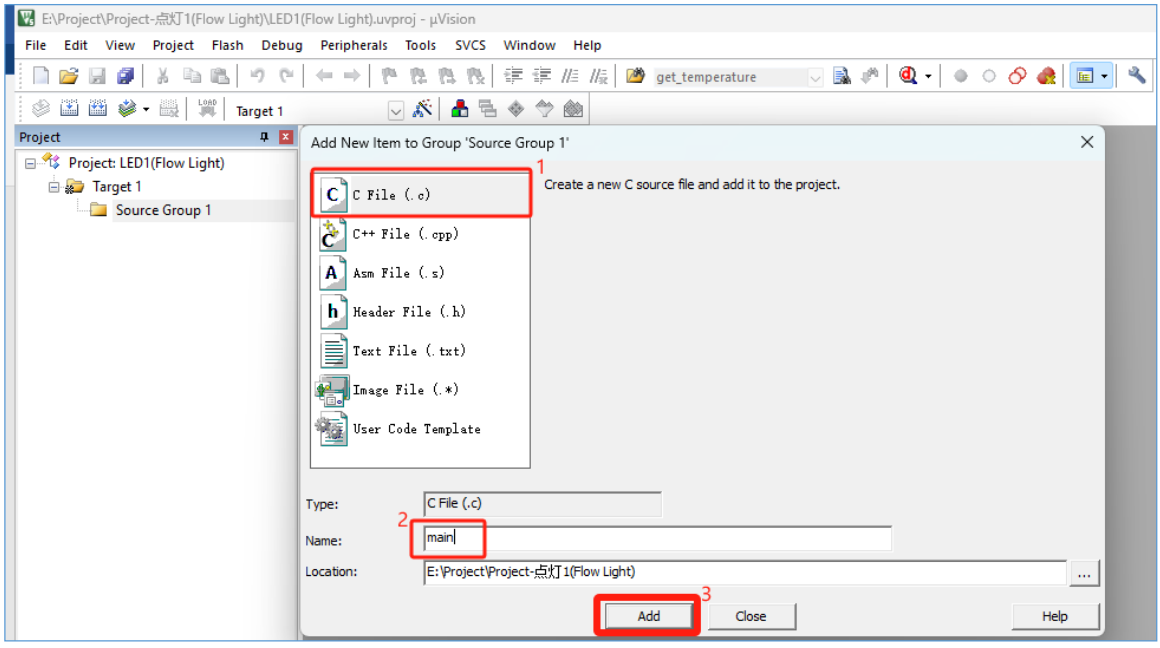


1.8.2.4　　添加源代码文件到项目 -- 给工程项目增加一个主程序文件

选择单片机后，工程界面还是一片灰白的。这时要给工程项目增加一个主程序文件，可以命名为 main.c。点击 Project 窗口上 Target1 的+，出现“Source Group1”，右键点击“Source Group1”，出现下图。

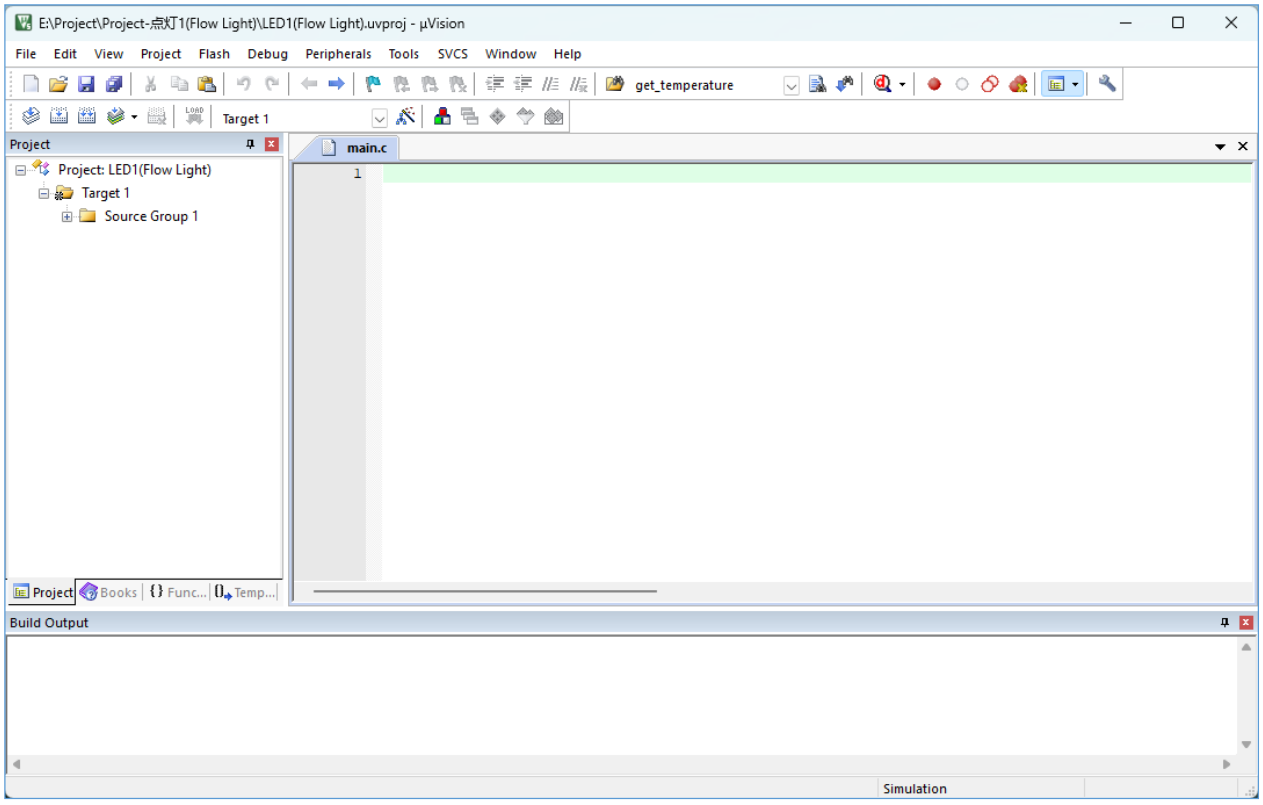


再选择快捷菜单中的“Add New Item to Group ‘Source Group 1’”菜单项，出现如下界面提示：



在上图中选择文件类型是：*.c

在 Name 输入框中输入文件名是：main 或其他任意文件名，最好是英文名
在 Location 输入框中选择当前项目的目录
然后点击 Add 按钮，将该新文件加入到当前项目。



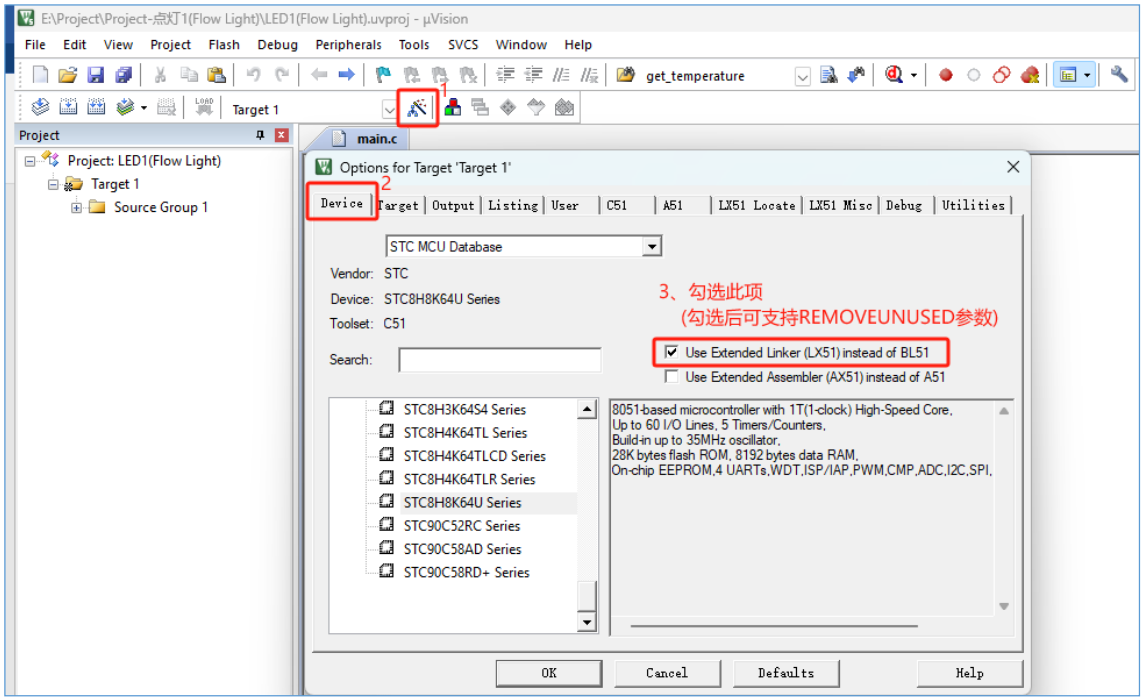
至此，生成了一个空的 main.c 主程序文件并且已经加入到当前项目。可以在该文件中编写代码了。后续的实验例程会演示如何将程序代码输入到其中。

当然，在编写代码前，最好先进行 Project 的各项基础设置。

1.8.3 项目的各种基础选项设置

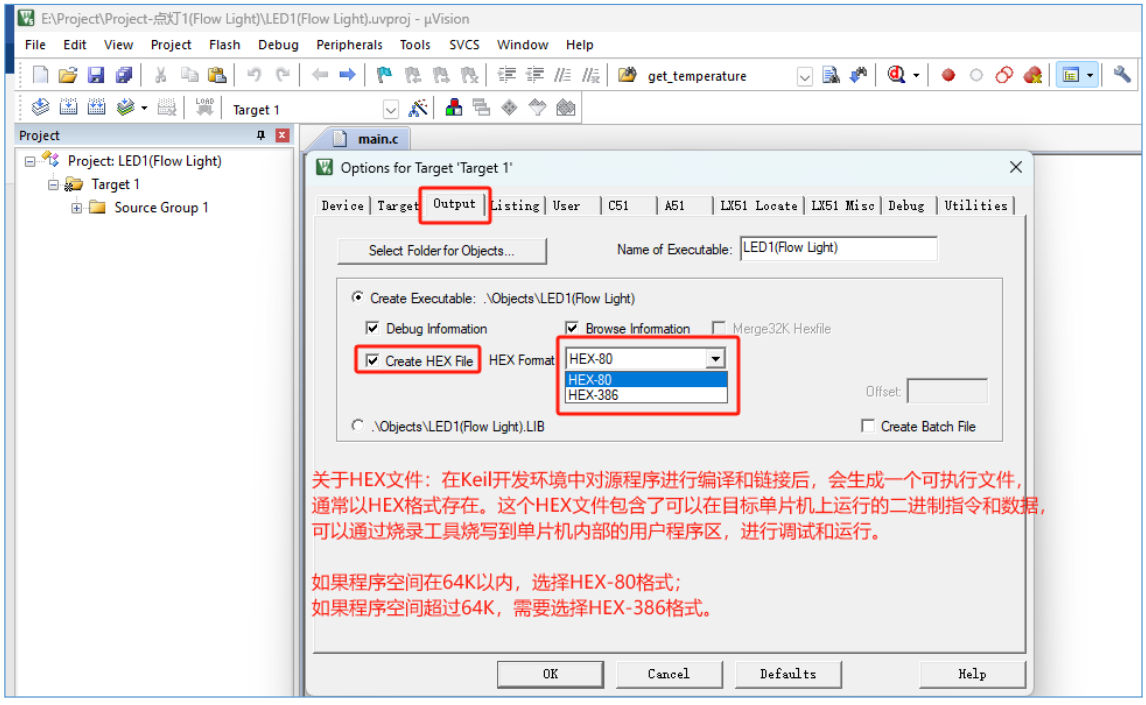
在 Build 工具栏找到一个类似魔术棒的工具，就是任务选项的设置。

1.8.3.1 Device 选项卡--勾选“Use Extended Linker（LX51）instead of BL51”可支持 REMOVEUNUSED 参数



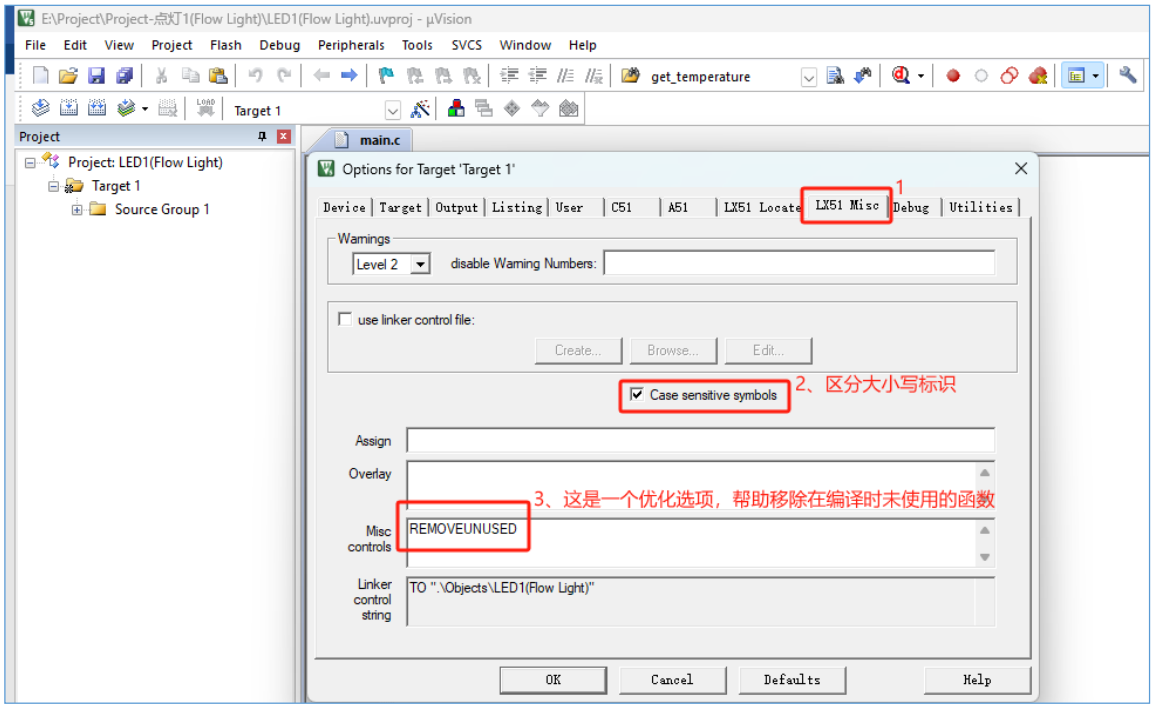
1.8.3.2 Output 选项卡--勾选 Create HEX File，选择 HEX-80

切换到 Output 选项卡，勾选‘Creat HEX File’选项，否则编译后是不会创建 HEX 文件的。HEX 文件是可以通过烧录工具烧录到单片机用户程序区。



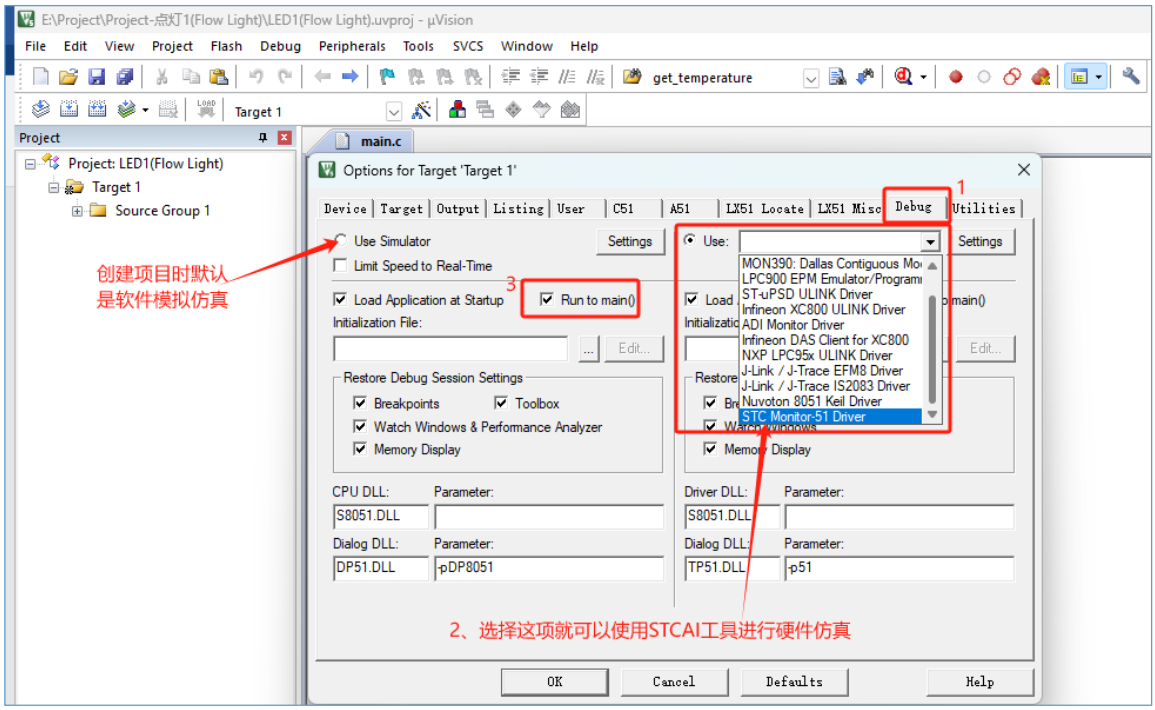
1.8.3.3 LX51 Misc 选项卡 -- 在 Misc controls 输入框输入‘REMOVEUNUSED’

切换到 LX51 Misc 选项卡，在 Misc controls 输入框中，输入大写的‘REMOVEUNUSED’，移除未使用的函数（如：库函数），以实现不使用的函数不进行链接。

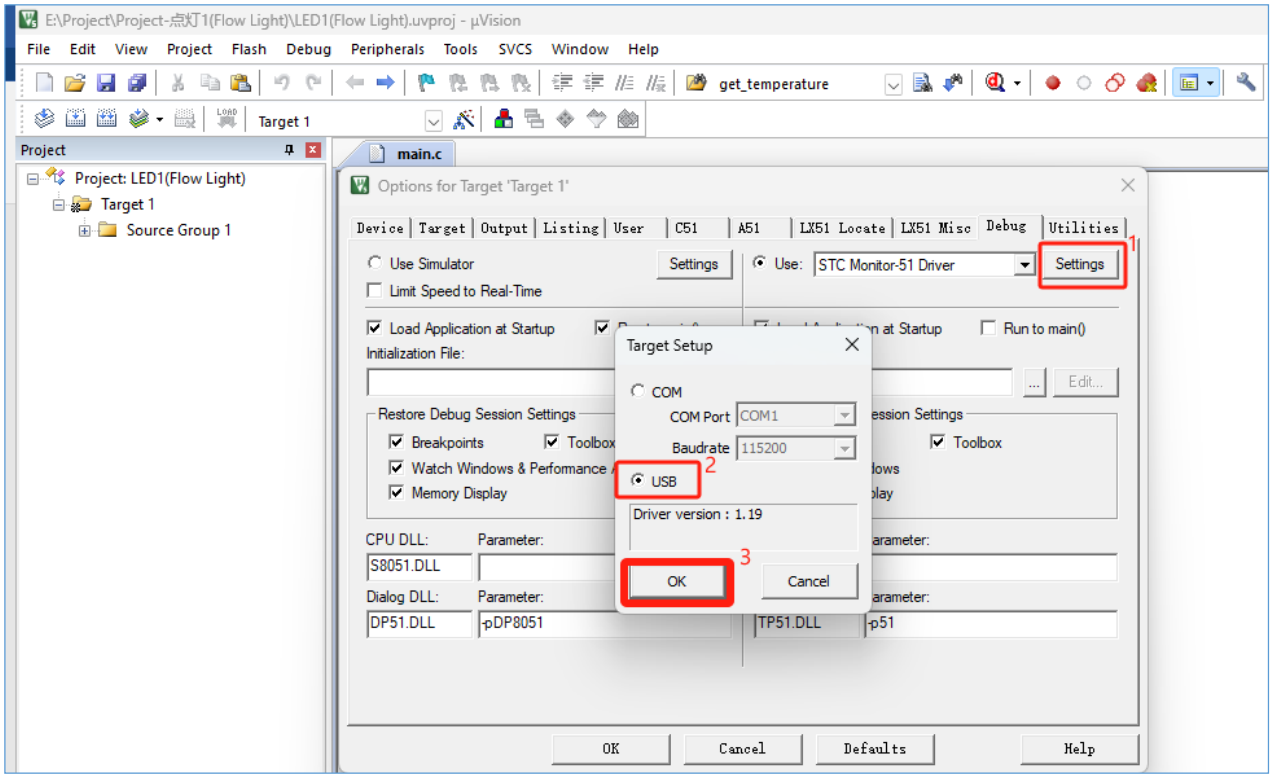


1.8.3.4 Debug 选项卡-- 使用 STCAI 工具进行硬件仿真设置

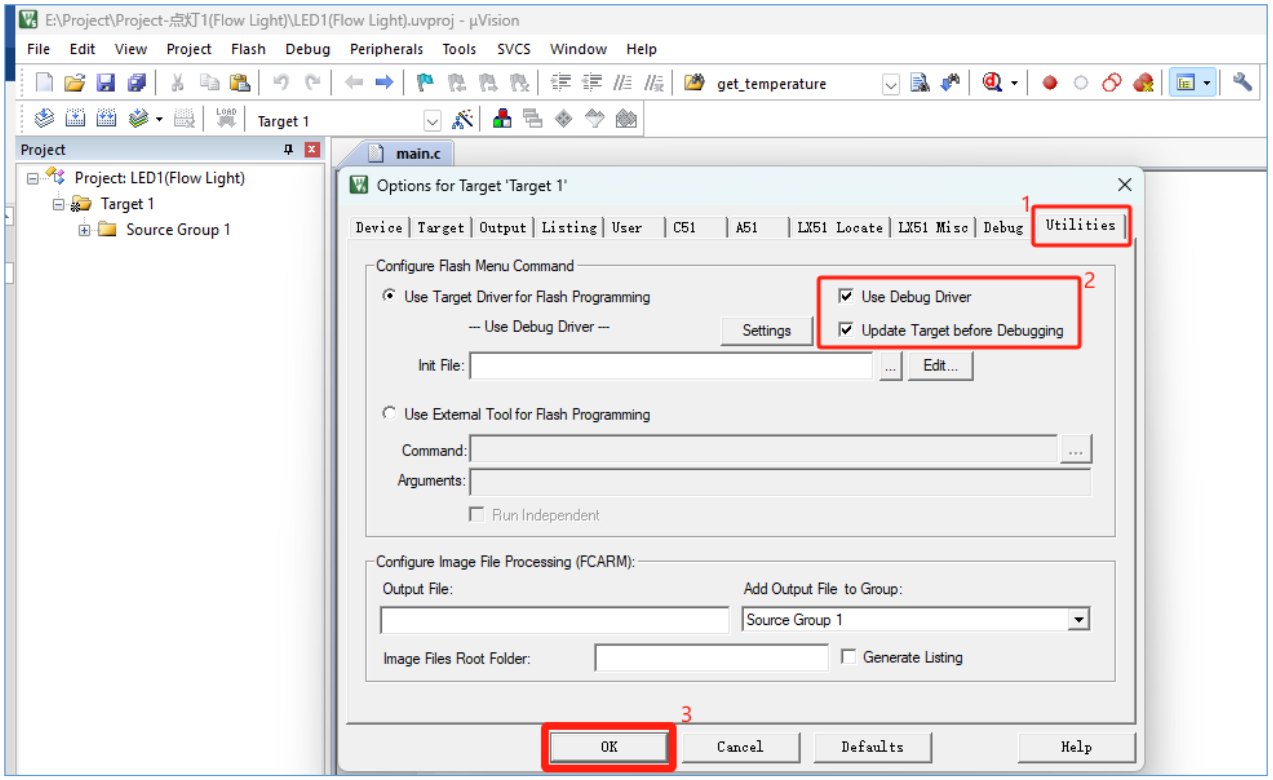
切换到 Debug 选项卡，创建项目时，Keil 默认是软件模拟仿真。我们可以借助 STCAI 的工具进行硬件仿真，如下图：



如果是选择通过芯片自带的硬件 USB 通讯仿真，则如下图设置：



切换到“Utilities”选项卡，勾选“Use Debug Driver”和“Update Target before Debugging”，点击“OK”，完成设置。

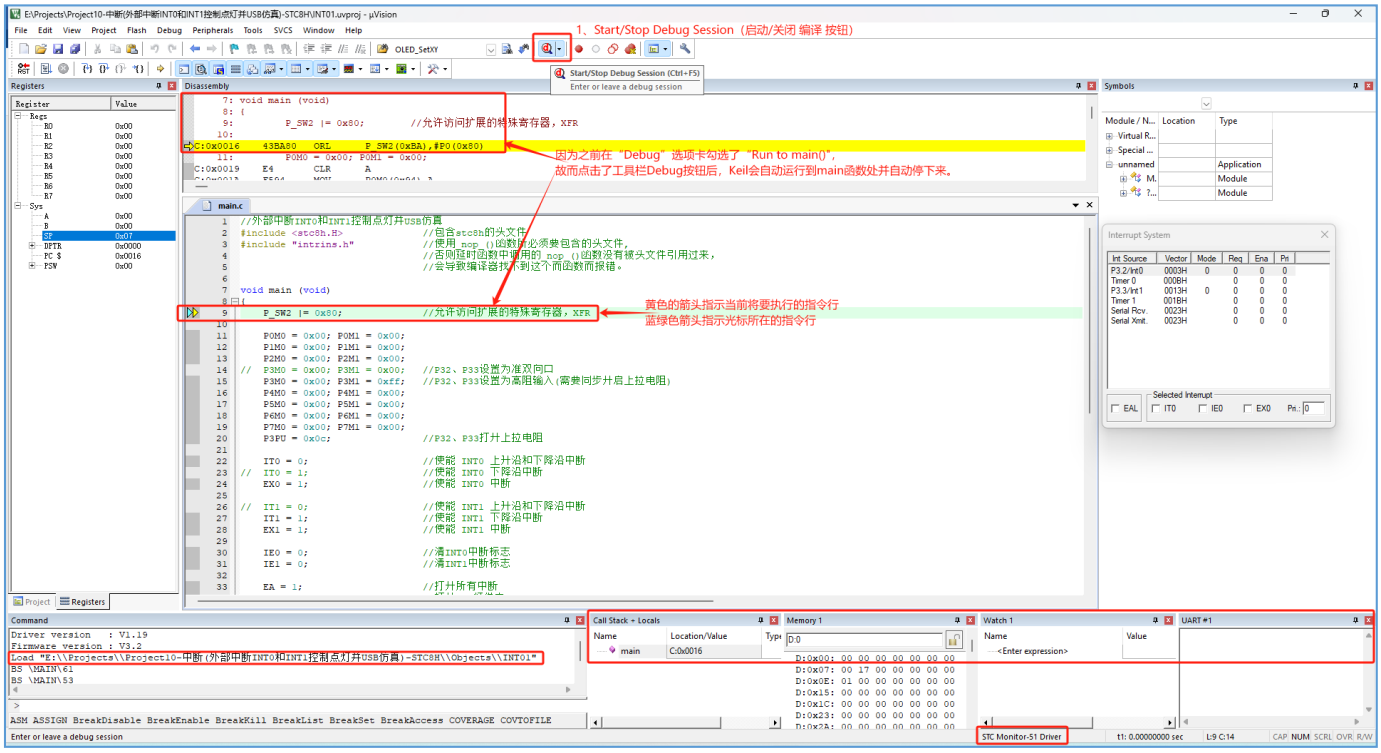


至此，工程项目的各种基础选项设置结束。可以进入下一节，在程序文件中添加实际的代码。

如果不使用仿真，USB 仿真设置可以跳过。

1.9　了解 File、Build、Debug 工具栏与仿真相关的常用操作，在 Keil 中

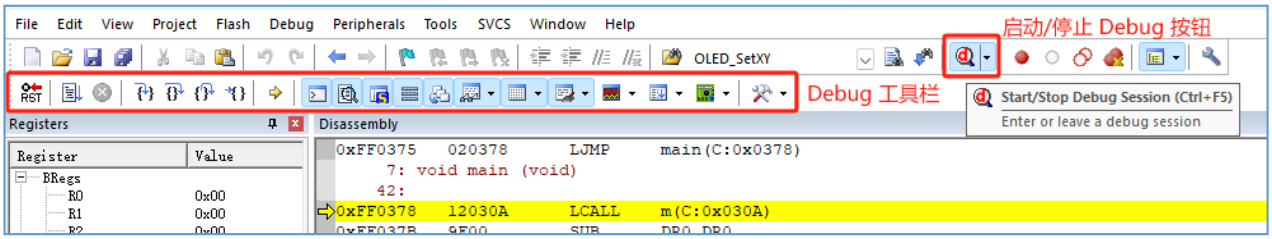
创建项目并对项目进行前面步骤的设置后，返回 Keil 界面，代码程序编译无误，点击工具栏的“Debug”（启动/关闭编译调试操作界面）按钮后，Keil 软件进入仿真界面，如下图：



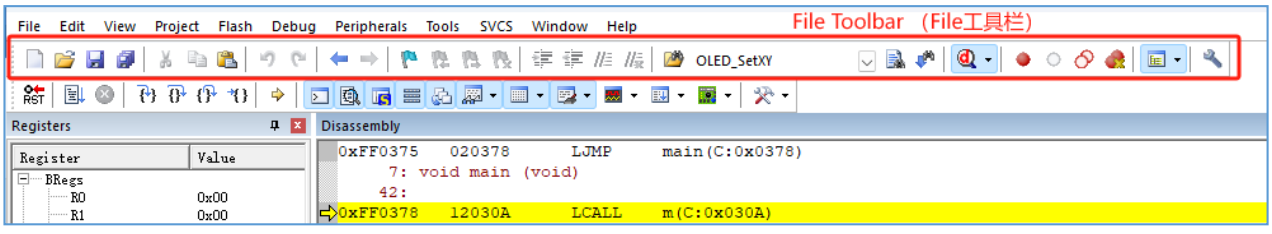
提示：退出 Debug 调试界面，同样点击工具栏的“Debug”按钮。

在 Debug 界面：

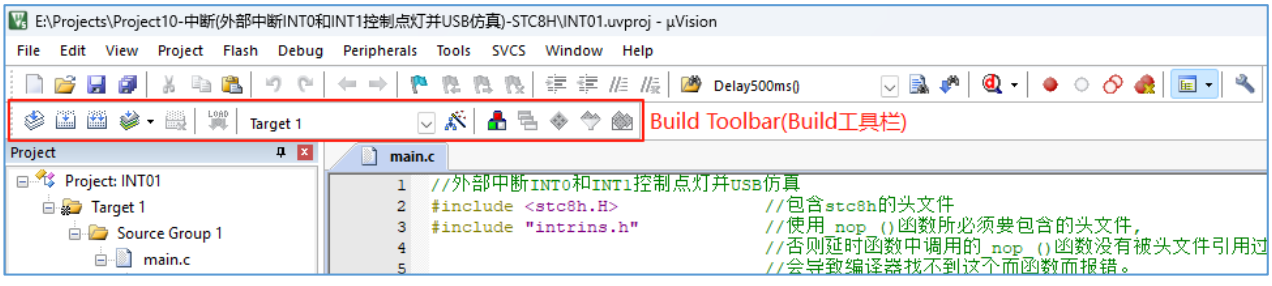
我们要认识一下 **Debug Toolbar（Debug 工具栏）**的常用调试/仿真按钮：



和 **File Toolbar（File 工具栏）**中在调试/仿真时的几个常用按钮：



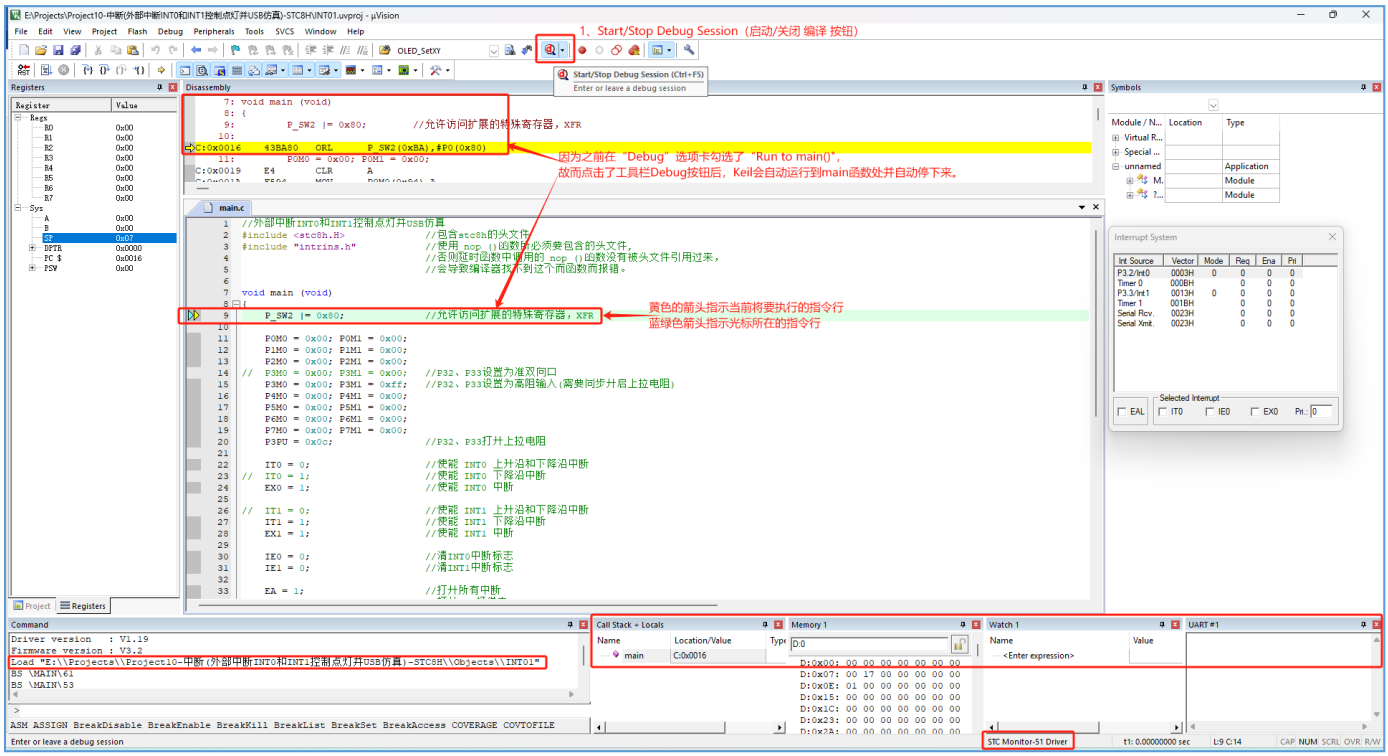
和 **Build Toolbar(Build 工具栏)**中在调试/仿真时的几个常用按钮:



1.9.1 File Toolbar（File 工具栏）

1.9.1.1 Start/Stop Debug Session，File Toolbar

Start/Stop Debug Session（Enter or leave a debug session）按钮：启动/停止 Debug 按钮。在软件开发过程中，调试是不可或缺的一环。Keil 提供了丰富的 Debug 调试工具，帮助我们快速定位和解决程序中的问题。点击此按钮，即刻进入 Debug 界面，如下图。再次点击此按钮，就会退出 Debug 界面。

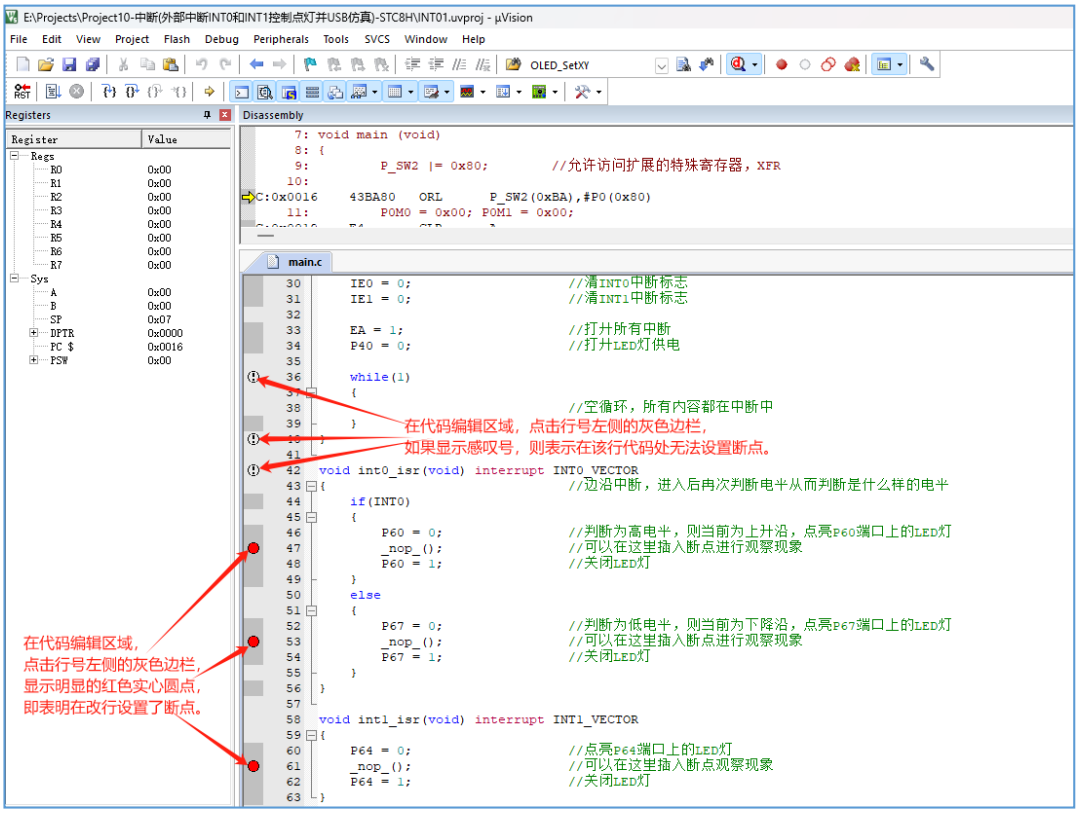


1.9.1.2 Breakpoint，File Toolbar

在调试/仿真程序时，会涉及一个很重要的工具：Breakpoint（断点）。

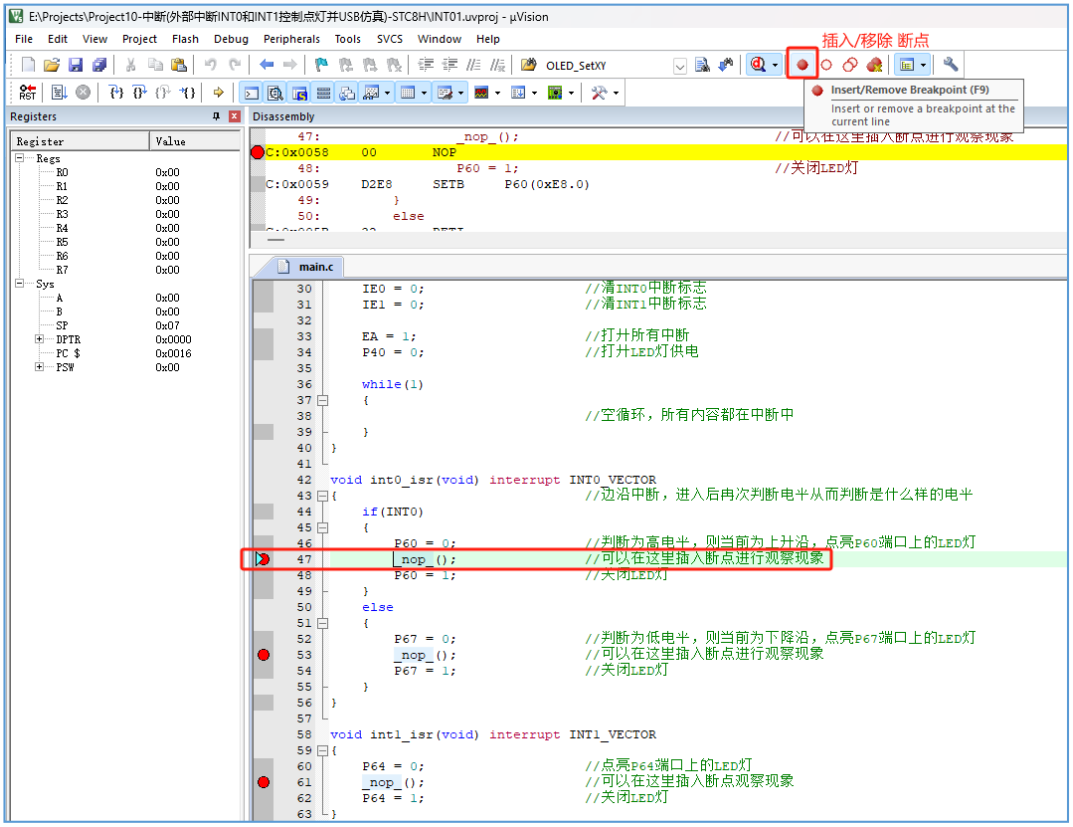
Breakpoint：断点。在 Keil 中，断点是一种强大的调试工具。断点本身是程序中的一个位置标记，当程序执行到断点位置时，程序会暂停执行，从而方便我们观察实验现象，定位、分析和排查程序中的问题等。在 Keil 环境中，设置断点有多种方式，下面我们介绍最简单的 2 种断点设置方法。

1、在代码编辑区域，点击行号左侧的灰色边栏，显示明显的红色实心圆点，即表明在该行代码处插入了断点。如果显示感叹号，则表示在该行代码处无法设置断点。在断点处点击断点（红色实心圆点）即为取消断点。



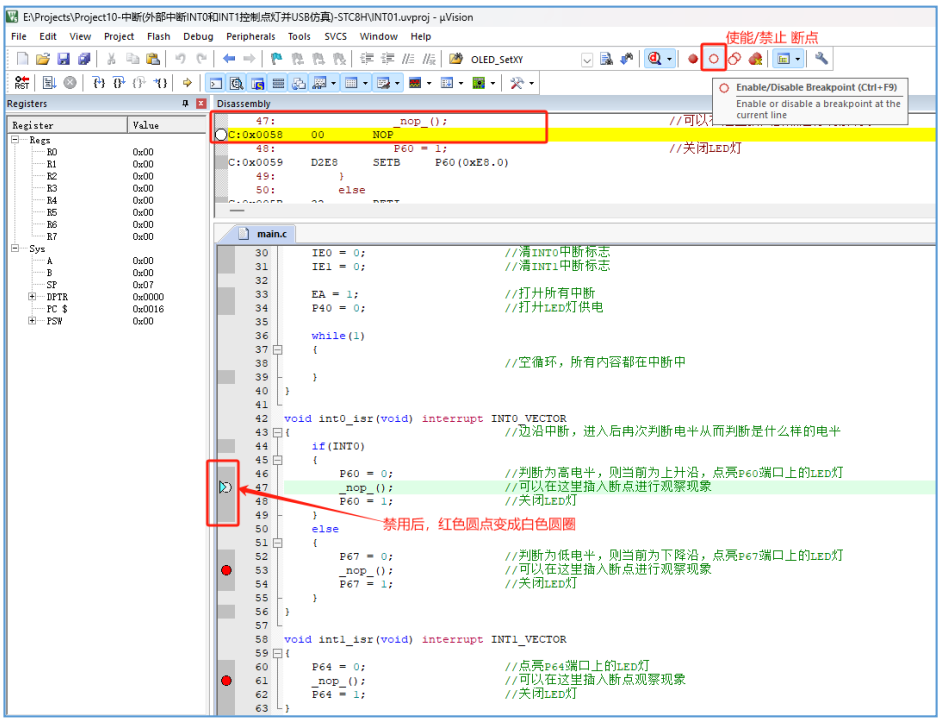
1.9.1.3　Insert/Remove Breakpoint，File Toolbar

2、Insert/Remove Breakpoint（Insert or remove a breakpoint at the current line）按钮：在当前光标所在行插入/移除断点。在当前光标所在行，该按钮如果显示红色，则表示可在该处插入断点。点击该按钮，即在当前行插入断点，当前行左侧就会显示一个红色圆点表示断点位置。该按钮如果显示灰色，则无法在该处插入断点。定位光标在断点所在代码行，点击该按钮，即移除该处断点，红色圆点消失。



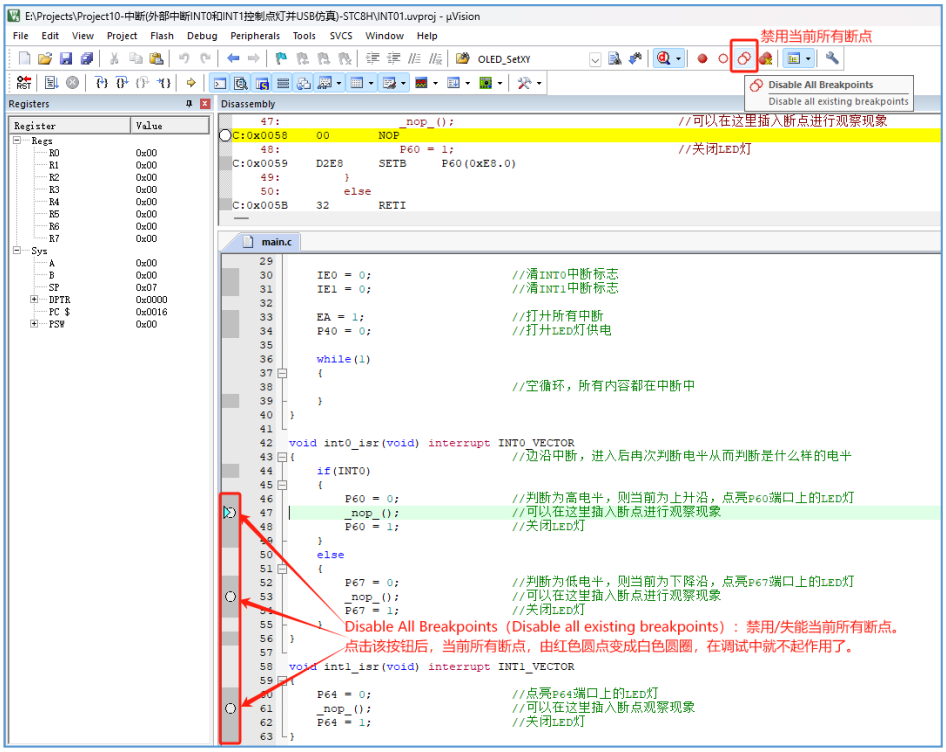
1.9.1.4 Enable/Disable Breakpoint，File Toolbar

Enable/Disable Breakpoint（Enable or disable a breakpoint at the current line）：使能/禁止断点。启用/禁用当前光标所在行的断点。断点在启用时，红色圆点显示。禁用后，红色圆点变成白色圆圈，在调试中就不起作用了。断点使能/禁止按钮，在设置断点后，你可以使用此功能按钮来启用或禁用断点，从而控制代码的暂停与继续。



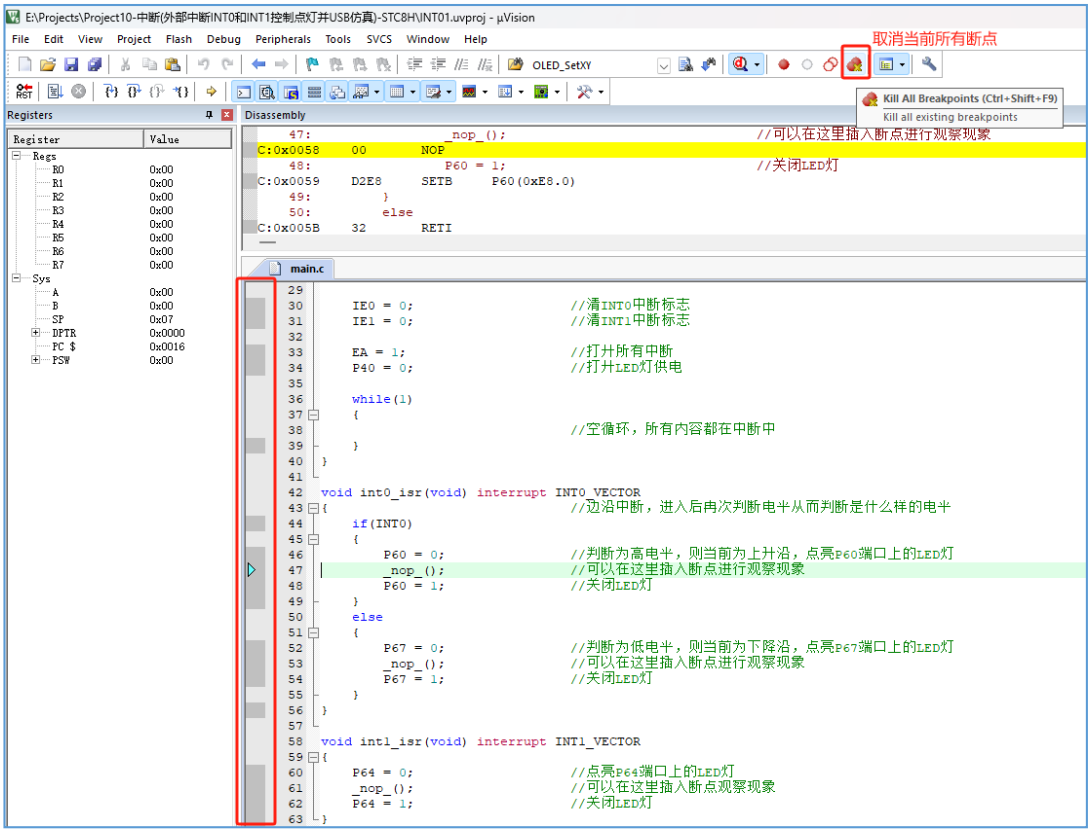
1.9.1.5 Disable All Breakpoints，File Toolbar

Disable All Breakpoints（Disable all existing breakpoints）：禁用/失能当前所有断点。点击该按钮后，当前所有断点，由红色圆点变成白色圆圈，在调试中就不起作用了。但此按钮不能再次使能所有断点。



1.9.1.6 Kill All Breakpoints，File Toolbar

Kill All Breakpoints（Kill all existing breakpoints）：取消当前所有断点。和 **Disable All Breakpoints** 不同，点击该按钮后，所有断点标记消失。

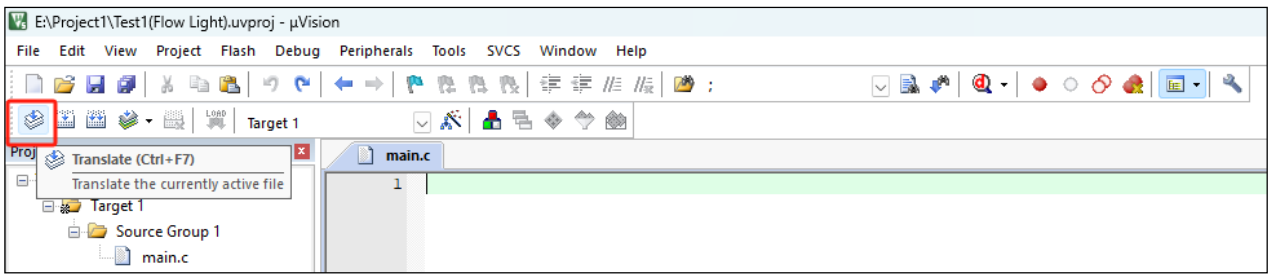


1.9.2 Build Toolbar（Build 工具栏）

Keil 是一款软件开发系统，输入工程项目的程序源代码后，要进行编译（Build）。编译过程是将 C 语言源代码转换为机器可以执行的机器代码（通常是 HEX 文件），这样程序才能在目标硬件上运行。编译过程中，编译器对源代码进行预处理，处理宏定义、条件编译指令等。然后将预处理后的源代码转换成汇编语言。这一步会检查语法错误，并将高级语言转换为低级语言。下面我们认识一下 Build 工具栏的几个重要的功能按钮。

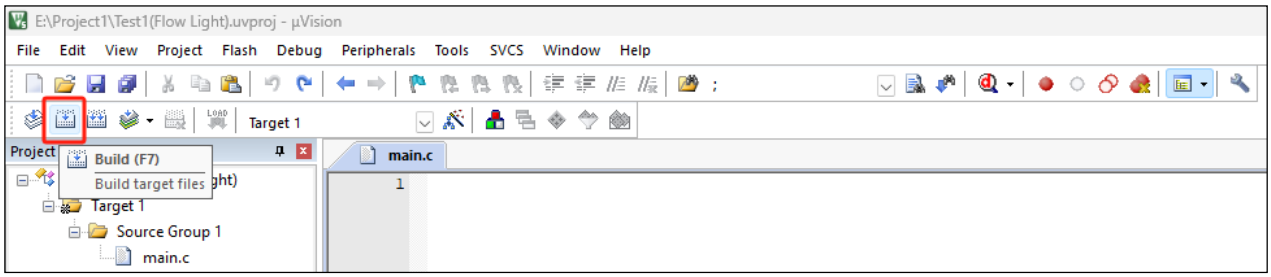
1.9.2.1 Translate（Translate the currently active file），Build Toolbar

Translate（Translate the currently active file）按钮：编译当前正在编辑的源文件(main.c 仅生成 main.o)，在 Translate 过程中会检查源文件中的语法错误，但并不生成最终可执行文件。



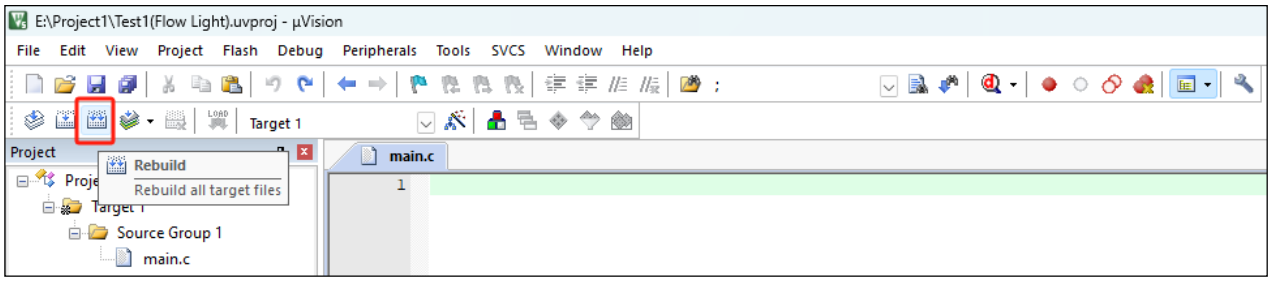
1.9.2.2 Build（Build target files），Build Toolbar

Build（Build target files）按钮：Build 在技术文档中是 increase build，所以理解 Build 按钮是编译工程中当前修改的文件及与此文件相关联的所有文件（模块），检查语法错误，并重新链接生成可执行文件。这样特别在调试大型项目时，因为不需要每次都从头开始编译整个项目，可以节省时间。平时项目开发调试验证过程中，也更多选择用 Build 编译生成可执行文件验证。如果工程是首次编译，会直接调用 Rebuild All。



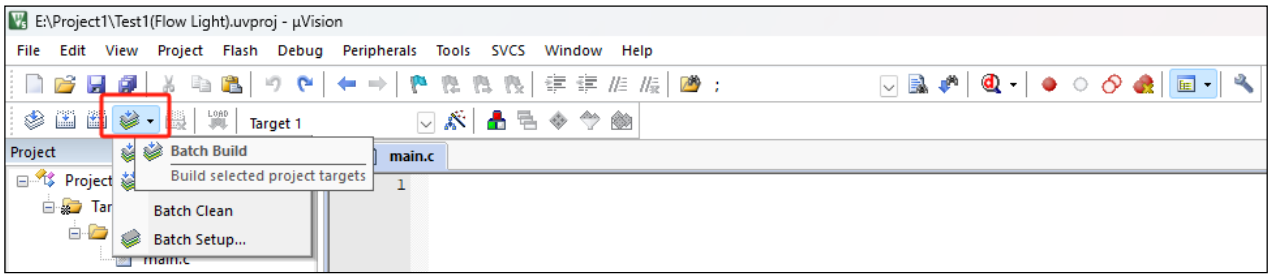
1.9.2.3 Rebuild（Rebuild all target files），Build Toolbar

Rebuild（Rebuild all target files）按钮：编译全部文件，检查语法错误，并重新链接生成可执行文件。



1.9.2.4 Batch Build（Build selected project targets），Build Toolbar

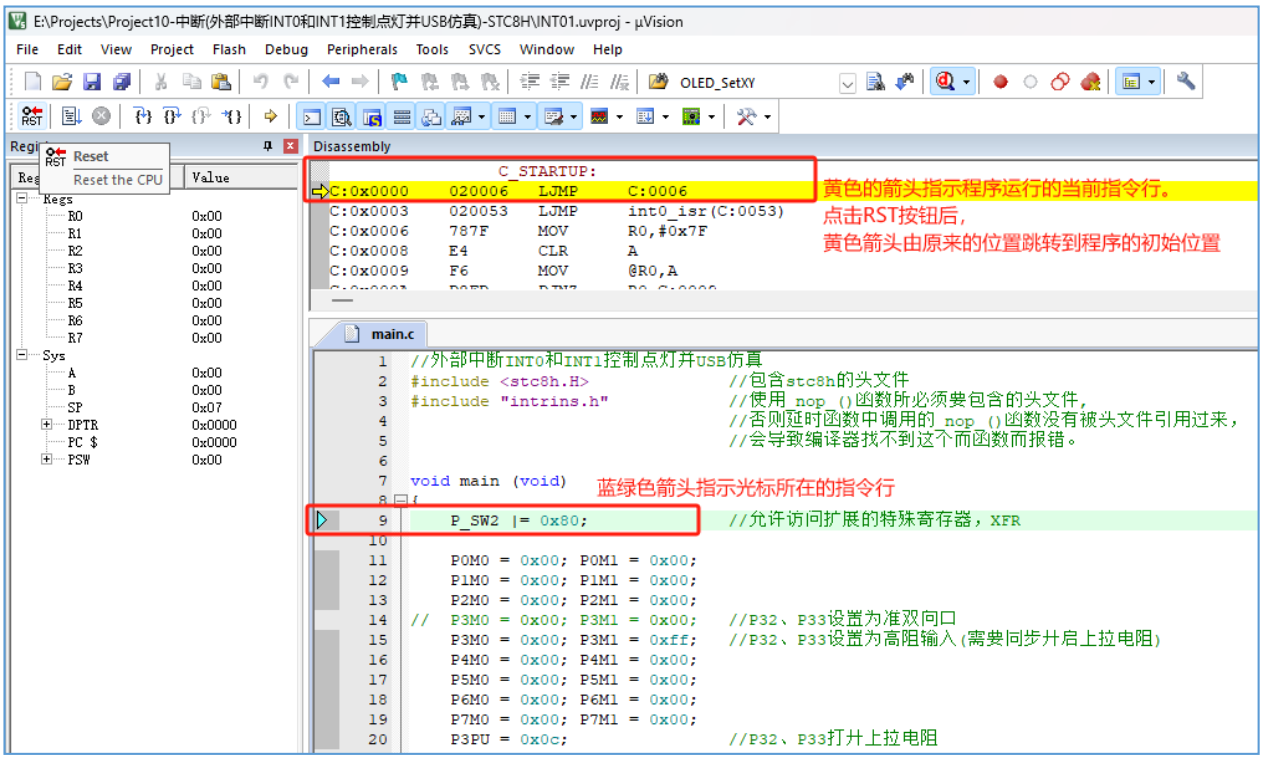
Batch Build（Build selected project targets）按钮：一次编译多个项目。在 Keil 中，可创建多个 projects，并通过 Batch Build 同时对这些 projects 进行编译。



1.9.3 Debug Toolbar（Debug 工具栏）

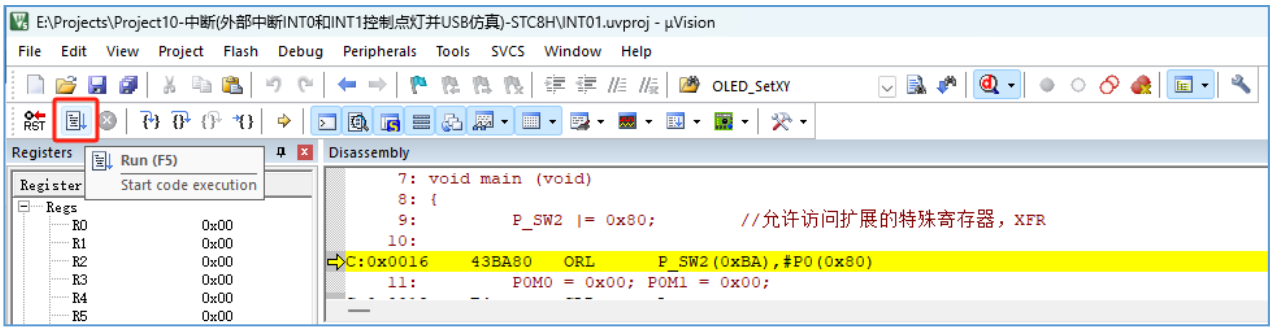
1.9.3.1 RST（Reset the CPU），Debug Toolbar

RST（Reset the CPU）：类似复位按钮，相当于实现了一次硬复位。点击该按钮之后，无论程序代码执行到哪个位置，代码都会重新从头开始执行，本例黄色箭头指向 C:0x0000（可以对比上图黄色箭头位置）。



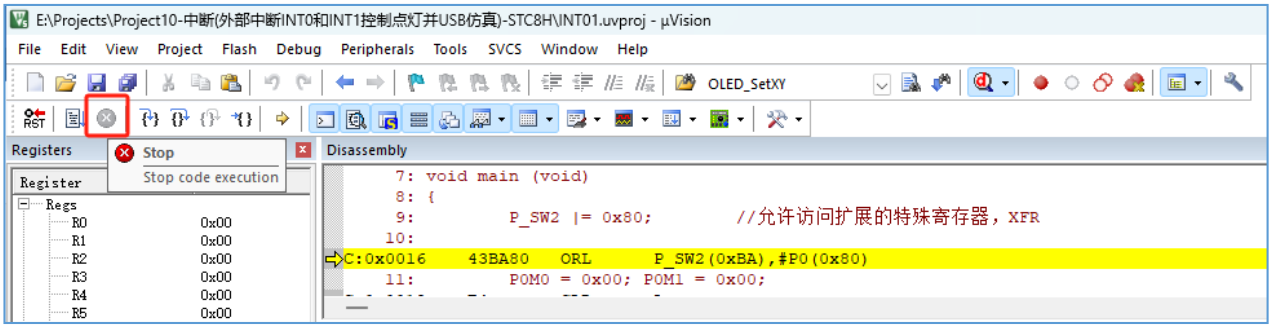
1.9.3.2 Run（Start code execution），Debug Toolbar

Run（Start code execution）：运行按钮。点击该按钮后，程序将一直处于运行状态，或者直接快速运行到断点处（前提是你在查看的地方设置了断点）。有时候你并不需要观看程序的每步是怎么执行的，而是想快速的执行到程序的某个地方看结果，这个按钮就可以实现这样的功能。



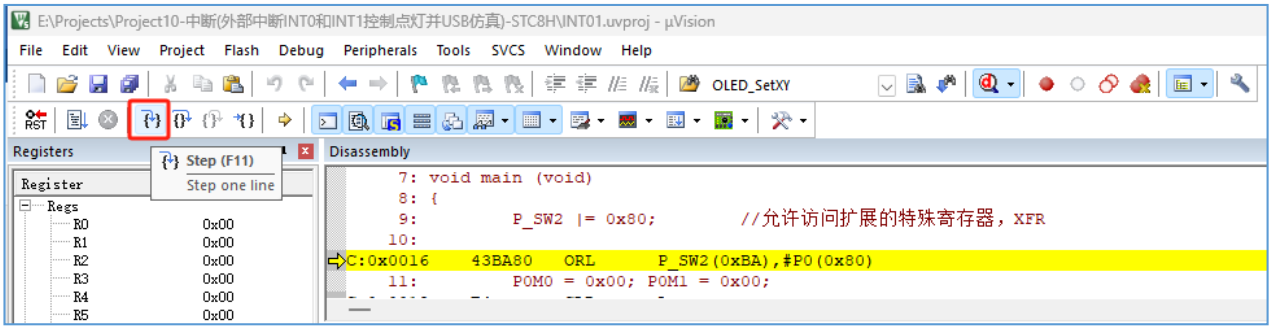
1.9.3.3 Stop（Stop code execution），Debug Toolbar

Stop（Stop code execution）：停止运行按钮。此按钮在程序一直执行的时候会变为有效，通过按该按钮，就可以使程序停止下来，进入到单步调试状态。



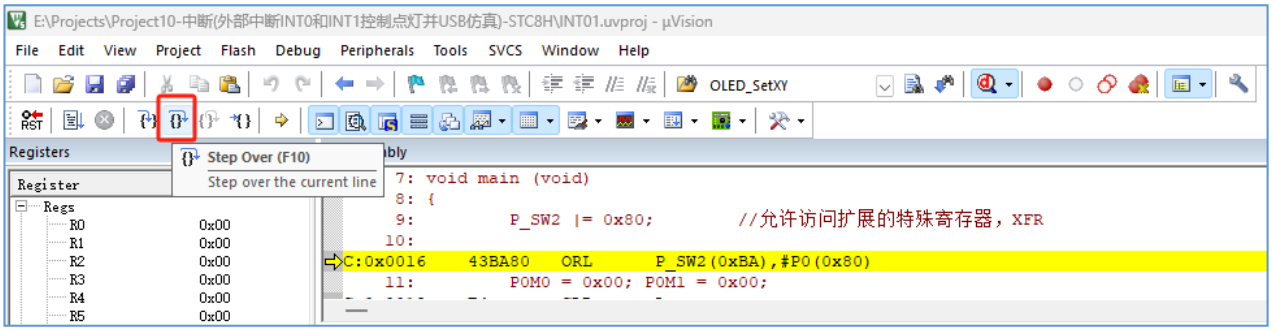
1.9.3.4 Step（Step one line），Debug Toolbar

Step（Step one line）：单步运行按钮。每点击一次该按钮，程序将执行单条语句。如果遇到函数，则会进入函数内部执行。这是调试程序时最常用的功能之一。此按钮在程序停止运行时有效。

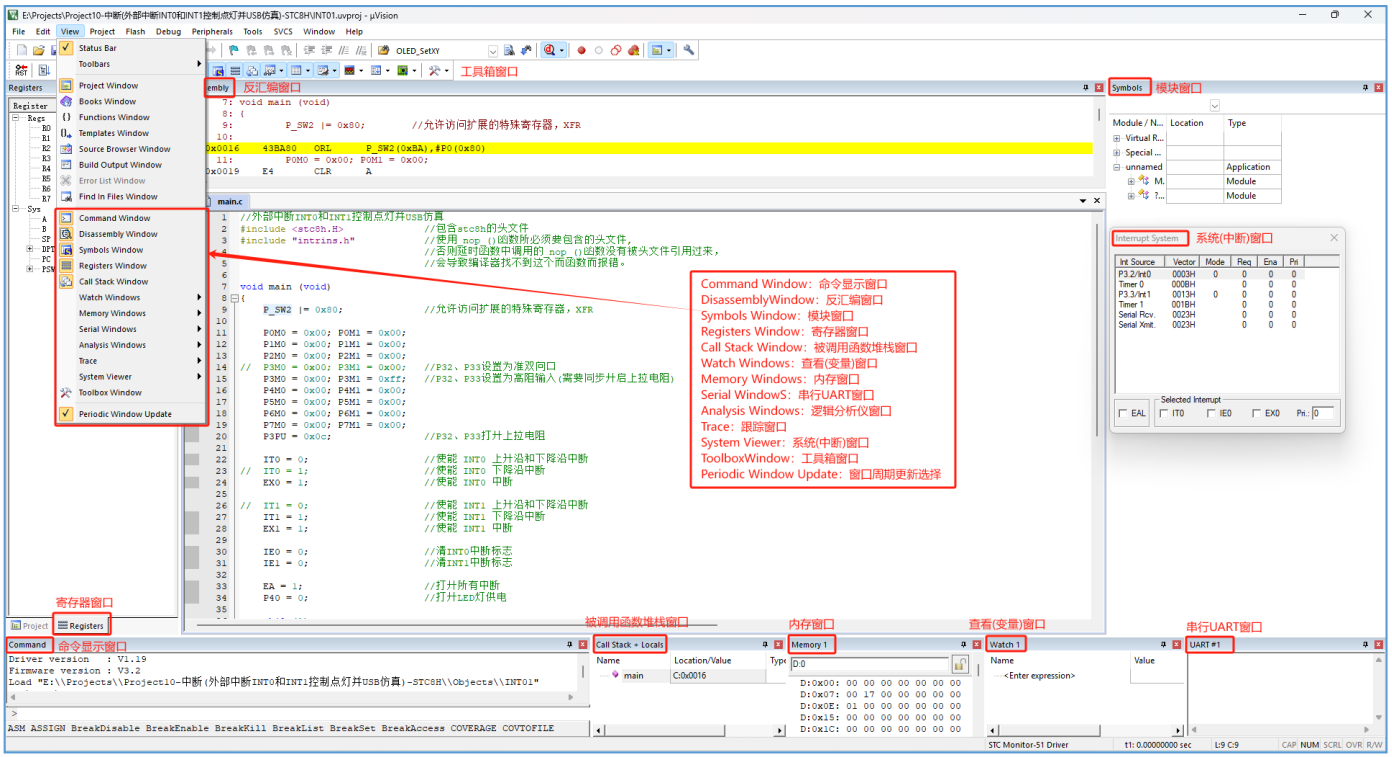


1.9.3.5 Step Over（Step over the current line），Debug Toolbar

Step Over（Step over the current line）：单步跳过按钮，程序运行一行，按单条语句执行。与 Step 按钮不同的是，Step Over 按钮遇到函数不会进入函数内部，而是直接全速运行函数，并跳到下一条语句。在碰到有函数（特别是延时函数）的地方，通过该按钮就可以单步执行过这个函数，而不进入这个函数单步执行。此按钮在程序停止运行时有效。



1.9.4　观察变量/寄存器/内存（数据存储器、程序存储器）窗口



2 实验

2.1 实验一：printf_usb("Hello World !\r\n")，第一个完整的 C 语言程序

看下面的程序，其实还是那个熟悉的标准 C 语言的 printf()，只是到了单片机 C 语言中，变成了 printf_usb()

2.1.1 printf( )，printf_usb( )实验程序代码

```
#include   "stc8h.h"          //头文件不分大小写，但关键字分大小写
#include   "ai_usb.h"         //USB 库和 USB 虚拟设备的头文件，需要在项目文件中添加对应的 USB 库
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。

void   main(void)
{
    P_SW2 |= 0x80;    //允许访问扩展的特殊寄存器，XFR

    usb_init();
    //初始化 USB 模块，并且会将【P3.0, P3.1】初始化为高阻输入，不影响 USB【D-, D+】通讯
    EA = 1;           //使能总中断

    while (1)
    {
        printf_usb("Hello World !\r\n");
                      //直接打印输出 “Hello World !” 到主机相应软件的 USB【CDC/HID-串口助手】
                      //  \r 是回车，跳到行首，\n 是换行，就是换到下一行；\r\n 不要交换次序
    }
}
```

以上是完整的第一个程序代码

printf_usb()函数：

printf_usb()函数和标准 C 语言的格式完全一致，详见 printf()函数章节
进入 printf_usb()函数后，首先会判断 USB 主机是否已经对 USB 设备配置完成，
如果没有配置完成，printf_usb()函数会直接返回退出。
如果已经配置完成，还会继续判断如有发送前一笔数据，则判断是否发送完成，
如果没有发送完成，会继续等待，直到前一笔数据发送完成，
才会发送新的 printf_usb()需发送的数据后再返回退出。
但如等待超过 50ms@24MHz，会直接退出。
如果主时钟不是 24MHz，20MHz 等待时间会变长，40MHz 等待时间会变短。
如果已经配置完成且前一笔数据已经发送完成，
则会直接发送新的 printf_usb()需发送的数据并返回退出。

如何获得上面"stc8h.h"和"ai_usb.h"这 2 个头文件

如已按照前面介绍的方法：
从 www.STCAI.com 网站，【软件工具 | 工具软件】处，下载安装最新的【AiCube-ISP 软件】，
选择其中的选项卡 【Keil 仿真设置 】，

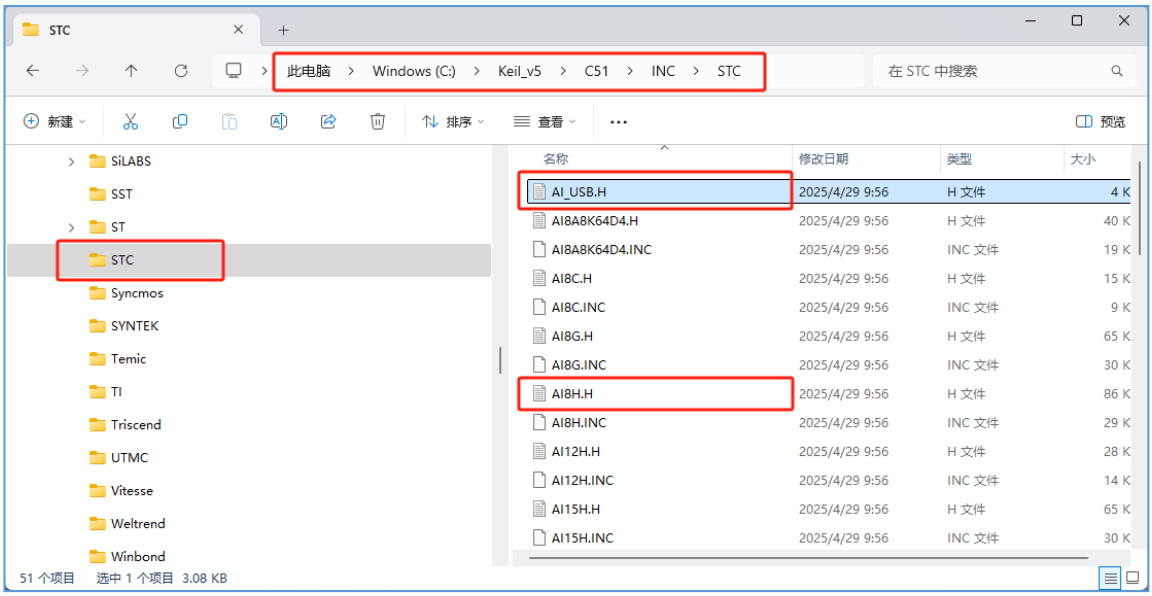
点击【添加型号和头文件到 Keil 中，并添加仿真器驱动到 Keil 中】按钮，
则 stc**8h.h** 和 **ai_usb.h** 都会自动复制到 Keil 中，包含进来就可以直接使用，
大家可以打开看下其中的内容，**ai_usb.h** 和 stc32_stc8_usb.h 是相似的，只需用其中的 1 个头文件，具体见 Keil_v5\C51\INC\STC 目录，或 Keil_v5\C251\INC\STC 目录。
如将对应的 USB 库文件添加到 项目中，后续用户程序中，USB 如收到命令字符串 "@STCISP#" USB 库函数就会自动软复位到系统 ISP 程序区，实现 USB 不停电下载，具体见相应部分的说明。

2.1.2　准备工作，如已准备，跳过此节

1、添加常用头文件 ai_usb.h

```
#include "ai_usb.h"                    //使用 usb 所需要包含的头文件。
```

这个头文件在使用 AiCube-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



2、添加 usb_cdc 库函数文件

因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载



下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_8h_data.LIB

【8 位的 8051 的 USB 库 可以用 BL51 连接器

BL51： 不支持 REMOVEUNUSED；BL51 支持 扩展中断号工具；

如果 不用 REMOVEUNUSED，可以 使用 BL51 ！

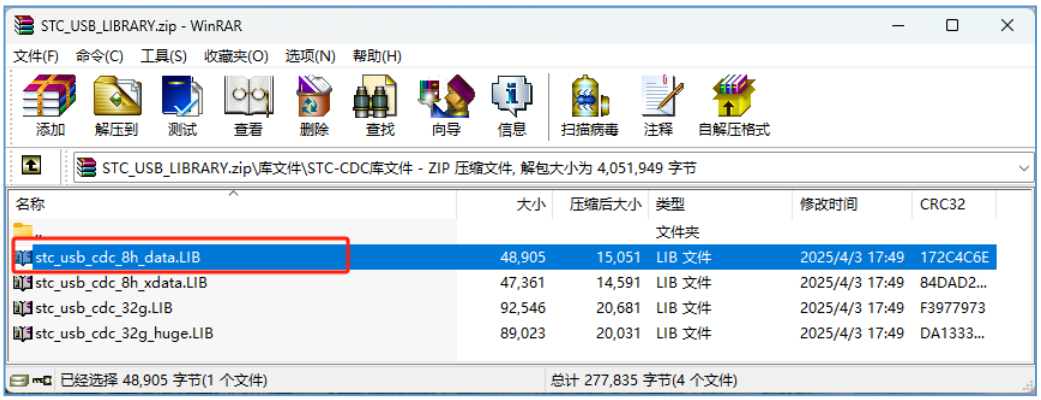
LX51： 支持 REMOVEUNUSED；LX51 不支持 扩展中断号工具；

如果 不用 扩展中断号工具，可以 使用 LX51 ！

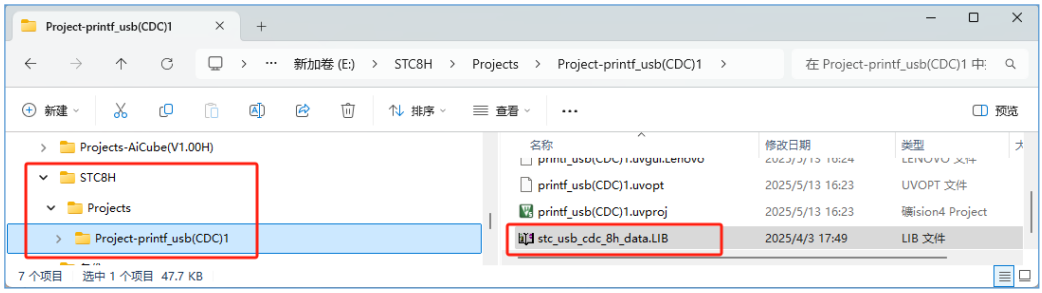
详情请参考本公司技术论坛链接：8 位的 8051 的 USB 库可以用 BL51；BL51 不支持 REMOVEUNUSED；

https://www.stcaimcu.com/thread-17105-1-1.html

(出处: 国芯人工智能技术交流网站)】

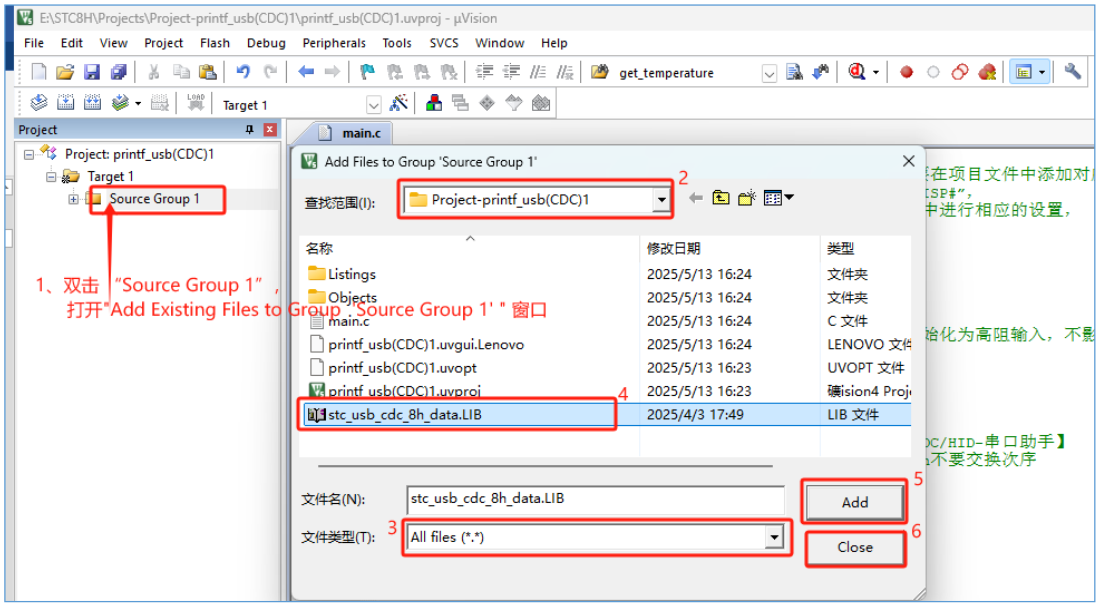


将库函数“stc_usb_cdc_8h_data.LIB”复制到创建工程所在的文件夹 E:\STC8H\Projects \Project-printf_usb(CDC)1 中：

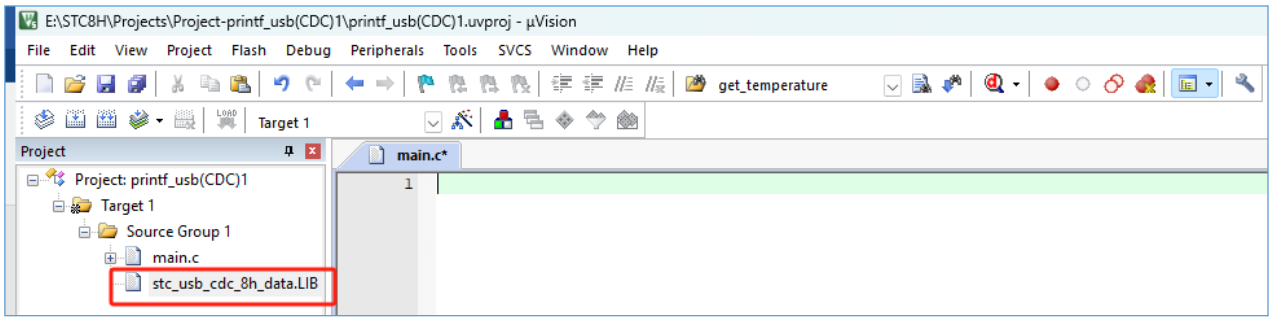


将库函数 stc_usb_cdc_8h_data.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在选择“Add Existing Files to Group 'Source Group 1”菜单项，打开“Add Files to Group 'Source Group 1”窗口中，将库函数 stc_usb_cdc_8h_data.LIB 添加进入此工程，就可以使用 USB-CDC 功能。



这样，Keil 界面变化如下图：



【注意】:

USB-HID 通讯不需要安装驱动，手机及 Windows 各种系统都自带 HID 驱动。

USB-CDC 通讯，手机不需要安装驱动，手机的各种系统已自带 USB-CDC 驱动。

USB-CDC 通讯，Win10（1903）及以后版本不需要安装 USB-CDC 驱动，较早的 Windows 版本才需要手动点击 ISP 软件中的【USB 驱动安装】，只有较早的 Windows 系统才需要安装 USB-CDC 驱动程序。

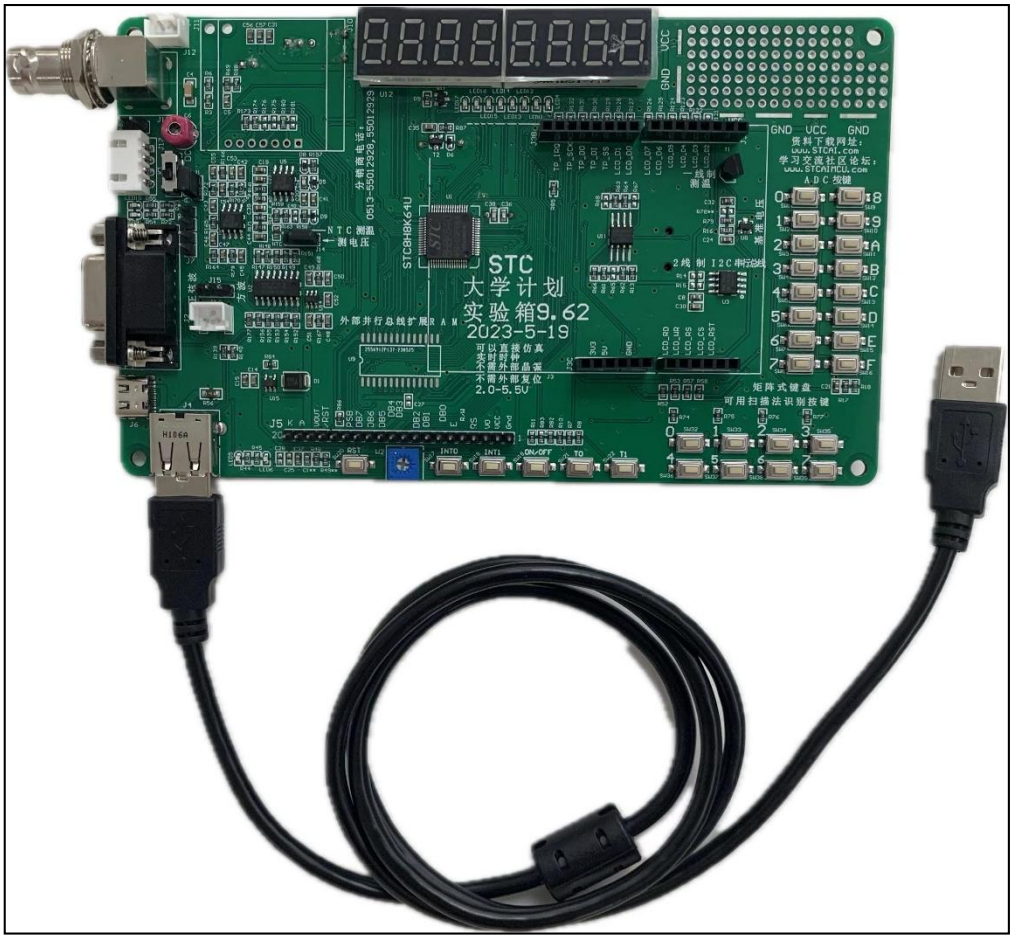
USB-CDC 虚拟串口类似传统串口的形式，但不需要进行波特率的设置，通讯时是以 12Mbps 的速度进行通讯，USB-CDC 虚拟串口转串口才需要设置转传统硬件串口需要的波特率、奇偶校验位等参数。

录入实验程序代码，保存，编译。

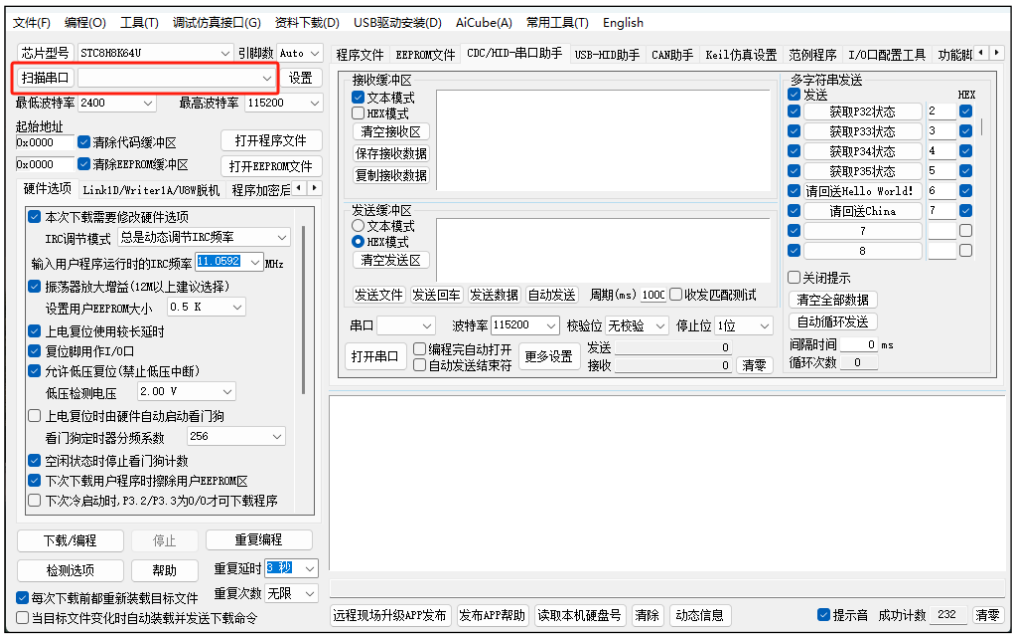
2.1.4　下载用户程序到实验箱，观察实验现象

2.1.4.1　连接实验箱和电脑

拿出 STC8H8K64U 实验箱中配置的 USB-TypeA 到 USB-TypeA 线将实验箱与电脑正确连接。

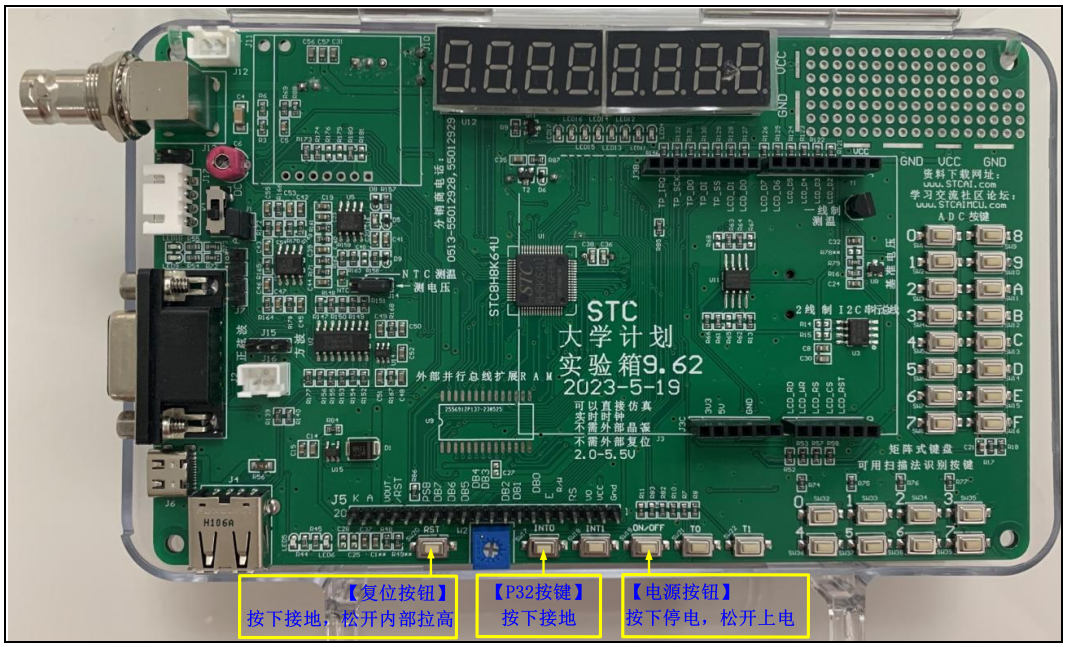


然后打开 AiCube-ISP-v6.95N 或以上版本。选择单片机型号：STC8H8K64U，如下图，我们看到“扫描串口”后面的端口列表框不是“（HID1）USB-Writer”，说明还不能下载程序到实验箱。

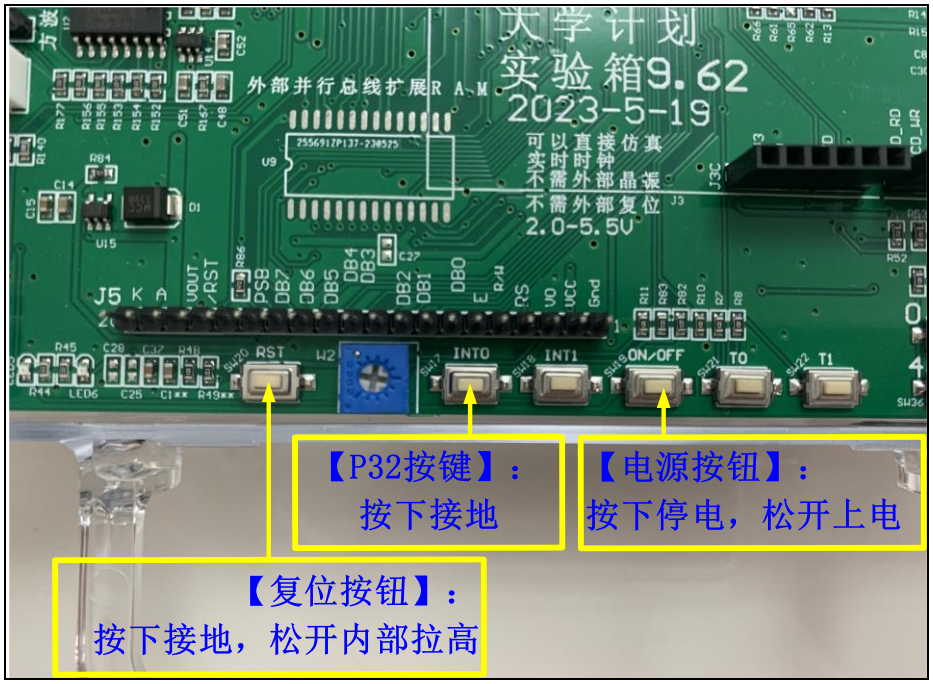


2.1.4.2　硬件 USB 下载用户程序步骤

下图是 STC8H8K64U 实验箱已经和电脑相连的正面图片，数码管是随机显示的，**此实验的目的把 STC8H8K64U 实验箱作为 USB-CDC 设备和电脑进行通讯，收发数据。**



局部放大实验箱图片，其正下方有一排按键，找到 P3.2 按键和 Power_SW 按键



【方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===本 STC8H8K64U 实验箱是：按下电源按钮停电，再松开电源按钮上电，

这时电脑端 ISP 下载软件会自动识别出“(HID1) USB Writer”，此时就与 P3.2 状态无关了，可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）

电脑端 ISP 下载软件提示：下载进行中…，几秒钟后，提示下载成功！

【方法二：复位管脚低电平复位下载】

USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

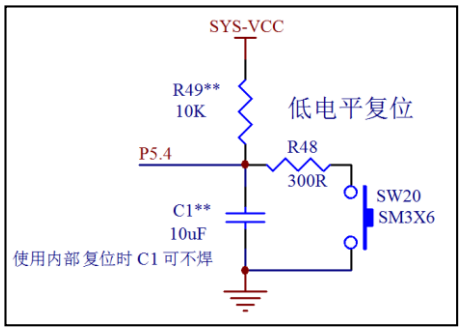
P5.4-RST 出厂时默认是 P5.4-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

1、按下复位管脚/P5.4-RST 外接的低电平复位按键，复位 MCU，再松开复位键，MCU 首先从系统程序区启动运行系统程序，判断是否要下载用户程序，

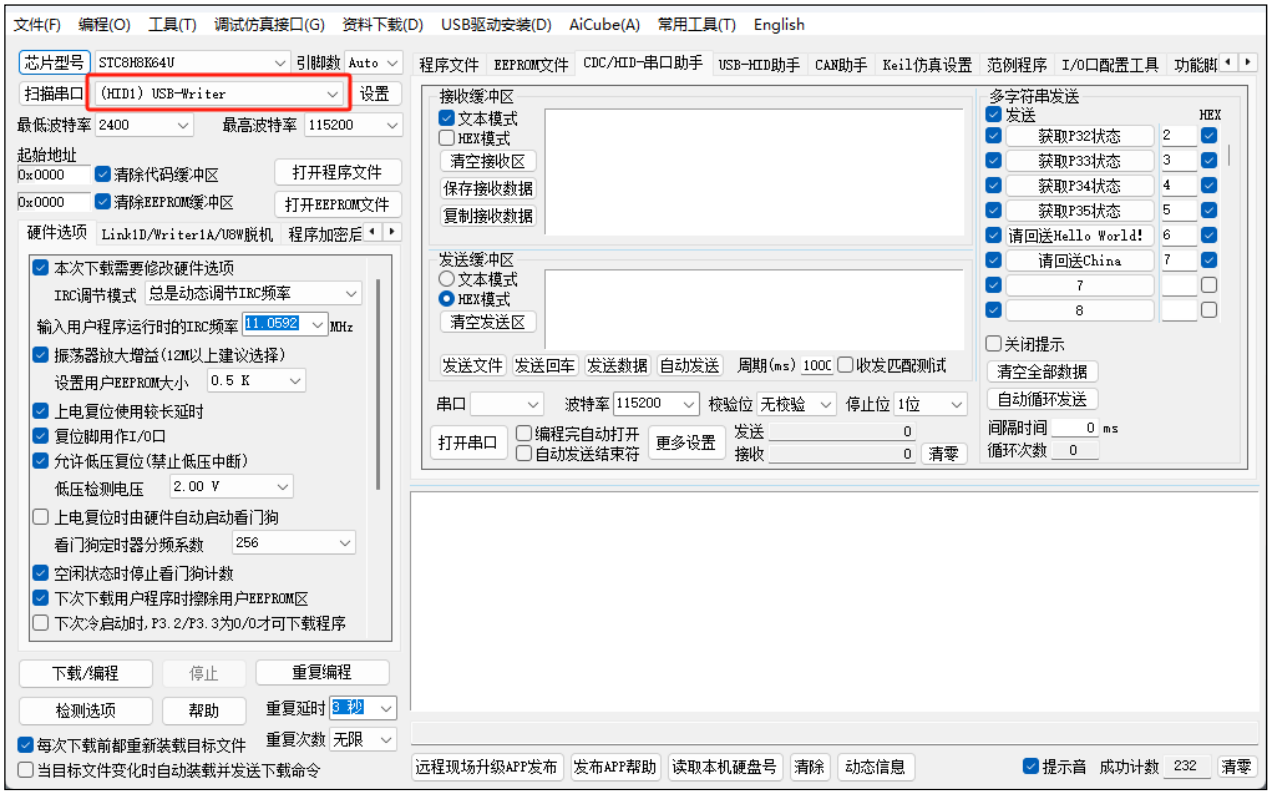
等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮

电脑端 ISP 下载软件提示：下载进行中…，下载成功！

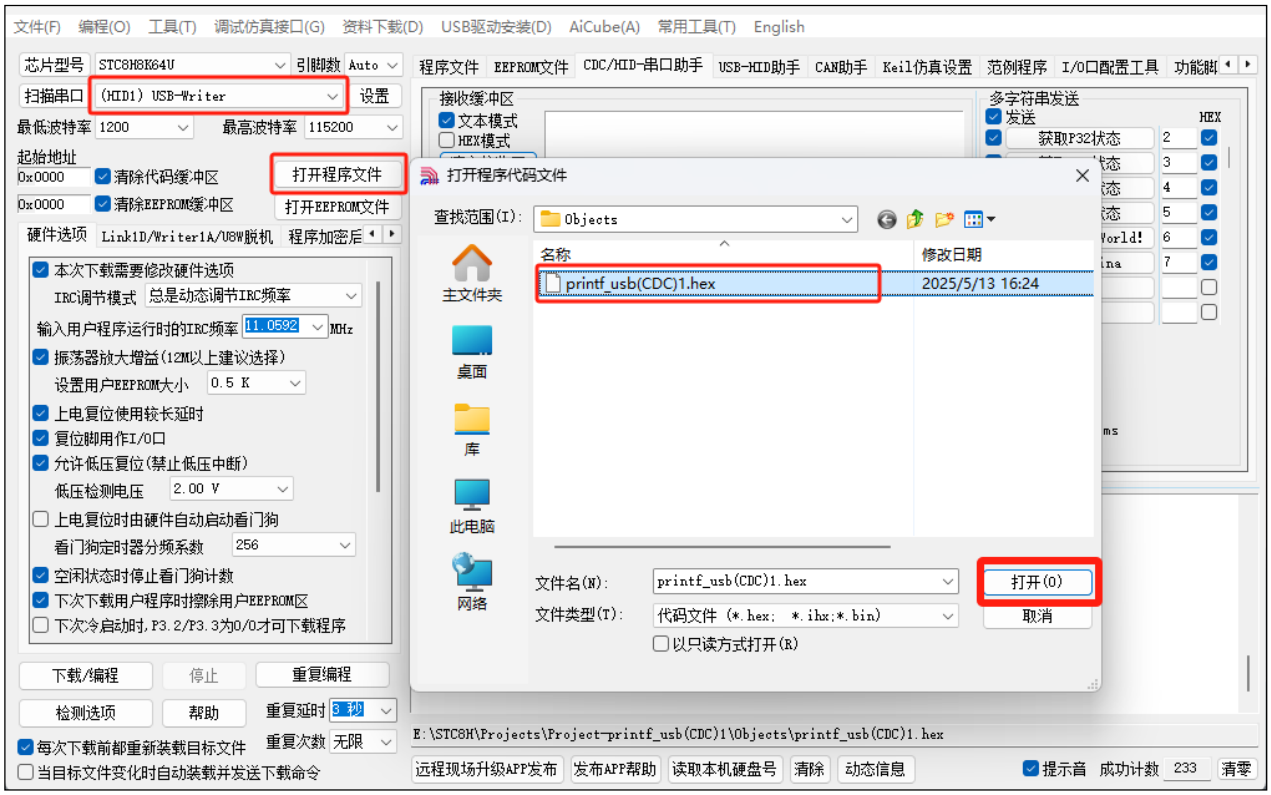


备注：如果目标 MCU 的用户程序区已有 USB 通讯程序(含 CDC、HID 等)，则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载

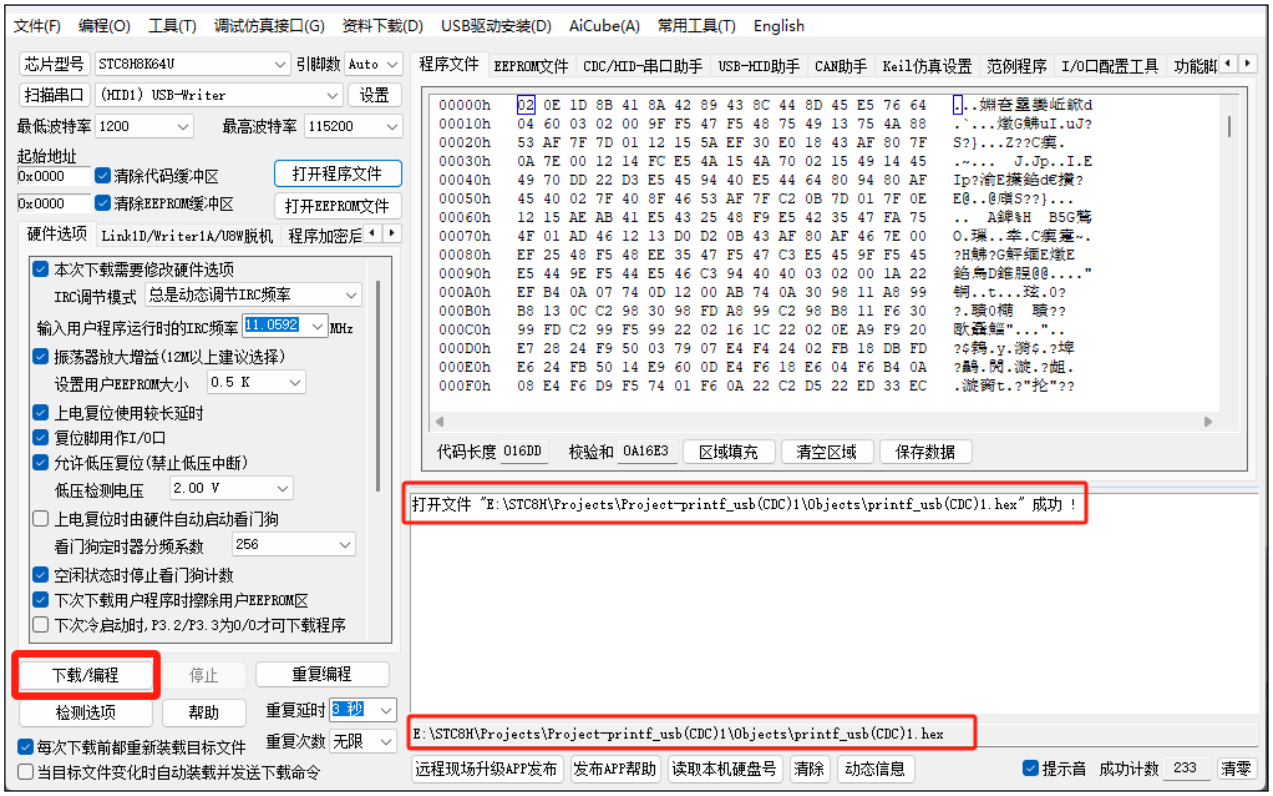


2.1.4.3　下载用户程序文件并查看实验现象

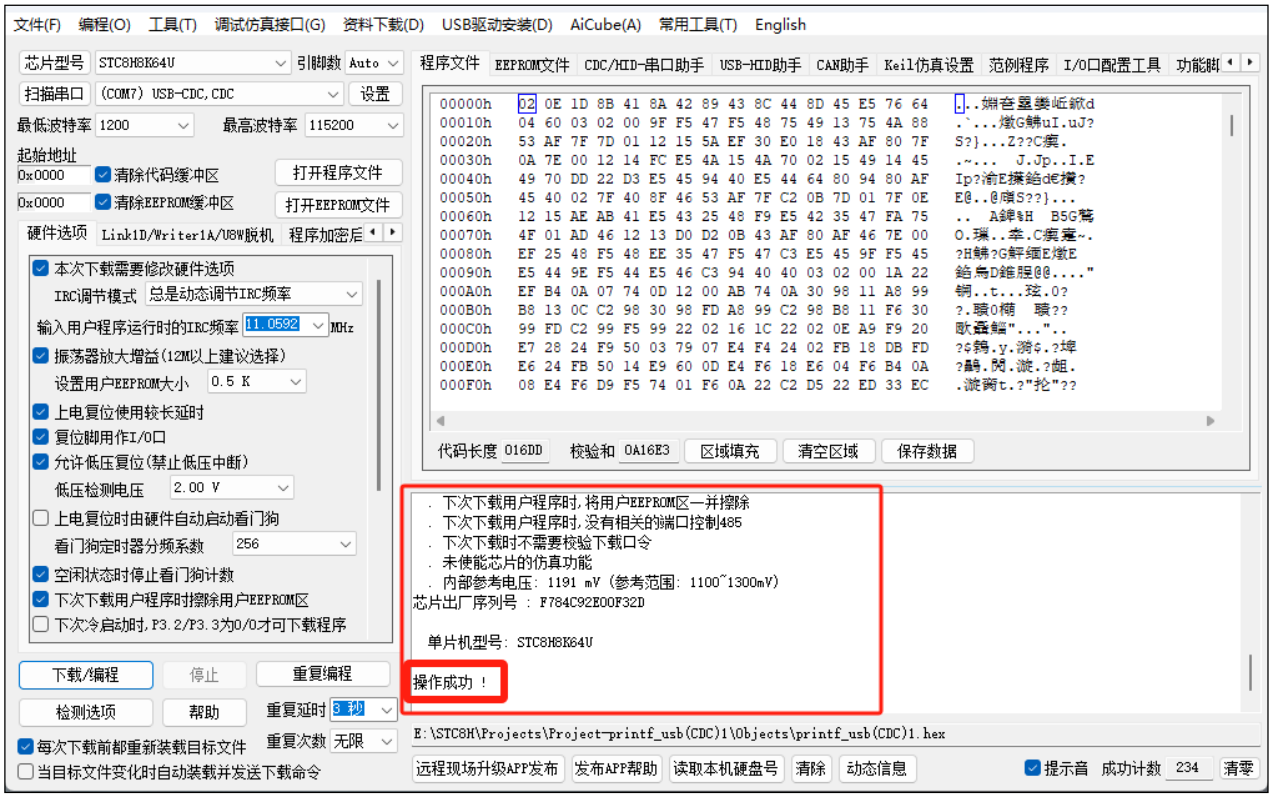
现在就打开用户程序文件，点击 AiCube-ISP 软件界面上的“打开程序文件”，调入在前面小节编译好的 HEX 文件：



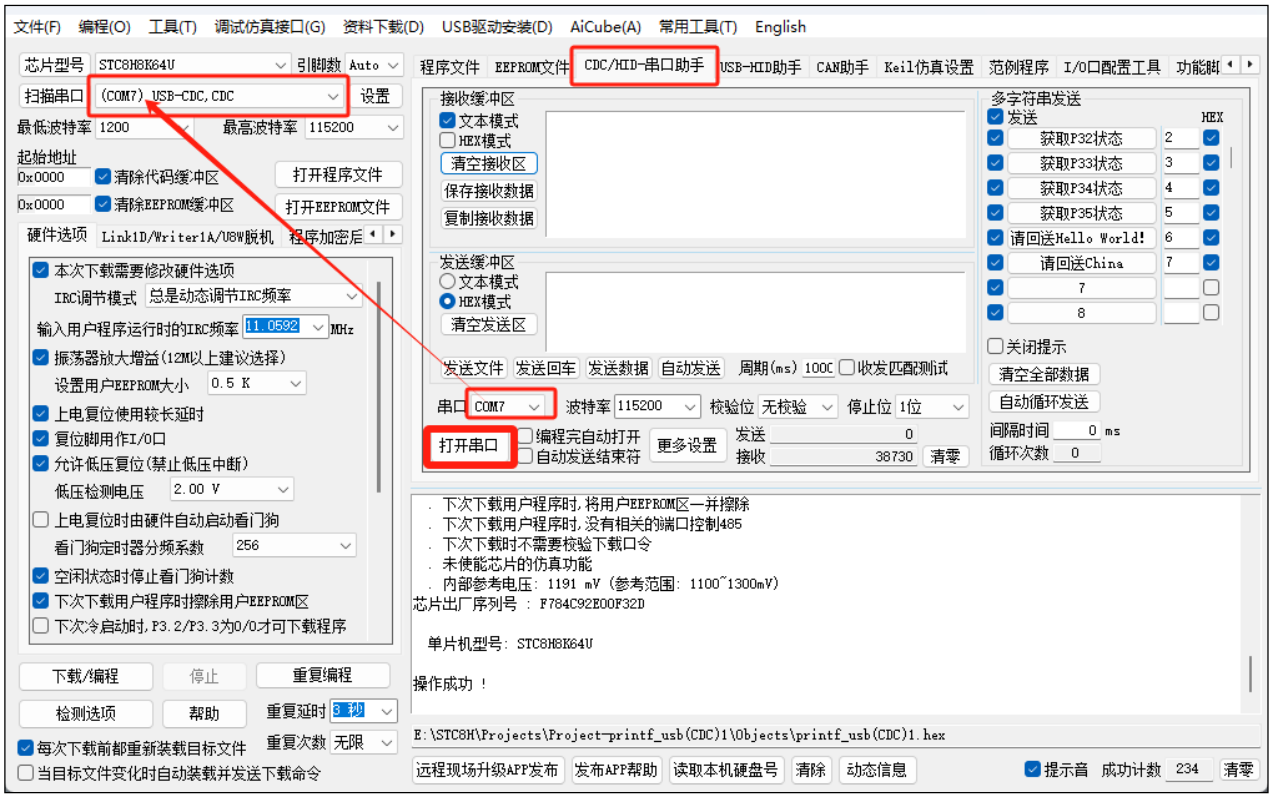
然后，AiCube-ISP 软件界面如下图所示，出现“‘打开文件***.hex’成功!”



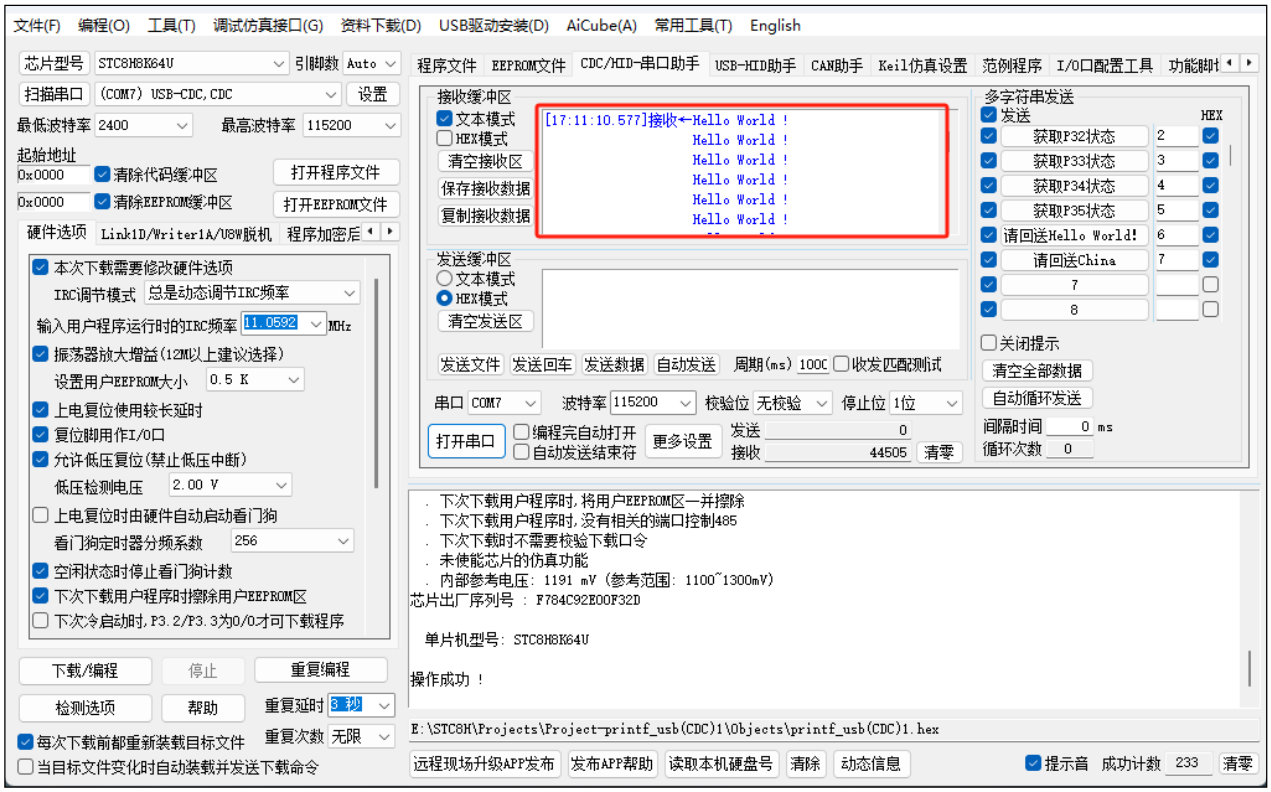
点击“下载/编程”，把 print_usb(CDC)1.hex 烧录到实验箱，AiCube-ISP 软件界面出现：操作成功！恭喜你！



我们看到此时串口显示：（COM7）USB-CDC，CDC，选择 AiCube-ISP 系统软件的“CDC/HID-串口助手”选项卡，找到“打开串口”按钮，如下图：



选择好对应的串口号，点击“打开串口”按钮，CDC/HID-串口助手的“接收缓冲区”，就会不停的显示“Hello World!”，如下图：



至此，printf_usb()显示“Hello World!”，实验成功！

2.1.5 使用【AiCube】工具创建 -- printf_usb("Hello World !\r\n"), 第一个完整的 C 语言程序

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit**、**Ai8051U-8Bit**、**STC32G12K128**、**STC8H8K64U**、**STC15F2K60S2**、**STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

2.1.5.1 安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj



下载下来的是压缩包文件及解压缩文件如下图：



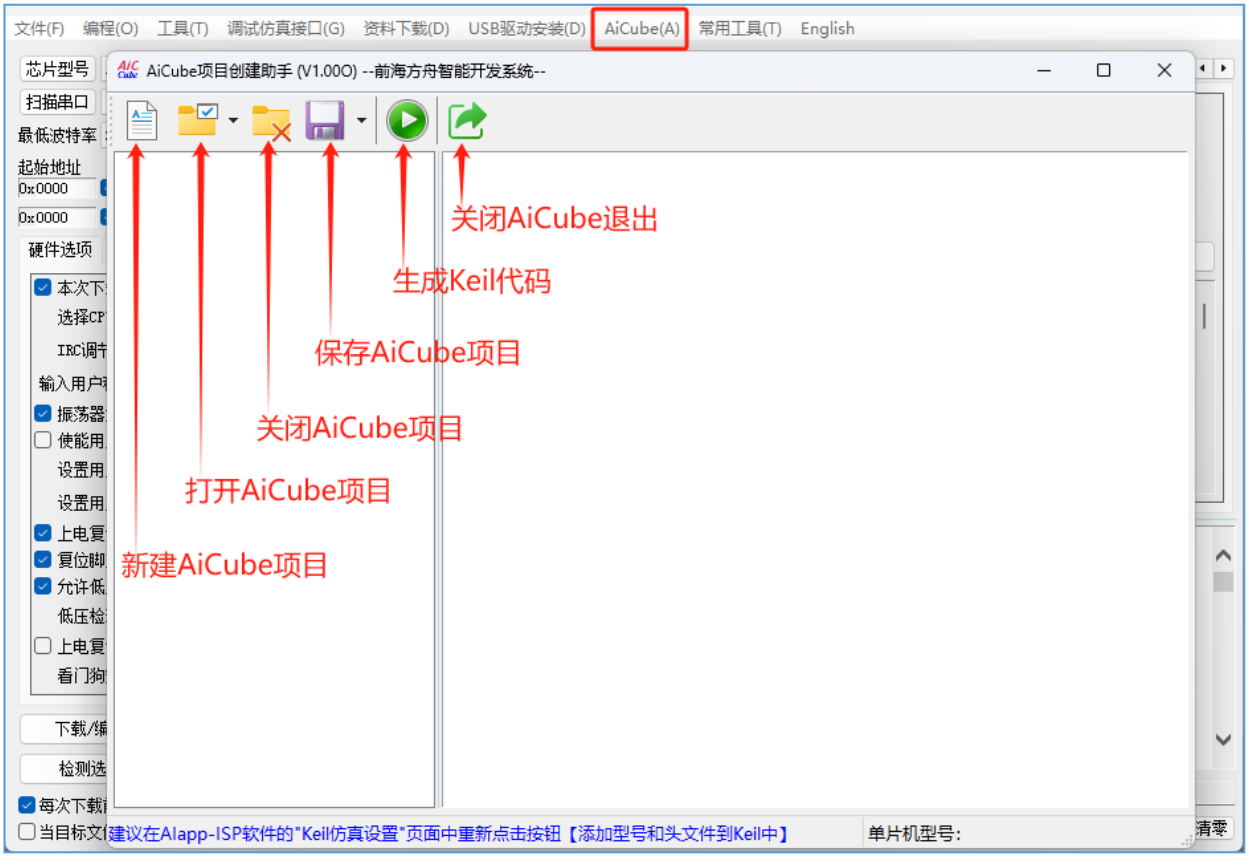
如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

2.1.5.2　使用【AiCube】工具创建项目

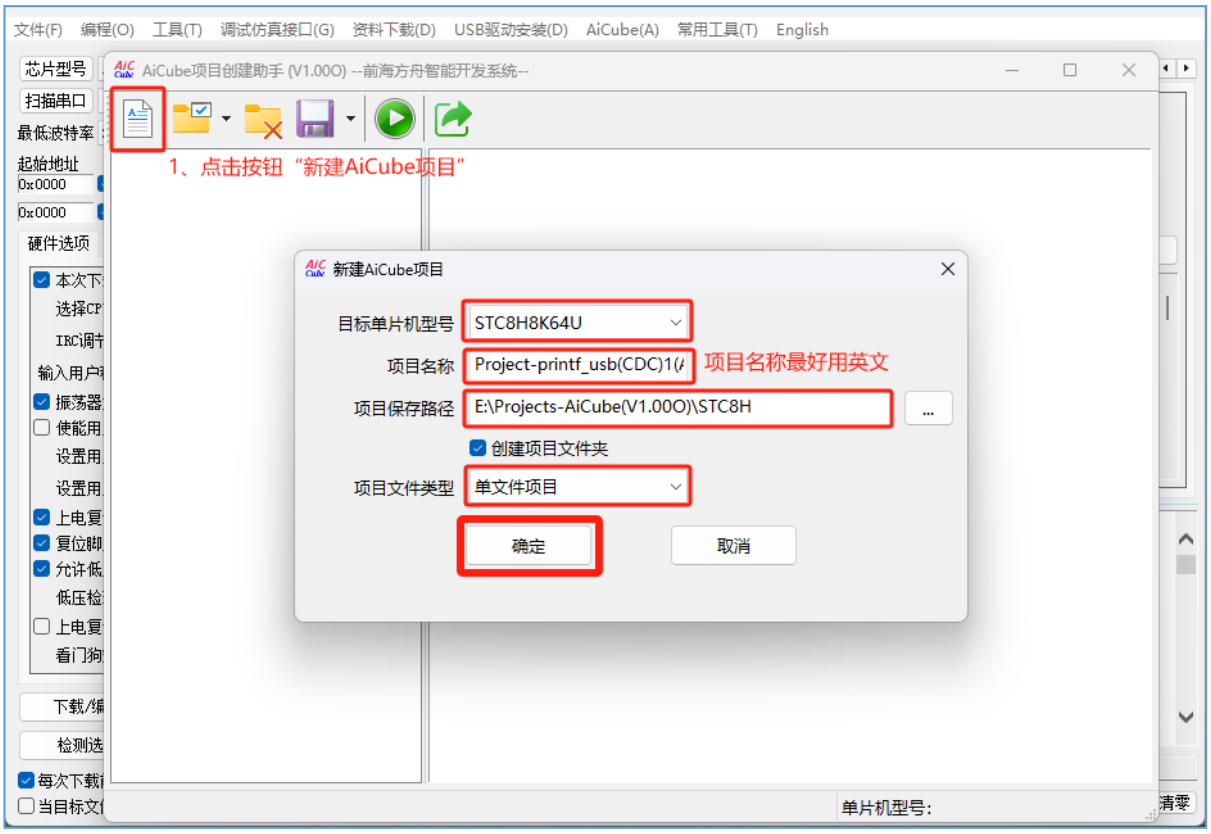
双击图标，打开 AiCube-ISP 软件界面，即可直接使用，如下图，在菜单栏上找到【AiCube(A)】菜单项：



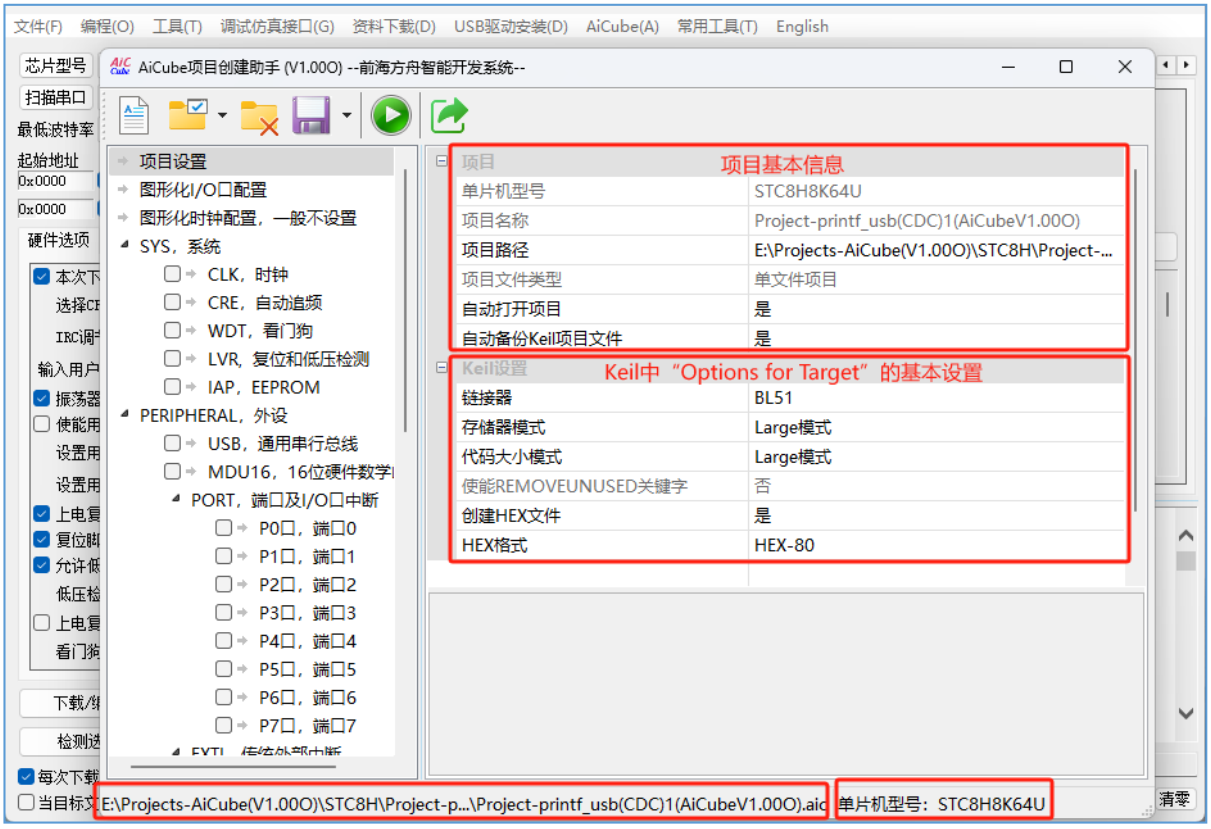
➲ **点击【AiCube（A）】**，弹出如下窗口：



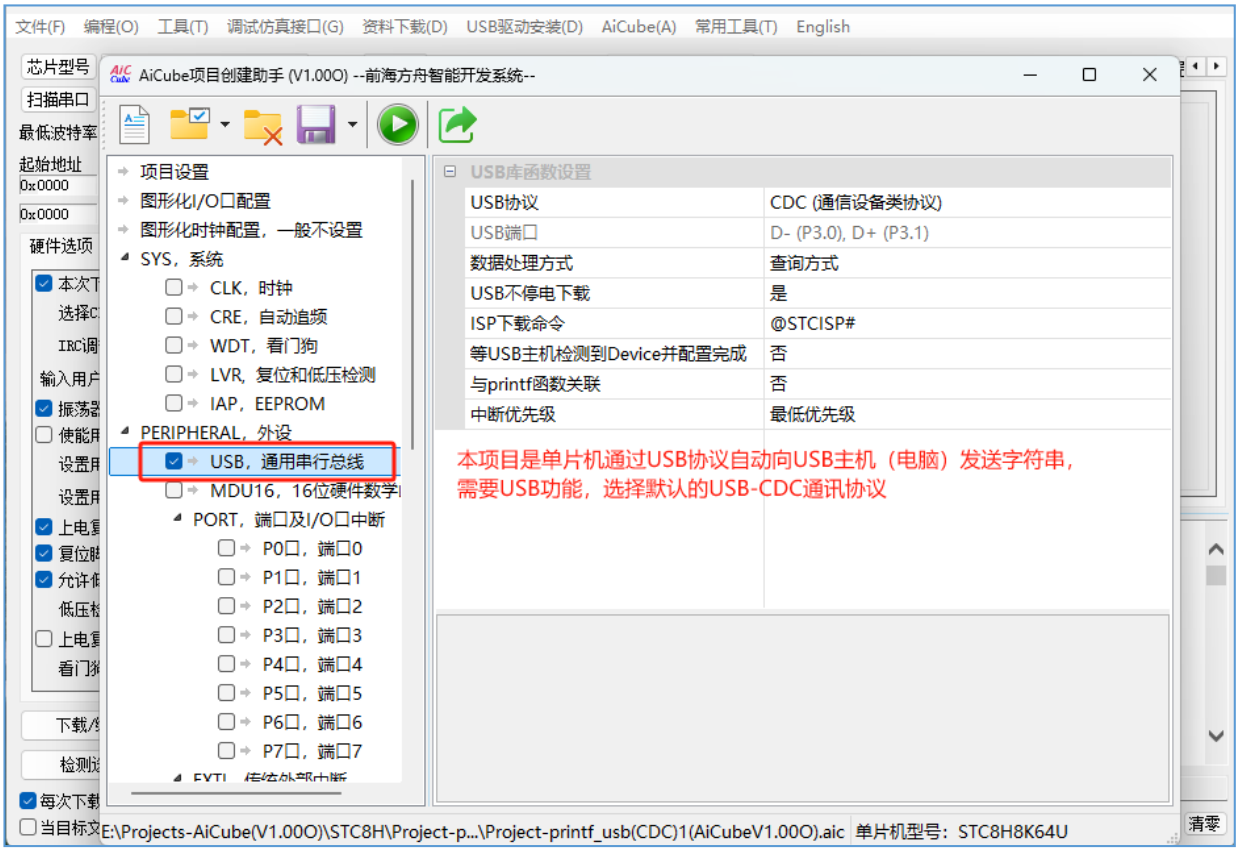
➲ **点击按钮“新建 AiCube 项目”**，出现如下浮窗，选择并填写项目基本信息，建议项目名称用英文！



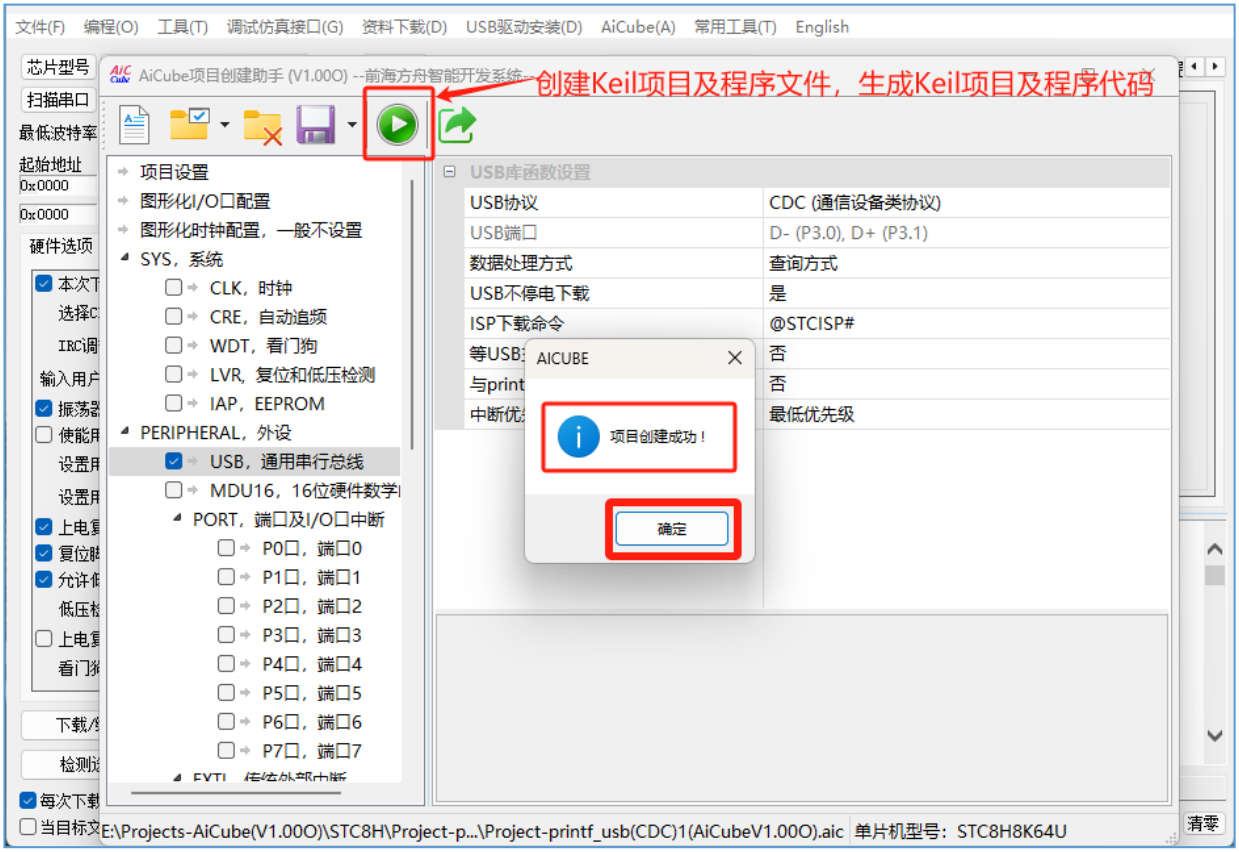
➲ **点击“确定”**，出现如下界面：



因本项目用到 USB-CDC 功能，就需要加载 USB 库函数、USB-CDC 协议，要对 USB 初始化等。本项目只需要使用 USB-CDC 发送数据，不需要读取，所以，不需要对 USB 外设进行任何设置。在【AiCube】助手中，根据本项目的实际需求，只需要勾选 USB 外设即可，如下图：



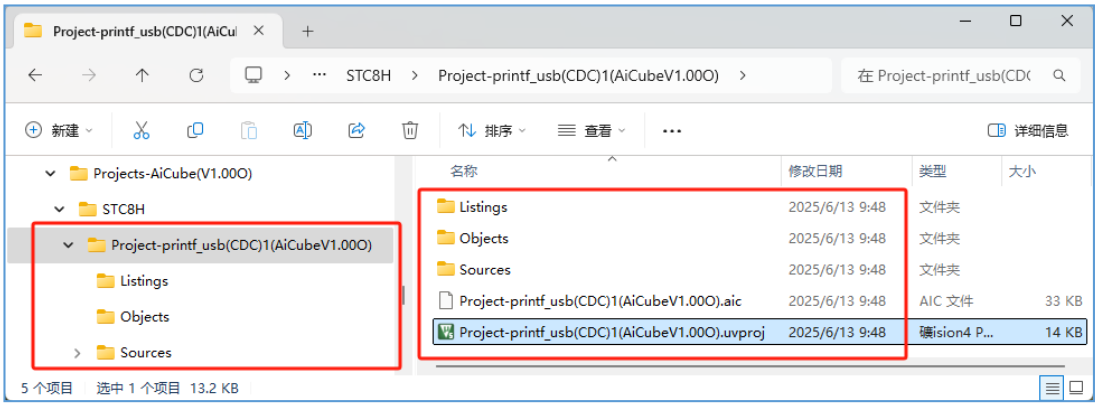
➲ **点击工具栏上的“创建 Keil 项目及程序文件”按钮，**【AiCube】提示：“项目创建成功！”，如下图：



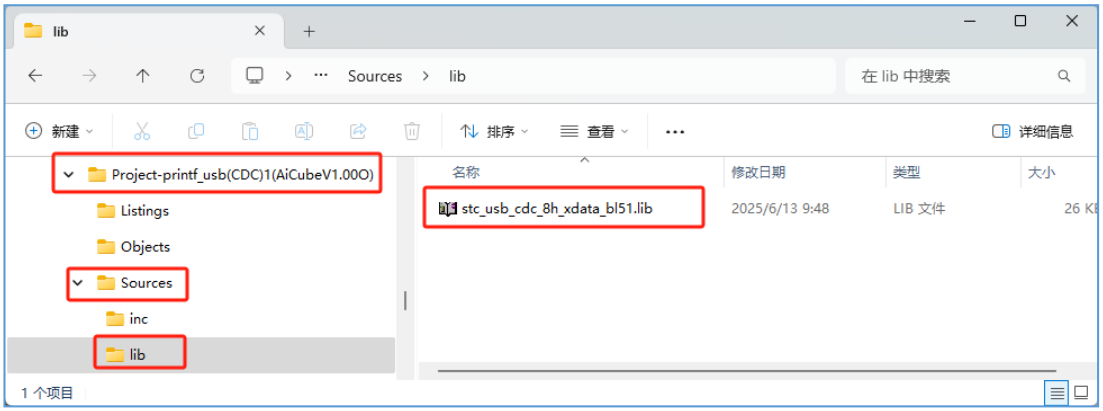
点击“确定”，立即打开 Keil 软件，进入该项目的 Keil 集成开发环境界面，即【AiCube】成功帮我们创建了项目及项目文件目录“Project-printf_usb(CDC)1(AiCubeV1.00O)”。至此，可以关闭【AiCube】工具。

➲ 找到并打开刚刚创建的项目目录

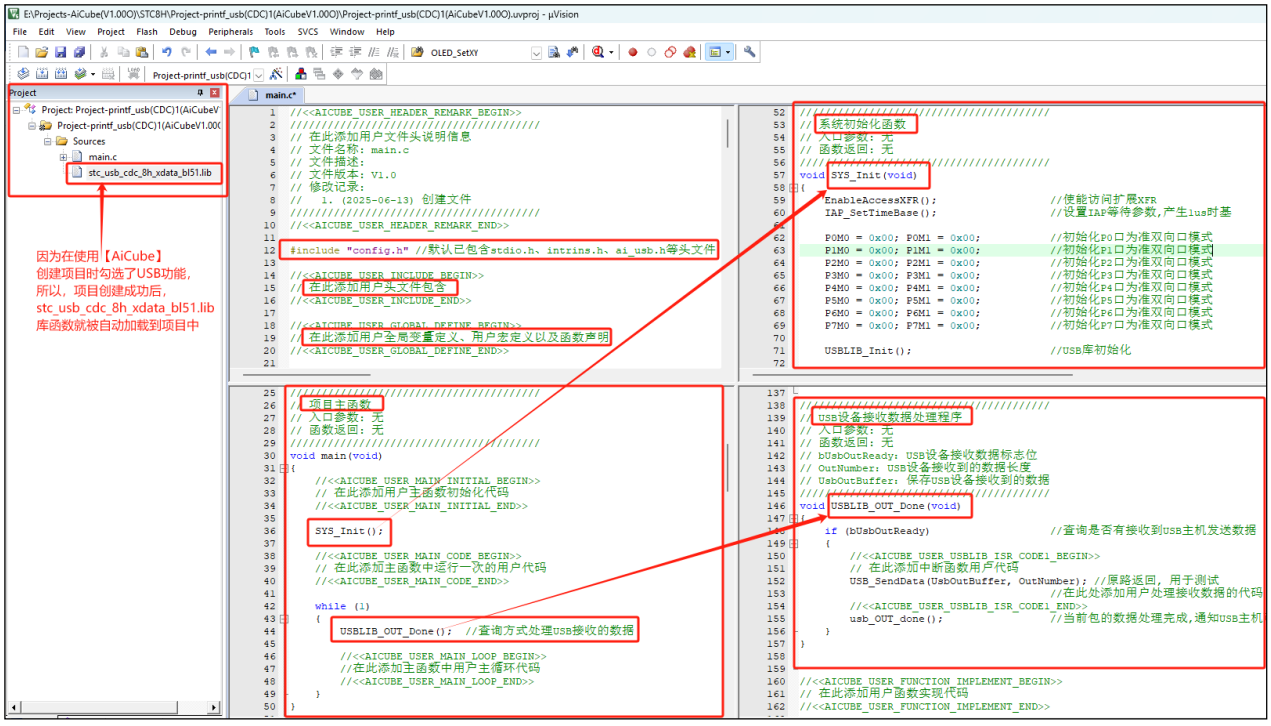
发现通过上述的操作过程,【AiCube】工具已经帮我们建立了项目目录及 Project-printf_usb(CDC)1(AiCube V1.00O).uvproj 文件，如下图：



同时打开 Sources>lib 子目录，发现，【AiCube】助手将相应的 USB 库函数加载进项目目录：



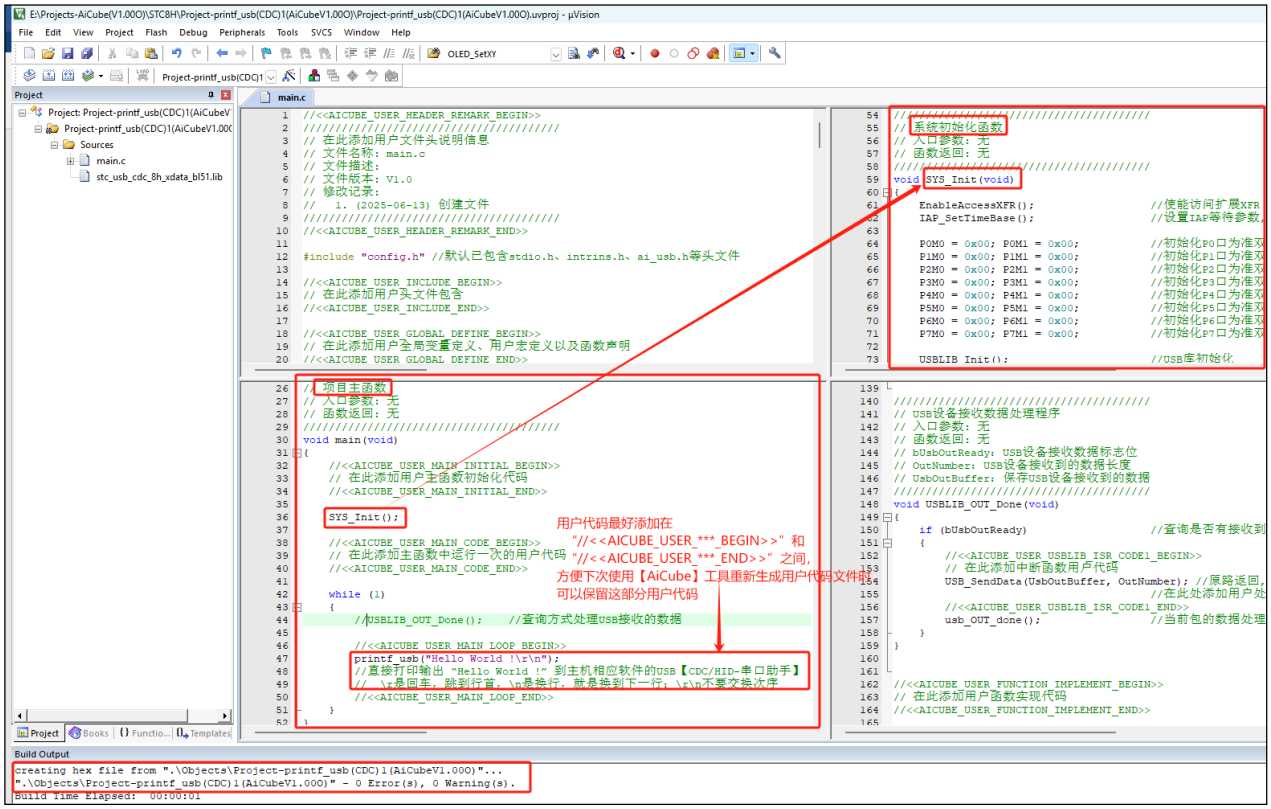
➲ 观察已打开的 Keil 项目文件：Project-printf_usb(CDC)1(AiCubeV1.00O).uvproj，如下图：



根据在【AiCube】工具中的配置，从上图中我们看到【AiCube】创建的本实验项目中不仅有了 main.c，还已经自动加载了 USB 库函数（stc_usb_cdc_8h_xdata_bl51.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、微秒及毫秒延时函数、USB 相关的配置及处理函数。

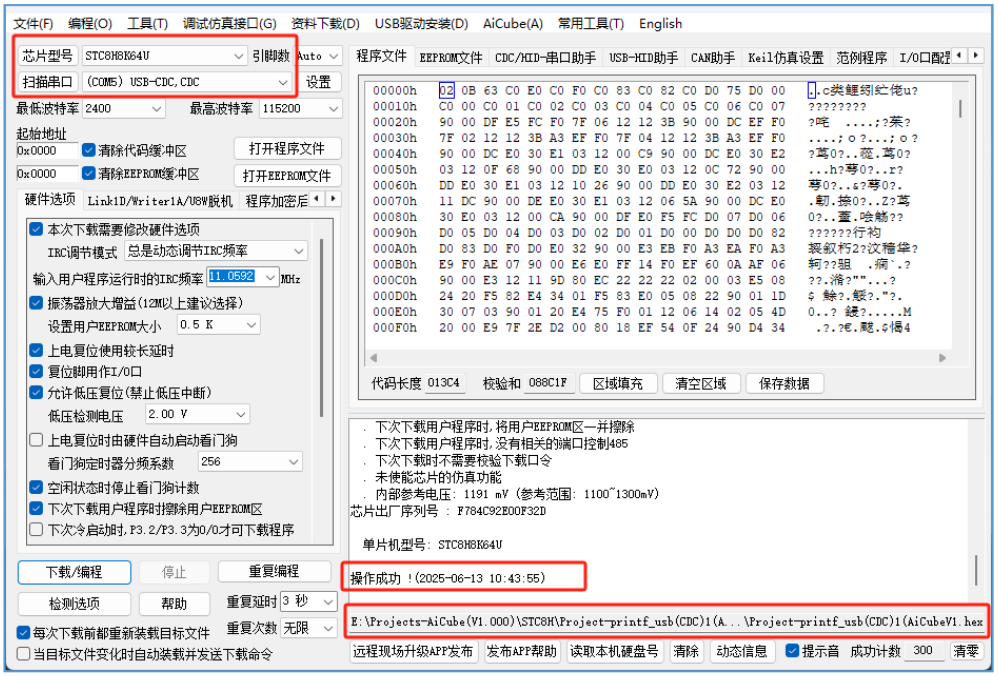
2.1.5.3　增加实验项目代码

为了实现单片机通过 USB 协议向 USB 主机（电脑）发送字符串"Hello World！"，我们现在只需要在主函数的 while 循环中加入“printf_usb("Hello World !\r\n");”代码即可，保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口分裂出 4 个窗口，对比观察）：

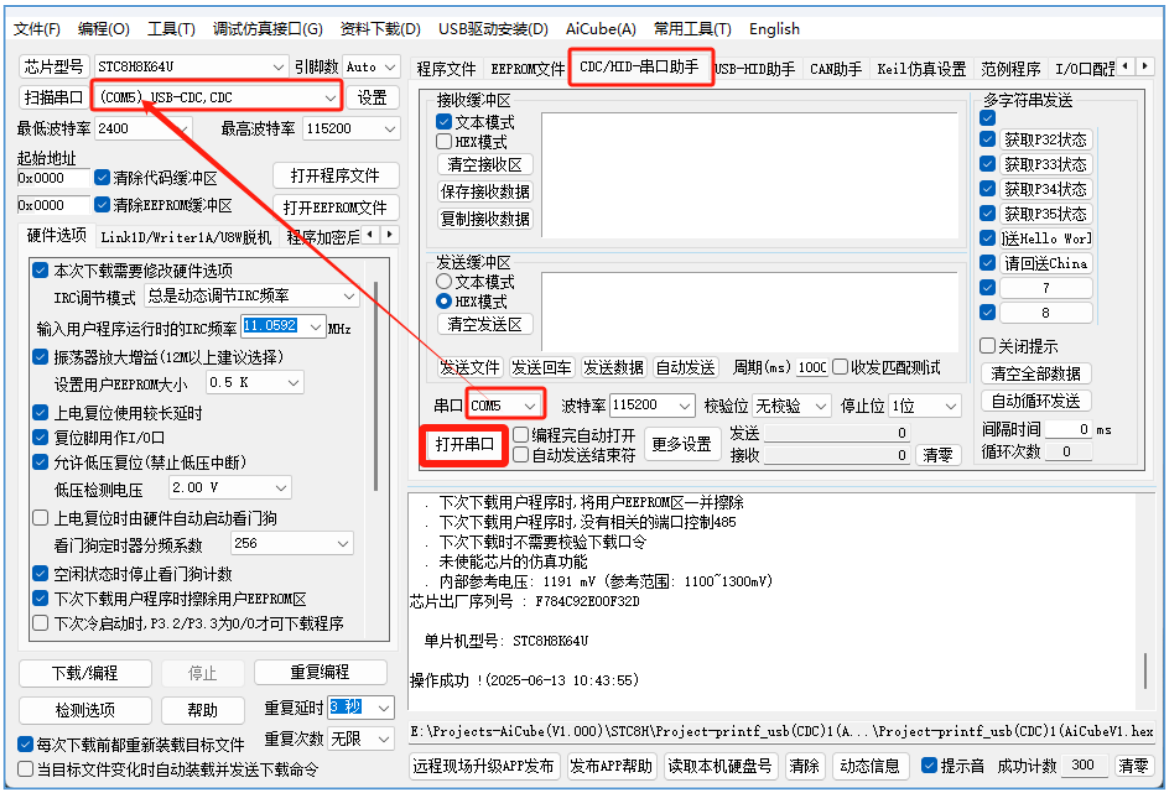


2.1.5.4　使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果

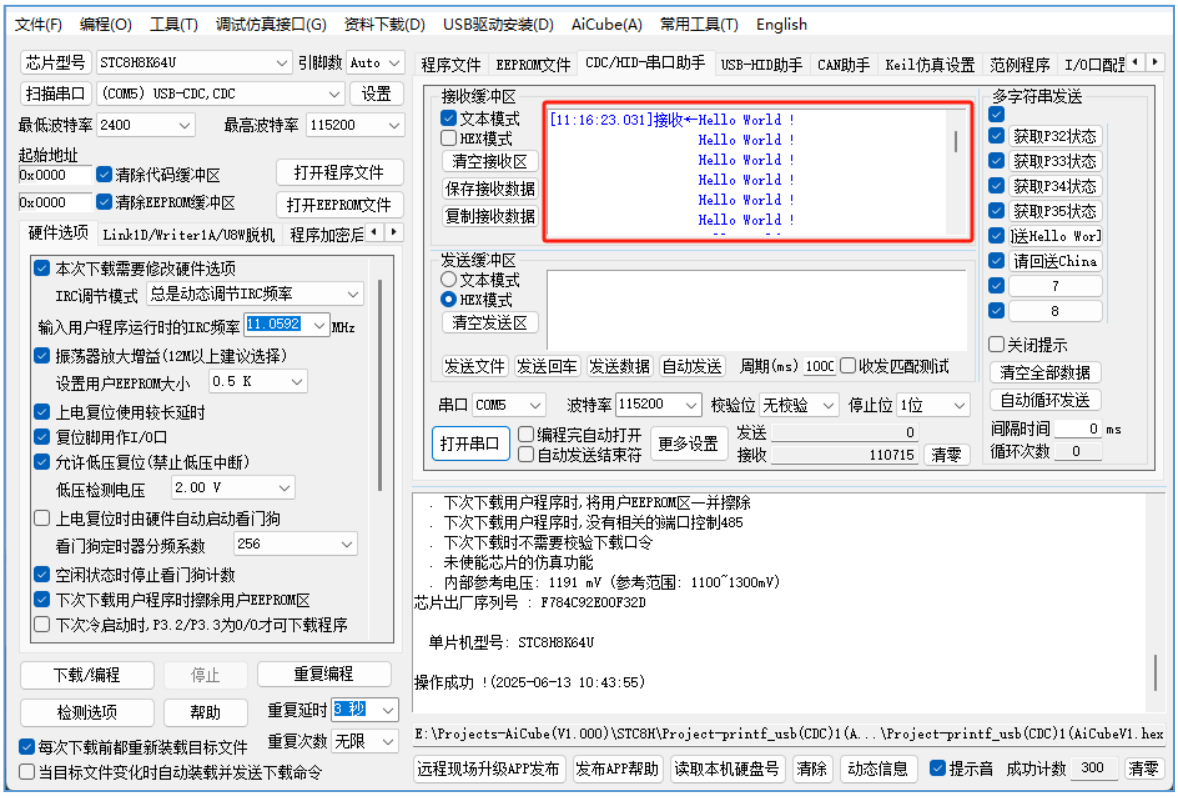
连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“Project-printf_usb(CDC)1 (AiCubeV1.00O).hex”，如下图：



我们看到此时串口显示："（COM5）USB-CDC，CDC"，打开 AiCube-ISP 系统软件的"CDC/HID-串口助手"选项卡，找到"打开串口"按钮，如下图：



选择好对应的串口号，点击"打开串口"按钮，在 CDC/HID-串口助手的"接收缓冲区"，就会不停的显示"Hello World!"，如下图：



至此，printf_usb()显示"Hello World!"实验成功！

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.1.6　USB 不停电下载，ISP 软件如何设置

在开发过程中，程序经常需要反复修改，重新编译后生成新的 HEX 文件，就需要停电/上电进入下载模式，很不方便。其实可以利用我们提供的 USB 不停电 ISP 下载功能，进行自动下载。

在有 USB 功能的实验中，都要加载 USB 库函数，并对 USB 进行初始化。由于在 USB 库函数的代码中已经定义了不停电 ISP 下载命令"@STCISP#"，只需要在下载软件的"收到用户命令后复位到 ISP 监控程序区"的选项卡中进行如下设置，即可实现【USB 不停电下载】功能。

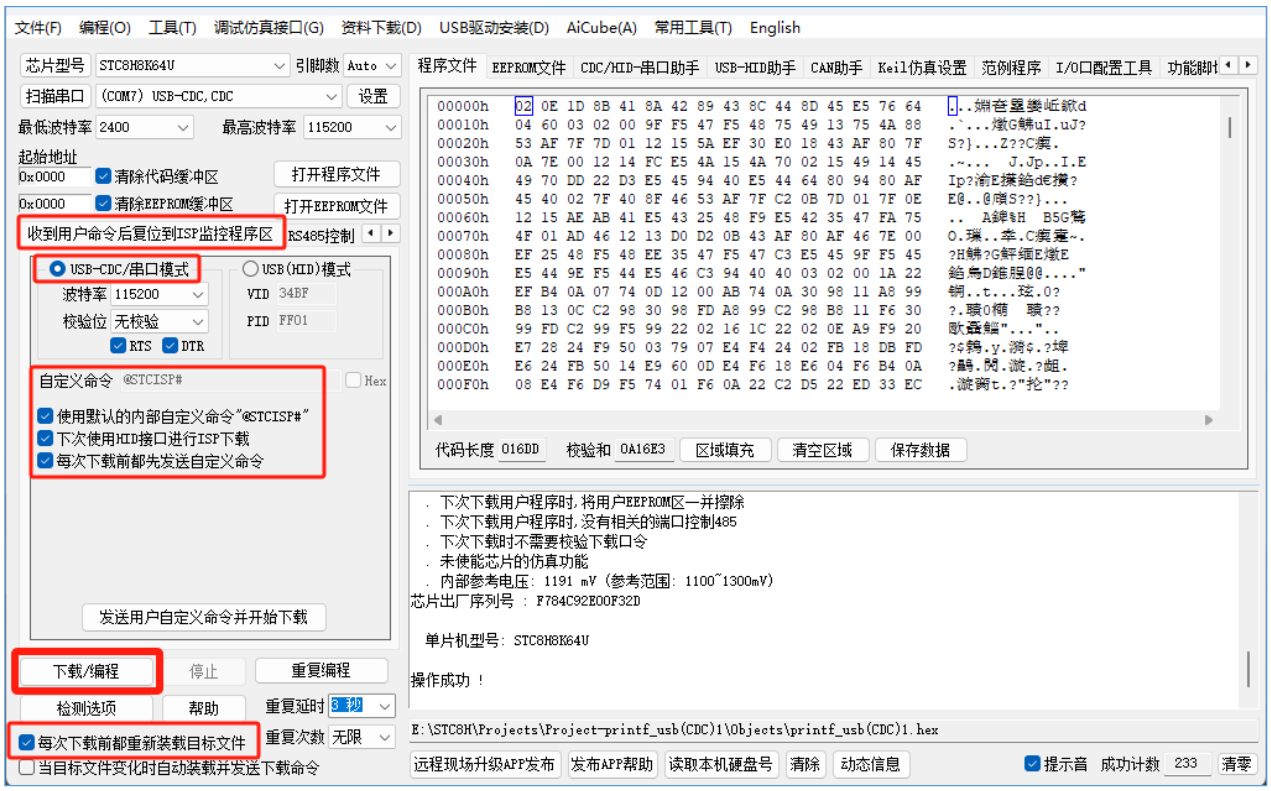
- 如使用 USB-CDC 库函数产生 USB-CDC 虚拟串口，可实现 USB-CDC 不停电自动 ISP 下载功能，并可忽略传统串口的波特率、校验位等参数的设置，下载时系统程序区还是使用的 USB-HID 通讯协议；
- 如果是传统的串口，需正确设置波特率、校验位等参数，才能实现串口的不停电自动 ISP 下载功能；
- 如果使用 USB-HID 库函数产生 USB-HID 设备，则选择"USB（HID）模式"就能实现不停电自动 ISP 下载功能，下载时系统程序区还是使用的 USB-HID 通讯协议。

ISP 软件如何设置：

2.1.6.1　手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载

如下图设置，就可以实现：

在点击"下载/编程"按钮后，ISP 软件会重新调入已打开的目标文件一次（不管目标文件有无变化），接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。

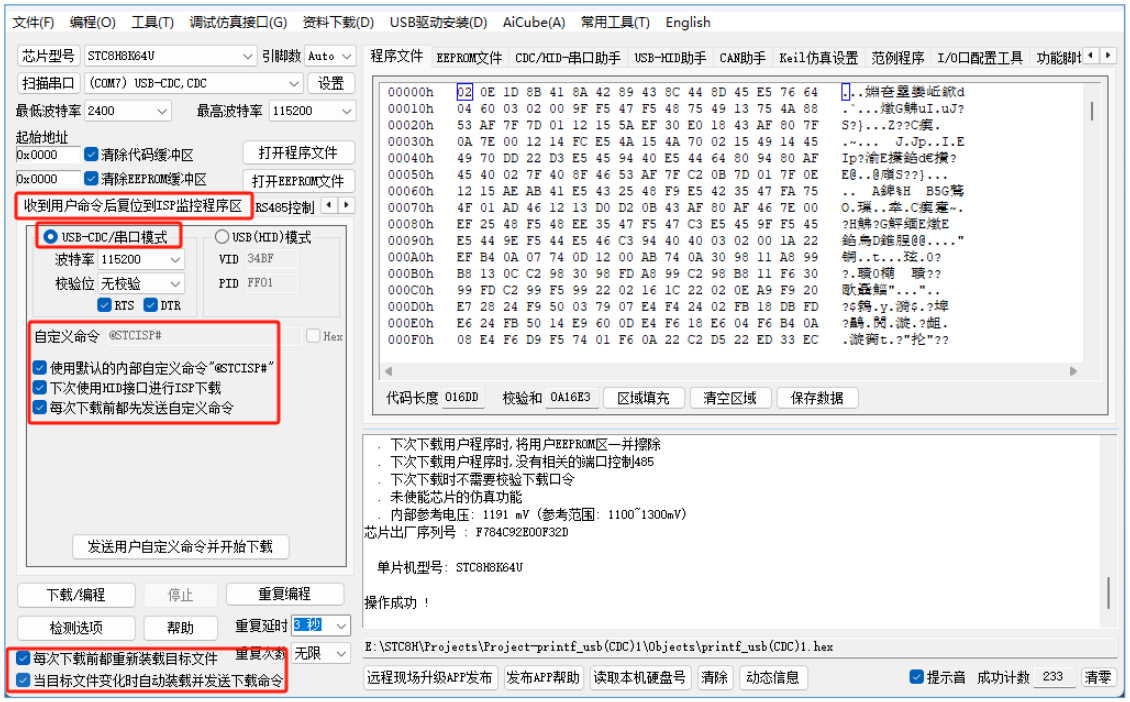


如上所示，不停电下载实验是成功的，但本设置还需要手工点击一次"下载/编程"按钮。

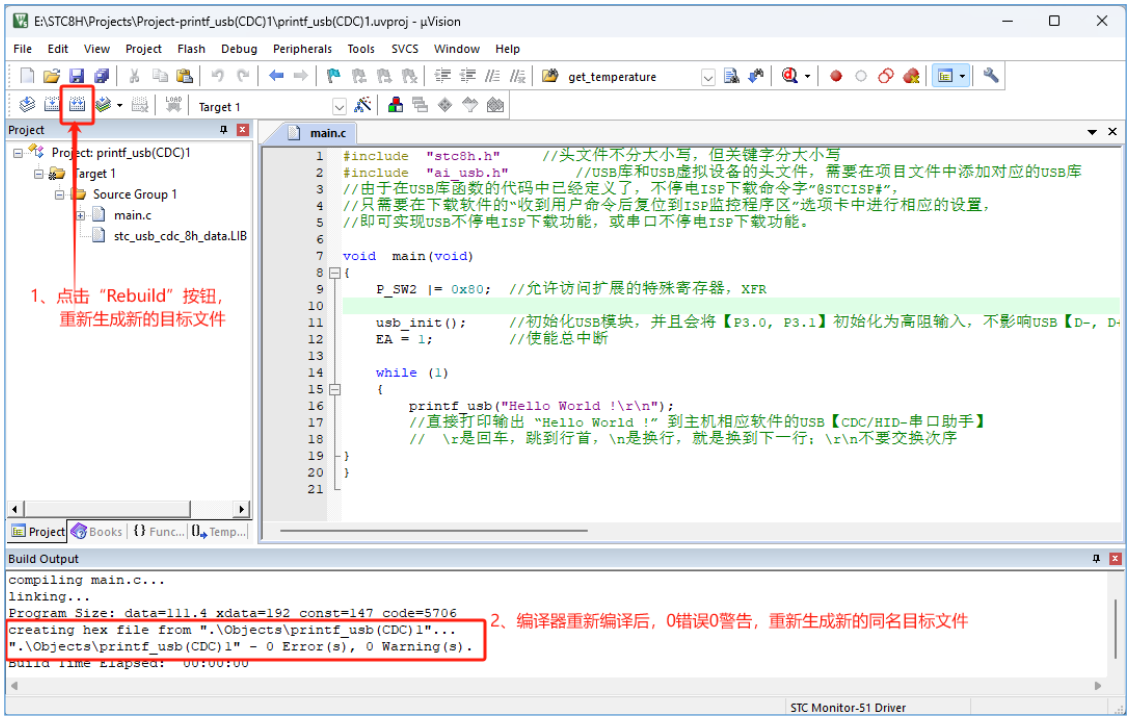
2.1.6.2　编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载

如下图设置：

下载软件会每秒检测多次目标文件被编译的时间是否已更新，如时间已更新，则 ISP 软件会自动重新调入已打开的目标文件，接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。



如上所示，不停电下载实验是成功的，但本设置的优点是不需要手工点击"下载/编程"按钮。



在编译集成环境中，点击"Rebuild"按钮，目标文件会被重新编译生成一次，ISP 软件发现目标文件被编译的时间发生变化后，ISP 软件就会执行前面描述的不停电下载动作。

2.2 2 实验二：查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他

看下面的程序，其实还是那个熟悉的标准 C 语言的 printf()，只是到了单片机 C 语言中，变成了 printf_usb()

2.2.1 printf( )，printf_usb( )实验程序代码

```
#include   "stc8h.h"          //头文件不分大小写，但关键字分大小写
#include   "ai_usb.h"    //USB 库和 USB 虚拟设备的头文件，需要在项目文件中添加对应的 USB 库
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。
void main(void)
{
    P_SW2 |= 0x80;    //允许访问扩展的特殊寄存器，XFR

    usb_init();          //初始化 USB 模块
    //初始化 USB 模块，并且会将【P3.0, P3.1】初始化为高阻输入，不影响 USB【D-, D+】通讯

    EA = 1;              //使能全局中断

    while(1)
    {
        if(bUsbOutReady)
        {    //查询方式：USB-Device 单片机查询是否有接收完成的 USB 主机送来的数据包
            //“bUsbOutReady”:主机发送给 HID/CDC 设备的数据状态标志。
            //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
            //状态位为 1 表示，主机发送给 HID/CDC 设备数据已完成，
            //设备已成功接收一包数据，等待对数据继续处理。

            if(UsbOutBuffer[0]==6)
                printf_usb("HelloWorld!\r\n");
                    //接收到的数据包的第 0 个字节是 6，打印“HelloWorld！”
            else if(UsbOutBuffer[0]==7)
                printf_usb("China!\r\n");
                    //接收到的数据包的第 0 个字节是 7 后，打印“China！”

            usb_OUT_done();  //USB 查询方式，Device 单片机通知【主机可以发下一个数据包】
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
                //如果是使用中断方式调用 USB 库函数，用户程序不需要调用“usb_OUT_done();”
                //中断方式，USB 库函数内部会自动处理。
        }
    }
}
```

以上是完整的第一个程序代码

printf_usb()函数：

printf_usb()函数和标准 C 语言的格式完全一致，详见 printf()函数章节

进入 printf_usb()函数后，首先会判断 USB 主机是否已经对 USB 设备配置完成，

如果没有配置完成，printf_usb()函数会直接返回退出。

如果已经配置完成，还会继续判断如有发送前一笔数据，则判断是否发送完成，

如果没有发送完成，会继续等待，直到前一笔数据发送完成，

才会发送新的 printf_usb()需发送的数据后再返回退出。

但如等待超过 50ms@24MHz，会直接退出。

如果主时钟不是 24MHz，20MHz 等待时间会变长，40MHz 等待时间会变短。

如果已经配置完成且前一笔数据已经发送完成，

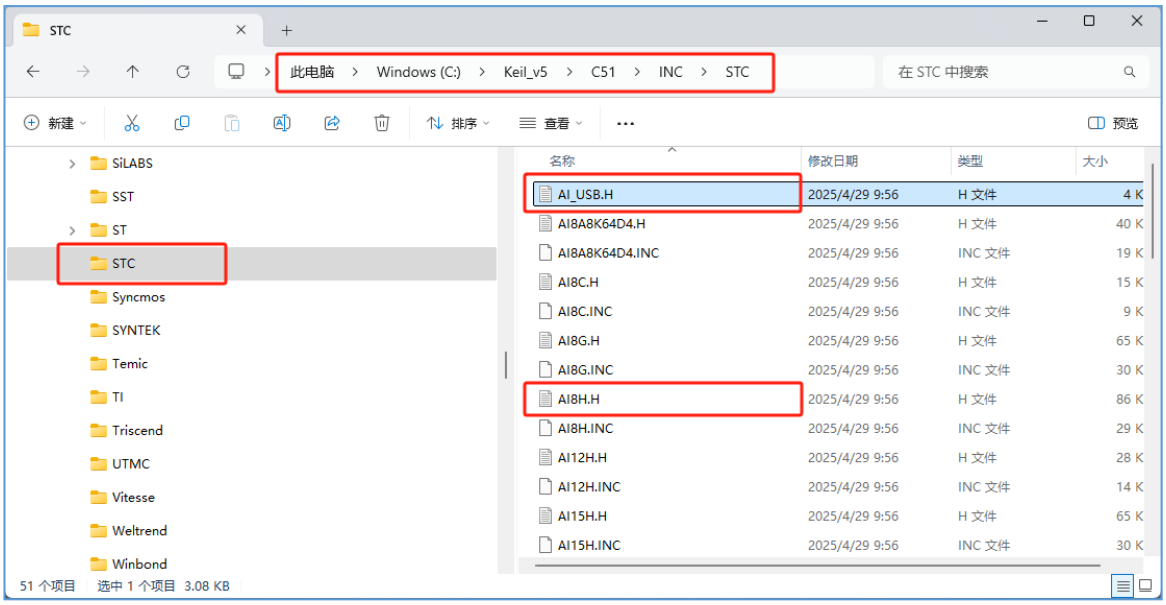
则会直接发送新的 printf_usb()需发送的数据并返回退出。

2.2.2　准备工作，如已准备，跳过此节

1、添加常用头文件 ai_usb.h

```
#include "ai_usb.h"                    //使用 usb 所需要包含的头文件。
```

这个头文件在使用 AiCube-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



2、添加 usb_cdc 库函数文件

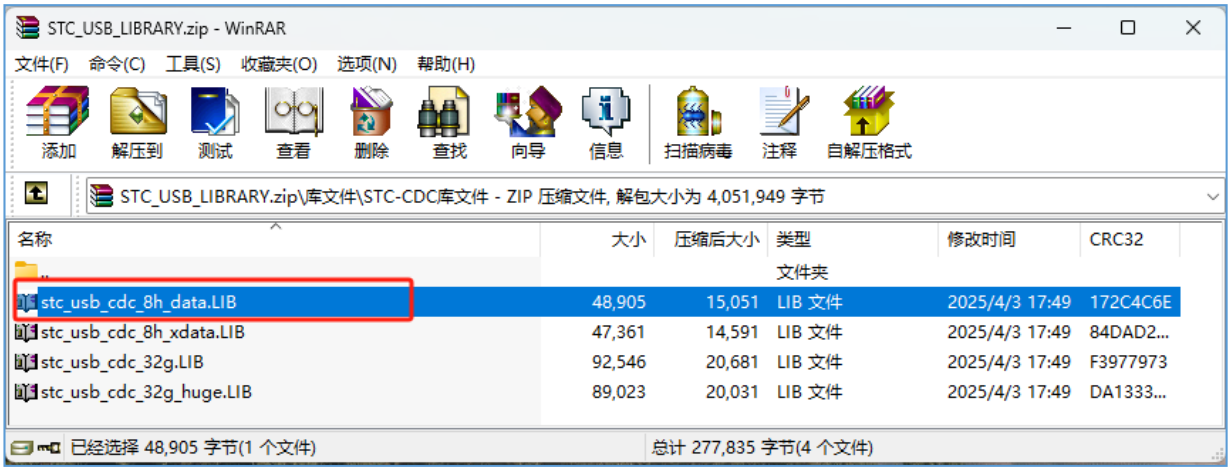
因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：

https://www.stcai.com/khs 的 USB 库文件中进行下载

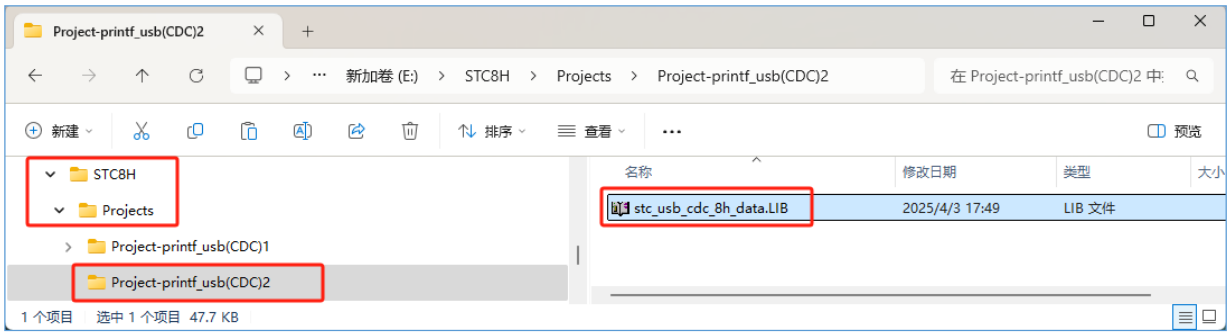


下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_8h_data.LIB

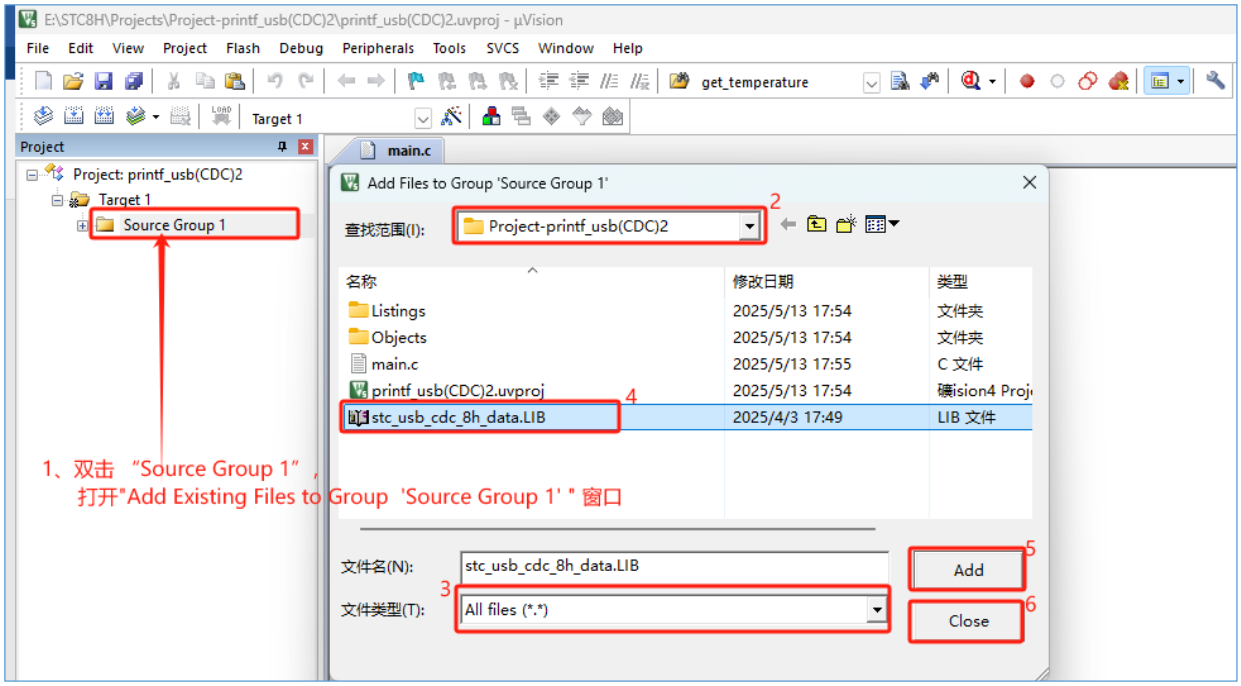
【8 位的 8051 的 USB 库 可以用 BL51 连接器
BL51： 不支持 REMOVEUNUSED；BL51 支持 扩展中断号工具；
如果 不用 REMOVEUNUSED，可以 使用 BL51 ！
LX51： 支持 REMOVEUNUSED；LX51 不支持 扩展中断号工具；
如果 不用 扩展中断号工具，可以 使用 LX51 ！
详情请参考本公司技术论坛链接:8 位的 8051 的 USB 库可以用 BL51;BL51 不支持 REMOVEUNUSED;
https://www.stcaimcu.com/thread-17105-1-1.html
(出处: 国芯人工智能技术交流网站)】



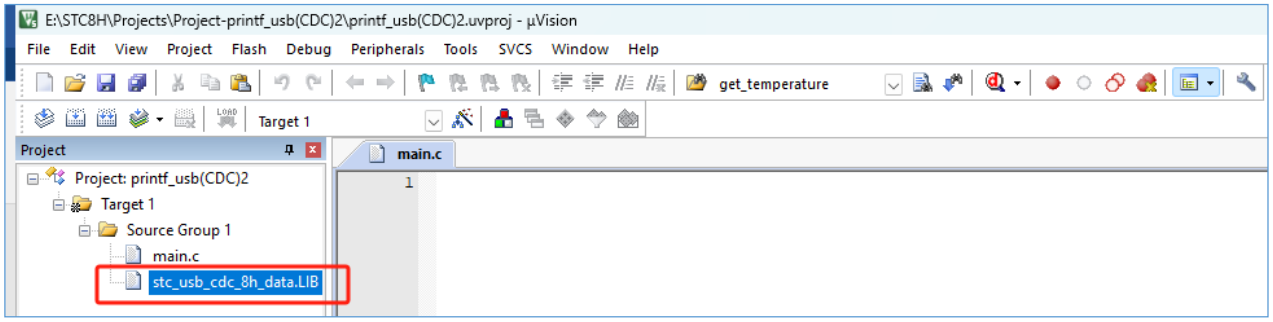
将库函数“stc_usb_cdc_8h_data.LIB”复制到创建工程所在的文件夹 E:\STC8\Projects\Project-printf_usb(CDC)2 中：



将库函数 stc_usb_cdc_8h_data.LIB 添加进入此工程项目中
在 Keil 中，双击 Source Group，在选择“Add Existing Files to Group 'Source Group 1”菜单项，打开“Add Files to Group 'Source Group 1”窗口中，将库函数 stc_usb_cdc_8h_data.LIB 添加进入此工程，就可以使用 USB-CDC 功能。



这样，Keil 界面变化如下图：



【注意】:

USB-HID 通讯不需要安装驱动，手机及 Windows 各种系统都自带 HID 驱动。

USB-CDC 通讯，手机不需要安装驱动，手机的各种系统已自带 USB-CDC 驱动。

USB-CDC 通讯，Win10（1903）及以后版本不需要安装 USB-CDC 驱动，较早的 Windows 版本才需要手动点击 ISP 软件中的【USB 驱动安装】，只有较早的 Windows 系统才需要安装 USB-CDC 驱动程序。

USB-CDC 虚拟串口类似传统串口的形式，但不需要进行波特率的设置，通讯时是以 12Mbps 的速度进行通讯，USB-CDC 虚拟串口转串口才需要设置转传统硬件串口需要的波特率、奇偶校验位等参数。

录入实验程序代码，保存，编译。

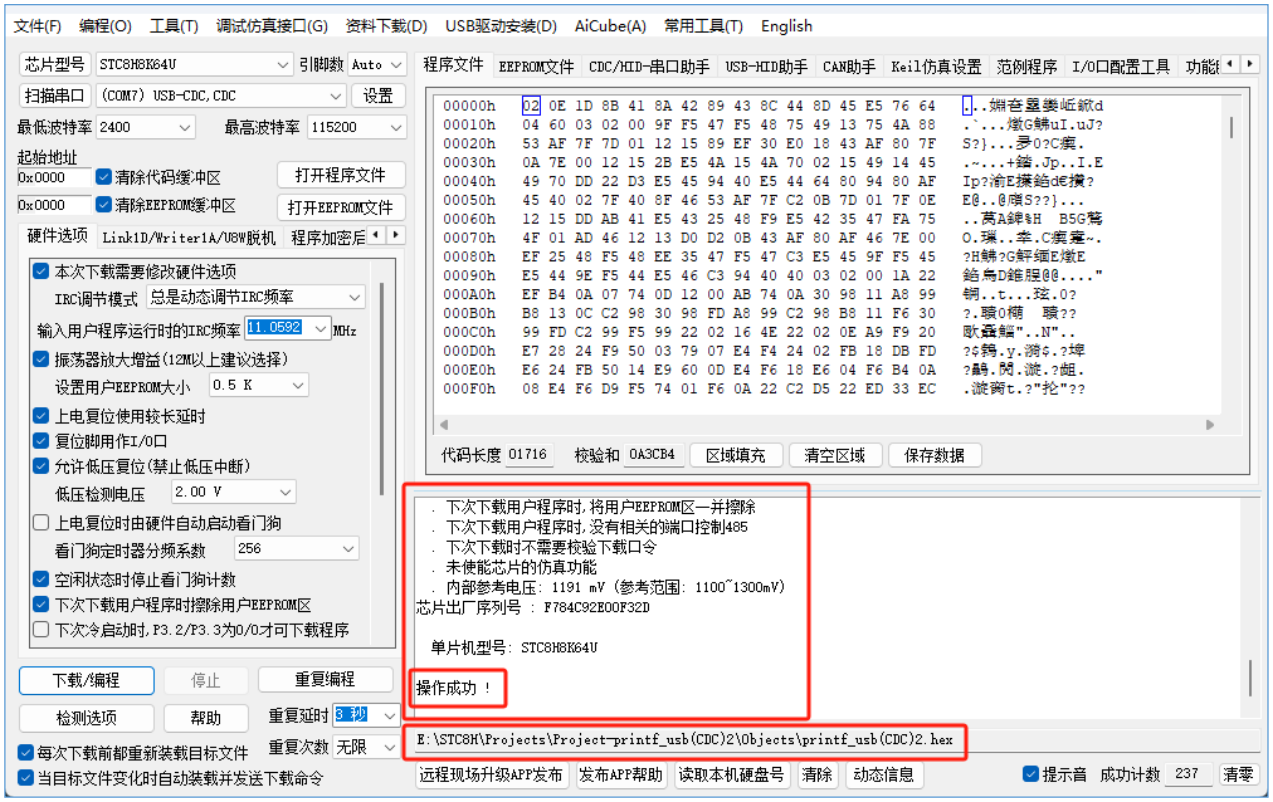
2.2.3　下载用户程序到 STC8H8K64U 实验箱

在录入代码，保存，编译无误后。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤“打开程序文件”printf_usb(CDC)2.hex（“打开程序文件”是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中）并“下载/编程”。

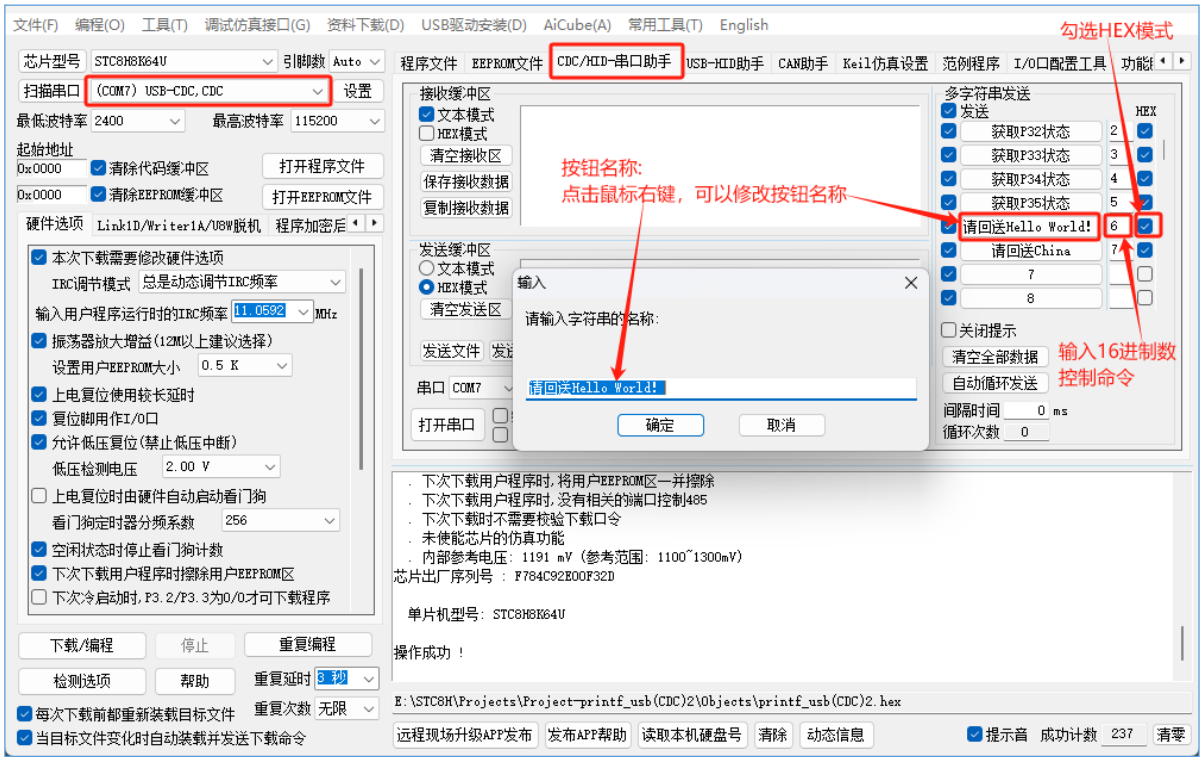
【AiCube-ISP 软件的安装】，请点击查阅本文档 1.1 安装 AiCube-ISP 下载/编程/烧录 工具，含强大的辅助开发工具

【STC8H8K64U 实验箱实物图、与电脑的连接、用户程序下载流程】，请点击查阅本文档的 1.2 STC8H8K64U 实验箱实物图及程序下载

AiCube-ISP 软件界面出现：操作成功！
恭喜你！



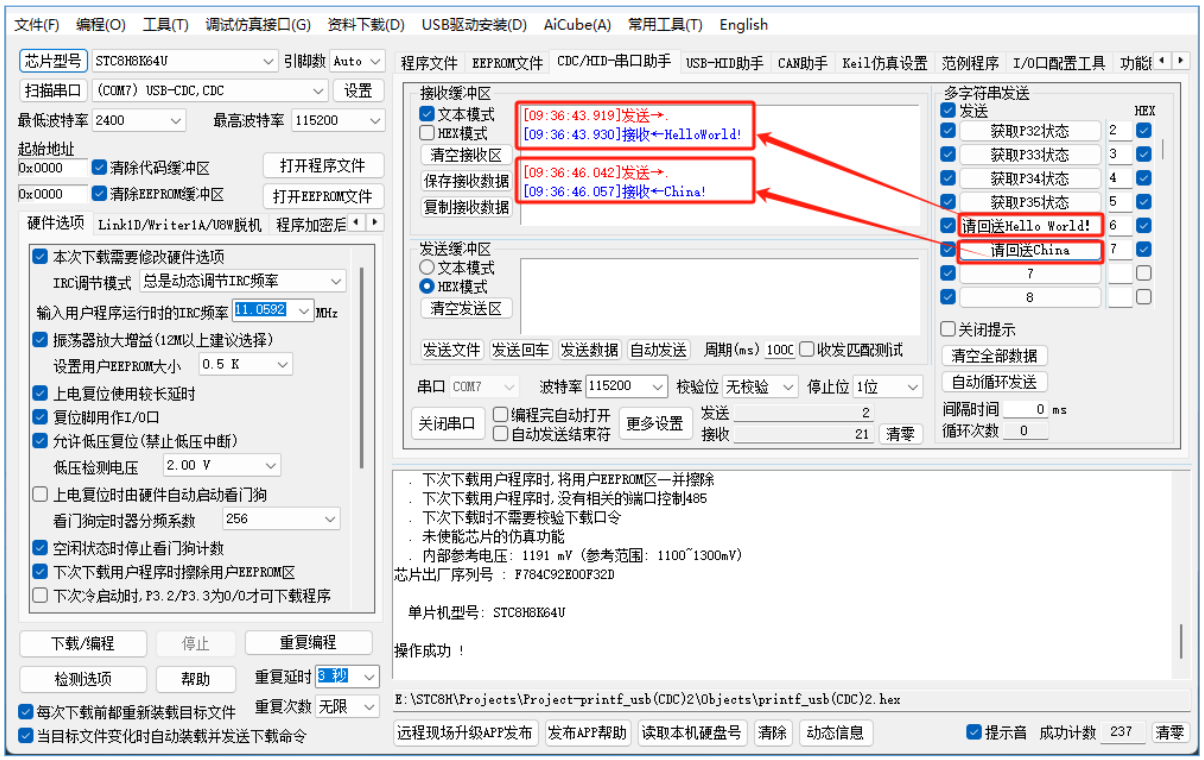
我们看到此时串口显示：“（COM7）USB-CDC，CDC”，选择 AiCube-ISP 系统软件的“CDC/HID-串口助手”选项卡，在“多字符串发送”窗口，设置好发送控制命令，以便观察实验现象，如下图：



一切准备就绪！

2.2.4　观察实验现象

点击“请回送 Hello World!”按钮，单片机收到 16 进制数 6 后，回送“Hello World!”给 USB 主机显示。
点击“请回送 China!”按钮，单片机收到 16 进制数 7 后，回送“China!”给 USB 主机显示。
如下图：



完成实验验证！

2.2.5　使用【AiCube】工具创建“查询方式，查询到电脑命令后，printf_usb("Hello World !\r\n")或其他”

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit**、**Ai8051U-8Bit**、**STC32G12K128**、**STC8H8K64U**、**STC15F2K60S2**、**STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

2.2.5.1　安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

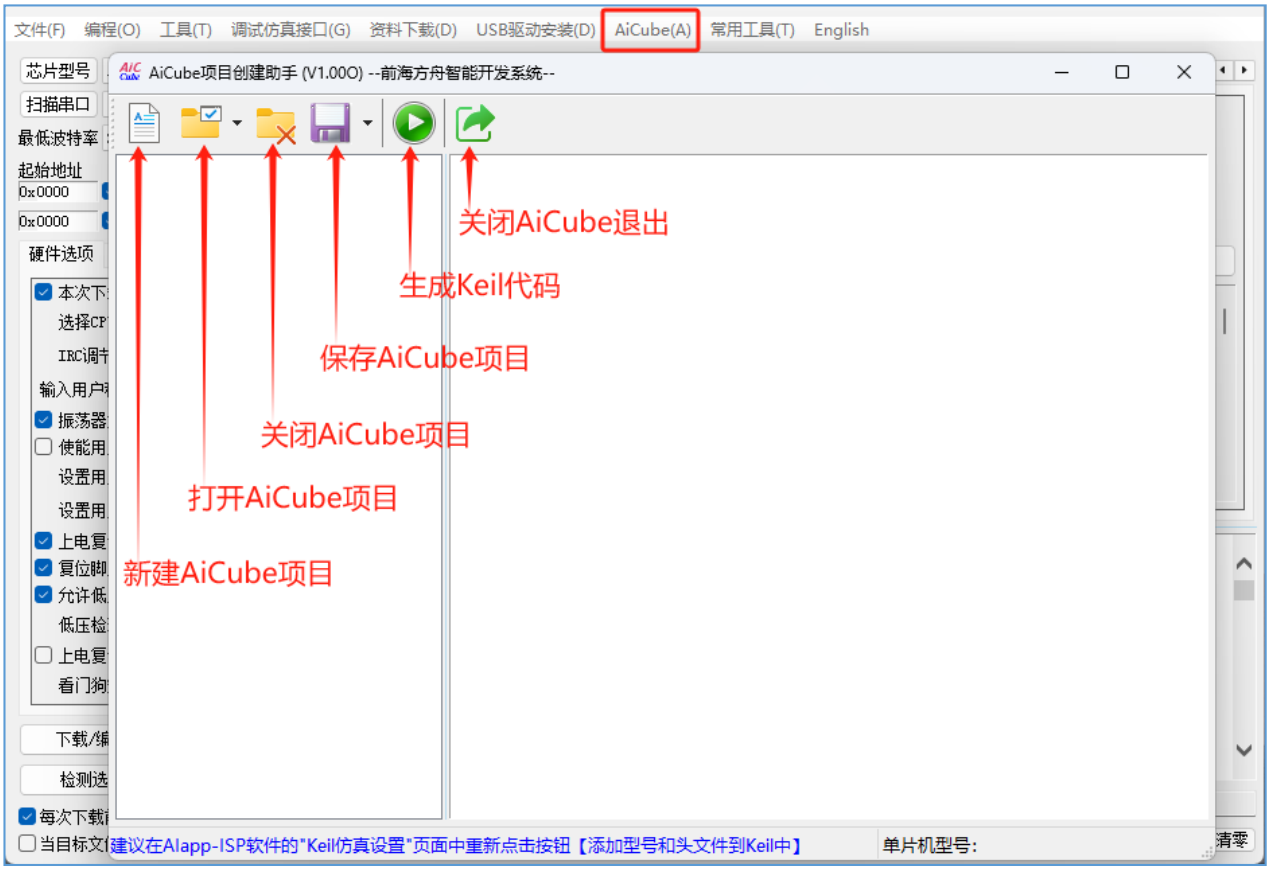
【AiCube-ISP 软件的安装】，请点击查阅本文档 1.1 安装 AiCube-ISP 下载/编程/烧录 工具，含强大的辅助开发工具

2.2.5.2　使用【AiCube】创建项目

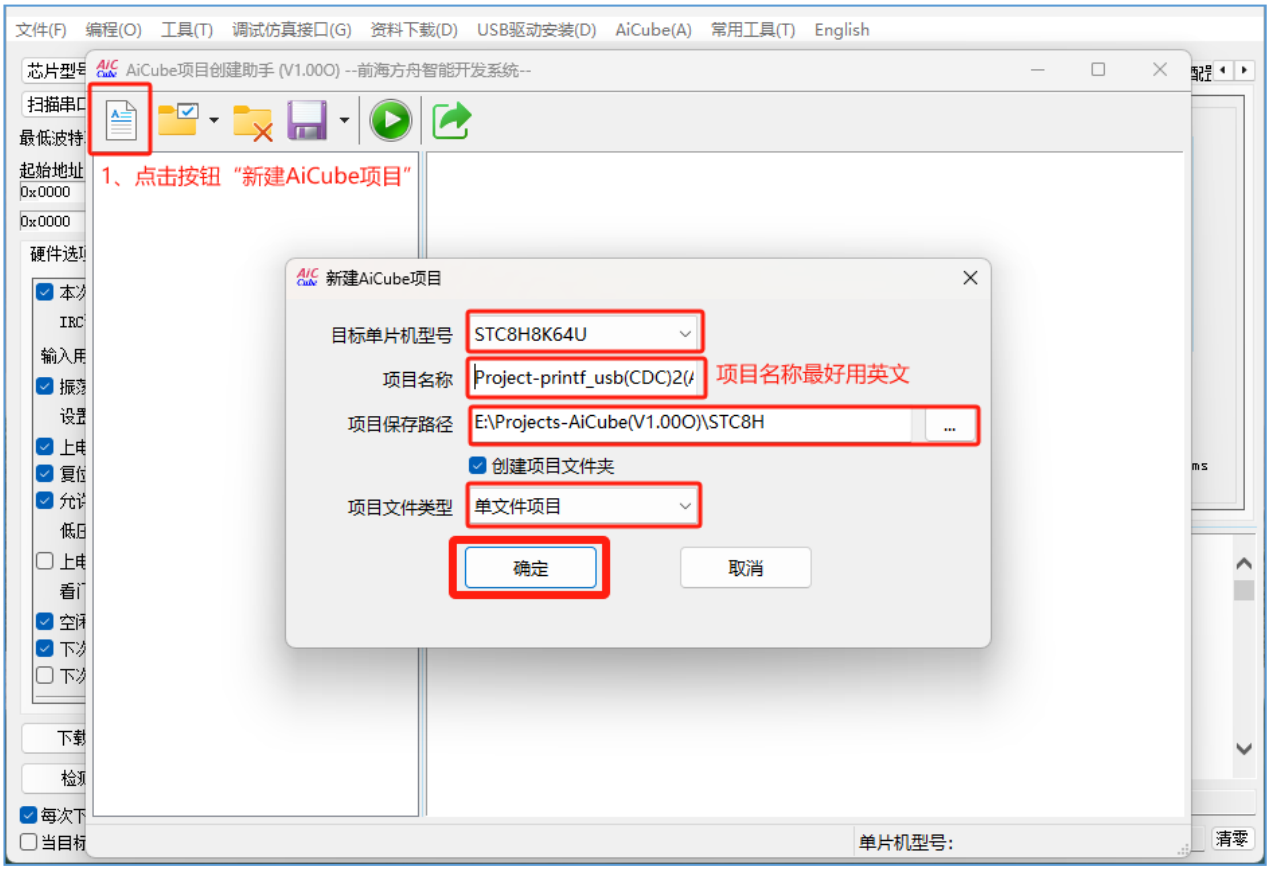
双击图标，打开 AiCube-ISP 软件界面，即可直接使用，如下图，在菜单栏上找到【AiCube(A)】菜单：



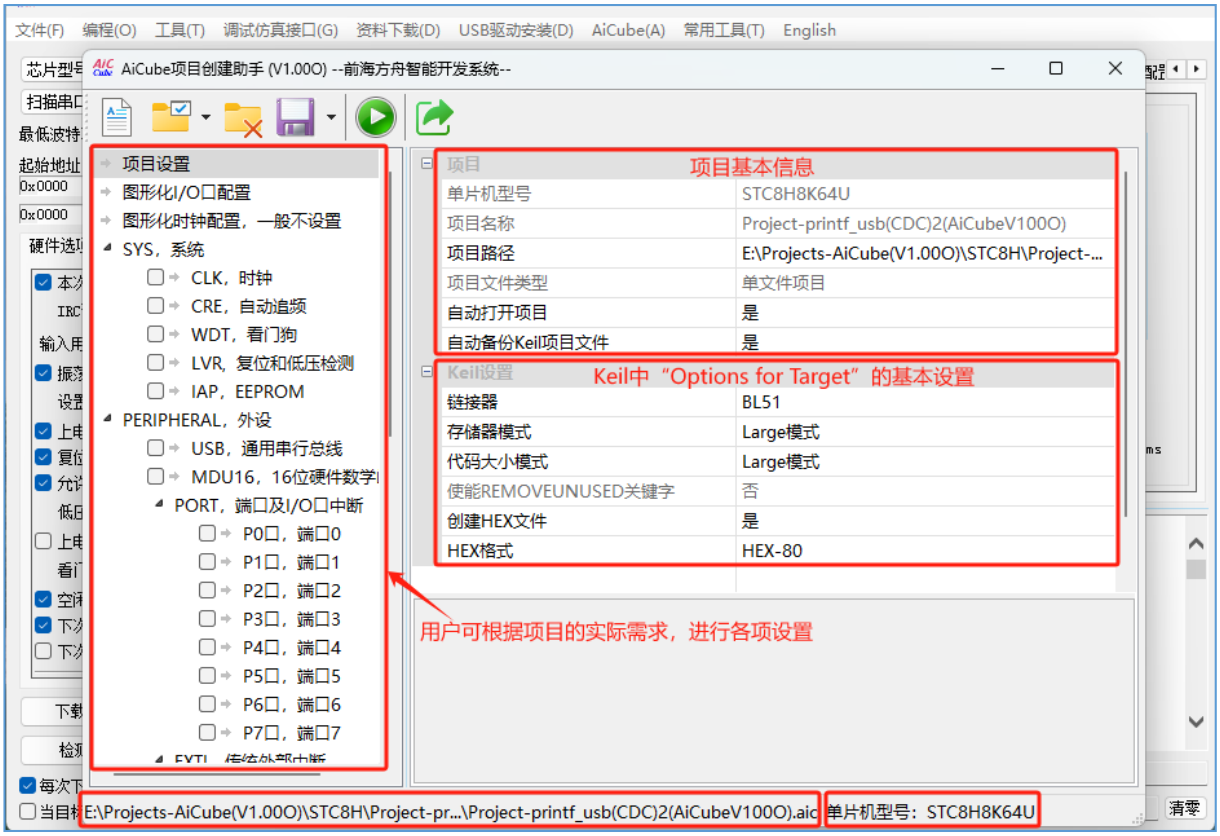
➲ **点击【AiCube(A)】**，弹出如下窗口：



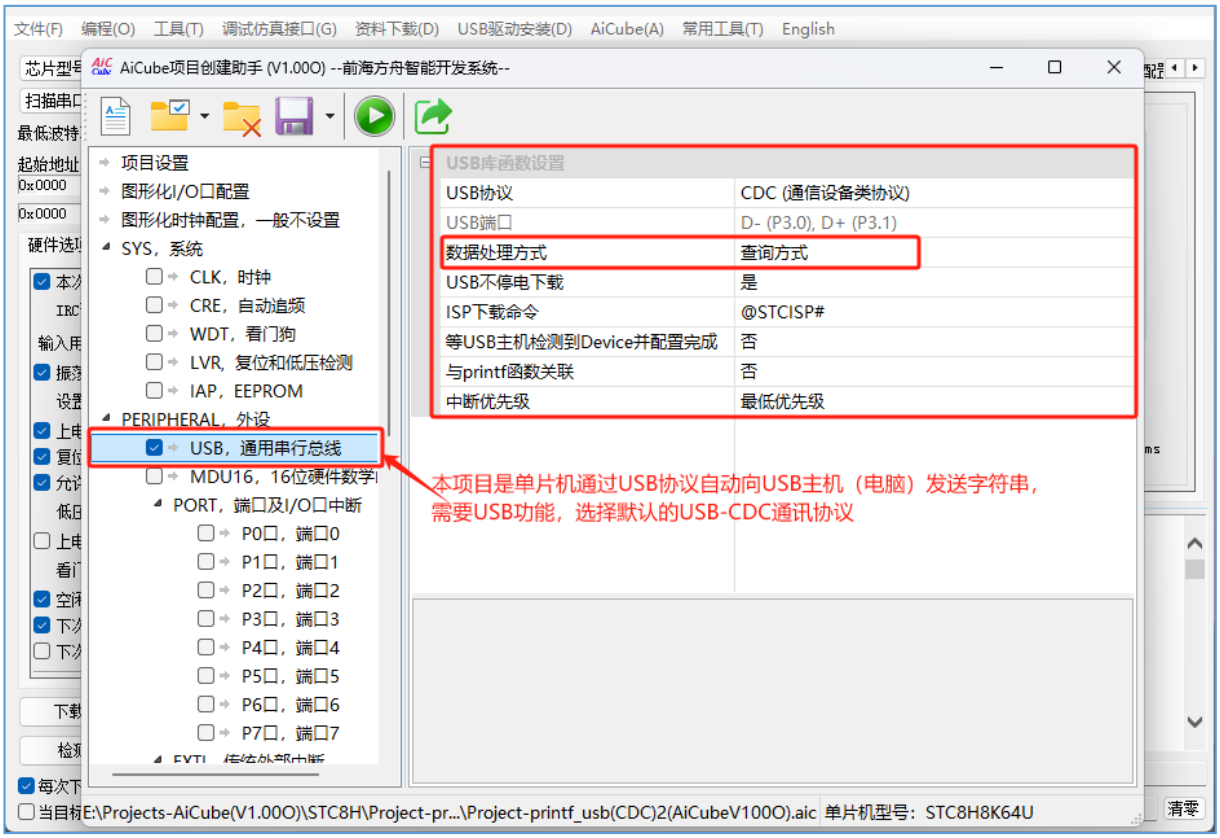
➲ **点击“新建 AiCube 项目”按钮**，出现如下界面，选择并填写项目基本信息，建议项目名称用英文！



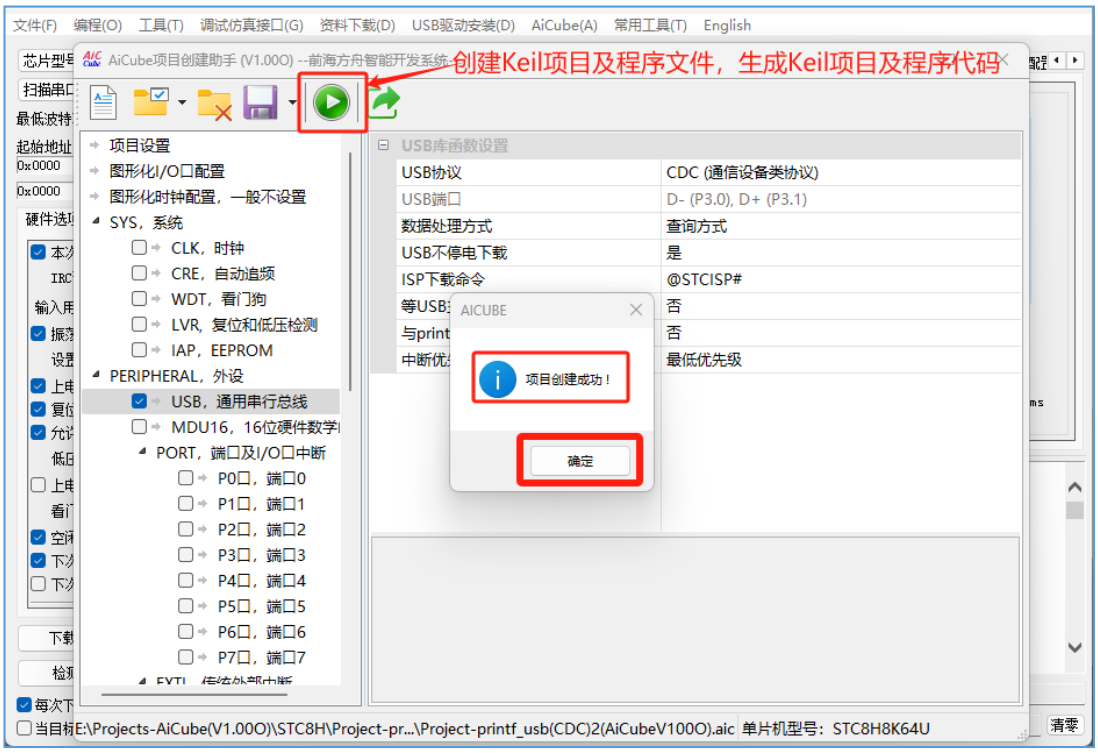
➲ 点击“确定”，出现如下界面：



因本项目用到 USB-CDC 功能，就需要加载 USB 库函数、USB-CDC 协议，要对 USB 初始化等。本项目是用到 USB-CDC 的查询方式，【AiCube】助手中 USB-CDC 默认的数据处理方式就是“查询方式”，根据本项目的实际需求，只需要勾选 USB 外设即可，如下图：



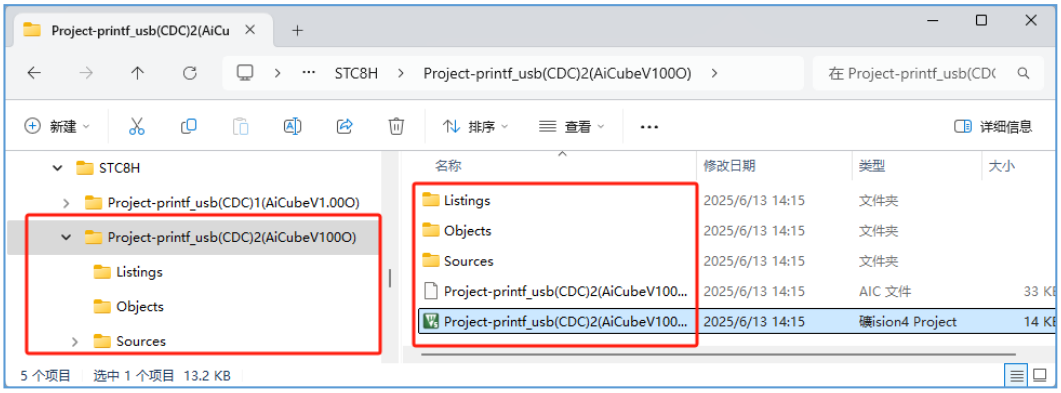
➲ 点击“创建 Keil 项目及程序文件”按钮，【AiCube】提示：“项目创建成功！”，如下图：



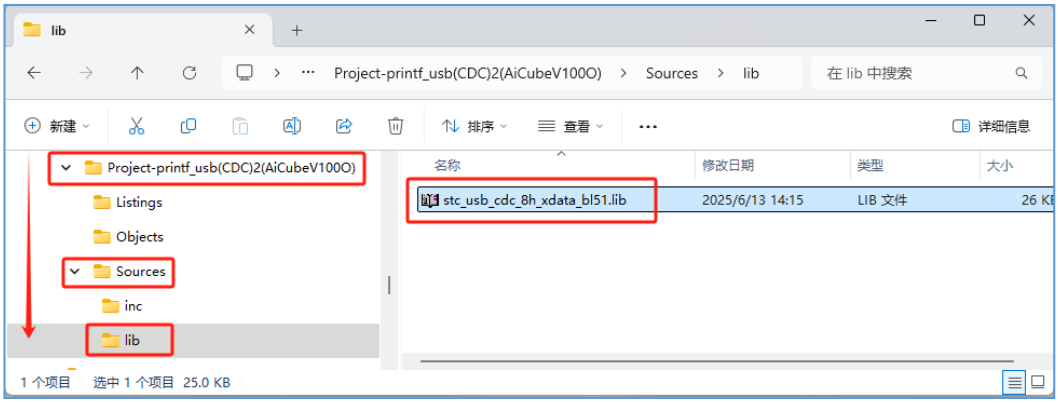
点击“确定”，立即打开 Keil 软件，进入该项目的 Keil 集成开发环境界面，即【AiCube】成功帮我们创建了项目及目录“Project-printf_usb(CDC)2(AiCubeV100O)”。至此，可以关闭【AiCube】工具。

➲ 找到并打开刚刚创建的项目目录

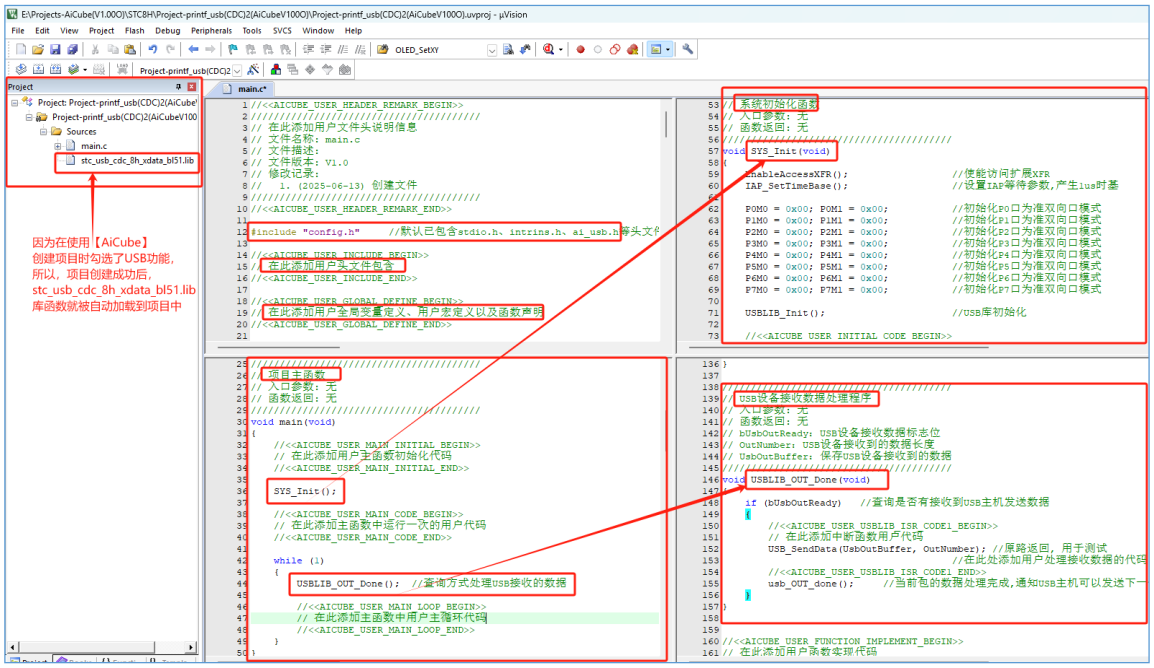
发现通过上述的操作过程,【AiCube】工具已经帮我们建立了项目目录及 Project-printf_usb(CDC)2(AiCube V100O).uvproj 文件，如下图：



同时打开 Sources>lib 子目录，发现，【AiCube】将相应的 USB 库函数加载进项目目录：



➲ 观察已打开的 Keil 项目文件：Project-printf_usb(CDC)2(AiCubeV100O).uvproj，如下图：



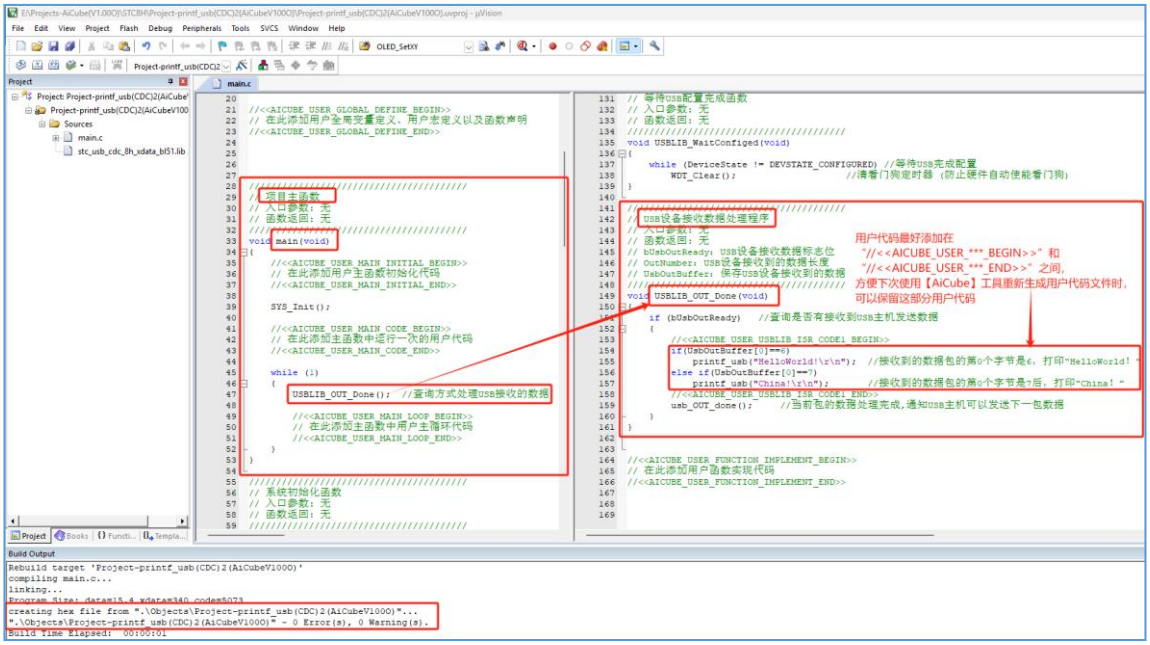
根据在【AiCube】中的配置，从上图中我们看到【AiCube】创建的本实验项目中不仅有了 main.c，还已经自动加载了 USB 库函数（stc_usb_cdc_8h_xdata_bl51.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、微秒及毫秒延时函数、USB 初始化及相关的配置及处理函数。

2.2.5.3　　增加实验项目代码

为了实现单片机通过 USB 协议接收 USB 主机（电脑）发送的命令后，再根据不同的命令回传不同的字符串到主机。我们现在只需要在“USB 设备接收数据处理程序”子函数的“查询是否有接收到 USB 主机发送数据”的 if 判断中加入如下代码即可：

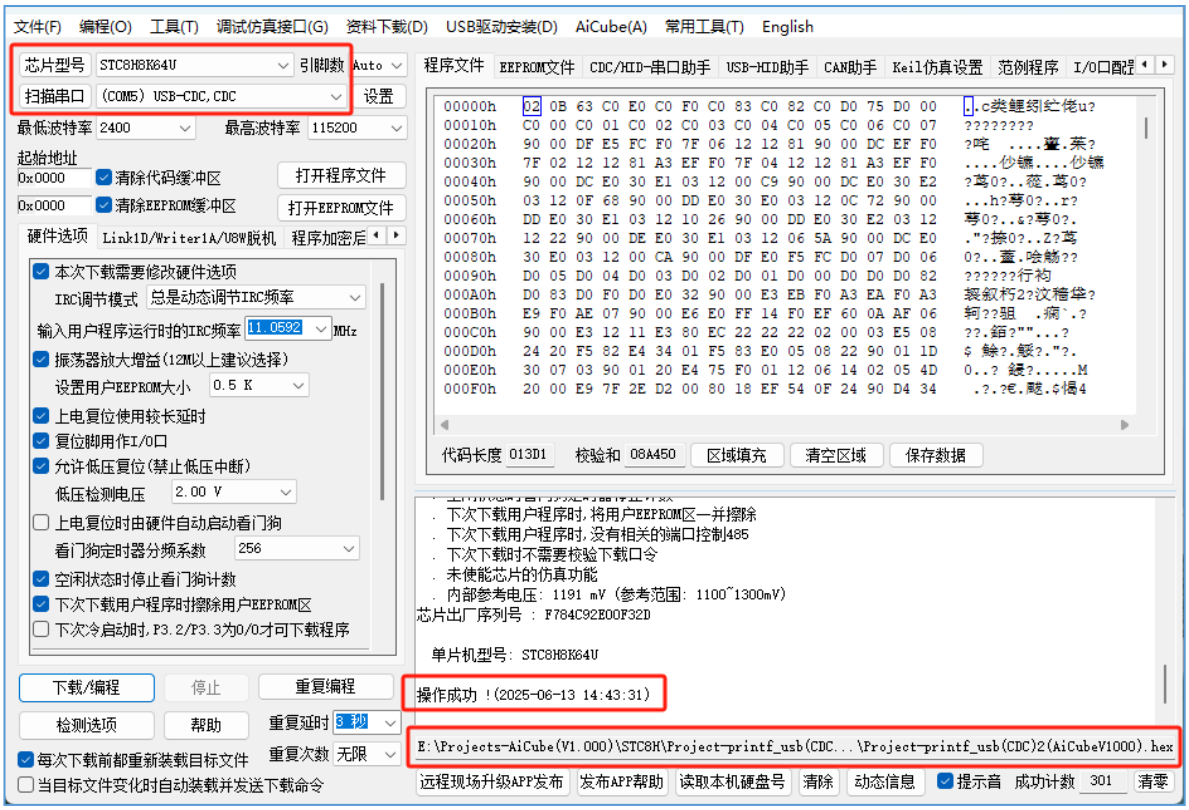
```
if(UsbOutBuffer[0]==6)
    printf_usb("HelloWorld!\r\n");      //接收到的数据包的第 0 个字节是 6，打印“HelloWorld！”
else if(UsbOutBuffer[0]==7)
    printf_usb("China!\r\n");           //接收到的数据包的第 0 个字节是 7 后，打印“China！”
```

保存，编译，如下图（使用菜单“Window”→“Split”将原窗口分裂出两个窗口，对比上图观察）：

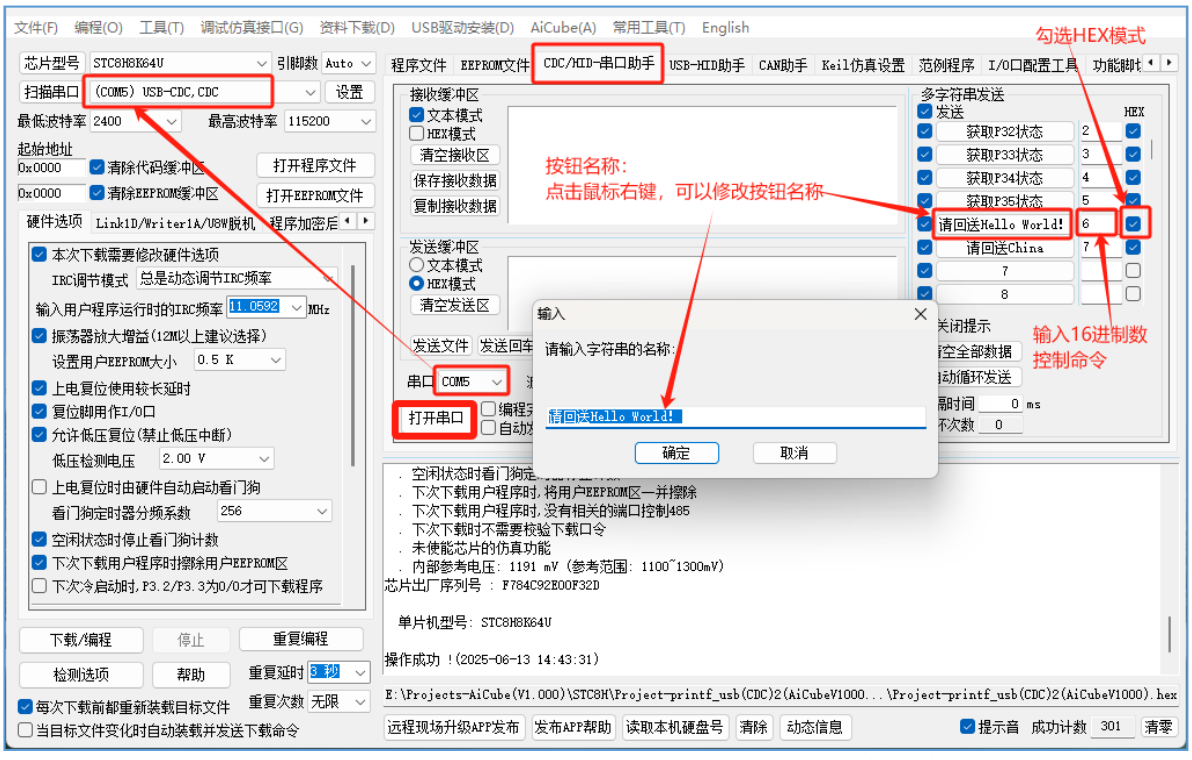


2.2.5.4　使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果

连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“Project-printf_usb (CDC)2(AiCubeV100O).hex”，如下图：



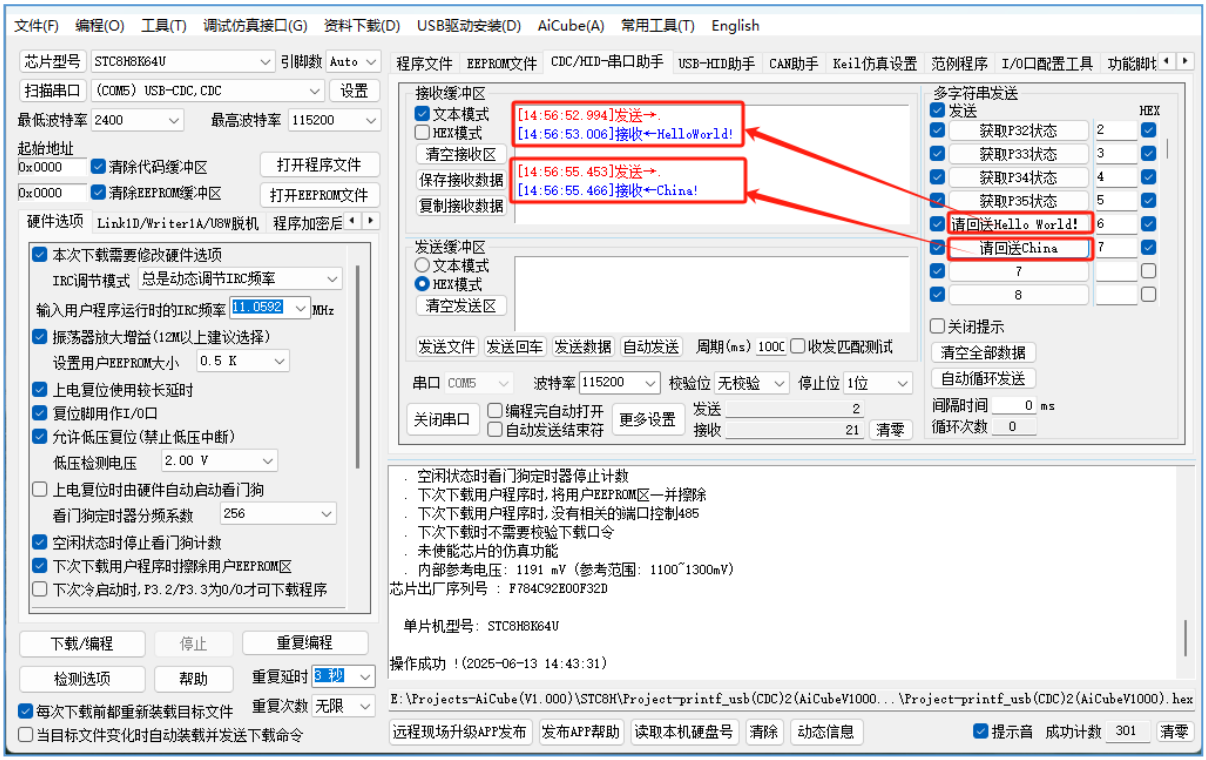
在 AiCube-ISP 系统软件界面，我们看到，此时串口显示“（COM5）USB-CDC，CDC”。打开“CDC/HID-串口助手”选项卡，在“多字符串发送”栏，设置好发送控制命令，以便观察实验效果，如下图：



在 CDC/HID-串口助手选项卡，找到并点击“打开串口”按钮，一切准备就绪！

点击“请回送 Hello World!”按钮，单片机收到 16 进制数 6 后，回送“Hello World!”给 USB 主机显示。
点击“请回送 China!”按钮，单片机收到 16 进制数 7 后，回送“China!”给 USB 主机显示。

在 CDC/HID-串口助手的“接收缓冲区”，如下图：



完成实验验证！

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.2.6　USB 不停电下载，ISP 软件如何设置

在开发过程中，程序经常需要反复修改，重新编译后生成新的 HEX 文件，就需要停电/上电进入下载模式，很不方便。其实可以利用我们提供的 USB 不停电 ISP 下载功能，进行自动下载。

在有 USB 功能的实验中，都要加载 USB 库函数，并对 USB 进行初始化。由于在 USB 库函数的代码中已经定义了不停电 ISP 下载命令"@STCISP#"，只需要在下载软件的"收到用户命令后复位到 ISP 监控程序区"的选项卡中进行如下设置，即可实现【USB 不停电下载】功能。

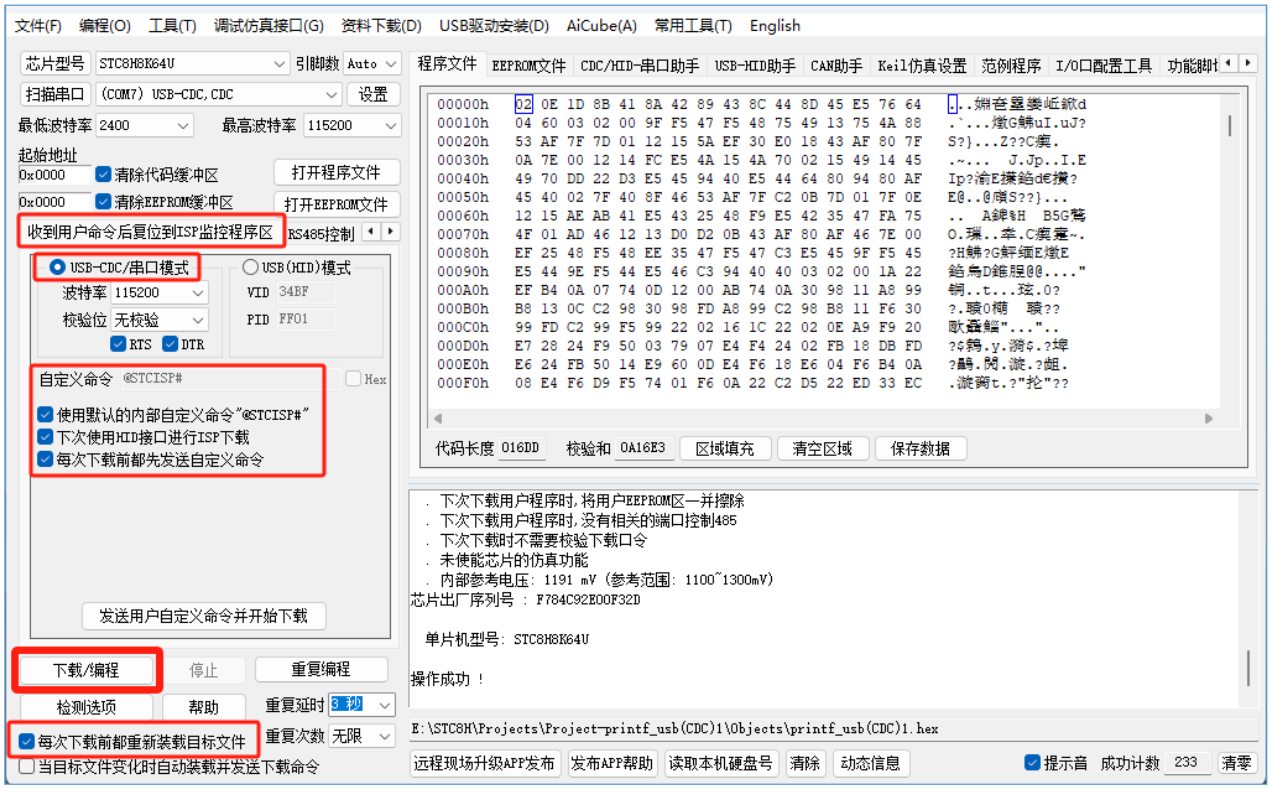
- 如使用 USB-CDC 库函数产生 USB-CDC 虚拟串口，可实现 USB-CDC 不停电自动 ISP 下载功能，并可忽略传统串口的波特率、校验位等参数的设置，下载时系统程序区还是使用的 USB-HID 通讯协议；
- 如果是传统的串口，需正确设置波特率、校验位等参数，才能实现串口的不停电自动 ISP 下载功能；
- 如果使用 USB-HID 库函数产生 USB-HID 设备，则选择"USB（HID）模式"就能实现不停电自动 ISP 下载功能，下载时系统程序区还是使用的 USB-HID 通讯协议。

ISP 软件如何设置：

2.2.6.1　手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载

如下图设置，就可以实现：

在点击"下载/编程"按钮后，ISP 软件会重新调入已打开的目标文件一次（不管目标文件有无变化），接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。

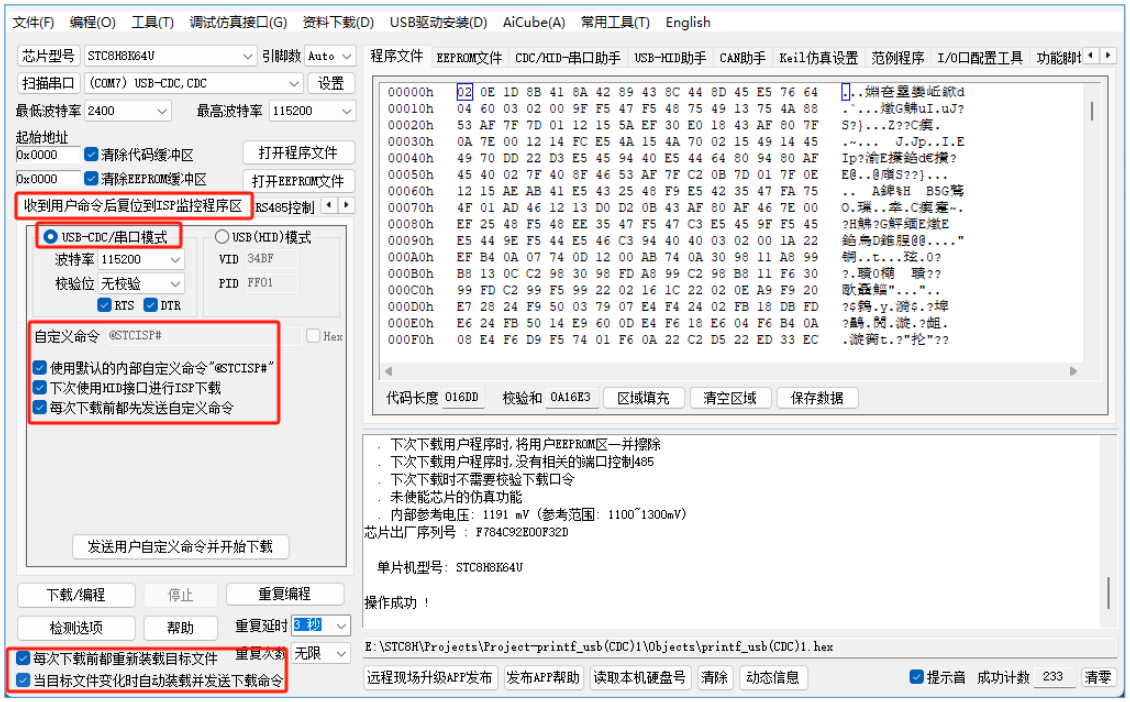


如上所示，不停电下载实验是成功的，但本设置还需要手工点击一次"下载/编程"按钮。

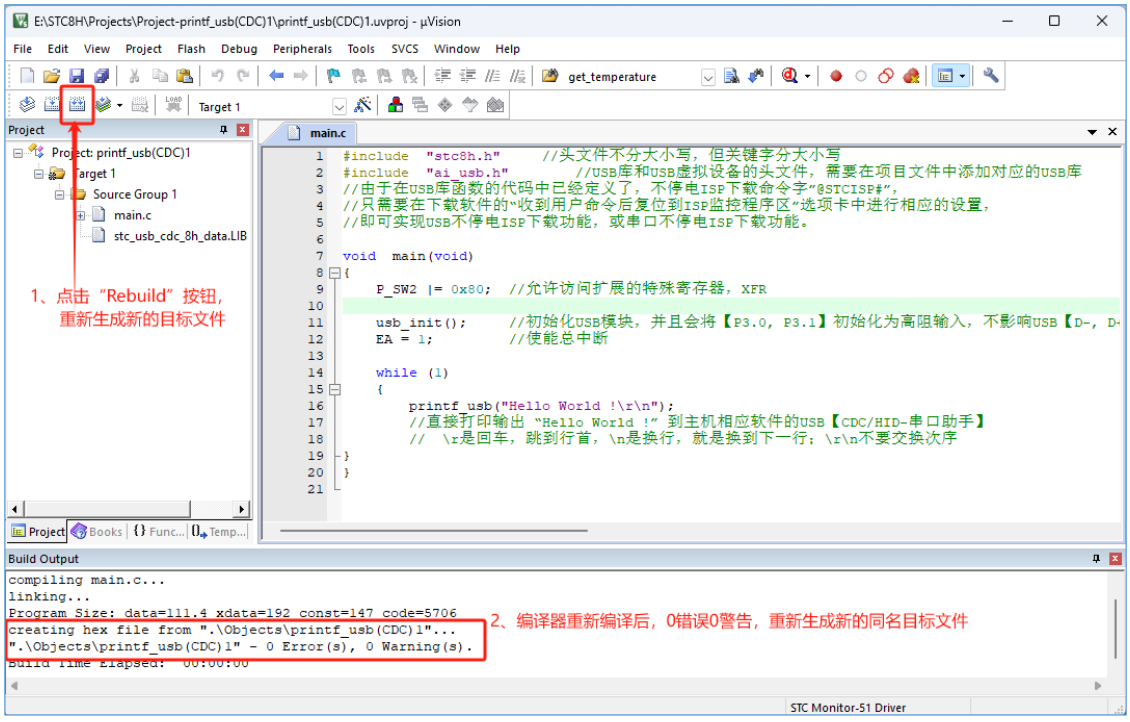
2.2.6.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载

如下图设置：

下载软件会每秒检测多次目标文件被编译的时间是否已更新，如时间已更新，则 ISP 软件会自动重新调入已打开的目标文件，接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。



如上所示，不停电下载实验是成功的，但本设置的优点是不需要手工点击"下载/编程"按钮。



在编译集成环境中，点击"Rebuild"按钮，目标文件会被重新编译生成一次，ISP 软件发现目标文件被编译的时间发生变化后，ISP 软件就会执行前面描述的不停电下载动作。

2.3　实验三：中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其他

看下面的程序，其实还是那个熟悉的标准 C 语言的 printf()，只是到了单片机 C 语言中，变成了 printf_usb()

2.3.1　printf( )，printf_usb( )实验程序代码

```
#include   "stc8h.h"          //头文件不分大小写，但关键字分大小写
#include   "ai_usb.h"         //USB 库和 USB 虚拟设备的头文件，需要在项目文件中添加对应的 USB 库
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。

// USB 接收中断服务程序中，调用用户实际需要执行的服务程序，
//  简称回调函数，用户根据实际应用场景需求编写
void USBLIB_OUT_Callback(void)
{    //用户希望 USB 库函数，中断方式回调执行的用户自己编写的任务函数，简称回调函数
     if (UsbOutBuffer[0] == 6)
          printf_usb("Hello World !\r\n");
          //接收到数据 6 后，直接打印输出“Hello World !”到主机相应软件的 USB【CDC/HID-串口助手】
          //   \r 是回车，跳到行首，\n 是换行，就是换到下一行；\r\n 不要交换次序
     else if (UsbOutBuffer[0] == 7)
          printf_usb("China !\r\n");
          //接收到数据 7 后，直接打印输出 “China !” 到主机相应软件的 USB【CDC/HID-串口助手】”
}

void main(void)
{
     P_SW2 |= 0x80;     //允许访问扩展的特殊寄存器，XFR

     usb_init();             //初始化 USB 模块
     //初始化 USB 模块，并且会将【P3.0, P3.1】初始化为高阻输入，不影响 USB【D-, D+】通讯

     set_usb_OUT_callback(USBLIB_OUT_Callback);  //设置 USB 中断服务程序可以调用的回调函数
     //此处 USBLIB_OUT_Callback 是前面用户自己编写的任务函数的函数名/地址，供 USB 库函数回调

     EA = 1;                 //使能全局中断

     while (1);
}
```

以上是完整的第一个程序代码

printf_usb()函数：

printf_usb()函数和标准 C 语言的格式完全一致，详见 printf()函数章节

进入 printf_usb()函数后，首先会判断 USB 主机是否已经对 USB 设备配置完成，

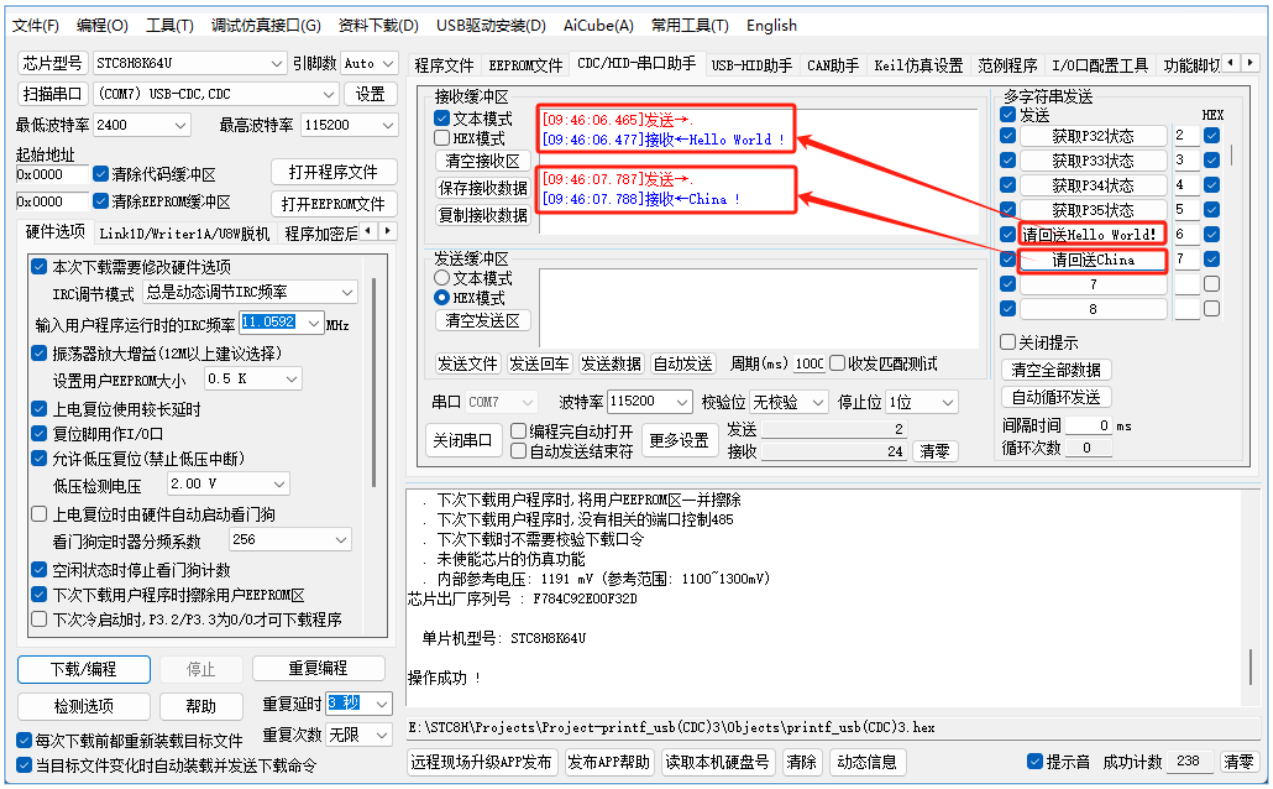
如果没有配置完成，printf_usb()函数会直接返回退出。

如果已经配置完成，还会继续判断如有发送前一笔数据，则判断是否发送完成，

如果没有发送完成，会继续等待，直到前一笔数据发送完成，
才会发送新的 printf_usb()需发送的数据后再返回退出。
但如等待超过 50ms@24MHz，会直接退出。
如果主时钟不是 24MHz，20MHz 等待时间会变长，40MHz 等待时间会变短。
如果已经配置完成且前一笔数据已经发送完成，
则会直接发送新的 printf_usb()需发送的数据并返回退出。

2.3.2　查看实验现象

按 printf_usb()实验二的步骤流程建立项目，编译通过，并将目标文件下载到 STC8H8K64U 实验箱。
运行效果如下图：



点击“请回送 Hello World!”按钮，单片机收到 16 进制数 6 后，回送“Hello World!”给 USB 主机显示。
点击“请回送 China!”按钮，单片机收到 16 进制数 7 后，回送“China!”给 USB 主机显示。

2.3.3　使用【AiCube】工具创建“中断方式，单片机 USB 中断接收服务函数收到命令后，printf_usb("Hello World !\r\n")或其他”

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit**、**Ai8051U-8Bit**、**STC32G12K128**、**STC8H8K64U**、**STC15F2K60S2**、**STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

2.3.3.1　安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

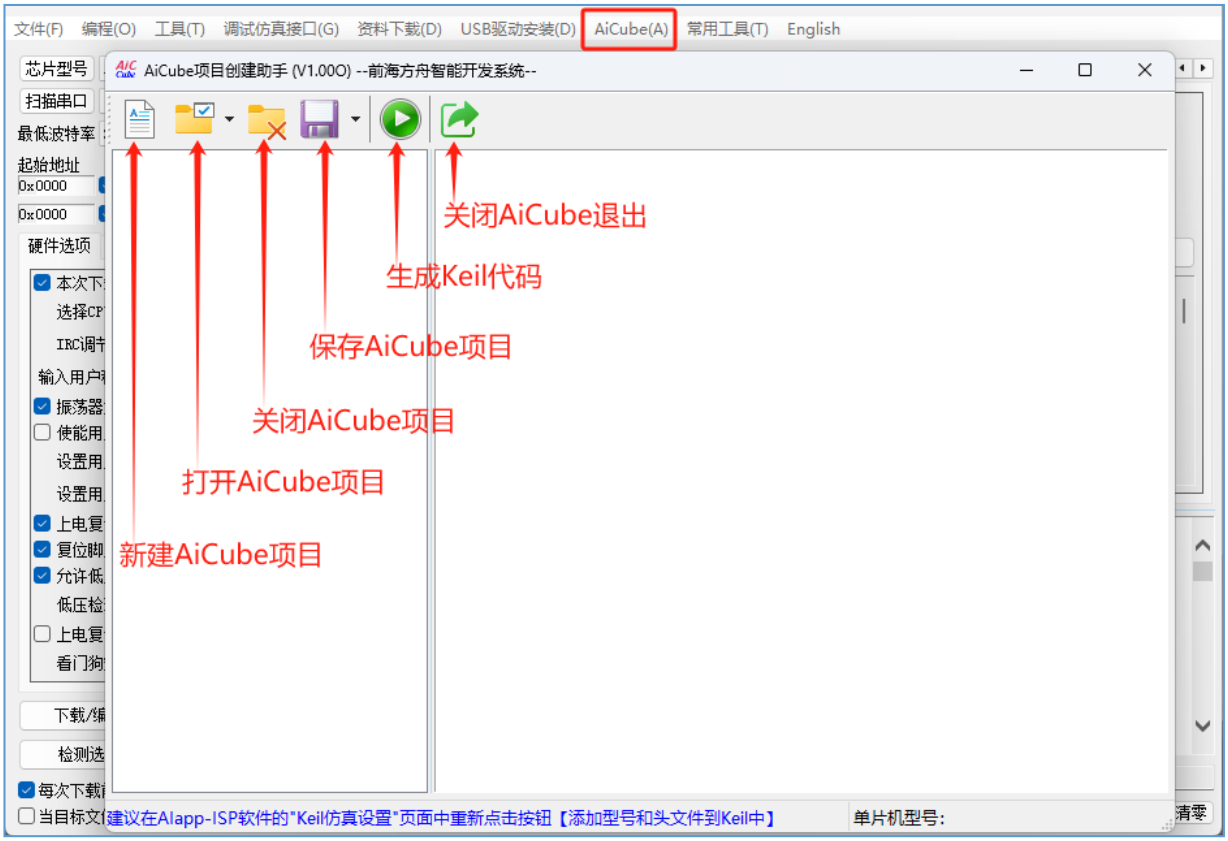
【AiCube-ISP 软件的安装】，请点击查阅本文档 1.1 安装 AiCube-ISP 下载/编程/烧录 工具，含强大的辅助开发工具

2.3.3.2　使用【AiCube】创建项目

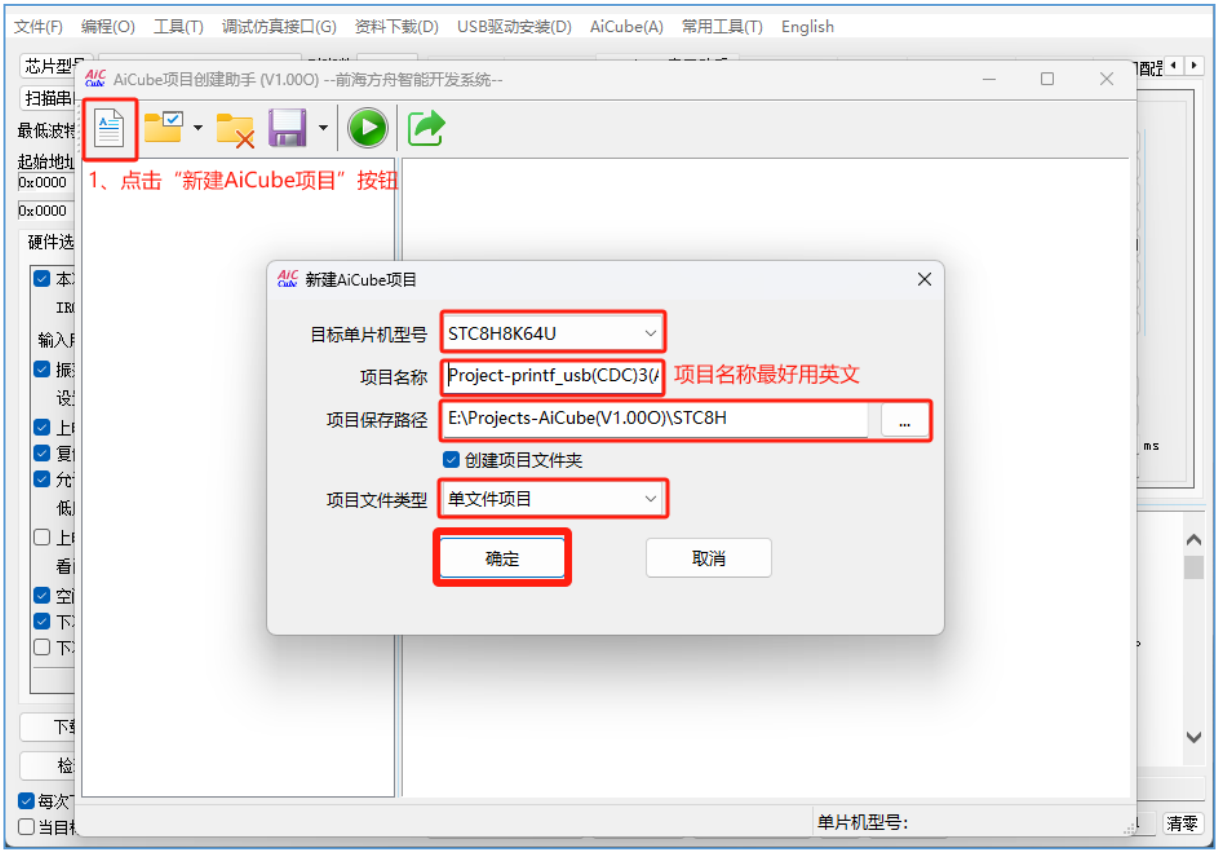
双击图标，打开 AiCube-ISP 软件界面，即可直接使用，如下图，在菜单栏上找到【AiCube(A)】菜单：



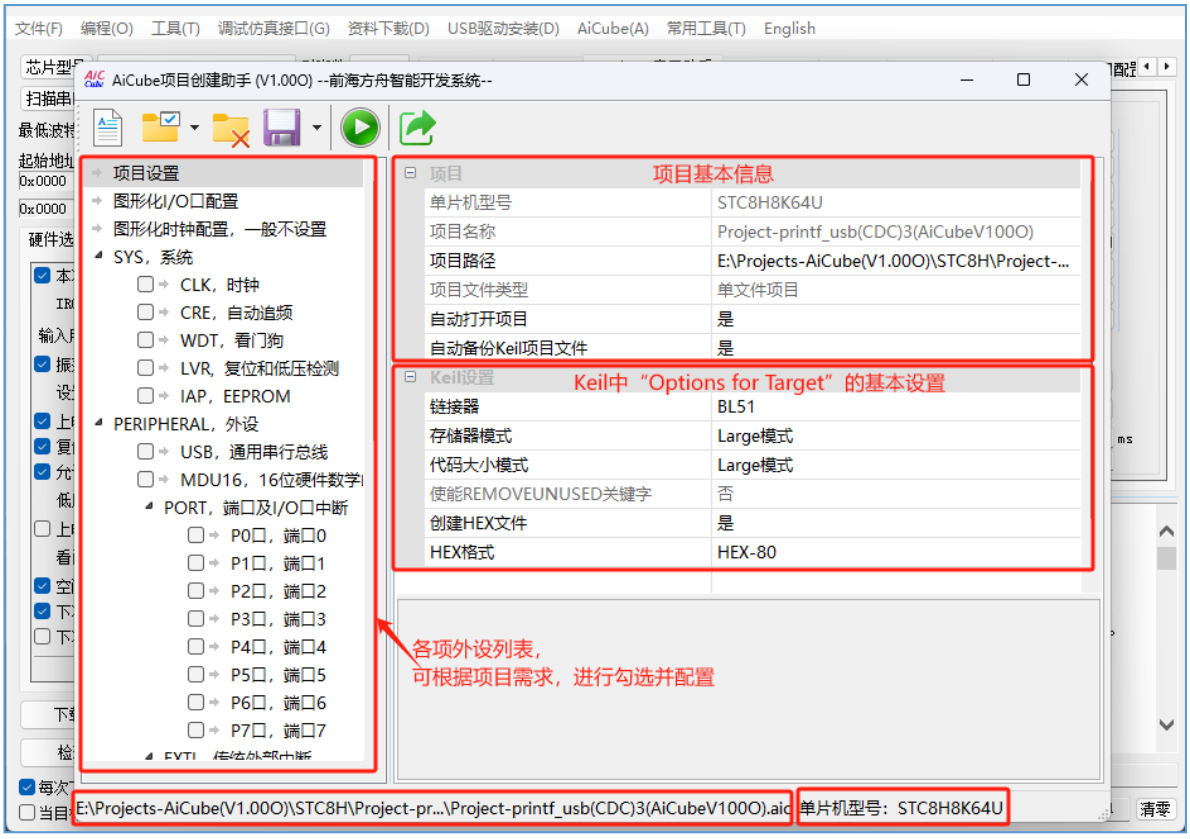
➲ 点击【AiCube(A)】，弹出如下窗口：



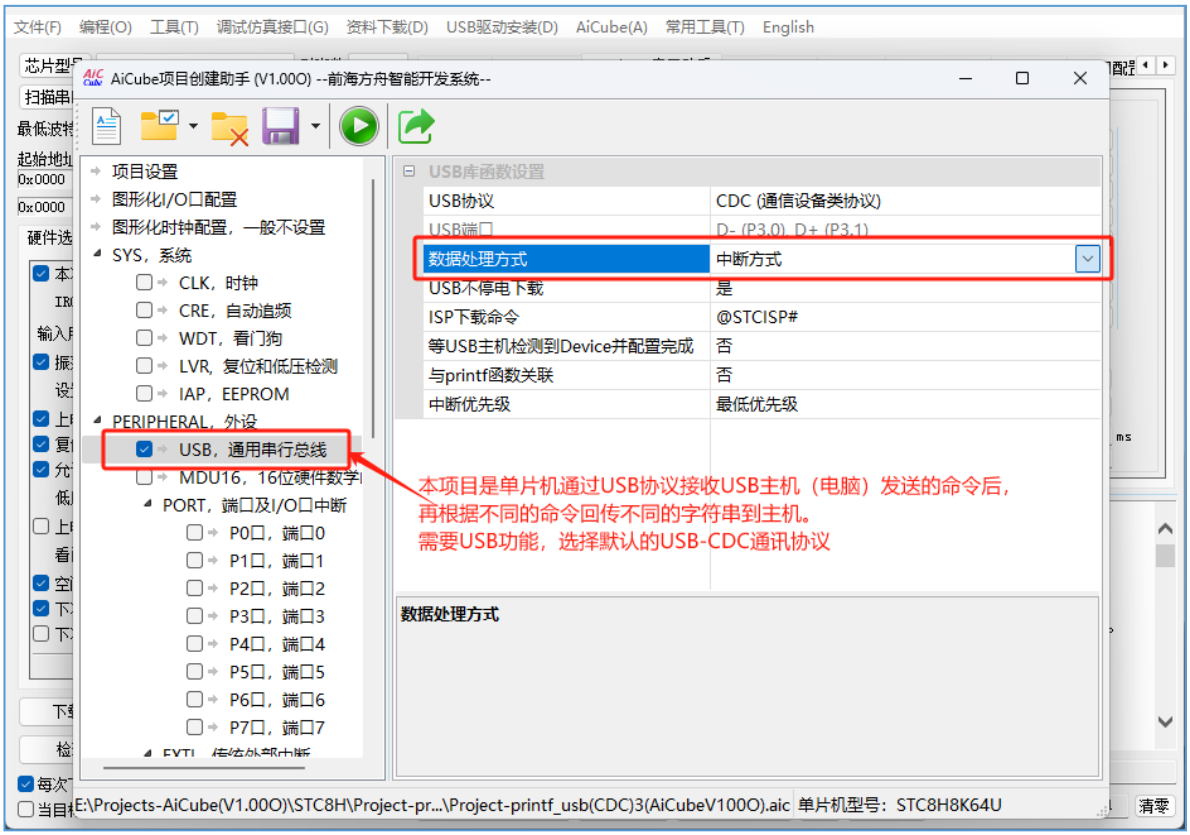
➲ 点击 “新建 AiCube 项目”按钮，出现如下配置界面，选择并填写项目基本信息，建议项目名称用英文！



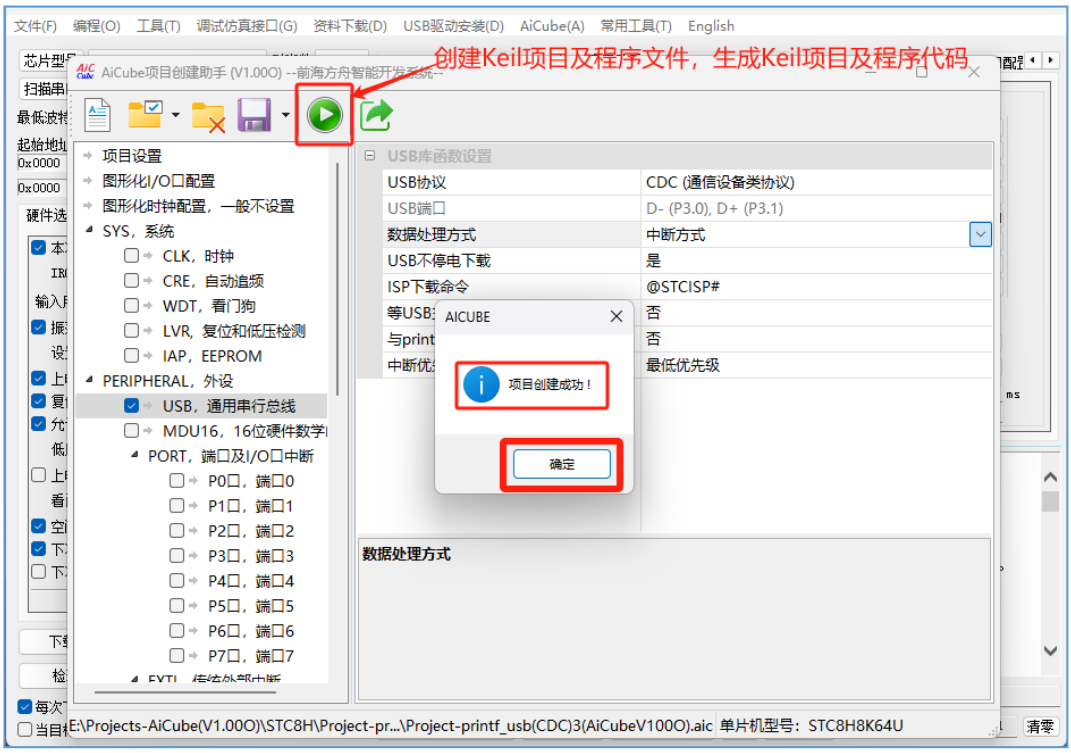
➲ 点击“确定”，出现如下界面：



因本项目用到 USB-CDC 功能，就需要加载 USB 库函数、USB-CDC 协议，要对 USB 初始化等。本项目是用到 USB-CDC 的中断方式，【AiCube】助手中 USB-CDC 默认的数据处理方式就是“查询方式”，根据本项目的实际需求，勾选 USB 外设，然后将“数据处理方式”设置为“中断方式”，如下图：



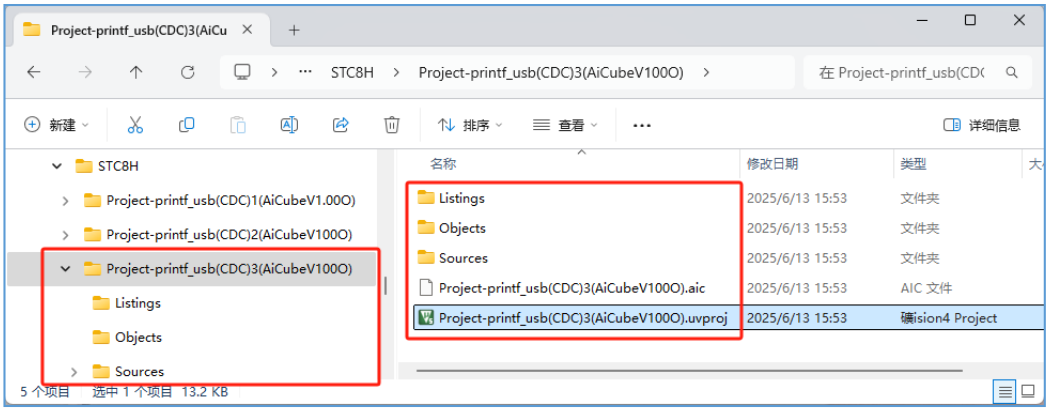
➲ 点击上图“创建 Keil 项目及程序文件”按钮，【AiCube】提示：“项目创建成功！”，如下图：



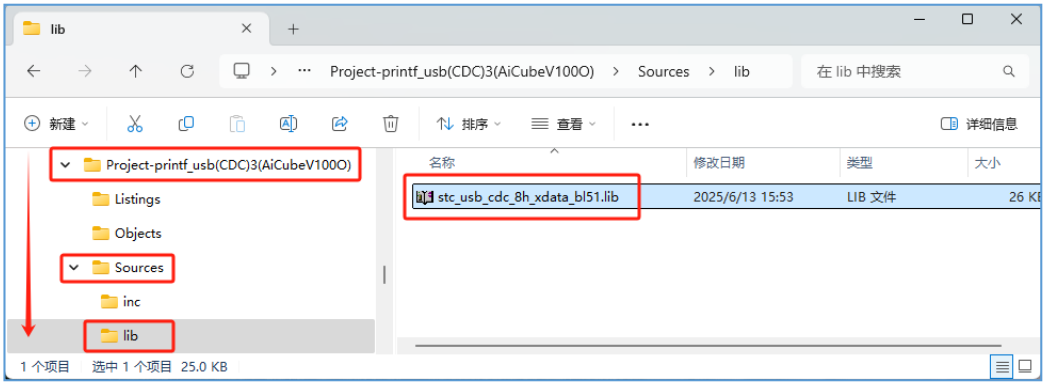
点击“确定”，立即打开 Keil 软件，进入该项目的 Keil 集成开发环境界面，即【AiCube】成功帮我们创建了项目及目录“Project-printf_usb(CDC)3(AiCubeV100O)”。至此，可以关闭【AiCube】工具。

➲ 找到并打开刚刚创建的项目目录

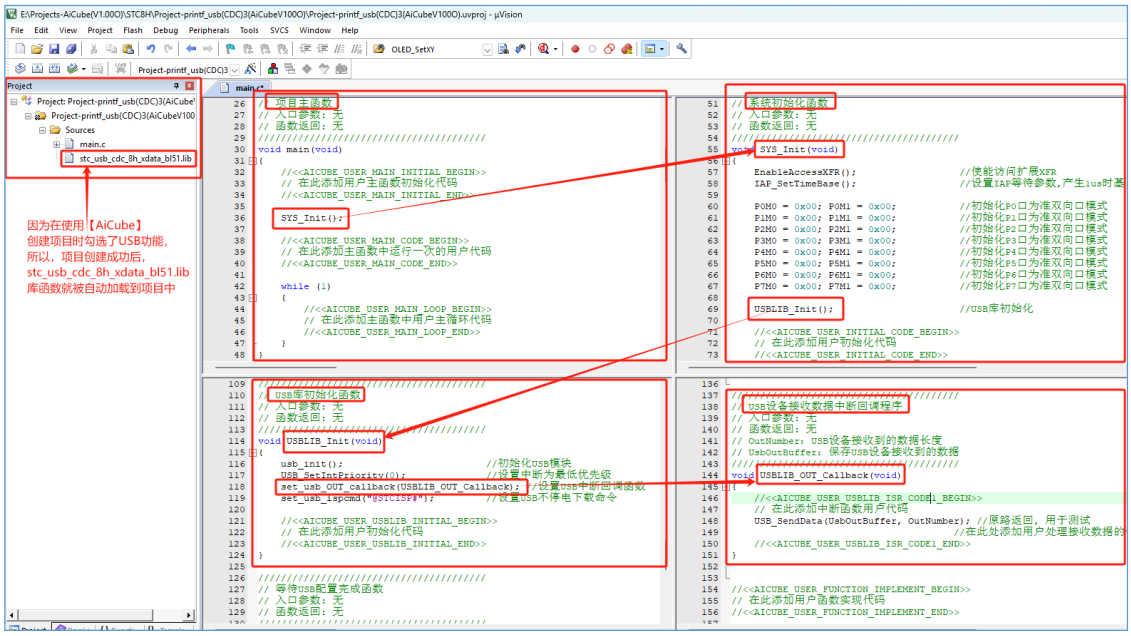
发现通过上述的操作过程,【AiCube】工具已经帮我们建立了项目目录及 Project-printf_usb(CDC)3(AiCube V100O).uvproj 文件，如下图：



同时打开 Sources>lib 子目录，发现，【AiCube】将相应的 USB 库函数加载进项目目录：



➲ 观察已打开的 Keil 项目文件：Project-printf_usb(CDC)3(AiCubeV100O).uvproj，如下图：



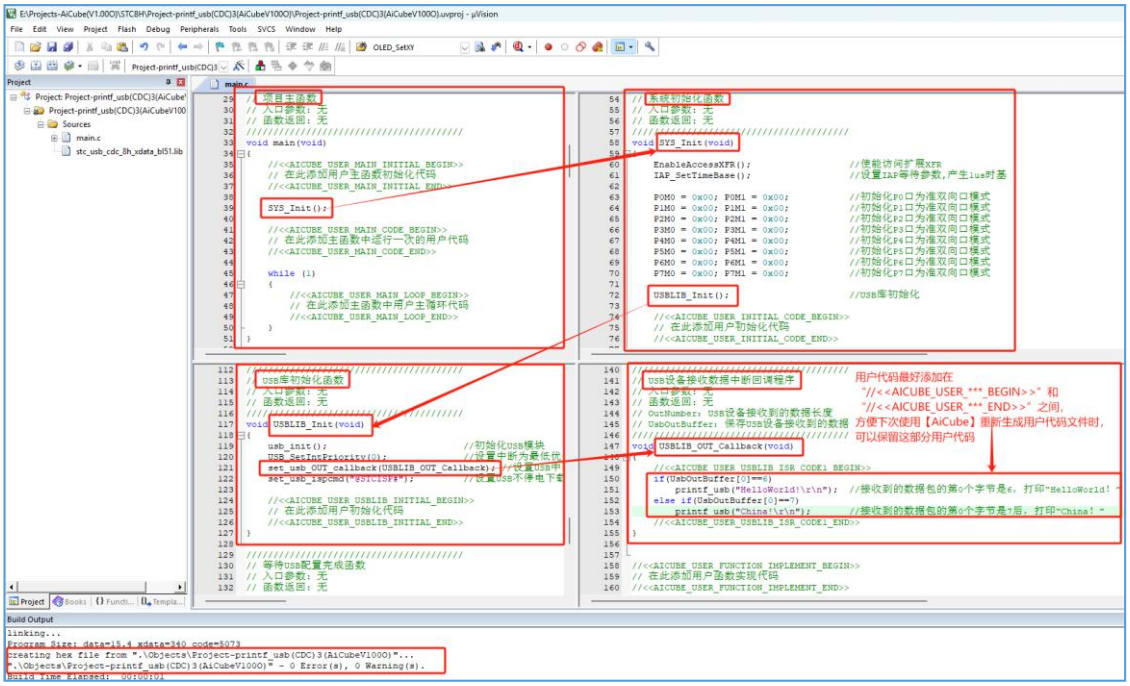
根据在【AiCube】中的配置，从上图中我们看到【AiCube】创建的本实验项目中不仅有了 main.c，还已经自动加载了 USB 库函数（stc_usb_cdc_8h_xdata_bl51.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、微秒及毫秒延时函数、USB 初始化及相关的配置与处理函数。

2.3.3.3　　增加实验项目代码

为了实现单片机通过 USB 协议接收 USB 主机（电脑）发送的命令后，再根据不同的命令回传不同的字符串到主机。我们现在只需要在“USB 设备接收数据中断回调程序”子函数中加入如下代码即可：

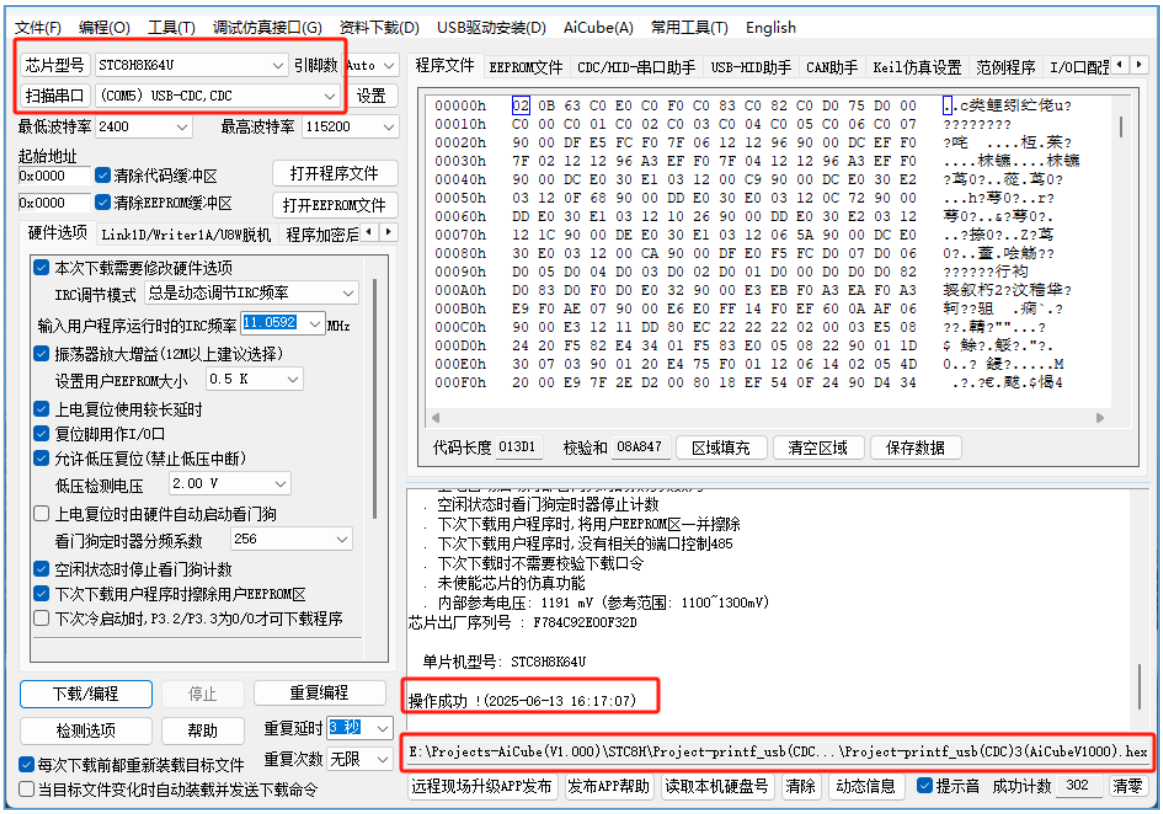
```
if(UsbOutBuffer[0]==6)
    printf_usb("HelloWorld!\r\n");      //接收到的数据包的第 0 个字节是 6，打印“HelloWorld！”
else if(UsbOutBuffer[0]==7)
    printf_usb("China!\r\n");           //接收到的数据包的第 0 个字节是 7 后，打印“China！”
```

保存，编译，如下图（使用菜单“Window”→“Split”将原窗口分裂出 4 个窗口，对比上图观察）：

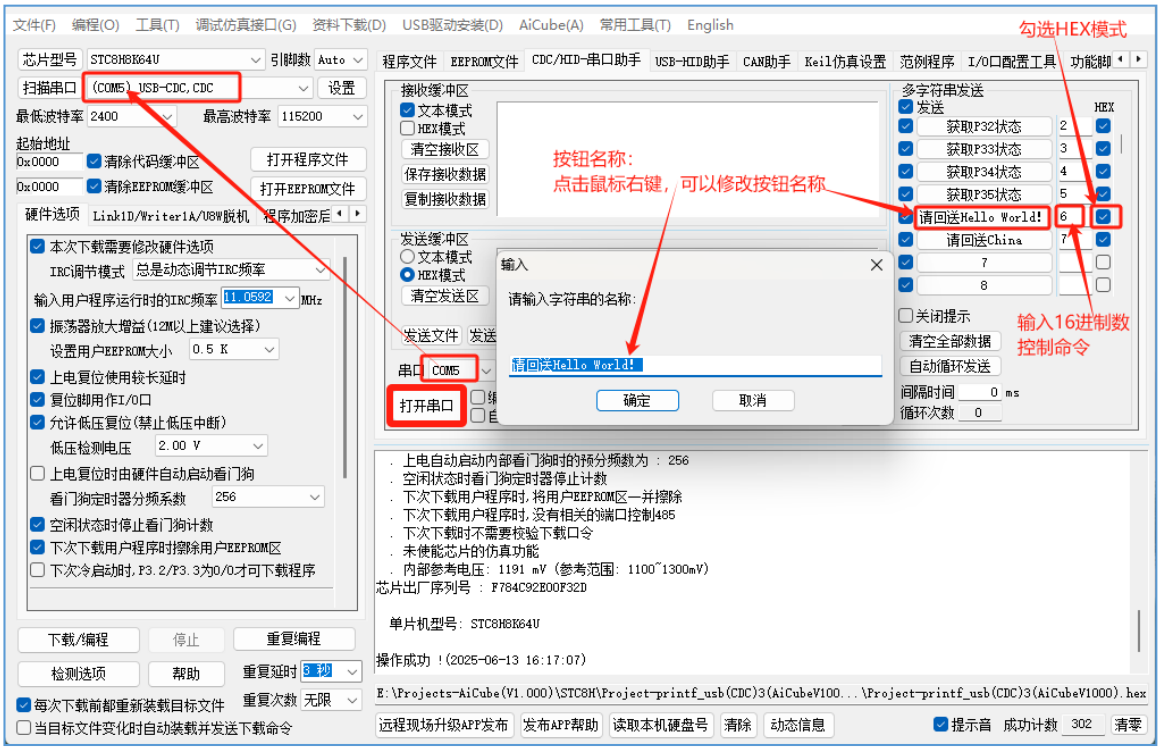


2.3.3.4　使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果

连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“Project-printf_usb(CDC)3 (AiCubeV100O).hex”，如下图：



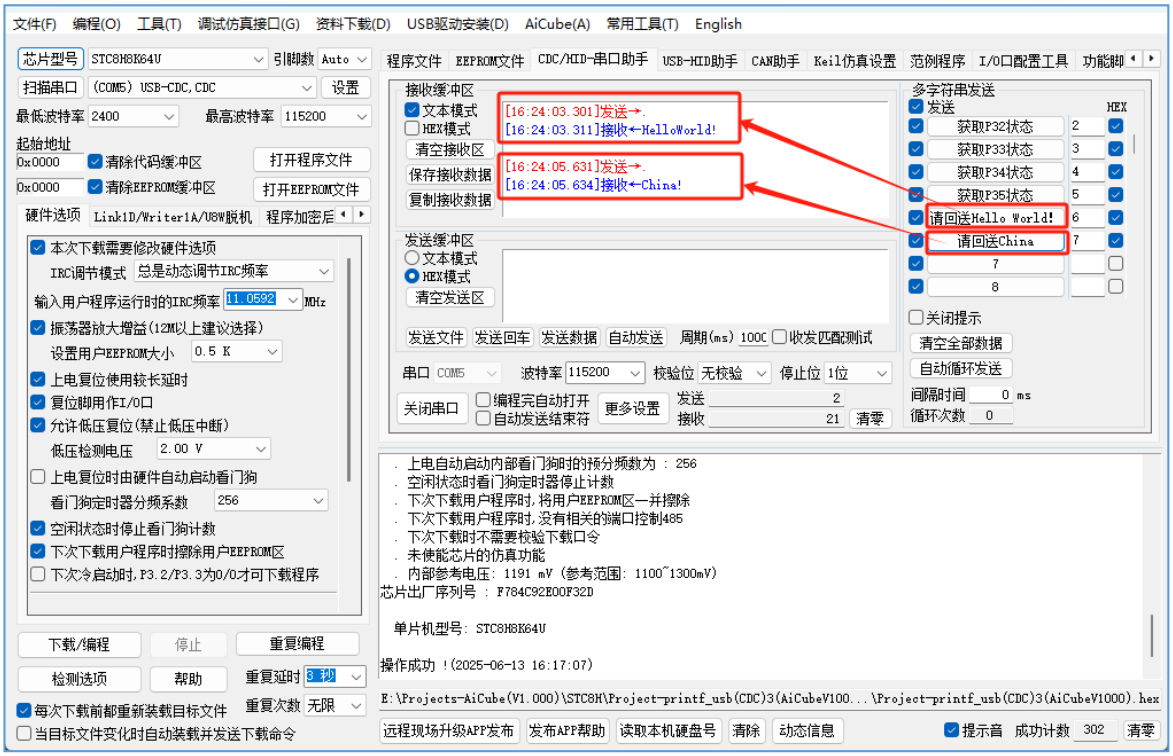
在 AiCube-ISP 系统软件，我们看到：此时串口显示“（COM5）USB-CDC，CDC”。打开“CDC/HID-串口助手”选项卡，在“多字符串发送”窗口，设置好发送控制命令，以便观察实验效果，如下图：



在 CDC/HID-串口助手选项卡，找到并点击“打开串口”按钮，一切准备就绪！

点击“请回送 Hello World!”按钮，单片机收到 16 进制数 6 后，回送“Hello World!”给 USB 主机显示。
点击“请回送 China!”按钮，单片机收到 16 进制数 7 后，回送“China!”给 USB 主机显示。

在 CDC/HID-串口助手的“接收缓冲区”，如下图：



完成实验验证！

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.3.4　USB 不停电下载，ISP 软件如何设置

在开发过程中，程序经常需要反复修改，重新编译后生成新的 HEX 文件，就需要停电/上电进入下载模式，很不方便。其实可以利用我们提供的 USB 不停电 ISP 下载功能，进行自动下载。

在有 USB 功能的实验中，都要加载 USB 库函数，并对 USB 进行初始化。由于在 USB 库函数的代码中已经定义了不停电 ISP 下载命令"@STCISP#"，只需要在下载软件的"收到用户命令后复位到 ISP 监控程序区"的选项卡中进行如下设置，即可实现【USB 不停电下载】功能。

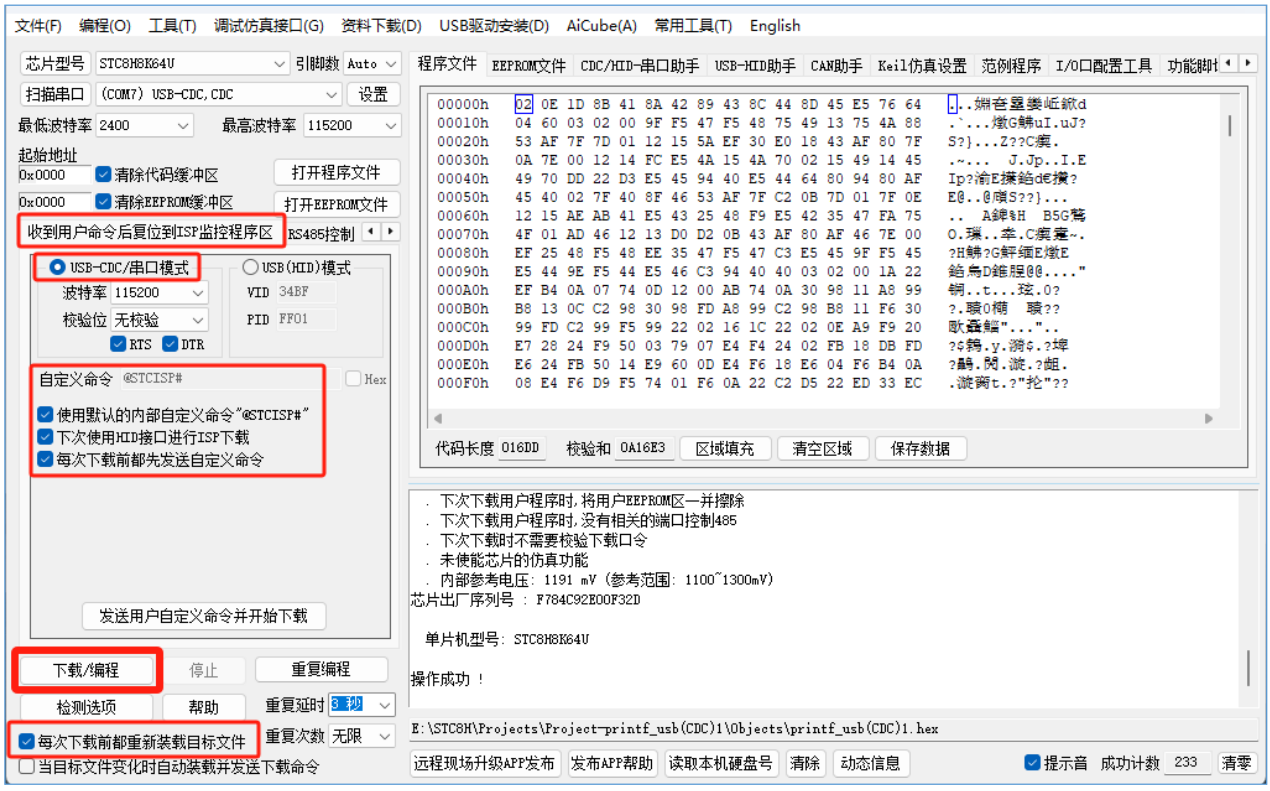
- 如使用 USB-CDC 库函数产生 USB-CDC 虚拟串口，可实现 USB-CDC 不停电自动 ISP 下载功能，并可忽略传统串口的波特率、校验位等参数的设置，下载时系统程序区还是使用的 USB-HID 通讯协议；
- 如果是传统的串口，需正确设置波特率、校验位等参数，才能实现串口的不停电自动 ISP 下载功能；
- 如果使用 USB-HID 库函数产生 USB-HID 设备，则选择"USB（HID）模式"就能实现不停电自动 ISP 下载功能，下载时系统程序区还是使用的 USB-HID 通讯协议。

ISP 软件如何设置：

2.3.4.1　手动点击【下载/编程】按钮，ISP 软件自动重新打开文件，不停电下载

如下图设置，就可以实现：

在点击"下载/编程"按钮后，ISP 软件会重新调入已打开的目标文件一次（不管目标文件有无变化），接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。

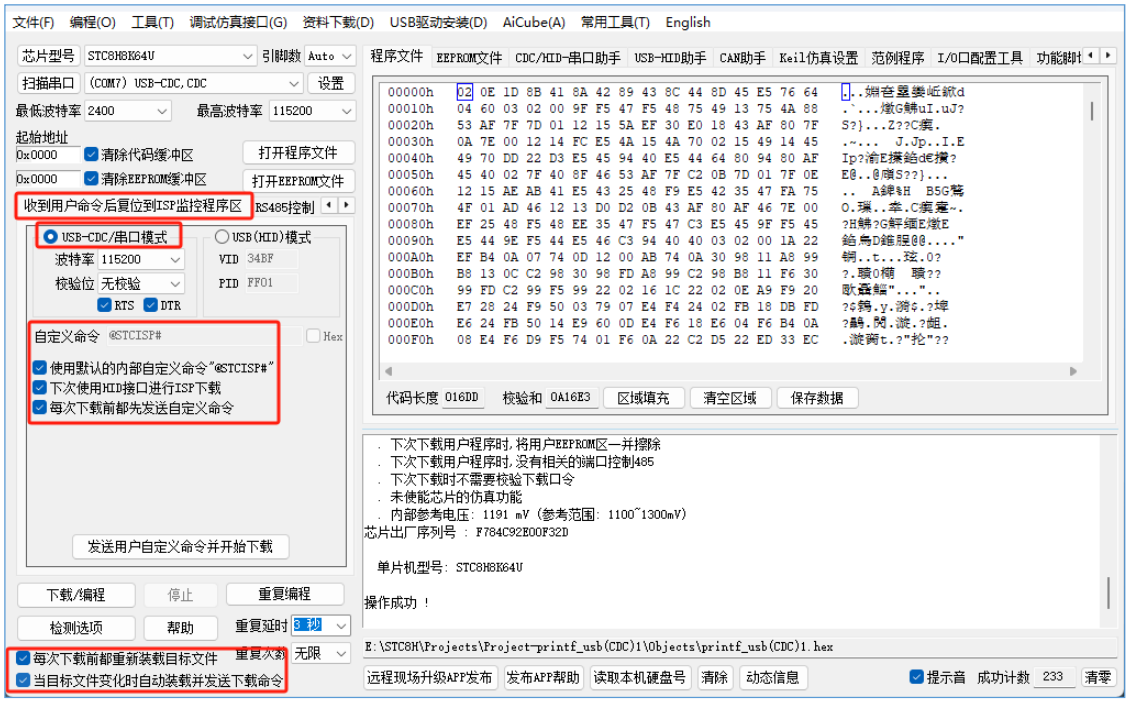


如上所示，不停电下载实验是成功的，但本设置还需要手工点击一次"下载/编程"按钮。

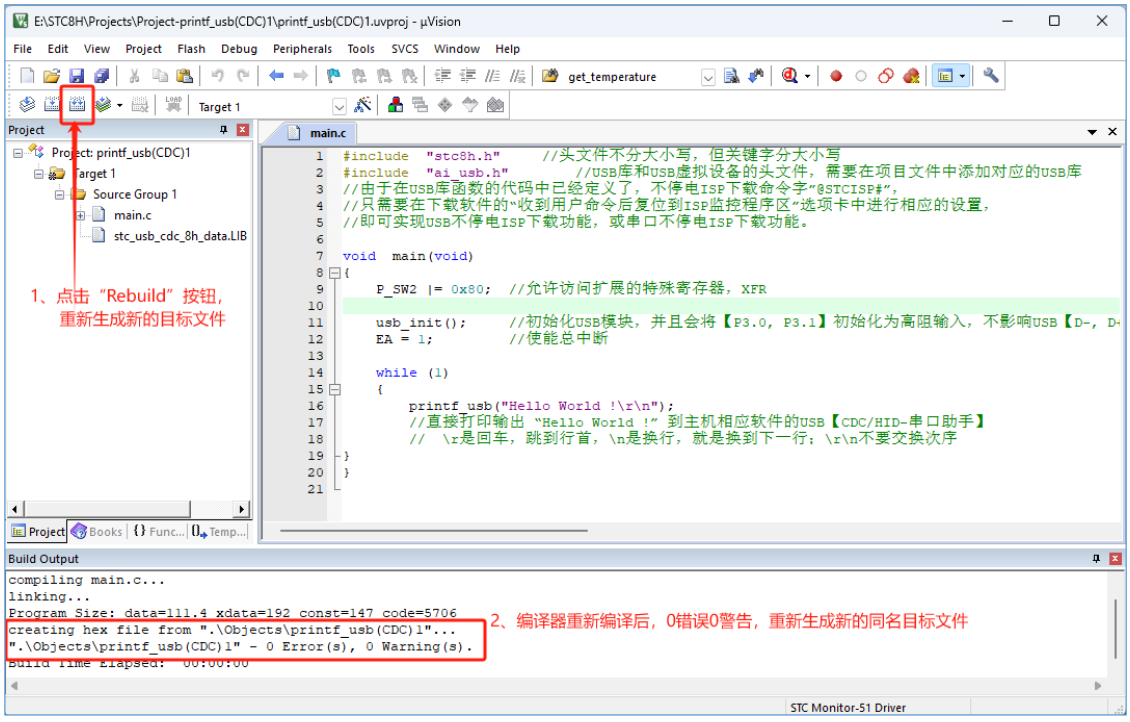
2.3.4.2 编译器重新编译后，ISP 软件会自动重新调入文件，进行不停电下载

如下图设置：

下载软件会每秒检测多次目标文件被编译的时间是否已更新，如时间已更新，则 ISP 软件会自动重新调入已打开的目标文件，接着通过 USB-CDC 通讯协议自动发送自定义下载命令如"@STCISP#"给单片机，单片机中的用户软件收到该命令后，就会软复位到单片机的系统监控程序区，等待 ISP 下载，本设置还指定了电脑端软件后续是通过 USB-HID 通讯协议传输用户程序到目标单片机进行 ISP 下载。



如上所示，不停电下载实验是成功的，但本设置的优点是不需要手工点击"下载/编程"按钮。

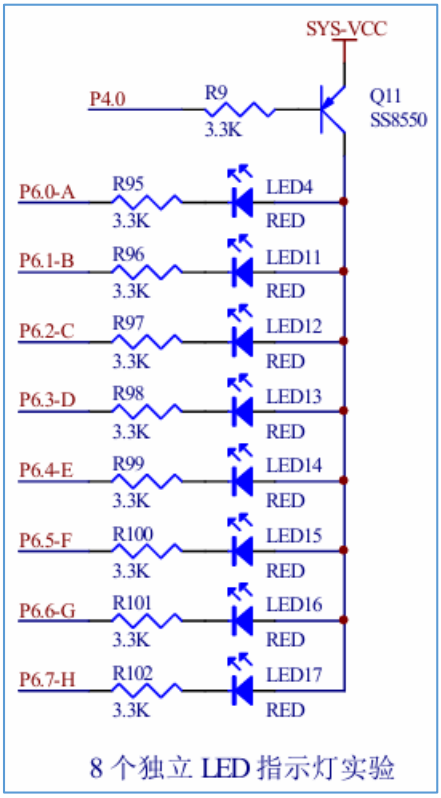


在编译集成环境中，点击"Rebuild"按钮，目标文件会被重新编译生成一次，ISP 软件发现目标文件被编译的时间发生变化后，ISP 软件就会执行前面描述的不停电下载动作。

2.4 点灯实验一：点亮 LED 灯，继续完成前面的工程项目

2.4.1 原理介绍

STC8H8K64U 实验箱原理图



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如果 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P6.0 输出 0，LED60 灯就会被点亮；如 P6.0 输出 1，LED60 灯就会熄灭；
如 P6.1 输出 0，LED61 灯就会被点亮；如 P6.1 输出 1，LED61 灯就会熄灭；
如 P6.2 输出 0，LED62 灯就会被点亮；如 P6.2 输出 1，LED62 灯就会熄灭；
如 P6.3 输出 0，LED63 灯就会被点亮；如 P6.3 输出 1，LED63 灯就会熄灭；
如 P6.4 输出 0，LED64 灯就会被点亮；如 P6.4 输出 1，LED64 灯就会熄灭；
如 P6.5 输出 0，LED65 灯就会被点亮；如 P6.5 输出 1，LED65 灯就会熄灭；
如 P6.6 输出 0，LED66 灯就会被点亮；如 P6.6 输出 1，LED66 灯就会熄灭；
如 P6.7 输出 0，LED67 灯就会被点亮；如 P6.7 输出 1，LED67 灯就会熄灭

如 P6 口对外全部送 0，则这 8 个灯全亮；
如 P6 口对外全部送 1，则这 8 个灯全灭

如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P6 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

2.4.2 代码部分

```
#include < STC8H.H >        //包含 STC8H 的头文件

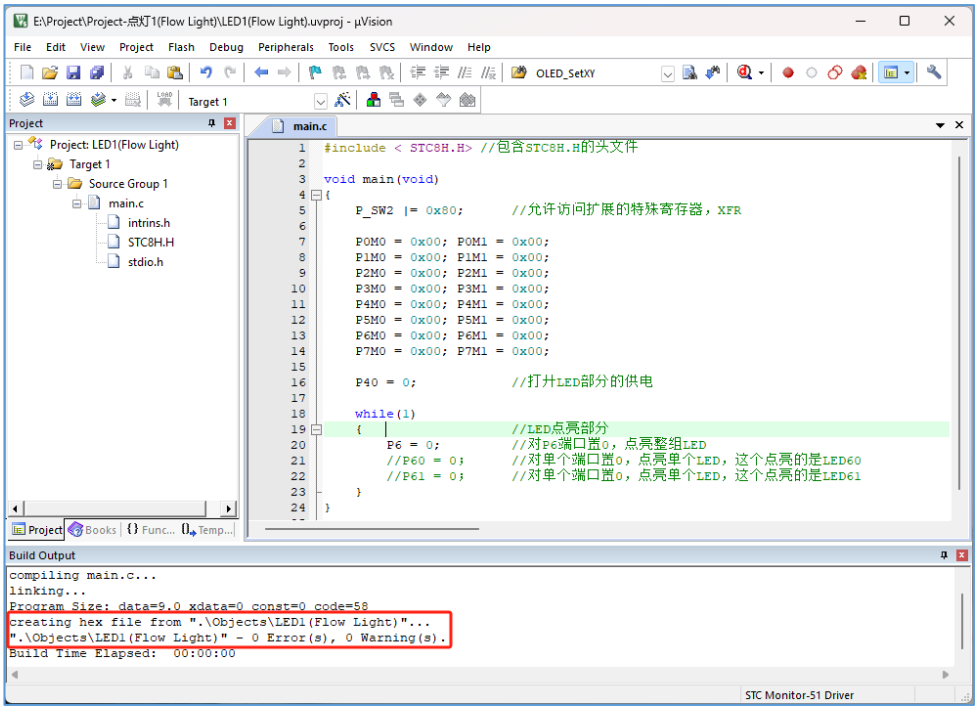
void main(void)
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式
//这段将所有 I/O 口都初始化为准双向口工作模式的代码，可以使用 AiCube-ISP 软件的
//“I/O 口配置工具”来自动生成。详见本节【小技巧--灵活运用 AiCube-ISP 软件工具自动生成代码】

    P40 = 0;                        //打开 LED 部分的供电

    while(1)
    {                               //LED 点亮部分
        P6 = 0;                     //对 P6 端口置 0，点亮整组 LED
        //P60 = 0;                  //对单个端口置 0，点亮单个 LED，这个点亮的是 LED60
        //P61 = 0;                  //对单个端口置 0，点亮单个 LED，这个点亮的是 LED61
    }
}
```

编写完成代码后，保存，点击工具栏上的【Build】按钮进行编译，如下图：

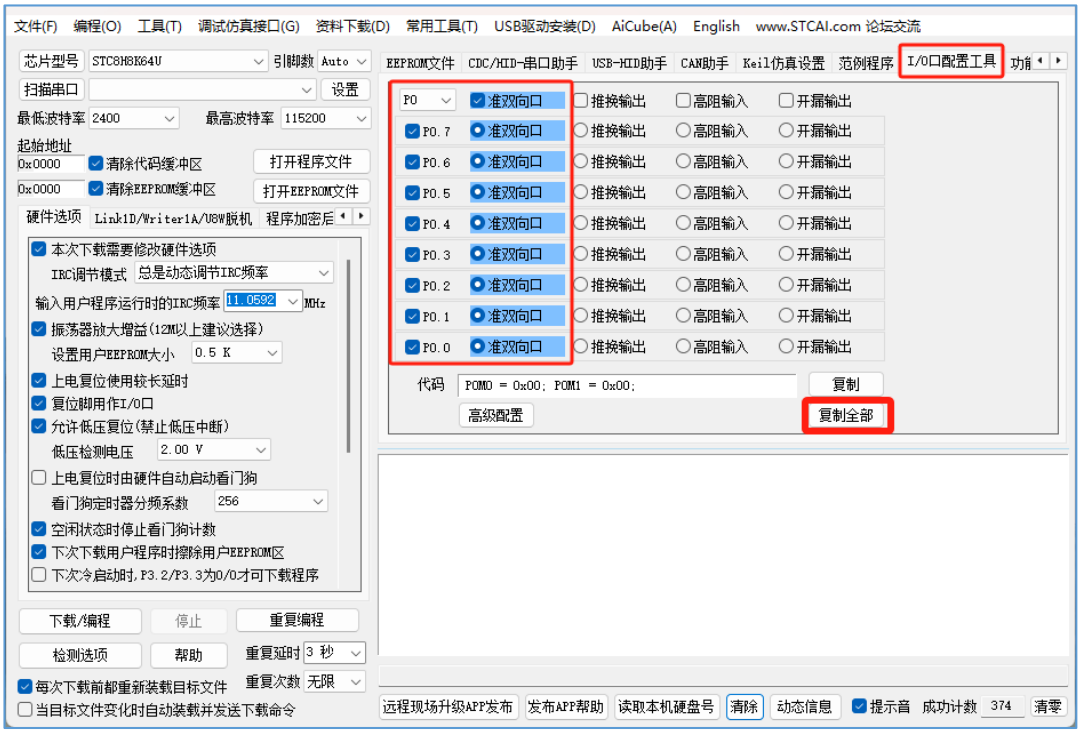


编译完成后，会在底部的 Build Output 窗口生成编译信息：
出现 creating hex file 和 0 Error(s) 0 Warning（s）即为编译成功，无报错且成功创建了 hex 文件。
此 hex 文件，一般是在对应工程目录下的 Objects 文件夹里。

【小技巧--灵活运用 AiCube-ISP 软件的工具自动生成代码】
例如上述代码中这一段将所有 I/O 口设置为准双向口模式的代码:

```
P0M0=0x00; P0M1=0x00:
P1M0=0x00; P1M1=0x00;
P2M0=0x00; P2M1=0x00;
P3M0=0x00; P3M1=0x00;
P4M0=0x00; P4M1=0x00:
P5M0=0x00; P5M1=0x00;
P6M0=0x00; P6M1=0x00;
P7M0=0x00; P7M1 =0x00;
```

就可以使用 **AiCube-ISP** 软件的**【I/O 口配置工具】**进行生成:



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

2.4.3 下载用户程序并查看实验现象

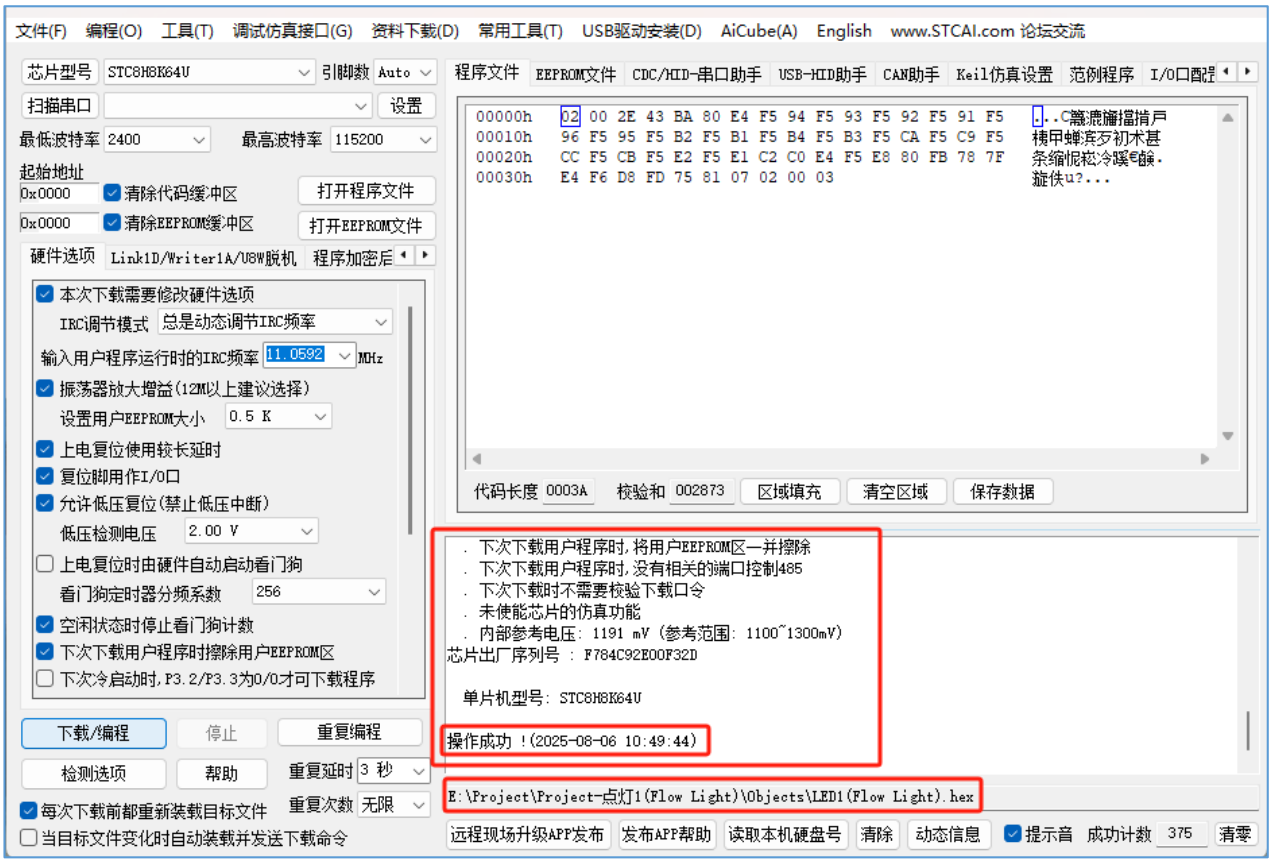
将 STC8H8K64U 实验箱通过 USB TypeA 到 USB TypeA 线与电脑连接，运行 AiCube-ISP 软件，按步骤“打开程序文件”LED1(Flow Light).hex（这里“打开程序文件”打开的是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在对应工程目录下的 Objects 文件夹里）并点击“下载/编程”，如下图：

【AiCube-ISP 软件的安装】，请点击查阅本文档 1.1 安装 AiCube-ISP 下载/编程/烧录 工具，含强大的辅助开发工具

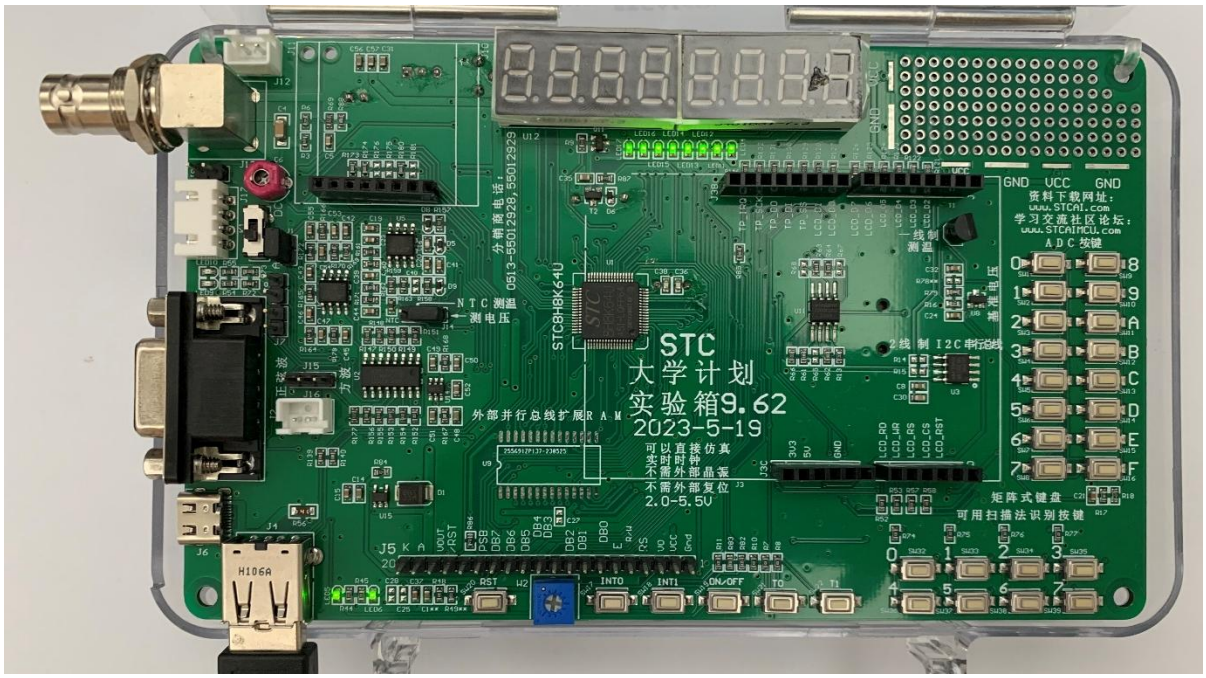
【STC8H8K64U 实验箱实物图、与电脑的连接、用户程序下载流程】，请点击查阅本文档的 1.2 STC8H8K64U 实验箱实物图及程序下载

AiCube-ISP 软件界面出现：操作成功！

恭喜你！



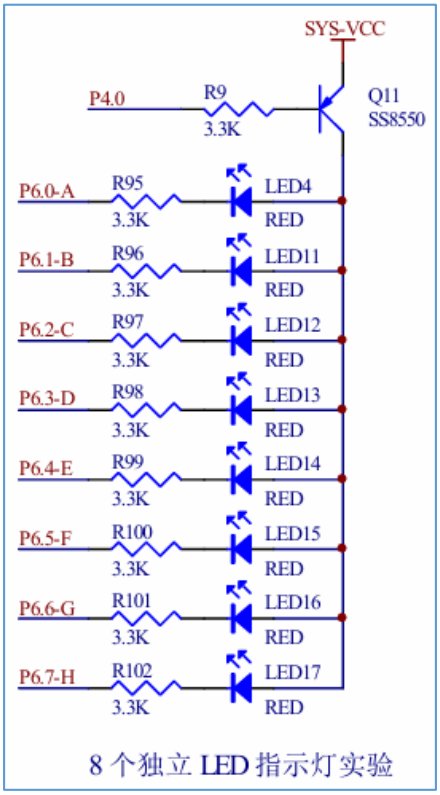
这时，我们看到 STC8H8K64U 实验箱上方刚刚暗淡无光的 8 个 LED 灯，齐齐亮了起来！至此，实验成功！



2.5 点灯实验二：让 LED 灯闪烁起来

2.5.1 原理介绍

STC8H8K64U 实验箱原理图



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如果 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P6.0 输出 0，LED60 灯就会被点亮；如 P6.0 输出 1，LED60 灯就会熄灭；
如 P6.1 输出 0，LED61 灯就会被点亮；如 P6.1 输出 1，LED61 灯就会熄灭；
如 P6.2 输出 0，LED62 灯就会被点亮；如 P6.2 输出 1，LED62 灯就会熄灭；
如 P6.3 输出 0，LED63 灯就会被点亮；如 P6.3 输出 1，LED63 灯就会熄灭；
如 P6.4 输出 0，LED64 灯就会被点亮；如 P6.4 输出 1，LED64 灯就会熄灭；
如 P6.5 输出 0，LED65 灯就会被点亮；如 P6.5 输出 1，LED65 灯就会熄灭；
如 P6.6 输出 0，LED66 灯就会被点亮；如 P6.6 输出 1，LED66 灯就会熄灭；
如 P6.7 输出 0，LED67 灯就会被点亮；如 P6.7 输出 1，LED67 灯就会熄灭

如 P6 口对外全部送 0，则这 8 个灯全亮；
如 P6 口对外全部送 1，则这 8 个灯全灭

如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P6 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

2.5.2 代码部分

```
#include < STC8H.H >          //包含 STC8H 的头文件
#include "intrins.h"          //使用_nop_()函数所必须要包含的头文件,
                              //否则延时函数中调用的_nop_()函数没有被头文件引用过来,
                              //会导致编译器找不到这个而函数而报错。

void Delay500ms(void)         //延时函数，@22.1184MHz
{
    unsigned char data i, j, k;    //定义变量

    i = 57;
    j = 27;
    k = 112;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

//上面延时函数部分代码可使用 **AiCube-ISP** 软件的“**软件延时计算器**”工具来生成，见本节后面的**【小技巧--灵活运用 AiCube-ISP 软件的工具自动生成代码】**

```
void main(void)
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

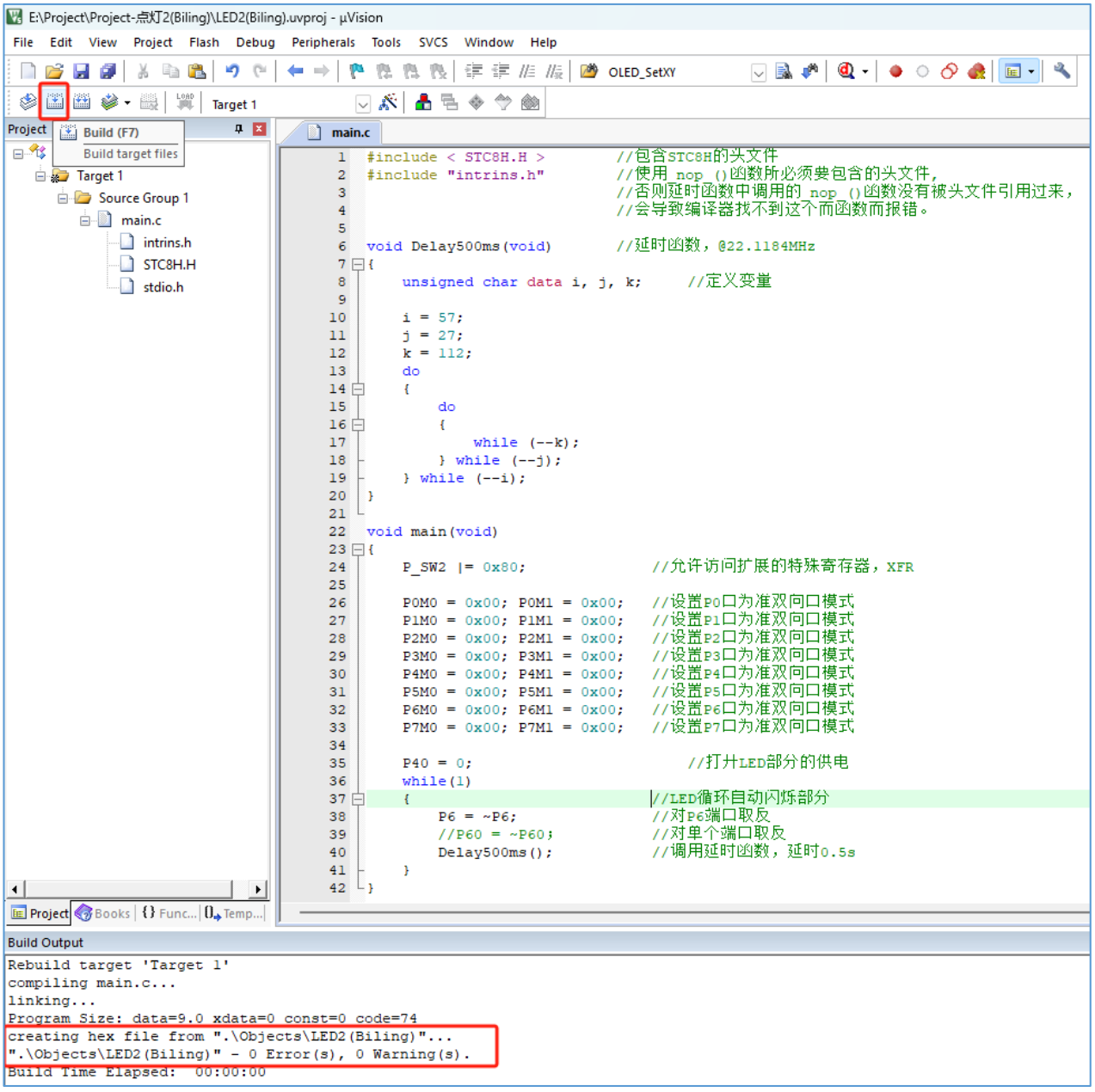
    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    P40 = 0;                        //打开 LED 部分的供电

    while(1)
    {                               //LED 循环自动闪烁部分
        P6 = ~P6;                   //对 P6 端口取反
        //P60 = ~P60;               //对单个端口取反
        Delay500ms();               //调用延时函数，延时 0.5s
```

```
    }
}
```

编写完成代码后，保存，点击工具栏上的【Build】按钮进行编译：



编译完成后，会在底部的 Build Output 生成编译信息

出现 creating hex file 和 0 Error(s) 0 Warning（s）即为编译成功，无报错且成功创建了 hex 文件。

此 hex 文件，一般是在对应工程目录下的 Objects 文件夹里。

【小技巧--灵活运用 AiCube-ISP 软件的工具自动生成代码】

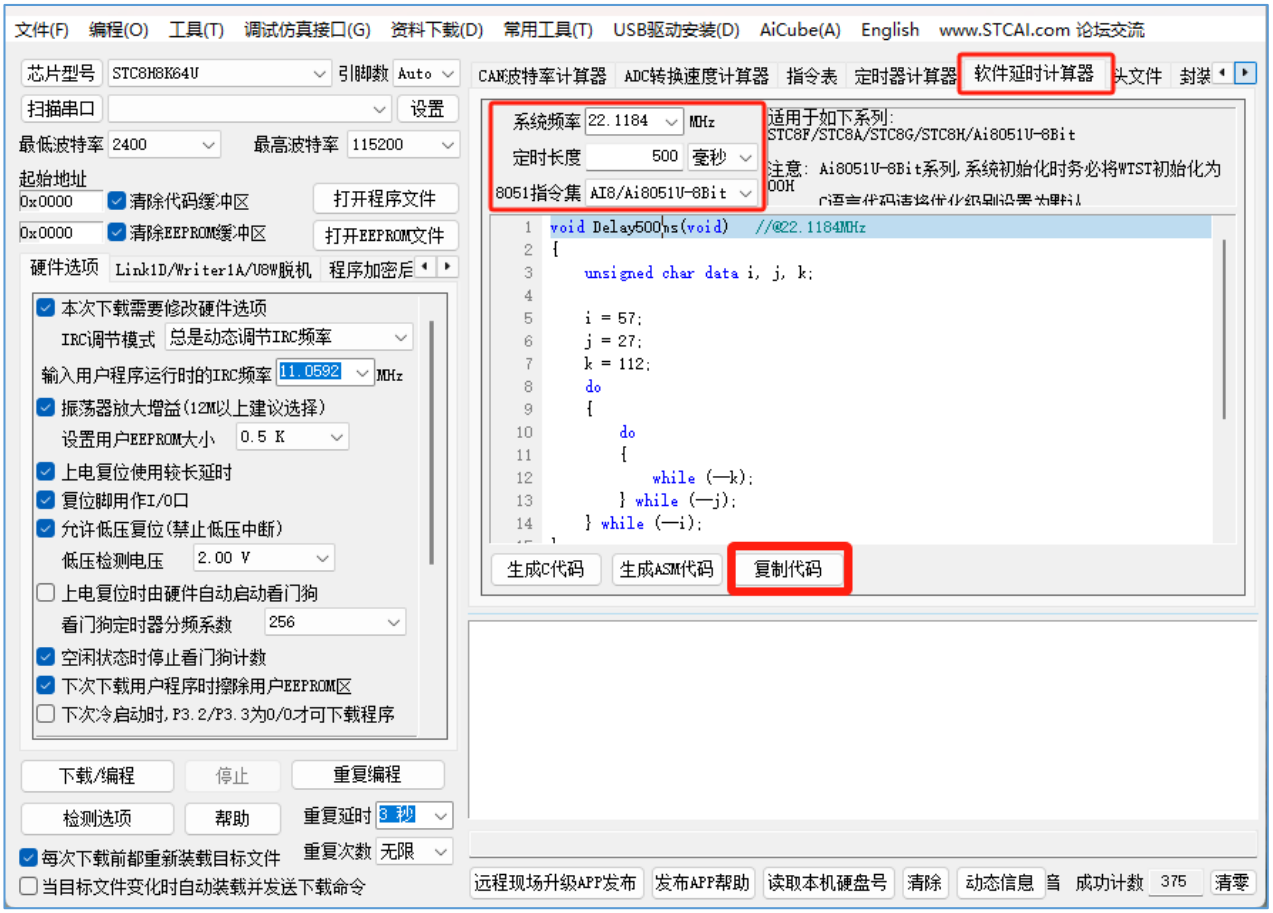
延时函数部分代码就可以使用 **AiCube-ISP** 软件的“**软件延时计算器**”来生成。

例如上述代码中这一段延时函数的代码：

```
void Delay500ms(void)　　//@22.1184MHz
{
    unsigned char data i, j, k;
```

```
    i = 57;
    j = 27;
    k = 112;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

就可以使用 AiCube-ISP 软件的“软件延时计算器”来生成：



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

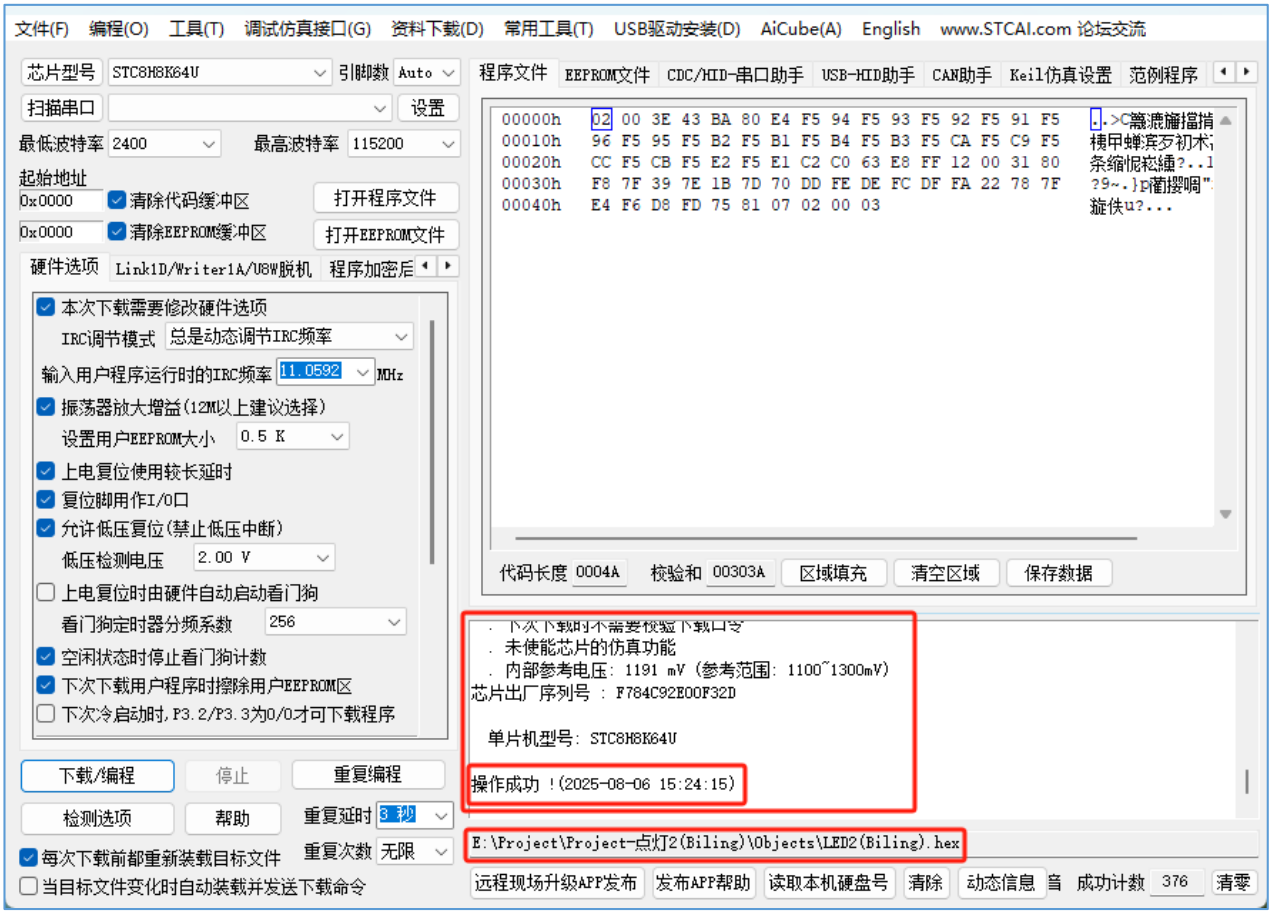
2.5.3　下载用户程序并查看实验现象

将 STC8H8K64U 实验箱通过 USB TypeA 到 USB TypeA 线与电脑连接，运行 AiCube-ISP 软件，按步骤“打开程序文件”LED2(Biling).hex（这里“打开程序文件”打开的是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在对应工程目录下的 Objects 文件夹里）并点击“下载/编程”，如下图：

【AiCube-ISP 软件的安装】，请点击查阅本文档 1.1 安装 AiCube-ISP 下载/编程/烧录 工具，含强大的辅助开发工具

【STC8H8K64U 实验箱实物图、与电脑的连接、用户程序下载流程】，请点击查阅本文档的 1.2 STC8H8K64U 实验箱实物图及程序下载

AiCube-ISP 软件界面出现：操作成功！
恭喜你！

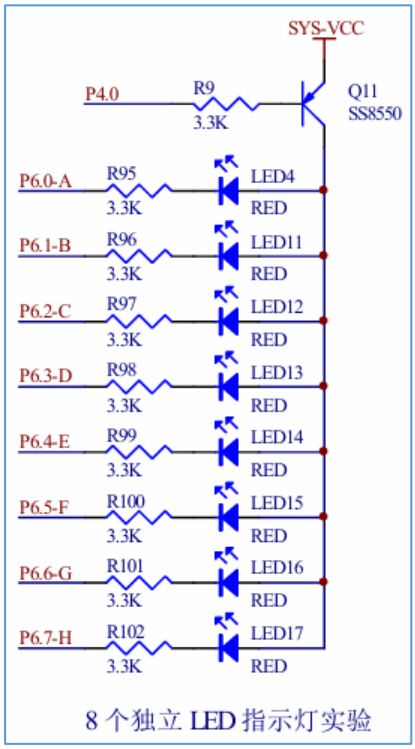


这时，STC8H8K64U 实验箱上方的刚刚暗淡无光的 8 个 LED 灯，齐齐一亮一灭闪烁起来了！至此，实验成功！

2.6 点灯实验三：通过 USB-CDC 方式控制 STC8H8K64U 进行 LED 点灯

2.6.1 原理介绍

STC8H8K64U 实验箱原理图



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。

如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。

如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。

如果 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。

如 P6.0 输出 0，LED60 灯就会被点亮；如 P6.0 输出 1，LED60 灯就会熄灭；
如 P6.1 输出 0，LED61 灯就会被点亮；如 P6.1 输出 1，LED61 灯就会熄灭；
如 P6.2 输出 0，LED62 灯就会被点亮；如 P6.2 输出 1，LED62 灯就会熄灭；
如 P6.3 输出 0，LED63 灯就会被点亮；如 P6.3 输出 1，LED63 灯就会熄灭；
如 P6.4 输出 0，LED64 灯就会被点亮；如 P6.4 输出 1，LED64 灯就会熄灭；
如 P6.5 输出 0，LED65 灯就会被点亮；如 P6.5 输出 1，LED65 灯就会熄灭；
如 P6.6 输出 0，LED66 灯就会被点亮；如 P6.6 输出 1，LED66 灯就会熄灭；
如 P6.7 输出 0，LED67 灯就会被点亮；如 P6.7 输出 1，LED67 灯就会熄灭

如 P6 口对外全部送 0，则这 8 个灯全亮；
如 P6 口对外全部送 1，则这 8 个灯全灭

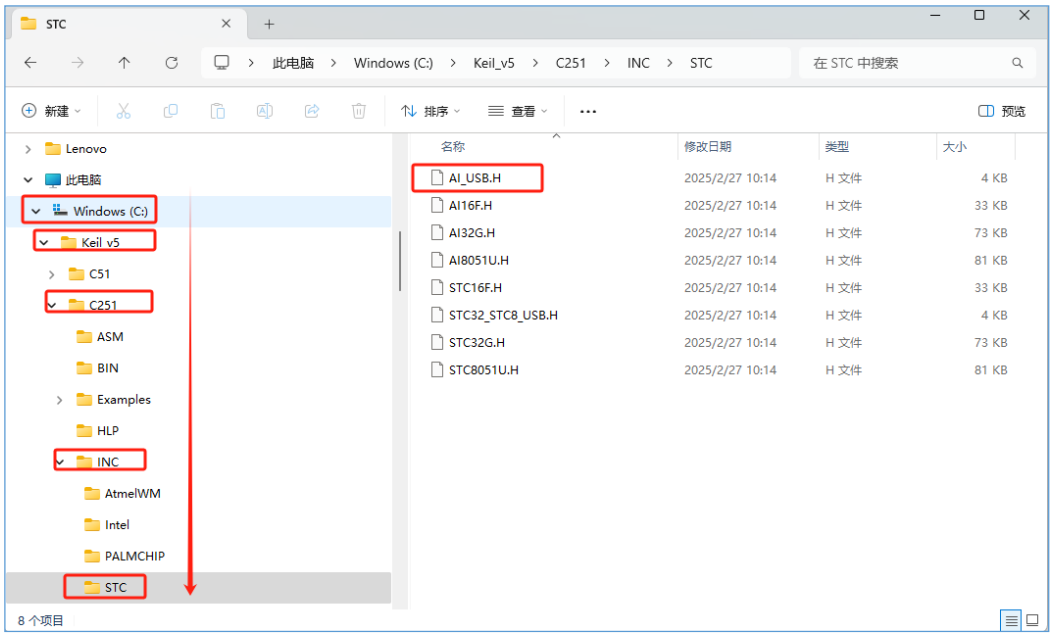
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P6 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

2.6.2　准备工作

1、添加常用头文件 ai_usb.h

#include "ai_usb.h"　　//使用 usb 所需要包含的头文件。

这个头文件在使用 AiCube-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



2、添加 usb_cdc 库函数文件

因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：

https://www.stcai.com/khs 的 USB 库文件中进行下载



下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，

在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_8h_data.LIB

【8 位的 8051 的 USB 库 可以用 BL51 连接器

BL51：　不支持 REMOVEUNUSED；BL51 支持 扩展中断号工具；

如果 不用 REMOVEUNUSED，可以 使用 BL51 ！

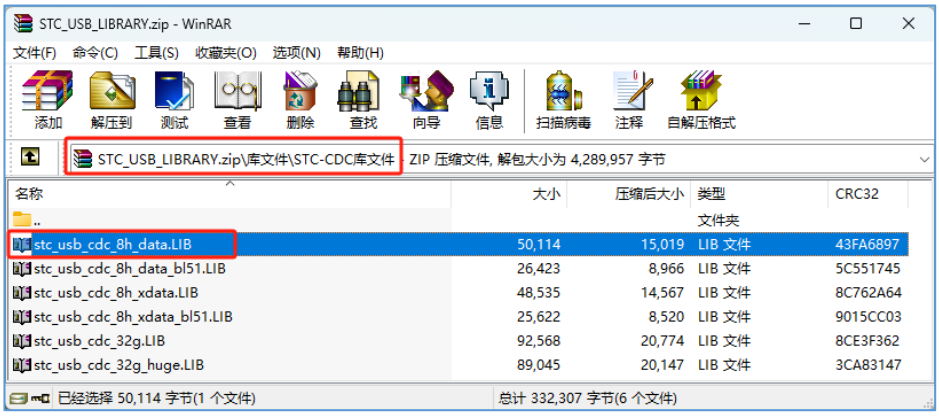
LX51：　支持 REMOVEUNUSED；LX51 不支持 扩展中断号工具；

如果 不用 扩展中断号工具，可以 使用 LX51 ！

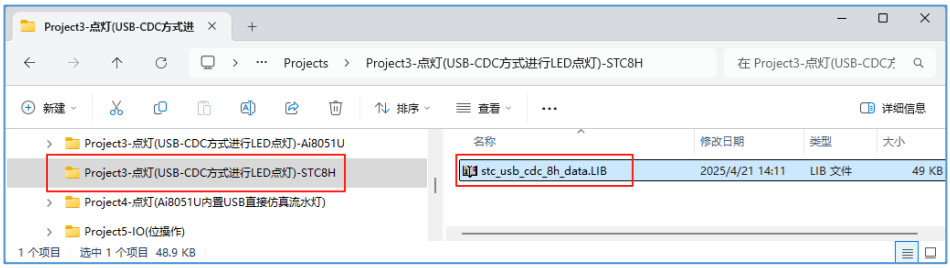
详情请参考本公司技术论坛链接：8 位的 8051 的 USB 库可以用 BL51；BL51 不支持 REMOVEUNUSED；

https://www.stcaimcu.com/thread-17105-1-1.html

(出处: 国芯人工智能技术交流网站)】

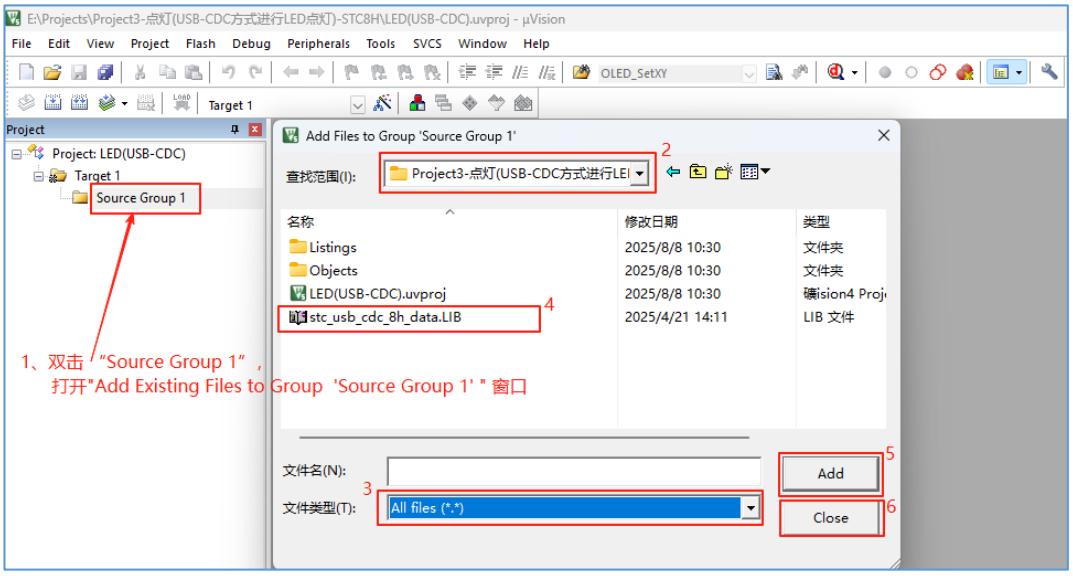


➲ 将库函数“stc_usb_cdc_8h_data.LIB”复制到创建工程所在的文件夹 E:\Projects\Project3-点灯(USB-CDC 方式进行 LED 点灯)-STC8H 中

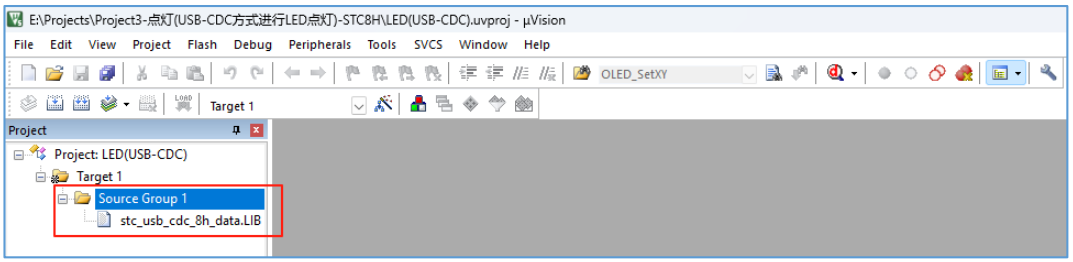


➲ 将库函数 stc_usb_cdc_8h_data.LIB 添加进入此工程项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_8h_data.LIB 添加进入此工程，就可以使用 USB-CDC 功能点亮 LED 灯。



这样，Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

2.6.3 代码部分

在工程项目的 main.c 文件中可输入如下代码：

```
//通过 USB-CDC 方式控制 STC8H8K64U 实验箱进行 LED 点灯
#include <STC8H.H>          //包含 STC8H 的头文件
#include "ai_usb.h"         //使用 usb 所需要包含的头文件，这个头文件在使用 AiCube-ISP 软件
                            //添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。

void main(void)
{
    BYTE dat;                           //定义变量保存接收的数据

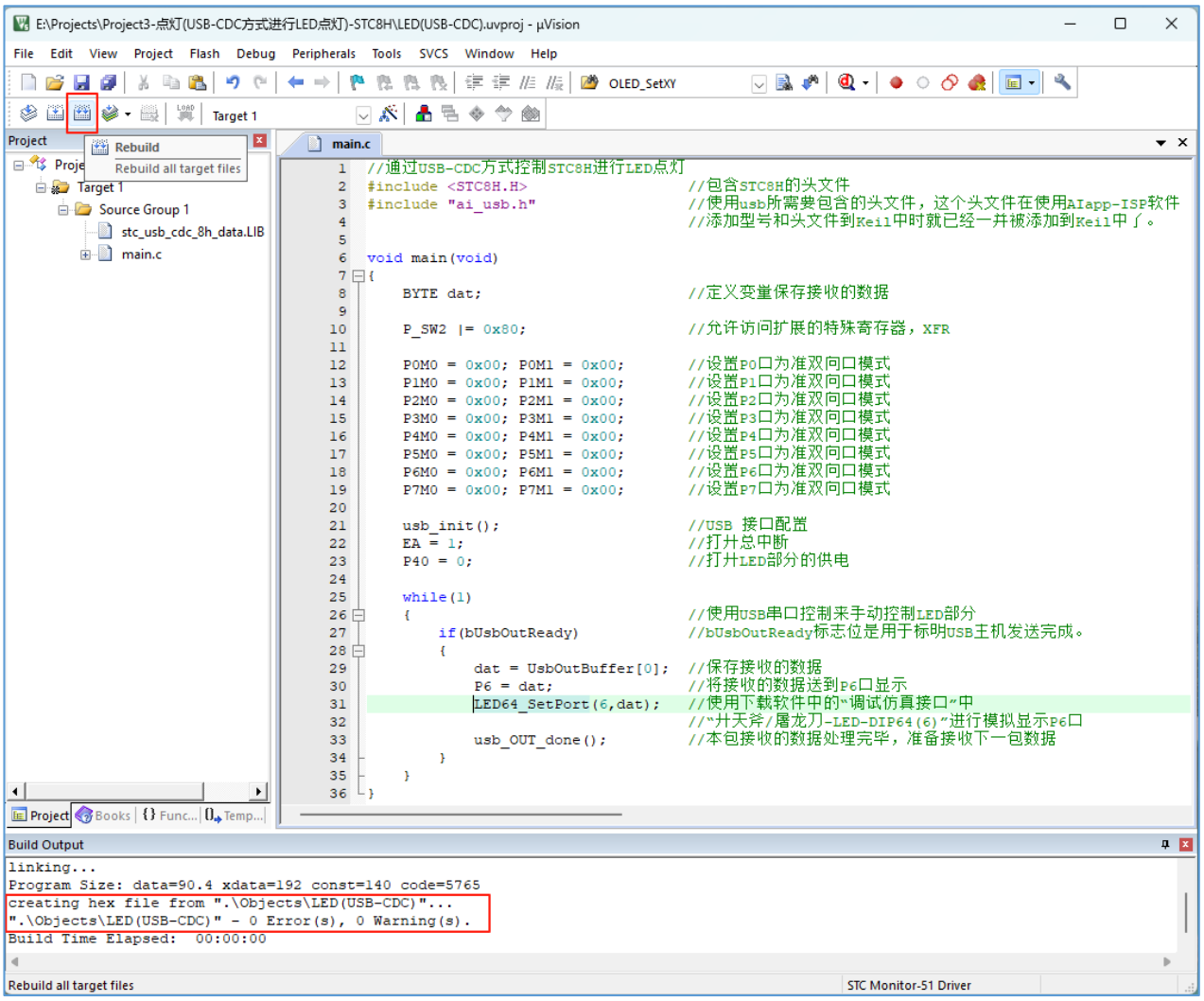
    P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;           //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;           //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;           //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;           //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;           //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;           //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;           //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;           //设置 P7 口为准双向口模式
//上面的将所有 I/O 口都初始化为准双向口工作模式的程序，
//可以使用 AiCube-ISP 软件的【I/O 口配置工具】来生成。

    usb_init();                         //USB 接口配置
    EA = 1;                             //打开总中断
    P40 = 0;                            //打开 LED 部分的供电

    while(1)
    {                                   //使用 USB 串口控制来手动控制 LED 部分
        if(bUsbOutReady)                //bUsbOutReady 标志位是用于标明 USB 主机发送完成。
        {
            dat = UsbOutBuffer[0];      //保存接收的数据
            P6 = dat;                   //将接收的数据送到 P6 口显示
            LED64_SetPort(6,dat);       //使用下载软件中的“调试仿真接口”中
                                        //“开天斧/屠龙刀-LED-DIP64(6)”进行模拟显示 P6 口
            usb_OUT_done();             //本包接收的数据处理完毕，准备接收下一包数据
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
        }
    }
}
```

代码编写完后，保存，点击工具栏上的 Rebuild 按钮进行编译



出现 creating hex file 和 0 Error(s) 0 Warning（s）即为编译成功，成功创建了 hex 文件且无报错。此 hex 文件，一般是在对应工程目录下的 Objects 文件夹里。

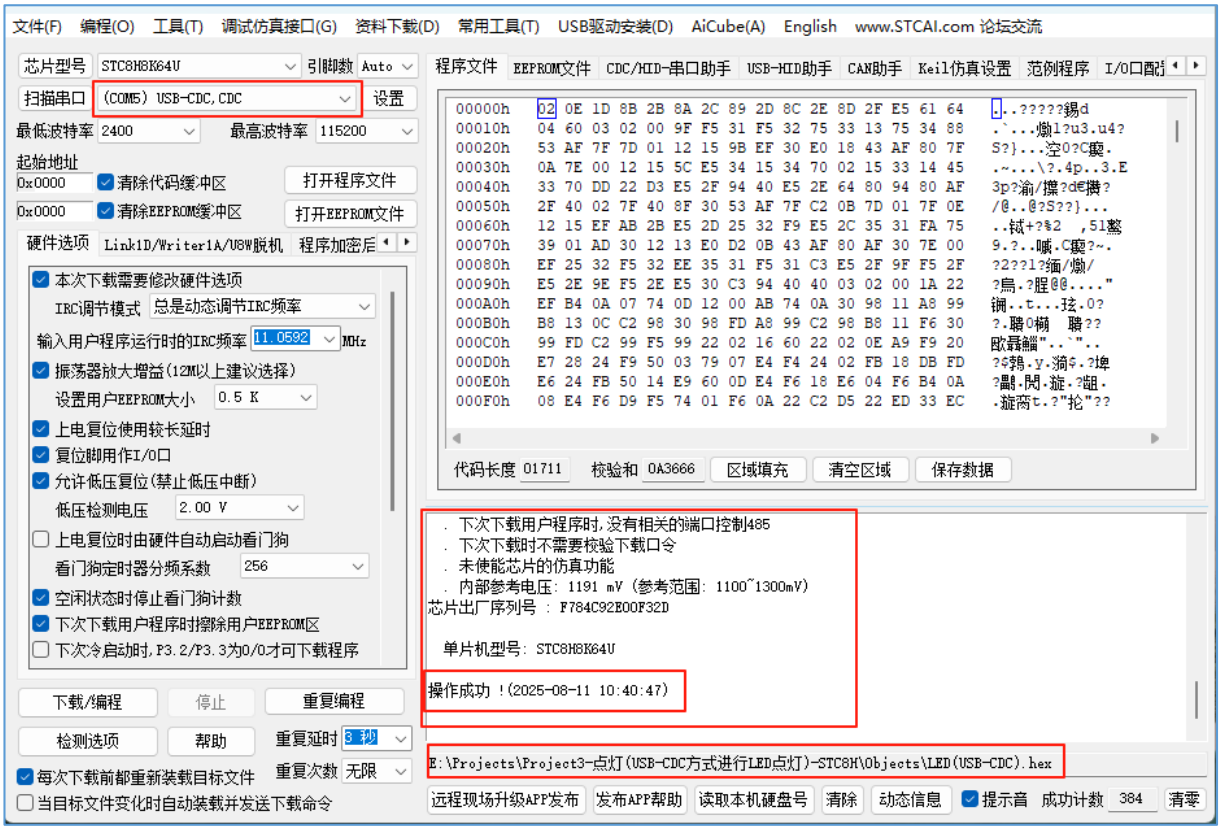
2.6.4　下载用户程序到 STC8H8K64U 实验箱并查看实验现象

录入代码，保存，编译无误。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤“打开程序文件”LED(USB-CDC).hex（“打开程序文件”是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中）并“下载/编程”，如下图：

【AiCube-ISP 软件的安装】，请点击查阅本文档 1.1 安装 AiCube-ISP 下载/编程/烧录 工具，含强大的辅助开发工具

【STC8H8K64U 实验箱实物图、与电脑的连接、用户程序下载流程】，请点击查阅本文档的 1.2 STC8H8K64U 实验箱实物图及程序下载

AiCube-ISP 软件界面出现：操作成功！
恭喜你！

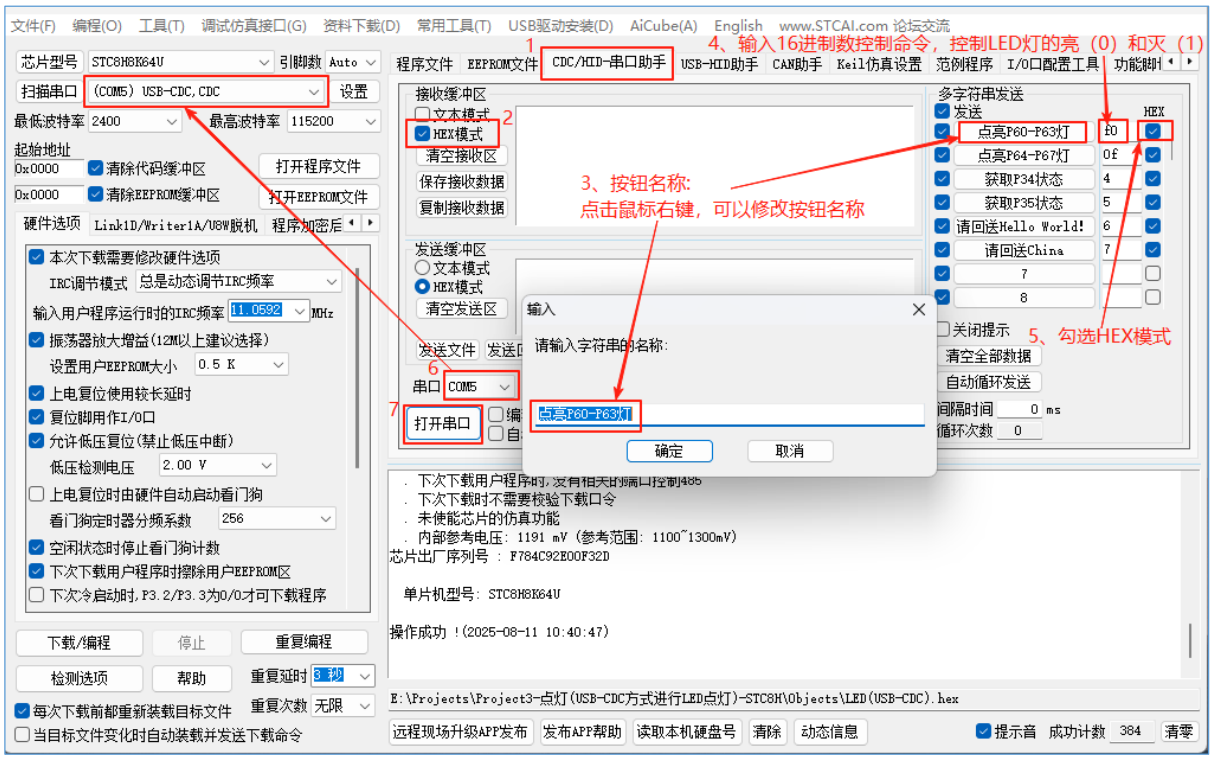


这时在 AiCube-ISP 软件界面，我们看到串口选项框中已经出现了一个(COM5)USB-CDC,CDC 串口，如果电脑系统并没有产生 CDC 端口，可能是因为系统版本较低（低于 Windows10 版本）。

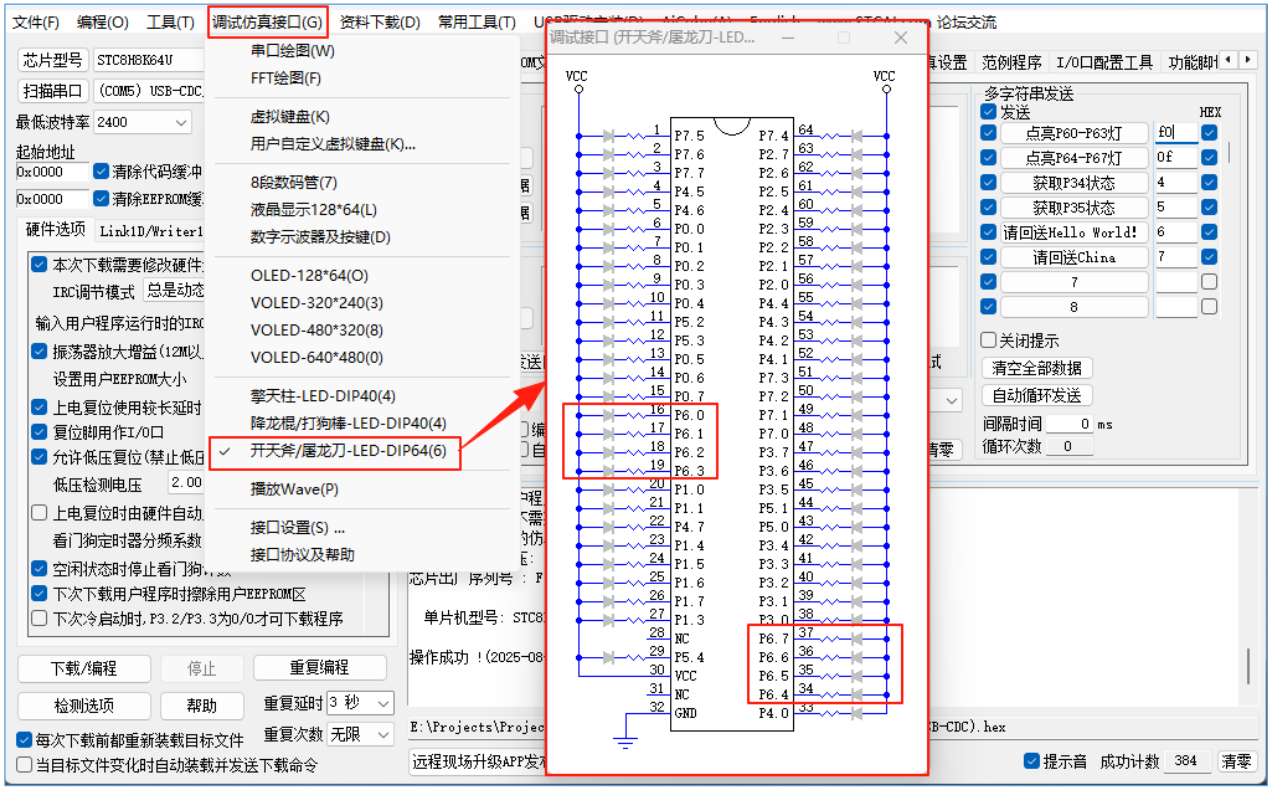
在 AiCube-ISP 软件界面，选择 CDC/HID-串口助手选项卡，然后按照下列步骤操作。

➲ 在“多字符串发送”窗口，输入 16 进制数控制命令并修改对应的按钮名称：
在输入窗口分别输入：f0（11110000），点亮 LED63 - P60 灯；
0f（00001111），点亮 LED67 - P64 灯。

➲ 选择与“扫描串口”中显示的 USB-CDC 相对应的串口号，并点击“打开串口”按钮。



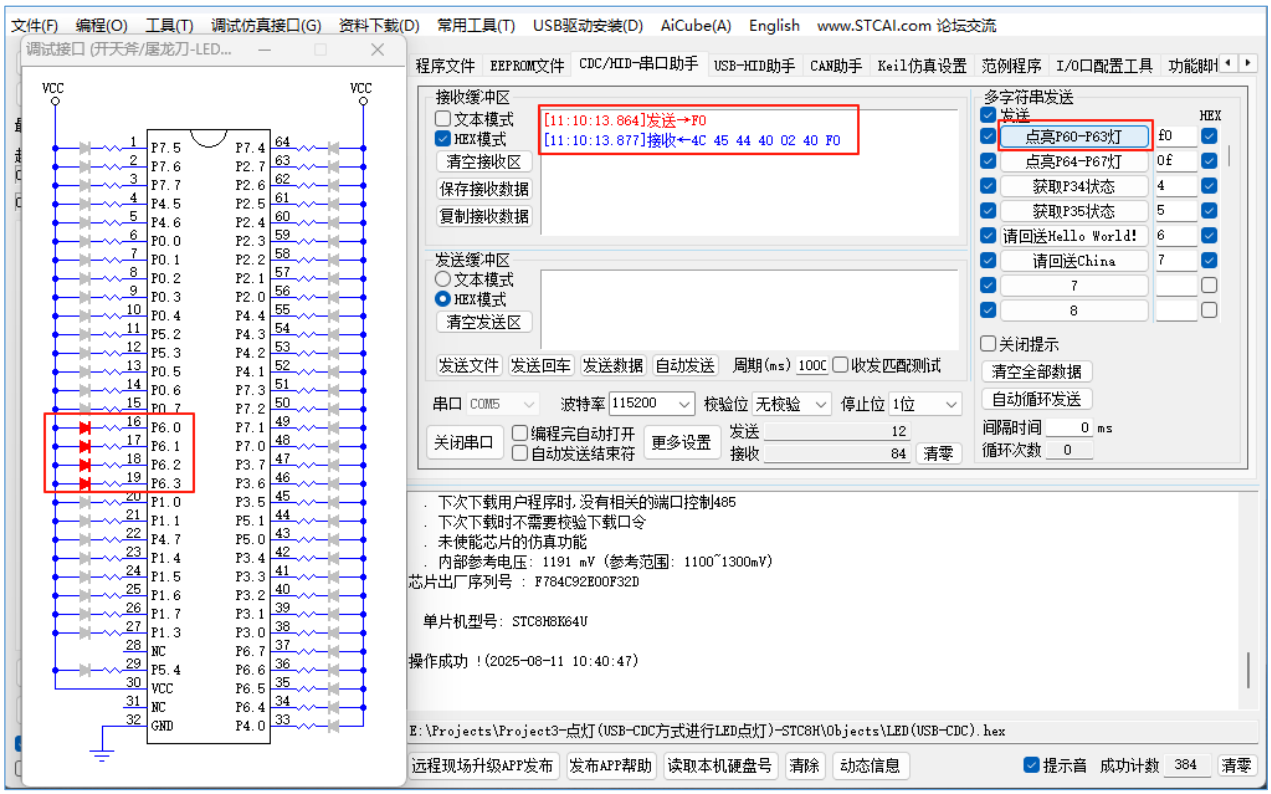
➲ 同时为了直观查看实验情况，可以打开 AiCube-ISP 软件的“调试仿真接口”菜单，选择“开天斧/屠龙刀-LED-DIP64(6)”进行模拟显示 P6 口：



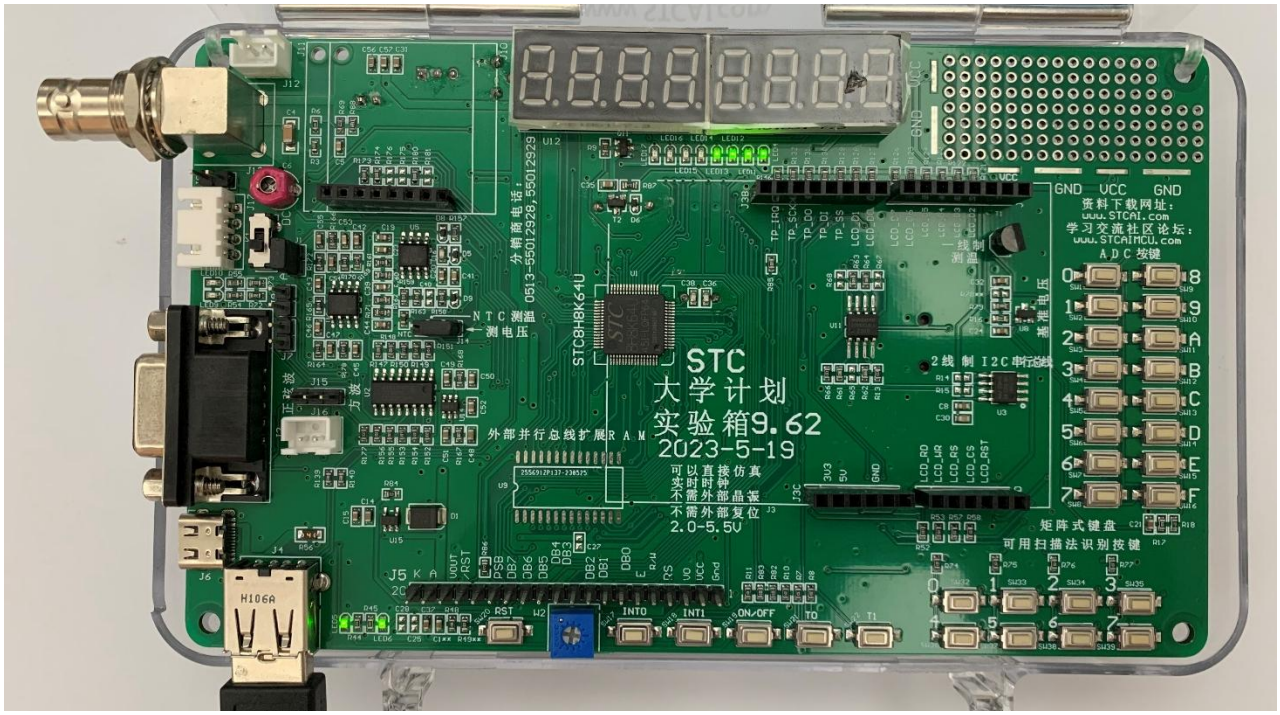
2.6.4.1　点击“多字符串发送”窗口中的【点亮 P60-P63 灯】按钮

1、观察 AiCube-ISP 软件的 CDC/HID-串口助手选项卡界面，如下图：

- 在接收缓冲区会出现发送、接收数据；
- “开天斧/屠龙刀-LED-DIP64(6)”浮窗上的 P6.0-P6.3 这 4 个虚拟灯被点亮



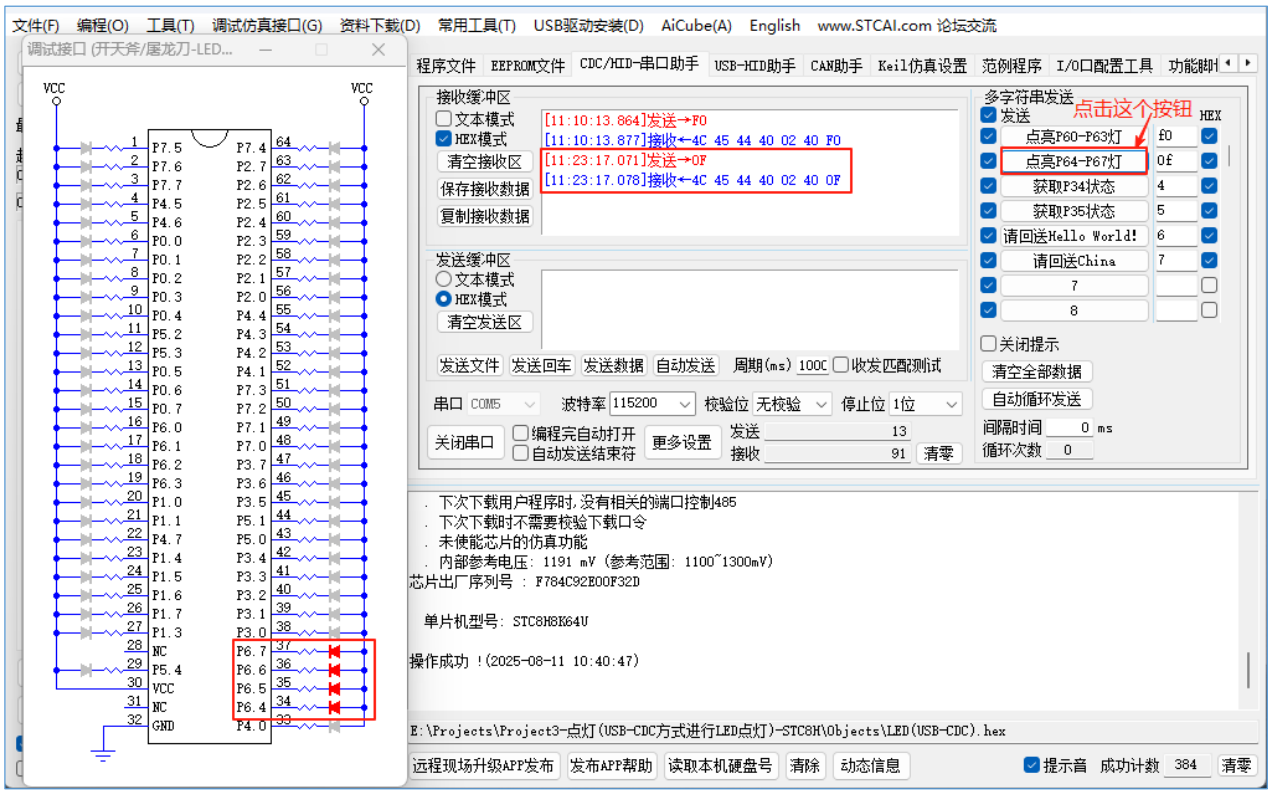
2、观察实验箱的 8 个 LED 灯，LED6.0 - LED6.3 这 4 个 LED 灯被点亮，如下图：



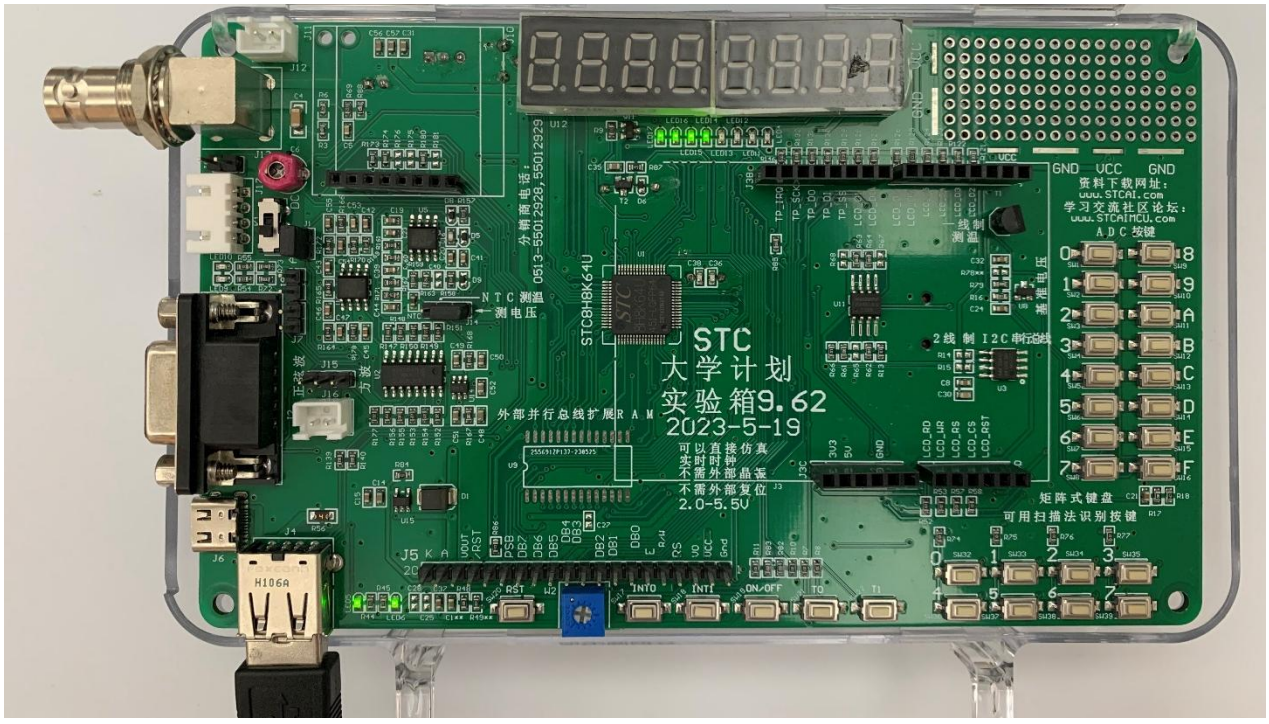
2.6.4.2　点击“多字符串发送”窗口中的【点亮 P64-P67 灯】按钮：

1、观察 AiCube-ISP 软件的 CDC/HID-串口助手选项卡界面，如下图：

- 在接收缓冲区会出现发送、接收数据；
- “开天斧/屠龙刀-LED-DIP64(6)”浮窗上的 P6.4-P6.7 这 4 个虚拟灯被点亮



2、观察实验箱的 8 个 LED 灯，LED6.4 - LED6.7 这 4 个 LED 灯被点亮，如下图：



实验成功！

2.6.5 使用【AiCube】项目创建助手工具创建通过 USB-CDC 方式控制 STC8H8K64U 进行 LED 点灯

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit、Ai8051U-8Bit、STC32G12K128、STC8H8K64U、STC15F2K60S2、STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

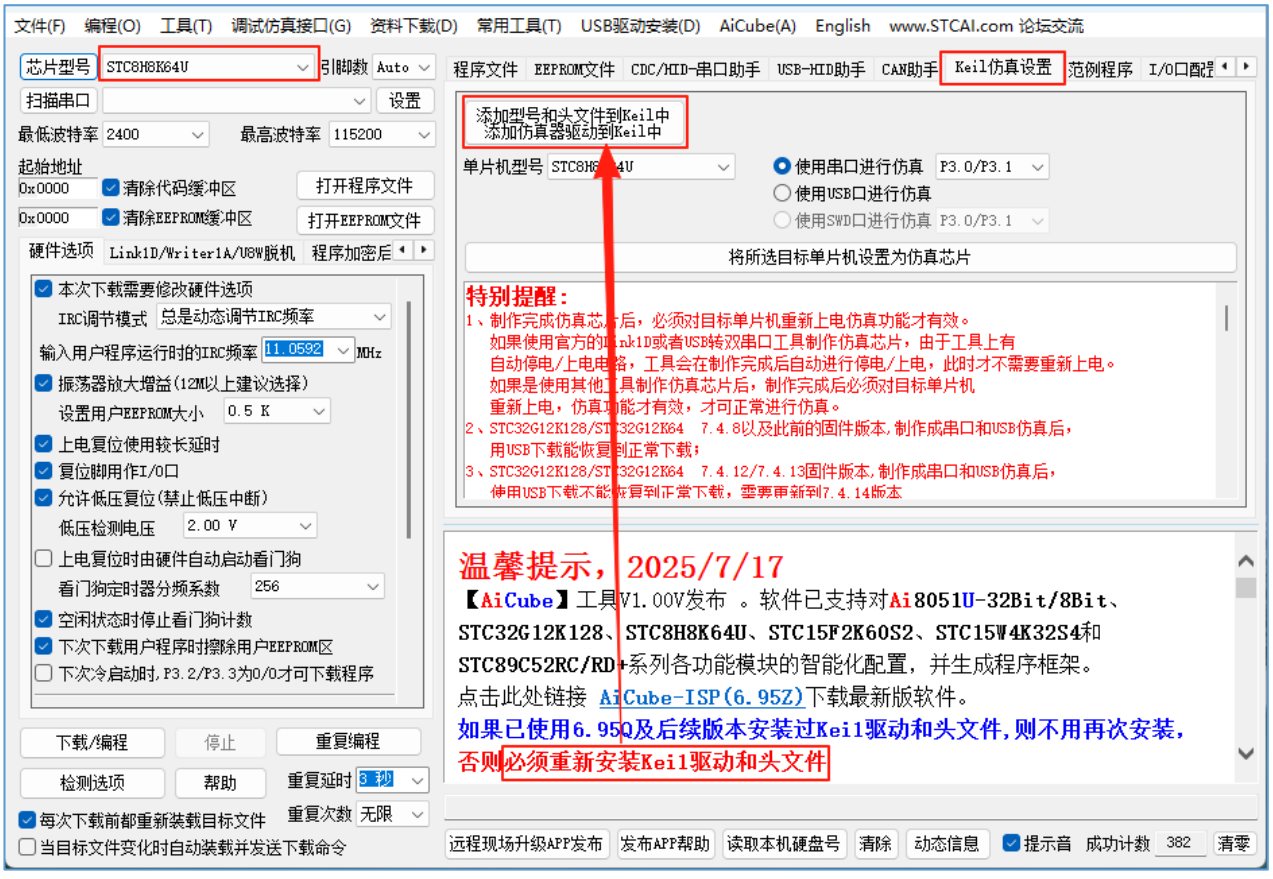
2.6.5.1 安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

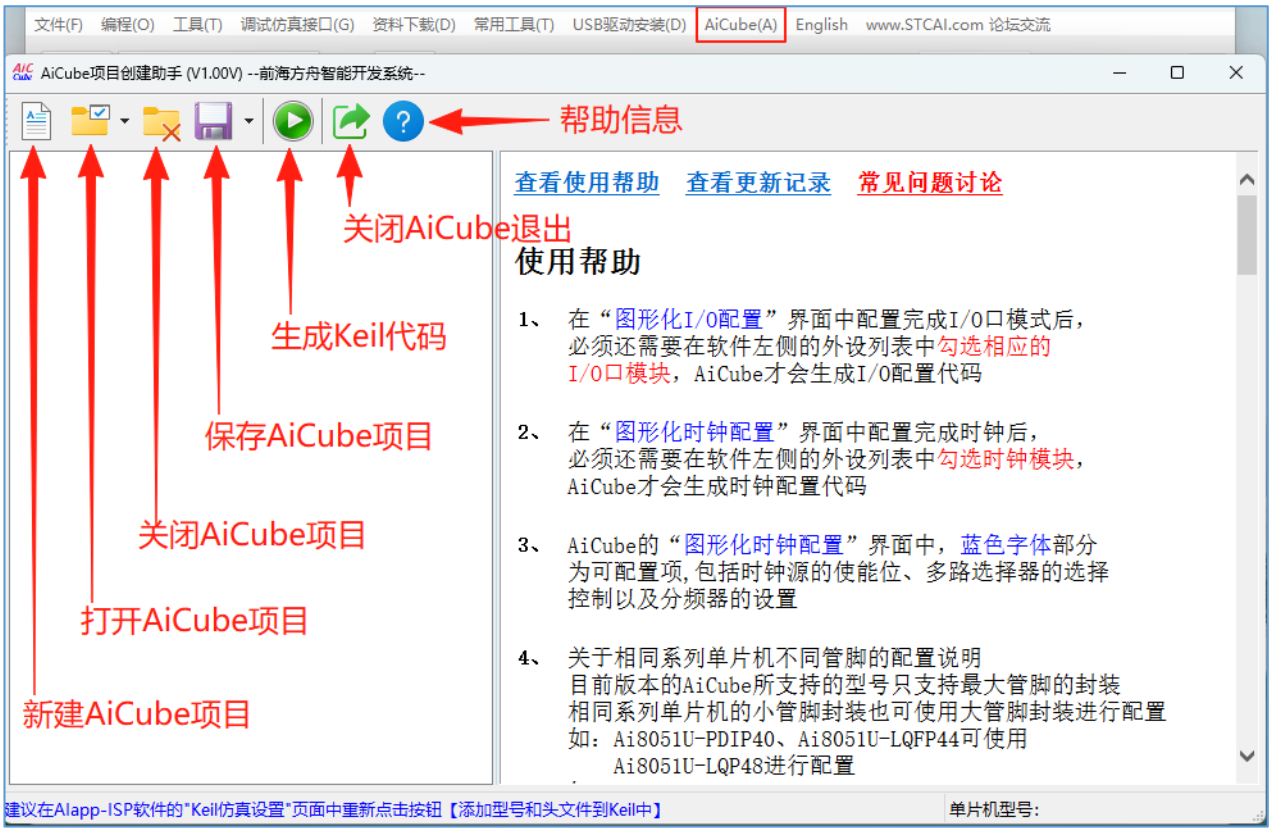
【AiCube-ISP 软件的安装】请点击查阅本文档 2.3 安装 AiCube-ISP 下载/编程/烧录 工具,含强大的辅助开发工具

2.6.5.2 使用【AiCube】助手创建项目

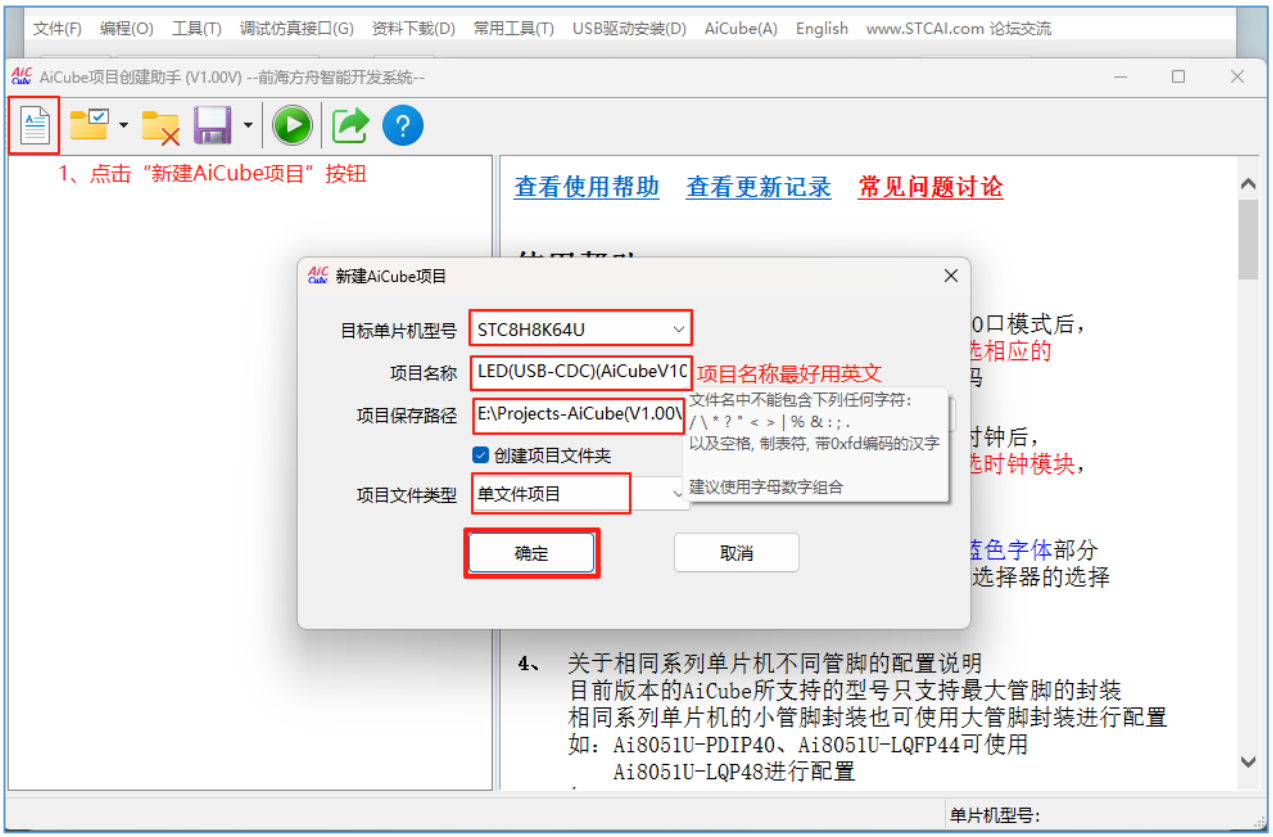
双击【AiCube-ISP】图标，打开 AiCube-ISP 软件界面。使用【AiCube】前，最好先“**重新安装 Keil 驱动和头文件**”如下图：



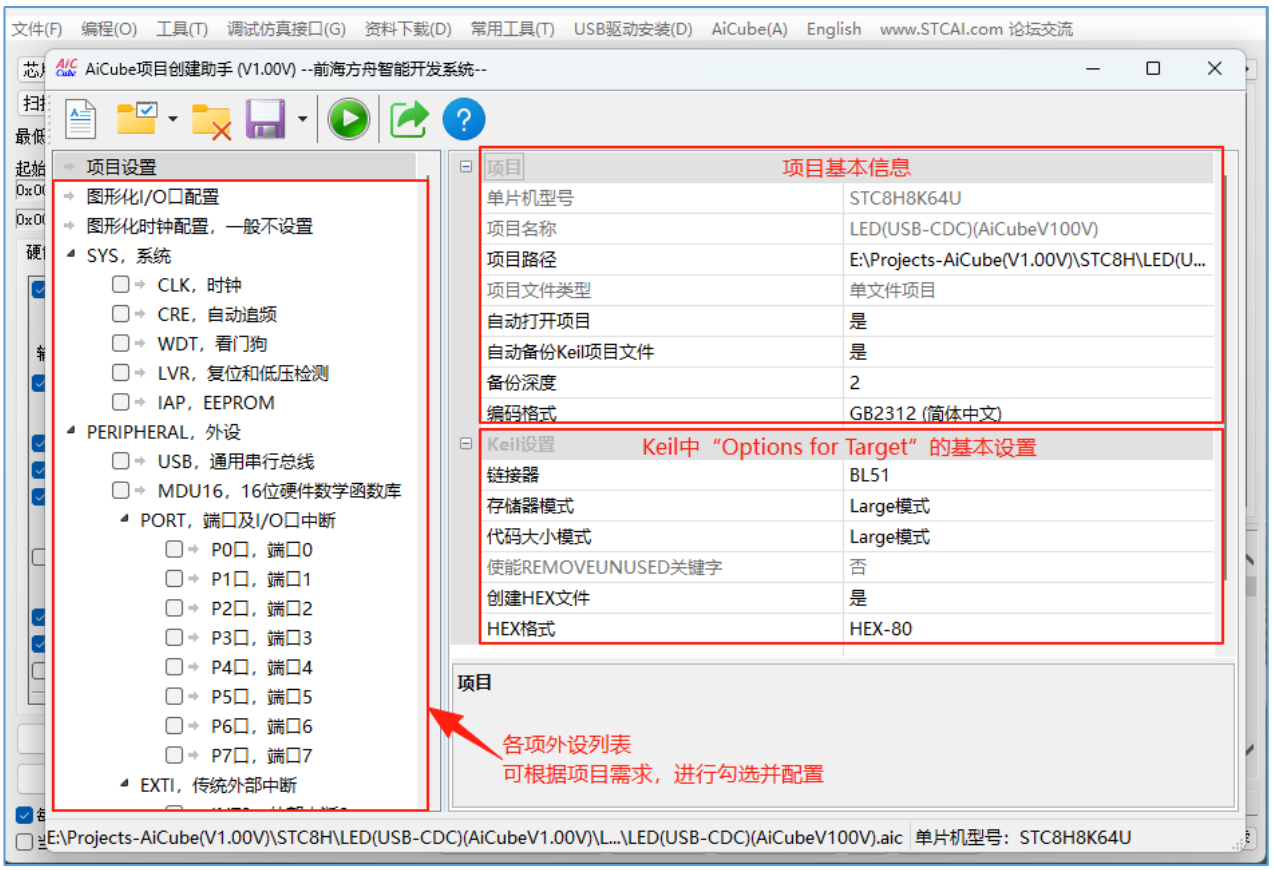
➲ **点击菜单栏的【AiCube（A）】，**弹出如下窗口：



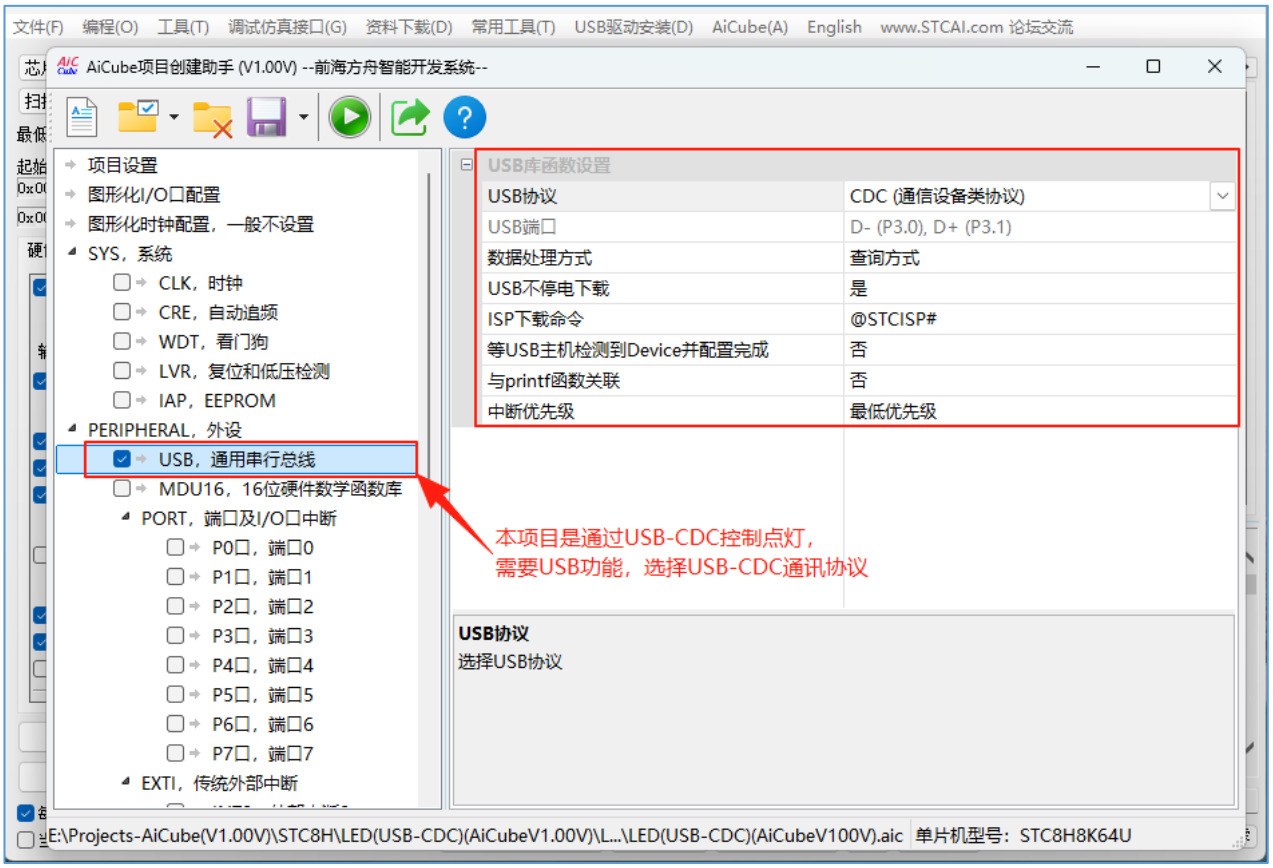
➲ **点击按钮“新建 AiCube 项目”，**出现如下浮窗，选择并填写项目基本信息，建议项目名称用英文！



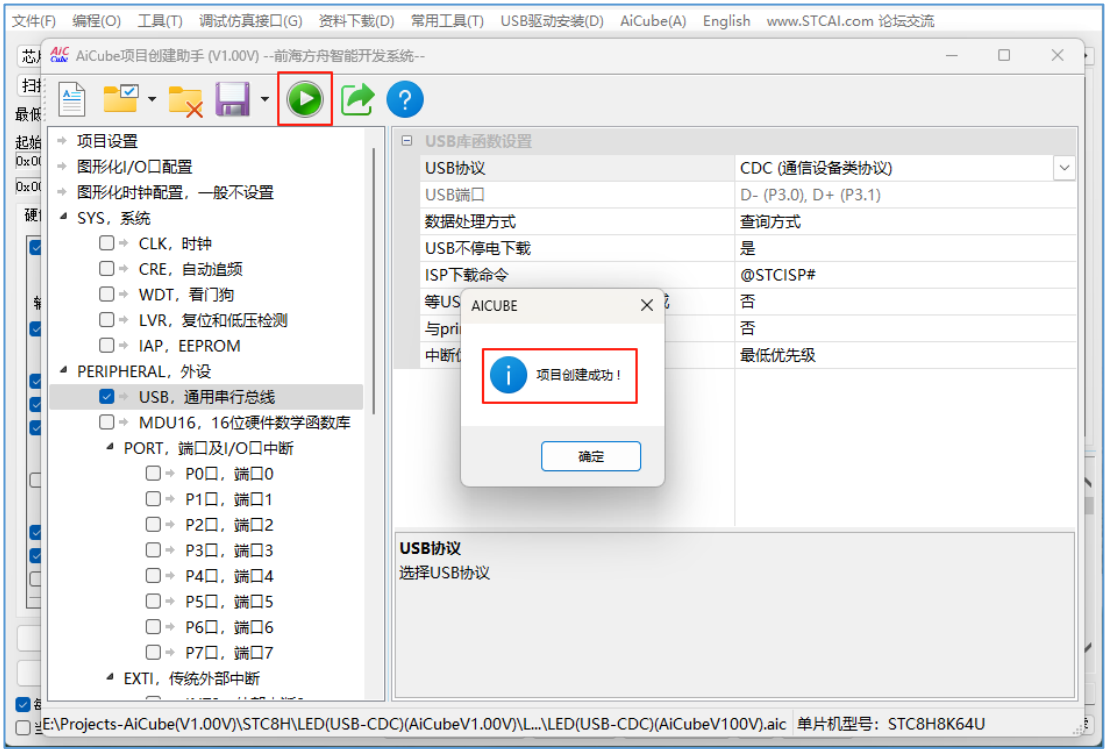
➲ 点击“确定”，出现如下配置界面：



因本项目用到USB 功能，要加载 USB 库函数、代码中要有 USB 初始化、USB-CDC 协议等，使用【AiCube】助手根据本项目的实际需求，对配置进行如下修改：



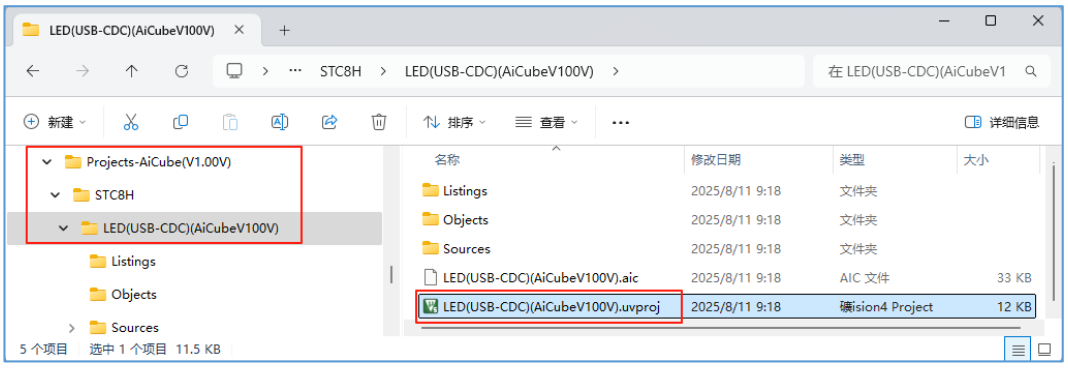
➲ **点击“创建 Keil 项目即程序文件”按钮**，【AiCube】助手提示：“项目创建成功！”，如下图：



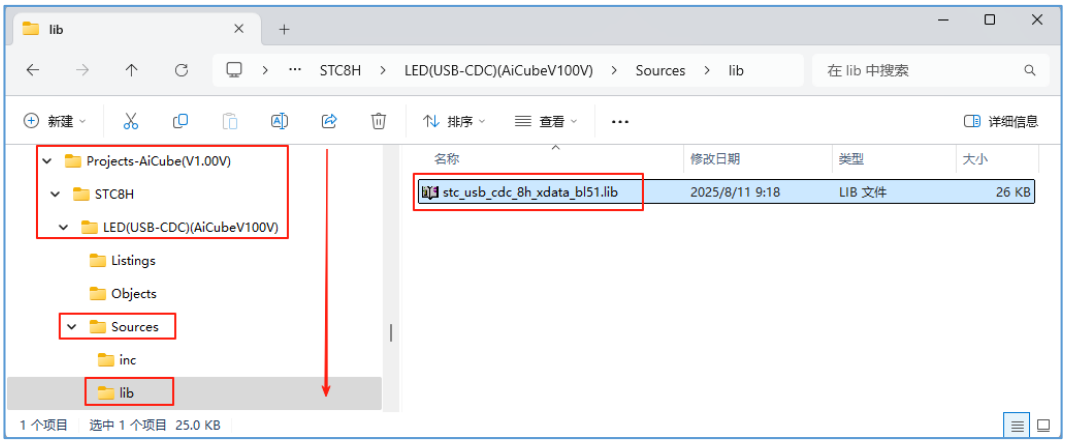
点击“确定”，立即自动打开 Keil 软件，进入该项目的 Keil 集成开发界面，即【AiCube】成功帮我们创建了项目及文件目录“LED(USB-CDC)(AiCubeV100V)”。至此，可以关闭【AiCube】。

➲ **找到并打开刚刚创建的项目目录**

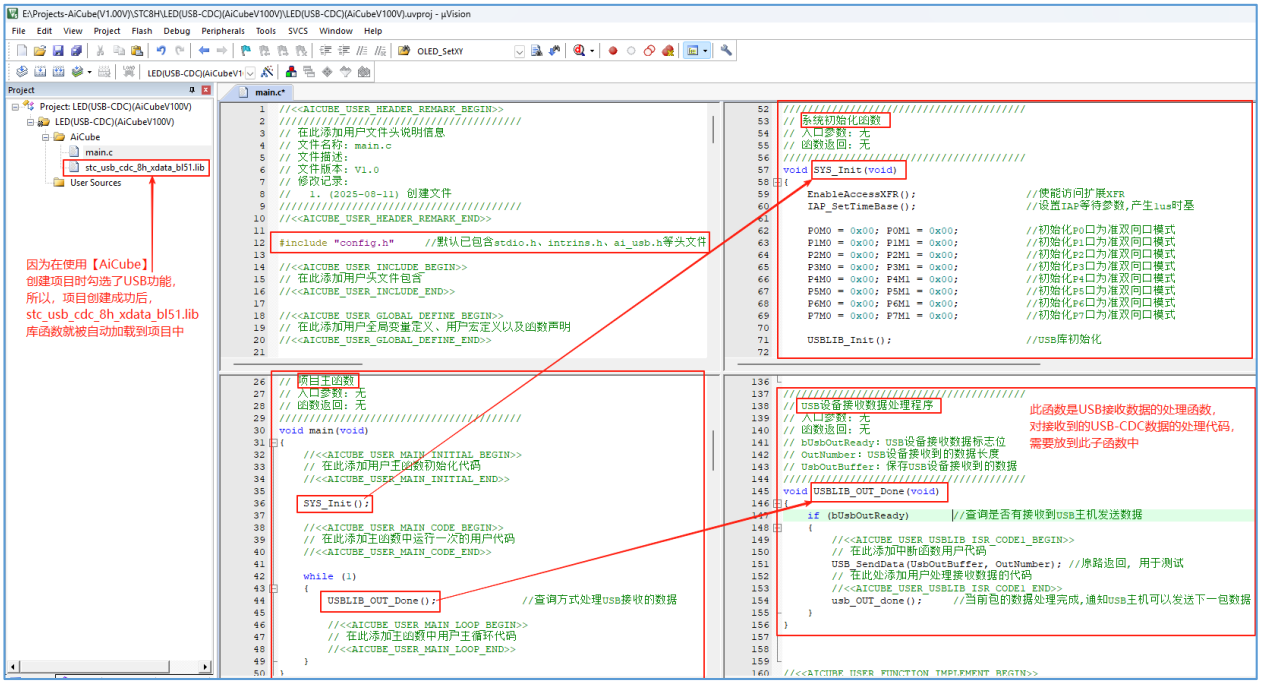
发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 LED(USB-CDC)(AiCubeV100V).uvproj 文件，如下图：



同时打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经给加载了：



➲ 观察已打开的 Keil 项目文件：LED(USB-CDC)(AiCubeV100V).uvproj，如下图：



根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中不仅有了 main.c，还已经自动加载了 USB 库函数（stc_usb_cdc_8h_xdata_bl51.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、毫秒和微秒延时函数、USB 相关的配置及处理函数。

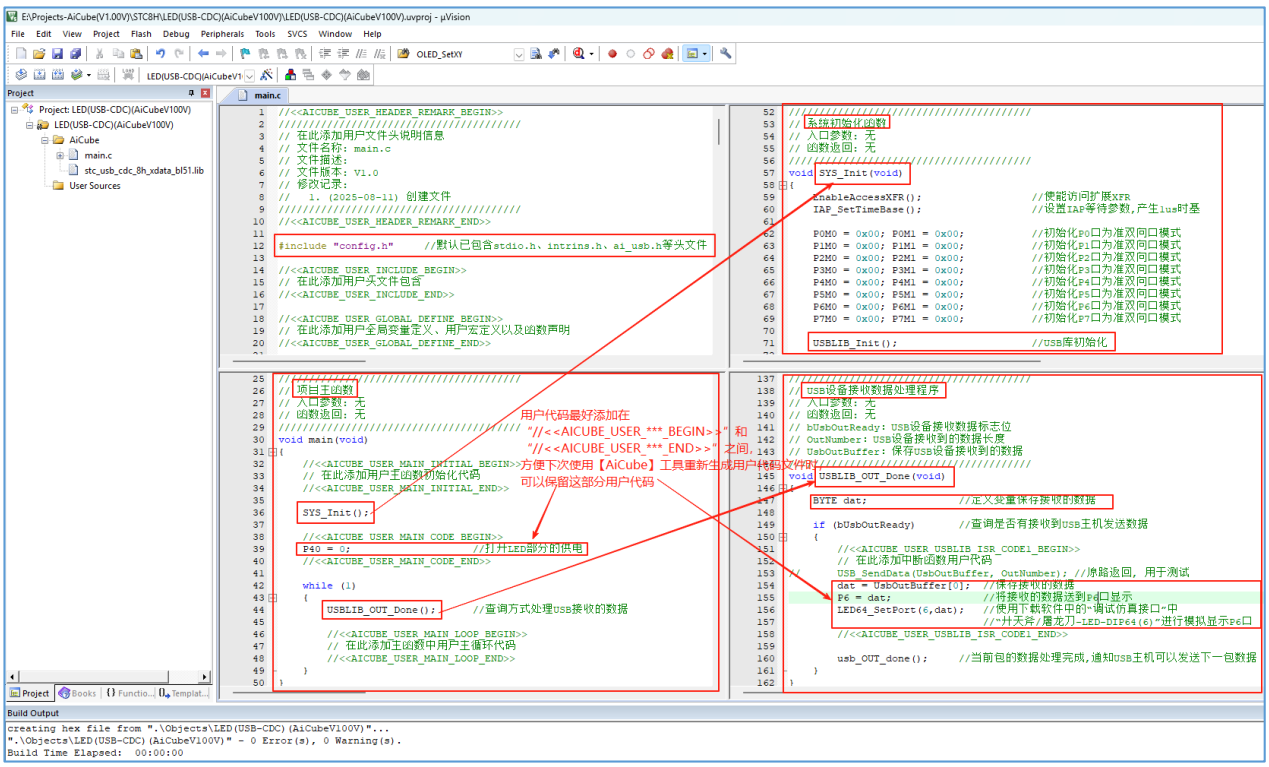
2.6.5.3　增加实验项目代码

为了实现 USB-CDC 方式控制 STC8H8K64U 进行 LED 点灯的功能，我们现在需要：

➲ 在主函数中加入“P40 = 0”打开 LED 供电的代码；

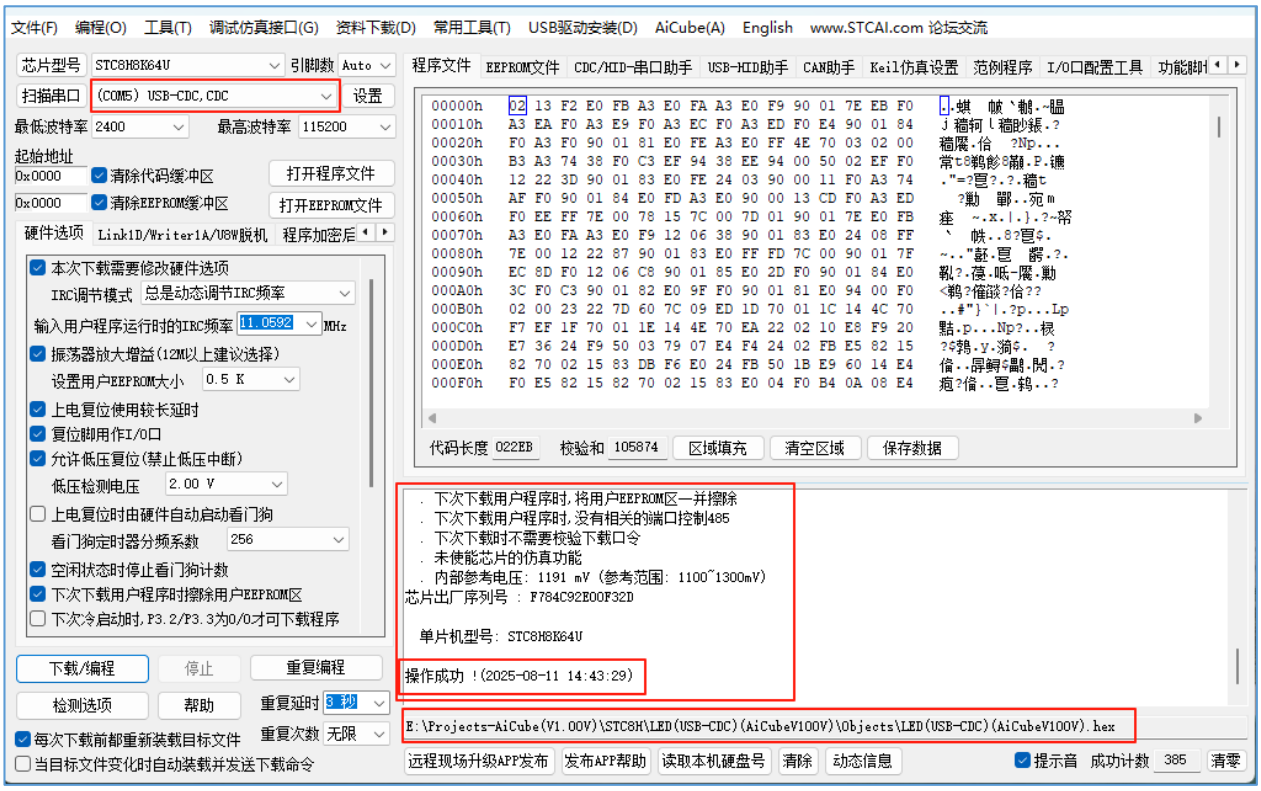
➲ 在“USB 设备接收数据处理程序”子函数中补充 USB 处理代码。

保存，编译，如下图（使用菜单“Window”→“Split”将原窗口分裂出 4 个窗口，对比观察）：



2.6.5.4　使用 AiCube-ISP 软件系统和 STC8H8K64U 实验箱查验实验效果

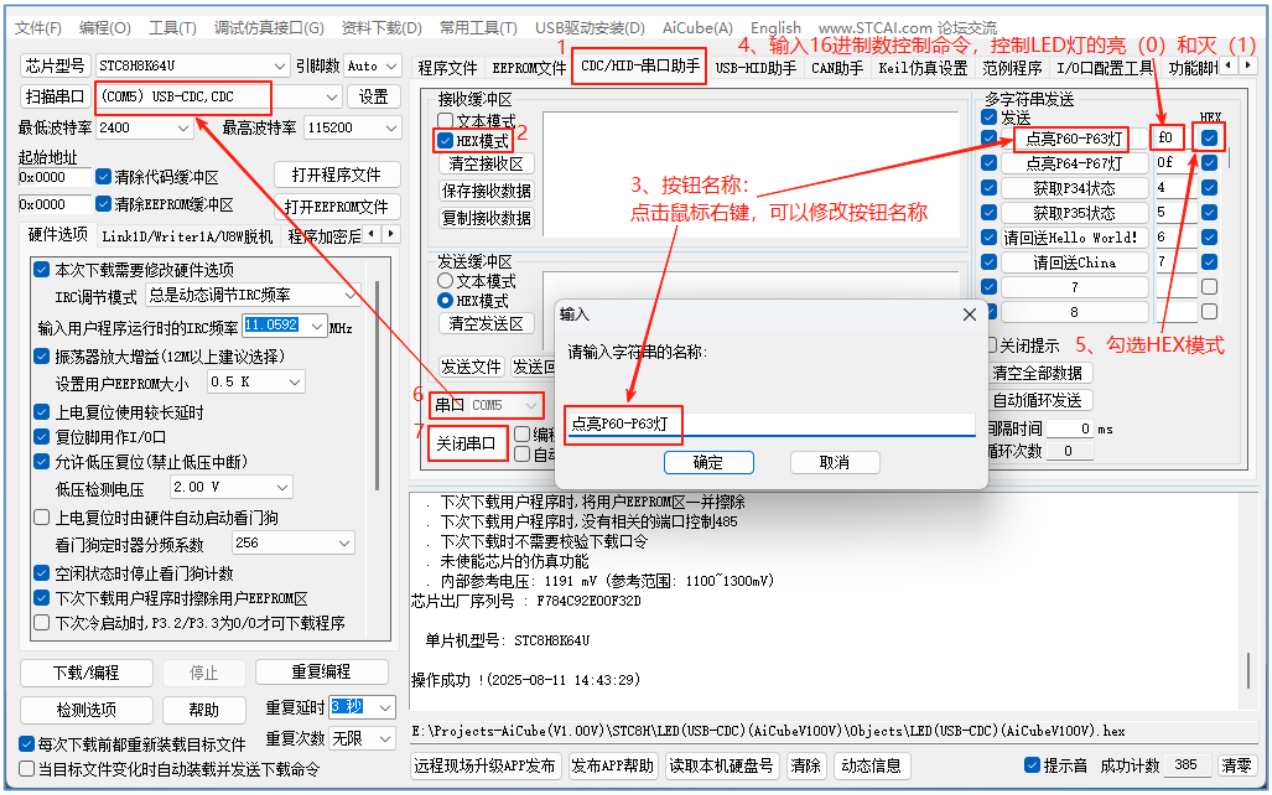
连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“LED(USB-CDC)(AiCubeV100V).hex”，如下图：



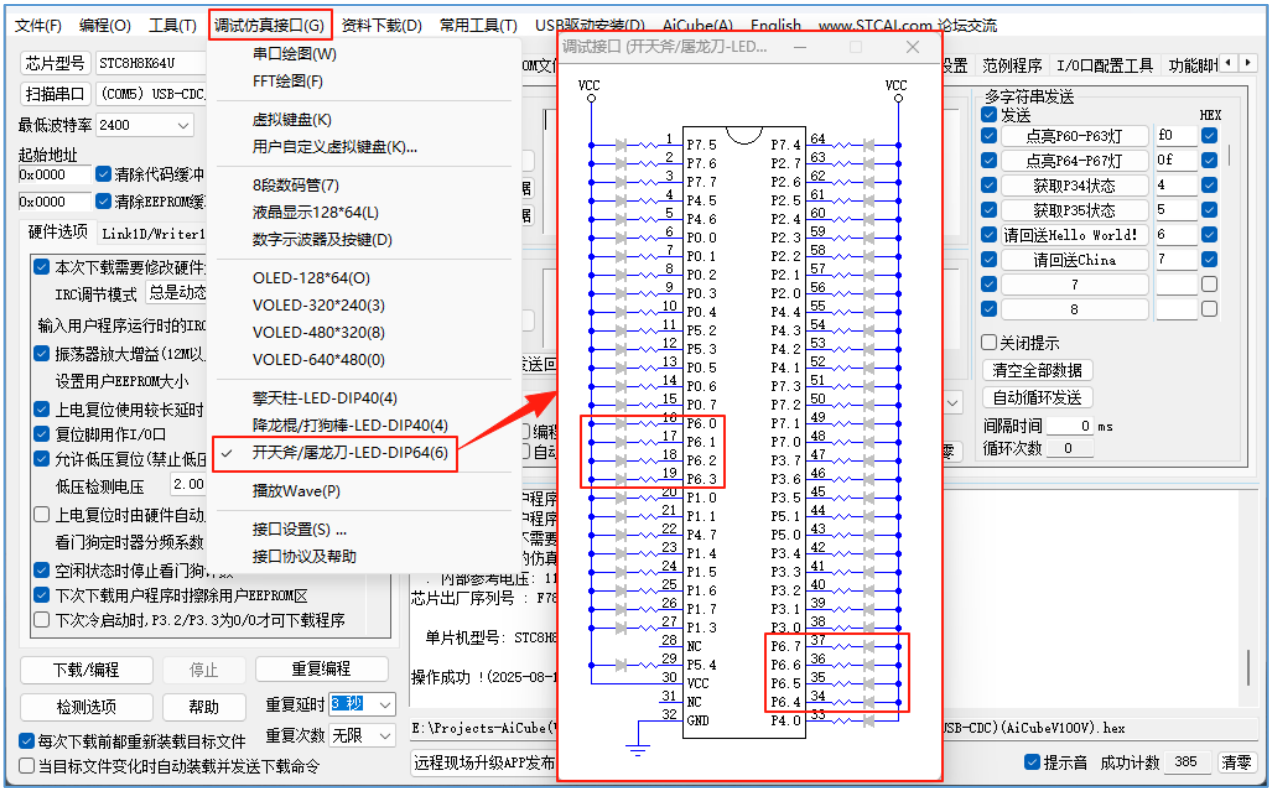
在 AiCube-ISP 软件界面，选择 CDC/HID-串口助手选项卡，然后按照下列步骤操作。

➲ 在“多字符串发送”窗口，输入 16 进制数控制命令并修改对应的按钮名称：
在控制命令窗口分别输入：f0（11110000），点亮 P63 - P60 灯；0f（00001111），点亮 P67 - P64 灯。

➲ 选择与“扫描串口”中显示的 USB-CDC 相对应的串口号，并点击“打开串口”按钮。



➲ 同时为了直观查看实验情况，可以打开 AiCube-ISP 软件的“调试仿真接口”菜单，选择“开天斧/屠龙刀-LED-DIP64(6)”进行模拟显示 P6 口：

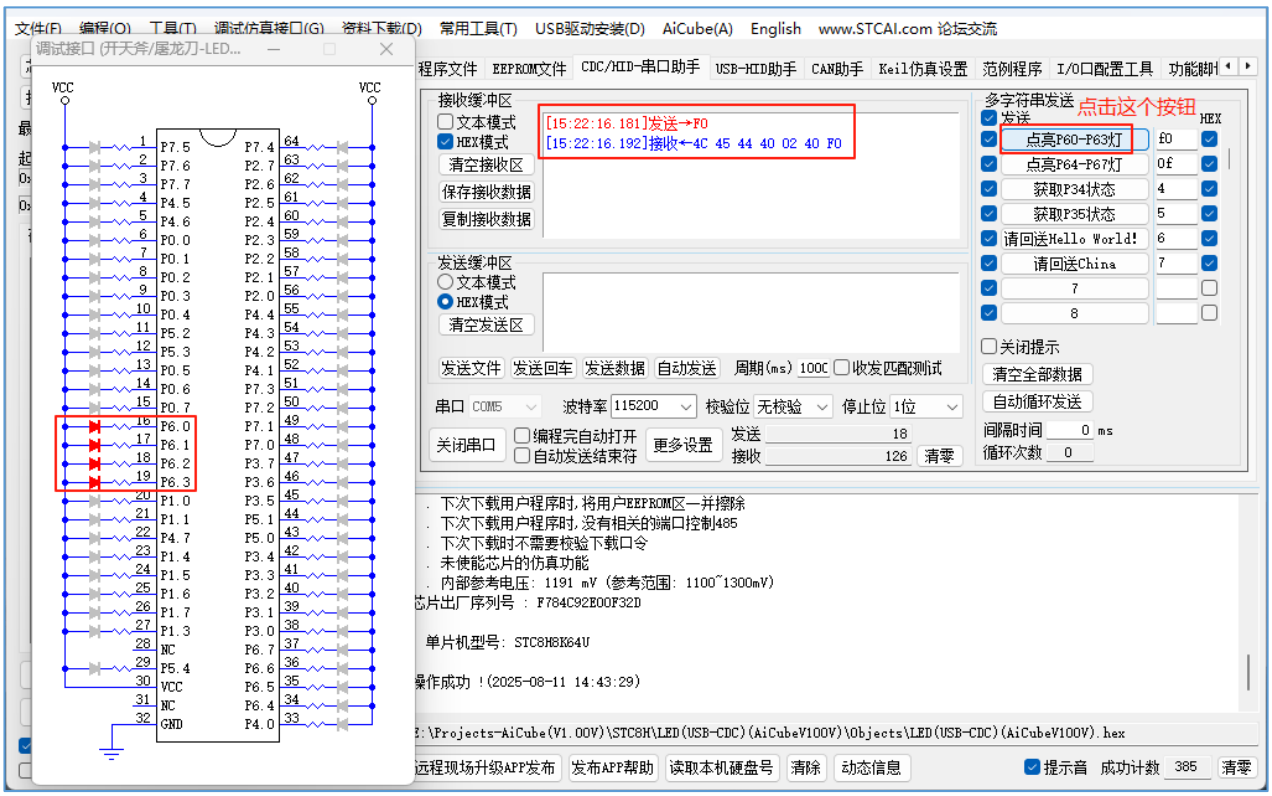


➲ 点击“多字符串发送”窗口中的【点亮 P60-P63 灯】按钮

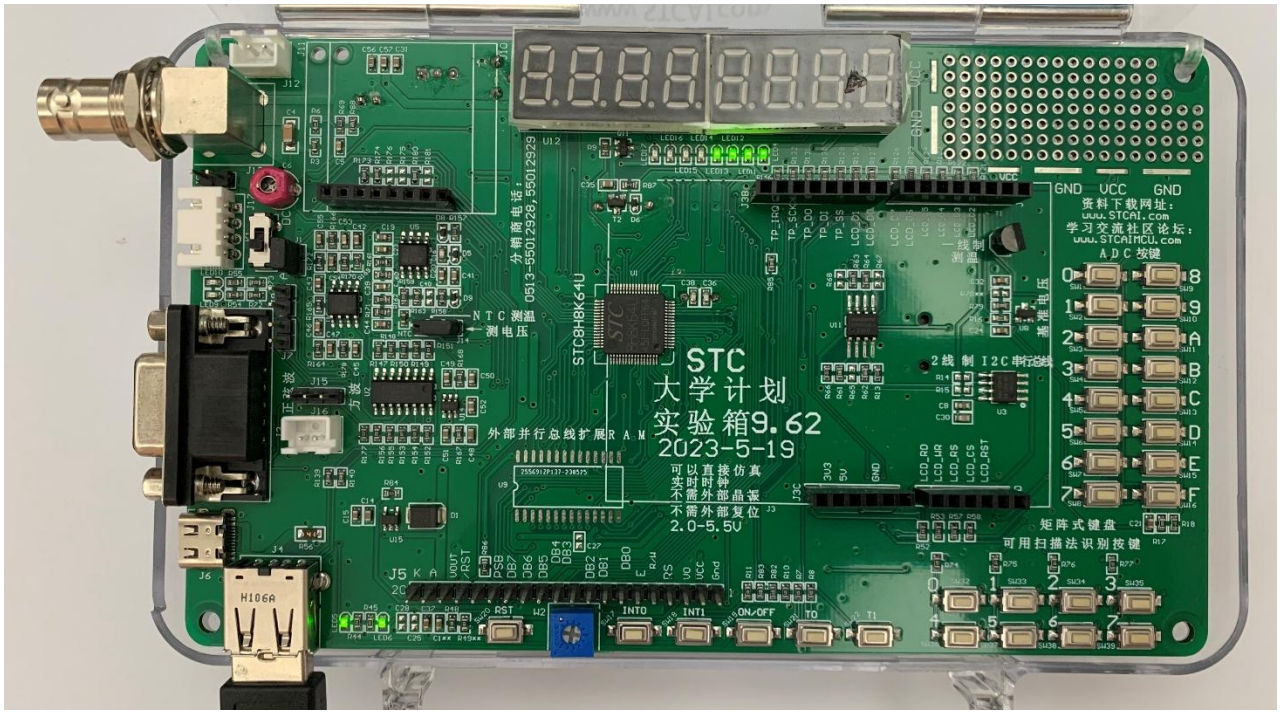
1、观察 AiCube-ISP 软件的 CDC/HID-串口助手选项卡界面，如下图：

➢ 在接收缓冲区会出现发送、接收数据；

➢ “开天斧/屠龙刀-LED-DIP64(6)”浮窗上的 P6.0-P6.3 这 4 个虚拟灯被点亮



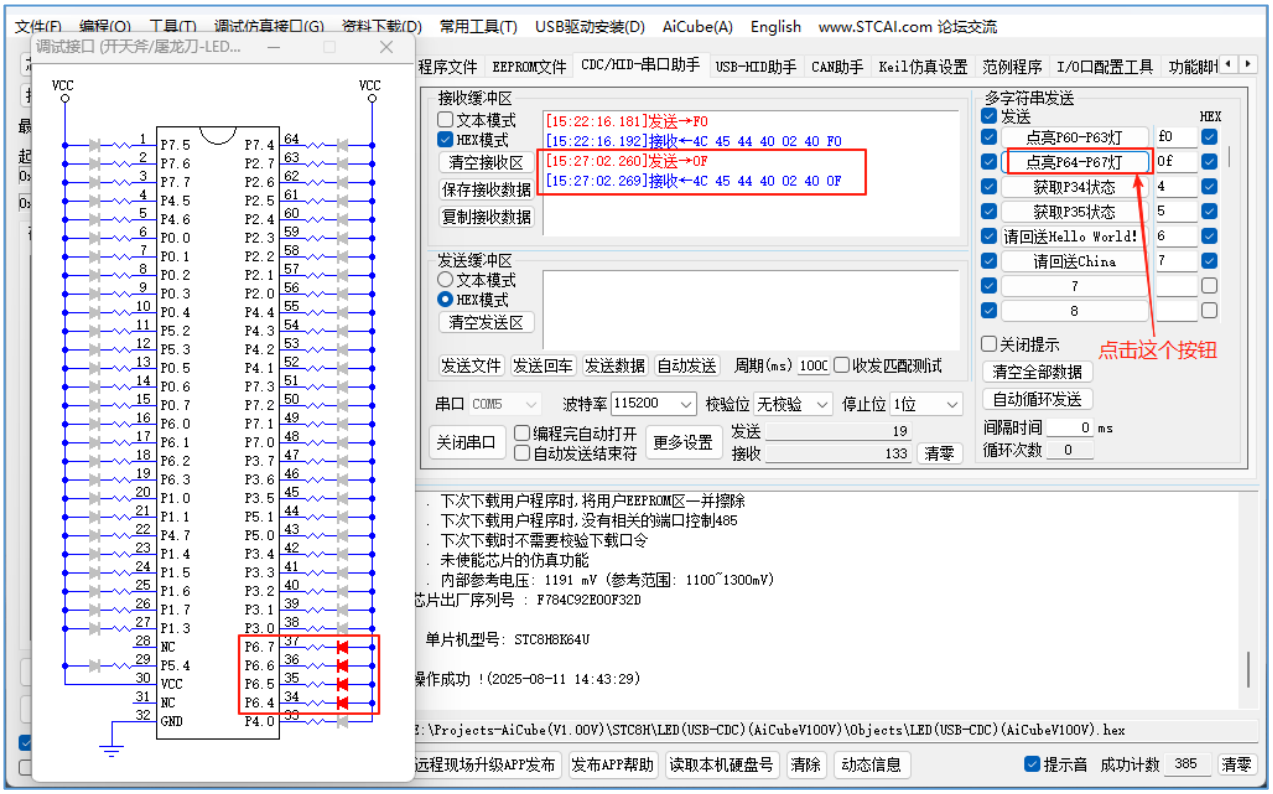
2、观察实验箱的 8 个 LED 灯，LED6.0 - LED6.3 这 4 个 LED 灯被点亮，如下图：



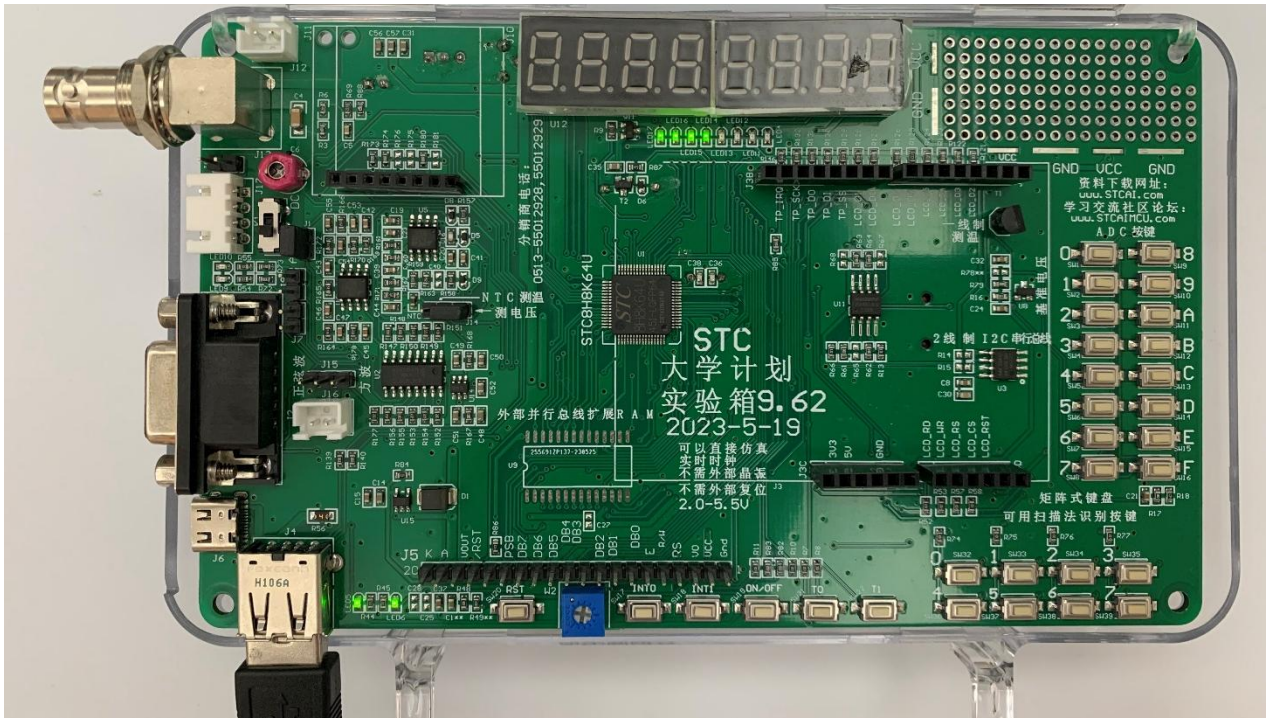
➲ 点击“多字符串发送”窗口中的【点亮 P64-P67 灯】按钮：

1、观察 AiCube-ISP 软件的 CDC/HID-串口助手选项卡界面，如下图：

- 在接收缓冲区会出现发送、接收数据；
- “开天斧/屠龙刀-LED-DIP64(6)”浮窗上的 P6.4-P6.7 这 4 个虚拟灯被点亮



2、观察实验箱的 8 个 LED 灯，LED6.4 - LED6.7 这 4 个 LED 灯被点亮，如下图：

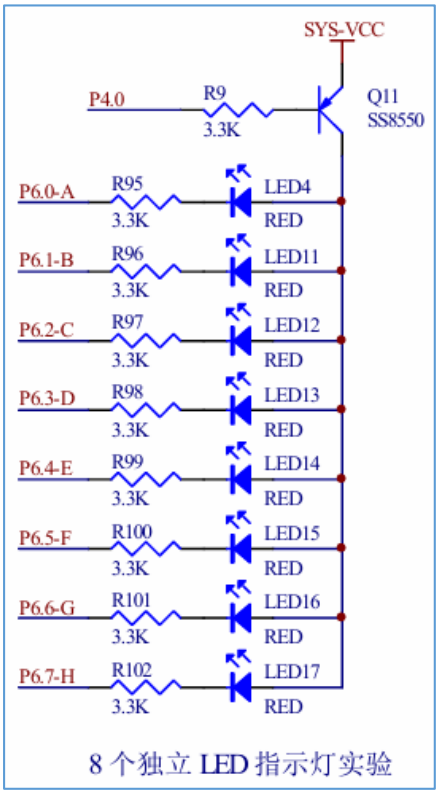


有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.7 点灯实验四：使用 STC8H8K64U 内置 USB 直接仿真流水灯

2.7.1 原理介绍

STC8H8K64U 实验箱原理图



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。

如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。

如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。

如果 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。

如 P6.0 输出 0，LED60 灯就会被点亮；如 P6.0 输出 1，LED60 灯就会熄灭；
如 P6.1 输出 0，LED61 灯就会被点亮；如 P6.1 输出 1，LED61 灯就会熄灭；
如 P6.2 输出 0，LED62 灯就会被点亮；如 P6.2 输出 1，LED62 灯就会熄灭；
如 P6.3 输出 0，LED63 灯就会被点亮；如 P6.3 输出 1，LED63 灯就会熄灭；
如 P6.4 输出 0，LED64 灯就会被点亮；如 P6.4 输出 1，LED64 灯就会熄灭；
如 P6.5 输出 0，LED65 灯就会被点亮；如 P6.5 输出 1，LED65 灯就会熄灭；
如 P6.6 输出 0，LED66 灯就会被点亮；如 P6.6 输出 1，LED66 灯就会熄灭；
如 P6.7 输出 0，LED67 灯就会被点亮；如 P6.7 输出 1，LED67 灯就会熄灭

如 P6 口对外全部送 0，则这 8 个灯全亮；
如 P6 口对外全部送 1，则这 8 个灯全灭

如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P6 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

2.7.2 实验程序 C 语言代码

```
//使用 STC8H8K64U 内置 USB 直接复制流水灯实验
#include <STC8H.H>          //包含 STC8H 的头文件
#include "intrins.h"        //使用_nop_()函数所必须要包含的头文件,
                            //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                            //会导致编译器找不到这个而函数而报错。
//unsigned char cnt;

void Delay500ms(void)       //延时 500 毫秒@11.0952MHz
{
    unsigned char data i, j, k;     //定义变量

    i = 29;
    j = 14;
    k = 54;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}

void main(void)
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    P40 = 0;                        //打开 LED 部分的供电
    while(1)
    {                               //实现 LED 流水灯
        P6 = 0;                     //对 P6 端口置 0，点亮整组 LED
        Delay500ms();

        P6 = 0xff;                  //对 P6 端口置 ff，全灭整组 LED
        Delay500ms();
```

```
            P60 = 0;                    //对单个端口 P60 置 0，点亮单个 LED，这个点亮是的 LED0
            Delay500ms();
            P60 = 1;                    //对单个端口 P60 置 1，熄灭单个 LED，这个熄灭是的 LED0
            Delay500ms();

            P61 = 0;                    //对单个端口 P61 置 0，点亮单个 LED，这个点亮是的 LED1
            Delay500ms();
            P61 = 1;                    //对单个端口 P61 置 1，熄灭单个 LED，这个熄灭是的 LED1
            Delay500ms();

            P62 = 0;                    //对单个端口 P62 置 0，点亮单个 LED，这个点亮是的 LED2
            Delay500ms();
            P62 = 1;                    //对单个端口 P62 置 1，熄灭单个 LED，这个熄灭是的 LED2
            Delay500ms();

            P63 = 0;                    //对单个端口 P63 置 0，点亮单个 LED，这个点亮是的 LED3
            Delay500ms();
            P63 = 1;                    //对单个端口 P63 置 1，熄灭单个 LED，这个熄灭是的 LED3
            Delay500ms();

            P64 = 0;                    //对单个端口 P64 置 0，点亮单个 LED，这个点亮是的 LED4
            Delay500ms();
            P64 = 1;                    //对单个端口 P64 置 1，熄灭单个 LED，这个熄灭是的 LED4
            Delay500ms();

            P65 = 0;                    //对单个端口 P65 置 0，点亮单个 LED，这个点亮是的 LED5
            Delay500ms();
            P65 = 1;                    //对单个端口 P65 置 1，熄灭单个 LED，这个熄灭是的 LED5
            Delay500ms();

            P66 = 0;                    //对单个端口 P66 置 0，点亮单个 LED，这个点亮是的 LED6
            Delay500ms();
            P66 = 1;                    //对单个端口 P66 置 1，熄灭单个 LED，这个熄灭是的 LED6
            Delay500ms();

            P67 = 0;                    //对单个端口 P67 置 0，点亮单个 LED，这个点亮是的 LED7
            Delay500ms();
            P67 = 1;                    //对单个端口 P67 置 1，熄灭单个 LED，这个熄灭是的 LED7
            Delay500ms();

//          P6 = (1<<(cnt%8));          //将对应亮起的位赋值给 P6 口
//          cnt++;                      //每次步进 1
//          Delay500ms();
        }
}
```

【小技巧--灵活运用 AiCube-ISP 软件的工具自动生成代码】

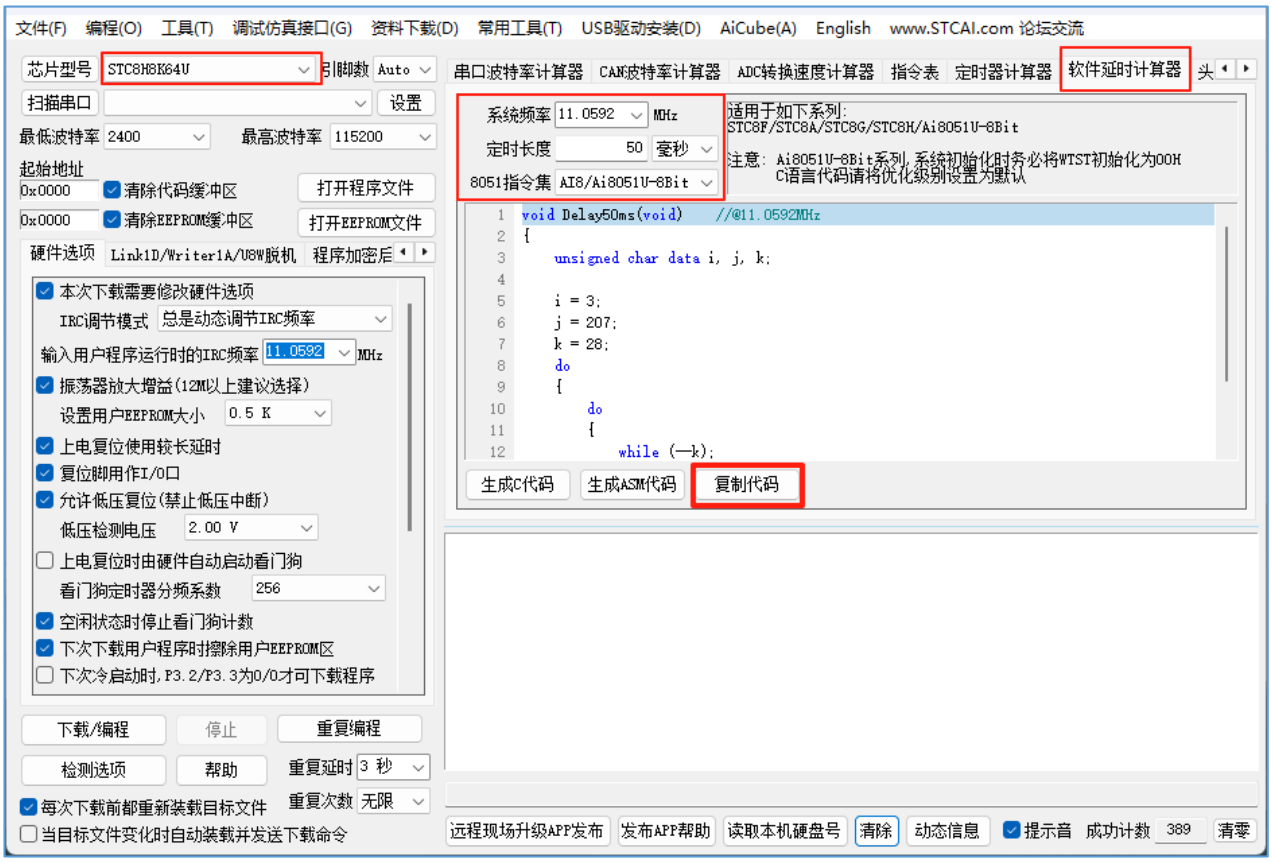
延时函数部分代码就可以使用 **AiCube-ISP** 软件的“**软件延时计算器**”来生成。

例如上述代码中这一段延时函数的代码：

```
void Delay500ms(void)		//延时 500 毫秒@11.0952MHz
{
	unsigned char data i, j, k;	//定义变量

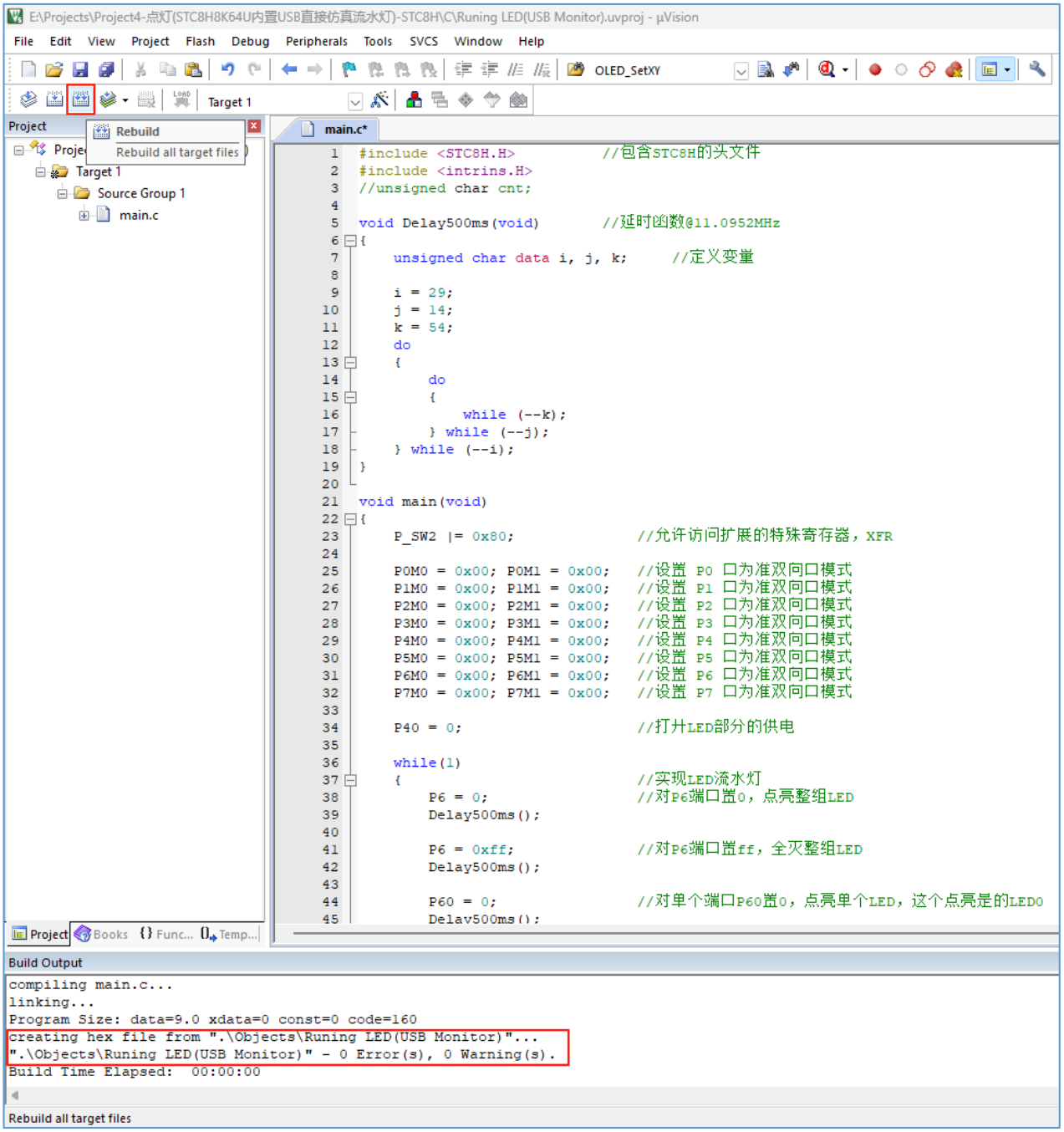
	i = 29;
	j = 14;
	k = 54;
	do
	{
		do
		{
			while (--k);
		} while (--j);
	} while (--i);
}
```

就可以使用 AiCube-ISP 软件的“软件延时计算器”来生成：



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

代码编写完后，保存，点击工具栏上的【Rebuild】按钮进行编译，如下图：



出现 creating hex file 和 0 Error(s) 0 Warning（s）即为编译成功，成功创建了 hex 文件且无报错。此 hex 文件，一般是在对应工程目录下的 Objects 文件夹里。

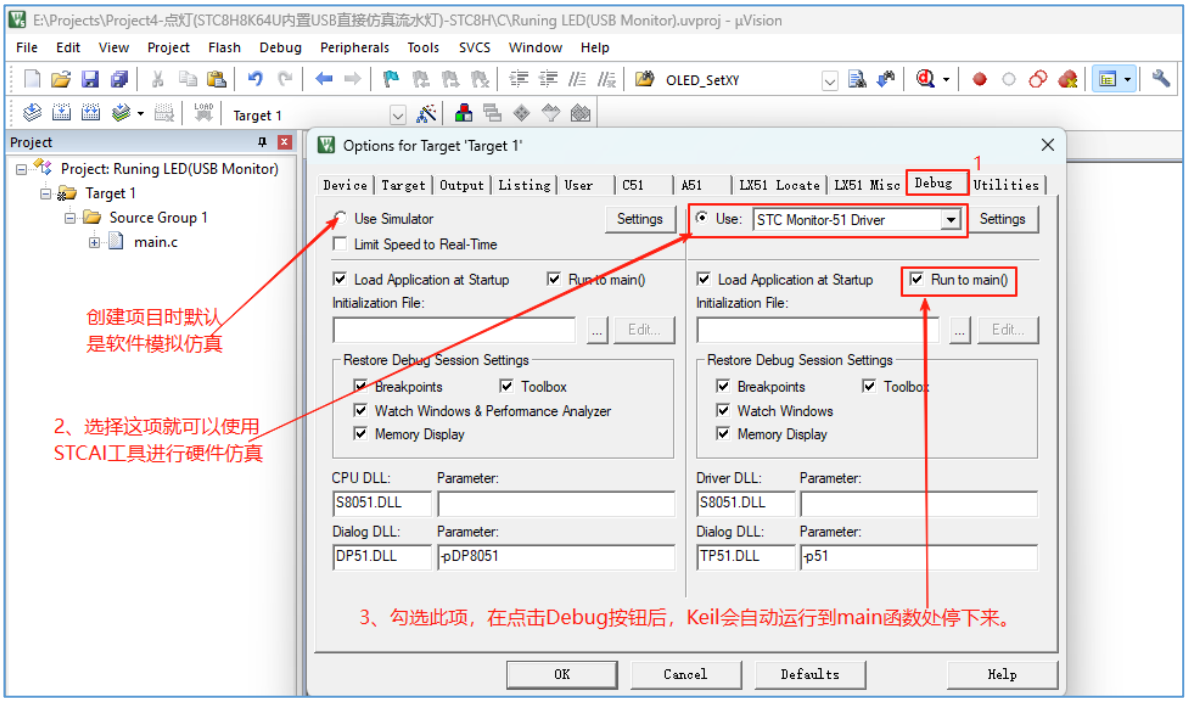
2.7.3　仿真设置，在 Keil 中

本项目是使用 STC8H8K64U 内置 USB 直接仿真流水灯，除了新建项目时必要设置的基础选项外，还要进行仿真设置：“Debug”选项卡和“Utilities”选项卡。

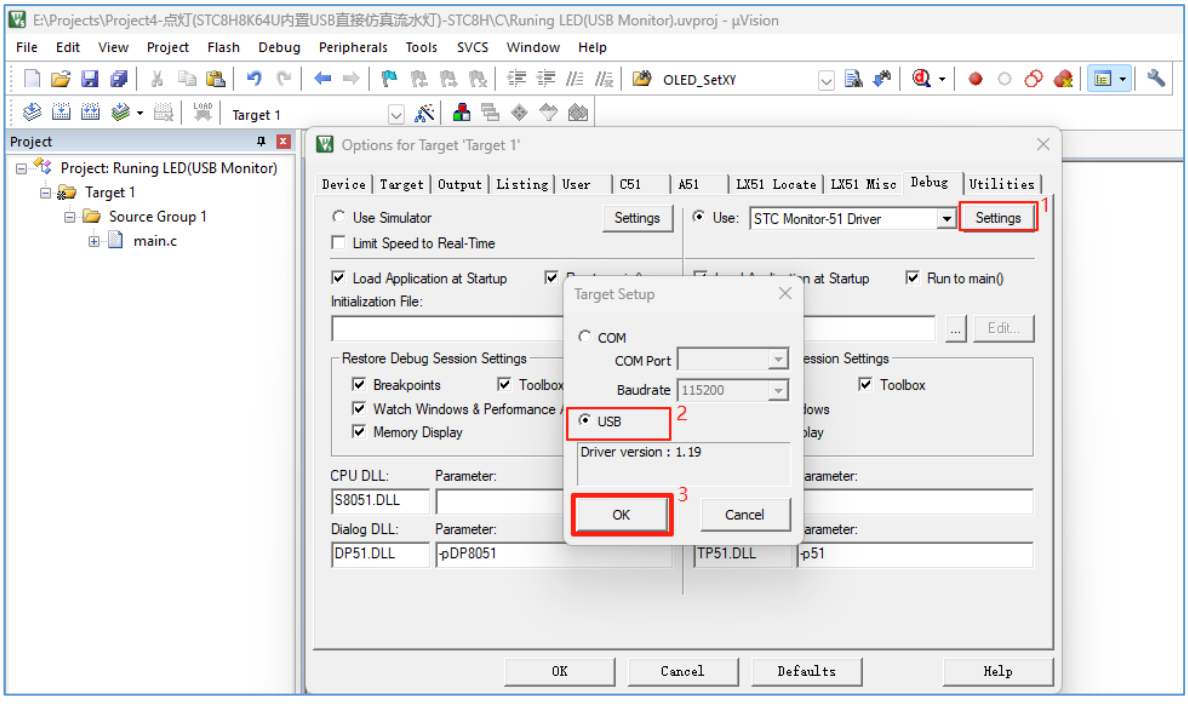
点击工具栏上的魔术棒的工具，打开“Options for Target ‘Target 1’”任务选项面板：

2.7.3.1　Debug 选项卡：

➲ **选择硬件仿真：**系统默认是软件模拟仿真，本项目是使用 STC8H8K64U 内置 USB 直接仿真流水灯，属于硬件仿真，要选择“STC Monitor-51 Driver”

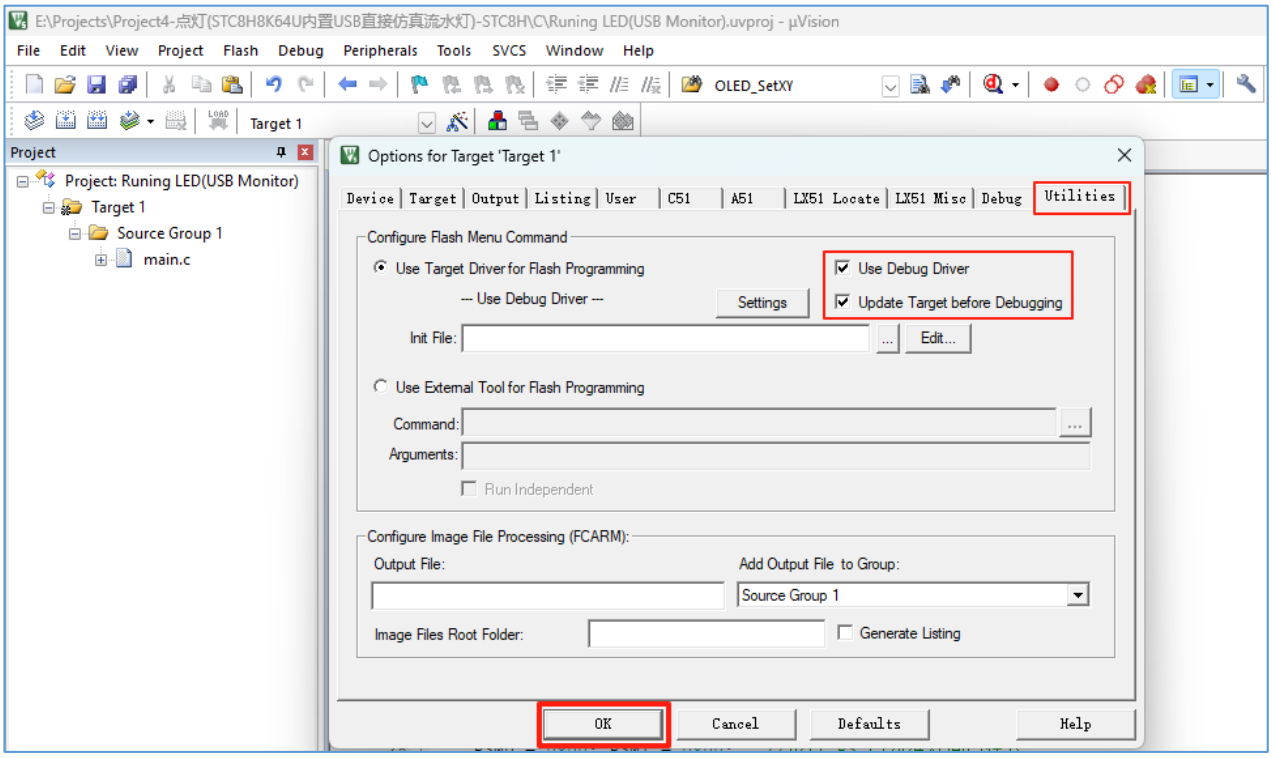


➲ **选择仿真接口：**STC8H8K64U 芯片内置的硬件 USB 支持直接仿真，本项目要选择 USB 为仿真接口。



2.7.3.2　再打开“Utilities”选项卡，进行如下勾选设置：

切换到“Utilities”选项卡，勾选“Use Debug Driver”，点击“OK”，完成设置。



2.7.4　使用 STC8H8K64U 实验箱仿真用户程序

录入代码，保存，编译。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统。

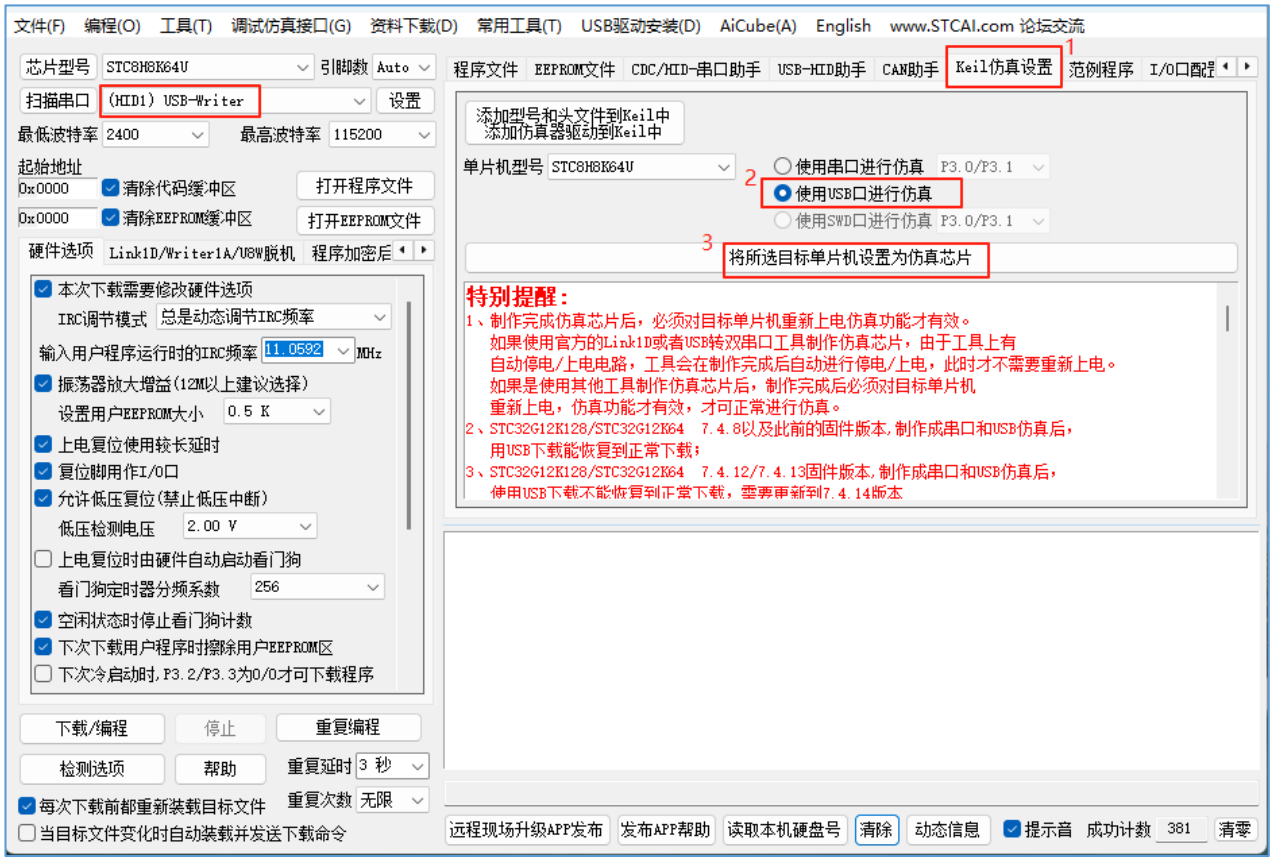
【AiCube-ISP 软件的安装】，请点击查阅本文档 1.1 安装 AiCube-ISP 下载/编程/烧录 工具，含强大的辅助开发工具

【STC8H8K64U 实验箱实物图、与电脑的连接、用户程序下载流程】，请点击查阅本文档的 1.2 STC8H8K64U 实验箱实物图及程序下载

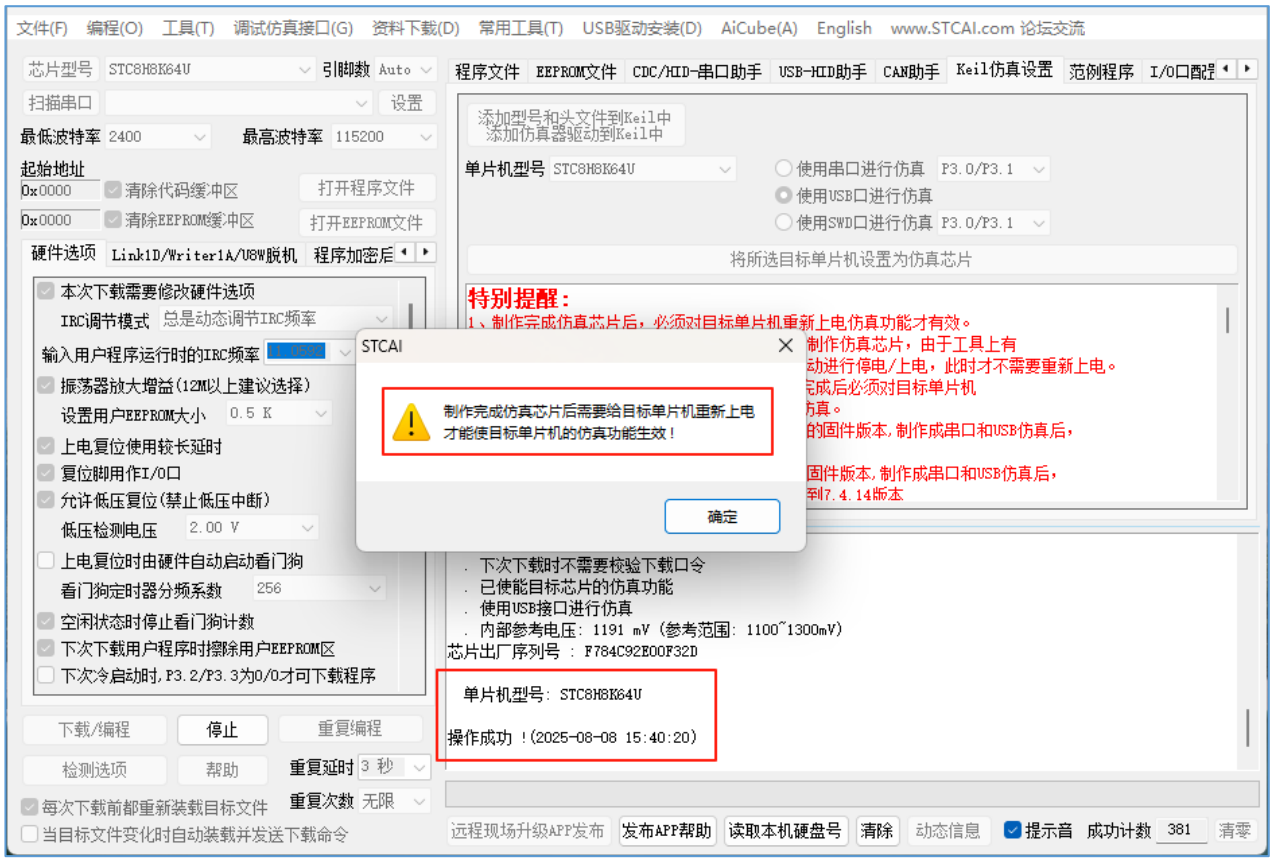
2.7.4.1　制作仿真芯片

在 AiCube-ISP 软件界面，看到扫描串口列表框显示“(HID1) USB Writer”：

- 点击“Keil 仿真设置”标签；
- 在“Keil 仿真设置”选项卡，选择“使用 USB 口进行仿真”
- 点击“将所选目标单片机设置为仿真芯片”

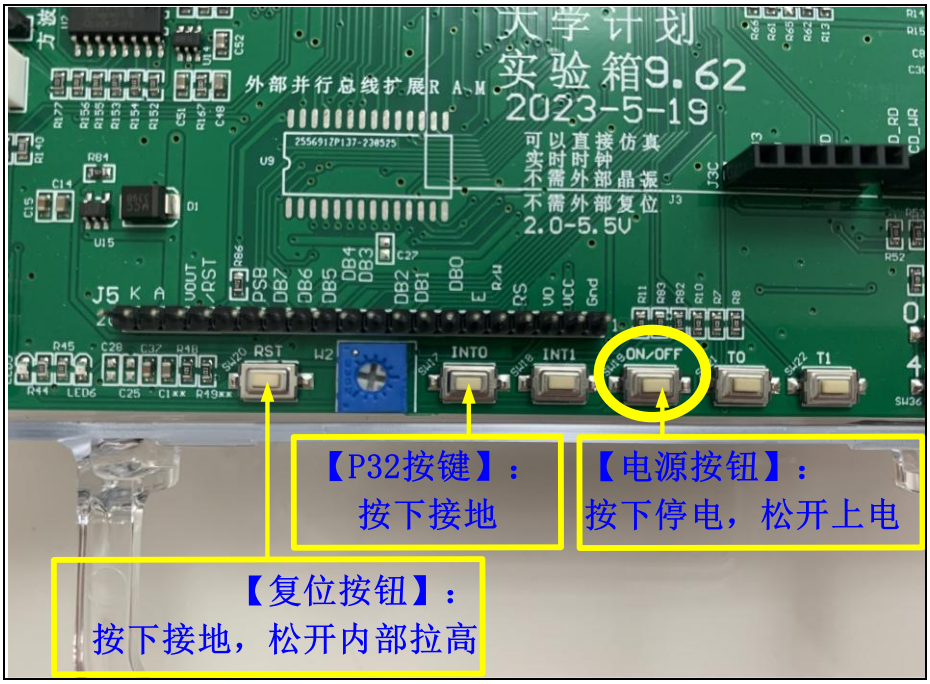


然后看到 AiCube-ISP 软件在自动进行仿真芯片的设置，界面下面的进度条在推进，然后“操作成功!”界面出现特别提示：“制作完成仿真芯片后需要给目标单片机重新上电才能使目标单片机的仿真生效!”如下图：



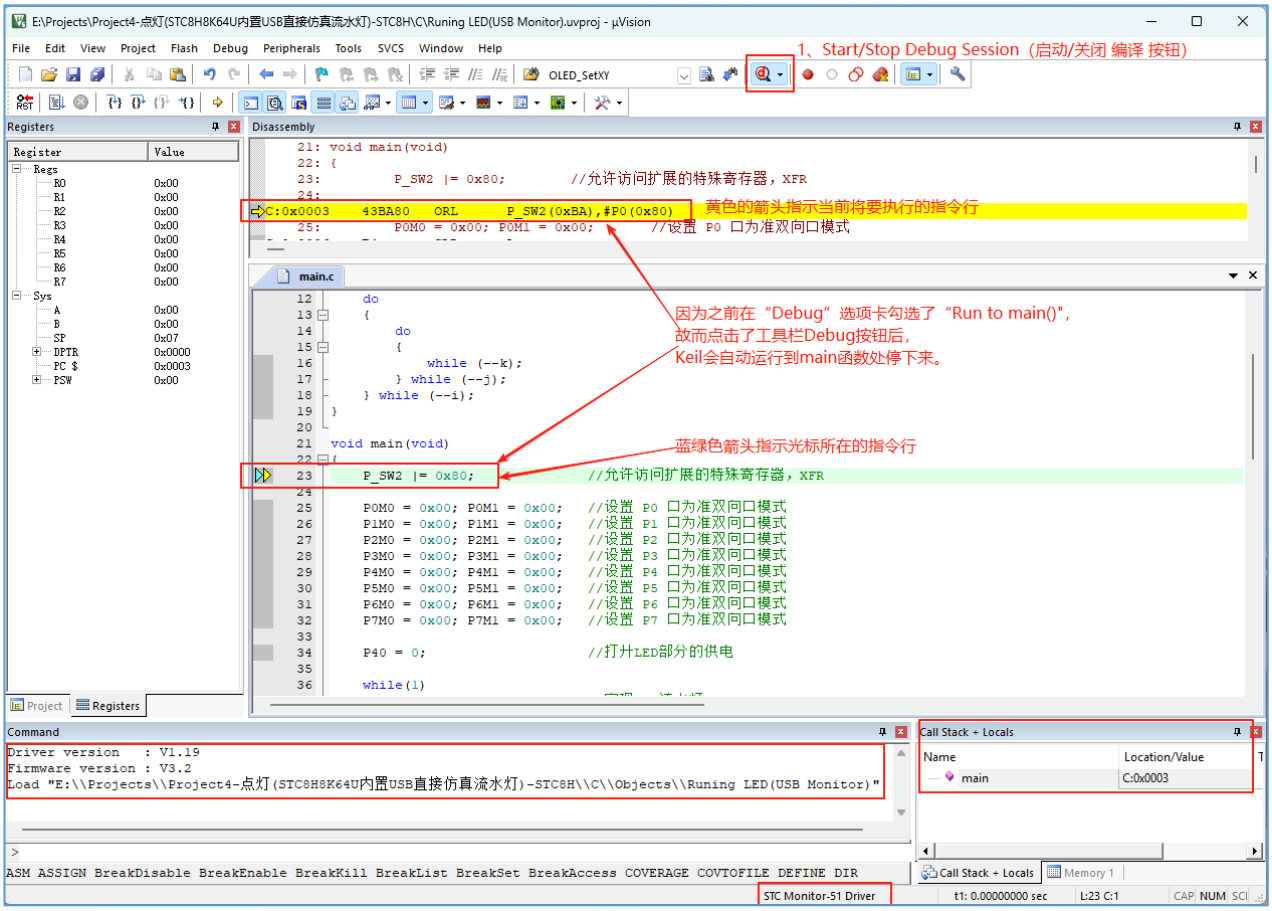
所以，按提示按一下 STC8H8K64U 实验箱正下方的“Power_SW”按钮，给 STC8H8K64U 重新上电使得

其仿真生效：



2.7.5　Keil 仿真并观察实验现象

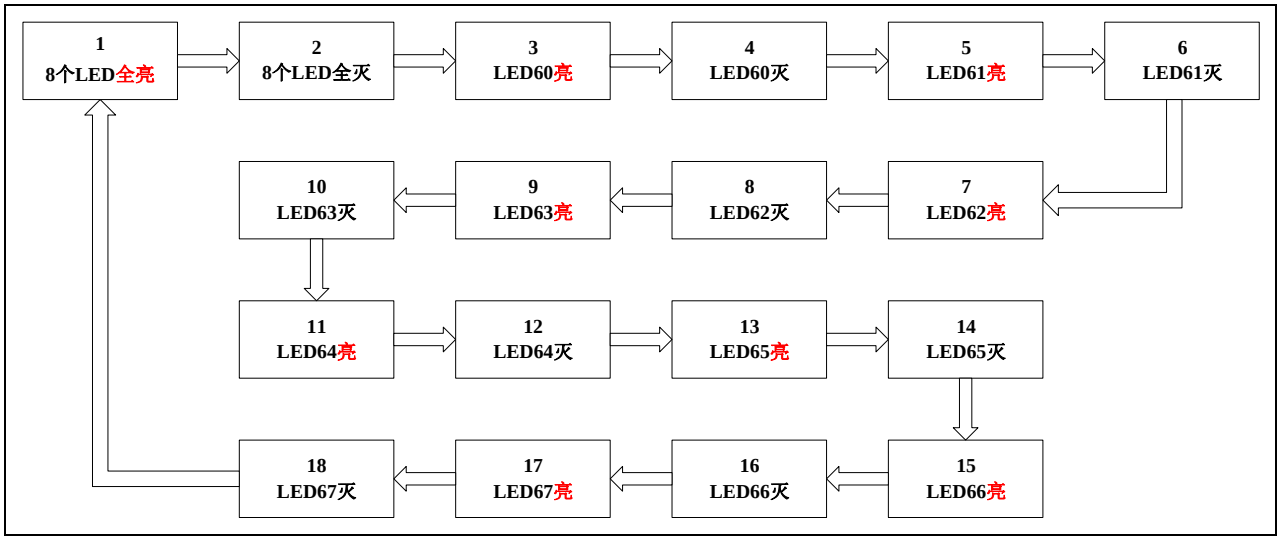
现在，返回 Keil 界面，点击工具栏的 Debug（启动/关闭编译调试操作界面）按钮后，Keil 软件进入仿真界面，如下图：



提示：退出 Debug 调试界面，同样点击工具栏的“Debug”按钮。

2.7.5.1　使用 Run 按钮，观察全速运行效果的实验现象

一切准备就绪，我们先 Run 一次程序，看看程序整体执行，实验箱仿真的实验结果。

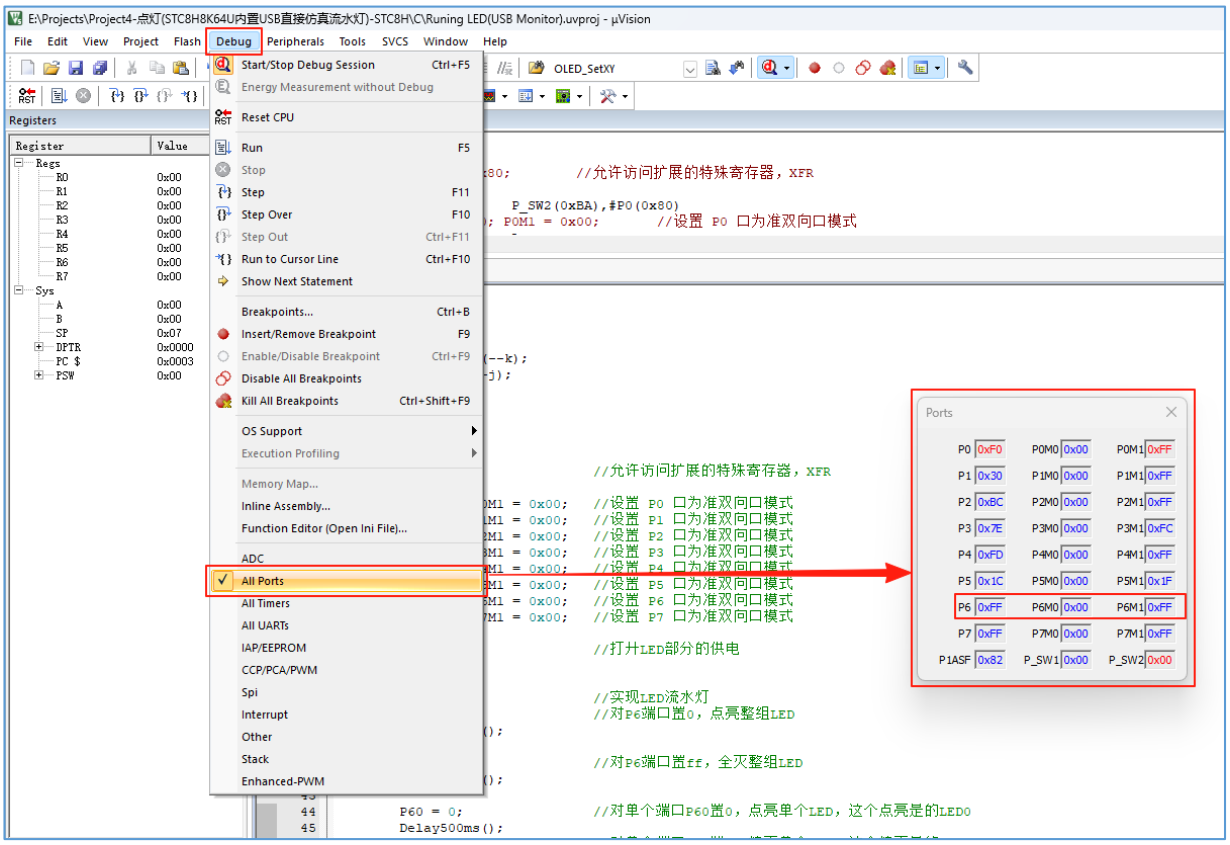


实现了我们想要的流水灯的实验现象。实验成功。

2.7.5.2　观察外设寄存器数据与实验现象

除了观察实验箱流水灯的状态，还可以用 Keil 软件来观察 P6 端口的数据，验证数据与实验箱上 LED 的亮灭现象是否对应。

选择菜单栏的“Debug”→勾选“All Ports”，出现“Ports”窗口

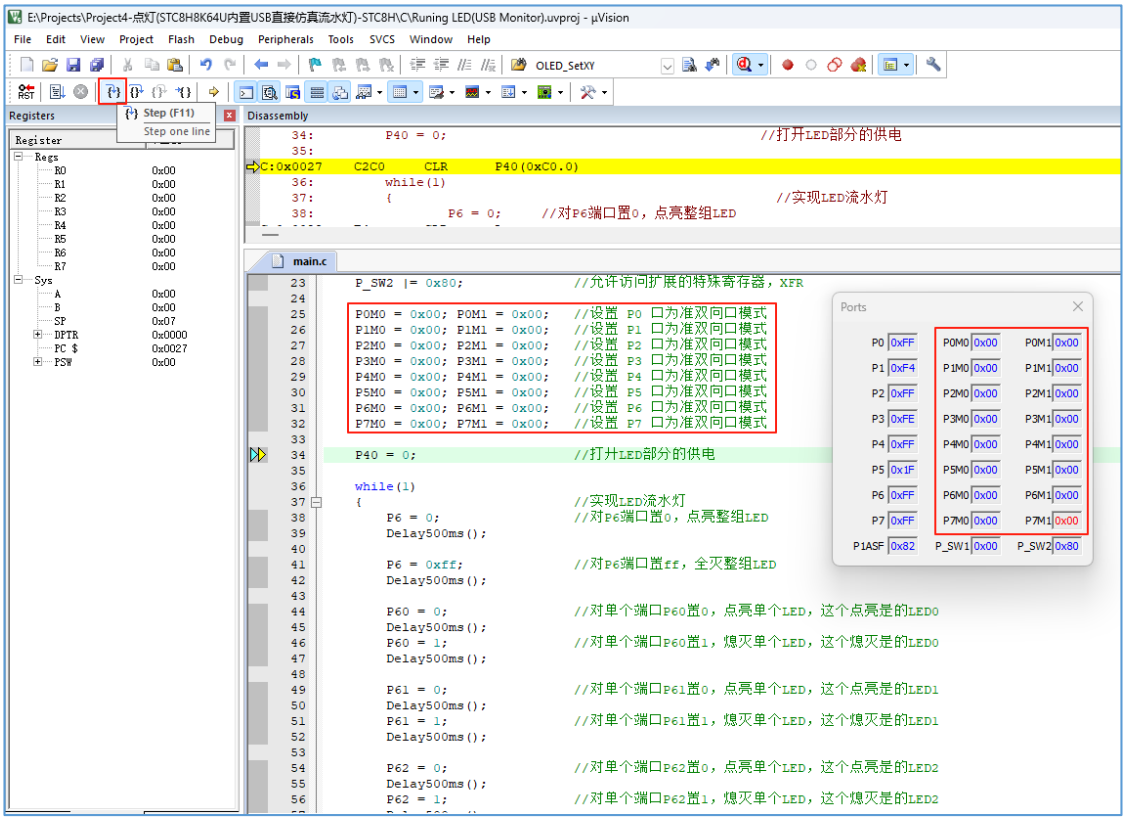


我们来单步运行程序，观察实验仿真现象。

依然将指示程序运行位置的黄色箭头停在 main 函数的第一条指令处，第 23 行：

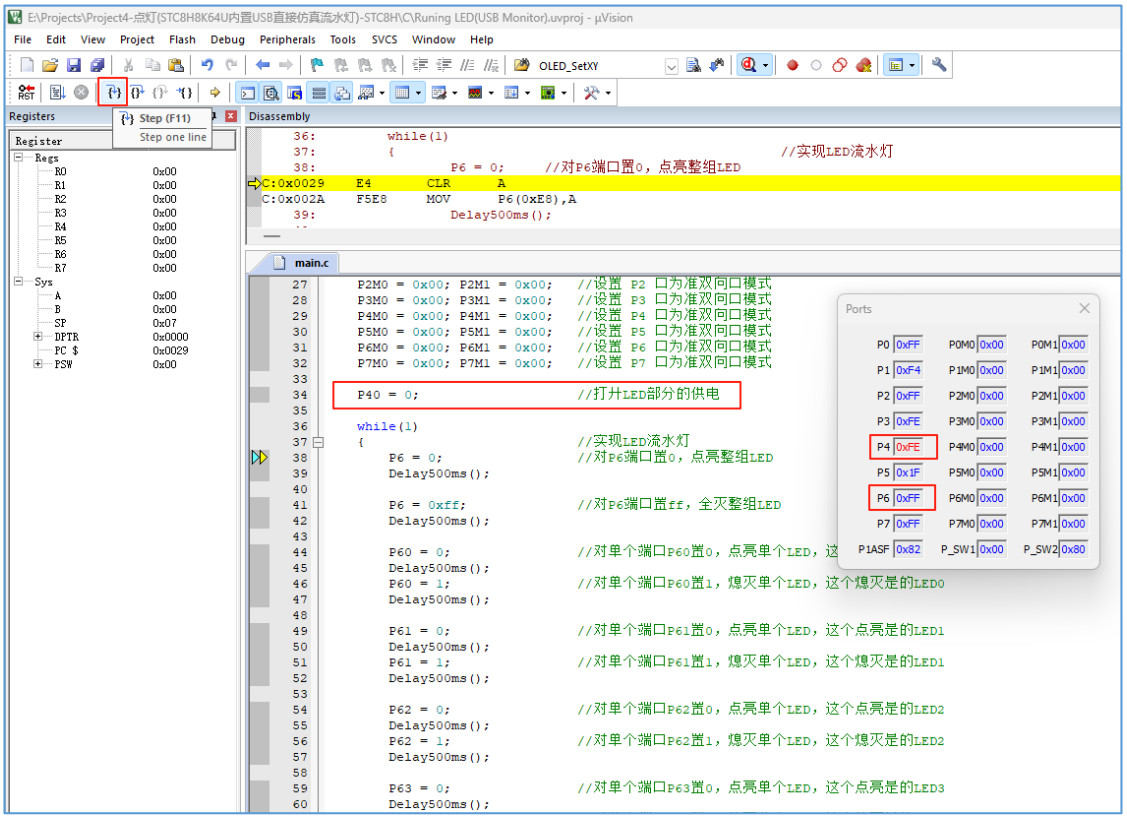
➔ **点击单步运行按钮【Step（Step one line）】**，一直箭头指向程序到第 34 行，我们看到：

- P0～P7 口依次被设置为准双向口模式，“Ports”窗口的 P0M0～P0M1，…，P7M0～P7M1 数据框里都显示 0x00。



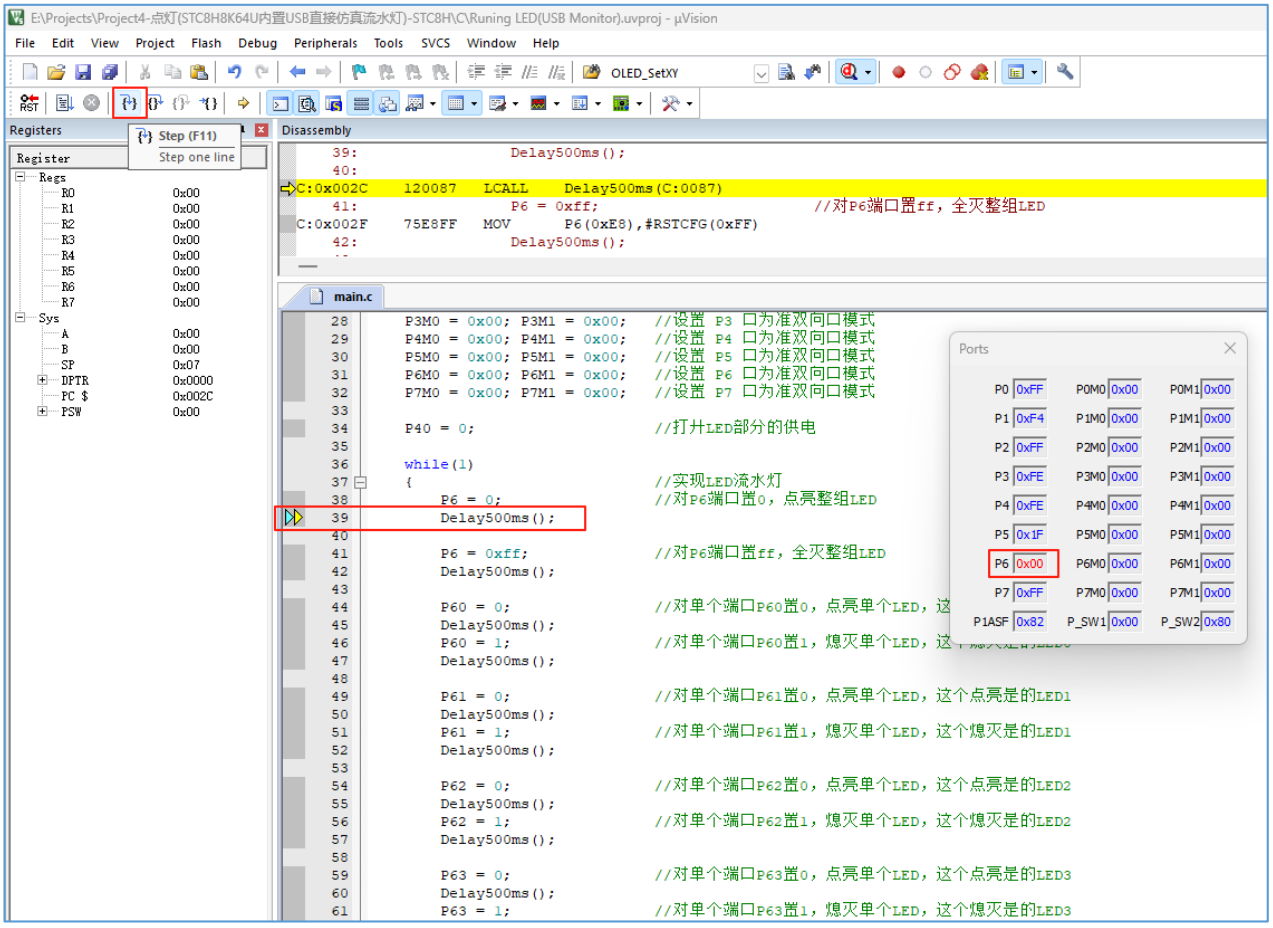
➔ **点击单步运行按钮【Step（Step one line）】**，P40 被置 0，箭头指向到程序第 38 行，这时：

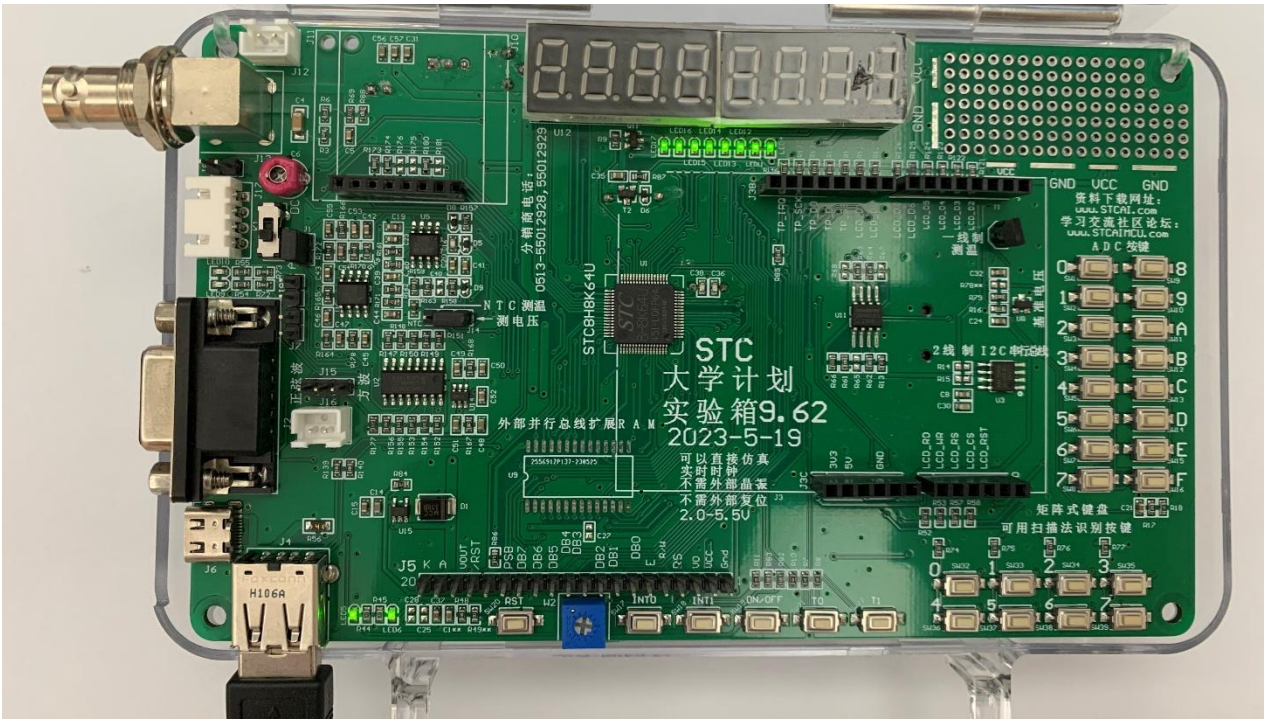
- “Ports”窗口 P4 显示 0xFE，给 LED 部分供电
- P6 依然为 0xFF，STC8H8K64U 实验箱上 LED 灯保持全灭。



➔ **点击单步运行按钮【Step（Step one line）】**，P6 被置 0，箭头指向到程序第 39 行：

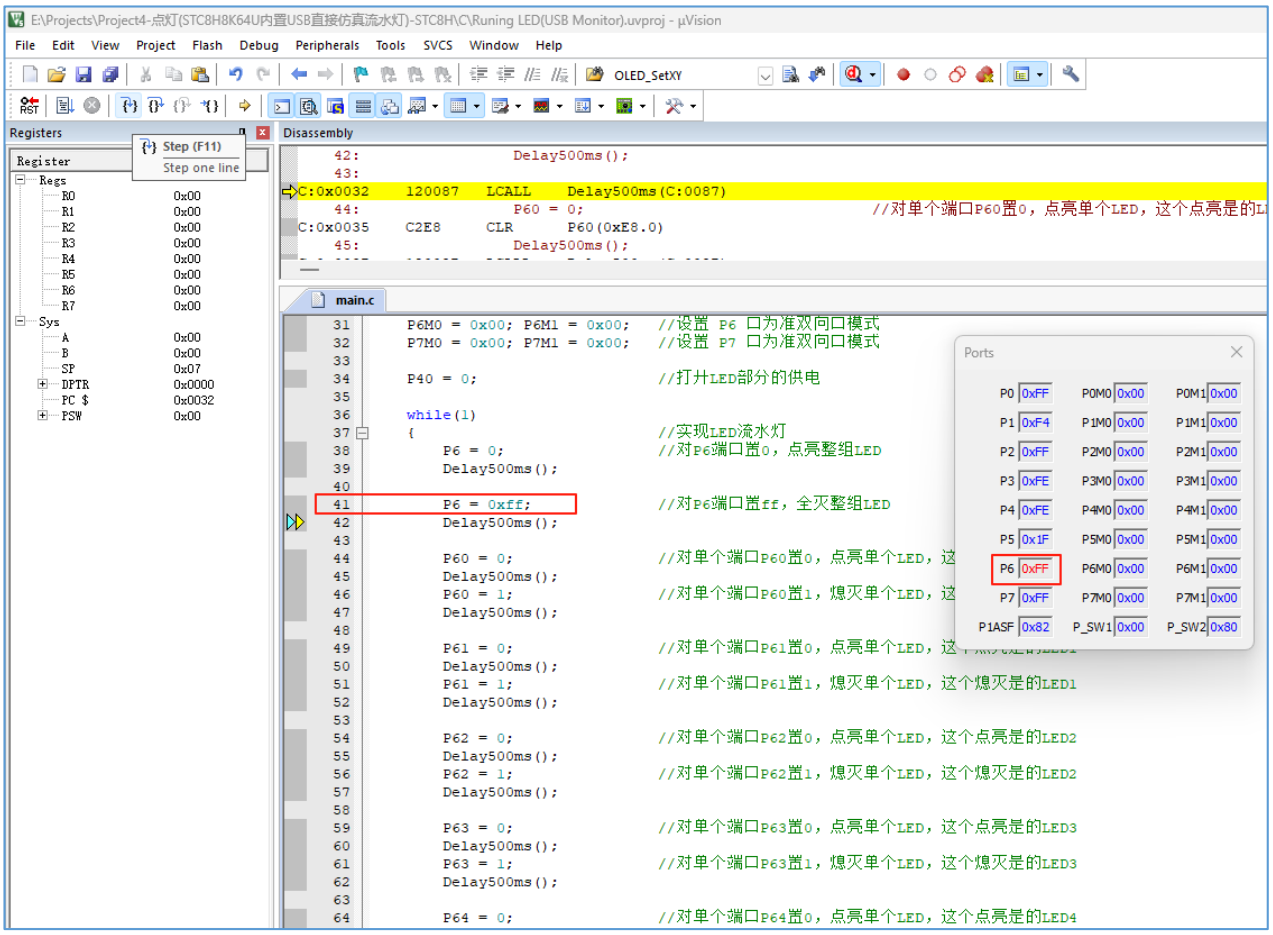
- “Ports”窗口 P6 显示 0x00
- STC8H8K64U 实验箱 LED60～LED67 全亮。

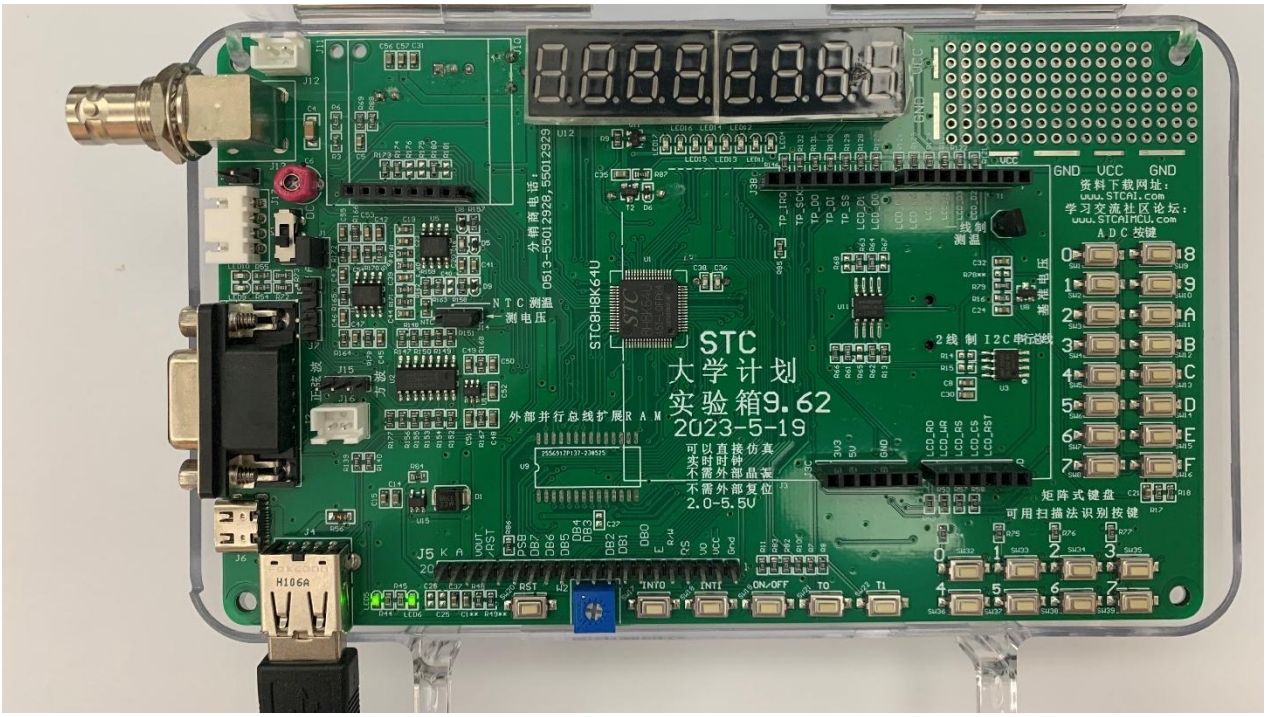




➔ 因接下来的第 39 条指令是延时函数，**点击【Step Over】按钮**跳过延时函数，箭头指向到程序第 41 行，**再点击单步运行按钮【Step（Step one line）】**，P6 被置 0xFF，箭头指向到程序第 42 行：

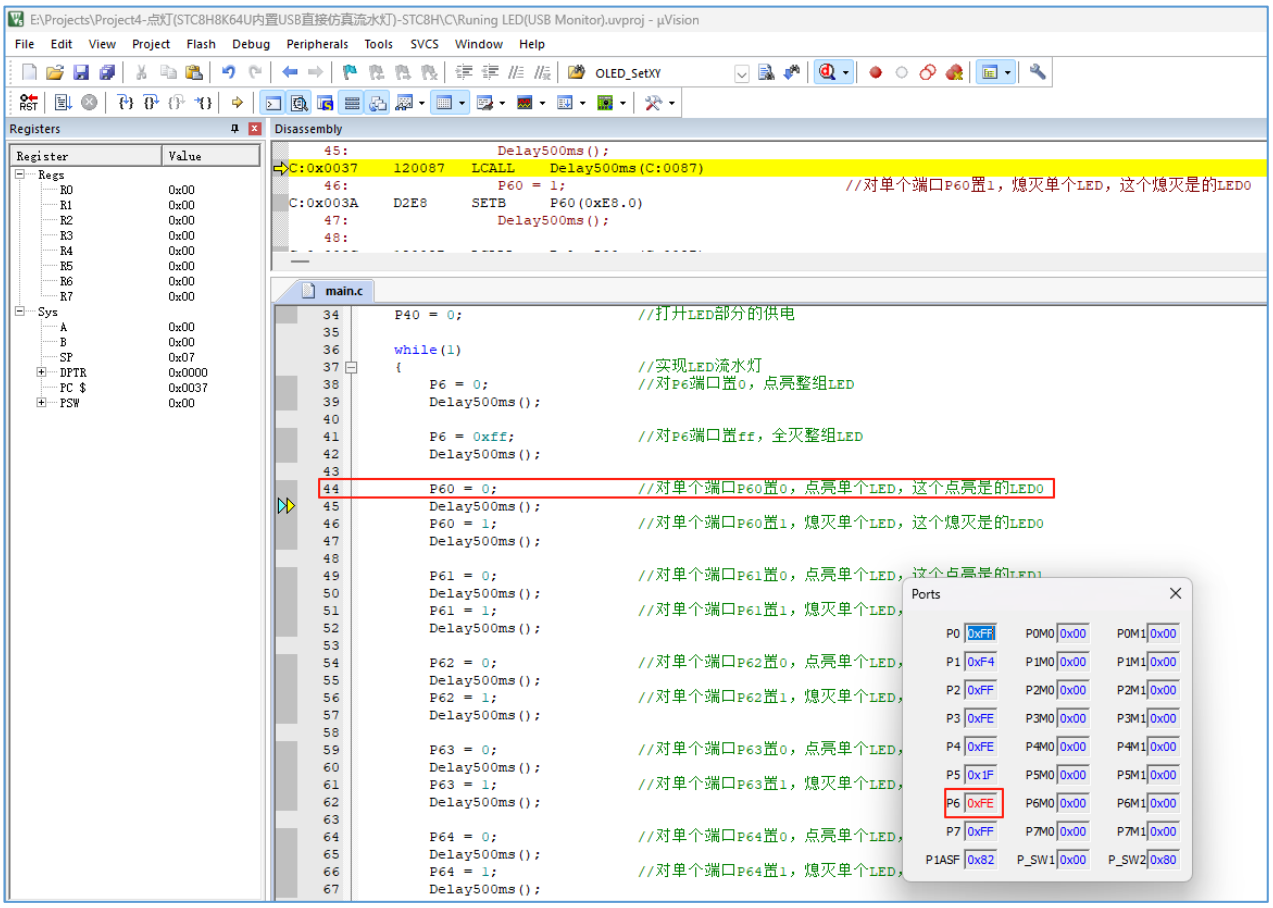
- “Ports”窗口 P6 显示 0xFF
- STC8H8K64U 实验箱 LED60～LED67 全灭。

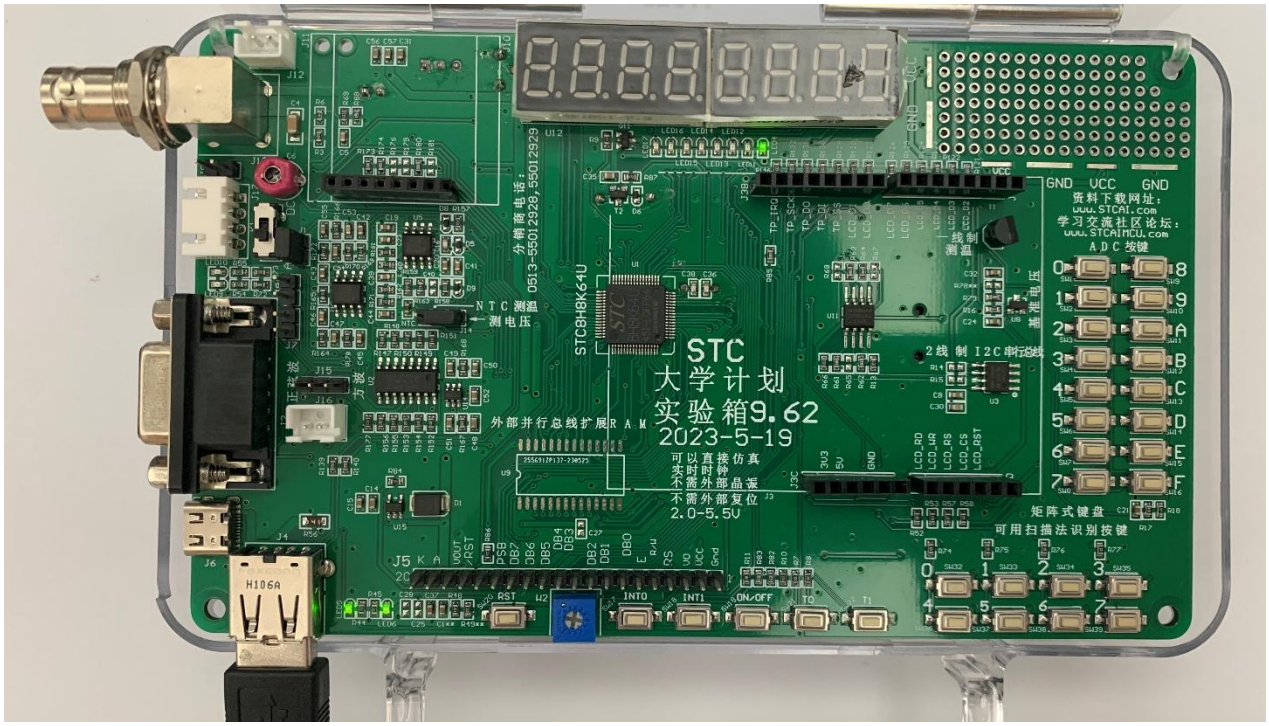




➔ 第 42 条指令是延时函数，**点击【Step Over】按钮**跳过延时函数，箭头指向到程序第 44 行，**再点击单步运行按钮【Step（Step one line）】**，P60 被置 0，箭头指向到程序第 45 行：

- “Ports”窗口 P6 显示 0xFE
- STC8H8K64U 实验箱 LED60 灯亮，LED61～LED67 灭。





重复上述操作，观察 P6 口数据变化，及对应的 STC8H8K64U 实验箱上 LED 灯的变化，是否一一对应。

2.7.6 实验程序汇编语言代码

;以下是控制 P6.0~P6.7 引脚上一个 LED 灯循环亮灭跑马灯的汇编语言程序代码。
;特别注意延时子程序汇编语言写法。试通过每一条指令的所花的机器时钟周期计算一下软件延时时间。
;51 汇编程序

```
$INCLUDE(STC8H.INC)
STACK_POIRTER       EQU     0D0H    ;堆栈开始地址

    ORG     0000H
    LJMP    START
    ORG     0100H

START:
    MOV     P_SW2, #80H             ;允许访问扩展的特殊寄存器，XFR
    MOV     SP, #STACK_POIRTER      ;堆栈栈顶设置
    ;端口初始化
    CLR     A
    MOV     P0M1, A     ;设置 P0 为准双向口
    MOV     P0M0, A
    MOV     P1M1, A     ;设置 P1 为准双向口
    MOV     P1M0, A
    MOV     P2M1, A     ;设置 P2 为准双向口
    MOV     P2M0, A
    MOV     P3M1, A
    MOV     P3M0, A
    MOV     P4M1, A
    MOV     P4M0, A
    MOV     P5M1, A
    MOV     P5M0, A
    MOV     P6M1, A
    MOV     P6M0, A
    MOV     P7M1, A
    MOV     P7M0, A
    MOV     P6, #0FFH
    ;变量初始化
    CLR     P40             ;流水灯总开关打开

LOOP:
    MOV     P6, #00H            ;P60 ~ P67 全亮
    ACALL   DELAY500MS      ;调用延时子程序

    MOV     P6, #0FFH           ;P60 ~ P67 全灭
    ACALL   DELAY500MS      ;调用延时子程序

    CLR     P60                 ;给 P6.0 清零，低电平时 led 亮
    ACALL   DELAY500MS      ;调用延时子程序
```

```
    SETB    P60             ;给 P6.0 置 1，高电平时 led 灭

    CLR     P61
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P61

    CLR     P62
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P62

    CLR     P63
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P63

    CLR     P64
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P64

    CLR     P65
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P65

    CLR     P66
    ACALL   DELAY500MS      ;调用延时子程序
    SETB    P66

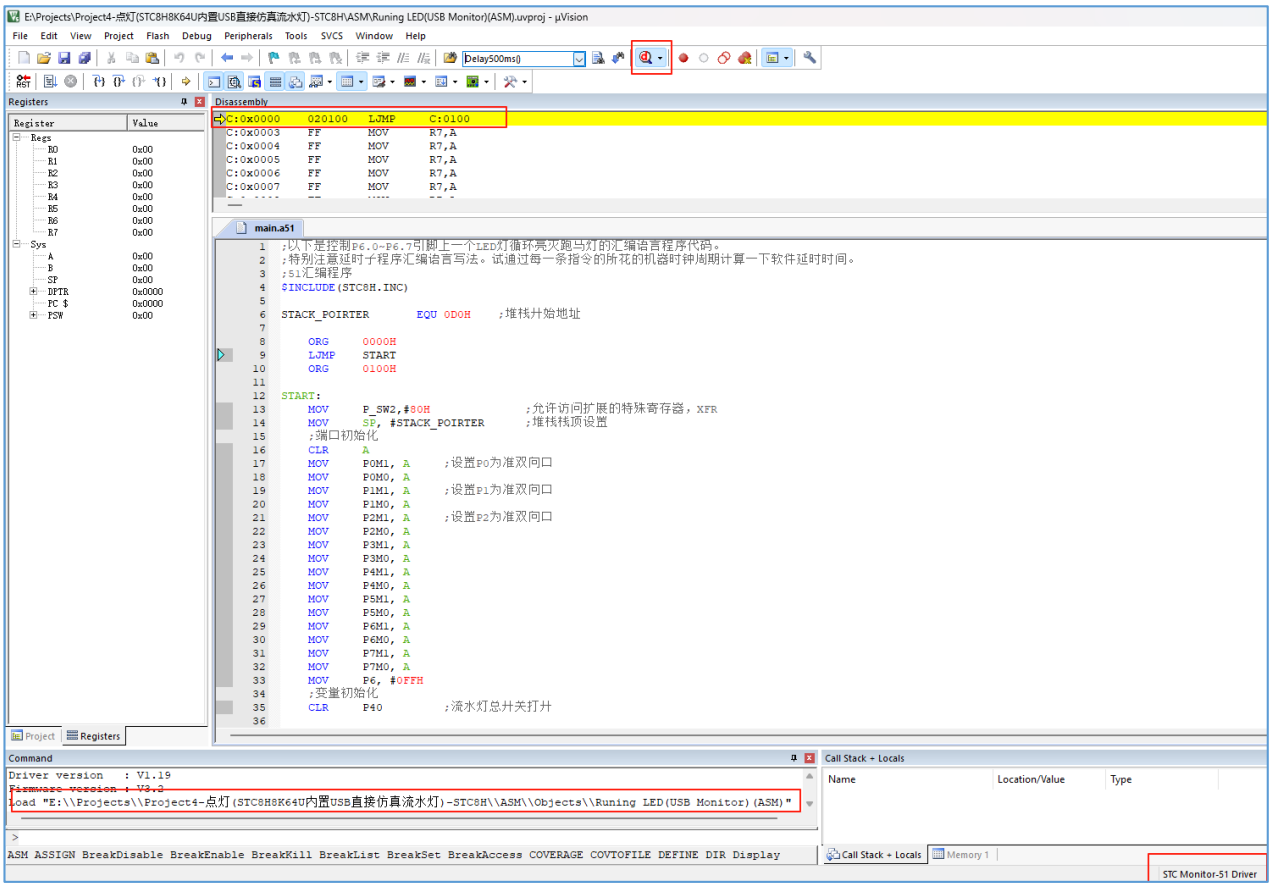
    CLR     P67
    ACALL   DELAY500MS
    SETB    P67

    SJMP    LOOP            ;无条件跳转至 LOOP 处

DELAY500MS:                 ;延时 500 毫秒@11.0592MHz
    PUSH    30H
    PUSH    31H
    PUSH    32H
    MOV     30H, #29
    MOV     31H, #14
    MOV     32H, #52
NEXT:
    DJNZ    32H, NEXT
    DJNZ    31H, NEXT
    DJNZ    30H, NEXT
    POP     32H
    POP     31H
    POP     30H
```

```
    RET
END                         ;汇编程序结束伪指令语句
```

录入代码（main 文件类型选 Asm File），保存，编译，如同 C 语言代码程序一样，打开 AiCube-ISP 软件，连接好 STC8H8K64U 实验箱，并下载文件到实验箱，就能观察到一样的流水灯现象。



2.7.7 使用【AiCube】项目创建助手工具创建使用 STC8H8K64U 内置 USB 仿真流水灯

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit、Ai8051U-8Bit、STC32G12K128、STC8H8K64U、STC15F2K60S2、STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

2.7.7.1 安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

【AiCube-ISP 软件的安装】，请点击查阅本文档 1.1 安装 AiCube-ISP 下载/编程/烧录工具,含强大的辅助开发工具

2.7.7.2 使用【AiCube】助手创建项目

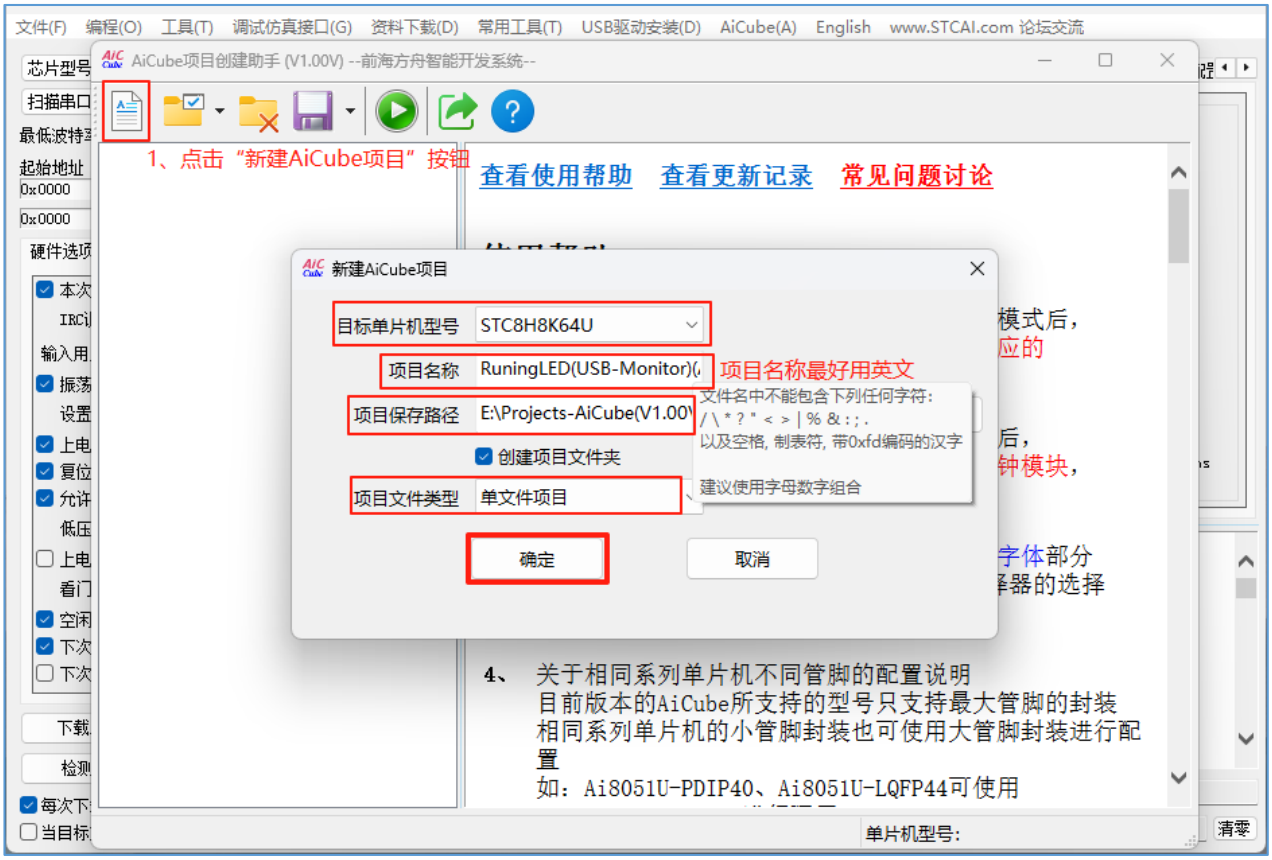
双击【AiCube-ISP】图标，打开 AiCube-ISP 软件界面。使用【AiCube】前，最好先“**重新安装 Keil 驱动和头文件**”如下图：



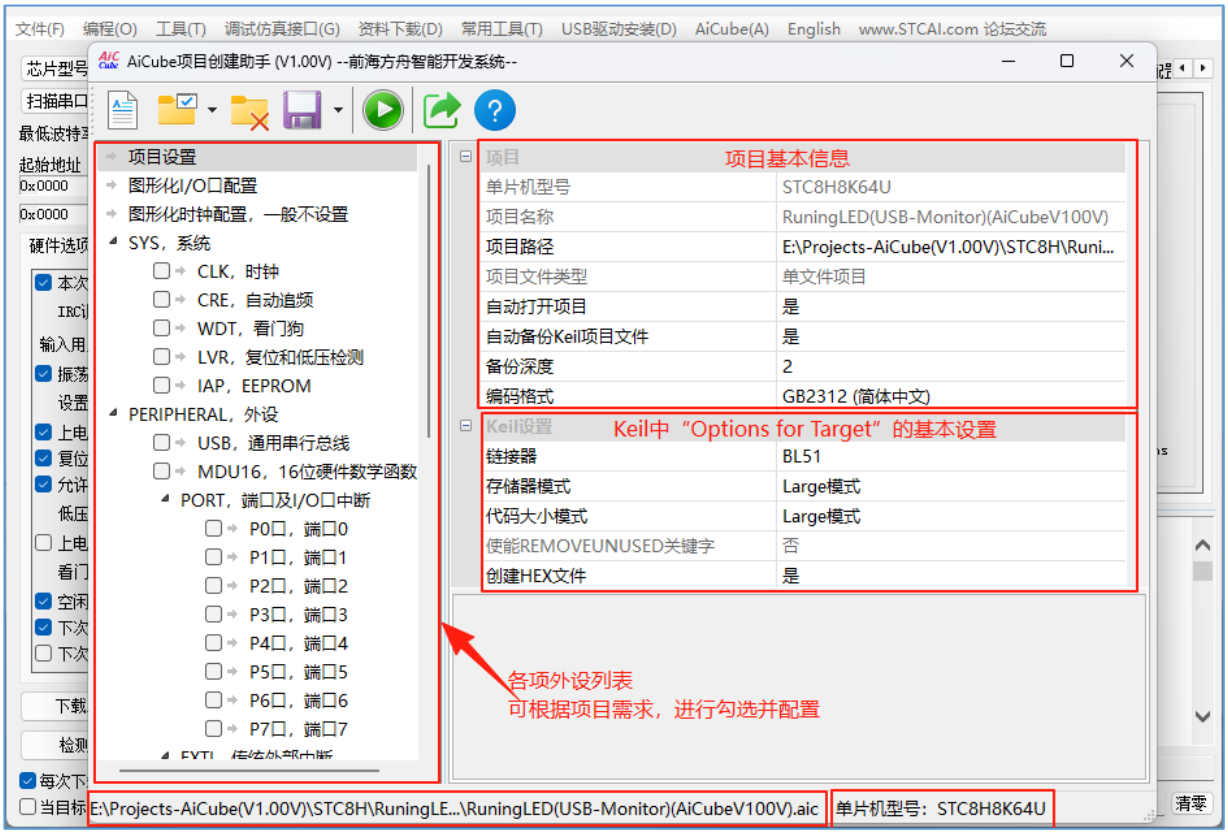
➲ **点击菜单栏的【AiCube（A）】**，弹出如下窗口：



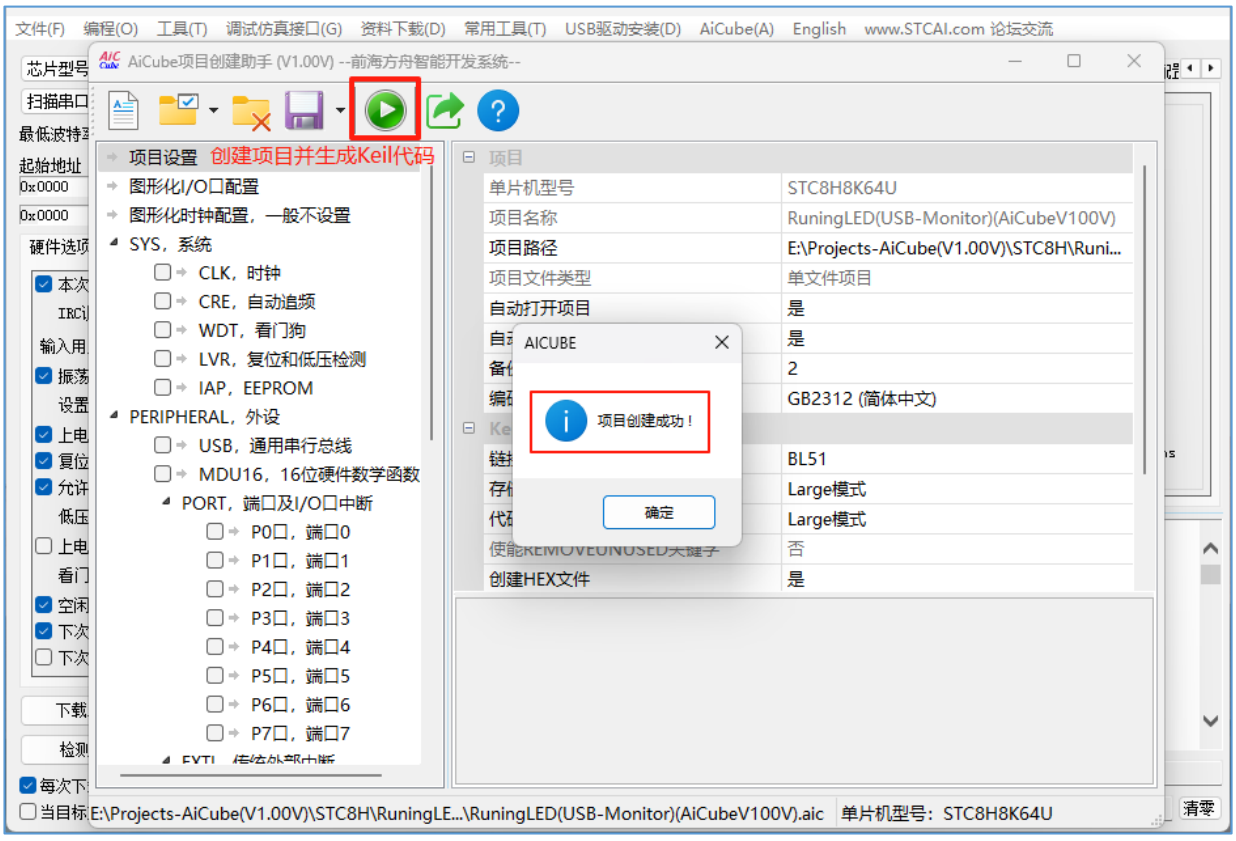
➲ **点击按钮“新建 AiCube 项目”**，出现如下浮窗，选择并填写项目基本信息，建议项目名称用英文！



➲ **点击“确定”**，出现如下配置界面：



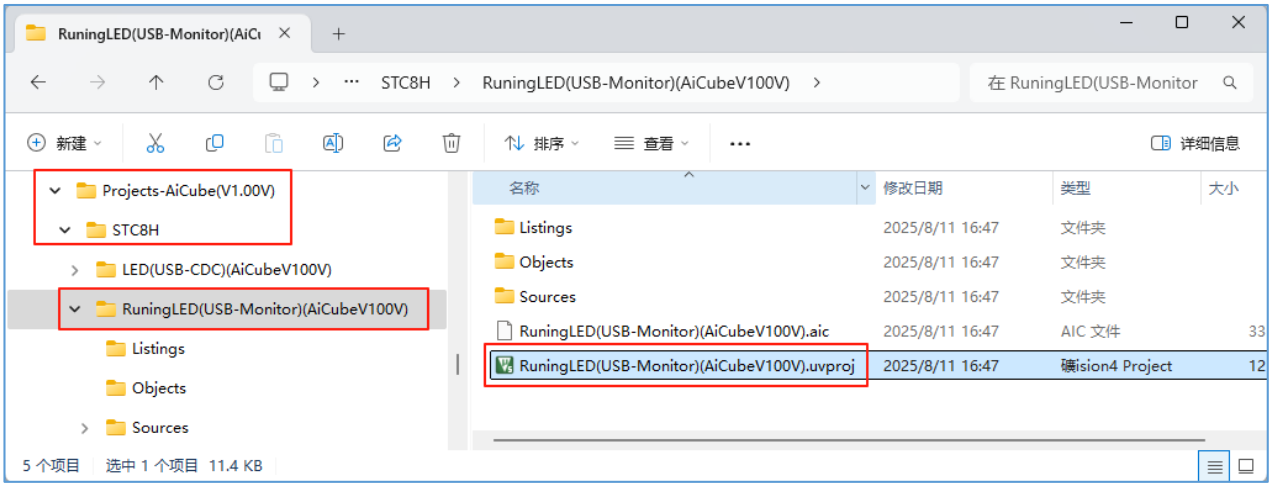
对于本项目，用 STC8H8K64U 内置的 USB 功能仿真实现流水灯的实验，非常简单，不需要对时钟或者其他选项做特殊配置。直接点击“创建项目并生成 Keil 代码”，【AiCube】助手提示：“项目创建成功！”，如下图：



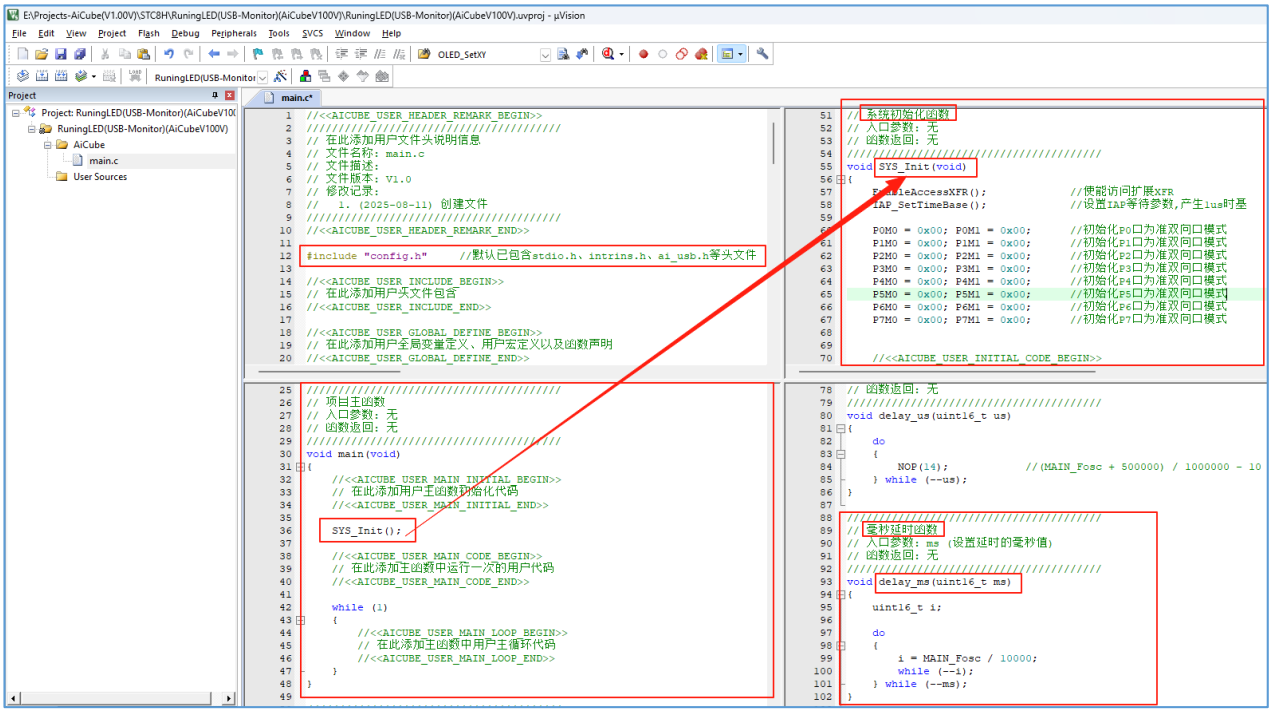
➲ **点击“确定”**，立即自动打开 Keil 软件，进入该项目的 Keil 集成开发界面，即【AiCube】成功帮我们创建了项目及文件目录“RuningLED(USB-Monitor)(AiCubeV100V)”。至此，可以关闭【AiCube】。

➲ **找到并打开刚刚创建的项目目录**

发现刚刚的操作过程，【AiCube】助手已经帮我们建立了项目目录及 RuningLED(USB-Monitor)(AiCubeV100V).uvproj 文件，如下图：



➲ **观察已打开的 Keil 项目文件：RuningLED(USB-Monitor)(AiCubeV100V).uvproj，如下图：**



根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、微秒和毫秒延时函数。

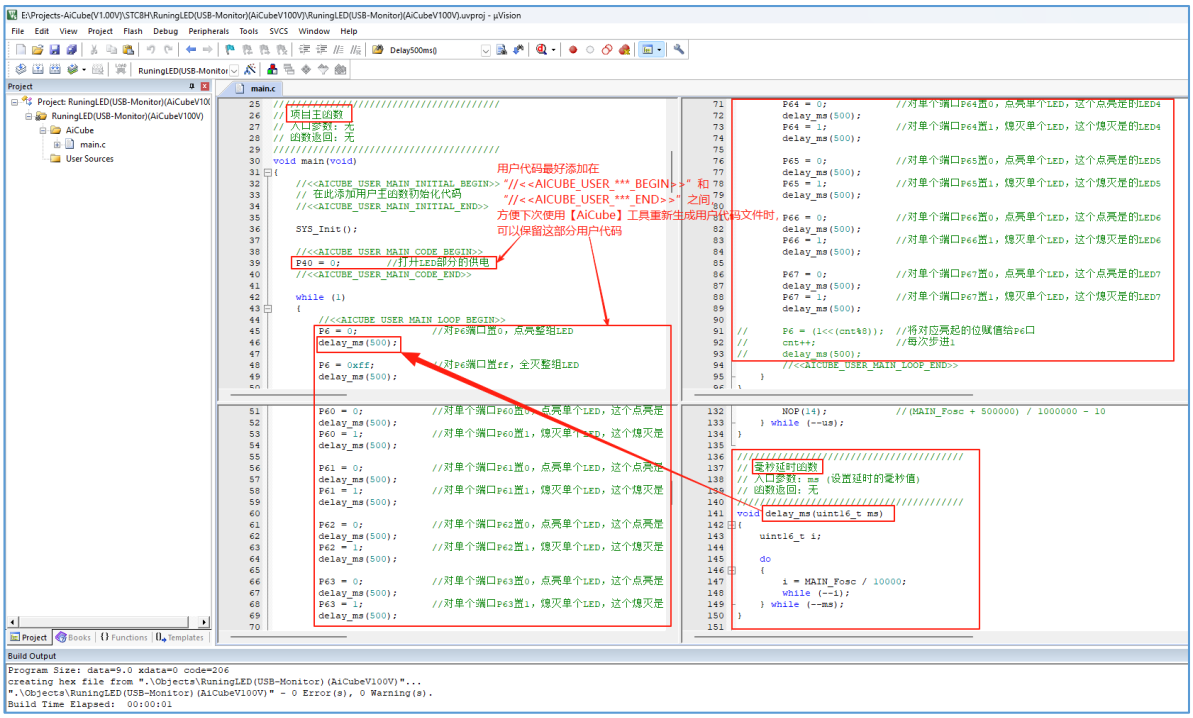
2.7.7.3　增加实验项目代码

为实现“用 STC8H8K64U 内置的 USB 功能仿真实现流水灯”的实验，我们现在需要在主函数中：

➲ 加入“P40 = 0”打开 LED 供电的代码

➲ 在 while 循环中补充实现流水灯的代码，其中延时函数根据需要直接调用。

保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成 4 个窗口）：

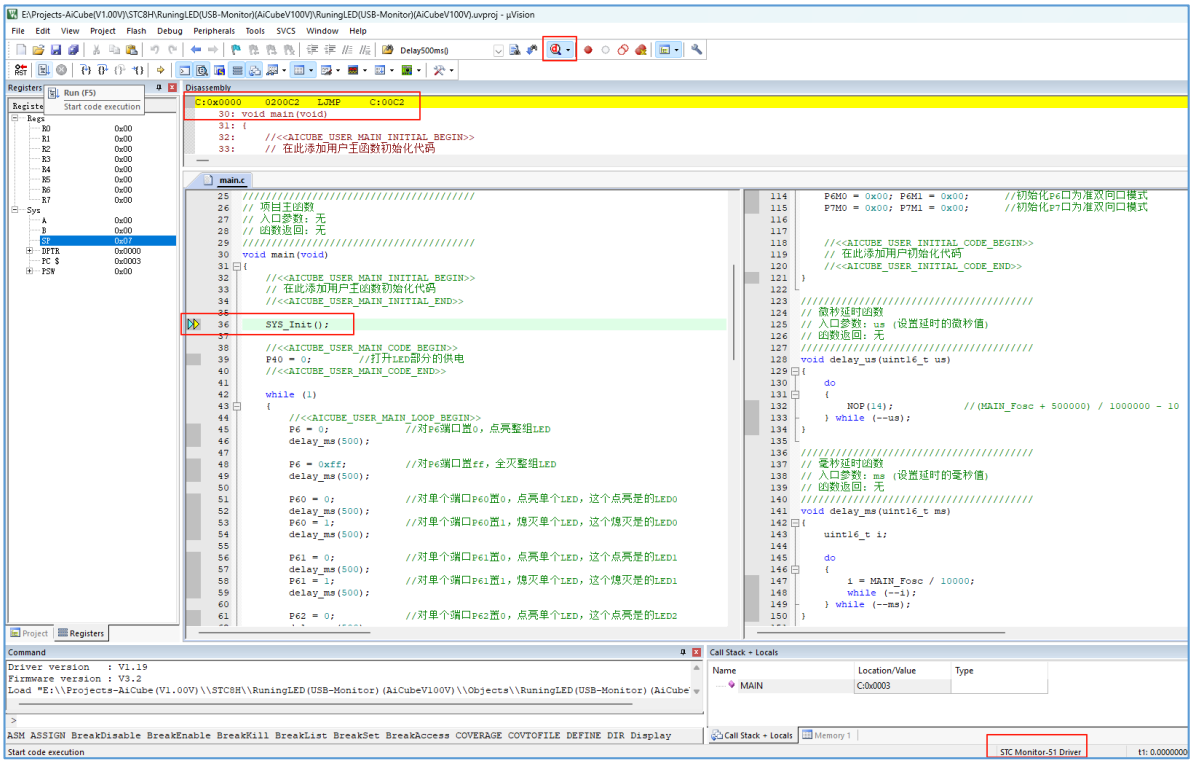


2.7.7.4　使用 AiCube-ISP 软件设置仿真芯片，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果

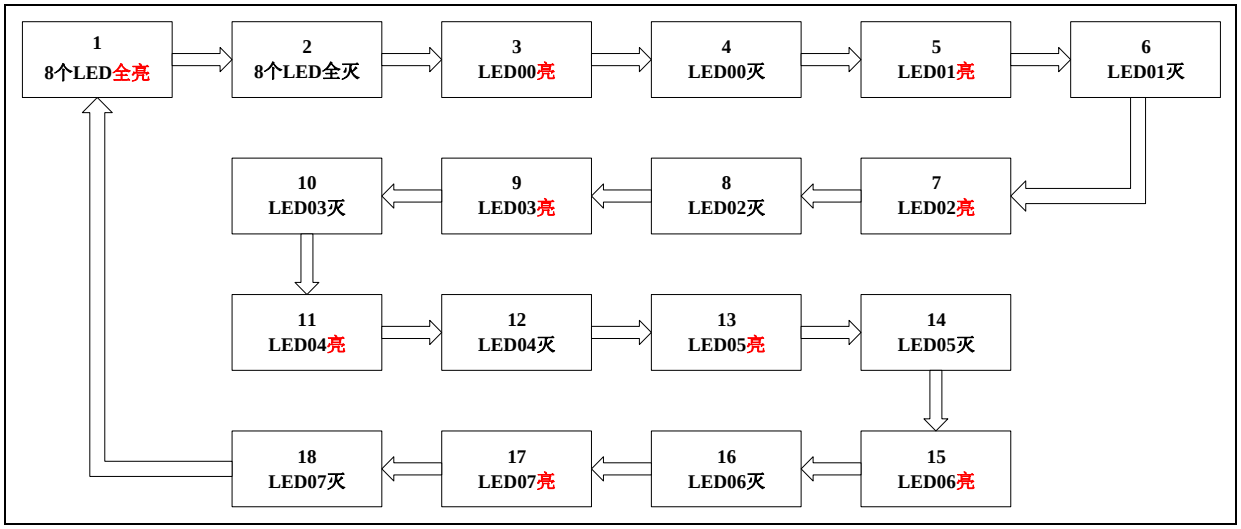
连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，在 Keil 仿真设置选项卡中，按步骤操作“将所选目标单片机设置为仿真芯片”。

回到 Keil 界面，在 Keil 中按步骤在“Options for Target”的 Debug 选项卡和 Utilities 选项卡中完成使用“STC Monitor-251 Driver”的 USB 仿真操作。

点击 File Toolbar 上的“Start/Stop Debug Session”按钮，进入仿真界面，如下图：



一切准备就绪，点击 Debug Toolbar 上的【Run】按钮，我们看到 STC8H8K64U 实验箱的 LED 灯按实验的预设跑动起来了。



当然，你也可以单步执行，打开 Debug 菜单项的【All Ports】观察 P6 口数据变化。

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.8 I/O 口实验 -- I/O 口读取按键状态并点灯

2.8.1 实验要求

可以通过正确设置 I/O 口模式来读取实验箱上按键的电平，然后根据对应按键的电平来设置 LED 灯的点亮/熄灭状态。

2.8.2 实验建议

STC8H8K64U 系列单片机所有的 I/O 口均有 4 种工作模式：

- ✓ 准双向口/弱上拉（标准 8051 输出口模式）
- ✓ 强推挽输出/强上拉
- ✓ 高阻输入（电流既不能流入也不能流出）
- ✓ 开漏模式（如果打开内部上拉电阻，就相当于准双向口；如果不加上拉并对外置 1，就相当于高阻输入）

1）高阻输入模式，内部配有 6K 上拉电阻；
为了读取外部状态，需要打开内部的 6K 上拉电阻。

2）开漏模式，内部同样配有 6K 上拉电阻；
打开内部 6K 上拉电阻后，其功能与准双向口/弱上拉模式相似。

3）读取 I/O 口的外部状态时，可以采用以下三种模式：

准双向口/弱上拉模式:

===要确保对外输出锁存为 1，关闭对内部对外强输出低的 MOS 管，
才能准确读到外部状态
===如对外输出锁存为 0，则外部被内部对外强输出低的 MOS 管拉低，
读到的是低电平，不能准确读到外部来的高输入/被内部拉低了

高阻输入模式，每个 I/O 口都有独立的 6K 上拉电阻:
===为了准确读取外部状态，需要打开内部的 6K 上拉电阻，或外部有上拉能力
===内部的对外输出锁存器对此模式无效，对外无输出高/低的能力
===电流流不进来，也留不出去，
高阻输入，无输出能力，不影响外部来的的输入状态

开漏模式，每个 I/O 口都有独立的 6K 上拉电阻：
启用内部 6K 上拉电阻后，其功能与准双向口/弱上拉模式相似
===要确保对外输出锁存为 1，关闭对内部对外强输出低的 MOS 管，才能准确读到外部状态
===如对外输出锁存为 0，则外部被内部对外强输出低的 MOS 管拉低，
读到的是低电平，不能准确读到外部来的高输入/被内部拉低了

4）I/O 口对外输出高或低电平时，可以采用以下三种模式：

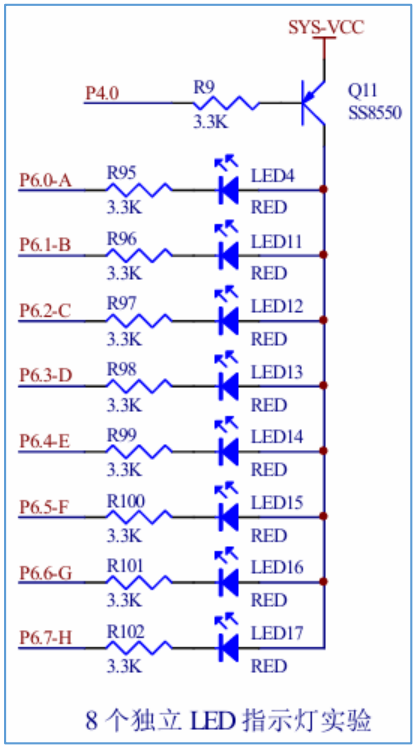
- ➢ 准双向口/弱上拉模式；
  对外输出低时，相当于强推挽输出低
- ➢ 强推挽输出模式；
- ➢ 开漏模式；

在对外输出低电平时，表现为强推挽输出的低电平状态。
在对外输出高电平时，表现为高阻输入状态，此时电流流不进来也流不出去。
如果用来控制 LED 灯，因为无电流流动，所以 LED 灯不亮。

5）P3.2/P3.3/P3.4/P3.5 端口均有按键，按下按键可接地，建议：
读取这些 I/O 口的状态，并将结果输出到 P6.2/P6.3/P6.4/P6.5 对应的指示灯上。

2.8.3 原理图

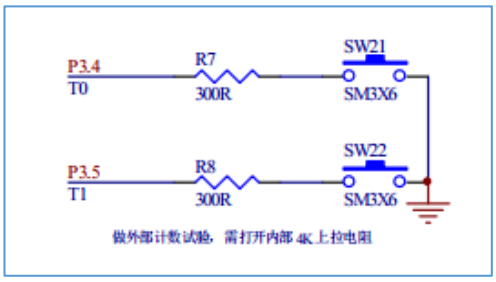
2.8.3.1 STC8H8K64U 实验箱原理图

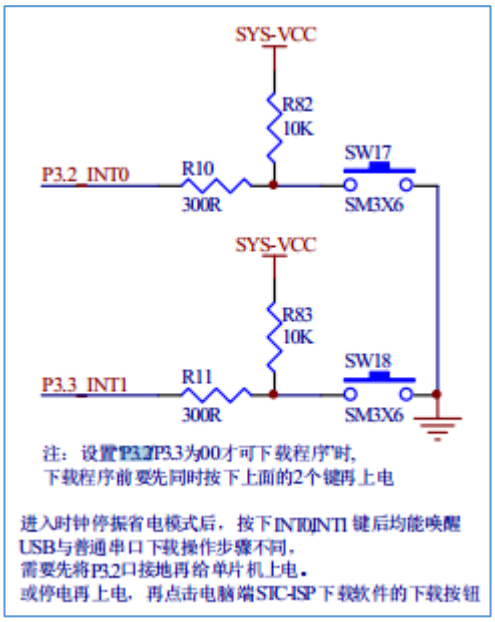


I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如果 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P6.0 输出 0，LED60 灯就会被点亮；如 P6.0 输出 1，LED60 灯就会熄灭；
如 P6.1 输出 0，LED61 灯就会被点亮；如 P6.1 输出 1，LED61 灯就会熄灭；
如 P6.2 输出 0，LED62 灯就会被点亮；如 P6.2 输出 1，LED62 灯就会熄灭；
如 P6.3 输出 0，LED63 灯就会被点亮；如 P6.3 输出 1，LED63 灯就会熄灭；
如 P6.4 输出 0，LED64 灯就会被点亮；如 P6.4 输出 1，LED64 灯就会熄灭；
如 P6.5 输出 0，LED65 灯就会被点亮；如 P6.5 输出 1，LED65 灯就会熄灭；
如 P6.6 输出 0，LED66 灯就会被点亮；如 P6.6 输出 1，LED66 灯就会熄灭；
如 P6.7 输出 0，LED67 灯就会被点亮；如 P6.7 输出 1，LED67 灯就会熄灭

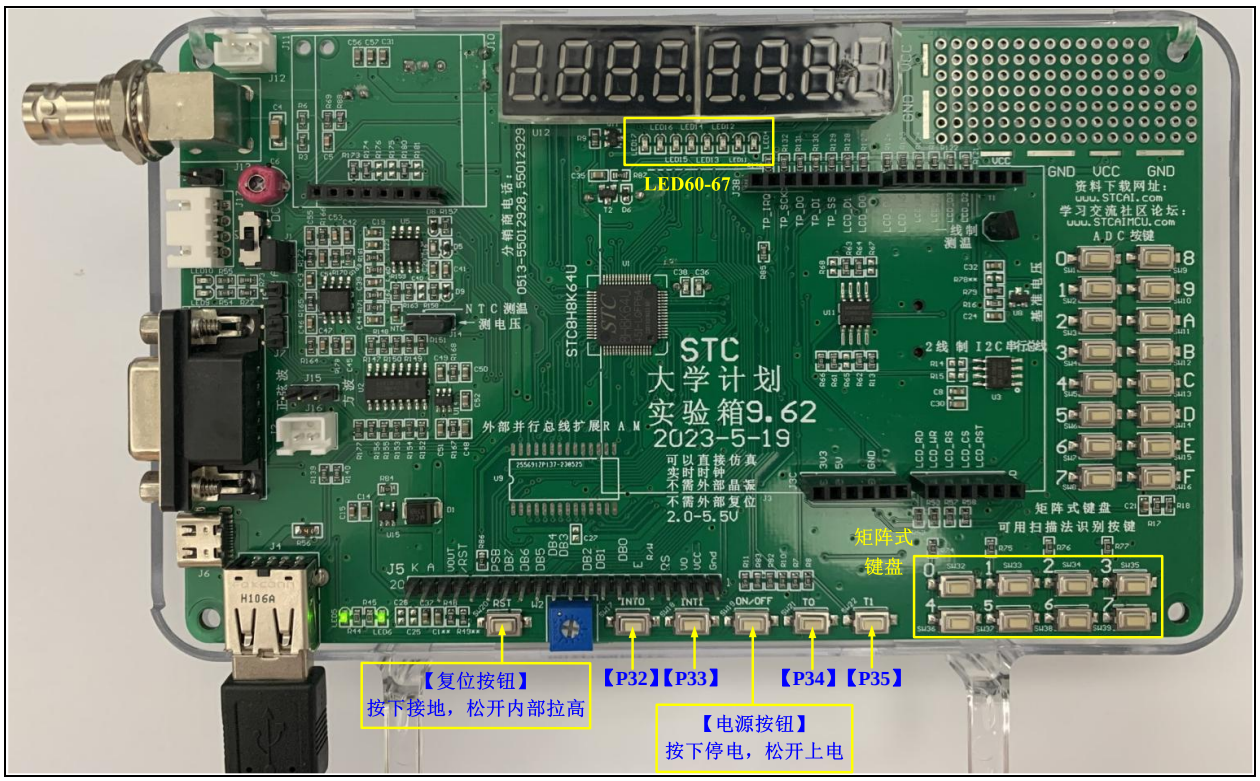
如 P6 口对外全部送 0，则这 8 个灯全亮；
如 P6 口对外全部送 1，则这 8 个灯全灭

如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P6 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。





2.8.3.2　　认识实验箱



2.8.4 I/O 口工作模式和内部上拉/下拉电阻介绍

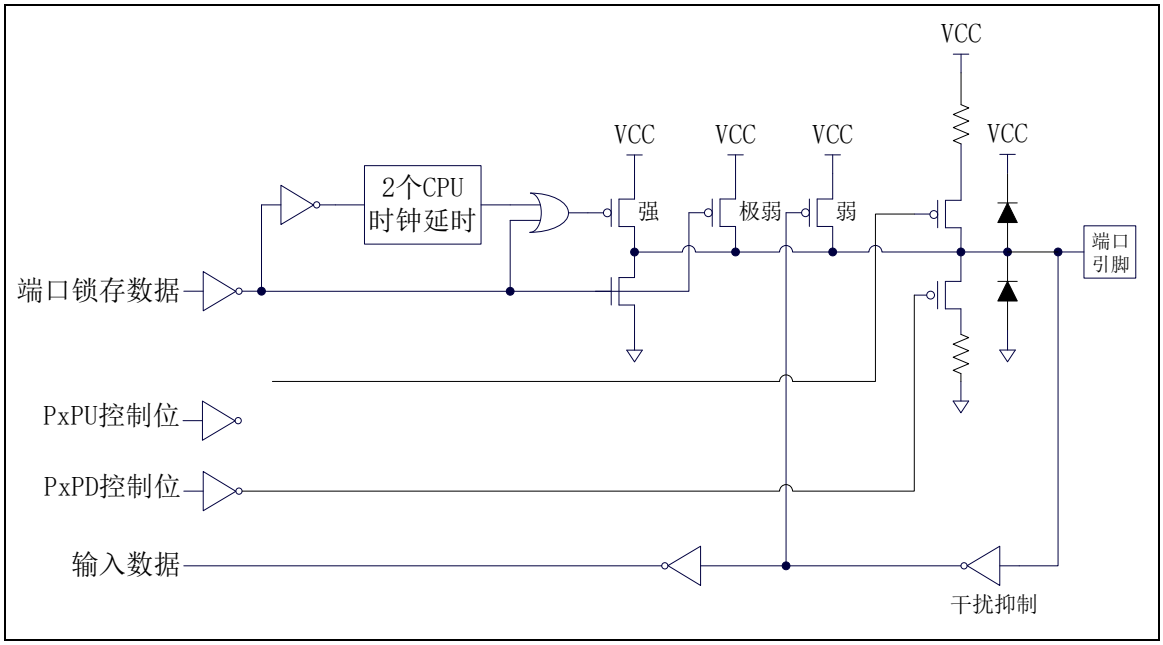
2.8.4.1 内部上拉电阻和外部上拉/下拉电阻的部分区别

外部上拉电阻随系统电源从 0V 上升到 MCU-VCC，到 3.3V 或 5V，这个时间是足够的长，系统上电复位后到初始化运行用户程序时，外部上拉电阻已将外部拉高到高电平。

内部上拉电阻是运行用户程序时才使能打开的，由于外部电路的分布电容效应，要等待 0.7RC 时间（R:电阻，C:电容）才能将外部拉高到 0.7MCU-VCC 高电平。建议将需要打开的内部上拉电阻使能后，直接多等待一会，如等待 1ms 后，就可认为外部已被拉高。

2.8.4.2 准双向口介绍

准双向口（弱上拉）输出类型可用作输出和输入功能而不需重新配置端口输出状态。这是因为当端口输出为 1 时驱动能力很弱，允许外部装置将其拉低。当引脚输出为低时，它的驱动能力很强，可吸收相当大的电流。准双向口输出低电平的驱动能力和强推挽的驱动能力是一样的。此处的上拉电阻和下拉电阻是独立的，和 I/O 口 4 种功能模式无关。



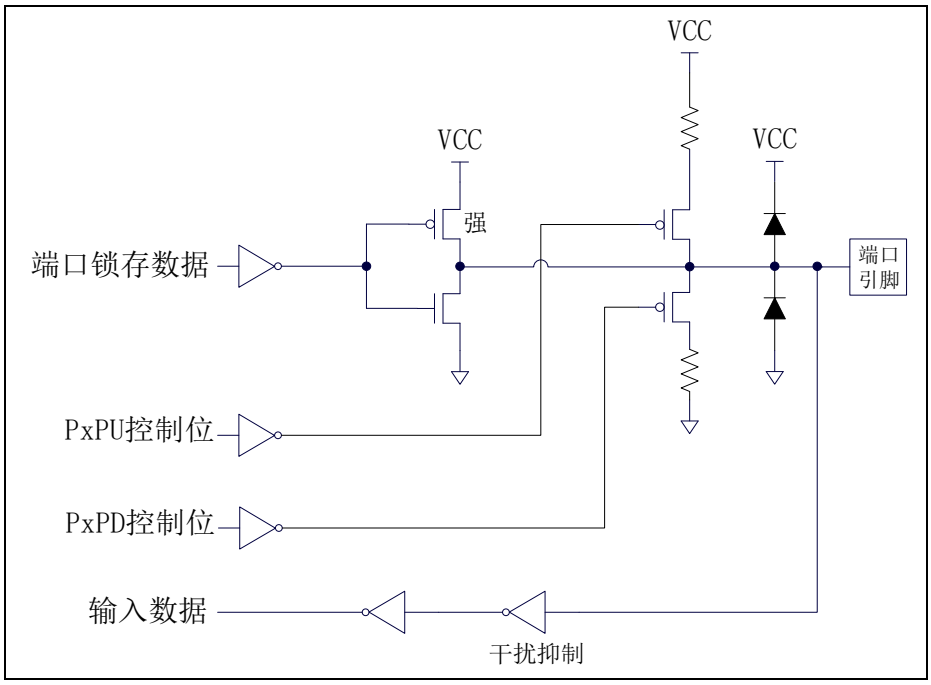
上拉电阻 / 电压	设计目标	实际测量
5.0V	4.0K	4.2K
3.3V	6.3K	5.8K

下拉电阻 / 电压	设计目标	实际测量
5.0V	47K	46K
3.3V	32K	35K

2.8.4.3 强推挽输出

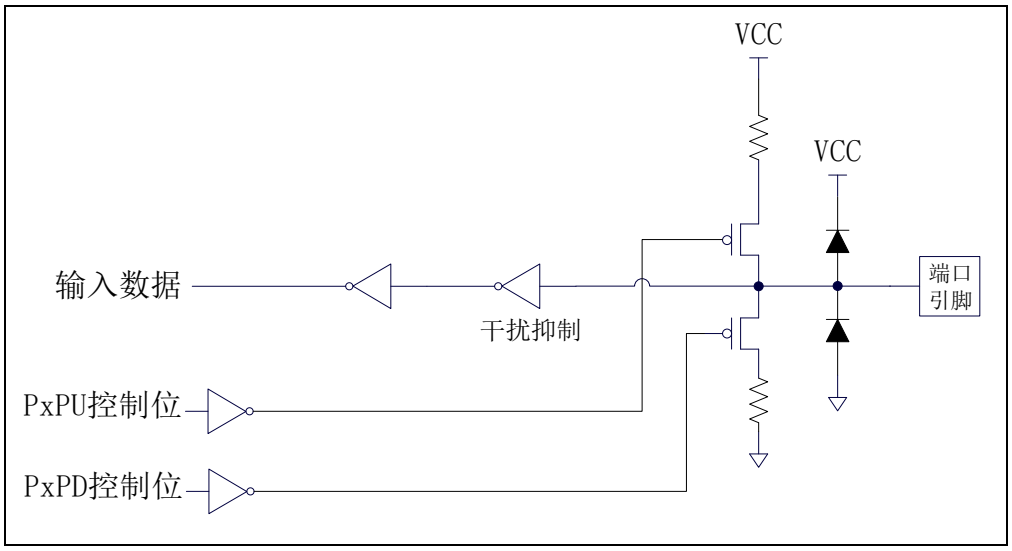
强推挽输出配置的下拉结构与开漏模式以及准双向口的下拉结构相同。

当对外输出锁存器设置为 0 时，强推挽的输出配置为强下拉结构，与开漏模式及准双向口的下拉是同一个 MOS 管，驱动能力是一样的。当对外输出锁存器设置为 1 时，对外输出强上拉。强推挽模式一般用于需要大驱动电流的情况。



2.8.4.4 高阻输入

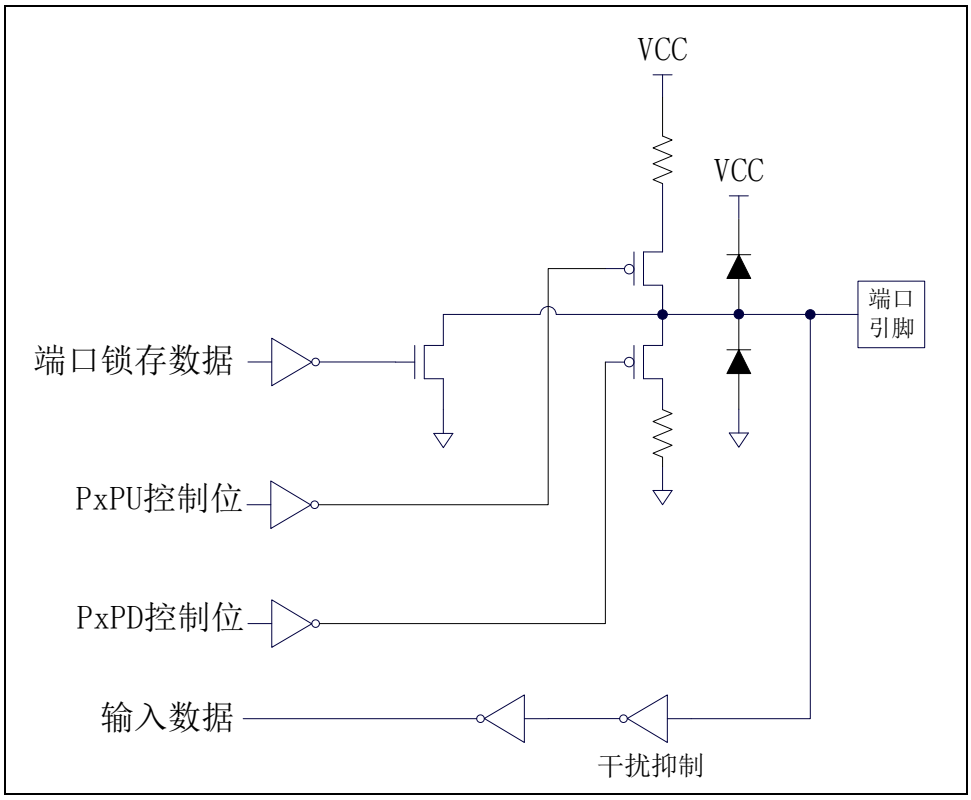
电流既不能流入也不能流出
输入口带有一个施密特触发输入以及一个抗干扰抑制电路



2.8.4.5 开漏模式

开漏工作模式，对外设置输出为 1，等同于高阻输入
开漏工作模式，打开内部上拉电阻 | 或外部加上拉电阻，简单等同于 准双向口

开漏模式如果打开内部上拉电阻或外接上拉电阻，可准确的读到外部状态。开漏模式对外可直接输出 0，但如果要输出 1，需打开内部上拉电阻或外接上拉电阻。



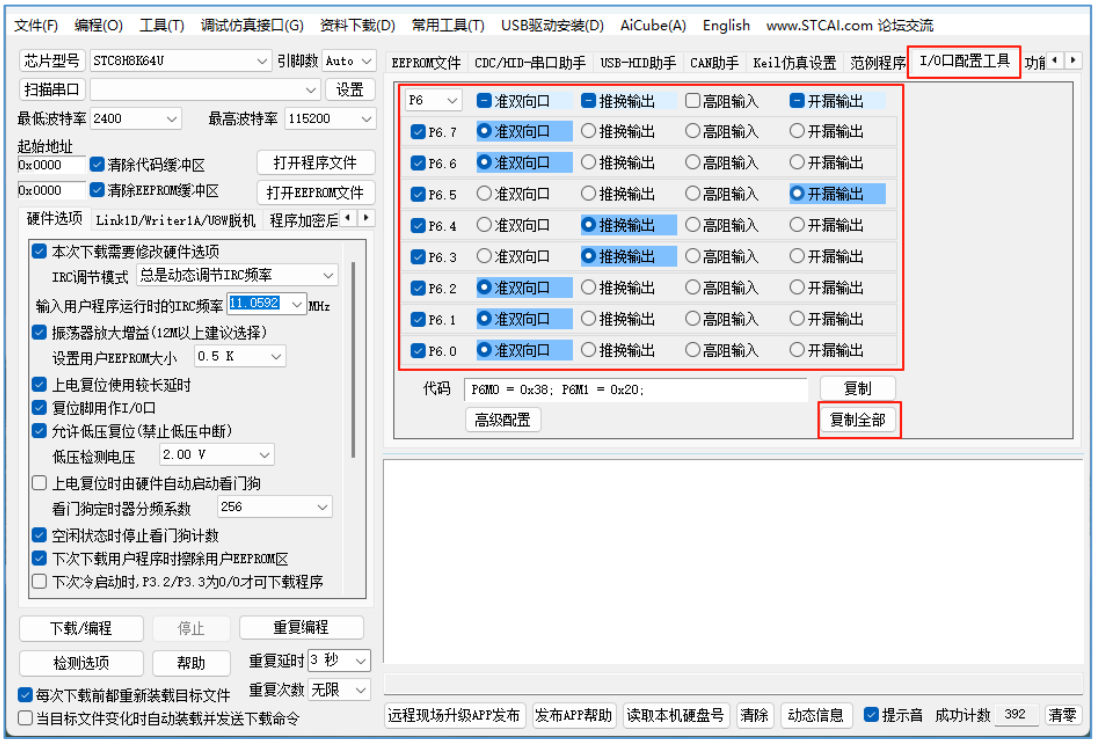
2.8.5　实验 1：位操作，用 USB 直接仿真来调试

单独读【P3.2 / P3.3 / P3.4 / P3.5】4 个独立 I/O 的状态，输出到【P6.2 / P6.3 / P6.4 / P6.5】对应的指示灯

2.8.5.1　I/O 口设置

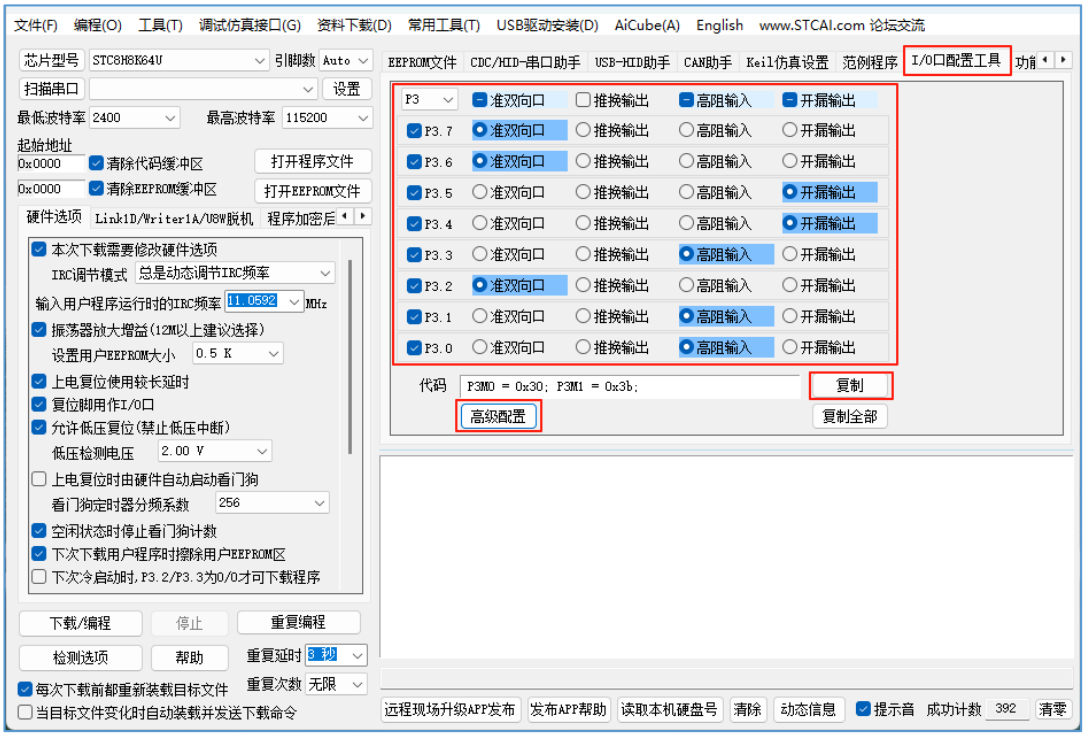
将不同的 I/O 口设置为不同的模式，尝试不同情况下依然可以正常输入/输出。打开 AiCube-ISP 软件：

➲ 配置输出端口的 P63、P64 为推挽输出，P65 为开漏输出（可以输出 0 点亮 LED）



➲ 输入配置

- P30、P31 为高阻输入(因为跟 USB 复用，设置为高阻输入，不干扰 USB 的通讯控制)
- 设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（设置为 1 等效高阻输入）
- 开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）



2.8.5.2　　实验程序 C 语言代码

```
#include <STC8H.H>          //包含 STC8H 的头文件
#include "intrins.h"        //使用_nop_()函数所必须要包含的头文件,
                            //否则延时函数中调用的_nop_()函数没有被头文件引用过来,
                            //会导致编译器找不到这个而函数而报错。

void Delay1ms(void)         //延时 1ms@11.0592MHz
{
    unsigned char data i, j;

    i = 15;
    j = 90;
    do
    {
        while (--j);
    } while (--i);
}

void main(void)
{
    P_SW2 |= 0x80;              //允许访问扩展的特殊寄存器，XFR
```

```
    P0M0 = 0x00; P0M1 = 0x00;     //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;     //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //设置 P2 口为准双向口模式

    P3M0 = 0x30; P3M1 = 0x3b;
    //P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出
    P3PU = 0x38;      //打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态
    P3 |= 0x3c;       //需要读的 I/O，对外输出锁存器先设置为 1，0011 1100B（0x3c）先和 P3 口逻辑或一下

    P4M0 = 0x00; P4M1 = 0x00;     //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;     //设置 P5 口为准双向口模式

    P6M0 = 0x38; P6M1 = 0x20;     //设置 P63、P64 为推挽输出，P65 为开漏输出，其余为准双向口模式

    P7M0 = 0x00; P7M1 = 0x00;     //设置 P7 口为准双向口模式

    Delay1ms();                   //延时 1ms，等待 I/O 口电平稳定，因为前面刚改变了模式
                                  //立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读

    EA = 1;                       //总中断允许位打开

    P40 = 0;                      //打开 LED 部分的供电

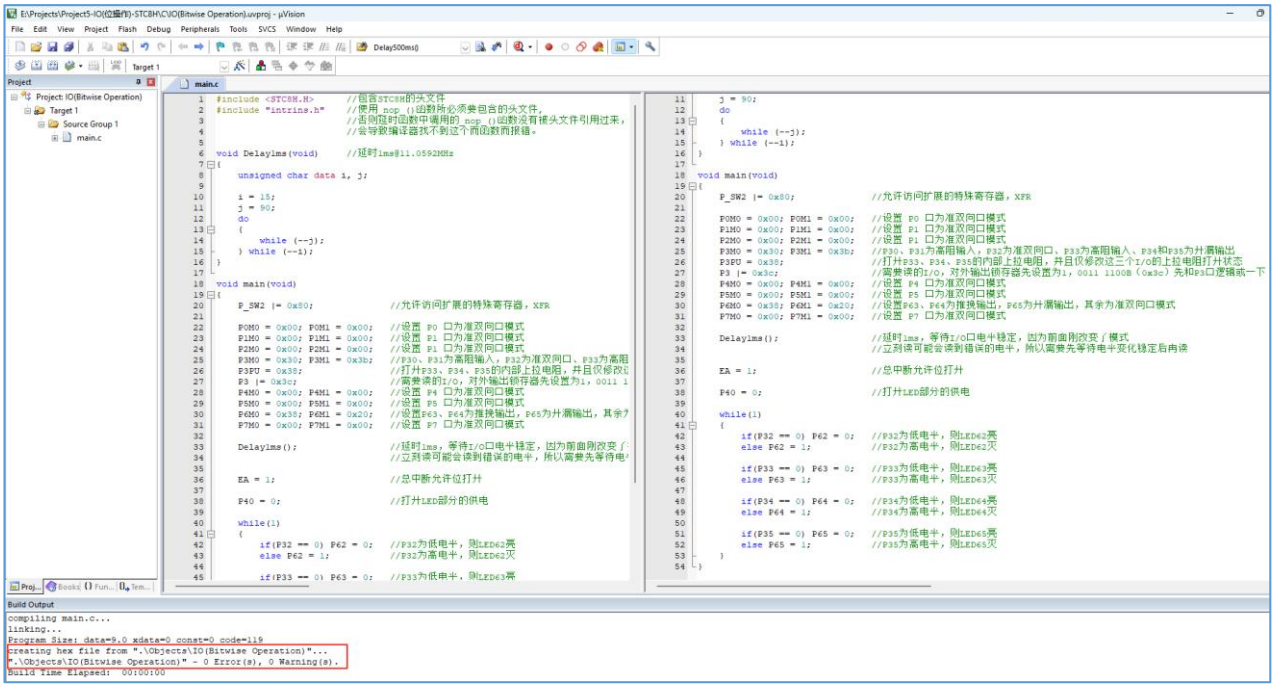
    while(1)
    {
        if(P32 == 0) P62 = 0;     //P32 为低电平，则 LED62 亮
        else P62 = 1;             //P32 为高电平，则 LED62 灭

        if(P33 == 0) P63 = 0;     //P33 为低电平，则 LED63 亮
        else P63 = 1;             //P33 为高电平，则 LED63 灭

        if(P34 == 0) P64 = 0;     //P34 为低电平，则 LED64 亮
        else P64 = 1;             //P34 为高电平，则 LED64 灭

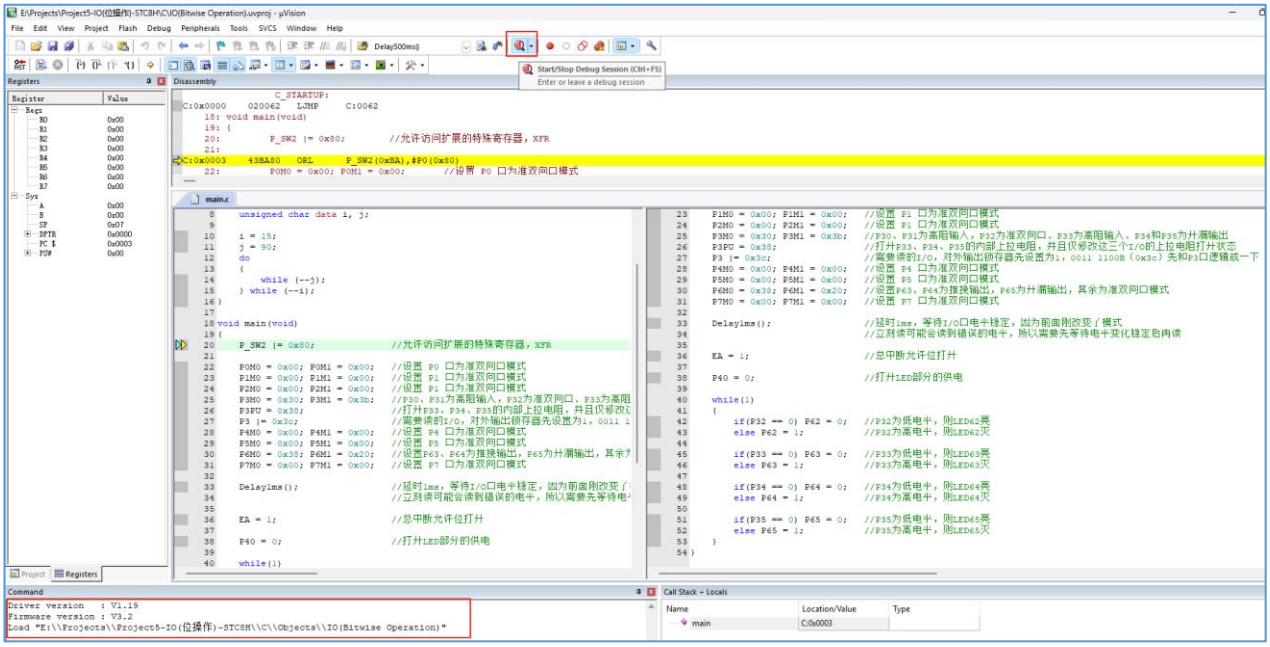
        if(P35 == 0) P65 = 0;     //P35 为低电平，则 LED65 亮
        else P65 = 1;             //P35 为高电平，则 LED65 灭
    }
}
```

录入代码，保存，编译。如下图：



2.8.5.3　　制作仿真芯片、仿真设置

- 制作仿真芯片：连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，在【Keil 仿真设置】选项卡中，按步骤操作“将所选目标单片机设置为仿真芯片”。
- 仿真设置：回到 Keil 界面，在 Keil 中按步骤在“Options for Target”的【Debug】选项卡和【Utilities】选项卡中完成使用“STC Monitor-251 Driver”的 STC8H8K64U 自带的 USB 仿真设置。
- 点击 File Toolbar 上的“Start/Stop Debug Session”按钮，进入仿真界面，如下图：



2.8.5.4　　观察实验现象

- 点击 Debug Toolbar 上的“Run”按钮：
  - 按下 STC8H8K64U 实验箱上的 P3.2 按键，对应的 LED62 灯亮；松开 P3.2 按键，LED62 灯灭。

- 按下 STC8H8K64U 实验箱上的 P3.3 按键，对应的 LED63 灯亮；松开 P3.3 按键，LED63 灯灭。
- 按下 STC8H8K64U 实验箱上的 P3.4 按键，对应的 LED64 灯亮；松开 P3.4 按键，LED64 灯灭。
- 按下 STC8H8K64U 实验箱上的 P3.5 按键，对应的 LED65 灯亮；松开 P3.5 按键，LED65 灯灭。

完成实验。

2.8.5.5 实验程序汇编语言代码

```
;IO 模式读取按键状态并点灯(位操作)

$INCLUDE(STC8H.INC)                 ;包含 STC8H 的头文件

STACK_POIRTER     EQU     0D0H      ;堆栈开始地址

    ORG     0000H                   ;程序复位地址
    LJMP    START
    ORG     0100H

START:
    MOV     P_SW2, #80H             ;允许访问扩展的特殊寄存器，XFR
    MOV     SP, #STACK_POIRTER      ;堆栈栈顶设置

    ;端口初始化
    MOV     P0M0, #00H      ;设置 P0 口为准双向口模式
    MOV     P0M1, #00H
    MOV     P1M0, #00H      ;设置 P1 口为准双向口模式
    MOV     P1M1, #00H
    MOV     P2M0, #00H      ;设置 P2 口为准双向口模式
    MOV     P2M1, #00H

    MOV     P3M0, #30H
            ;P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出
    MOV     P3M1, #3BH

    MOV     P4M0, #00H      ;设置 P4 口为准双向口模式
    MOV     P4M1, #00H
    MOV     P5M0, #00H      ;设置 P5 口为准双向口模式
    MOV     P5M1, #00H

    MOV     P6M0, #38H      ;设置 P63、P64 为推挽输出，P65 为开漏输出，其余为准双向口模式
    MOV     P6M1, #20H

    MOV     P7M0, #00H      ;设置 P7 口为准双向口模式
    MOV     P7M1, #00H

    MOV     DPTR, #P3PU
```

```
        ;打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态
    MOV     A, #38H
    MOVX    @DPTR, A

    ORL     P3, #3CH
        ;需要读的 I/O，对外输出锁存器先设置为 1，0011 1100B（0x3c）先和 P3 口逻辑或一下

    LCALL   DELAY1MS        ;延时 1ms，等待 IO 口电平稳定，因为前面刚改变了模式
                            ;立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读

    CLR     P40             ;打开 LED 部分的供电
;------------------------------------------------
MAINLOOP:
    JNB     P3.2, P32ISLOW  ;判断 P32 的电平

P32ISHIGH:
    SETB    P6.2            ;P32 为高电平，则 LED62 灭
    SJMP    CHECKP33        ;跳转到判断 P33

P32ISLOW:
    CLR     P6.2            ;P32 为低电平，则 LED62 亮

CHECKP33:
    JNB     P3.3, P33ISLOW  ;判断 P33 的电平

P33ISHIGH:
    SETB    P6.3            ;P33 为高电平，则 LED63 灭
    SJMP    CHECKP34        ;跳转到判断 P34

P33ISLOW:
    CLR     P6.3            ;P33 为低电平，则 LED63 亮

CHECKP34:
    JNB     P3.4, P34ISLOW  ;判断 P34 的电平

P34ISHIGH:
    SETB    P6.4            ;P34 为高电平，则 LED64 灭
    SJMP    CHECKP35        ;跳转到判断 P35

P34ISLOW:
    CLR     P6.4            ;P34 为低电平，则 LED64 亮

CHECKP35:
    JNB     P3.5, P35ISLOW  ;判断 P35 的电平

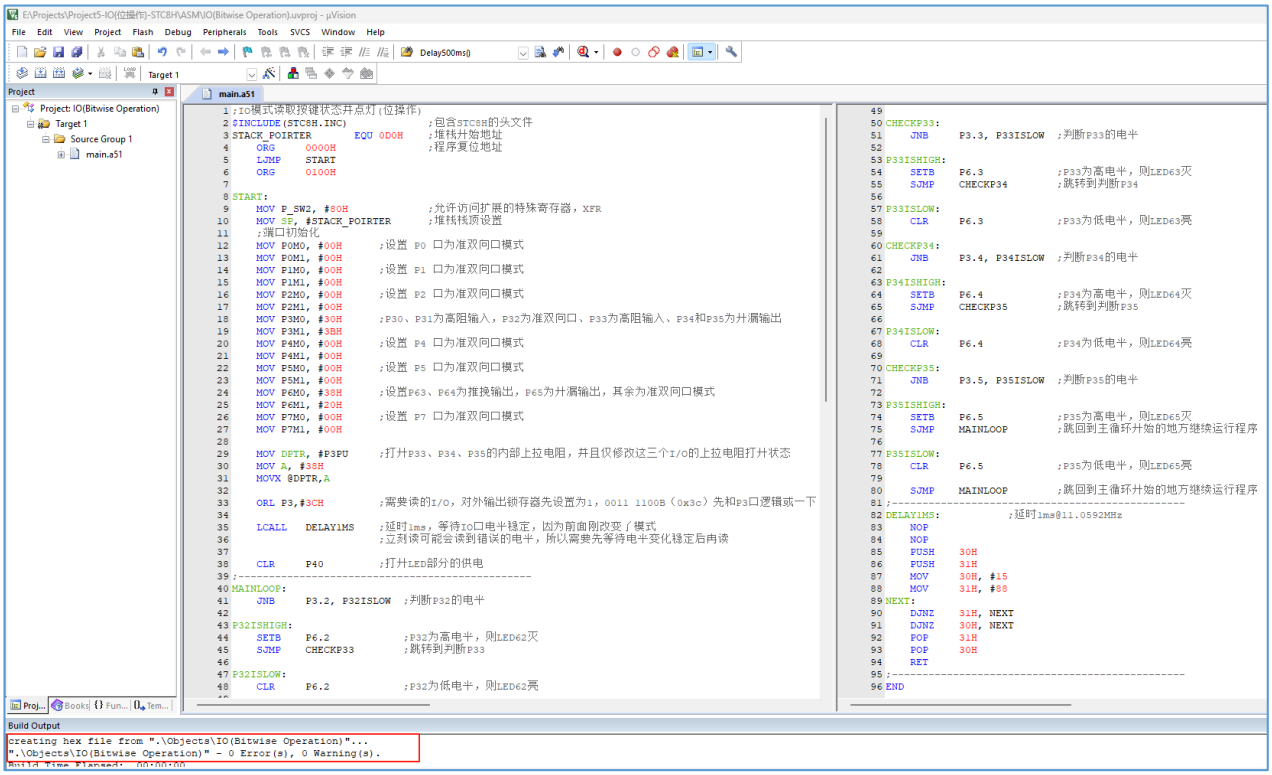
P35ISHIGH:
```

```
    SETB    P6.5            ;P35 为高电平，则 LED65 灭
    SJMP    MAINLOOP        ;跳回到主循环开始的地方继续运行程序

P35ISLOW:
    CLR     P6.5            ;P35 为低电平，则 LED65 亮

    SJMP    MAINLOOP        ;跳回到主循环开始的地方继续运行程序
;--------------------------------------------
DELAY1MS:           ;@11.0592MHz
    NOP
    NOP
    PUSH    30H
    PUSH    31H
    MOV     30H, #15
    MOV     31H, #88
NEXT:
    DJNZ    31H, NEXT
    DJNZ    30H, NEXT
    POP     31H
    POP     30H
    RET
;--------------------------------------------
    END
```

录入代码，保存，编译。



制作仿真芯片，仿真设置

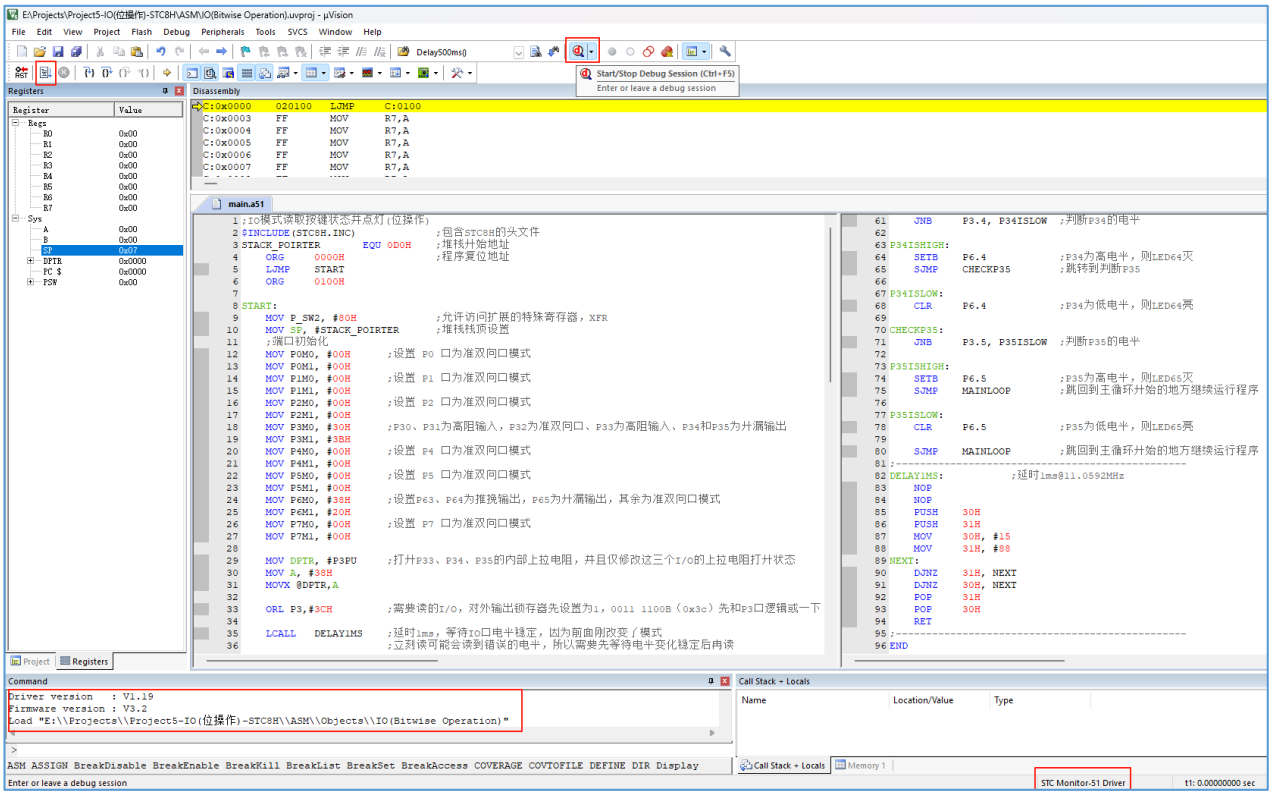
➲ 制作仿真芯片：连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，在【Keil 仿真设置】选

项卡中，按步骤操作“将所选目标单片机设置为仿真芯片”。

➲ 仿真设置：回到 Keil 界面，在 Keil 中按步骤在“Options for Target”的【Debug】选项卡和【Utilities】选项卡中完成使用“STC Monitor-251 Driver”的 USB 仿真设置。

【关于制作仿真芯片及相关设置，详细内容请查阅本文：USB 直接通信仿真，STC8H8K64U 仿真】

➲ 点击 File Toolbar 上的“Start/Stop Debug Session”按钮，进入仿真界面，如下图：



观察实验现象

➲ 点击 Debug Toolbar 上的“Run”按钮：

- 按下 STC8H8K64U 实验箱上 P3.2 按键，对应的 LED62 灯亮；松开 P3.2 按键，LED62 灯灭。
- 按下 STC8H8K64U 实验箱上 P3.3 按键，对应的 LED63 灯亮；松开 P3.3 按键，LED63 灯灭。
- 按下 STC8H8K64U 实验箱上 P3.4 按键，对应的 LED64 灯亮；松开 P3.4 按键，LED64 灯灭。
- 按下 STC8H8K64U 实验箱上 P3.5 按键，对应的 LED65 灯亮；松开 P3.5 按键，LED65 灯灭。

完成实验。

2.8.6 实验 1A--使用【AiCube】项目助手实现“位操作，用 USB 仿真来点灯”

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit、Ai8051U-8Bit、STC32G12K128、STC8H8K64U、STC15F2K60S2、STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

2.8.6.1 安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj



下载下来的是压缩包文件，图标如下图：



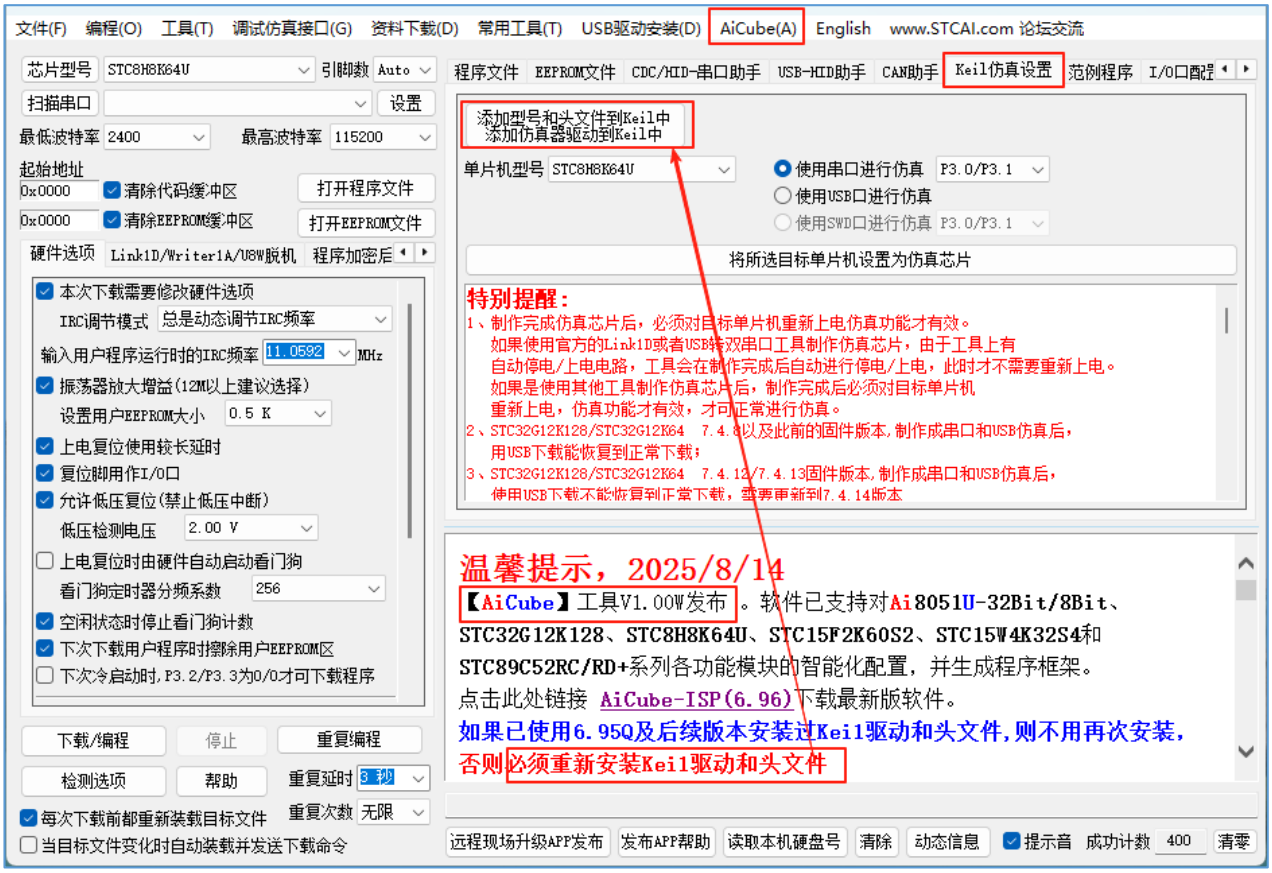
解压缩后的 AiCube-ISP 文件，图标如下图：



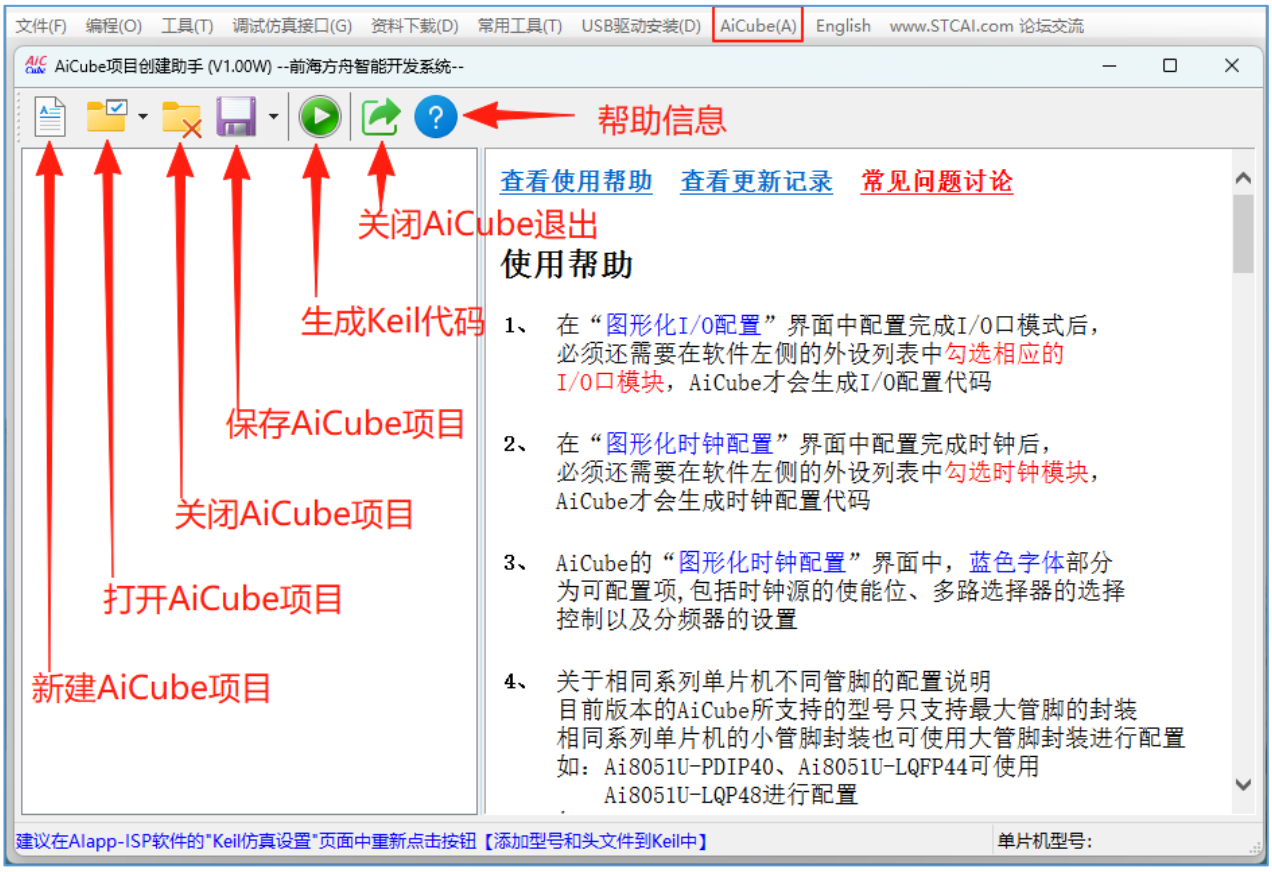
如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

2.8.6.2　使用【AiCube】助手创建项目

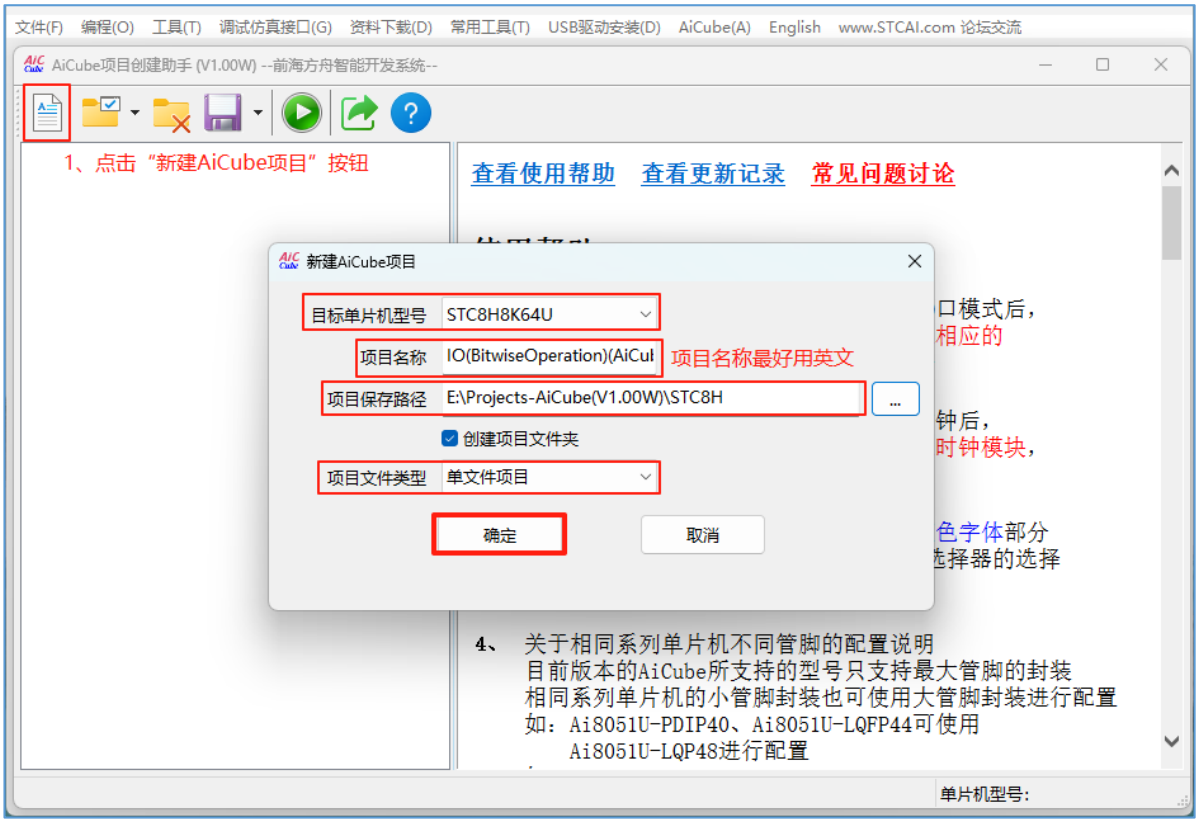
➲　双击图标，打开 AiCube-ISP 软件界面，如下图：



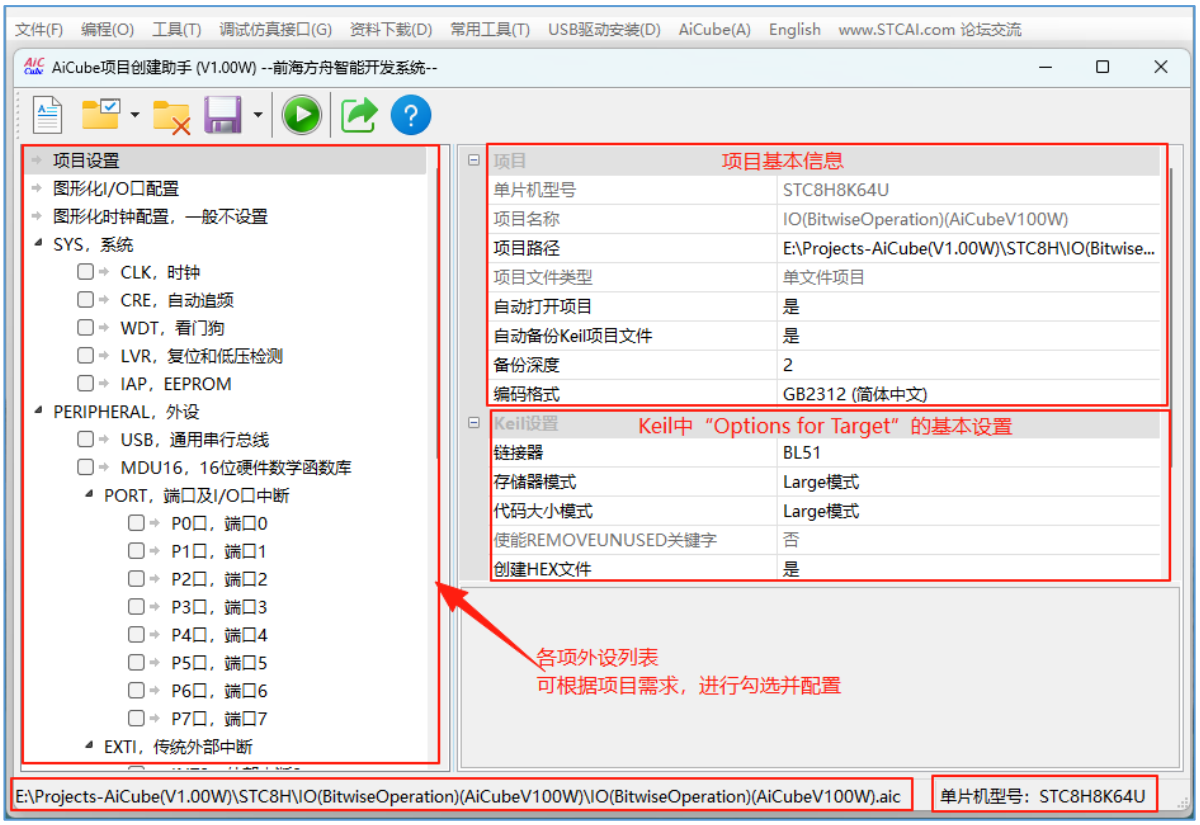
➲　点击【AiCube（A）】，弹出如下窗口：



➲ 点击“新建 AiCube 项目”按钮，选择并填写项目基本信息，建议项目名称用英文！



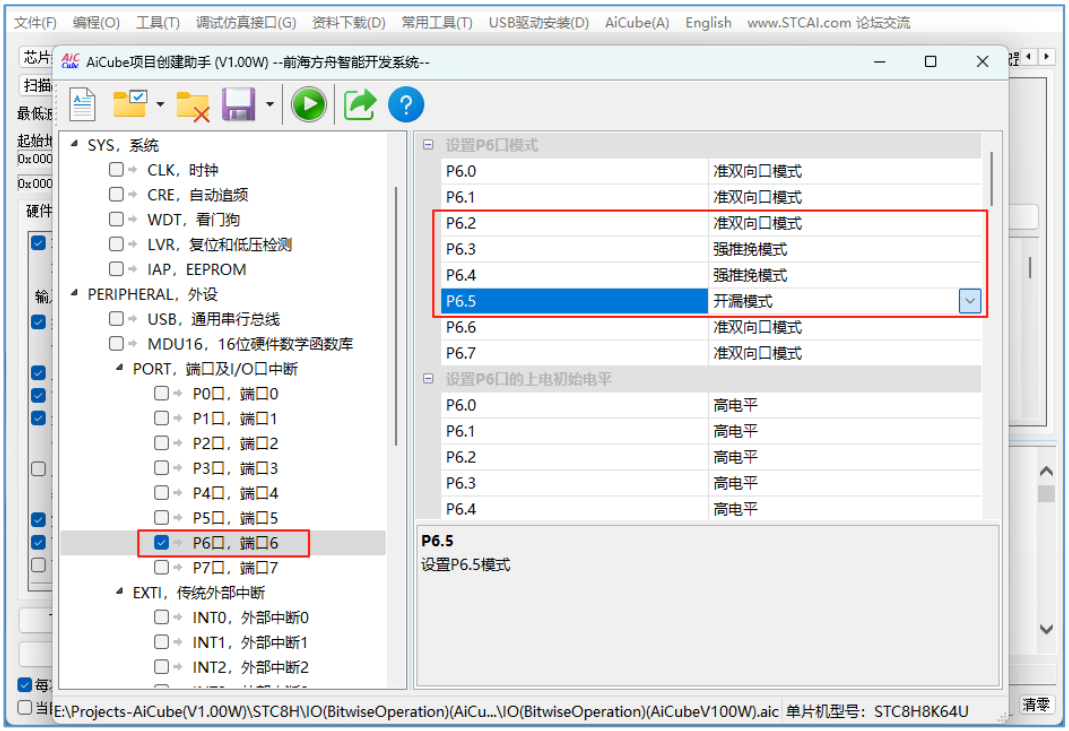
➲ 点击“确定”，出现如下配置界面：



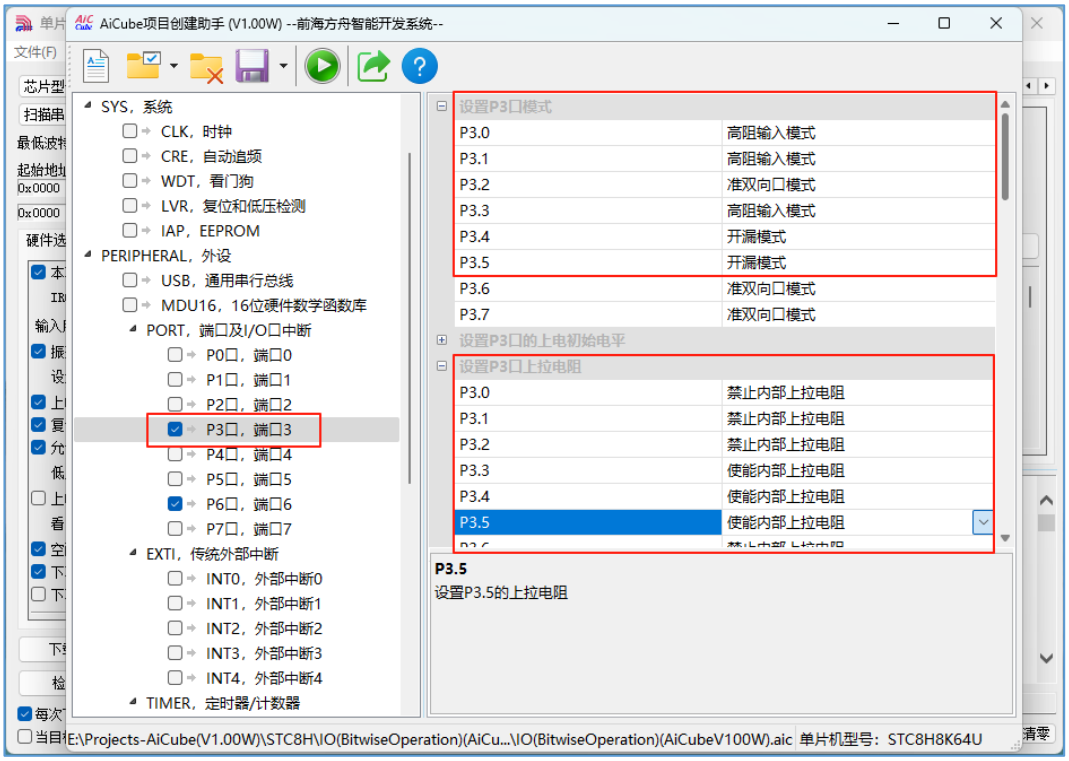
对于本项目，需要单片机读【P3.2 / P3.3 / P3.4 / P3.5】这 4 个 I/O 的状态，
输出到【P6.2 / P6.3 / P6.4 / P6.5】对应的指示灯

针对 I/O 有 4 种不同的模式，我们将【P3.2 / P3.3 / P3.4 / P3.5】这 4 个不同的 I/O 口设置为不同的模式，以验证不同情况下依然可以正常输入/输出。

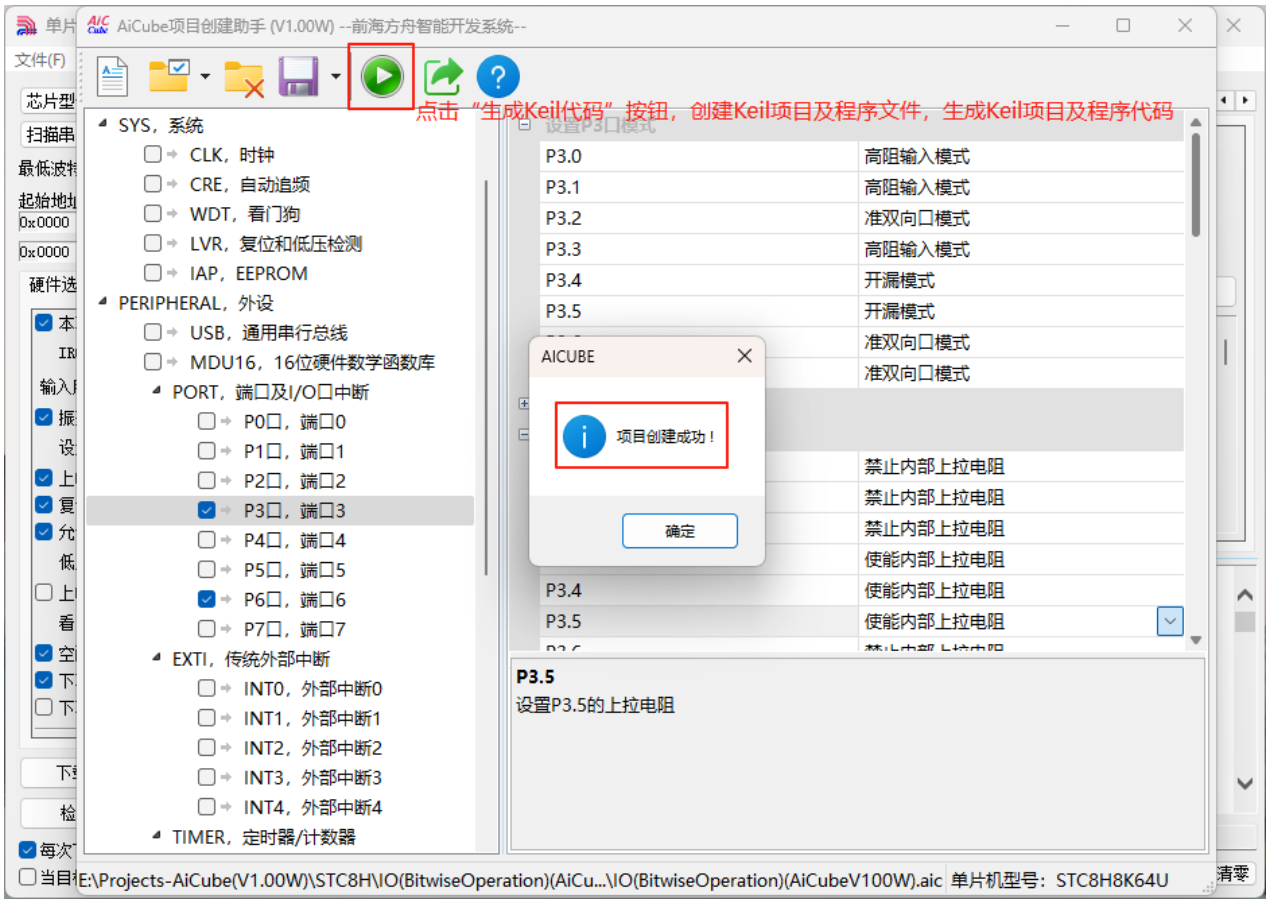
➲ 【AiCube】助手中，端口默认是准双向口模式，我们配置**输出端口**的 P62 保持准双向口模式，P63、P64 为强推挽输出，P65 为开漏输出（可以输出 0 点亮 LED），如下图：



➲ **输入配置** P30、P31 为高阻输入(因为跟 USB 复用，建议设置为高阻输入，不干扰 USB 通讯控制)
设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（设置为 1 等效高阻输入）
然后开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）
【AiCube】助手中，端口默认是准双向口模式，内部上拉电阻默认是禁用设置。现在我们按上述要求配置 P30、P31 为高阻输入；P32 为保持准双向口；P33 为高阻输入；P34、P35 为开漏模式；打开 P33、P34、P35 的内部上拉电阻。如下图：



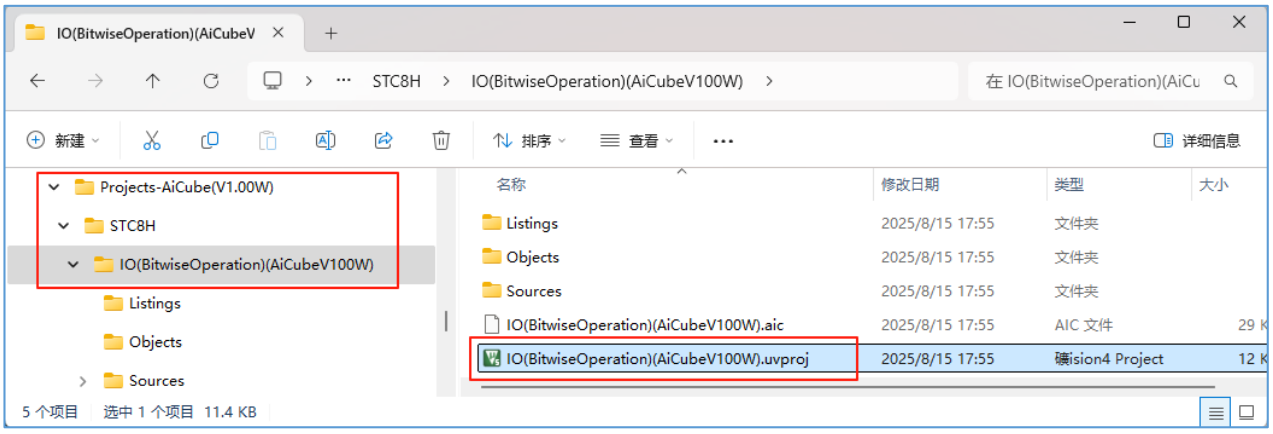
配置完毕，点击“创建 Keil 项目即程序文件”按钮，【AiCube】助手提示：“项目创建成功！”，如下图：



点击“确定”，立即自动打开 Keil 软件，进入该项目的 Keil 集成开发界面，即【AiCube】成功帮我们创建了项目及文件目录“IO(BitwiseOperation)(AiCubeV100W)”。至此，可以关闭【AiCube】。

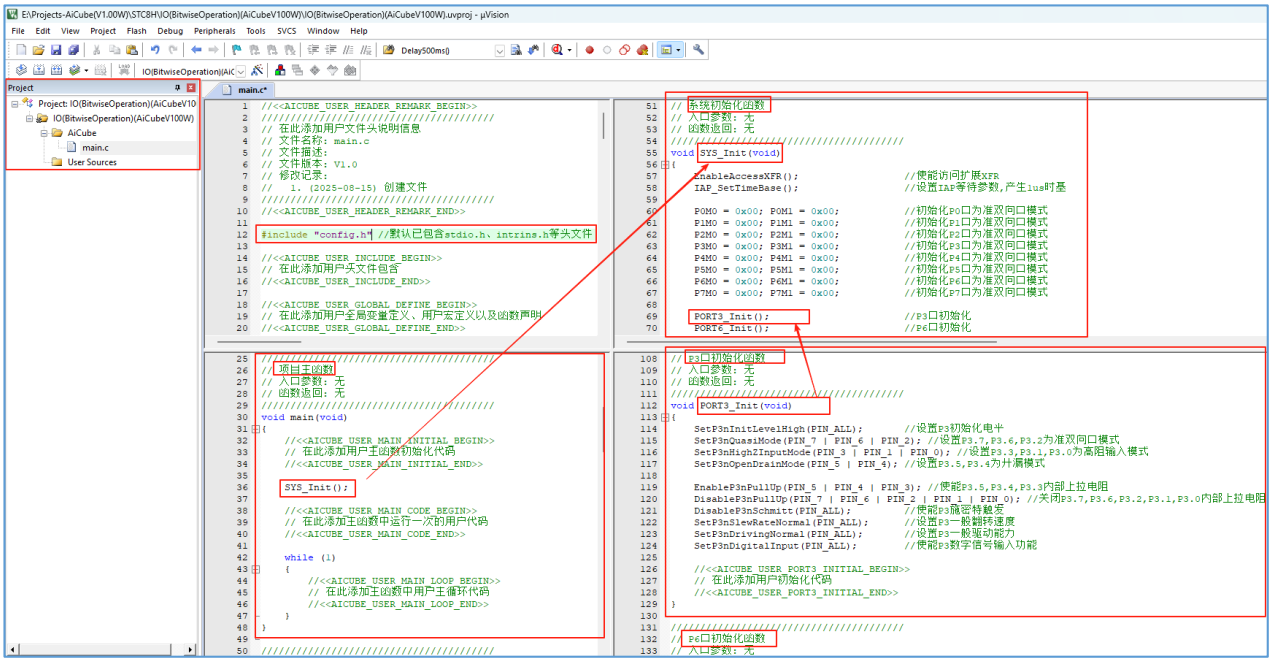
➲ 找到创建的项目目录

发现通过上述操作，【AiCube】助手已经帮我们在指定位置建立了项目目录及 IO(BitwiseOperation)(AiCubeV100W).uvproj 文件，如下图：



➲ 观察已打开的 Keil 项目文件：IO(BitwiseOperation)(AiCubeV100W).uvproj

如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，全代码观察）：

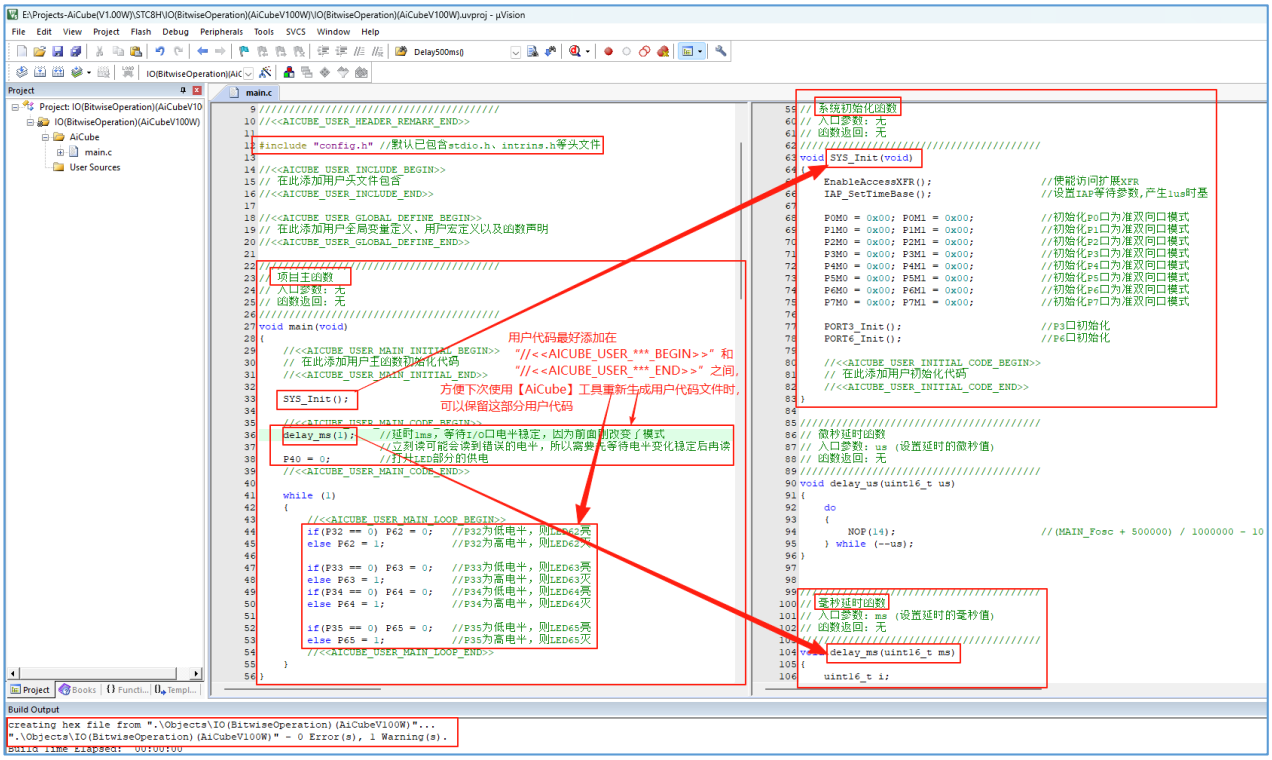


根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、毫秒和微秒延时函数、P3 口和 P6 口初始化函数。

2.8.6.3　　增加实验项目代码

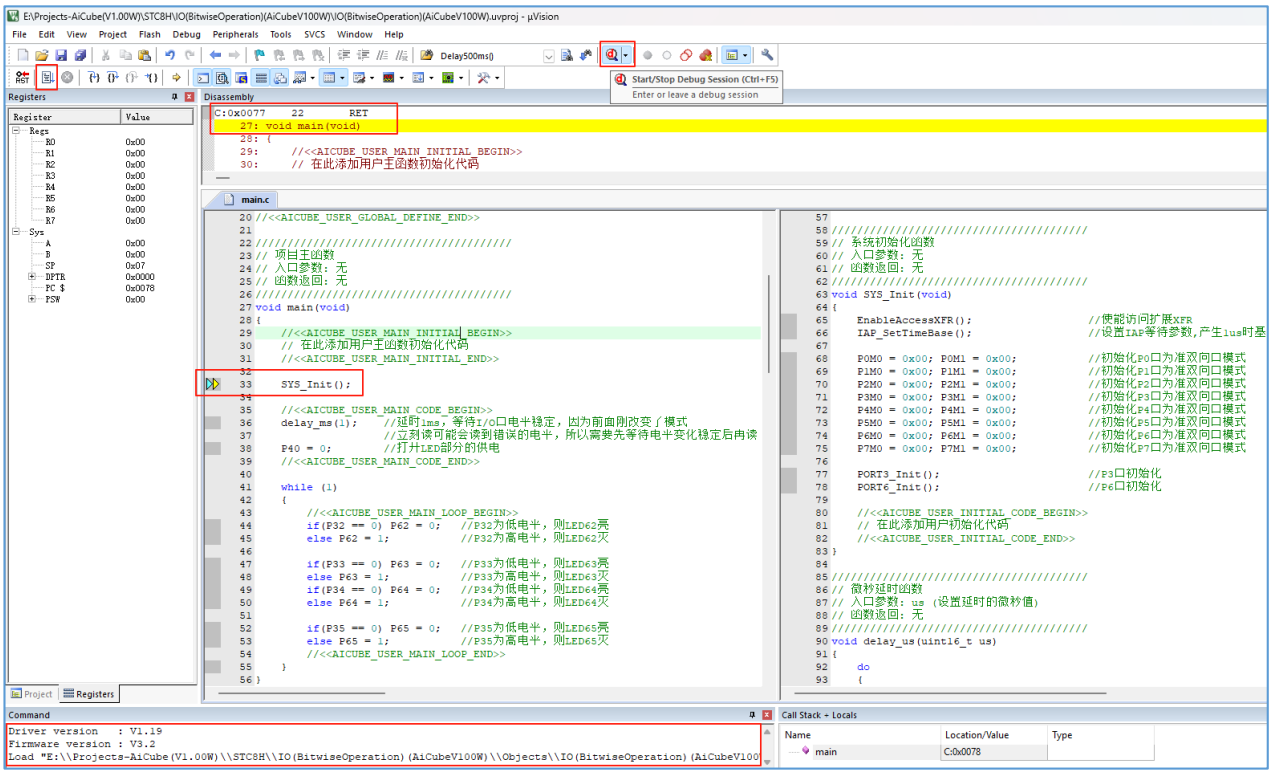
为实现“I/O 口位操作点灯，用 USB 仿真点灯”的实验，我们现在需要在主函数中：

- ➲ 加入调用延时函数“delay_ms(1)”延时 1ms（等待 I/O 口电平稳定）；
- ➲ 加入“P40 = 0”打开 LED 供电的代码；
- ➲ 在 while 循环中加入 I/O 口位操作点灯代码。
- ➲ 保存，编译，如下图（使用菜单“Window”→“Split”将原窗口拆分成两个窗口，对比观察）：



2.8.6.4　使用 AiCube-ISP 软件设置仿真芯片，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果

- 制作仿真芯片：连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，在“Keil 仿真设置”选项卡中，按步骤操作“将所选目标单片机设置为仿真芯片”。
- 仿真设置：回到 Keil 界面，在 Keil 中按步骤在“Options for Target”的 Debug 选项卡和 Utilities 选项卡中完成使用“STC Monitor-251 Driver”的 USB 仿真设置。
- 点击 File Toolbar 上的“Start/Stop Debug Session”按钮，进入仿真界面，如下图：



一切准备就绪。

- 点击 Debug Toolbar 上的“Run”按钮，
  - 按下 STC8H8K64U 实验箱上的 P3.2 按键，对应的 LED62 灯亮；松开 P3.2 按键，LED62 灯灭。
  - 按下 STC8H8K64U 实验箱上的 P3.3 按键，对应的 LED63 灯亮；松开 P3.3 按键，LED63 灯灭。
  - 按下 STC8H8K64U 实验箱上的 P3.4 按键，对应的 LED64 灯亮；松开 P3.4 按键，LED64 灯灭。
  - 按下 STC8H8K64U 实验箱上的 P3.5 按键，对应的 LED65 灯亮；松开 P3.5 按键，LED65 灯灭。

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.8.7 实验 2：整组 8 位端口操作，用 USB 直接仿真来调试

整组 读【P3.2/P3.3/P3.4/P3.5】这 4 个 I/O 的状态，一次性读全部 P3 口
整体 输出到【P6.2/P6.3/P6.4/P6.5】在的 P6 口对应的指示灯

2.8.7.1 实验程序 C 代码

```
#include <STC8H.H>          //包含 STC8H 的头文件
#include "intrins.h"        //使用_nop_()函数所必须要包含的头文件,
                            //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                            //会导致编译器找不到这个而函数而报错。

void Delay1ms(void)         //延时 1ms@11.0592MHz
{
    unsigned char data i, j;

    i = 15;
    j = 90;
    do
    {
        while (--j);
    } while (--i);
}

void main(void)
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式

    P3M0 = 0x30; P3M1 = 0x3b;
                    //P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出
    P3PU = 0x38;    //打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态
    P3 |= 0x3c;     //需要读的 I/O，对外输出锁存器先设置为 1，0011 1100B（0x3c）先和 P3 口逻辑或一下

    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式

    P6M0 = 0x38; P6M1 = 0x20;       //设置 P63、P64 为推挽输出，P65 为开漏输出，其余为准双向口模式

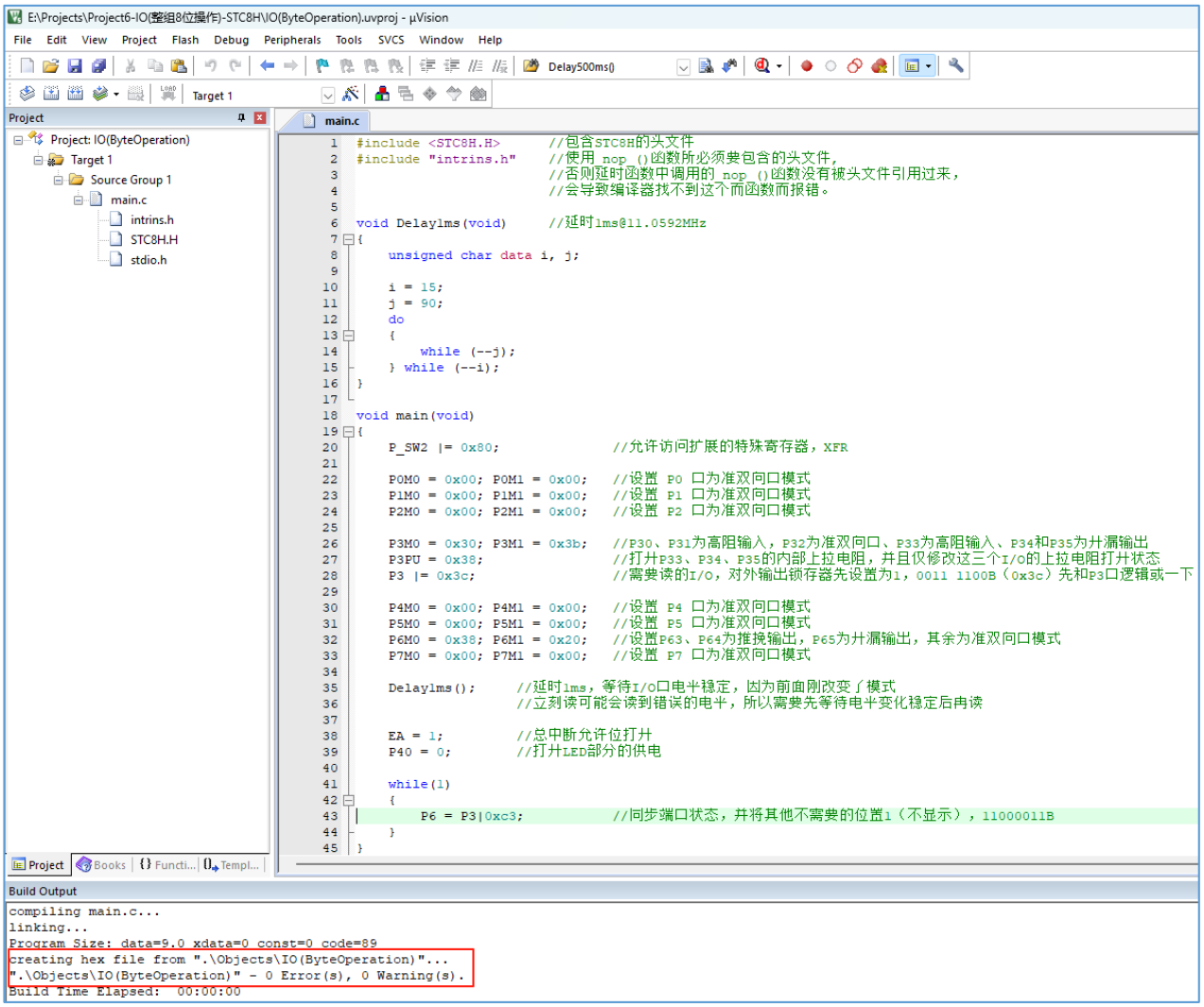
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    Delay1ms();                     //延时 1ms，等待 I/O 口电平稳定，因为前面刚改变了模式
                                    //立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读
```

```
    EA = 1;                         //总中断允许位打开
    P40 = 0;                        //打开 LED 部分的供电

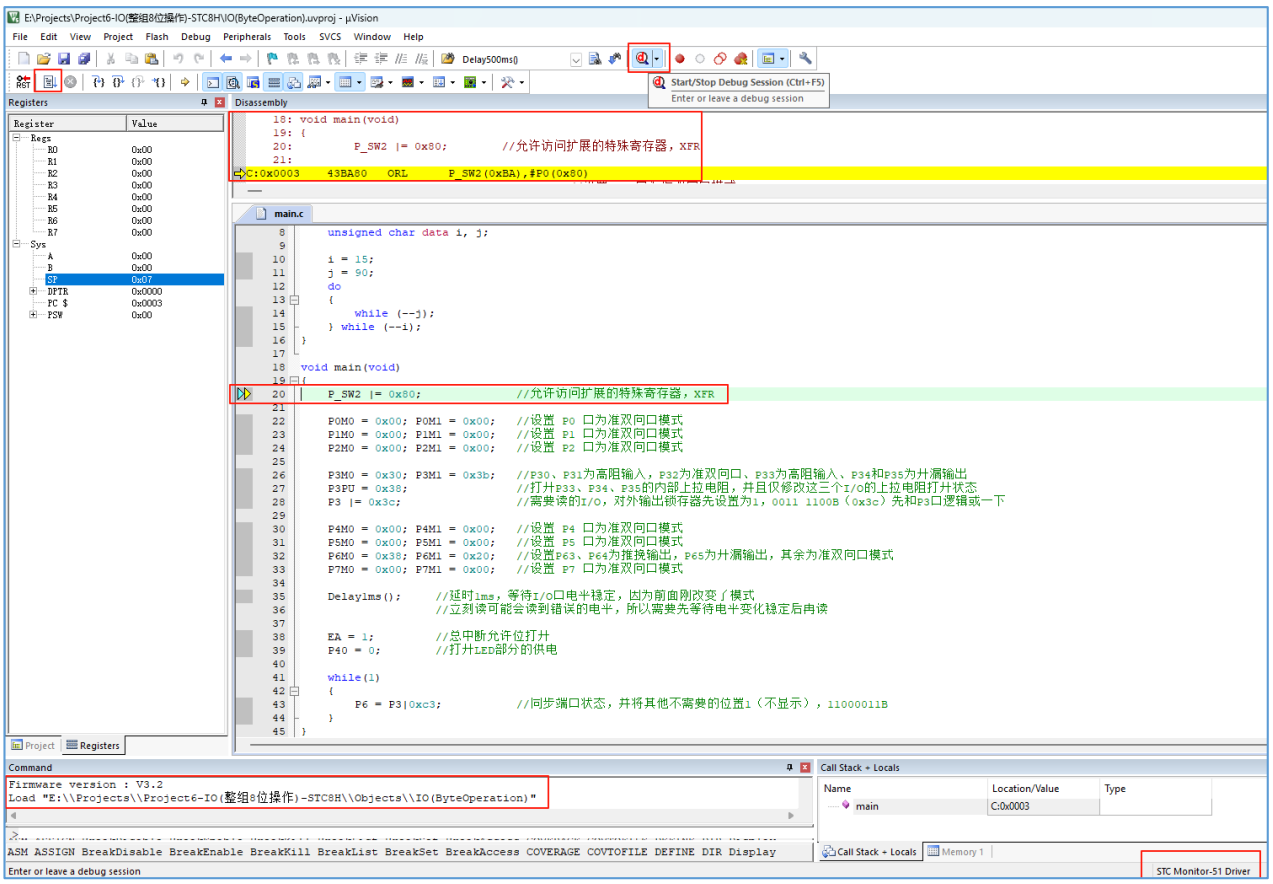
    while(1)
    {
        P6 = P3|0xc3;               //同步端口状态，并将其他不需要的位置 1（不显示），11000011B
    }
}
```

录入代码，保存，编译。



2.8.7.2　制作仿真芯片、仿真设置

- 制作仿真芯片：连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，在【Keil 仿真设置】选项卡中，按步骤操作“将所选目标单片机设置为仿真芯片”。
- 仿真设置：回到 Keil 界面，在 Keil 中按步骤在“Options for Target”的【Debug】选项卡和【Utilities】选项卡中完成使用“STC Monitor-251 Driver”的 STC8H8K64U 自带的 USB 仿真设置。
  【关于制作仿真芯片及相关设置，详细内容请查阅本文：USB 直接通信仿真，STC8H8K64U 仿真】
- 点击 File Toolbar 上的“Start/Stop Debug Session”按钮，进入仿真界面，如下图：



2.8.7.3　观察实验现象

- 点击 Debug Toolbar 上的“Run”按钮：
  - 按下 STC8H8K64U 实验箱上 P3.2 按键，对应的 LED62 灯亮；松开 P3.2 按键，LED62 灯灭。
  - 按下 STC8H8K64U 实验箱上 P3.3 按键，对应的 LED63 灯亮；松开 P3.3 按键，LED63 灯灭。
  - 按下 STC8H8K64U 实验箱上 P3.4 按键，对应的 LED64 灯亮；松开 P3.4 按键，LED64 灯灭。
  - 按下 STC8H8K64U 实验箱上 P3.5 按键，对应的 LED65 灯亮；松开 P3.5 按键，LED65 灯灭。

完成实验。

2.8.8 实验 2A--使用【AiCube】项目助手实现“整组 8 位端口操作，用 USB 仿真来点灯”

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit、Ai8051U-8Bit、STC32G12K128、STC8H8K64U、STC15F2K60S2、STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

2.8.8.1 安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj



下载下来的是压缩包文件图标，如下图：



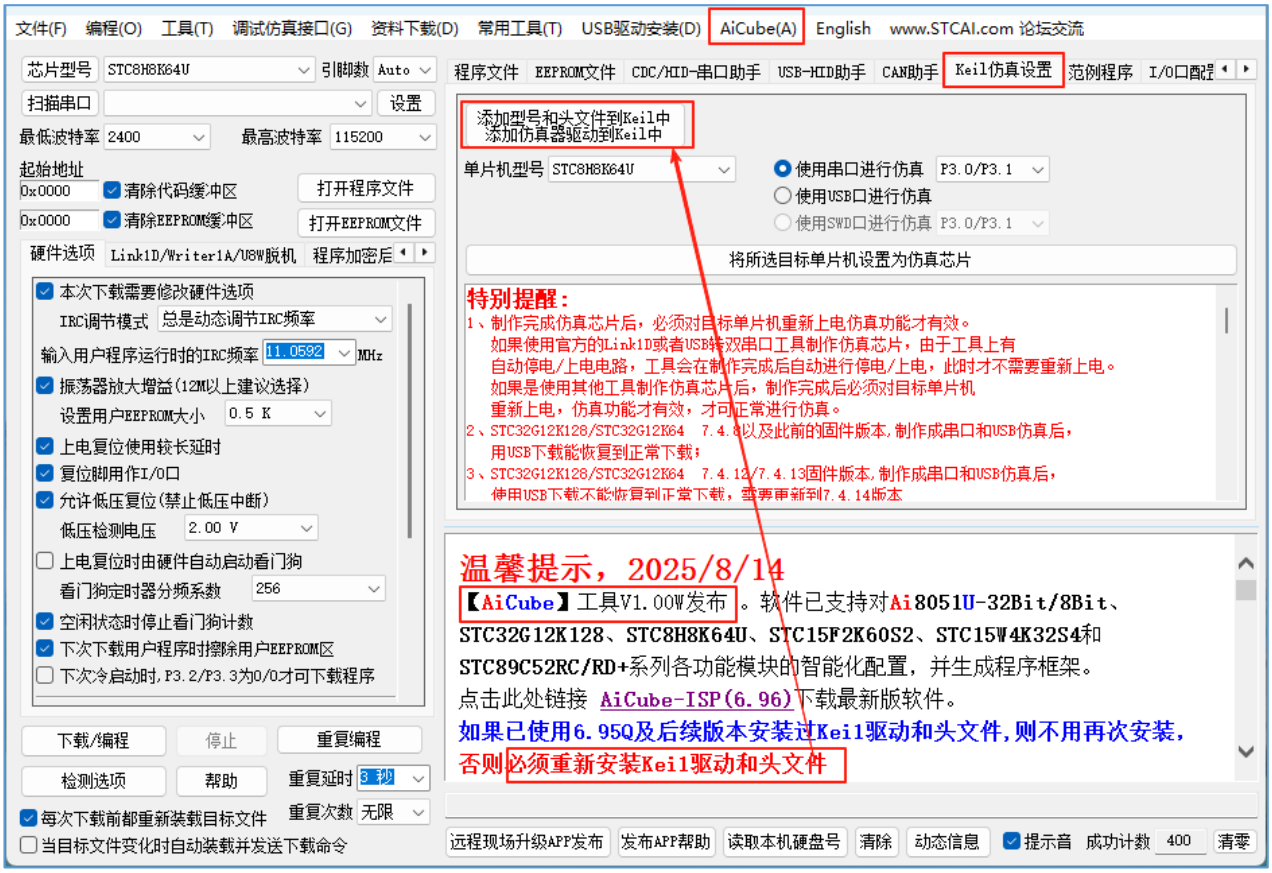
解压缩后的 AiCube-ISP 文件图标，如下图：



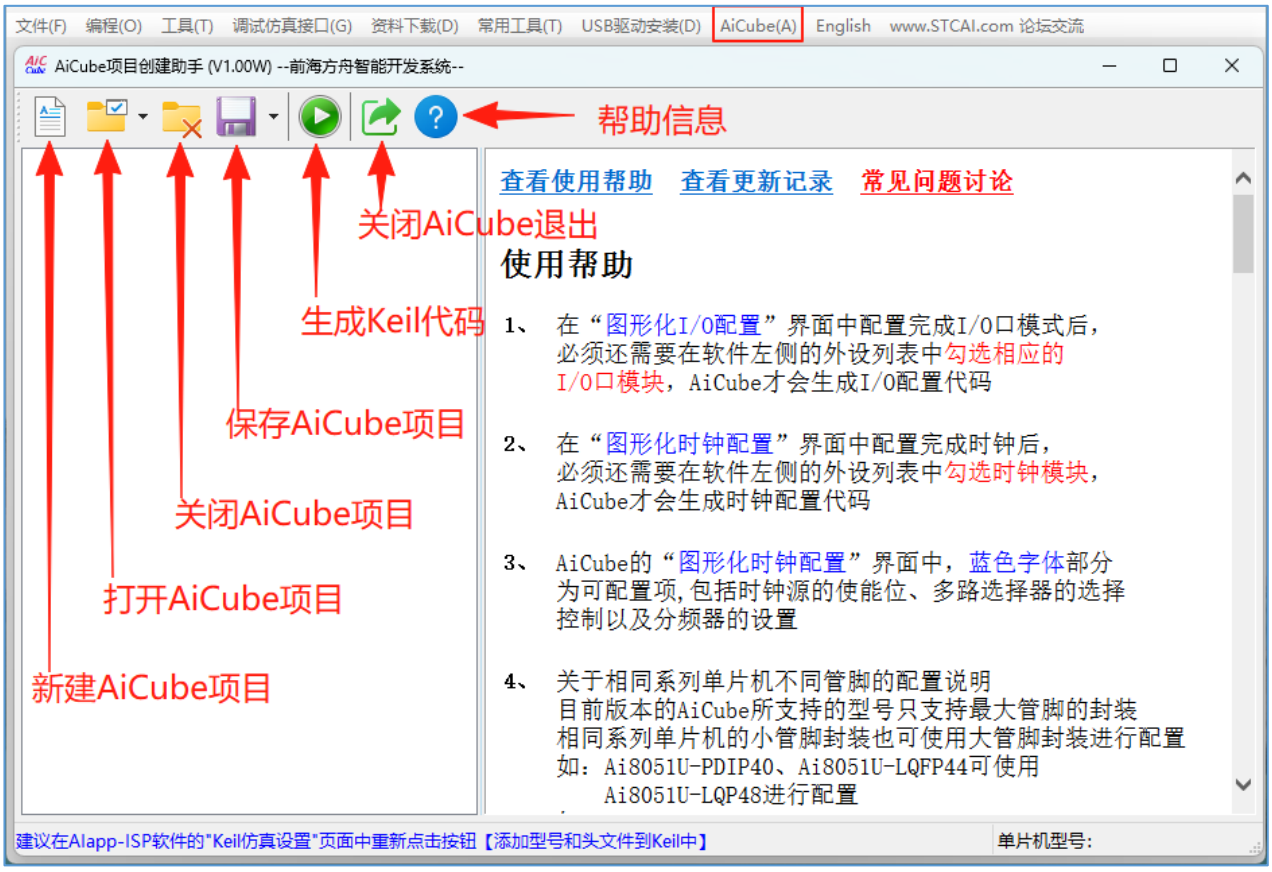
如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

2.8.8.2　使用【AiCube】助手创建项目

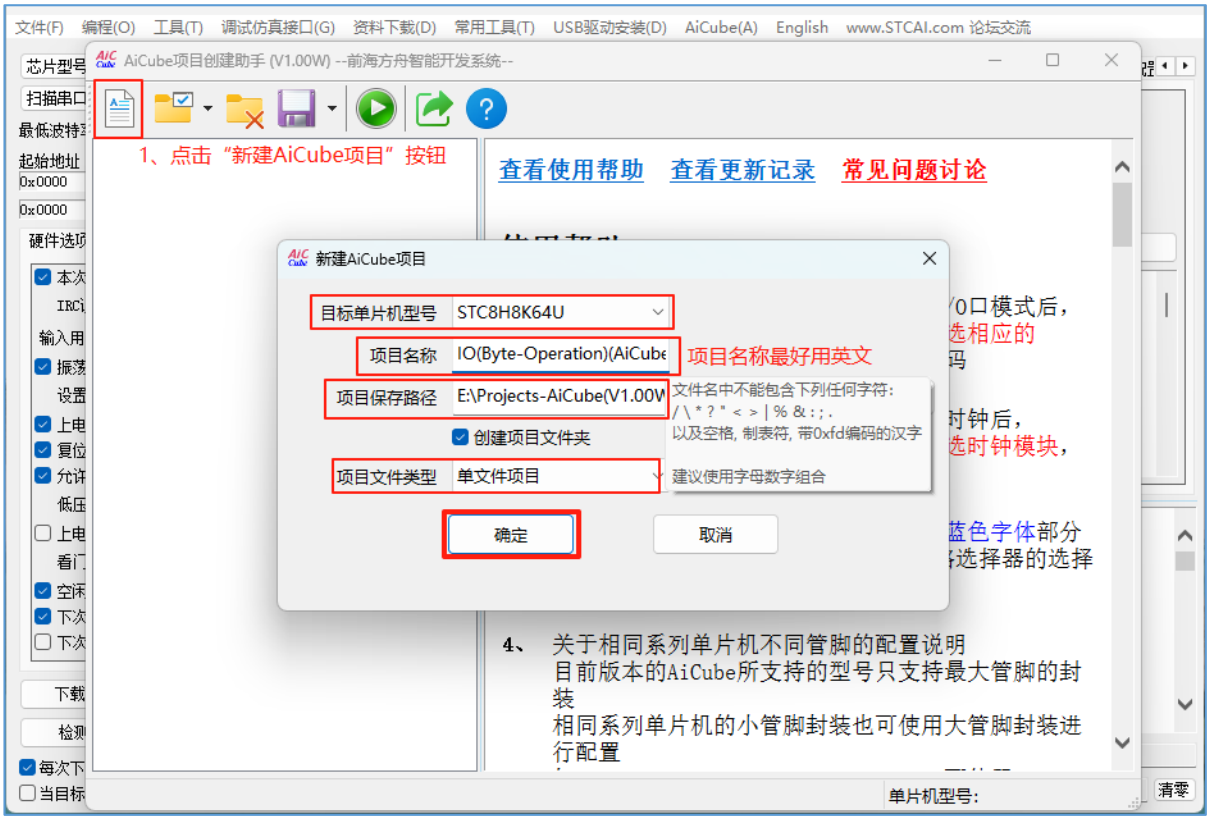
➲　双击图标，打开 AiCube-ISP 软件界面，如下图：



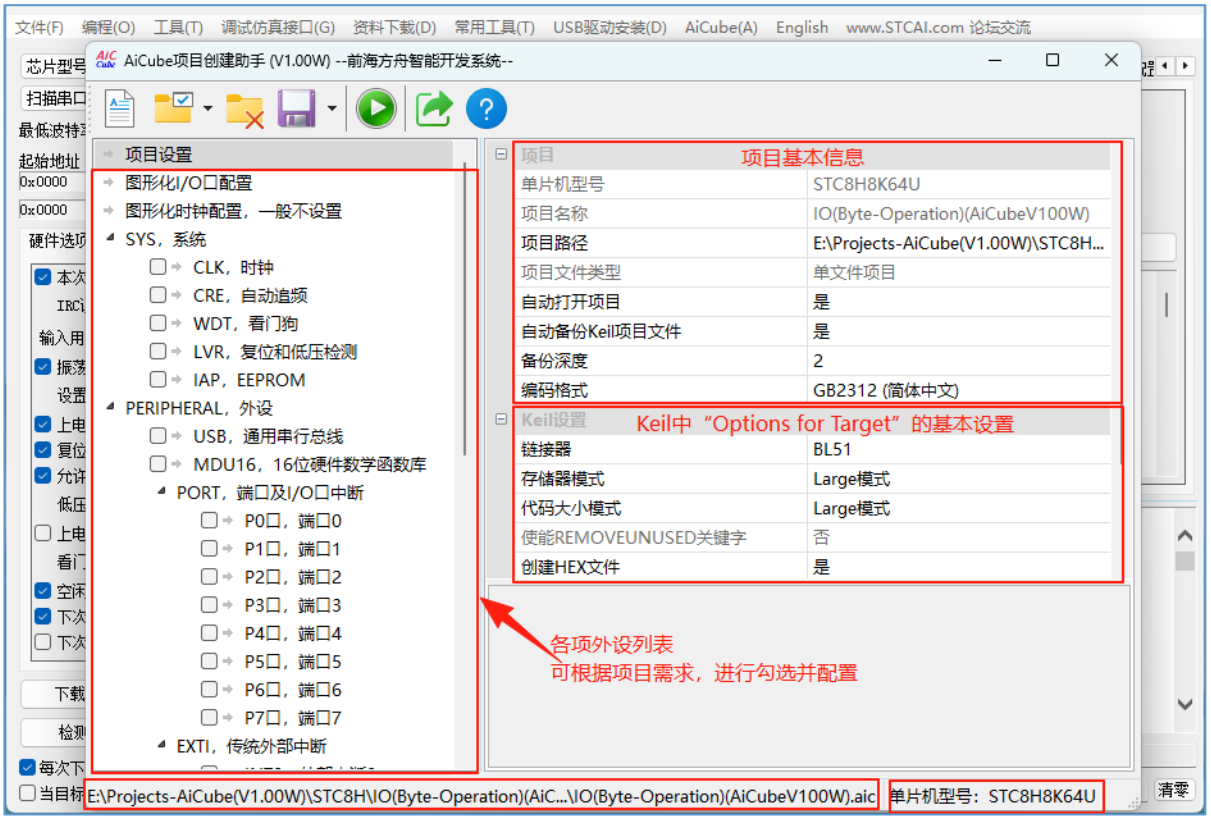
➲　点击【AiCube（A）】，弹出如下窗口：



➲ 点击“新建 AiCube 项目”按钮，选择并填写项目基本信息，建议项目名称用英文！



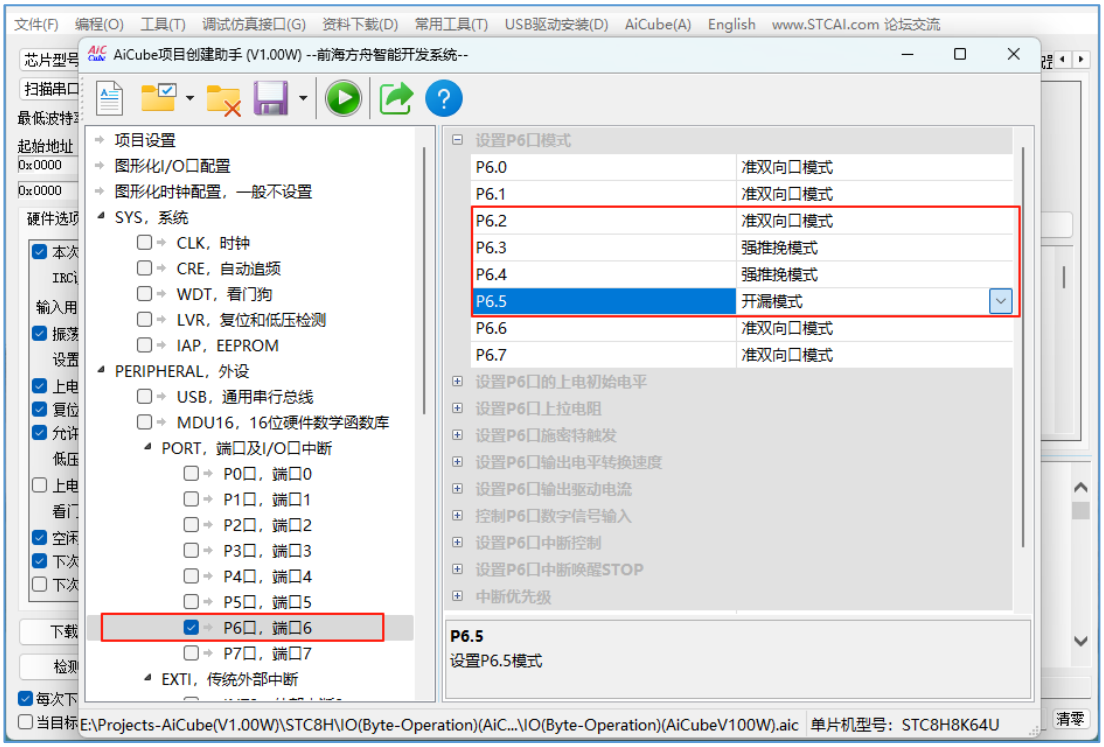
➲ 点击“确定”按钮，出现如下配置界面：



对于本项目，需要单片机读【P3.2 / P3.3 / P3.4 / P3.5】这 4 个 I/O 的状态，
输出到【P6.2 / P6.3 / P6.4 / P6.5】对应的指示灯

针对 I/O 有 4 种不同的模式，我们将【P3.2 / P3.3 / P3.4 / P3.5】这 4 个不同的 I/O 口设置为不同的模式，以验证不同情况下依然可以正常输入/输出。

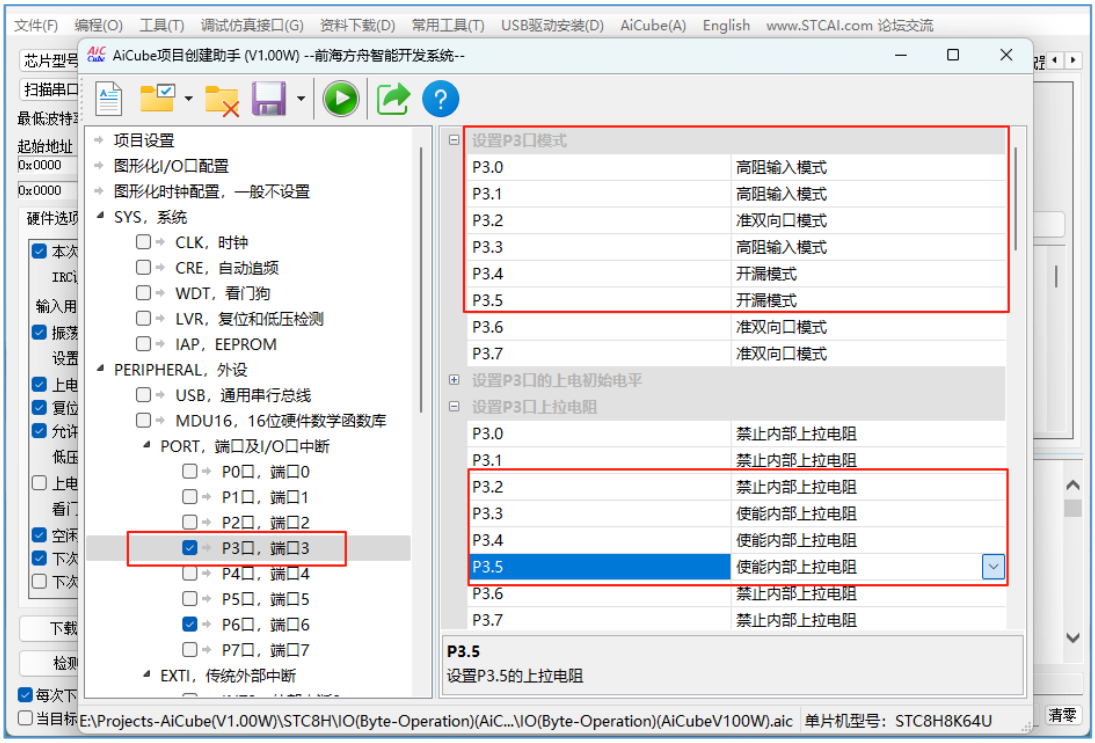
【AiCube】助手中，端口默认是准双向口模式，我们配置输出端口的 P62 保持准双向口模式，P63、P64 为强推挽输出，P65 为开漏输出（可以输出 0 点亮 LED）。如下图：



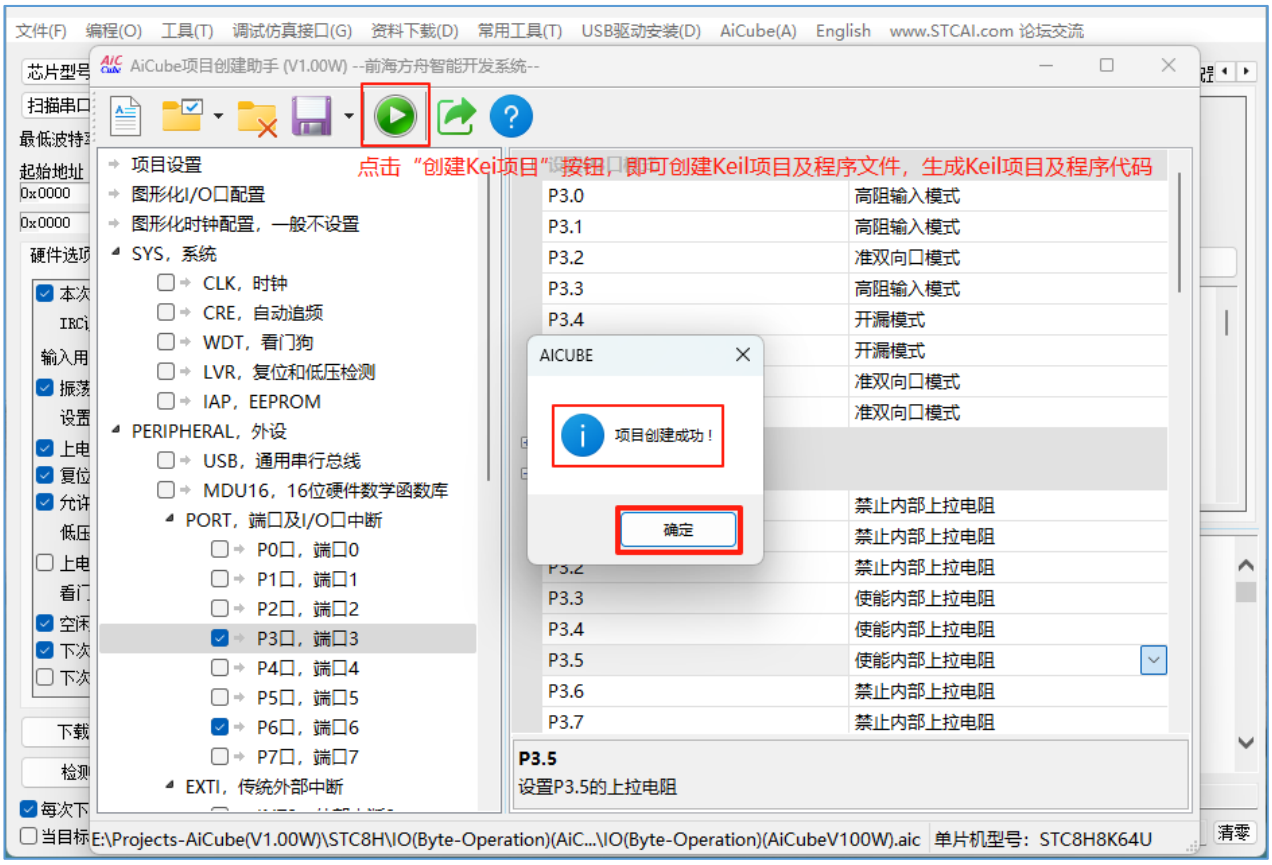
➲ 输入配置

- P30、P31 为高阻输入(因为跟 USB 复用，建议设置为高阻输入，不干扰 USB 的通讯控制)
- 设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（设置为 1 等效高阻输入）
- 然后开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）

【AiCube】助手中，端口默认是准双向口模式，内部上拉电阻默认是禁用设置。现在我们按上述要求配置 P30、P31 为高阻输入；P32 为保持准双向口；P33 为高阻输入；P34、P35 为开漏模式；打开 P33、P34、P35 的内部上拉电阻。如下图：



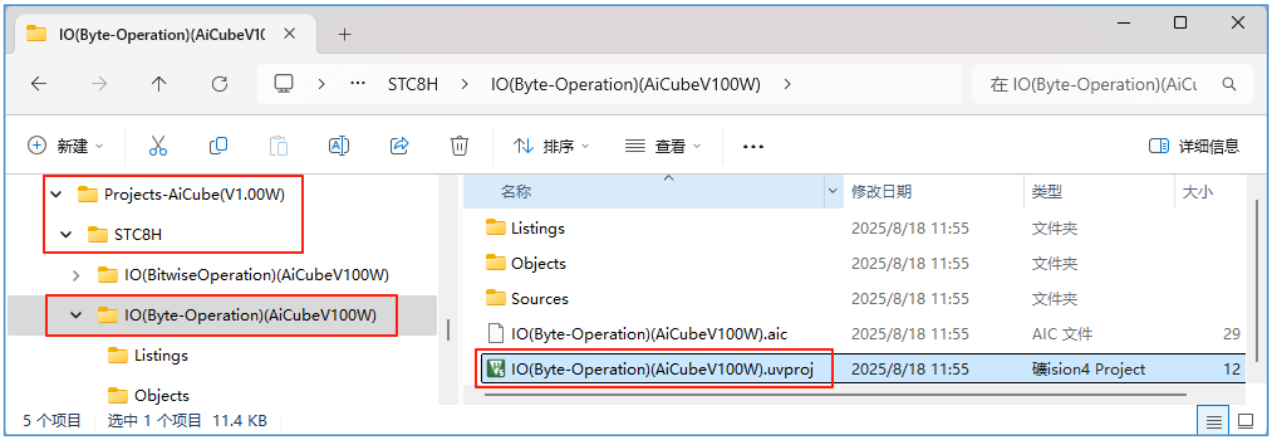
➲ 配置完毕，点击“创建 Keil 项目即程序文件”按钮，【AiCube】助手提示：“项目创建成功！”，如下图：



➲ 点击“确定”，立即自动打开 Keil 软件，进入该项目的 Keil 集成开发界面，即【AiCube】成功帮我们创建了项目及文件目录“IO(Byte-Operation)(AiCubeV100W)”。至此，可以关闭【AiCube】。

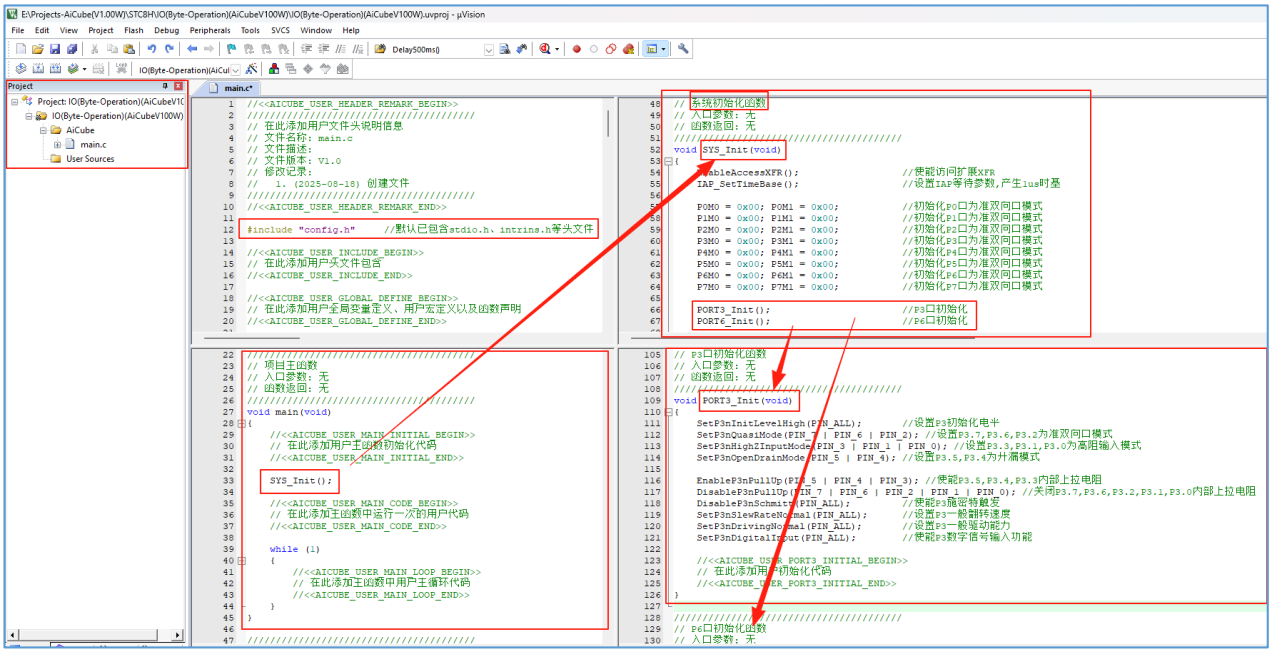
➲ 找到创建的项目目录

发现通过上述操作，【AiCube】助手已经帮我们建立了项目目录及 IO(Byte-Operation)(AiCubeV100W).uvproj 文件，如下图：



➲ 观察已打开的 Keil 项目文件：IO(Byte-Operation)(AiCubeV100W).uvproj

如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，全代码观察）：

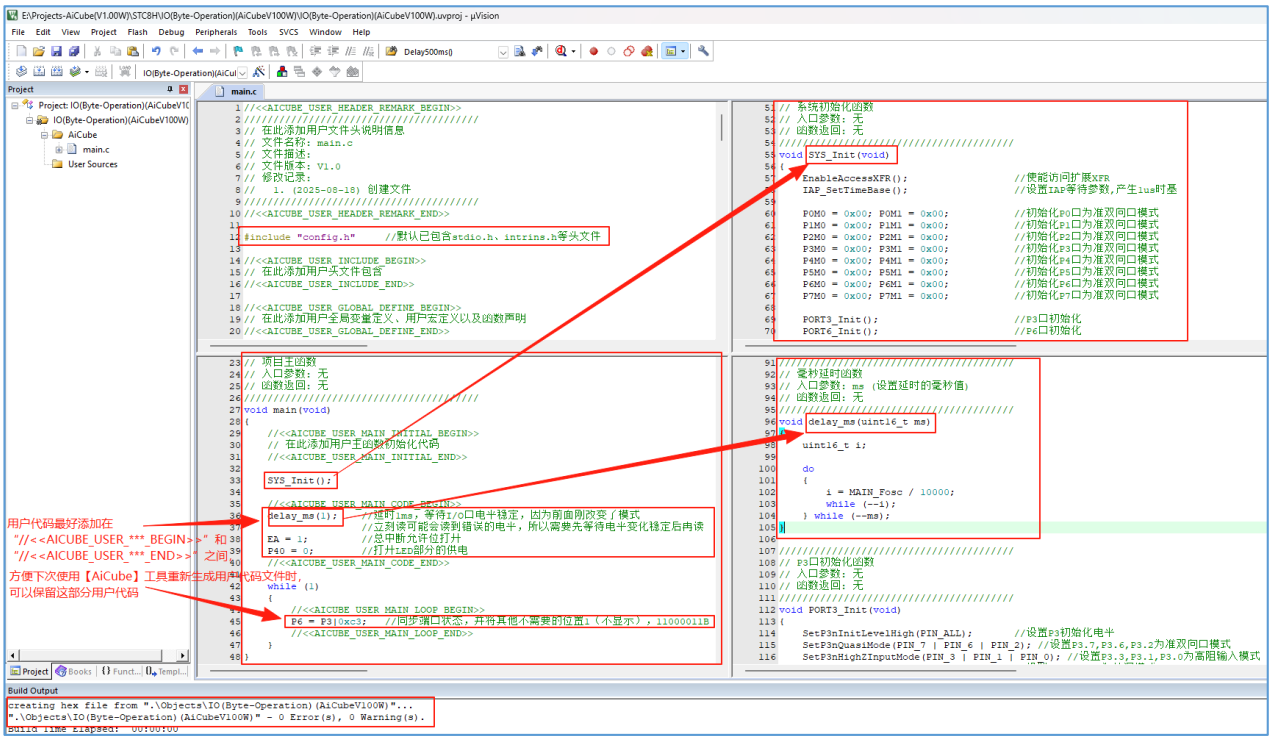


根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、微秒和毫秒延时函数、P0 口和 P3 口初始化函数。

2.8.8.3　　增加实验项目代码

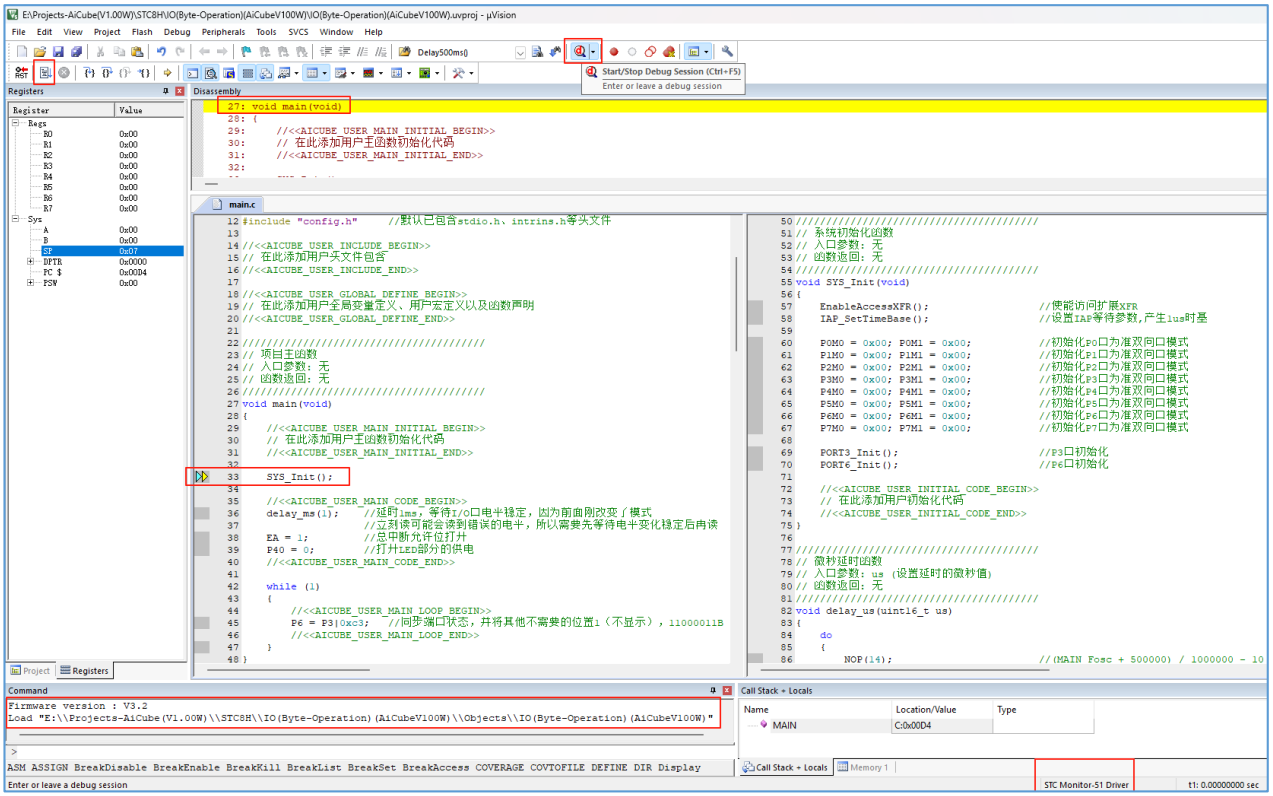
为实现“I/O 整组 8 位操作点灯，用 USB 仿真调试”的实验，我们现在需要在主函数中：

- 调用延时函数“delay_ms(1);”，延时 1ms（等待 I/O 口电平稳定）；
- 加入“P40 = 0;”，打开 LED 供电的代码；
- 在 while 循环中加入 I/O 口 8 位整组操作点灯代码“P0 = P3|0xc3;”。
- 保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，对比观察）：



2.8.8.4 使用 AiCube-ISP 软件设置仿真芯片，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果

- **制作仿真芯片：**连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，在 Keil 仿真设置选项卡中，按步骤操作“将所选目标单片机设置为仿真芯片”。
- **仿真设置：**回到 Keil 界面，在 Keil 中按步骤在“Options for Target”的 Debug 选项卡和 Utilities 选项卡中完成使用“STC Monitor-251 Driver”的 USB 仿真设置。
- 点击 File Toolbar 上的“Start/Stop Debug Session”按钮，进入仿真界面，如下图：



一切准备就绪。

- 点击 Debug Toolbar 上的“Run”按钮，
  - 按下 STC8H8K64U 实验箱上的 P3.2 按键，对应的 LED62 灯亮；松开 P3.2 按键，LED62 灯灭。
  - 按下 STC8H8K64U 实验箱上的 P3.3 按键，对应的 LED63 灯亮；松开 P3.3 按键，LED63 灯灭。
  - 按下 STC8H8K64U 实验箱上的 P3.4 按键，对应的 LED64 灯亮；松开 P3.4 按键，LED64 灯灭。
  - 按下 STC8H8K64U 实验箱上的 P3.5 按键，对应的 LED65 灯亮；松开 P3.5 按键，LED65 灯灭。

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.8.9 实验 3：用 USB-CDC 读取按键状态在串口助手显示并点亮 LED 灯

2.8.9.1 实验介绍

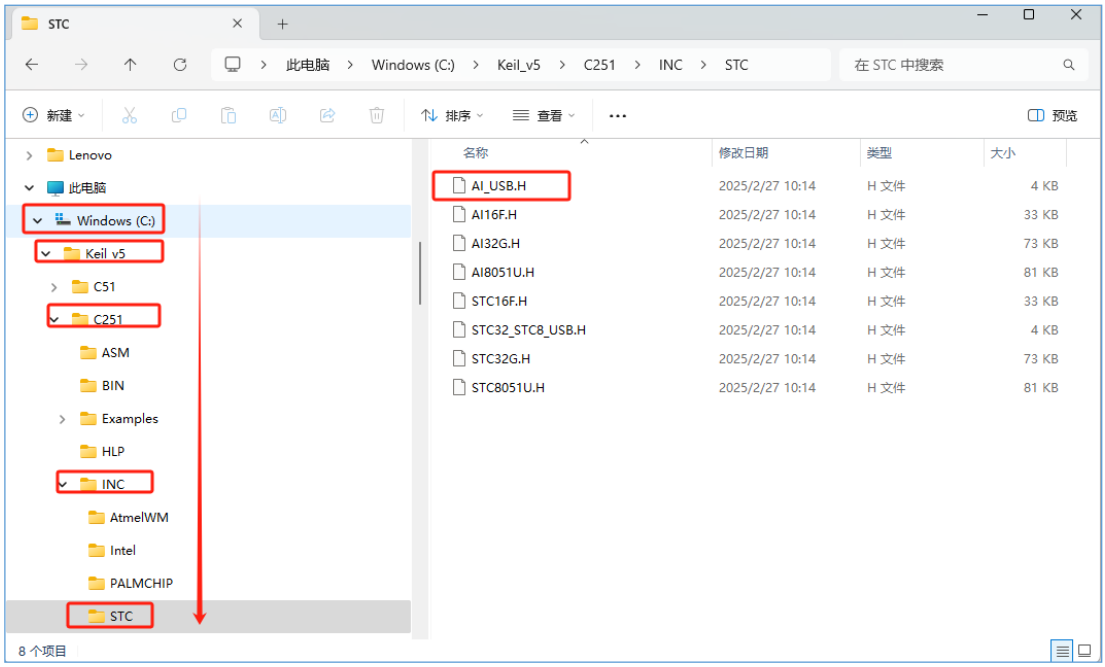
- 电脑 AiCube-ISP 软件中的，USB-CDC 串口助手通知单片机读【P3.2/P3.3/P3.4/P3.5】的状态，送回 USB-CDC 串口助手显示，同时将相应的状态送到 LED【P6.2/P6.3/P6.4/P6.5】显示（点亮/熄灭）
- 通过 USB-CDC 接口在串口助手接收窗口能直观的查看到对应端口的状态，并且提示点亮/熄灭了对应的 LED 灯。

2.8.9.2 准备工作--添加头文件和库函数

1、添加常用头文件 ai_usb.h

#include "ai_usb.h" //使用 usb 所需要包含的头文件。

这个头文件在使用 AiCube-ISP 软件【添加型号和头文件到 Keil 中】时就已经一并被添加到 Keil 中了。



2、添加 usb_cdc 库函数文件到项目所在的文件夹和项目文件中

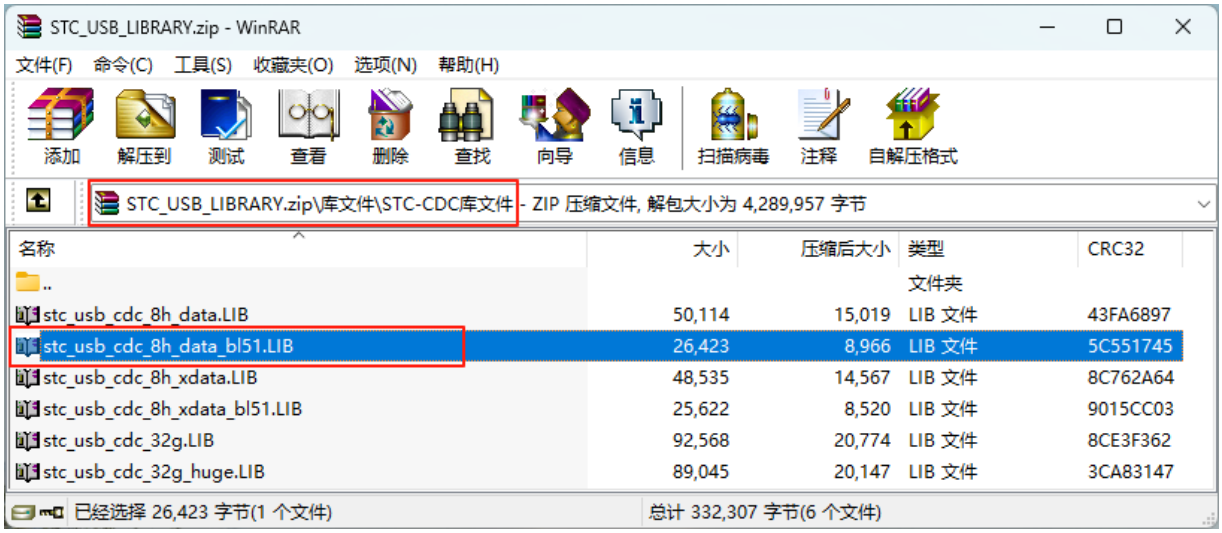
- 下载 **usb_cdc 库函数文件**

因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载

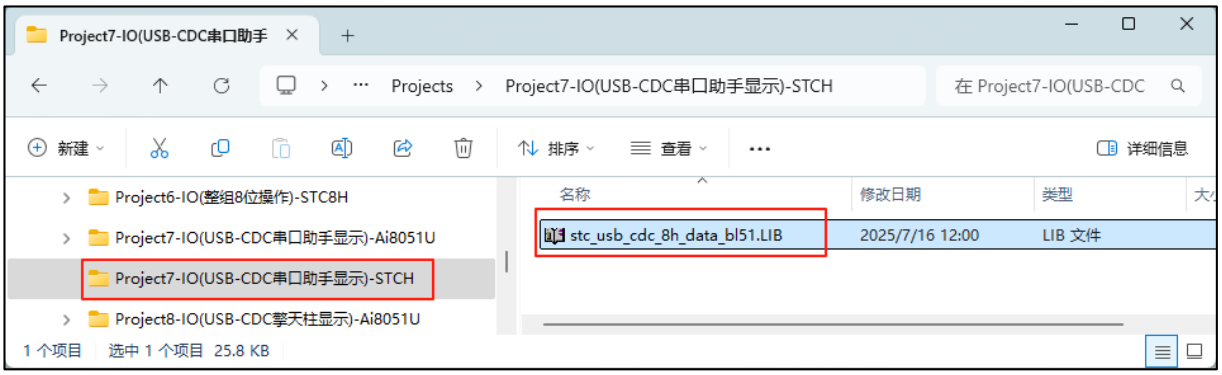


下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_8h_data_bl51.LIB

【8 位的 8051 的 USB 库 可以用 BL51 连接器
BL51： 不支持 REMOVEUNUSED；BL51 支持 扩展中断号工具；
如果 不用 REMOVEUNUSED，可以 使用 BL51 ！
LX51： 支持 REMOVEUNUSED；LX51 不支持 扩展中断号工具；
如果 不用 扩展中断号工具，可以 使用 LX51 ！
详情请参考本公司技术论坛链接:8 位的 8051 的 USB 库可以用 BL51;BL51 不支持 REMOVEUNUSED;
https://www.stcaimcu.com/thread-17105-1-1.html
(出处: 国芯人工智能技术交流网站)】

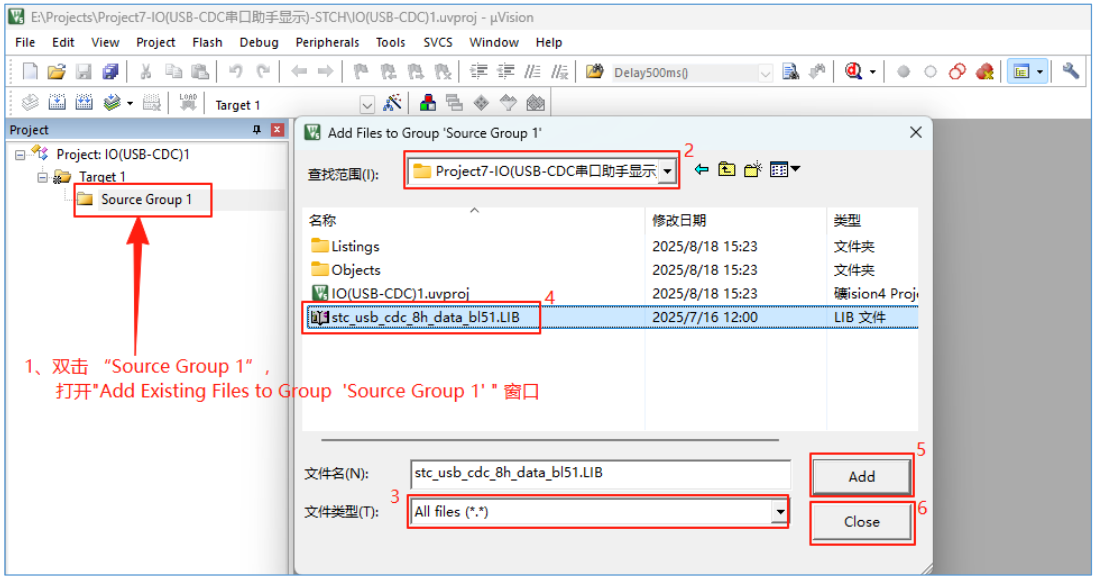


➲ **将库函数“stc_usb_cdc_8h_data_bl51.LIB”复制到创建项目所在的文件夹中**

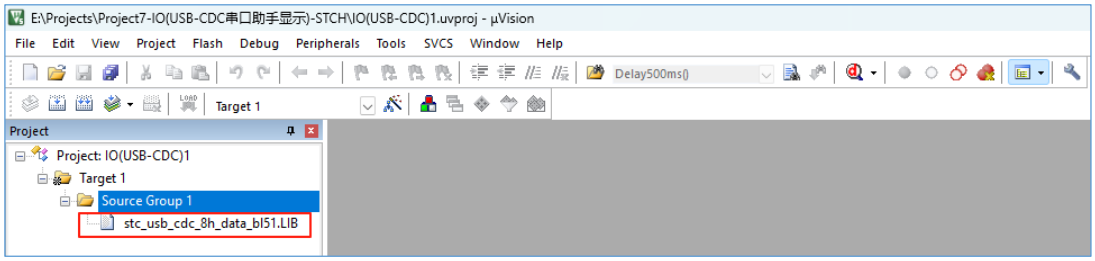


➲ **将库函数“stc_usb_cdc_8h_data_bl51.LIB”添加进入此项目重新文件中**

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_8h_data_bl51.LIB 添加进入此工程，就可以使用 USB-CDC 功能进行端口通讯。



Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

2.8.9.3　　实验程序 C 代码

```
#include <STC8H.H>     //包含 STC8H 的头文件
#include   "ai_usb.h"   //USB 库和 USB 虚拟设备的头文件
                        //由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
                        //只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
                        //即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。
#include "intrins.h"    //使用_nop_()函数所必须要包含的头文件，
                        //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                        //会导致编译器找不到这个而函数而报错。

void Delay1ms(void)     //延时 1ms@11.0592MHz
{
    unsigned char data i, j;

    i = 15;
    j = 90;
    do
    {
        while (--j);
    } while (--i);
```

```
}

void main(void)
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x30; P3M1 = 0x3b;
                    //P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出
    P3PU = 0x38;    //打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态
    P3 |= 0x3c;     //需要读的 I/O，对外输出锁存器先设置为 1，0011 1100B（0x3c）先和 P3 口逻辑或一下
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x38; P6M1 = 0x20;       //设置 P63、P64 为推挽输出，P65 为开漏输出，其余为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    Delay1ms();                     //延时 1ms，等待 I/O 口电平稳定，因为前面刚改变了模式
                                    //立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读

    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接 */

    EA = 1;                         //总中断允许位打开
    P40 = 0;                        //打开 LED 部分的供电

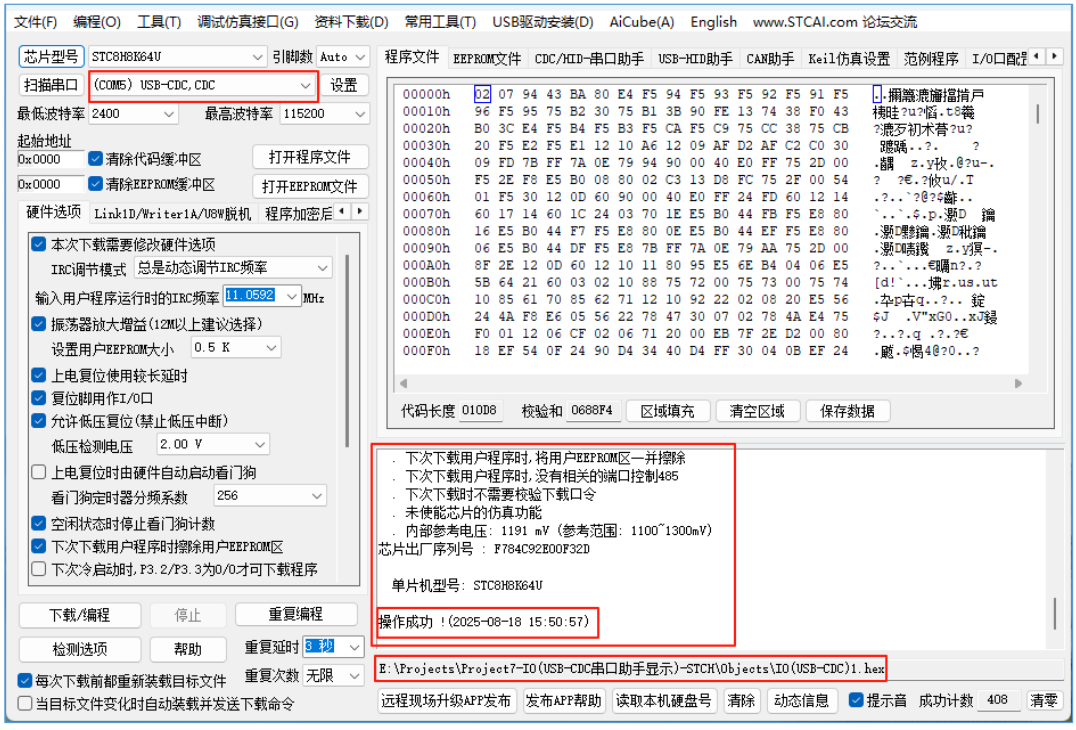
    while(1)
    {                               //使用 USB 串口控制来手动控制 LED 部分
        if(bUsbOutReady)
            //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
        {
            printf_usb("P3%d 当前的状态为:%d\r\n",(int)UsbOutBuffer[0],(int)(P3>>UsbOutBuffer[0])&1);
            //传入 hex 的位号来标识需要读取哪个位的状态，
            //读取后通过对整组 I/O 口数据的移位操作读取到对应 I/O 的状态
            //直接打印输出 “P3%d 当前的状态为: %d” 到主机相应软件的 USB【CDC/HID-串口助手】
//          P6 = P3|~(1<<UsbOutBuffer[0]);      //同步端口状态，并将其他不需要的位置 1（不显示）
//下面的 switch 等同于上一句，上一句 C 语言高手才能看懂
            switch (UsbOutBuffer[0])            //判断接收的第一个字节数据
            {
                case 2: P6 = P3 | 0xfb; break;  //如果接收到 2 则将 P3.2 状态送 P6.2，
                                                //其他不需要的位置 1（不显示），1111 1011B
                case 3: P6 = P3 | 0xf7; break;  //如果接收到 3 则将 P3.3 状态送 P6.3，
```

```
                                                    //其他不需要的位置 1（不显示），1111 0111B
                case 4: P6 = P3 | 0xef; break;      //如果接收到 4 则将 P3.4 状态送 P6.4，
                                                    //其他不需要的位置 1（不显示），1110 1111B
                case 5: P6 = P3 | 0xdf; break;      //如果接收到 5 则将 P3.5 状态送 P6.5，
                                                    //其他不需要的位置 1（不显示），1101 1111B
            }
            printf_usb("已将状态同步给 LED6%d\r\n",(int)UsbOutBuffer[0]);        //显示同步完成
            //直接打印输出 “已将状态同步给 LED0%d” 到主机相应软件的 USB【CDC/HID-串口助手】

            usb_OUT_done();                 // 本包接收的数据处理完毕，准备接收下一包数据
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
        }
    }
}
```

2.8.9.4　程序下载，观察实验现象

录入代码，保存，编译。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤“打开程序文件”IO(USB-CDC)1.hex（“打开程序文件”是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中）并“下载/编程”，如下图：



这时在【AiCube-ISP】界面，我们看到串口选项框中已经出现了一个(COM5)USB-CDC,CDC 串口，表明现在可以通过这个 USB-CDC 端口进行数据通讯了。如果电脑系统并没有产生 CDC 端口，可能是因为系统版本较低（低于 Windows10 版本）。

现在，打开【AiCube-ISP】界面的“CDC/HID-串口助手”选项卡，

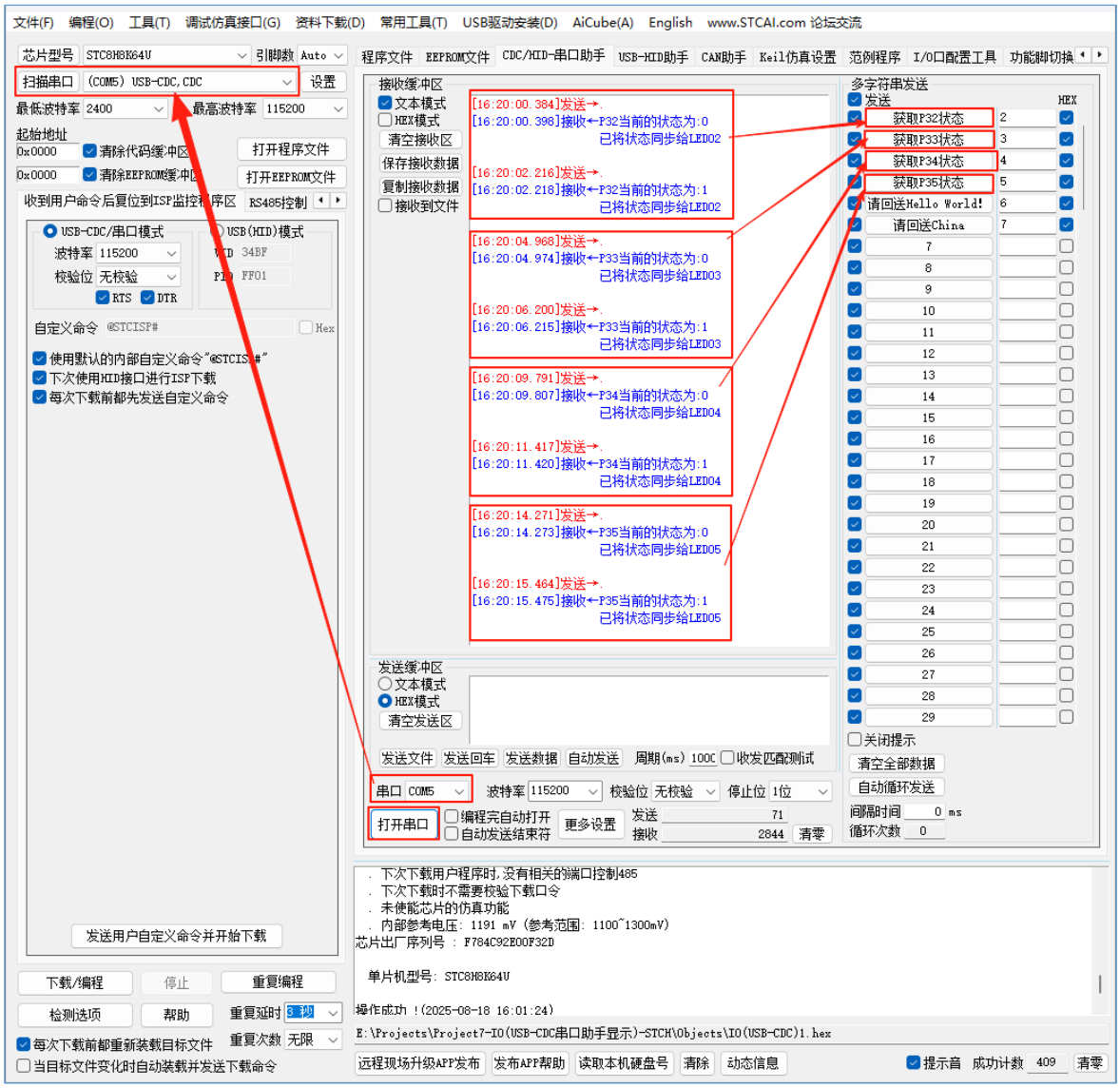
➲ 在右侧的“多字符串发送”窗口，通过设置对应的文字和需要发送的内容。

➲ 串口：选择与“扫描串口”窗口显示的 USB-CDC 端口一致

➲ 点击“打开端口”按钮

- ➢ 按住 P3.2 键，点击“获取 P32 的状态”按钮，MCU 读取实验箱上 P32 端口的状态并进行回传电脑的串口助手显示“P32 当前状态为 0”，同时实验箱上对应的 LED62 亮；
- ➢ 松开 P3.2 键，LED62 依然是亮的，再点击“获取 P32 的状态”按钮，MCU 读取实验箱上 P32 端口的状态并进行回传电脑的串口助手显示“P32 当前状态为 1”，同时实验箱上 LED62 灭；
- ➢ 按住 P3.3 键，点击“获取 P33 的状态”按钮，MCU 读取实验箱上 P33 端口的状态并进行回传电脑的串口助手显示“P33 当前状态为 0”，同时实验箱上对应的 LED63 亮；
- ➢ 松开 P3.3 键，LED63 依然是亮的，再点击“获取 P33 的状态”按钮，MCU 读取实验箱上 P33 端口的状态并进行回传电脑的串口助手显示“P33 当前状态为 1”，同时实验箱上 LED63 灭；
- ➢ 按住 P3.4 键，点击“获取 P34 的状态”按钮，MCU 读取实验箱上 P34 端口的状态并进行回传电脑的串口助手显示“P34 当前状态为 0”，同时实验箱上对应的 LED64 亮；
- ➢ 松开 P3.4 键，LED64 依然是亮的，再点击“获取 P34 的状态”按钮，MCU 读取实验箱上 P34 端口的状态并进行回传电脑的串口助手显示“P34 当前状态为 1”，同时实验箱上 LED64 灭；
- ➢ 按住 P3.5 键，点击“获取 P35 的状态”按钮，MCU 读取实验箱上 P35 端口的状态并进行回传电脑的串口助手显示“P35 当前状态为 0”，同时实验箱上对应的 LED65 亮；
- ➢ 松开 P3.5 键，LED65 依然是亮的，再点击“获取 P35 的状态”按钮，MCU 读取实验箱上 P35 端口的状态并进行回传电脑的串口助手显示“P35 当前状态为 1”，同时实验箱上 LED65 灭。

如下图：



2.8.10 实验 3A--使用【AiCube】项目助手实现“用 USB-CDC 读取按键状态在串口助手显示并点亮 LED 灯”

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit、Ai8051U-8Bit、STC32G12K128、STC8H8K64U、STC15F2K60S2、STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

2.8.10.1 安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj



下载下来的是压缩包文件图标，如下图：



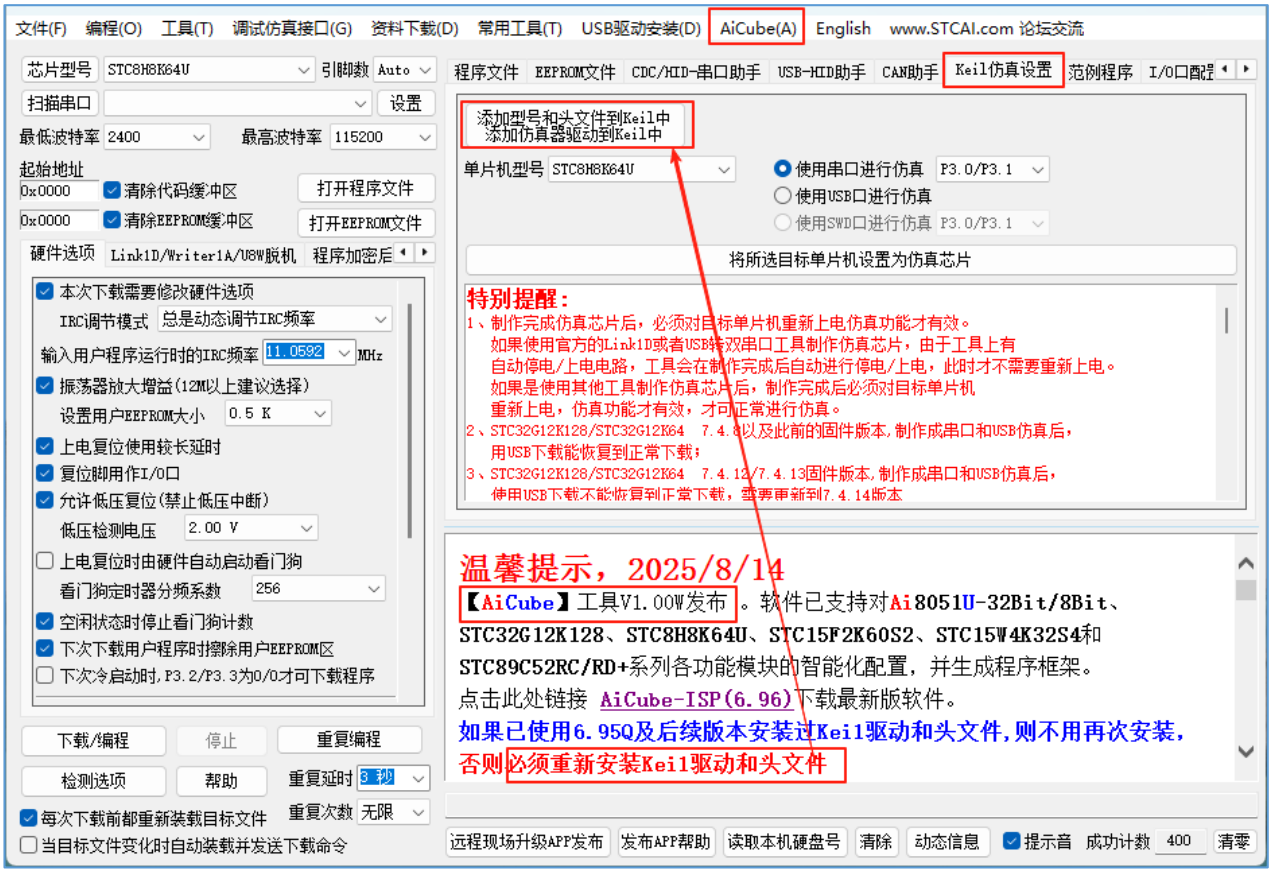
解压缩后的 AiCube-ISP 文件图标，如下图：



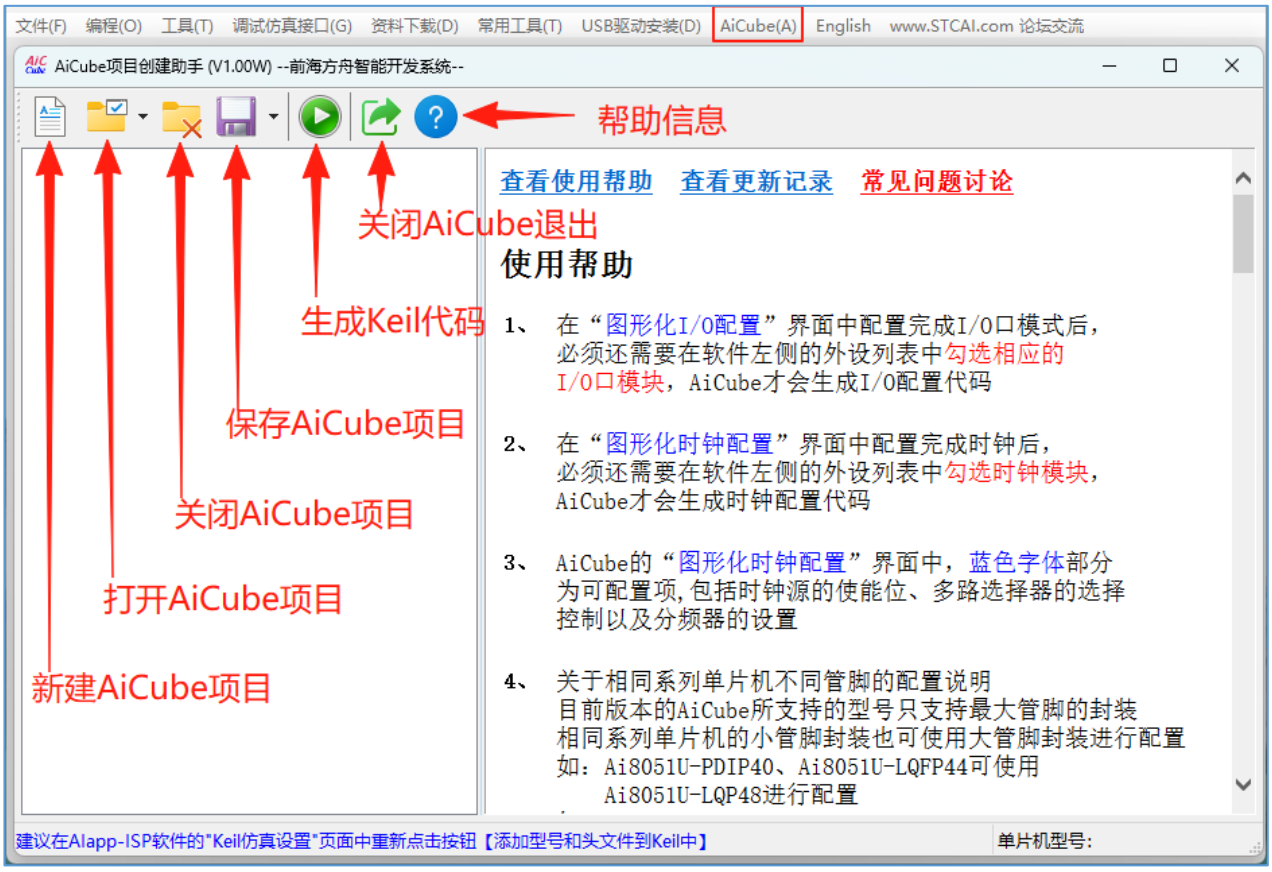
如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

2.8.10.2　使用【AiCube】助手创建项目

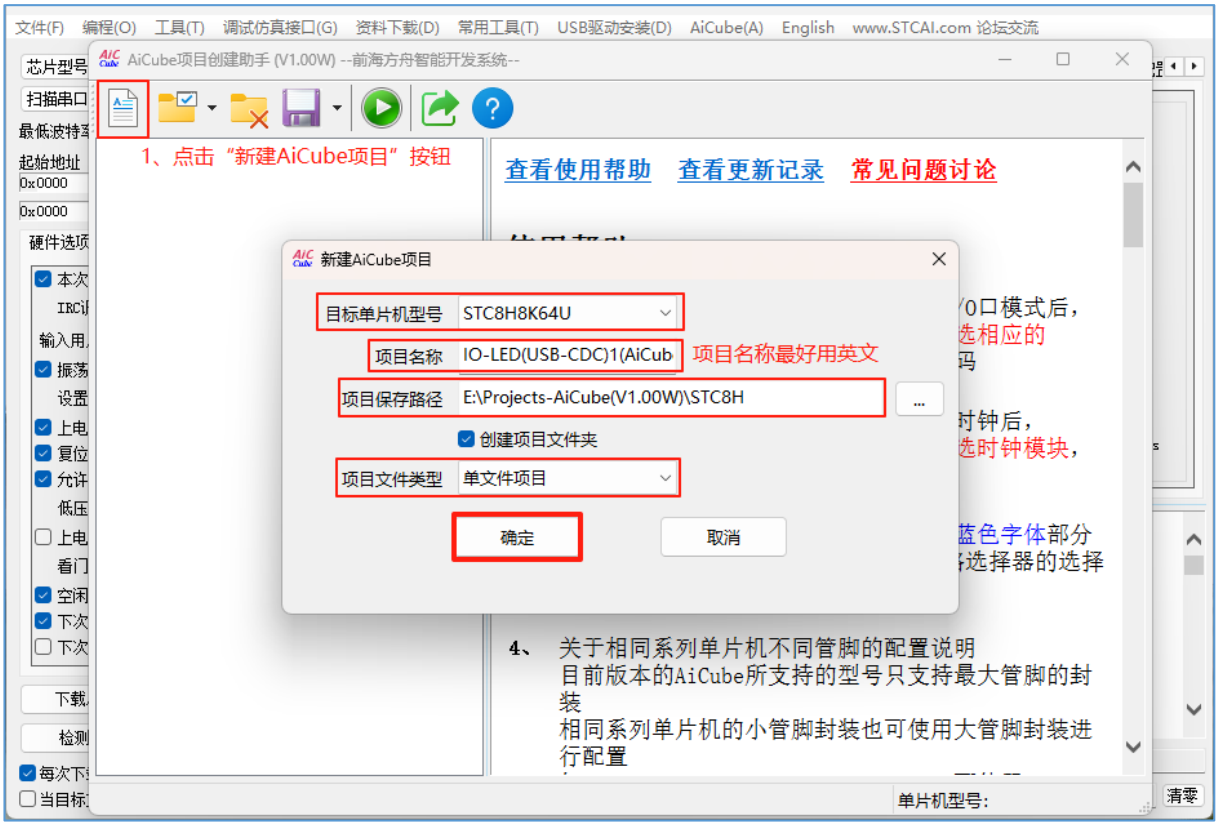
➲　双击图标，打开 AiCube-ISP 软件界面，如下图：



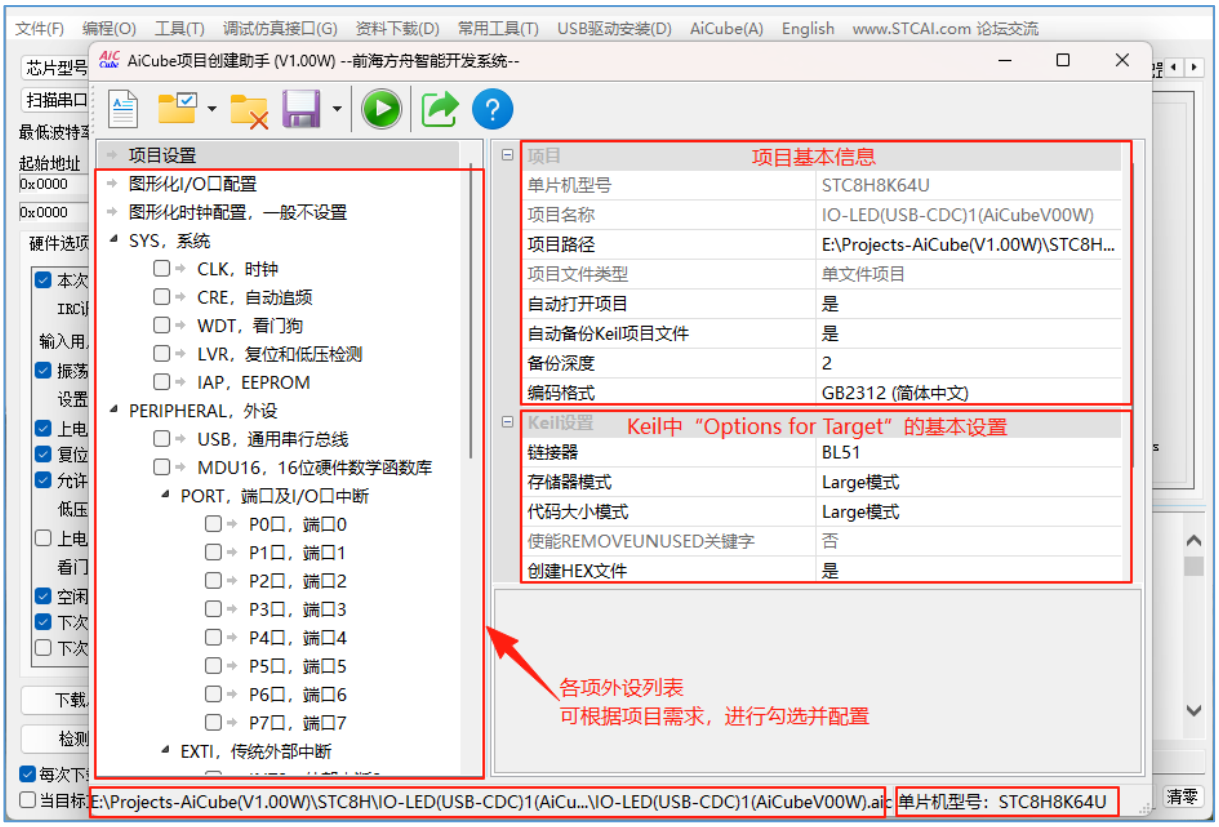
➲　点击【AiCube（A）】，弹出如下窗口：



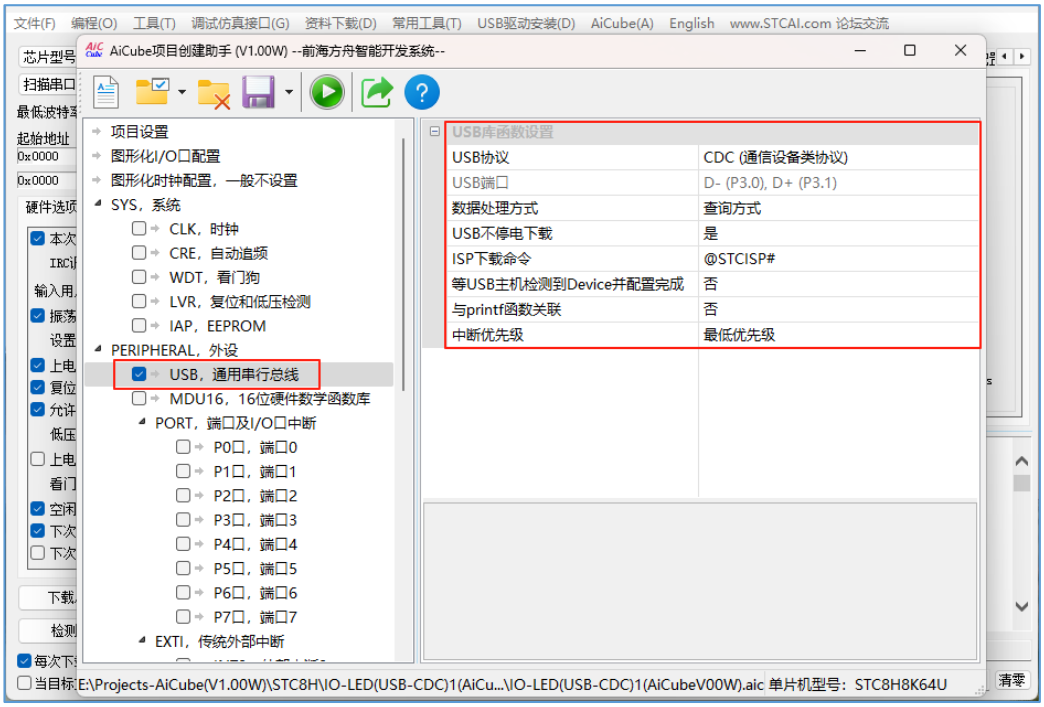
➲ 点击“新建 AiCube 项目”按钮，选择并填写项目基本信息，建议项目名称用英文！



➲ 点击“确定”按钮，出现如下配置界面：



➲ 本实验是通过【AiCube-ISP】系统软件中的 USB-CDC 串口助手通知单片机读【P3.2/P3.3/P3.4/P3.5】的状态，送回 USB-CDC 串口助手显示，同时将相应的状态送到 LED【P6.2/P6.3/P6.4/P6.5】显示（点亮/熄灭）。显然要勾选 USB 选项，默认“CDC（通信设备类协议）”，如下图：

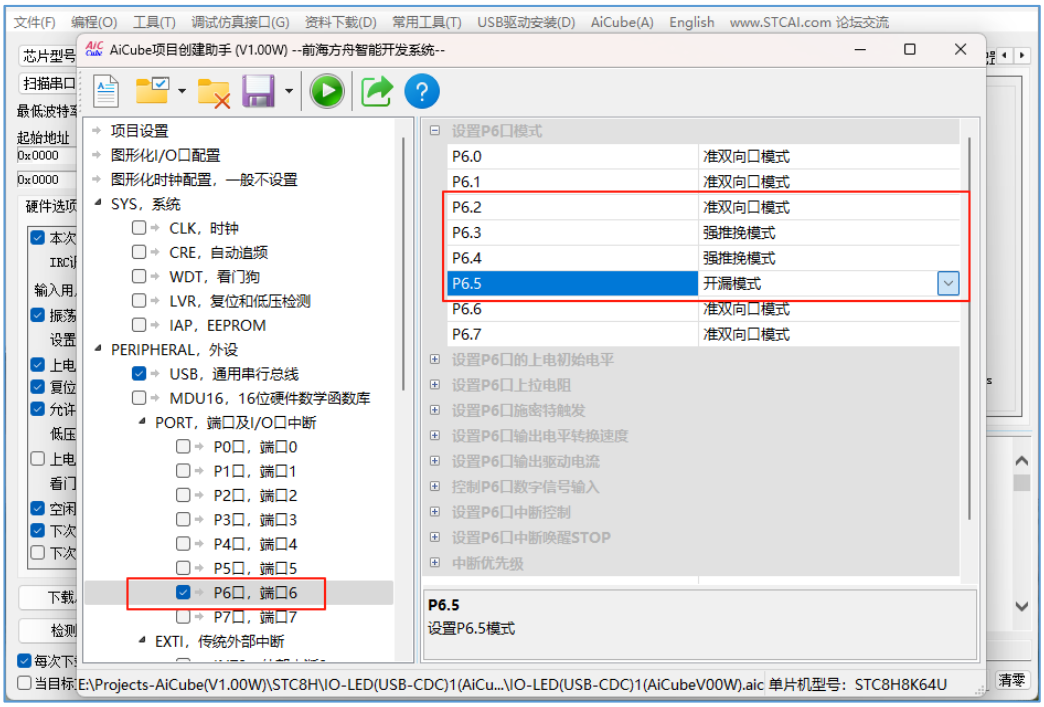


对于本项目，需要单片机读【P3.2 / P3.3 / P3.4 / P3.5】这 4 个 I/O 的状态，
输出到【P6.2 / P6.3 / P6.4 / P6.5】对应的指示灯

➲ I/O 配置

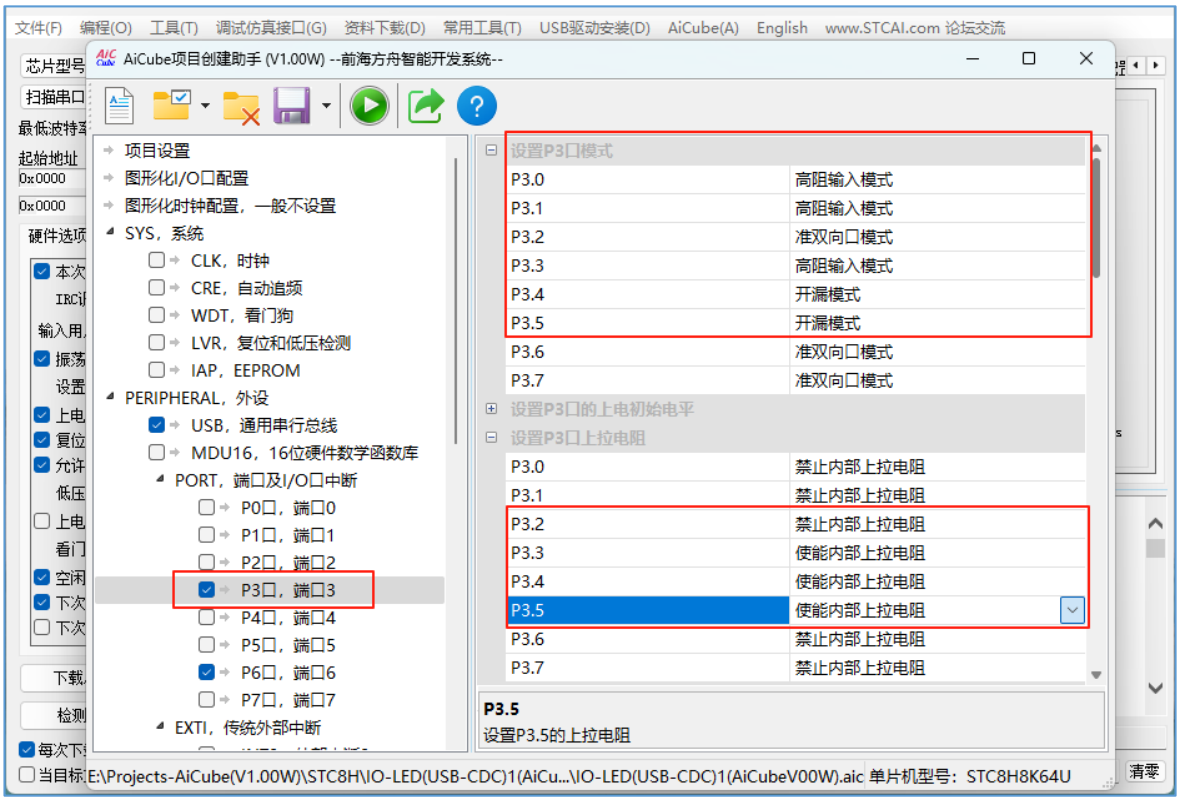
针对 I/O 有 4 种不同的模式，我们将【P3.2 / P3.3 / P3.4 / P3.5】不同的 I/O 口设置为不同的模式，验证不同情况下依然可以正常输入/输出。

- 【AiCube】助手中，端口默认是准双向口模式，我们配置**输出端口**的 P62 保持准双向口模式，P63、P64 为强推挽输出，P65 为开漏输出（可以输出 0 点亮 LED）。如下图：

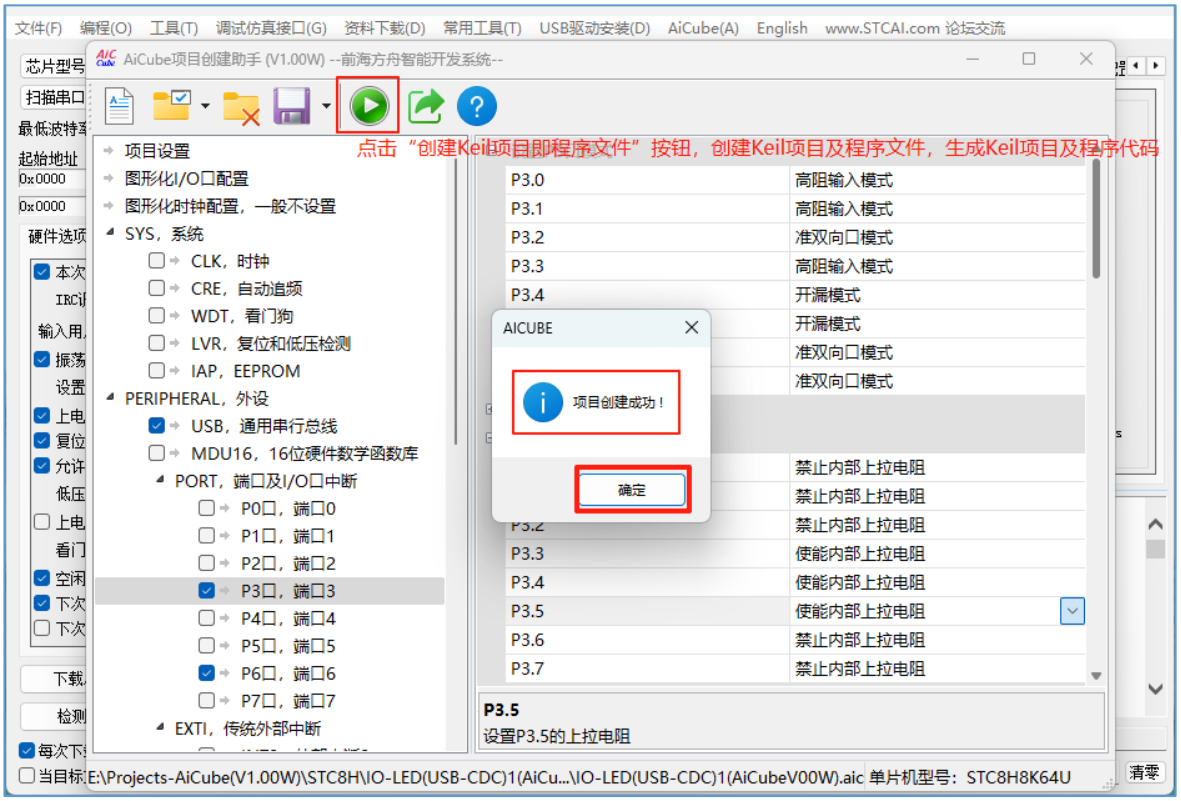


- **输入配置**
  - ✓ P30、P31 为高阻输入(因为跟 USB 复用，建议设置为高阻输入，不干扰 USB 的通讯控制)
  - ✓ 设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（设置为 1 等效高阻输入）
  - ✓ 然后开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）

【AiCube】助手中，端口默认是准双向口模式，内部上拉电阻默认是禁用设置。现在我们按上述要求配置 P30、P31 为高阻输入；P32 为保持准双向口；P33 为高阻输入；P34、P35 为开漏模式；打开 P33、P34、P35 的内部上拉电阻。如下图：



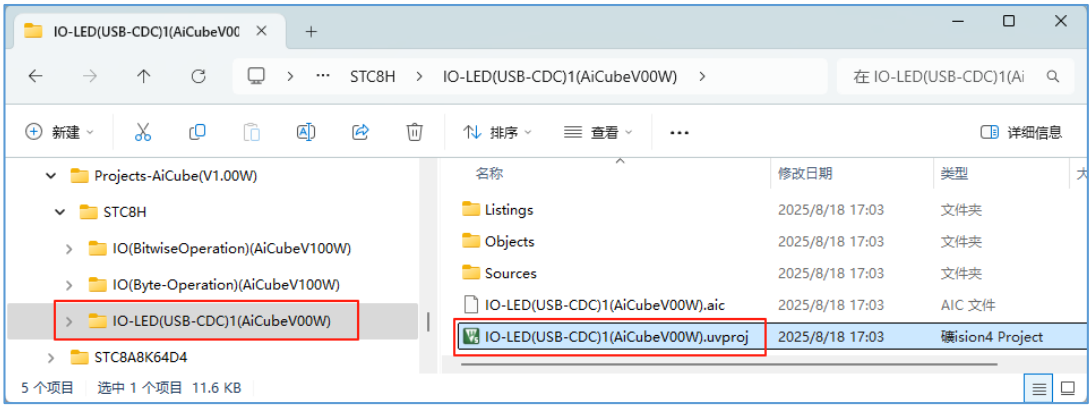
➲ 点击“创建 Keil 项目及程序文件”按钮，【AiCube】助手提示：“项目创建成功！”，如下图：



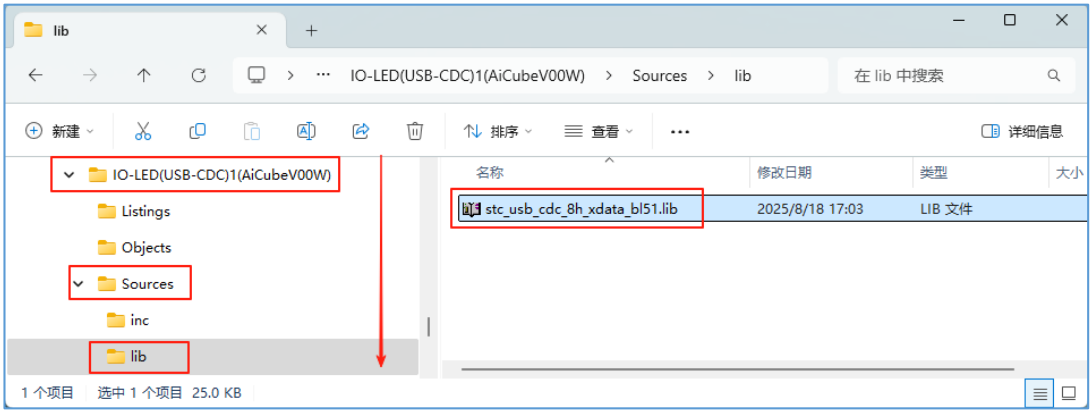
➲ 点击“确定”，立即自动打开 Keil 软件，进入该项目的 Keil 集成开发界面，即【AiCube】成功帮我们创建了项目及文件目录“IO-LED(USB-CDC)1(AiCubeV100W)”。至此，可以关闭【AiCube】。

➲ 找到创建的项目目录

发现通过上述操作，【AiCube】助手已经帮我们建立了项目目录及 IO-LED(USB-CDC)1(AiCube V100W).uvproj 文件，如下图：

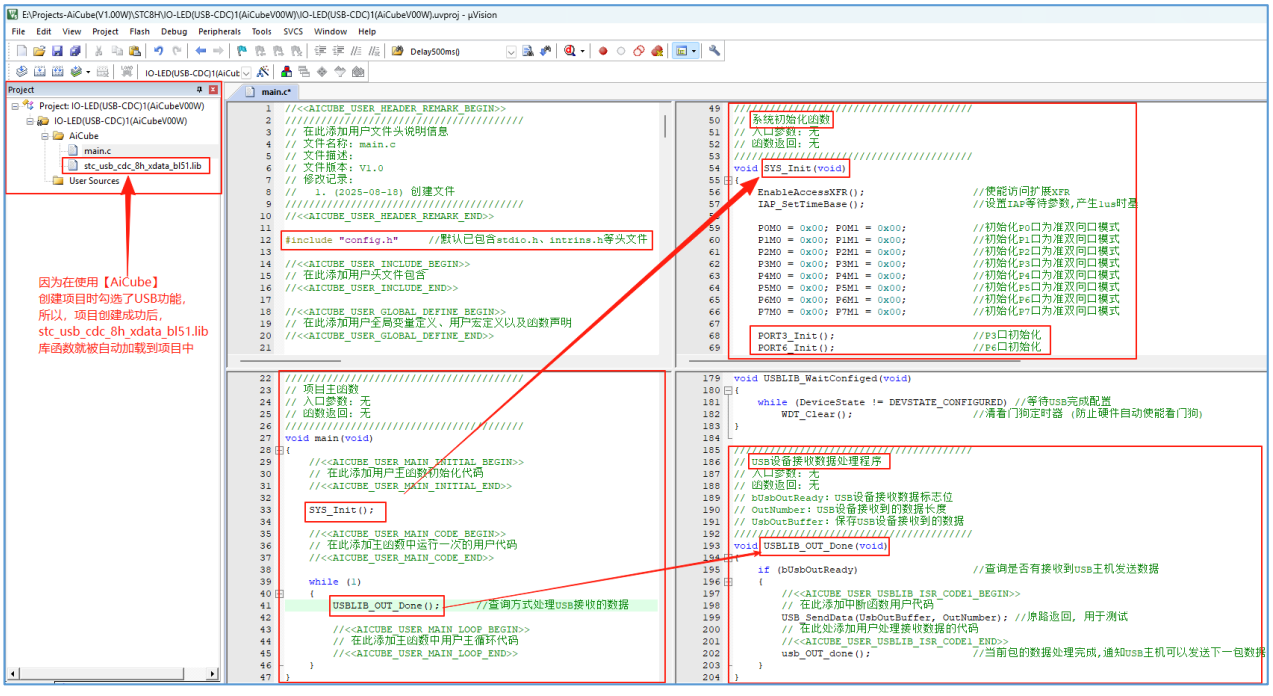


因为勾选了 USB 选项，打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经加载了：



➲ 观察已打开的 Keil 项目文件：IO-LED(USB-CDC)1(AiCubeV100W).uvproj

如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，全代码观察）：

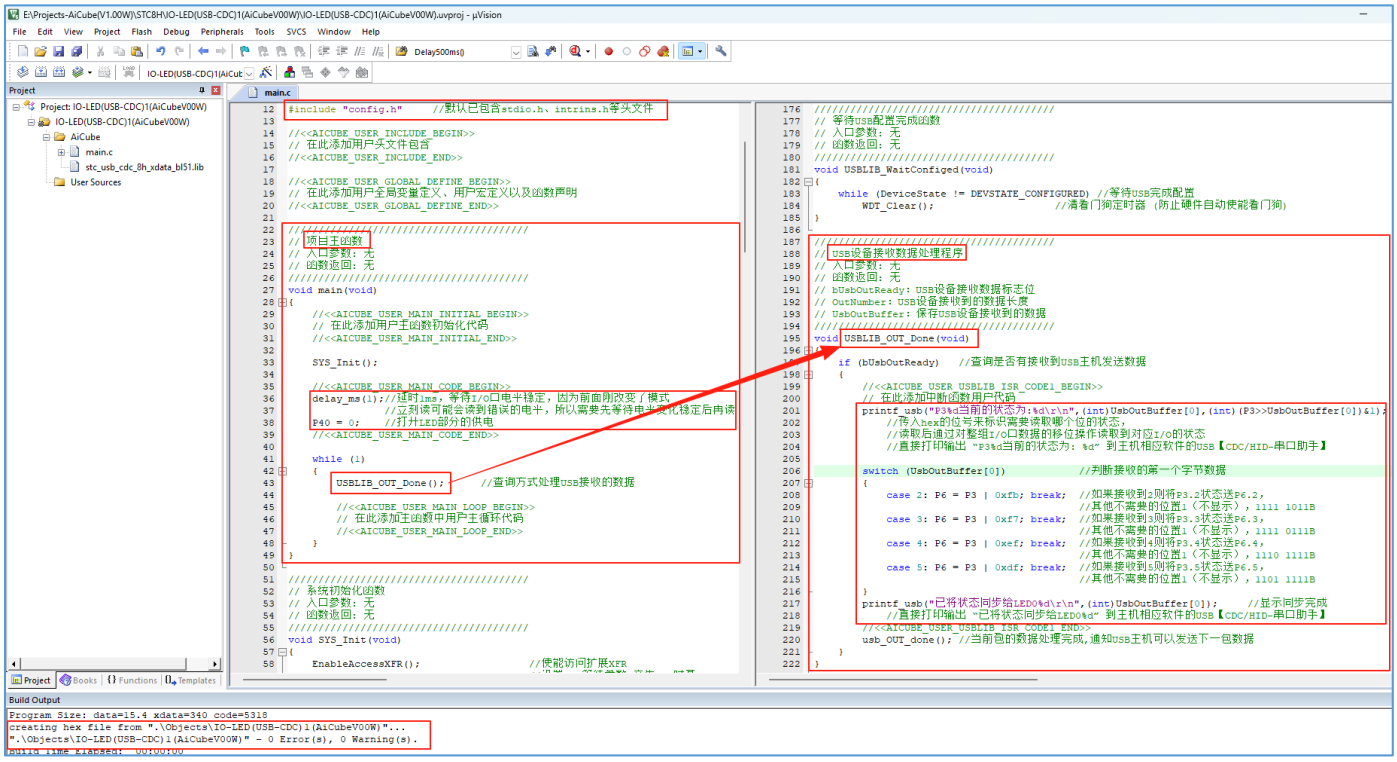


根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c，并自动加载 USB 库函数（stc_usb_cdc_8h_xdata_bl51.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、毫秒和微秒延时函数、P0 口和 P3 口初始化函数、USB 库初始化函数、等待 USB 配置完成函数、USB 设备接收数据处理程序。

2.8.10.3　增加实验项目代码

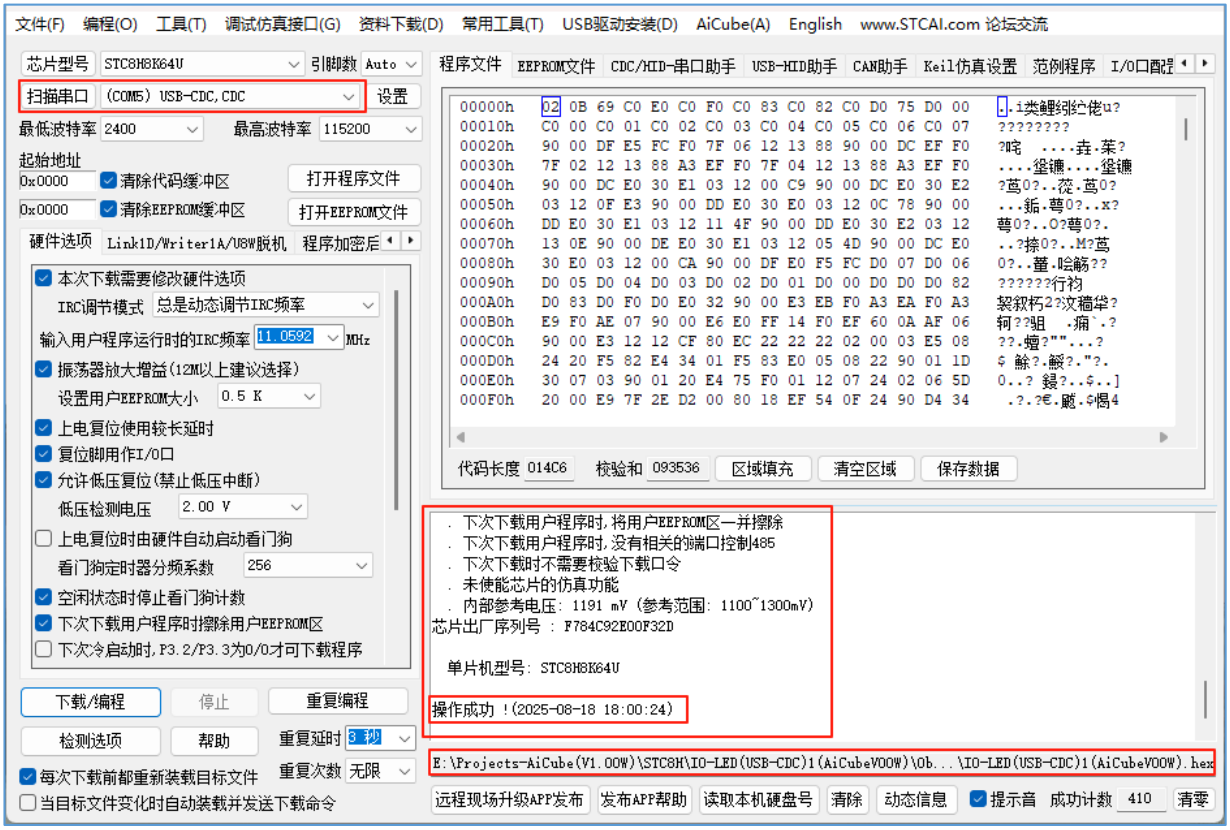
为实现通过 AiCube-ISP 系统软件中的 USB-CDC 串口助手读取【P3.2/P3.3/P3.4/P3.5】的状态，送回 USB-CDC 串口助手显示，同时将相应的状态送到 LED 的【P6.2/P6.3/P6.4/P6.5】显示（点亮/熄灭）。我们现在需要：

- 在主函数中，调用延时函数“delay_ms(1);”，延时 1ms（等待 I/O 口电平稳定）；
- 在主函数中，加入“P40 = 0;”，打开 LED 供电的代码；
- 在“USB 设备接收数据处理程序”子函数中，补充 USB 控制代码；
- 保存，编译，如下图：



2.8.10.4　使用 AiCube-ISP 软件系统，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果

连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“IO-LED(USB-CDC)1(AiCubeV100W).hex”到实验箱，如下图：

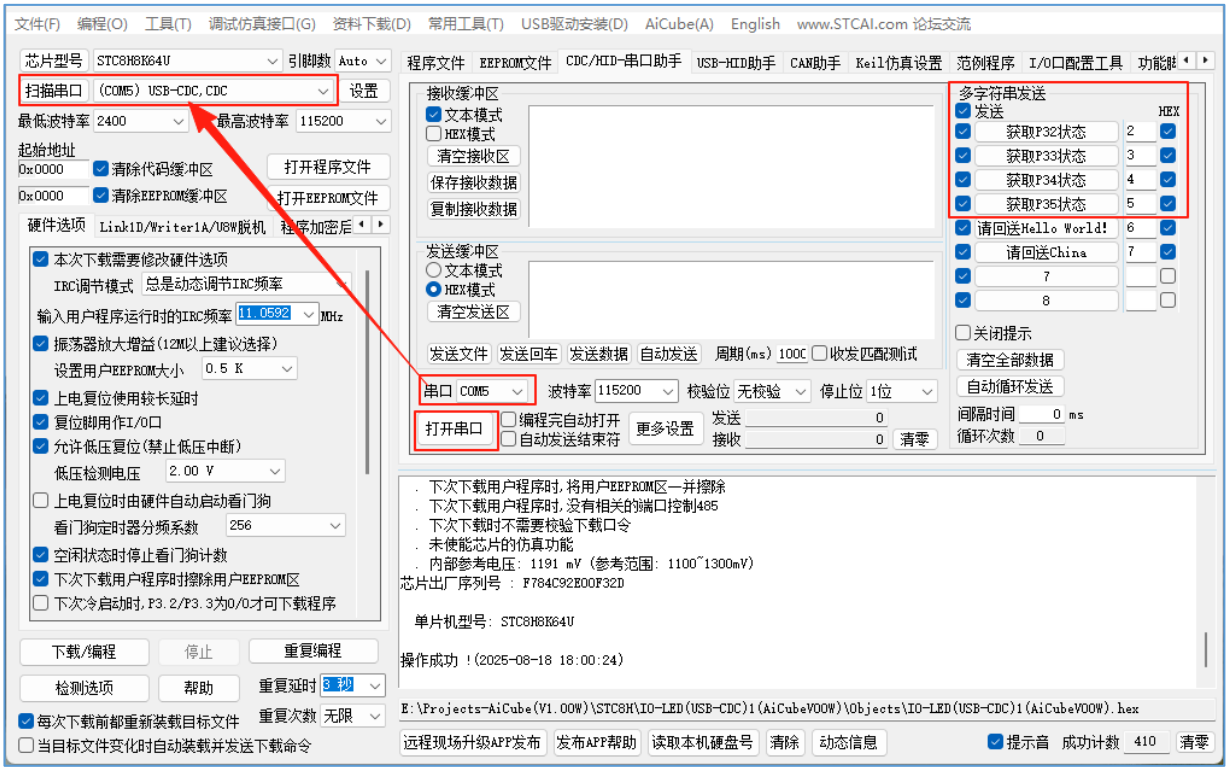


这时在【AiCube-ISP】界面，我们看到串口选项框中已经出现了一个(COM5)USB-CDC,CDC 串口，表明现在可以通过这个 USB-CDC 端口进行数据通讯了。

打开“CDC/HID-串口助手”选项卡，

➲ 在右侧的“多字符串发送”窗口，设置好发送控制命令，以便观察实验效果。

➲ 串口：选择与“扫描串口”窗口显示的 USB-CDC 端口一致

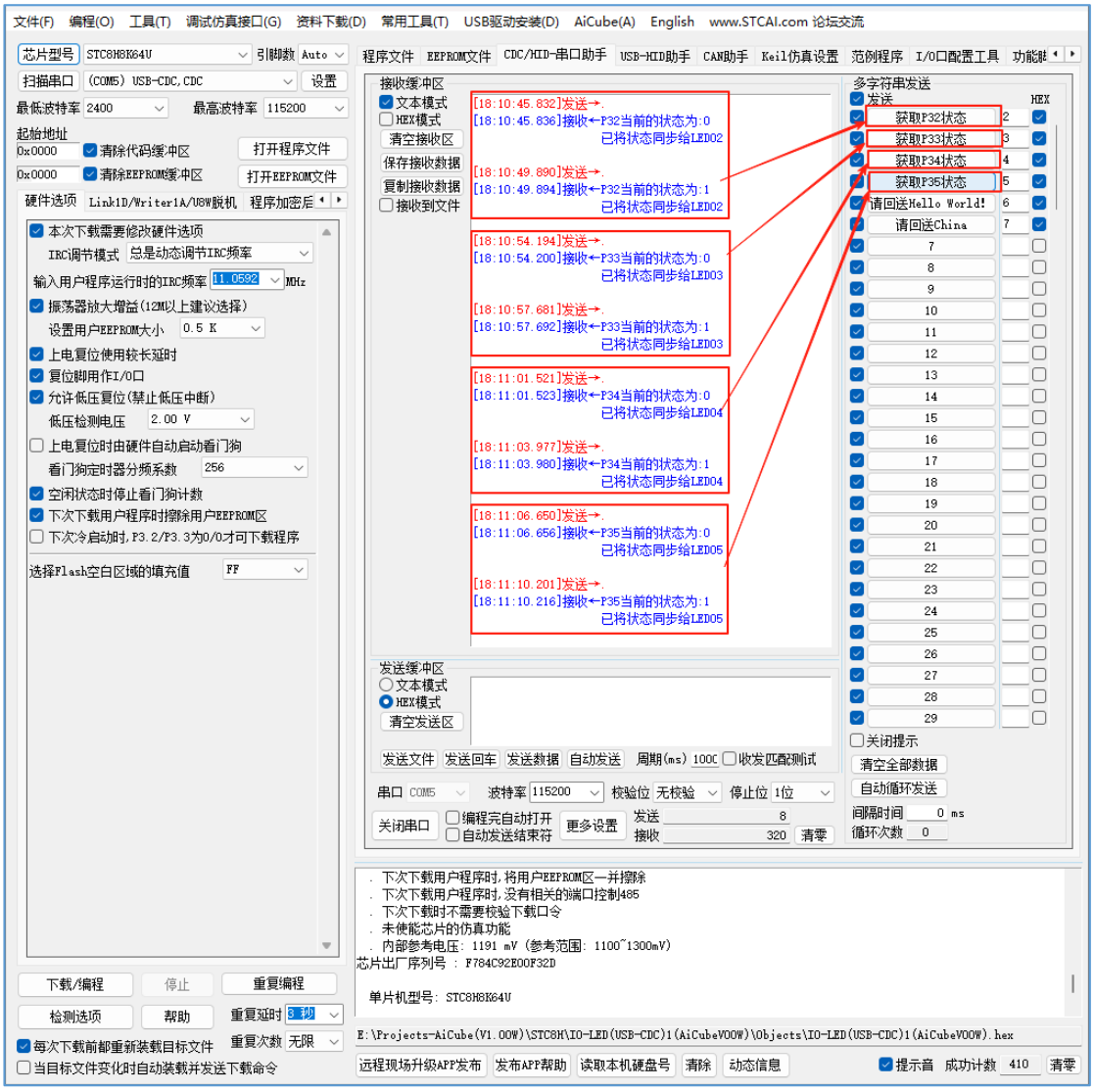


一切准备就绪！

➲ 点击“打开串口口”按钮

- ➢ 按住 P3.2 键，点击“获取 P32 的状态”按钮，MCU 读取实验箱上 P32 端口的状态并进行回传电脑的串口助手显示“P32 当前状态为 0”，同时实验箱上对应的 LED62 亮；
- ➢ 松开 P3.2 键，LED62 依然是亮的，再点击“获取 P32 的状态”按钮，MCU 读取实验箱上 P32 端口的状态并进行回传电脑的串口助手显示“P32 当前状态为 1”，同时实验箱上 LED62 灭；
- ➢ 按住 P3.3 键，点击“获取 P33 的状态”按钮，MCU 读取实验箱上 P33 端口的状态并进行回传电脑的串口助手显示“P33 当前状态为 0”，同时实验箱上对应的 LED63 亮；
- ➢ 松开 P3.3 键，LED63 依然是亮的，再点击“获取 P33 的状态”按钮，MCU 读取实验箱上 P33 端口的状态并进行回传电脑的串口助手显示“P33 当前状态为 1”，同时实验箱上 LED63 灭；
- ➢ 按住 P3.4 键，点击“获取 P34 的状态”按钮，MCU 读取实验箱上 P34 端口的状态并进行回传电脑的串口助手显示“P34 当前状态为 0”，同时实验箱上对应的 LED64 亮；
- ➢ 松开 P3.4 键，LED64 依然是亮的，再点击“获取 P34 的状态”按钮，MCU 读取实验箱上 P34 端口的状态并进行回传电脑的串口助手显示“P34 当前状态为 1”，同时实验箱上 LED64 灭；
- ➢ 按住 P3.5 键，点击“获取 P35 的状态”按钮，MCU 读取实验箱上 P35 端口的状态并进行回传电脑的串口助手显示“P35 当前状态为 0”，同时实验箱上对应的 LED65 亮；
- ➢ 松开 P3.5 键，LED65 依然是亮的，再点击“获取 P35 的状态”按钮，MCU 读取实验箱上 P35 端口的状态并进行回传电脑的串口助手显示“P35 当前状态为 1”，同时实验箱上 LED65 灭。

如下图：



有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.8.11 实验 4：用 USB-CDC 读取按键状态，在 ISP 软件的 【开天斧/屠龙刀-LED-DIP64(6)】 虚拟显示

电脑 AiCube-ISP 软件中的，USB-CDC 串口助手通知单片机读【P3.2/P3.3/P3.4/P3.5】的状态，送到 AiCube-ISP 软件中的【调试仿真接口 | 开天斧/屠龙刀-LED-DIP64(6)】虚拟显示，同时将相应的状态送到 LED【P6.2/P6.3/P6.4/P6.5】显示（点亮/熄灭）。

2.8.11.1 通信协议

AiCube-ISP 软件界面的“调试仿真接口”→“接口协议及帮助”→“DIP64 封装 LED 接口”

功能3： 控制DIP64的指定组管脚上LED的状态

命令格式： 4CH 45H 44H 40H 02H x1 px

命令说明：

第1～4字节： 命令头

第5字节： 后面有效数据长度

第6字节： P0～P7端口的屏蔽位
(01H:P0, 02H:P1, 04H:P2, 08H:P3,
10H:P4, 20H:P5, 40H:P6, 80H:P7)

第7字节： 端口的状态

使用串口通讯请参考下面的示例发送串口数据

示例1： 发送4CH 45H 44H 40H 02H 01H 55H
设置P0口的状态为55H

示例2： 发送4CH 45H 44H 40H 02H 04H AAH
设置P2口的状态为AAH

库函数声明：
```
void LED64_SetPort(BYTE port, BYTE dat);
```
*** 包含库函数的头文件后，不用另外声明 ***

项目中包含库函数后，可直接调用下面的库函数

库函数调用：
```
LED64_SetPort(0, 0x55);     //设置P0口输出55H
LED64_SetPort(4, 0xAA);     //设置P4口输出AAH
```

2.8.11.2 实验介绍

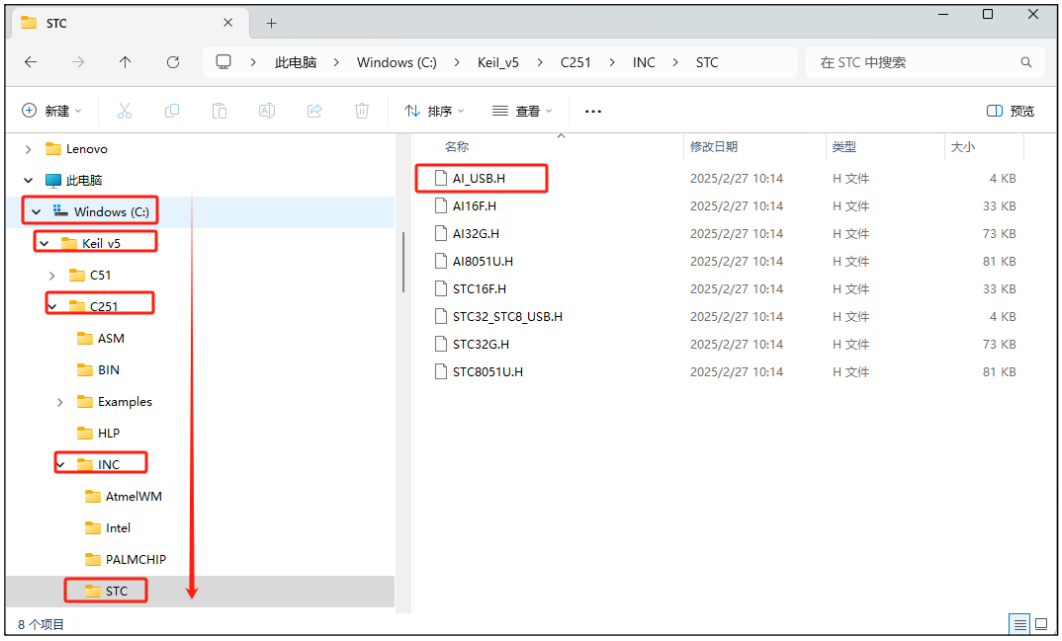
通过 USB-CDC 接口在“开天斧/屠龙刀-LED-DIP64(6)”能直观的查看到对应端口的状态，并且点亮/熄灭了对应的 LED 灯。

2.8.11.3 准备工作--添加头文件和库函数

1、添加常用头文件 ai_usb.h

```
#include "ai_usb.h"                    //使用 usb 所需要包含的头文件。
```

这个头文件在使用 AiCube-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



2、添加 usb_cdc 库函数文件

➲ 下载 usb_cdc 库函数文件

因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：
https://www.stcai.com/khs 的 USB 库文件中进行下载

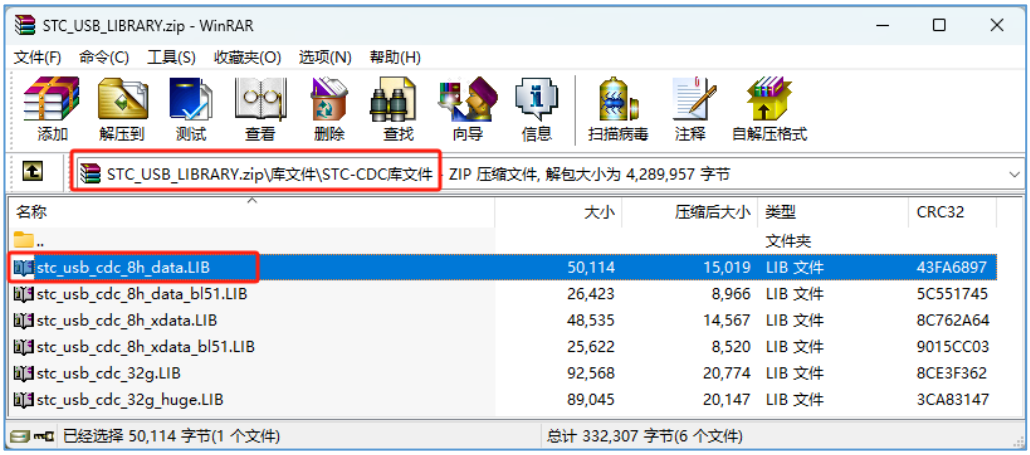


下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 **stc_usb_cdc_8h_data.LIB**

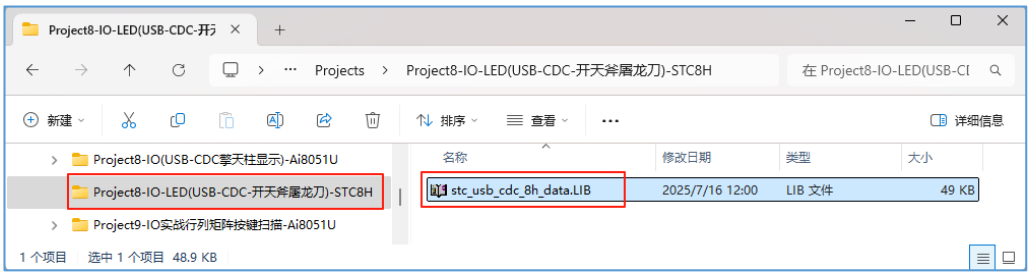
【8 位的 8051 的 USB 库 可以用 BL51 连接器
BL51：　不支持 REMOVEUNUSED；BL51 支持 扩展中断号工具；
如果 不用 REMOVEUNUSED，可以 使用 BL51 ！

LX51：　支持 REMOVEUNUSED；LX51 不支持 扩展中断号工具；
如果 不用 扩展中断号工具，可以 使用 LX51 ！

详情请参考本公司技术论坛链接：8 位的 8051 的 USB 库可以用 BL51；BL51 不支持 REMOVEUNUSED；
https://www.stcaimcu.com/thread-17105-1-1.html
(出处: 国芯人工智能技术交流网站)】

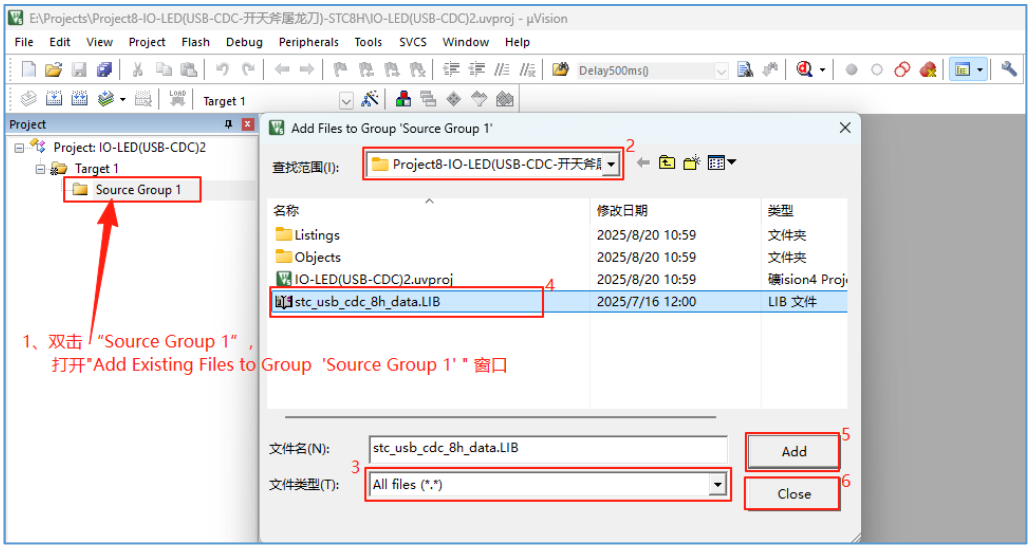


➲ **将库函数“stc_usb_cdc_8h_data.LIB”复制到创建项目所在的文件夹中**

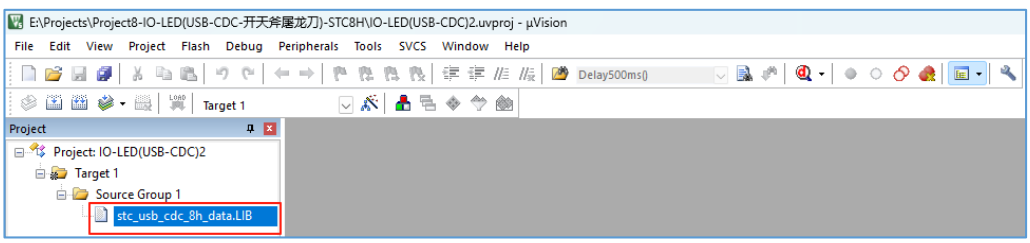


➲ **将库函数“stc_usb_cdc_8h_data.LIB”添加进入此项目重新文件中**

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 **stc_usb_cdc_8h_data.LIB** 添加进入此工程，就可以使用 USB-CDC 功能进行端口通讯。



Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

2.8.11.4 实验程序 C 代码

```
#include <STC8H.H>    //包含 STC8H 的头文件
#include   "ai_usb.h"  //USB 库和 USB 虚拟设备的头文件
                       //由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
                       //只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
                       //即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。
#include "intrins.h"   //使用_nop_()函数所必须要包含的头文件，
                       //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                       //会导致编译器找不到这个而函数而报错。

void Delay1ms(void)    //延时 1ms@11.0592MHz
{
    unsigned char data i, j;

    i = 15;
    j = 90;
    do
    {
        while (--j);
    } while (--i);
}

void main(void)
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式

    P3M0 = 0x30; P3M1 = 0x3b;
                       //P30、P31 为高阻输入，P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏输出
    P3PU = 0x38;       //打开 P33、P34、P35 的内部上拉电阻，并且仅修改这三个 I/O 的上拉电阻打开状态
    P3 |= 0x3c;        //需要读的 I/O，对外输出锁存器先设置为 1，0011 1100B（0x3c）先和 P3 口逻辑或一下

    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式

    P6M0 = 0x38; P6M1 = 0x20;       //设置 P63、P64 为推挽输出，P65 为开漏输出，其余为准双向口模式

    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    Delay1ms();                     //延时 1ms，等待 I/O 口电平稳定，因为前面刚改变了模式
                                    //立刻读可能会读到错误的电平，所以需要先等待电平变化稳定后再读
```

```
    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */

    EA = 1;                         //总中断允许位打开

    P40 = 0;                        //打开 LED 部分的供电

    while(1)
    {                               //使用 USB 串口控制来手动控制 LED 部分
        if(bUsbOutReady)
            //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
        {
            LED64_SetPort(6, P3 | ~ (1<<UsbOutBuffer[0]));
                                    //使用 AiCube-ISP 软件菜单“调试仿真接口”中的
                                    //“开天斧/屠龙刀-LED-DIP64(6)”进行模拟显示 P6 口

//          P6 = P3 | ~(1<<UsbOutBuffer[0]);    //同步端口状态，并将其他不需要的位置 1（不显示）
//下面的 switch 等同于上一句，上一句 C 语言高手才能看懂
            switch (UsbOutBuffer[0])            //判断接收的第一个字节数据
            {
                case 2: P6 = P3 | 0xfb; break;  //如果接收到 2 则将 P3.2 状态送 P6.2，
                                                //其他不需要的位置 1（不显示），1111 1011B
                case 3: P6 = P3 | 0xf7; break;  //如果接收到 3 则将 P3.3 状态送 P6.3，
                                                //其他不需要的位置 1（不显示），1111 0111B
                case 4: P6 = P3 | 0xef; break;  //如果接收到 4 则将 P3.4 状态送 P6.4，
                                                //其他不需要的位置 1（不显示），1110 1111B
                case 5: P6 = P3 | 0xdf; break;  //如果接收到 5 则将 P3.5 状态送 P6.5，
                                                //其他不需要的位置 1（不显示），1101 1111B
            }
            usb_OUT_done();     //本包接收的数据处理完毕，准备接收下一包数据
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
        }
    }
}
```

2.8.11.5　程序下载

录入代码，保存，编译无误。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤“打开程序文件”IO-LED(USB-CDC)2.hex（“打开程序文件”是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中）并“下载/编程”，如下图：



这时在【AiCube-ISP】界面，我们看到串口选项框中已经出现了一个(COM5)USB-CDC,CDC 串口，表明现在可以通过这个 USB-CDC 端口进行数据通讯了。如果电脑系统并没有产生 CDC 端口，可能是因为系统版本较低（低于 Windows10 版本）。

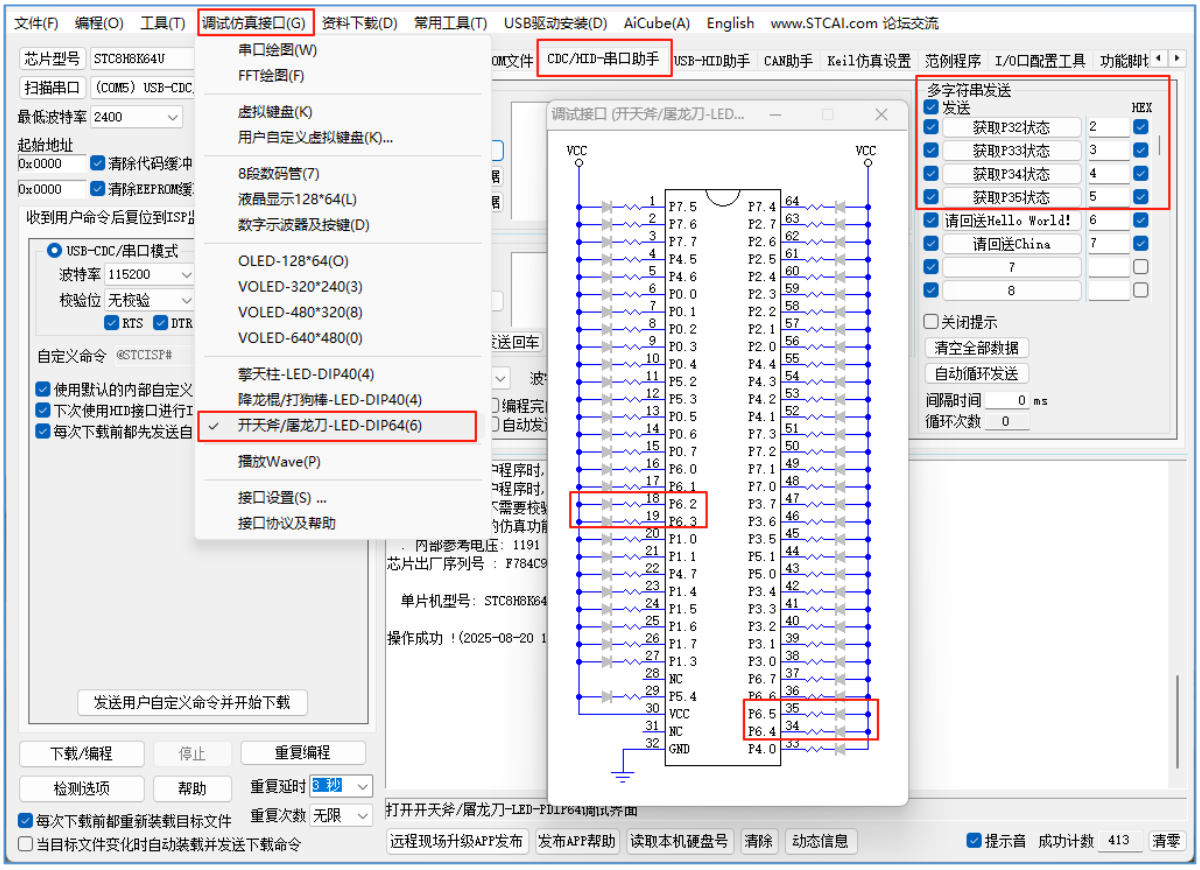
2.8.11.6　程序实现效果，观察实验现象

程序编写好，编译无误后加载到实验箱，就可以通过 AiCube-ISP 软件界面的“调试仿真接口”→“开天斧/屠龙刀-LED-DIP64(6)”虚拟显示对应端口的状态。实现**按下对应按键后，**“开天斧/屠龙刀-LED-DIP64(6)”显示点亮/熄灭了对应的 LED 灯。

打开【AiCube-ISP】界面的“CDC/HID-串口助手”选项卡，

- ➲ 在右侧的“多字符串发送”窗口，设置对应的文字和需要发送的内容。
  在输入窗口分别输入：2，则将 P3.2 状态送 P6.2
  3，则将 P3.3 状态送 P6.3
  4，则将 P3.4 状态送 P6.4
  5，则将 P3.5 状态送 P6.5
- ➲ 串口：选择与“扫描串口”窗口显示的 USB-CDC 端口一致
- ➲ 打开菜单“调试仿真接口”→“开天斧/屠龙刀-LED-DIP64(6)”，虚拟显示对应端口的状态。

如下图：

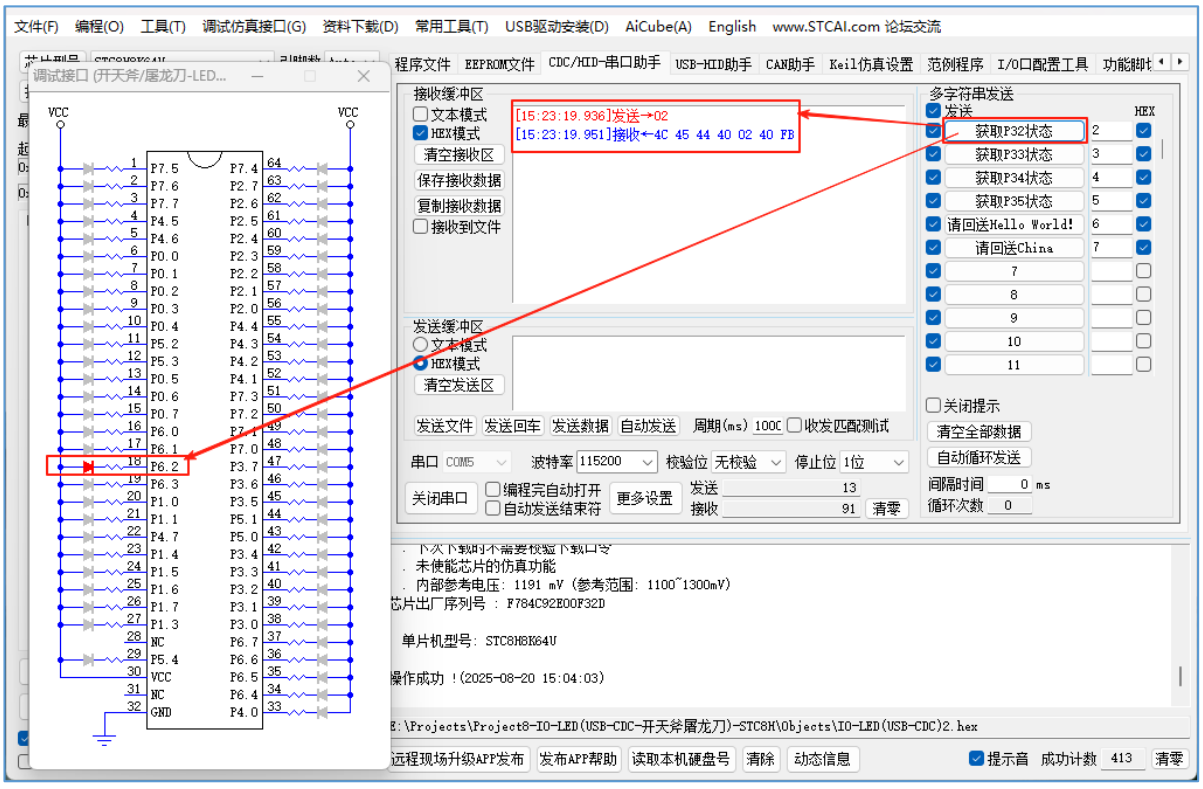


一切准备就绪！

➲ 点击“打开端口”按钮

➢ 按住 P3.2 键，点击“获取 P32 的状态”按钮

MCU 读取实验箱上 P32 端口的状态为“0”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，点亮 P6.2；同时实验箱上对应的 LED62 亮；



- 松开 P3.2 键，“开天斧/屠龙刀-LED-DIP64(6)”上的 P6.2 是亮的，实验箱上 LED62 也是亮的，再点击“获取 P32 的状态”按钮
  MCU 读取实验箱上 P32 端口的状态为“1”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，P6.2 灭，同时实验箱上 LED62 灭；

- 按住 P3.3 键，点击“获取 P33 的状态”按钮
  MCU 读取实验箱上 P33 端口的状态为“0”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，点亮 P6.3；同时实验箱上对应的 LED63 亮；
- 松开 P3.3 键，“开天斧/屠龙刀-LED-DIP64(6)”上的 P6.3 是亮的，实验箱上 LED63 也是亮的，再点击“获取 P33 的状态”按钮
  MCU 读取实验箱上 P33 端口的状态为“1”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，P6.3 灭，同时实验箱上 LED63 灭；

- 按住 P3.4 键，点击“获取 P34 的状态”按钮
  MCU 读取实验箱上 P34 端口的状态为“0”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，点亮 P6.4；同时实验箱上对应的 LED64 亮；
- 松开 P3.4 键，“开天斧/屠龙刀-LED-DIP64(6)”上的 P6.4 是亮的，实验箱上 LED64 也是亮的，再点击“获取 P34 的状态”按钮
  MCU 读取实验箱上 P34 端口的状态为“1”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，P6.4 灭，同时实验箱上 LED64 灭；

- 按住 P3.5 键，点击“获取 P35 的状态”按钮
  MCU 读取实验箱上 P35 端口的状态为“0”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，点亮 P6.5；同时实验箱上对应的 LED65 亮；
- 松开 P3.5 键，“开天斧/屠龙刀-LED-DIP64(6)”上的 P6.5 是亮的，实验箱上 LED65 也是亮的，再点击“获取 P35 的状态”按钮
  MCU 读取实验箱上 P35 端口的状态为“1”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，P6.5 灭，同时实验箱上 LED65 灭；

完成实验。

2.8.12 实验 4A--使用【AiCube】项目助手实现“用 USB-CDC 读取按键状态，并在【开天斧/屠龙刀-LED-DIP64(6)】 虚拟显示”

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit、Ai8051U-8Bit、STC32G12K128、STC8H8K64U、STC15F2K60S2、STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

2.8.12.1 安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj



下载下来的是压缩包文件图标，如下图：



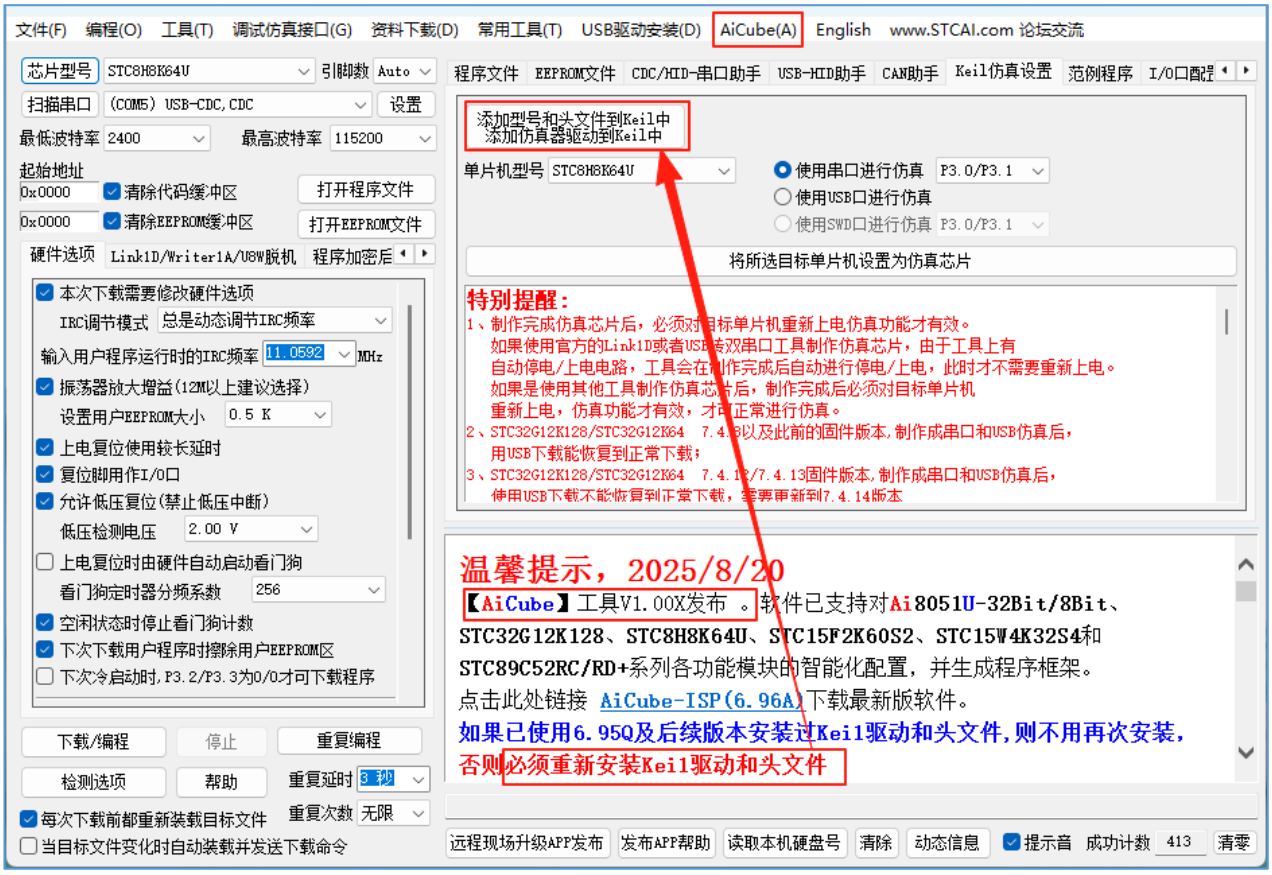
解压缩后的 AiCube-ISP 文件图标，如下图：



如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

2.8.12.2　使用【AiCube】助手创建项目

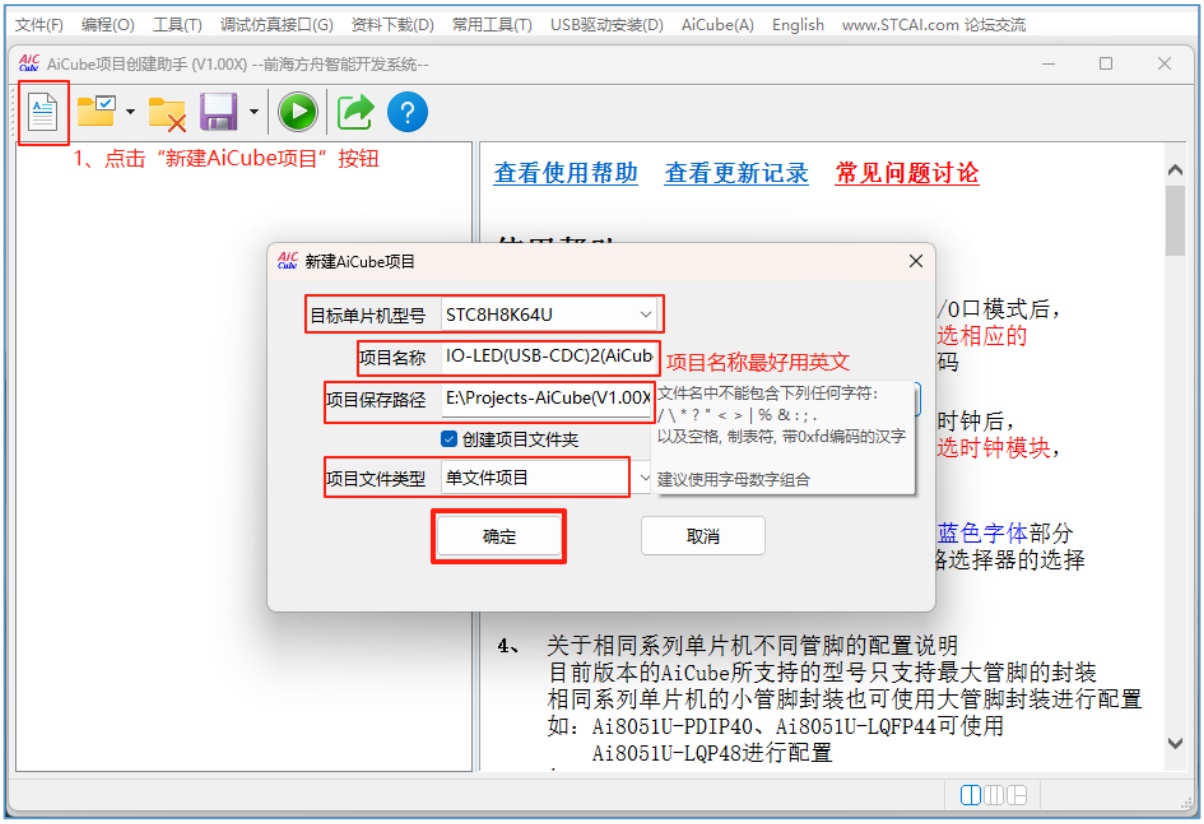
➲　双击图标，打开 AiCube-ISP 软件界面，如下图：



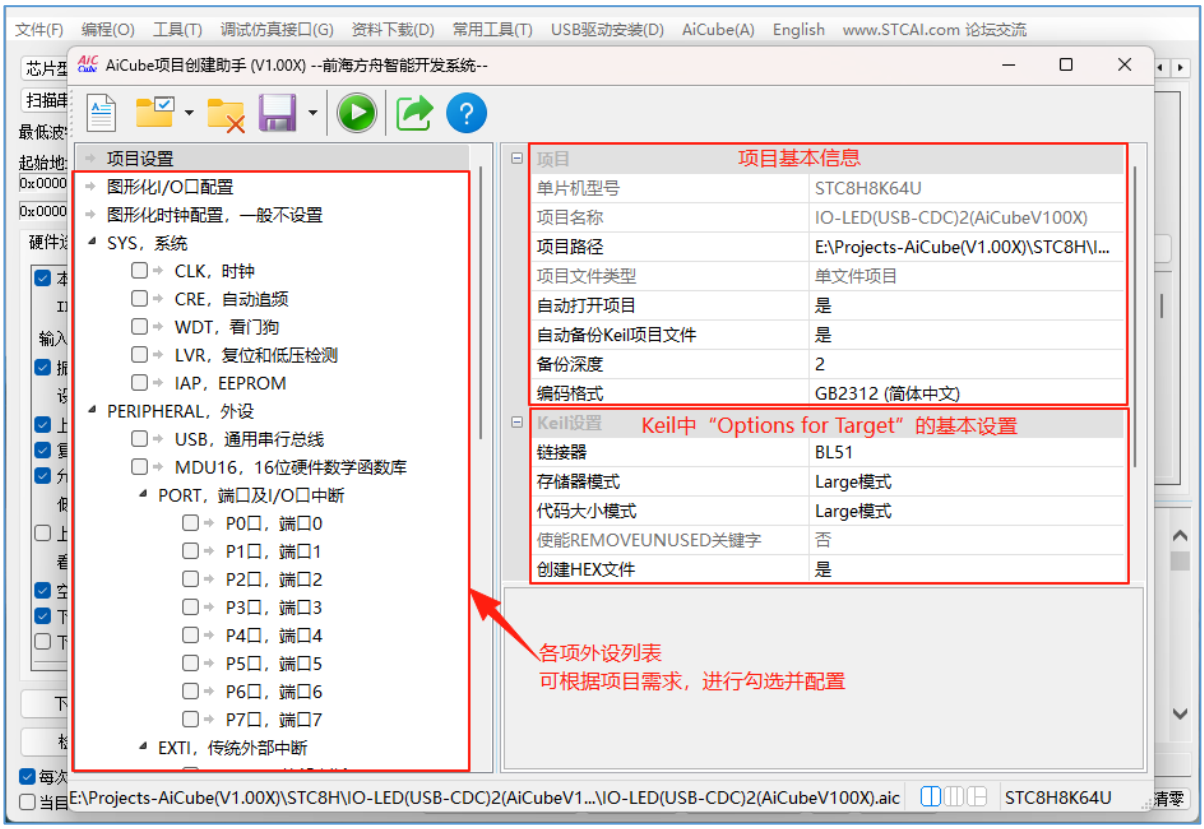
➲　点击【AiCube（A）】，弹出如下窗口：



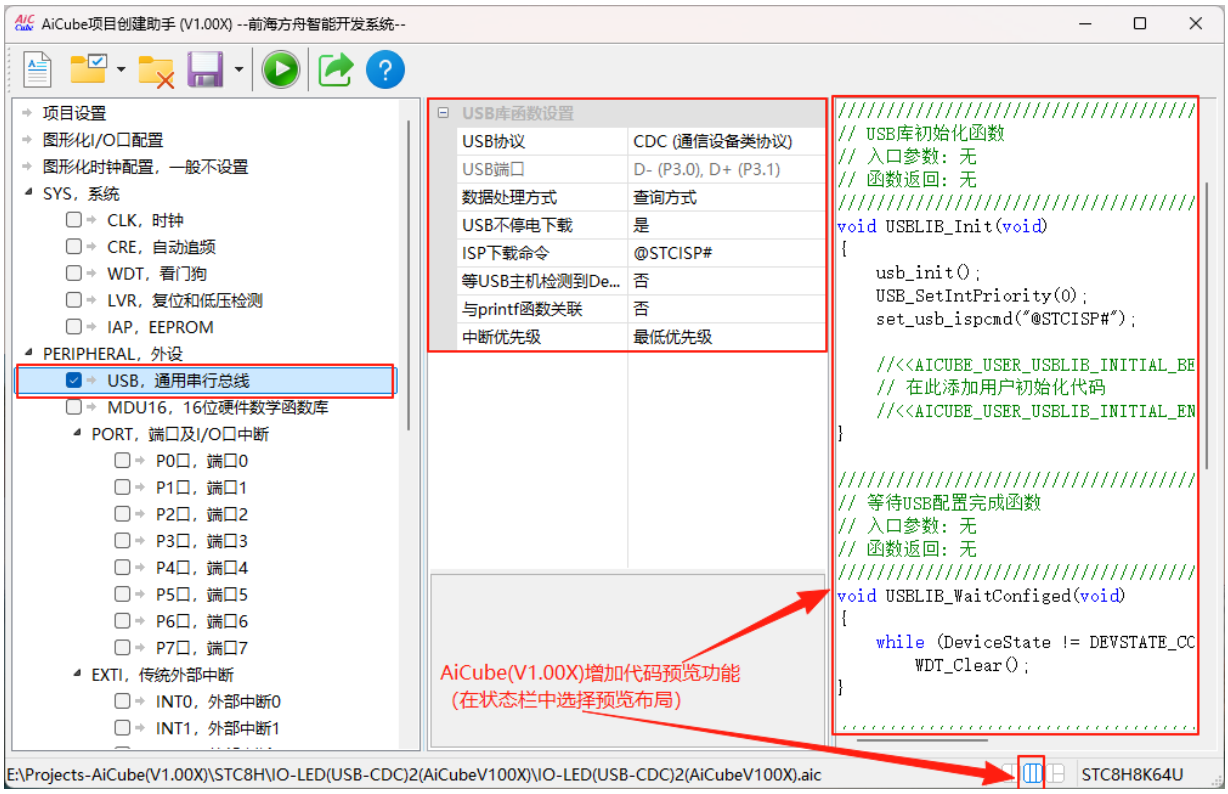
➲ **点击“新建 AiCube 项目”按钮**，选择并填写项目基本信息，建议项目名称用英文！



➲ **点击“确定”按钮**，出现如下配置界面：



➲ 本实验是通过【AiCube-ISP】系统软件中的 USB-CDC 串口助手通知单片机读【P3.2/P3.3/P3.4/P3.5】的状态，送到【调试仿真接口 | 开天斧/屠龙刀-LED-DIP64(6)】虚拟显示，同时将相应的状态送到 LED【P6.2/P6.3/P6.4/P6.5】显示（点亮/熄灭）。显然要**勾选 USB 选项**，选择“CDC（通信设备类协议）”，如下图：

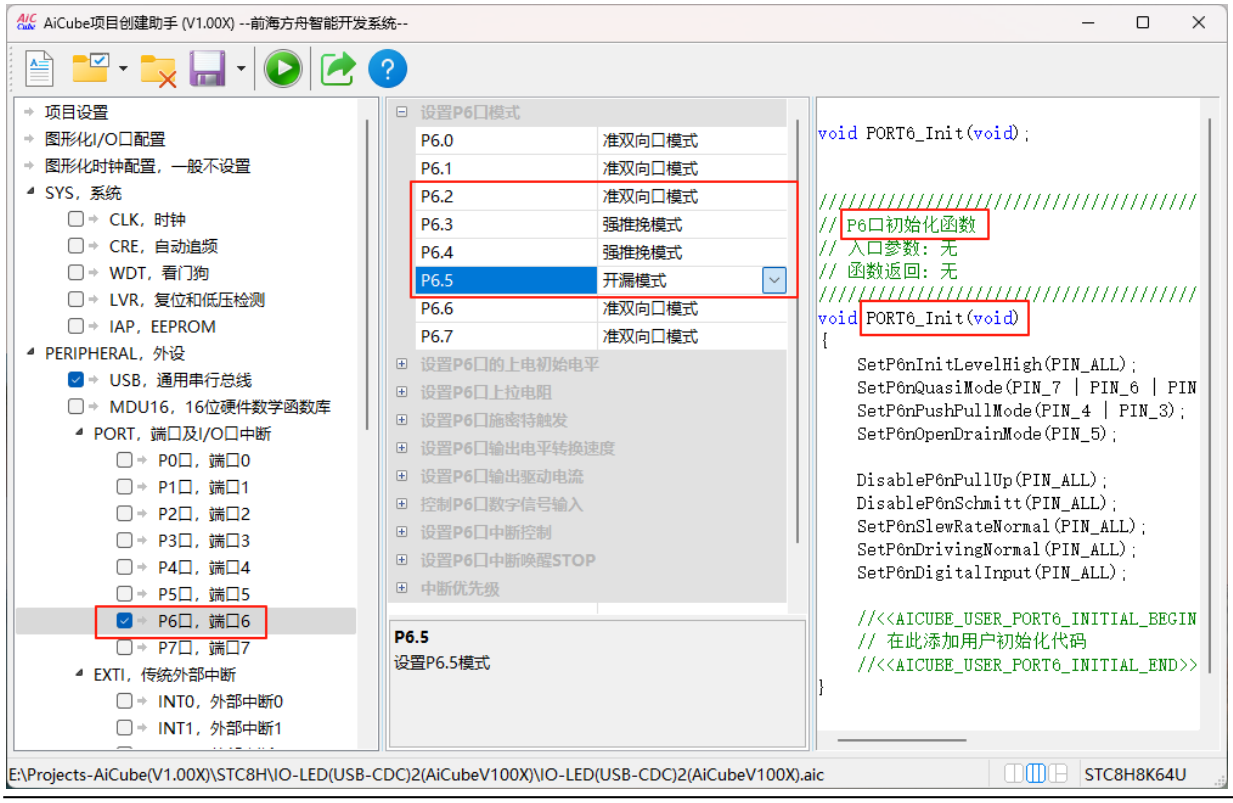


对于本项目，需要单片机读【P3.2 / P3.3 / P3.4 / P3.5】这 4 个 I/O 的状态，
输出到【P6.2 / P6.3 / P6.4 / P6.5】对应的指示灯。

➲ I/O 配置

针对 I/O 有 4 种不同的模式，我们将【P3.2 / P3.3 / P3.4 / P3.5】不同的 I/O 口设置为不同的模式，验证不同情况下依然可以正常输入/输出。

- 【AiCube】助手中，端口默认是准双向口模式，我们配置**输出端口**的 P62 保持准双向口模式，P63、P64 为强推挽输出，P65 为开漏输出（可以输出 0 点亮 LED）。如下图：



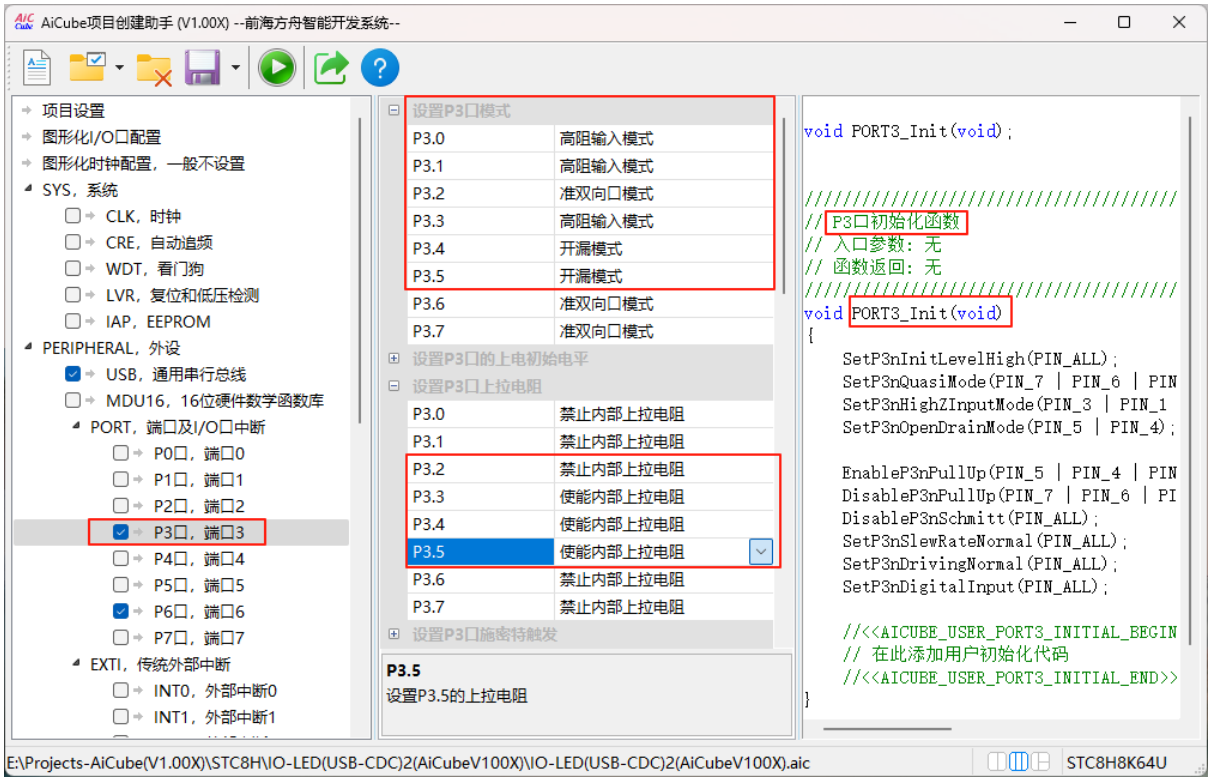
➢ **输入配置**

✓ P30、P31 为高阻输入(因为跟 USB 复用，建议设置为高阻输入，不干扰 USB 的通讯控制)

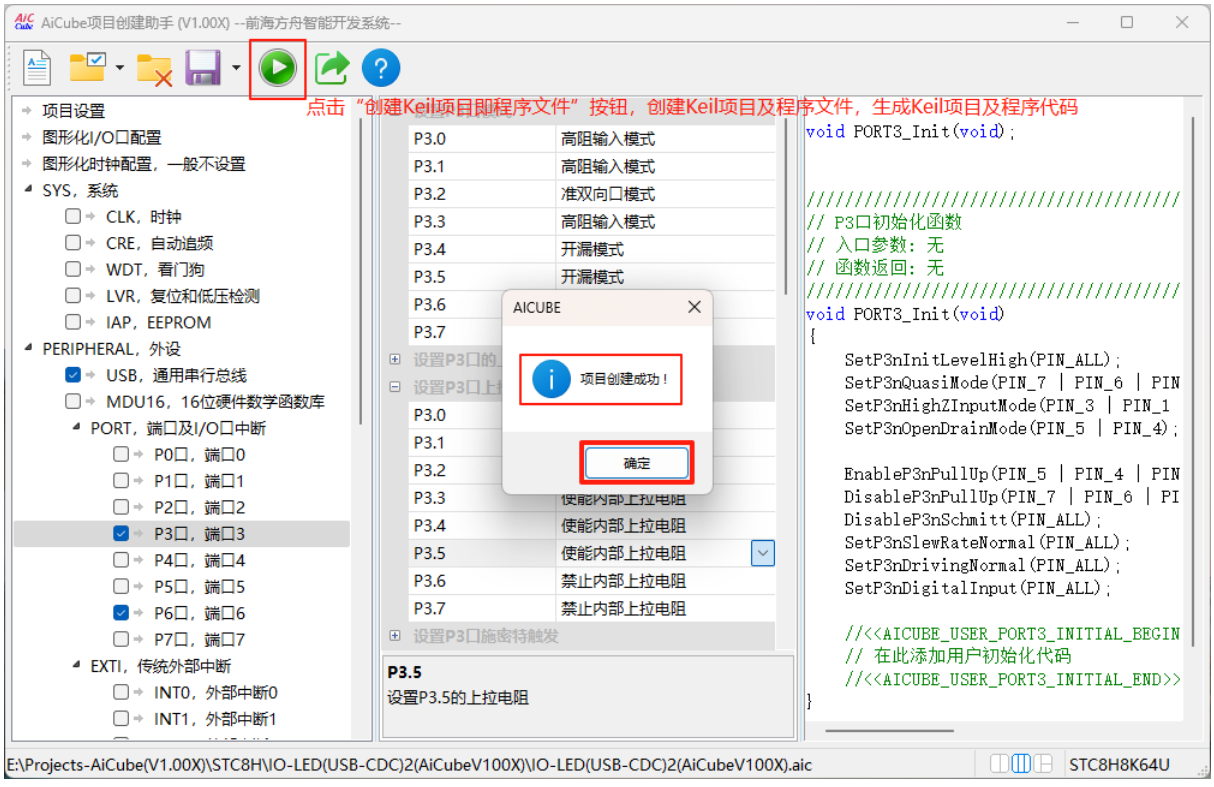
✓ 设置 P32 为准双向口、P33 为高阻输入、P34 和 P35 为开漏模式（等效高阻输入）

✓ 开启 P33、P34、P35 的内部上拉电阻以稳定电平状态（不开启内部上拉电阻读外部状态，高阻输入/开漏模式，外部都需要有上拉电阻配合）

【AiCube】助手中，端口默认是准双向口模式，内部上拉电阻默认是禁用设置。如下图：



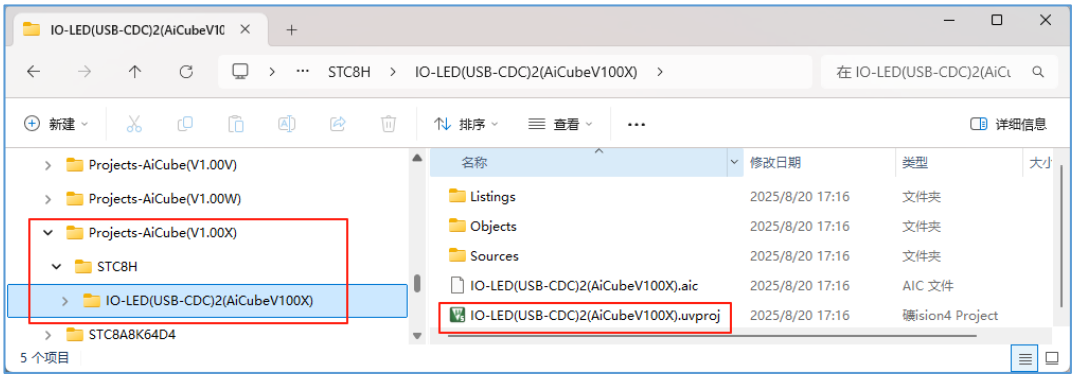
➲ **点击“创建 Keil 项目及程序文件”按钮**，【AiCube】助手提示：“项目创建成功！”，如下图：



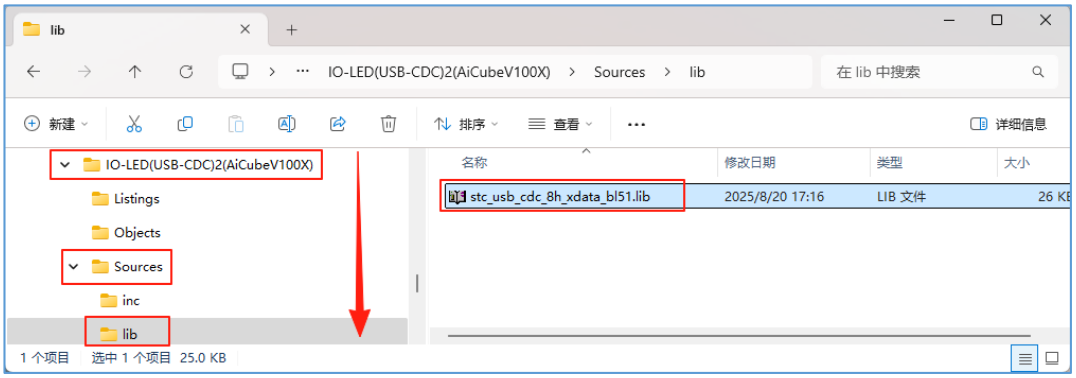
➲ **点击“确定”**，立即自动打开 Keil 软件，进入该项目的 Keil 集成开发界面，即【AiCube】成功帮我们创建了项目及文件目录“IO-LED(USB-CDC)2(AiCubeV100X)”。至此，可以关闭【AiCube】。

➲ 找到创建的项目目录

发现通过上述操作，【AiCube】助手已经帮我们建立了项目目录及 IO-LED(USB-CDC)2(AiCubeV100X).uvproj 文件，如下图：

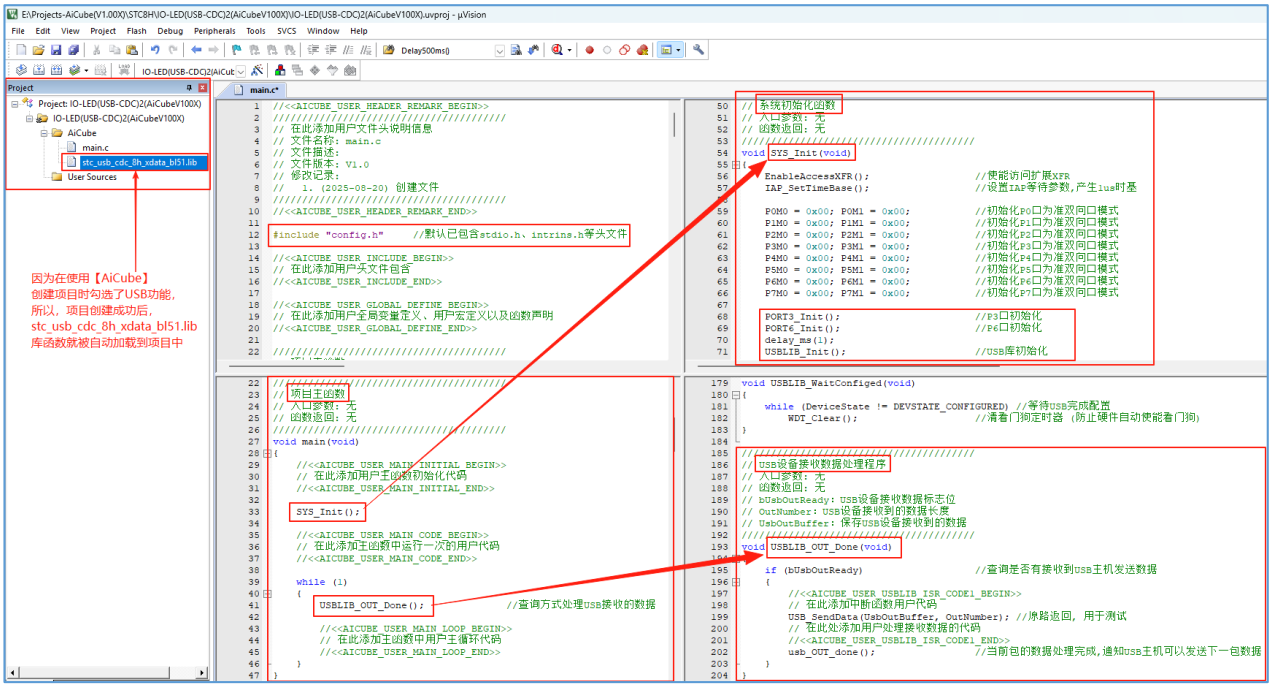


因为勾选了 USB 选项，打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经加载了：



➲ 观察已打开的 Keil 项目文件：IO-LED(USB-CDC)2(AiCubeV100X).uvproj

如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，全代码观察）：

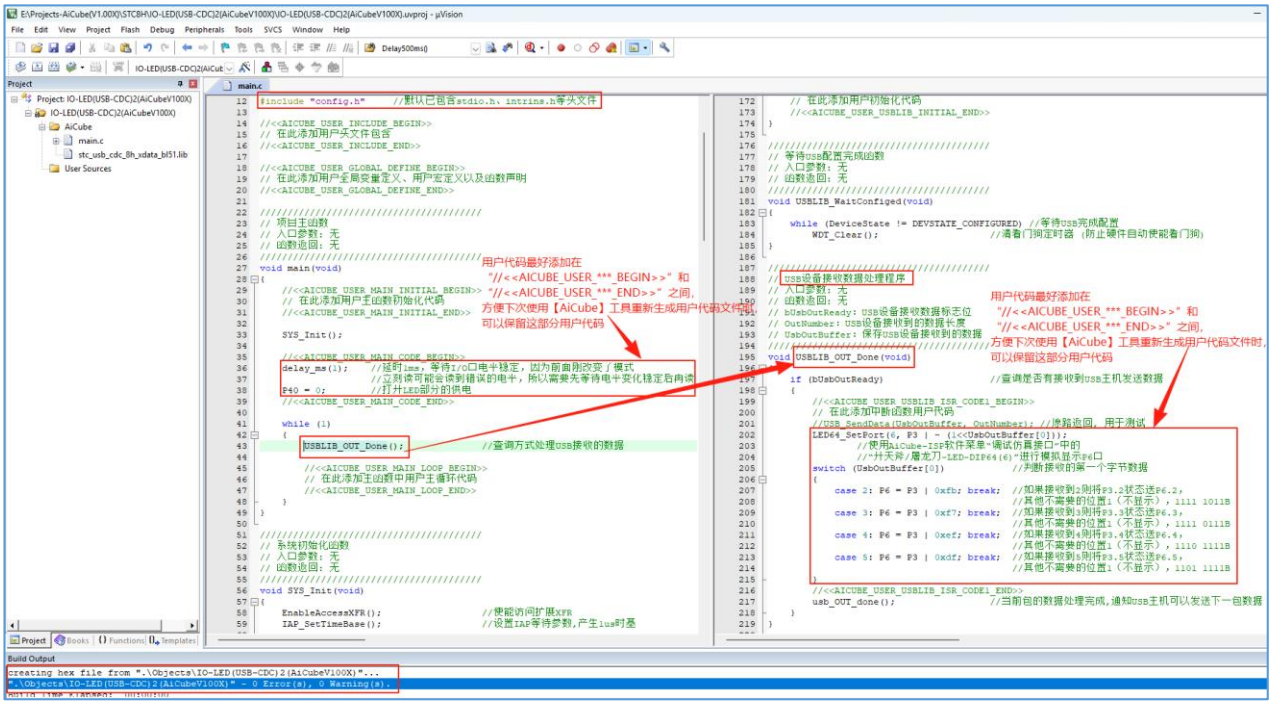


根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c，并自动加载 USB 库函数（stc_usb_cdc_8h_xdata_bl51.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、毫秒和微秒延时函数、P0 口和 P3 口初始化函数、USB 库初始化函数、等待 USB 配置完成函数、USB 设备接收数据处理程序。

2.8.12.3　增加实验项目代码

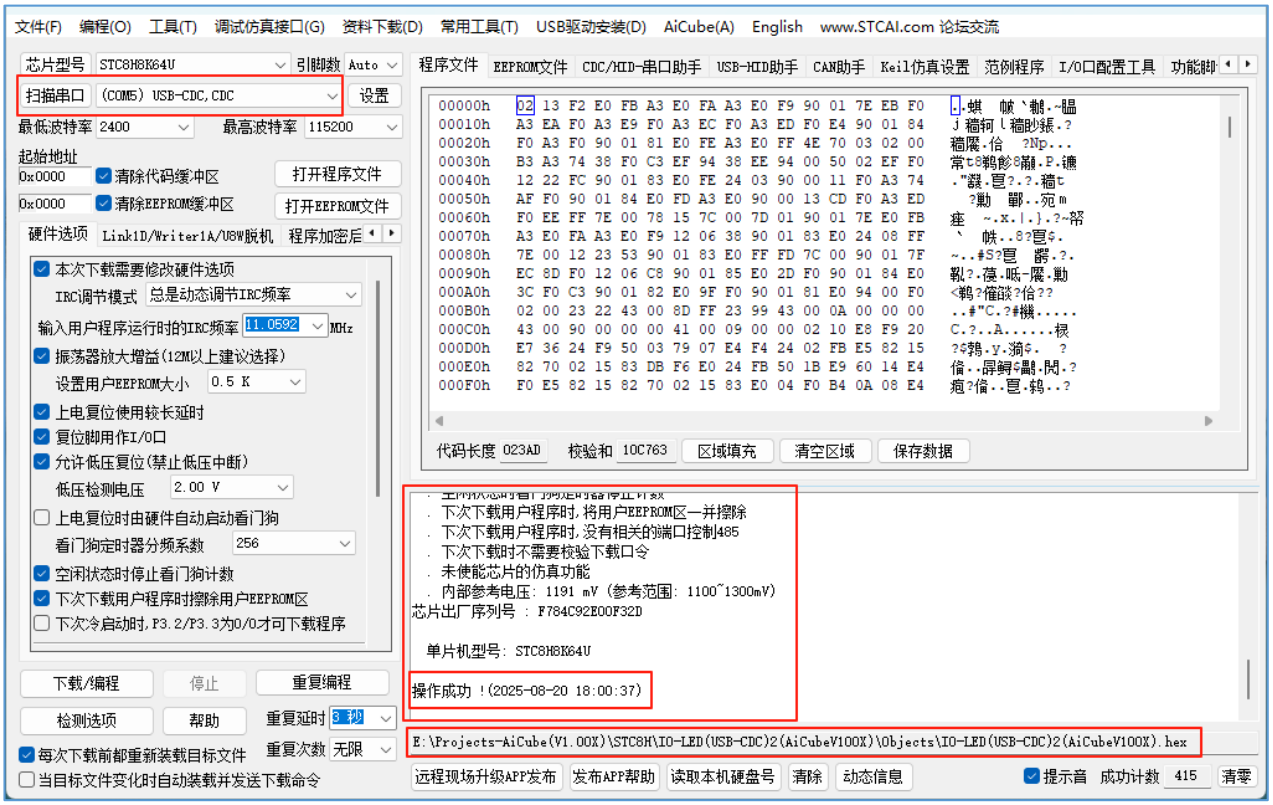
为实现通过 AiCube-ISP 系统软件中的 USB-CDC 串口助手读取【P3.2/P3.3/P3.4/P3.5】的状态，送回 USB-CDC 串口助手显示，同时将相应的状态送到 LED 的【P6.2/P6.3/P6.4/P6.5】显示（点亮/熄灭）。我们现在需要：

- 在主函数中，调用延时函数“delay_ms(1);”，延时 1ms（等待 I/O 口电平稳定）；
- 在主函数中，加入“P40 = 0;”，打开 LED 供电的代码；
- 在“USB 设备接收数据处理程序”子函数中，补充 USB 控制代码；
- 保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成两个窗口，对比观察）：



2.8.12.4　使用 AiCube-ISP 软件系统，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果

连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“IO-LED(USB-CDC)2(AiCubeV100X).hex”到实验箱，如下图：



这时在【AiCube-ISP】界面，我们看到串口选项框中已经出现了一个(COM5)USB-CDC,CDC 串口，表明现在可以通过这个 USB-CDC 端口进行数据通讯了。

程序编写好，编译无误后加载到实验箱，就可以通过 AiCube-ISP 软件界面的“调试仿真接口”→“开天斧/屠龙刀-LED-DIP64(6)”虚拟显示对应端口的状态。实现**按下对应按键后，**“开天斧/屠龙刀-LED-DIP64(6)”显示点亮/熄灭了对应的 LED 灯。

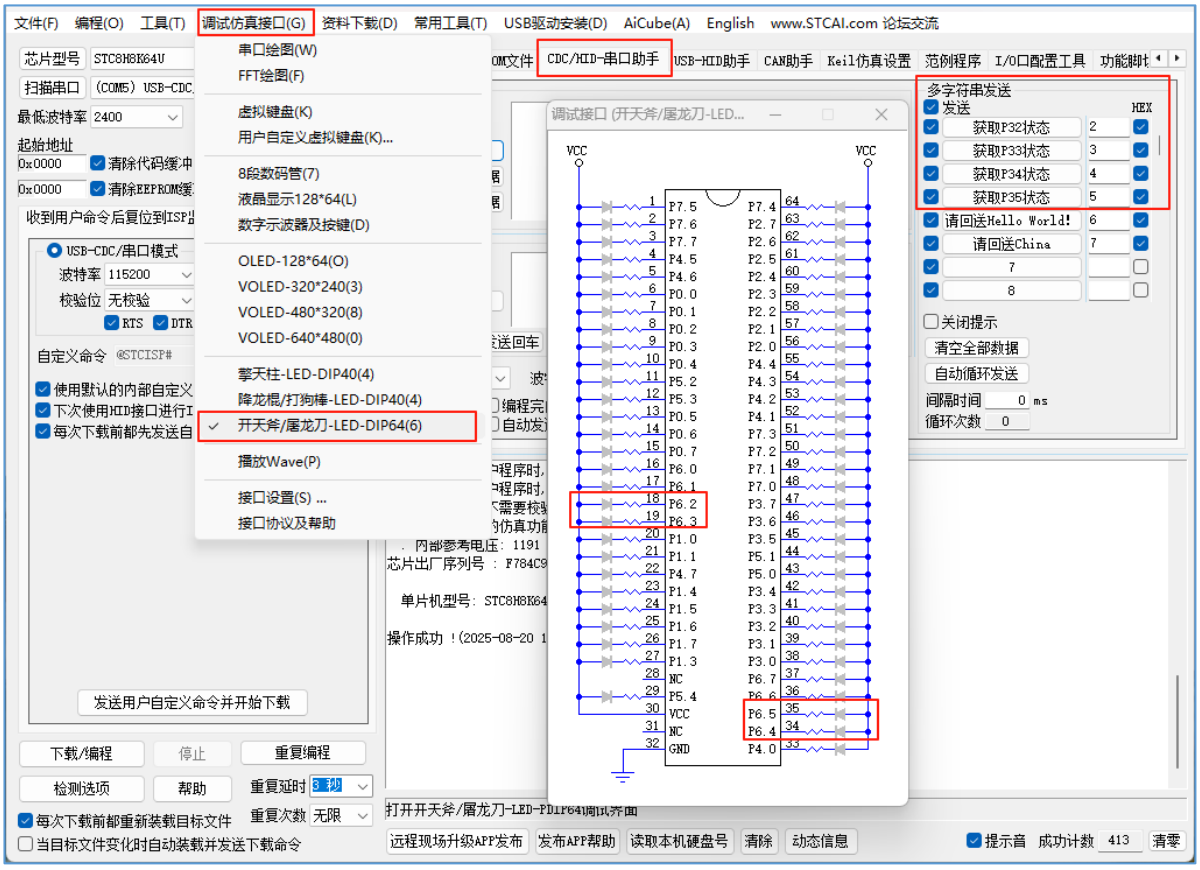
打开【AiCube-ISP】界面的“CDC/HID-串口助手”选项卡，

➲ 在右侧的“多字符串发送”窗口，设置对应的文字和需要发送的内容。
在输入窗口分别输入：2，则将 P3.2 状态送 P6.2
3，则将 P3.3 状态送 P6.3
4，则将 P3.4 状态送 P6.4
5，则将 P3.5 状态送 P6.5

➲ 串口：选择与“扫描串口”窗口显示的 USB-CDC 端口一致

➲ 打开菜单“调试仿真接口”→“开天斧/屠龙刀-LED-DIP64(6)”，虚拟显示对应端口的状态。

如下图：

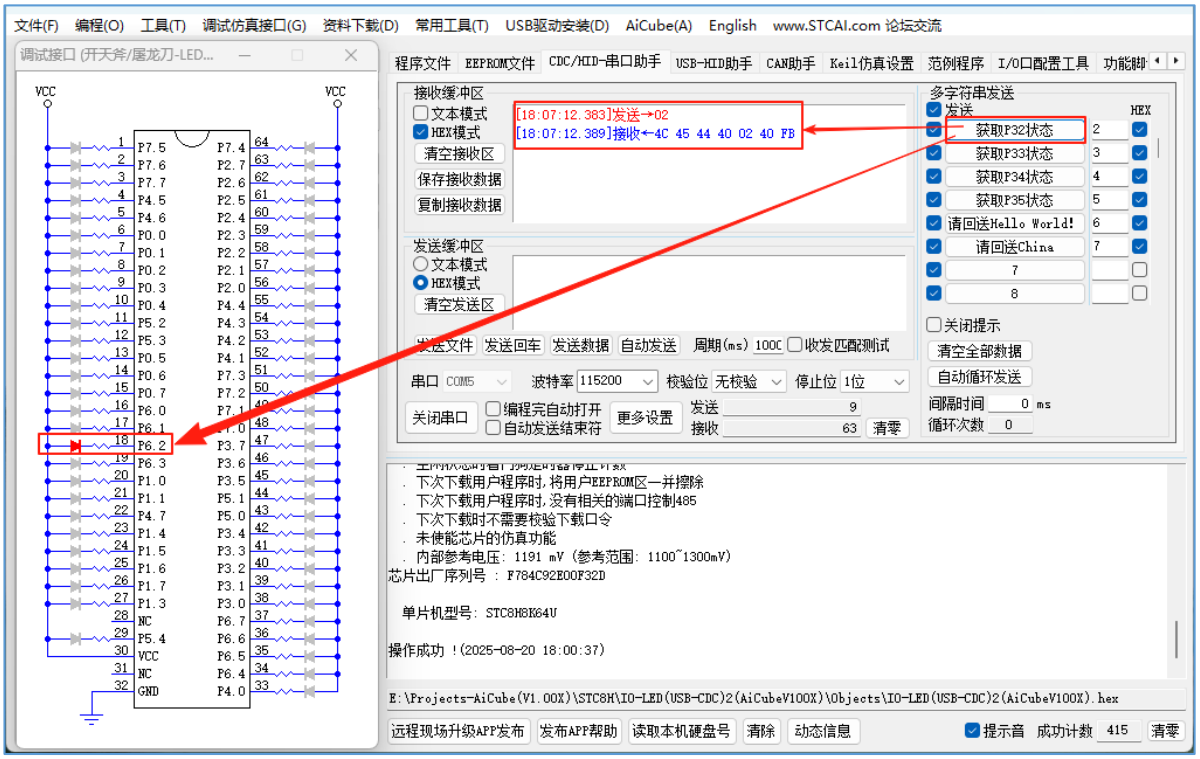


一切准备就绪！

➲ 点击“打开端口”按钮

➢ 按住 P3.2 键，点击“获取 P32 的状态”按钮

MCU 读取实验箱上 P32 端口的状态为“0”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，点亮 P6.2；同时实验箱上对应的 LED62 亮；



- 松开 P3.2 键，“开天斧/屠龙刀-LED-DIP64(6)”上的 P6.2 是亮的，实验箱上 LED62 也是亮的，再点击“获取 P32 的状态”按钮
  MCU 读取实验箱上 P32 端口的状态为“1”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，P6.2 灭，同时实验箱上 LED62 灭；

- 按住 P3.3 键，点击“获取 P33 的状态”按钮
  MCU 读取实验箱上 P33 端口的状态为“0”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，点亮 P6.3；同时实验箱上对应的 LED63 亮；
- 松开 P3.3 键，“开天斧/屠龙刀-LED-DIP64(6)”上的 P6.3 是亮的，实验箱上 LED63 也是亮的，再点击“获取 P33 的状态”按钮
  MCU 读取实验箱上 P33 端口的状态为“1”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，P6.3 灭，同时实验箱上 LED63 灭；

- 按住 P3.4 键，点击“获取 P34 的状态”按钮
  MCU 读取实验箱上 P34 端口的状态为“0”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，点亮 P6.4；同时实验箱上对应的 LED64 亮；
- 松开 P3.4 键，“开天斧/屠龙刀-LED-DIP64(6)”上的 P6.4 是亮的，实验箱上 LED64 也是亮的，再点击“获取 P34 的状态”按钮
  MCU 读取实验箱上 P34 端口的状态为“1”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，P6.4 灭，同时实验箱上 LED64 灭；

- 按住 P3.5 键，点击“获取 P35 的状态”按钮
  MCU 读取实验箱上 P35 端口的状态为“0”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，点亮 P6.5；同时实验箱上对应的 LED65 亮；
- 松开 P3.5 键，“开天斧/屠龙刀-LED-DIP64(6)”上的 P6.5 是亮的，实验箱上 LED65 也是亮的，再点击“获取 P35 的状态”按钮
  MCU 读取实验箱上 P35 端口的状态为“1”，并将此状态回传到“开天斧/屠龙刀-LED-DIP64(6)”调试助手，P6.5 灭，同时实验箱上 LED65 灭；

2.8.13 实验 5：I/O 口行列按键扫描，蜂鸣器发声实战

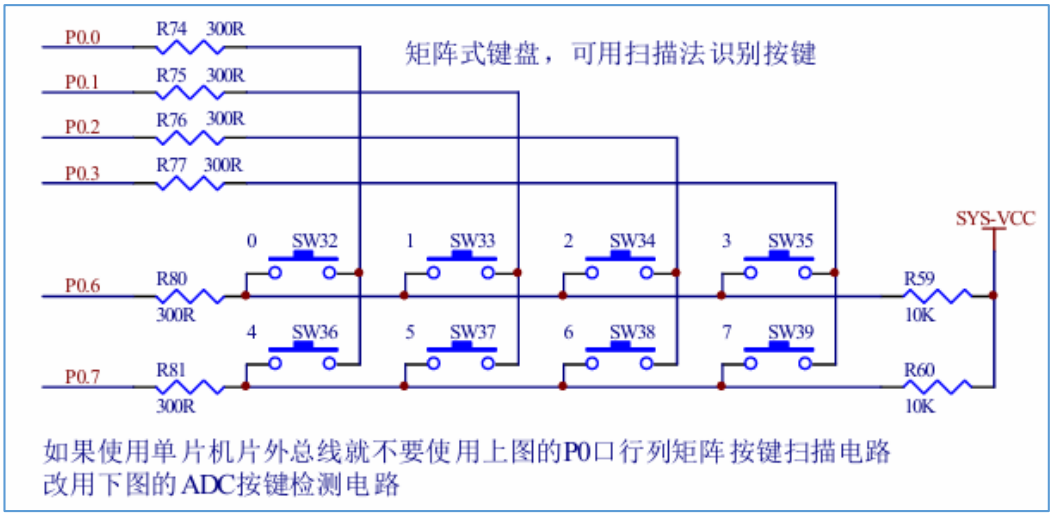
2.8.13.1 实验介绍

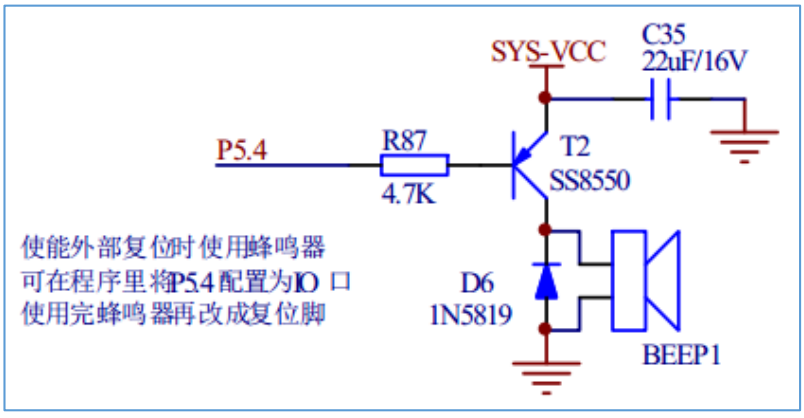
用行列扫描方法扫描 2 行 4 列 8 按键的矩阵式键盘，按键按下时驱动蜂鸣器发声，同时通过 USB-CDC 将按下按键的键码发到【CDC/HID-串口助手】显示。

➲ 按键扫描说明：

如 1 个按键要判断为正常按下，而不是抖动，
假定需要该按键保持按下持续保持为低状态时间是 50ms ~ 200ms；
假定 1 个主循环的运行时间大概 10ms ~ 30ms
则如发现某个按键按下，需要连续 6 次主循环都能读到该按键按下
===如发现多个按键按下，则归 0，重新开始
调整按键的灵敏度只需要修改，
===【判断按键已按下的时间，即读到为低经历了多少次主循环的次数】

2.8.13.2 原理图



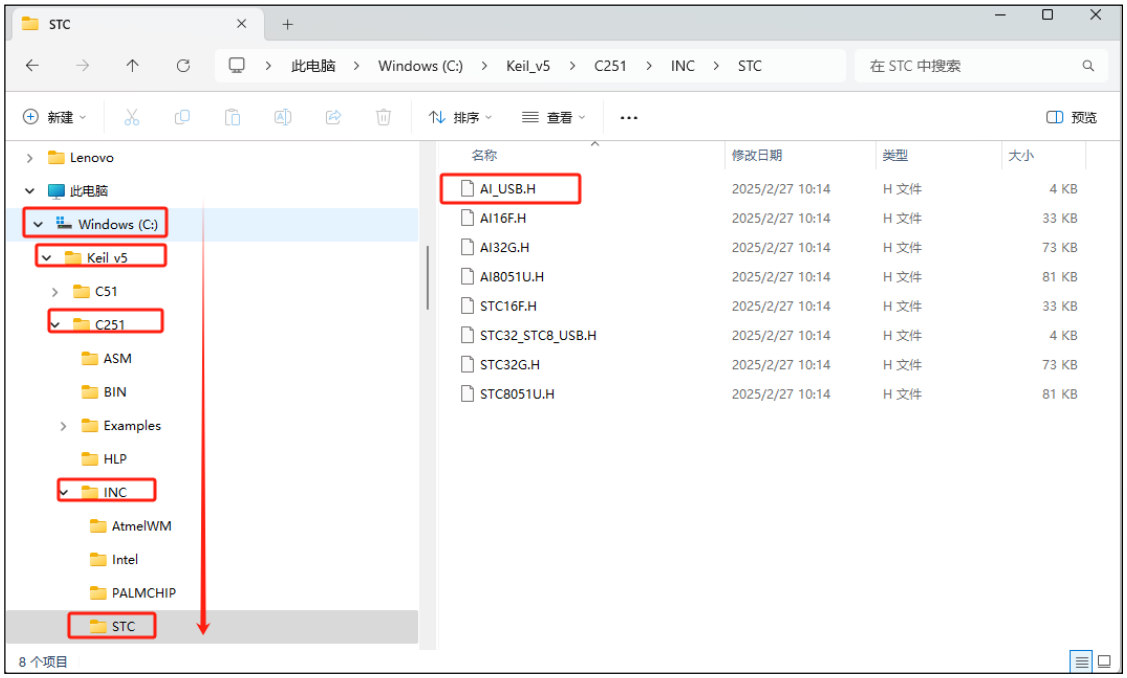


2.8.13.3 准备工作--添加头文件和库函数

1、添加常用头文件 ai_usb.h

```
#include "ai_usb.h"                    //使用 usb 所需要包含的头文件。
```

这个头文件在使用 AiCube-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



2、添加 usb_cdc 库函数文件

这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载



下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，

在’库文件\查询模式 STC-CDC 库文件’中，复制出来 **stc_usb_cdc_8h_data.LIB**

【8 位的 8051 的 USB 库 可以用 BL51 连接器

BL51： 不支持 REMOVEUNUSED；BL51 支持 扩展中断号工具；

如果 不用 REMOVEUNUSED，可以 使用 BL51 ！

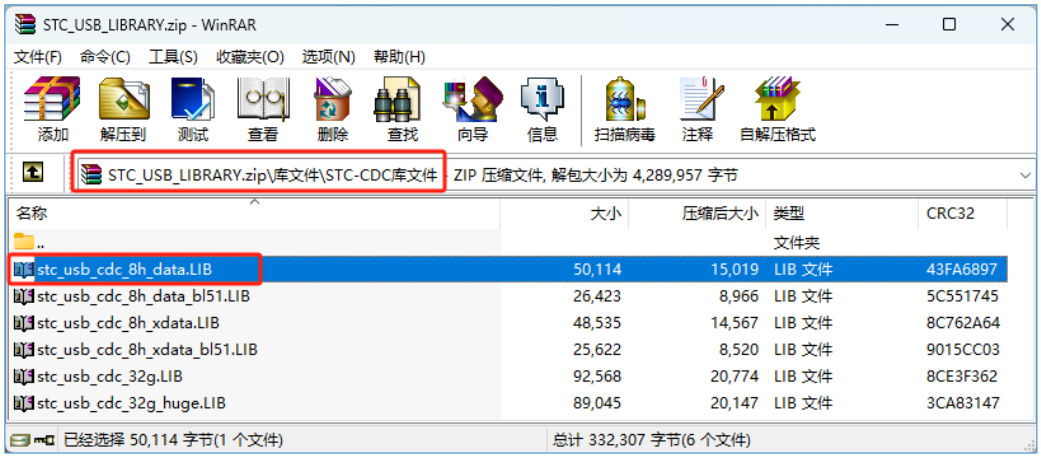
LX51： 支持 REMOVEUNUSED；LX51 不支持 扩展中断号工具；

如果 不用 扩展中断号工具，可以 使用 LX51 ！

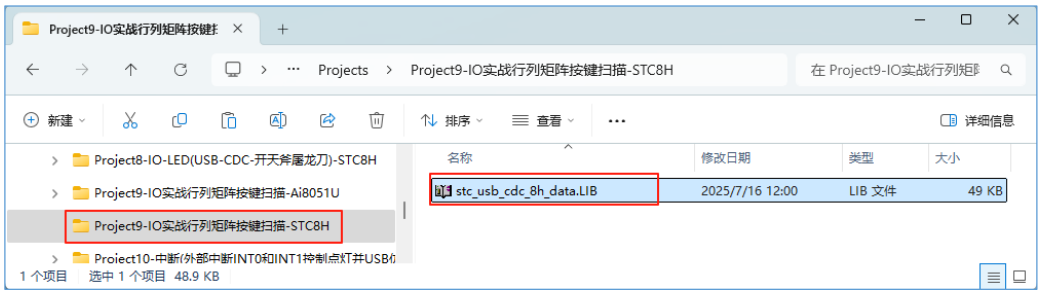
详情请参考本公司技术论坛链接：8 位的 8051 的 USB 库可以用 BL51；BL51 不支持 REMOVEUNUSED；

https://www.stcaimcu.com/thread-17105-1-1.html

(出处：国芯人工智能技术交流网站)】

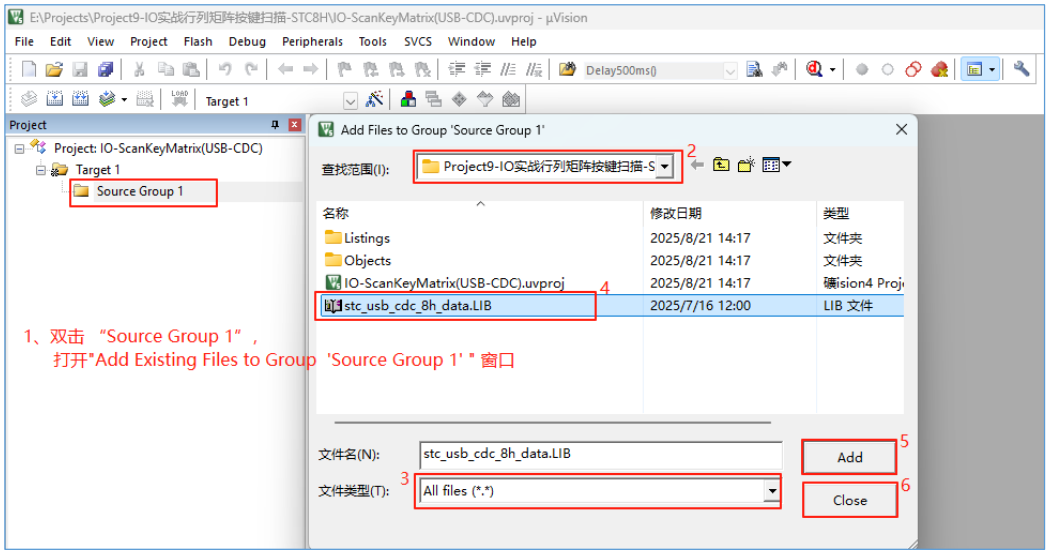


➲ **将库函数“stc_usb_cdc_8h_data.LIB”复制到创建项目所在的文件夹中**

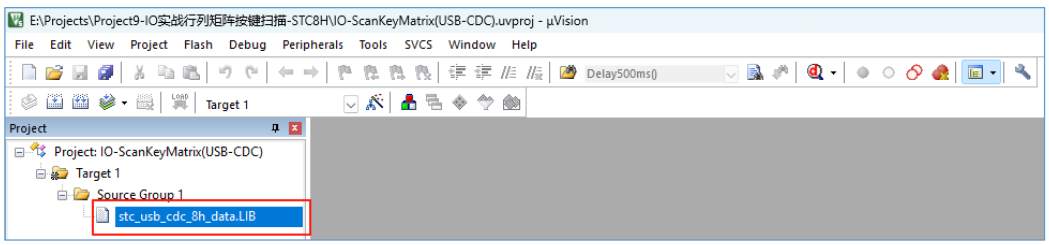


➲ **将库函数“stc_usb_cdc_8h_data.LIB”添加进入此项目重新文件中**

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 **stc_usb_cdc_8h_data.LIB** 添加进入此工程，就可以使用 USB-CDC 功能进行端口通讯。



Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

2.8.13.4 实验程序 C 代码

```
#include <STC8H.H>          //包含 STC8H 的头文件
#include "ai_usb.h"         //使用 usb 所需要包含的头文件
#include "intrins.h"        //使用_nop_()函数所必须要包含的头文件,
                            //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                            //会导致编译器找不到这个而函数而报错。

bit   keyok;                //按键按下标志
unsigned char keycode;      //定义变量,保存按键键码
unsigned char debounce;     //按键去抖动

void Delay1ms(void)         //延时 1ms@11.0592MHz
{
    unsigned char data i, j;

    i = 15;
    j = 90;
    do
    {
        while (--j);
    } while (--i);
}

void DelayXms(int n)        //延时函数，延时 x(ms)
{
    while (n--)
        Delay1ms();
}

void BootLogo(void)         //开机画面，LED60-LED67 闪烁 3 次
{
    P6 = 0x00;              //8 个 LED 全部点亮
    DelayXms(200);          //延时 200ms
    P6 = 0xff;              //8 个 LED 全部熄灭
    DelayXms(200);          //延时 200ms
    P6 = 0x00;              //8 个 LED 全部点亮
    DelayXms(200);          //延时 200ms
    P6 = 0xff;              //8 个 LED 全部熄灭
    DelayXms(200);          //延时 200ms
    P6 = 0x00;              //8 个 LED 全部点亮
    DelayXms(200);          //延时 200ms
    P6 = 0xff;              //8 个 LED 全部熄灭
    DelayXms(200);          //延时 200ms
}
```

```
void DetectTemp(void)       //检测当前温度
{
}

void ControlTemp(void)      //控制当前温度
{
}

void OtherEvent(void)       //其他事件
{
    DelayXms(10);           //假定其他事件处理时间为 10ms
}

void Beep()                 //蜂鸣器程序
{                           //实验箱上的是有源蜂鸣器,只需要输出高电平蜂鸣器就会发声
    P54 = 0;                //打开蜂鸣器控制三极管
    DelayXms(50);           //延时
    P54 = 1;                //关闭蜂鸣器控制三极管
}

void KeyEvents(void)            //执行有效按键对应的任务
{
    if (keyok)                  //判断按键按下标志
    {
        keyok =  0;             //清除标志
        Beep();                 //蜂鸣器发声
    }
}

void ScanKey(void)                  //检测按键状态
{
    unsigned char key;              //按键扫描键码

    key  = 0xff;                    //初始化扫描变量
    P0 = 0xff;                      //P0 输出高使能内部若上拉,准备读取按键状态
    P06 = 0; P07 = 1;               //准备扫描第 1 行
    NOP12();                        //等待 12 个 nop 再读取按键状态，
                                    //NOP12()这个宏已在 STC8H.H 的头文件中定义
    if (!P00) key =0;               //检查第 1 行第 1 列按键是否被按下
    else if    (!P01) key = 1;      //检查第 1 行第 2 列按键是否被按下
    else if    (!P02) key = 2;      //检查第 1 行第 3 列按键是否被按下
    else if    (!P03) key = 3;      //检查第 1 行第 4 列按键是否被按下
    else                            //如果第一行没有按键按下则扫描下一行
    {
        P07 = 0;                    //准备扫描第 2 行
        P06 = 1;
```

```
        NOP12();                        //等待 12 个 nop 再读取按键状态
                                        //NOP12()这个宏已在 STC8H.H 的头文件中定义
        if (!P00) key =4;               //检查第 2 行第 1 列按键是否被按下
        else if   (!P01) key = 5;       //检查第 2 行第 2 列按键是否被按下
        else if   (!P02) key = 6;       //检查第 2 行第 3 列按键是否被按下
        else if   (!P03) key = 7;       //检查第 2 行第 4 列按键是否被按下
    }
    P0 = 0xff;                          //恢复 P0 口状态

    if (key    != keycode)              //判断按键状态是否有变化
    {
        keycode  = key;                 //保存键码
        debounce = 5;                   //初始化去抖动变量
    }
    else
    {
        if (debounce !=     0)          //判断按键扫描是否完成
        {
            if (--debounce == 0)        //判断去抖动是否已经完成
            {
                if (keycode    != 0xff)
//判断是松开按键还是按下按键,键码为 0xff 表示是松开按键,非 0xff 为按下按键
                {
                    keyok =  1;   //设置按键按下标志

                    printf_usb("Key%d 被按下\r\n", (int)keycode);
//直接打印输出 “Key%d 被按下” 到主机相应软件的 USB【CDC/HID-串口助手】
                }
            }
        }
    }
}

void main(void)
{
    P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 =  0x00;      //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 =  0x00;      //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 =  0x00;      //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 =  0x00;      //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 =  0x00;      //设置 P4 口为准双向口模式
    P5M0 = 0x10; P5M1 =  0x00;      //设置 P5.4  口为强推挽，其他设置为准双向口模式
    P6M0 = 0x00; P6M1 =  0x00;      //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 =  0x00;      //设置 P7 口为准双向口模式
```

```
    keycode   = 0xff;                   //初始化键码(0~7 为有效键码,0xff 为无效键码)
    keyok =   0;                        //初始化按键标志

    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */
    EA = 1;                             //总中断允许位打开

    P54  = 1;                           //关闭蜂鸣器
    P40  = 0;                           //打开 LED 部分的供电

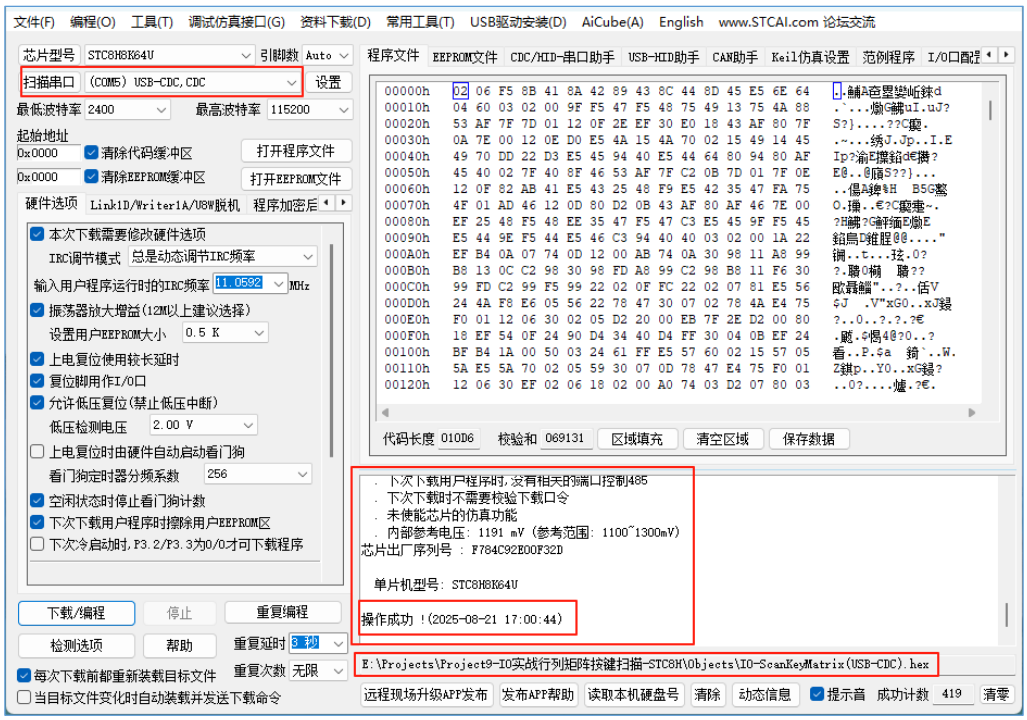
    BootLogo();                         //显示开机画面，  8 个 LED 同时闪烁 3 次

    while(1)
    {                                   //假定整个主循环的执行时间为 10ms 左右
        ScanKey();                      //按键扫描
        KeyEvents();                    //按键事件处理

        DetectTemp();                   //温度检查任务
        ControlTemp();                  //温度控制任务
        OtherEvent();                   //其他事件
    }
}
```

2.8.13.5 程序下载

录入代码，保存，编译。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤“打开程序文件”：IO-ScanKeyMatrix(USB-CDC).hex（“打开程序文件”是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中）并“下载/编程”，如下图：



这时在【AiCube-ISP】界面，我们看到串口选项框中已经出现了一个(COM5)USB-CDC,CDC 串口，表明现在可以通过这个 USB-CDC 端口进行数据通讯了。

2.8.13.6　　程序实现效果，观察实验现象

程序编写好以后，编译无误后下载程序到 STC8H8K64U 实验箱。

✓ **实验箱上 8 个 LED 灯立即就会同时闪烁 3 次。**

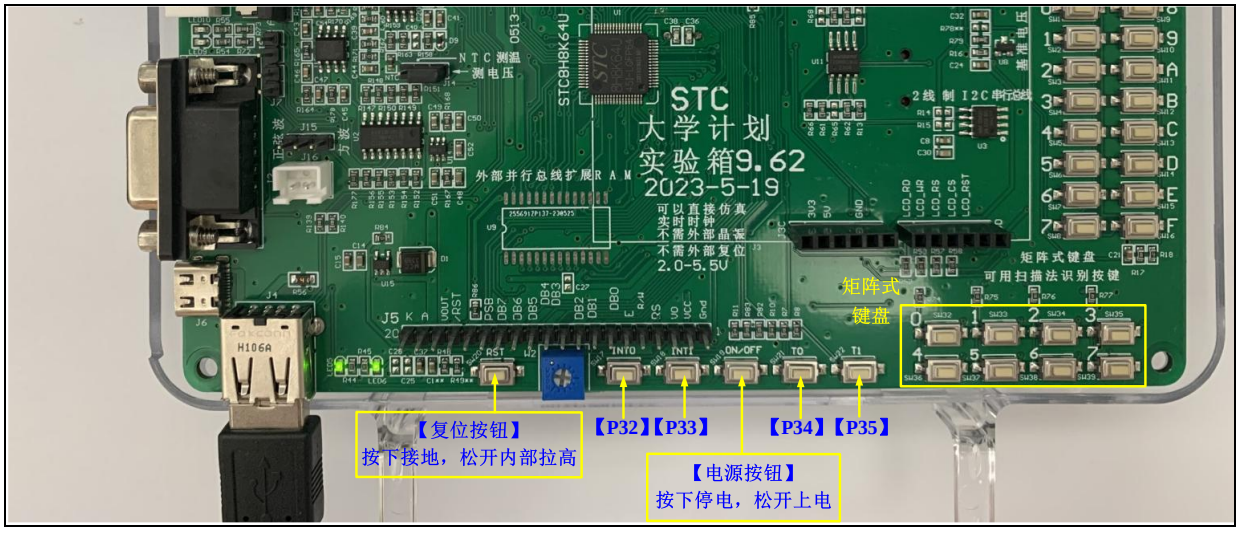
打开【AiCube-ISP】界面的“CDC/HID-串口助手”选项卡，

➲ 接收缓冲区：勾选“文本模式”

➲ 串口：选择与“扫描串口”窗口显示的 USB-CDC 端口一致

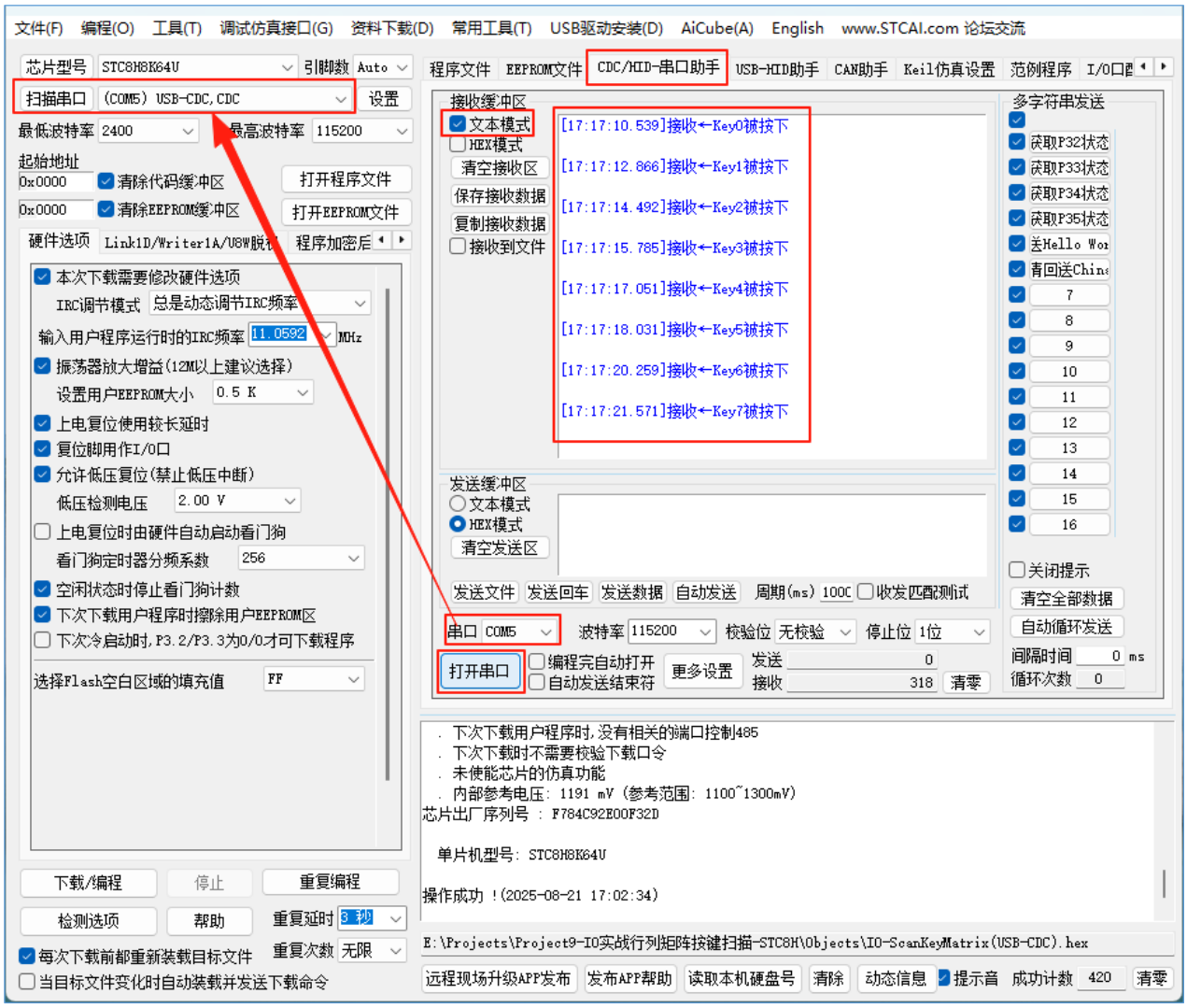
➲ 点击“打开串口”按钮

➲ 分别按下矩阵式键盘的 0-7 个按键（如下图）后



✓ **蜂鸣器会发出声音**

✓ **AiCube-ISP 软件“CDC/HID-串口助手”选项卡的接收缓冲区显示如下：**


文件(F) 编程(O) 工具(T) 调试仿真接口(G) 资料下载(D) 常用工具(T) USB驱动安装(D) AiCube(A) English www.STCAI.com 论坛交流
芯片型号 STC8H8K64U 引脚数 Auto
扫描串口 (COM5) USB-CDC,CDC 设置
最低波特率 2400 最高波特率 115200
起始地址
0x0000 清除代码缓冲区 打开程序文件
0x0000 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 Link1D/Writer1A/U8W脱机 程序加密后
本次下载需要修改硬件选项
IRC调节模式 总是动态调节IRC频率
输入用户程序运行时的IRC频率 11.0592 MHz
振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 2.00 V
上电复位时由硬件自动启动看门狗
看门狗定时器分频系数 256
空闲状态时停止看门狗计数
下次下载用户程序时擦除用户EEPROM区
下次冷启动时,P3.2/P3.3为0/0才可下载程序
选择Flash空白区域的填充值 FF
下载/编程 停止 重复编程
检测选项 帮助 重复延时 3 秒
每次下载前都重新装载目标文件 重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件 EEPROM文件 CDC/HID-串口助手 USB-HID助手 CAN助手 Keil仿真设置 范例程序 I/O口
接收缓冲区
文本模式
HEX模式
清空接收区
保存接收数据
复制接收数据
接收到文件
[17:17:10.539]接收←Key0被按下
[17:17:12.866]接收←Key1被按下
[17:17:14.492]接收←Key2被按下
[17:17:15.785]接收←Key3被按下
[17:17:17.051]接收←Key4被按下
[17:17:18.031]接收←Key5被按下
[17:17:20.259]接收←Key6被按下
[17:17:21.571]接收←Key7被按下
多字符串发送
获取P32状态
获取P33状态
获取P34状态
获取P35状态
关Hello Wor
青回送Chins
7
8
9
10
11
12
13
14
15
16
关闭提示
清空全部数据
自动循环发送
发送缓冲区
文本模式
HEX模式
清空发送区
发送文件 发送回车 发送数据 自动发送 周期(ms) 100 收发匹配测试
串口 COM5 波特率 115200 校验位 无校验 停止位 1位
打开串口 编程完自动打开 自动发送结束符 更多设置
发送 0
接收 318 清零
间隔时间 0 ms
循环次数 0
. 下次下载用户程序时,没有相关的端口控制485
. 下次下载时不需要校验下载口令
. 未使能芯片的仿真功能
. 内部参考电压: 1191 mV (参考范围: 1100~1300mV)
芯片出厂序列号 : F784C92E00F32D
单片机型号: STC8H8K64U
操作成功 !(2025-08-21 17:02:34)
E:\Projects\Project9-IO实战行列矩阵按键扫描-STC8H\Objects\IO-ScanKeyMatrix(USB-CDC).hex
远程现场升级APP发布 发布APP帮助 读取本机硬盘号 清除 动态信息 提示音 成功计数 420 清零

完成实验。

2.9 传统外部中断实验 -- INT0，INT1，INT2，INT3

2.9.1 中断实验 1(外部中断 INT0/P3.2，INT1/P3.3 控制点灯并 USB 仿真)

1、 了解传统的外部中断 INT0/P3.2，INT1/P3.3 的使用
2、 了解 STC8H8K64U 对传统外部中断 INT0/P3.2，INT1/P3.3 功能的扩展
3、 学习使用 USB 进行仿真
4、 学习使用在中断中设置标志位，在主循环中查询并处理中断事件的处理方式

2.9.1.1 实验介绍

1，边缘触发中断，INT0/P3.2 外部中断实验演示程序：

P3.2 设置为高阻输入
或 P3.2 设置为准双向口
INT0/P3.2 设置为边缘触发中断。

【实验步骤】:

- 先点击 Run（运行）按钮，按下 P3.2 按键，保持不松开（为了验证下一步的实验），
  进 INT0 中断后，读到 INT0/P3.2 口是 低，下降沿，点亮 LED6.7，退出中断服务程序。
- 点击 Run（运行）按钮，松开 P3.2 按键
  进 INT0 中断后，读到 INT0/P3.2 口是 高，上升沿，点亮 LED6.0，退出中断服务程序。
- 利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪，是下降沿，还是上升沿

2，下降沿触发中断，INT1/P3.3 外部中断实验演示程序：

P3.3 设置为高阻输入
或 P3.3 设置为准双向口
INT1/P3.3 设置为下降沿触发中断。

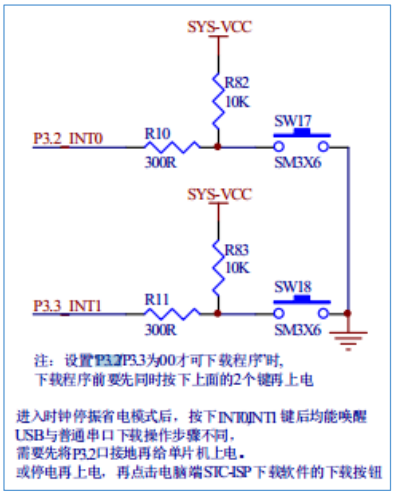
【实验步骤】:

- 先点击 Run（运行）按钮，按一下 P3.3 按键
  进 INT1 中断后，点亮 LED6.4，退出中断服务程序。
- 利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪。

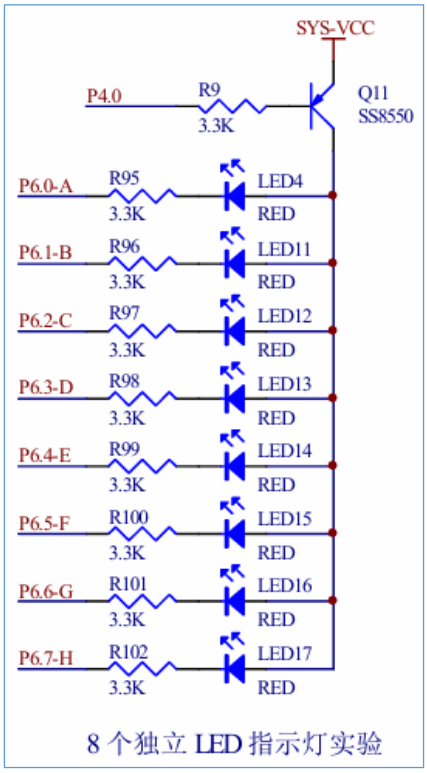
2.9.1.2 原理图

- **外部中断【INT0/P3.2，INT1/P3.3】输入原理图@STC8H8K64U 实验箱**

  将[INT0/P3.2，INT1/P3.3]设置为：高阻输入
  或 准双向口
  或 开漏模式
  [INT0/P3.2，INT1/P3.3]设置为边缘触发中断。



➢ 外部中断实验辅助显示 LED 灯原理图



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。
如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。
如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。
如果 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。
如 P6.0 输出 0，LED60 灯就会被点亮；如 P6.0 输出 1，LED60 灯就会熄灭；
如 P6.1 输出 0，LED61 灯就会被点亮；如 P6.1 输出 1，LED61 灯就会熄灭；
如 P6.2 输出 0，LED62 灯就会被点亮；如 P6.2 输出 1，LED62 灯就会熄灭；
如 P6.3 输出 0，LED63 灯就会被点亮；如 P6.3 输出 1，LED63 灯就会熄灭；
如 P6.4 输出 0，LED64 灯就会被点亮；如 P6.4 输出 1，LED64 灯就会熄灭；
如 P6.5 输出 0，LED65 灯就会被点亮；如 P6.5 输出 1，LED65 灯就会熄灭；
如 P6.6 输出 0，LED66 灯就会被点亮；如 P6.6 输出 1，LED66 灯就会熄灭；
如 P6.7 输出 0，LED67 灯就会被点亮；如 P6.7 输出 1，LED67 灯就会熄灭

如 P6 口对外全部送 0，则这 8 个灯全亮；
如 P6 口对外全部送 1，则这 8 个灯全灭
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P6 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

2.9.1.3　实验程序 C 语言代码

```
#include <STC8H.H>          //包含 STC8H 的头文件
#include "intrins.h"        //使用_nop_()函数所必须要包含的头文件,
                            //否则延时函数中调用的_nop_()函数没有被头文件引用过来,
                            //会导致编译器找不到这个而函数而报错。
```

```
bit   int0_flag = 0; //定义 1 个位变量，INT0 事件位变量标志，记录 INT0 已产生中断
// 供主循环查询 INT0 是否已产生中断，在主循环中处理 INT0 的中断事件任务，不堵塞其他中断
bit   int1_flag = 0; //定义 1 个位变量，INT1 事件位变量标志，记录 INT1 已产生中断
// 供主循环查询 INT1 是否已产生中断，在主循环中处理 INT1 的中断事件任务，不堵塞其他中断

void main (void)
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
//  P3M0 = 0x00; P3M1 = 0x00;       //P32、P33 设置为准双向口
    P3M0 = 0x00; P3M1 = 0xff;       //P32、P33 设置为高阻输入(需要同步开启上拉电阻)
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;

    P3PU = 0x0c;                    //P32、P33 打开上拉电阻

    IT0 = 0;                        //使能 INT0 上升沿和下降沿中断
//  IT0 = 1;                        //使能 INT0 下降沿中断
    EX0 = 1;                        //使能 INT0 中断

//  IT1 = 0;                        //使能 INT1 上升沿和下降沿中断
    IT1 = 1;                        //使能 INT1 下降沿中断
    EX1 = 1;                        //使能 INT1 中断

    IE0 = 0;                        //清 INT0 中断标志
    IE1 = 0;                        //清 INT1 中断标志

    EA = 1;                         //总中断允许位打开
    P40 = 0;                        //打开 LED 灯供电

    int0_flag = 0;                  //初始化用户标志位
    int1_flag = 0;                  //初始化用户标志位

    while(1)
    {
        if(int0_flag)   //主循环中查询，INT0 是否已产生中断，是否有需要处理的 INT 0 事件
        {
            int0_flag = 0;          //清 0，INT0 事件位变量标志
            _nop_();                //用户在此添加需要处理的事件
            _nop_();
        }
```

```
        if(int1_flag)   //主循环中查询，INT1 是否已产生中断，是否有需要处理的 INT1 事件
        {
            int1_flag = 0;      //清 0，INT1 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
    }
}

void int0_isr(void) interrupt INT0_VECTOR
{
    int0_flag = 1;  // int0_flag 置 1 是通知主循环处理部分 INT0 中断事件不需要特急处理的任务
                    //置 1，记录 INT0 已产生中断，供主循环查询判断有无需处理的 INT0 任务

    if(INT0)                //边沿中断，进入后再次判断电平从而判断是什么样的电平
    {
        P60 = 0;            //判断为高电平，则当前为上升沿，点亮 P60 端口上的 LED 灯
        _nop_();            //可以在这里插入断点进行观察现象
        P60 = 1;            //关闭 LED 灯
    }
    else
    {
        P67 = 0;            //判断为低电平，则当前为下降沿，点亮 P77 端口上的 LED 灯
        _nop_();            //可以在这里插入断点进行观察现象
        P67 = 1;            //关闭 LED 灯
    }
}

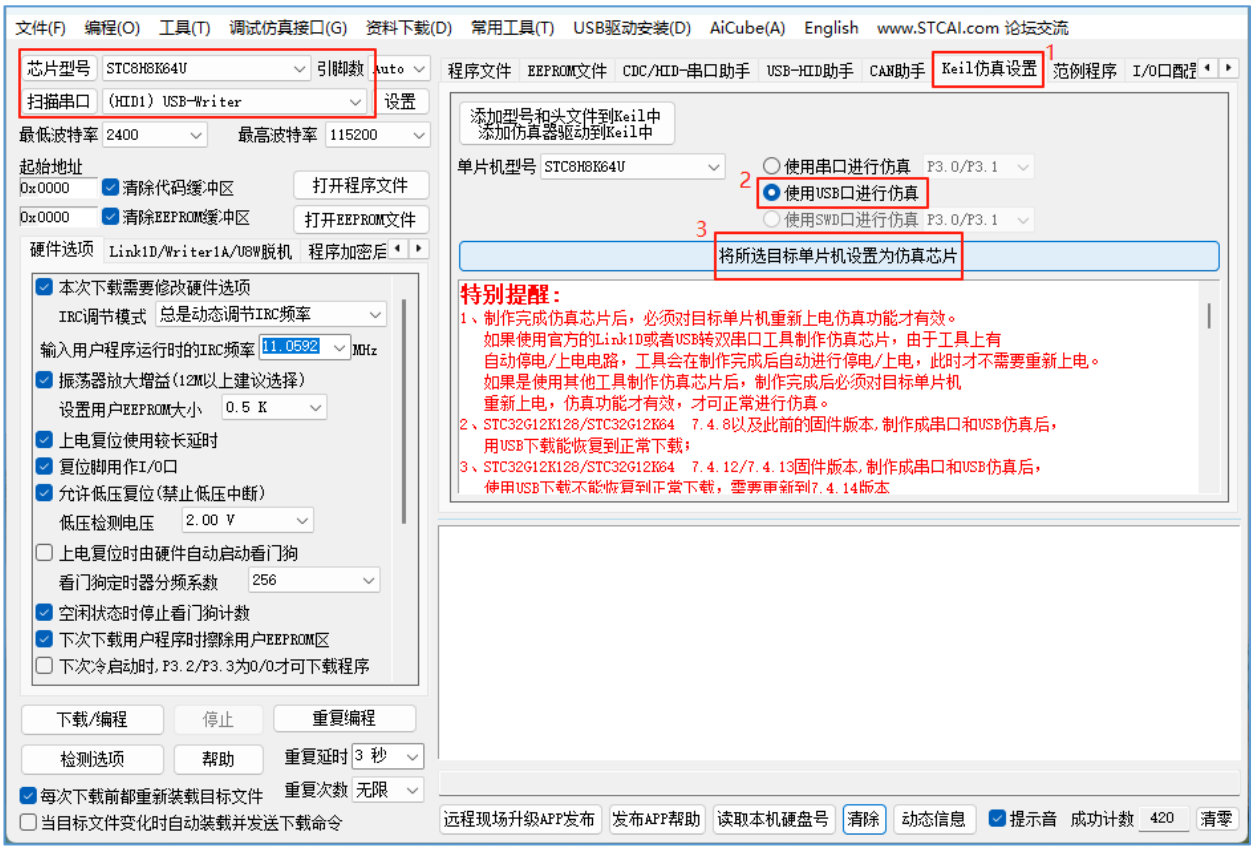
void int1_isr(void) interrupt INT1_VECTOR
{
    int1_flag = 1;  // int1_flag 置 1 是通知主循环处理部分 INT1 中断事件不需要特急处理的任务
                    //置 1，记录 INT1 已产生中断，供主循环查询判断有无需处理的 INT1 任务

    P64 = 0;                //点亮 P64 端口上的 LED 灯
    _nop_();                //可以在这里插入断点观察现象
    P64 = 1;                //关闭 LED 灯
}
```

2.9.1.4　制作仿真芯片

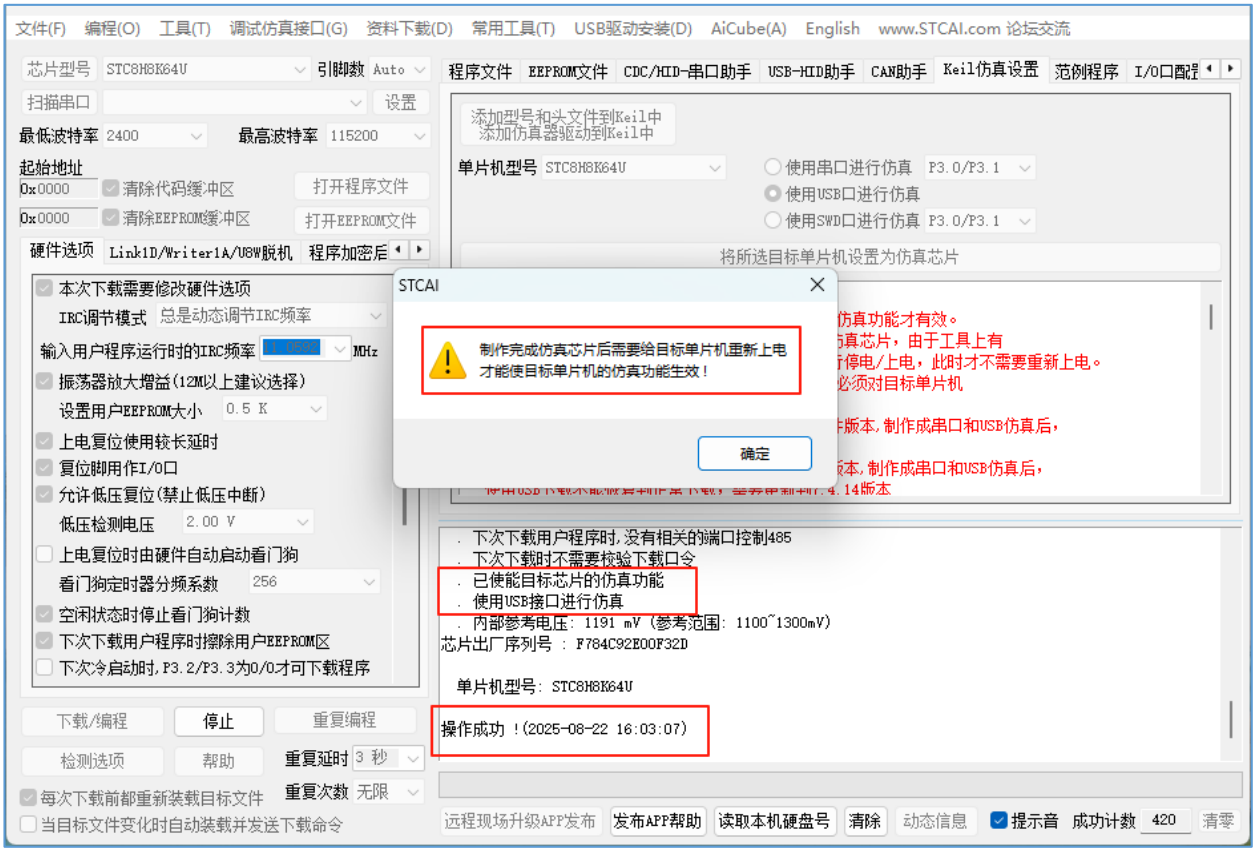
在 AiCube-ISP 软件界面，看到扫描串口列表框显示“(HID1) USB Writer”：

- 点击“Keil 仿真设置”标签；
- 在“Keil 仿真设置”选项卡，选择“使用 USB 口进行仿真”
- 点击“将所选目标单片机设置为仿真芯片”

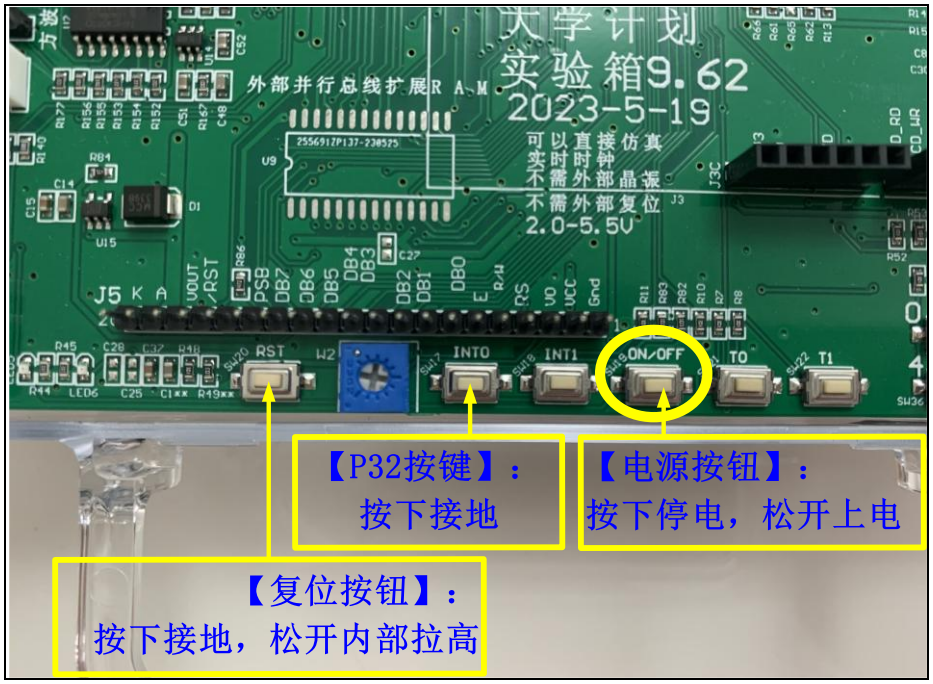


然后看到 AiCube-ISP 软件在自动进行仿真芯片的设置，界面下面的进度条在推进，然后“操作成功!”

界面出现特别提示：“制作完成仿真芯片后需要给目标单片机重新上电才能使目标单片机的仿真生效!”如下图：



所以，按提示按一下实验箱上的“Power_SW”按钮：



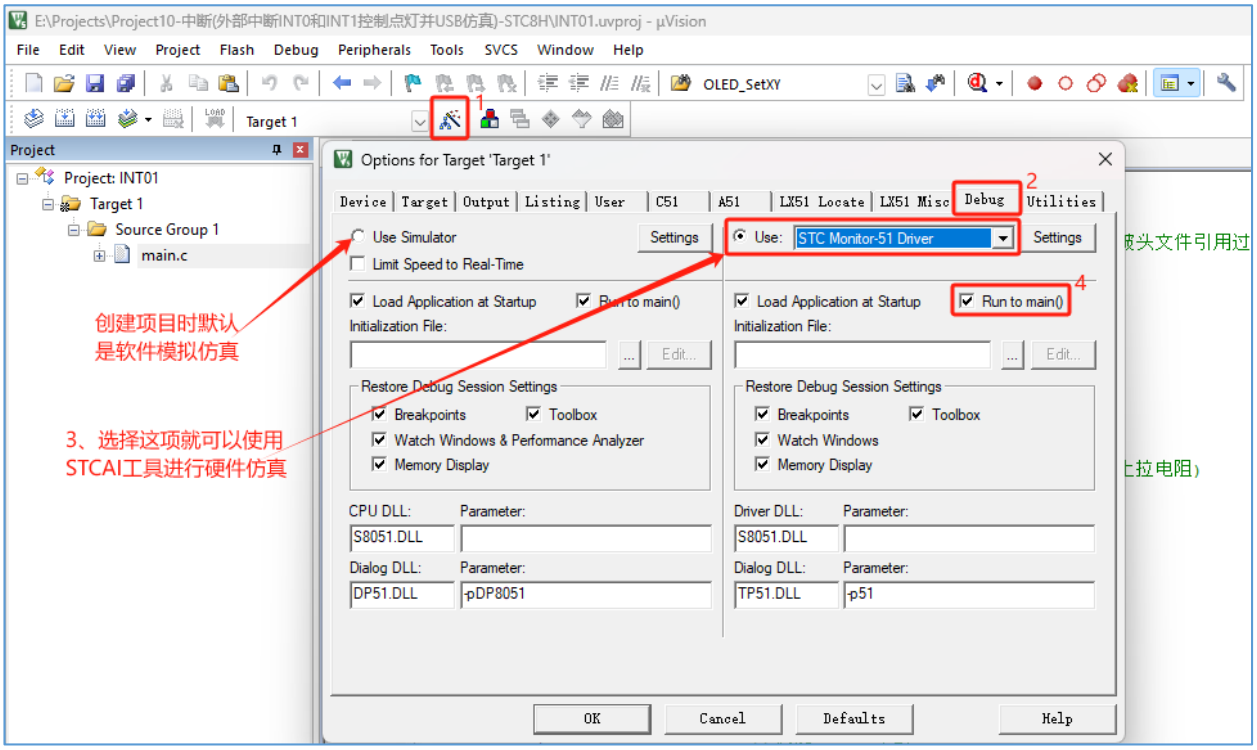
2.9.1.5　仿真设置，在 Keil 中

本项目是使用 STC8H8K64U 内置 USB 直接仿真，除了新建项目时必要设置的基础选项外，还要进行仿真设置：“Debug”选项卡和“Utilities”选项卡。

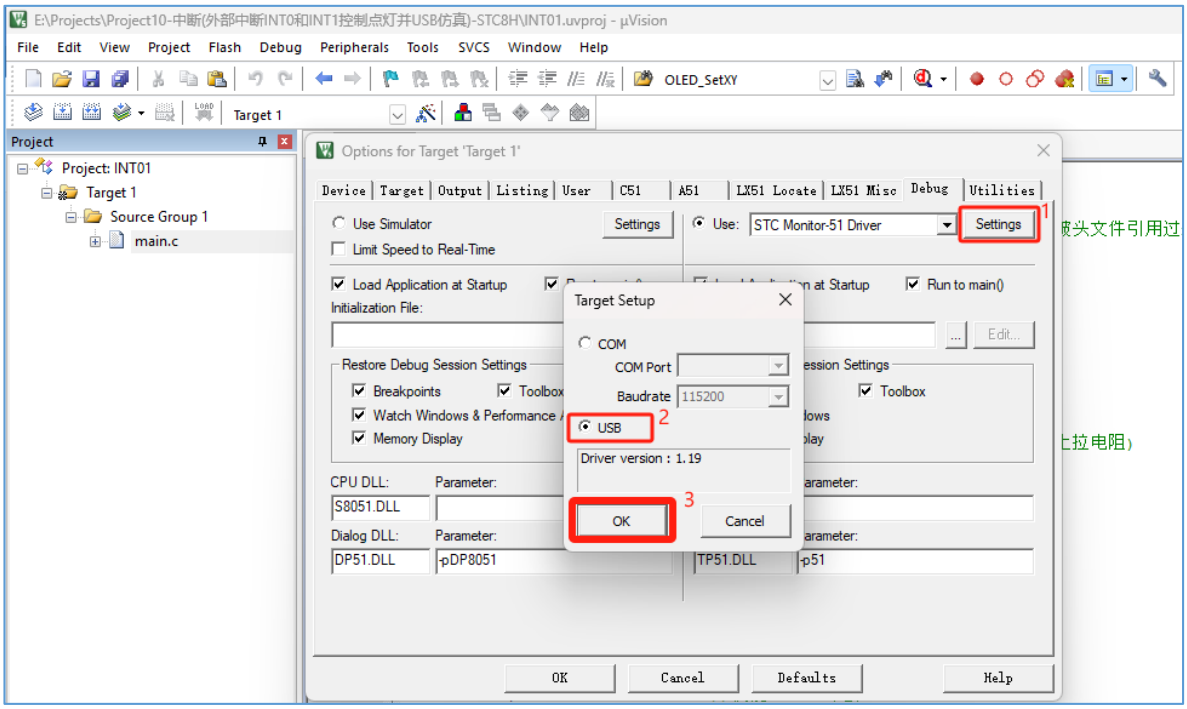
点击工具栏上的魔术棒的工具，打开“Options for Target ‘Target 1’”任务选项面板：

2.9.1.5.1　设置 Debug 选项卡

- **选择硬件仿真：**系统默认是软件模拟仿真，本项目是使用 STC8H8K64U 内置 USB 直接仿真，属于硬件仿真，要选择“STC Monitor-251 Driver”

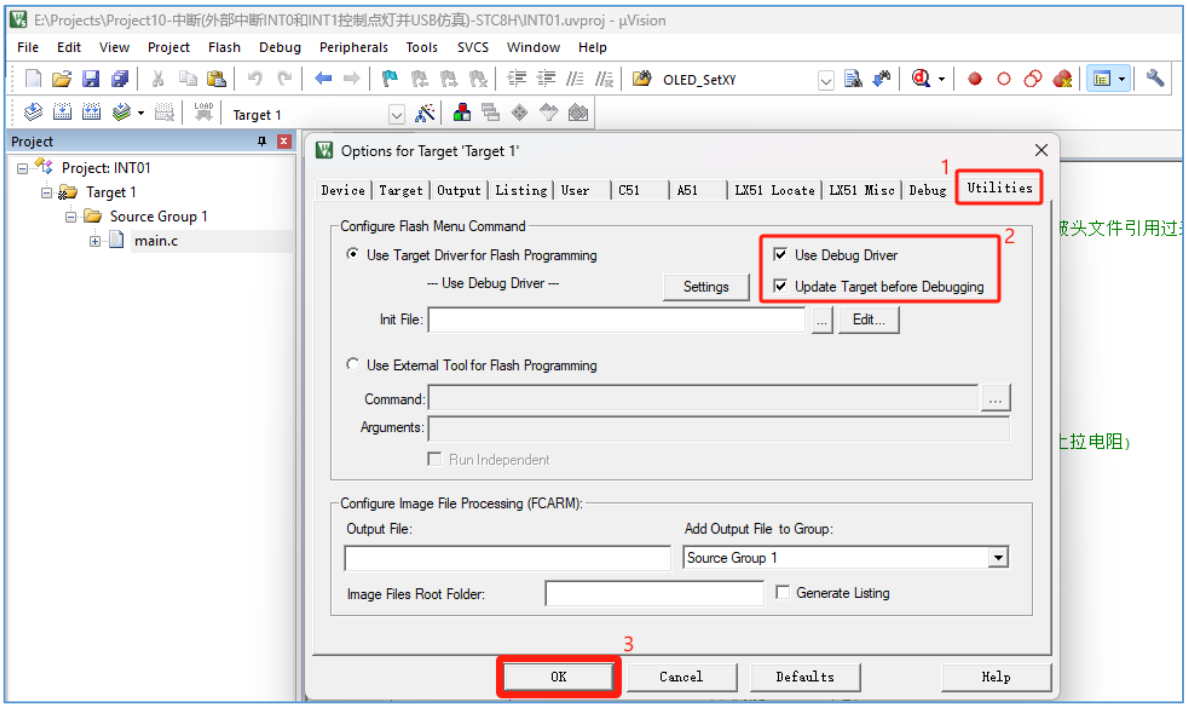


➲ **选择仿真接口**：STC8H8K64U 芯片内置的硬件 USB 支持直接仿真，本项目要选择 USB 为仿真接口。



2.9.1.5.2　设置 Utilities 选项卡

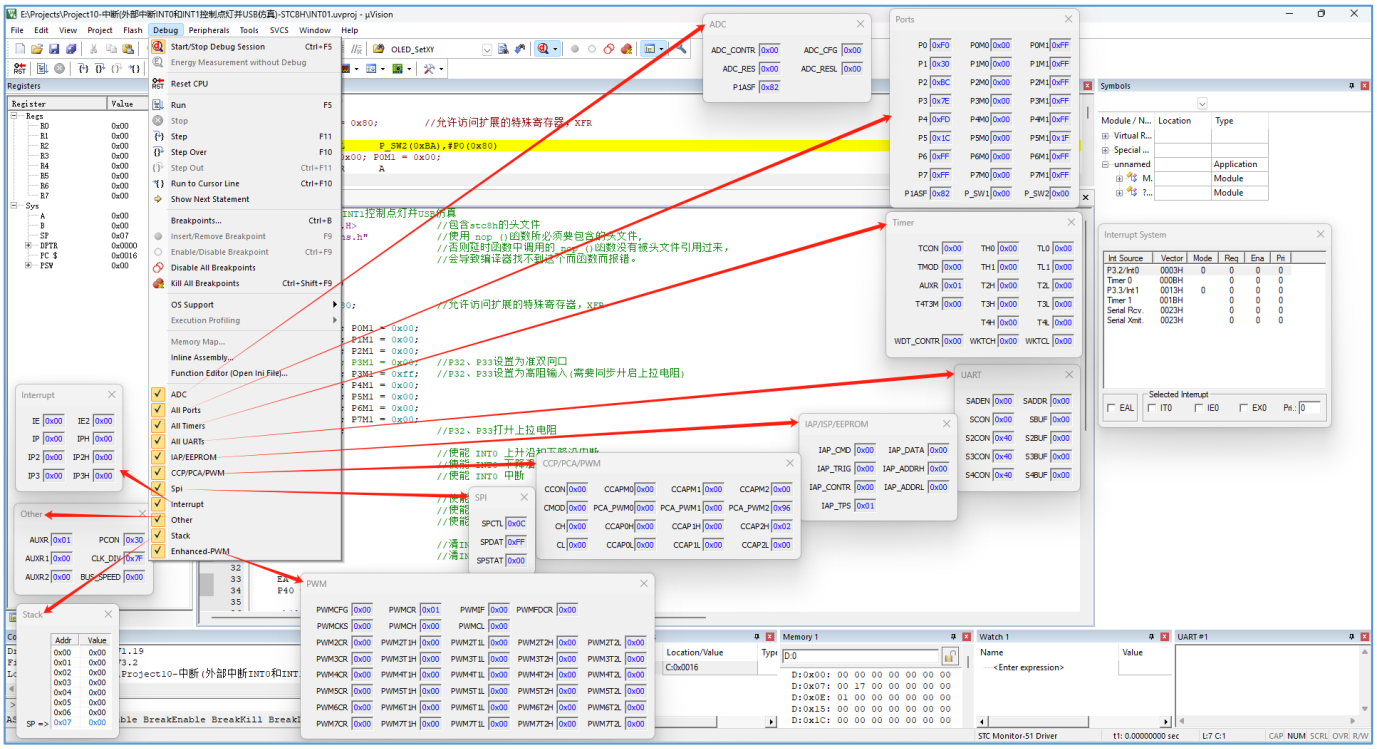
切换到“Utilities”选项卡，勾选“Use Debug Driver”，点击“OK”，完成设置。



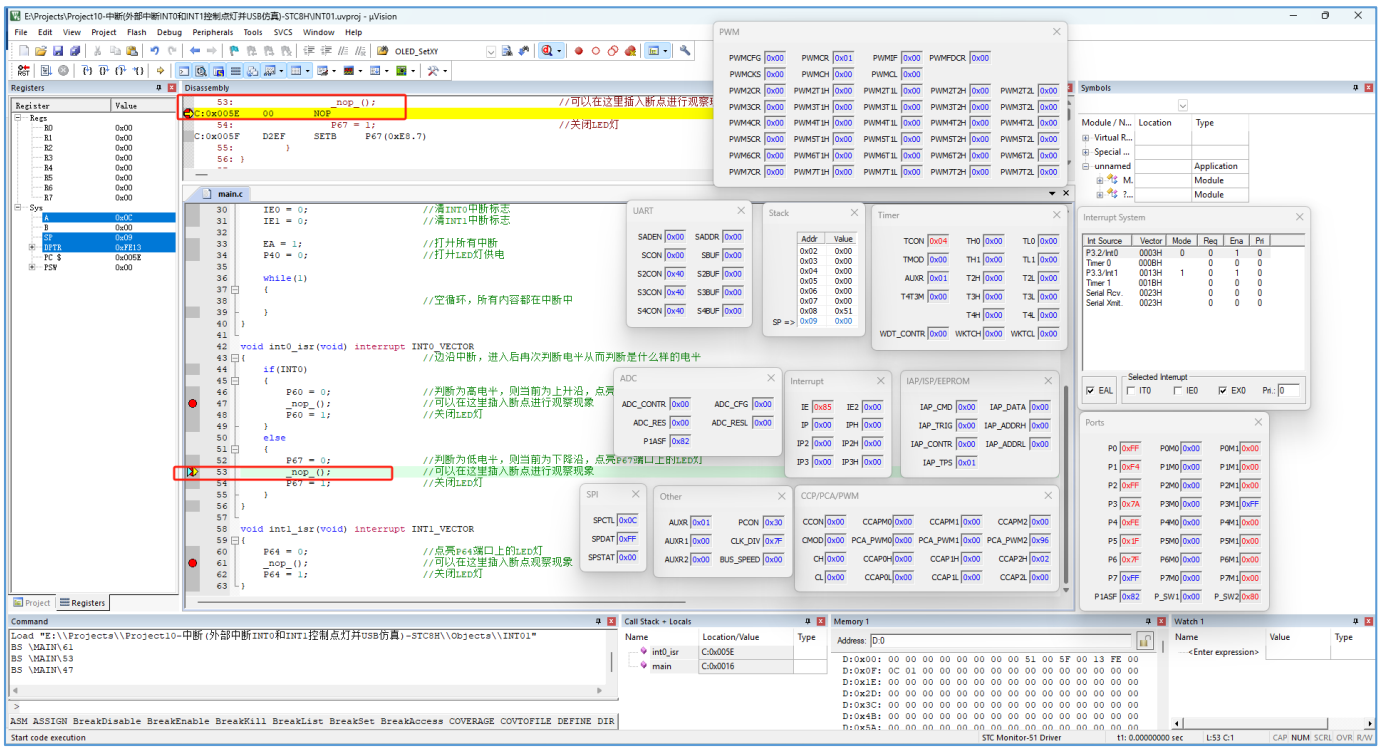
2.9.1.6　程序实现效果，观察实验现象

步骤一：利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪，INT0/P3.2 是下降沿还是上升沿

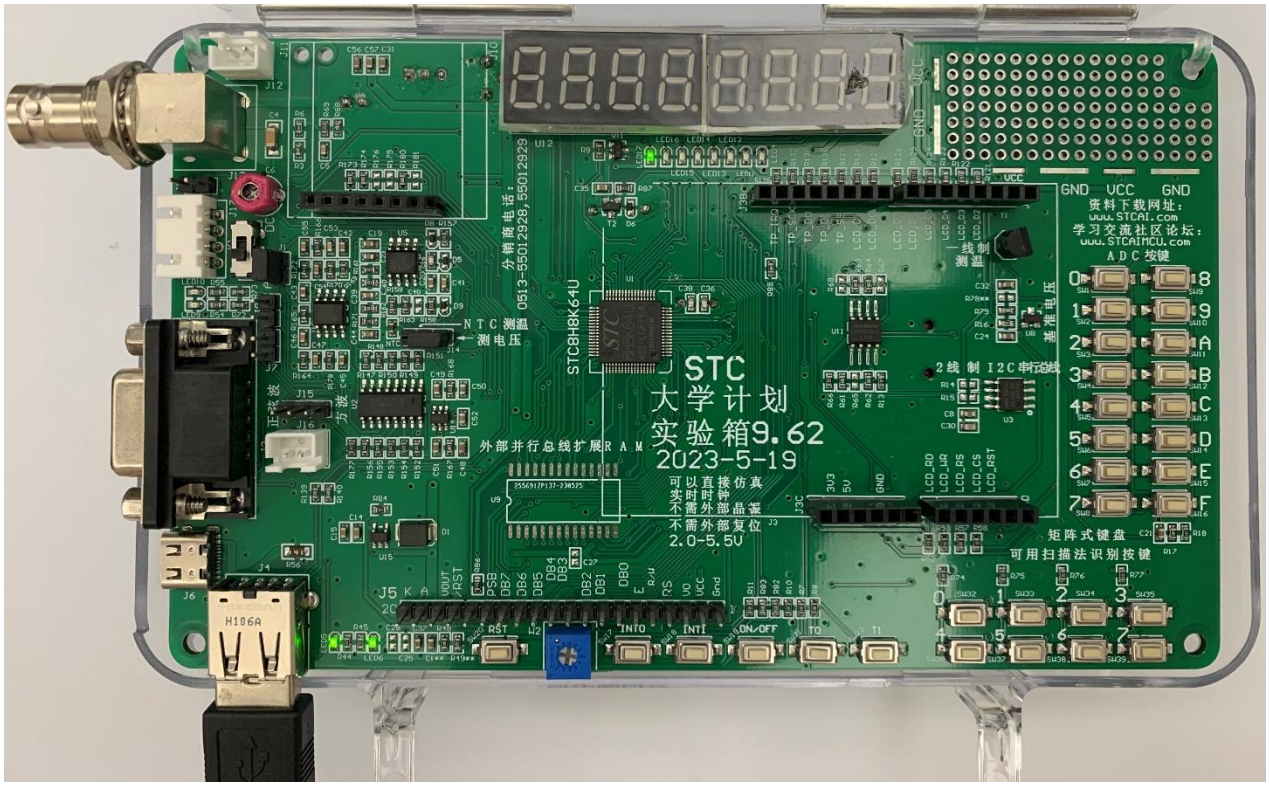
程序代码编译无误后，点击工具栏的 Debug（启动/关闭编译调试操作界面）按钮后，Keil 软件进入仿真界面。为方便观察实验现象，需在中断服务程序中设置如下 3 个断点。当然，你也可以打开全部或部分外设窗口观察各外设数据在 Debug 过程中的变化：



先点击 Run（运行）按钮，再按下 P3.2 按键，保持不松开（为了验证下一步的实验），程序会停在如下图所示断点处。同时，相较于上图，我们观察到：Ports / Timer / Interrupt / Interrupt System / Call Stack /Memory1 / Registers / Disassembly 这些窗口显示的数据都有可见的变化。

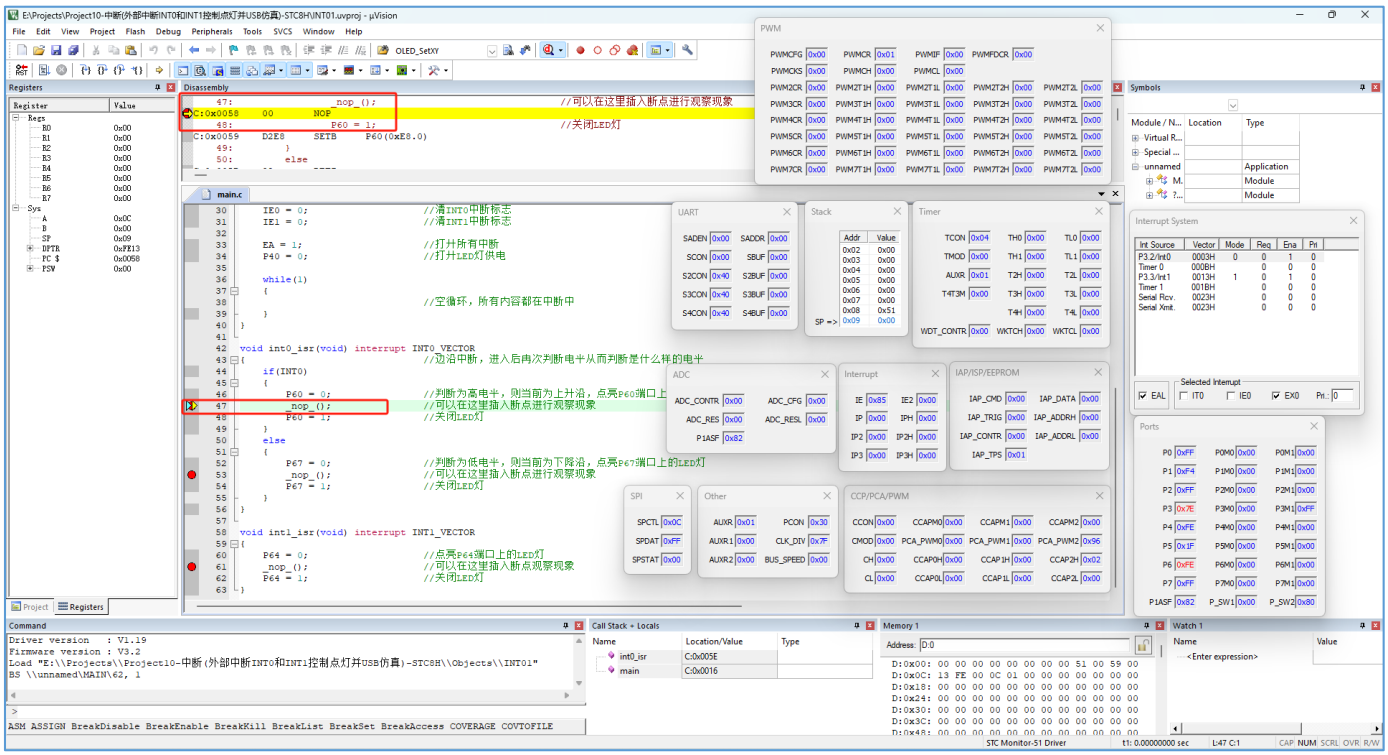


进 INT0 中断后，读到 INT0/P3.2 口是 低，下降沿，点亮 LED6.7，STC8H8K64U 实验箱如下图：

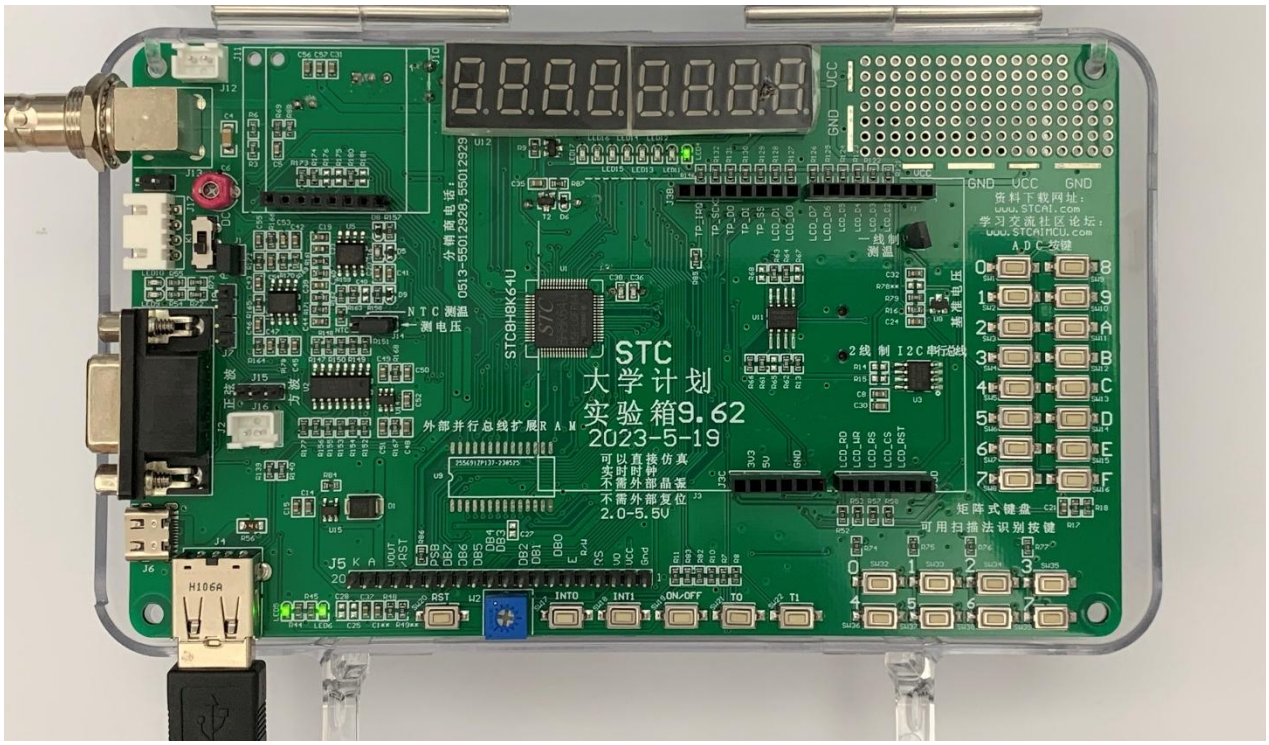


继续点击 Run（运行）按钮运行，退出中断服务程序。LED6.7 熄灭。

再松开 P3.2 按键，程序会停在如下图所示断点处。同时，相较于上图，我们观察到：Ports /Memory1 / Registers / Disassembly 这些窗口显示的数据都有可见的变化。



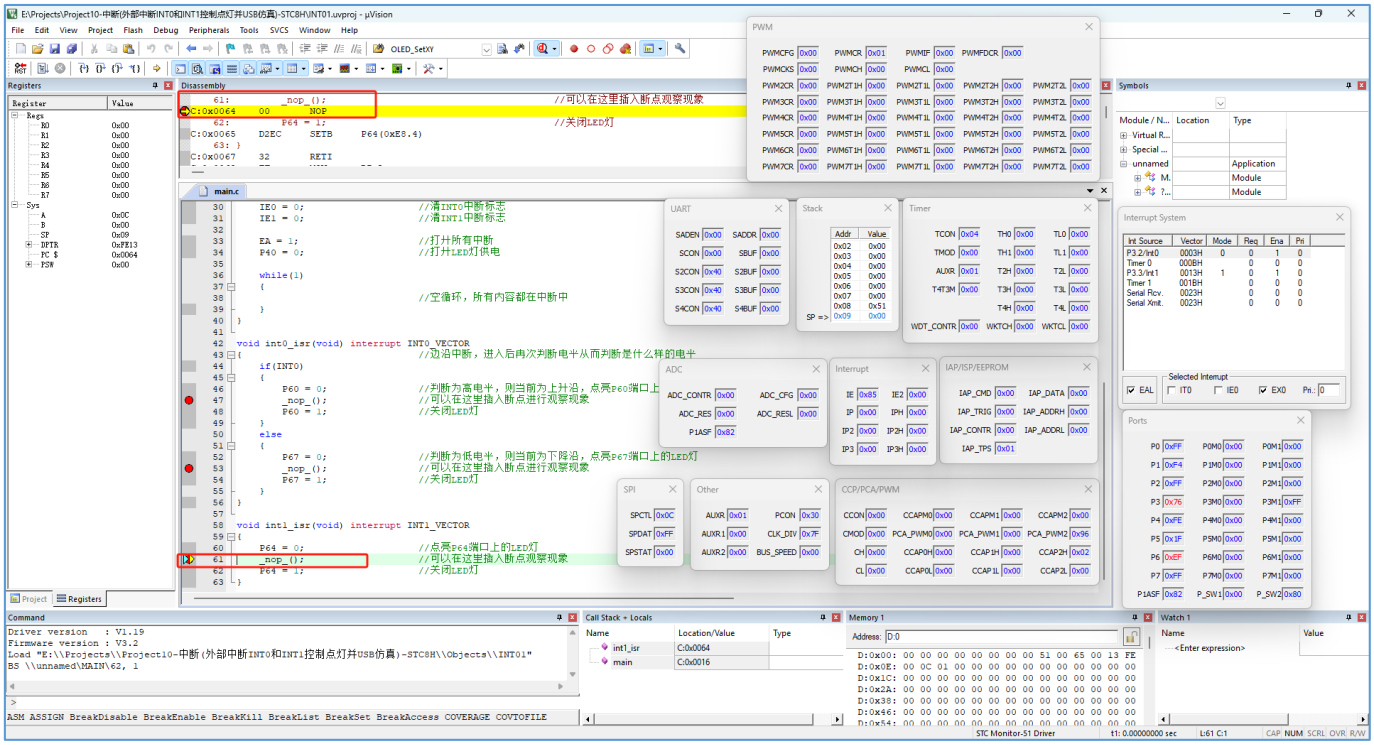
进 INT0 中断后，读到 INT0/P3.2 口是 高，上升沿，点亮 LED6.0，STC8H8K64U 实验箱如下图：



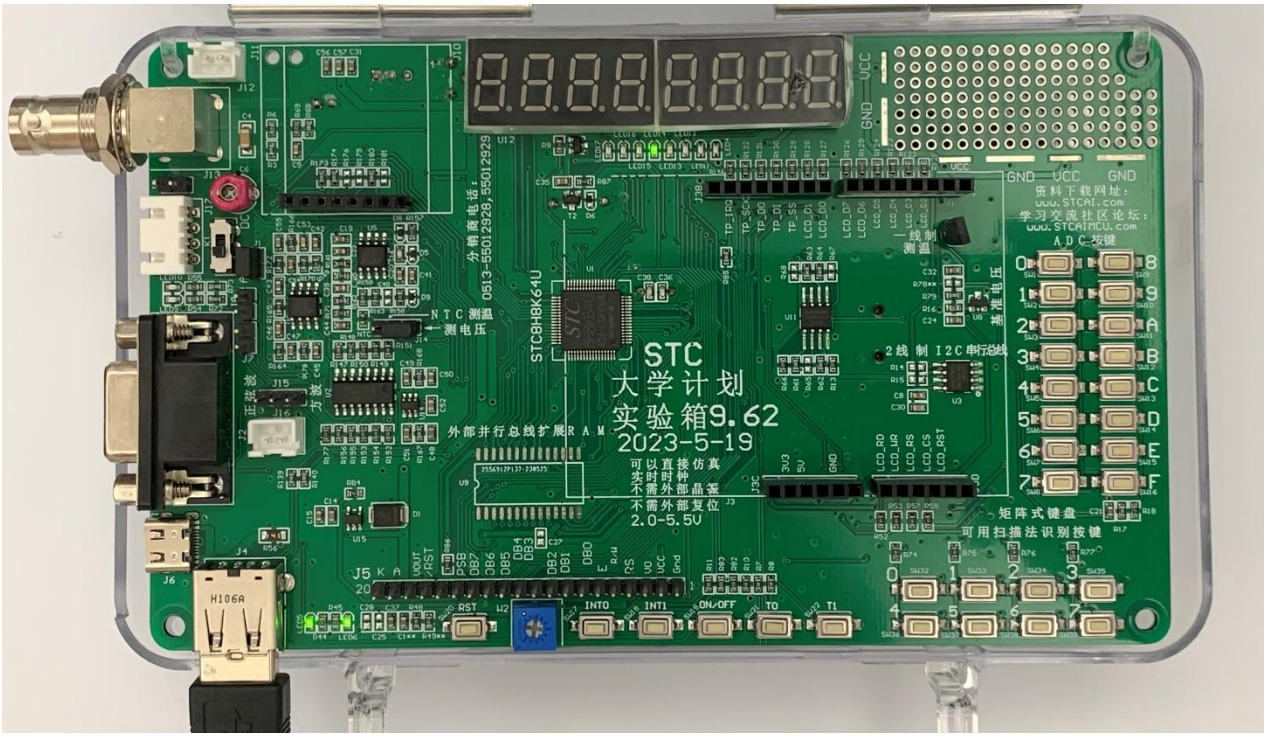
继续点击 Run（运行）按钮运行，退出中断服务程序。LED6.0 熄灭。

步骤二：利用仿真，可以设置运行到 INT1/P3.3 外部中断的断点处

如果程序没有在运行，先点击 Run（运行）按钮，再按一下 P3.3 按键，程序会停在如下图所示断点处。同时，相较于上图，我们观察到：Ports / Call Stack /Memory1 / Registers / Disassembly 这些窗口显示的数据都有可见的变化。



进 INT1 中断后，点亮 LED6.4，STC8H8K64U 实验箱如下图：



继续点击 Run（运行）按钮运行，退出中断服务程序。LED6.4 熄灭。
完成本实验。

2.9.1.7　　实验程序汇编语言代码

```
;外部中断 INT0 和 INT1 控制点灯并 USB 仿真
$INCLUDE(STC8H.INC)          ;包含 STC8H 的头文件
;-----------------------------------------------
    ORG     0000H            ;程序复位地址
    LJMP    START            ;跳转到复位地址开始运行程序

    ORG     0003H            ;INT0 中断入口地址
    LJMP    INT0_ISR         ;跳转到 INT0 中断服务程序

    ORG     0013H            ;INT1 中断入口地址
    LJMP    INT1_ISR         ;跳转到 INT1 中断服务程序
;-----------------------------------------------
    ORG     0100H

INT0_ISR:                    ;INT0 中断服务程序
    JNB     INT0, P32ISLOW   ;边沿中断，进入后再次判断电平从而判断是什么样的电平

P32ISHIGH:
    CLR     P6.0             ;判断为高电平，则当前为上升沿，点亮 P60 端口上的 LED 灯
    NOP                      ;可以在这里插入断点进行观察现象
    SETB    P6.0             ;关闭 LED 灯
    SJMP    INT0_ISR_EXIT    ;跳转到 INT0 中断程序返回的地方
```

```
P32ISLOW:
    CLR     P6.7            ;判断为低电平，则当前为下降沿，点亮 P67 端口上的 LED 灯
    NOP                     ;可以在这里插入断点进行观察现象
    SETB    P6.7            ;关闭 LED 灯

INT0_ISR_EXIT:
    RETI                    ;中断返回
;-----------------------------------------------
INT1_ISR:                   ;INT1 中断服务程序
    CLR     P6.4            ;点亮 P64 端口上的 LED 灯
    NOP                     ;可以在这里插入断点观察现象
    SETB    P6.4            ;关闭 LED 灯
    RETI                    ;中断返回
;-----------------------------------------------
START:

    MOV     P_SW2,#80H      ;允许访问扩展的特殊寄存器，XFR

    MOV     P0M0, #00H      ;设置 P0 口为准双向口模式
    MOV     P0M1, #00H
    MOV     P1M0, #00H      ;设置 P1 口为准双向口模式
    MOV     P1M1, #00H
    MOV     P2M0, #00H      ;设置 P2 口为准双向口模式
    MOV     P2M1, #00H
    MOV     P3M0, #00H      ;P32、P33 设置为高阻输入(需要同步开启上拉电阻)
    MOV     P3M1, #0CH
    MOV     P4M0, #00H      ;设置 P4 口为准双向口模式
    MOV     P4M1, #00H
    MOV     P5M0, #00H      ;设置 P5 口为准双向口模式
    MOV     P5M1, #00H
    MOV     P6M0, #00H      ;设置 P6 口为准双向口模式
    MOV     P6M1, #00H
    MOV     P7M0, #00H      ;设置 P7 口为准双向口模式
    MOV     P7M1, #00H

    MOV     DPTR, #P3PU     ;P32、P33 打开上拉电阻
    MOV     A, #0CH
    MOVX    @DPTR, A

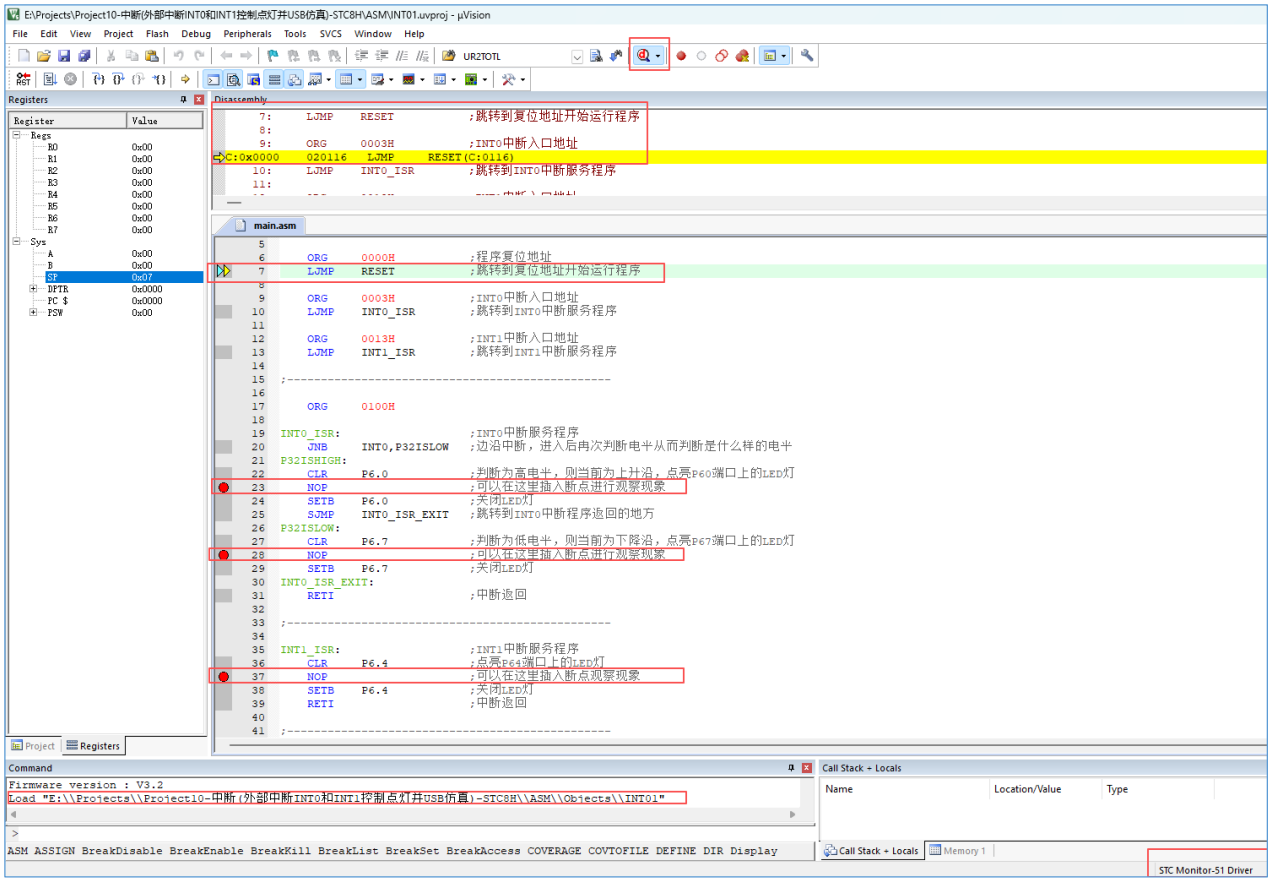
    CLR     IT0             ;使能 INT0 上升沿和下降沿中断
;   SETB    IT0             ;使能 INT0 下降沿中断
    SETB    EX0             ;使能 INT0 中断
;   CLR     IT1             ;使能 INT1 上升沿和下降沿中断
    SETB    IT1             ;使能 INT1 下降沿中断
    SETB    EX1             ;使能 INT1 中断
```

```
    CLR     IE0             ;清 INT0 中断标志
    CLR     IE1             ;清 INT1 中断标志

    SETB    EA              ;打开总中断开关
    CLR     P40             ;打开 LED 灯供电
;-----------------------------------------------
MAINLOOP:
    SJMP    MAINLOOP        ;主循环为空循环，所有内容都在中断中

;-----------------------------------------------
END
```

使用 AiCube-ISP 系统软件及 STC8H8K64U 实验箱制作仿真芯片，上述代码编译无误，点击“Debug”按钮，进入仿真界面。为方便观察实验现象，需在中断服务程序中设置如下 3 个断点，如下图：



实验步骤及实验现象与 C 语言代码编写的该项目完全相同。

2.9.2 中断实验 2(单片机主动触发外部中断 INT2/P3.6，INT3/P3.7 控制点灯并 USB 仿真)

1、 了解 STC8H8K64U 新增的外部中断 INT2/P3.6，INT3/P3.7 功能
2、 学习使用 USB 进行仿真
3、 学习使用在中断中设置标志位，在主循环中查询并处理中断事件的处理方式

2.9.2.1 实验介绍

1，下降沿触发中断，单片机内部主动触发 INT2/P3.6 口，产生中断实验演示程序：
P3.6 设置为高阻输入，打开内部上拉电阻
或 P3.6 设置为准双向口，打开内部上拉电阻
INT2/P3.6 固定为下降沿触发中断
2，下降沿触发中断，单片机内部主动触发 INT3/P3.7 口，产生中断实验演示程序：
P3.7 设置为高阻输入，打开内部上拉电阻
或 P3.7 设置为准双向口，打开内部上拉电阻
INT3/P3.7 固定为下降沿触发中断

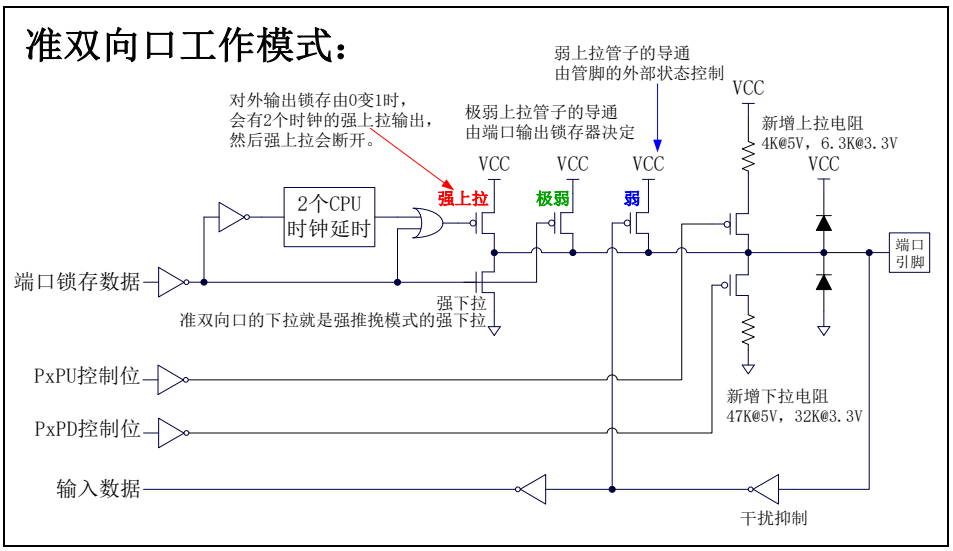
【实验步骤】:

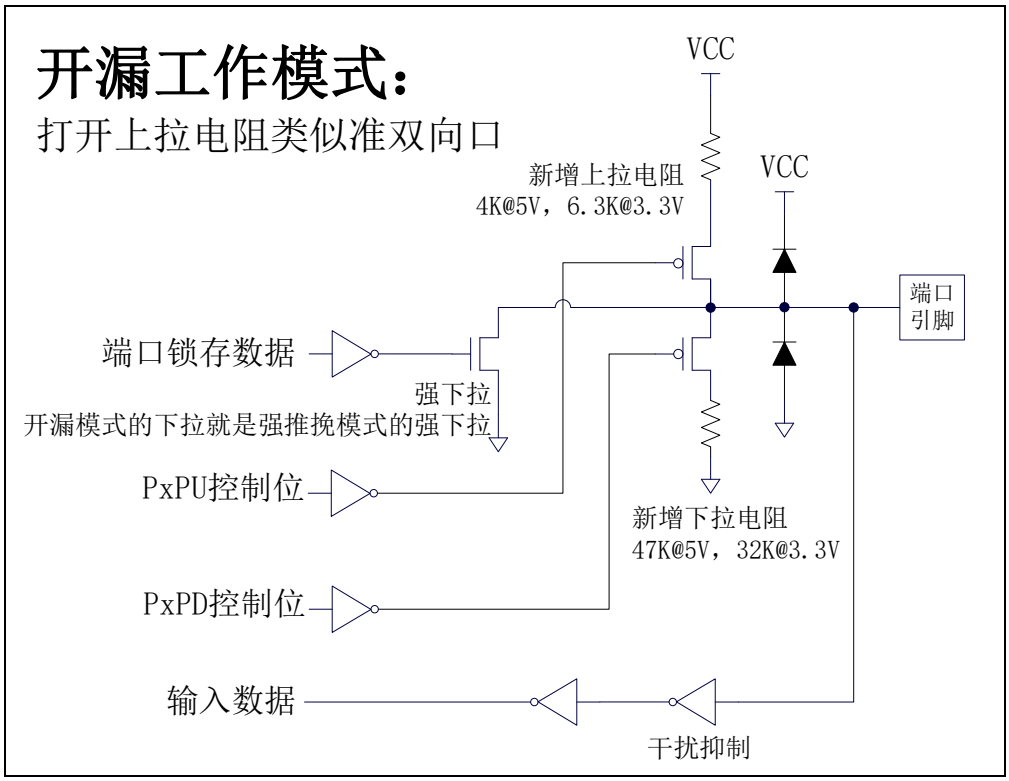
- 先点击 Run（运行）按钮，运行到单片机内部主动触发 INT2/P3.6 时，会进入 INT2 中断，点亮 LED6.0 灯，退出中断服务程序，继续运行。
- 运行到单片机内部主动触发 INT3/P3.7 口，会进入 INT3 中断，点亮 LED6.7 灯，退出中断服务程序。

利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪。

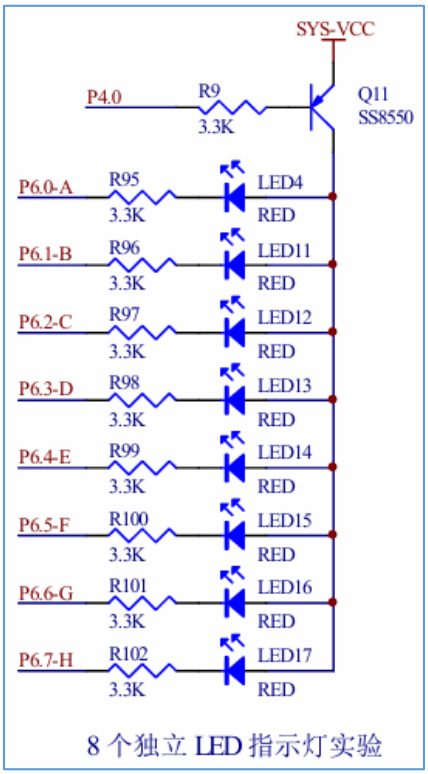
2.9.2.2 原理图

- **外部中断【INT2/P3.6，INT3/P3.7】实验原理图@STC8H8K64U 实验箱**
  在本实验中,【INT2/P3.6,INT3/P3.7】没有设计外部按键输入，而采用内部将【INT2/P3.6,INT3/P3.7】拉低产生下降沿的方式产生中断的方法。【INT2/P3.6，INT3/P3.7】工作在准双向口/开漏模式时，输出是独立的，输入也是独立的，【INT2/P3.6，INT3/P3.7】对外输出低后，【INT2/P3.6，INT3/P3.7】读到的外部状态就是低；输出高，在打开内部上拉电阻或有上拉电阻的状态下，读到的就是高，除非外界有强行拉低的电路。





➢ **外部中断实验辅助显示 LED 灯原理图:**



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。

如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。

如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。

如果 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。

如 P6.0 输出 0，LED60 灯就会被点亮；如 P6.0 输出 1，LED60 灯就会熄灭；
如 P6.1 输出 0，LED61 灯就会被点亮；如 P6.1 输出 1，LED61 灯就会熄灭；
如 P6.2 输出 0，LED62 灯就会被点亮；如 P6.2 输出 1，LED62 灯就会熄灭；
如 P6.3 输出 0，LED63 灯就会被点亮；如 P6.3 输出 1，LED63 灯就会熄灭；
如 P6.4 输出 0，LED64 灯就会被点亮；如 P6.4 输出 1，LED64 灯就会熄灭；
如 P6.5 输出 0，LED65 灯就会被点亮；如 P6.5 输出 1，LED65 灯就会熄灭；
如 P6.6 输出 0，LED66 灯就会被点亮；如 P6.6 输出 1，LED66 灯就会熄灭；
如 P6.7 输出 0，LED67 灯就会被点亮；如 P6.7 输出 1，LED67 灯就会熄灭

如 P6 口对外全部送 0，则这 8 个灯全亮；
如 P6 口对外全部送 1，则这 8 个灯全灭

如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P6 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

2.9.2.3 实验程序 C 代码

```
#include <STC8H.H>
#include "intrins.h"            //使用_nop_()函数所必须要包含的头文件,
                                //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                                //会导致编译器找不到这个而函数而报错。

bit   int2_flag = 0;  //定义 1 个位变量，INT2 事件位变量标志，记录 INT2 已产生中断
//  供主循环查询 INT2 是否已产生中断，在主循环中处理 INT2 的中断事件任务，不堵塞其他中断
bit   int3_flag = 0;  //定义 1 个位变量，INT3 事件位变量标志，记录 INT3 已产生中断
//  供主循环查询 INT3 是否已产生中断，在主循环中处理 INT3 的中断事件任务，不堵塞其他中断

void main (void)
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;       //P36、P37 设置为准双向口
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;

    P3PU = 0xc0;                //P36、P37 打开上拉电阻

    AUXINTIF = 0x00;        //清空 INT2、INT3 的中断标志位，防止仿真的时候被错误触发一次
    INTCLKO |= EX2;;        //使能  INT2  中断，固定只有下降沿中断
    INTCLKO |= EX3;         //使能  INT3  中断，固定只有下降沿中断
```

```
    EA = 1;                 //总中断允许位打开
    P40 = 0;                //打开 LED 灯供电

    P36 = 1;                //INT2 所在 I/O 端口
    NOP12();                //可以作为断点插入，用于观察现象
    P36 = 0;                //手动创造下降沿，这里就会直接跳转到 INT2 中断了
    NOP20();                //芯片内部对下降沿信号进行采样，此处不可设置断点
    NOP12();                //可以作为断点插入，用于观察现象

    P37 = 1;                //INT3 所在 I/O 端口
    NOP12();                //可以作为断点插入，用于观察现象
    P37 = 0;                //手动创造下降沿，这里就会直接跳转到 INT3 中断了
    NOP20();                //芯片内部对下降沿信号进行采样，此处不可设置断点
    NOP12();                //可以作为断点插入，用于观察现象

    P37 = 1;                //手动创造上升沿，此时不会进入中断
    NOP12();                //不会进入中断，用于观察现象
    NOP12();                //可以作为断点插入，用于观察现象

    while(1)
    {
        if(int2_flag)   //主循环中查询，INT2 是否已产生中断，是否有需要处理的 INT2 事件
        {
            int2_flag = 0;      //清 0，INT2 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(int3_flag)   //主循环中查询，INT3 是否已产生中断，是否有需要处理的 INT3 事件
        {
            int3_flag = 0;      //清 0，INT3 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
    }
}

void int2_isr(void) interrupt INT2_VECTOR
{
    int2_flag = 1;  // int2_flag 置 1 是通知主循环处理部分 INT2 中断事件不需要特急处理的任务
                    //置 1，记录 INT2 已产生中断，供主循环查询判断有无需处理的 INT2 任务

    P60 = 0;        //点亮 P60 端口上的 LED 灯
    NOP12();        //可以在这里插入断点观察现象
    P60 = 1;        //关闭 LED 灯
}
```

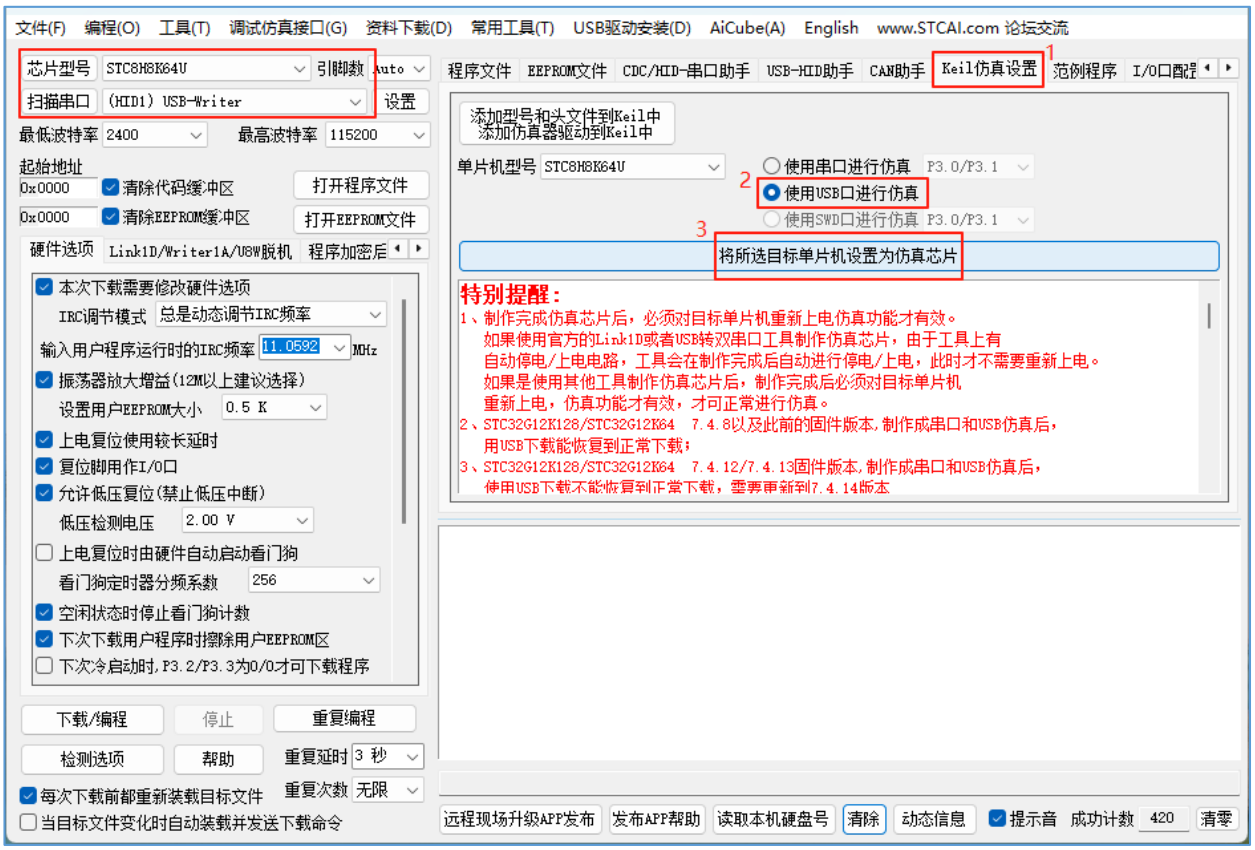
```
void int3_isr(void) interrupt INT3_VECTOR
{
    int3_flag = 1;  // int3_flag 置 1 是通知主循环处理部分 INT3 中断事件不需要特急处理的任务
                    //置 1，记录 INT3 已产生中断，供主循环查询判断有无需处理的 INT3 任务

    P67 = 0;        //点亮 P67 端口上的 LED 灯
    NOP12();        //可以在这里插入断点观察现象
    P67 = 1;        //关闭 LED 灯
}
```

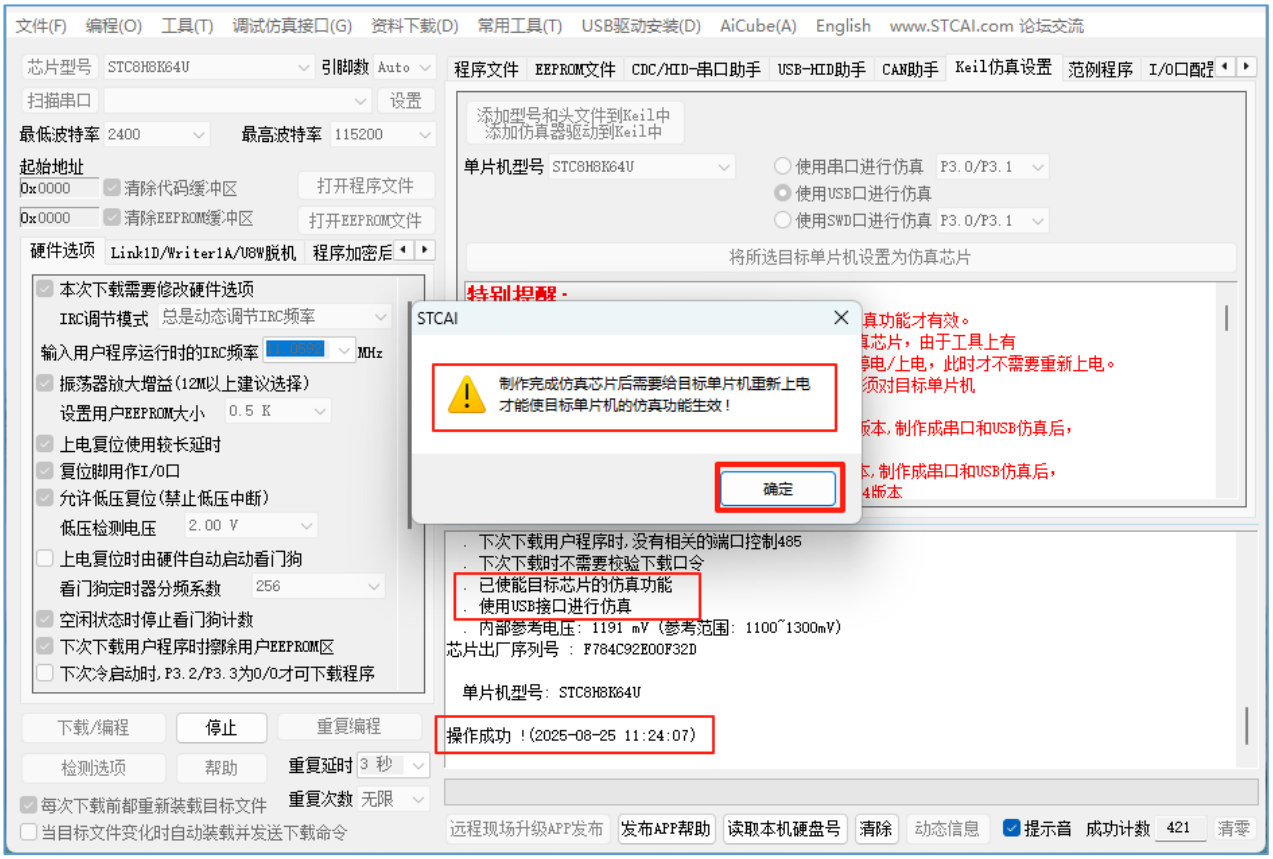
2.9.2.4　制作仿真芯片

在 AiCube-ISP 软件界面，看到扫描串口列表框显示“(HID1) USB Writer”：

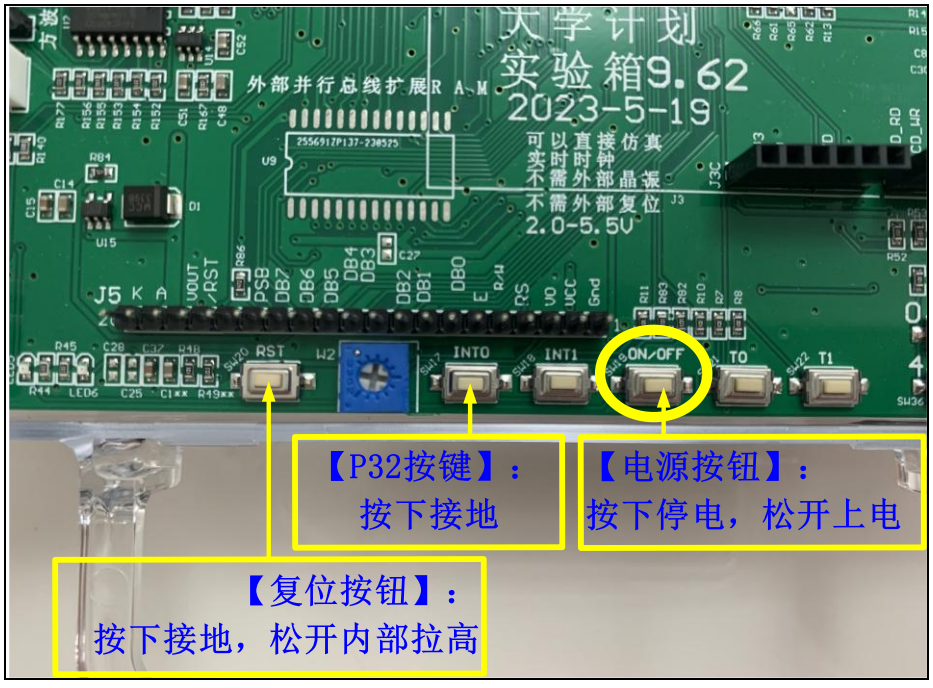
- 点击“Keil 仿真设置”标签；
- 在“Keil 仿真设置”选项卡，选择“使用 USB 口进行仿真”
- 点击“将所选目标单片机设置为仿真芯片”



然后看到 AiCube-ISP 软件在自动进行仿真芯片的设置，界面下面的进度条在推进，然后“操作成功!”界面出现特别提示：“制作完成仿真芯片后需要给目标单片机重新上电才能使目标单片机的仿真生效!”如下图：



所以，按提示按一下实验箱上的“Power_SW”按钮：



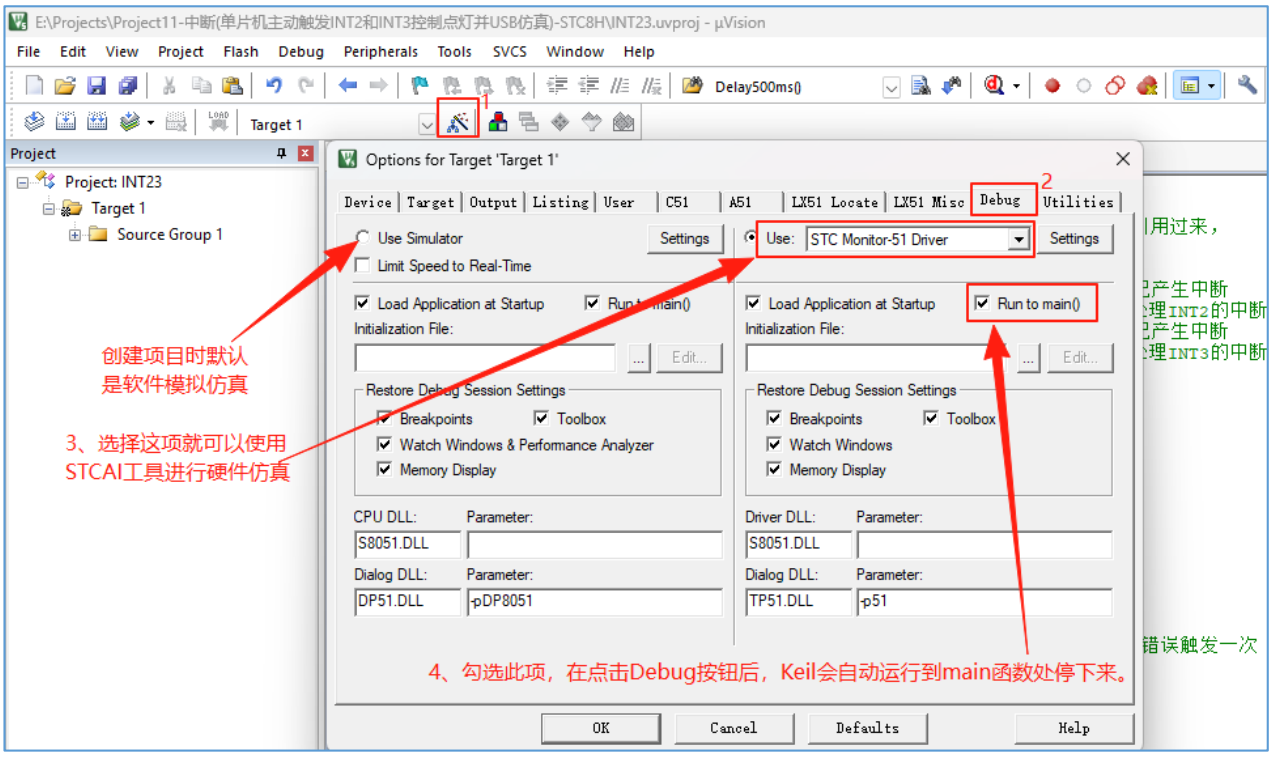
2.9.2.5　仿真设置，在 Keil 中

本项目是使用 STC8H8K64U 内置 USB 直接仿真，除了新建项目时必要设置的基础选项外，还要进行仿真设置：“Debug”选项卡和“Utilities”选项卡。

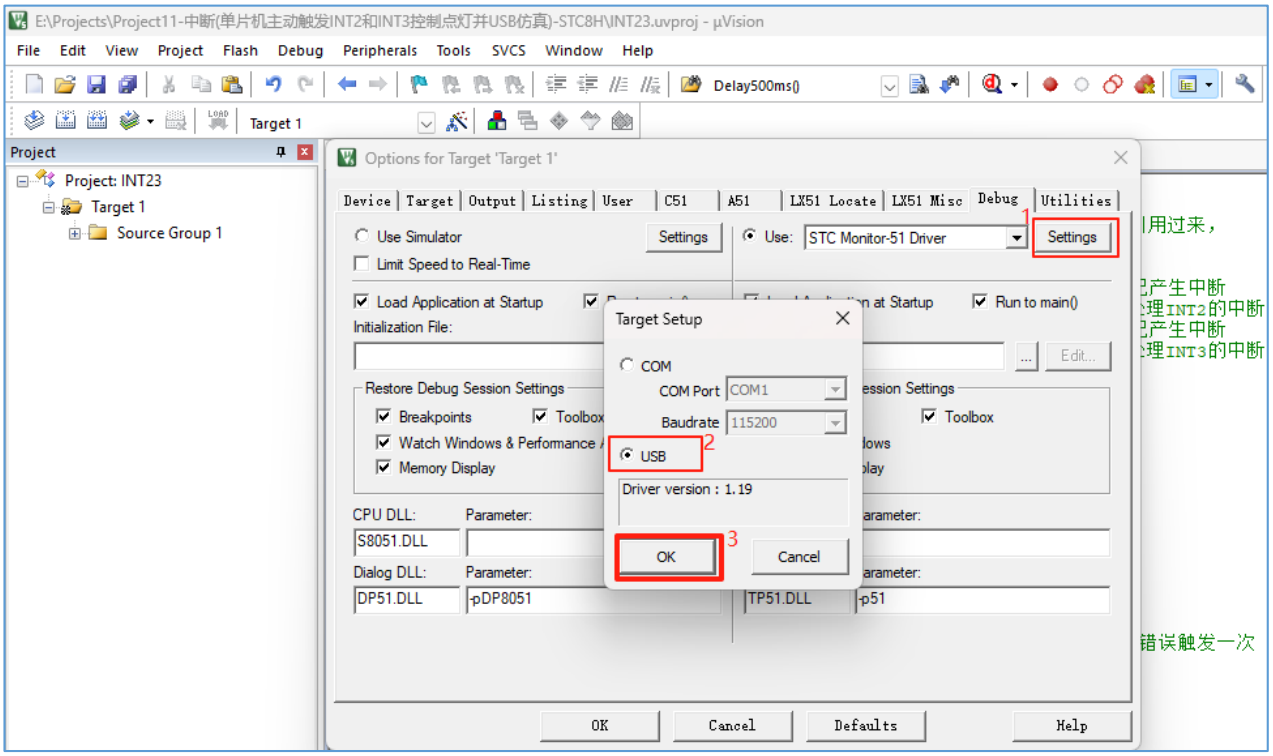
点击工具栏上的魔术棒的工具，打开“Options for Target ‘Target 1’”任务选项面板：

2.9.2.5.1　设置 Debug 选项卡

➲ **选择硬件仿真：**系统默认是软件模拟仿真，本项目是使用 STC8H8K64U 内置 USB 直接仿真，属于硬件仿真，要选择“STC Monitor-251 Driver”

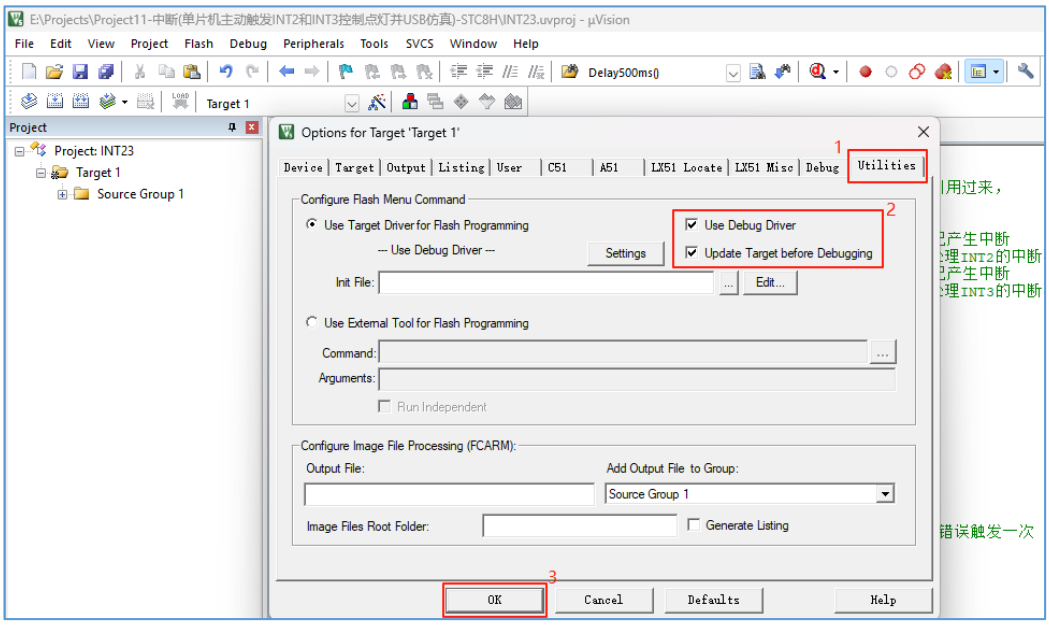


➲ **选择仿真接口：**STC8H8K64U 芯片内置的硬件 USB 支持直接仿真，本项目要选择 USB 为仿真接口。



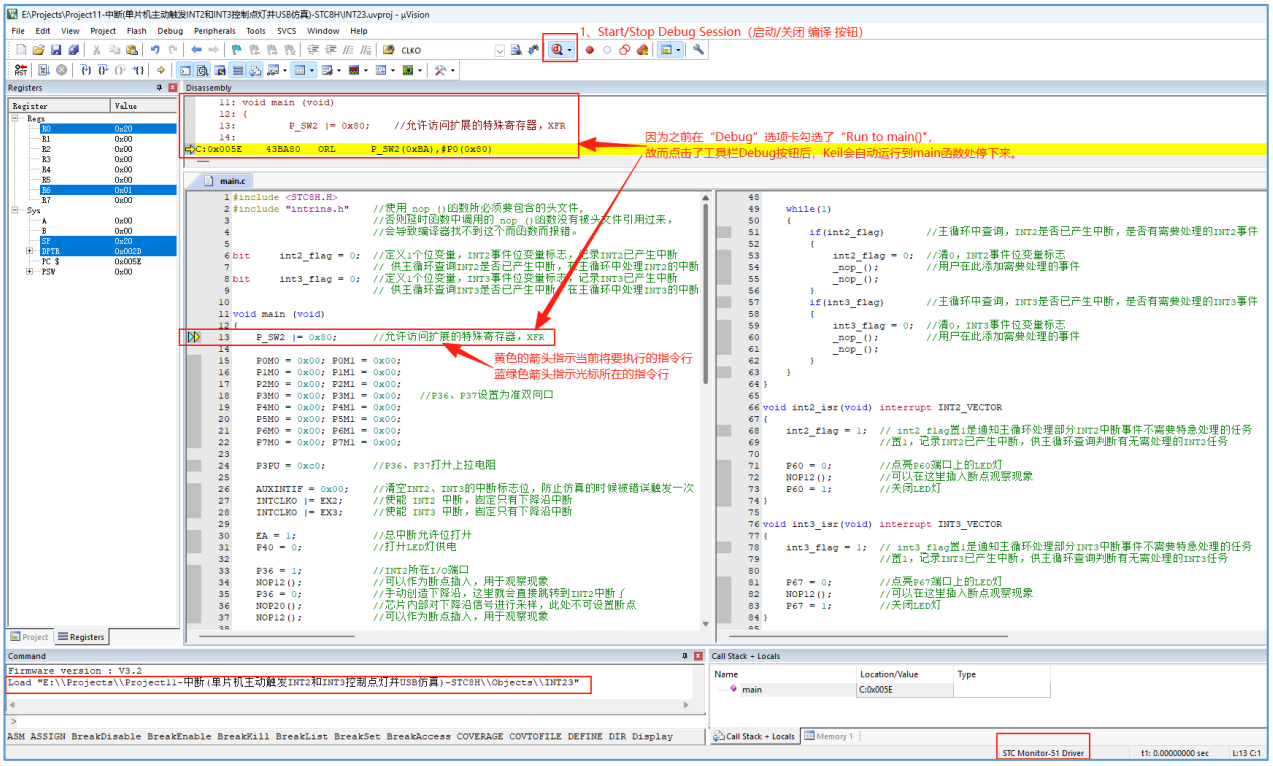
2.9.2.5.2　设置 Utilities 选项卡

切换到“Utilities”选项卡，勾选“Use Debug Driver”，点击“OK”，完成设置。



2.9.2.6　程序实现效果，观察实验现象

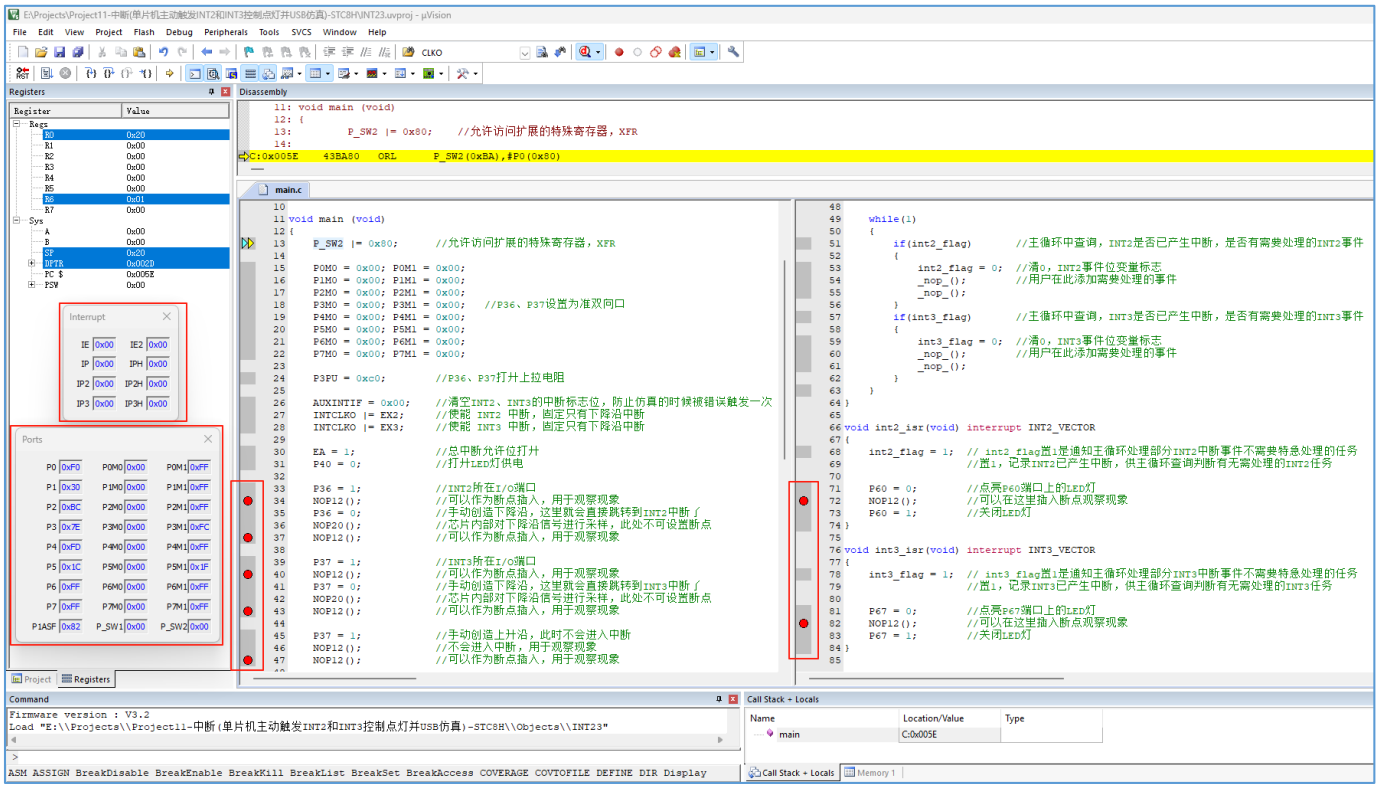
程序代码编译无误后，点击工具栏的 Debug（启动/关闭编译调试操作界面）按钮后，Keil 软件进入仿真界面，如下图：



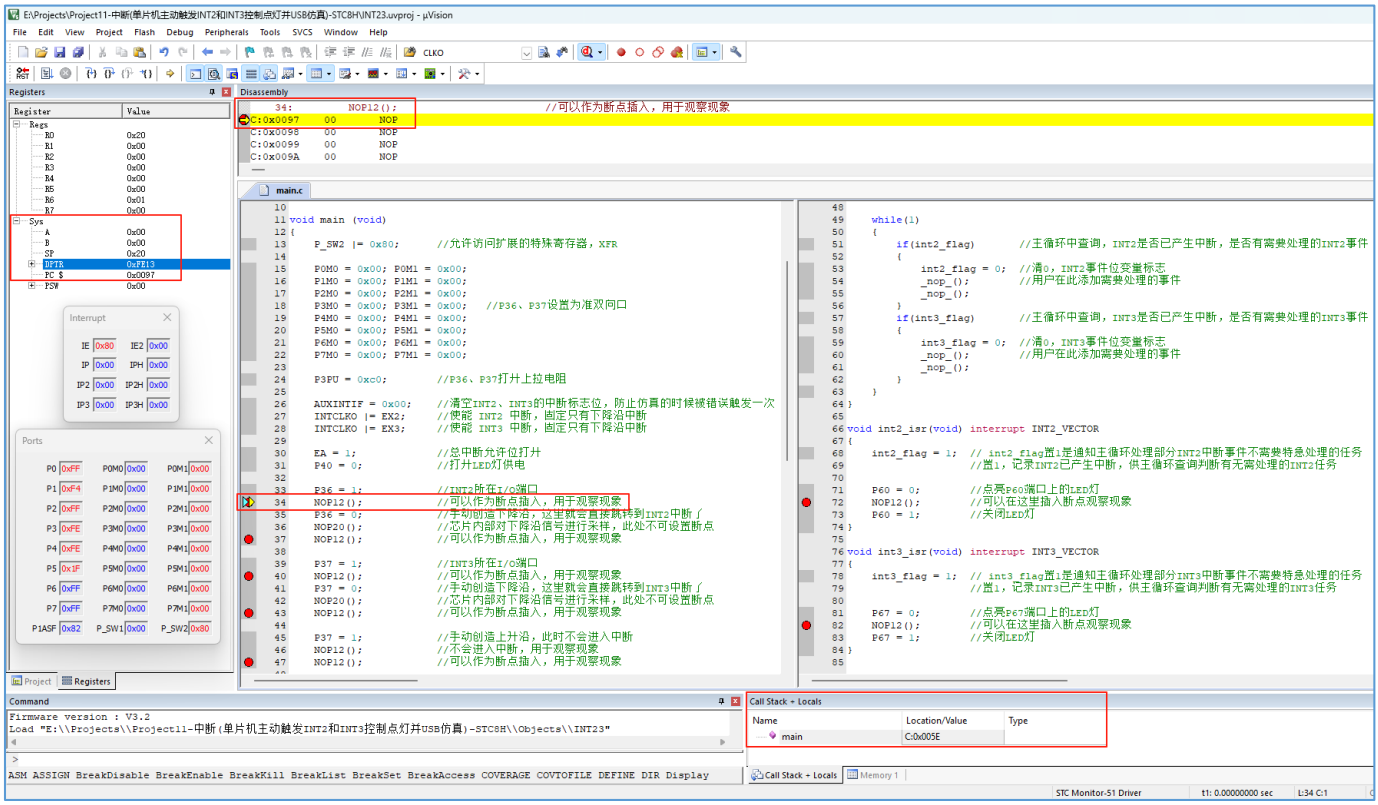
提示：退出 Debug 调试界面，同样点击工具栏的“Debug”按钮。

利用仿真，可以设置运行到断点处，捕捉当前程序运行到哪

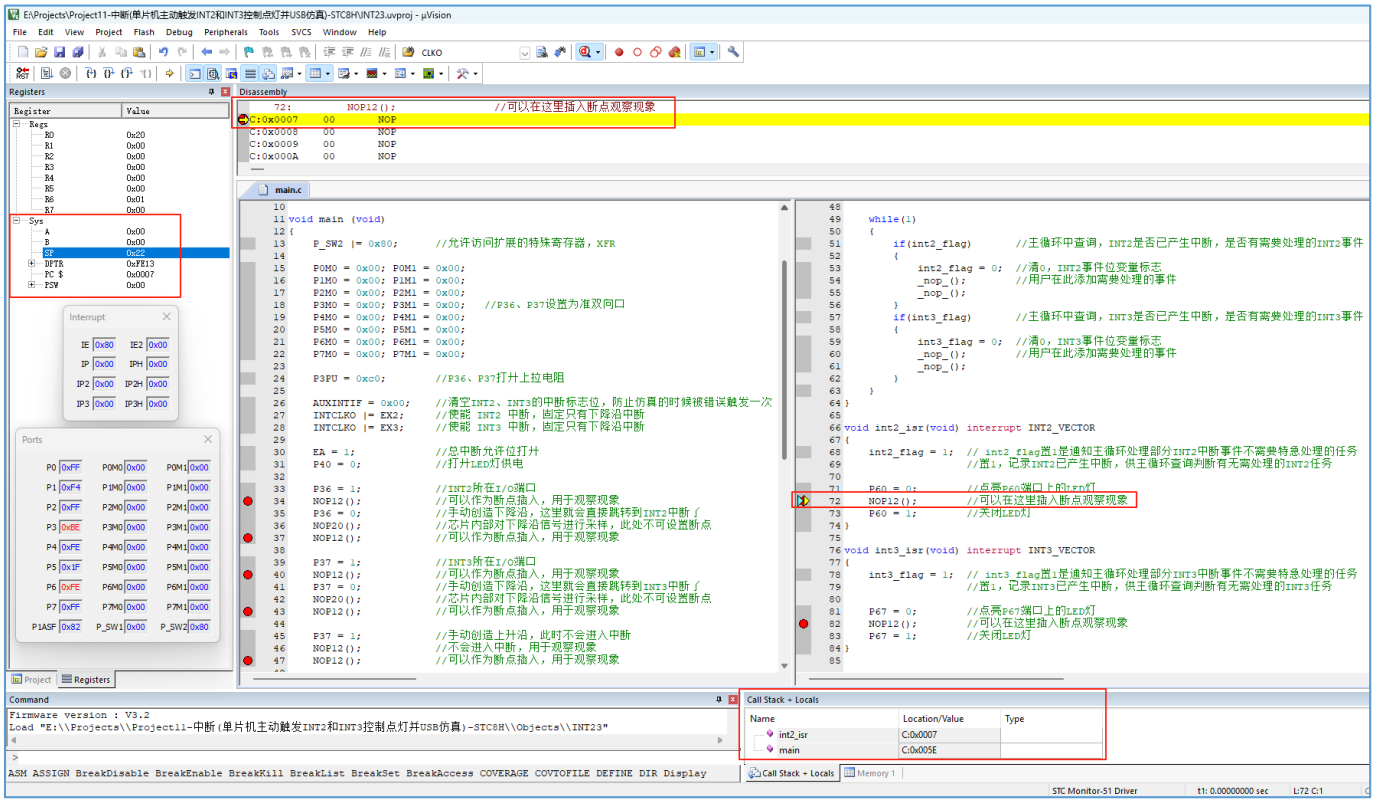
为方便观察，需在中断服务程序中设置如下 7 个断点，同时，打开菜单项“Debug”→“All Ports”和“Interrupt”，一起观察：



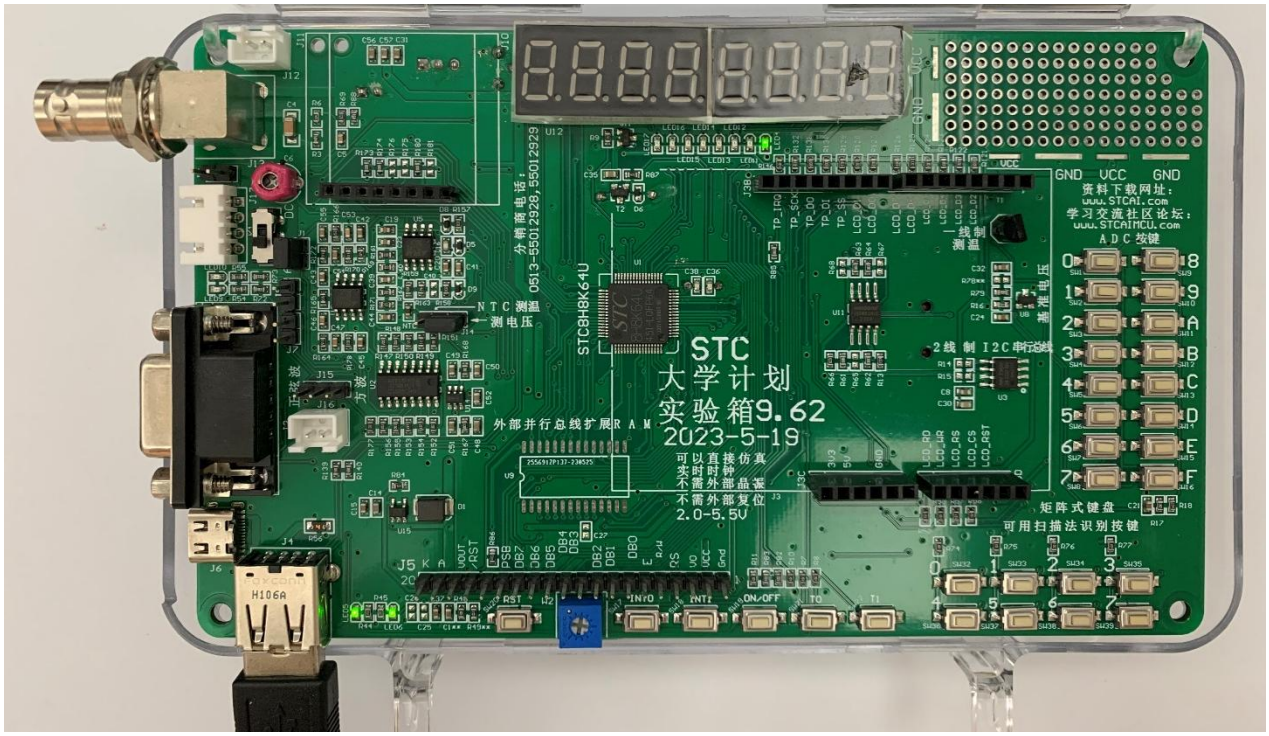
1、先点击 Run（运行）按钮，程序会停在如下图所示断点处（准备内部自动触发 INT2）：



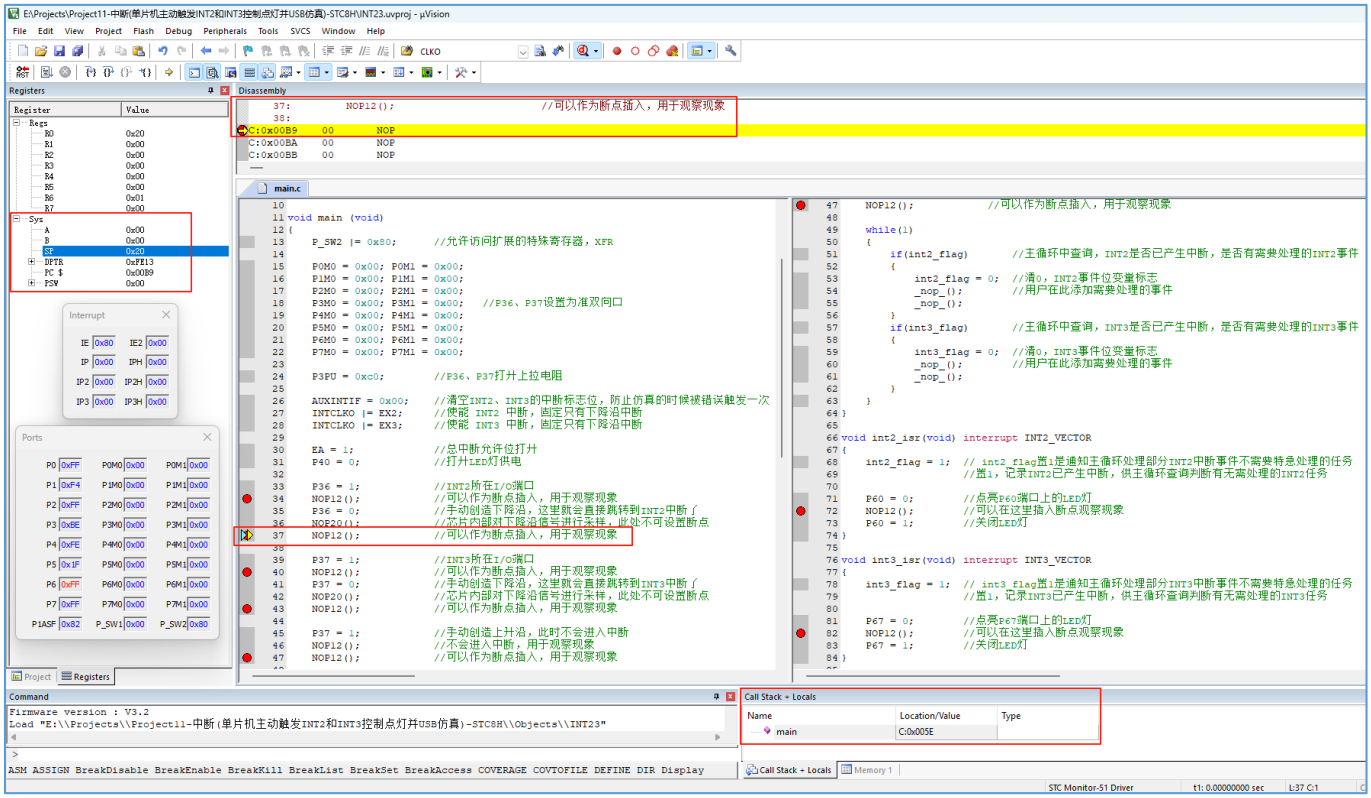
2、点击 Run（运行）按钮，继续运行，单片机内部主动触发 INT2/P3.6，会进入 INT2 中断，点亮 LED6.0 灯，程序会停在如下图所示断点处：



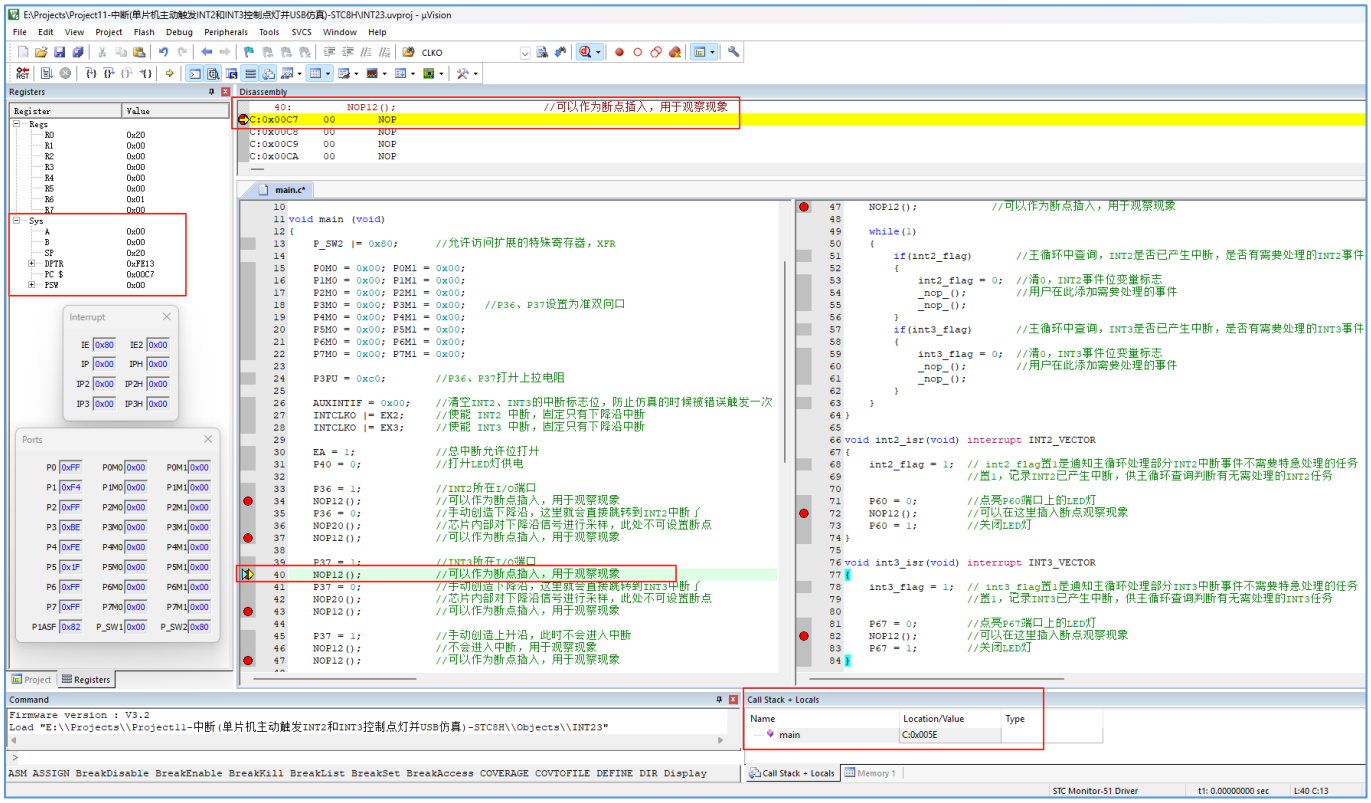
实验箱如图，点亮 LED6.0 灯：



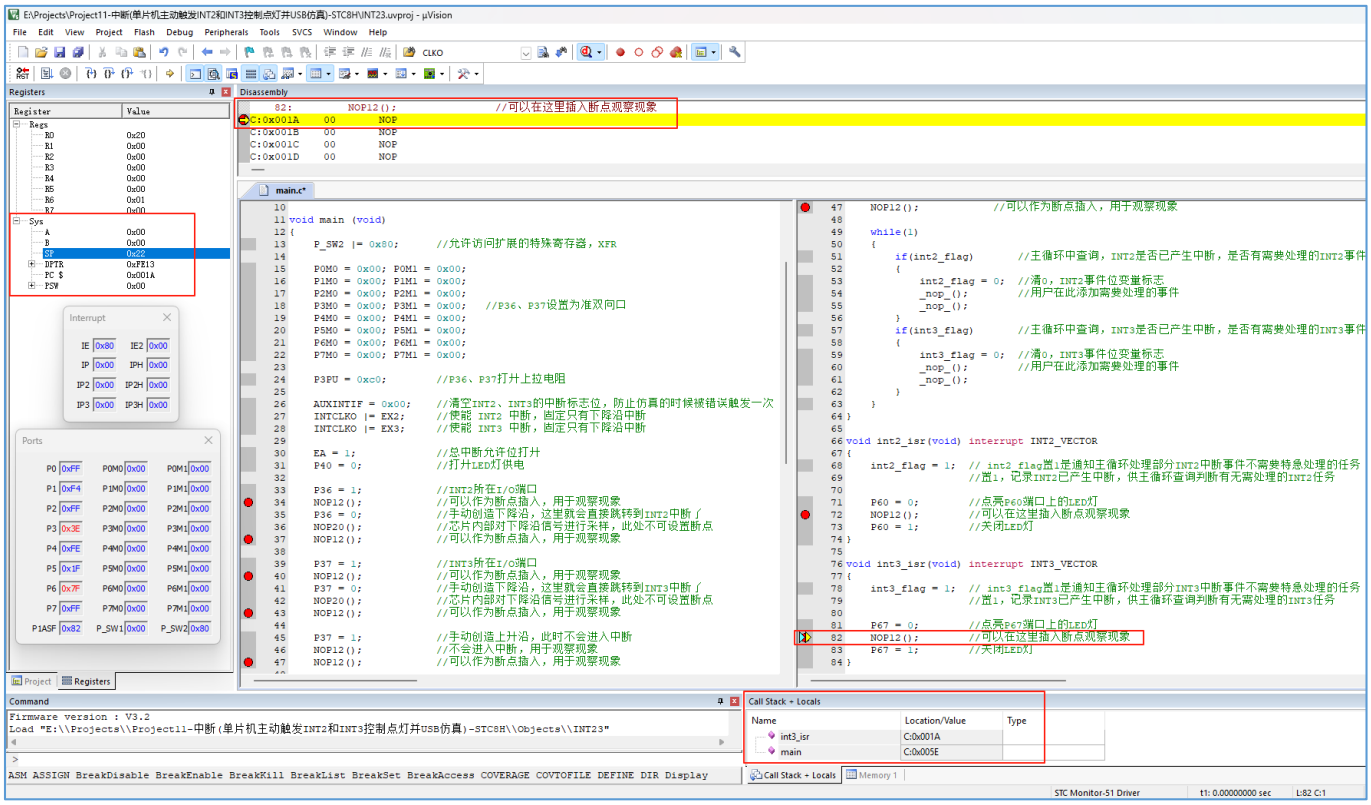
3、点击 Run（运行）按钮，继续运行。实验箱上 LED6.0 灯灭，程序会停在如下图所示断点处（中断返回）：



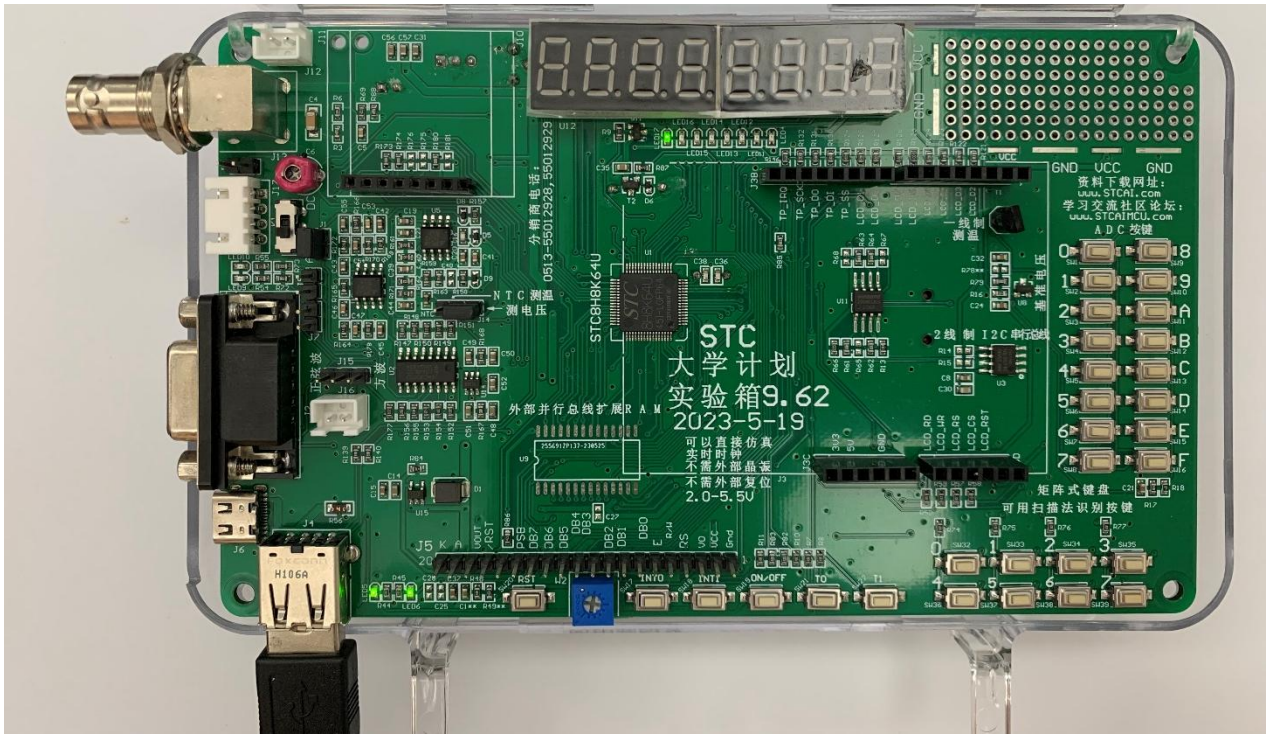
4、点击 Run（运行）按钮，继续运行，程序会停在如下图所示断点处（准备内部自动触发 INT3）：



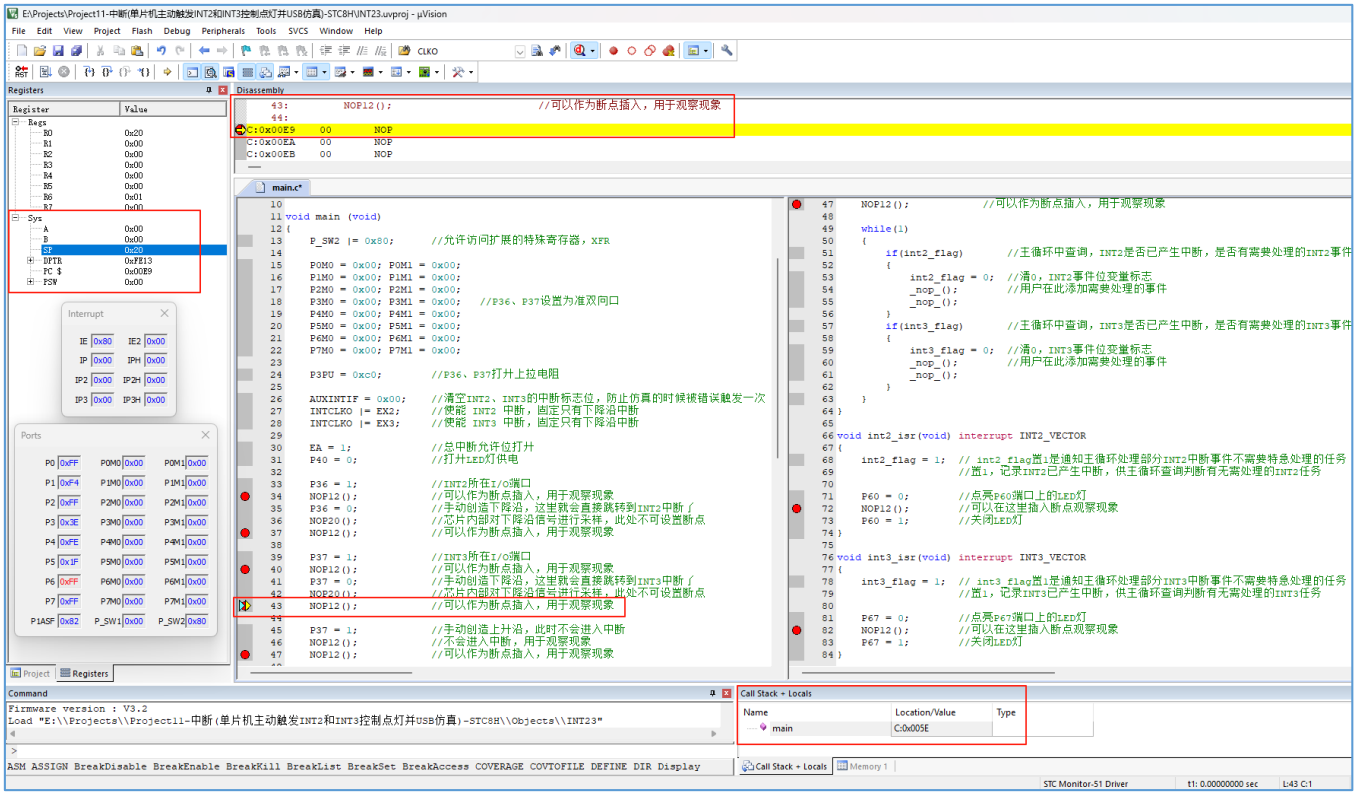
5、点击 Run（运行）按钮，继续运行，单片机内部主动触发 INT3/P3.7，会进入 INT3 中断，点亮 LED6.7 灯，程序会停在如下图所示断点处：



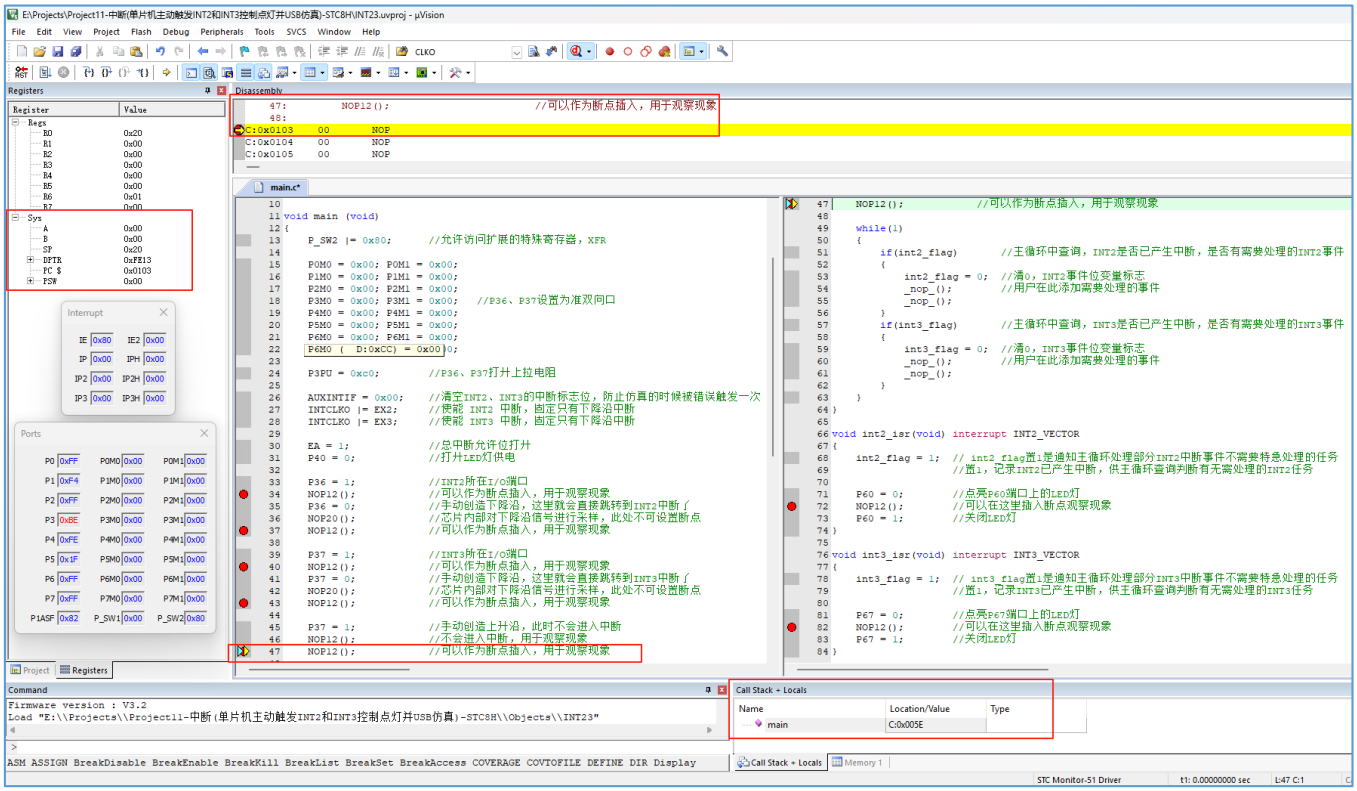
实验箱如图，点亮 LED6.7 灯：



6、点击 Run（运行）按钮，继续运行，实验箱上 LED6.7 灯灭，程序会停在如下图所示断点处（中断返回）：



7、点击 Run（运行）按钮，继续运行，程序会停在如下图所示断点处（测试上升沿不触发中断）：



点击 Run（运行）按钮，程序进入主循环，实验结束。

2.10 定时器实验 -- 定时器 0/1，16 位定时器

2.10.1 定时器 0/1，16 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息）

1、 了解传统的定时器 0，定时器 1 的使用
2、 了解 STC8H8K64U 对定时器 0，定时器 1 功能的扩展（增加 16 位自动重载模式）
3、 学习使用 USB-CDC 和电脑进行数据传输
4、 学习使用在中断中设置标志位，在主循环中查询并处理中断事件的处理方式

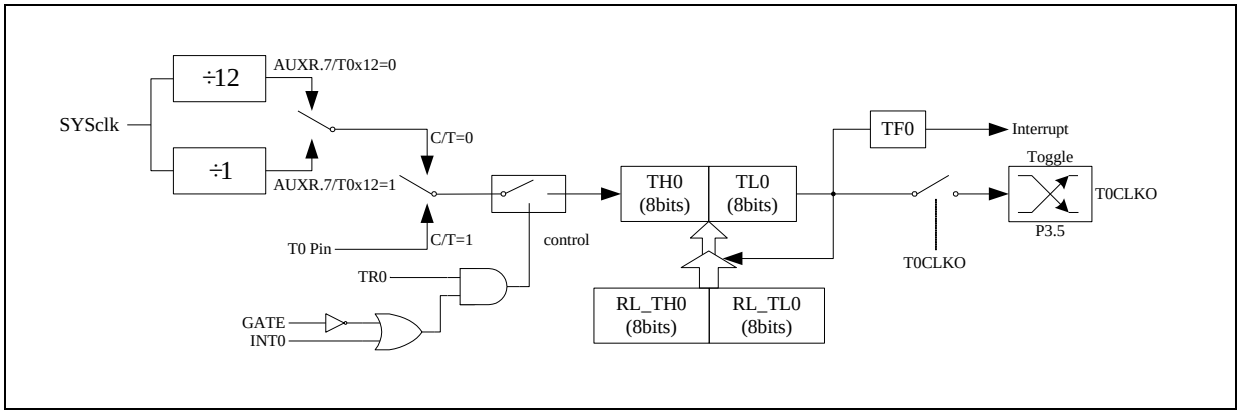
2.10.1.1 实验介绍

定时器 0（Timer0）定时 2s 产生中断，LED60 灯闪烁，CDC 窗口显示“Timer0!”
定时器 1（Timer1）定时 500ms 产生中断，LED67 灯闪烁，CDC 窗口显示“Timer1!”
实验现象：

- AiCube-ISP 软件的 CDC 窗口每显示 4 次“Timer1”，显示 1 次“Timer0”
- 实验箱，LED67 亮灭各 2 次，LED60 亮 1 次

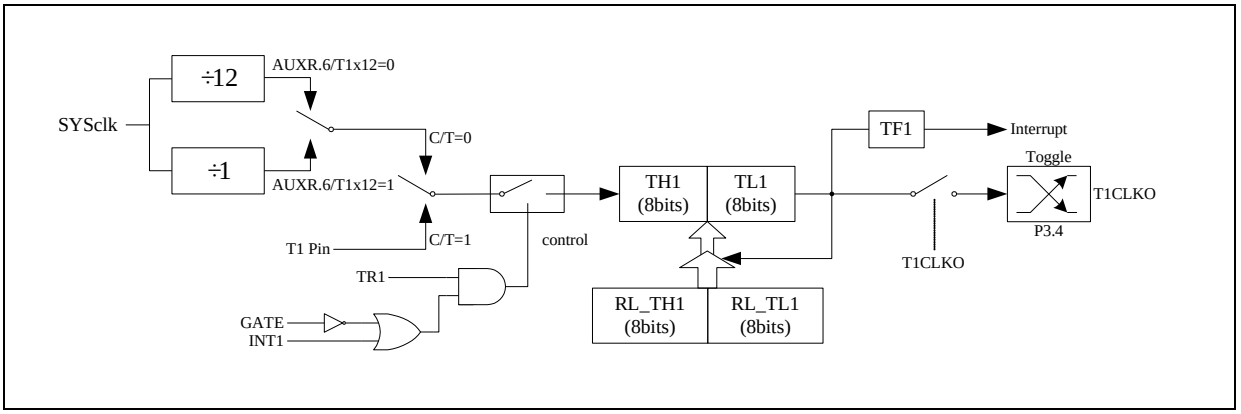
2.10.1.2 原理图

- **定时器 0 模式 0 结构图：此模式下定时器/计数器 0 作为可自动重装载的 16 位计数器。如下图所示：**



定时器/计数器 0 的模式 0：16 位自动重装载模式

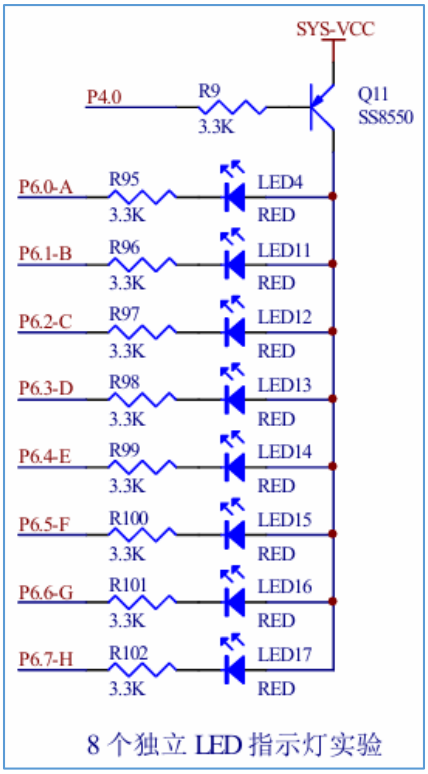
- **定时器 1 模式 0 结构图：此模式下定时器/计数器 1 作为可自动重装载的 16 位计数器。如下图所示：**



定时器/计数器 1 的模式 0：16 位自动重装载模式

➲ **LED 指示灯原理图**

P6 口都有单独的指示灯



I/O 口 P4.0 控制 PNP 三极管的通断，控制这 8 个灯的整体供电。

如果 P4.0 输出 1，则 PNP 三极管不导通，这 8 个灯都不会获得电源，都不会有电流流过，都不会被点亮。

如果 P4.0 输出 0，则 PNP 三极管导通，这 8 个灯才有可能会流过电流，才有可能会被点亮。

如果 P4.0 输出 0，PNP 三极管导通，特定 LED 灯的控制 I/O 口如输出 0，则该 LED 灯会有电流流过，会被点亮。

如 P6.0 输出 0，LED60 灯就会被点亮；如 P6.0 输出 1，LED60 灯就会熄灭；
如 P6.1 输出 0，LED61 灯就会被点亮；如 P6.1 输出 1，LED61 灯就会熄灭；
如 P6.2 输出 0，LED62 灯就会被点亮；如 P6.2 输出 1，LED62 灯就会熄灭；
如 P6.3 输出 0，LED63 灯就会被点亮；如 P6.3 输出 1，LED63 灯就会熄灭；
如 P6.4 输出 0，LED64 灯就会被点亮；如 P6.4 输出 1，LED64 灯就会熄灭；
如 P6.5 输出 0，LED65 灯就会被点亮；如 P6.5 输出 1，LED65 灯就会熄灭；
如 P6.6 输出 0，LED66 灯就会被点亮；如 P6.6 输出 1，LED66 灯就会熄灭；
如 P6.7 输出 0，LED67 灯就会被点亮；如 P6.7 输出 1，LED67 灯就会熄灭

如 P6 口对外全部送 0，则这 8 个灯全亮；
如 P6 口对外全部送 1，则这 8 个灯全灭

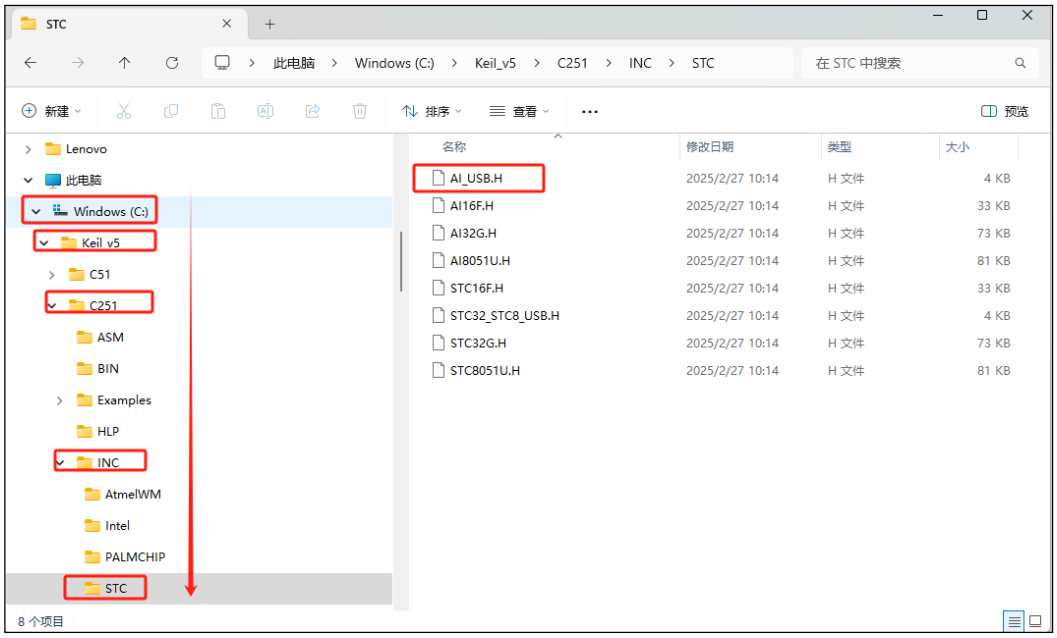
如果 P4.0 输出 1，则 PNP 三极管不导通，不管 P6 口对外全部送 0 或送 1，这 8 个灯都不会有电流流过，都不会亮。P4.0 控制 PNP 三极管，作为这 8 个 LED 指示灯的总电源开关。

2.10.1.3 准备工作--添加头文件和库函数

1、添加常用头文件 ai_usb.h

```
#include "ai_usb.h"                    //使用 usb 所需要包含的头文件。
```

这个头文件在使用 AiCube-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



2、添加 usb_cdc 库函数文件

➲ 下载 **usb_cdc 库函数文件**

因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载



下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，

在’库文件\查询模式 STC-CDC 库文件’中，复制出来 **stc_usb_cdc_8h_data.LIB**

【8 位的 8051 的 USB 库 可以用 BL51 连接器

BL51： 不支持 **REMOVEUNUSED；BL51** 支持 **扩展中断号工具；**

如果 不用 REMOVEUNUSED，可以 使用 BL51 ！

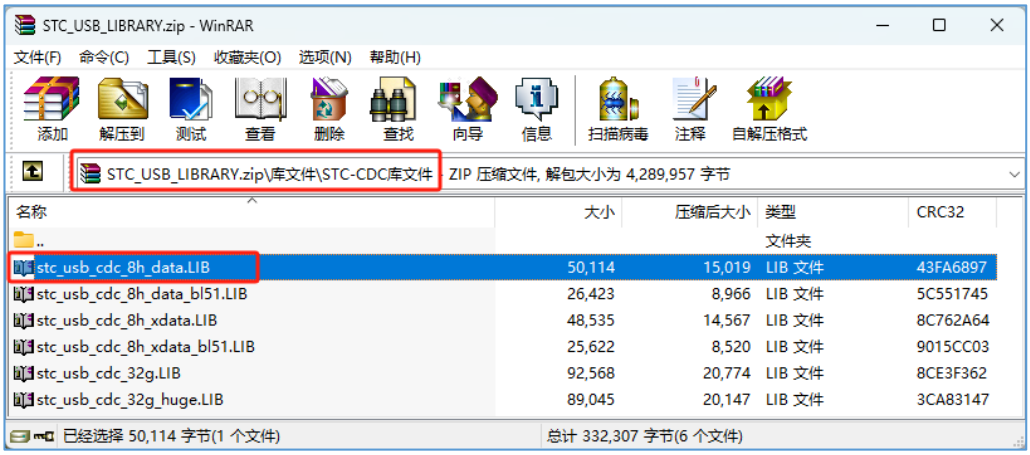
LX51： 支持 **REMOVEUNUSED；LX51** 不支持 **扩展中断号工具；**

如果 不用 扩展中断号工具，可以 使用 LX51 ！

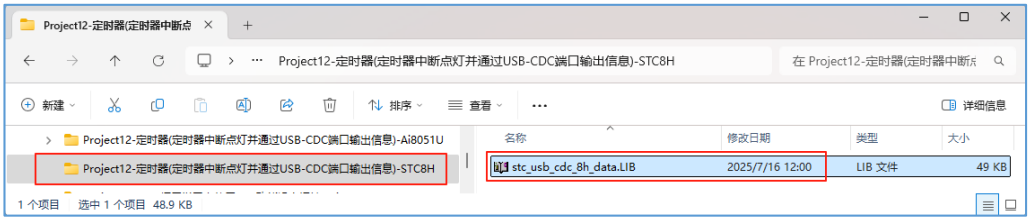
详情请参考本公司技术论坛链接：8 位的 8051 的 USB 库可以用 BL51；BL51 不支持 REMOVEUNUSED；

https://www.stcaimcu.com/thread-17105-1-1.html

(出处: 国芯人工智能技术交流网站)】

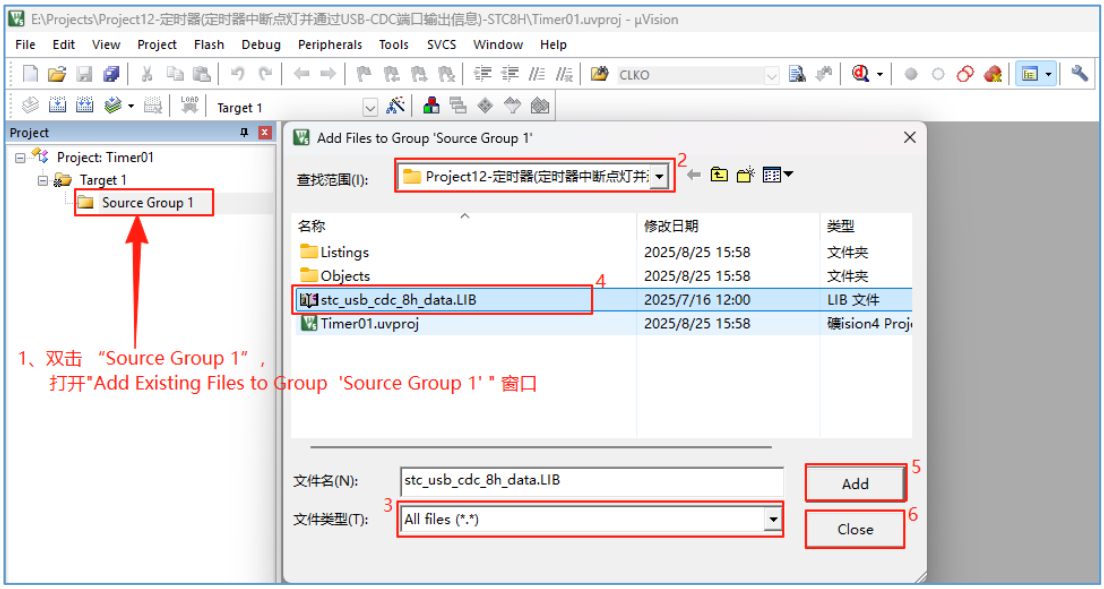


➲ **将库函数“stc_usb_cdc_8h_data.LIB”复制到创建项目所在的文件夹中**

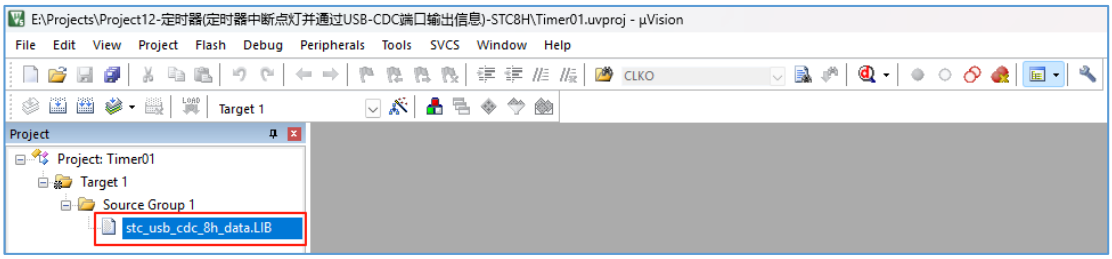


➲ **将库函数“stc_usb_cdc_8h_data.LIB”添加进入此项目重新文件中**

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 **stc_usb_cdc_8h_data.LIB** 添加进入此工程，就可以使用 USB-CDC 功能进行端口通讯。



Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

2.10.1.4　　实验程序代码

```
#include <STC8H.H>          // STC8H 的通用头文件，头文件不分大小写
#include   "ai_usb.h"        //USB 库和 USB 虚拟设备的头文件
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。

bit   t0_flag = 0;    //定义 1 个位变量，T0 事件位变量标志，记录定时器 0 已产生中断
//  供主循环查询定时器 0 是否已产生中断，在主循环中处理定时器 0 的中断事件任务，不堵塞其他中断
bit   t1_flag = 0;    //定义 1 个位变量，T1 事件位变量标志，记录定时器 1 已产生中断
//  供主循环查询定时器 1 是否已产生中断，在主循环中处理定时器 1 的中断事件任务，不堵塞其他中断

int cnt40 = 0;        //定时器 0 中断次数计数器
int cnt10 = 0;        //定时器 1 中断次数计数器

void Timer0_Isr(void) interrupt TMR0_VECTOR      //定时器 0 中断服务程序
{
    if (cnt40++ < 40)   //定时器 0 中断次数累加
        return;         //如果没有达到 40 次（50ms*40=2s），就直接返回
    cnt40 = 0;          //达到 40 次后，计数器重新计数

    P60 = ~P60;   //P60 灯闪烁，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                  //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                  //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t0_flag = 1;  // t0_flag 置 1 是通知主循环处理部分 T0 中断事件不需要特急处理的任务
                  //置 1，记录定时器 0 已产生中断，供主循环查询判断有无需处理的定时器 0 任务
}
//定时器 0 中断服务程序，TMR0_VECTOR 在 STC8H.H 头文件中已宏定义为 1
void Timer0_Init(void)        //定时器 0 初始化，50ms@11.0592MHz
{
    AUXR &= 0x7F;         //定时器时钟 12T 模式
    TMOD &= 0xF0;         //设置定时器模式
    TL0 = 0x00;           //设置定时初始值
    TH0 = 0x4C;           //设置定时初始值
    TF0 = 0;              //清除 TF0 标志
    TR0 = 1;              //定时器 0 开始计时
    ET0 = 1;              //使能定时器 0 中断
}

void   Timer1_Isr(void)   interrupt   TMR1_VECTOR
{
    if (cnt10++ < 10)   //定时器 1 中断次数累加
        return;         //如果没有达到 10 次（50ms*10=500ms），就直接返回
    cnt10 = 0;          //达到 10 次后，计数器重新计数
```

```
    P67 = ~P67;         //P67 灯闪烁，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t1_flag = 1;        // t1_flag 置 1 是通知主循环处理部分 T1 中断事件不需要特急处理的任务
                        //置 1，记录定时器 1 已产生中断，供主循环查询判断有无需处理的定时器 1 任务
}
//定时器 1 中断服务程序，TMR1_VECTOR 在 STC8H.H 头文件中已宏定义为 3
void Timer1_Init(void)      //定时器 1 初始化，50ms@11.0592MHz
{
    AUXR &= 0xBF;           //定时器时钟 12T 模式
    TMOD &= 0x0F;           //设置定时器模式
    TL1 = 0x00;             //设置定时初始值
    TH1 = 0x4C;             //设置定时初始值
    TF1 = 0;                //清除 TF1 标志
    TR1 = 1;                //定时器 1 开始计时
    ET1 = 1;                //使能定时器 1 中断
}

void   sys_init(void)       //系统初始化
{
    P_SW2 |= 0x80;          //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;     //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;     //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;     //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;     //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;     //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;     //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;     //设置 P7 口为准双向口模式
}

void   main(void)
{
    sys_init();     //调用系统初始化函数
    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */
    EA = 1;             //总中断允许位打开

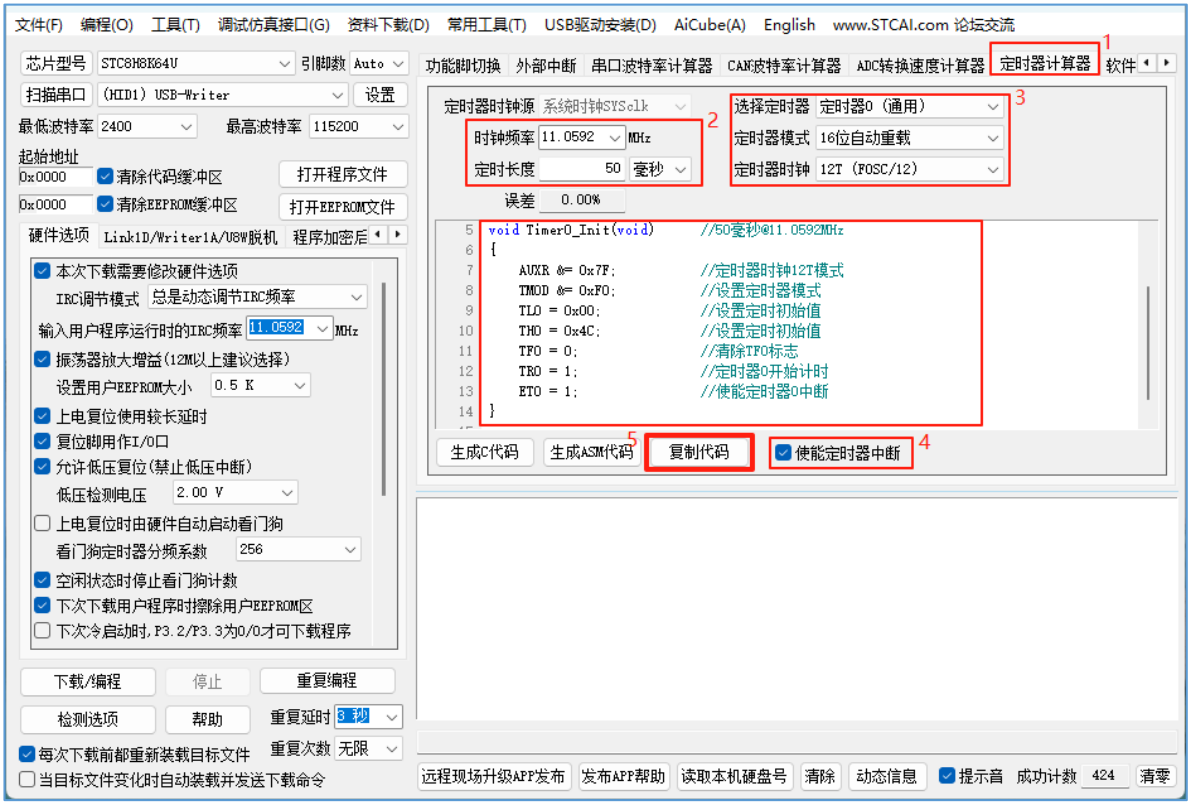
    Timer0_Init();      //调用定时器 0 初始化函数
    Timer1_Init();      //调用定时器 1 初始化函数
```

```
    P40 = 0;                //给 LED 灯供电
    while(1)                //主循环中查询需要处理的各种事件，如 T0/T1 中断事件，并打印输出状态
    {
        if(t0_flag)         //主循环中查询，定时器 0 是否已产生中断，是否有需要处理的定时器 0 事件
        {
            t0_flag = 0;            //清 0，T0 事件位变量标志
            printf_usb("Timer0！\r\n");
            //向电脑 USB-CDC 串口助手输出“Timer0!”字符串，代表主循环在处理 T0 不急的任务
        }
        if(t1_flag)         //主循环中查询，定时器 1 是否已产生中断，是否有需要处理的定时器 1 事件
        {
            t1_flag = 0;            //清 0，T1 事件位变量标志
            printf_usb("Timer1！\r\n");
            //向电脑 USB-CDC 串口助手输出“Timer1!”字符串，代表主循环在处理 T1 不急的任务
        }
    }
}
```

【小技巧 -- 灵活运用 AiCube-ISP 软件的工具自动生成代码】

定时器 0 初始化代码部分就可以使用 **AiCube-ISP** 软件的“**定时器计算器**”来生成。

上述代码中 Time0_Init(void)这一段代码，按要求设置：时钟频率 11.0592Hz，定时长度 50ms，选择定时器 0、16 位自动重载、定时器时钟 12T。**AiCube-ISP 软件“定时器计算器”窗口就会自动生成如下代码：**

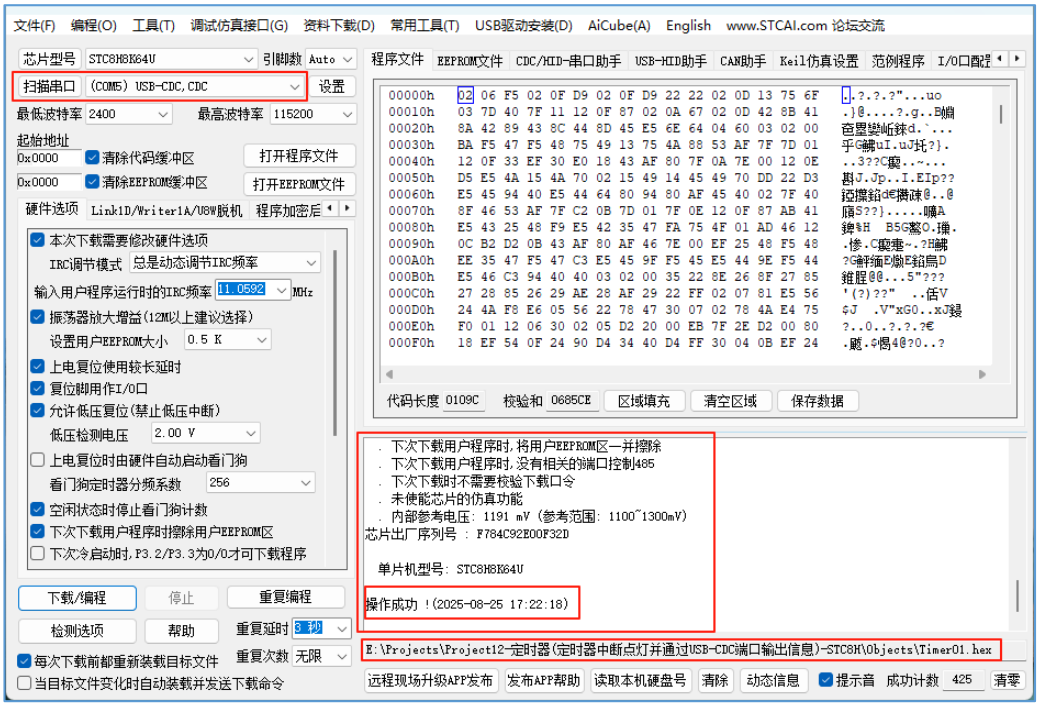


将复制出来的内容直接粘贴到代码文件中，不用手动输入。

定时器 1 的初始化代码部分获取方法同上操作。

2.10.1.5　程序下载

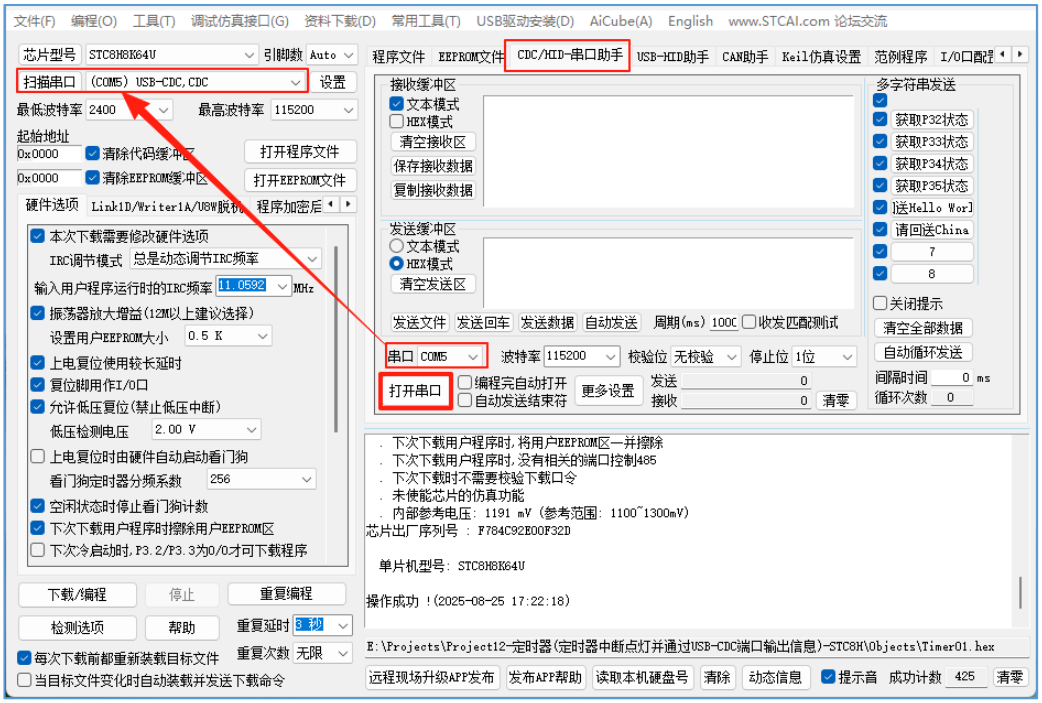
录入代码，保存，编译无误。连接 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤“打开程序文件”Timer01.hex（“打开程序文件”是项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中）并“下载/编程”，如下图：



这时在【AiCube-ISP】界面，我们看到串口选项框中已经出现了一个(COM5)USB-CDC,CDC 串口，表明现在可以通过这个 USB-CDC 端口进行数据通讯了。如果电脑系统并没有产生 CDC 端口，可能是因为系统版本较低（低于 Windows10 版本）。

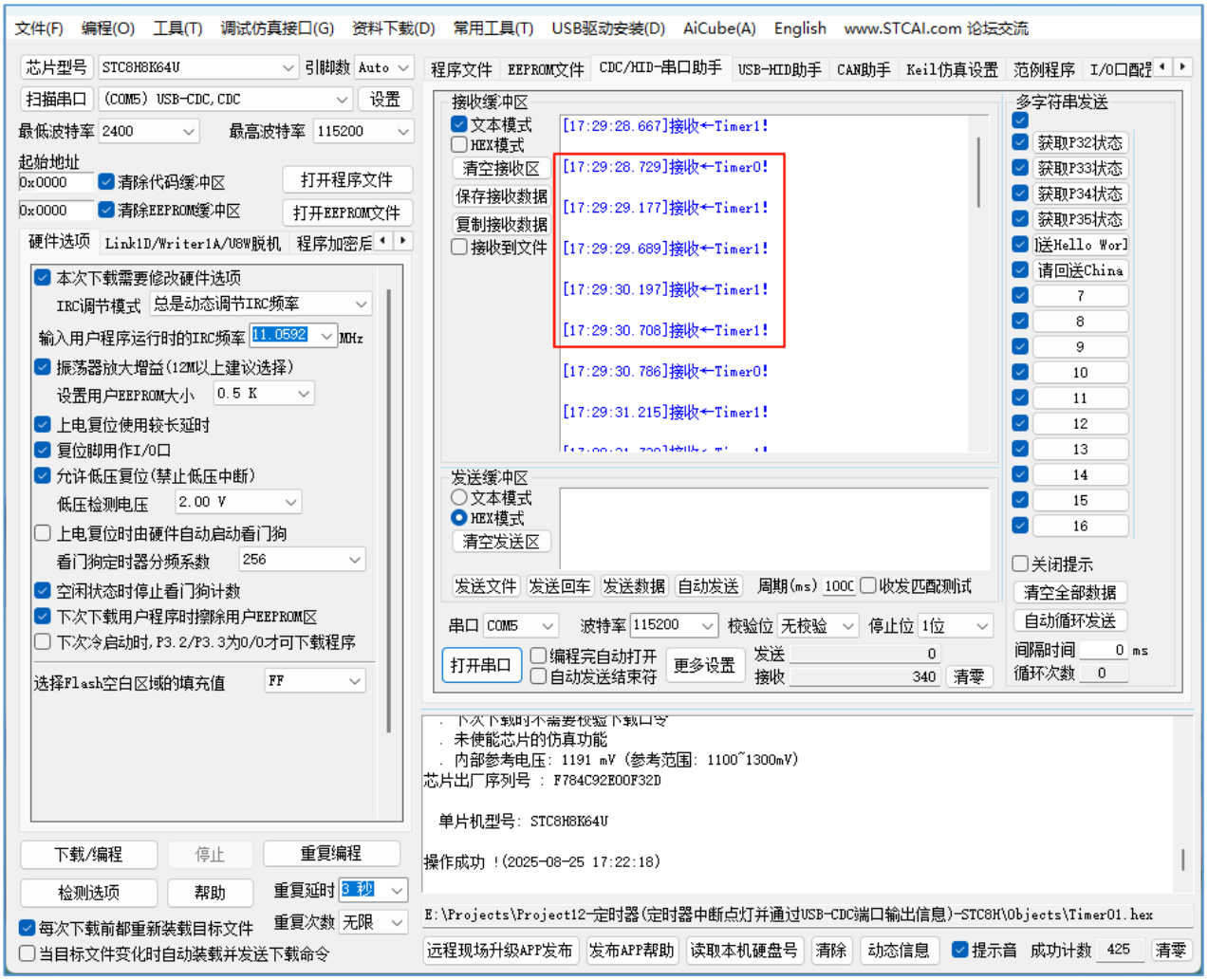
2.10.1.6　程序实现效果，观察实验现象

这时，在 AiCube-ISP 软件的“CDC/HID-串口助手”选项卡中找到“打开串口”按钮：



点击“打开串口”按钮。我们观察到：

➲ 在 AiCube-ISP 软件的接收缓冲区窗口，一直显示接收时间及接收的字符串内容“Time0”和“Time1”，并且如本实验代码中设置的每触发 4 次 Time1 中断，触发 1 次 Time0。如下图：



➲ **STC8H8K64U 实验箱上，LED67 灯每亮灭各 2 次，LED60 灯亮 1 次。**

达成实验预设，实验结束。

2.10.2　使用【AiCube】工具实现：定时器 0/1，16 位定时器实验（定时器中断点灯并通过 USB-CDC 端口输出信息）

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit**、**Ai8051U-8Bit**、**STC32G12K128**、**STC8H8K64U**、**STC15F2K60S2**、**STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

2.10.2.1　安装 AiCube-ISP 下载/编程/烧录/软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj



下载下来的是压缩包文件图标，如下图：



解压缩后的 AiCube-ISP 文件图标，如下图：



如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

2.10.2.2　使用【AiCube】创建项目

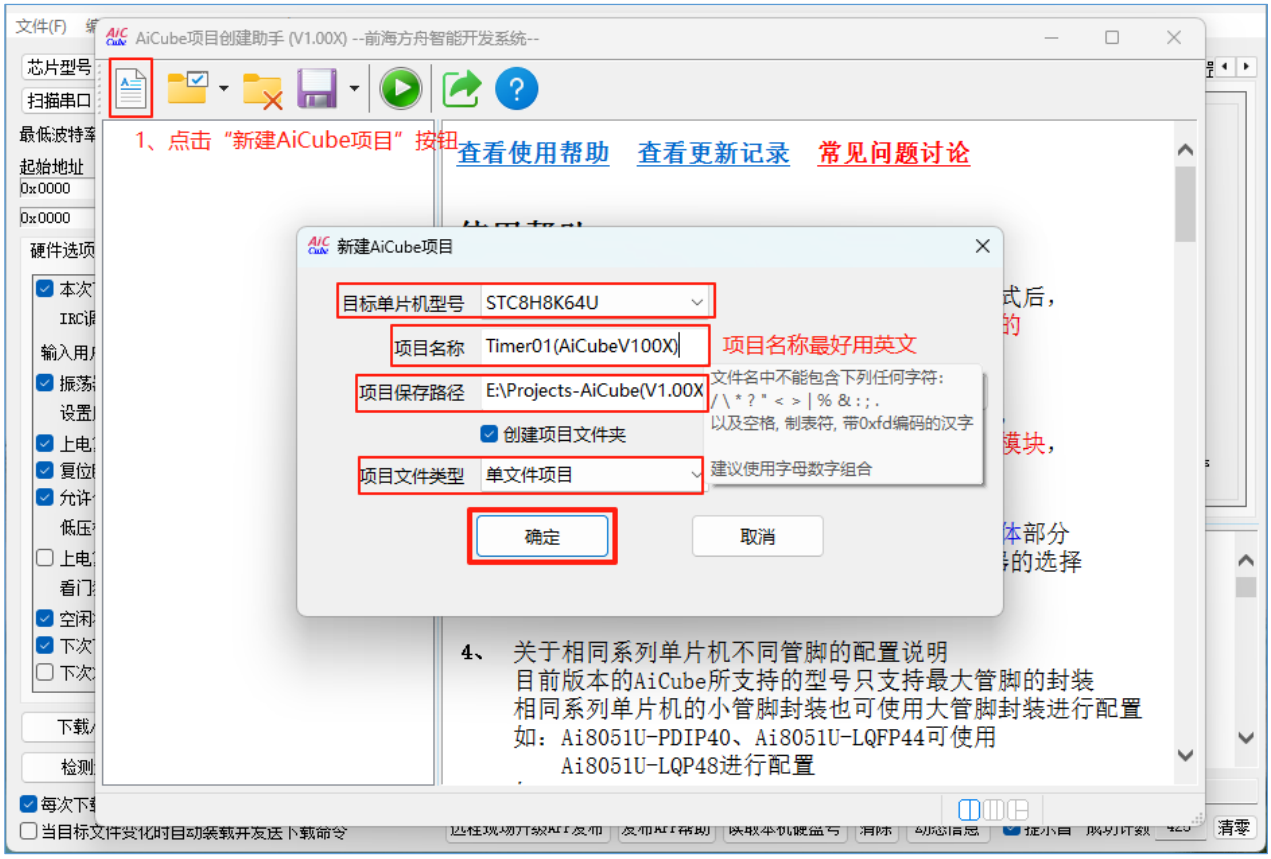
➲　双击图标，打开 AiCube-ISP 软件界面，如下图：



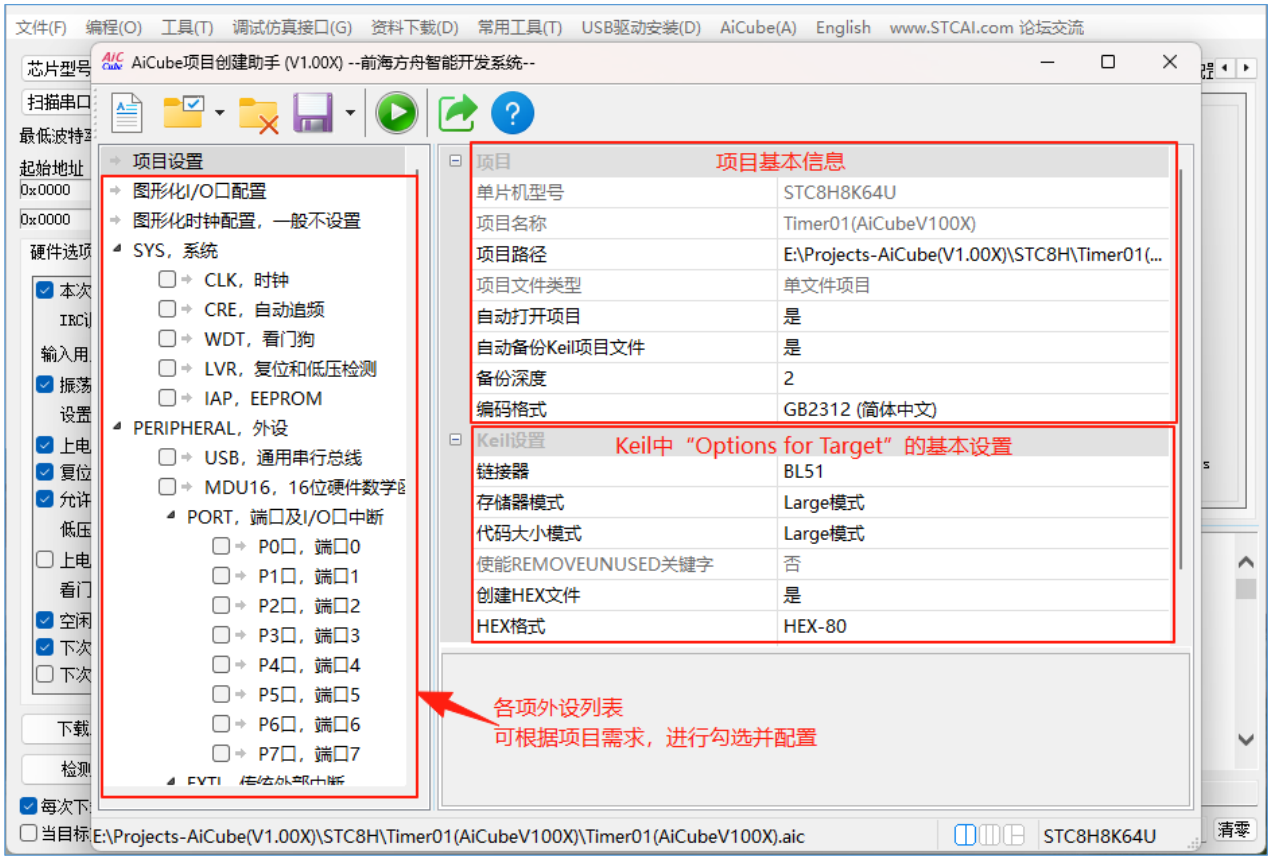
➲　点击【AiCube（A）】，弹出如下窗口：



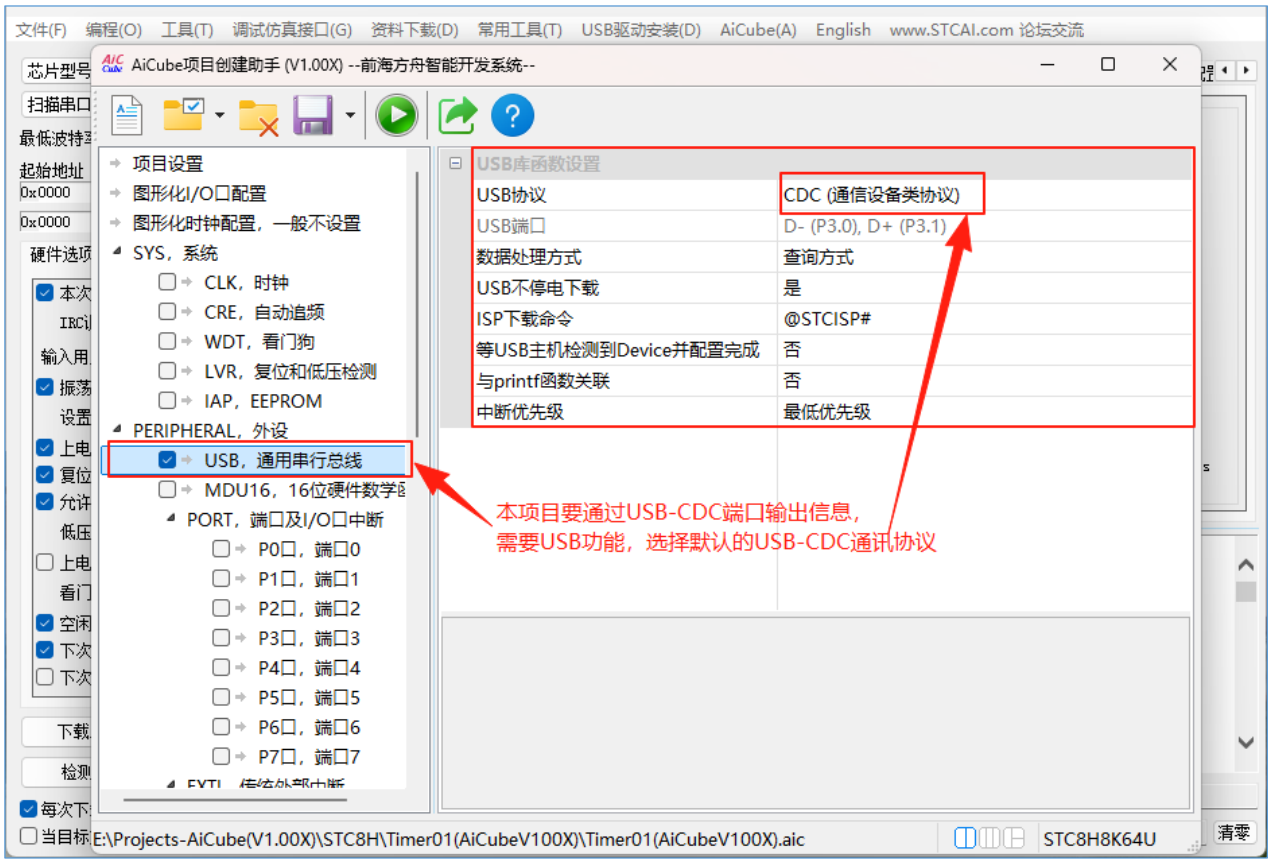
➲ **点击“新建 AiCube 项目”按钮**，选择并填写项目基本信息，建议项目名称用英文！



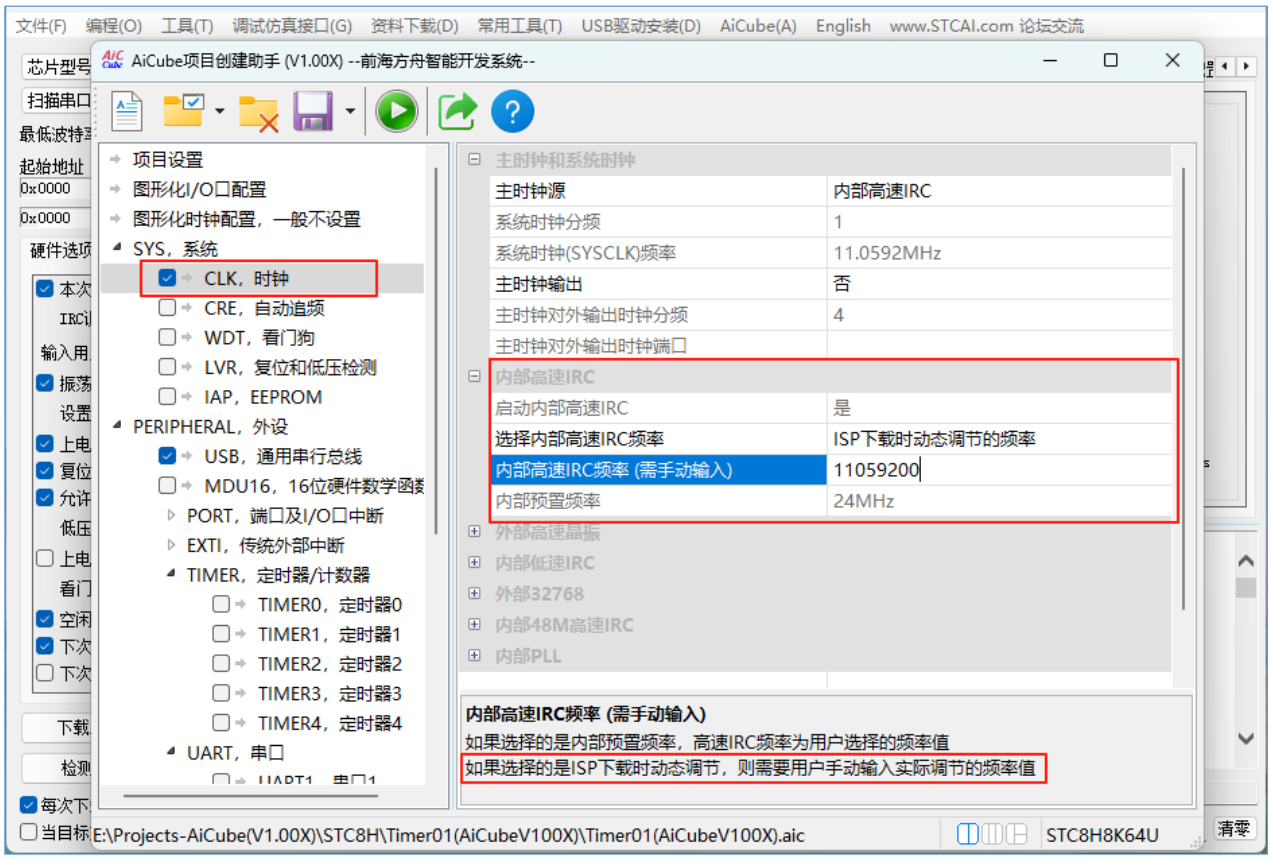
➲ **选择并填写项目基本信息**，建议项目名称用英文！**点击“确定”按钮**，出现如下配置界面：



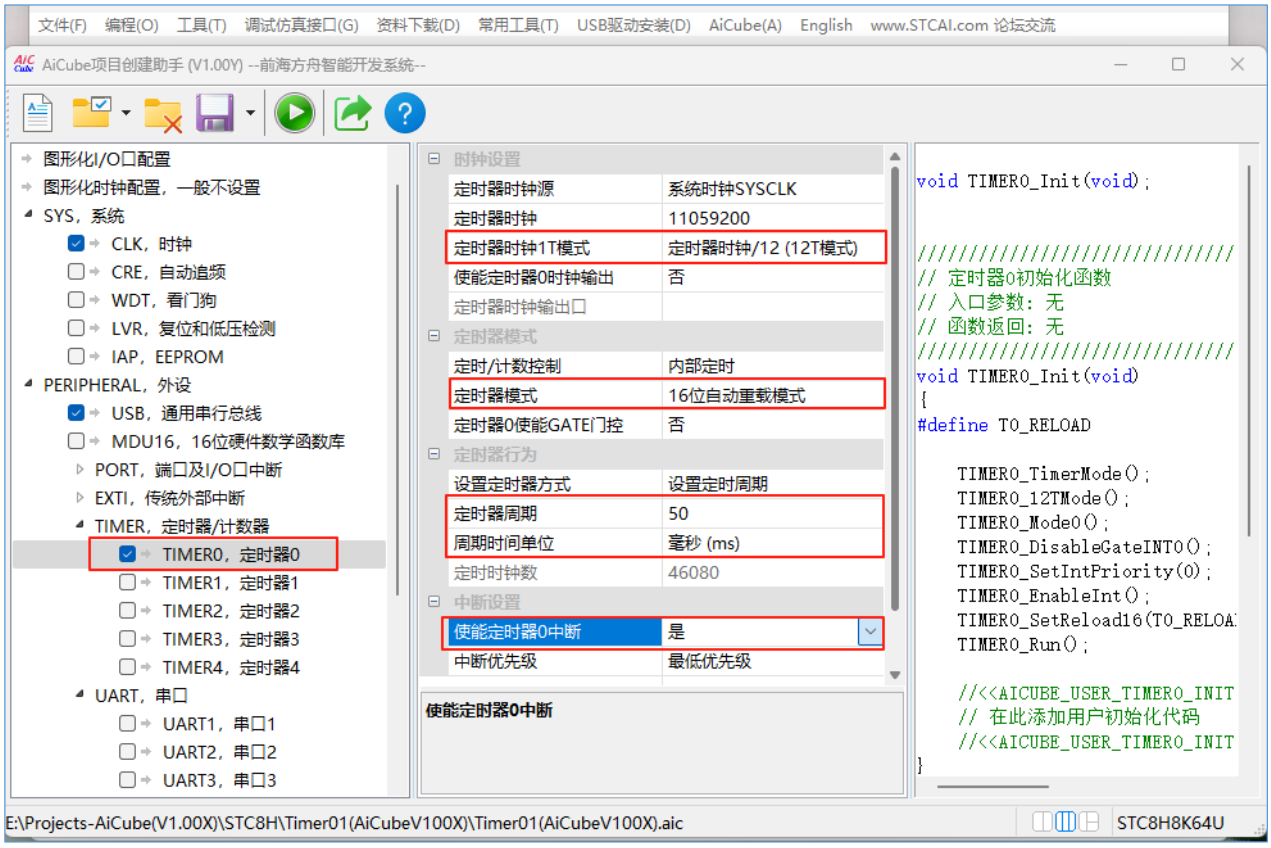
➲ 因本项目用到 USB 功能，要加载 USB 库函数、代码中要有 USB 初始化、USB-CDC 协议等，使用【AiCube】助手根据本项目的实际需求，**勾选“USB，通用串行总线”**：



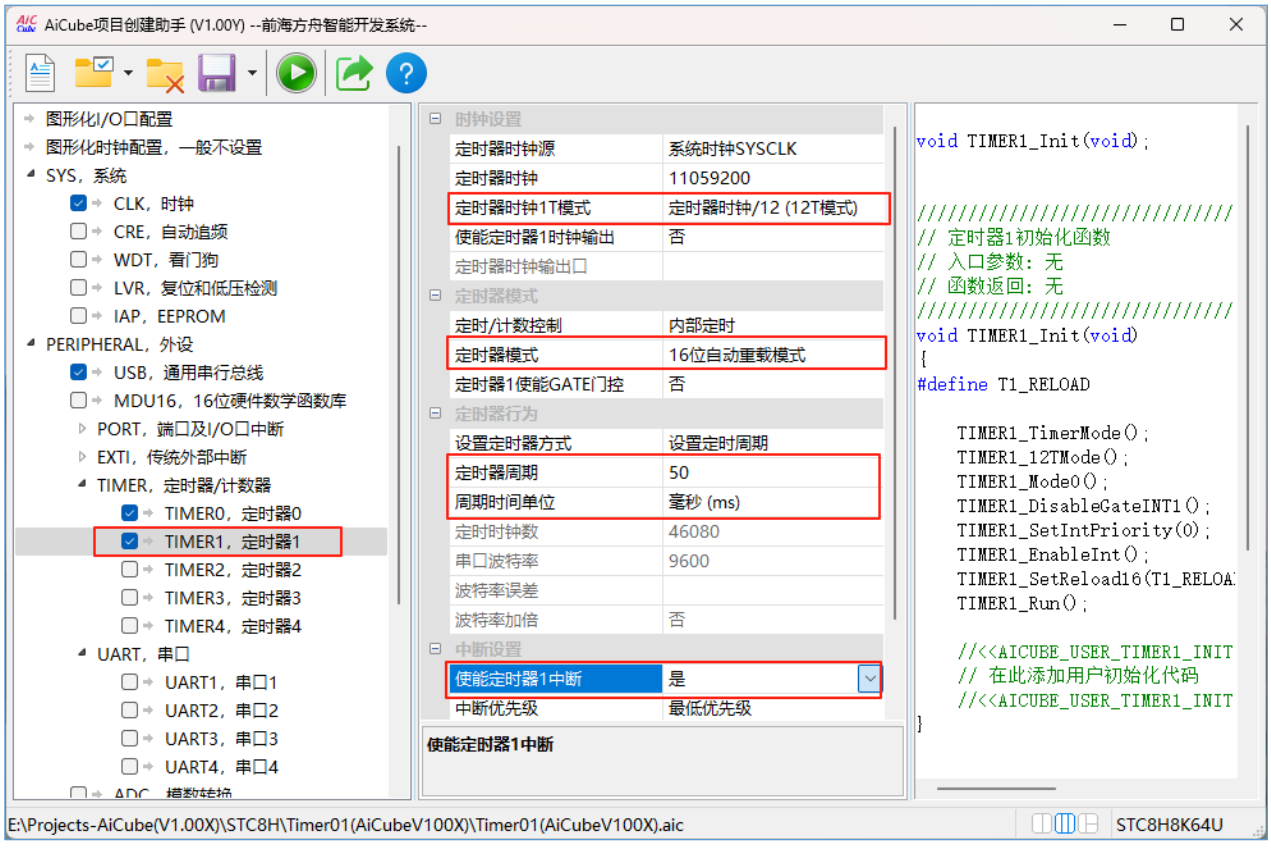
➲ 项目中需要实现定时器 0/1 分别定时 2s/500ms 产生中断，由于本实验使用的主频是 11.0592MHz，AiCube 默认是 24MHz，所以在配置定时器前需要**先勾选“时钟(CLK)”，进行时钟配置**：



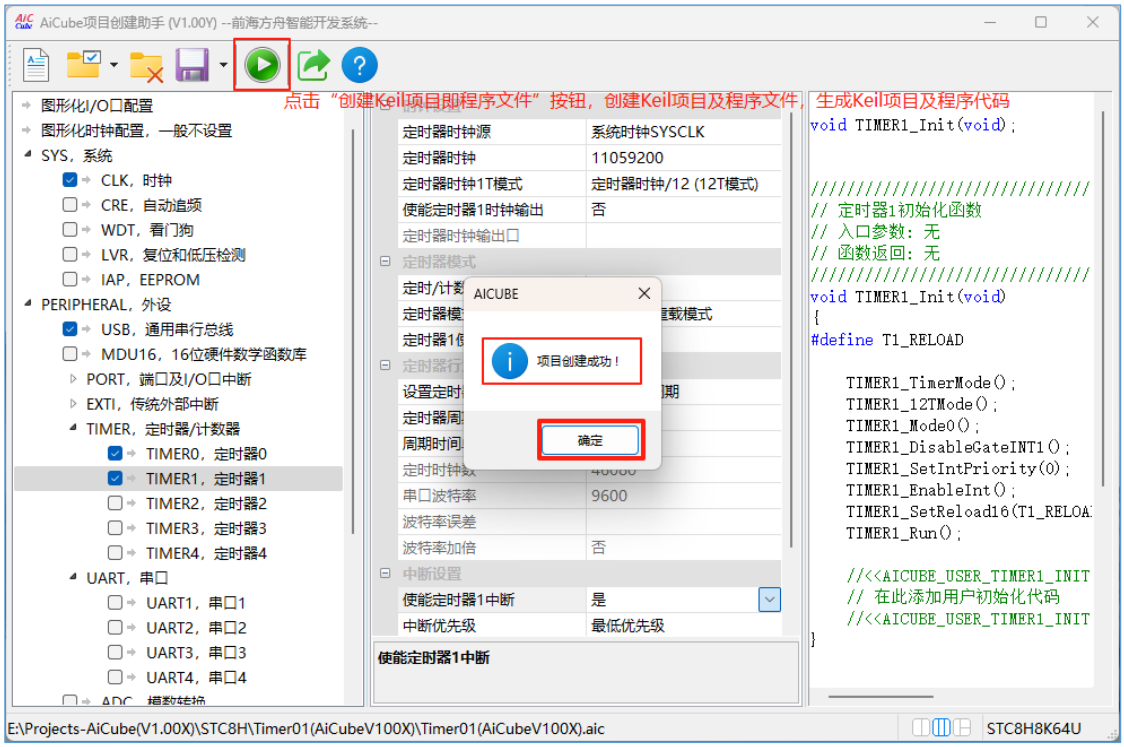
➲ 项目中需要实现【定时器 0（Timer0）定时 2s 产生中断，P60 灯闪烁，CDC 窗口显示“Timer0！”】，所以，需要**勾选“Timer0，定时器 0”，对定时器 0 进行配置**，如下图：



➲ 项目中需要实现【定时器 1（Timer1）定时 500ms 产生中断，P67 灯闪烁，CDC 窗口显示“Timer1！”】，所以，需要**勾选“Timer0，定时器 0”，对定时器 1 进行配置**：



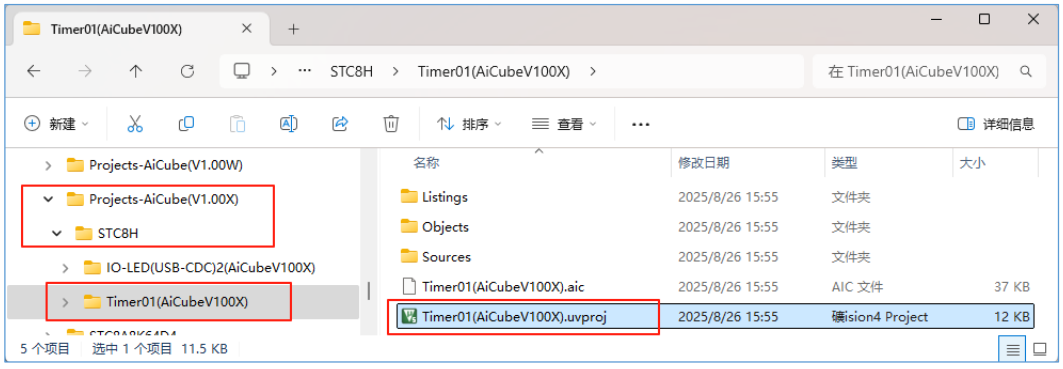
➲ 点击“创建 Keil 项目及程序文件”按钮，【AiCube】助手提示：“项目创建成功！”，如下图：



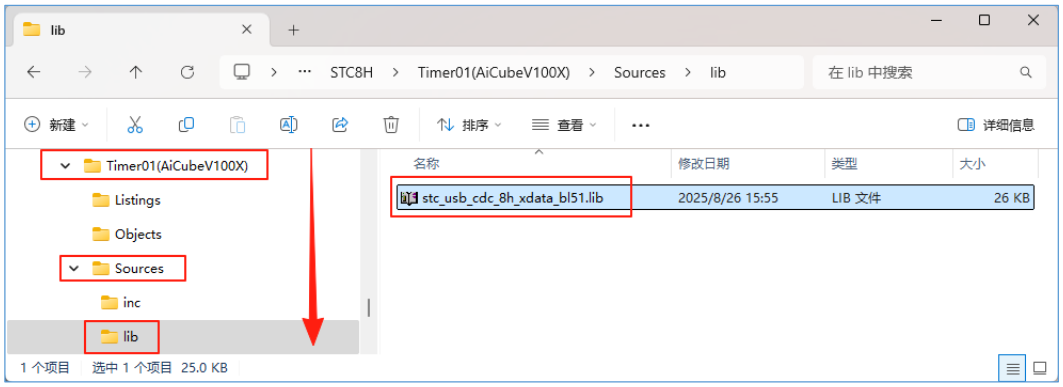
➲ **点击“确定”，立即自动打开 Keil 软件，进入该项目的 Keil 集成开发界面**，即【AiCube】成功帮我们创建了项目及文件目录“Time01(AiCubeV100X)”。**至此，可以关闭【AiCube】。**

➲ **找到创建的项目目录**

发现通过上述操作，【AiCube】助手已经帮我们建立了项目目录及 Time01(AiCubeV100X).uvproj 文件，如下图：

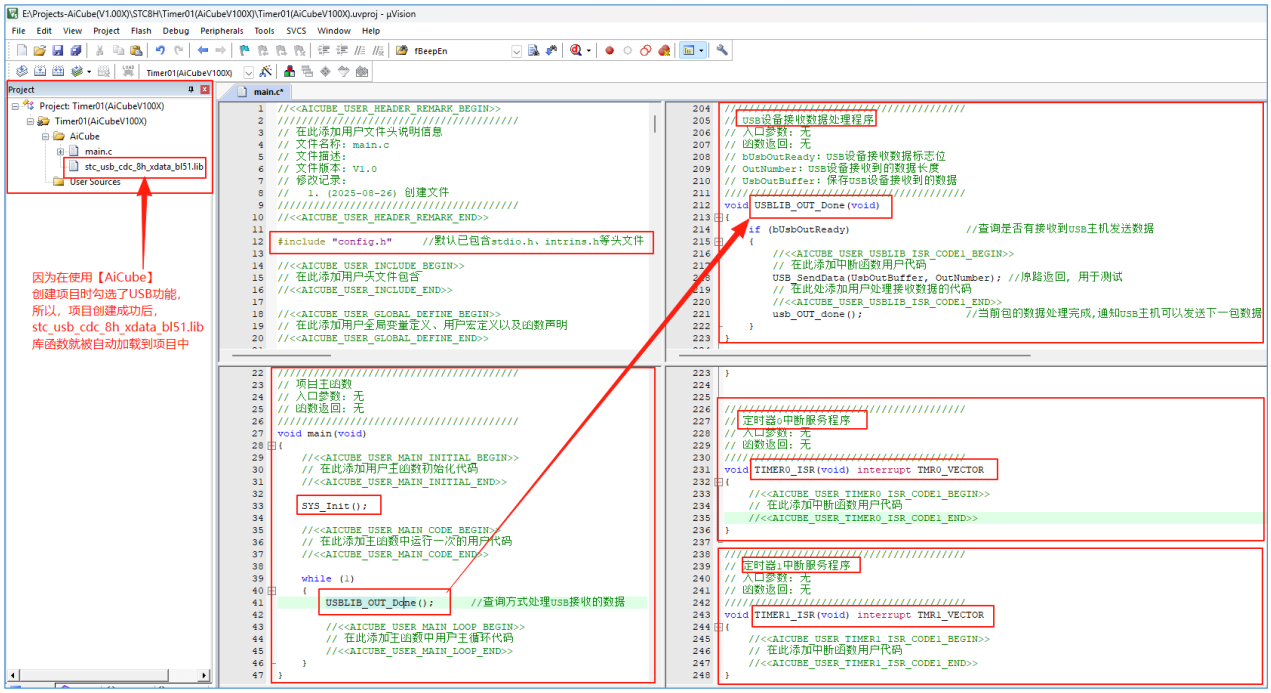


因为勾选了 USB 选项，打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经加载了：



➲ 观察已打开的 Keil 项目文件：Time01(AiCubeV100X).uvproj

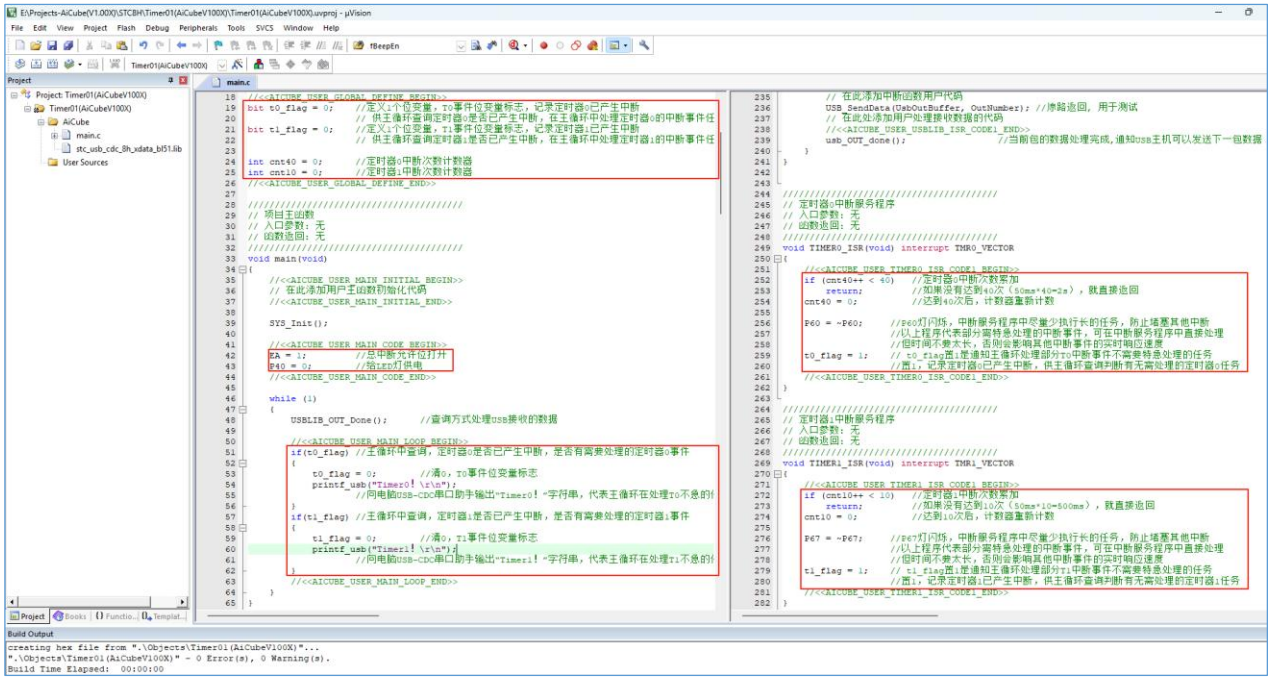
如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成 4 个窗口，全代码观察）：



根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c，并自动加载 USB 库函数（stc_usb_cdc_8h_xdata_bl51.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、毫秒和微秒延时函数、时钟初始化函数、定时器 0 和定时器 1 初始化函数、USB 库初始化函数、等待 USB 配置完成函数、USB 设备接收数据处理程序、定时器 0 和定时器 1 中断服务函数。

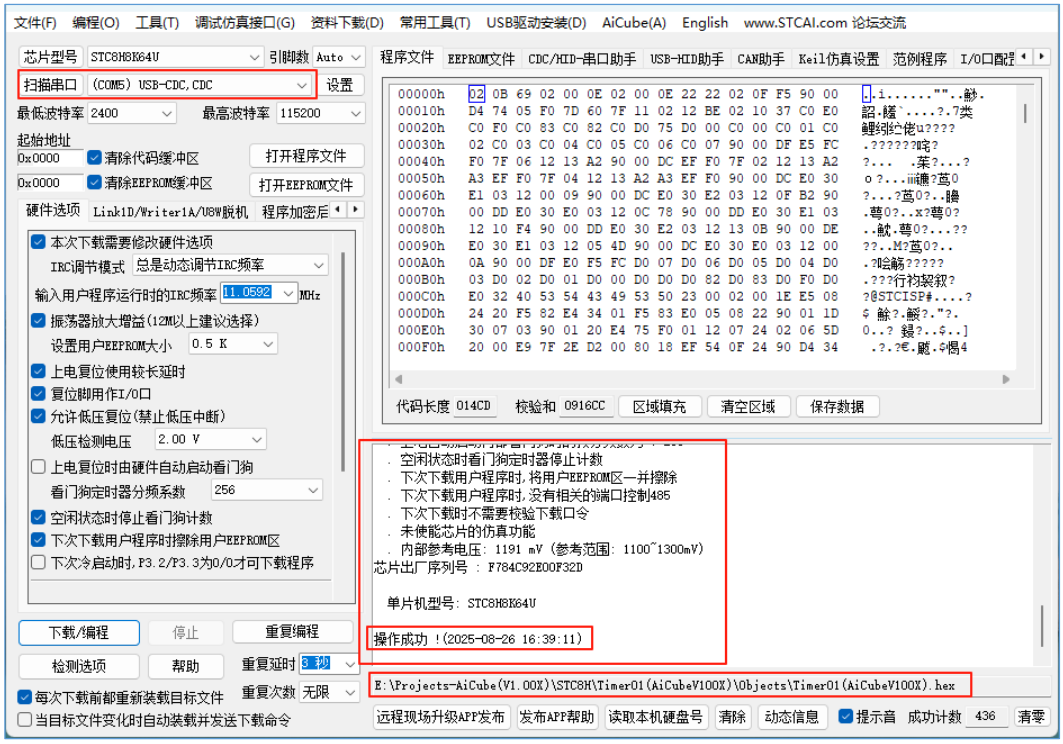
2.10.2.3　增加实验项目代码

【AiCube】助手帮我们搭建了程序框架，为了实现本实验效果，我们现在需要在主函数、定时器 Time0 中断服务函数、定时器 Time1 中断服务函数中补充相应代码，保存，编译，如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分出 2 个窗口，对比观察）：



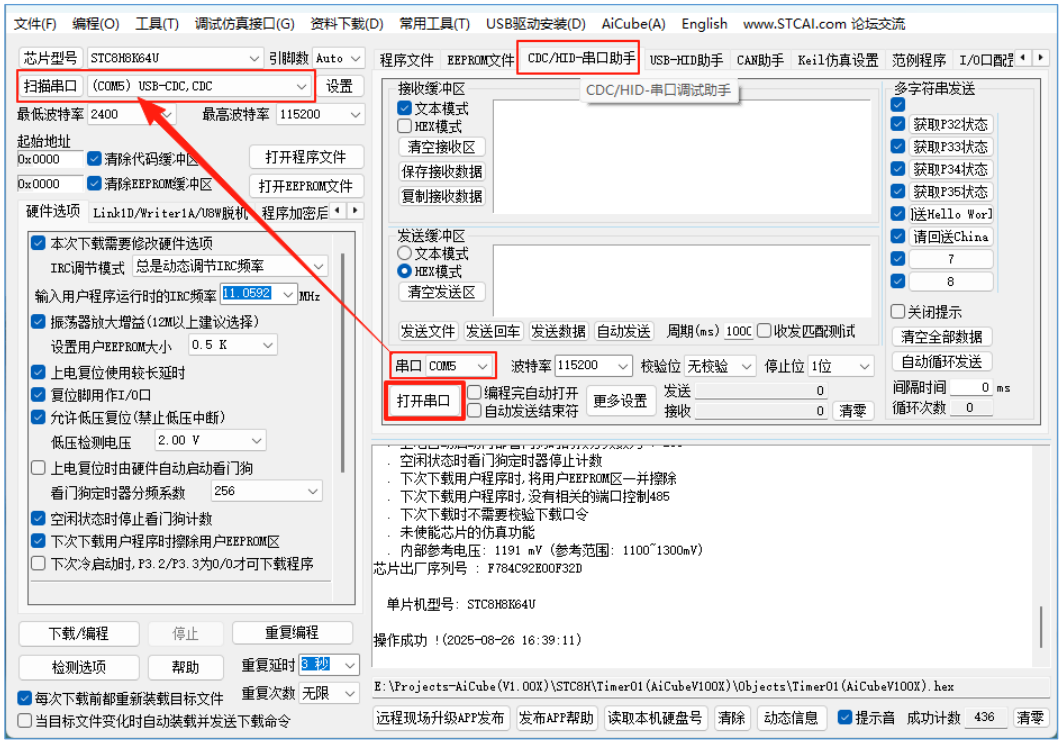
2.10.2.4　使用 STC8H8K64U 实验箱观察实验现象

连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“Time01(AiCube V100X).hex”（“打开程序文件”是打开项目程序文件编译无误后生成的 HEX 文件，一般默认存放在该项目的 Objects 目录中），如下图：



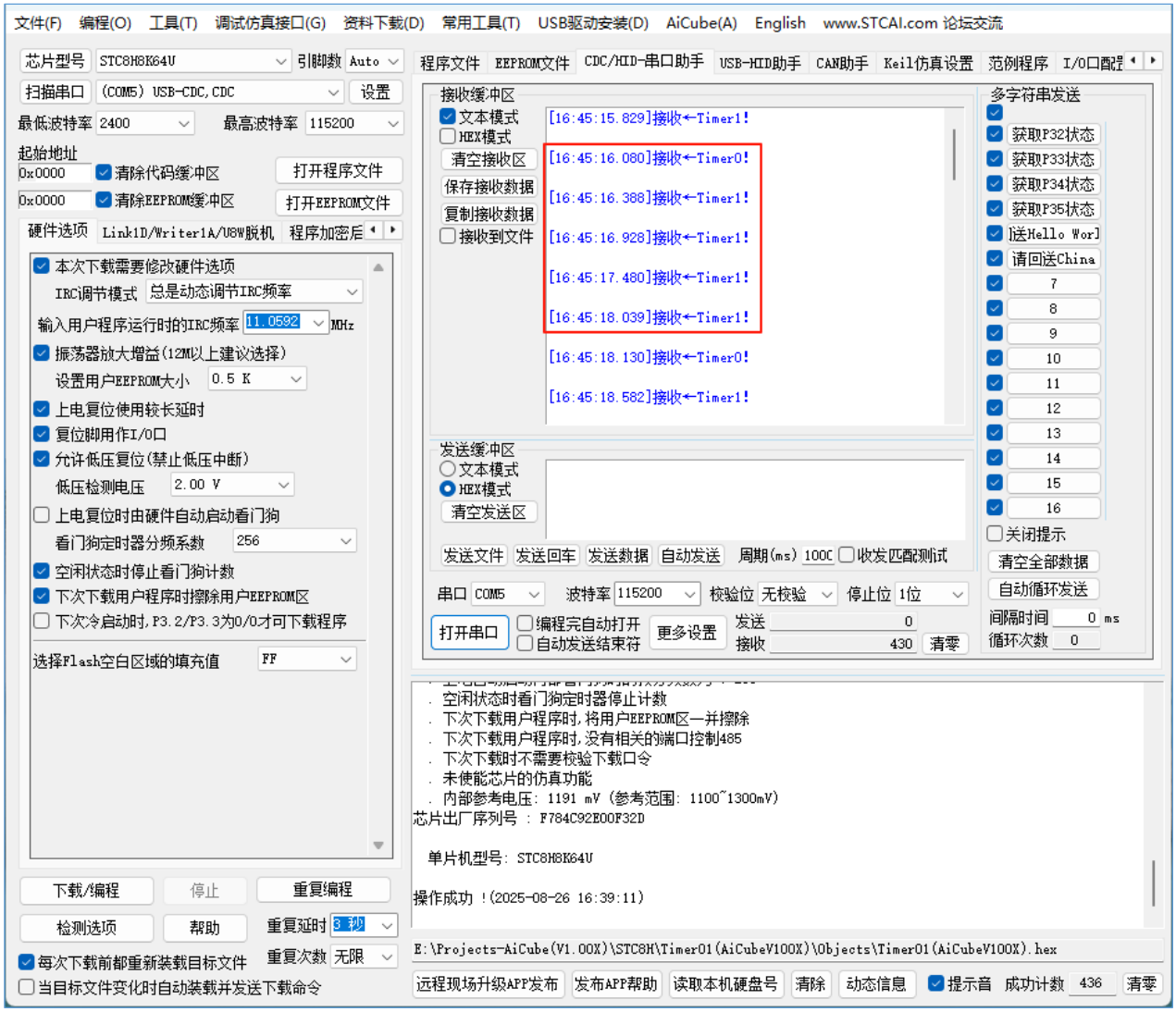
这时在【AiCube-ISP】界面，我们看到串口选项框中已经出现了一个(COM5)USB-CDC,CDC 串口，表明现在可以通过这个 USB-CDC 端口进行数据通讯了。如果电脑系统并没有产生 CDC 端口，可能是因为系统版本较低（低于 Windows10 版本）。

这时，在 AiCube-ISP 软件的“CDC/HID-串口助手”选项卡中找到“打开串口”按钮：



点击“打开串口”按钮。我们观察到：

➲ 在 AiCube-ISP 软件的接收缓冲区窗口，一直显示接收时间及接收的字符串内容“Time0”和“Time1”，并且如本实验代码中设置的每触发 4 次 Time1 中断，触发 1 次 Time0。如下图：



➲ **实验箱上，LED67 灯每亮灭各 2 次，LED60 灯亮 1 次。**

达成实验预设，实验结束。

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

2.10.3 定时器 0/1，INT0/INT1，中断组合应用实验，演示程序

```
//C 语言程序代码
#include <STC8H.H>          //包含 STC8H 的头文件

#include   "ai_usb.h"    //USB 库和 USB 虚拟设备的头文件，需要在项目文件中添加对应的 USB 库
           //由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字”@STCISP#”，
           //只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
           //即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。

#include "intrins.h"        //使用_nop_()函数所必须要包含的头文件,
                            //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                            //会导致编译器找不到这个而函数而报错。

bit   int0_flag = 0;  //定义 1 个位变量，INT0 事件位变量标志，记录 INT0 已产生中断
      // 供主循环查询 INT0 是否已产生中断，在主循环中处理 INT0 的中断事件任务，不堵塞其他中断
bit   int1_flag = 0;  //定义 1 个位变量，INT1 事件位变量标志，记录 INT1 已产生中断
      // 供主循环查询 INT1 是否已产生中断，在主循环中处理 INT1 的中断事件任务，不堵塞其他中断

bit   t0_flag = 0;    //定义 1 个位变量，T0 事件位变量标志，记录定时器 0 已产生中断
      // 供主循环查询定时器 0 是否已产生中断，在主循环中处理定时器 0 的中断事件任务，不堵塞其他中断
bit   t1_flag = 0;    //定义 1 个位变量，T1 事件位变量标志，记录定时器 1 已产生中断
      // 供主循环查询定时器 1 是否已产生中断，在主循环中处理定时器 1 的中断事件任务，不堵塞其他中断

void Timer0_Init(void)        //定时器 0 初始化，50ms@11.0592MHz
{
    AUXR &= 0x7F;             //定时器时钟 12T 模式
    TMOD &= 0xF0;             //设置定时器模式
    TL0 = 0x00;               //设置定时初始值
    TH0 = 0x4C;               //设置定时初始值
    TF0 = 0;                  //清除 TF0 标志
    TR0 = 1;                  //定时器 0 开始计时
    ET0 = 1;                  //使能定时器 0 中断
}

void Timer1_Init(void)        //定时器 1 初始化，50ms@11.0592MHz
{
    AUXR &= 0xBF;             //定时器时钟 12T 模式
    TMOD &= 0x0F;             //设置定时器模式
    TL1 = 0x00;               //设置定时初始值
    TH1 = 0x4C;               //设置定时初始值
    TF1 = 0;                  //清除 TF1 标志
    TR1 = 1;                  //定时器 1 开始计时
    ET1 = 1;                  //使能定时器 1 中断
}
```

```
void main (void)
{
    P_SW2 |= 0x80;          //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;     //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;     //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //设置 P2 口为准双向口模式
//  P3M0 = 0x00; P3M1 = 0x00;     //设置 P3 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x0c;     //P32、P33 设置为高阻输入(需要同步开启上拉电阻)
    P4M0 = 0x00; P4M1 = 0x00;     //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;     //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;     //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;     //设置 P7 口为准双向口模式

    P3PU = 0x0c;      //P32、P33 打开上拉电阻

    IT0 = 0;          //使能 INT0 上升沿和下降沿中断
//  IT0 = 1;          //使能 INT0 下降沿中断
    EX0 = 1;          //使能 INT0 中断

//  IT1 = 0;          //使能 INT1 上升沿和下降沿中断
    IT1 = 1;          //使能 INT1 下降沿中断
    EX1 = 1;          //使能 INT1 中断

    IE0 = 0;          //清 INT0 中断标志
    IE1 = 0;          //清 INT1 中断标志

    int0_flag = 0;    //初始化用户标志位
    int1_flag = 0;    //初始化用户标志位

    Timer0_Init();    //调用定时器 0 初始化函数
    Timer1_Init();    //调用定时器 1 初始化函数

    usb_init();     //调用 USB 初始化函数，不需要立即判断电脑已正确识别到 USB 从设备
    /* USB 型单片机从设备，如需要主动向电脑发送数据，
       在执行 USB_SendData( )函数和 printf_usb( )函数时，
       这两个函数已增加了判断电脑是否已正确识别到 USB 从设备的程序。
       如果电脑要主动发送数据给 USB 从设备，电脑自己会主动判断与 USB 从设备是否已正确连接。
    */

    EA = 1;           //总中断允许位打开
    P40 = 0;          //打开 LED 灯供电
    while(1)      //主循环中查询需要处理的各种事件，如 T0/T1 中断事件，并打印输出状态
    {   /*  本演示程序中，主循环查询各中断有无需要继续处理的事件的次序，
            依次是 INT0/INT1/T0/T1, 用户可以自己根据实际情况，
            调整查询各中断有无需要继续处理的事件的优先次序  */
```

```
        if(int0_flag)   //主循环中查询，INT0 是否已产生中断，是否有需要处理的 INT0 事件
        {
            int0_flag = 0;      //清 0，INT0 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(int1_flag)   //主循环中查询，INT1 是否已产生中断，是否有需要处理的 INT1 事件
        {
            int1_flag = 0;      //清 0，INT1 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(t0_flag)     //主循环中查询，定时器 0 是否已产生中断，是否有需要处理的定时器 0 事件
        {
            t0_flag = 0;        //清 0，T0 事件位变量标志
            printf_usb("Timer0！\r\n");
            //向电脑 USB-CDC 串口助手输出“Timer0!”字符串，代表主循环在处理 T0 不急的任务
        }

        if(t1_flag)     //主循环中查询，定时器 1 是否已产生中断，是否有需要处理的定时器 1 事件
        {
            t1_flag = 0;        //清 0，T1 事件位变量标志
            printf_usb("Timer1！\r\n");
            //向电脑 USB-CDC 串口助手输出“Timer1!”字符串，代表主循环在处理 T1 不急的任务
        }
    }
}

void int0_isr(void) interrupt INT0_VECTOR
{
    int0_flag = 1;  // int0_flag 置 1 是通知主循环处理部分 INT0 中断事件不需要特急处理的任务
                    //置 1，记录 INT0 已产生中断，供主循环查询判断有无需处理的 INT0 任务

    if(INT0)                //边沿中断，进入后再次判断电平从而判断是什么样的电平
    {
    //  P61 = 0;            //判断为高电平，则当前为上升沿，点亮 P61 端口上的 LED 灯
    //  _nop_();            //可以在这里插入断点进行观察现象
    //  P61 = 1;            //关闭 LED 灯
        P61 = ~P61;         //如果用外部按键触发 INT0，则需另外添加去抖动处理
    }
    else
    {
    //  P66 = 0;            //判断为低电平，则当前为下降沿，点亮 P66 端口上的 LED 灯
```

```
    //  _nop_();            //可以在这里插入断点进行观察现象
    //  P66 = 1;            //关闭 LED 灯
        P66 = ~P66;         //如果用外部按键触发 INT0，则需另外添加去抖动处理
    }
}
//INT0 中断服务程序，INT0_VECTOR 在 STC8H.H 头文件中已宏定义为 0

void int1_isr(void) interrupt INT1_VECTOR
{
    int1_flag = 1;  // int1_flag 置 1 是通知主循环处理部分 INT1 中断事件不需要特急处理的任务
                    //置 1，记录 INT1 已产生中断，供主循环查询判断有无需处理的 INT1 任务

//  P62= 0;             //点亮 P62 端口上的 LED 灯
//  _nop_();            //可以在这里插入断点观察现象
//  P62 = 1;            //关闭 LED 灯
    P62 = ~P62;         //如果用外部按键触发 INT1，则需另外添加去抖动处理

}
//INT1 中断服务程序，INT1_VECTOR 在 STC8H.H 头文件中已宏定义为 2

void Timer0_Isr(void) interrupt TMR0_VECTOR      //定时器 0 中断服务程序
{
    P60 = ~P60;   //P60 灯闪烁，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                  //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                  //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t0_flag = 1;  // t0_flag 置 1 是通知主循环处理部分 T0 中断事件不需要特急处理的任务
                  //置 1，记录定时器 0 已产生中断，供主循环查询判断有无需处理的定时器 0 任务
}
//定时器 0 中断服务程序，TMR0_VECTOR 在 STC8H.H 头文件中已宏定义为 1

void Timer1_Isr(void) interrupt TMR1_VECTOR
{
    P67 = ~P67;   //P67 灯闪烁，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                  //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                  //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t1_flag = 1;  // t1_flag 置 1 是通知主循环处理部分 T1 中断事件不需要特急处理的任务
                  //置 1，记录定时器 1 已产生中断，供主循环查询判断有无需处理的定时器 1 任务
}
//定时器 1 中断服务程序，TMR1_VECTOR 在 STC8H.H 头文件中已宏定义为 3
```

2.11　大型综合实战 -- 课程设计：时钟+数码管+矩阵按键+蜂鸣器发声

2.11.1　实验介绍

1、熟悉矩阵式按键扫描方法
2、了解使用 P6、P7 口控制 LED 数码管
3、软件模拟 RTC，实现 RTC 的时、分、秒控制和设置
4、I/O 控制有源蜂鸣器发声
5、扩展部分：如何实现长按键
6、熟悉 STC8H8K64U 实验箱布局及电路图
7、熟练使用 AiCube-ISP 系统软件
8、熟悉如何管理多文件项目

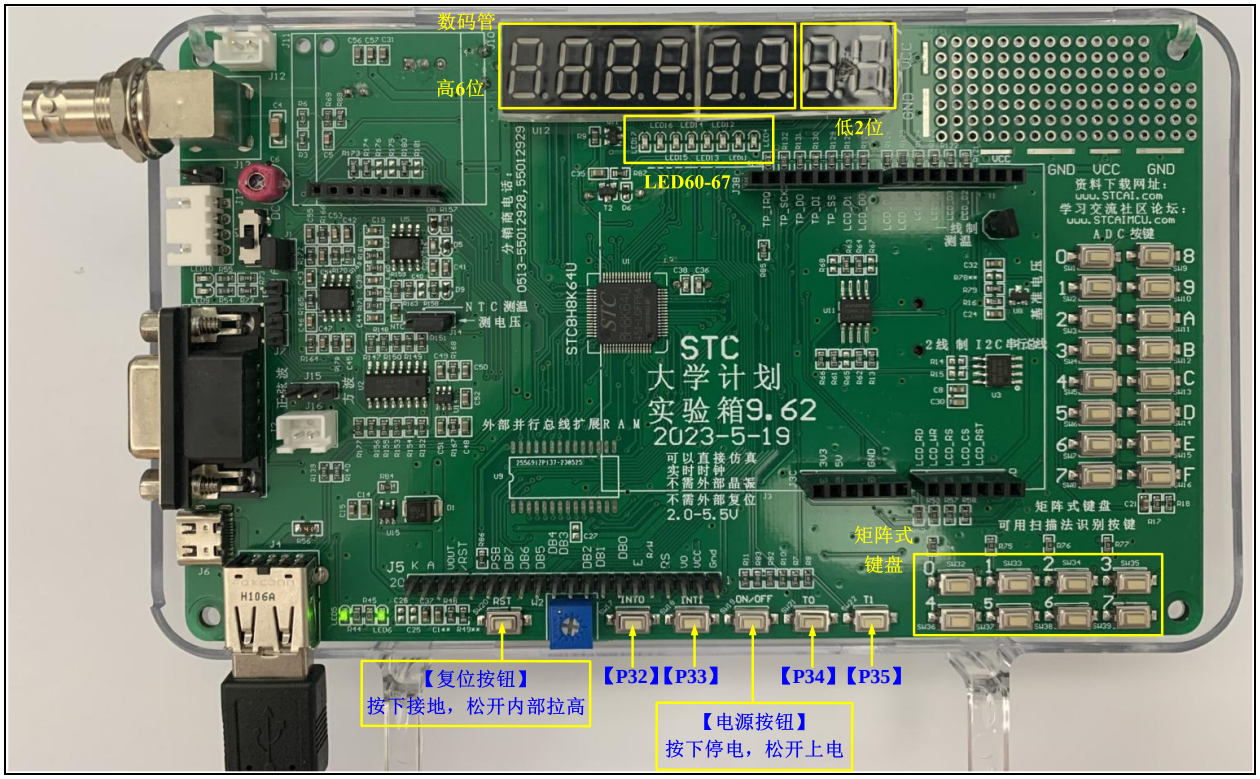
2.11.1.1　按键扫描说明

如按键要判断为正常按下，而不是抖动，需要该按键保持按下持续保持为低状态时间是 20ms ~ 50ms；主循环中有个 1ms 的时间基准，将按键扫描程序作为 1ms 事件，按键状态维持 20ms 不变，即可对按键去抖动，得到稳定的键码。

2.11.1.2　实验现象和操作说明

将本项目的 HEX 文件下载到 STC8H8K64U 的实验箱后
1、在 LED 数码管上面高 6 位会动态显示当前的 RTC 时间
2、按下两行 4 列的 8 个矩阵按键后：
- 在 LED 数码管的最低位会显示当前的按键键码
- 同时按键会发出按键音

（特别注意：STC8H8K64U 实验箱上的蜂鸣器是有源蜂鸣器）



3、矩阵按键中的按键 0，是 RTC 模式设置键

- 上电后 RTC 为正常工作模式，此时数码管动态显示 RTC 时间
- 按一次按键 0，RTC 会切换为小时设置模式，此时 RTC 的小时值会 0.5 秒闪烁一次
- 再按一次按键 0，RTC 会切换为分钟设置模式，此时 RTC 的分钟值会 0.5 秒闪烁一次
- 再按一次按键 0，RTC 会切换为秒钟设置模式，此时 RTC 的秒钟值会 0.5 秒闪烁一次
- 在按一次按键 0，恢复为 RTC 正常工作模式

4、矩阵按键中的按键 4 和 5 分别是上调键和下调键

- 只有在 RTC 的时/分/秒设置模式时，上调键和下调键才有效

5、时钟设置模式时，长按按键 4 和 5，可实现快速设置功能

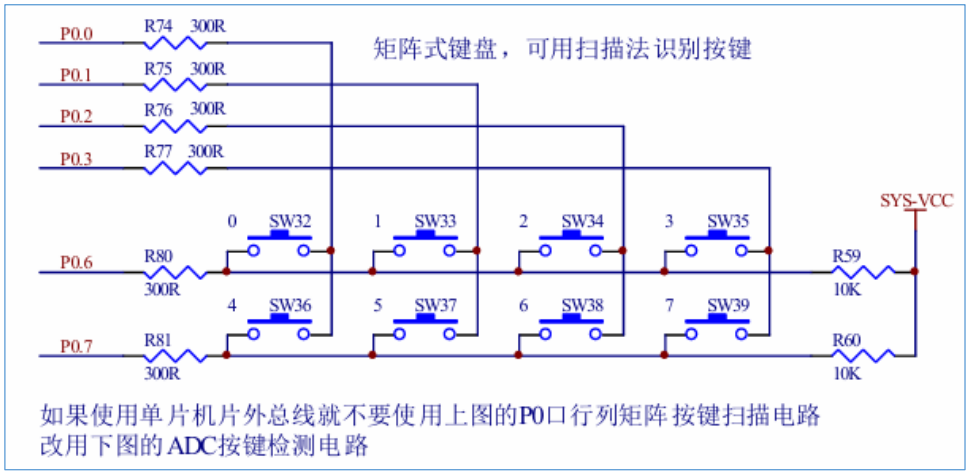
2.11.1.3 Keil 环境下多文件项目管理说明

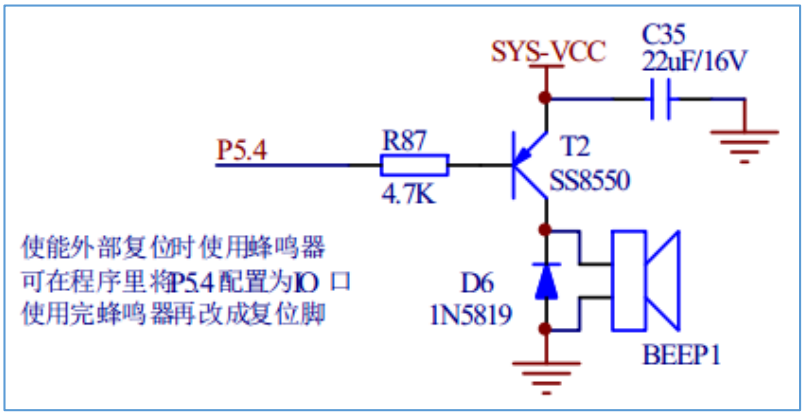
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以**方便分工合作、代码复用、模块化管理、增强可读性和可维护性**。

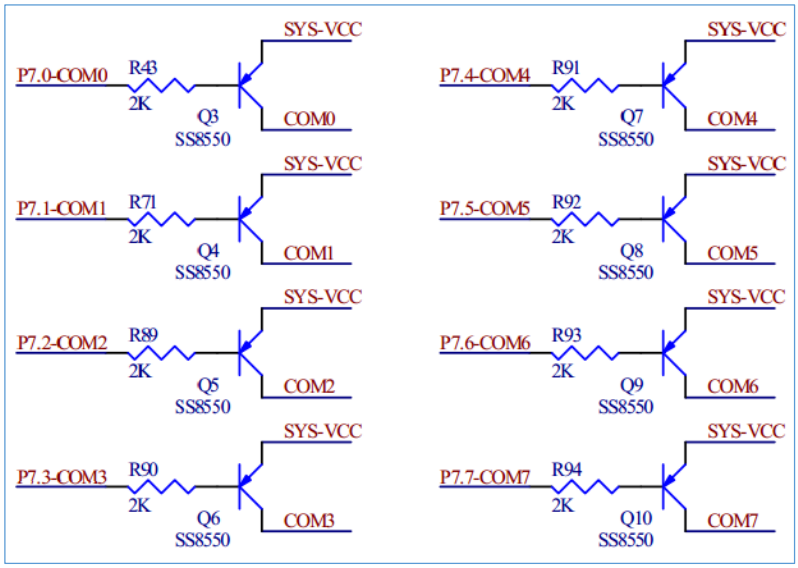
- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

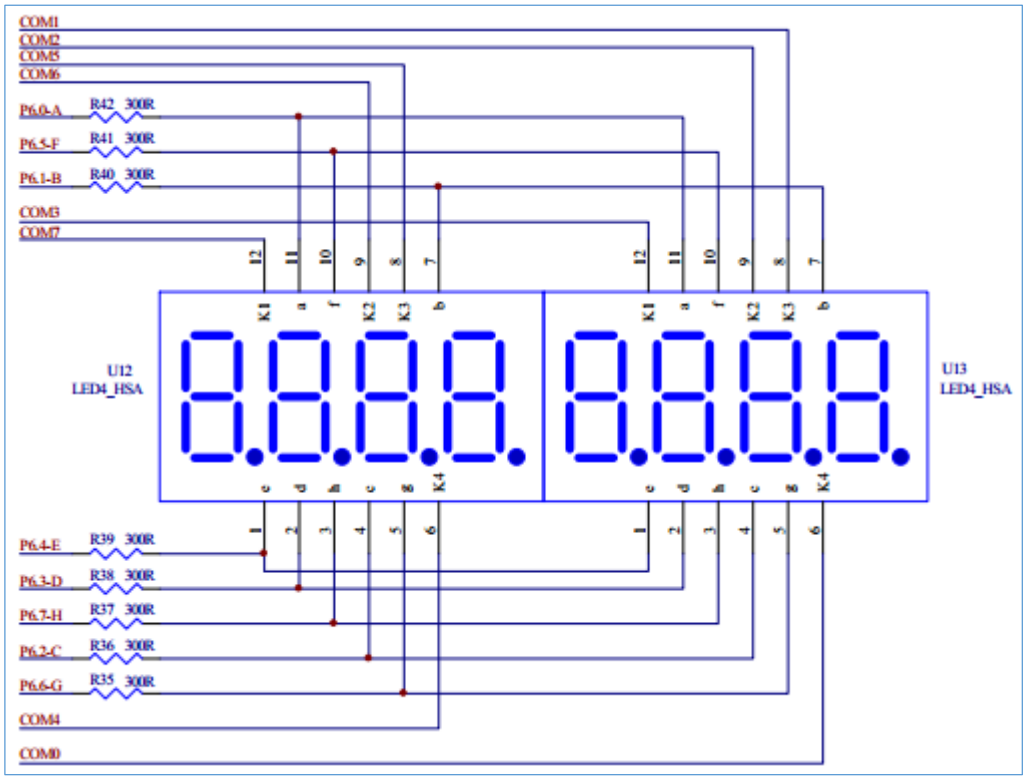
```
#ifndef XXXX
#define XXXX
...
#endif
```

2.11.2 原理图



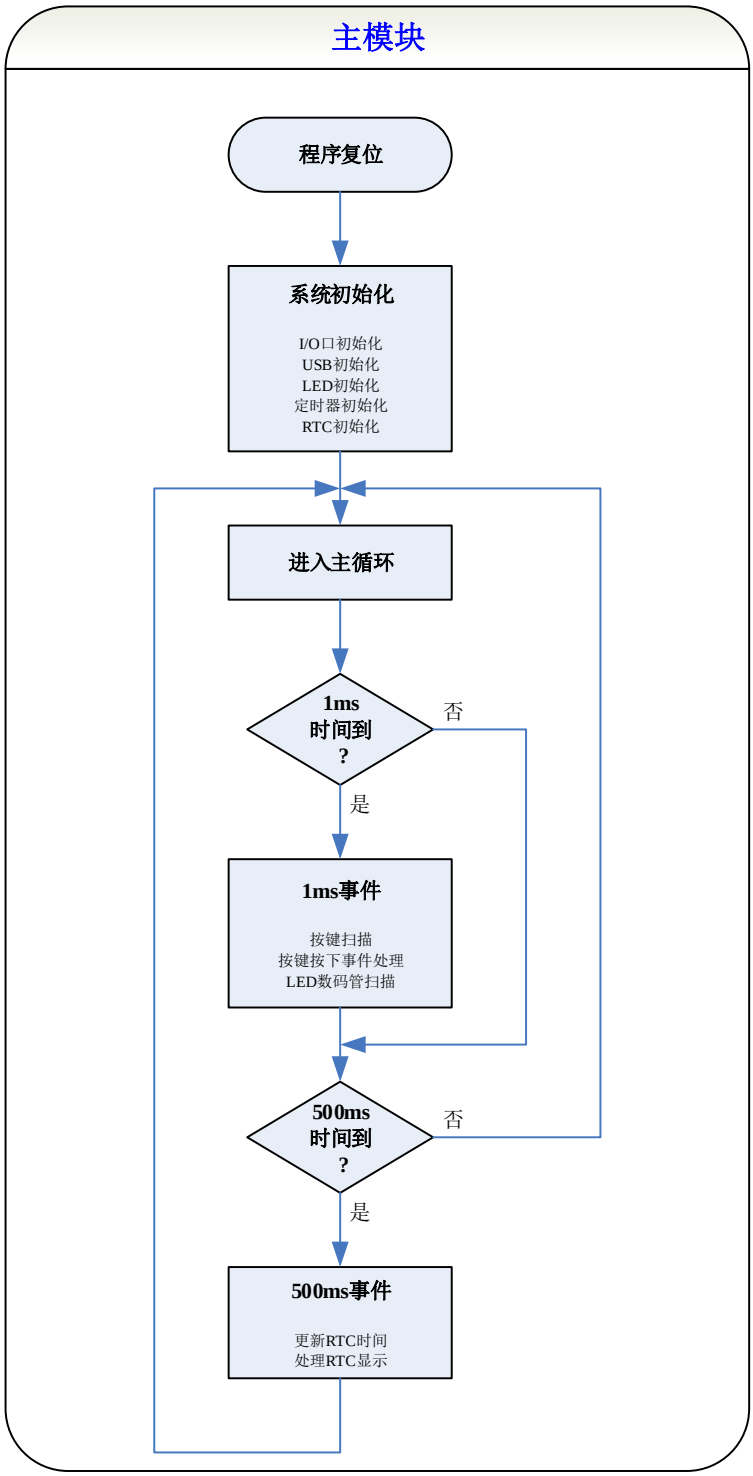

SYS-VCC
C35
22uF/16V
P5.4
R87
4.7K
T2
SS8550
使能外部复位时使用蜂鸣器
可在程序里将P5.4配置为IO 口
使用完蜂鸣器再改成复位脚
D6
1N5819
BEEP1


P7.0-COM0
R43
2K
Q3
SS8550
SYS-VCC
COM0
P7.1-COM1
R71
2K
Q4
SS8550
COM1
P7.2-COM2
R89
2K
Q5
SS8550
COM2
P7.3-COM3
R90
2K
Q6
SS8550
COM3
P7.4-COM4
R91
2K
Q7
SS8550
COM4
P7.5-COM5
R92
2K
Q8
SS8550
COM5
P7.6-COM6
R93
2K
Q9
SS8550
COM6
P7.7-COM7
R94
2K
Q10
SS8550
COM7

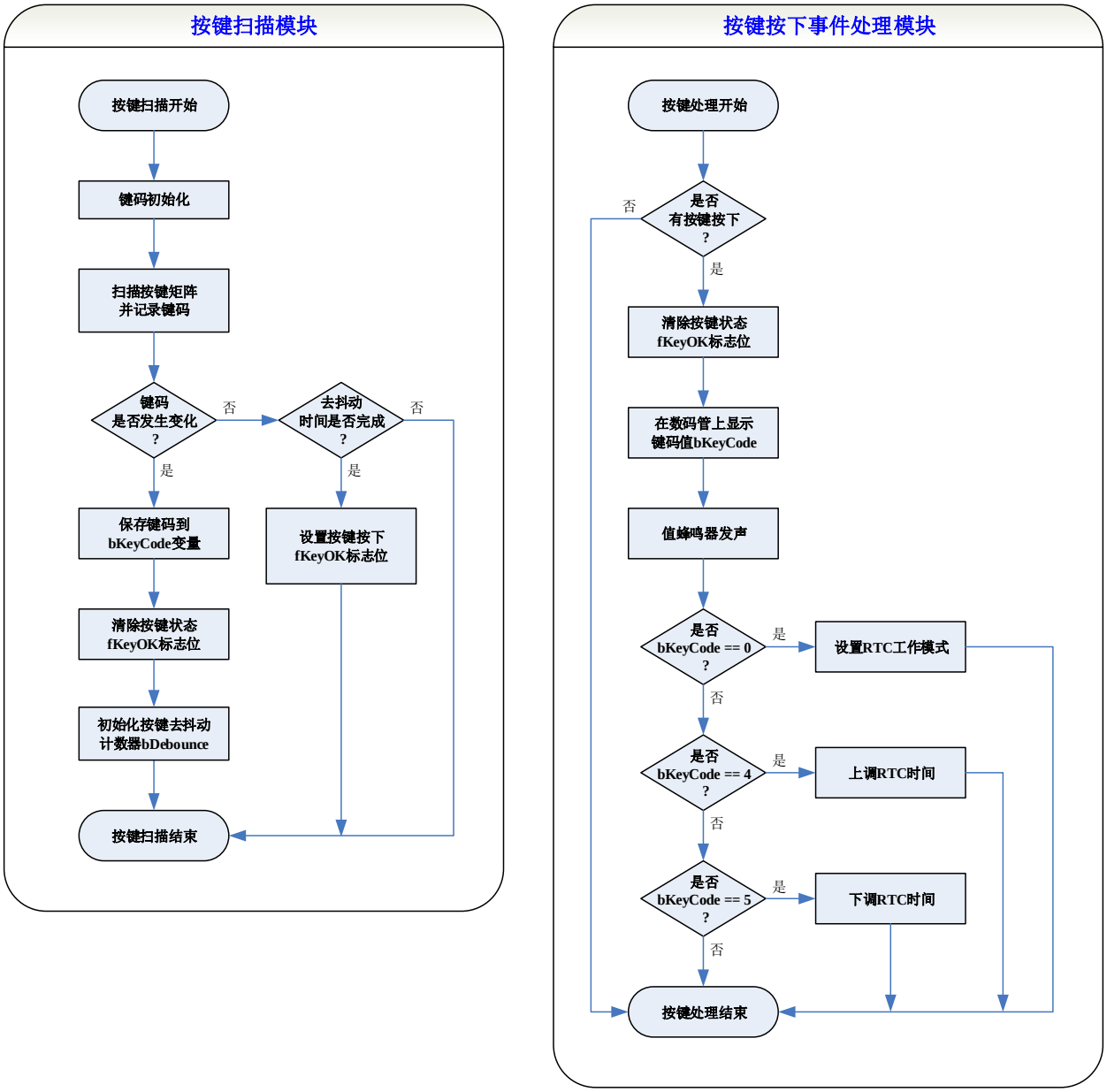

COM1
COM2
COM5
COM6
P6.0-A
R42 300R
P6.5-F
R41 300R
P6.1-B
R40 300R
COM3
COM7
U12
LED4_HSA
U13
LED4_HSA
P6.4-E
R39 300R
P6.3-D
R38 300R
P6.7-H
R37 300R
P6.2-C
R36 300R
P6.6-G
R35 300R
COM4
COM0

2.11.3 流程图

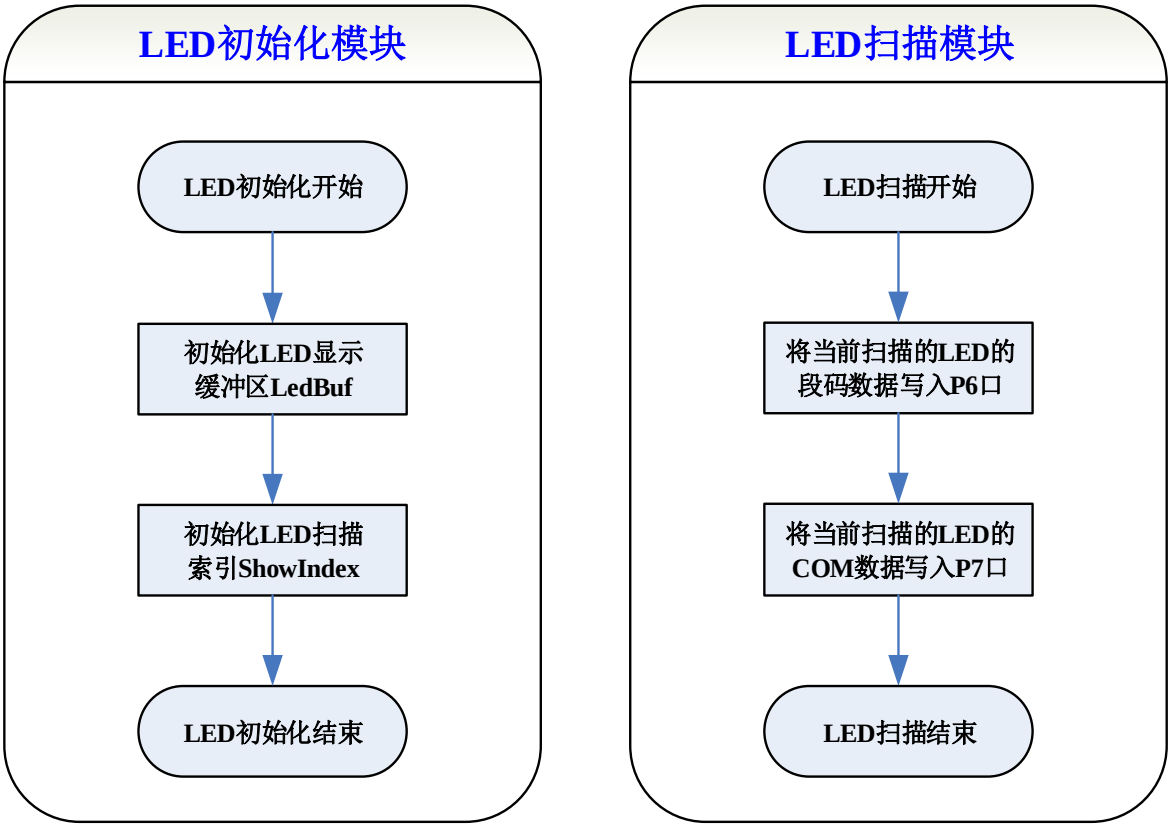
2.11.3.1 main -- 程序主函数 流程图



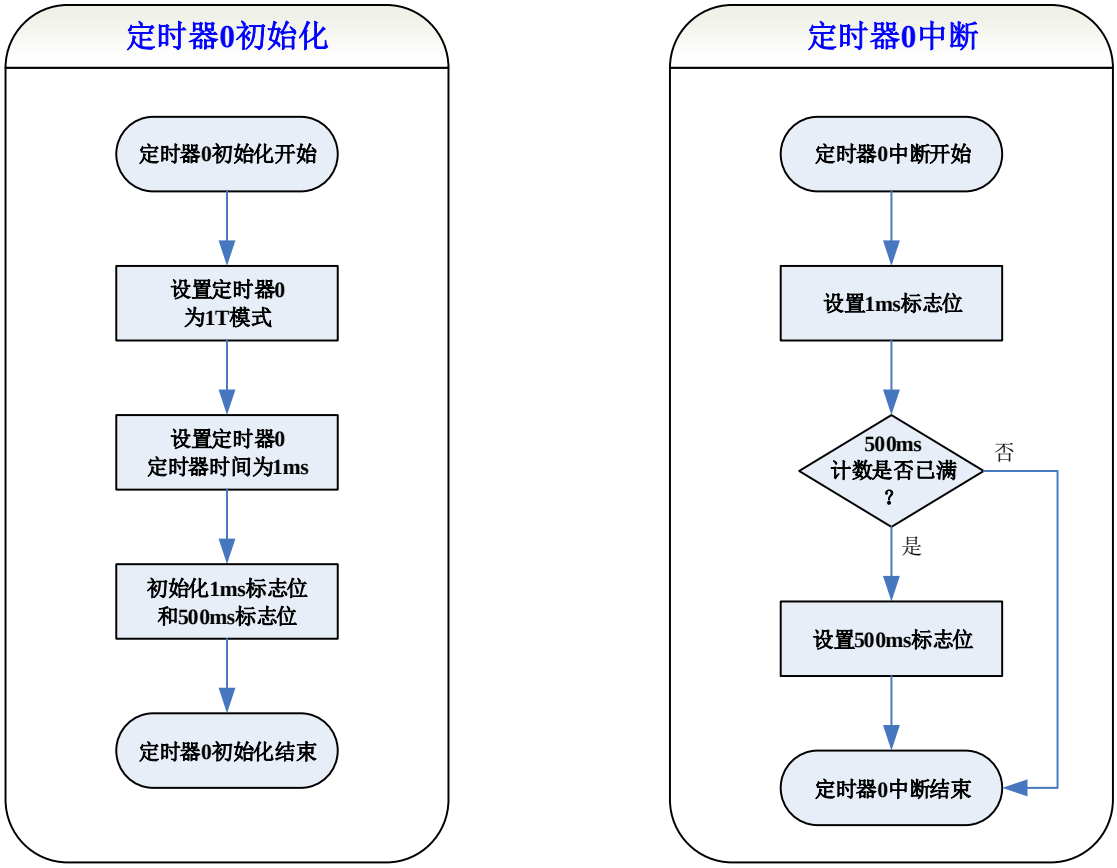
2.11.3.2 key -- 按键扫描以及按键事件处理 流程图



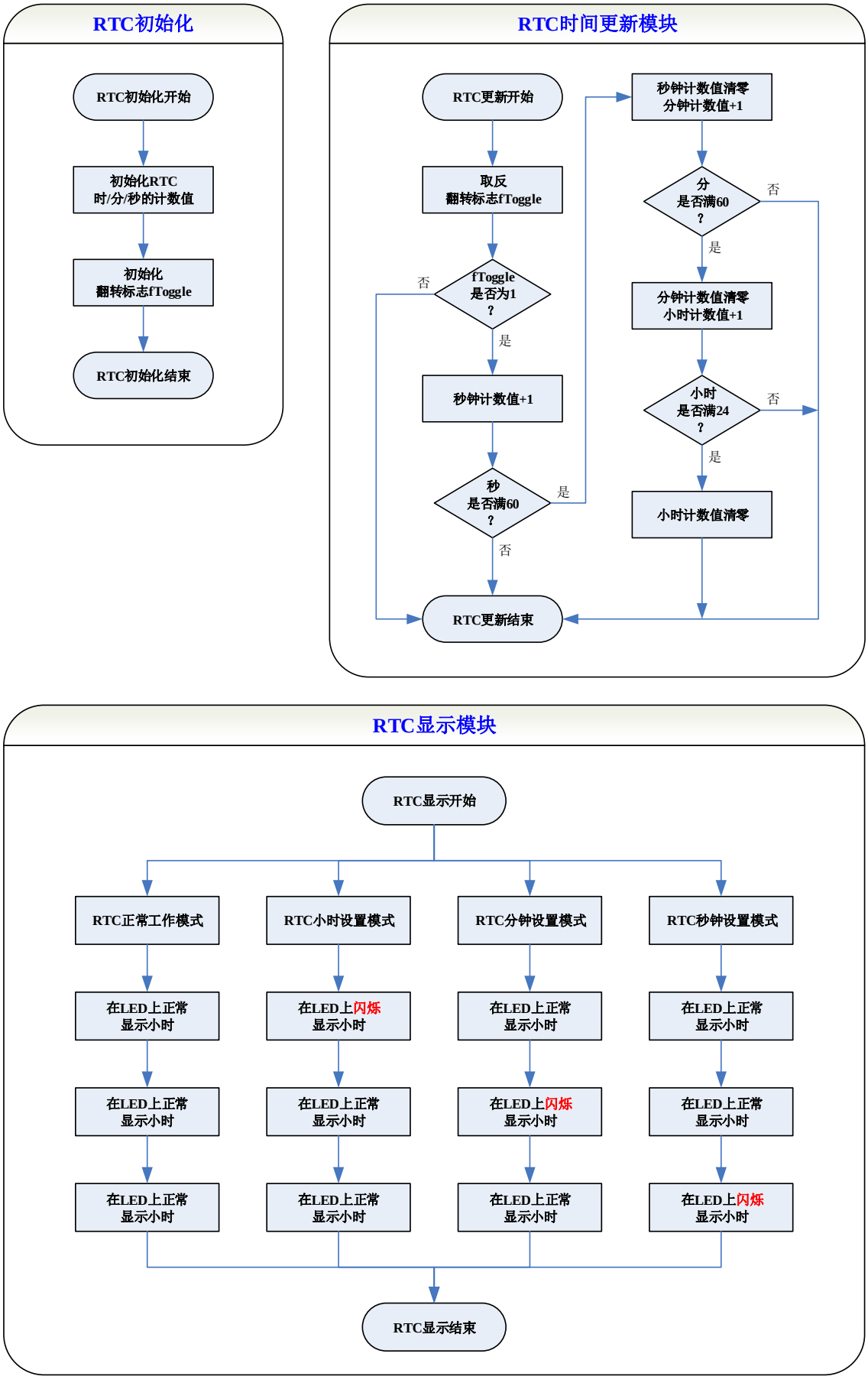
2.11.3.3 led -- 扫描 LED 数码管 流程图

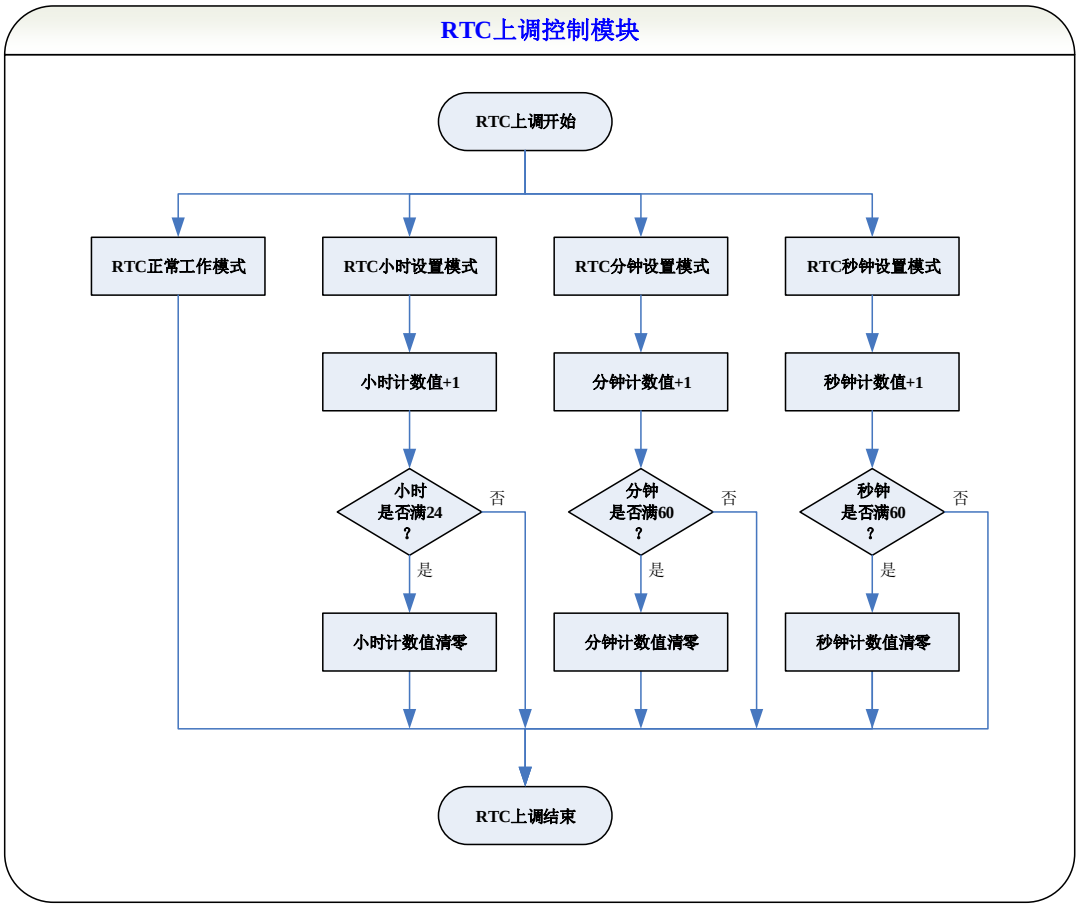


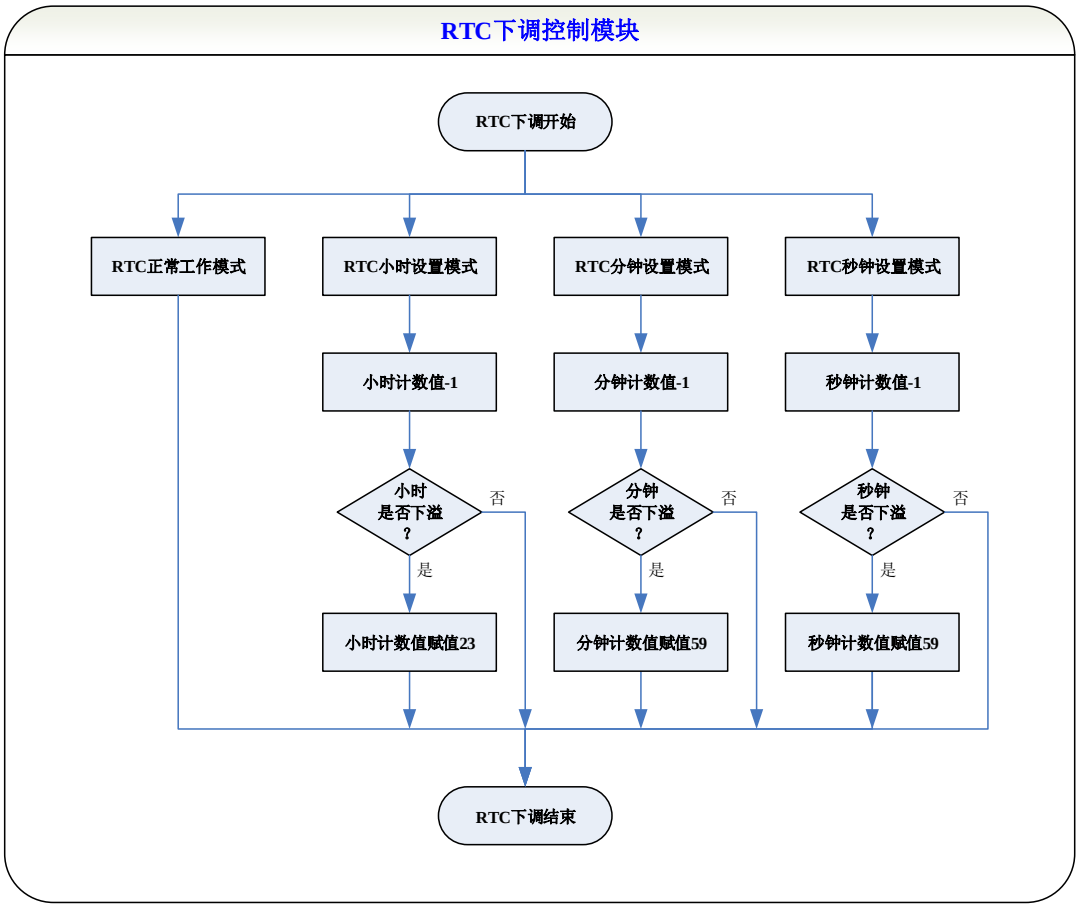
2.11.3.4 timer -- 定时器初始化以及定时器中断服务程序 流程图



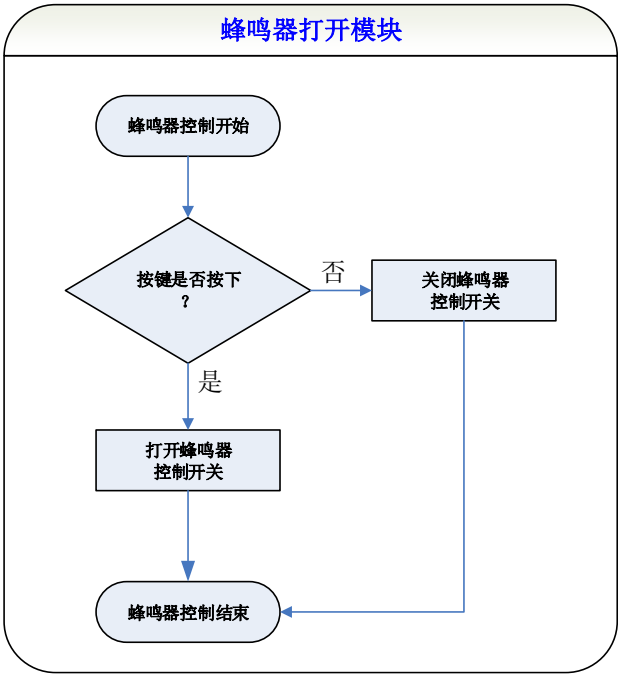
2.11.3.5 rtc -- 软件模拟 RTC 程序 流程图




RTC上调控制模块
RTC上调开始
RTC正常工作模式
RTC小时设置模式
RTC分钟设置模式
RTC秒钟设置模式
小时计数值+1
分钟计数值+1
秒钟计数值+1
小时是否满24？
分钟是否满60？
秒钟是否满60？
否
是
小时计数值清零
分钟计数值清零
秒钟计数值清零
RTC上调结束


RTC下调控制模块
RTC下调开始
RTC正常工作模式
RTC小时设置模式
RTC分钟设置模式
RTC秒钟设置模式
小时计数值-1
分钟计数值-1
秒钟计数值-1
小时是否下溢？
分钟是否下溢？
秒钟是否下溢？
否
是
小时计数值赋值23
分钟计数值赋值59
秒钟计数值赋值59
RTC下调结束

2.11.3.6 beep -- 蜂鸣器发声控制程序 流程图



2.11.4 实验程序代码

2.11.4.1 main.c -- 程序主函数

```
//main.c
//程序主函数
#include "config.h"                 //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

void main()
{
    SYS_Init();                     //系统初始化

    while (1)
    {
        if (f1ms)                   //判断 1ms 标志位
        {
            f1ms = 0;               //清除 1ms 标志，并处理 1ms 事件
            ScanKey();              //按键扫描
            KeyEvents();            //按键处理事件
            ScanLed();              //LED 数码管扫描
        }
        if (f500ms)                 //判断 500ms 标志位
        {
            f500ms = 0;             //清除 500ms 标志，并处理 500ms 事件
            RTC_Update();           //更新软件模拟的 RTC 时间
            RTC_Show();             //在 LED 数模管上显示 RTC 时间
```

```
        }
    }
}

void SYS_Init(void)         //系统初始化函数
{
    P_SW2 |= 0x80;          //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;     //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;     //初始化 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;     //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;     //初始化 P4 口为准双向口模式
    P5M0 = 0x10; P5M1 = 0x00;     //设置 P5.4 口为强推挽, 其他设置为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;     //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;     //初始化 P7 口为准双向口模式

    Led_Init();             //LED 数码管初始化
    Timer0_Init();          //定时器 0 初始化,用于产生 1ms 和 500ms 的时间基准
    RTC_Init();             //软件模拟 RTC 时间初始化

    EA = 1;                 //总中断允许位打开
}
```

2.11.4.2 key.c -- 按键扫描以及按键事件处理

```
//key.c
//按键扫描以及按键事件处理
#include "config.h"         //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

bit fKeyOK;                 //按键按下标志
bit fKeyHold;               //按键长按标志,长按 2 秒后有效,用于快速设置时间
BYTE bKeyCode;              //定义变量,保存按键键码
BYTE bDebounce;             //按键去抖动
WORD wHoldCount;            //按键长按计数器

void ScanKey()                  //按键扫描程序 (需每 1ms 执行一次!!!)
{
    BYTE key;                   //按键扫描键码

    key = 0xff;                 //初始化扫描变量
    P0 = 0xff;                  //P0 输出输出高使能内部若上拉,准备读取按键状态
    P06 = 0; P07 = 1;           //准备扫描第 1 行(P0.6)
    _nop_();                    //等待 2 个 nop 再读取按键状态
    _nop_();
    if (!P00) key = 0;          //检查第 1 行第 1 列按键是否被按下
```

```
    else if (!P01) key = 1;             //检查第 1 行第 2 列按键是否被按下
    else if (!P02) key = 2;             //检查第 1 行第 3 列按键是否被按下
    else if (!P03) key = 3;             //检查第 1 行第 4 列按键是否被按下
    else                                //如果第一行没有按键按下则扫描下一行
    {
        P07 = 0; P06 = 1;               //准备扫描第 2 行(P0.7)
        _nop_();                        //等待 2 个 nop 再读取按键状态
        _nop_();
        if (!P00) key = 4;              //检查第 2 行第 1 列按键是否被按下
        else if (!P01) key = 5;         //检查第 2 行第 2 列按键是否被按下
        else if (!P02) key = 6;         //检查第 2 行第 3 列按键是否被按下
        else if (!P03) key = 7;         //检查第 2 行第 4 列按键是否被按下
    }

    P0 = 0xff;                          //恢复 P0 口状态

    if (key != bKeyCode)                //判断按键状态是否有变化
    {
        fKeyOK = 0;                     //清除按键按下标志
        fKeyHold = 0;                   //清除按键长按标志
        bKeyCode = key;                 //保存键码
        bDebounce = 20;                 //初始化去抖动变量(去抖动时间设置为 20ms)
        wHoldCount = 2000;              //初始化按键长按计数器(按键按住不放维持时间超过 2000ms,
                                        //按键长按状态有效)
    }
    else
    {
        if (bDebounce != 0)             //判断按键扫描是否完成
        {
            if (--bDebounce == 0)       //判断去抖动是否已经完成
            {
                fKeyOK = 1;             //设置按键扫描完成标志
            }
        }
        if (--wHoldCount == 0)          //判断长按是否完成
        {
            wHoldCount = 100;           //长按有效后,每 100ms 自动触发一次按键处理
            fKeyHold = 1;               //设置按键长按标志位
            fKeyOK = 1;                 //设置按键扫描完成标志
        }
    }
}

void KeyEvents()                        //按键事件处理程序  (需每 1ms 执行一次!!!)
{
    if (fKeyOK)                         //判断按键按下标志
```

```
    {
        fKeyOK = 0;                 //清除标志

        SetLed(0, bKeyCode, 0);     //在 LED 数码管的第 0 位显示按键键码
        BeepOn(bKeyCode);           //按键使蜂鸣器发声

        if (fKeyHold)               //判断是否按键长按有效
        {
            fKeyHold = 0;           //清除按键长按标志位
            switch (bKeyCode)       //处理不同的按键键码
            {
                case 4:
                    RTC_SetUp();        //长按按键 4,快速向上调整 RTC 时间
                    break;
                case 5:
                    RTC_SetDown();  //长按按键 5,快速向下调整 RTC 时间
                    break;
            }
        }
        else                        //否则为普通按键按下
        {
            switch (bKeyCode)       //处理不同的按键键码
            {
                case 0:
                    RTC_SetMode();  //按下按键 0,设置 RTC 工作模式
                    break;
                case 4:
                    RTC_SetUp();        //按下按键 4,向上调整 RTC 时间
                    break;
                case 5:
                    RTC_SetDown();  //按下按键 5,向上调整 RTC 时间
                    break;
            }
        }
    }
}
```

2.11.4.3 led.c -- 扫描 LED 数码管

```
//led.c
//使用 P6 和 P7 口动态扫描 LED 数码管
//由于 LED 数码管是由 8 个 SEG 和 8 个 COM 组成的矩阵 LED
//所以 LED 数码管也需要扫描显示
#include "config.h"                 //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

#define   LED_OFF     0x00      //定义关闭 LED 数码管的段码数据,LED 数码管为共阴极,
```

```
                                    //所以 SEG 线全部输出 0 即可关闭一个 8 段数码管
#define   LED_POINT  0x80           //定义 LED 数码管小数点的段码数据

BYTE    ShowIndex;                  //LED 数码管扫描显示索引(8 个数码管轮流扫描显示)
BYTE    LedBuf[8];                  //8 个 LED 数码管显示数据缓冲区
BYTE code LED_PATTERN[10] =         //定义 LED 数码管显示数据 0~9 的段码数据
{
    0x3F,     //0                   //此表为共阴极数码管的段码数据
    0x06,     //1
    0x5B,     //2
    0x4F,     //3
    0x66,     //4
    0x6D,     //5
    0x7D,     //6
    0x07,     //7
    0x7F,     //8
    0x6F,     //9
};

void Led_Init(void)                 //LED 数码管初始化程序
{
    LedBuf[0] = LED_OFF;            //初始化 LED 数码管显示数据缓冲区,上电关闭显示
    LedBuf[1] = LED_OFF;
    LedBuf[2] = LED_OFF;
    LedBuf[3] = LED_OFF;
    LedBuf[4] = LED_OFF;
    LedBuf[5] = LED_OFF;
    LedBuf[6] = LED_OFF;
    LedBuf[7] = LED_OFF;

    ShowIndex = 0;                  //初始化扫描显示索引
}

void ScanLed(void)                  //LED 数码管扫描显示程序,  (需每 1ms 执行一次!!!)
{
    P6 = (~LedBuf[ShowIndex]);      //将当前扫描到的 LED 数码管的段码移入到 P6 口
    P7 = (~(1 << ShowIndex));       //将当前扫描到的 LED 数码管的 COM 数据移入到 P7 口

    if (++ShowIndex >= 8)           //扫描索引值递增
        ShowIndex = 0;              //扫描完一轮后索引值归零
}

void SetLed   (BYTE nIndex,         //设置 LED 显示函数,nIndex:设置目标数码管,0~7 分别对应 8 个数码
               BYTE nNum,           //nNum:设置显示的数据,有效值 0~9,其他值则关闭显示
               BOOL bShowPoint      //bShowPoint:设置是否显示小数点
               )
```

```
{
    if (nIndex < 8)                 //目标数码管的有效值为 0~7
    {
        if (nNum < 10)              //判断显示的值是否为 0~9
            LedBuf[nIndex] = LED_PATTERN[nNum];  //将相应数据的段码赋值到 LED 数码管显示数据缓冲区
        else
            LedBuf[nIndex] = LED_OFF;           //如果是其他值则关闭显示

        if (bShowPoint)                         //判断是否需要显示小数点
            LedBuf[nIndex] |= LED_POINT;        //如果需要显示,则增加小数点的段码数据
    }
}
```

2.11.4.4 timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:1ms 定时，产生 1ms 和 500ms 的标志位
#include "config.h"             //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

bit f1ms;                       //1ms 标志位
bit f500ms;                     //500ms 标志位
int cnt500;                     //500 次计数器,用于产生 500ms 的时间

void Timer0_Init(void)          //1 毫秒@11.0592MHz
{
    AUXR |= 0x80;               //定时器 0 时钟 1T 模式
    TMOD &= 0xF0;               //设置定时器 0 为模式 0
    TL0 = 0xCD;                 //设置定时器 0 初始值
    TH0 = 0xD4;                 //设置定时器 0 初始值
    TF0 = 0;                    //清除 TF0 标志
    TR0 = 1;                    //定时器 0 开始计时
    ET0 = 1;                    //使能定时器 0 中断

    cnt500 = 0;                 //初始化 500 次计数器

    f1ms = 0;                   //初始化 1m 标志位
    f500ms = 0;                 //初始化 500ms 标志位
}

void Timer0_Isr(void) interrupt 1   //定时器 0 中断服务程序
{
    f1ms = 1;                   //设置 1ms 标志位

    if (cnt500++ >= 500)        //判断 500 此计数是否已满
    {
```

```
            cnt500 = 0;         //清除计数值,开始开始计数
            f500ms = 1;         //设置 500ms 标志位
        }
}
```

2.11.4.5 rtc.c -- 软件模拟 RTC 程序

```
//rtc.c
//软件模拟 RTC 程序
//包括 RTC 时间更新控制
//RTC 显示控制以及 RTC 时间调整控制
#include "config.h"             //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

bit fToggle;                    //0.5 秒->1 秒的控制位(两次计数器),用于在设置时间时进行闪烁控制
BYTE bHour;                     //RTC 小时计数值
BYTE bMinute;                   //RTC 分钟计数值
BYTE bSecond;                   //RTC 秒钟计数值
BYTE bMode;                     //RTC 工作模式控制
                                //0:正常工作模式
                                //1:小时设置模式
                                //2:分钟设置模式
                                //3:秒钟设置模式

void RTC_Init()                 //RTC 初始化函数
{
    bHour = 12;                 //初始化时间为 12:30:00
    bMinute = 30;
    bSecond = 0;
    bMode = 0;                  //初始化 RTC 工作模式为正常工作模式
    fToggle = 0;                //初始化翻转控制
}

void RTC_Update()               //RTC 时间更新函数,  (需每 500ms 执行一次!!!)
{
    fToggle = !fToggle;         //翻转控制位

    if ((fToggle == 1) &&       //如果 1 秒时间到并且为正常工作模式,则更新 RTC 时间
       (bMode == 0))
    {
        if (++bSecond >= 60)            //秒钟递增 1,并判断秒钟是否记满 60
        {
            bSecond = 0;                //秒钟归零
            if (++bMinute >= 60)        //分钟递增 1,并判断分钟是否记满 60
            {
                bMinute = 0;            //分钟归零
                if (++bHour >= 24)      //小时递增 1,并判断小时是否记满 24
```

```
                {
                    bHour = 0;          //小时归零
                }
            }
        }
    }
}

void RTC_Show()                         //RTC 显示控制程序,不同的模式显示方式不一样
{
    switch (bMode)                      //对 RTC 工作模式进行分支处理
    {
    case 0:                                         //正常工作模式
        SetLed(7, (BYTE)(bHour / 10), 0);           //在数码 7 上显示小时的十位
        SetLed(6, (BYTE)(bHour % 10), 1);           //在数码 6 上显示小时的个位
        SetLed(5, (BYTE)(bMinute / 10), 0);         //在数码 5 上显示分钟的十位
        SetLed(4, (BYTE)(bMinute % 10), 1);         //在数码 4 上显示分钟的个位
        SetLed(3, (BYTE)(bSecond / 10), 0);         //在数码 3 上显示秒钟的十位
        SetLed(2, (BYTE)(bSecond % 10), 0);         //在数码 2 上显示秒钟的个位
        break;
    case 1:                                         //设置 RTC 的小时模式
        if (fToggle)                                //闪烁控制
        {
            SetLed(7, (BYTE)(bHour / 10), 0);       //在数码 7 上显示小时的十位
            SetLed(6, (BYTE)(bHour % 10), 1);       //在数码 6 上显示小时的个位
        }
        else
        {
            SetLed(7, -1, 0);                       //在数码 7 上关闭显示,产生闪烁效果
            SetLed(6, -1, 1);                       //在数码 6 上关闭显示
        }
        SetLed(5, (BYTE)(bMinute / 10), 0);         //在数码 5 上显示分钟的十位
        SetLed(4, (BYTE)(bMinute % 10), 1);         //在数码 4 上显示分钟的个位
        SetLed(3, (BYTE)(bSecond / 10), 0);         //在数码 3 上显示秒钟的十位
        SetLed(2, (BYTE)(bSecond % 10), 0);         //在数码 2 上显示秒钟的个位
        break;
    case 2:                                         //设置 RTC 的分钟模式
        SetLed(7, (BYTE)(bHour / 10), 0);           //在数码 7 上显示小时的十位
        SetLed(6, (BYTE)(bHour % 10), 1);           //在数码 6 上显示小时的个位
        if (fToggle)                                //闪烁控制
        {
            SetLed(5, (BYTE)(bMinute / 10), 0);     //在数码 5 上显示分钟的十位
            SetLed(4, (BYTE)(bMinute % 10), 1);     //在数码 4 上显示分钟的个位
        }
        else
        {
```

```
            SetLed(5, -1, 0);                       //在数码 5 上关闭显示,产生闪烁效果
            SetLed(4, -1, 1);                       //在数码 4 上关闭显示
        }
        SetLed(3, (BYTE)(bSecond / 10), 0);         //在数码 3 上显示秒钟的十位
        SetLed(2, (BYTE)(bSecond % 10), 0);         //在数码 2 上显示秒钟的个位
        break;
    case 3:                                         //设置 RTC 的秒钟模式
        SetLed(7, (BYTE)(bHour / 10), 0);           //在数码 7 上显示小时的十位
        SetLed(6, (BYTE)(bHour % 10), 1);           //在数码 6 上显示小时的个位
        SetLed(5, (BYTE)(bMinute / 10), 0);         //在数码 5 上显示分钟的十位
        SetLed(4, (BYTE)(bMinute % 10), 1);         //在数码 4 上显示分钟的个位
        if (fToggle)                                //闪烁控制
        {
            SetLed(3, (BYTE)(bSecond / 10), 0);     //在数码 3 上显示秒钟的十位
            SetLed(2, (BYTE)(bSecond % 10), 0);     //在数码 2 上显示秒钟的个位
        }
        else
        {
            SetLed(3, -1, 0);                       //在数码 3 上关闭显示,产生闪烁效果
            SetLed(2, -1, 0);                       //在数码 2 上关闭显示
        }
        break;
    }
}

void RTC_SetMode()              //RTC 工作模式控制
{
    if (++bMode > 3)            //RTC 工作模式转换(0:正常工作模式->1:小时设置模式->
                                //2:分钟设置模式->3:秒钟设置模式->0:正常工作模式->...)
    bMode = 0;                  //如果 RTC 模式大于 3,则 RTC 重归正常工作模式
}

void RTC_SetUp()                //RTC 时间上调控制
{
    if (bMode == 0)             //如果是正常工作模式,上调无意义
        return;                 //直接返回

    switch (bMode)              //根据工作模式调整时/分/秒
    {
    case 1:                     //小时设置模式
        if (++bHour >= 24)      //小时计数值向上递增
            bHour = 0;          //溢出后自动归零
        break;
    case 2:                     //分钟设置模式
        if (++bMinute >= 60)    //分钟计数值向上递增
            bMinute = 0;        //溢出后自动归零
```

```
            break;
        case 3:                         //秒钟设置模式
            if (++bSecond >= 60)        //秒钟计数值向上递增
                bSecond = 0;            //溢出后自动归零
            break;
        }

        fToggle = 1;                    //调整完后立即强制显示,不用等待 500ms
        RTC_Show();                     //显示调整的时间
}

void RTC_SetDown()                      //RTC 时间下调控制
{
        if (bMode == 0)                 //如果是正常工作模式,下调无意义
        return;                         //直接返回

        switch (bMode)                  //根据工作模式调整时/分/秒
        {
        case 1:                         //小时设置模式
            if (--bHour >= 24)          //小时计数值向下递减
                bHour = 23;             //溢出后自动设置上限值
            break;
        case 2:                         //分钟设置模式
            if (--bMinute >= 60)        //分钟计数值向下递减
                bMinute = 59;           //溢出后自动设置上限值
            break;
        case 3:                         //秒钟设置模式
            if (--bSecond >= 60)        //秒钟计数值向下递减
                bSecond = 59;           //溢出后自动设置上限值
            break;
        }

        fToggle = 1;                    //调整完后立即强制显示,不用等待 500ms
        RTC_Show();                     //显示调整的时间
}
```

2.11.4.6 beep.c -- 蜂鸣器发声控制程序

```
//beep.c
//蜂鸣器发声控制程序
//STC8H8K64U 实验箱上的蜂鸣器为有源蜂鸣器
//P5.4 控制口输出 0，使能蜂鸣器发声
//P5.4 控制口输出 1，关闭蜂鸣器
#include "config.h"       //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

void BeepOn(BYTE bBeep)         //蜂鸣器发声控制程序
```

```
{
    if (bBeep != 0xff)
        P54 = 0;                    //蜂鸣器发声
    else
        P54 = 1;                    //关闭蜂鸣器发声控制位
}
```

2.11.4.7 config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__            //防止头文件被重复包含
#define __CONFIG_H__

#define HIRC            11059200UL
#define FOSC            11059200UL
#define SYSCLK          FOSC
#define MAIN_Fosc       FOSC

#include <stc8h.h>      //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"        //包含项目头文件
#include "timer.h"
#include "key.h"
#include "led.h"
#include "rtc.h"
#include "beep.h"

void SYS_Init(void);    //函数声明

#endif
```

2.11.4.8 key.h -- 项目配置文件的头文件

```
#ifndef __KEY_H__           //防止头文件被重复包含
#define __KEY_H__

void ScanKey();             //函数声明
void KeyEvents();           //函数声明

#endif
```

2.11.4.9 led.h -- 项目配置文件的头文件

```
#ifndef __LED_H__           //防止头文件被重复包含
#define __LED_H__
```

```
void Led_Init(void);            //函数声明
void ScanLed(void);             //函数声明
void SetLed(BYTE nIndex, BYTE nNum, BOOL bShowPoint);

#endif
```

2.11.4.10 timer.h -- 项目配置文件的头文件

```
#ifndef __TIMER_H__         //防止头文件被重复包含
#define __TIMER_H__

void Timer0_Init(void);         //函数声明

extern bit f1ms;                //外部变量声明
extern bit f500ms;              //外部变量声明

#endif
```

2.11.4.11 rtc.h -- 项目配置文件的头文件

```
#ifndef __RTC_H__           //防止头文件被重复包含
#define __RTC_H__

void RTC_Init();            //函数声明
void RTC_Update();          //函数声明
void RTC_Show();            //函数声明
void RTC_SetMode();         //函数声明
void RTC_SetUp();           //函数声明
void RTC_SetDown();         //函数声明

#endif
```

2.11.4.12 beep.h -- 蜂鸣器发声控制程序的头文件

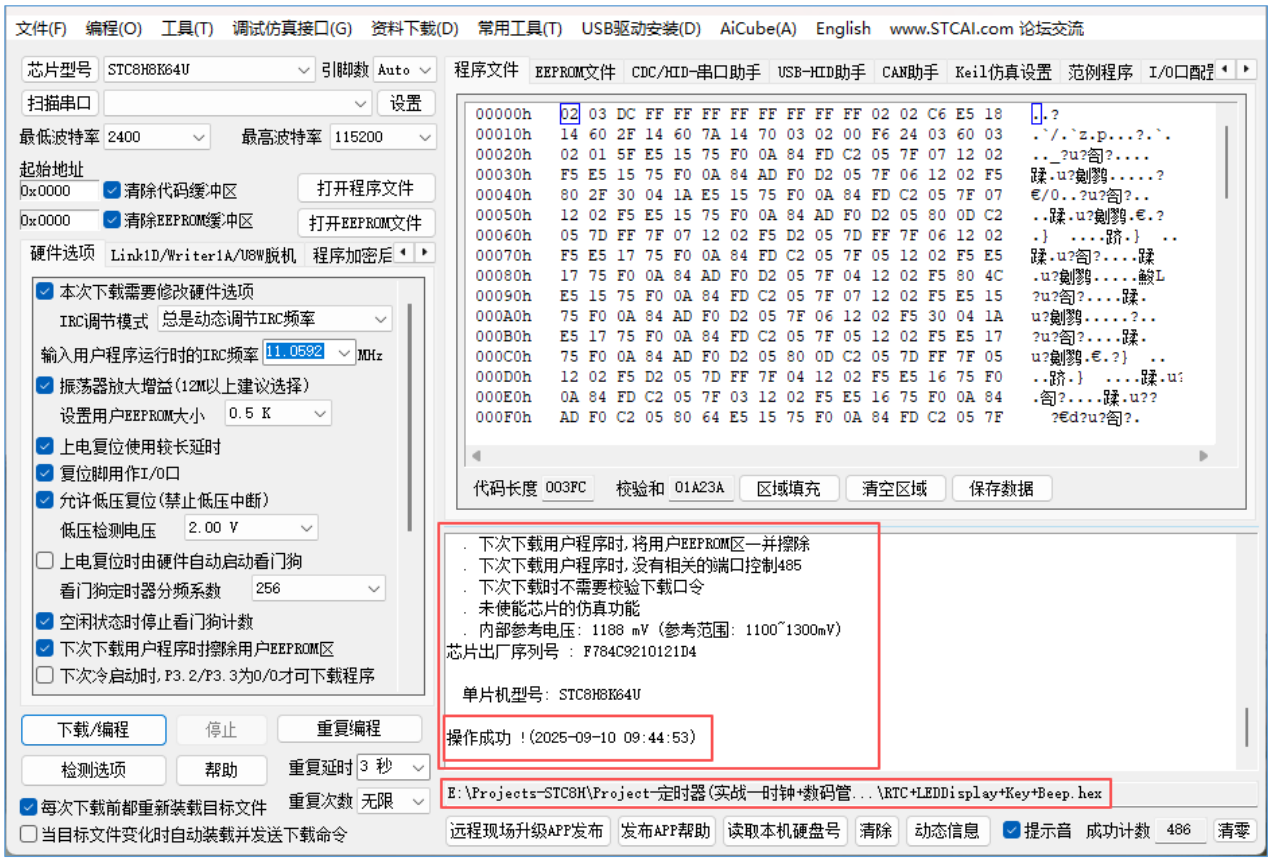
```
#ifndef __BEEP_H__
#define __BEEP_H__

void BeepOn(BYTE bBeep);        //函数声明

#endif
```

2.11.5 程序下载，观察实验现象

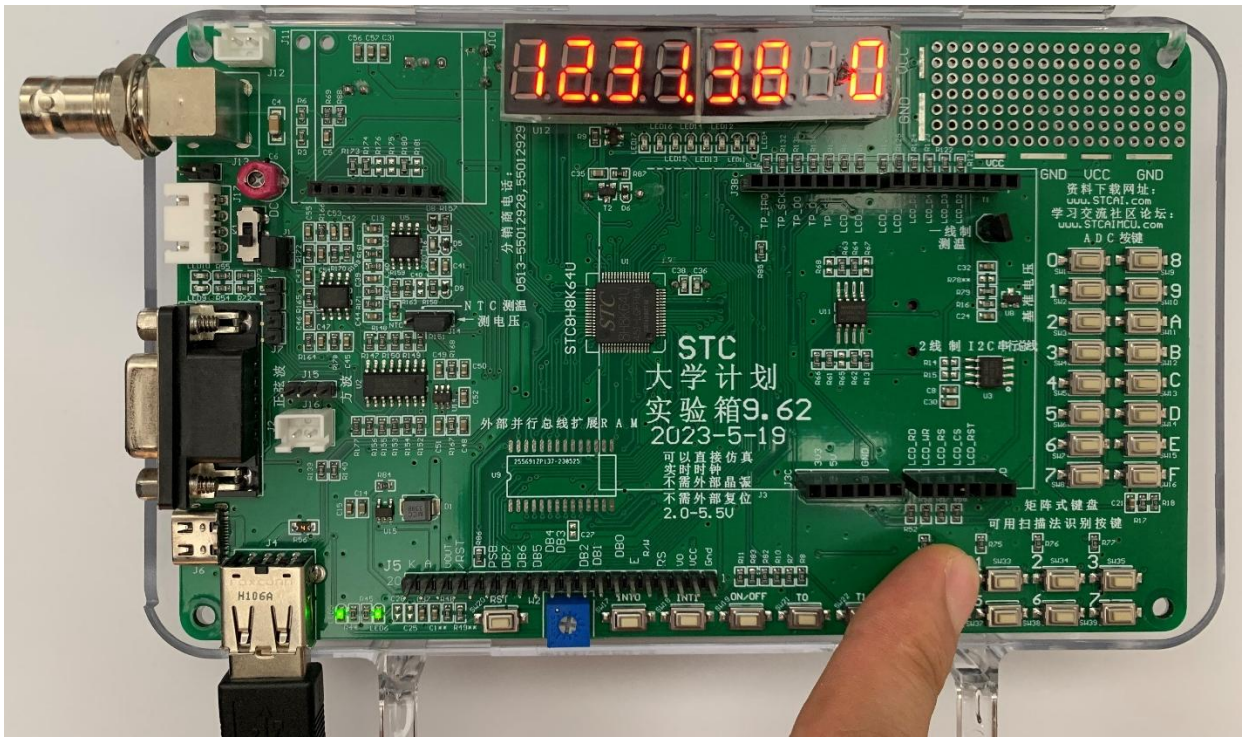
➲ 录入代码，保存，编译。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“RTC+LEDDisplay+Key+Beep.hex”，如下图：



➲ **观察 STC8H8K64U 实验箱：**程序下载“操作成功”，实验箱上的数码管高 6 位立即显示“12:30:00”，同时秒显示走动。因本实验 RTC 的初始化时间为 12:30:00，那么 STC8H8K64U 实验箱数码高 6 位显示计时从 12:30:00 开始。

在 STC8H8K64U 实验箱右下方找到矩阵式键盘 0 - 7 这 8 个按键

➲ **分别按下 0 - 7 这 8 个按键：**在 STC8H8K64U 实验箱数码管最低位会显示当前的按键键码，并且蜂鸣器会发出“Be…”音。如下图，按下 0 按键，数码管最低位会显示 0，蜂鸣器发出“Be…”音：



- **矩阵按键中的按键 0，是 RTC 模式设置键**
  - 上电后 RTC 为正常工作模式，此时数码管动态显示 RTC 时间
  - 按一次按键 0，RTC 会切换为小时设置模式，此时 RTC 的小时值会 0.5 秒闪烁一次
  - 再按一次按键 0，RTC 会切换为分钟设置模式，此时 RTC 的分钟值会 0.5 秒闪烁一次
  - 再按一次按键 0，RTC 会切换为秒钟设置模式，此时 RTC 的秒钟值会 0.5 秒闪烁一次
  - 在按一次按键 0，恢复为 RTC 正常工作模式
- **矩阵按键中的按键 4 和 5 分别是上调键和下调键**
  - 只有在 RTC 的时/分/秒设置模式式，上调键和下调键才有效
- **时钟设置模式时，长按按键 4 和 5，可实现快速设置功能**

完成实验预设，实验验证完毕。

2.12 RTC 实验

2.12.1 RTC 实验(拓展学习之使用 RTC 功能设定闹钟)

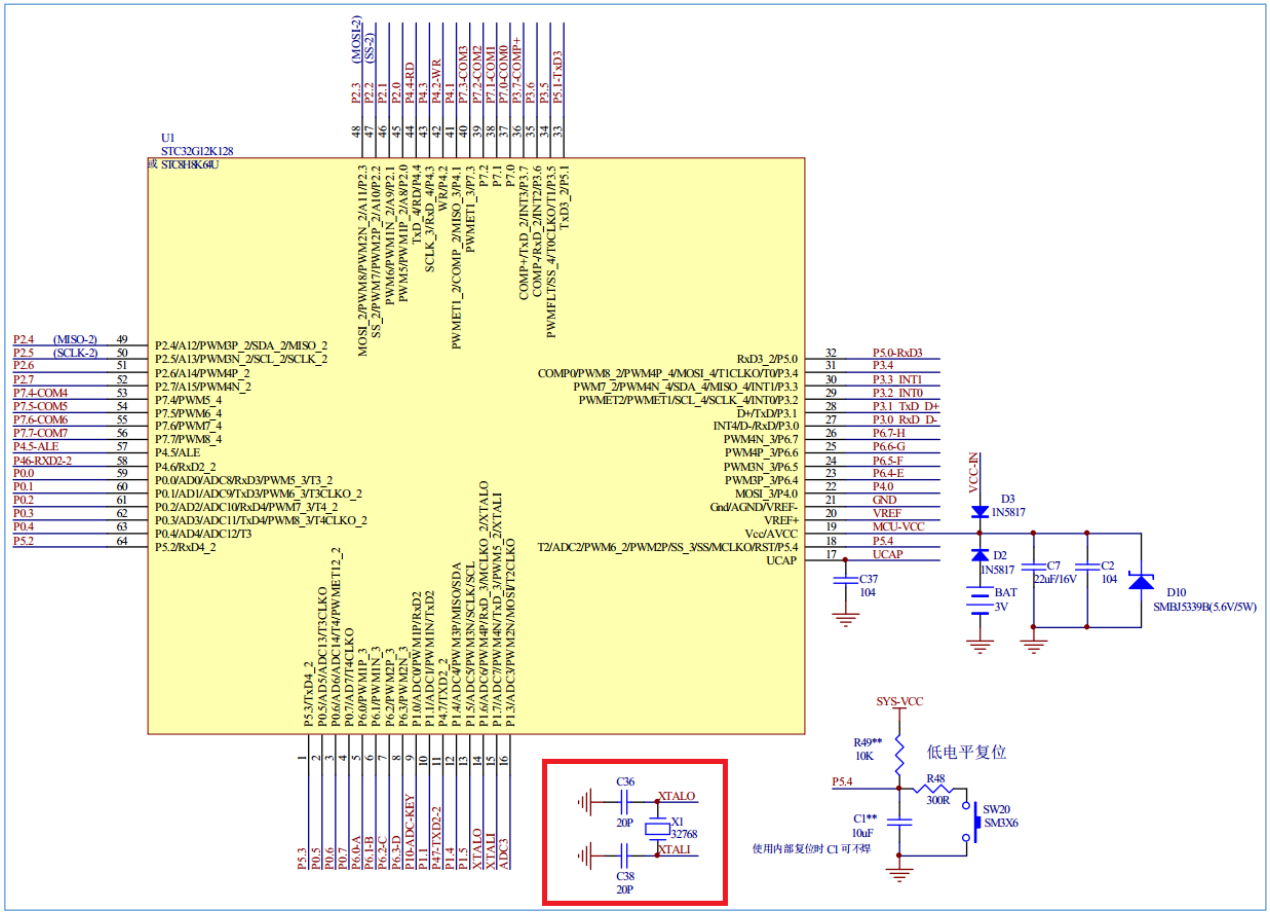
2.12.1.1 实验介绍

1、学习使用 STC8H8K64U 的 RTC 实时时钟功能
2、学习使用 STC8H8K64U 的 RTC 的闹钟功能
3、使用拓展中断号工具解除 Keil 中 32 个中断号的限制
4、熟练使用 USB-CDC 功能
5、熟练使用 AiCube-ISP 软件

程序运行约 5s 后，AiCube-ISP 软件的 CDC 窗口显示“RTC Start”，RTC 闹钟 10s 产生中断，打印数据在 CDC 窗口显示“Alarm!”
CDC 串口发送任意数据，读取 RTC 的秒值，并在 CDC 窗口显示
观察到实验现象：

- 程序运行 5s 后，AiCube-ISP 软件 CDC 的接收缓冲区窗口显示“RTC Start”，再 10s 后，窗口显示“Alarm！”
- 在 USB-CDC 串口发送任意数据，在 CDC 的接收缓冲区窗口显示当前的 RTC 秒值

2.12.1.2 原理图

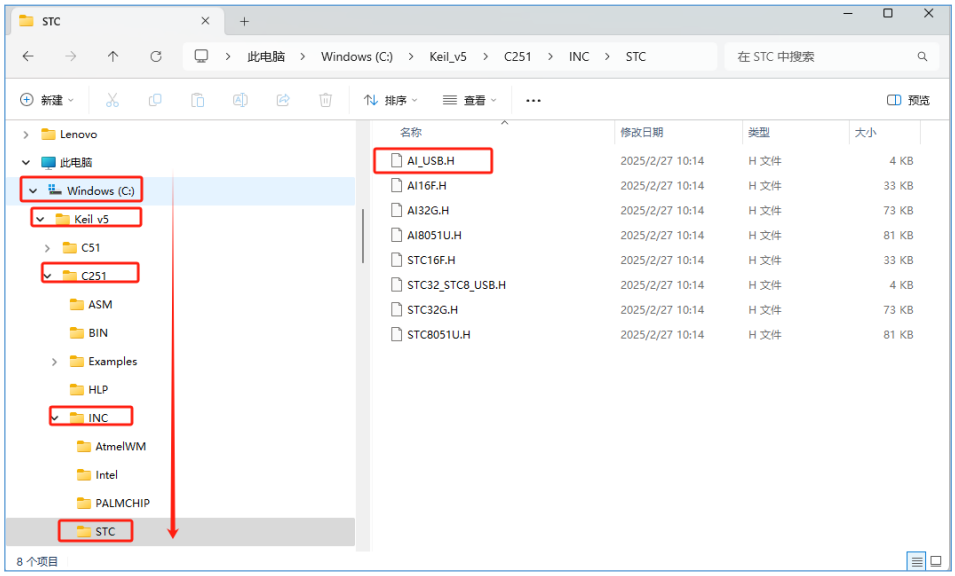


2.12.1.3　准备工作--添加头文件、Keil 中断拓展插件、库函数

1、添加常用头文件 ai_usb.h

#include "ai_usb.h"　　//使用 usb 所需要包含的头文件。

这个头文件在使用 AiCube-ISP 软件添加型号和头文件到 Keil 中时就已经一并被添加到 Keil 中了。



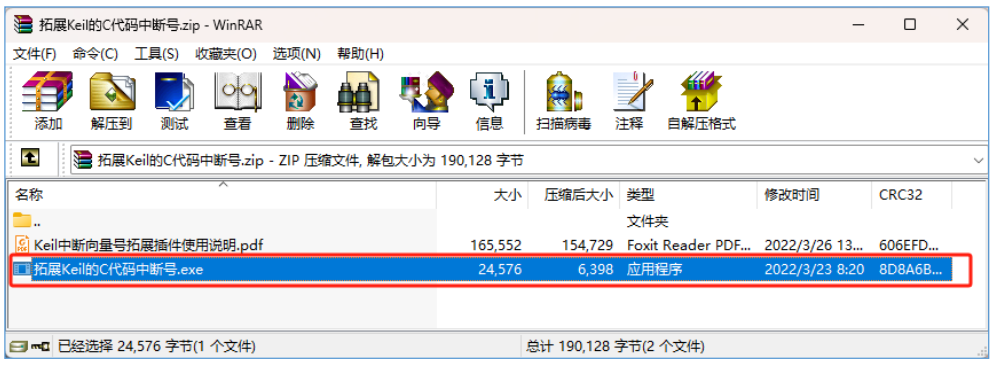
2、添加 Keil 中断拓展插件

Keil C51/C251 编译器只支持 31 以内的中断号（0~31），超过 31 编译会报错。**RTC 的中断向量 RTC_VECTOR 为 36**，这就需要 Keil 中断号拓展工具，可将中断号拓展到 254。

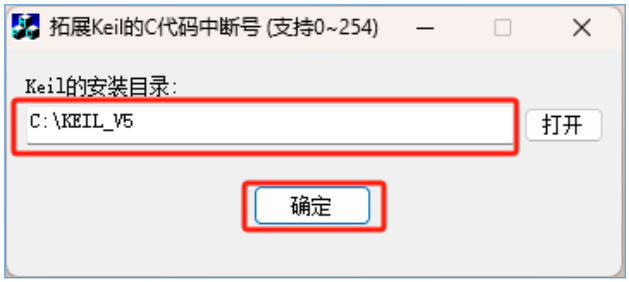
这部分可以到 STCAI 的官网：https://www.stcai.com/gjrj 的软件工具➔工具软件中“Keil 中断拓展插件”进行下载。



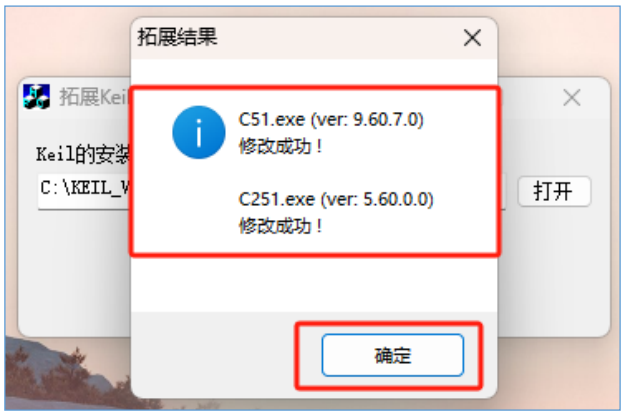
下载完成后，打开压缩包“拓展 Keil 的 C 代码中断号.zip”，找到“拓展 Keil 的 C 代码中断号.exe”，这是绿色免安装软件。



直接双击“拓展 Keil 的 C 代码中断号.exe”，出现如下界面，选择安装路径，按照默认直接安装到 C:\Keil_V5 目录下，点击“确定”



如下图，点击“确定”，Keil 的 C 代码中断号到 254



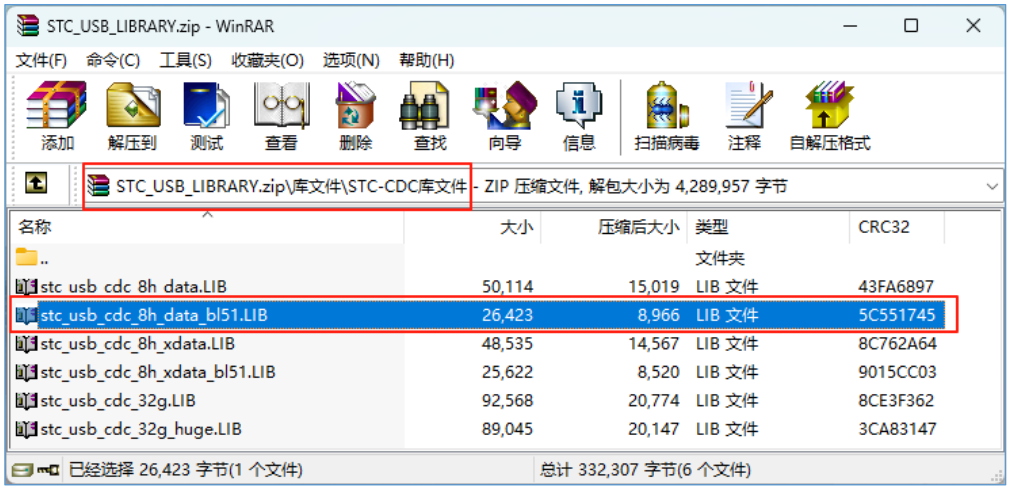
3、添加 usb_cdc 库函数文件

➲ 因为要使用到 USB 功能，必须在项目中添加 USB 库函数。这部分可以到 STCAI 的官网：https://www.stcai.com/khs 的 USB 库文件中进行下载

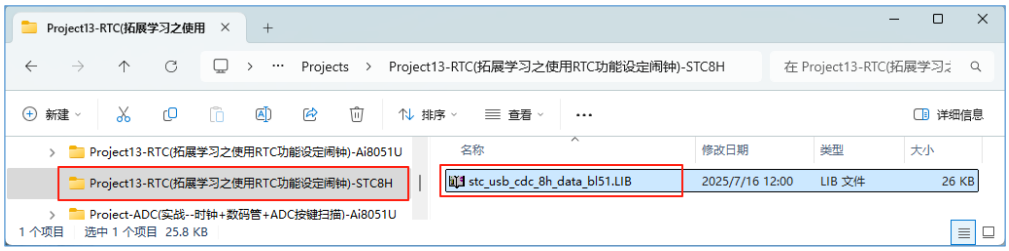


下载完成后，打开压缩包“STC_USB_LIBRARY.zip”，
在’库文件\查询模式 STC-CDC 库文件’中，复制出来 stc_usb_cdc_8h_data_bl51.LIB

【8 位的 8051 的 USB 库 可以用 BL51 连接器
BL51：　不支持 REMOVEUNUSED；BL51 支持 扩展中断号工具；
如果 不用 REMOVEUNUSED，可以 使用 BL51 ！
LX51：　支持 REMOVEUNUSED；LX51 不支持 扩展中断号工具；
如果 不用 扩展中断号工具，可以 使用 LX51 ！
详情请参考本公司技术论坛链接：8 位的 8051 的 USB 库可以用 BL51；BL51 不支持 REMOVEUNUSED；
https://www.stcaimcu.com/thread-17105-1-1.html
(出处：国芯人工智能技术交流网站)】

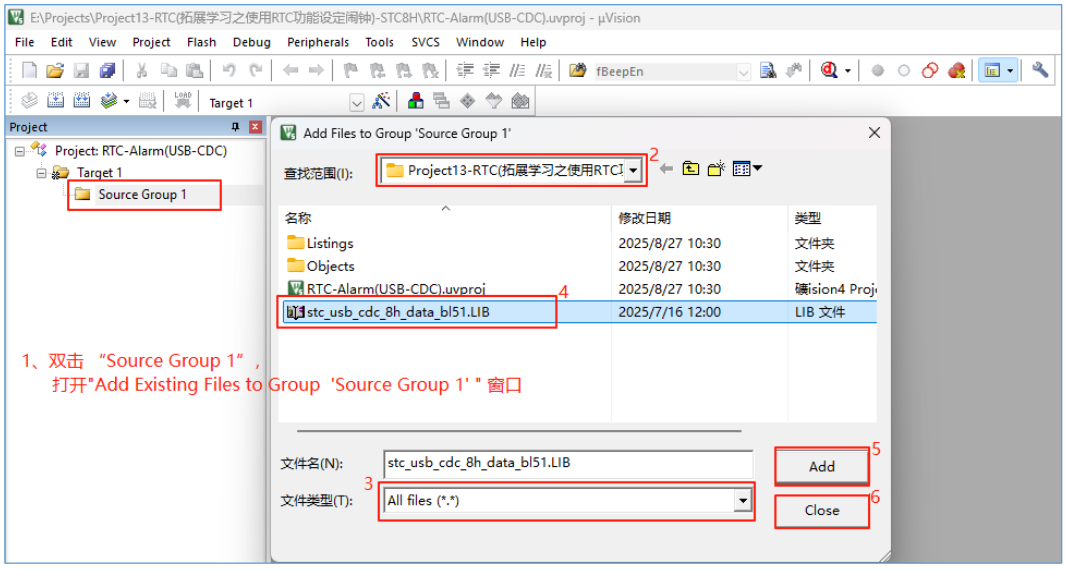


➲ 将库函数“stc_usb_cdc_8h_data_bl51.LIB”复制到创建项目所在的文件夹中

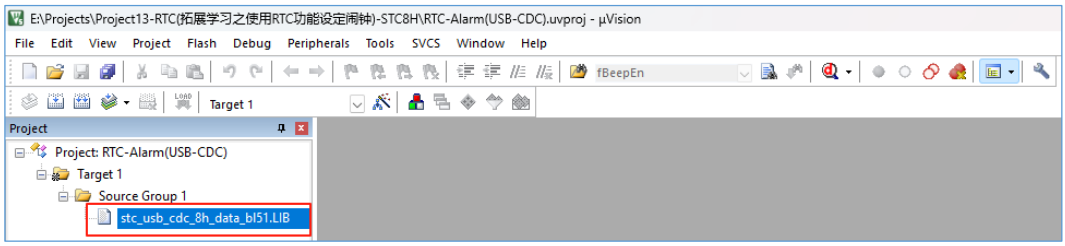


➲ 将库函数 stc_usb_cdc_8h_data_bl51.LIB 添加进入此项目中

在 Keil 中，双击 Source Group，在弹出的 Add Files to Group 'Source Group 1 窗口中，将库函数 stc_usb_cdc_8h_data_bl51.LIB 添加进入此工程项目，就可以使用 USB-CDC 功能。



这样，Keil 界面变化如下图：



【注意】：USB-CDC 串口则更类似传统串口的形式，使用 COM 口进行通讯，但是只有 Win10 及其以上才默认携带 USB-CDC 的驱动，较低的系统版本可能要手动安装驱动才能使用。

2.12.1.4 实验程序代码

```
#include <STC8H.H>
#include   "ai_usb.h"    //USB 库和 USB 虚拟设备的头文件
//由于在 USB 库函数的代码中已经定义了，不停电 ISP 下载命令字"@STCISP#"，
//只需要在下载软件的“收到用户命令后复位到 ISP 监控程序区”选项卡中进行相应的设置，
//即可实现 USB 不停电 ISP 下载功能，或串口不停电 ISP 下载功能。

#include "intrins.h"          //使用_nop_()函数所必须要包含的头文件,
                              //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                              //会导致编译器找不到这个而函数而报错。
bit alarm_flag = 0;           //定义 RTC 中断标志

void rtc_isr(void) interrupt RTC_VECTOR  //RTC 中断服务程序
{
    alarm_flag = 1;
    RTCIF = 0;                //清除 RTC 中断标志
}

void sys_init(void)           //系统初始化
{
    P_SW2 |= 0x80;            //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;     //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;     //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;     //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;     //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;     //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;     //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;     //设置 P7 口为准双向口模式
}

void Delay1000ms(void)    //延时 1000ms@11.0592MHz
{
    unsigned char data i, j, k;

    i = 57;
    j = 27;
    k = 112;
    do
    {
        do
        {
            while (--k);
        } while (--j);
```

```
    } while (--i);
}

void main(void)
{
    sys_init();                     //调用系统初始化函数
    usb_init();                     //调用 USB 初始化函数
    EA = 1;                         //总中断允许位打开
    P40 = 0;                        //给 LED 灯供电

    Delay1000ms ();                 //调用延时 5s，否则根本看不到 USB-CDC 窗口显示“RTC Start”
    Delay1000ms ();
    Delay1000ms ();
    Delay1000ms ();
    Delay1000ms ();

    X32KCR = 0xc0;                  //启动外部 32K 晶振
    while (!(X32KCR & 0x01));       //等待时钟稳定
    RTCCFG &= ~0x02;                //选择外部 32K 作为 RTC 时钟源

    RTCIF = 0;                  //清除 RTC 中断标志
    RTCIEN = 0x80;              //使能 RTC 闹钟中断
    ALASEC = 10;                //闹钟秒设定，这里设定 10s
                                //因为没有设置初始值，所以初始值默认都是从 00 开始走的
                                //这里设置 10 就相当于从 RTC 初始化完成后开始 10s 就会接收到闹钟中断

    RTCCR = 0x01;               // RTC 模块使能
    RTCCFG |= 0x01;             //触发 RTC 寄存器初始化
    while(RTCCFG & 0x01);       //等待 RTC 初始化完成

    printf_usb("RTC Start\r\n");
        //直接打印输出 “RTC Start” 到主机相应软件的 USB【CDC/HID-串口助手】

    while(1)                    //判断 RTC 闹钟中断，打印数据
    {
        if(alarm_flag)          //闹钟中断到了
        {
            alarm_flag = 0;
            printf_usb("Alarm!\r\n");           //打印一下闹钟到达的信息
        }

        if(bUsbOutReady)        //读取是否发送消息
            //USB 库中有定义 bUsbOutReady 位变量，查询方式必须调用一次“usb_OUT_done();”
        {
            printf_usb("This_sec:%d\r\n",(int)RTCSEC);          //发送消息的时候返回当前的秒数
                //直接打印输出 “This_sec:%d” 到主机相应软件的 USB【CDC/HID-串口助手】
```

```
            usb_OUT_done();         //每次处理完 OUT 信息，最后都要调用这个函数
                //当使用查询方式 if(bUsbOutReady)，查询调用 USB 库函数时，在数据处理完成后，
                //必须调用一次“usb_OUT_done();”，这样才能接收下一包数据。
        }
    }
}
```

【小技巧 -- 灵活运用 AiCube-ISP 软件工具自动生成代码】

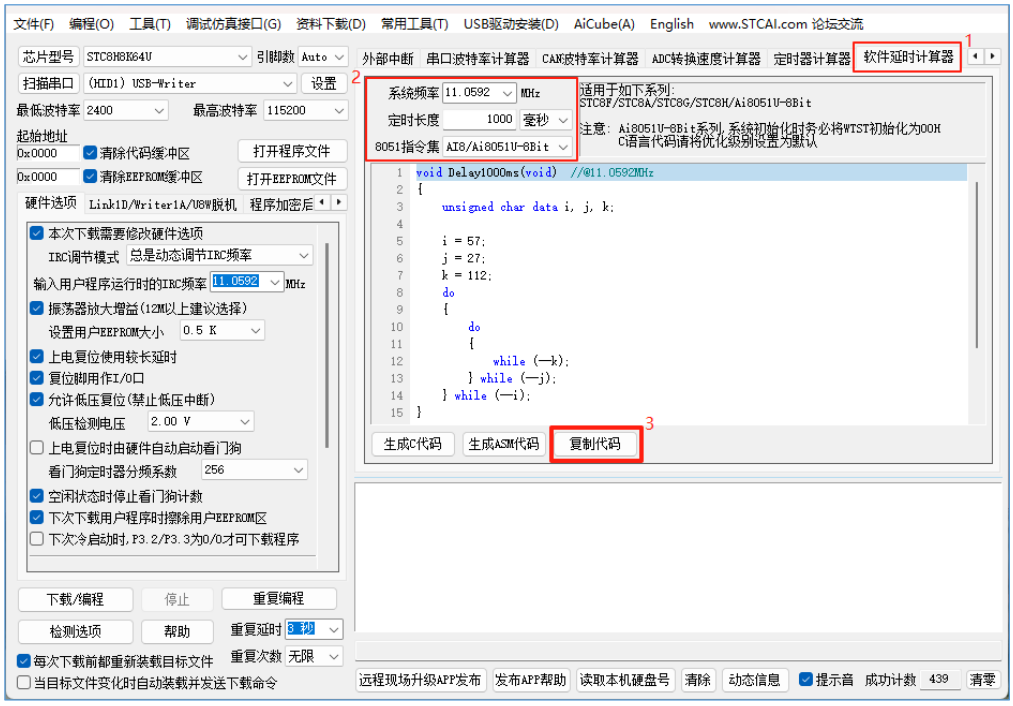
延时函数部分代码就可以使用 **AiCube-ISP** 软件的“**软件延时计算器**”来生成。

例如上述代码中这一段延时函数的代码：

```
void Delay1000ms(void)      //延时 1000ms@11.0592MHz
{
    unsigned char data i, j, k;

    i = 57;
    j = 27;
    k = 112;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

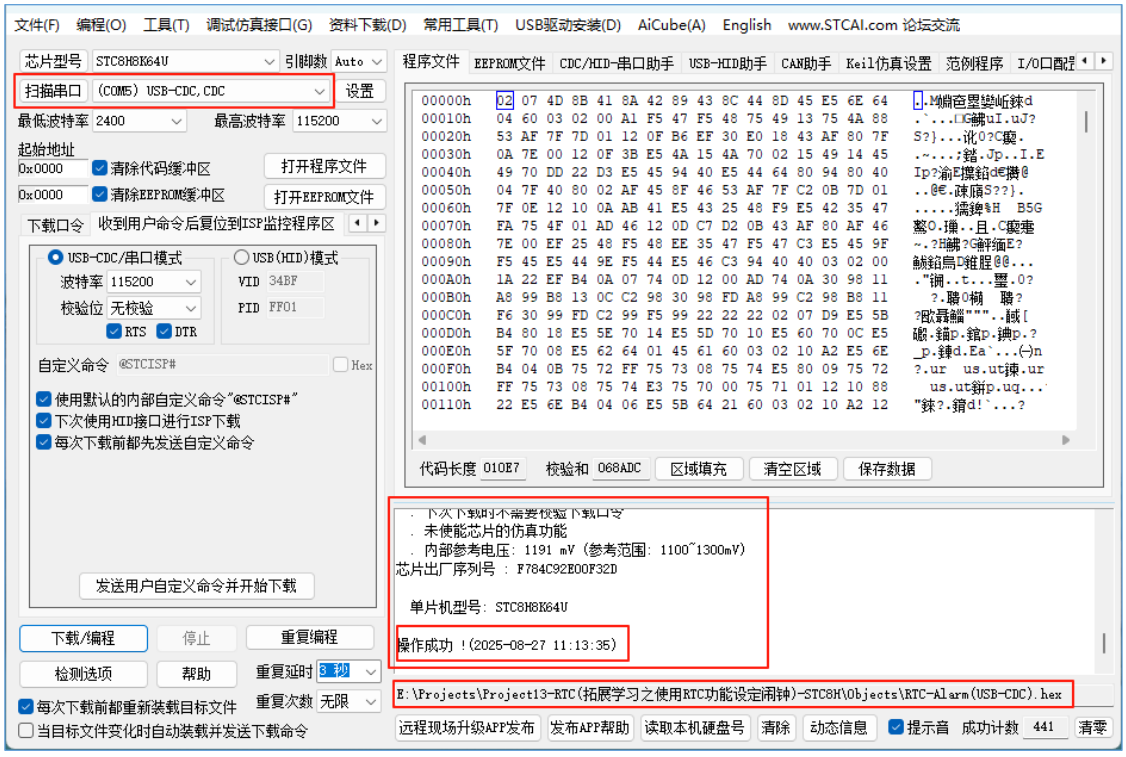
就可以使用 AiCube-ISP 软件的“软件延时计算器”来生成：



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

2.12.1.5　程序下载，观察实验现象

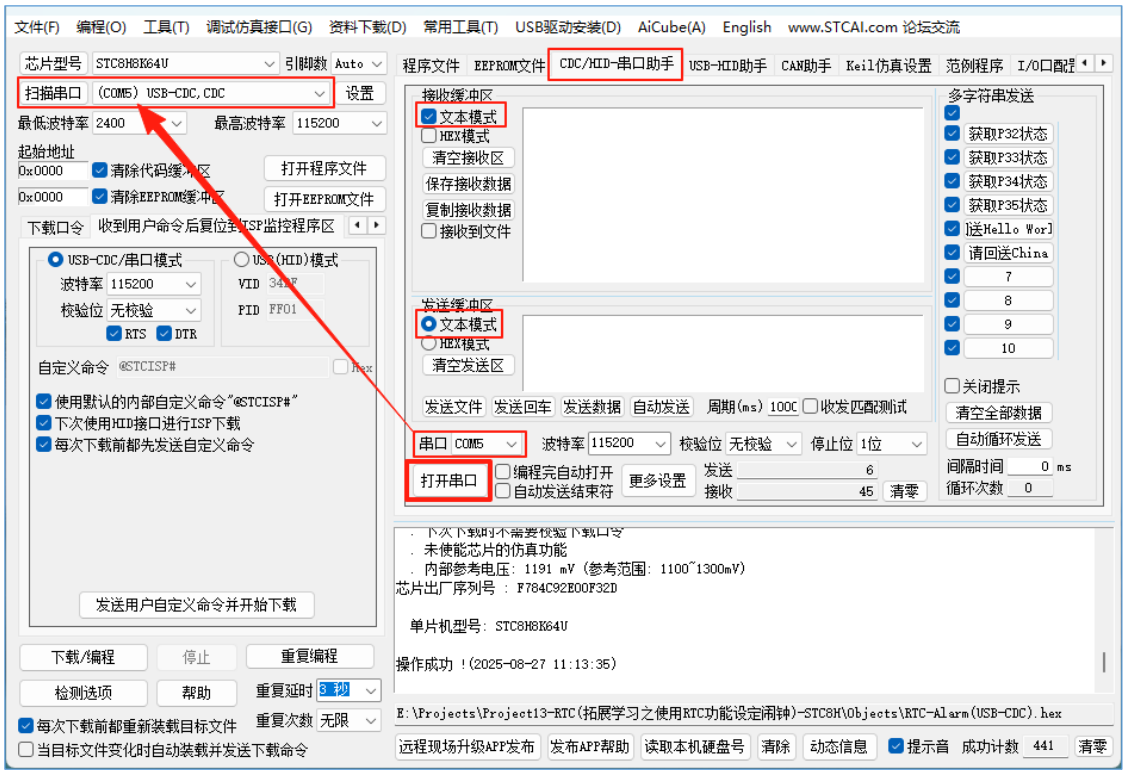
➲ 录入代码，保存，编译。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“RTC-Alarm(USB-CDC).hex”，如下图：



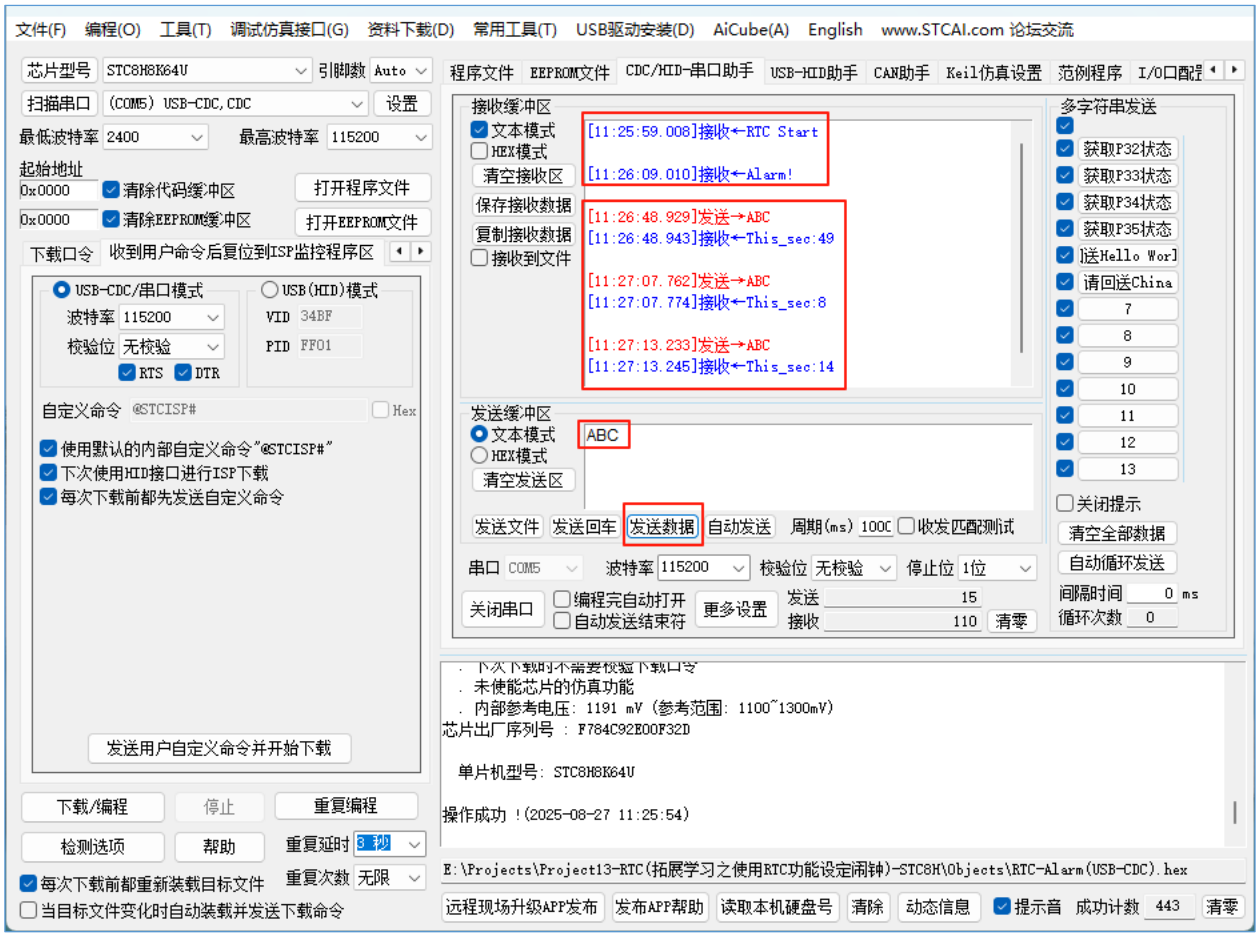
➲ 我们观察到：

➢ “操作成功!”，并且 AiCube-ISP 软件系统的扫描串口窗口出现(COM5)USB-CDC,CDC 端口。表明此时就可以通过这个 USB-CDC 端口进行数据通讯了。

➲ 打开 AiCube-ISP 软件的 CDC 串口助手选项卡，找到“打开串口”按钮：



- 点击“打开串口”按钮
- 我们观察到：
  - 在 AiCube-ISP 软件的 CDC 串口助手窗口，会显示“RTC start”，10s 后显示及“Alarm!”。
  - 在发送缓冲区输入“ABC”，点击“发送数据”，则接收缓冲区显示发送的内容及发送当时的秒值。如下图：



达成实验预设，实验结束。

2.12.2 使用【AiCube】助手实现 RTC 实验(测试 RTC，使用 USB-CDC 打印年/月/日/时/分/秒)

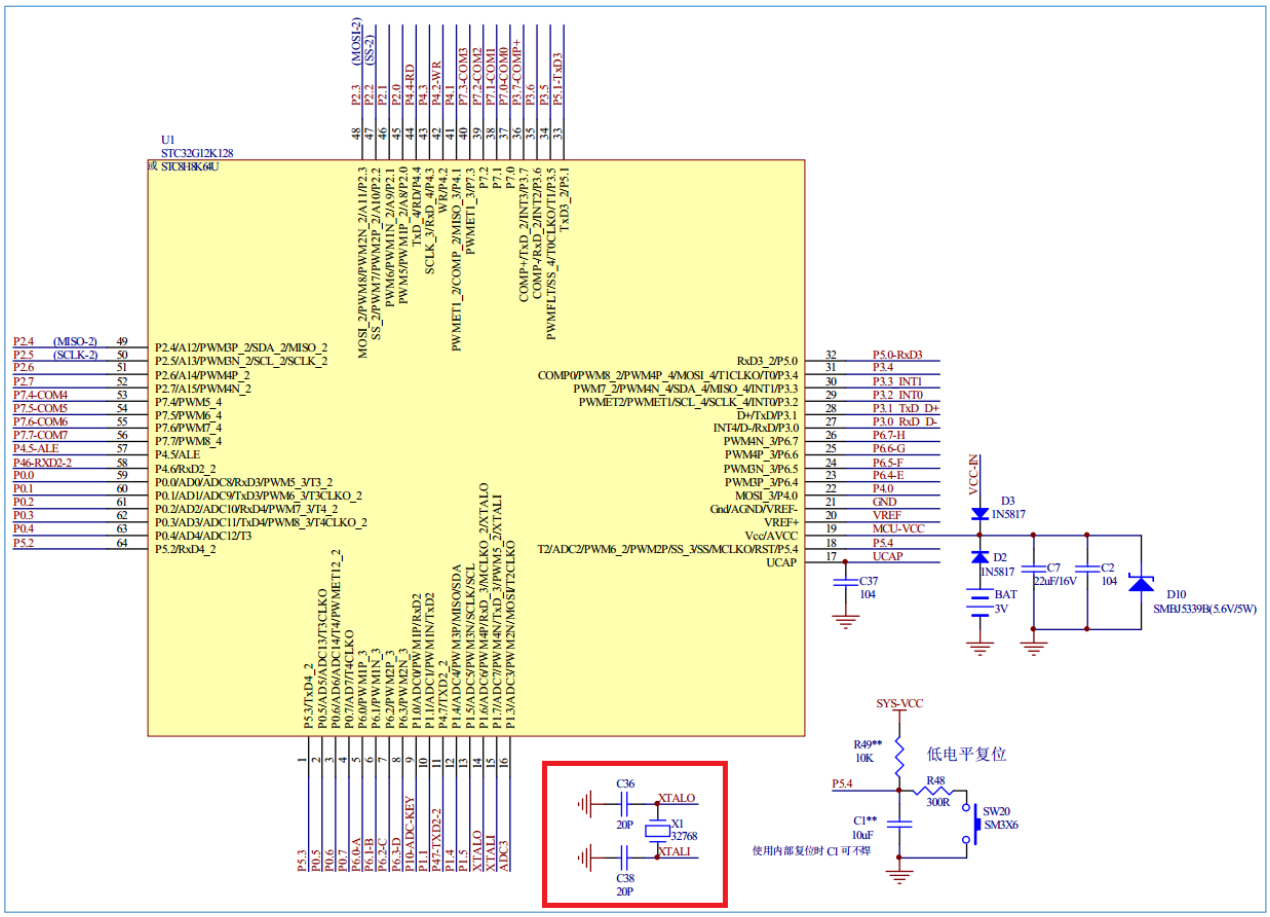
2.12.2.1 实验介绍

测试 STC8H8K64U 芯片的 RTC 功能，产生闹钟中断、天中断、小时中断、分钟中断、秒钟中断，并使用 AiCube-ISP 系统软件的 USB-CDC 功能打印中断数据及年/月/日、时/分/秒数据。

实验观察到的现象：

- 程序运行后，在 AiCube-ISP 系统软件的 USB-CDC 窗口显示闹钟中断，每秒/分/小时/天 产生的中断，并显示当前 RTC 的时间。

2.12.2.2 原理图



2.12.2.3 准备工作--添加 Keil 中断拓展插件

Keil C51/C251 编译器只支持 31 以内的中断号（0~31），超过 31 编译会报错。RTC 的中断向量 RTC_VECTOR 为 36，这就需要 Keil 中断号拓展工具，可将中断号拓展到 254。

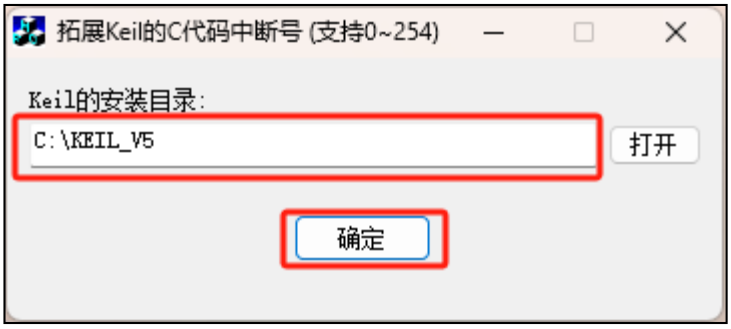
这部分可以到 STCAI 的官网：https://www.stcai.com/gjrj 的软件工具➔工具软件中“Keil 中断拓展插件”进行下载。



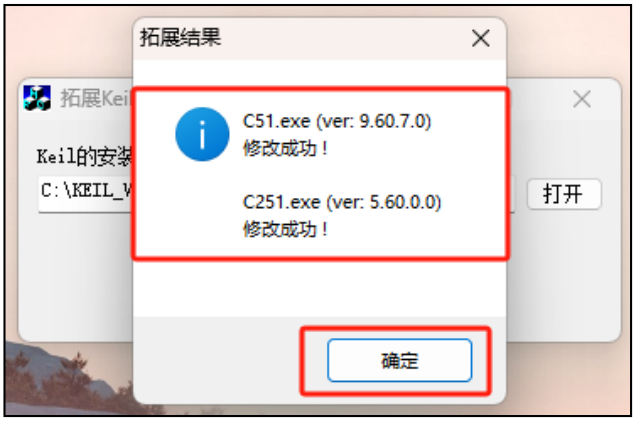
下载完成后，打开压缩包“拓展 Keil 的 C 代码中断号.zip”，找到“拓展 Keil 的 C 代码中断号.exe”，这是绿色免安装软件。



直接双击“拓展 Keil 的 C 代码中断号.exe”，出现如下界面，选择安装路径，按照默认直接安装到 C:\Keil_V5 目录下，点击“确定”



如下图，点击“确定”，Keil 的 C 代码中断号到 254



2.12.2.4　安装 AiCube-ISP 下载/编程/烧录/软件

STCAI 在最新版的 AiCube-ISP 软件中，推出【AiCube】项目创建工具，目前支持 **Ai8051U-32Bit、Ai8051U-8Bit、STC32G12K128、STC8H8K64U、STC15F2K60S2、STC15W4K32S4** 和 **STC89C52RC** 系列各功能模块的智能化配置，并生成程序框架。这项功能在程序员建立项目时，简化了项目的初始化难度，提高效率，避免因不太熟悉或不完全了解该单片机的性能而忽略个别配置，引起编译错误等问题。尤其对刚涉足单片机的初学者，此工具不仅引导其了解单片机编程流程，还极大的降低了入门门槛，缩短了摸索的时间，提高入门的兴趣和信心。现在我们使用这个工具来生成实现此实验功能的配置和代码。

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj



下载下来的是压缩包文件图标，如下图：



解压缩后的 AiCube-ISP 文件图标，如下图：



如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

2.12.2.5　使用【AiCube】助手创建项目

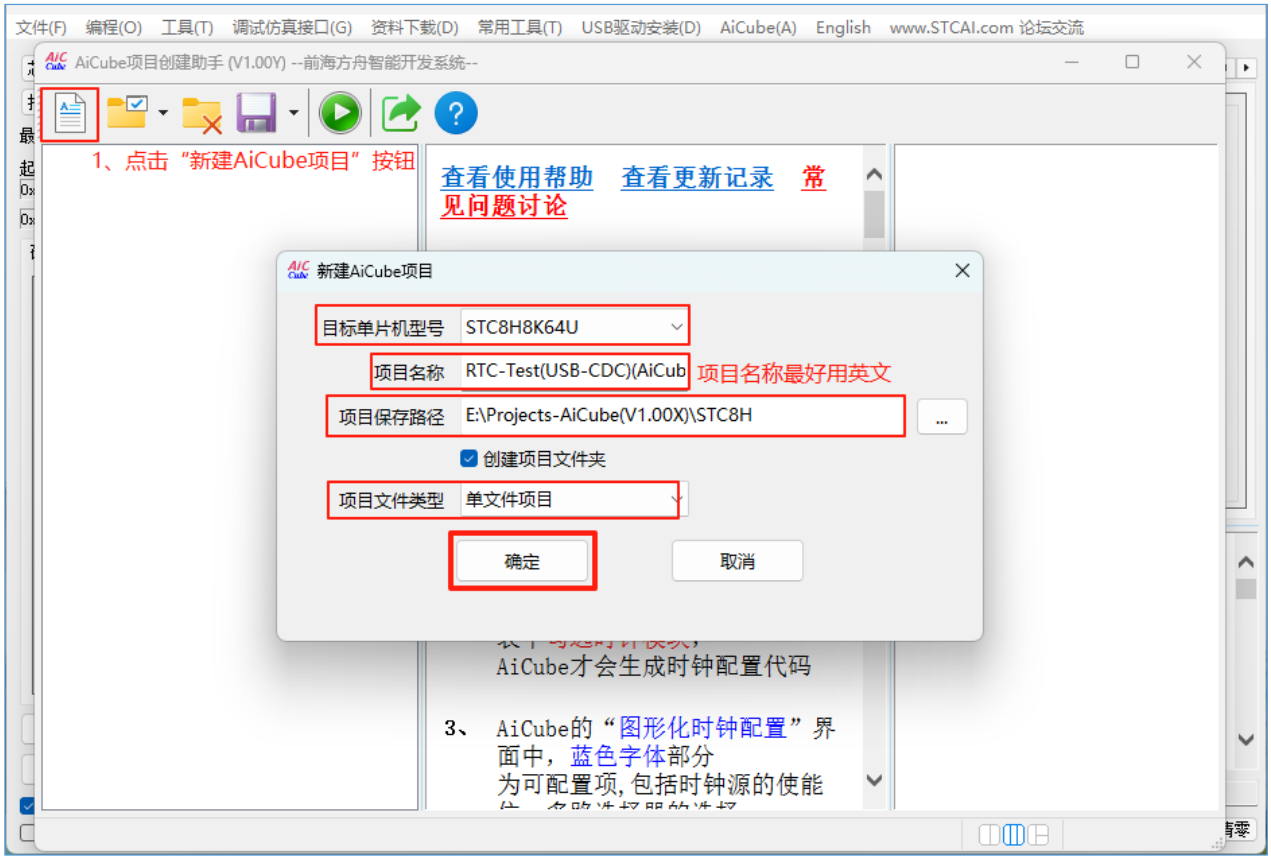
➲　双击图标，打开 AiCube-ISP 软件界面，如下图：



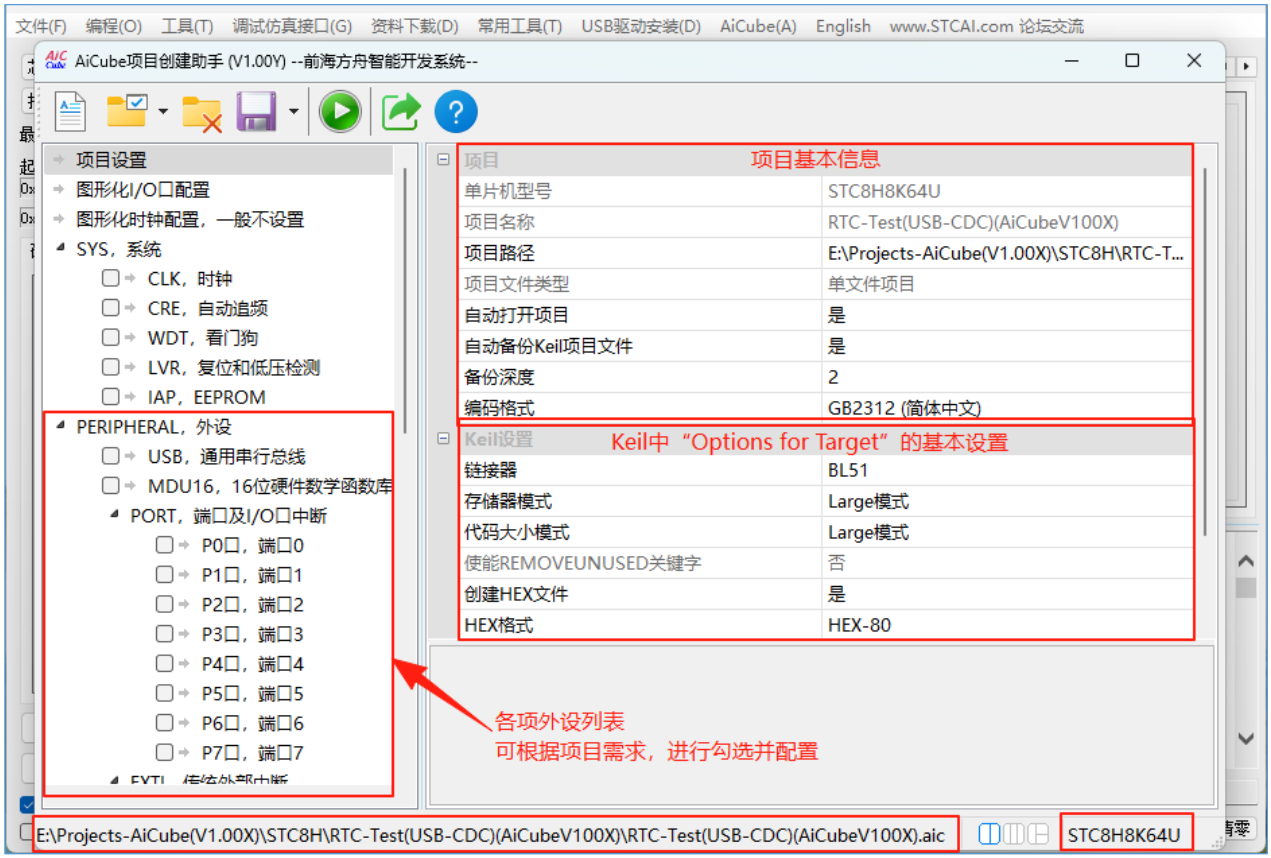
➲　点击【AiCube（A）】，弹出如下窗口：



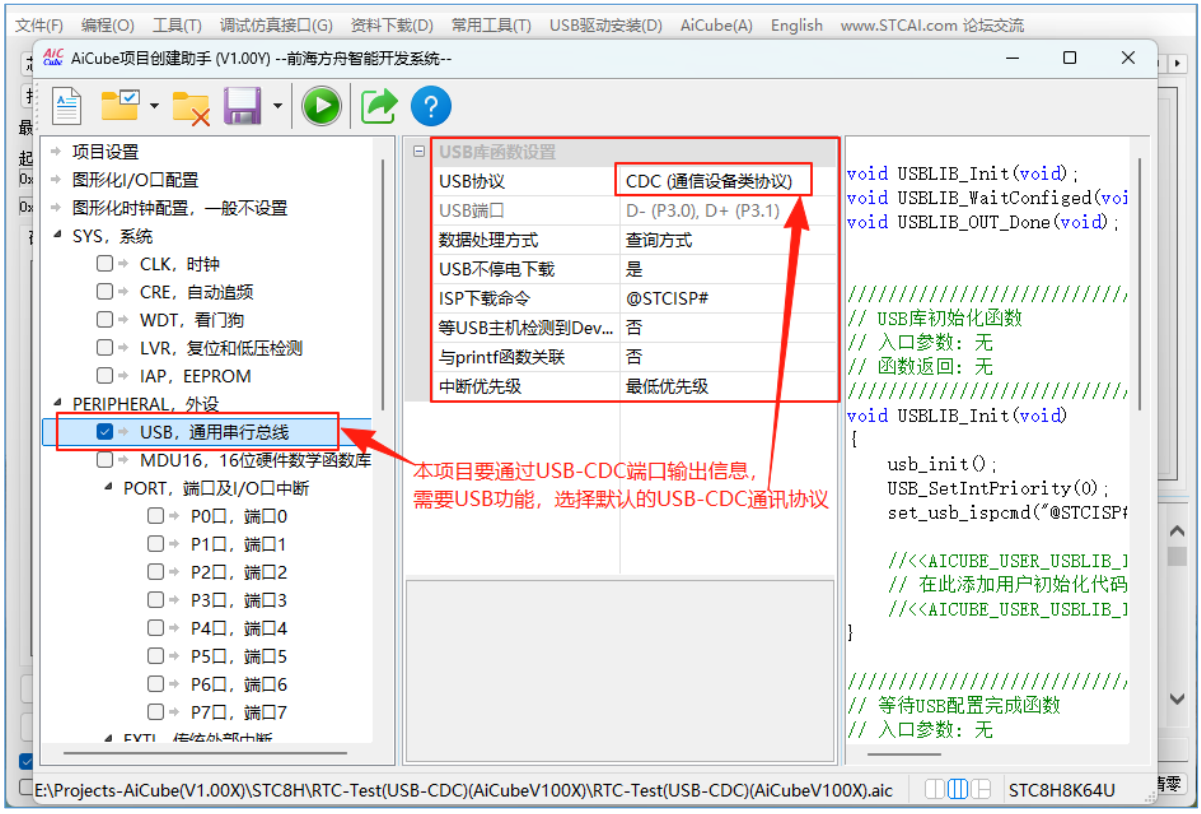
➲ **点击“新建 AiCube 项目”按钮**，选择并填写项目基本信息，建议项目名称用英文！



➲ **选择并填写项目基本信息，**建议项目名称用英文！**点击“确定”按钮，**出现如下配置界面：

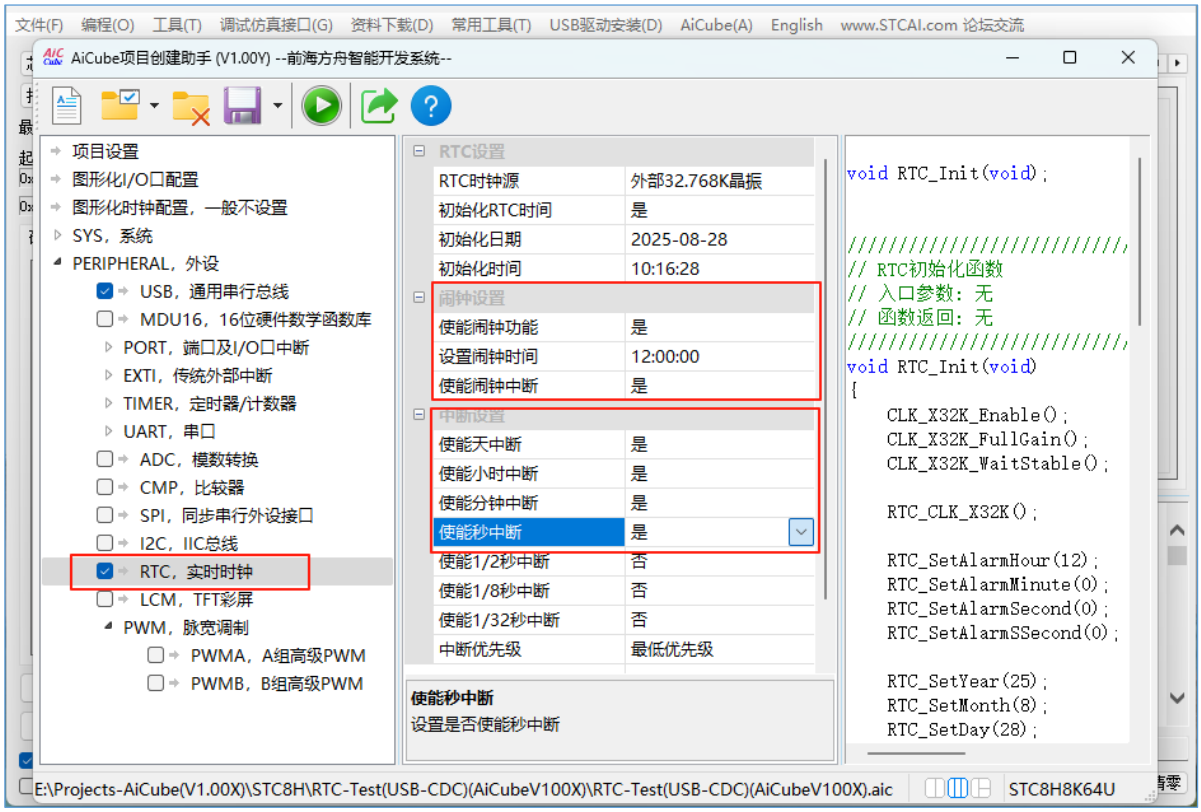


➲ 因本项目用到 USB 功能，要加载 USB 库函数、代码中要有 USB 初始化、USB-CDC 协议等，使用【AiCube】助手根据本项目的实际需求，**勾选“USB，通用串行总线”**：

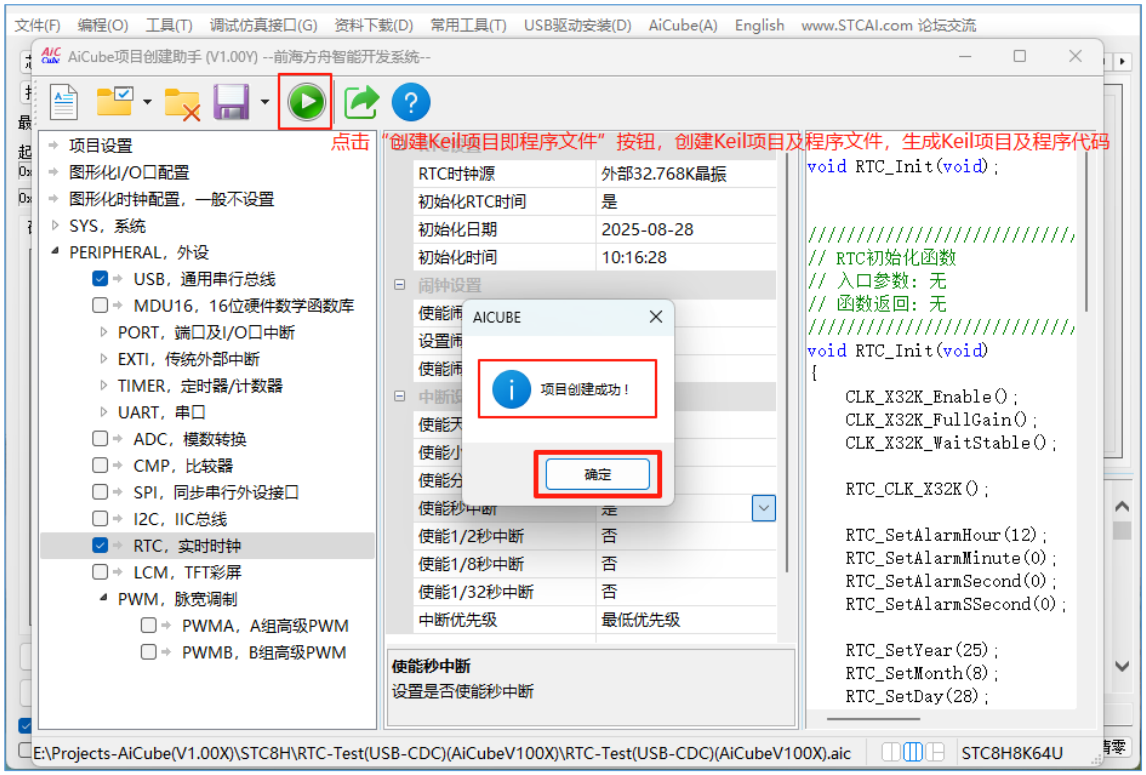


本实验是测试 STC8H8K64U 芯片的 RTC 功能，产生闹钟中断、天中断、小时中断、分钟中断、秒钟中断，并使用 USB-CDC 打印中断数据及年月日、时分秒数据。

➲ 当然要设置 RTC 模块，设置闹钟中断、天中断、小时中断、分钟中断、秒钟中断为“是”，如下图：



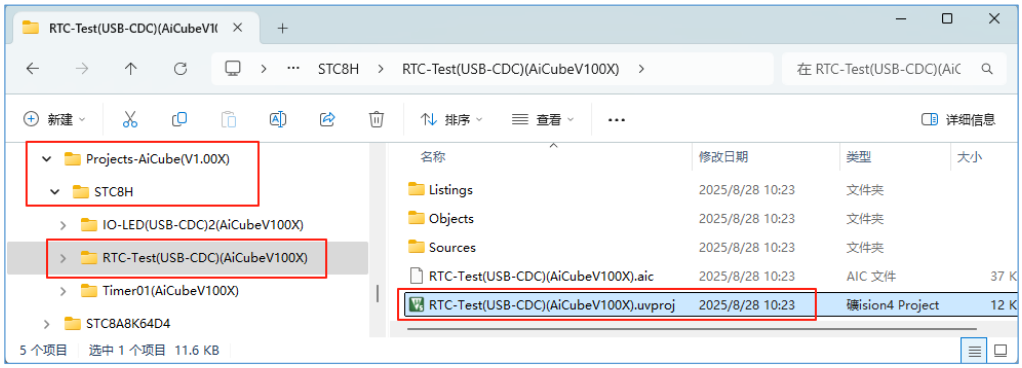
➲ 点击“创建 Keil 项目及程序文件”按钮，【AiCube】助手提示：“项目创建成功！”，如下图：



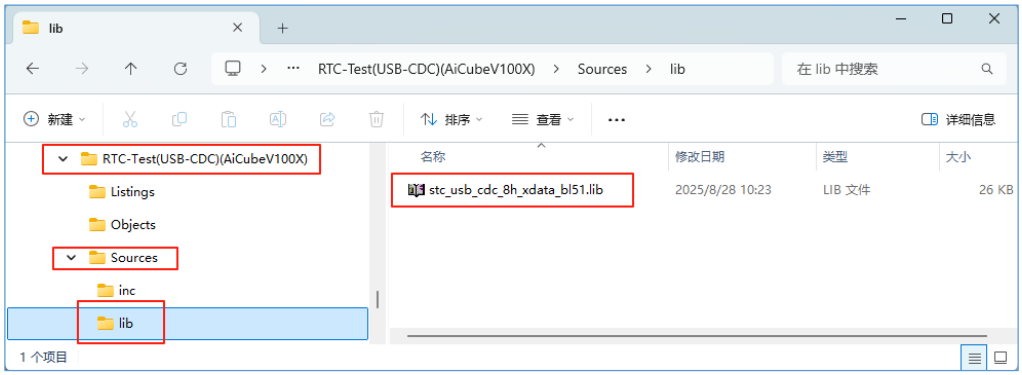
➲ 点击“确定”，立即自动打开 Keil 软件，进入该项目的 Keil 集成开发界面，即【AiCube】成功帮我们创建了项目及文件目录“RTC-Test(USB-CDC)(AiCubeV100X)”。至此，可以关闭【AiCube】。

➲ 找到创建的项目目录

发现通过上述操作，【AiCube】助手已经帮我们建立了项目目录及 RTC-Test(USB-CDC)(AiCubeV100X).uvproj 文件，如下图：

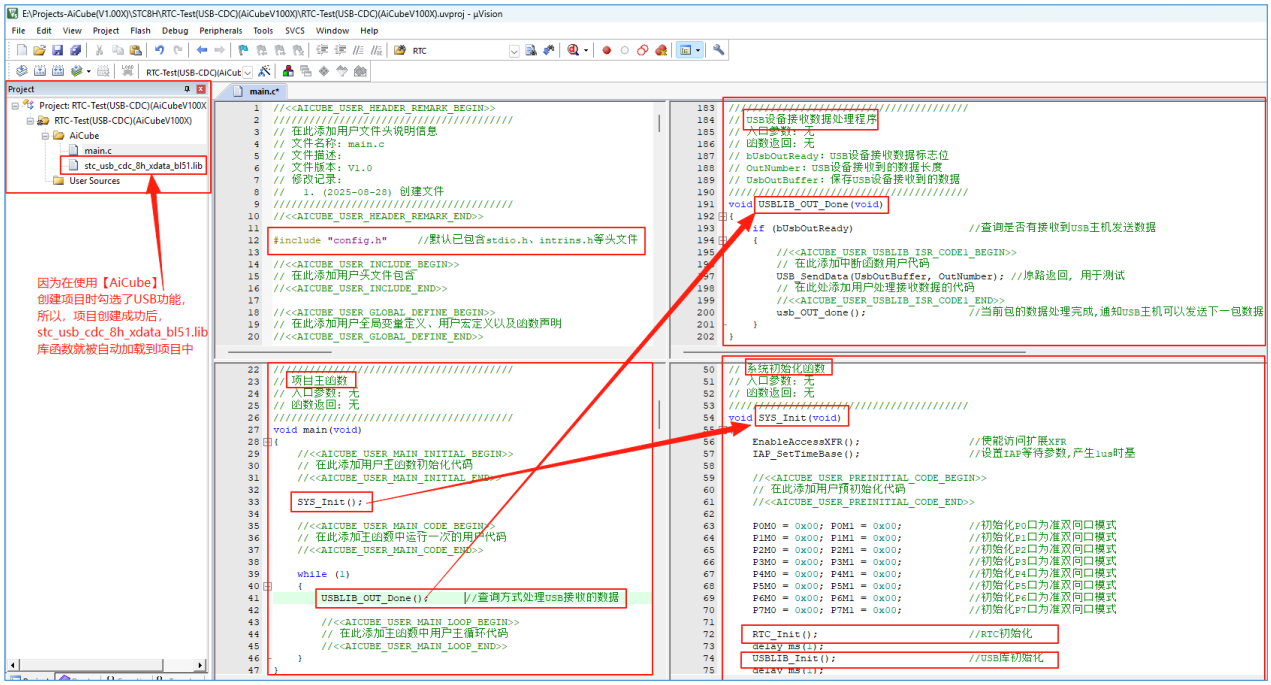


因为勾选了 USB 选项，打开 Sources>lib 子目录，发现，【AiCube】助手连 USB 库函数也已经加载了：



➲ **观察已打开的 Keil 项目文件：**RTC-Test(USB-CDC)(AiCubeV100X)**.uvproj**

如下图（为方便查看，使用菜单“Window”→“Split”将原窗口拆分成 4 个窗口，全代码观察）：



根据在【AiCube】助手中的配置，从上图中我们看到【AiCube】助手创建的本实验项目中有了 main.c，还已经自动加载了 USB 库函数（stc_usb_cdc_8h_xdata_bl51.lib）。main.c 中按模块化设计，已经包含了：必要的头文件、主函数、系统初始化函数、微秒/毫秒延时函数、RTC 初始化函数、RTC 中断服务程序、USB 库初始化函数、等待 USB 配置完成函数、USB 设备接收数据处理程序。

2.12.2.6　　增加实验项目代码

为测试 STC8H8K64U 芯片的 RTC 功能，产生闹钟中断、天中断、小时中断、分钟中断、秒钟中断，并使用 USB-CDC 打印中断数据及年月日、时分秒数据。我们现在需要：

➲ 在主函数前定义中断标志位（闹钟中断、天中断、小时中断、分钟中断、秒钟中断）；

```
bit fAlarm = 0;          //RTC 闹钟查询标志，在中断程序中设置，主循环中查询并清除
bit fDay = 0;            //RTC 天查询标志，在中断程序中设置，主循环中查询并清除
bit fHour = 0;           //RTC 小时查询标志，在中断程序中设置，主循环中查询并清除
bit fMinute = 0;         //RTC 分钟查询标志，在中断程序中设置，主循环中查询并清除
bit fSecond = 0;         //RTC 秒钟查询标志，在中断程序中设置，主循环中查询并清除
```

➲ 在主函数的 while 循环中，加入 RTC 各种中断判断代码；

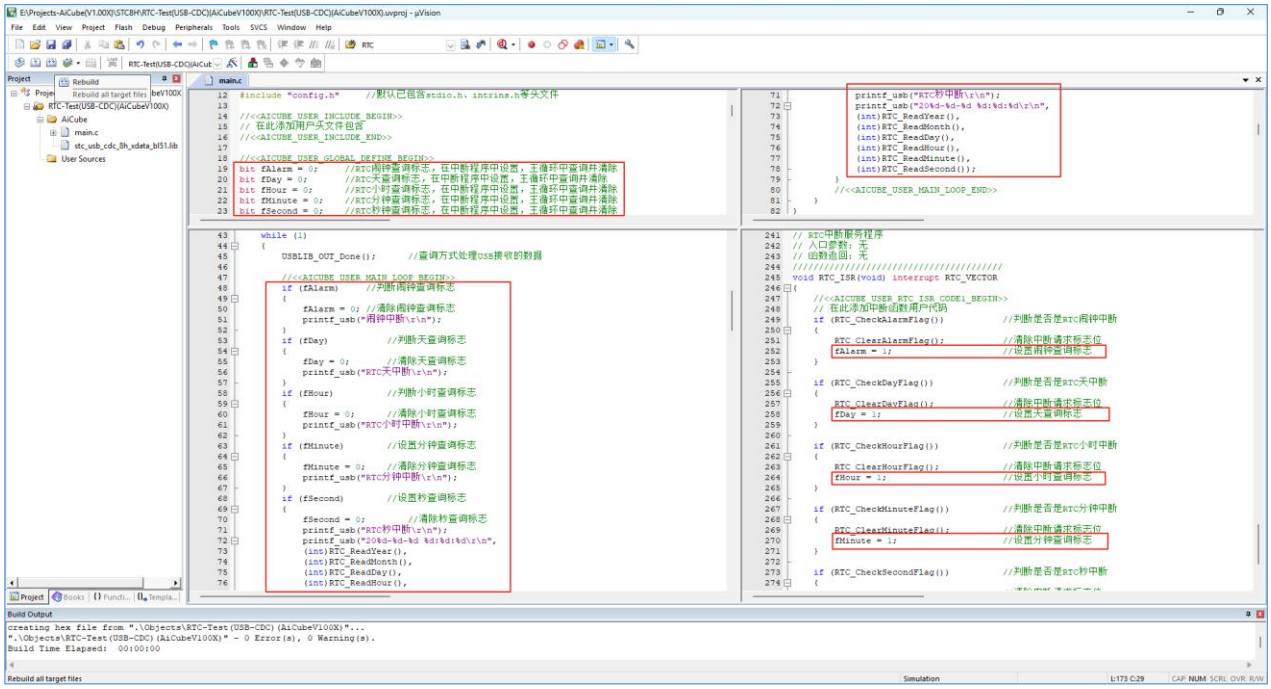
```
if (fAlarm)              //判断闹钟查询标志
{
    fAlarm = 0;    //清除闹钟查询标志
    printf_usb("闹钟中断\r\n");
}
if (fDay)                //判断天查询标志
{
    fDay = 0;      //清除天查询标志
    printf_usb("RTC 天中断\r\n");
}
```

```
    if (fHour)          //判断小时查询标志
    {
        fHour = 0;      //清除小时查询标志
        printf_usb("RTC 小时中断\r\n");
    }
    if (fMinute)        //设置分钟查询标志
    {
        fMinute = 0;    //清除分钟查询标志
        printf_usb("RTC 分钟中断\r\n");
    }
    if (fSecond)        //设置秒查询标志
    {
        fSecond = 0;    //清除秒查询标志
        printf_usb("RTC 秒中断\r\n");
        printf_usb("20%d-%d-%d %d:%d:%d\r\n",
                                            (int)RTC_ReadYear(),
                                            (int)RTC_ReadMonth(),
                                            (int)RTC_ReadDay(),
                                            (int)RTC_ReadHour(),
                                            (int)RTC_ReadMinute(),
                                            (int)RTC_ReadSecond());
    }
```

➲ 在“RTC 中断服务程序”子函数中，补充设置中断标志的代码；

```
    fAlarm = 1;         //设置闹钟查询标志
    fDay = 1;           //设置天查询标志
    fHour = 1;          //设置小时查询标志
    fMinute = 1;        //设置分钟查询标志
    fSecond = 1;        //设置秒查询标志
```

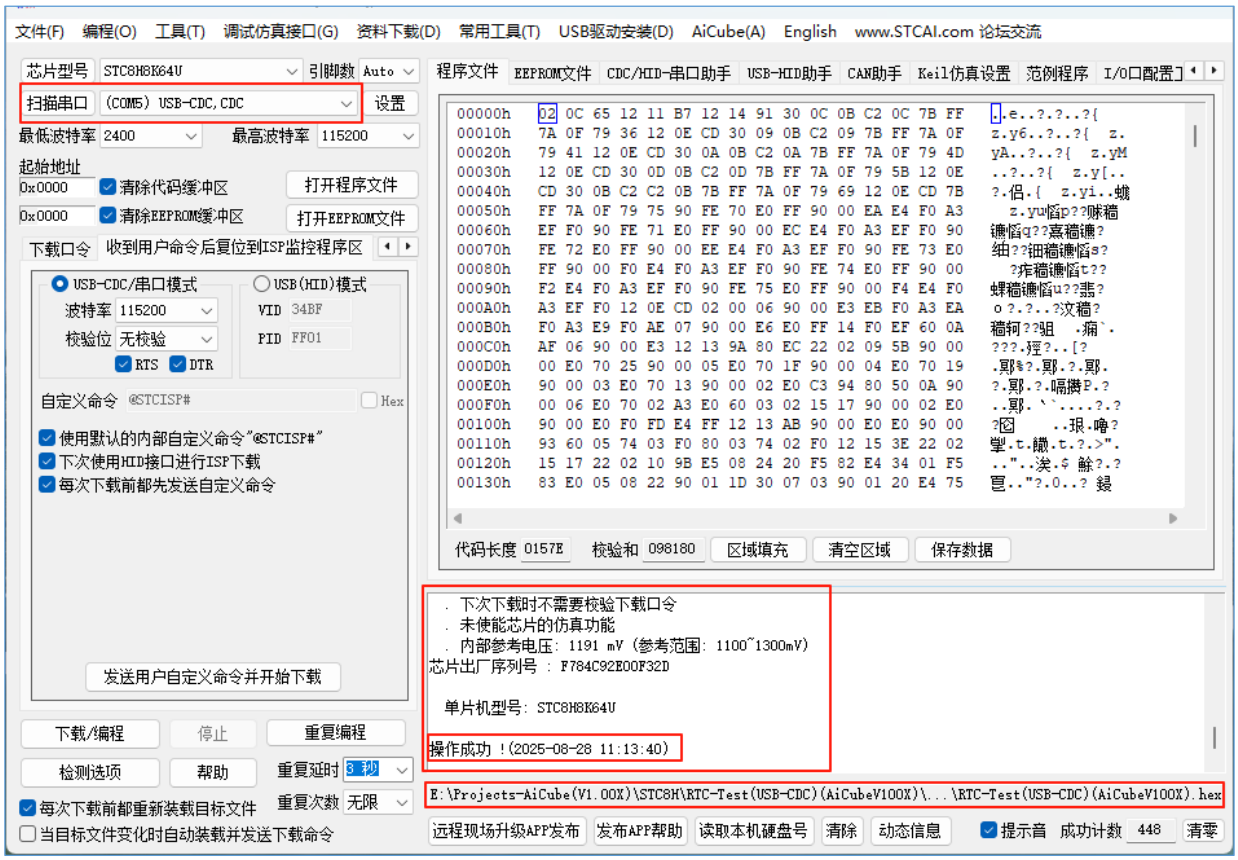
➲ 保存，编译，如下图（使用菜单“Window”→“Split”将原窗口拆分成 4 个窗口，对比观察）：



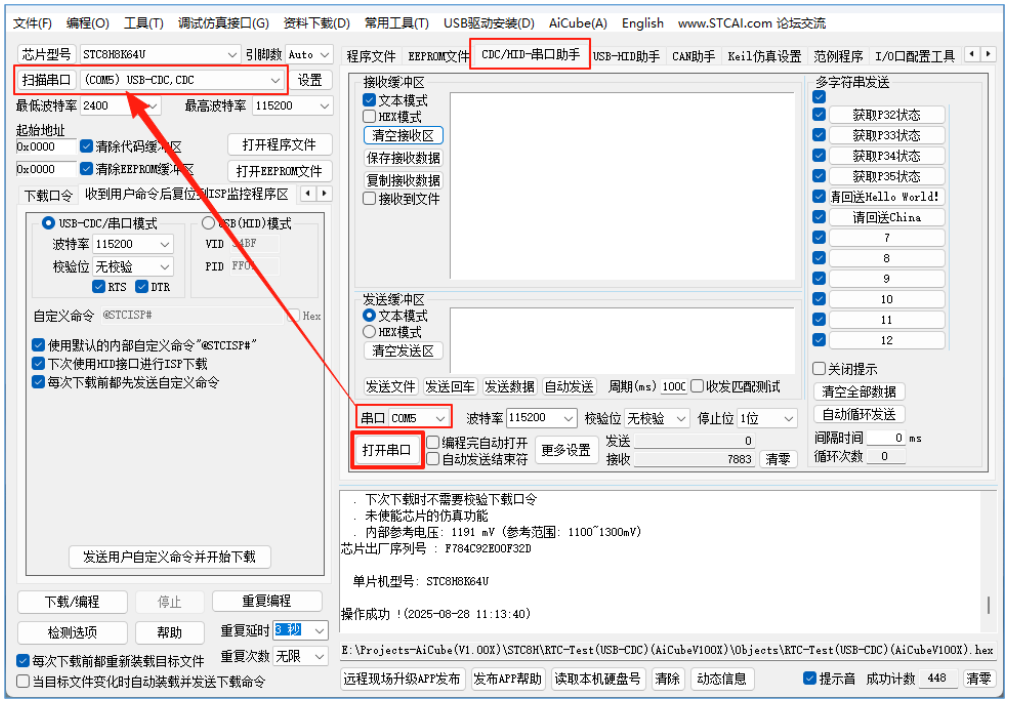
2.12.2.7　使用 AiCube-ISP 软件系统，连接 STC8H8K64U 实验箱，观察实验现象，验证实验效果

➲ 下载程序到 STC8H8K64U 实验箱

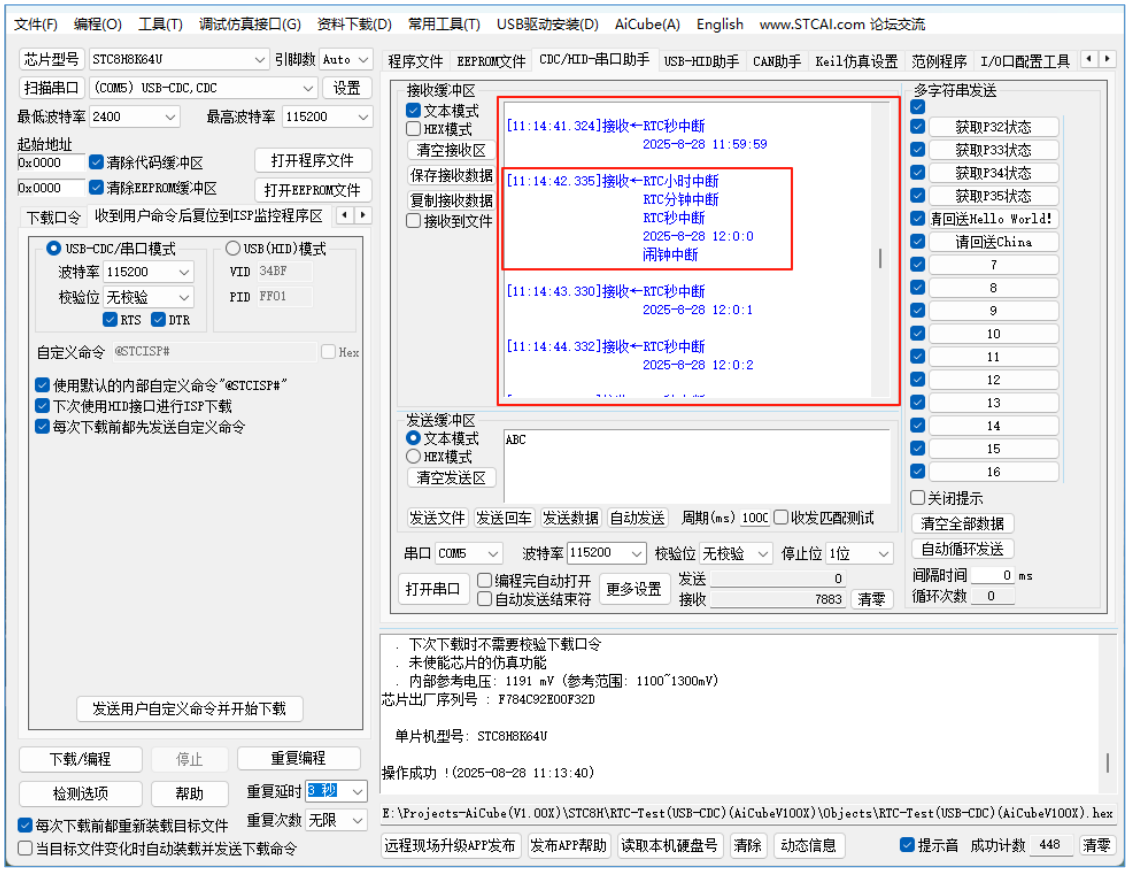
➢ 连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“RTC(USB-CDC)(AiCube).hex”到实验箱，如下图：



- 打开 AiCube-ISP 软件的 CDC 串口助手选项卡
  - 找到“打开串口”按钮



- 点击“打开串口”
- 我们观察到：
  - 在 AiCube-ISP 软件的 CDC 串口助手窗口，会显示年月日及时钟中断、分钟中断、秒钟中断。
  - 并显示发送的内容及发送当时的秒值。如下图：



达成实验预设，实验结束。

有了【AiCube】助手，单片机编程就是这么简单！快用起来吧！

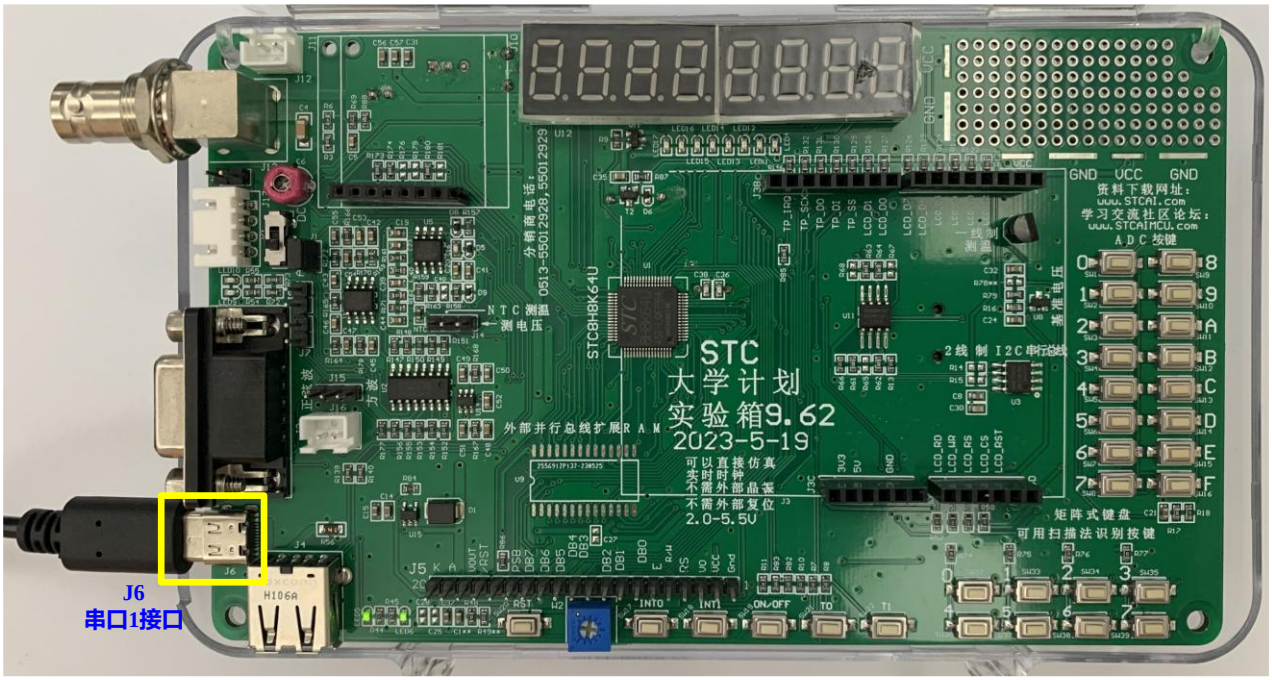
2.13 串口实验 -- 串口通讯(硬件 UART1 与电脑收发通讯)

2.13.1 实验介绍

1、 了解串口 1/UART1 通讯原理
2、 熟悉 STC8H8K64U 实验箱及其原理图
3、 熟练使用 AiCube-ISP 软件
4、 了解 USB Link1D 工具的原理及使用
5、 熟悉如何管理多文件项目

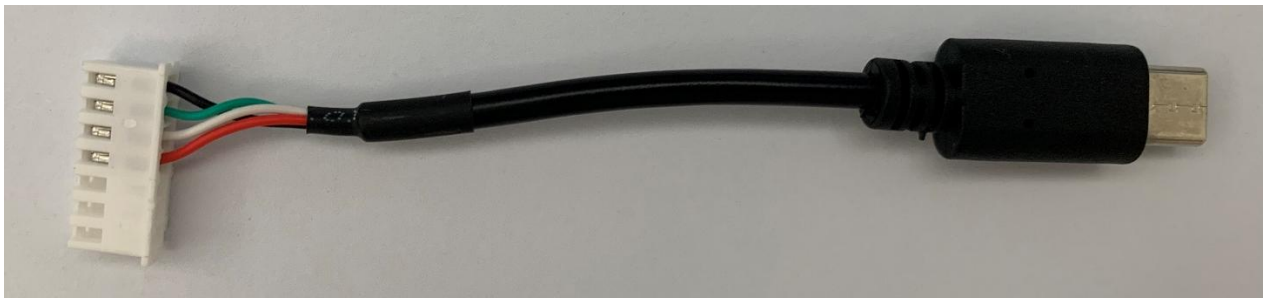
2.13.1.1 认识 STC8H8K64U 实验箱

STC8H8K64U 的实验箱正面图：

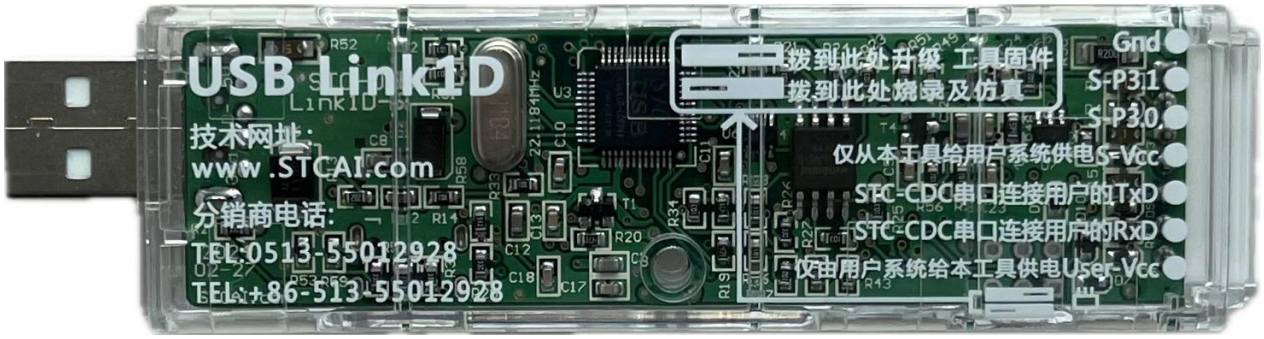


J6： 串口 1 接口。本实验需要用专门定制的 USB 转串口/TTL（连接 USB Link1D 工具） - TypeC（连接实验箱 J6）接口线相连。（详见 J6 原理图）

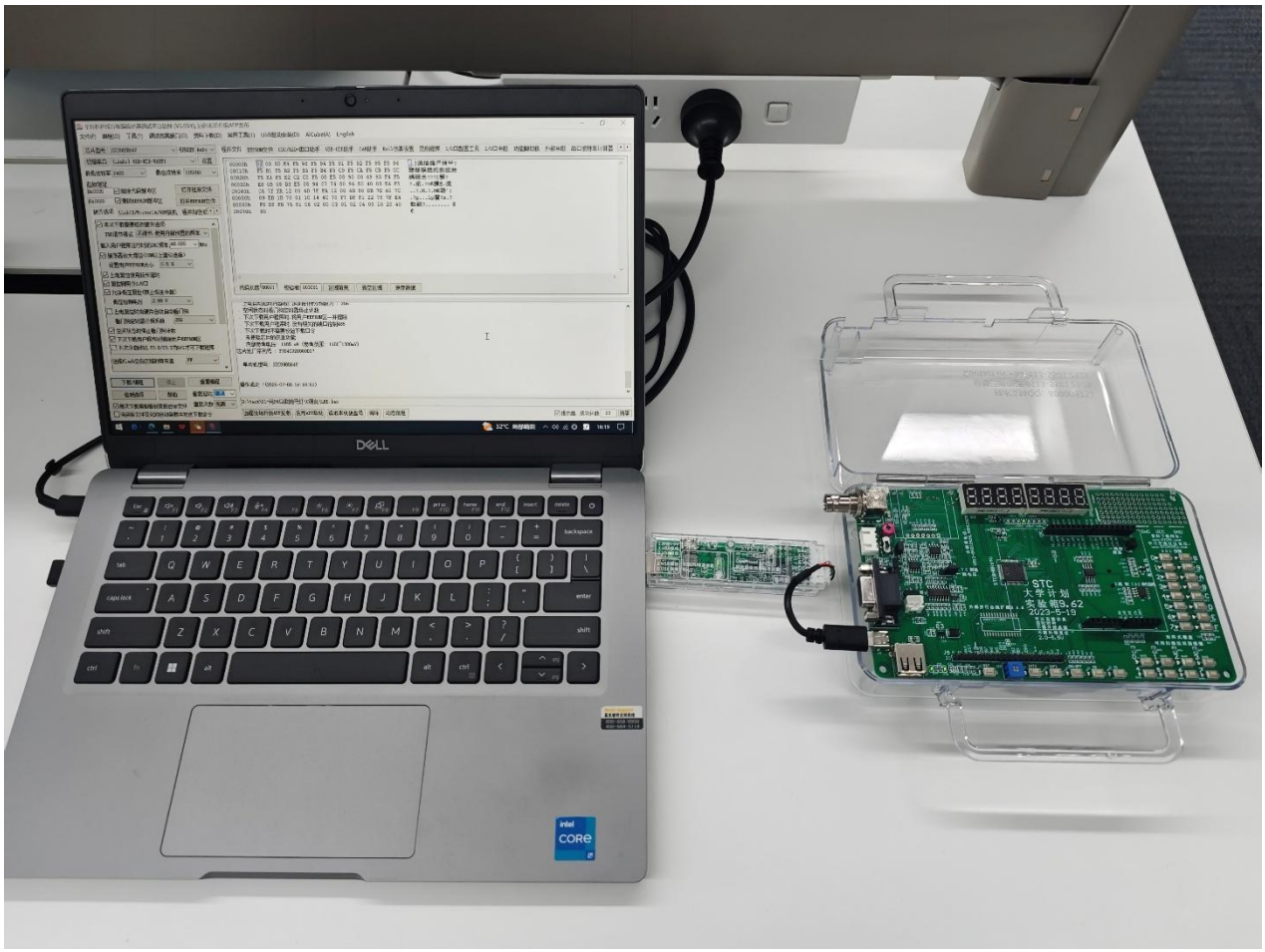
2.13.1.2 专门定制的 USB 转串口/TTL - TypeC 接口线



2.13.1.3 USB Link1D 工具



2.13.1.4 STC8H8K64U 实验箱由 TTL - TypeC 接口线通过 USB Link1D 工具与电脑连接图



2.13.1.5 Keil 环境下多文件项目管理说明

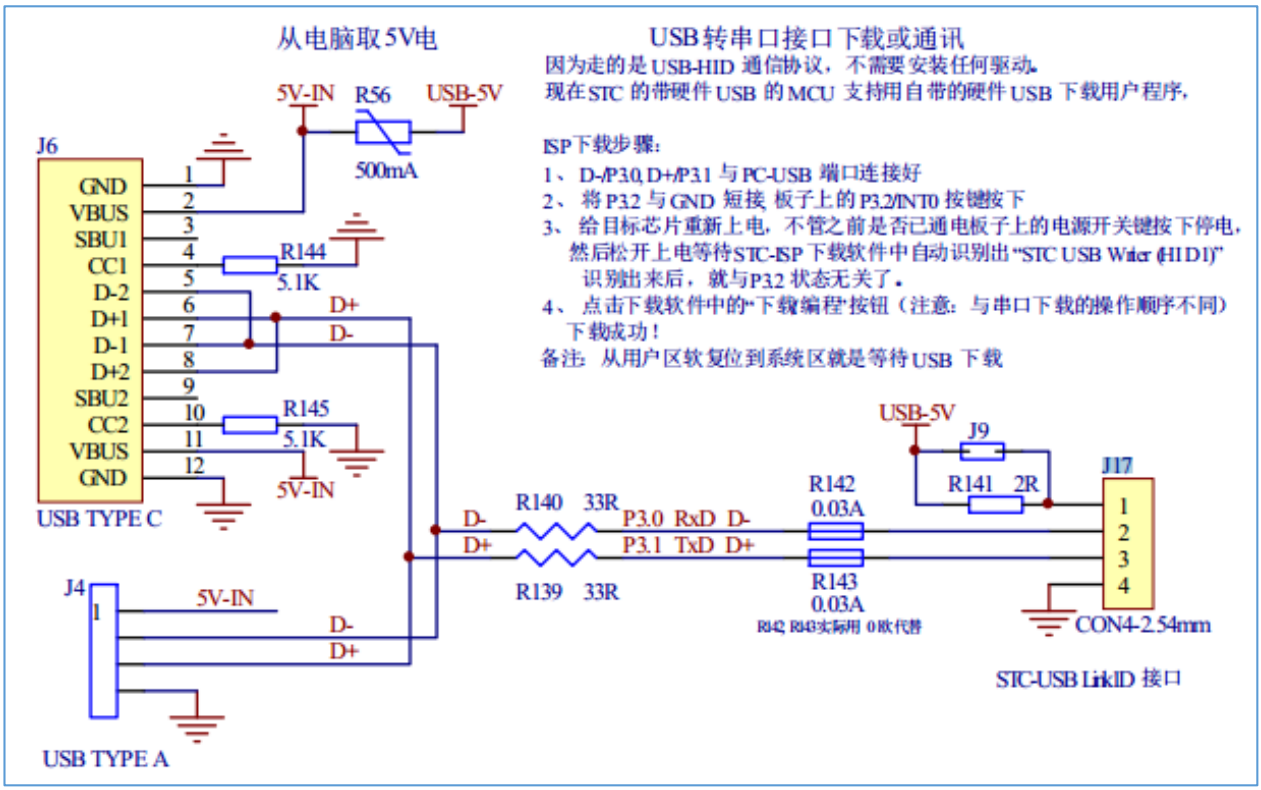
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以方便分工合作、代码复用、模块化管理、

增强可读性和可维护性。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

```
#ifndef XXXX
#define XXXX
...
#endif
```

2.13.2 原理图



2.13.3 实验程序代码

2.13.5.1. main.c -- 程序主函数

```
//main.c
//程序主函数
//USB 转串口：TTL-TypeC 连接线连接电脑与实验箱 J6 接口，串口助手打开 CDC1 对应的 COM 口
```

```
#include "config.h"       //头文件中已包含 stc8h.h 以及其他头文件

void main()
{
    SYS_Init();           //系统初始化
    while (1)
    {
        if((TX1_Cnt != RX1_Cnt) && (!B_TX1_Busy))
            //发送计数不等于接收计数（说明收到的数据没有全部发送出去），并且发送空闲
        {
            SBUF = RX1_Buffer[TX1_Cnt++];     //发送缓冲区里收到的数据
            B_TX1_Busy = 1;                   //设置发送忙标志
        }
    }
}

void SYS_Init(void)      //系统初始化函数
{
    P_SW2 |= 0x80;           //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;       //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //初始化 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //初始化 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //初始化 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //初始化 P7 口为准双向口模式

    P_SW1 &= ~0xc0;       //UART1 通道选择, 00: P3.0 P3.1, 01: P3.6 P3.7, 10: P1.6 P1.7, 11: P4.3 P4.4

    Uart1_Init();             //UART1 接口初始化

    EA = 1;                   //总中断允许位打开
}
```

2.13.5.2. uart.c -- 初始化 UART 接口

```
//uart.c
//初始化 UART 接口
//中断处理 UART 收发数据结果

#include "config.h"       //头文件中已包含 stc8h.h 以及其他头文件

u8   TX1_Cnt;             //发送计数
```

```
u8   RX1_Cnt;          //接收计数
bit  B_TX1_Busy;       //发送忙标志

u8   xdata RX1_Buffer[256];      //接收缓冲

void Uart1_Isr(void) interrupt 4
{
    if (TI)                     //检测串口 1 发送中断
    {
        TI = 0;                 //清除串口 1 发送中断请求位
        B_TX1_Busy = 0;         //清除发送忙标志
    }

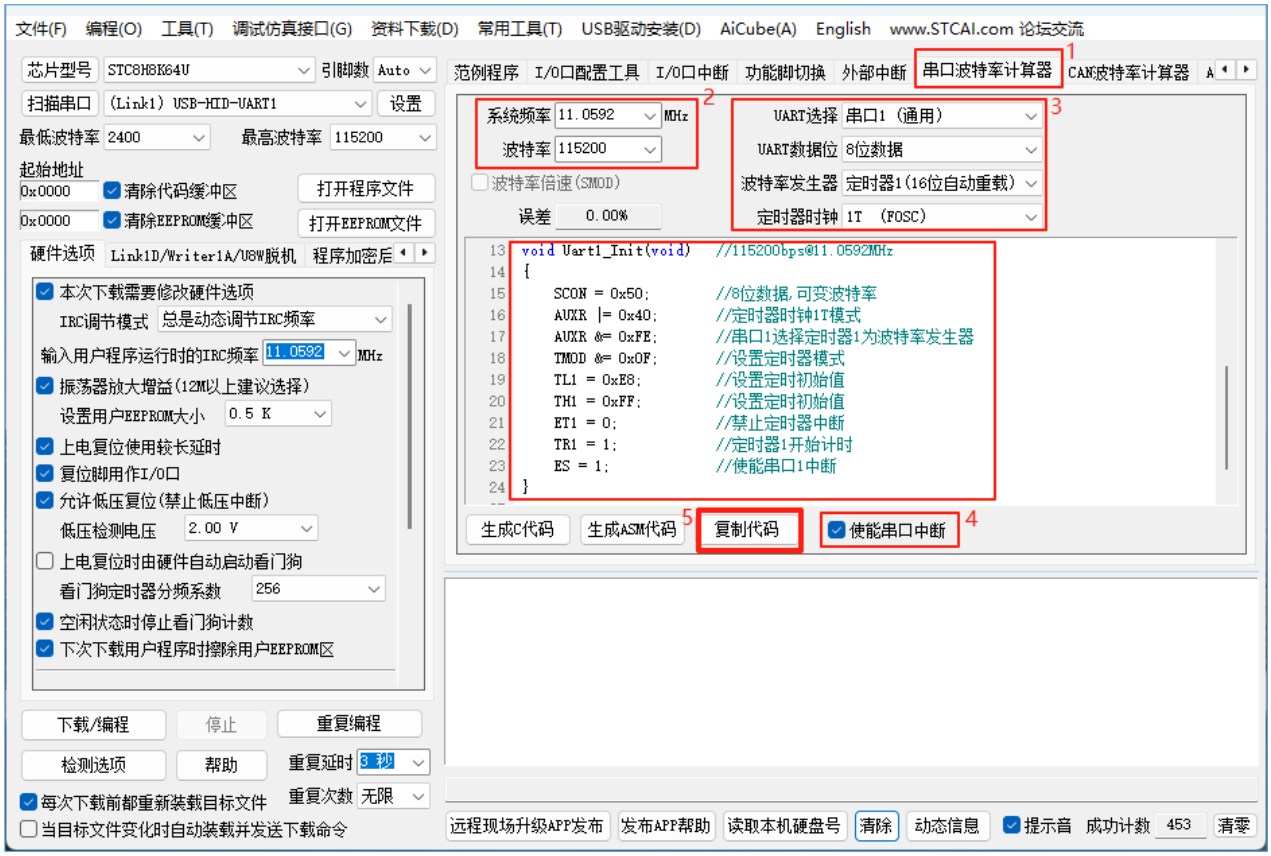
    if (RI)                     //检测串口 1 接收中断
    {
        RI = 0;                 //清除串口 1 接收中断请求位
        RX1_Buffer[RX1_Cnt++] = SBUF;    //将收到数据存入缓冲区
    }
}

void Uart1_Init(void)           //115200bps@11.0592MHz
{
    SCON = 0x50;                //8 位数据,可变波特率
    AUXR |= 0x40;               //定时器时钟 1T 模式
    AUXR &= 0xFE;               //串口 1 选择定时器 1 为波特率发生器
    TMOD &= 0x0F;               //设置定时器模式
    TL1 = 0xE8;                 //设置定时初始值
    TH1 = 0xFF;                 //设置定时初始值
    ET1 = 0;                    //禁止定时器中断
    TR1 = 1;                    //定时器 1 开始计时
    ES = 1;                     //使能串口 1 中断

    B_TX1_Busy = 0;             //变量初始化
    TX1_Cnt = 0;
    RX1_Cnt = 0;
}
```

【小技巧 -- 灵活运用 AiCube-ISP 软件工具自动生成代码】

上述串口波特率部分代码就可以使用 **AiCube-ISP** 软件的“**串口波特率计算器**”来生成。
如下图：



将复制出来的内容直接粘贴到代码文件中，不用手动输入。

2.13.5.3. config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__          //防止头文件被重复包含
#define __CONFIG_H__

#define HIRC        11059200UL
#define FOSC        11059200UL
#define SYSCLK      FOSC
#define MAIN_Fosc   FOSC

#include <stc8h.h>            //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"              //包含项目头文件
#include "uart.h"

void SYS_Init(void);          //函数声明

#endif
```

2.13.5.4. uart.h -- 项目配置文件的头文件

```
#ifndef __UART_H__          //防止头文件被重复包含
#define __UART_H__

extern    u8    TX1_Cnt;              //发送计数
extern    u8    RX1_Cnt;              //接收计数
extern    bit   B_RX1_TimeOut;        //发送忙标志

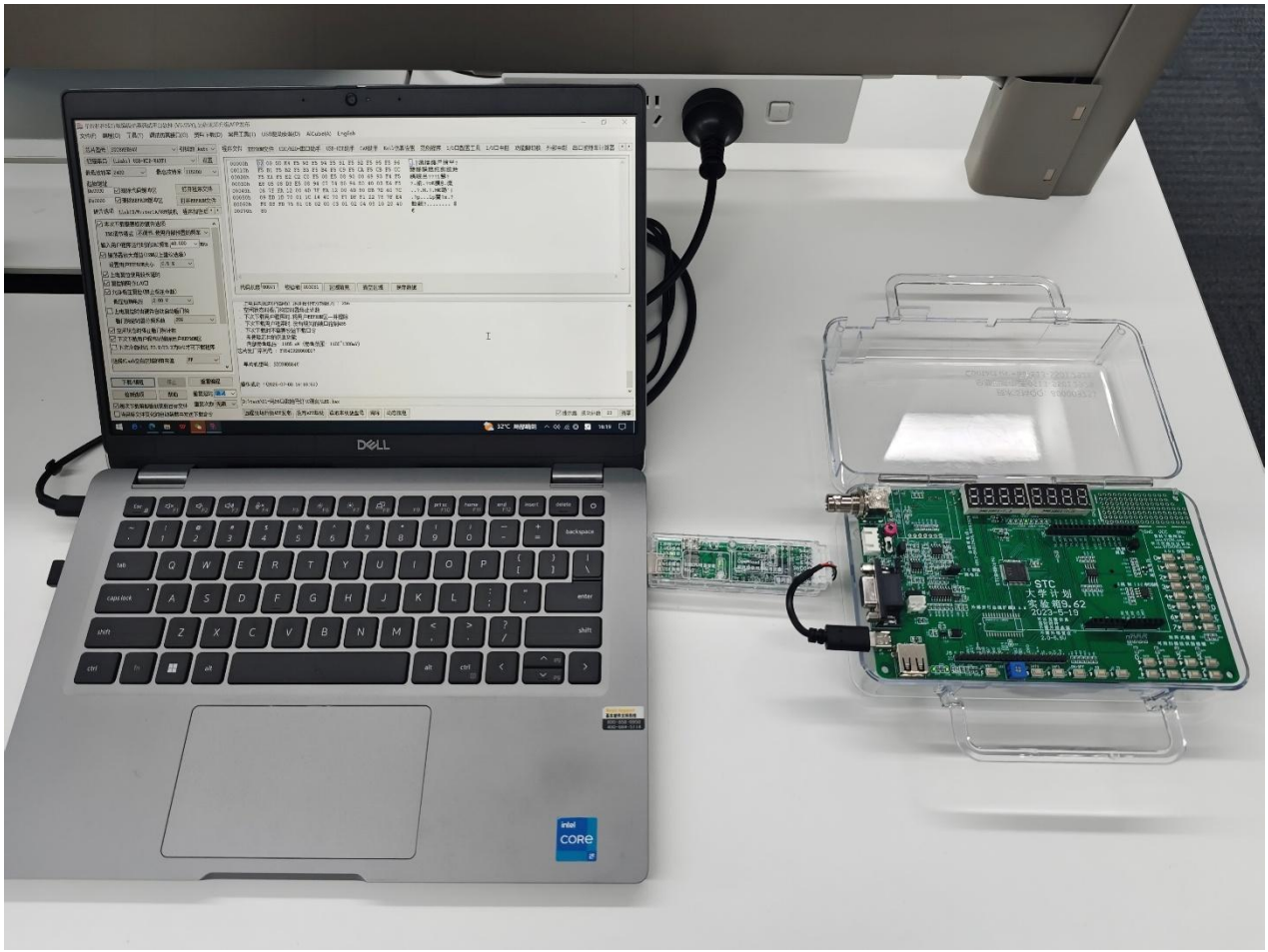
extern    u8    xdata RX1_Buffer[256];      //接收缓冲

void      Uart1_Init(void);           //函数声明

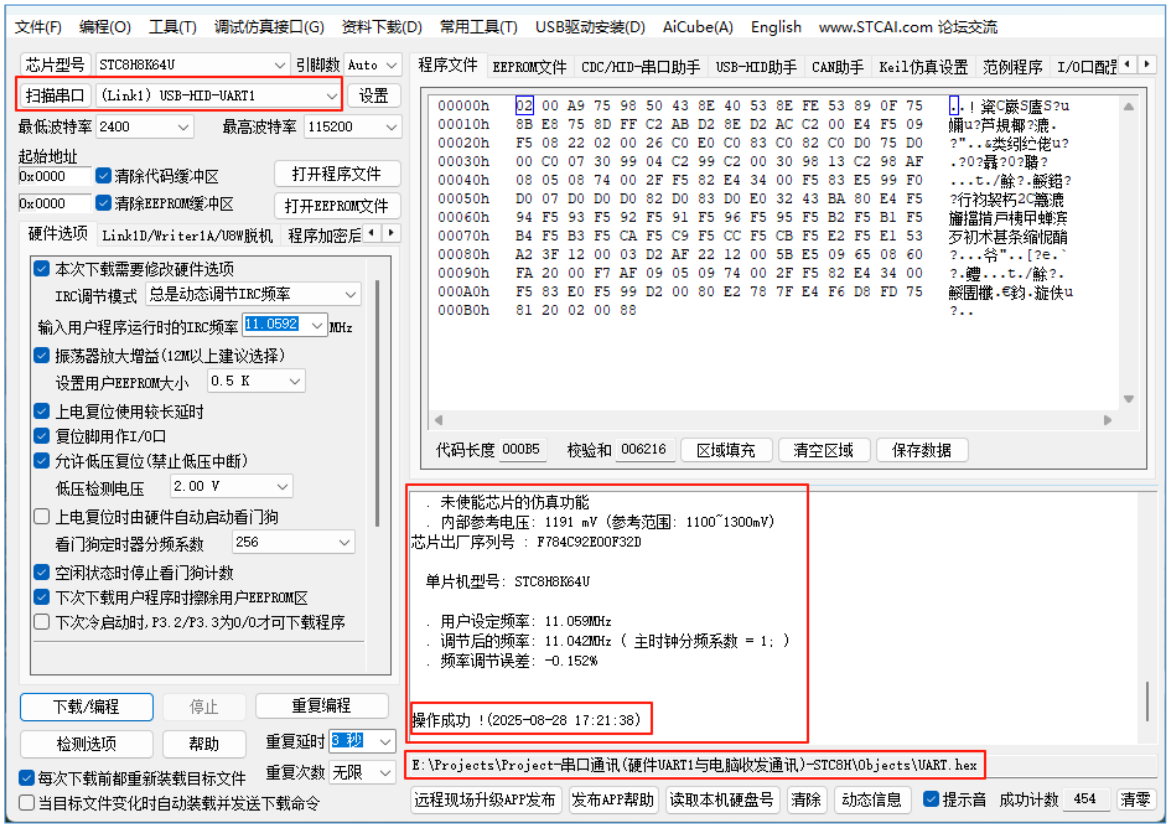
#endif
```

2.13.4 程序下载，观察实验现象，验证实验效果

➲ 录入代码，保存，编译。按实验要求连接并设置好 STC8H8K64U 实验箱，如下图：

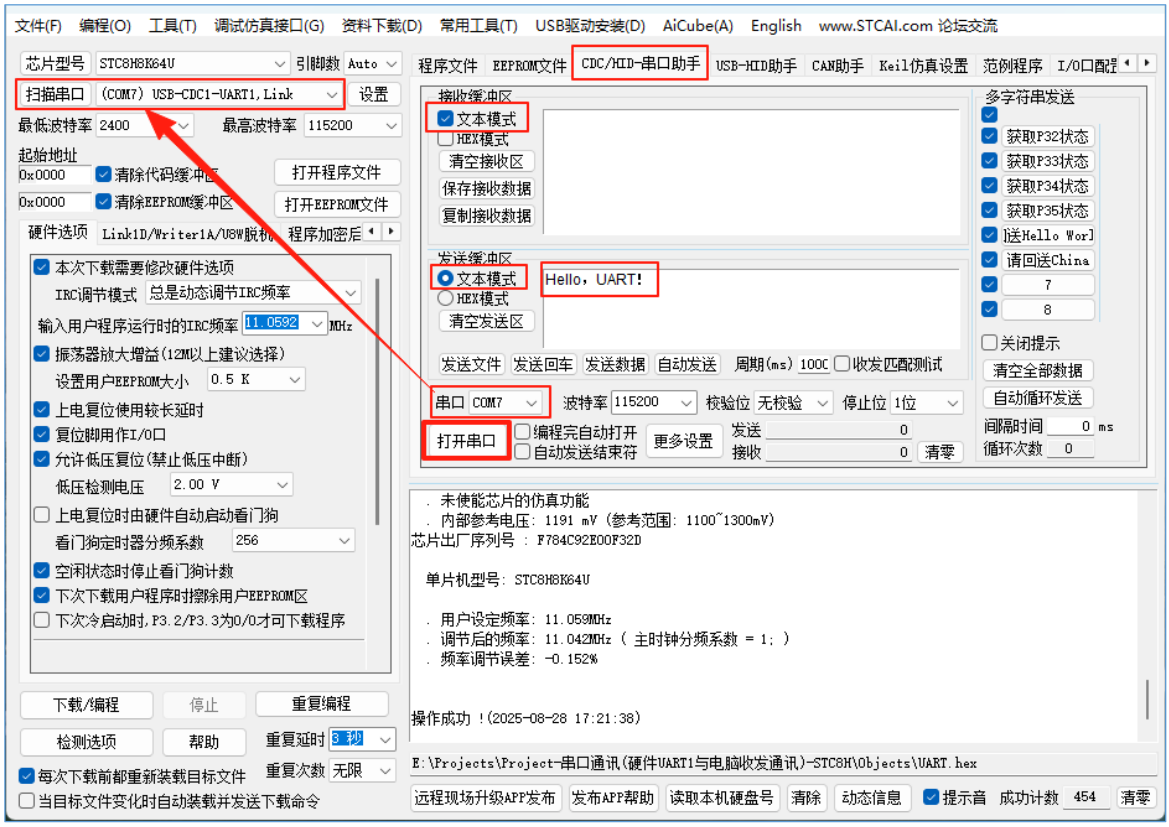


➲ 运行 AiCube-ISP 软件系统，按步骤打开并下载程序“uart.hex”，如下图：



➲ 点击“CDC/HID-串口助手”选项卡：

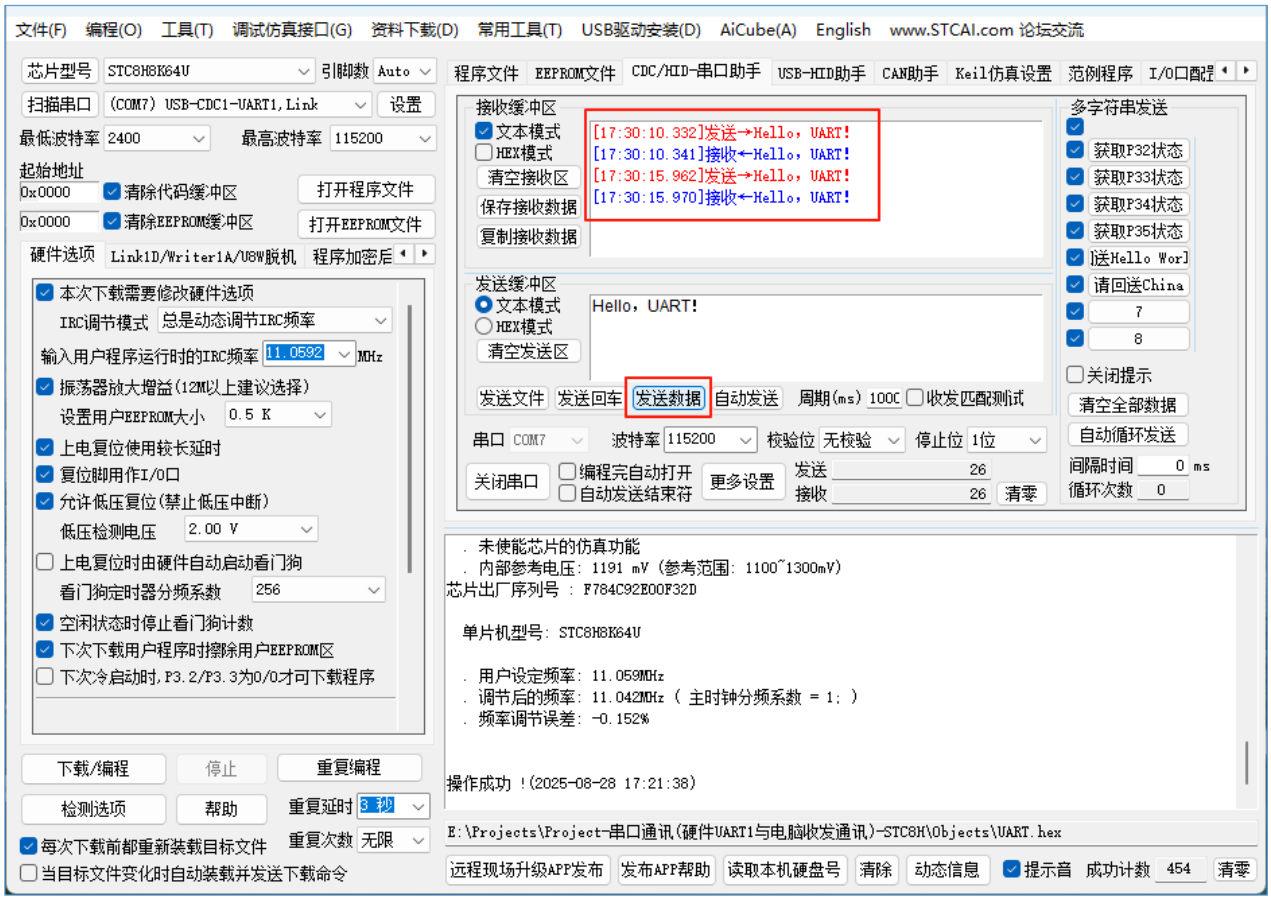
- ➢ 选择与“扫描串口”栏 USB-CDC1-UART1 对应的 COM7 串口
- ➢ 在发送缓冲区中输入发送数据“Hello，UART!”
- ➢ 找到“打开串口”按钮，如下图：



一切准备就绪！

- 点击“打开串口”按钮
- 点击“发送数据”按钮

我们看到在接收缓冲区就会收到串口 1/UART1 发送给电脑的数据，如下图：



完成实验验证。

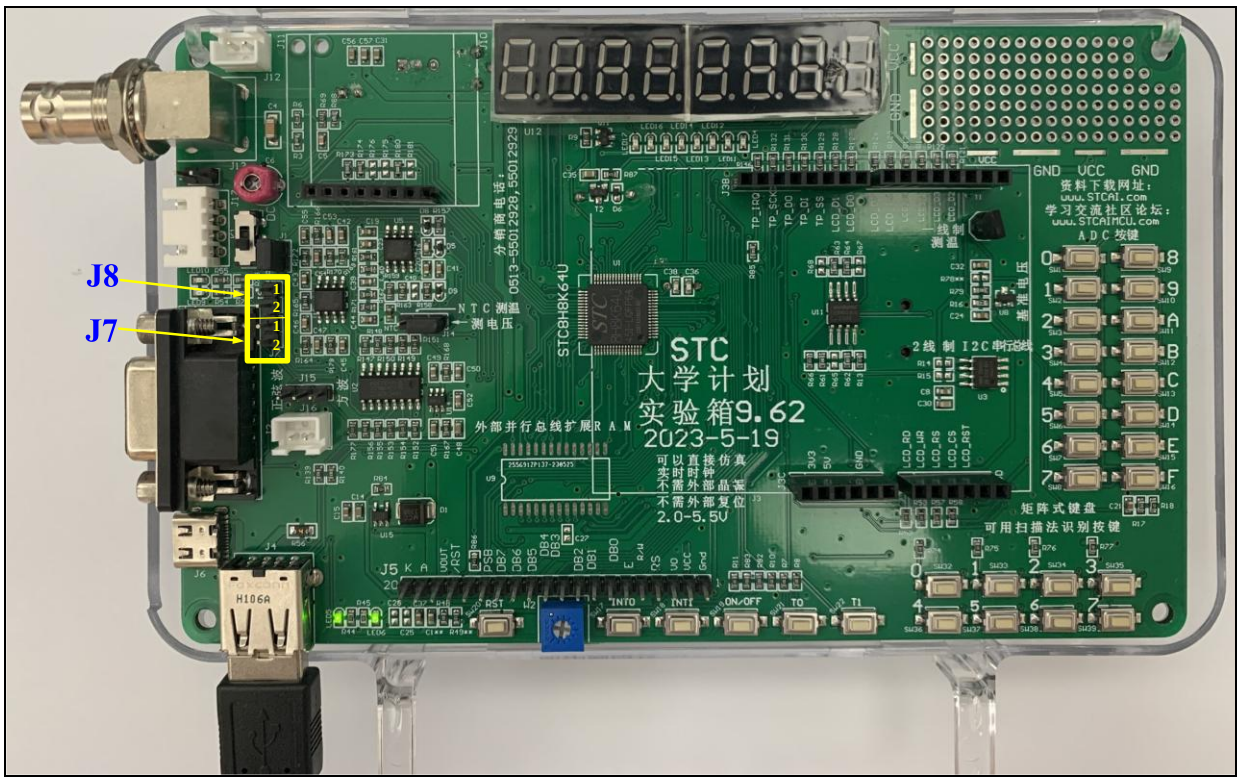
2.14 串口实验 -- 串口通讯(硬件 UART2 与 UART3 收发通讯)

2.14.1 实验介绍

1、 熟悉串口 2/UART2 和串口 3/UART3 的通讯原理
2、 熟悉 STC8H8K64U 实验箱及其原理图
3、 熟练使用 AiCube-ISP 软件
4、 认识 USB 转双串口工具@Ai8H2K12U、杜邦线、USB 转串口/TTL 连接线
5、 熟悉如何管理多文件项目

2.14.1.1 认识 STC8H8K64U 实验箱

STC8H8K64U 实验箱正面图：



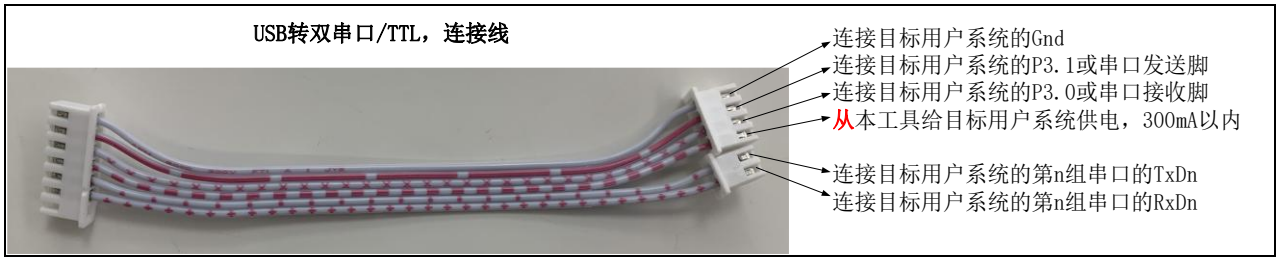
1、J8： 将 STC8H8K64U 实验箱上“J8 跳线”跳线连上，使主控芯片 UART2 的 P4.7/TxD2_2 与 UART3 的 P5.0/TxD3 联通。实现本实验要求的串口 2/UART2 和串口 3/UART3 收发通讯。（详见 J7 和 J8 跳线原理图）

2、J7： STC8H8K64U 实验箱上 J7 的 Pin2 通过 1 根杜邦线连接 UART3 的 P5.1/TxD3 到双串口工具（Ai8H2K12U）S-TxD2，使主控芯片 UART3 的 P5.1/TxD3 发送的串口数据在电脑端显示。（详见 J7 和 J8 跳线原理图）

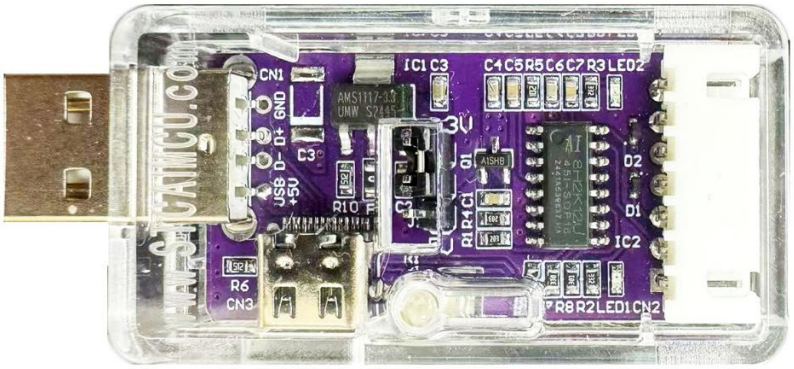
2.14.1.2 2 根杜邦线（1 公头、1 母头）



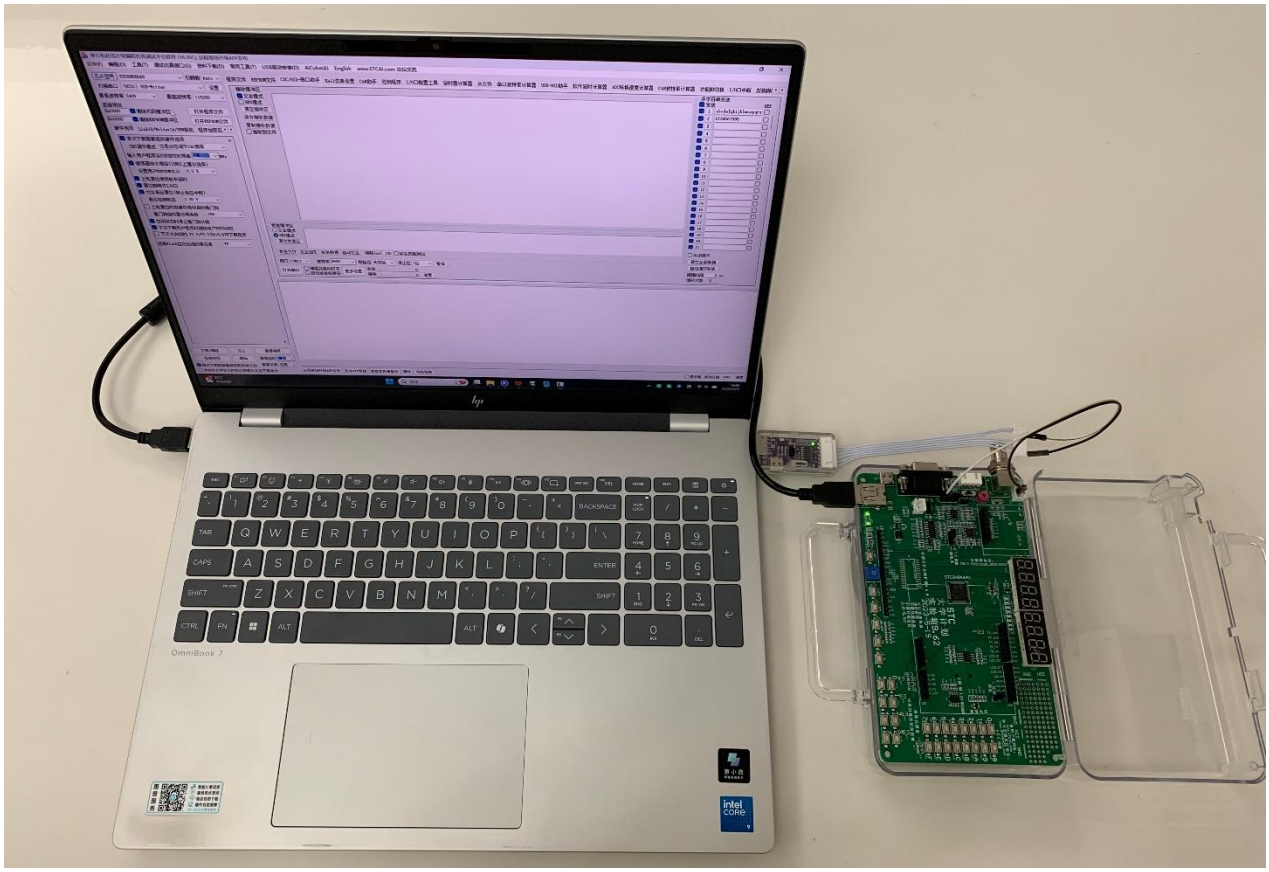
2.14.1.3 USB 转串口/TTL 连接线



2.14.1.4 USB 转双串口工具@Ai8H2K12U：正面



2.14.1.5 本实验实验箱与电脑连接实物图



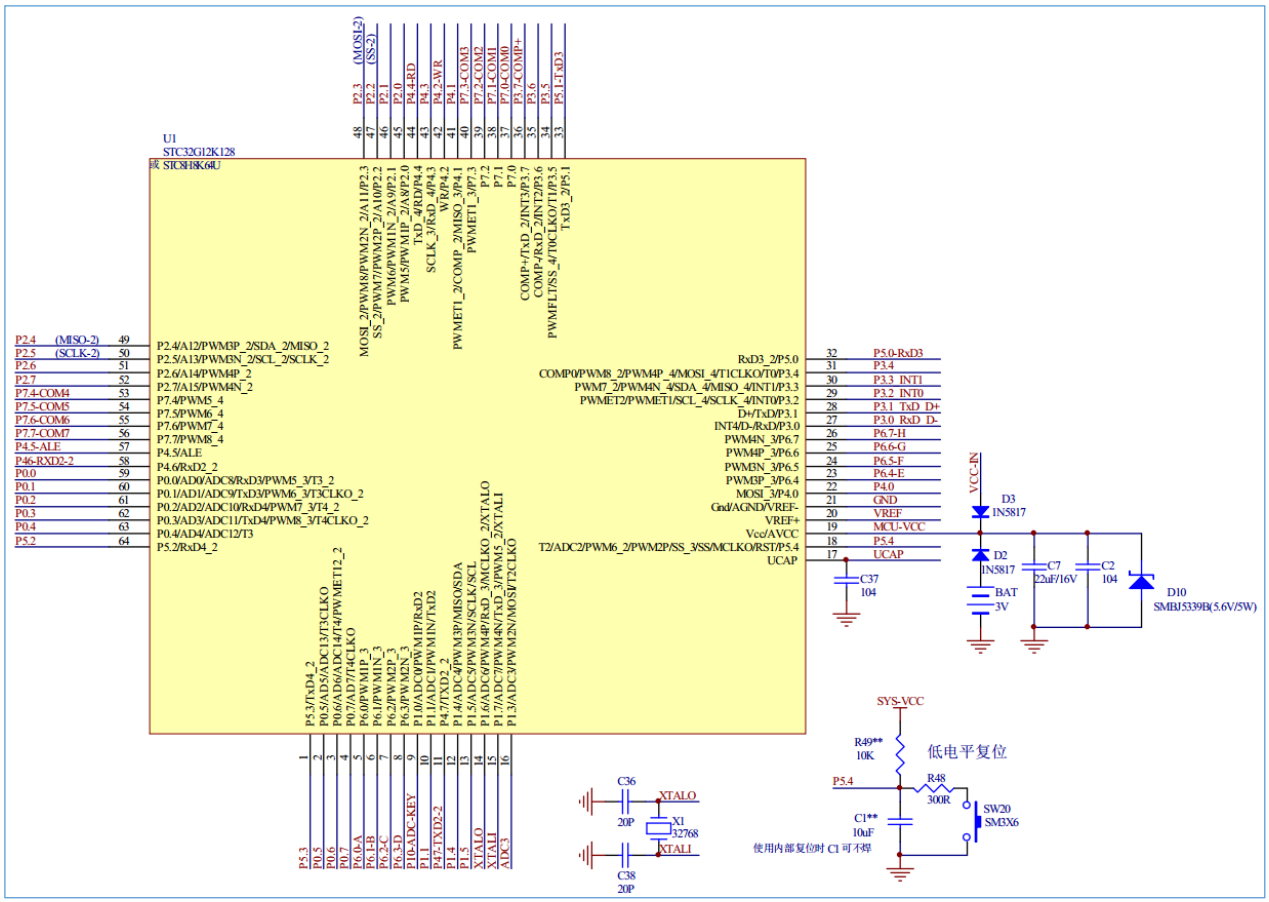
2.14.1.6 Keil 环境下多文件项目管理说明

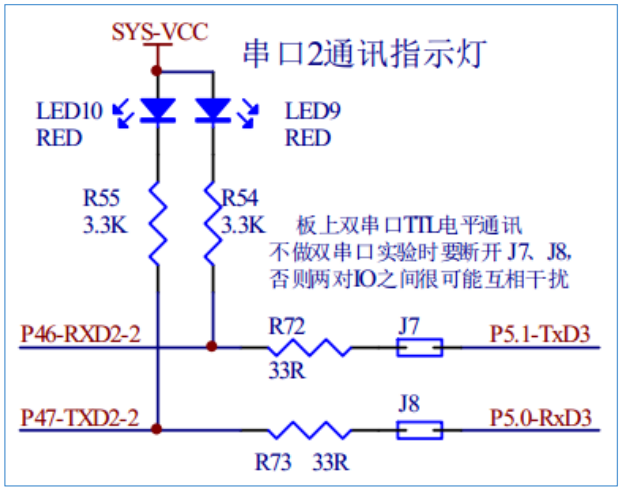
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以**方便分工合作、代码复用、模块化管理、增强可读性和可维护性**。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

```
#ifndef XXXX
#define XXXX
...
#endif
```

2.14.2 原理图





2.14.3 实验程序代码

2.14.3.1 main.c -- 程序主函数

```
//main.c
//程序主函数
//短接 J8 跳线
//J7 通过 2 根杜邦线连接 UART3 的 P5.1/TxD3 到双串口工具
//串口助手就能输出串口 2 每秒钟发送给串口 3 的数据

#include "config.h"        //头文件中已包含 stc8h.h 以及其他头文件

void main()
{
    SYS_Init();          //系统初始化

    while (1)
    {
        if(f1s)          //判断是否完成一串数据的接收
        {
            f1s = 0;   //清除接收一串数据完成标志
            printf("STC8H8K64U UART2-UART3 Test Programme!\r\n");        //UART2 发送一个字符串
        }
        if (RX3_Cnt != TX3_Cnt)
            Uart3_Send(RX3_Buffer[TX3_Cnt++]);      //将串口 3 收到的数据，从串口 3 发送出去
    }
}

void SYS_Init(void)      //系统初始化函数
{
    P_SW2 = 0x80;                    //使能访问扩展 XFR

    P0M0 = 0x00; P0M1 = 0x00;        //初始化 P0 口为准双向口模式
```

```
    P1M0 = 0x00; P1M1 = 0x00;     //初始化 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;     //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;     //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;     //初始化 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;     //初始化 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;     //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;     //初始化 P7 口为准双向口模式

    P_SW2 |= UART2_S2;            //UART2 通道选择: 0: P1.0 P1.1,   1: P4.6 P4.7
    P_SW2 |= UART3_S2;            //UART3 通道选择: 0: P0.0 P0.1,   1: P5.0 P5.1

    Uart2_Init();       //UART1 接口初始化
    Uart3_Init();       //UART2 接口初始化
    Timer0_Init();      //定时器 0 初始化,用于产生 1s 的时间基准

    EA = 1;             //总中断允许位打开
}
```

2.14.3.2　　uart.c -- 初始化 UART 接口

```
//uart.c
//初始化 UART 接口
//中断处理 UART 收发数据结果

#include "config.h"       //头文件中已包含 stc8h.h 以及其他头文件

bit   B_TX2_Busy;     //发送忙标志
bit   B_TX3_Busy;     //发送忙标志

u8    RX2_Cnt;        //接收计数
u8    RX3_Cnt;        //接收计数
u8    TX3_Cnt;        //发送计数

u8    xdata RX2_Buffer[256];      //接收缓冲
u8    xdata RX3_Buffer[256];      //接收缓冲

void Uart2_Isr(void) interrupt UART2_VECTOR
{
    if (S2CON & 0x02)           //检测串口 2 发送中断
    {
        S2CON &= ~0x02;         //清除串口 2 发送中断请求位
        B_TX2_Busy = 0;         //清除发送忙标志
    }
    if (S2CON & 0x01)           //检测串口 2 接收中断
    {
        S2CON &= ~0x01;         //清除串口 2 接收中断请求位
```

```
            RX2_Buffer[RX2_Cnt++] = S2BUF;    //将收到数据存入缓冲区
        }
}

void Uart2_Init(void)     //115200bps@11.0592MHz
{
    S2CON = 0x50;     //8 位数据,可变波特率
    AUXR |= 0x04;     //定时器时钟 1T 模式
    T2L = 0xE8;       //设置定时初始值
    T2H = 0xFF;       //设置定时初始值
    AUXR |= 0x10;     //定时器 2 开始计时
    IE2 |= 0x01;      //使能串口 2 中断

    RX2_Cnt = 0;
    B_TX2_Busy = 0;
}

void Uart2_Send(u8 dat)       //通过串口 2 发送数据
{
    S2BUF = dat;              //写入需要发送的数据
    B_TX2_Busy = 1;           //设置发送忙标志
    while(B_TX2_Busy);        //判断是否发送完成
}

char putchar(char c)      //将 printf 函数映射到 UART2 接口
{
    Uart2_Send(c);        //调用串口 2 发送函数（如果要映射到串口 3 的话，这里修改为串口 3 发送函数）
    return c;
}

void Uart3_Isr(void) interrupt UART3_VECTOR
{
    if (S3CON & 0x02)             //检测串口 3 发送中断
    {
        S3CON &= ~0x02;       //清除串口 3 发送中断请求位
        B_TX3_Busy = 0;       //清除发送忙标志
    }
    if (S3CON & 0x01)             //检测串口 3 接收中断
    {
        S3CON &= ~0x01;       //清除串口 3 接收中断请求位
        RX3_Buffer[RX3_Cnt++] = S3BUF;    //将收到数据存入缓冲区
    }
}

void Uart3_Init(void)         //115200bps@11.0592MHz
{
```

```
    S3CON = 0x10;           //8 位数据,可变波特率
    S3CON &= 0xBF;          //串口 3 选择定时器 2 为波特率发生器
    AUXR |= 0x04;           //定时器时钟 1T 模式
    T2L = 0xE8;             //设置定时初始值
    T2H = 0xFF;             //设置定时初始值
    AUXR |= 0x10;           //定时器 2 开始计时
    IE2 |= 0x08;          //使能串口 3 中断

    RX3_Cnt = 0;
    TX3_Cnt = 0;
    B_TX3_Busy = 0;
}

void Uart3_Send(u8 dat)     //通过串口 3 发送数据
{
    S3BUF = dat;            //写入需要发送的数据
    B_TX3_Busy = 1;         //设置发送忙标志
    while(B_TX3_Busy);      //判断是否发送完成
}
```

2.14.3.3 timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:1s 定时，产生 1s 的标志位

#include "config.h"             //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

bit   f1s;                      //1s 标志位
int cnt1000;

void Timer0_Init(void)          //1 毫秒@11.0592MHz
{
    AUXR |= 0x80;               //定时器时钟 1T 模式
    TMOD &= 0xF0;               //设置定时器模式
    TL0 = 0xCD;                 //设置定时初始值
    TH0 = 0xD4;                 //设置定时初始值
    TF0 = 0;                    //清除 TF0 标志
    TR0 = 1;                    //定时器 0 开始计时
    ET0 = 1;                    //使能定时器 0 中断

    cnt1000 = 0;
}

void Timer0_Isr(void) interrupt 1   //定时器 0 中断服务程序
{
```

```
    if (cnt1000++ >= 1000)
    {
        cnt1000 = 0;
        f1s = 1;                    //设置 1s 标志位
    }
}
```

2.14.3.4 config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__            //防止头文件被重复包含
#define __CONFIG_H__

#define HIRC            11059200UL
#define FOSC            11059200UL
#define SYSCLK          FOSC
#define MAIN_Fosc       FOSC

#include <stc8h.h>              //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"                //包含项目头文件
#include "uart.h"
#include "timer.h"

void SYS_Init(void);            //函数声明

#endif
```

2.14.3.5 uart.h -- 项目配置文件的头文件

```
#ifndef __UART_H__          //防止头文件被重复包含
#define __UART_H__

extern    u8    RX2_Cnt;    //接收计数
extern    u8    RX3_Cnt;    //接收计数
extern    u8    TX3_Cnt;    //发送计数

extern    u8    xdata    RX2_Buffer[256];  //接收缓冲
extern    u8    xdata    RX3_Buffer[256];  //接收缓冲

void Uart2_Init(void);          //函数声明
void Uart3_Init(void);          //函数声明
void Uart2_Send(u8 dat);        //函数声明
void Uart3_Send(u8 dat);        //函数声明
```

```
#endif
```

2.14.3.6 timer.h -- 项目配置文件的头文件

```
#ifndef __TIMER_H__        //防止头文件被重复包含
#define __TIMER_H__

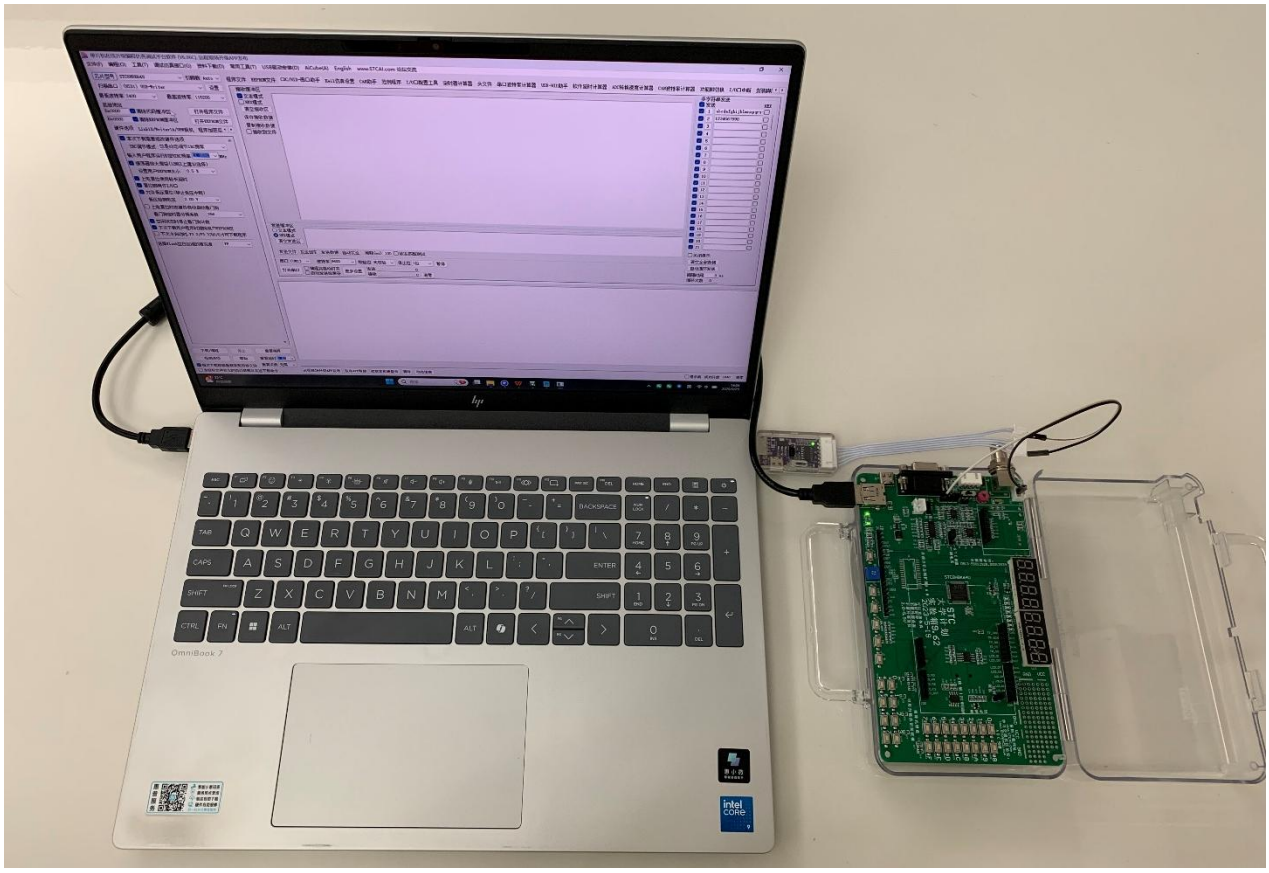
void Timer0_Init(void);        //函数声明

extern bit f1s;                //外部变量声明

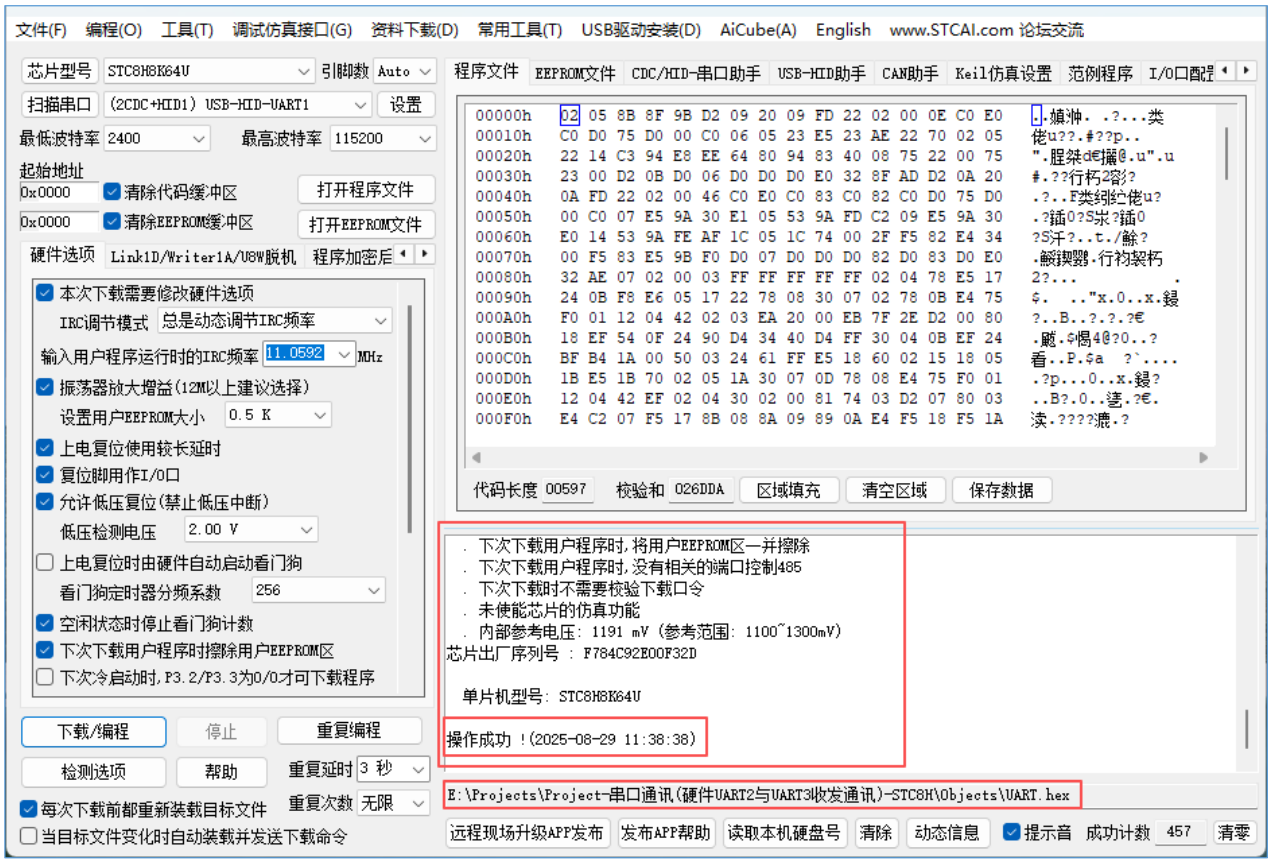
#endif
```

2.14.4 程序下载，观察实验现象，验证实验效果

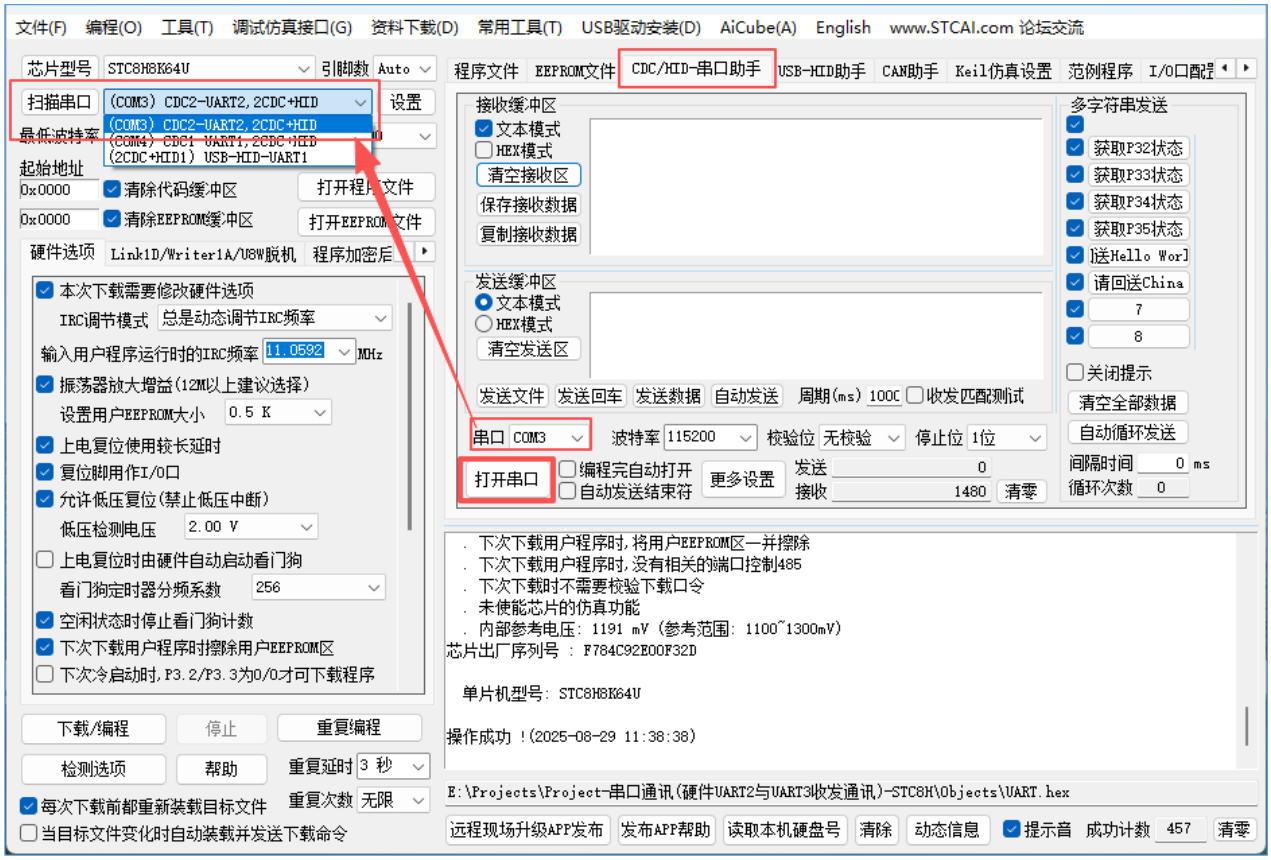
➲ 录入代码，保存，编译。按实验要求连接并设置好 STC8H8K64U 实验箱，如下图：



➲ 运行 AiCube-ISP 软件系统，按步骤打开并下载程序“uart.hex”，如下图：



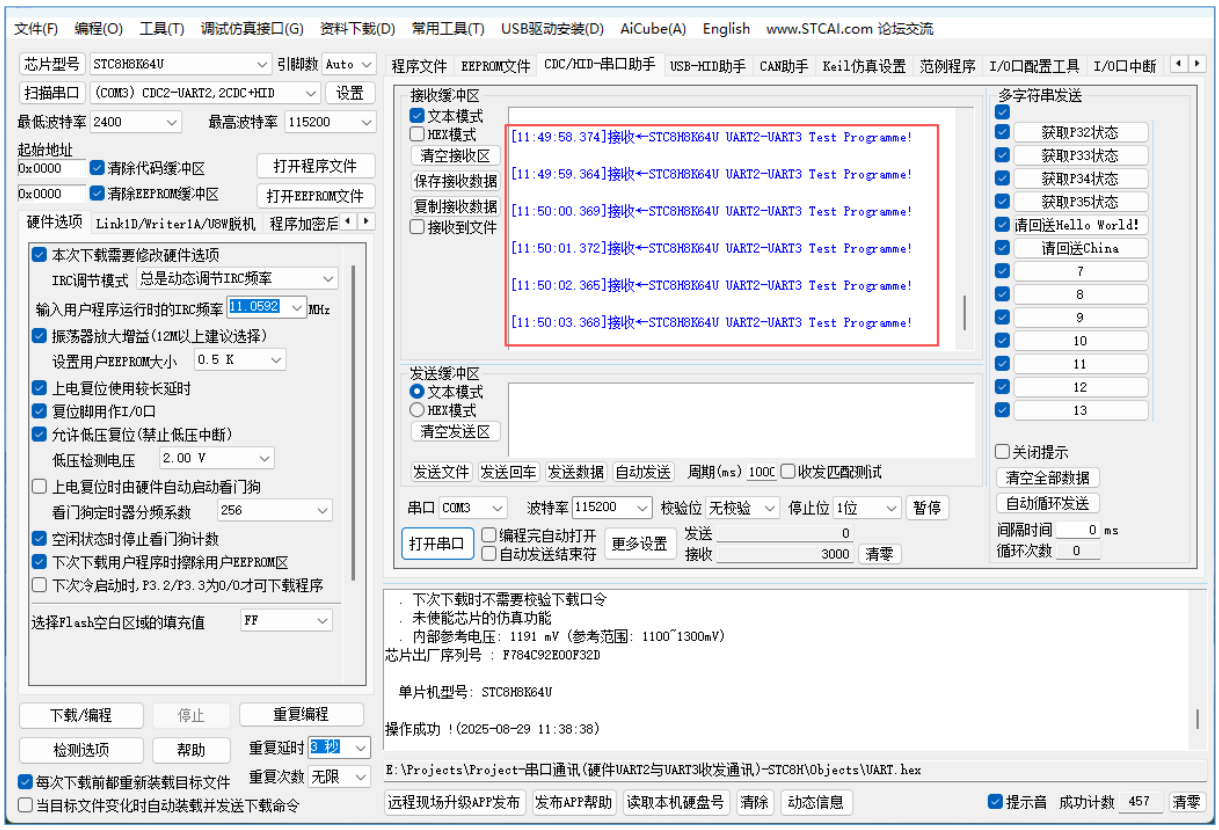
➲ 点击“CDC/HID-串口助手”选项卡，“串口”要选择与“扫描串口”栏里 CDC2 对应的 COM3 口，找到“打开串口”按钮，如下图：



一切准备就绪！

➲ 点击“打开串口”按钮，我们观察到：

➢ “CDC/HID-串口助手”选项卡中的接收缓冲区就会收到串口 2 每秒钟发送给串口 3 的数据“STC 8H8K64U UART2-UART3 Test Programme!”，如下图：



➢ 观察 STC8H8K64U 实验箱，我们会看到 J7 旁边的 LED10 在闪烁。

➢ 再看示波器采集的串口 2 发送字母‘S’（0101 0011）的数据图，1bit 起始位（0）+8bit 数据位+1bit 停止位（1）：



完成实验验证。

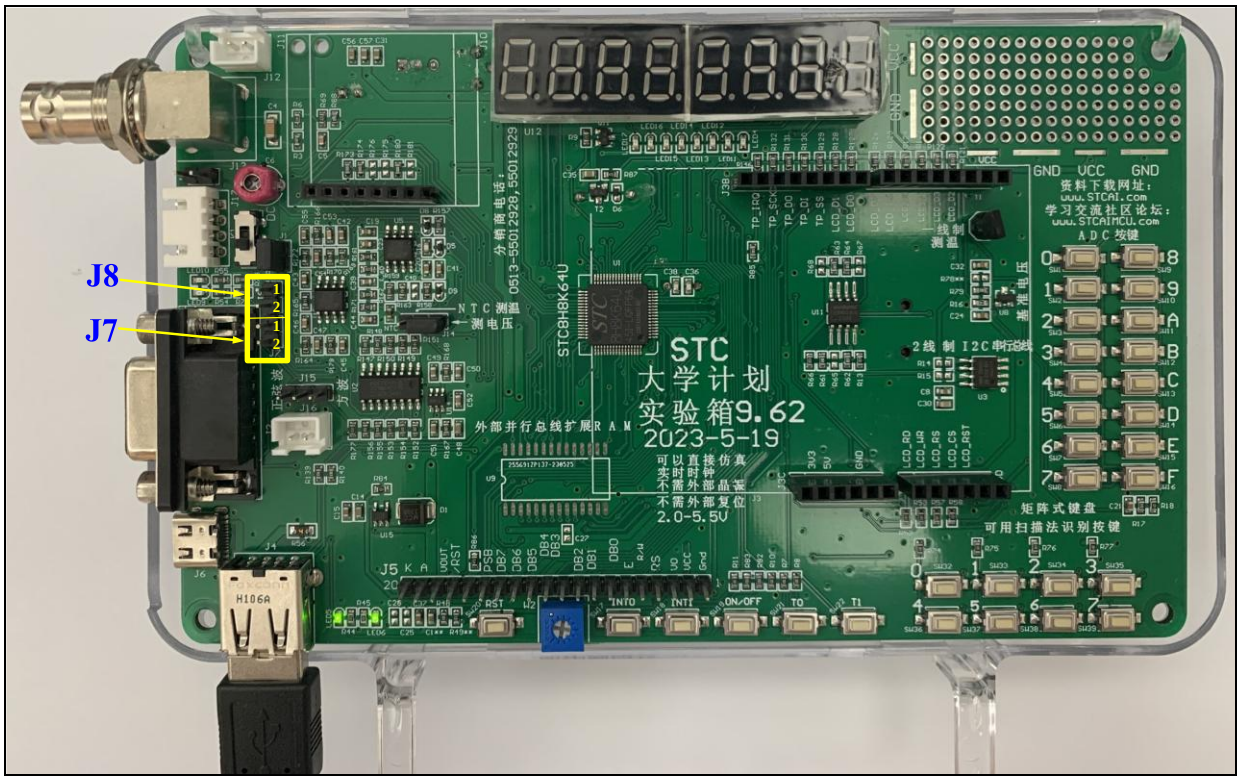
2.15 串口实验 -- 串口通讯(硬件 UART2 与 UART3 数据透传)

2.15.1 实验介绍

1、 了解双串口数据透传的原理及应用
2、 熟悉 STC8H8K64U 实验箱及其原理图
3、 熟练使用 AiCube-ISP 系统软件
4、 认识 USB 转双串口工具@Ai8H2K12U、杜邦线、USB 转串口/TTL 连接线
5、 熟悉如何管理多文件项目

2.15.1.1 认识 STC8H8K64U 实验箱

STC8H8K64U 的实验箱正面图：



1、J8： STC8H8K64U 实验箱上 J8 的 Pin1 通过 1 根杜邦线将 UART2 的 P4.7/TxD2_2 发送脚与双串口工具（Ai8H2K12U）的 S-TxD 相连接；

STC8H8K64U 实验箱上 J8 的 Pin2 通过 1 根杜邦线将 UART3 的 P5.0/RxD3 接收脚与双串口工具（Ai8H2K12U）的 S-RxD2 相连接。

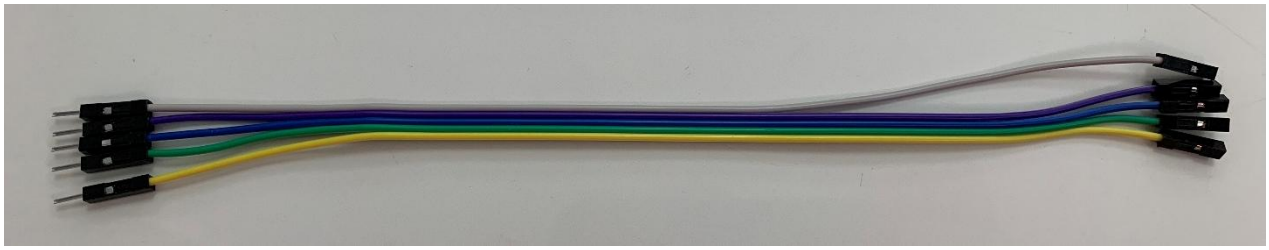
（详见 J7 和 J8 跳线原理图）

2、J7： STC8H8K64U 实验箱上 J7 的 Pin1 通过 1 根杜邦线将 UART2 的 P4.6/RxD2_2 接收脚与双串口工具（Ai8H2K12U）的 S-RxD 相连接；

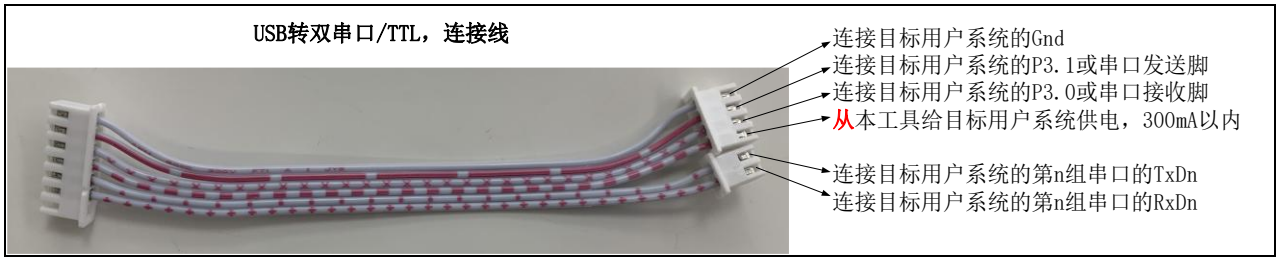
STC8H8K64U 实验箱上 J7 的 Pin2 通过 1 根杜邦线将 UART3 的 P5.1/TxD3 发送脚与双串口工具（Ai8H2K12U）的 S-TxD2 相连接。

（详见 J7 和 J8 跳线原理图）

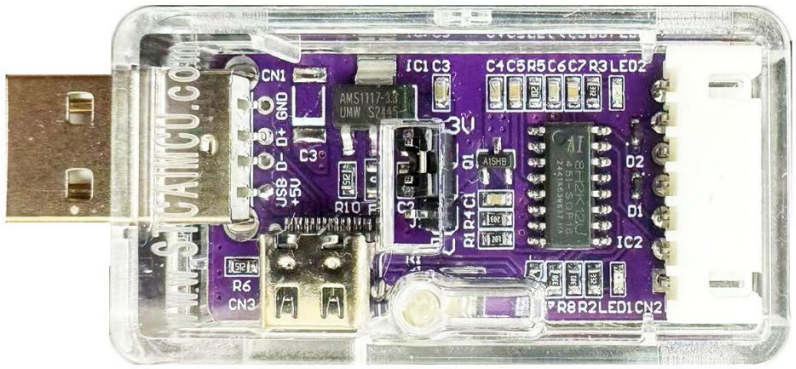
2.15.1.2 5 根杜邦线（1 公头、1 母头）



2.15.1.3 USB 转串口/TTL 连接线



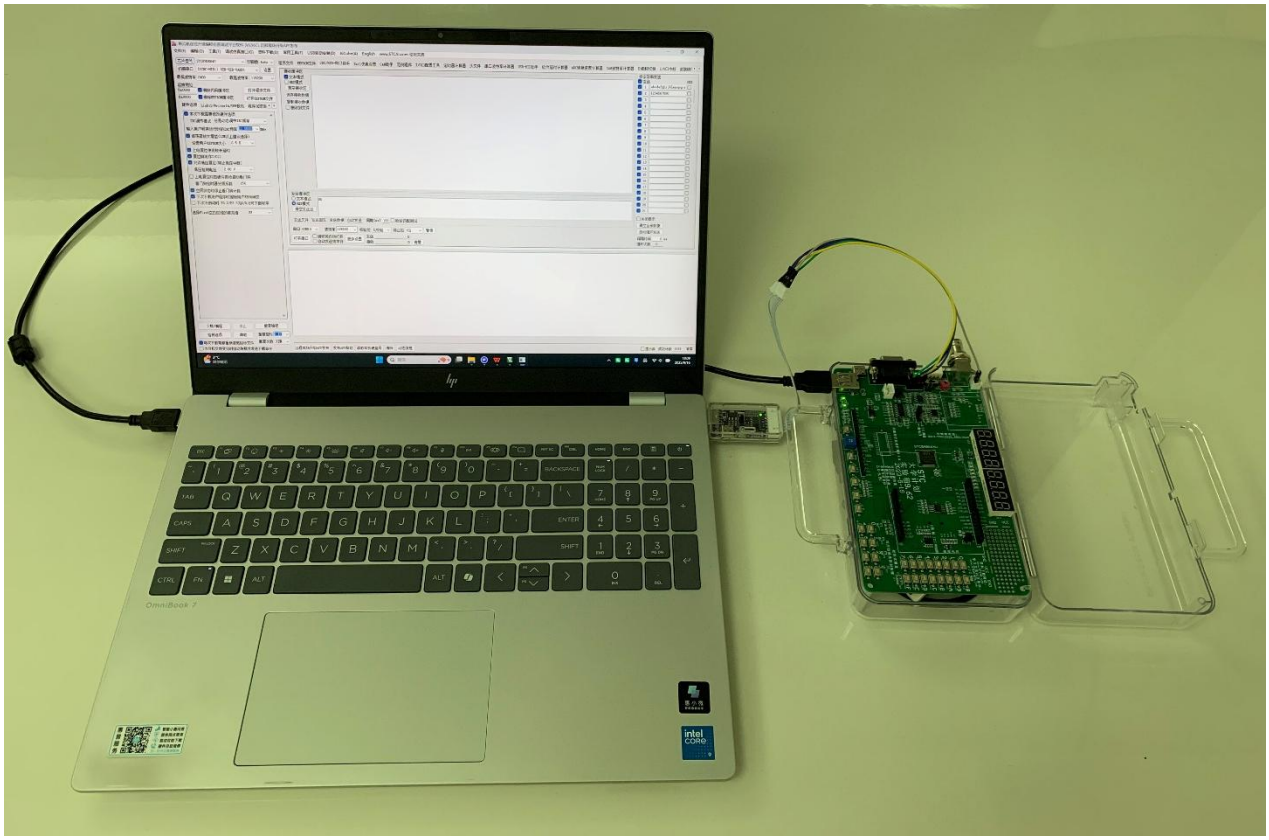
2.15.1.4 USB 转双串口工具@Ai8H2K12U：正面



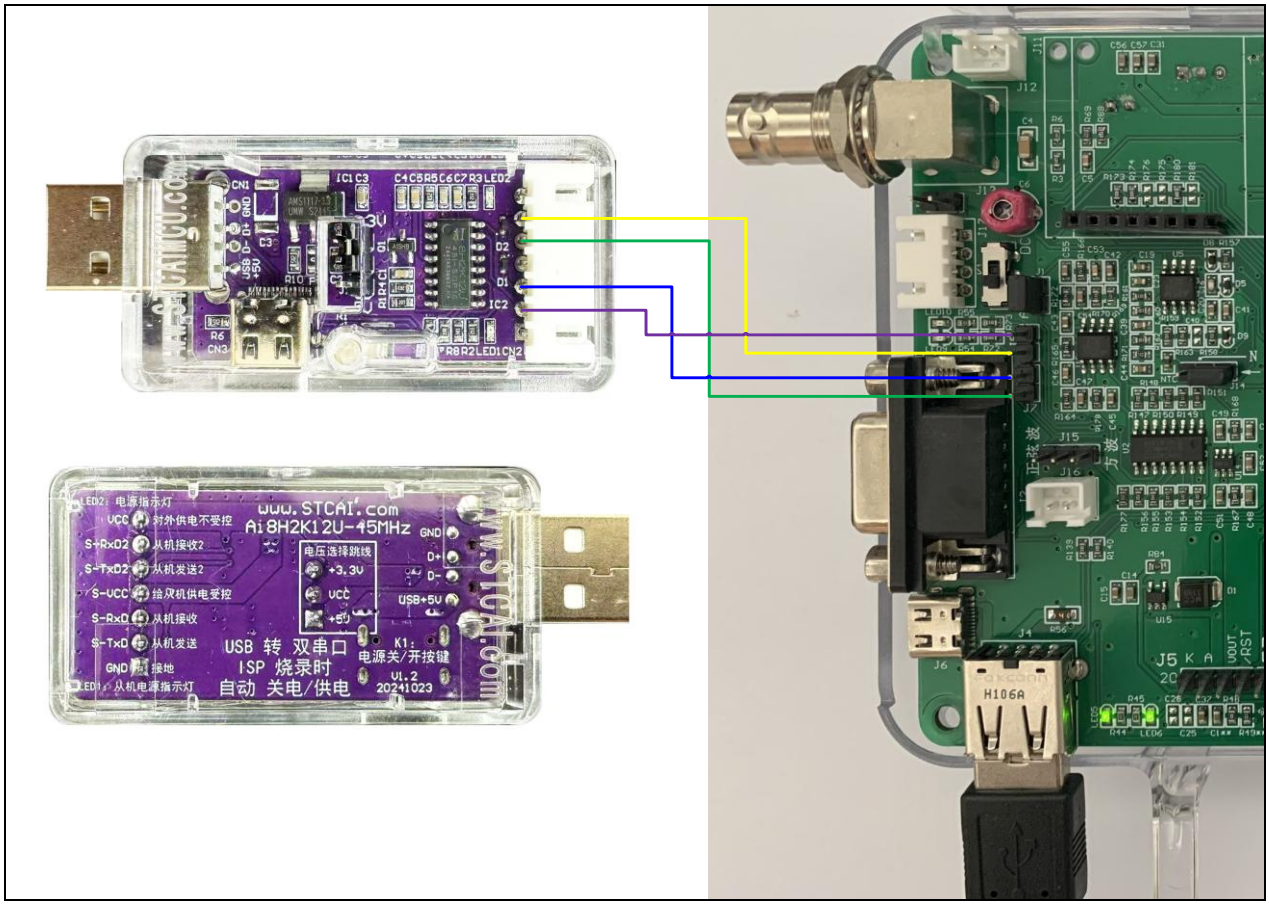
2.15.1.5 USB 转双串口工具@Ai8H2K12U：背面



2.15.1.6 本实验实验箱与电脑连接实物图



详细的连线图如下：



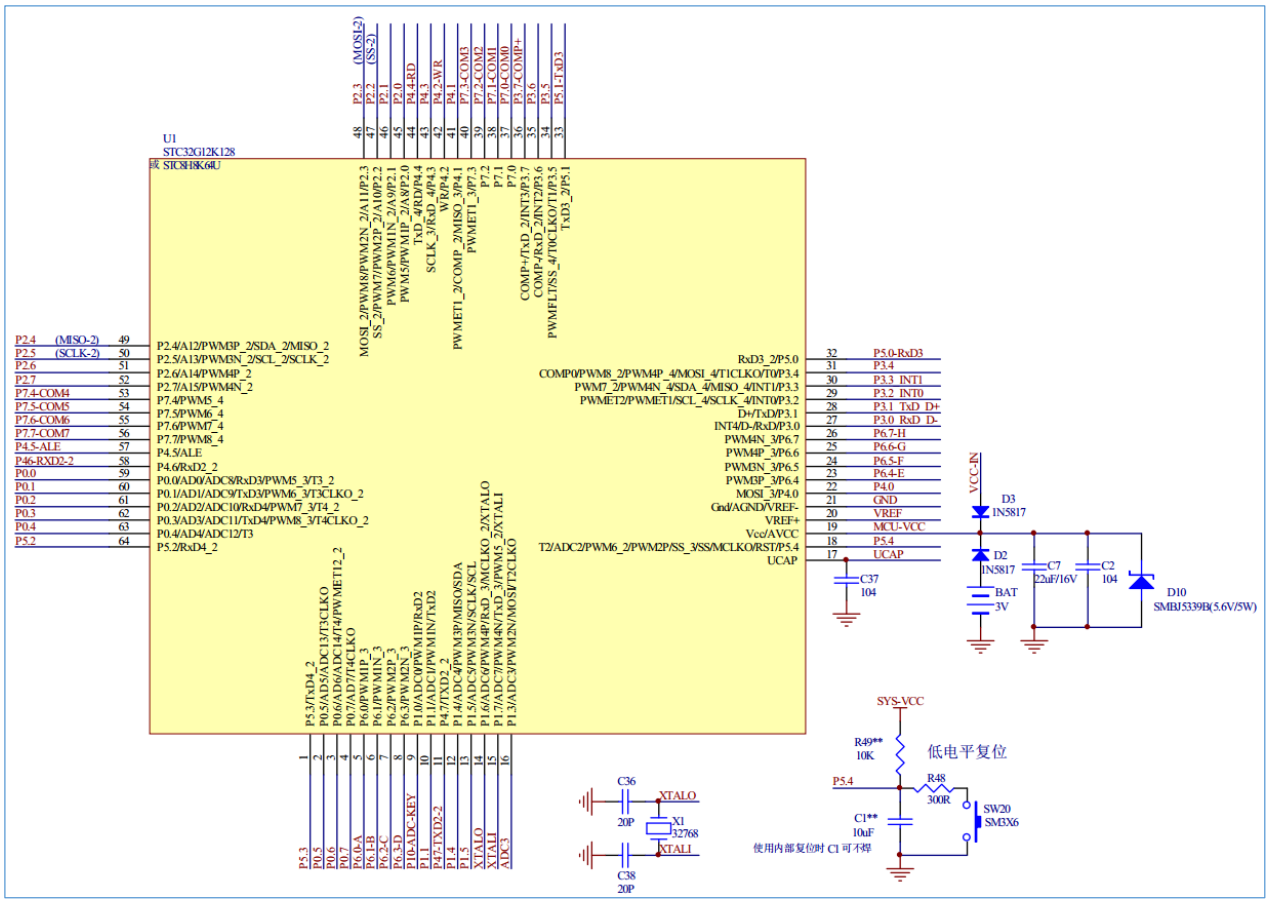
2.15.1.7 Keil 环境下多文件项目管理说明

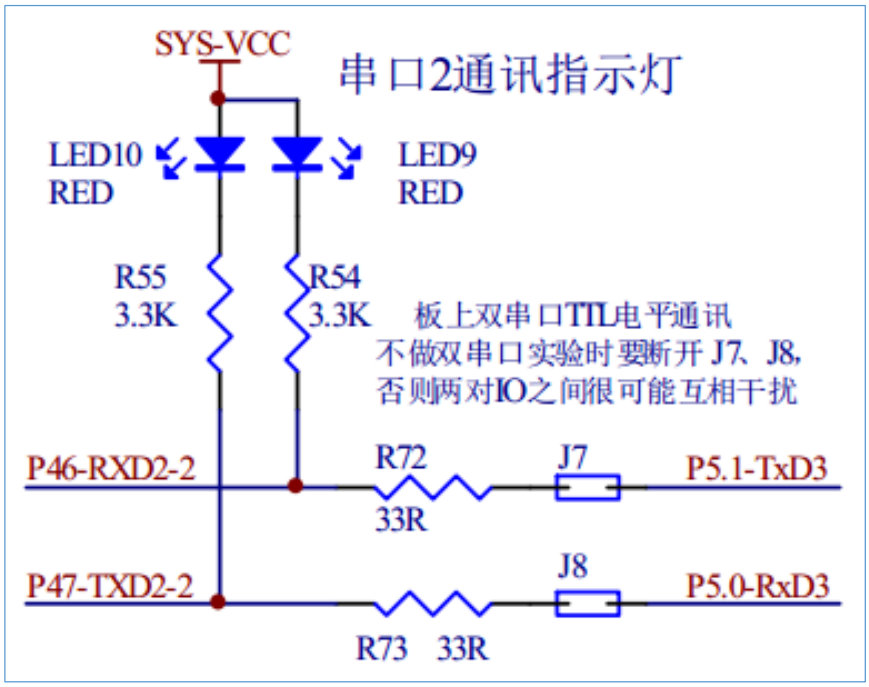
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以**方便分工合作、代码复用、模块化管理、增强可读性和可维护性**。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

```
#ifndef XXXX
#define XXXX
...
#endif
```

2.15.2 原理图





2.15.3 实验程序代码

2.15.3.1 main.c -- 程序主函数

```
//main.c
//程序主函数
//短接 J7、J8 通过 4 根杜邦线连接双串口工具
//USB 线连接电脑与双串口工具，开启两个串口助手，分别打开 CDC2，CDC3 对应的 COM 口
//CDC2 串口助手发送的内容会被 CDC3 串口助手接收
//CDC3 串口助手发送的内容会被 CDC2 串口助手接收

#include "config.h"        //头文件中已包含 stc8h.h 以及其他头文件

void main()
{
    SYS_Init();            //系统初始化

    while (1)
    {
        if (RX2_Cnt != TX2_Cnt)
            Uart3_Send(RX2_Buffer[TX2_Cnt++]);   //将串口 1 收到的数据，从串口 2 发送出去

        if (RX3_Cnt != TX3_Cnt)
            Uart2_Send(RX3_Buffer[TX3_Cnt++]);   //将串口 2 收到的数据，从串口 3 发送出去
    }
}

void SYS_Init(void)      //系统初始化函数
```

```
{
    P_SW2 = 0x80;                   //使能访问扩展 XFR

    P0M0 = 0x00; P0M1 = 0x00;       //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //初始化 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //初始化 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //初始化 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //初始化 P7 口为准双向口模式

    P_SW2 |= UART2_S2;              //UART2 通道选择: 0: P1.0 P1.1,   1: P4.6 P4.7
    P_SW2 |= UART3_S2;              //UART3 通道选择: 0: P0.0 P0.1,   1: P5.0 P5.1

    Uart2_Init();       //UART2 接口初始化
    Uart3_Init();       //UART3 接口初始化

    EA = 1;             //总中断允许位打开
}
```

2.15.3.2 uart.c -- 初始化 UART 接口

```
//uart.c
//初始化 UART 接口
//中断处理 UART 收发数据结果

#include "config.h"     //头文件中已包含 stc8h.h 以及其他头文件

bit   B_TX2_Busy;       //发送忙标志
bit   B_TX3_Busy;       //发送忙标志

u8    RX2_Cnt;          //接收计数
u8    RX3_Cnt;          //接收计数
u8    TX2_Cnt;          //发送计数
u8    TX3_Cnt;          //发送计数

u8    xdata RX2_Buffer[256];    //接收缓冲
u8    xdata RX3_Buffer[256];    //接收缓冲

void Uart2_Isr(void) interrupt UART2_VECTOR
{
    if (S2CON & 0x02)           //检测串口 2 发送中断
    {
        S2CON &= ~0x02;         //清除串口 2 发送中断请求位
        B_TX2_Busy = 0;         //清除发送忙标志
```

```
    }
    if (S2CON & 0x01)           //检测串口 2 接收中断
    {
        S2CON &= ~0x01;         //清除串口 2 接收中断请求位
        RX2_Buffer[RX2_Cnt++] = S2BUF;    //将收到数据存入缓冲区
    }
}

void Uart2_Init(void)    //115200bps@11.0592MHz
{
    S2CON = 0x50;    //8 位数据,可变波特率
    AUXR |= 0x04;    //定时器时钟 1T 模式
    T2L = 0xE8;      //设置定时初始值
    T2H = 0xFF;      //设置定时初始值
    AUXR |= 0x10;    //定时器 2 开始计时
    IE2 |= 0x01;     //使能串口 2 中断

    RX2_Cnt = 0;
    TX2_Cnt = 0;
    B_TX2_Busy = 0;
}

void Uart2_Send(u8 dat)     //通过串口 2 发送数据
{
    S2BUF = dat;            //写入需要发送的数据
    B_TX2_Busy = 1;         //设置发送忙标志
    while(B_TX2_Busy);      //判断是否发送完成
}

void Uart3_Isr(void) interrupt UART3_VECTOR
{
    if (S3CON & 0x02)           //检测串口 3 发送中断
    {
        S3CON &= ~0x02;         //清除串口 3 发送中断请求位
        B_TX3_Busy = 0;         //清除发送忙标志
    }
    if (S3CON & 0x01)           //检测串口 3 接收中断
    {
        S3CON &= ~0x01;         //清除串口 3 接收中断请求位
        RX3_Buffer[RX3_Cnt++] = S3BUF;    //将收到数据存入缓冲区
    }
}

void Uart3_Init(void)       //115200bps@11.0592MHz
{
    S3CON = 0x10;           //8 位数据,可变波特率
```

```
    S3CON &= 0xBF;      //串口 3 选择定时器 2 为波特率发生器
    AUXR |= 0x04;       //定时器时钟 1T 模式
    T2L = 0xE8;         //设置定时初始值
    T2H = 0xFF;         //设置定时初始值
    AUXR |= 0x10;       //定时器 2 开始计时
    IE2 |= 0x08;        //使能串口 3 中断

    RX3_Cnt = 0;
    TX3_Cnt = 0;
    B_TX3_Busy = 0;
}

void Uart3_Send(u8 dat)     //通过串口 3 发送数据
{
    S3BUF = dat;            //写入需要发送的数据
    B_TX3_Busy = 1;         //设置发送忙标志
    while(B_TX3_Busy);      //判断是否发送完成
}
```

2.15.3.3 config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__            //防止头文件被重复包含
#define __CONFIG_H__

#define HIRC            11059200UL
#define FOSC            11059200UL
#define SYSCLK          FOSC
#define MAIN_Fosc       FOSC

#include <stc8h.h>              //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"                //包含项目头文件
#include "uart.h"

void SYS_Init(void);            //函数声明

#endif
```

2.15.3.4 uart.h -- 项目配置文件的头文件

```
#ifndef __UART_H__          //防止头文件被重复包含
#define __UART_H__

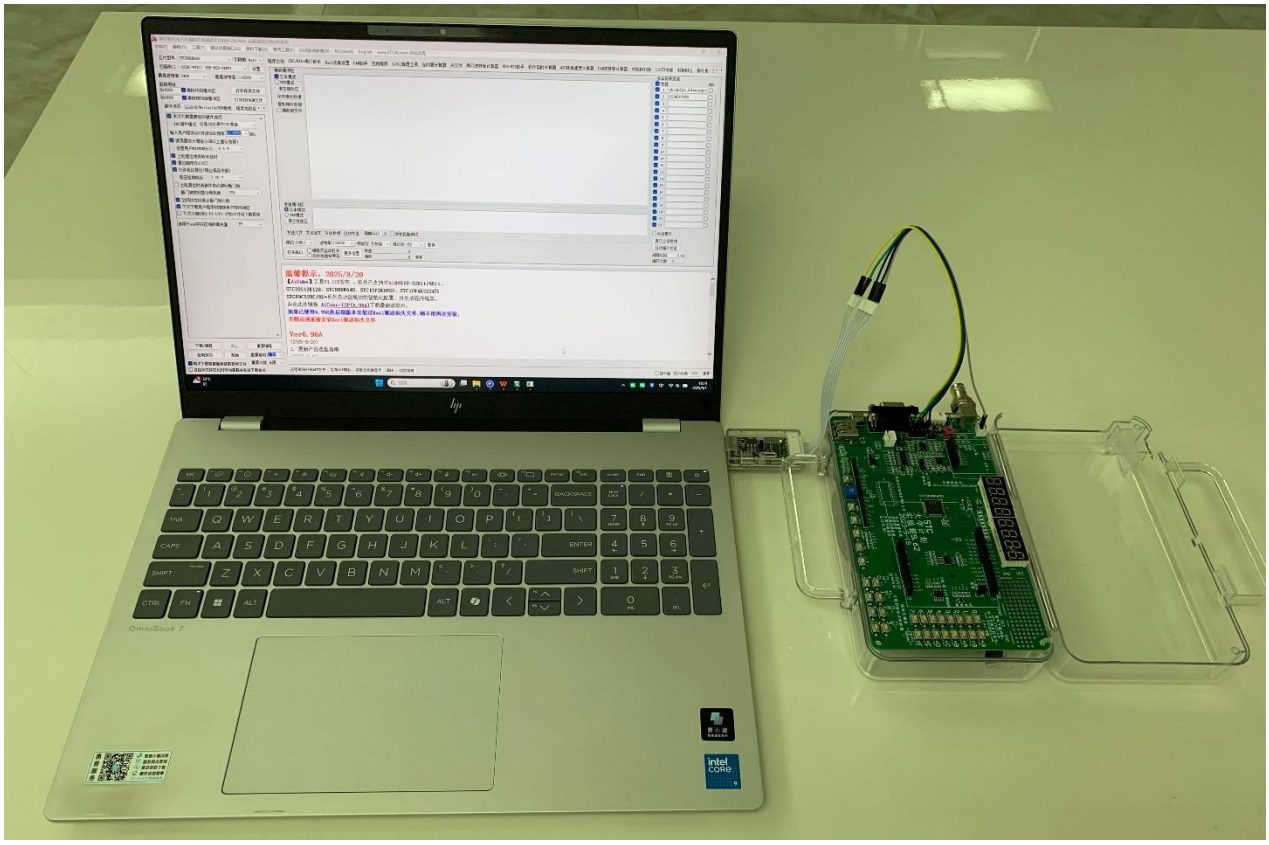
extern    u8    RX2_Cnt;    //接收计数
```

```
extern    u8    RX3_Cnt;     //接收计数
extern    u8    TX2_Cnt;     //发送计数
extern    u8    TX3_Cnt;     //发送计数
extern    u8    xdata    RX2_Buffer[256];  //接收缓冲
extern    u8    xdata    RX3_Buffer[256];  //接收缓冲
void Uart2_Init(void);          //函数声明
void Uart3_Init(void);          //函数声明
void Uart2_Send(u8 dat);        //函数声明
void Uart3_Send(u8 dat);        //函数声明

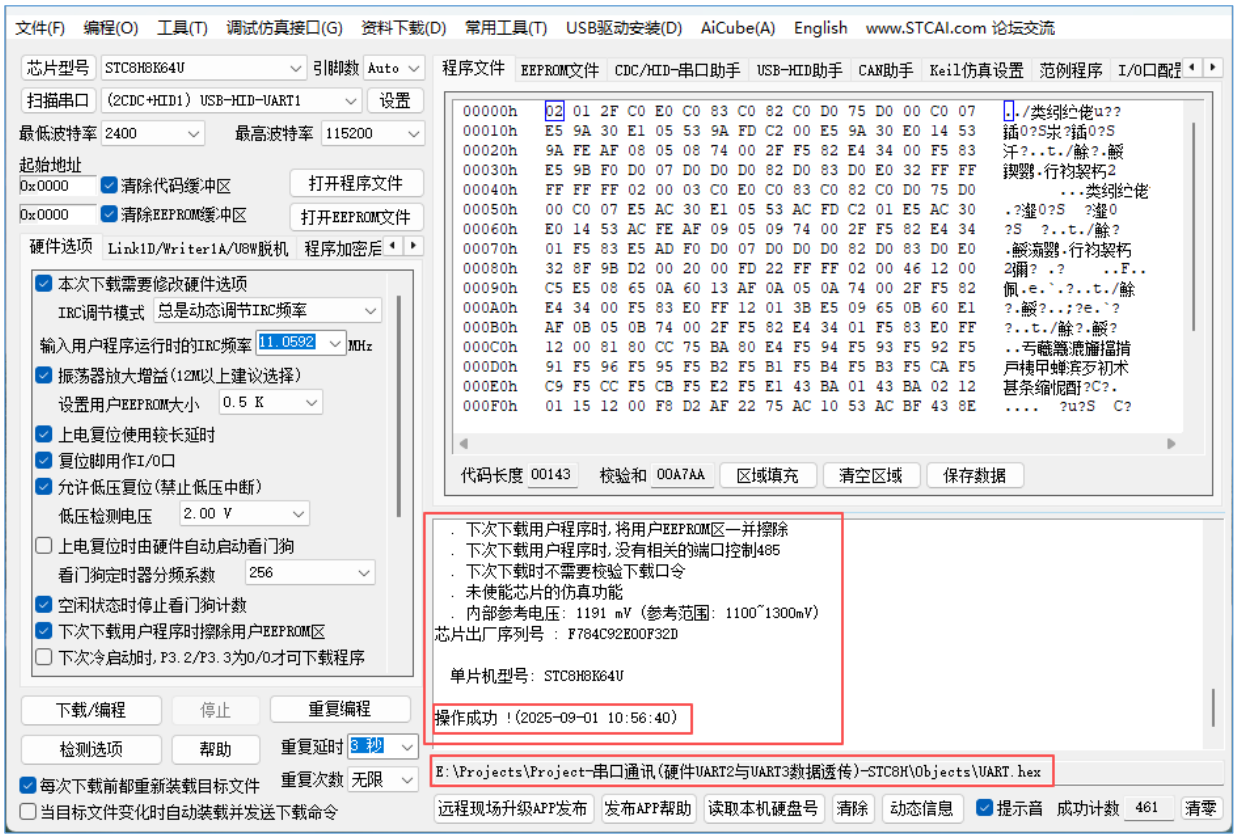
#endif
```

2.15.4　程序下载、观察实验现象，验证实验效果

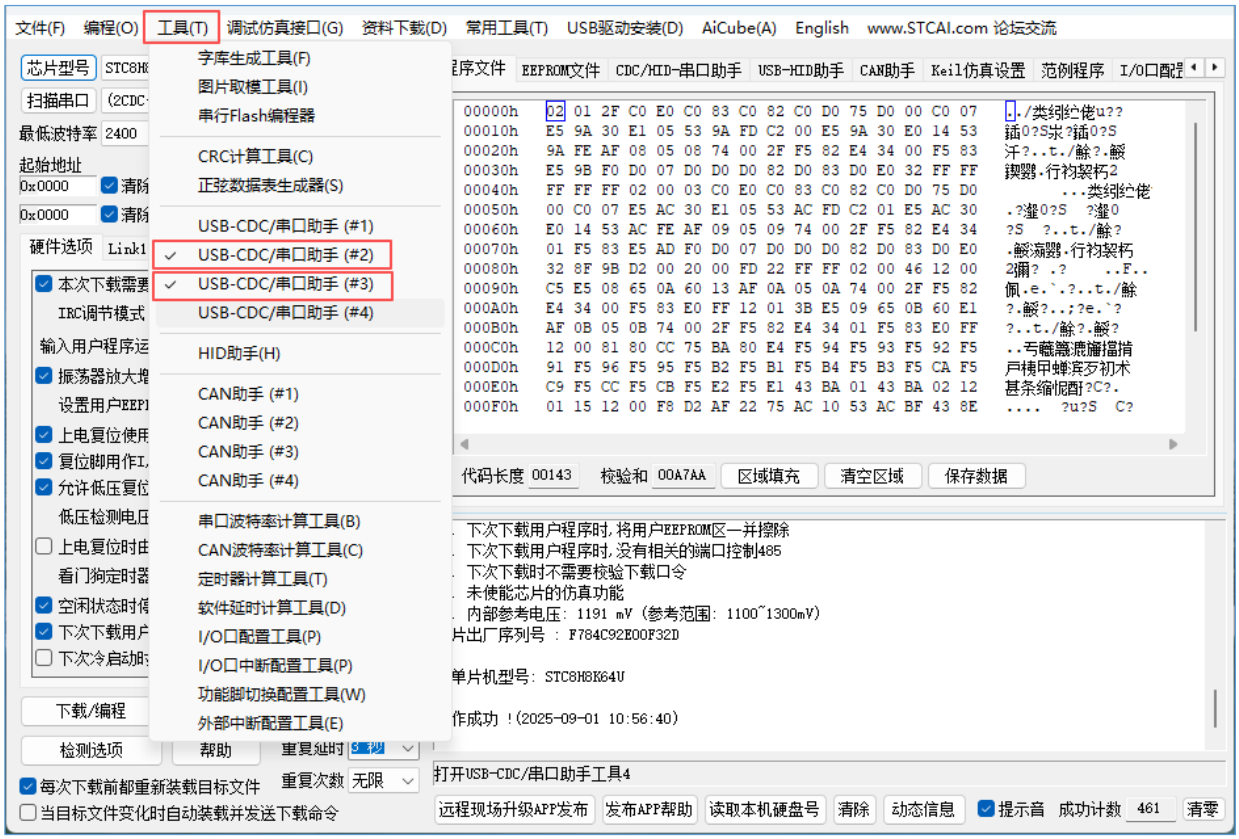
➲ 录入代码，保存，编译。按实验要求连接并设置好 STC8H8K64U 实验箱，如下图：



➲ 程序下载：运行 AiCube-ISP 软件系统，按步骤打开并下载程序“uart.hex”，如下图：



➲ 在 AiCube-ISP 软件系统的菜单栏，点击“工具”→勾选“USB-CDC/串口助手(#2)”和“USB-CDC/串口助手(#3)”，如下图：



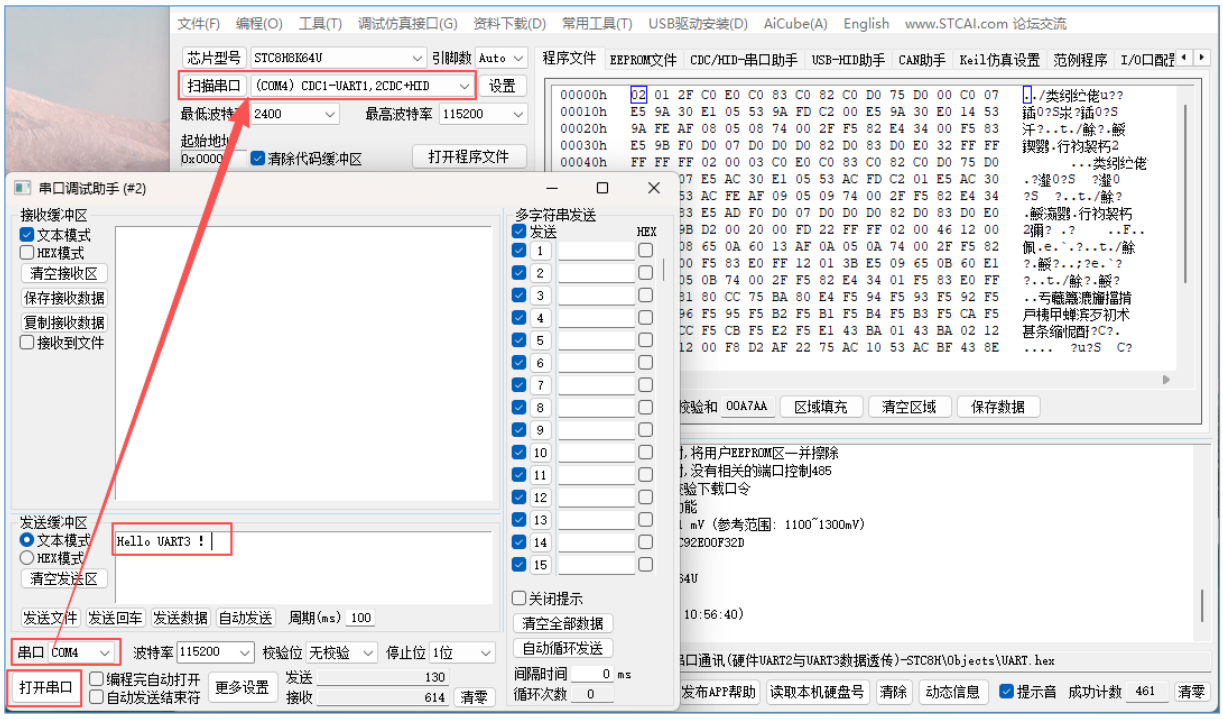
就会打开“串口调试助手(#2)”和“串口调试助手(#3)”两个浮窗。

➲ 在“串口调试助手(#2)”窗口：

➢ 选择与 CDC1 对应的 COM4 串口，

➢ 在“串口调试助手(#2)”窗口的发送缓冲区输入“Hello UART3！”

➢ 在点击“打开串口”按钮。如下图：

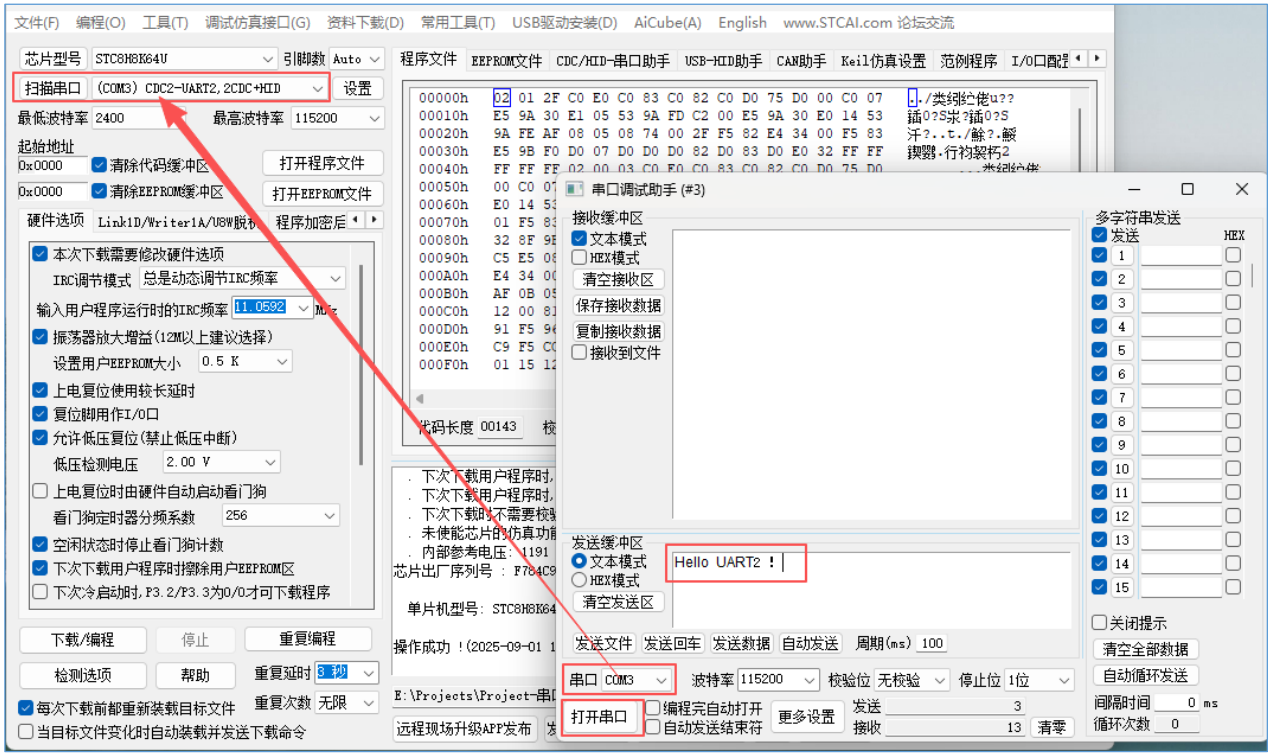


➲ 在“串口调试助手(#3)”窗口：

➢ 选择与 CDC2 对应的 COM3 串口

➢ 在“串口调试助手(#3)”窗口的发送缓冲区输入“Hello UART2！”

➢ 点击“打开串口”按钮。如下图：



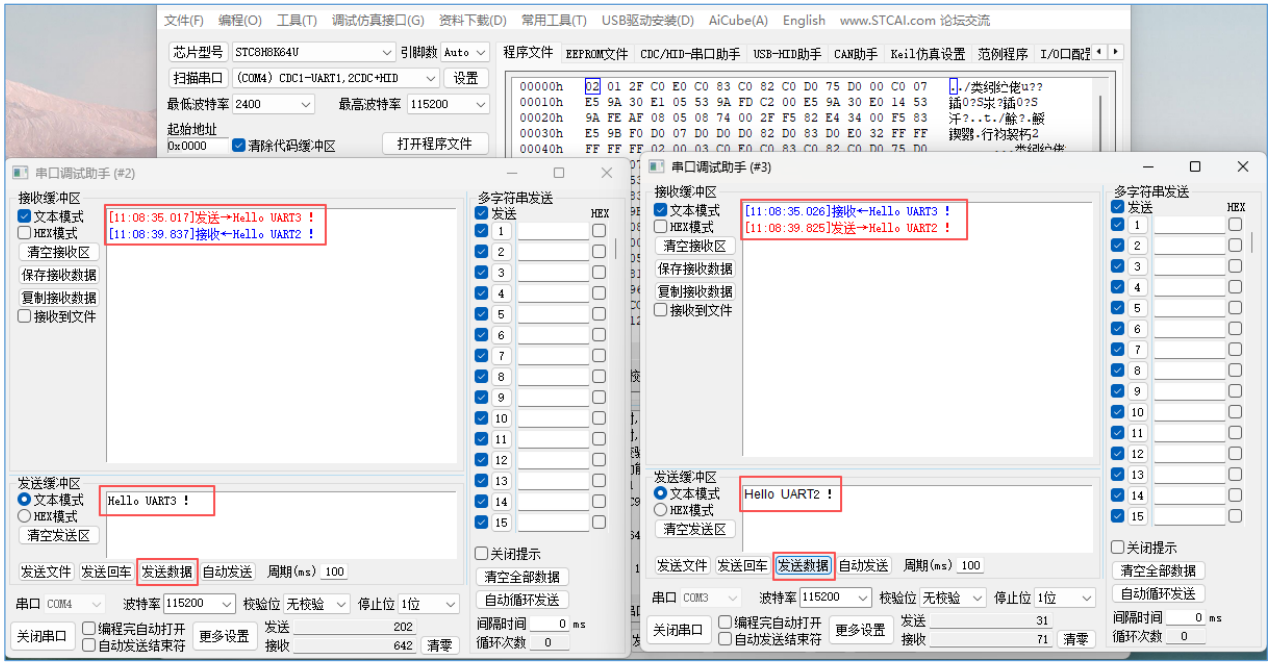
一切准备就绪！

➲　分别点击两个串口调试助手浮窗的“发送数据”按钮，我们发现：

在“串口调试助手#2”发送“Hello UART3!”，立即会在“串口调试助手#3”收到“Hello UART3!”

在“串口调试助手#3”发送“Hello UART2!”，立即会在“串口调试助手#2”收到“Hello UART2!”

如下图：



完成实验验证。

2.16 串口实验 -- 串口 1_2_3_4(中断设置标志位，主循环查询)，演示实验

```
/*演示串口 1、串口 2、串口 3、串口 4 中断同时打开时，
在各自的中断程序中设置中断事件标志位，
标志已产生相应的中断，在主循环中查询中断事件标志位，
来处理不是特急处理的任务，不堵塞其他中断*/

#include < STC8H.H >        //包含 STC8H 的头文件
#include "intrins.h"        //使用_nop_()函数所必须要包含的头文件,
                            //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                            //会导致编译器找不到这个而函数而报错。

unsigned char uart1_txflag = 0;//定义 1 个位变量，UART1 事件位变量标志，记录 UART1 已产生发送中断
// 供主循环查询 UART1 是否已产生发送中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断
unsigned char uart1_rxflag = 0;      //定义 1 个位变量，UART1 事件位变量标志，记录 UART1 已产生接收中断
// 供主循环查询 UART1 是否已产生接收中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断

unsigned char uart2_txflag = 0;//定义 1 个位变量，UART2 事件位变量标志，记录 UART2 已产生发送中断
// 供主循环查询 UART2 是否已产生发送中断，在主循环中处理 UART2 的中断事件任务，不堵塞其他中断
unsigned char uart2_rxflag = 0;      //定义 1 个位变量，UART2 事件位变量标志，记录 UART2 已产生接收中断
// 供主循环查询 UART2 是否已产生接收中断，在主循环中处理 UART2 的中断事件任务，不堵塞其他中断

unsigned char uart3_txflag = 0;//定义 1 个位变量，UART3 事件位变量标志，记录 UART3 已产生发送中断
// 供主循环查询 UART3 是否已产生发送中断，在主循环中处理 UART3 的中断事件任务，不堵塞其他中断
unsigned char uart3_rxflag = 0;      //定义 1 个位变量，UART3 事件位变量标志，记录 UART3 已产生接收中断
// 供主循环查询 UART3 是否已产生接收中断，在主循环中处理 UART3 的中断事件任务，不堵塞其他中断

unsigned char uart4_txflag = 0;//定义 1 个位变量，UART4 事件位变量标志，记录 UART4 已产生发送中断
// 供主循环查询 UART4 是否已产生发送中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断
unsigned char uart4_rxflag = 0;      //定义 1 个位变量，UART4 事件位变量标志，记录 UART4 已产生接收中断
// 供主循环查询 UART4 是否已产生接收中断，在主循环中处理 UART4 的中断事件任务，不堵塞其他中断

void Uart1_Init(void)       //115200bps@11.0592MHz
{
    SCON = 0x50;            //8 位数据,可变波特率
    AUXR |= 0x01;           //串口 1 选择定时器 2 为波特率发生器
    AUXR |= 0x04;           //定时器时钟 1T 模式
    T2L = 0xE8;             //设置定时初始值
    T2H = 0xFF;             //设置定时初始值
    AUXR |= 0x10;           //定时器 2 开始计时
    ES = 1;                 //使能串口 1 中断
}

void Uart2_Init(void)       //115200bps@11.0592MHz
{
    S2CON = 0x50;           //8 位数据,可变波特率
```

```
    AUXR |= 0x04;           //定时器时钟 1T 模式
    T2L = 0xE8;             //设置定时初始值
    T2H = 0xFF;             //设置定时初始值
    AUXR |= 0x10;           //定时器 2 开始计时
    IE2 |= 0x01;            //使能串口 2 中断
}

void Uart3_Init(void)       //115200bps@11.0592MHz
{
    S3CON = 0x10;           //8 位数据,可变波特率
    S3CON &= 0xBF;          //串口 3 选择定时器 2 为波特率发生器
    AUXR |= 0x04;           //定时器时钟 1T 模式
    T2L = 0xE8;             //设置定时初始值
    T2H = 0xFF;             //设置定时初始值
    AUXR |= 0x10;           //定时器 2 开始计时
    IE2 |= 0x08;            //使能串口 3 中断
}

void Uart4_Init(void)       //115200bps@11.0592MHz
{
    S4CON = 0x10;           //8 位数据,可变波特率
    S4CON &= 0xBF;          //串口 4 选择定时器 2 为波特率发生器
    AUXR |= 0x04;           //定时器时钟 1T 模式
    T2L = 0xE8;             //设置定时初始值
    T2H = 0xFF;             //设置定时初始值
    AUXR |= 0x10;           //定时器 2 开始计时
    IE2 |= 0x10;            //使能串口 4 中断
}

void main (void)
{
    P_SW2 = 0x80;                   //使能访问扩展 XFR

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x0c;       //P32、P33 设置为高阻输入(需要同步开启上拉电阻)
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    uart1_txflag = 0;               //初始化用户标志位
    uart1_rxflag = 0;               //初始化用户标志位
    uart2_txflag = 0;               //初始化用户标志位
```

```
    uart2_rxflag = 0;               //初始化用户标志位
    uart3_txflag = 0;               //初始化用户标志位
    uart3_rxflag = 0;               //初始化用户标志位
    uart4_txflag = 0;               //初始化用户标志位
    uart4_rxflag = 0;               //初始化用户标志位

    Uart1_Init();                   //调用 UART1 初始化函数
    Uart2_Init();                   //调用 UART2 初始化函数
    Uart3_Init();                   //调用 UART3 初始化函数
    Uart4_Init();                   //调用 UART4 初始化函数

    EA = 1;                         //总中断允许位打开
    P40 = 0;                        //打开 LED 灯供电
    while(1)                        //主循环中查询需要处理的各种事件
    {
        /*  本演示程序中，主循环查询各中断有无需要继续处理的事件的次序，
        依次是 UART/UART2/UART3/UART4, 用户可以自己根据实际情况，
        调整查询各中断有无需要继续处理的事件的优先次序  */

        if(uart1_txflag)
        {   //主循环中查询，UART1 是否已产生发送中断，是否有需要处理的 UART1 发送事件
            uart1_txflag = 0;   //清 0，UART1 发送事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart1_rxflag)
        {   //主循环中查询，UART1 是否已产生接收中断，是否有需要处理的 UART1 接收事件
            uart1_rxflag = 0;   //清 0，UART1 接收事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart2_txflag)
        {   //主循环中查询，UART2 是否已产生发送中断，是否有需要处理的 UART2 发送事件
            uart2_txflag = 0;   //清 0，UART2 发送事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart2_rxflag)
        {   //主循环中查询，UART2 是否已产生接收中断，是否有需要处理的 UART2 接收事件
            uart2_rxflag = 0;   //清 0，UART2 接收事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
```

```
        if(uart3_txflag)
        {   //主循环中查询，UART3 是否已产生发送中断，是否有需要处理的 UART3 发送事件
            uart3_txflag = 0;   //清 0，UART3 发送事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart3_rxflag)
        {   //主循环中查询，UART3 是否已产生接收中断，是否有需要处理的 UART3 接收事件
            uart3_rxflag = 0;   //清 0，UART3 接收事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart4_txflag)
        {   //主循环中查询，UART4 是否已产生发送中断，是否有需要处理的 UART4 发送事件
            uart4_txflag = 0;   //清 0，UART4 发送事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart4_rxflag)
        {   //主循环中查询，UART4 是否已产生接收中断，是否有需要处理的 UART4 接收事件
            uart4_rxflag = 0;   //清 0，UART4 接收事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
    }
}

void Uart1_Isr(void) interrupt UART1_VECTOR
{
    if (TI)                     //检测串口 1 发送中断
    {
        TI = 0;                 //清除串口 1 发送中断请求位
        _nop_();                //特急处理
            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
            //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart1_txflag = 1;
            // uart1_txflag 置 1 是通知主循环处理部分串口 1 发送中断事件不需要特急处理的任务
            //置 1，记录 UART1 已产生发送中断，供主循环查询判断有无需处理的 UART1 发送任务
    }
    if (RI)                     //检测串口 1 接收中断
    {
        RI = 0;                 //清除串口 1 接收中断请求位
```

```
            _nop_();                    //特急处理
                //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                //但时间不要太长，否则会影响其他中断事件的实时响应速度
            uart1_rxflag = 1;
                // uart1_rxflag 置 1 是通知主循环处理部分串口 1 接收中断事件不需要特急处理的任务
                //置 1，记录 UART1 已产生接收中断，供主循环查询判断有无需处理的 UART1 接收任务
        }
    }
    //UART1 中断服务程序，UART1_VECTOR 在 STC8H.H 头文件中已宏定义为 4

    void Uart2_Isr(void) interrupt UART2_VECTOR
    {
        if (S2CON & 0x02)           //检测串口 2 发送中断
        {
            S2CON &= ~0x02;         //清除串口 2 发送中断请求位
            _nop_();                //特急处理
                //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                //但时间不要太长，否则会影响其他中断事件的实时响应速度
            uart2_txflag = 1;
                // uart2_txflag 置 1 是通知主循环处理部分串口 2 发送中断事件不需要特急处理的任务
                //置 1，记录 UART2 已产生发送中断，供主循环查询判断有无需处理的 UART2 发送任务
        }

        if (S2CON & 0x01)           //检测串口 2 接收中断
        {
            S2CON &= ~0x01;         //清除串口 2 接收中断请求位
            _nop_();                //特急处理
                //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                //但时间不要太长，否则会影响其他中断事件的实时响应速度
            uart2_rxflag = 1;
                // uart2_rxflag 置 1 是通知主循环处理部分串口 2 接收中断事件不需要特急处理的任务
                //置 1，记录 UART2 已产生接收中断，供主循环查询判断有无需处理的 UART2 接收任务
        }
    }
    //UART2 中断服务程序，UART2_VECTOR 在 STC8H.H 头文件中已宏定义为 8

    void Uart3_Isr(void) interrupt UART3_VECTOR
    {
        if (S3CON & 0x02)           //检测串口 3 发送中断
        {
            S3CON &= ~0x02;         //清除串口 3 发送中断请求位
            _nop_();                //特急处理
                //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                //但时间不要太长，否则会影响其他中断事件的实时响应速度
            uart3_txflag = 1;
                // uart3_txflag 置 1 是通知主循环处理部分串口 3 发送中断事件不需要特急处理的任务
```

```
            //置 1，记录 UART3 已产生发送中断，供主循环查询判断有无需处理的 UART3 发送任务
    }

    if (S3CON & 0x01)           //检测串口 3 接收中断
    {
        S3CON &= ~0x01;         //清除串口 3 接收中断请求位
        _nop_();                //特急处理
            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
            //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart3_rxflag = 1;
            // uart3_rxflag 置 1 是通知主循环处理部分串口 3 接收中断事件不需要特急处理的任务
            //置 1，记录 UART3 已产生接收中断，供主循环查询判断有无需处理的 UART3 接收任务
    }
}
//UART3 中断服务程序，UART3_VECTOR 在 STC8H.H 头文件中已宏定义为 17

void Uart4_Isr(void) interrupt UART4_VECTOR
{
    if (S4CON & 0x02)           //检测串口 4 发送中断
    {
        S4CON &= ~0x02;         //清除串口 4 发送中断请求位
        _nop_();                //特急处理
            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
            //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart4_txflag = 1;
            // uart4_txflag 置 1 是通知主循环处理部分串口 4 发送中断事件不需要特急处理的任务
            //置 1，记录 UART4 已产生发送中断，供主循环查询判断有无需处理的 UART4 发送任务
    }

    if (S4CON & 0x01)           //检测串口 4 接收中断
    {
        S4CON &= ~0x01;         //清除串口 4 接收中断请求位
        _nop_();                //特急处理
            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
            //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart4_rxflag = 1;
            // uart4_rxflag 置 1 是通知主循环处理部分串口 4 接收中断事件不需要特急处理的任务
            //置 1，记录 UART4 已产生接收中断，供主循环查询判断有无需处理的 UART4 接收任务
    }
}
//UART4 中断服务程序，UART4_VECTOR 在 STC8H.H 头文件中已宏定义为 18
```

2.17 串口实验 -- 串口 1_2_3_4 + 定时器 0_1_3_4 + 外部中断 INT0_1_2_3 (中断设置标志位，主循环查询)，演示实验

```
/*演示串口 1/串口 2/串口 3/串口 4 + 定时器 0/1/3/4 + INT0/1/2/3 中断同时打开时，
在各自的中断程序中设置中断事件标志位，标志已产生相应的中断，
在主循环中查询中断事件标志位，来处理不是特急处理的任务，不堵塞其他中断*/

#include < STC8H.H >        //包含 STC8H 的头文件

#include "intrins.h"        //使用_nop_()函数所必须要包含的头文件,
                            //否则延时函数中调用的_nop_()函数没有被头文件引用过来，
                            //会导致编译器找不到这个而函数而报错。

unsigned char int0_flag = 0;  //定义 1 个位变量，INT0 事件位变量标志，记录 INT0 已产生中断
// 供主循环查询 INT0 是否已产生中断，在主循环中处理 INT0 的中断事件任务，不堵塞其他中断
unsigned char int1_flag = 0;  //定义 1 个位变量，INT1 事件位变量标志，记录 INT1 已产生中断
// 供主循环查询 INT1 是否已产生中断，在主循环中处理 INT1 的中断事件任务，不堵塞其他中断
unsigned char int2_flag = 0;  //定义 1 个位变量，INT2 事件位变量标志，记录 INT2 已产生中断
// 供主循环查询 INT2 是否已产生中断，在主循环中处理 INT2 的中断事件任务，不堵塞其他中断
unsigned char int3_flag = 0;  //定义 1 个位变量，INT3 事件位变量标志，记录 INT3 已产生中断
// 供主循环查询 INT3 是否已产生中断，在主循环中处理 INT3 的中断事件任务，不堵塞其他中断

unsigned char t0_flag = 0;          //定义 1 个位变量，T0 事件位变量标志，记录定时器 0 已产生中断
// 供主循环查询定时器 0 是否已产生中断，在主循环中处理定时器 0 的中断事件任务，不堵塞其他中断
unsigned char t1_flag = 0;          //定义 1 个位变量，T1 事件位变量标志，记录定时器 1 已产生中断
// 供主循环查询定时器 1 是否已产生中断，在主循环中处理定时器 1 的中断事件任务，不堵塞其他中断
unsigned char t3_flag = 0;          //定义 1 个位变量，T3 事件位变量标志，记录定时器 3 已产生中断
// 供主循环查询定时器 3 是否已产生中断，在主循环中处理定时器 3 的中断事件任务，不堵塞其他中断
unsigned char t4_flag = 0;          //定义 1 个位变量，T4 事件位变量标志，记录定时器 4 已产生中断
// 供主循环查询定时器 4 是否已产生中断，在主循环中处理定时器 4 的中断事件任务，不堵塞其他中断

unsigned char uart1_txflag = 0;   //定义 1 个位变量，UART1 事件位变量标志，记录 UART1 已产生发送中断
// 供主循环查询 UART1 是否已产生发送中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断
unsigned char uart1_rxflag = 0;   //定义 1 个位变量，UART1 事件位变量标志，记录 UART1 已产生接收中断
// 供主循环查询 UART1 是否已产生接收中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断

unsigned char uart2_txflag = 0;   //定义 1 个位变量，UART2 事件位变量标志，记录 UART2 已产生发送中断
// 供主循环查询 UART2 是否已产生发送中断，在主循环中处理 UART2 的中断事件任务，不堵塞其他中断
unsigned char uart2_rxflag = 0;   //定义 1 个位变量，UART2 事件位变量标志，记录 UART2 已产生接收中断
// 供主循环查询 UART2 是否已产生接收中断，在主循环中处理 UART2 的中断事件任务，不堵塞其他中断

unsigned char uart3_txflag = 0;   //定义 1 个位变量，UART3 事件位变量标志，记录 UART3 已产生发送中断
// 供主循环查询 UART3 是否已产生发送中断，在主循环中处理 UART3 的中断事件任务，不堵塞其他中断
unsigned char uart3_rxflag = 0;   //定义 1 个位变量，UART3 事件位变量标志，记录 UART3 已产生接收中断
// 供主循环查询 UART3 是否已产生接收中断，在主循环中处理 UART3 的中断事件任务，不堵塞其他中断
```

```
unsigned char uart4_txflag = 0;    //定义 1 个位变量，UART4 事件位变量标志，记录 UART4 已产生发送中断
// 供主循环查询 UART4 是否已产生发送中断，在主循环中处理 UART1 的中断事件任务，不堵塞其他中断
unsigned char uart4_rxflag = 0;    //定义 1 个位变量，UART4 事件位变量标志，记录 UART4 已产生接收中断
// 供主循环查询 UART4 是否已产生接收中断，在主循环中处理 UART4 的中断事件任务，不堵塞其他中断

void Timer0_Init(void)        //定时器 0 初始化，2 秒@11.0592MHz
{
    TM0PS = 0x1C;    //设置定时器时钟预分频（注意:并非所有系列都有此寄存器,详情请查看数据手册 ）
    AUXR &= 0x7F;        //定时器时钟 12T 模式
    TMOD &= 0xF0;        //设置定时器模式
    TL0 = 0xB9;          //设置定时初始值
    TH0 = 0x07;          //设置定时初始值
    TF0 = 0;             //清除 TF0 标志
    TR0 = 1;             //定时器 0 开始计时
    ET0 = 1;             //使能定时器 0 中断
}

void Timer1_Init(void)        //定时器 1 初始化，500 毫秒@11.0592MHz
{
    TM1PS = 0x07;    //设置定时器时钟预分频（注意:并非所有系列都有此寄存器,详情请查看数据手册 ）
    AUXR &= 0xBF;        //定时器时钟 12T 模式
    TMOD &= 0x0F;        //设置定时器模式
    TL1 = 0x00;          //设置定时初始值
    TH1 = 0x1F;          //设置定时初始值
    TF1 = 0;             //清除 TF1 标志
    TR1 = 1;             //定时器 1 开始计时
    ET1 = 1;             //使能定时器 1 中断
}

void Timer3_Init(void)        //定时器 3 初始化，100 毫秒@11.0592MHz
{
    TM3PS = 0x10;    //设置定时器时钟预分频（注意:并非所有系列都有此寄存器,详情请查看数据手册 ）
    T4T3M |= 0x02;       //定时器时钟 1T 模式
    T3L = 0xE2;          //设置定时初始值
    T3H = 0x01;          //设置定时初始值
    T4T3M |= 0x08;       //定时器 3 开始计时
    IE2 |= 0x20;         //使能定时器 3 中断
}

void Timer4_Init(void)        //定时器 4 初始化，200 毫秒@11.0592MHz
{
    TM4PS = 0x21;    //设置定时器时钟预分频（注意:并非所有系列都有此寄存器,详情请查看数据手册 ）
    T4T3M |= 0x20;       //定时器时钟 1T 模式
    T4L = 0xE2;          //设置定时初始值
    T4H = 0x01;          //设置定时初始值
    T4T3M |= 0x80;       //定时器 4 开始计时
```

```
    IE2 |= 0x40;                //使能定时器 4 中断
}

void Uart1_Init(void)           //115200bps@11.0592MHz
{
    SCON = 0x50;                //8 位数据,可变波特率
    AUXR |= 0x01;               //串口 1 选择定时器 2 为波特率发生器
    AUXR |= 0x04;               //定时器时钟 1T 模式
    T2L = 0xE8;                 //设置定时初始值
    T2H = 0xFF;                 //设置定时初始值
    AUXR |= 0x10;               //定时器 2 开始计时
    ES = 1;                     //使能串口 1 中断
}

void Uart2_Init(void)           //115200bps@11.0592MHz
{
    S2CON = 0x50;               //8 位数据,可变波特率
    AUXR |= 0x04;               //定时器时钟 1T 模式
    T2L = 0xE8;                 //设置定时初始值
    T2H = 0xFF;                 //设置定时初始值
    AUXR |= 0x10;               //定时器 2 开始计时
    IE2 |= 0x01;                //使能串口 2 中断
}

void Uart3_Init(void)           //115200bps@11.0592MHz
{
    S3CON = 0x10;               //8 位数据,可变波特率
    S3CON &= 0xBF;              //串口 3 选择定时器 2 为波特率发生器
    AUXR |= 0x04;               //定时器时钟 1T 模式
    T2L = 0xE8;                 //设置定时初始值
    T2H = 0xFF;                 //设置定时初始值
    AUXR |= 0x10;               //定时器 2 开始计时
    IE2 |= 0x08;                //使能串口 3 中断
}

void Uart4_Init(void)           //115200bps@11.0592MHz
{
    S4CON = 0x10;               //8 位数据,可变波特率
    S4CON &= 0xBF;              //串口 4 选择定时器 2 为波特率发生器
    AUXR |= 0x04;               //定时器时钟 1T 模式
    T2L = 0xE8;                 //设置定时初始值
    T2H = 0xFF;                 //设置定时初始值
    AUXR |= 0x10;               //定时器 2 开始计时
    IE2 |= 0x10;                //使能串口 4 中断
}
```

```
void main (void)
{
    P_SW2 = 0x80;                   //使能访问扩展 XFR

    P0M0 = 0x00; P0M1 = 0x00;       //设置 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //设置 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //设置 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //设置 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //设置 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //设置 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //设置 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //设置 P7 口为准双向口模式

    int0_flag = 0;                  //初始化用户标志位
    int1_flag = 0;                  //初始化用户标志位
    int2_flag = 0;                  //初始化用户标志位
    int3_flag = 0;                  //初始化用户标志位

    t0_flag = 0;                    //初始化用户标志位
    t1_flag = 0;                    //初始化用户标志位
    t3_flag = 0;                    //初始化用户标志位
    t4_flag = 0;                    //初始化用户标志位

    uart1_txflag = 0;               //初始化用户标志位
    uart1_rxflag = 0;               //初始化用户标志位
    uart2_txflag = 0;               //初始化用户标志位
    uart2_rxflag = 0;               //初始化用户标志位
    uart3_txflag = 0;               //初始化用户标志位
    uart3_rxflag = 0;               //初始化用户标志位
    uart4_txflag = 0;               //初始化用户标志位
    uart4_rxflag = 0;               //初始化用户标志位

    IT0 = 0;                        //使能 INT0 上升沿和下降沿中断
    //IT0 = 1;                      //使能 INT0 下降沿中断
    EX0 = 1;                        //使能 INT0 中断
    IE0 = 0;                        //清 INT0 中断标志

    //IT1 = 0;                      //使能 INT1 上升沿和下降沿中断
    IT1 = 1;                        //使能 INT1 下降沿中断
    EX1 = 1;                        //使能 INT1 中断
    IE1 = 0;                        //清 INT1 中断标志

    INTCLKO |= 0x10;                //使能 INT2 中断
    INTCLKO |= 0x20;                //使能 INT3 中断

    Timer0_Init();                  //调用定时器 0 初始化函数
```

```
    Timer1_Init();              //调用定时器 1 初始化函数
    Timer3_Init();              //调用定时器 3 初始化函数
    Timer4_Init();              //调用定时器 4 初始化函数

    Uart1_Init();               //调用 UART1 初始化函数
    Uart2_Init();               //调用 UART2 初始化函数
    Uart3_Init();               //调用 UART3 初始化函数
    Uart4_Init();               //调用 UART4 初始化函数

    EA = 1;                     //总中断允许位打开

    while(1)                    //主循环中查询需要处理的各种事件
    {
        /*  本演示程序中，主循环查询各中断有无需要继续处理的事件的次序，
        依次是 INTx/TIMERx/UARTx, 用户可以自己根据实际情况，
        调整查询各中断有无需要继续处理的事件的优先次序  */

        //查询外部中断事件
        if(int0_flag)           //主循环中查询，INT0 是否已产生中断，是否有需要处理的 INT0 事件
        {
            int0_flag = 0;      //清 0，INT0 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(int1_flag)           //主循环中查询，INT1 是否已产生中断，是否有需要处理的 INT1 事件
        {
            int1_flag = 0;      //清 0，INT1 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(int2_flag)           //主循环中查询，INT2 是否已产生中断，是否有需要处理的 INT2 事件
        {
            int2_flag = 0;      //清 0，INT2 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }

        if(int3_flag)           //主循环中查询，INT3 是否已产生中断，是否有需要处理的 INT3 事件
        {
            int3_flag = 0;      //清 0，INT3 事件位变量标志
            _nop_();            //用户在此添加需要处理的事件
            _nop_();
        }
```

```
//查询定时中断事件
if(t0_flag)     //主循环中查询，定时器 0 是否已产生中断，是否有需要处理的定时器 0 事件
{
    t0_flag = 0;        //清 0，T0 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(t1_flag)     //主循环中查询，定时器 1 是否已产生中断，是否有需要处理的定时器 1 事件
{
    t1_flag = 0;        //清 0，T1 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(t3_flag)     //主循环中查询，定时器 3 是否已产生中断，是否有需要处理的定时器 3 事件
{
    t3_flag = 0;        //清 0，T3 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(t4_flag)     //主循环中查询，定时器 4 是否已产生中断，是否有需要处理的定时器 4 事件
{
    t4_flag = 0;        //清 0，T4 事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

//查询串口中断事件
if(uart1_txflag)    //主循环中查询，UART1 是否已产生发送中断，是否有需要处理的 UART1 发送事件
{
    uart1_txflag = 0;   //清 0，UART1 发送事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart1_rxflag)    //主循环中查询，UART1 是否已产生接收中断，是否有需要处理的 UART1 接收事件
{
    uart1_rxflag = 0;   //清 0，UART1 接收事件位变量标志
    _nop_();            //用户在此添加需要处理的事件
    _nop_();
}

if(uart2_txflag)    //主循环中查询，UART2 是否已产生发送中断，是否有需要处理的 UART2 发送事件
{
```

```
            uart2_txflag = 0;    //清 0，UART2 发送事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart2_rxflag)     //主循环中查询，UART2 是否已产生接收中断，是否有需要处理的 UART2 接收事件
        {
            uart2_rxflag = 0;    //清 0，UART2 接收事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart3_txflag)     //主循环中查询，UART3 是否已产生发送中断，是否有需要处理的 UART3 发送事件
        {
            uart3_txflag = 0;    //清 0，UART3 发送事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart3_rxflag)     //主循环中查询，UART3 是否已产生接收中断，是否有需要处理的 UART3 接收事件
        {
            uart3_rxflag = 0;    //清 0，UART3 接收事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart4_txflag)     //主循环中查询，UART4 是否已产生发送中断，是否有需要处理的 UART4 发送事件
        {
            uart4_txflag = 0;    //清 0，UART4 发送事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }

        if(uart4_rxflag)     //主循环中查询，UART4 是否已产生接收中断，是否有需要处理的 UART4 接收事件
        {
            uart4_rxflag = 0;    //清 0，UART4 接收事件位变量标志
            _nop_();             //用户在此添加需要处理的事件
            _nop_();
        }
    }
}

void int0_isr(void) interrupt INT0_VECTOR
{
    _nop_();             //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                         //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
```

```
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
    int0_flag = 1;      // int0_flag 置 1 是通知主循环处理部分 INT0 中断事件不需要特急处理的任务
                        //置 1，记录 INT0 已产生中断，供主循环查询判断有无需处理的 INT0 任务

    if(INT0)            //边沿中断，进入后再次判断电平从而判断是什么样的电平
    {
        _nop_();        //判断为高电平，则当前为上升沿
        _nop_();        //可以在这里插入断点进行观察现象
    }
    else
    {
        _nop_();        //判断为低电平，则当前为下降沿
        _nop_();        //可以在这里插入断点进行观察现象
    }
}
//INT0 中断服务程序，INT0_VECTOR 在 STC8H.H 头文件中已宏定义为 0

void int1_isr(void) interrupt INT1_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
    int1_flag = 1;      // int1_flag 置 1 是通知主循环处理部分 INT1 中断事件不需要特急处理的任务
                        //置 1，记录 INT1 已产生中断，供主循环查询判断有无需处理的 INT1 任务
}
//INT1 中断服务程序，INT1_VECTOR 在 STC8H.H 头文件中已宏定义为 2

void int2_isr(void) interrupt INT2_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
    int2_flag = 1;      // int2_flag 置 1 是通知主循环处理部分 INT2 中断事件不需要特急处理的任务
                        //置 1，记录 INT2 已产生中断，供主循环查询判断有无需处理的 INT2 任务
}
//INT2 中断服务程序，INT2_VECTOR 在 STC8H.H 头文件中已宏定义为 10

void int3_isr(void) interrupt INT3_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
    int3_flag = 1;      // int3_flag 置 1 是通知主循环处理部分 INT3 中断事件不需要特急处理的任务
                        //置 1，记录 INT3 已产生中断，供主循环查询判断有无需处理的 INT3 任务
}
//INT3 中断服务程序，INT3_VECTOR 在 STC8H.H 头文件中已宏定义为 11
```

```
void Timer0_Isr(void) interrupt TMR0_VECTOR      //定时器 0 中断服务程序
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t0_flag = 1;        // t0_flag 置 1 是通知主循环处理部分 T0 中断事件不需要特急处理的任务
                        //置 1，记录定时器 0 已产生中断，供主循环查询判断有无需处理的定时器 0 任务
}
//定时器 0 中断服务程序，TMR0_VECTOR 在 STC8H.H 头文件中已宏定义为 1

void Timer1_Isr(void) interrupt TMR1_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t1_flag = 1;        // t1_flag 置 1 是通知主循环处理部分 T1 中断事件不需要特急处理的任务
                        //置 1，记录定时器 1 已产生中断，供主循环查询判断有无需处理的定时器 1 任务
}
//定时器 1 中断服务程序，TMR1_VECTOR 在 STC8H.H 头文件中已宏定义为 3

void Timer3_Isr(void) interrupt TMR3_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t3_flag = 1;        // t3_flag 置 1 是通知主循环处理部分 T3 中断事件不需要特急处理的任务
                        //置 1，记录定时器 3 已产生中断，供主循环查询判断有无需处理的定时器 3 任务
}
//定时器 3 中断服务程序，TMR3_VECTOR 在 STC8H.H 头文件中已宏定义为 19

void Timer4_Isr(void) interrupt TMR4_VECTOR
{
    _nop_();            //特急处理，中断服务程序中尽量少执行长的任务，防止堵塞其他中断
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
    t4_flag = 1;        // t4_flag 置 4 是通知主循环处理部分 T4 中断事件不需要特急处理的任务
                        //置 1，记录定时器 4 已产生中断，供主循环查询判断有无需处理的定时器 4 任务
}
//定时器 4 中断服务程序，TMR4_VECTOR 在 STC8H.H 头文件中已宏定义为 20

void Uart1_Isr(void) interrupt UART1_VECTOR
{
    if (TI)                 //检测串口 1 发送中断
    {
        TI = 0;             //清除串口 1 发送中断请求位
        _nop_();            //特急处理
```

```
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart1_txflag = 1;   // uart1_txflag 置 1 是通知主循环处理部分串口 1 发送中断事件不需要特急处理的任务
                            //置 1，记录 UART1 已产生发送中断，供主循环查询判断有无需处理的 UART1 发送任务
    }

    if (RI)                 //检测串口 1 接收中断
    {
        RI = 0;             //清除串口 1 接收中断请求位
        _nop_();            //特急处理
                            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                            //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart1_rxflag = 1;   // uart1_rxflag 置 1 是通知主循环处理部分串口 1 接收中断事件不需要特急处理的任务
                            //置 1，记录 UART1 已产生接收中断，供主循环查询判断有无需处理的 UART1 接收任务
    }
}
//UART1 中断服务程序，UART1_VECTOR 在 STC8H.H 头文件中已宏定义为 4

void Uart2_Isr(void) interrupt UART2_VECTOR
{
    if (S2CON & 0x02)       //检测串口 2 发送中断
    {
        S2CON &= ~0x02;     //清除串口 2 发送中断请求位
        _nop_();            //特急处理
                            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                            //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart2_txflag = 1;   // uart2_txflag 置 1 是通知主循环处理部分串口 2 发送中断事件不需要特急处理的任务
                            //置 1，记录 UART2 已产生发送中断，供主循环查询判断有无需处理的 UART2 发送任务
    }

    if (S2CON & 0x01)       //检测串口 2 接收中断
    {
        S2CON &= ~0x01;     //清除串口 2 接收中断请求位
        _nop_();            //特急处理
                            //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                            //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart2_rxflag = 1;   // uart2_rxflag 置 1 是通知主循环处理部分串口 2 接收中断事件不需要特急处理的任务
                            //置 1，记录 UART2 已产生接收中断，供主循环查询判断有无需处理的 UART2 接收任务
    }
}
//UART2 中断服务程序，UART2_VECTOR 在 STC8H.H 头文件中已宏定义为 8

void Uart3_Isr(void) interrupt UART3_VECTOR
{
    if (S3CON & 0x02)       //检测串口 3 发送中断
    {
```

```
        S3CON &= ~0x02; //清除串口 3 发送中断请求位
        _nop_();        //特急处理
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart3_txflag = 1;   // uart3_txflag 置 1 是通知主循环处理部分串口 3 发送中断事件不需要特急处理的任务
                            //置 1，记录 UART3 已产生发送中断，供主循环查询判断有无需处理的 UART3 发送任务
    }

    if (S3CON & 0x01)       //检测串口 3 接收中断
    {
        S3CON &= ~0x01; //清除串口 3 接收中断请求位
        _nop_();        //特急处理
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart3_rxflag = 1;   // uart3_rxflag 置 1 是通知主循环处理部分串口 3 接收中断事件不需要特急处理的任务
                            //置 1，记录 UART3 已产生接收中断，供主循环查询判断有无需处理的 UART3 接收任务
    }
}
//UART3 中断服务程序，UART3_VECTOR 在 STC8H.H 头文件中已宏定义为 17

void Uart4_Isr(void) interrupt UART4_VECTOR
{
    if (S4CON & 0x02)       //检测串口 4 发送中断
    {
        S4CON &= ~0x02; //清除串口 4 发送中断请求位
        _nop_();        //特急处理
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart4_txflag = 1;   // uart4_txflag 置 1 是通知主循环处理部分串口 4 发送中断事件不需要特急处理的任务
                            //置 1，记录 UART4 已产生发送中断，供主循环查询判断有无需处理的 UART4 发送任务
    }

    if (S4CON & 0x01)       //检测串口 4 接收中断
    {
        S4CON &= ~0x01; //清除串口 4 接收中断请求位
        _nop_();        //特急处理
                        //以上程序代表部分需特急处理的中断事件，可在中断服务程序中直接处理
                        //但时间不要太长，否则会影响其他中断事件的实时响应速度
        uart4_rxflag = 1;   // uart4_rxflag 置 1 是通知主循环处理部分串口 4 接收中断事件不需要特急处理的任务
                            //置 1，记录 UART4 已产生接收中断，供主循环查询判断有无需处理的 UART4 接收任务
    }
}
//UART4 中断服务程序，UART4_VECTOR 在 STC8H.H 头文件中已宏定义为 18
```

2.18 串口实验 -- 串口通讯(串口通讯,串口绘图,ADC 检测按键,ADC 热敏电阻测温)

2.18.1 实验介绍

1、本实验项目主要目的：

- 掌握串口通讯，了解串口绘图工具
- 继续熟悉 ADC 的应用，熟悉 ADC 检测按键，熟悉 ADC 热敏电阻（NTC）测温

2、掌握 STC8H8K64U 实验箱原理图中的串口通信部分电路，ADC 应用部分电路

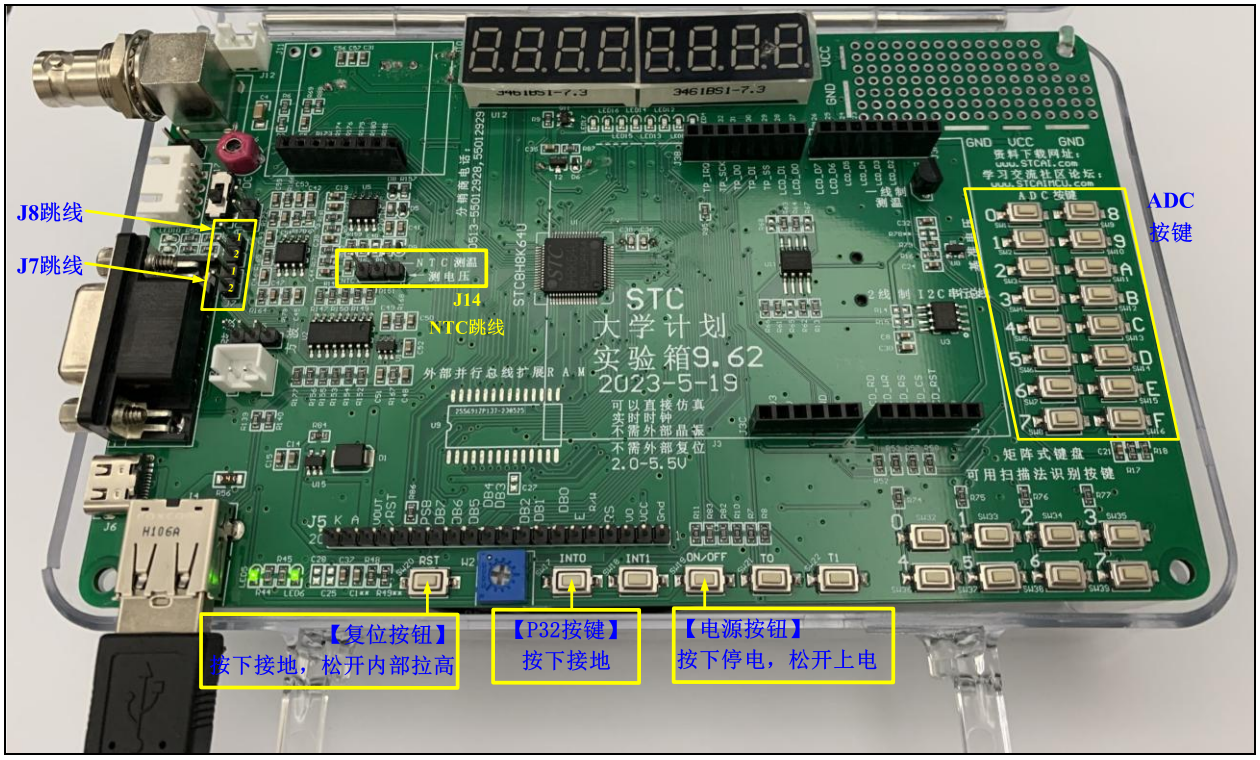
3、继续熟悉 AiCube-ISP 系统软件中串口助手工具，了解串口绘图工具

4、掌握 USB 转双串口工具@Ai8H2K12U 的原理及应用

5、继续熟悉如何管理多文件项目

2.18.1.1 认识 STC8H8K64U 实验箱

STC8H8K64U 的实验箱正面图（稍微倾斜角度拍摄，便于看清跳线引脚）：



1、J8： STC8H8K64U 实验箱上 J8 的 PIN1 通过 1 根杜邦线将 UART2 的 P4.7/TxD2_2 发送脚和双串口工具（Ai8H2K12U）的 S-TxD2 相连接。（详见 J7 和 J8 跳线原理图）

2、J7： STC8H8K64U 实验箱上 J7 的 PIN1 通过 1 根杜邦线将 UART2 的 P4.6/RxD2_2 接收脚和双串口工具（Ai8H2K12U）的 S-RxD2 相连接。（详见 J7 和 J8 跳线原理图）

3、NTC 跳线：Negative Temperature Coefficient thermistor，即“负温度系数热敏电阻”，其电阻值随温度升高而减小，ADC 测温通过 ADC3 通道采集 NTC 温敏电阻的电压值来计算出温度值。详见 NTC 原理图所示，NTC 温敏电阻与 10K 精密电阻 R171 串联分压后的电压输入到 ADC3 脚位，而温敏电阻随温度变化，阻值会产生变化，从而导致分压电压跟着改变，于是通过

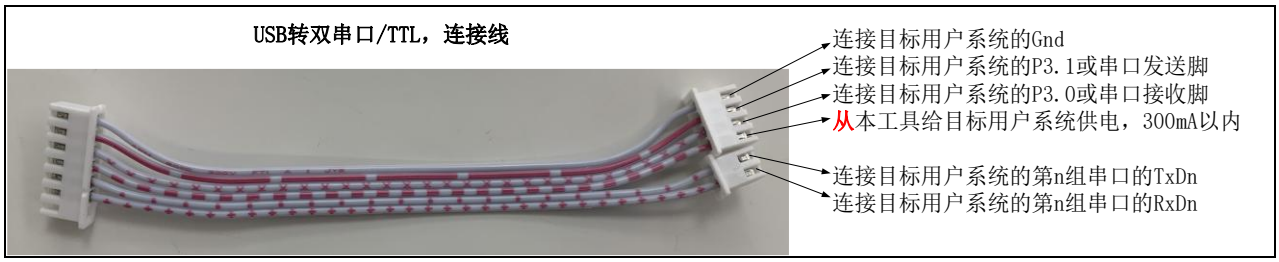
ADC 采集分压电压值进行 ADC 转换，将转换的 ADC 结果通过 UART2 串口发送到“串口绘图”显示实时采样数据。

4、ADC 按键：按下不同的 ADC 按键可以改变 P1.0-ADC-KEY 脚的电压，ADC 通过采集 P1.0-ADC-KEY 脚的电压进行 ADC 转换，然后将转换的 ADC 结果通过 UART2 串口发送到“串口绘图”显示实时采样数据。

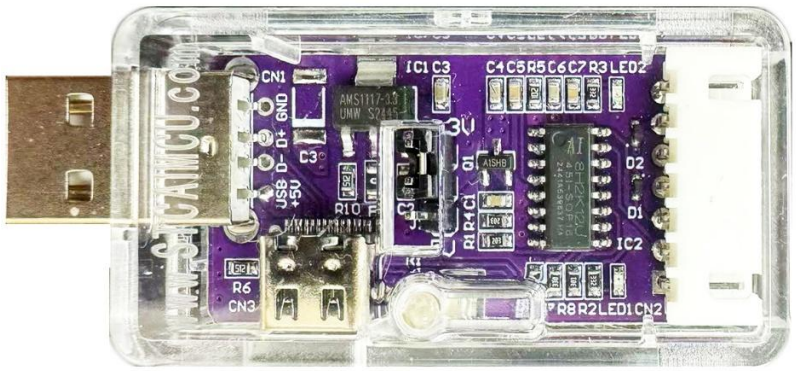
2.18.1.2　2 根杜邦线（1 公头、1 母头）



2.18.1.3　USB 转串口/TTL 连接线



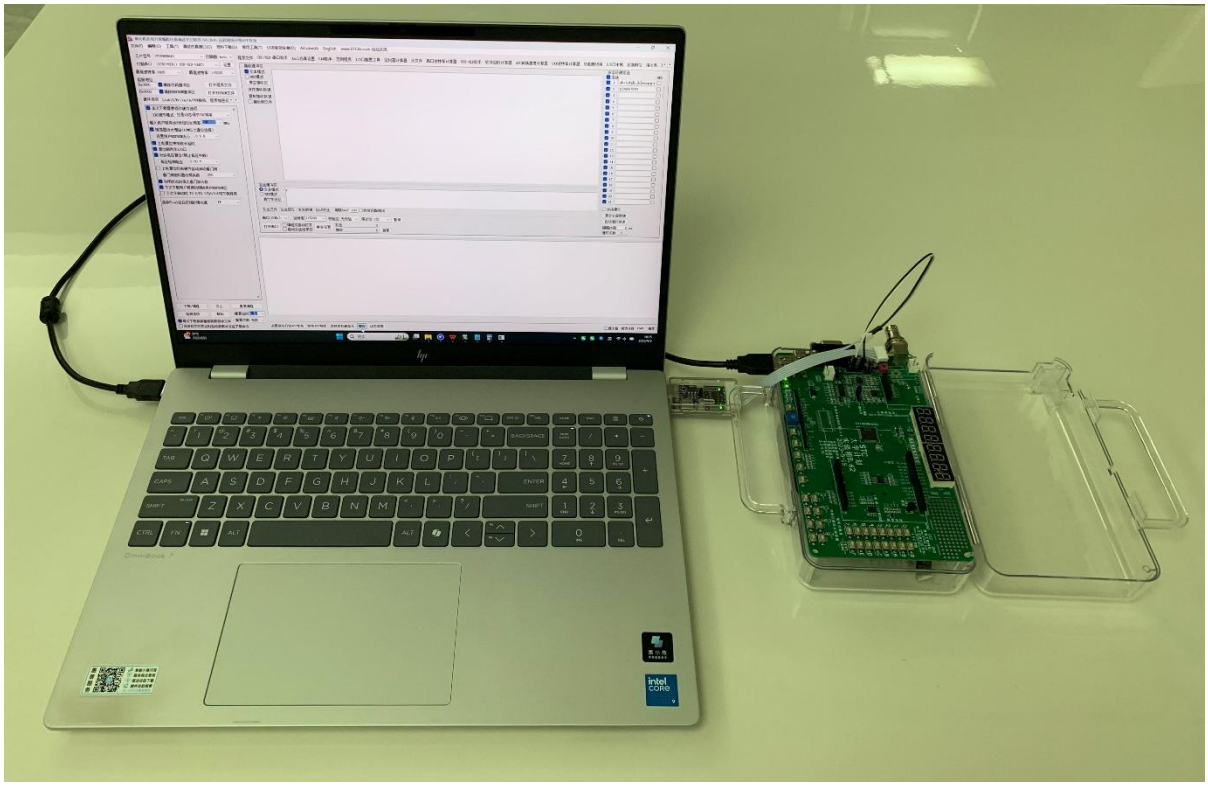
2.18.1.4　USB 转双串口工具@Ai8H2K12U：正面



2.18.1.5　USB 转双串口工具@Ai8H2K12U：背面



2.18.1.6 本实验实验箱与电脑连接实物图



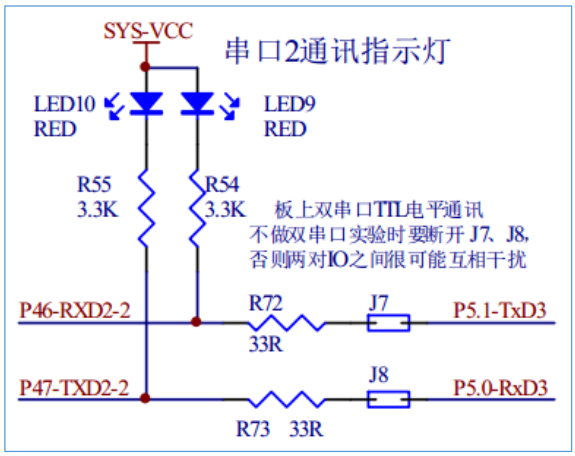
2.18.1.7 Keil 环境下多文件项目管理说明

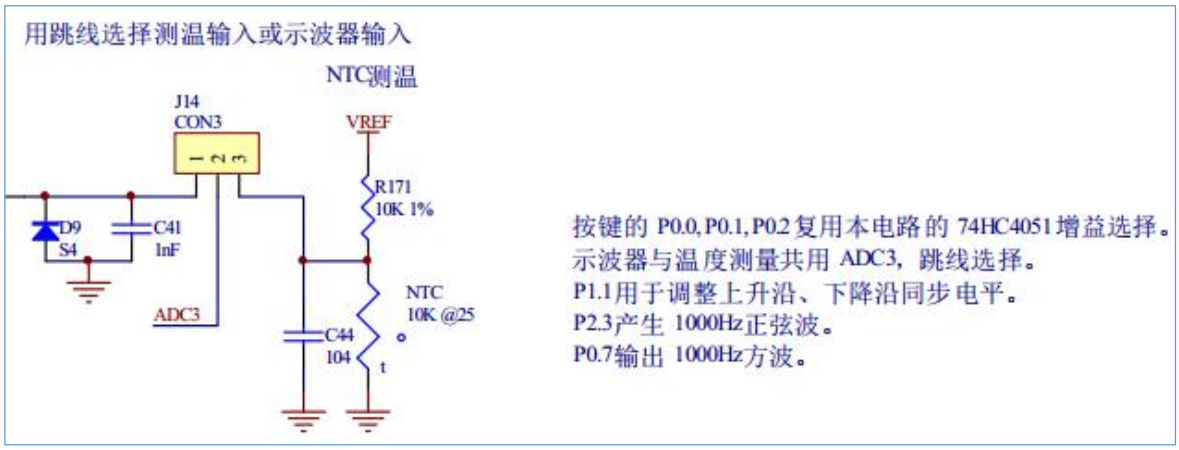
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以**方便分工合作、代码复用、模块化管理、增强可读性和可维护性**。

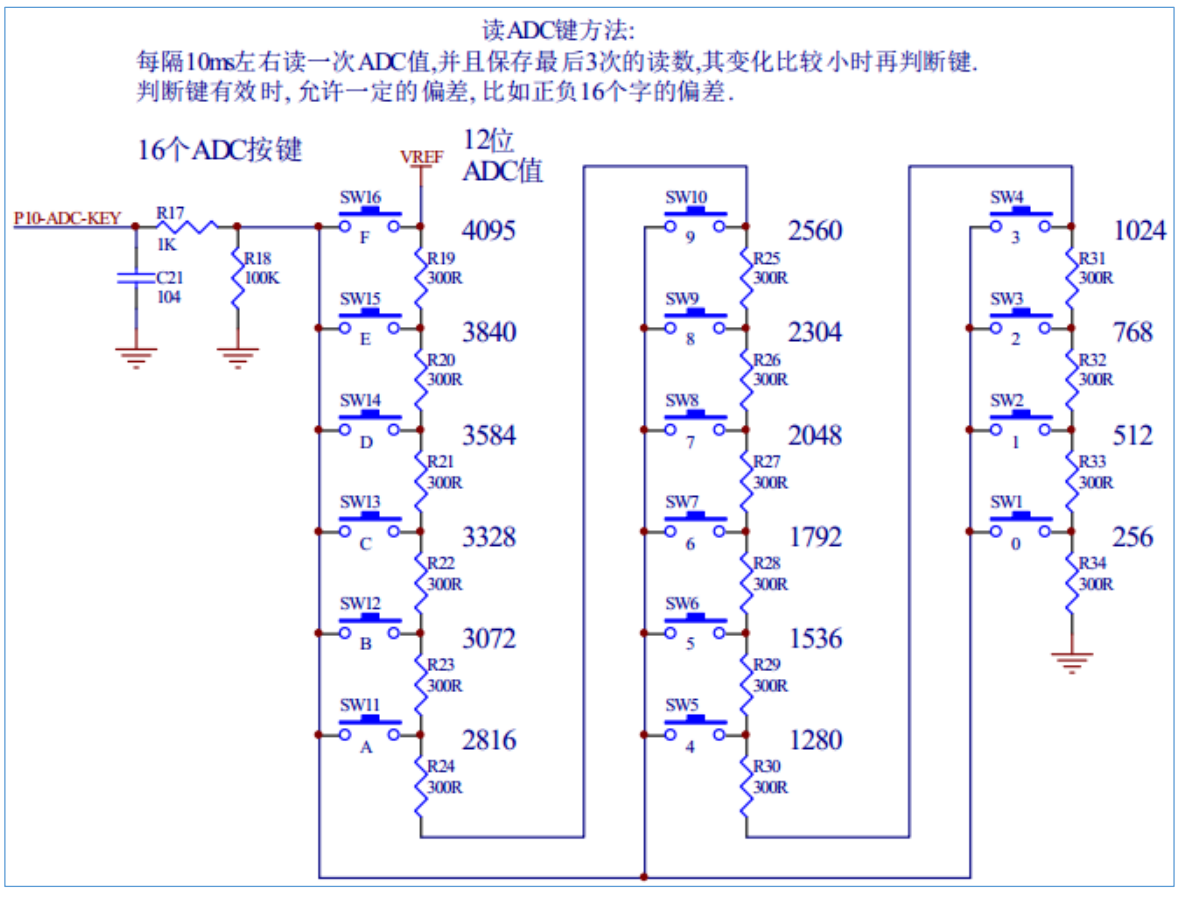
- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

```
#ifndef XXXX
#define XXXX
...
#endif
```

2.18.2 原理图


SYS-VCC
串口2通讯指示灯
LED10
RED
LED9
RED
R55
3.3K
R54
3.3K
板上双串口TTL电平通讯
不做双串口实验时要断开 J7、J8,
否则两对IO之间很可能互相干扰
P46-RXD2-2
R72
33R
J7
P5.1-TxD3
P47-TXD2-2
J8
P5.0-RxD3
R73 33R


用跳线选择测温输入或示波器输入
NTC测温
J14
CON3
VREF
R171
10K 1%
D9
S4
C41
1nF
ADC3
NTC
10K @25
C44
104
按键的 P0.0, P0.1, P0.2 复用本电路的 74HC4051 增益选择。
示波器与温度测量共用 ADC3，跳线选择。
P1.1用于调整上升沿、下降沿同步电平。
P2.3产生 1000Hz正弦波。
P0.7输出 1000Hz方波。


读ADC键方法:
每隔10ms左右读一次ADC值,并且保存最后3次的读数,其变化比较小时再判断键.
判断键有效时, 允许一定的偏差, 比如正负16个字的偏差.
16个ADC按键
12位
ADC值
VREF
P10-ADC-KEY
R17
1K
R18
100K
C21
104
SW16
F
4095
R19
300R
SW15
E
3840
R20
300R
SW14
D
3584
R21
300R
SW13
C
3328
R22
300R
SW12
B
3072
R23
300R
SW11
A
2816
R24
300R
SW10
9
2560
R25
300R
SW9
8
2304
R26
300R
SW8
7
2048
R27
300R
SW7
6
1792
R28
300R
SW6
5
1536
R29
300R
SW5
4
1280
R30
300R
SW4
3
1024
R31
300R
SW3
2
768
R32
300R
SW2
1
512
R33
300R
SW1
0
256
R34
300R

2.18.3 实验程序代码

2.18.3.1 main.c -- 程序主函数

```
//main.c
//程序主函数
//短接 NTC 跳线
//串口 2/UART2 通过杜邦线与双串口工具相连
//AiCube-ISP 软件"CDC/HID-串口助手"页面，串口选择 CDC2 对应的 COM 口(先不打开串口)
//AiCube-ISP 软件"调试仿真接口" → "接口设置" 选择将所有调试接口绑定到 USB-CDC/串口助手
//AiCube-ISP 软件"调试仿真接口" → "串口绘图" 先点"设置"按钮，数据格式选择"双字节(高位在前)"
//点击"串口绘图"界面的"打开端口"按钮，开始进行串口绘图

#include "config.h"       //头文件中已包含 stc8h.h 以及其他头文件

void main()
{
    u16 Res;

    SYS_Init();          //系统初始化

    while (1)
    {
        if (f100ms)          //判断 100ms 标志位
        {
            f100ms = 0;    //清除 100ms 标志，并处理 100ms 事件

            Res = Get_ADC12bitResult(0);    //检测 ADC 按键值
            Res = Get_ADC12bitResult(0);    //检测 ADC 按键值
            Res = Get_ADC12bitResult(0);    //检测 ADC 按键值
            Uart2_Send((u8)(Res>>8));       //发送采样结果高 8 位数据
            Uart2_Send((u8)Res);            //发送采样结果低 8 位数据

            Res = Get_ADC12bitResult(3);    //采集 NTC 脚位 AD 值
            Res = Get_ADC12bitResult(3);    //采集 NTC 脚位 AD 值
            Res = Get_ADC12bitResult(3);    //采集 NTC 脚位 AD 值
            Uart2_Send((u8)(Res>>8));       //发送采样结果高 8 位数据
            Uart2_Send((u8)Res);            //发送采样结果低 8 位数据
        }
    }
}

void SYS_Init(void)      //系统初始化函数
{
    P_SW2 = 0x80;      //使能访问扩展 XFR
```

```
    P0M0 = 0x00; P0M1 = 0x00;     //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x09;     //初始化 P1 口为准双向口模式, P1.0,P1.3 高阻输入
    P2M0 = 0x00; P2M1 = 0x00;     //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;     //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;     //初始化 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;     //初始化 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;     //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;     //初始化 P7 口为准双向口模式

    P_SW2 |= UART2_S2;            //UART2 通道选择: 0: P1.0 P1.1,   1: P4.6 P4.7

    Timer0_Init();      //定时器 0 初始化,用于产生 1ms 的时间基准
    Uart2_Init();       //UART 接口初始化
    ADC_Init();         //ADC 初始化

    EA = 1;             //总中断允许位打开
}
```

2.18.3.2 uart.c -- 初始化 UART 接口

```
//uart.c
//初始化 UART 接口

#include "config.h"       //头文件中已包含 stc8h.h 以及其他头文件

void Uart2_Init(void)     //115200bps@11.0592MHz
{
    S2CON = 0x50;     //8 位数据,可变波特率
    AUXR |= 0x04;     //定时器时钟 1T 模式
    T2L = 0xE8;       //设置定时初始值
    T2H = 0xFF;       //设置定时初始值
    AUXR |= 0x10;     //定时器 2 开始计时
}

void Uart2_Send(u8 dat)           //通过串口 2 发送数据
{
    S2BUF = dat;                  //写入需要发送的数据
    while(!(S2CON & S2TI));       //判断发送是否完成
    S2CON &= ~S2TI;               //清除发送完成标志
}
```

2.18.3.3 adc.c -- ADC 初始化和 ADC 数据采集

```
//adc.c
//ADC 程序
//包括 ADC 初始化和 ADC 数据采集
```

```
#include "config.h"        //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

void ADC_Init()            //ADC 初始化函数
{
    ADCTIM = 0x3f;              //设置 ADC 内部时序，ADC 采样时间建议设最大值
    ADCCFG = 0x2f;              //设置 ADC 时钟为系统时钟/2/16
    ADC_CONTR = 0x80;           //使能 ADC 模块
}

u16 Get_ADC12bitResult(u8 channel)        //channel = 0~15
{
    ADC_RES = 0;
    ADC_RESL = 0;

    ADC_CONTR = (ADC_CONTR & 0xf0) | channel;   //设置 ADC 转换通道
    ADC_CONTR |= ADC_START;         //启动 ADC 转换
    _nop_();
    _nop_();
    _nop_();
    _nop_();

    while((ADC_CONTR & ADC_FLAG) == 0);     //wait for ADC finish
    ADC_CONTR &= ~ADC_FLAG;         //清除 ADC 结束标志

    return    (((u16)ADC_RES << 8) | ADC_RESL);
}
```

2.18.3.4 timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:100ms 定时,产生 100ms 的标志位

#include "config.h"        //头文件中已包含 stc8h.h 以及其他头文件

bit f100ms;                //100ms 标志位
int cnt100;

void Timer0_Init(void)          //1 毫秒@11.0592MHz
{
    AUXR |= 0x80;           //定时器时钟 1T 模式
    TMOD &= 0xF0;           //设置定时器模式
    TL0 = 0xCD;             //设置定时初始值
    TH0 = 0xD4;             //设置定时初始值
    TF0 = 0;                //清除 TF0 标志
    TR0 = 1;                //定时器 0 开始计时
```

```
    ET0 = 1;                    //使能定时器 0 中断

    f100ms = 0;                 //初始化 100ms 标志位
    cnt100 = 0;
}

void Timer0_Isr(void) interrupt 1       //定时器 0 中断服务程序
{
    if (cnt100++ >= 100)
    {
        cnt100 = 0;
        f100ms = 1;             //设置 100ms 标志位
    }
}
```

2.18.3.5 config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__            //防止头文件被重复包含
#define __CONFIG_H__

#define HIRC            11059200UL
#define FOSC            11059200UL
#define SYSCLK          FOSC
#define MAIN_Fosc       FOSC

#include < stc8h.h>             //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"                //包含项目头文件
#include "uart.h"
#include "adc.h"
#include "timer.h"

void SYS_Init(void);                //函数声明

#endif
```

2.18.3.6 uart.h -- 项目配置文件的头文件

```
#ifndef __UART_H__          //防止头文件被重复包含
#define __UART_H__

void    Uart2_Init();               //函数声明
void    Uart2_Send(u8 dat);     //函数声明
```

```
#endif
```

2.18.3.7 adc.h -- 项目配置文件的头文件

```
#ifndef __ADC_H__          //防止头文件被重复包含
#define __ADC_H__

void ADC_Init();           //函数声明
u16 Get_ADC12bitResult(u8 channel);      //函数声明

#endif
```

2.18.3.8 timer.h -- 项目配置文件的头文件

```
#ifndef __TIMER_H__        //防止头文件被重复包含
#define __TIMER_H__

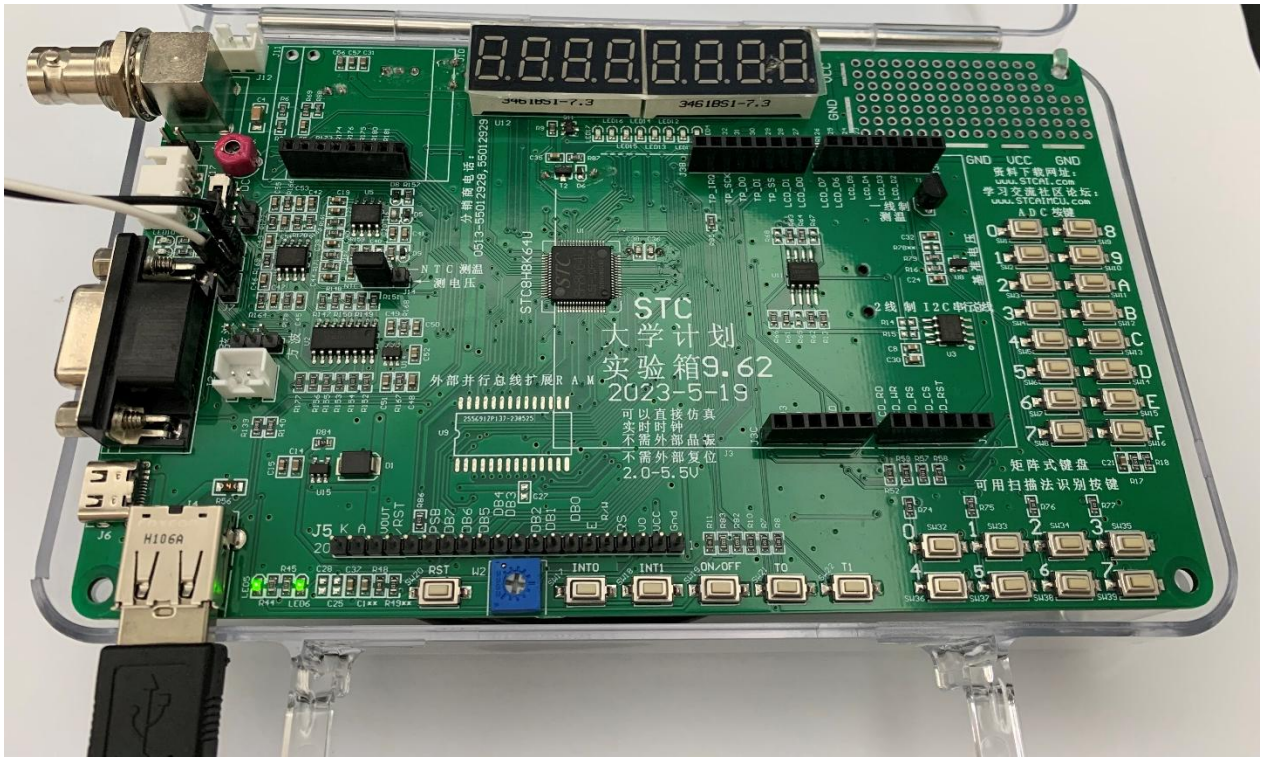
void Timer0_Init(void);    //函数声明

extern bit f100ms;         //外部变量声明

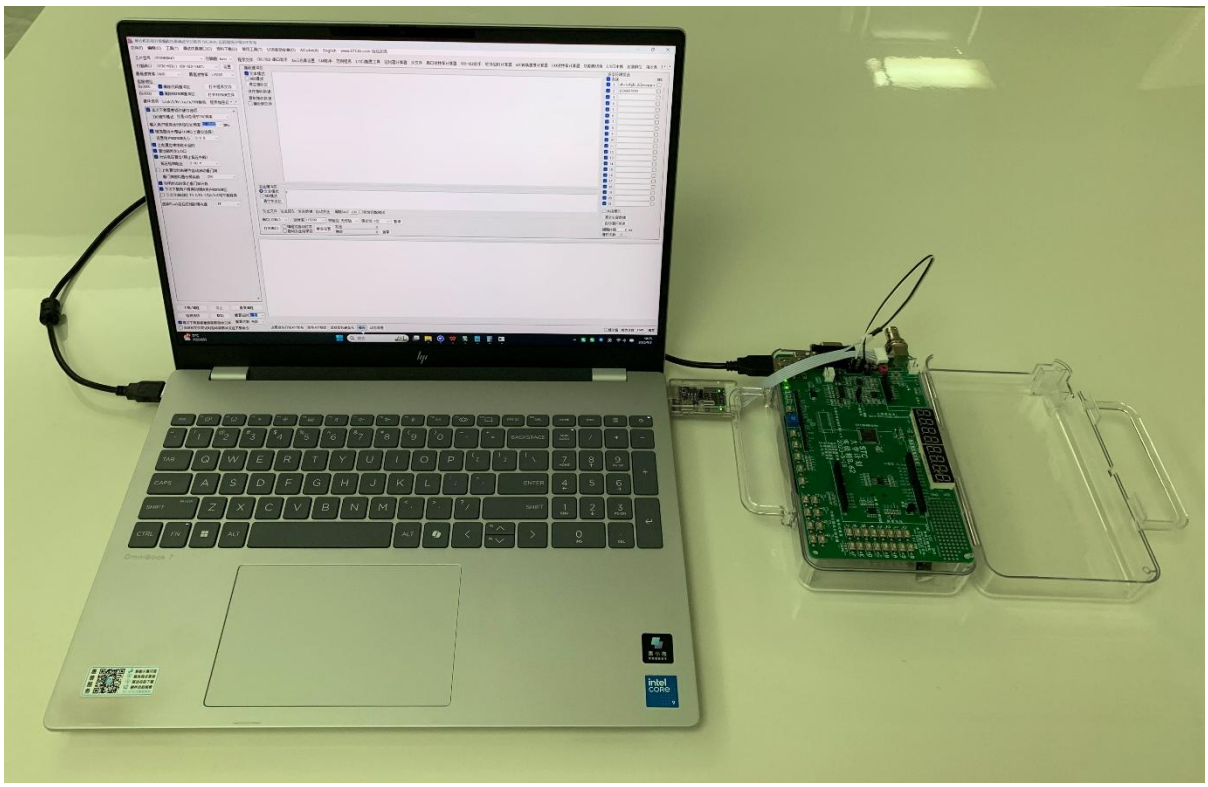
#endif
```

2.18.4 程序下载，串口绘图相关设置

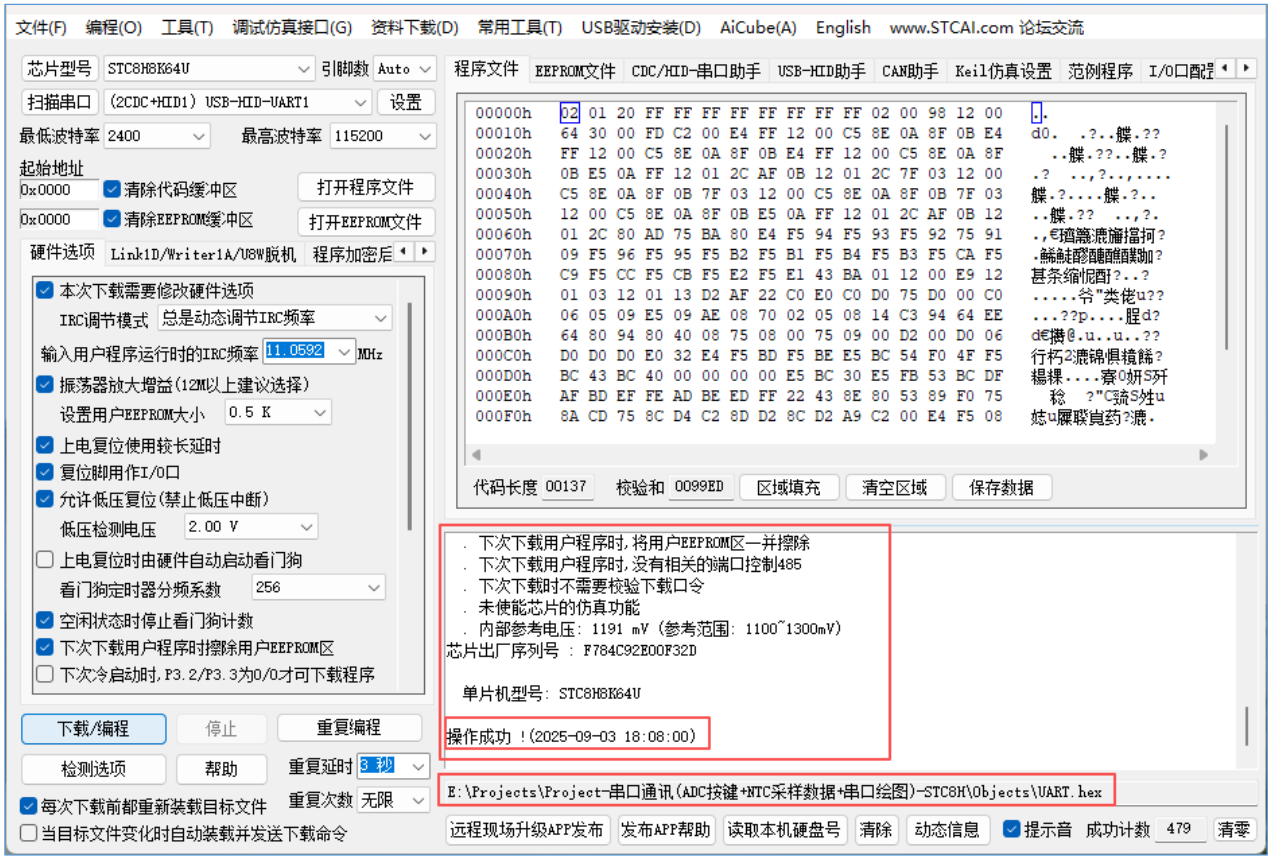
录入代码，保存，编译。按实验要求连接并设置好 STC8H8K64U 实验箱，如下图：



STC8H8K64U 实验箱与电脑连接实物图：

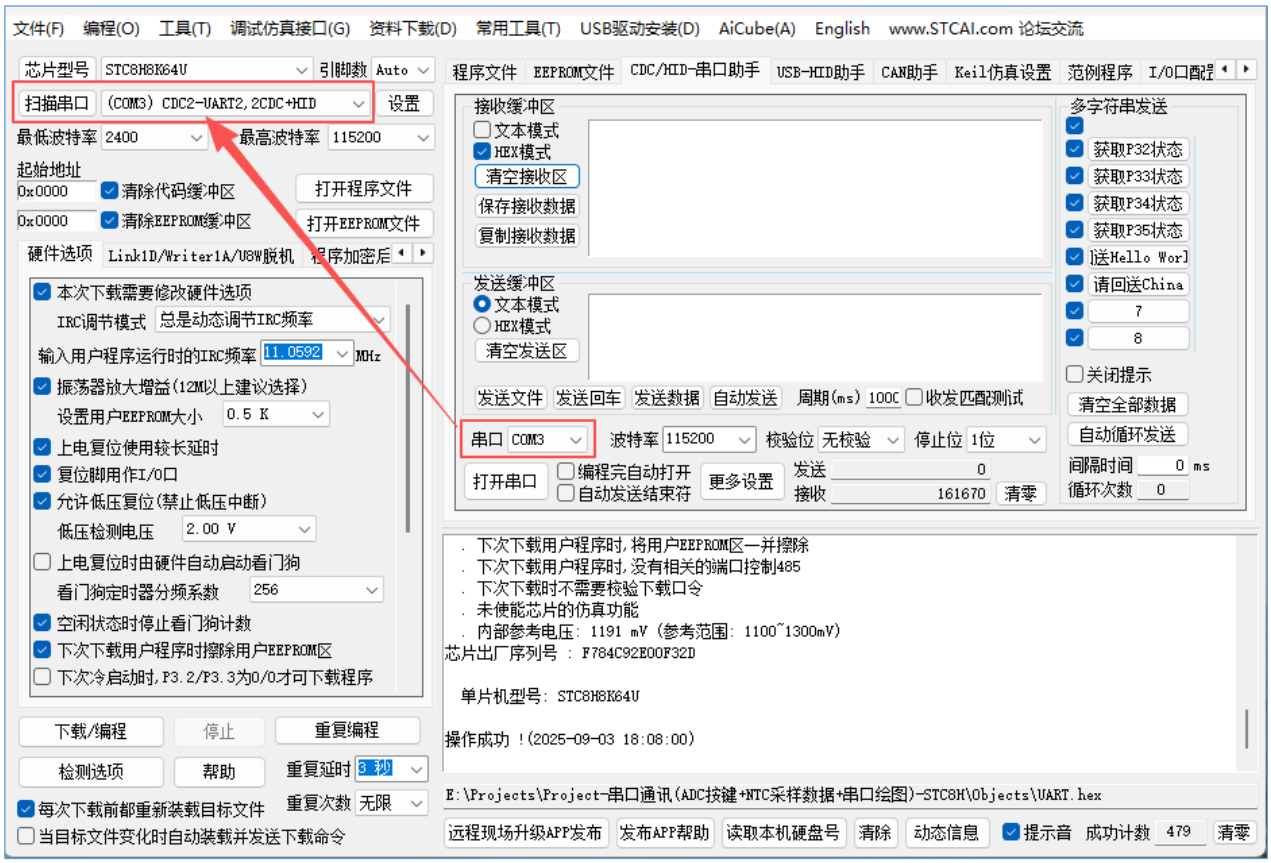


运行 AiCube-ISP 软件系统，按步骤打开并下载程序“uart.hex”，如下图：



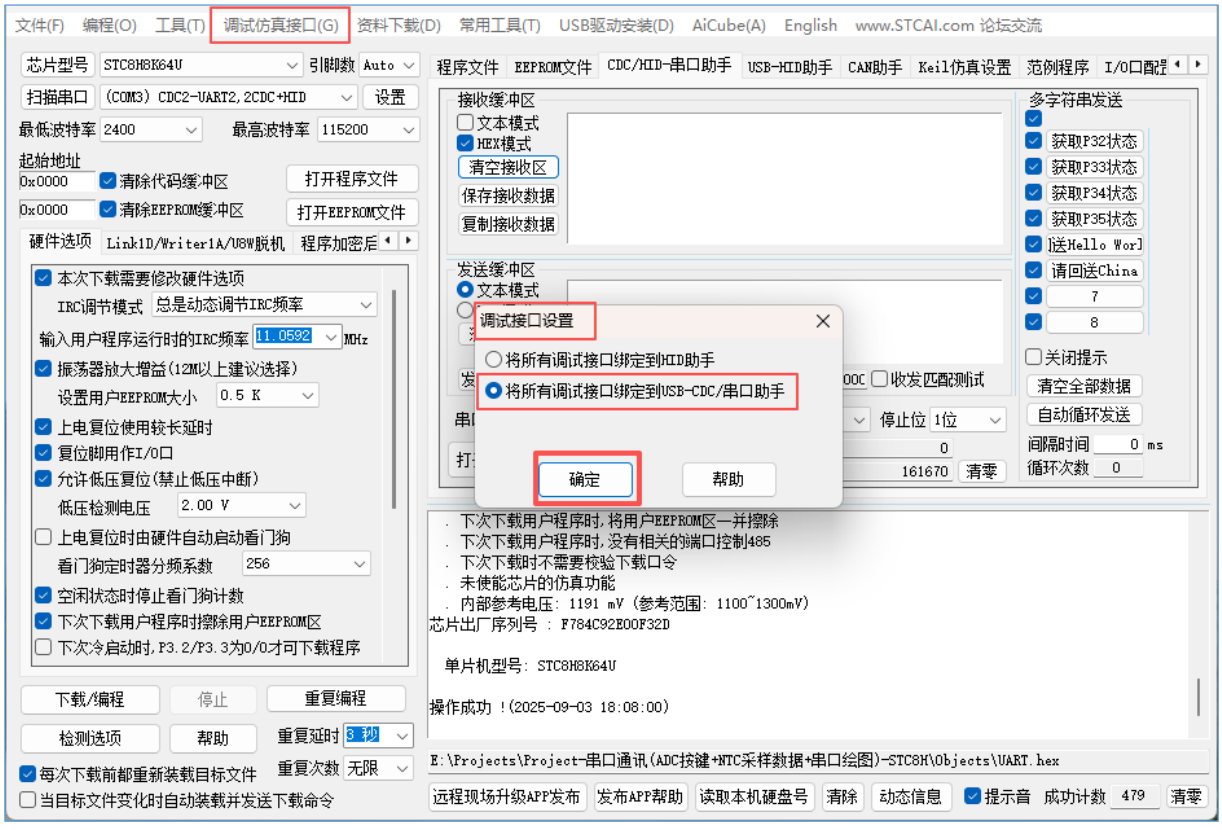
➲ 在 AiCube-ISP 软件系统界面，点击“CDC/HID-串口助手”选项卡：

➢ 串口选择 CDC2 对应的 COM 口，如下图：



➲ 在 AiCube-ISP 软件系统的菜单栏，点击“调试仿真接口”→点击“接口设置”：

➢ 选择“将所有调试接口绑定到 USB-CDC/串口助手”，如下图：

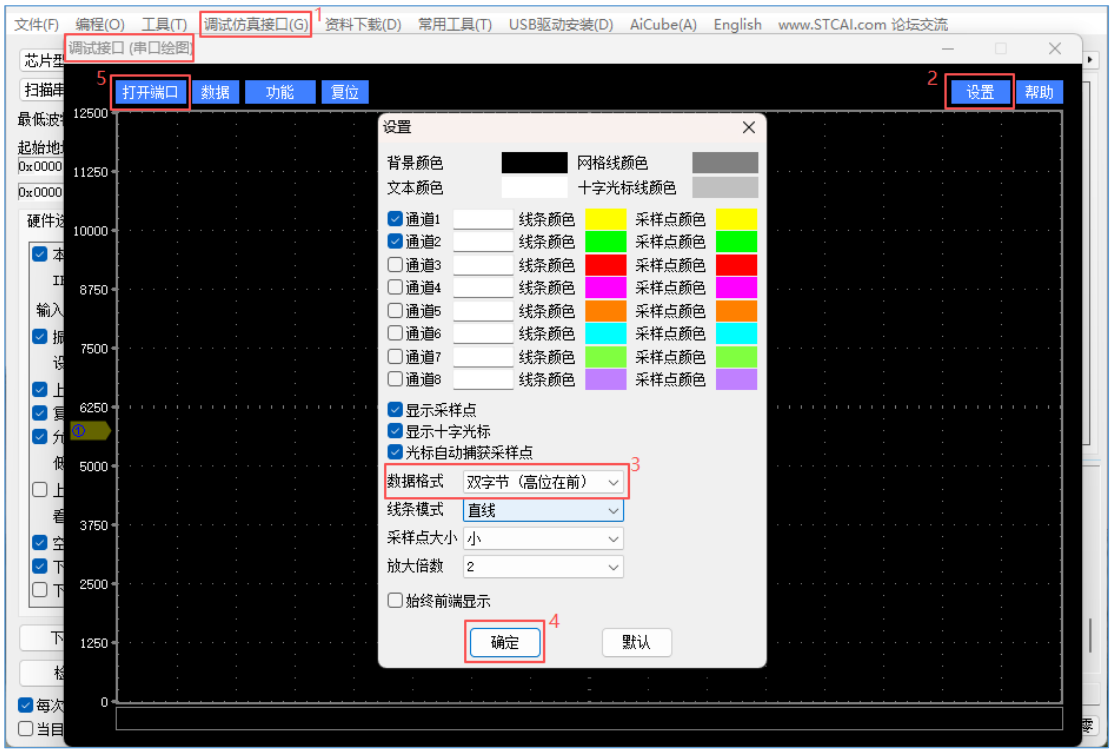


➲ 在 AiCube-ISP 软件系统的菜单栏，点击“调试仿真接口”→点击“串口绘图”：

➢ 在“串口绘图”浮窗，点击“设置”按钮；

➢ 在“设置”窗口，数据格式：选择“双字节(高位在前)”。

➢ 找到“打开端口”按钮，如下图：

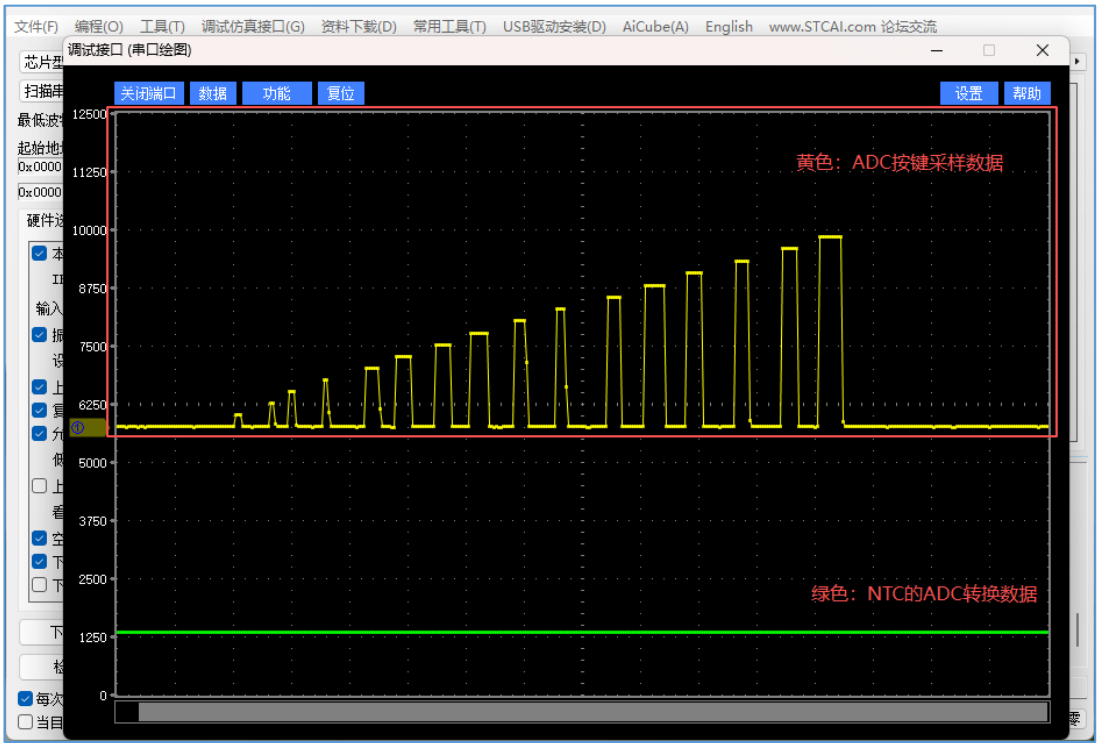


一切准备就绪！

2.18.5　观察实验现象，验证实验效果

在“串口绘图”界面，点击“打开端口”按钮，开始进行串口绘图。

随意按 STC8H8K64U 实验箱上的 ADC 按键“0～F”，串口绘图曲线如下图：



完成实验。

2.19　大型综合实战 -- 模拟 RTC 时钟+数码管+ADC 检测按键(查询)

2.19.1　实验介绍

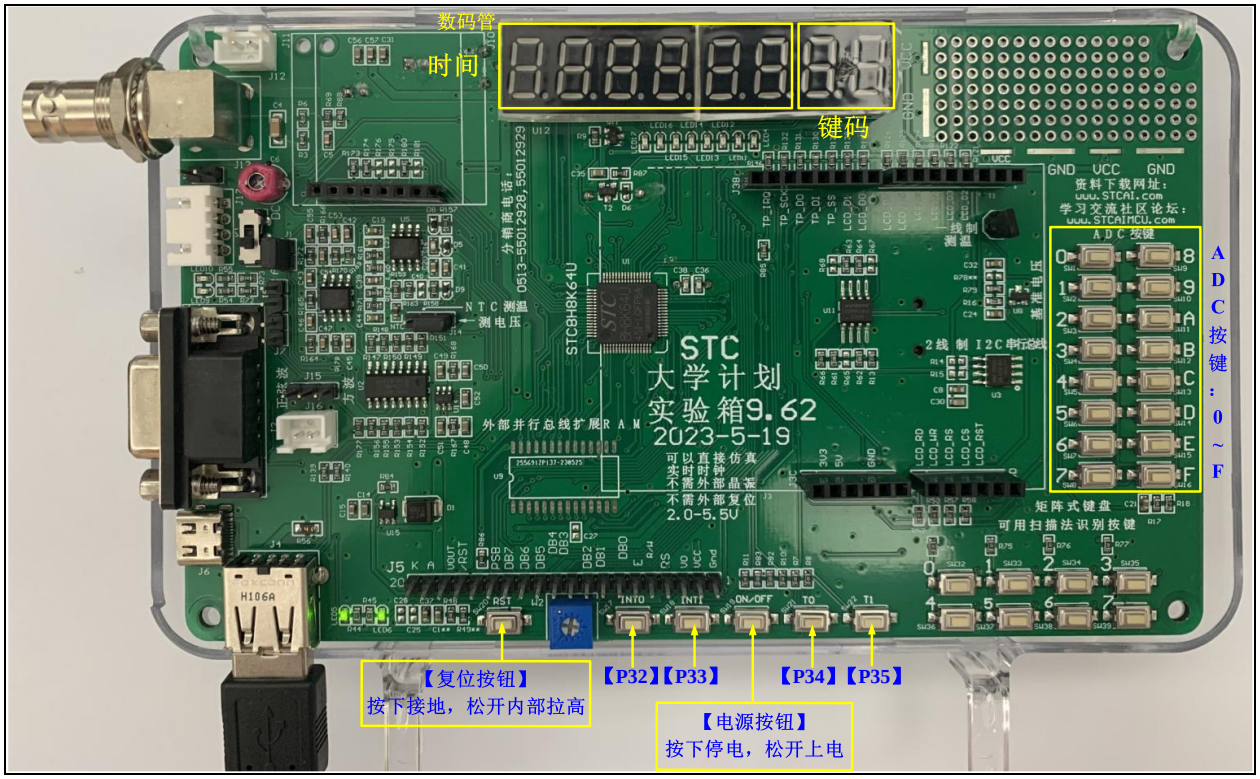
1、熟悉 ADC 按键扫描方法
2、了解使用 P6、P7 口控制 LED 数码管
3、熟悉软件模拟 RTC，实现 RTC 的时、分、秒控制和设置
4、扩展部分：如何实现长按键
5、熟悉 STC8H8K64U 实验箱布局及原理图，熟练使用 AiCube-ISP 系统软件
6、熟悉如何管理多文件项目

2.19.1.1　ADC 键盘计算键码说明

本 ADC 键盘方案在很多实际产品设计中，验证了其稳定可靠，即使按键使用导电膜，都很可靠。
12 位 ADC 的转换结果范围位 0～4095，16 个键直接使用相同的电阻进行分压，
理论上各个键对应的 ADC 值为 (4096 / 16) * k = 256 * k，k = 1～16。
特别的，k=16 时，对应的 ADC 值是 4095。
但是实际会有偏差，则判断时限制这个偏差，ADC_OFFSET 为+/-偏差，
则 ADC 值在 (256 * k - ADC_OFFSET) 与 (256 * k + ADC_OFFSET)之间为键有效。
间隔一定的时间，就采样一次 ADC，比如 10ms。
为了避免偶然的 ADC 值误判，或者避免 ADC 在上升或下降时误判，使用连续 3 次 ADC 值均在偏差范围内时，ADC 值才认为有效。
以上算法，能保证读键非常可靠。

2.19.1.2　ADC 按键实验现象和操作说明

将本项目的 HEX 文件下载到 STC8H8K64U 的实验箱后
1、在 LED 数码管的高 6 位会动态显示当前的 RTC 时间，低 2 位会显示按下的 ADC 按键的键码



2、按下 16 个 ADC 按键之一后：

- 在 LED 数码管的低 2 位会显示当前的按键键码

3、ADC 按键中的按键 F，是 RTC 模式设置键

- 上电后 RTC 为正常工作模式，此时数码管动态显示 RTC 时间
- 按一次按键 F，RTC 会切换为小时设置模式，此时 RTC 的小时值会 0.5 秒闪烁一次
- 再按一次按键 F，RTC 会切换为分钟设置模式，此时 RTC 的分钟值会 0.5 秒闪烁一次
- 再按一次按键 F，RTC 会切换为秒钟设置模式，此时 RTC 的秒钟值会 0.5 秒闪烁一次
- 再按一次按键 F，恢复为 RTC 正常工作模式

4、ADC 按键中的按键 D 和 E 分别是上调键和下调键

- 只有在 RTC 的时/分/秒设置模式时，上调键和下调键才有效

5、时钟设置模式时，长按按键 D 或 E，可实现快速设置功能

2.19.1.3 按键扫描说明

如按键要判断为正常按下，而不是抖动，需要该按键保持按下持续时间是 20ms ~ 50ms；
主循环中有个 10ms 的时间基准，将按键扫描程序作为 10ms 事件，按键的值检测 3 次都在有效范围内，即 30ms 去抖动处理，得到稳定的键码。

2.19.1.4 Keil 环境下多文件项目管理说明

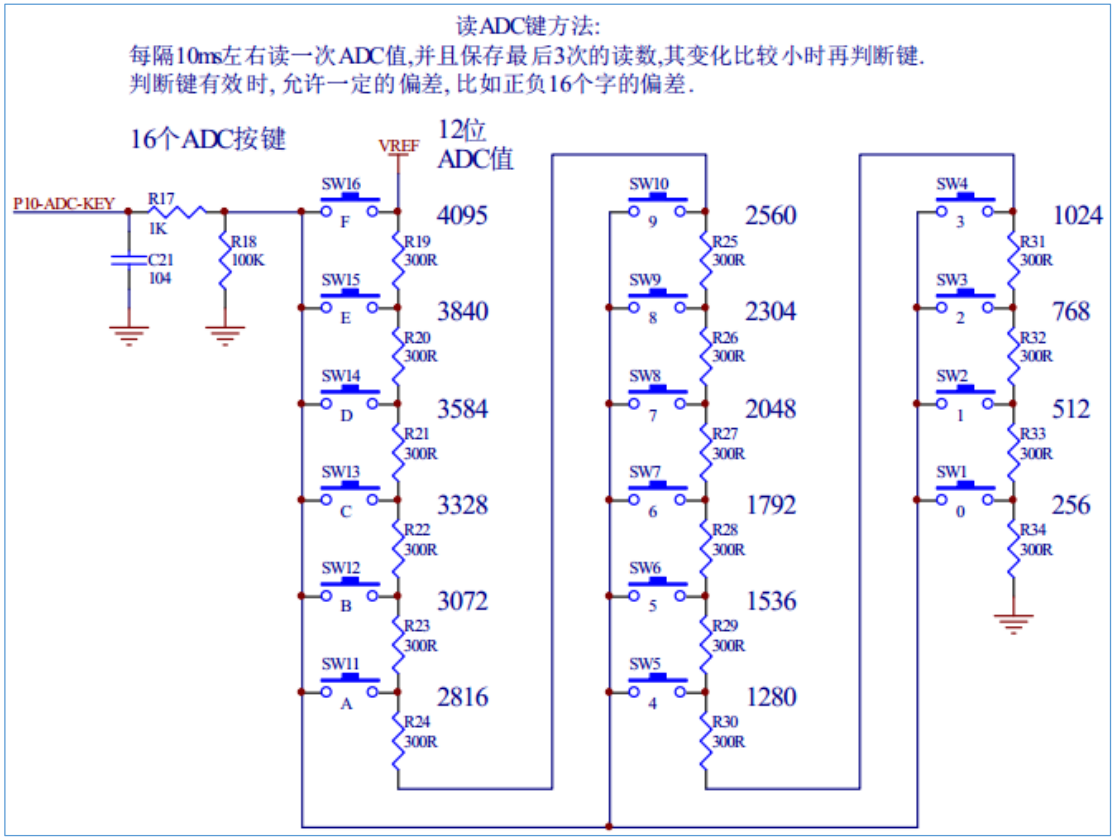
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以**方便分工合作、代码复用、模块化管理、增强可读性和可维护性**。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

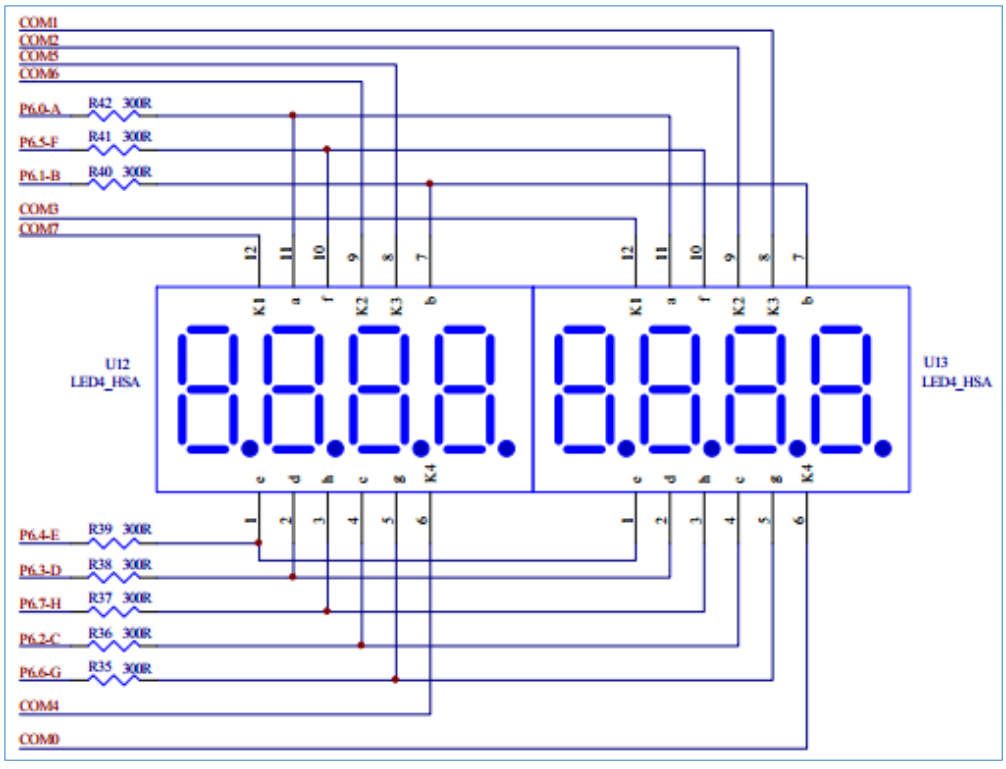
```
#ifndef XXXX
#define XXXX
...
#endif
```

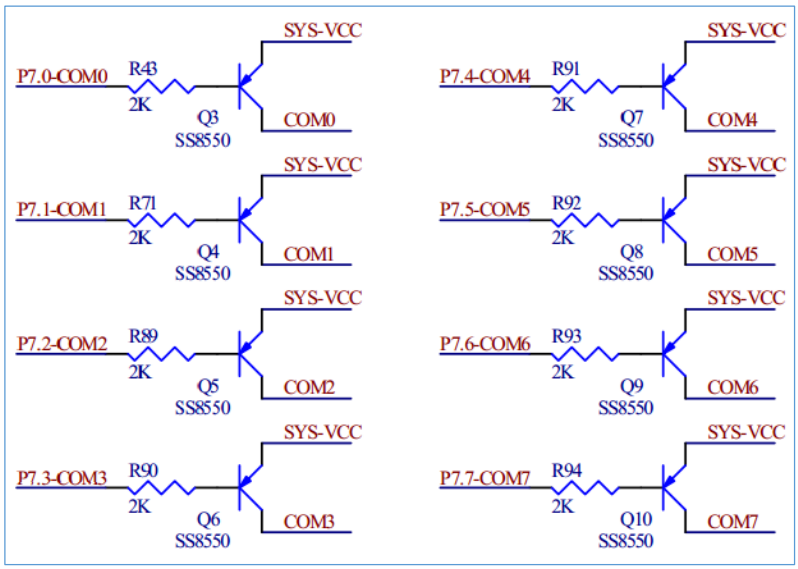
2.19.2 原理图

2.19.2.1 ADC 按键原理图



2.19.2.2 P6、P7 口控制数码管原理图





2.19.3 实验程序代码

2.19.3.1 main.c -- 程序主函数

```
//main.c
//程序主函数

#include "config.h"              //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

void main()
{
    u16 Res;

    SYS_Init();                  //系统初始化

    while (1)
    {
        if (f1ms)                //判断 1ms 标志位
        {
            f1ms = 0;            //清除 1ms 标志，并处理 1ms 事件
            ScanLed();           //LED 数码管扫描
        }

        if (f10ms)               //判断 10ms 标志位
        {
            f10ms = 0;           //清除 10ms 标志，并处理 10ms 事件
            Res = Get_ADC12bitResult(0);    //检测 ADC 按键
            if(Res < 4096)       //判断采样数值是否有效
            {
                CalculateAdcKey(Res);       //判断按键
            }
```

```
            KeyEvents();        //按键处理事件
        }

        if (f500ms)             //判断 500ms 标志位
        {
            f500ms = 0;         //清除 500ms 标志，并处理 500ms 事件
            RTC_Update();       //更新软件模拟的 RTC 时间
            RTC_Show();         //在 LED 数码管上显示 RTC 时间
        }
    }
}

void SYS_Init(void) //系统初始化函数
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;       //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x01;       //初始化 P1 口为准双向口模式, P1.0 高阻输入
    P2M0 = 0x00; P2M1 = 0x00;       //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //初始化 P4 口为准双向口模式
    P5M0 = 0x01; P5M1 = 0x00;       //设置 P5.0 口为强推挽, 其他设置为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //初始化 P7 口为准双向口模式

    Led_Init();         //LED 数码管初始化
    Timer0_Init();      //定时器 0 初始化,用于产生 1ms 和 500ms 的时间基准
    RTC_Init();         //软件模拟 RTC 时间初始化
    ADC_Init();         //ADC 初始化

    EA = 1;             //总中断允许位打开
}
```

2.19.3.2 key.c -- 按键扫描以及按键事件处理

```
//key.c
//按键扫描以及按键事件处理

#include "config.h"             //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

u8      ADC_KeyState=0;
u8      ADC_KeyState1=0;
u8      ADC_KeyState2=0;
u8      ADC_KeyState3=0;    //键状态
u16     ADC_KeyHoldCnt=0;   //键按下计时
u8      KeyCode=0;          //给用户使用的键码, 1~16 有效
```

```
/***************** ADC 键盘计算键码 *****************************
本 ADC 键盘方案在很多实际产品设计中, 验证了其稳定可靠, 即使按键使用导电膜,都很可靠.
16 个键,理论上各个键对应的 ADC 值为 (4096 / 16) * k = 256 * k, k = 1 ~ 16, 特别的, k=16 时,对应的 ADC 值是 4095.
但是实际会有偏差,则判断时限制这个偏差, ADC_OFFSET 为+/-偏差, 则 ADC 值在 (256*k-ADC_OFFSET) 与 (256*k+ADC_OFFSET)之间为键有效.
间隔一定的时间,就采样一次 ADC,比如 10ms.
为了避免偶然的 ADC 值误判, 或者避免 ADC 在上升或下降时误判, 使用连续 3 次 ADC 值均在偏差范围内时, ADC 值才认为有效.
以上算法, 能保证读键非常可靠.
*********************************************/
#define   ADC_OFFSET     64
void CalculateAdcKey(u16 adc)
{
    u8 i;
    u16 j;

    if(adc < (256-ADC_OFFSET))      //判断按键是否处于释放状态
    {
        ADC_KeyState = 0;           //键状态归 0
        ADC_KeyHoldCnt = 0;
    }

    j = 256;                        //设置初始值，KEY0 理论 ADC 值为 256
    for(i=1; i<=16; i++)            //判断当前 ADC 值是否在 16 个按键有效范围之内
    {
        if((adc >= (j - ADC_OFFSET)) && (adc <= (j + ADC_OFFSET)))   break;
                                    //判断是否在偏差范围内，是的话退出循环
        j += 256;                   //否则判断是否在下一个按键有效范围
    }

    ADC_KeyState3 = ADC_KeyState2;          //保存上次状态
    ADC_KeyState2 = ADC_KeyState1;          //保存当前状态

    if(i > 16)   ADC_KeyState1 = 0;         //键无效
    else                                    //键有效
    {
        ADC_KeyState1 = i;                  //保存当前键码
        if((ADC_KeyState3 == ADC_KeyState2) && (ADC_KeyState2 == ADC_KeyState1) &&
           (ADC_KeyState3 > 0) && (ADC_KeyState2 > 0) && (ADC_KeyState1 > 0))
                                            //三次状态相同，并且键码非 0，才认为按键有效
        {
            if(ADC_KeyState == 0)           //第一次检测到
            {
                KeyCode   = i;              //保存键码
                ADC_KeyState = i;           //保存键状态
```

```
                ADC_KeyHoldCnt = 0;
            }

            if(ADC_KeyState == i)           //连续检测到同一键按着
            {
                if(++ADC_KeyHoldCnt >= 100)     //按下 1 秒后,以 10 次每秒的速度 Repeat Key
                {
                    ADC_KeyHoldCnt = 90;
                                    //重设长按计数器,后续每 10*10ms=100ms 设置一次有效键码
                    KeyCode   = i;          //保存键码
                }
            }
            else ADC_KeyHoldCnt = 0;        //按下时间计数归 0
        }
    }
}

void KeyEvents()        //按键事件处理程序 (需每 1ms 执行一次!!!)
{
    if(KeyCode > 0)     //有键按下
    {
        SetLed(0, (BYTE)(KeyCode / 10), 0);     //在数码 0 上显示键码的十位
        SetLed(1, (BYTE)(KeyCode % 10), 0);     //在数码 1 上显示键码的个位

        switch (KeyCode)                        //处理不同的按键键码
        {
            case 16:
                RTC_SetMode();                  //按下按键 F,设置 RTC 工作模式
                break;
            case 14:
                RTC_SetUp();                    //按下按键 E,向上调整 RTC 时间
                break;
            case 15:
                RTC_SetDown();                  //按下按键 D,向上调整 RTC 时间
                break;
        }

        KeyCode = 0;        //清除键码，防止重复处理
    }
}
```

2.19.3.3 led.c -- 扫描 LED 数码管

```
//led.c
//使用 P6 和 P7 口动态扫描 LED 数码管
//由于 LED 数码管是由 8 个 SEG 和 8 个 COM 组成的矩阵 LED
```

```
//所以 LED 数码管也需要扫描显示

#include "config.h"             //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

#define   LED_OFF     0x00
//定义关闭 LED 数码管的段码数据,LED 数码管为共阴极,所以 SEG 线全部输出 0 即可关闭一个 8 段数码管
#define   LED_POINT  0x80      //定义 LED 数码管小数点的段码数据

BYTE    ShowIndex;              //LED 数码管扫描显示索引(8 个数码管轮流扫描显示)
BYTE    LedBuf[8];              //8 个 LED 数码管显示数据缓冲区

BYTE    code LED_PATTERN[10] =      //定义 LED 数码管显示数据 0~9 的段码数据
{
    0x3F,     //0                         //此表为共阴极数码管的段码数据
    0x06,     //1
    0x5B,     //2
    0x4F,     //3
    0x66,     //4
    0x6D,     //5
    0x7D,     //6
    0x07,     //7
    0x7F,     //8
    0x6F,     //9
};

void Led_Init(void)                 //LED 数码管初始化程序
{
    LedBuf[0] = LED_OFF;        //初始化 LED 数码管显示数据缓冲区,上电关闭显示
    LedBuf[1] = LED_OFF;
    LedBuf[2] = LED_OFF;
    LedBuf[3] = LED_OFF;
    LedBuf[4] = LED_OFF;
    LedBuf[5] = LED_OFF;
    LedBuf[6] = LED_OFF;
    LedBuf[7] = LED_OFF;

    ShowIndex = 0;              //初始化扫描显示索引
}

void ScanLed(void)                  //LED 数码管扫描显示程序,  (需每 1ms 执行一次!!!)
{
    P6 = (~LedBuf[ShowIndex]);      //将当前扫描到的 LED 数码管的段码移入到 P6 口
    P7 = (~(1 << ShowIndex));       //将当前扫描到的 LED 数码管的 COM 数据移入到 P7 口

    if (++ShowIndex >= 8)       //扫描索引值递增
        ShowIndex = 0;          //扫描完一轮后索引值归零
```

```
}

void SetLed (BYTE nIndex,       //设置 LED 显示函数,nIndex:设置目标数码管,0~7 分别对应 8 个数码
             BYTE nNum,         //nNum:设置显示的数据,有效值 0~9,其他值则关闭显示
             BOOL bShowPoint    //bShowPoint:设置是否显示小数点
            )
{
    if (nIndex < 8)             //目标数码管的有效值为 0~7
    {
        if (nNum < 10)          //判断显示的值是否为 0~9
            LedBuf[nIndex] = LED_PATTERN[nNum];
                                //将相应数据的段码赋值到 LED 数码管显示数据缓冲区
        else
            LedBuf[nIndex] = LED_OFF;       //如果是其他值则关闭显示

        if (bShowPoint)         //判断是否需要显示小数点
            LedBuf[nIndex] |= LED_POINT;        //如果需要显示,则增加小数点的段码数据
    }
}
```

2.19.3.4 timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:1ms 定时,产生 1ms 和 500ms 的标志位

#include "config.h"     //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

bit     f1ms;           //1ms 标志位
bit     f10ms;          //10ms 标志位
bit     f500ms;         //500ms 标志位
char    cnt10;          //10 次计数器,用于产生 10ms 的时间
int     cnt500;         //500 次计数器,用于产生 500ms 的时间

void Timer0_Init(void)      //1 毫秒@11.0592MHz
{
    AUXR |= 0x80;           //定时器 0 时钟 1T 模式
    TMOD &= 0xF0;           //设置定时器 0 为模式 0
    TL0 = 0xCD;             //设置定时器 0 初始值
    TH0 = 0xD4;             //设置定时器 0 初始值
    TF0 = 0;                //清除 TF0 标志
    TR0 = 1;                //定时器 0 开始计时
    ET0 = 1;                //使能定时器 0 中断

    cnt10 = 0;              //初始化 10 次计数器
    cnt500 = 0;             //初始化 500 次计数器
```

```
    f1ms = 0;                   //初始化 1m 标志位
    f10ms = 0;                  //初始化 10m 标志位
    f500ms = 0;                 //初始化 500ms 标志位
}

void Timer0_Isr(void) interrupt 1       //定时器 0 中断服务程序
{
    f1ms = 1;                   //设置 1ms 标志位

    if (cnt10++ >= 10)          //判断 10 次计数是否已满
    {
        cnt10 = 0;              //清除计数值,开始开始计数
        f10ms = 1;              //设置 10ms 标志位
    }

    if (cnt500++ >= 500)        //判断 500 次计数是否已满
    {
        cnt500 = 0;             //清除计数值,开始开始计数
        f500ms = 1;             //设置 500ms 标志位
    }
}
```

2.19.3.5 rtc.c -- 软件模拟 RTC 程序

```
//rtc.c
//软件模拟 RTC 程序
//包括 RTC 时间更新控制
//RTC 显示控制以及 RTC 时间调整控制

#include "config.h"       //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

bit     fToggle;          //0.5 秒->1 秒的控制位(两次计数器),用于在设置时间时进行闪烁控制
BYTE    bHour;            //RTC 小时计数值
BYTE    bMinute;          //RTC 分钟计数值
BYTE    bSecond;          //RTC 秒钟计数值

BYTE    bMode;            //RTC 工作模式控制
                          //0:正常工作模式
                          //1:小时设置模式
                          //2:分钟设置模式
                          //3:秒钟设置模式

void RTC_Init()           //RTC 初始化函数
{
    bHour = 12;           //初始化时间为 12:30:00
    bMinute = 30;
```

```
    bSecond = 0;
    bMode = 0;          //初始化 RTC 工作模式为正常工作模式
    fToggle = 0;        //初始化翻转控制
}

void RTC_Update()               //RTC 时间更新函数,  (需每 500ms 执行一次!!!)
{
    fToggle = !fToggle;         //翻转控制位

    if ((fToggle == 1) &&       //如果 1 秒时间到并且为正常工作模式,则更新 RTC 时间
      (bMode == 0))
    {
        if (++bSecond >= 60)            //秒钟递增 1,并判断秒钟是否记满 60
        {
            bSecond = 0;                //秒钟归零
            if (++bMinute >= 60)        //分钟递增 1,并判断分钟是否记满 60
            {
                bMinute = 0;            //分钟归零
                if (++bHour >= 24)      //小时递增 1,并判断小时是否记满 24
                {
                    bHour = 0;          //小时归零
                }
            }
        }
    }
}

void RTC_Show()                 //RTC 显示控制程序,不同的模式显示方式不一样
{
    switch (bMode)              //对 RTC 工作模式进行分支处理
    {
        case 0:                         //正常工作模式
            SetLed(7, (BYTE)(bHour / 10), 0);           //在数码 7 上显示小时的十位
            SetLed(6, (BYTE)(bHour % 10), 1);           //在数码 6 上显示小时的个位
            SetLed(5, (BYTE)(bMinute / 10), 0);         //在数码 5 上显示分钟的十位
            SetLed(4, (BYTE)(bMinute % 10), 1);         //在数码 4 上显示分钟的个位
            SetLed(3, (BYTE)(bSecond / 10), 0);         //在数码 3 上显示秒钟的十位
            SetLed(2, (BYTE)(bSecond % 10), 0);         //在数码 2 上显示秒钟的个位
            break;

        case 1:                         //设置 RTC 的小时模式
            if (fToggle)                //闪烁控制
            {
                SetLed(7, (BYTE)(bHour / 10), 0);       //在数码 7 上显示小时的十位
                SetLed(6, (BYTE)(bHour % 10), 1);       //在数码 6 上显示小时的个位
            }
```

```
            else
            {
                SetLed(7, -1, 0);                   //在数码 7 上关闭显示,产生闪烁效果
                SetLed(6, -1, 1);                   //在数码 6 上关闭显示
            }
            SetLed(5, (BYTE)(bMinute / 10), 0);     //在数码 5 上显示分钟的十位
            SetLed(4, (BYTE)(bMinute % 10), 1);     //在数码 4 上显示分钟的个位
            SetLed(3, (BYTE)(bSecond / 10), 0);     //在数码 3 上显示秒钟的十位
            SetLed(2, (BYTE)(bSecond % 10), 0);     //在数码 2 上显示秒钟的个位
            break;

        case 2:                                     //设置 RTC 的分钟模式
            SetLed(7, (BYTE)(bHour / 10), 0);       //在数码 7 上显示小时的十位
            SetLed(6, (BYTE)(bHour % 10), 1);       //在数码 6 上显示小时的个位
            if (fToggle)                            //闪烁控制
            {
                SetLed(5, (BYTE)(bMinute / 10), 0); //在数码 5 上显示分钟的十位
                SetLed(4, (BYTE)(bMinute % 10), 1); //在数码 4 上显示分钟的个位
            }
            else
            {
                SetLed(5, -1, 0);                   //在数码 5 上关闭显示,产生闪烁效果
                SetLed(4, -1, 1);                   //在数码 4 上关闭显示
            }
            SetLed(3, (BYTE)(bSecond / 10), 0);     //在数码 3 上显示秒钟的十位
            SetLed(2, (BYTE)(bSecond % 10), 0);     //在数码 2 上显示秒钟的个位
            break;

        case 3:                                     //设置 RTC 的秒钟模式
            SetLed(7, (BYTE)(bHour / 10), 0);       //在数码 7 上显示小时的十位
            SetLed(6, (BYTE)(bHour % 10), 1);       //在数码 6 上显示小时的个位
            SetLed(5, (BYTE)(bMinute / 10), 0);     //在数码 5 上显示分钟的十位
            SetLed(4, (BYTE)(bMinute % 10), 1);     //在数码 4 上显示分钟的个位
            if (fToggle)                            //闪烁控制
            {
                SetLed(3, (BYTE)(bSecond / 10), 0); //在数码 3 上显示秒钟的十位
                SetLed(2, (BYTE)(bSecond % 10), 0); //在数码 2 上显示秒钟的个位
            }
            else
            {
                SetLed(3, -1, 0);                   //在数码 3 上关闭显示,产生闪烁效果
                SetLed(2, -1, 0);                   //在数码 2 上关闭显示
            }
            break;
    }
}
```

```
void RTC_SetMode()              //RTC 工作模式控制
{
    if (++bMode > 3)            //RTC 工作模式转换(0:正常工作模式->1:小时设置模式->
                                //2:分钟设置模式->3:秒钟设置模式->0:正常工作模式->...)
    bMode = 0;                  //如果 RTC 模式大于 3,则 RTC 重归正常工作模式
}

void RTC_SetUp()                //RTC 时间上调控制
{
    if (bMode == 0)             //如果是正常工作模式,上调无意义
        return;                 //直接返回

    switch (bMode)              //根据工作模式调整时/分/秒
    {
        case 1:                     //小时设置模式
            if (++bHour >= 24)      //小时计数值向上递增
                bHour = 0;          //溢出后自动归零
            break;
        case 2:                     //分钟设置模式
            if (++bMinute >= 60)    //分钟计数值向上递增
                bMinute = 0;        //溢出后自动归零
            break;
        case 3:                     //秒钟设置模式
            if (++bSecond >= 60)    //秒钟计数值向上递增
                bSecond = 0;        //溢出后自动归零
            break;
    }

    fToggle = 1;                //调整完后立即强制显示,不用等待 500ms
    RTC_Show();                 //显示调整的时间
}

void RTC_SetDown()              //RTC 时间下调控制
{
    if (bMode == 0)             //如果是正常工作模式,下调无意义
    return;                     //直接返回

    switch (bMode)              //根据工作模式调整时/分/秒
    {
        case 1:                     //小时设置模式
            if (--bHour >= 24)      //小时计数值向下递减
                bHour = 23;         //溢出后自动设置上限值
            break;
        case 2:                     //分钟设置模式
            if (--bMinute >= 60)    //分钟计数值向下递减
```

```
                bMinute = 59;       //溢出后自动设置上限值
            break;
        case 3:                     //秒钟设置模式
            if (--bSecond >= 60)    //秒钟计数值向下递减
                bSecond = 59;       //溢出后自动设置上限值
            break;
    }

    fToggle = 1;                    //调整完后立即强制显示,不用等待 500ms
    RTC_Show();                     //显示调整的时间
}
```

2.19.3.6 adc.c -- ADC 初始化和 ADC 查询方式检测按键

```
//adc.c
//ADC 程序
//包括 ADC 初始化和 ADC 数据采集

#include "config.h"       //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

void ADC_Init()           //ADC 初始化函数
{
    ADCTIM = 0x3f;        //设置 ADC 内部时序，ADC 采样时间建议设最大值
    ADCCFG = 0x2f;        //设置 ADC 时钟为系统时钟/2/16
    ADC_CONTR = 0x80;     //使能 ADC 模块
}

u16 Get_ADC12bitResult(u8 channel)        //channel = 0~15
{
    ADC_RES = 0;
    ADC_RESL = 0;

    ADC_CONTR = (ADC_CONTR & 0xf0) | channel;          //设置 ADC 转换通道
    ADC_CONTR |= ADC_START;                 //启动 ADC 转换_nop_();
    _nop_();
    _nop_();
    _nop_();

    while((ADC_CONTR & ADC_FLAG) == 0);     //wait for ADC finish
        ADC_CONTR &= ~ADC_FLAG;             //清除 ADC 结束标志

    return    (((u16)ADC_RES << 8) | ADC_RESL);
}
```

2.19.3.7 config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__          //防止头文件被重复包含
#define __CONFIG_H__

#define  HIRC                 11059200UL
#define  FOSC                 11059200UL
#define  SYSCLK               FOSC
#define  MAIN_Fosc            FOSC

#include <stc8h.h>            //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"              //包含项目头文件
#include "timer.h"
#include "key.h"
#include "led.h"
#include "rtc.h"
#include "adc.h"

void SYS_Init(void);          //函数声明

#endif
```

2.19.3.8 key.h -- 项目配置文件的头文件

```
#ifndef __KEY_H__             //防止头文件被重复包含
#define __KEY_H__

void CalculateAdcKey(u16 adc); //函数声明
void KeyEvents();              //函数声明

#endif
```

2.19.3.9 led.h -- 项目配置文件的头文件

```
#ifndef __LED_H__             //防止头文件被重复包含
#define __LED_H__

extern BYTE LedBuf[8];

void Led_Init(void);          //函数声明
void ScanLed(void);           //函数声明
void SetLed(BYTE nIndex, BYTE nNum, BOOL bShowPoint);

#endif
```

2.19.3.10 timer.h -- 项目配置文件的头文件

```
#ifndef __TIMER_H__        //防止头文件被重复包含
#define __TIMER_H__

void Timer0_Init(void);    //函数声明
void Timer1_Init(void);    //函数声明

extern bit f1ms;           //外部变量声明
extern bit f10ms;          //外部变量声明
extern bit f500ms;         //外部变量声明

#endif
```

2.19.3.11 rtc.h -- 项目配置文件的头文件

```
#ifndef __RTC_H__          //防止头文件被重复包含
#define __RTC_H__

void RTC_Init();           //函数声明
void RTC_Update();         //函数声明
void RTC_Show();           //函数声明
void RTC_SetMode();        //函数声明
void RTC_SetUp();          //函数声明
void RTC_SetDown();        //函数声明

#endif
```

2.19.3.12 adc.h -- 项目配置文件的头文件

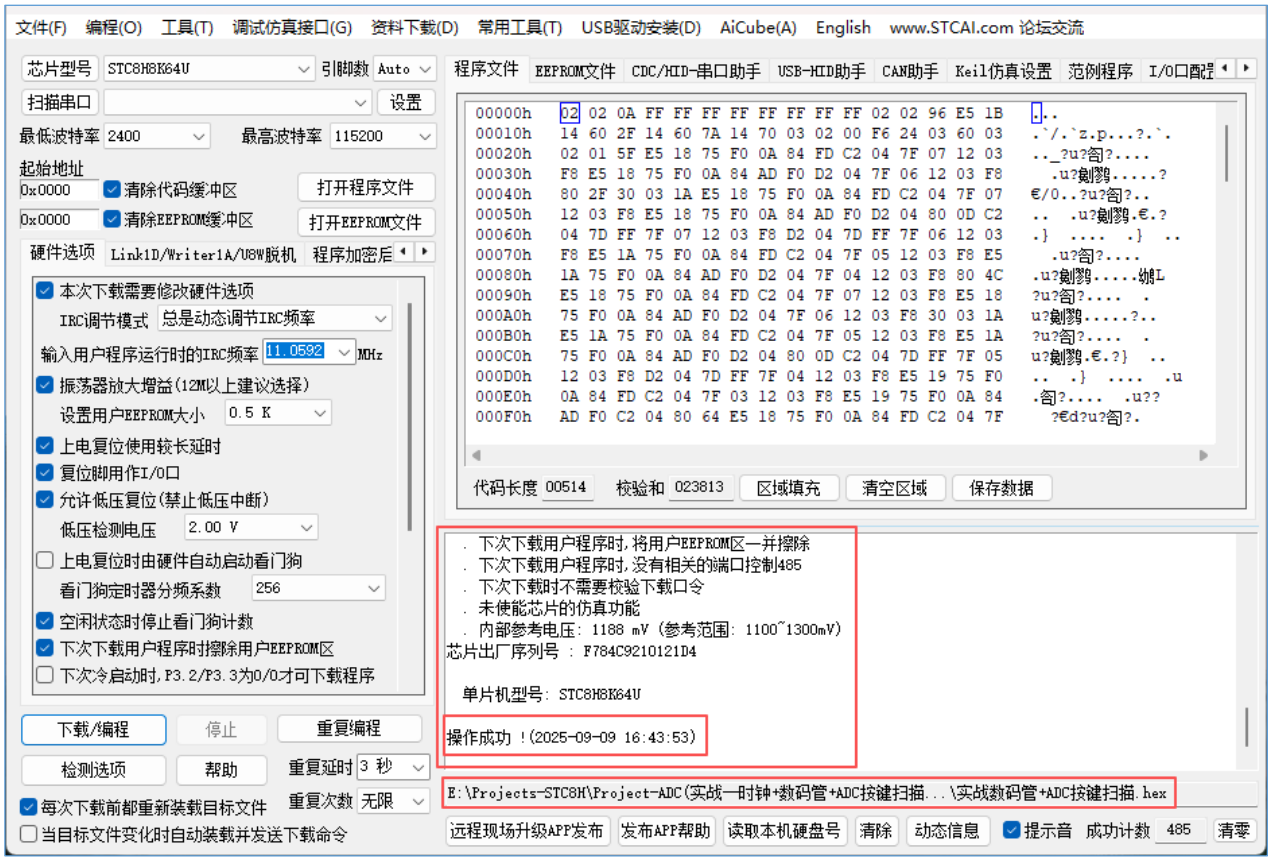
```
#ifndef __ADC_H__                       //防止头文件被重复包含
#define __ADC_H__

void ADC_Init();                        //函数声明
u16 Get_ADC12bitResult(u8 channel);     //函数声明

#endif
```

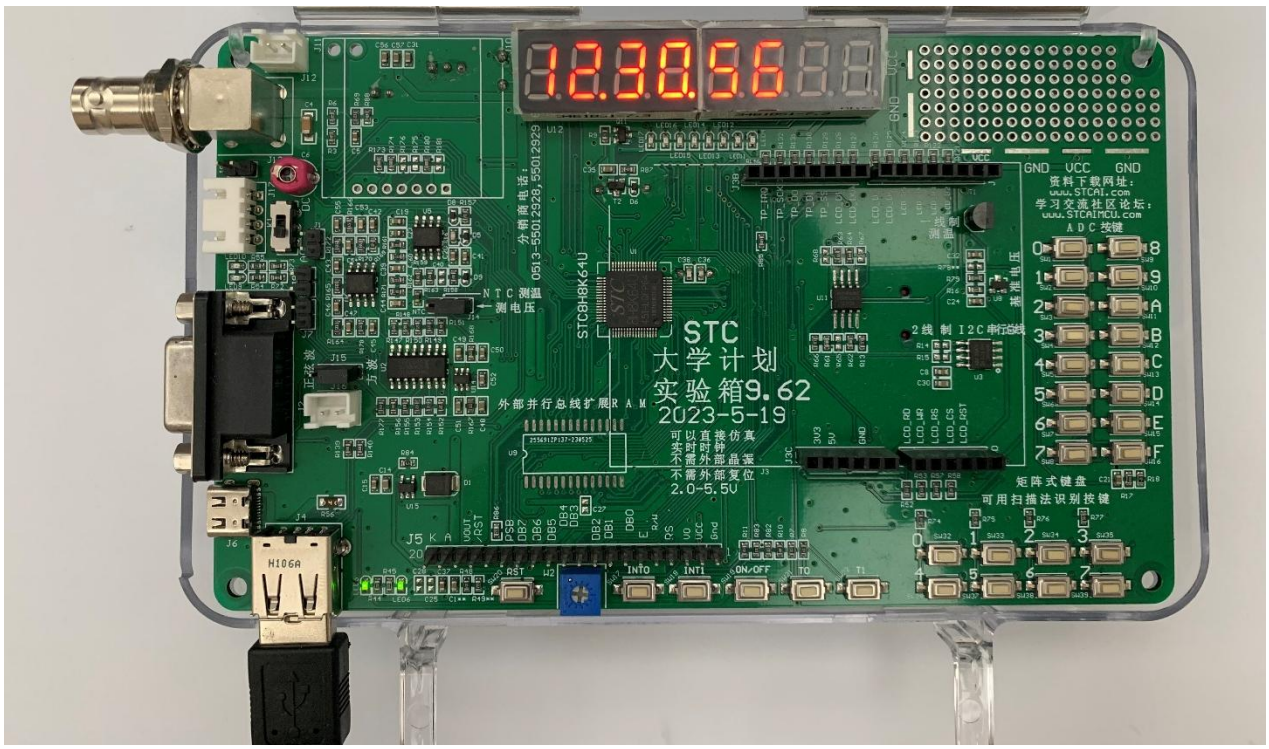
2.19.4　程序下载，观察实验现象，验证实现效果

录入代码，保存，编译。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“实战数码管+ADC 按键扫描.hex”，如下图：

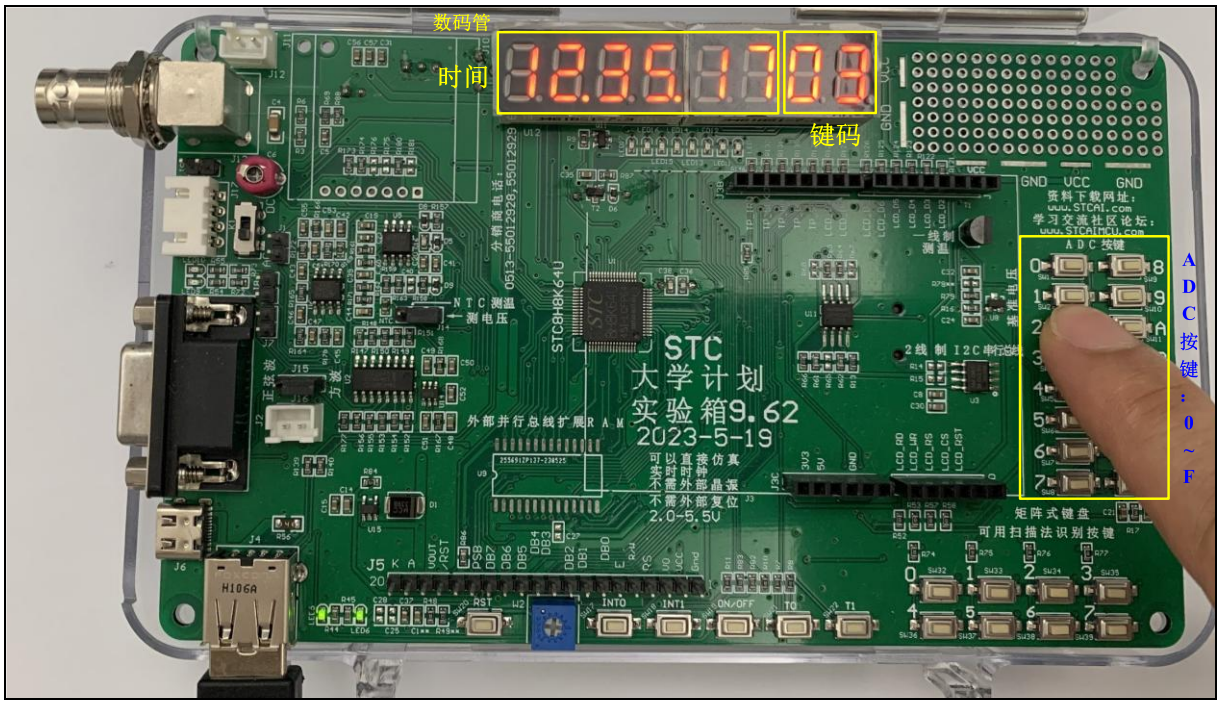


观察 STC8H8K64U 实验箱：

➲ 程序下载“操作成功”，实验箱上的数码管高 6 位立即显示“12:30:00”，同时秒显示走动。因本实验 RTC 的初始化时间为 12:30:00，所以 STC8H8K64U 实验箱数码高 6 位显示计时从 12:30:00 开始。



➲ 在 LED 数码管上面高 6 位会动态显示当前的 RTC 时间，低 2 位显示按下的 ADC 按键的键码：



1、按下 16 个 ADC 按键之一后：

- 在 LED 数码管的最高位会显示当前的按键键码

2、ADC 按键中的按键 F，是 RTC 模式设置键

- 上电后 RTC 为正常工作模式，此时数码管动态显示 RTC 时间
- 按一次按键 F，RTC 会切换为小时设置模式，此时 RTC 的小时值会 0.5 秒闪烁一次
- 再按一次按键 F，RTC 会切换为分钟设置模式，此时 RTC 的分钟值会 0.5 秒闪烁一次
- 再按一次按键 F，RTC 会切换为秒钟设置模式，此时 RTC 的秒钟值会 0.5 秒闪烁一次
- 再按一次按键 F，恢复为 RTC 正常工作模式

3、矩阵按键中的按键 D 和 E 分别是上调键和下调键

- 只有在 RTC 的时/分/秒设置模式时，上调键和下调键才有效

4、时钟设置模式时，长按按键 D 或 E，可实现快速设置功能。

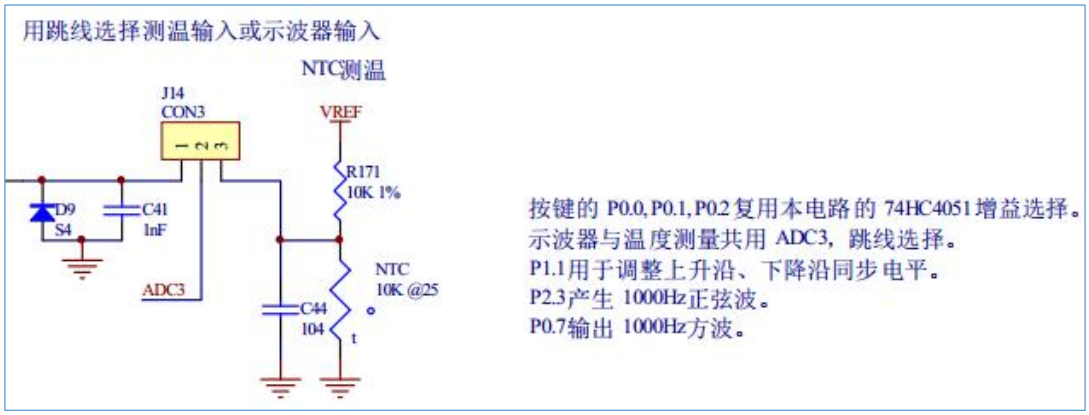
2.20 ADC 实验 -- 数码管+ NTC 热敏电阻+ADC 测温(中断方式)

2.20.1 实验介绍

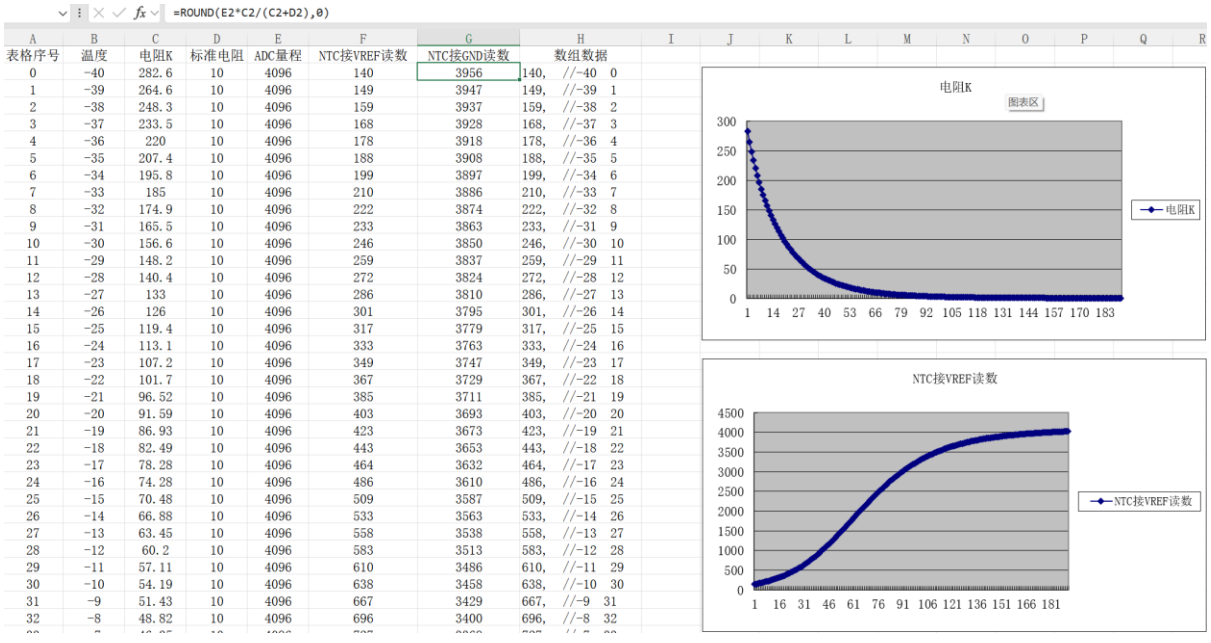
1、熟悉 ADC 工作原理，了解使用 ADC 数据采集热敏电阻 NTC 方法
2、NTC 介绍，了解 NTC 工作原理
3、熟练使用 P6、P7 口控制 LED 数码管
4、了解二分搜索算法
5、熟悉 STC8H8K64U 实验箱原理图与 AiCube-ISP 系统软件的应用
6、本项目主要熟悉如何管理多文件项目

2.20.1.1 ADC 通过采集 NTC 热敏电阻的电压值进行测温

NTC thermistor，即 Negative Temperature Coefficient thermistor，即“负温度系数热敏电阻”，其电阻值随温度升高而减小，ADC 测温通过 ADC3 通道采集 NTC 热敏电阻的电压值来计算出温度值。如图所示，NTC 热敏电阻与 10K 精密电阻 R171 串联分压后的电压输入到 ADC3 脚位，而热敏电阻随温度变化，阻值会产生变化，从而导致分压电压跟着改变，于是通过 ADC 采集分压电压值，再根据热敏电阻的温度与阻值变化特性关系就能计算出温度值。



本实验程序通过查表法计算温度值，使用 Excel 软件生成温度与 ADC 读数对应表，放到程序 temp_table 数组里，供查表计算使用：



表格存放从 -40~120 度对应的 ADC 数据，分度值为 1 度，ADC 采集到的 NTC 数据与表格进行比较，从表格里找到最接近的数据所对应的温度就是当前 NTC 感应到的温度。

为了从表格里快速查找最接近的数据，我们采用二分法查表加快查找速度。例如 -40 度数据存放在数组 temp_table[0]，120 度数据存放在数组 temp_table[160]，取中间的 temp_table[80]数据跟 ADC 采样数据进行对比，如果 ADC 采样数据小于 temp_table[80]，就可以排除后面一半表格的数据，从前面一半数据里面再次进行对分比较，依此类推：

160 / 2 = 80
80 / 2 = 40
40 / 2 = 20
20 / 2 = 10
10 / 2 = 5

经过 5 次对分查表后，剩下的数据只有 temp_table[n]~temp_table[n+5] 这几个数据，然后进行逐个比较，如果 ADC 采样数据跟其中一个数据正好相等的话，那么当前温度值就是这个数据所对应的温度。如果 ADC 采样数据在两个表格数据之间，则进行线性插补。

例如 ADC 采样数据为 2063，经过比较后发现在表格数据的 2048(25 度)与 2093(26 度)之间，则计算其百分比值作为温度的小数值。为了方便计算，我们将最终数据放大 10 倍，这样就可以使用整数计算，提高计算速度：

1 度的差值：2093 - 2048 = 45
当前读数不满 1 度的值：2063 - 2048 = 15
计算小数部分，放大 10 倍：15*10 / 45 = 3　　//取整
从而得出当前温度值为 25.3 度。

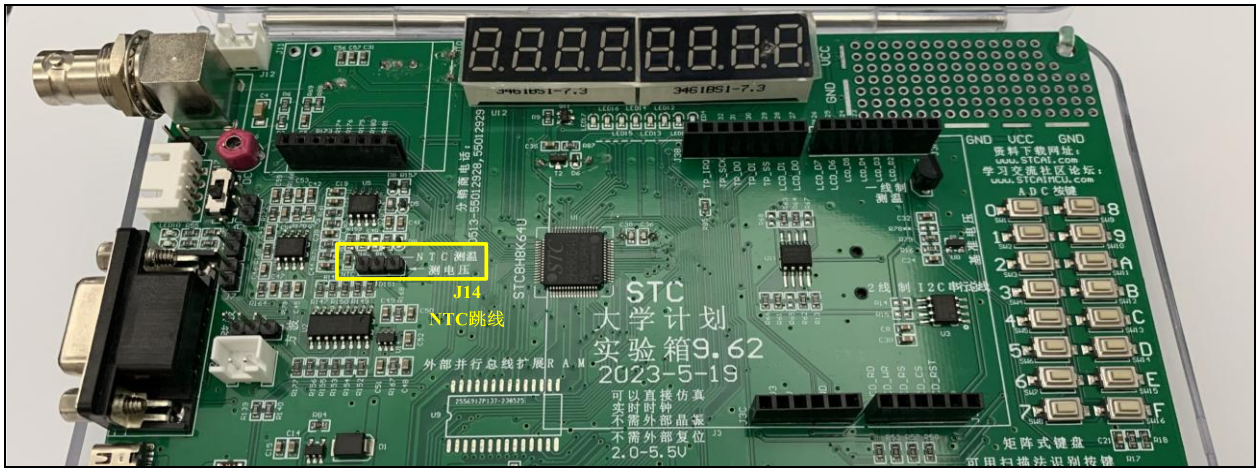
为了避免浮点计算，本实验使用整数来处理，人为规定一个 0 点（即定点计算），并且温度值扩大 10 倍（即分辨率为 0.1 度），表格中 0 度对应的数据序号是 40，扩大 10 倍为 400，即 400 表示 0 度，大于 400 为正的温度，小于 400 为负的温度。

比如查表计算的结果是 653，则实际温度为（653-400）/10 = 25.3 度。
比如查表计算的结果是 300，则实际温度为（300-400）/10 = -10.0 度。

2.20.1.2　操作说明和实验现象

将本项目的 HEX 文件下载到 STC8H8K64U 的实验箱后

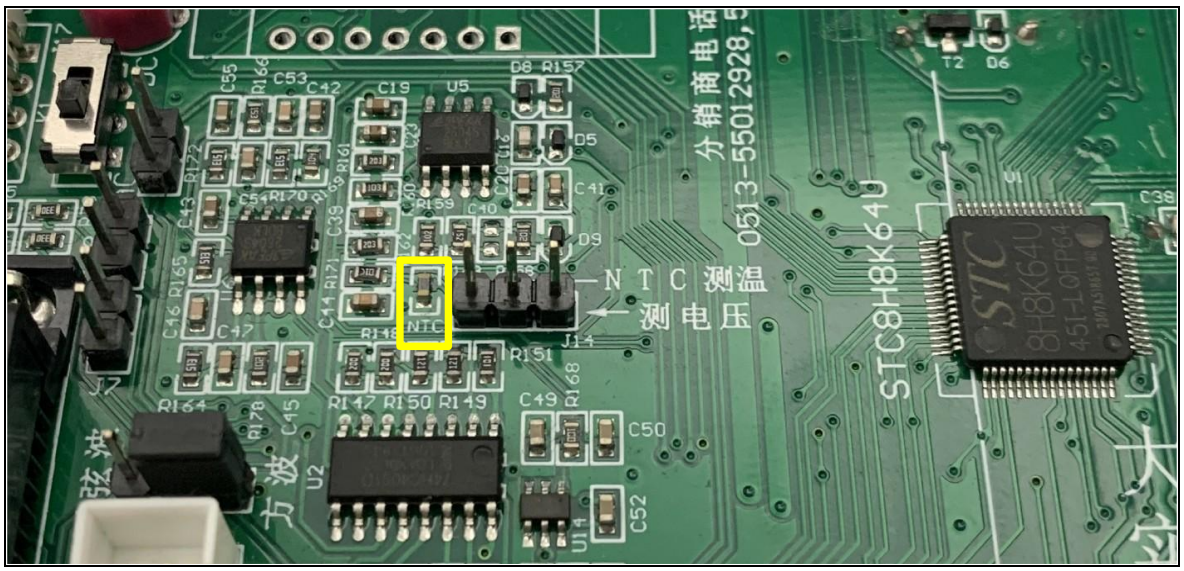
1、测量 NTC 温度，需要短接试验箱 J14 左侧两个跳线，才能连接 NTC 到芯片 ADC 脚：



2、在 LED 数码管上面低 4 位会动态显示当前的 NTC 测量的温度，高 4 位显示 NTC 的 ADC 值补码：



3、NTC 热敏电阻在 NTC 跳线旁边，用手指触摸，或者向其吹气，可改变测量的温度值：



2.20.1.3　Keil 环境下多文件项目管理说明

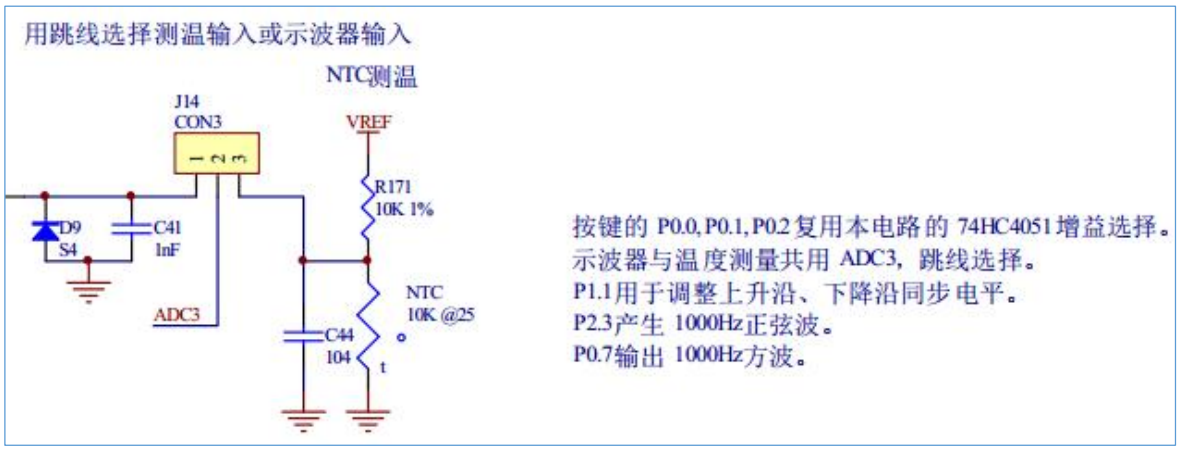
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以**方便分工合作、代码复用、模块化管理、增强可读性和可维护性**。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

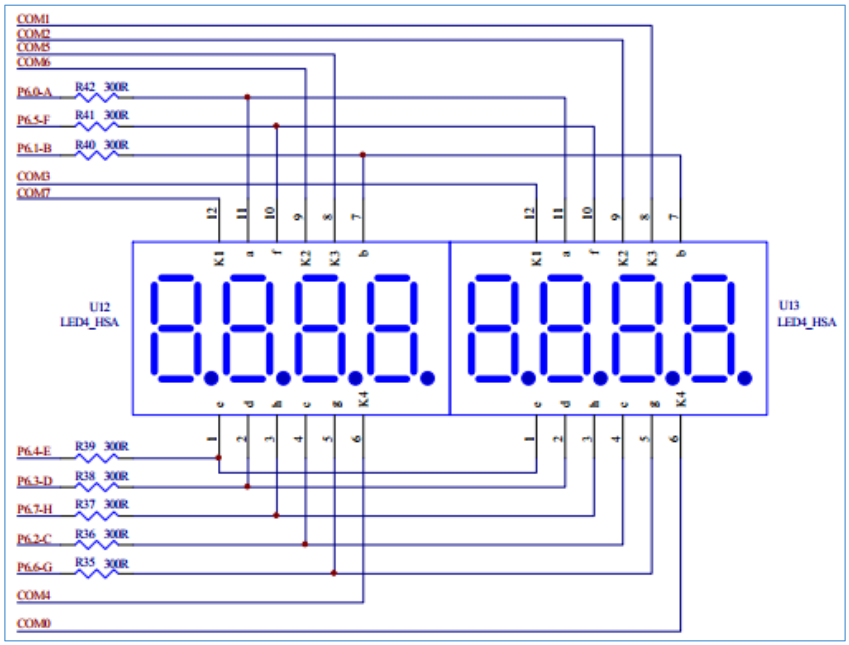
```
#ifndef XXXX
#define XXXX
...
#endif
```

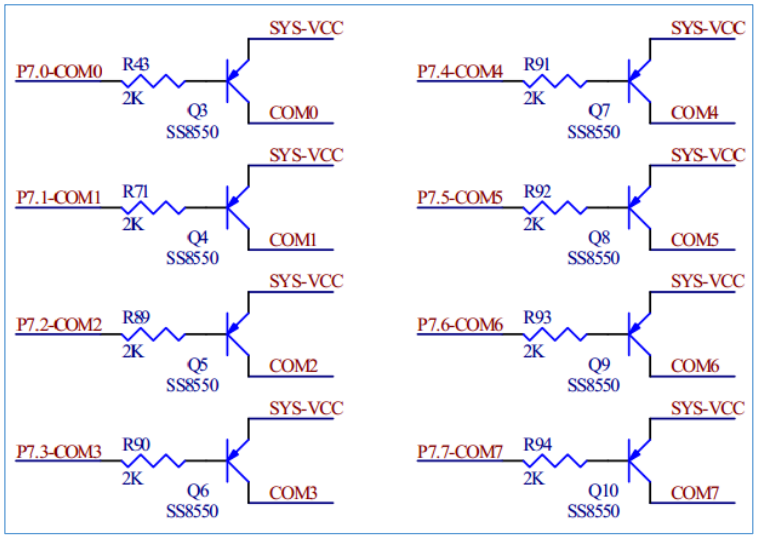
2.20.2 原理图

2.20.2.1 NTC 温度检测原理图



2.20.2.2 P6、P7 口控制数码管原理图





2.20.3 实验程序代码

2.20.3.1 main.c -- 程序主函数

```
//main.c
//程序主函数

#include "config.h"       //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

void Delay2ms(void);

void main()
{
    SYS_Init();         //系统初始化
    Delay2ms();         //ADC 电源开启需要等待 2ms 后才能开始转换
    Get_ADC(3);         //启动 ADC 转换

    while (1)
    {
        if (f1ms)           //判断 1ms 标志位
        {
            f1ms = 0;       //清除 1ms 标志，并处理 1ms 事件
            ScanLed();      //LED 数码管扫描
        }

        if (f500ms)         //判断 500ms 标志位
        {
            f500ms = 0;     //清除 500ms 标志，并处理 500ms 事件
            NTC_Show(ADC_Data);     //在 LED 数码管上显示 NTC 温度
            Get_ADC(3);  //启动 ADC 转换
        }
    }
}

void SYS_Init(void)         //系统初始化函数
{
    P_SW2 |= 0x80;          //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;       //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x08;       //初始化 P1 口为准双向口模式, P1.3 高阻输入
    P2M0 = 0x00; P2M1 = 0x00;       //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //初始化 P4 口为准双向口模式
    P5M0 = 0x02; P5M1 = 0x00;       //初始化 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //初始化 P7 口为准双向口模式
```

```
    Led_Init();         //LED 数码管初始化
    Timer0_Init();      //定时器 0 初始化,用于产生 1ms 和 500ms 的时间基准
    ADC_Init();         //ADC 初始化

    EA = 1;             //总中断允许位打开
}

void Delay2ms(void)     //@11.0592MHz
{
    unsigned char data i, j;

    i = 29;
    j = 183;
    do
    {
        while (--j);
    } while (--i);
}
```

2.20.3.2 led.c -- 扫描 LED 数码管

```
//led.c
//使用 P6 和 P7 口动态扫描 LED 数码管
//由于 LED 数码管是由 8 个 SEG 和 8 个 COM 组成的矩阵 LED
//所以 LED 数码管也需要扫描显示

#include "config.h"             //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

#define   LED_OFF     0x00
//定义关闭 LED 数码管的段码数据,LED 数码管为共阴极,所以 SEG 线全部输出 0 即可关闭一个 8 段数码管
#define   LED_        0x40      //定义 LED 数码管中杠的段码数据
#define   LED_POINT   0x80      //定义 LED 数码管小数点的段码数据

BYTE    ShowIndex;          //LED 数码管扫描显示索引(8 个数码管轮流扫描显示)
BYTE    LedBuf[8];          //8 个 LED 数码管显示数据缓冲区

BYTE    code LED_PATTERN[10] =      //定义 LED 数码管显示数据 0~9 的段码数据
{
    0x3F,   //0                     //此表为共阴极数码管的段码数据
    0x06,   //1
    0x5B,   //2
    0x4F,   //3
    0x66,   //4
    0x6D,   //5
    0x7D,   //6
    0x07,   //7
```

```
    0x7F,    //8
    0x6F,    //9
};

void Led_Init(void)                 //LED 数码管初始化程序
{
    LedBuf[0] = LED_OFF;        //初始化 LED 数码管显示数据缓冲区,上电关闭显示
    LedBuf[1] = LED_OFF;
    LedBuf[2] = LED_OFF;
    LedBuf[3] = LED_OFF;
    LedBuf[4] = LED_OFF;
    LedBuf[5] = LED_OFF;
    LedBuf[6] = LED_OFF;
    LedBuf[7] = LED_OFF;

    ShowIndex = 0;                  //初始化扫描显示索引
}

void ScanLed(void)                      //LED 数码管扫描显示程序,  (需每 1ms 执行一次!!!)
{
    P6 = (~LedBuf[ShowIndex]);      //将当前扫描到的 LED 数码管的段码移入到 P6 口
    P7 = (~(1 << ShowIndex));       //将当前扫描到的 LED 数码管的 COM 数据移入到 P7 口

    if (++ShowIndex >= 8)           //扫描索引值递增
        ShowIndex = 0;              //扫描完一轮后索引值归零
}

void SetLed (BYTE nIndex,       //设置 LED 显示函数,nIndex:设置目标数码管,0~7 分别对应 8 个数码
             BYTE nNum,         //nNum:设置显示的数据,有效值 0~9,其他值则关闭显示
             BOOL bShowPoint    //bShowPoint:设置是否显示小数点
            )
{
    if (nIndex < 8)             //目标数码管的有效值为 0~7
    {
        if (nNum < 10)          //判断显示的值是否为 0~9
            LedBuf[nIndex] = LED_PATTERN[nNum];
                                //将相应数据的段码赋值到 LED 数码管显示数据缓冲区
        else if (nNum == 11)    //判断显示的值是否为"-"
            LedBuf[nIndex] = LED_;          //显示"-"
        else
            LedBuf[nIndex] = LED_OFF;       //如果是其他值则关闭显示

        if (bShowPoint)         //判断是否需要显示小数点
            LedBuf[nIndex] |= LED_POINT;  //如果需要显示,则增加小数点的段码数据
    }
}
```

2.20.3.3 timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:1ms 定时,产生 1ms 和 500ms 的标志位

#include "config.h"       //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

bit       f1ms;           //1ms 标志位
bit       f500ms;         //500ms 标志位
int       cnt500;         //500 次计数器,用于产生 500ms 的时间

void Timer0_Init(void)        //1 毫秒@11.0592MHz
{
    AUXR |= 0x80;             //定时器 0 时钟 1T 模式
    TMOD &= 0xF0;             //设置定时器 0 为模式 0
    TL0 = 0xCD;               //设置定时器 0 初始值
    TH0 = 0xD4;               //设置定时器 0 初始值
    TF0 = 0;                  //清除 TF0 标志
    TR0 = 1;                  //定时器 0 开始计时
    ET0 = 1;                  //使能定时器 0 中断

    cnt500 = 0;               //初始化 500 次计数器
    f1ms = 0;                 //初始化 1m 标志位
    f500ms = 0;               //初始化 500ms 标志位
}

void Timer0_Isr(void) interrupt 1        //定时器 0 中断服务程序
{
    f1ms = 1;                 //设置 1ms 标志位

    if (cnt500++ >= 500)      //判断 500 次计数是否已满
    {
        cnt500 = 0;           //清除计数值,开始开始计数
        f500ms = 1;           //设置 500ms 标志位
    }
}
```

2.20.3.4 adc.c -- ADC 初始化和 ADC 中断方式数据采集程序

```
//adc.c
//ADC 程序
//包括 ADC 初始化和 ADC 数据采集

#include "config.h"           //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件
```

```
u16 ADC_Data;

void ADC_Init()                 //ADC 初始化函数
{
    ADCTIM = 0x3f;              //设置 ADC 内部时序，ADC 采样时间建议设最大值
    ADCCFG = 0x2f;              //设置 ADC 时钟为系统时钟/2/16
    ADC_CONTR = 0x80;           //使能 ADC 模块
    EADC = 1;                   //使能 ADC 中断
}

void Get_ADC(u8 channel)    //channel = 0~15
{
    ADC_CONTR = (ADC_CONTR & 0xf0) | channel;       //设置 ADC 转换通道
    ADC_CONTR |= ADC_START;    //启动 ADC 转换}

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~ADC_FLAG;                         //清除 ADC 结束标志
    ADC_Data = ((u16)ADC_RES << 8) | ADC_RESL;      //读取 ADC 结果
}
```

2.20.3.5 ntc.c -- ADC 中断方式检测 NTC 热敏电阻测温程序

```
//ntc.c
//ADC 采集热敏电阻 NTC 程序

#include "config.h" //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

//    MF52E 10K at 25, B = 3950, ADC = 12 bits
u16 code temp_table[]={
                        140,    //;-40    0
                        149,    //;-39    1
                        159,    //;-38    2
                        168,    //;-37    3
                        178,    //;-36    4
                        188,    //;-35    5
                        199,    //;-34    6
                        210,    //;-33    7
                        222,    //;-32    8
                        233,    //;-31    9
                        246,    //;-30    10
                        259,    //;-29    11
                        272,    //;-28    12
                        286,    //;-27    13
                        301,    //;-26    14
                        317,    //;-25    15
```

```
333,    //;-24    16
349,    //;-23    17
367,    //;-22    18
385,    //;-21    19
403,    //;-20    20
423,    //;-19    21
443,    //;-18    22
464,    //;-17    23
486,    //;-16    24
509,    //;-15    25
533,    //;-14    26
558,    //;-13    27
583,    //;-12    28
610,    //;-11    29
638,    //;-10    30
667,    //;-9     31
696,    //;-8     32
727,    //;-7     33
758,    //;-6     34
791,    //;-5     35
824,    //;-4     36
858,    //;-3     37
893,    //;-2     38
929,    //;-1     39
965,    //;0      40
1003,   //;1      41
1041,   //;2      42
1080,   //;3      43
1119,   //;4      44
1160,   //;5      45
1201,   //;6      46
1243,   //;7      47
1285,   //;8      48
1328,   //;94     9
1371,   //;10     50
1414,   //;11     51
1459,   //;12     52
1503,   //;13     53
1548,   //;14     54
1593,   //;15     55
1638,   //;16     56
1684,   //;17     57
1730,   //;18     58
1775,   //;19     59
1821,   //;20     60
1867,   //;21     61
```

```
1912,   //;22   62
1958,   //;23   63
2003,   //;24   64
2048,   //;25   65
2093,   //;26   66
2137,   //;27   67
2182,   //;28   68
2225,   //;29   69
2269,   //;30   70
2312,   //;31   71
2354,   //;32   72
2397,   //;33   73
2438,   //;34   74
2479,   //;35   75
2519,   //;36   76
2559,   //;37   77
2598,   //;38   78
2637,   //;39   79
2675,   //;40   80
2712,   //;41   81
2748,   //;42   82
2784,   //;43   83
2819,   //;44   84
2853,   //;45   85
2887,   //;46   86
2920,   //;47   87
2952,   //;48   88
2984,   //;49   89
3014,   //;50   90
3044,   //;51   91
3073,   //;52   92
3102,   //;53   93
3130,   //;54   94
3157,   //;55   95
3183,   //;56   96
3209,   //;57   97
3234,   //;58   98
3259,   //;59   99
3283,   //;60   100
3306,   //;61   101
3328,   //;62   102
3351,   //;63   103
3372,   //;64   104
3393,   //;65   105
3413,   //;66   106
3432,   //;67   107
```

```
3452,   //;68   108
3470,   //;69   109
3488,   //;70   110
3506,   //;71   111
3523,   //;72   112
3539,   //;73   113
3555,   //;74   114
3571,   //;75   115
3586,   //;76   116
3601,   //;77   117
3615,   //;78   118
3628,   //;79   119
3642,   //;80   120
3655,   //;81   121
3667,   //;82   122
3679,   //;83   123
3691,   //;84   124
3702,   //;85   125
3714,   //;86   126
3724,   //;87   127
3735,   //;88   128
3745,   //;89   129
3754,   //;90   130
3764,   //;91   131
3773,   //;92   132
3782,   //;93   133
3791,   //;94   134
3799,   //;95   135
3807,   //;96   136
3815,   //;97   137
3822,   //;98   138
3830,   //;99   139
3837,   //;100  140
3844,   //;101  141
3850,   //;102  142
3857,   //;103  143
3863,   //;104  144
3869,   //;105  145
3875,   //;106  146
3881,   //;107  147
3887,   //;108  148
3892,   //;109  149
3897,   //;110  150
3902,   //;111  151
3907,   //;112  152
3912,   //;113  153
```

```
                    3917,   //;114   154
                    3921,   //;115   155
                    3926,   //;116   156
                    3930,   //;117   157
                    3934,   //;118   158
                    3938,   //;119   159
                    3942    //;120   160
                    };

/********************  计算温度 ********************************************/
// 计算结果: 0 对应-40.0 度, 400 对应 0 度, 650 对应 25.0 度, 最大 1600 对应 120.0 度.
// 为了通用, ADC 输入为 12bit 的 ADC 值.
/***************************************************/

#define  D_SCALE    10  //结果放大倍数, 放大 10 倍就是保留一位小数
u16 get_temperature(u16 adc)
{
    u16     code *p;
    u16     i;
    u8      j,k,min,max;

    p = temp_table;         //指针指向温度表格
    if(adc < p[0])      return (0xfffe);
    if(adc > p[160])    return (0xffff);

    min = 0;                //-40 度
    max = 160;              //120 度

    for(j=0; j<5; j++)      //对分查表 5 次，缩小查表范围
    {
        k = min / 2 + max / 2;
        if(adc <= p[k])     max = k;
        else                min = k;
    }

    if(adc == p[min])       i = min * D_SCALE;
                            //如果 ADC 值正好等于最小指针对应值，则取其序号作为温度参数并放大 10 倍
    else if(adc == p[max])  i = max * D_SCALE;
                            //如果 ADC 值正好等于最大指针对应值，则取其序号作为温度参数并放大 10 倍
    else                    // min < temp < max
    {
        while(min <= max)       //剩下部分逐个查表对比
        {
            min++;
            if(adc == p[min])   {i = min * D_SCALE; break;}
                                //如果 ADC 值正好等于 p[min] ，则取其序号作为温度参数并放大 10 倍
```

```
            else if(adc < p[min])      //如果 ADC 值在 p[min-1] 与 p[min] 之间，则增加补偿
            {
                min--;
                i = p[min];             //min
                j = (adc - i) * D_SCALE / (p[min+1] - i);
                i = min;
                i *= D_SCALE;
                i += j;
                break;
            }
        }
    }
    return i;
}

void NTC_Show(u16 adc)     //NTC 温度计算显示程序
{
    u16 temp;

    if(temp > 4095) return;  //判断 ADC 值是否有效，无效则直接退出

    adc = 4096 - adc;               //Rt 接地

    //在数码管左侧显示 ADC 采样数据
    if((adc / 1000) == 0)
    {
        SetLed(7, 12, 0);           //千位为 0，在数码 7 上显示空白
    }
    else
    {
        SetLed(7, (BYTE)(adc / 1000), 0);           //在数码 7 上显示 ADC 的千位
    }

    SetLed(6, (BYTE)((adc % 1000) / 100), 0);       //在数码 6 上显示 ADC 的百位
    SetLed(5, (BYTE)((adc % 100) / 10), 0);         //在数码 5 上显示 ADC 的十位
    SetLed(4, (BYTE)(adc % 10), 0);                 //在数码 4 上显示 ADC 的个位

    temp = get_temperature(adc);            //计算温度值

    if(temp >= 400)     F0 = 0, temp -= 400;        //温度 >= 0 度
    else                F0 = 1, temp = 400 - temp;  //温度 <  0 度

    if(F0)
    {
        SetLed(3, 11, 0);           //温度为零下，在数码 3 上显示"-"
    }
```

```
    else if((temp / 1000) == 0)
    {
        SetLed(3, 12, 0);          //千位为 0，在数码 3 上显示空白
    }
    else
    {
        SetLed(3, (BYTE)(temp / 1000), 0);          //在数码 3 上显示 ADC 的千位
    }

    SetLed(2, (BYTE)((temp % 1000) / 100), 0);      //在数码 2 上显示 ADC 的百位
    SetLed(1, (BYTE)((temp % 100) / 10), 1);        //在数码 1 上显示 ADC 的十位，并显示小数点
    SetLed(0, (BYTE)(temp % 10), 0);                //在数码 0 上显示 ADC 的个位
}
```

2.20.3.6 config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__            //防止头文件被重复包含
#define __CONFIG_H__

#define   HIRC              11059200UL
#define   FOSC              11059200UL
#define   SYSCLK            FOSC
#define   MAIN_Fosc         FOSC

#include < stc8h.h>           //包含外部头文件
#include <stdio.h>
#include <intrins.h>

#include "def.h"              //包含项目头文件
#include "timer.h"
#include "led.h"
#include "ntc.h"
#include "adc.h"

void SYS_Init(void);          //函数声明

#endif
```

2.20.3.7 led.h -- 项目配置文件的头文件

```
#ifndef __LED_H__             //防止头文件被重复包含
#define __LED_H__

extern BYTE LedBuf[8];

void Led_Init(void);          //函数声明
```

```
void ScanLed(void);             //函数声明
void SetLed(BYTE nIndex, BYTE nNum, BOOL bShowPoint);

#endif
```

2.20.3.8 timer.h -- 项目配置文件的头文件

```
#ifndef __TIMER_H__         //防止头文件被重复包含
#define __TIMER_H__

void Timer0_Init(void);     //函数声明

extern bit f1ms;            //外部变量声明
extern bit f10ms;           //外部变量声明
extern bit f500ms;          //外部变量声明

#endif
```

2.20.3.9 adc.h -- 项目配置文件的头文件

```
#ifndef __ADC_H__           //防止头文件被重复包含
#define __ADC_H__

extern u16 ADC_Data;

void ADC_Init();            //函数声明
void Get_ADC(u8 channel);   //函数声明

#endif
```

2.20.3.10 ntc.h -- 项目配置文件的头文件

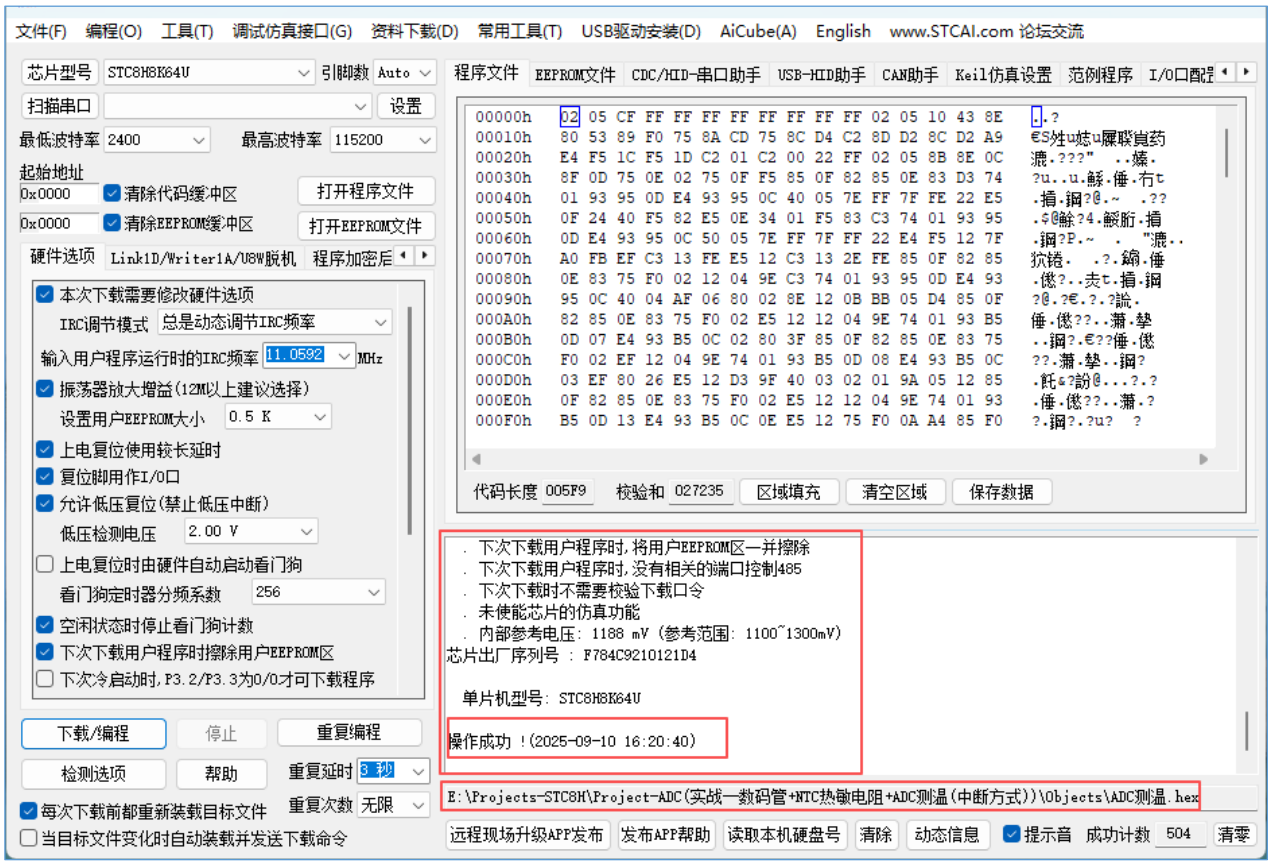
```
#ifndef __NTC_H__           //防止头文件被重复包含
#define __NTC_H__

void NTC_Show(u16 adc);             //函数声明

#endif
```

2.20.4　　程序下载，观察实验现象

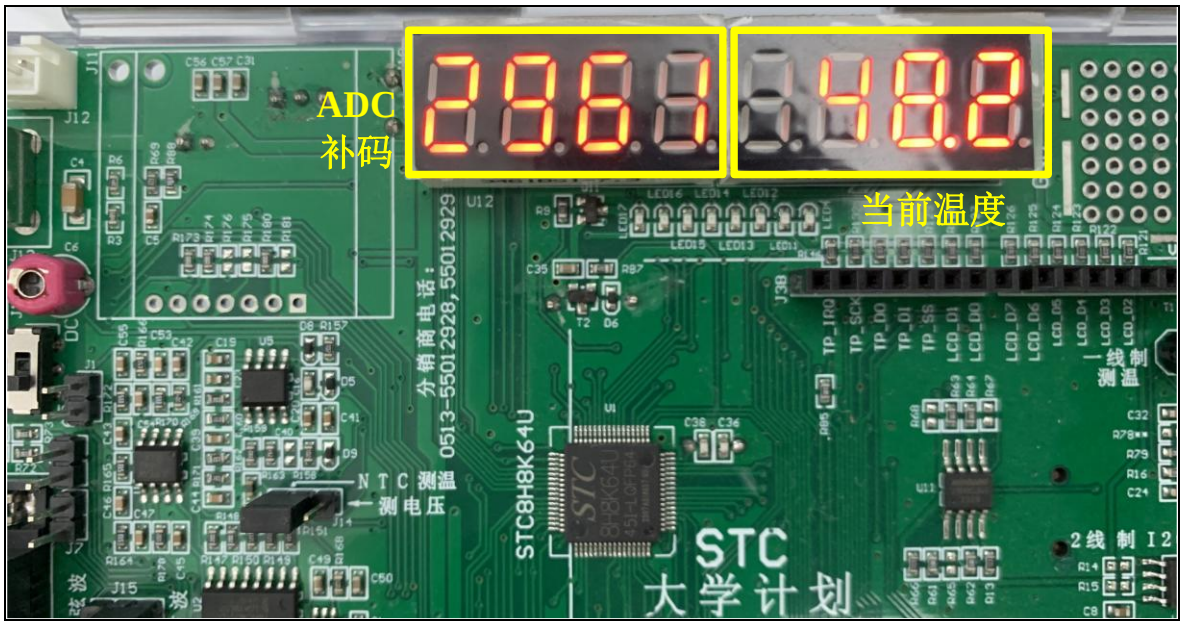
录入代码，保存，编译。连接好 STC8H8K64U 实验箱，运行 AiCube-ISP 软件系统，按步骤打开并下载程序“ADC 测温.hex”，如下图：



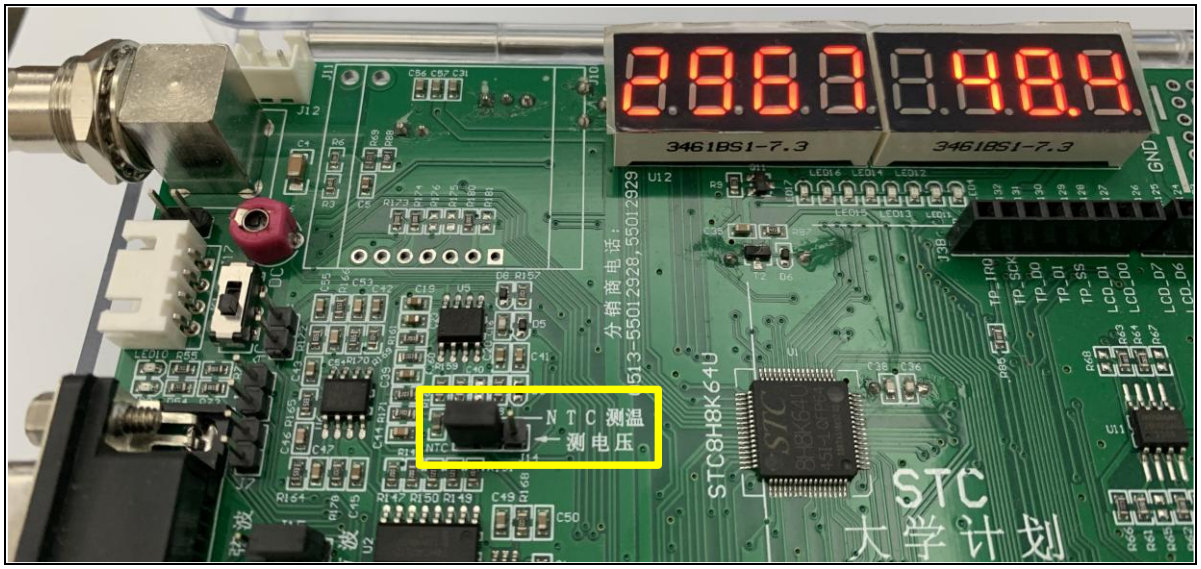
下载/烧录成功！

现在，观察 STC8H8K64U 实验箱：

1、下载/烧录成功后，STC8H8K64U 实验箱的 LED 数码管上高 4 位立即动态显示 NTC 的 ADC 值补码，低 4 位动态显示当前的 NTC 测量的温度：



2、NTC 热敏电阻在 NTC 跳线的左边，用手指触摸，或者向其吹气，可改变测量的温度值：



2.21 大型综合实战 -- 课程设计：硬件 SPI 读写串行 Flash

2.21.1 实验介绍

1、本实验项目主要目的：

- 掌握串口通讯
- 硬件 SPI 读写串行 Flash

2、掌握 STC8H8K64U 实验箱原理图中的串口通信电路、硬件 SPI 电路

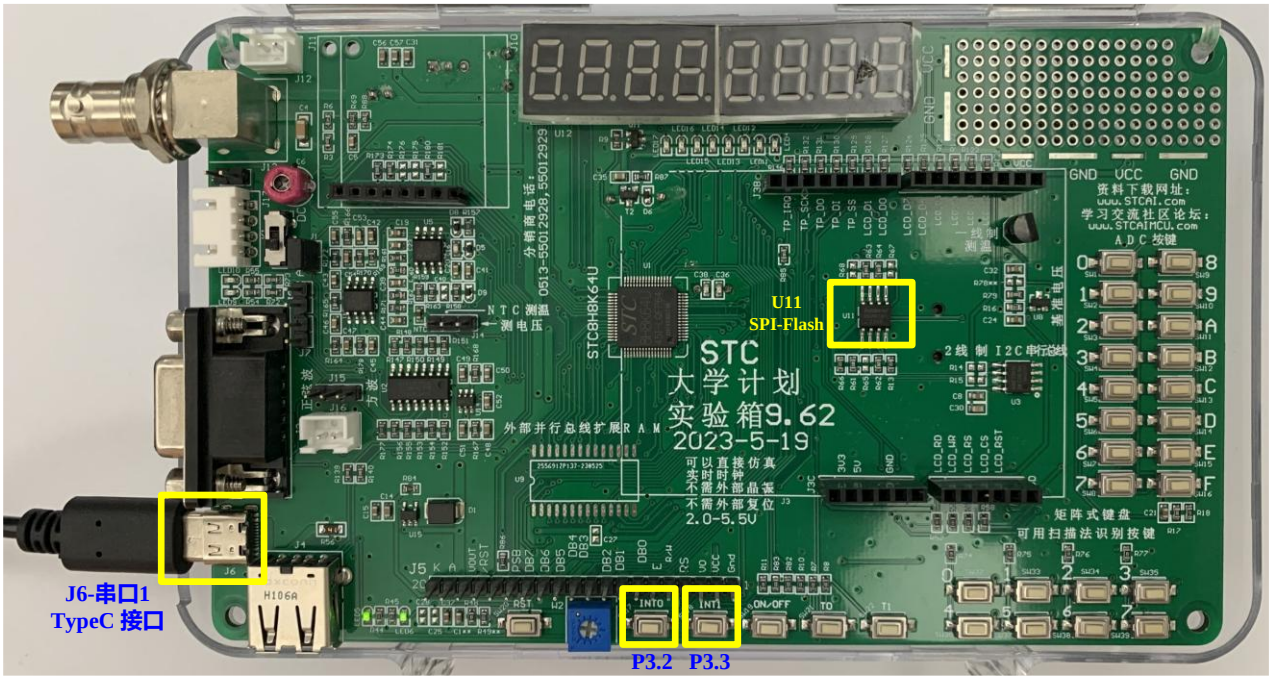
3、熟练使用 AiCube-ISP 系统软件

4、了解 USB Link1D 工具的原理及使用

5、熟悉如何管理多文件项目

2.21.1.1 认识 STC8H8K64U 的实验箱

STC8H8K64U 的实验箱正面图：



J6： 串口 1 的 TypeC 接口。本实验需要用专门定制的 USB 转串口/TTL（连接 USB Link1D 工具） - TypeC（连接实验箱 J6）接口线相连。（详见 J6 原理图）

SPI-Flash：详见 STC8H8K64U 实验箱中 U6 原理图

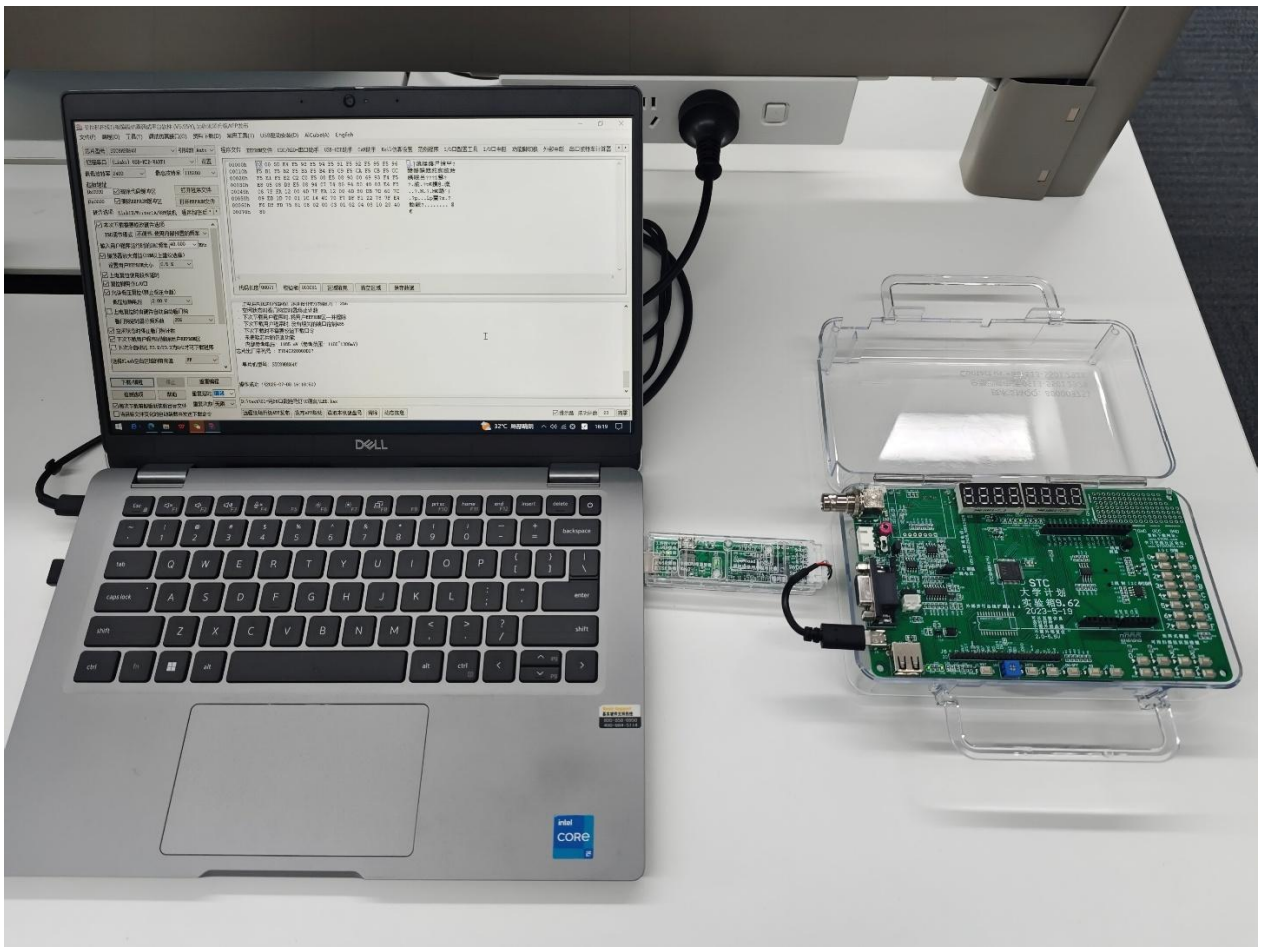
2.21.1.2 专门定制的 USB 转串口/TTL - TypeC 接口线



2.21.1.3 USB Link1D 工具



2.21.1.4 STC8H8K64U 实验箱通过 USB Link1D 工具与电脑连接图



2.21.1.5 按键去抖动说明

如按键要判断为正常按下，而不是抖动，需要该按键保持按下持续保持为低状态时间是 20ms ~ 50ms；主循环中有个 1ms 的时间基准，将按键扫描程序作为 1ms 事件，按键状态维持 50ms 不变，即可对按键去抖动，得到稳定的键码。

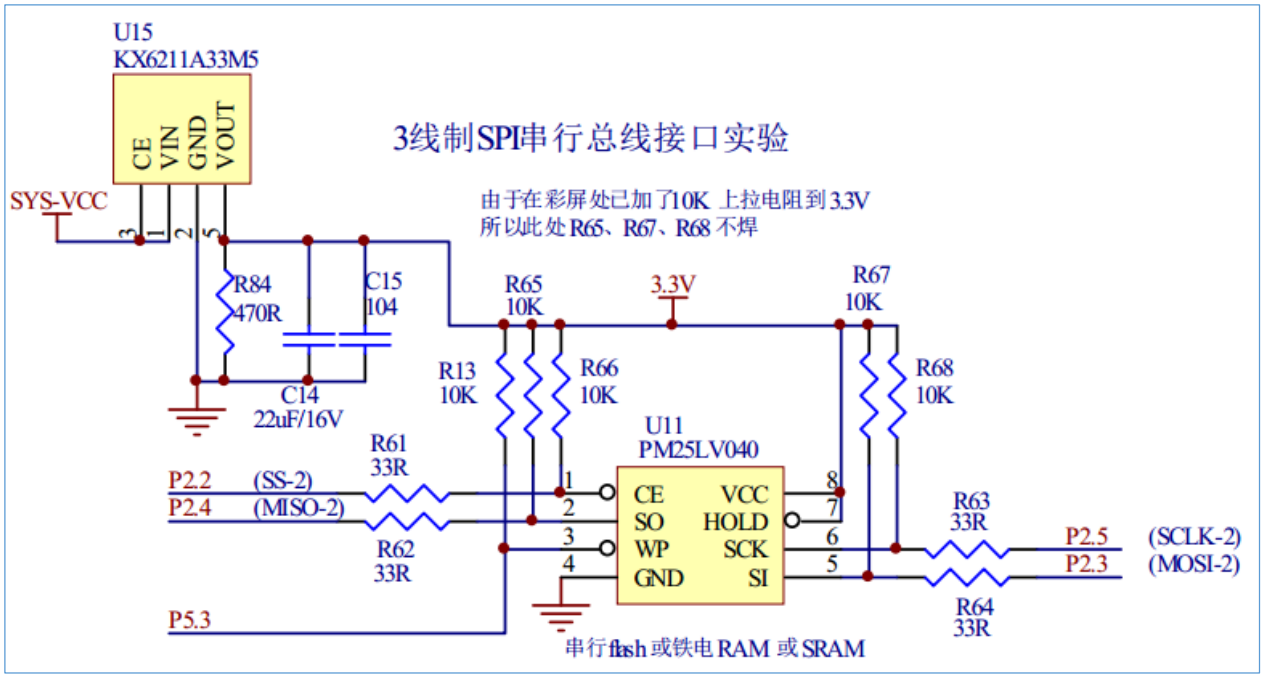
2.21.1.6 Keil 环境下多文件项目管理说明

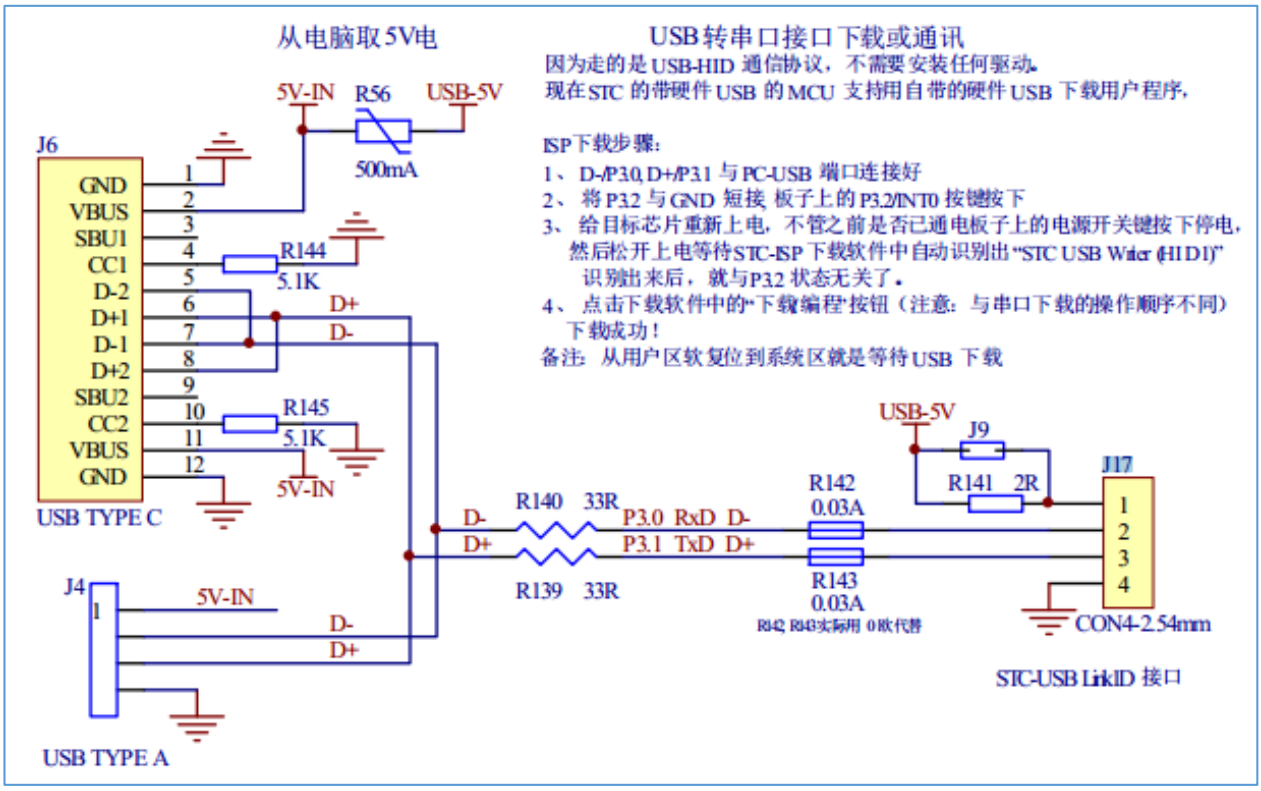
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以**方便分工合作、代码复用、模块化管理、增强可读性和可维护性**。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

```
#ifndef XXXX
#define XXXX
...
#endif
```

2.21.2 原理图





2.21.3　实验程序代码

2.21.3.1　main.c -- 程序主函数

```
//main.c
//程序主函数
//用 USB 转串口/TTL（连接 USB Link1D 工具） - TypeC（连接实验箱 J6）接口线相连电脑和实验箱
//在 AiCube-ISP 中，串口助手打开 CDC1 对应的 COM 口。
//通过按键触发硬件 SPI 对串行 Flash 做扇区擦除、写入、读出的操作。
//P32 按键读取串行 Flash ID，以及 Flash 指定地址数据。
//P33 按键短按擦除指定地址 Flash 扇区，并写入计数值，然后读出，通过串口打印操作结果。
//P33 按键长按擦除指定地址 Flash 扇区，重置计数值。

#include "config.h"        //头文件中已包含 stc8h.h 以及其他头文件

void main()
{
    SYS_Init();          //系统初始化

    while (1)
    {
        if (f1ms)              //判断 1ms 标志位
        {
            f1ms = 0;        //清除 1ms 标志，并处理 1ms 事件

            ScanKey();     //按键扫描
```

```
            KeyEvents();  //按键处理事件
        }
    }
}

void SYS_Init(void)     //系统初始化函数
{
    P_SW2 |= 0x80;                  //允许访问扩展的特殊寄存器，XFR

    P0M0 = 0x00; P0M1 = 0x00;       //初始化 P0 口为准双向口模式
    P1M0 = 0x00; P1M1 = 0x00;       //初始化 P1 口为准双向口模式
    P2M0 = 0x00; P2M1 = 0x00;       //初始化 P2 口为准双向口模式
    P3M0 = 0x00; P3M1 = 0x00;       //初始化 P3 口为准双向口模式
    P4M0 = 0x00; P4M1 = 0x00;       //初始化 P4 口为准双向口模式
    P5M0 = 0x00; P5M1 = 0x00;       //初始化 P5 口为准双向口模式
    P6M0 = 0x00; P6M1 = 0x00;       //初始化 P6 口为准双向口模式
    P7M0 = 0x00; P7M1 = 0x00;       //初始化 P7 口为准双向口模式

    P3PU |= 0x0c;                   //使能 P3.2,P3.3 内部上拉

    P_SW1 &= ~0xc0;                 //UART1 通道选择, 00: P3.0 P3.1,   01: P3.6 P3.7,   10: P1.6 P1.7,   11: P4.3 P4.4
    P_SW1 = (P_SW1 & ~0x0c) | 0x04;     //SPI 通道选择, 00: P1.2 P1.3 P1.4 P1.5,   01: P2.2 P2.3 P2.4 P2.5,
                                        //              10: P5.4 P4.0 P4.1 P4.3,   11: P3.5 P3.4 P3.3 P3.2

    SPI_Init();         //SPI 接口初始化
    Uart1_Init();       //UART 接口初始化
    Timer0_Init();      //定时器 0 初始化,用于产生 1ms 的时间基准

    EA = 1;             //总中断允许位打开
}
```

2.21.3.2 spi.c -- 初始化 SPI 接口、串行 Flash 操作函数

```
//spi.c
//初始化 SPI 接口、串行 Flash 操作函数

#include "config.h"     //头文件中已包含 stc8h.h 以及其他头文件

u8  B_FlashOK;          //Flash 状态
u8  PM25LV040_ID, PM25LV040_ID1, PM25LV040_ID2;

void SPI_Init()         //SPI 初始化函数
{
    SPCTL = (SPCTL & ~3) | 1;       //SPI  时钟频率选择, 0: 4T, 1: 8T, 2: 16T, 3: 2T
    SPCTL |= SSIG | SPEN | MSTR;    //允许 SPI，设为主机,忽略 SS 脚
}
```

```
void SPI_WriteByte(u8 out)
{
    SPDAT = out;                    //写入需要发送的数据
    while ((SPSTAT & SPIF) == 0);   //等待传输完成
    SPSTAT = SPIF | WCOL;           //清 SPIF, WCOL 标志
}

/*************************************************************************/
u8 SPI_ReadByte(void)
{
    SPDAT = 0xff;                   //MOSI 发送 0xFF(高电平)产生 CLK，同时读取 MISO 数据
    while ((SPSTAT & SPIF) == 0);   //等待传输完成
    SPSTAT = SPIF | WCOL;           //清 SPIF, WCOL 标志
    return (SPDAT);                 //返回 MISO 读取数据
}
/************************************************
检测 Flash 是否准备就绪
入口参数: 无
出口参数:
    0 : 没有检测到正确的 Flash
    1 : Flash 准备就绪
************************************************/
void FlashCheckID(void)
{
    SPI_CE_Low();                   //SPI 片选使能
    SPI_WriteByte(SFC_RDID);        //发送读取 ID 命令
    SPI_WriteByte(0x00);            //空读 3 个字节
    SPI_WriteByte(0x00);
    SPI_WriteByte(0x00);
    PM25LV040_ID1 = SPI_ReadByte();  //读取制造商 ID1
    PM25LV040_ID = SPI_ReadByte();   //读取设备 ID
    PM25LV040_ID2 = SPI_ReadByte();  //读取制造商 ID2
    SPI_CE_High();                   //SPI 片选禁能

    printf("制造商 ID1 = 0x%02X",PM25LV040_ID1);
    printf("\r\n      ID2 = 0x%02X",PM25LV040_ID2);
    printf("\r\n 设备 ID = 0x%02X\r\n",PM25LV040_ID);

    if((PM25LV040_ID1 == 0x9d) && (PM25LV040_ID2 == 0x7f))      B_FlashOK = 1;
                                                                //检测是否为 PM25LVxx 系列的 Flash
    else if(PM25LV040_ID == 0x12)       B_FlashOK = 2;    //检测是否为 W25X4x 系列的 Flash
    else if(PM25LV040_ID == 0x13)       B_FlashOK = 3;    //检测是否为 W25X8x 系列的 Flash
    else if(PM25LV040_ID == 0x17)       B_FlashOK = 4;    //检测是否为 W25X128 系列的 Flash
    else                                B_FlashOK = 0;    //没有检测到指定型号的 Flash

    if(!B_FlashOK)    printf("未检测到指定型号的 Flash!\r\n");
```

```
    else
    {
        if(B_FlashOK == 1)
        {
            printf("检测到 PM25LV040!\r\n");
        }
        else if(B_FlashOK == 2)
        {
            printf("检测到 W25X40CL!\r\n");
        }
        else if(B_FlashOK == 3)
        {
            printf("检测到 W25Q80BV!\r\n");
        }
        else if(B_FlashOK == 4)
        {
            printf("检测到 W25Q128FV!\r\n");
        }
    }
}
/*************************************************
检测 Flash 的忙状态
入口参数: 无
出口参数:
    0 : Flash 处于空闲状态
    1 : Flash 处于忙状态
*************************************************/
u8 CheckFlashBusy(void)
{
    u8    dat;

    SPI_CE_Low();           //SPI 片选使能
    SPI_WriteByte(SFC_RDSR);      //发送读取状态命令
    dat = SPI_ReadByte();   //读取状态
    SPI_CE_High();          //SPI 片选禁能

    return (dat);           //状态值的 Bit0 即为忙标志
}

/*************************************************
使能 Flash 写命令
入口参数: 无
出口参数: 无
*************************************************/
void FlashWriteEnable(void)
{
```

```
    while(CheckFlashBusy() > 0);      //Flash 忙检测
    SPI_CE_Low();                     //SPI 片选使能
    SPI_WriteByte(SFC_WREN);          //发送写使能命令
    SPI_CE_High();                    //SPI 片选禁能
}

/*********************************************
擦除扇区, 一个扇区 4KB
入口参数: 无
出口参数: 无
*********************************************/
void FlashSectorErase(u32 addr)
{
    if(B_FlashOK)                     //判断是否检测到指定型号 Flash
    {
        FlashWriteEnable();           //使能 Flash 写命令
        SPI_CE_Low();                 //SPI 片选使能

        if(B_FlashOK == 1)            //判断 Flash 类型，不同型号擦除命令不同
        {
            SPI_WriteByte(SFC_SECTORER1);   //发送扇区擦除命令
        }
        else
        {
            SPI_WriteByte(SFC_SECTORER2);   //发送扇区擦除命令
        }

        SPI_WriteByte(((u8 *)&addr)[1]);          //设置起始地址
        SPI_WriteByte(((u8 *)&addr)[2]);
        SPI_WriteByte(((u8 *)&addr)[3]);

        SPI_CE_High();                            //SPI 片选禁能
    }
}

/*********************************************
从 Flash 中读取数据
入口参数:
    addr   : 地址参数
    buffer : 缓冲从 Flash 中读取的数据
    size   : 数据块大小
出口参数 : 无
*********************************************/
void SPI_Read_Nbytes(u32 addr, u8 *buffer, u16 size)
{
    if(size == 0)         return;         //操作长度为 0 则退出
```

```
    if(!B_FlashOK)     return;          //没检测到指定型号 Flash 则退出
    while(CheckFlashBusy() > 0);        //Flash 忙检测

    SPI_CE_Low();                       //SPI 片选使能
    SPI_WriteByte(SFC_READ);            //发送读命令

    SPI_WriteByte(((u8 *)&addr)[1]);        //设置起始地址
    SPI_WriteByte(((u8 *)&addr)[2]);
    SPI_WriteByte(((u8 *)&addr)[3]);

    do{
        *buffer = SPI_ReadByte();   //读取 1 个字节内容存入缓冲区
        buffer++;
        }while(--size);                 //连续读取指定个数内容
    SPI_CE_High();                      //SPI 片选禁能
}

/*************************************************
写数据到 Flash 中
入口参数:
    addr   : 地址参数
    buffer : 缓冲需要写入 Flash 的数据
    size   : 数据块大小
出口参数: 无
*************************************************/
void SPI_Write_Nbytes(u32 addr, u8 *buffer, u8 size)
{
    if(size == 0)       return;         //操作长度为 0 则退出
    if(!B_FlashOK)      return;         //没检测到指定型号 Flash 则退出
    while(CheckFlashBusy() > 0);        //Flash 忙检测

    FlashWriteEnable();                 //使能 Flash 写命令

    SPI_CE_Low();                           //SPI 片选使能
    SPI_WriteByte(SFC_PAGEPROG);            //发送页编程命令
    SPI_WriteByte(((u8 *)&addr)[1]);        //设置起始地址
    SPI_WriteByte(((u8 *)&addr)[2]);
    SPI_WriteByte(((u8 *)&addr)[3]);

    do{
        SPI_WriteByte(*buffer++);       //连续页内写
        addr++;
        if ((addr & 0xff) == 0) break;      //地址跨页则退出
        }while(--size);
    SPI_CE_High();                          //SPI 片选禁能
}
```

2.21.3.3 uart.c -- 初始化 UART 接口

```
//uart.c
//初始化 UART 接口

#include "config.h"             //头文件中已包含 stc8h.h 以及其他头文件

void Uart1_Init(void)           //115200bps@40.000MHz
{
    SCON = 0x50;                //8 位数据,可变波特率
    AUXR |= 0x40;               //定时器时钟 1T 模式
    AUXR &= 0xFE;               //串口 1 选择定时器 1 为波特率发生器
    TMOD &= 0x0F;               //设置定时器模式
    TL1 = 0xE8;                 //设置定时初始值
    TH1 = 0xFF;                 //设置定时初始值
    ET1 = 0;                    //禁止定时器中断
    TR1 = 1;                    //定时器 1 开始计时
    ES = 1;                     //使能串口 1 中断
}

void Uart1_Send(u8 dat)         //通过串口 1 发送数据
{
    SBUF = dat;                 //写入需要发送的数据
    while(!TI);                 //判断发送是否完成
    TI = 0;                     //清除发送完成标志
}

char putchar(char c)        //将 printf 函数映射到 UART1 接口
{
    Uart1_Send(c);      //调用串口 1 发送函数（如果要映射到串口 2 的话，这里修改为串口 2 发送函数）
    return c;
}
```

2.21.3.4 key.c -- 按键扫描以及按键事件处理

```
//key.c
//按键扫描以及按键事件处理

#include "config.h"                     //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

bit   Key1_Flag;
bit   Key2_Flag;
bit   Key2_Short_Flag;

bit   Key1_Function;
bit   Key2_Short_Function;
```

```
bit   Key2_Long_Function;

u16  Key1_cnt;
u16  Key2_cnt;

union LongType temp;

void KeyScan(void)
{
    //单纯短按按键
    if(!KEY1)
    {
        if(!Key1_Flag)      //判断按键是否已经有效触发，如果已经处于触发状态则不再检测，避免重复触发
        {
            Key1_cnt++;                 //按键计数器加 1
            if(Key1_cnt >= 50)          //50ms 防抖
            {
                Key1_Flag = 1;          //设置按键状态，防止重复触发
                Key1_Function = 1;      //设置按键操作标志
            }
        }
    }
    else
    {
        Key1_cnt = 0;       //按键计数器清零
        Key1_Flag = 0;      //按键释放
    }

    //长按短按按键
    if(!KEY2)
    {
        if(!Key2_Flag)      //判断按键是否已经有效触发，如果已经处于触发状态则不再检测，避免重复触发
        {
            Key2_cnt++;  //按键计数器加 1
            if(Key2_cnt >= 1000)            //长按 1s
            {
                Key2_Short_Flag = 0;        //清除短按标志
                Key2_Flag = 1;              //设置按键状态，防止重复触发
                Key2_Long_Function = 1;     //设置长按操作标志
            }
            else if(Key2_cnt >= 50)         //50ms 防抖
            {
                Key2_Short_Flag = 1;        //设置短按标志
            }
        }
    }
```

```
    else
    {
        if(Key2_Short_Flag)                 //判断是否短按
        {
            Key2_Short_Flag = 0;            //清除短按标志
            Key2_Short_Function = 1;        //设置短按操作标志
        }
        Key2_cnt = 0;       //按键计数器清零
        Key2_Flag = 0;      //按键释放
    }
}

void KeyEvents(void)
{
    u8 buf[4];

    if(Key1_Function)               //判断 KEY1 是否按下
    {
        Key1_Function = 0;      //清按键标志位
        FlashCheckID();         //检测 Flash 型号

        SPI_Read_Nbytes(FLASH_ADDR,temp.cdata,4);       //读取 4 字节内容
        printf("读取内容：0x%08lx\r\n",temp.ldata);          //打印读取内容，printf 打印 32 位数据需要加"l"
        temp.ldata++;                   //数值加 1
        //temp.ldata = 0x12345678;  //设置写入内容
    }

    if(Key2_Short_Function)             //判断 KEY2 是否短按
    {
        Key2_Short_Function = 0;    //清按键标志位

        FlashSectorErase(FLASH_ADDR);     //擦除 Flash
        printf("写入内容：0x%08lx\r\n",temp.ldata);          //打印写入内容，printf 打印 32 位数据需要加"l"
        SPI_Write_Nbytes(FLASH_ADDR,temp.cdata,4);      //写 4 个字节内容

        SPI_Read_Nbytes(FLASH_ADDR,buf,4);              //读取 4 字节内容
        printf("读取内容：0x%02x%02x%02x%02x\r\n",buf[0],buf[1],buf[2],buf[3]);       //打印读取内容

        temp.ldata++;                   //数值加 1
    }

    if(Key2_Long_Function)              //判断 KEY2 是否长按
    {
        Key2_Long_Function = 0;     //清按键标志位
        printf("擦除\xfd Flash\r\n");
        FlashSectorErase(FLASH_ADDR);     //擦除 Flash
```

```
            temp.ldata = 0;                 //重置写入内容
        }
}
```

2.21.3.5 timer.c -- 定时器初始化以及定时器中断服务程序

```
//timer.c
//定时器初始化以及定时器中断服务程序
//定时器 0:1ms 定时，产生 1ms 的标志位

#include "config.h"                 //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件

bit    f1ms;                        //1ms 标志位

void Timer0_Init(void)              //1 毫秒@11.0592MHz
{
    AUXR |= 0x80;                   //定时器 0 时钟 1T 模式
    TMOD &= 0xF0;                   //设置定时器 0 为模式 0
    TL0 = 0xCD;                     //设置定时器 0 初始值
    TH0 = 0xD4;                     //设置定时器 0 初始值
    TF0 = 0;                        //清除 TF0 标志
    TR0 = 1;                        //定时器 0 开始计时
    ET0 = 1;                        //使能定时器 0 中断

    f1ms = 0;                       //初始化 1m 标志位
}

void Timer0_Isr(void) interrupt 1   //定时器 0 中断服务程序
{
    f1ms = 1;                       //设置 1ms 标志位
}
```

2.21.3.6 config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__                //防止头文件被重复包含
#define __CONFIG_H__

#define HIRC            11059200UL
#define FOSC            11059200UL
#define SYSCLK          FOSC
#define MAIN_Fosc       FOSC

#include <stc8h.h>      //包含外部头文件
#include <stdio.h>
#include <intrins.h>
```

```
#include "def.h"                //包含项目头文件
#include "timer.h"
#include "key.h"
#include "spi.h"
#include "uart.h"

void SYS_Init(void);            //函数声明

#endif
```

2.21.3.7 spi.h -- 项目配置文件的头文件

```
#ifndef __SPI_H__               //防止头文件被重复包含
#define __SPI_H__

sbit    SPI_CE   = P2^2;        //PIN1
sbit    SPI_SO   = P2^4;        //PIN2
sbit    SPI_SI   = P2^3;        //PIN5
sbit    SPI_SCK = P2^5;         //PIN6

#define SFC_WREN            0x06    //串行 Flash 命令集
#define SFC_WRDI            0x04
#define SFC_RDSR            0x05
#define SFC_WRSR            0x01
#define SFC_READ            0x03
#define SFC_FASTREAD        0x0B
#define SFC_RDID            0xAB
#define SFC_PAGEPROG        0x02
#define SFC_RDCR            0xA1
#define SFC_WRCR            0xF1
#define SFC_SECTORER1       0xD7    //PM25LV040  扇区擦除指令
#define SFC_SECTORER2       0x20    //W25Xxx  扇区擦除指令
#define SFC_BLOCKER         0xD8
#define SFC_CHIPER          0xC7

#define FLASH_ADDR          0x00

#define SPI_CE_High()       SPI_CE = 1    // set CE high
#define SPI_CE_Low()        SPI_CE = 0    // clear CE low

void SPI_Init();                    //函数声明
void FlashCheckID(void);            //函数声明
void FlashSectorErase(u32 addr);
void SPI_Read_Nbytes( u32 addr, u8 *buffer, u16 size);
void SPI_Write_Nbytes(u32 addr, u8 *buffer,    u8 size);
```

```
#endif
```

2.21.3.8 uart.h -- 项目配置文件的头文件

```
#ifndef __UART_H__          //防止头文件被重复包含
#define __UART_H__

void Uart1_Init();          //函数声明
void Uart1_Send(u8 dat);    //函数声明

#endif
```

2.21.3.9 key.h -- 项目配置文件的头文件

```
#ifndef __KEY_H__           //防止头文件被重复包含
#define __KEY_H__

sbit KEY1 = P3^2;
sbit KEY2 = P3^3;

union LongType
{
    u32 ldata;
    u8 cdata[4];
};

extern union LongType temp;

void KeyScan ();            //函数声明
void KeyEvents();           //函数声明

#endif
```

2.21.3.10 timer.h -- 项目配置文件的头文件

```
#ifndef __TIMER_H__         //防止头文件被重复包含
#define __TIMER_H__

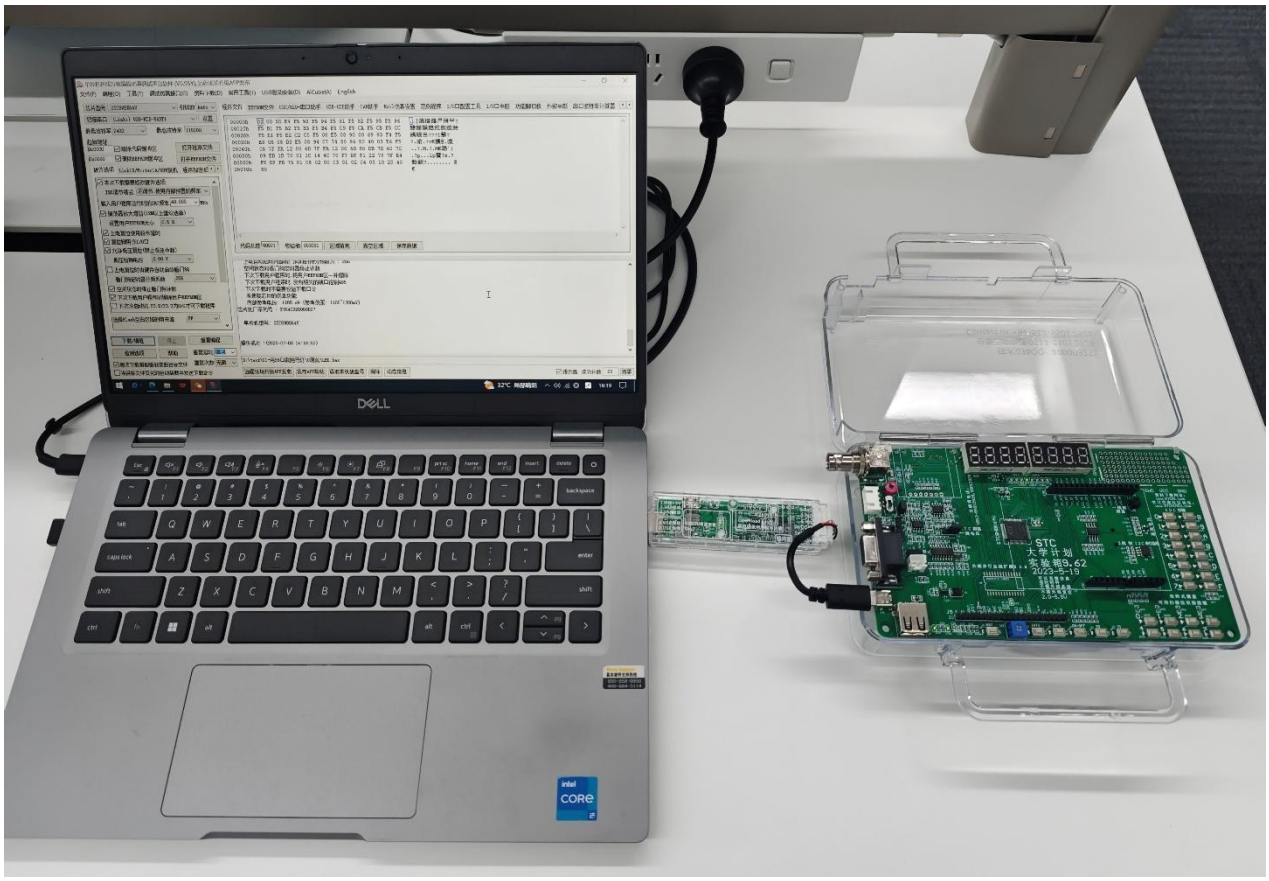
void Timer0_Init(void);     //函数声明

extern bit f1ms;            //外部变量声明

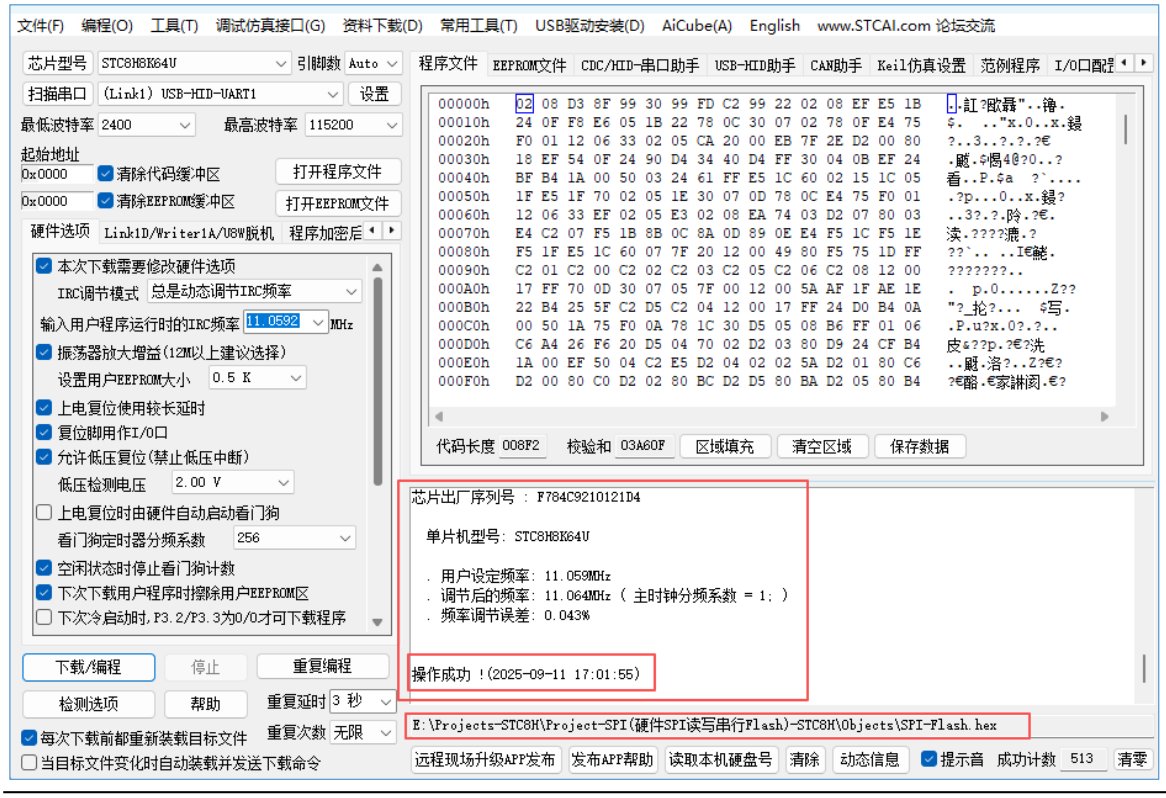
#endif
```

2.21.4　程序下载，CDC 串口助手相关设置

录入代码，保存，编译。用 USB 转串口/TTL - TypeC 接口线连接 USB Link 1D 和 STC8H8K64U 实验箱，如下图：



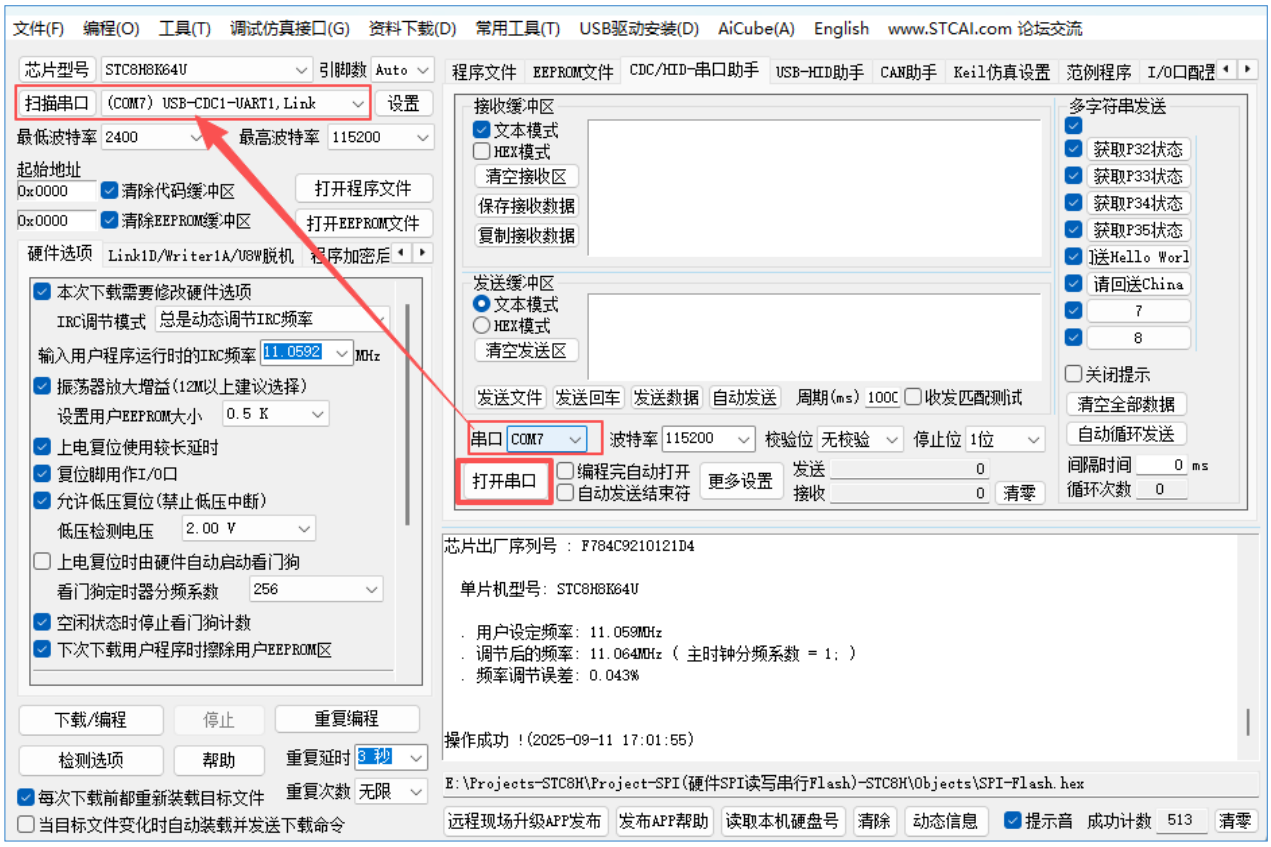
运行 AiCube-ISP 软件系统，按步骤打开并下载程序“SPI-Flash.hex”，如下图：



在 AiCube-ISP 软件系统界面，点击“CDC/HID-串口助手”选项卡：

➲ 串口选择 CDC1 对应的 COM 口

➲ 找到“打开串口”按钮，如下图：



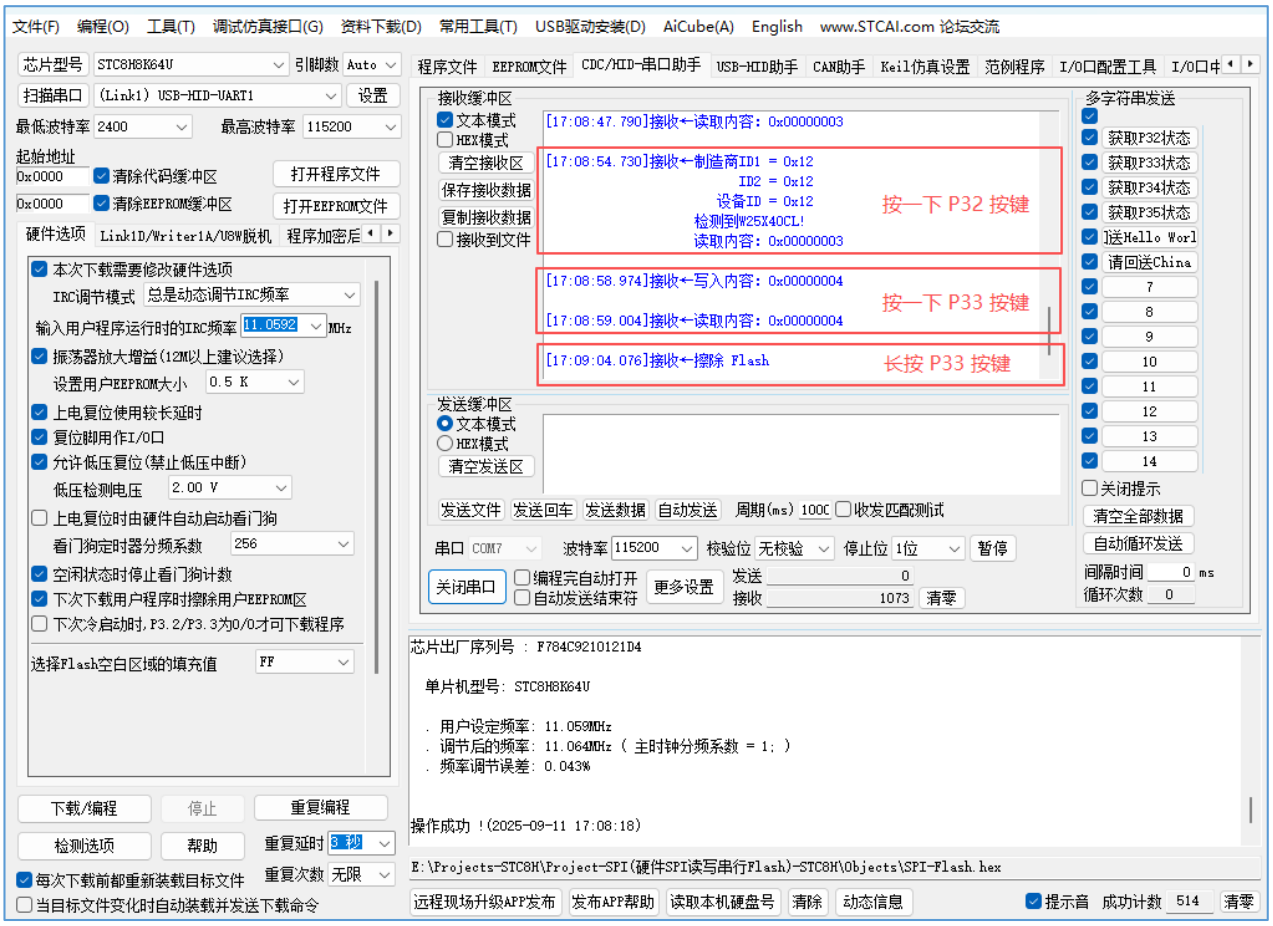
一切准备就绪！

2.21.5　观察实验现象，验证实验效果

在上述 AiCube-ISP 软件系统界面：

- 点击“打开串口”按钮
- 按一下 P32 按键：读取串行 Flash ID，以及 Flash 指定地址数据。
- 按一下 P33 按键（短按）：擦除指定地址 Flash 扇区，并写入计数值，然后读出，通过串口打印操作结果。
- 长按 P33 按键：擦除指定地址 Flash 扇区，重置计数值。

观察接收缓冲区，如下图：



完成实验验证。

2.22 SPI 实验 -- 硬件 SPI 驱动 OLED 显示图片、图形、字符

2.22.1 实验介绍

1、本实验项目主要目的：

- 硬件 SPI 驱动 OLED 显示
- 如何实现显示图片
- 如何实现 6*8 点阵字符、8*16 点阵字符、16*16 点阵汉字
- 如何画直线、方框、圆

2、掌握 STC8H8K64U 实验箱原理图中 SPI 驱动 OLED 电路

3、了解图片取模的原理，并熟练使用 AiCube-ISP 系统软件中图片取模工具

4、了解字库生成原理，并熟练使用 AiCube-ISP 系统软件中字库生成工具

5、使用 AiCube-ISP 系统软件图片取模工具生成 ASCII 的点阵字符 C 代码

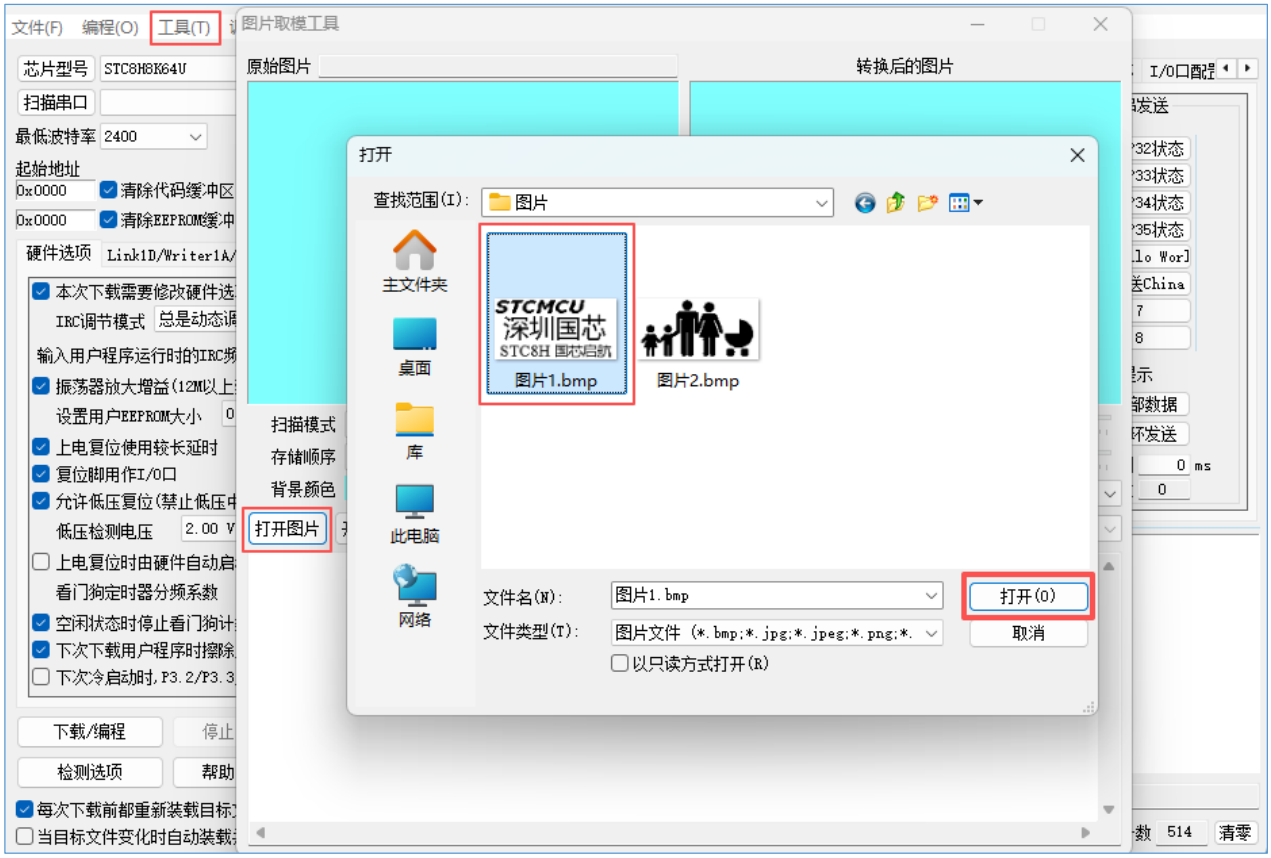
6、继续熟悉如何管理多文件项目

2.22.1.1 AiCube-ISP 系统软件图片取模工具的使用

打开 AiCube-ISP 系统软件：

- 点击菜单栏“工具”→“图片取模工具”，打开“图片取模工具”浮窗
- 点击“打开图片”按钮，选择准备要取模的图片
- 点击“打开”按钮

如下图：



此时，在打开的“图片取模工具”浮窗，我们看到被选择的图片显示在原始图片区和转换后的图片区：

- 扫描模式：选择“垂直扫描，从左到右，从上到下”
- 存储顺序：选择“高位在前”
- 数据格式：选择“C 表格格式”
- 勾选“图片反色”
- 设置：宽度：128；高度：64；输出格式：1 位单色
- 点击“开始转换”

就会生成相应的点阵图片 C 代码，如下图：



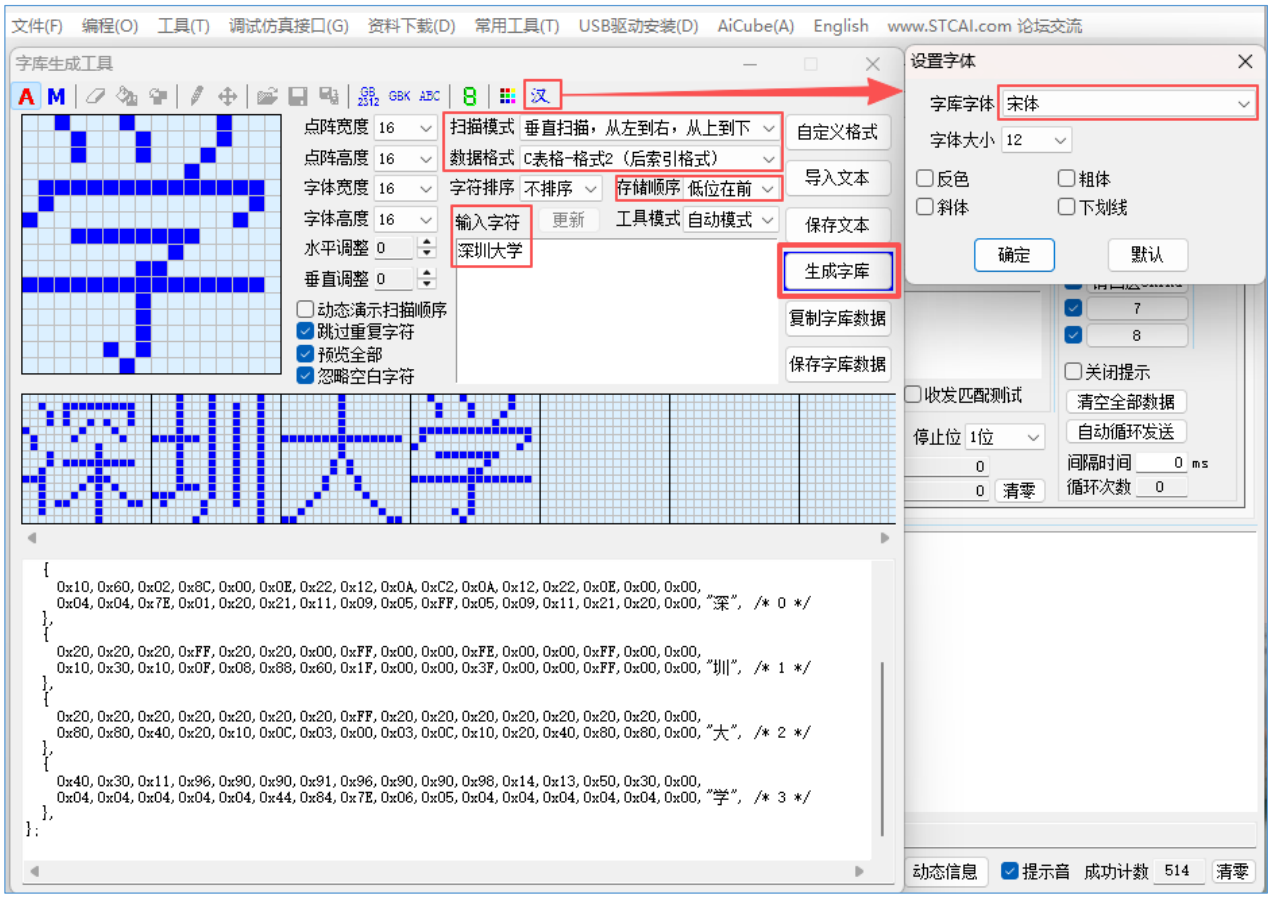
将点阵图片生成的 C 代码数据拷贝出来，本例是将此段代码放在“pic.h”中，形成表格数据，方便程序加载并在 OLED 上显示出来。

2.22.1.2　AiCube-ISP 系统软件字库生成工具的使用

打开 AiCube-ISP 系统软件，点击菜单栏“工具”→“字库生成工具”，打开“字库生成工具”浮窗：

- 在“输入字符”区输入想要取模的文字，本例输入“深圳大学”
- 点击第一行的工具栏上的“汉”按钮，打开“设置字体”浮窗，字库字体选择“宋体”
- 扫描模式：选择“垂直扫描，从左到右，从上到下”
- 数据格式：选择“C 表格-格式 2（后索引模式）”
- 存储顺序：选择“低位在前”
- 点击“生成字库”

就会生成相应的点阵字库代码，如下图：



将点阵字库生成的 C 代码拷贝出来，本例是将此段代码放在“font.h”中，形成表格数据，方便程序查询并提取字库。

2.22.1.3　使用 AiCube-ISP 系统软件图片取模工具生成 ASCII 的点阵字符 C 代码

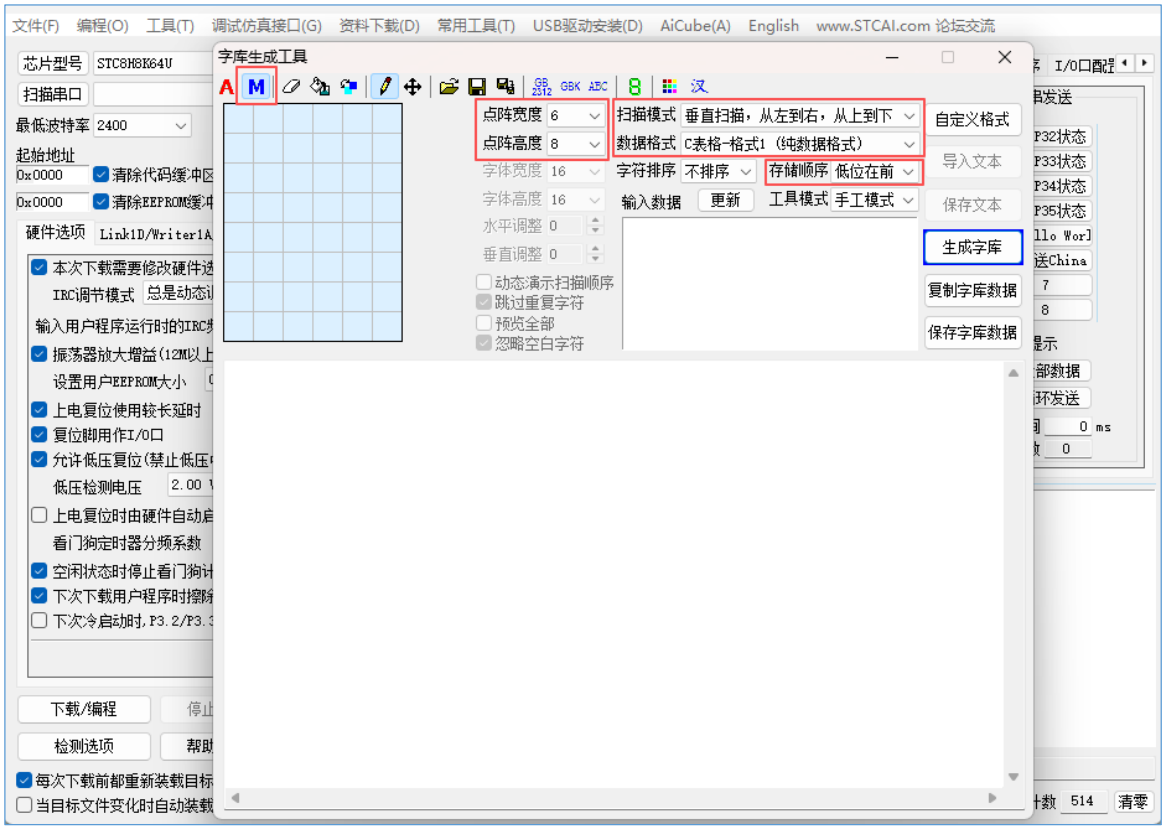
在 AiCube-ISP 系统软件的字库生成工具中，我们除了可以像上述操作步骤生成标准点阵字符 C 代码，还可以自己手动描点生成自己想要的点阵字符 C 代码。

打开 AiCube-ISP 系统软件，点击菜单栏“工具”→“字库生成工具”，打开“字库生成工具”浮窗。

先进行相应的设置：

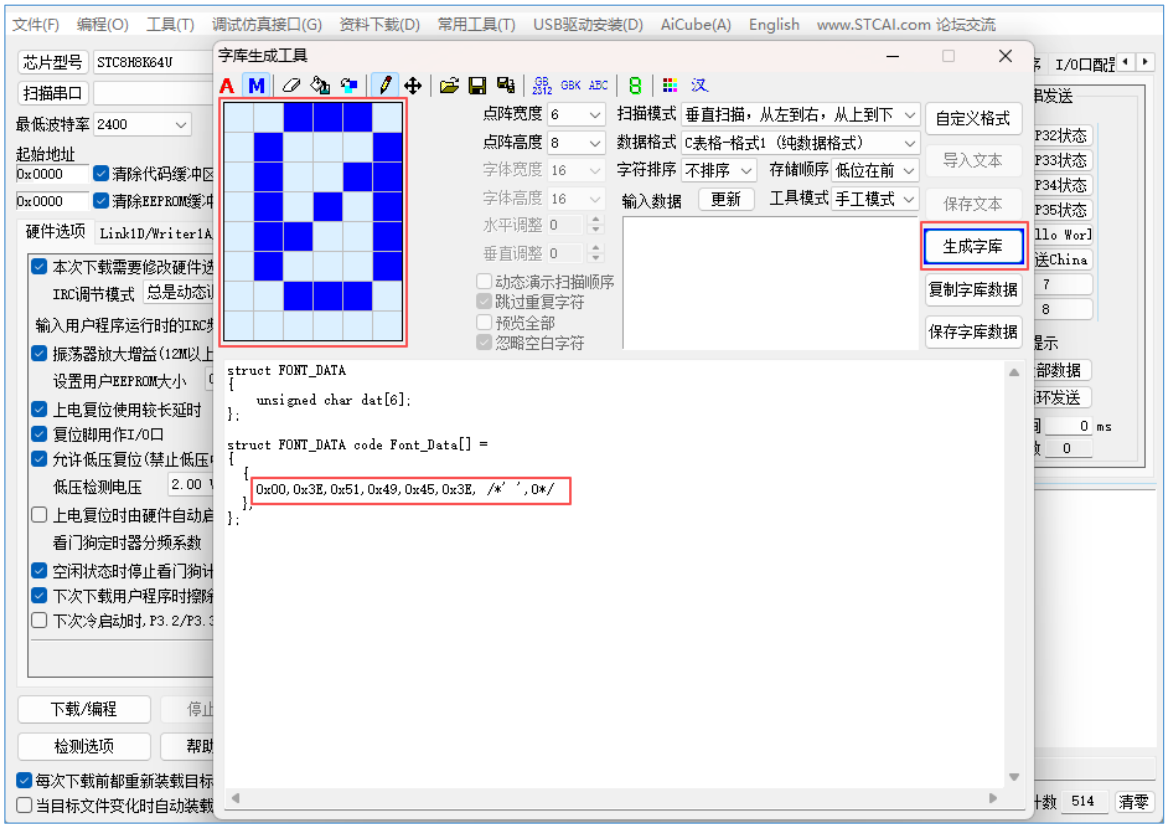
- 点击“字库生成工具”浮窗第一行的“M”按钮，清空字符点阵区域
- 确定点阵宽度：6；点阵高度：8（本例中有 6*8 的点阵 ASCII 字符 C 代码数据和 8*16 的点阵 ASCII 字符代码数据）
- 扫描模式：选择“垂直扫描，从左到右，从上到下”
- 数据格式：选择“C 表格-格式 1（纯数据格式）”
- 存储顺序：选择“低位在前”

如下图：



现在手动用鼠标在窗口左上角点阵描图区域，进行描图。

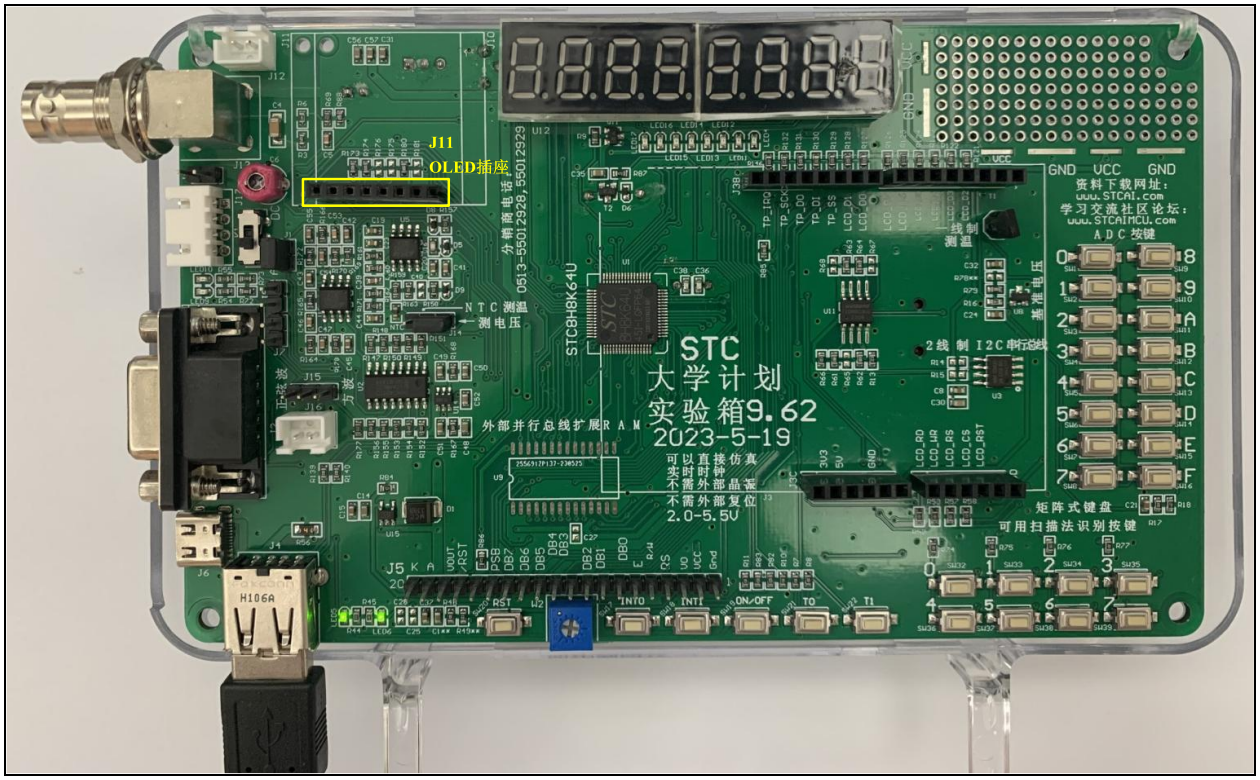
以 ASCII 码的“0”为例，进行如下图中的描绘，再点击“生成字库”按钮，即可生成相应的字库 C 代码数据。以此类推，生成自己想要的全部 ASCII 码的点阵字库 C 代码数据。



将点阵字库生成的 C 代码拷贝出来，本例是将此段代码放在“ascii.h”中，形成表格数据，方便程序查询并提取字库。

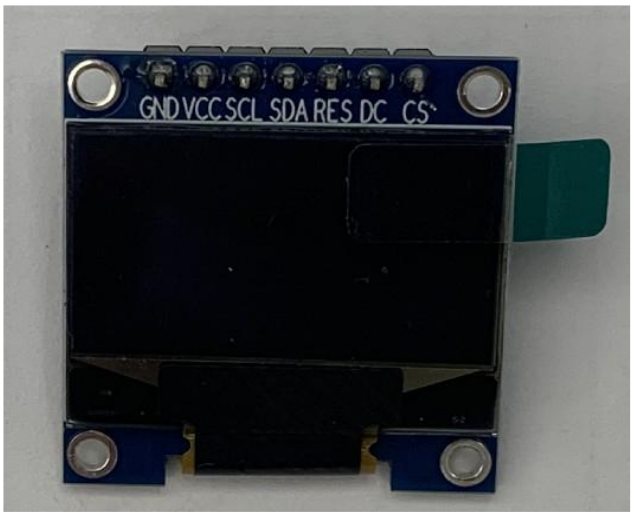
2.22.1.4 认识 STC8H8K64U 实验箱和 OLED 屏

STC8H8K64U 实验箱：

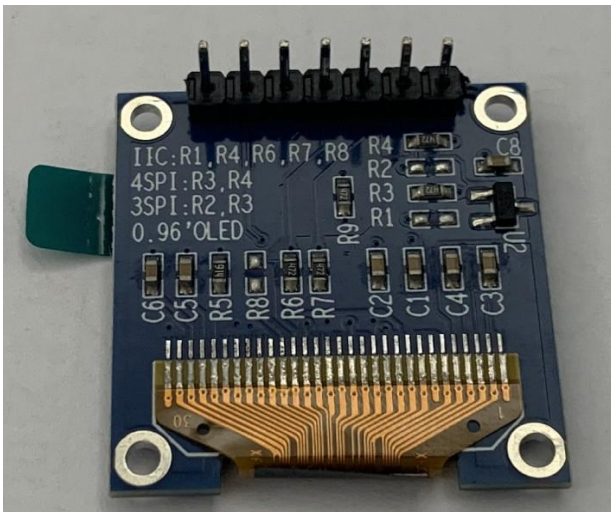


J11：OLED 插座。STC8H8K64U 实验箱上的 J11 插座可插 7 线或 8 线的 OLED 屏，本实验使用的 OLED 屏是 7 线，第 8 脚浮空。详见 STC8H8K64U 实验箱中 J11 原理图。

OLED 屏



OLED正面



OLED背面

2.22.1.5 Keil 环境下多文件项目管理说明

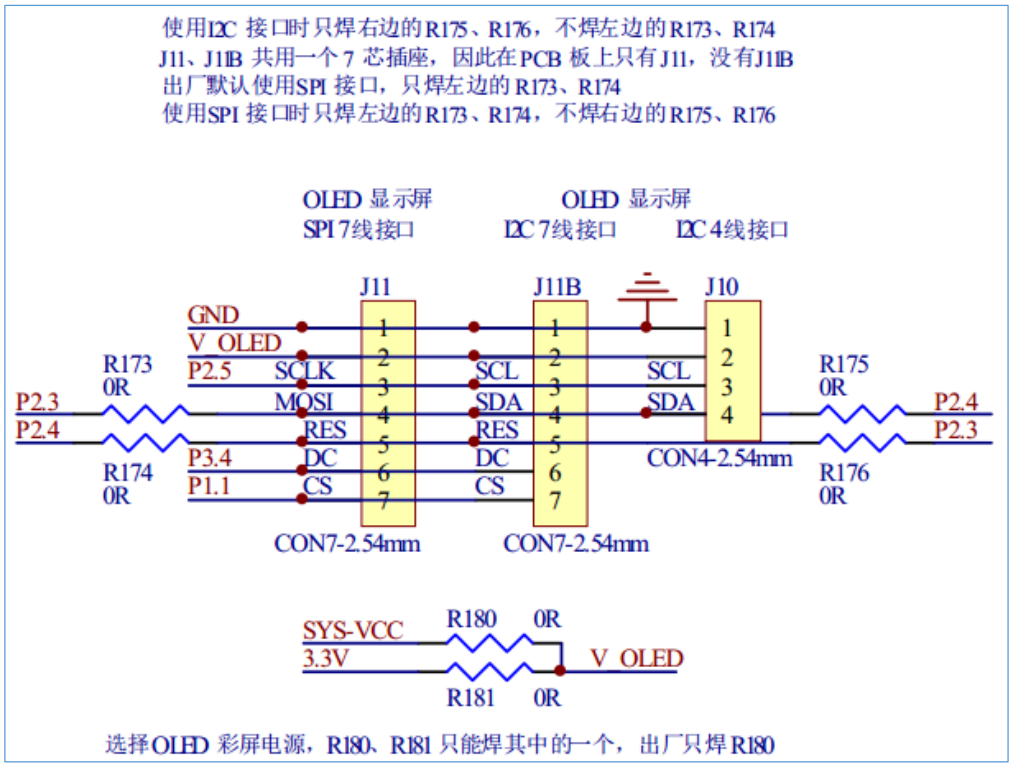
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以**方便分工合作、代码复用、模块化管理、增强可读性和可维护性**。

- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。

- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，可避免在同一个.c 文件中对同一个.h 头文件进行多次包含：

```
#ifndef XXXX
#define XXXX
...
#endif
```

2.22.2 原理图



2.22.3 实验程序代码

2.22.3.1 main.c -- 程序主函数

```
//main.c
//程序主函数
//SPI 驱动 OLED 显示图片、图形、字符

#include "config.h"        //头文件中已包含 stc8h.h, ai_usb.h 以及其他头文件
#include "pic.h"
```

```
void delay_ms(unsigned int ms);

void main()
{
    SYS_Init();             //系统初始化
    OLED_Init();            //初始化
    OLED_CLS();             //清屏
//  OLED_Light(0);          //最低亮度
//  OLED_Light(0xFF);       //最高亮度
    OLED_Light(0x80);       //中间亮度

    while (1)
    {
        OLED_BMP(0,0,128,8,OLED12864_IMG0[0],1);     //显式 BMP 图
        delay_ms(2000);

        OLED_BMP(0,0,128,8,OLED12864_IMG1[0],0);
        OLED_String(0,0,"AiMCU",0,1);                //图上叠加文字...
        delay_ms(2000);

        OLED_CLS();     //清屏
        OLED_String(5,1,"深圳大学+STC",0,1);         //中西文混合显示
        OLED_String(5,3,"深圳大学+STC",1,1);         //正显/反显
        OLED_Str5x7(20,6,"STC8H8K64U",0,1);  //正显
        OLED_Str5x7(20,7,"STC8H8K64U",1,1);  //反显
        delay_ms(2000);

        OLED_CLS();     //清屏
        OLED_LineH(10,10,108,0);        //画一水平线 作图算法...
        OLED_LineV(10,10,44,0);         //画一垂直线
        OLED_Line(10,10,118,54,0);      //两点一线 不同方向画线...
        OLED_Line(10,54,118,10,0);      //两点一线
        OLED_Line(118,54,118,10,0);     //两点一线
        OLED_Line(118,54,10,54,0);      //两点一线
        OLED_Box(0,0,127,63,1);         //画一方框
        OLED_Circle(64,32,30,1);        //画一个圆
        delay_ms(2000);
    }
}

void SYS_Init(void)     //系统初始化函数
{
    P_SW2 |= 0x80;                      //允许访问扩展的特殊寄存器，XFR

    P0M1 = 0x00;    P0M0 = 0x00;        //设置为准双向口
    P1M1 = 0x00;    P1M0 = 0x02;        //设置为准双向口, P1.1 设置为推挽输出
```

```
    P2M1 = 0x00;    P2M0 = 0x38;        //设置为准双向口, P2.3,P2.4,P2.5 设置为推挽输出
    P3M1 = 0x00;    P3M0 = 0x10;        //设置为准双向口, P3.4 设置为推挽输出
    P4M1 = 0x00;    P4M0 = 0x00;        //设置为准双向口
    P5M1 = 0x00;    P5M0 = 0x00;        //设置为准双向口
    P6M1 = 0x00;    P6M0 = 0x00;        //设置为准双向口
    P7M1 = 0x00;    P7M0 = 0x00;        //设置为准双向口

    P2SR = 0x00;        //电平转换速度快（改善 IO 口高速翻转信号）
    P2DR = 0xff;        //端口驱动电流增强//   EA = 1;   //使能全局中断
}

//========================================================================
// 函数: void delay_ms(unsigned int ms)
// 描述: 毫秒级延时函数。
// 参数: ms,要延时的 ms 数,自动适应主时钟.
//========================================================================
void delay_ms(unsigned int ms)
{
    unsigned int i;
    do{
        i = MAIN_Fosc / 6000;
        while(--i);
    } while(--ms);
}
```

2.22.3.2 oled.c -- SPI 驱动 OLED 显示图片、图形、字符

```
//oled.c
//SPI 驱动 OLED 显示图片、图形、字符

#include "config.h"       //头文件中已包含 stc8h.h 以及其他头文件
#include "font.h"
#include "ascii.h"

#define   UseSPI   1     //0: IO 模拟 SPI;   1: 使用硬件 SPI

//OLED 屏引脚定义
sbit      OLED_SCK  =   P2^5;    //D0      SPI or IIC 的时钟脚
sbit      OLED_SDA  =   P2^3;    //D1      SPI or IIC 的数据脚
sbit      OLED_RES  =   P2^4;    //RES     复位脚, 低电平复位
sbit      OLED_DC   =   P3^4;    //DC      数据或命令脚
sbit      OLED_CS   =   P1^1;    //CS      片选脚

//SPI 定义
sbit      SPI_SCK   =   P2^5;    //PIN3
sbit      SPI_MOSI  =   P2^3;    //PIN4
```

```
unsigned char xdata ShowBuf[8][128];        //OLED 全局缓存

extern void delay_ms(unsigned int ms);

//==========================================================================
// 函数名称: OLED_WR_CMD(unsigned char Cmd)
// 函数功能: OLED_写命令字
//==========================================================================
void OLED_WR_CMD(unsigned char Cmd)
{
#if(UseSPI==0)
    unsigned char i;
#endif

    OLED_SCK= 0;    //时钟低;
    OLED_DC = 0;    //写命令
    OLED_CS = 0;    //片选中

#if(UseSPI)
    SPDAT = Cmd;
    while((SPSTAT & SPIF) == 0);
    SPSTAT |= SPIF | WCOL;    //清 SPIF/WCOL 标志
#else
    for(i=0;i<8;i++)
    {
        if(Cmd&0x80)
            OLED_SDA=1;
        else
            OLED_SDA=0;

        Cmd <<= 1;          //左移位 //
        OLED_SCK = 1;
        OLED_SCK = 0;
    }
#endif

    OLED_CS = 1;            //片不选
}

//==========================================================================
// 函数名称: OLED_WR_DAT(unsigned char Dat)
// 函数功能: OLED_写数据字
//==========================================================================
void OLED_WR_DAT(unsigned char Dat)
{
#if(UseSPI==0)
```

```
    unsigned char i;
#endif

    OLED_SCK= 0;    //时钟低;
    OLED_DC = 1;    //写数据
    OLED_CS = 0;    //片选中

#if(UseSPI)
    SPDAT = Dat;
    while((SPSTAT & SPIF) == 0);
    SPSTAT |= SPIF | WCOL;    //清 SPIF/WCOL 标志
#else
    for(i=0;i<8;i++)
    {
        if(Dat&0x80)
            OLED_SDA=1;
        else
            OLED_SDA=0;

        Dat <<= 1;          //左移位
        OLED_SCK = 1;
        OLED_SCK = 0;
    }
#endif

    OLED_CS = 1;    //片不选
}

//==============================================================================
// 函数名称: OLED_Disp(unsigned char OnOff)
// 函数功能: OLED 开关显示 参数: 1:开 0:关
//==============================================================================
void OLED_Disp(unsigned char On)    //开关显示
{
    if(On==1)
    {
        OLED_WR_CMD(0x8D);    //电荷泵使能
        OLED_WR_CMD(0x14);    //开启电荷泵
        OLED_WR_CMD(0xAF);    //点亮屏幕
    }
    else
    {
        OLED_WR_CMD(0x8D);    //电荷泵使能
        OLED_WR_CMD(0x10);    //关闭电荷泵
        OLED_WR_CMD(0xAF);    //关闭屏幕
    }
```

```
}

//==========================================================================
// 函数名称: OLED_Init()  显示屏初始化函数
//==========================================================================
void OLED_Init(void)
{
#if(UseSPI)
    P_SW1 = (P_SW1 & ~0x0c) | 0x04;    //SPI 通道选择, 00: P1.2 P1.3 P1.4 P1.5,  01: P2.2 P2.3 P2.4 P2.5,
                                       //            10: P5.4 P4.0 P4.1 P4.3,  11: P3.5 P3.4 P3.3 P3.2

    SPCTL = (SPCTL & ~3) | 1;          //SPI 时钟频率选择, 0: 4T, 1: 8T,  2: 16T,  3: 2T
    SPCTL |= SSIG | SPEN | MSTR;       //允许 SPI，设为主机,忽略 SS 脚

    SPI_SCK  = 0;                      //时钟信号置低
    SPI_MOSI = 1;
    SPSTAT |= SPIF | WCOL;             //清 SPIF/WCOL 标志
#endif
    OLED_SCK = 0;                      //时钟端低
    OLED_SDA = 1; delay_ms(100);       //数据端高
    OLED_RES = 1; delay_ms(200);
    OLED_RES = 0; delay_ms(200);       //复位
    OLED_RES = 1;

    OLED_WR_CMD(0xAE);  //--turn off oled panel
    OLED_WR_CMD(0x00);  //--set low column address
    OLED_WR_CMD(0x10);  //--set high column address
    OLED_WR_CMD(0x40);  //--set start line address  Set Mapping RAM Display Start Line (0x00~0x3F)
    OLED_WR_CMD(0x81);  //--set contrast control register
    OLED_WR_CMD(0xCF);  //--Set SEG Output Current Brightness  亮度...
    OLED_WR_CMD(0xA1);  //--Set SEG/Column Mapping      0xA0 左右反转  0xA1 正常
    OLED_WR_CMD(0xC8);  //--Set COM/Row Scan Direction 0xC0 上下反转  0xC8 正常
    OLED_WR_CMD(0xA6);  //--set normal display
    OLED_WR_CMD(0xA8);  //--set multiplex ratio(1 to 64)
    OLED_WR_CMD(0x3f);  //--1/64 duty
    OLED_WR_CMD(0xD3);  //--set display offsetShift Mapping RAM Counter (0x00~0x3F)
    OLED_WR_CMD(0x00);  //--not ofset
    OLED_WR_CMD(0xd5);  //--set display clock divide ratio/oscillator frequency
    OLED_WR_CMD(0x80);  //--set divide ratio, Set Clock as 100 Frames/Sec
    OLED_WR_CMD(0xD9);  //--set pre-charge period
    OLED_WR_CMD(0xF1);  //--Set Pre-Charge as 15 Clocks & Discharge as 1 Clock
    OLED_WR_CMD(0xDA);  //--set com pins hardware configuration
    OLED_WR_CMD(0x12);
    OLED_WR_CMD(0xDB);  //--set vcomh
    OLED_WR_CMD(0x40);  //--Set VCOM Deselect Level
    OLED_WR_CMD(0x20);  //--Set Page Addressing Mode (0x00/0x01/0x02)
```

```
    OLED_WR_CMD(0x00);
    OLED_WR_CMD(0x8D);    //--set Charge Pump enable/disable
    OLED_WR_CMD(0x14);    //--set(0x10) disable
    OLED_WR_CMD(0xA4);    //--Disable Entire Display On (0xa4/0xa5)
    OLED_WR_CMD(0xA6);    //--Disable Inverse Display On (0xa6/a7)
    OLED_WR_CMD(0xAF);
    OLED_Disp(1);                //开显示
}

//==============================================================================
// 函数名称: OLED_SetXY
// 函数功能: OLED 设置显示位置
// 入口参数: X-水平坐标 0..127,Y-垂直坐标 0..63
//==============================================================================
void OLED_SetXY(unsigned char X, unsigned char Y)
{
    OLED_WR_CMD((u8)(0xB0+Y));
    OLED_WR_CMD((u8)(((0xF0&X)>>4)|0x10));
    OLED_WR_CMD((u8)(((0x0f&X)|0x01)));
}

//==============================================================================
// 函数名称: OLED_Show
// 函数功能: OLED 刷新显示(将显示缓存的数据刷到显示屏显示出来)
//==============================================================================
void OLED_Show(void)
{
    unsigned char xdata i,n;

    for(i=0;i<8;i++)                    //8 行(页)循环...
    {
        OLED_WR_CMD((u8)(0xB0+i));      //设置行起始地址
        OLED_WR_CMD(0x00);              //设置低列起始地址
        OLED_WR_CMD(0x10);              //设置高列起始地址

        for(n=0;n<128;n++)                  //每行(页)128 点(列)
            OLED_WR_DAT(ShowBuf[i][n]);     //写数据...
    }
}

//==============================================================================
// 函数名称: OLED_CLS 清屏函数
// 函数功能: OLED 刷新显示(将显示缓存的 0 数据刷到显示屏显示出来)
//==============================================================================
void OLED_CLS(void)
{
```

```
    memset(ShowBuf,0,128*8);        //清空缓存
    OLED_Show();                    //刷新显示
}

//==============================================================================
// 函数名称: OLED_Light(unsigned char num)
// 函数功能: OLED 亮度级设置
//==============================================================================
void OLED_Light(unsigned char Level)
{
    OLED_WR_CMD(0x81);
    OLED_WR_CMD(Level);
    OLED_WR_CMD(0xDB);          //--set vcomh
    OLED_WR_CMD(0x20);          //Set VCOM Deselect Level
}

//==============================================================================
// 函数名称: OLED_Point
// 函数功能: OLED 显示一个点
// 入口参数: X：水平点 0..127 Y:垂直点 0..63
//==============================================================================
void OLED_Point(unsigned char X,unsigned char Y,u8 mode)
{
    ShowBuf[Y/8][X] |= 1<<(Y%8);        //垂直 8 点,高位在下,低位在上.
    if(mode)
        OLED_Show();
}

//==============================================================================
// 函数名称: OLED_BMP
// 函数功能: OLED 缓存写入 BMP 格式图片
// 入口参数: X：水平起点 0..127 Y:垂直起点 0..63  W: 宽度 H:高度 *BMP 图片数组
//==============================================================================
void OLED_BMP(unsigned char X, unsigned char Y,unsigned char W, unsigned char H,unsigned char BMP[],u8
mode)
{
    unsigned int xdata num=0;
    unsigned char i,l;

    for( i=0;i<H;i++ )
    {
        for(l=0;l<W;l++)
        {
            ShowBuf[Y+i][X+l] = BMP[num++];
        }
    }
```

```
    if(mode)
    {
        OLED_Show();
    }
}

//==============================================================================
// 画一条水平线
//==============================================================================
void OLED_LineH(unsigned char X,unsigned char Y,unsigned char W,u8 mode)
{
    unsigned char i;

    for(i=0;i<W;i++)
    {
        OLED_Point((u8)(X+i),Y,0);
    }

    if(mode)
    {
        OLED_Show();
    }
}

//==============================================================================
// 画一条垂直线
//==============================================================================
void OLED_LineV(unsigned char X,unsigned char Y,unsigned char H,u8 mode)
{
    unsigned char i;

    for(i=0;i<H;i++)
    {
        OLED_Point(X,(u8)(Y+i),0);
    }

    if(mode)
    {
        OLED_Show();
    }
}
//==============================================================================
// 函数名称: OLED_BuffShowPoint
// 函数功能: OLED 显示一条线
// 入口参数: X0 起点  Y0 起点  X1 终点  Y1 终点
```

```
//==========================================================================
void OLED_Line( unsigned char X0, unsigned char Y0, unsigned char X1,unsigned char Y1,u8 mode)
{
    unsigned char x,y;
    if(X0>X1)   //为从左到右而交换坐标.
    {
        x=X0;X0=X1;X1=x;
        y=Y0;Y0=Y1;Y1=y;
    }

    if(X0!=X1)
    {
        for(x=0;x<(X1-X0);x++ )
        {
            if(Y1>Y0)
                OLED_Point((u8)(X0+x),(u8)(Y0+(u16)(Y1-Y0)*(u16)x/(u16)(X1-X0)),0);
            else
                OLED_Point((u8)(X0+x),(u8)(Y0-(u16)(Y0-Y1)*(u16)x/(u16)(X1-X0)),0);
        }
    }
    else
    {
        if(Y0>Y1)
        {
            for(y=Y1; y<=Y0; y++ )
                OLED_Point(X0,y,0);
        }
        else
        {
            for(y=Y0; y<=Y1; y++ )
                OLED_Point(X0,y,0);
        }
    }

    if(mode)
    {
        OLED_Show();
    }
}
//==========================================================================
// 画一个方框
//==========================================================================
void OLED_Box(unsigned char X,unsigned char Y,unsigned char W,unsigned char H,u8 mode)
{
    OLED_LineH(X,Y,W,0);
    OLED_LineV(X,Y,H,0);
```

```
    OLED_LineH(X,(u8)(Y+H),W,0);
    OLED_LineV((u8)(X+W),Y,H,0);

    if(mode)
    {
        OLED_Show();
    }
}
//==========================================================================
// 函数名称: OLED_Circle
// 函数功能: OLED 显示一个圆形
// 入口参数: X 点   Y 点   r:半径
//         turn: 0:正显示  1:反显示   mode: 1:立即刷新显示  0:仅写缓存
//==========================================================================
void OLED_Circle(unsigned char X,unsigned char Y,unsigned char r,u8 mode)
{
    int a, b, di;
    a = 0;
    b = r;
    di = 3 - (r << 1);          //判断下个点位置的标志

    while (a <= b)
    {
        OLED_Point((u8)(X+a),(u8)(Y-b),0);          //5
        OLED_Point((u8)(X+b),(u8)(Y-a),0);          //0
        OLED_Point((u8)(X+b),(u8)(Y+a),0);          //4
        OLED_Point((u8)(X+a),(u8)(Y+b),0);          //6
        OLED_Point((u8)(X-a),(u8)(Y+b),0);          //1
        OLED_Point((u8)(X-b),(u8)(Y+a),0);          //3
        OLED_Point((u8)(X-a),(u8)(Y-b),0);          //2
        OLED_Point((u8)(X-b),(u8)(Y-a),0);          //7
        a++;

        //使用 Bresenham 算法画圆
        if (di < 0)
        {
            di += 4 * a + 6;
        }
        else
        {
            di += 10 + 4 * (a - b);
            b--;
        }
    }

    if(mode)
```

```
    {
        OLED_Show();
    }
}
//==========================================================================
// 函数名称: OLED_A08
// 函数功能: OLED 显示一个 6*8 字符
// 入口参数: X: 水平坐标列 0..127  Y: 垂直坐标行(页) 0..7
//       turn: 0:正显示 1:反显示  mode: 1:立即刷新显示 0:仅写缓存
//==========================================================================
void OLED_A08(unsigned char X,unsigned char Y,char Ch,u8 turn,u8 mode)      //OLED 显示一个 5*7 字符
{
    unsigned char i;
    for( i=0;i<6;i++ )
    {
        if(turn==0)
            ShowBuf[Y][X+i]= ASC8[Ch-' '][i];
        else
            ShowBuf[Y][X+i]=~ASC8[Ch-' '][i];
    }

    if(mode)
    {
        OLED_Show();
    }
}
//==========================================================================
// 函数名称: OLED_A16
// 函数功能: OLED 显示一个 8*16 字符
// 入口参数: X: 水平坐标列 0..127  Y: 垂直坐标行(页) 0..7
//       turn: 0:正显示 1:反显示  mode: 1:立即刷新显示 0:仅写缓存
//==========================================================================
void OLED_A16(unsigned char X,unsigned char Y,char Ch,u8 turn,u8 mode)      //OLED 显示一个 8*16 字
符
{
    unsigned char i,j;
    for( j=0;j<2;j++ )
    {
        for( i=0;i<8;i++ )
        {
            if(turn==0)
                ShowBuf[Y+j][X+i]= ASC16[Ch-' '][i+j*8];
            else
                ShowBuf[Y+j][X+i]=~ASC16[Ch-' '][i+j*8];
        }
    }
```

```
    if(mode)
    {
        OLED_Show();
    }
}
//==============================================================================
// 函数名称: OLED_HZ16
// 函数功能: OLED 显示一个 16*16 汉字
// 入口参数: X: 水平坐标列 0..127  Y: 垂直坐标行(页) 0..7
//       turn: 0:正显示 1:反显示  mode: 1:立即刷新显示 0:仅写缓存
//==============================================================================
void OLED_HZ16(unsigned char X,unsigned char Y,char *Hz,u8 turn,u8 mode)
{
    unsigned char i,j;
    unsigned int m;

    //查找字库汉字表
    for(m=0;m<FONT_NUM;m++)
    {
        if((Font_Data[m].txt[0]==Hz[0])&&(Font_Data[m].txt[1]==Hz[1]))
        {
            for( j=0;j<2;j++ )
            {
                for( i=0;i<16;i++ )
                {
                    if(turn==0)
                        ShowBuf[Y+j][X+i]= Font_Data[m].dat[i+j*16];
                    else
                        ShowBuf[Y+j][X+i]=~Font_Data[m].dat[i+j*16];
                }
            }
        }
    }

    if(mode)
    {
        OLED_Show();
    }
}
//==============================================================================
// 函数名称: OLED_Str5x7
// 函数功能: OLED 显示 5*7 点阵字符串
// 入口参数: X: 水平坐标列 0..127  Y: 垂直坐标行(页) 0..7
//       turn: 0:正显示 1:反显示  mode: 1:立即刷新显示 0:仅写缓存
//==============================================================================
```

```
void OLED_Str5x7(unsigned char X,unsigned char Y,char *s,u8 turn,u8 mode)
{
    while(*s != '\0')           //字符串不为空，循环
    {
        OLED_A08(X,Y,*s,turn,0);
        X += 6;
        s++;
    }

    if(mode)
    {
        OLED_Show();
    }
}
//==============================================================================
// 函数名称: OLED_String
// 函数功能: OLED 显示字符串。
// 入口参数: X: 水平坐标列 0..127   Y: 垂直坐标行(页) 0..7
//       turn: 0:正显示 1:反显示   mode: 1:立即刷新显示 0:仅写缓存
//==============================================================================
void OLED_String(unsigned char X,unsigned char Y,char *s,u8 turn,u8 mode)
{
    char hz[2];
    while(*s != '\0')           //字符串不为空，循环
    {
        if ((unsigned char)*s < 0x80)       //根据输入数据的大小判断是字符还是汉字，
        {
            OLED_A16(X,Y,*s,turn,0);
            X += 8;
            s++;
        }
        else
        {
            hz[0] = *s ;
            hz[1] = *(s+1) ;
            OLED_HZ16(X,Y,hz,turn,0);
            X += 16;
            s += 2;
        }

        if(X>127)           //行
        {
            X = 0;
            Y += 2;
        }
    }
```

```
    if(mode)
    {
        OLED_Show();
    }
}
```

2.22.3.3 config.h -- 项目配置的头文件

```
#ifndef __CONFIG_H__            //防止头文件被重复包含
#define __CONFIG_H__

#define HIRC            11059200UL
#define FOSC            11059200UL
#define SYSCLK          FOSC
#define MAIN_Fosc       FOSC

#include <stc8h.h>              //包含外部头文件
#include <string.h>
#include <stdlib.h>

#include "def.h"                //包含项目头文件
#include "oled.h"

void SYS_Init(void);            //函数声明

#endif
```

2.22.3.4 oled.h -- 项目配置文件的头文件

```
#ifndef __OLED_H__              //防止头文件被重复包含
#define __OLED_H__

/*  OLED Pixel */
#define WIDTH 128
#define HEIGHT 64
#define PAGES   8

/*  OLED Brightness */
#define BRIGHTNESS_MIN 1
#define BRIGHTNESS_MAX 25

/*  OLED Driver */
void OLED_Init(void);       //初始化
void OLED_Show(void);       //显示(缓存数据刷屏)
void OLED_Disp(unsigned char On);       //开关显示
```

```
void OLED_CLS(void);        //清屏
void OLED_Light(unsigned char Level);     //亮度设置
void OLED_SetXY(unsigned char X,unsigned char Y);            //设定坐标
void OLED_Point(unsigned char X,unsigned char Y,u8 mode);   //画一个点
void OLED_LineH(unsigned char X,unsigned char Y,unsigned char W,u8 mode);     //画一条水平线
void OLED_LineV(unsigned char X,unsigned char Y,unsigned char H,u8 mode);     //画一条垂直线
void OLED_Line( unsigned char X0, unsigned char Y0, unsigned char X1,unsigned char Y1,u8 mode);
void OLED_Box(unsigned char X,unsigned char Y,unsigned char W,unsigned char H,u8 mode);     //画一个方框
void OLED_Circle(unsigned char X, unsigned char Y, unsigned char r,u8 mode);     //画一个圆
void OLED_BMP(unsigned char X,unsigned char Y,unsigned char W,unsigned char H,unsigned char BMP[],u8 mode);
void OLED_A08(unsigned char X,unsigned char Y,char Ch, u8 turn,u8 mode);    //OLED 显示一个 5*7 字符
void OLED_A16(unsigned char X,unsigned char Y,char Ch, u8 turn,u8 mode);    //OLED 显示一个 8*16 字符
void OLED_HZ16(unsigned char X,unsigned char Y,char *Hz, u8 turn,u8 mode); //OLED 显示一个 16*16 汉字
void OLED_Str5x7(unsigned char X,unsigned char Y,char *s,u8 turn,u8 mode); //OLED 显示 5*7 点阵字符串
void OLED_String(unsigned char X,unsigned char Y,char *s,u8 turn,u8 mode); //OLED 显示 16 点阵高字符串

#endif
```

2.22.3.5 font.h -- 项目配置文件的头文件

```
#ifndef __FONT_H__          //防止头文件被重复包含
#define __FONT_H__

/* 自定义中文字库，16*16，宋体 */
#define FONT_NUM    sizeof(Font_Data)/sizeof(Font_Data[0])       //自动统计汉字数目

struct FONT_DATA
{
    unsigned char dat[32];
    char txt[2];
};

struct FONT_DATA code Font_Data[] =
{
    {
        0x10,0x60,0x02,0x8C,0x00,0x0E,0x22,0x12,0x0A,0xC2,0x0A,0x12,0x22,0x0E,0x00,0x00,
        0x04,0x04,0x7E,0x01,0x20,0x21,0x11,0x09,0x05,0xFF,0x05,0x09,0x11,0x21,0x20,0x00,"深", /* 0 */
    },
    {
        0x20,0x20,0x20,0xFF,0x20,0x20,0x00,0xFF,0x00,0x00,0xFE,0x00,0x00,0xFF,0x00,0x00,
        0x10,0x30,0x10,0x0F,0x08,0x88,0x60,0x1F,0x00,0x00,0x3F,0x00,0x00,0xFF,0x00,0x00,"圳", /* 1 */
    },
    {
        0x20,0x20,0x20,0x20,0x20,0x20,0x20,0xFF,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x00,
        0x80,0x80,0x40,0x20,0x10,0x0C,0x03,0x00,0x03,0x0C,0x10,0x20,0x40,0x80,0x80,0x00,"大", /* 2 */
    },
```

```
    {
        0x40,0x30,0x11,0x96,0x90,0x90,0x91,0x96,0x90,0x90,0x98,0x14,0x13,0x50,0x30,0x00,
        0x04,0x04,0x04,0x04,0x04,0x44,0x84,0x7E,0x06,0x05,0x04,0x04,0x04,0x04,0x04,0x00,"学", /* 3 */
    },
};

#endif
```

2.22.3.6 pic.h -- 项目配置文件的头文件

```
#ifndef __PIC_H__           //防止头文件被重复包含
#define __PIC_H__

#define IMG_WIDTH       128
#define IMG_HEIGHT      64
#define IMG_DEPTH       1

u8 code OLED12864_IMG0[8][128] =
{
    0x00,0x00,0x00,0xE0,0xF0,0xF8,0xF8,0x9C,0x1C,0x1C,0x1C,0x1C,0x1C,0x38,0x38,0x18,
    0x00,0x10,0x18,0x1C,0x1C,0xDC,0xFC,0xFC,0xFC,0x1C,0x1C,0x1C,0x1C,0x0C,0x04,0xC0,
    0xE0,0xF0,0x78,0x38,0x38,0x1C,0x1C,0x1C,0x1C,0x3C,0x38,0x08,0x00,0x00,0xC0,0xF0,
    0xFC,0xFC,0x7C,0xFC,0xF0,0x00,0x00,0xC0,0xC0,0xE0,0xF8,0xFC,0xFC,0xFC,0x7C,0x0C,
    0x00,0xC0,0xC0,0xF0,0xF0,0x78,0x38,0x1C,0x1C,0x1C,0x1C,0x1C,0x38,0x38,0x08,0x00,
    0x80,0xF0,0xFC,0xFC,0xFC,0x00,0x00,0x00,0x00,0x00,0xC0,0xFC,0xFC,0xFC,0x0C,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x70,0x70,0xF0,0xE1,0xE3,0xE3,0xE3,0xE3,0xE7,0x7F,0x7F,0x3E,0x1C,0x00,0x00,
    0x00,0x00,0x00,0xF0,0xFE,0xFF,0x7F,0x0F,0x03,0x00,0x00,0x00,0x00,0x00,0x1F,0x3F,
    0x7F,0x7F,0xF0,0xE0,0xE0,0xE0,0xE0,0x70,0x70,0x10,0x00,0x00,0xC0,0xF0,0xFF,0xFF,
    0x7F,0x03,0x00,0x03,0x0F,0x0F,0x0F,0x03,0xC3,0xF0,0xFF,0xFF,0x7F,0x03,0x00,0x00,
    0x1F,0x3F,0x3F,0x7F,0xF0,0xF0,0xE0,0xE0,0xE0,0x70,0x70,0x70,0x00,0x00,0x00,0x3C,
    0x3F,0x7F,0xFF,0xF3,0xE0,0xE0,0xE0,0xE0,0x70,0x7E,0x3F,0x1F,0x0F,0x03,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x10,0x18,0x30,0x60,0xE0,
    0x40,0x00,0xF0,0xF0,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,
    0x30,0xF0,0xF0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xF8,0xF8,0x00,0x00,0x00,0x00,0x00,
    0xF8,0xF8,0x00,0x00,0x00,0x00,0x00,0xF0,0xF0,0x00,0x00,0x00,0x00,0x00,0xF8,0xF8,0x00,0x00,
    0x00,0x00,0x00,0xF0,0xF0,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,
    0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0xF0,0xF0,0x00,0x00,0x00,0x00,0xC0,0xC0,0xC0,0xC0,
    0xC0,0xC0,0xC0,0xC0,0xFE,0xFE,0xC0,0xC0,0xC0,0xC0,0xC0,0xC0,0xC0,0xC0,0xC0,0xFE,0xFE,0xC0,
    0xC0,0xC0,0xC0,0xC0,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x1C,0x18,0x38,0x70,0x20,
    0x00,0x00,0x00,0x90,0xB0,0x98,0x9C,0x8C,0x86,0x83,0x82,0xF0,0xF0,0x82,0x83,0x86,0x8C,
    0x9C,0xB8,0x90,0x80,0x00,0x00,0x06,0x06,0x06,0x06,0xFF,0xFF,0x06,0x06,0x06,0x00,0x00,
```

```
    0xFF,0xFF,0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0x00,
    0x00,0x00,0xFF,0xFF,0x00,0x00,0xC3,0xC3,0xC3,0xC3,0xC3,0xC3,0xC3,0xFF,0xFF,0xC3,
    0xC3,0xC3,0xC3,0xC3,0xC3,0x00,0x00,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x80,
    0x80,0x00,0x80,0x8F,0x0F,0x00,0x00,0x10,0x78,0xF0,0xC0,0x00,0x00,0x0F,0x0F,0x00,
    0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xF8,0x7E,0x0C,
    0x00,0x00,0x81,0xC1,0xC1,0x61,0x31,0x19,0x0D,0x07,0xFF,0xFF,0x07,0x1D,0x31,0x61,
    0xC1,0xC1,0x81,0x01,0x00,0x60,0x60,0x30,0x30,0x3F,0x1F,0x18,0x18,0x0C,0xCC,0xF0,
    0x7F,0x0F,0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0x00,0x00,0x00,0xFF,0xFF,0x00,0x00,
    0x00,0x00,0xFF,0xFF,0x00,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x3F,0x3F,0x30,
    0x33,0x36,0x3E,0x34,0x30,0x30,0x00,0xFF,0xFF,0x00,0x00,0x00,0x40,0xF0,0x7E,0x0F,
    0x01,0x00,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x03,0x07,0x02,0x00,0x00,0xF0,0xE0,
    0x01,0x07,0x3F,0xF8,0x40,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x0F,0x03,0x00,0x00,
    0x01,0x03,0x01,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0F,0x0F,0x00,0x00,0x00,0x00,
    0x00,0x01,0x03,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x0E,0x07,0x03,0x01,
    0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x01,0x00,0x00,0x00,0x00,0x0F,0x0F,0x00,0x00,
    0x00,0x00,0x0F,0x0F,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x06,
    0x06,0x06,0x06,0x06,0x06,0x06,0x06,0x0F,0x0F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x07,0x0F,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0F,0x07,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x38,0x6C,0xC4,0xC4,0x84,0x88,0x3C,0x00,0x00,
    0x0C,0x04,0x04,0x04,0xFC,0x04,0x04,0x04,0x04,0x00,0x00,0xE0,0xF0,0x08,0x0C,0x04,
    0x04,0x04,0x08,0x08,0x30,0x00,0x00,0x38,0x6C,0xC4,0xC4,0xA4,0x18,0x00,0x00,0x04,
    0xFC,0xFC,0x84,0x80,0x80,0x80,0x84,0xFC,0xFC,0x04,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0xFE,0x02,0x4A,0x4A,0x4A,0x4A,0xFA,0x4A,0xCA,0x4A,0x4A,0x02,0xFE,0x00,0x08,0x08,
    0x88,0x08,0xBE,0x08,0x08,0xC8,0x88,0x08,0x3E,0x08,0x88,0x08,0x08,0x00,0x00,0xFC,
    0x24,0x24,0x24,0x24,0x26,0x26,0x24,0x24,0x24,0x24,0x3C,0x00,0x80,0xFC,0x84,0x96,
    0xA4,0xFC,0x00,0x08,0xE8,0x2A,0x2C,0x28,0xE8,0x08,0x08,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1E,0x08,0x10,0x10,0x11,0x11,0x0F,0x00,0x00,
    0x00,0x00,0x10,0x10,0x1F,0x10,0x10,0x00,0x00,0x00,0x00,0x03,0x0F,0x08,0x18,0x10,
    0x10,0x10,0x10,0x08,0x04,0x00,0x00,0x0E,0x19,0x10,0x10,0x11,0x0F,0x00,0x00,0x10,
    0x1F,0x1F,0x10,0x00,0x00,0x00,0x10,0x1F,0x1F,0x10,0x00,0x00,0x00,0x00,0x00,0x00,
    0x3F,0x10,0x14,0x14,0x14,0x14,0x17,0x14,0x15,0x17,0x14,0x10,0x3F,0x00,0x08,0x07,
    0x01,0x00,0x3F,0x20,0x20,0x20,0x21,0x20,0x30,0x1E,0x00,0x07,0x0C,0x20,0x1C,0x07,
    0x00,0x3F,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x11,0x3F,0x00,0x30,0x1F,0x20,0x26,
    0x20,0x3F,0x20,0x18,0x07,0x00,0x00,0x00,0x3F,0x20,0x3E,0x00,0x00,0x00,0x00,0x00,
};

u8 code OLED12864_IMG1[8][128] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0xC0,0xF0,0xF8,0xF8,0xFC,0xFC,0xFC,0xF8,0xF8,0xF0,0xE0,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0xE0,0xE0,0xF0,0xF0,0xF0,0xE0,0xE0,
    0xC0,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xC0,0xC0,0xC0,0xC0,
0xC1,0xC7,0xCF,0xDF,0xDF,0xDF,0xDF,0xDF,0xCF,0xC7,0xC3,0xC0,0xC0,0xC0,0xC0,0x80,
0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x07,0x1F,0x3F,0x7F,0x7F,0x7F,0x7F,0x7F,0x3F,
0x1F,0x0F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0x1F,0xBF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x1F,0xFF,0xFF,0xFF,
0xFF,0x00,0x00,0xE0,0xF8,0xFE,0xFF,0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0x7F,0xFE,0xFC,0xF0,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0xC0,0x80,0x80,
0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0xE0,0xE0,0xF0,0xF0,
0xE0,0xE0,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xC0,0xE0,
0xF0,0xF0,0xF0,0xE0,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0x00,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0xFF,0xFF,0xFF,
0xFF,0xFC,0xFF,0xFF,0x1F,0xC7,0xF0,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFC,0xE3,0x0F,0x7F,0xFF,0xFE,0xF0,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFE,0xFC,0xF8,0xF0,0xE0,0x80,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,0xC7,0xEF,0xCF,0xCF,0xCF,
0xCF,0xEF,0xC7,0xC0,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xC3,0xE7,0xCF,
0xCF,0xDF,0xCF,0xCF,0xE7,0xE1,0xE0,0xF0,0xF8,0x78,0x7F,0x3F,0x1F,0x1F,0x00,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0x0F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x1F,0x1F,0x1F,
0x1F,0x1F,0x87,0xF0,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xF8,0xC3,0x0F,0x1F,0x1F,0x1E,0x00,0x00,0xE0,0xE0,0xE0,
0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFC,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x60,0x78,0x7C,0x3E,0x9F,0xEF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0x9F,0x3E,0x7C,0x78,0x70,0x7C,0x3E,0x1F,0x0F,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,
0x00,0x00,0x01,0x01,0x01,0x01,0x0B,0xFF,0xFF,0xFF,0xFF,0xFF,0x01,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0x01,0x01,0x01,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x03,0x0F,0x3F,
0x7F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x7F,0x1F,0x07,0x01,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0C,0x0F,0x0F,0xFF,0xFF,0xFF,0x5F,0x0F,
0xFF,0xFF,0xFF,0x0F,0x0F,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,
0xFF,0x01,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,

```
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0xFF,0xFF,0xFF,
    0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xC0,
    0xE0,0xE0,0xE1,0xE3,0xE3,0xE7,0xC7,0x0F,0x0F,0x0F,0x0F,0x0F,0x0F,0x0F,0xCF,0xEF,
    0xE7,0xE7,0xE7,0xE3,0xE1,0xC1,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x1F,0x1F,0x1F,0x00,0x00,
    0x1F,0x1F,0x1F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0F,0x1F,0x1F,
    0x1F,0x00,0x1F,0x1F,0x1F,0x0F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x07,
    0x1F,0x1F,0x1F,0x1F,0x0F,0x00,0x0F,0x1F,0x1F,0x1F,0x1F,0x0F,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x1F,0x1F,0x1F,0x1F,0x0F,0x00,0x07,0x1F,0x1F,
    0x1F,0x1F,0x0F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0F,
    0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x0F,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0F,0x1F,
    0x1F,0x1F,0x1F,0x1F,0x1F,0x0F,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};

#endif
```

2.22.3.7 ascii.h -- 项目配置文件的头文件

```
#ifndef __ASCII_H__          //防止头文件被重复包含
#define __ASCII_H__

/*****6*8 的点阵 ASCII 字符**************************************/
unsigned char code ASC8[][6] =
{
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,    // sp
    0x00, 0x00, 0x00, 0x2f, 0x00, 0x00,    // !
    0x00, 0x00, 0x07, 0x00, 0x07, 0x00,    // "
    0x00, 0x14, 0x7f, 0x14, 0x7f, 0x14,    // #
    0x00, 0x24, 0x2a, 0x7f, 0x2a, 0x12,    // $
    0x00, 0x62, 0x64, 0x08, 0x13, 0x23,    // %
    0x00, 0x36, 0x49, 0x55, 0x22, 0x50,    // &
    0x00, 0x00, 0x05, 0x03, 0x00, 0x00,    // '
    0x00, 0x00, 0x1c, 0x22, 0x41, 0x00,    // (
    0x00, 0x00, 0x41, 0x22, 0x1c, 0x00,    // )
    0x00, 0x14, 0x08, 0x3E, 0x08, 0x14,    // *
    0x00, 0x08, 0x08, 0x3E, 0x08, 0x08,    // +
    0x00, 0x00, 0x00, 0xA0, 0x60, 0x00,    // ,
    0x00, 0x08, 0x08, 0x08, 0x08, 0x08,    // -
    0x00, 0x00, 0x60, 0x60, 0x00, 0x00,    // .
    0x00, 0x20, 0x10, 0x08, 0x04, 0x02,    // /
    0x00, 0x3E, 0x51, 0x49, 0x45, 0x3E,    // 0
    0x00, 0x00, 0x42, 0x7F, 0x40, 0x00,    // 1
    0x00, 0x42, 0x61, 0x51, 0x49, 0x46,    // 2
    0x00, 0x21, 0x41, 0x45, 0x4B, 0x31,    // 3
    0x00, 0x18, 0x14, 0x12, 0x7F, 0x10,    // 4
    0x00, 0x27, 0x45, 0x45, 0x45, 0x39,    // 5
```

```
0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30,  // 6
0x00, 0x01, 0x71, 0x09, 0x05, 0x03,  // 7
0x00, 0x36, 0x49, 0x49, 0x49, 0x36,  // 8
0x00, 0x06, 0x49, 0x49, 0x29, 0x1E,  // 9
0x00, 0x00, 0x36, 0x36, 0x00, 0x00,  // :
0x00, 0x00, 0x56, 0x36, 0x00, 0x00,  // ;
0x00, 0x08, 0x14, 0x22, 0x41, 0x00,  // <
0x00, 0x14, 0x14, 0x14, 0x14, 0x14,  // =
0x00, 0x00, 0x41, 0x22, 0x14, 0x08,  // >
0x00, 0x02, 0x01, 0x51, 0x09, 0x06,  // ?
0x00, 0x32, 0x49, 0x59, 0x51, 0x3E,  // @
0x00, 0x7C, 0x12, 0x11, 0x12, 0x7C,  // A
0x00, 0x7F, 0x49, 0x49, 0x49, 0x36,  // B
0x00, 0x3E, 0x41, 0x41, 0x41, 0x22,  // C
0x00, 0x7F, 0x41, 0x41, 0x22, 0x1C,  // D
0x00, 0x7F, 0x49, 0x49, 0x49, 0x41,  // E
0x00, 0x7F, 0x09, 0x09, 0x09, 0x01,  // F
0x00, 0x3E, 0x41, 0x49, 0x49, 0x7A,  // G
0x00, 0x7F, 0x08, 0x08, 0x08, 0x7F,  // H
0x00, 0x00, 0x41, 0x7F, 0x41, 0x00,  // I
0x00, 0x20, 0x40, 0x41, 0x3F, 0x01,  // J
0x00, 0x7F, 0x08, 0x14, 0x22, 0x41,  // K
0x00, 0x7F, 0x40, 0x40, 0x40, 0x40,  // L
0x00, 0x7F, 0x02, 0x0C, 0x02, 0x7F,  // M
0x00, 0x7F, 0x04, 0x08, 0x10, 0x7F,  // N
0x00, 0x3E, 0x41, 0x41, 0x41, 0x3E,  // O
0x00, 0x7F, 0x09, 0x09, 0x09, 0x06,  // P
0x00, 0x3E, 0x41, 0x51, 0x21, 0x5E,  // Q
0x00, 0x7F, 0x09, 0x19, 0x29, 0x46,  // R
0x00, 0x46, 0x49, 0x49, 0x49, 0x31,  // S
0x00, 0x01, 0x01, 0x7F, 0x01, 0x01,  // T
0x00, 0x3F, 0x40, 0x40, 0x40, 0x3F,  // U
0x00, 0x1F, 0x20, 0x40, 0x20, 0x1F,  // V
0x00, 0x3F, 0x40, 0x38, 0x40, 0x3F,  // W
0x00, 0x63, 0x14, 0x08, 0x14, 0x63,  // X
0x00, 0x07, 0x08, 0x70, 0x08, 0x07,  // Y
0x00, 0x61, 0x51, 0x49, 0x45, 0x43,  // Z
0x00, 0x00, 0x7F, 0x41, 0x41, 0x00,  // [
0x00, 0x55, 0x2A, 0x55, 0x2A, 0x55,  // \
0x00, 0x00, 0x41, 0x41, 0x7F, 0x00,  // ]
0x00, 0x04, 0x02, 0x01, 0x02, 0x04,  // ^
0x00, 0x40, 0x40, 0x40, 0x40, 0x40,  // _
0x00, 0x00, 0x01, 0x02, 0x04, 0x00,  // '
0x00, 0x20, 0x54, 0x54, 0x54, 0x78,  // a
0x00, 0x7F, 0x48, 0x44, 0x44, 0x38,  // b
0x00, 0x38, 0x44, 0x44, 0x44, 0x20,  // c
```

```
    0x00, 0x38, 0x44, 0x44, 0x48, 0x7F,     // d
    0x00, 0x38, 0x54, 0x54, 0x54, 0x18,     // e
    0x00, 0x08, 0x7E, 0x09, 0x01, 0x02,     // f
    0x00, 0x18, 0xA4, 0xA4, 0xA4, 0x7C,     // g
    0x00, 0x7F, 0x08, 0x04, 0x04, 0x78,     // h
    0x00, 0x00, 0x44, 0x7D, 0x40, 0x00,     // i
    0x00, 0x40, 0x80, 0x84, 0x7D, 0x00,     // j
    0x00, 0x7F, 0x10, 0x28, 0x44, 0x00,     // k
    0x00, 0x00, 0x41, 0x7F, 0x40, 0x00,     // l
    0x00, 0x7C, 0x04, 0x18, 0x04, 0x78,     // m
    0x00, 0x7C, 0x08, 0x04, 0x04, 0x78,     // n
    0x00, 0x38, 0x44, 0x44, 0x44, 0x38,     // o
    0x00, 0xFC, 0x24, 0x24, 0x24, 0x18,     // p
    0x00, 0x18, 0x24, 0x24, 0x18, 0xFC,     // q
    0x00, 0x7C, 0x08, 0x04, 0x04, 0x08,     // r
    0x00, 0x48, 0x54, 0x54, 0x54, 0x20,     // s
    0x00, 0x04, 0x3F, 0x44, 0x40, 0x20,     // t
    0x00, 0x3C, 0x40, 0x40, 0x20, 0x7C,     // u
    0x00, 0x1C, 0x20, 0x40, 0x20, 0x1C,     // v
    0x00, 0x3C, 0x40, 0x30, 0x40, 0x3C,     // w
    0x00, 0x44, 0x28, 0x10, 0x28, 0x44,     // x
    0x00, 0x1C, 0xA0, 0xA0, 0xA0, 0x7C,     // y
    0x00, 0x44, 0x64, 0x54, 0x4C, 0x44,     // z
    0x14, 0x14, 0x14, 0x14, 0x14, 0x14,     // horiz lines
};

/****************************************8*16 的点阵************************************/
unsigned char code ASC16[][16]=
{
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,     // - -
    0x00,0x00,0x38,0xFC,0xFC,0x38,0x00,0x00,0x00,0x00,0x00,0x0D,0x0D,0x00,0x00,0x00,     // -!-
    0x00,0x0E,0x1E,0x00,0x00,0x1E,0x0E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,     // -"-
    0x20,0xF8,0xF8,0x20,0xF8,0xF8,0x20,0x00,0x02,0x0F,0x0F,0x02,0x0F,0x0F,0x02,0x00,     // -#-
    0x38,0x7C,0x44,0x47,0x47,0xCC,0x98,0x00,0x03,0x06,0x04,0x1C,0x1C,0x07,0x03,0x00,     // -$-
    0x30,0x30,0x00,0x80,0xC0,0x60,0x30,0x00,0x0C,0x06,0x03,0x01,0x00,0x0C,0x0C,0x00,     // -%-
    0x80,0xD8,0x7C,0xE4,0xBC,0xD8,0x40,0x00,0x07,0x0F,0x08,0x08,0x07,0x0F,0x08,0x00,     // -&-
    0x00,0x10,0x1E,0x0E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,     // -'-
    0x00,0x00,0xF0,0xF8,0x0C,0x04,0x00,0x00,0x00,0x00,0x00,0x03,0x07,0x0C,0x08,0x00,0x00,     // -(-
    0x00,0x00,0x04,0x0C,0xF8,0xF0,0x00,0x00,0x00,0x00,0x00,0x08,0x0C,0x07,0x03,0x00,0x00,     // -)-
    0x80,0xA0,0xE0,0xC0,0xC0,0xE0,0xA0,0x80,0x00,0x02,0x03,0x01,0x01,0x03,0x02,0x00,     // -*-
    0x00,0x80,0x80,0xE0,0xE0,0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x03,0x03,0x00,0x00,0x00,     // -+-
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x10,0x1E,0x0E,0x00,0x00,0x00,     // -,-
    0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,     // ---
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0C,0x0C,0x00,0x00,0x00,     // -.-
    0x00,0x00,0x00,0x80,0xC0,0x60,0x30,0x00,0x0C,0x06,0x03,0x01,0x00,0x00,0x00,0x00,0x00,     // -/-
    0xF8,0xFC,0x04,0xC4,0x24,0xFC,0xF8,0x00,0x07,0x0F,0x09,0x08,0x08,0x0F,0x07,0x00,     // -0-
```

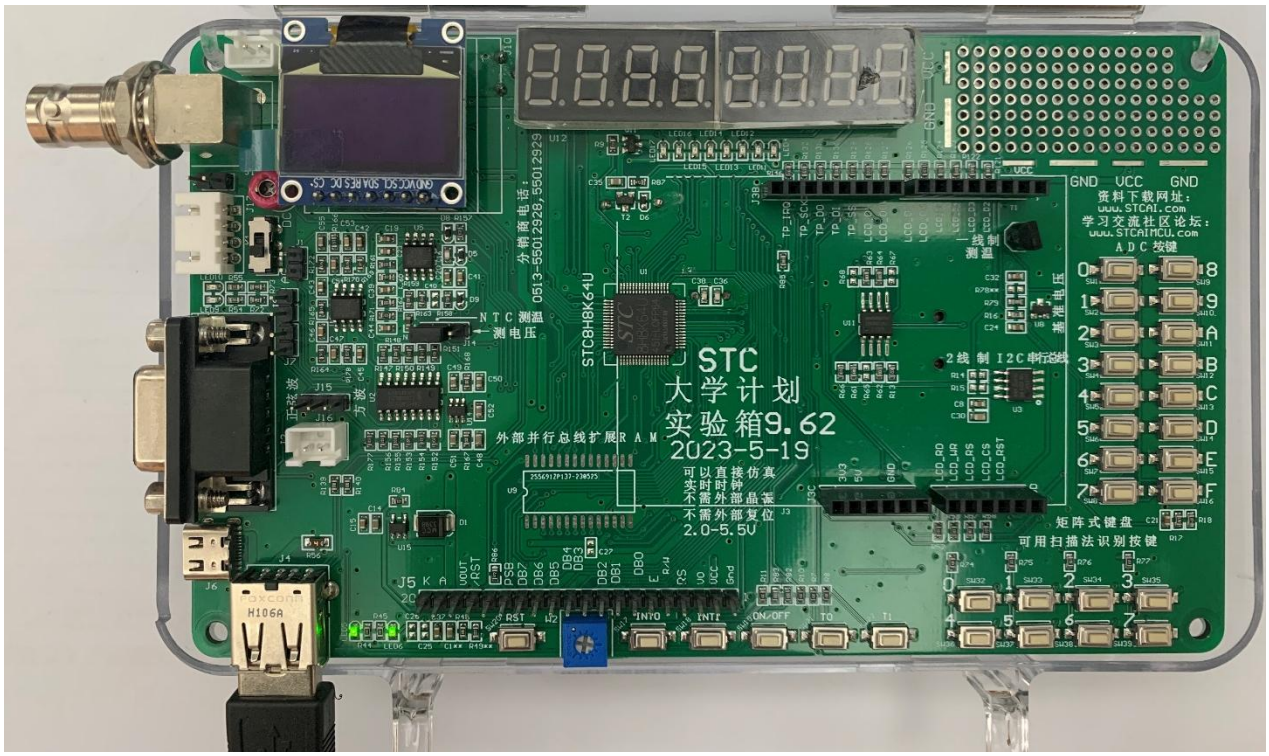
```
0x00,0x10,0x18,0xFC,0xFC,0x00,0x00,0x00,0x00,0x08,0x08,0x0F,0x0F,0x08,0x08,0x00,   // -1-
0x08,0x0C,0x84,0xC4,0x64,0x3C,0x18,0x00,0x0E,0x0F,0x09,0x08,0x08,0x0C,0x0C,0x00,   // -2-
0x08,0x0C,0x44,0x44,0x44,0xFC,0xB8,0x00,0x04,0x0C,0x08,0x08,0x08,0x0F,0x07,0x00,   // -3-
0xC0,0xE0,0xB0,0x98,0xFC,0xFC,0x80,0x00,0x00,0x00,0x00,0x08,0x0F,0x0F,0x08,0x00,   // -4-
0x7C,0x7C,0x44,0x44,0xC4,0xC4,0x84,0x00,0x04,0x0C,0x08,0x08,0x08,0x0F,0x07,0x00,   // -5-
0xF0,0xF8,0x4C,0x44,0x44,0xC0,0x80,0x00,0x07,0x0F,0x08,0x08,0x08,0x0F,0x07,0x00,   // -6-
0x0C,0x0C,0x04,0x84,0xC4,0x7C,0x3C,0x00,0x00,0x00,0x00,0x0F,0x0F,0x00,0x00,0x00,   // -7-
0xB8,0xFC,0x44,0x44,0x44,0xFC,0xB8,0x00,0x07,0x0F,0x08,0x08,0x08,0x0F,0x07,0x00,   // -8-
0x38,0x7C,0x44,0x44,0x44,0xFC,0xF8,0x00,0x00,0x08,0x08,0x08,0x0C,0x07,0x03,0x00,   // -9-
0x00,0x00,0x00,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x06,0x00,0x00,   // -:-
0x00,0x00,0x00,0x30,0x30,0x00,0x00,0x00,0x00,0x00,0x00,0x08,0x0E,0x06,0x00,0x00,   // -;-
0x00,0x80,0xC0,0x60,0x30,0x18,0x08,0x00,0x00,0x00,0x01,0x03,0x06,0x0C,0x08,0x00,   // -<-
0x40,0x40,0x40,0x40,0x40,0x40,0x40,0x00,0x02,0x02,0x02,0x02,0x02,0x02,0x02,0x00,   // -=-
0x00,0x08,0x18,0x30,0x60,0xC0,0x80,0x00,0x00,0x08,0x0C,0x06,0x03,0x01,0x00,0x00,   // ->-
0x18,0x1C,0x04,0xC4,0xE4,0x3C,0x18,0x00,0x00,0x00,0x00,0x0D,0x0D,0x00,0x00,0x00,   // -?-
0xF0,0xF8,0x08,0xC8,0xC8,0xF8,0xF0,0x00,0x07,0x0F,0x08,0x0B,0x0B,0x0B,0x01,0x00,   // -@-
0xE0,0xF0,0x98,0x8C,0x98,0xF0,0xE0,0x00,0x0F,0x0F,0x00,0x00,0x00,0x0F,0x0F,0x00,   // -A-
0x04,0xFC,0xFC,0x44,0x44,0xFC,0xB8,0x00,0x08,0x0F,0x0F,0x08,0x08,0x0F,0x07,0x00,   // -B-
0xF0,0xF8,0x0C,0x04,0x04,0x0C,0x18,0x00,0x03,0x07,0x0C,0x08,0x08,0x0C,0x06,0x00,   // -C-
0x04,0xFC,0xFC,0x04,0x0C,0xF8,0xF0,0x00,0x08,0x0F,0x0F,0x08,0x0C,0x07,0x03,0x00,   // -D-
0x04,0xFC,0xFC,0x44,0xE4,0x0C,0x1C,0x00,0x08,0x0F,0x0F,0x08,0x08,0x0C,0x0E,0x00,   // -E-
0x04,0xFC,0xFC,0x44,0xE4,0x0C,0x1C,0x00,0x08,0x0F,0x0F,0x08,0x00,0x00,0x00,0x00,   // -F-
0xF0,0xF8,0x0C,0x84,0x84,0x8C,0x98,0x00,0x03,0x07,0x0C,0x08,0x08,0x07,0x0F,0x00,   // -G-
0xFC,0xFC,0x40,0x40,0x40,0xFC,0xFC,0x00,0x0F,0x0F,0x00,0x00,0x00,0x0F,0x0F,0x00,   // -H-
0x00,0x00,0x04,0xFC,0xFC,0x04,0x00,0x00,0x00,0x00,0x08,0x0F,0x0F,0x08,0x00,0x00,   // -I-
0x00,0x00,0x00,0x04,0xFC,0xFC,0x04,0x00,0x07,0x0F,0x08,0x08,0x0F,0x07,0x00,0x00,   // -J-
0x04,0xFC,0xFC,0xC0,0xF0,0x3C,0x0C,0x00,0x08,0x0F,0x0F,0x00,0x01,0x0F,0x0E,0x00,   // -K-
0x04,0xFC,0xFC,0x04,0x00,0x00,0x00,0x00,0x08,0x0F,0x0F,0x08,0x08,0x0C,0x0E,0x00,   // -L-
0xFC,0xFC,0x38,0x70,0x38,0xFC,0xFC,0x00,0x0F,0x0F,0x00,0x00,0x00,0x0F,0x0F,0x00,   // -M-
0xFC,0xFC,0x38,0x70,0xE0,0xFC,0xFC,0x00,0x0F,0x0F,0x00,0x00,0x00,0x0F,0x0F,0x00,   // -N-
0xF0,0xF8,0x0C,0x04,0x0C,0xF8,0xF0,0x00,0x03,0x07,0x0C,0x08,0x0C,0x07,0x03,0x00,   // -O-
0x04,0xFC,0xFC,0x44,0x44,0x7C,0x38,0x00,0x08,0x0F,0x0F,0x08,0x00,0x00,0x00,0x00,   // -P-
0xF8,0xFC,0x04,0x04,0x04,0xFC,0xF8,0x00,0x07,0x0F,0x08,0x0E,0x3C,0x3F,0x27,0x00,   // -Q-
0x04,0xFC,0xFC,0x44,0xC4,0xFC,0x38,0x00,0x08,0x0F,0x0F,0x00,0x00,0x0F,0x0F,0x00,   // -R-
0x18,0x3C,0x64,0x44,0xC4,0x9C,0x18,0x00,0x06,0x0E,0x08,0x08,0x08,0x0F,0x07,0x00,   // -S-
0x00,0x1C,0x0C,0xFC,0xFC,0x0C,0x1C,0x00,0x00,0x00,0x08,0x0F,0x0F,0x08,0x00,0x00,   // -T-
0xFC,0xFC,0x00,0x00,0x00,0xFC,0xFC,0x00,0x07,0x0F,0x08,0x08,0x08,0x0F,0x07,0x00,   // -U-
0xFC,0xFC,0x00,0x00,0x00,0xFC,0xFC,0x00,0x01,0x03,0x06,0x0C,0x06,0x03,0x01,0x00,   // -V-
0xFC,0xFC,0x00,0x80,0x00,0xFC,0xFC,0x00,0x03,0x0F,0x0E,0x03,0x0E,0x0F,0x03,0x00,   // -W-
0x0C,0x3C,0xF0,0xC0,0xF0,0x3C,0x0C,0x00,0x0C,0x0F,0x03,0x00,0x03,0x0F,0x0C,0x00,   // -X-
0x00,0x3C,0x7C,0xC0,0xC0,0x7C,0x3C,0x00,0x00,0x00,0x08,0x0F,0x0F,0x08,0x00,0x00,   // -Y-
0x1C,0x0C,0x84,0xC4,0x64,0x3C,0x1C,0x00,0x0E,0x0F,0x09,0x08,0x08,0x0C,0x0E,0x00,   // -Z-
0x00,0x00,0xFC,0xFC,0x04,0x04,0x00,0x00,0x00,0x00,0x0F,0x0F,0x08,0x08,0x00,0x00,   // -[-
0x38,0x70,0xE0,0xC0,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x03,0x07,0x0E,0x00,   // -\-
0x00,0x00,0x04,0x04,0xFC,0xFC,0x00,0x00,0x00,0x00,0x08,0x08,0x0F,0x0F,0x00,0x00,   // -]-
0x08,0x0C,0x06,0x03,0x06,0x0C,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,   // -^-
```

```
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x20,0x20,0x20,0x20,0x20,0x20,0x20,0x20,   // -_-
    0x00,0x00,0x03,0x07,0x04,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,   // -`-
    0x00,0xA0,0xA0,0xA0,0xE0,0xC0,0x00,0x00,0x07,0x0F,0x08,0x08,0x07,0x0F,0x08,0x00,   // -a-
    0x04,0xFC,0xFC,0x20,0x60,0xC0,0x80,0x00,0x08,0x0F,0x07,0x08,0x08,0x0F,0x07,0x00,   // -b-
    0xC0,0xE0,0x20,0x20,0x20,0x60,0x40,0x00,0x07,0x0F,0x08,0x08,0x08,0x0C,0x04,0x00,   // -c-
    0x80,0xC0,0x60,0x24,0xFC,0xFC,0x00,0x00,0x07,0x0F,0x08,0x08,0x07,0x0F,0x08,0x00,   // -d-
    0xC0,0xE0,0xA0,0xA0,0xA0,0xE0,0xC0,0x00,0x07,0x0F,0x08,0x08,0x08,0x0C,0x04,0x00,   // -e-
    0x40,0xF8,0xFC,0x44,0x0C,0x18,0x00,0x00,0x08,0x0F,0x0F,0x08,0x00,0x00,0x00,0x00,   // -f-
    0xC0,0xE0,0x20,0x20,0xC0,0xE0,0x20,0x00,0x27,0x6F,0x48,0x48,0x7F,0x3F,0x00,0x00,   // -g-
    0x04,0xFC,0xFC,0x40,0x20,0xE0,0xC0,0x00,0x08,0x0F,0x0F,0x00,0x00,0x0F,0x0F,0x00,   // -h-
    0x00,0x00,0x20,0xEC,0xEC,0x00,0x00,0x00,0x00,0x00,0x08,0x0F,0x0F,0x08,0x00,0x00,   // -i-
    0x00,0x00,0x00,0x00,0x20,0xEC,0xEC,0x00,0x00,0x30,0x70,0x40,0x40,0x7F,0x3F,0x00,   // -j-
    0x04,0xFC,0xFC,0x80,0xC0,0x60,0x20,0x00,0x08,0x0F,0x0F,0x01,0x03,0x0E,0x0C,0x00,   // -k-
    0x00,0x00,0x04,0xFC,0xFC,0x00,0x00,0x00,0x00,0x00,0x08,0x0F,0x0F,0x08,0x00,0x00,   // -l-
    0xE0,0xE0,0x60,0xC0,0x60,0xE0,0xC0,0x00,0x0F,0x0F,0x00,0x0F,0x00,0x0F,0x0F,0x00,   // -m-
    0x20,0xE0,0xC0,0x20,0x20,0xE0,0xC0,0x00,0x00,0x0F,0x0F,0x00,0x00,0x0F,0x0F,0x00,   // -n-
    0xC0,0xE0,0x20,0x20,0x20,0xE0,0xC0,0x00,0x07,0x0F,0x08,0x08,0x08,0x0F,0x07,0x00,   // -o-
    0x20,0xE0,0xC0,0x20,0x20,0xE0,0xC0,0x00,0x40,0x7F,0x7F,0x48,0x08,0x0F,0x07,0x00,   // -p-
    0xC0,0xE0,0x20,0x20,0xC0,0xE0,0x20,0x00,0x07,0x0F,0x08,0x48,0x7F,0x7F,0x40,0x00,   // -q-
    0x20,0xE0,0xC0,0x60,0x20,0x60,0xC0,0x00,0x08,0x0F,0x0F,0x08,0x00,0x00,0x00,0x00,   // -r-
    0x40,0xE0,0xA0,0x20,0x20,0x60,0x40,0x00,0x04,0x0C,0x09,0x09,0x0B,0x0E,0x04,0x00,   // -s-
    0x20,0x20,0xF8,0xFC,0x20,0x20,0x00,0x00,0x00,0x00,0x07,0x0F,0x08,0x0C,0x04,0x00,   // -t-
    0xE0,0xE0,0x00,0x00,0xE0,0xE0,0x00,0x00,0x07,0x0F,0x08,0x08,0x07,0x0F,0x08,0x00,   // -u-
    0x00,0xE0,0xE0,0x00,0x00,0xE0,0xE0,0x00,0x00,0x03,0x07,0x0C,0x0C,0x07,0x03,0x00,   // -v-
    0xE0,0xE0,0x00,0x00,0x00,0xE0,0xE0,0x00,0x07,0x0F,0x0C,0x07,0x0C,0x0F,0x07,0x00,   // -w-
    0x20,0x60,0xC0,0x80,0xC0,0x60,0x20,0x00,0x08,0x0C,0x07,0x03,0x07,0x0C,0x08,0x00,   // -x-
    0xE0,0xE0,0x00,0x00,0x00,0xE0,0xE0,0x00,0x47,0x4F,0x48,0x48,0x68,0x3F,0x1F,0x00,   // -y-
    0x60,0x60,0x20,0xA0,0xE0,0x60,0x20,0x00,0x0C,0x0E,0x0B,0x09,0x08,0x0C,0x0C,0x00,   // -z-
    0x00,0x40,0x40,0xF8,0xBC,0x04,0x04,0x00,0x00,0x00,0x00,0x00,0x07,0x0F,0x08,0x08,0x00,   // -{-
    0x00,0x00,0x00,0xBC,0xBC,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0F,0x0F,0x00,0x00,0x00,   // -|-
    0x00,0x04,0x04,0xBC,0xF8,0x40,0x40,0x00,0x00,0x08,0x08,0x0F,0x07,0x00,0x00,0x00,   // -}-
    0x08,0x0C,0x04,0x0C,0x08,0x0C,0x04,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,   // -~-
    0x80,0xC0,0x60,0x30,0x60,0xC0,0x80,0x00,0x07,0x07,0x04,0x04,0x04,0x07,0x07,0x00,   // -□-
};

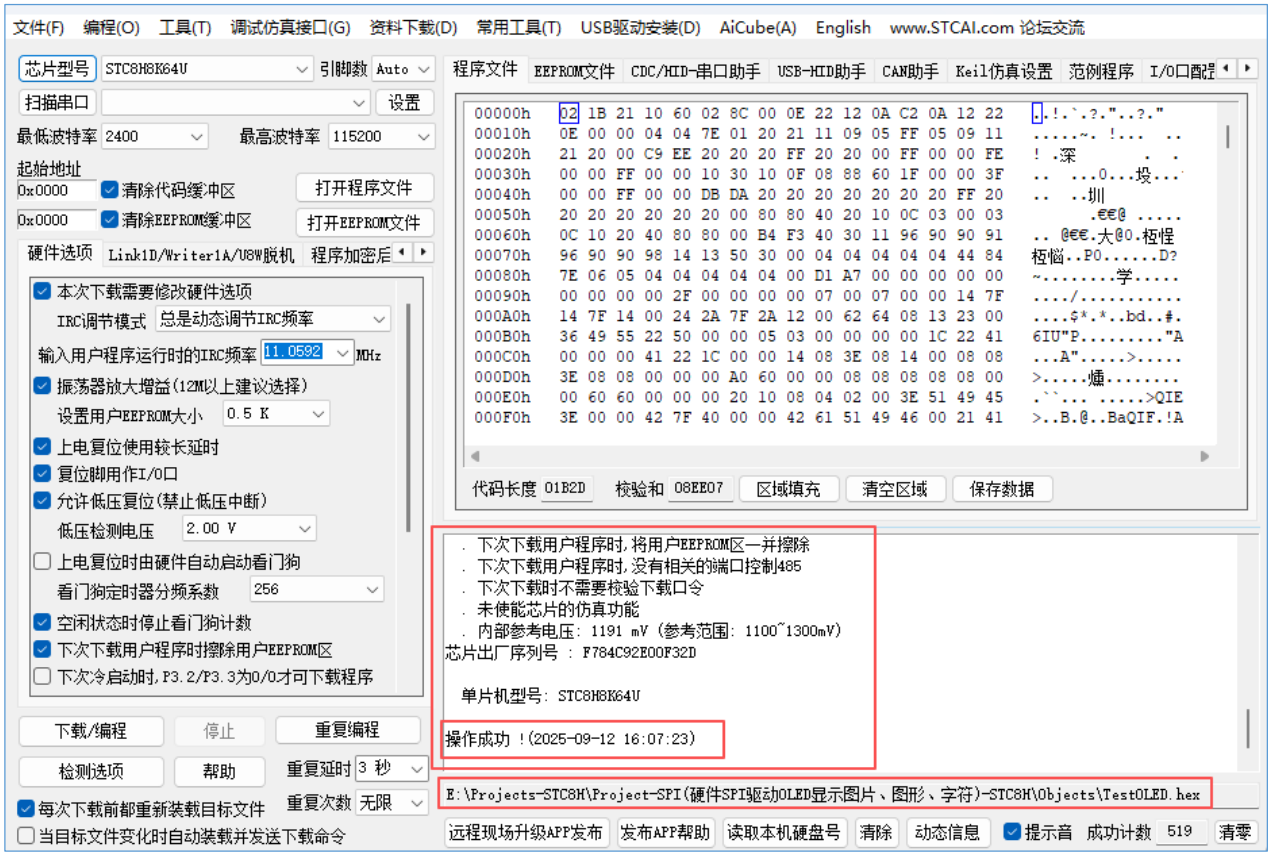
#endif
```

2.22.4　　程序下载，观察实验现象，验证实验效果

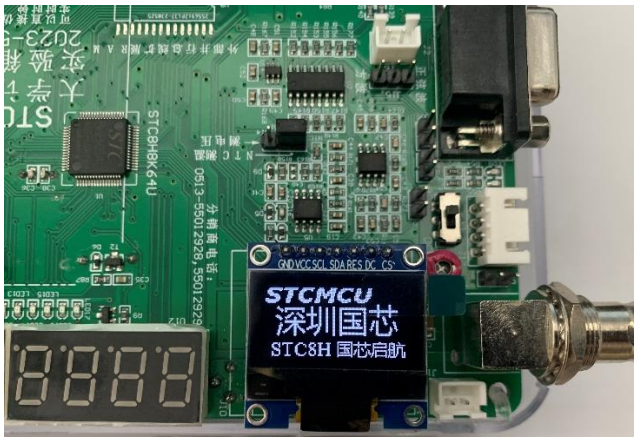
录入代码，保存，编译。将 OLED 屏正确的插到 STC8H8K64U 实验箱的 J11 接口，此时 OLED 屏上无任何图案显示，如下图：

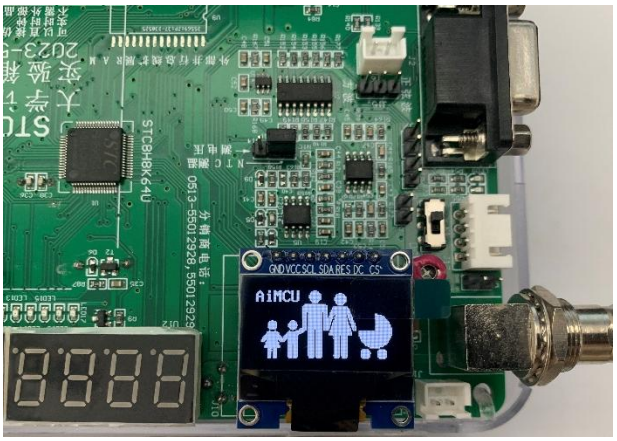


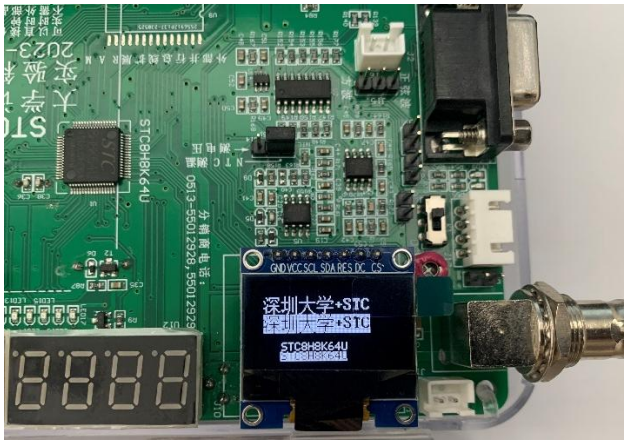
运行 AiCube-ISP 软件系统，按步骤打开并下载程序“TestOLED.hex”，如下图：

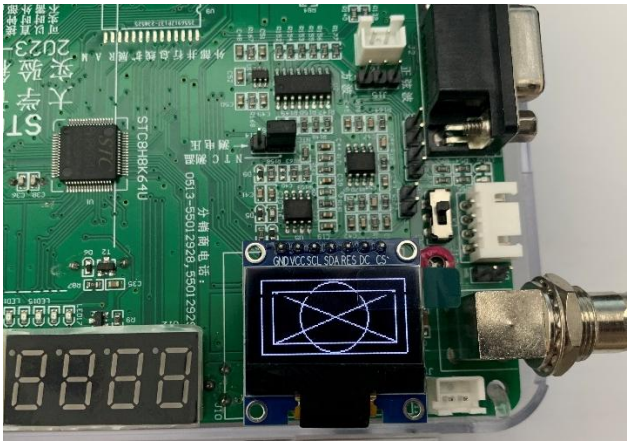


现在观察 STC8H8K64U 实验箱上的 OLED 屏幕，下列 4 幅图片在 OLED 屏上循环显示：









完成实验验证。

3 仿真 -- STC8H8K64U 实验箱仿真

3.1 USB 直接通信仿真 -- STC8H8K64U 实验箱仿真

在编写较大的程序时，往往会遇到代码编译没问题，但运行异常的情况，或者要观察项目运行状况，这时就可以借助仿真来查找问题，进行程序调试，或者对项目实时监控，进行逻辑验证。

STC8H 可支持仿真功能。STC8H 芯片出厂时默认仿真功能是关闭的，需要使用 AiCube-ISP 软件将目标单片机设置为仿真芯片，才能开启芯片的仿真功能。

STC8H 均支持串口通信仿真，其中 STC8H2K12U、STC8H2K32U、STC8H8K64U 芯片既可 USB 直接通信仿真，也可串口通信仿真。

USB 直接通信仿真，不需要任何仿真器，单片机本身就是仿真器，直接连接电脑就可进行仿真。

下面我们就以“中断实验 1(外部中断 INT0/P3.2，INT1/P3.3 控制点灯并 USB 仿真)”实验为例，使用 **STC8H8K64U 实验箱仿真 -- USB 直接通信仿真**，并观察实验现象。

3.1.1 安装 AiCube-ISP 软件，制作仿真芯片

3.1.1.1 安装 AiCube-ISP 下载/编程/烧录 工具软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

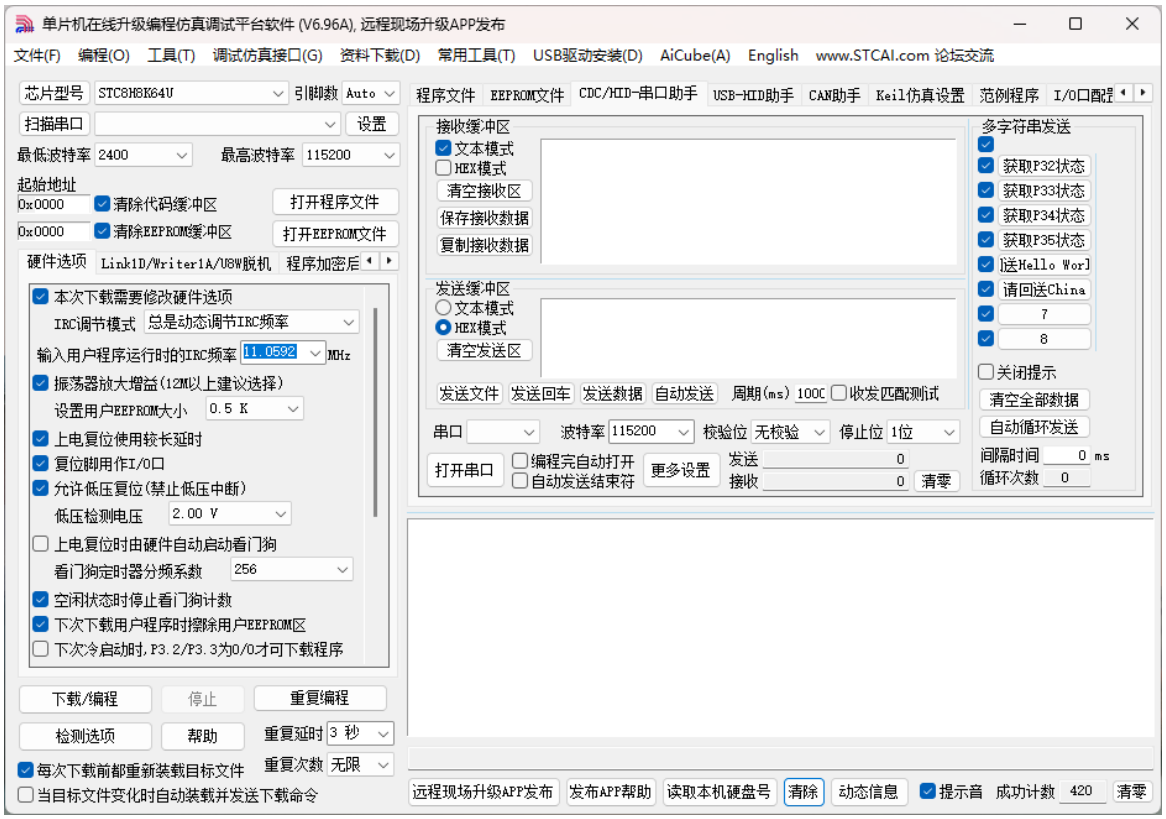


下载下来的是压缩包文件图标和解压缩后的 AiCube-ISP 文件图标，如下图：



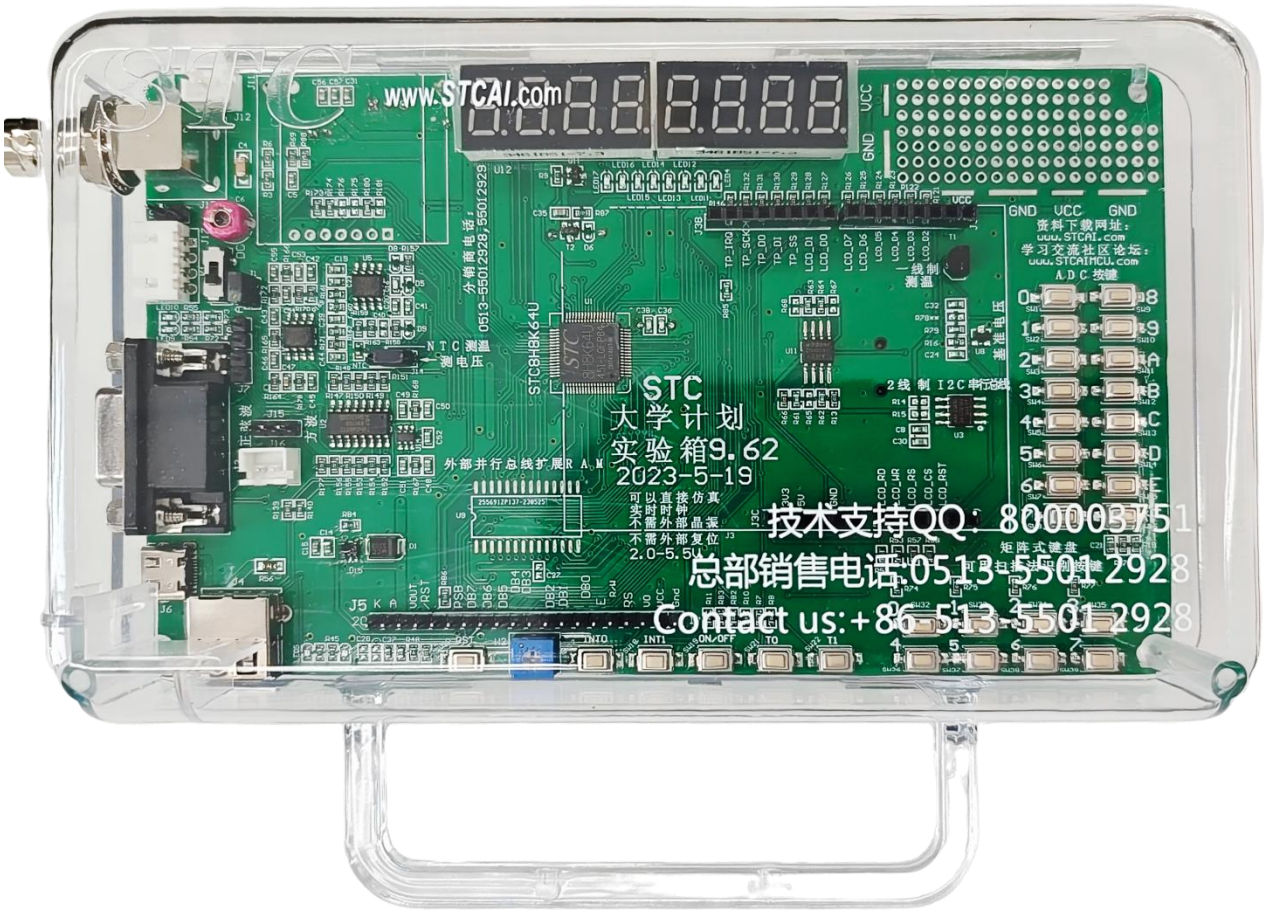
如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

双击 AiCube-ISP 系统软件图标，打开 AiCube-ISP 软件界面，即可直接使用，如下图：

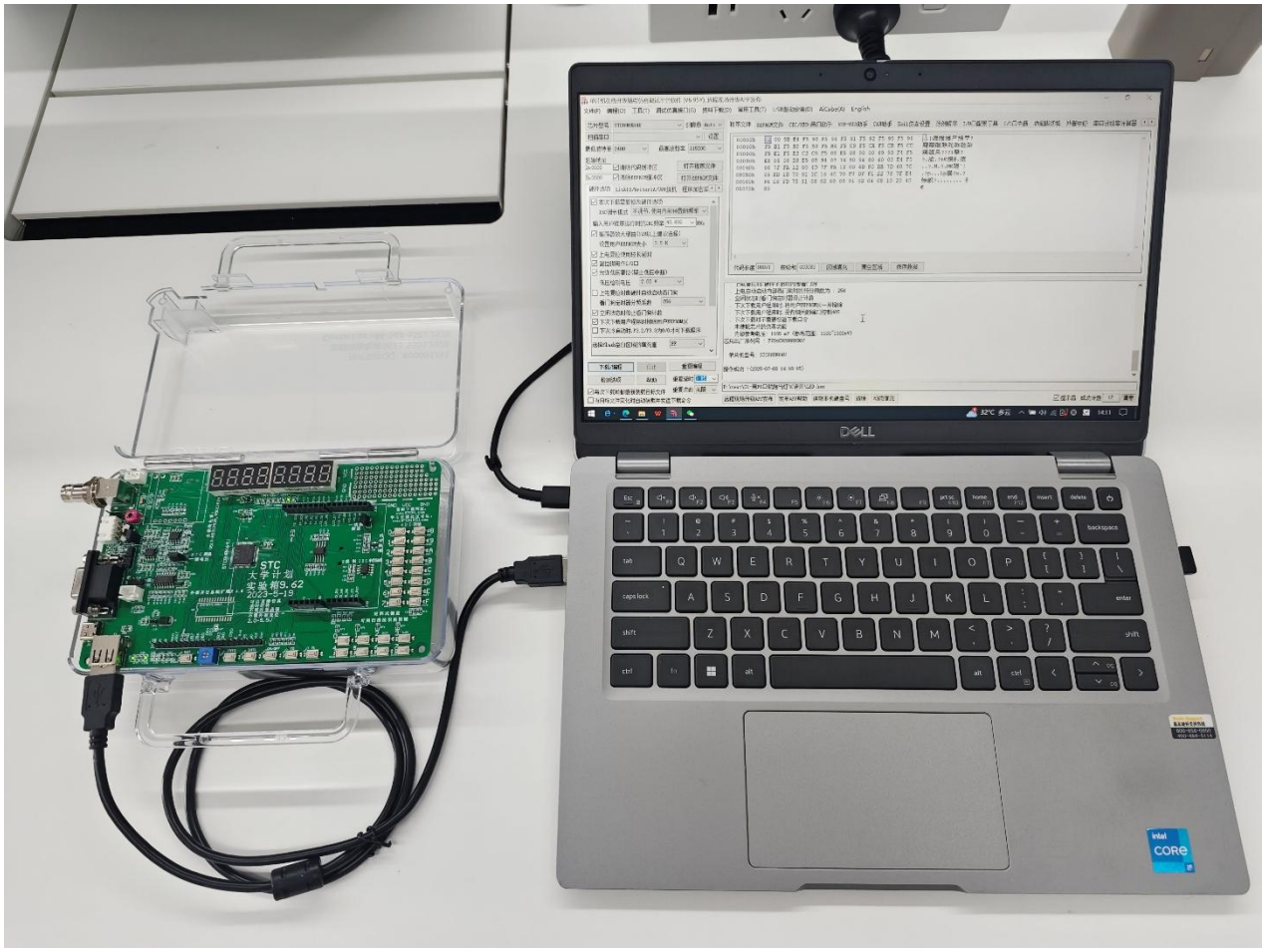


3.1.1.2　　STC8H8K64U 实验箱实物图及与电脑连接实物图

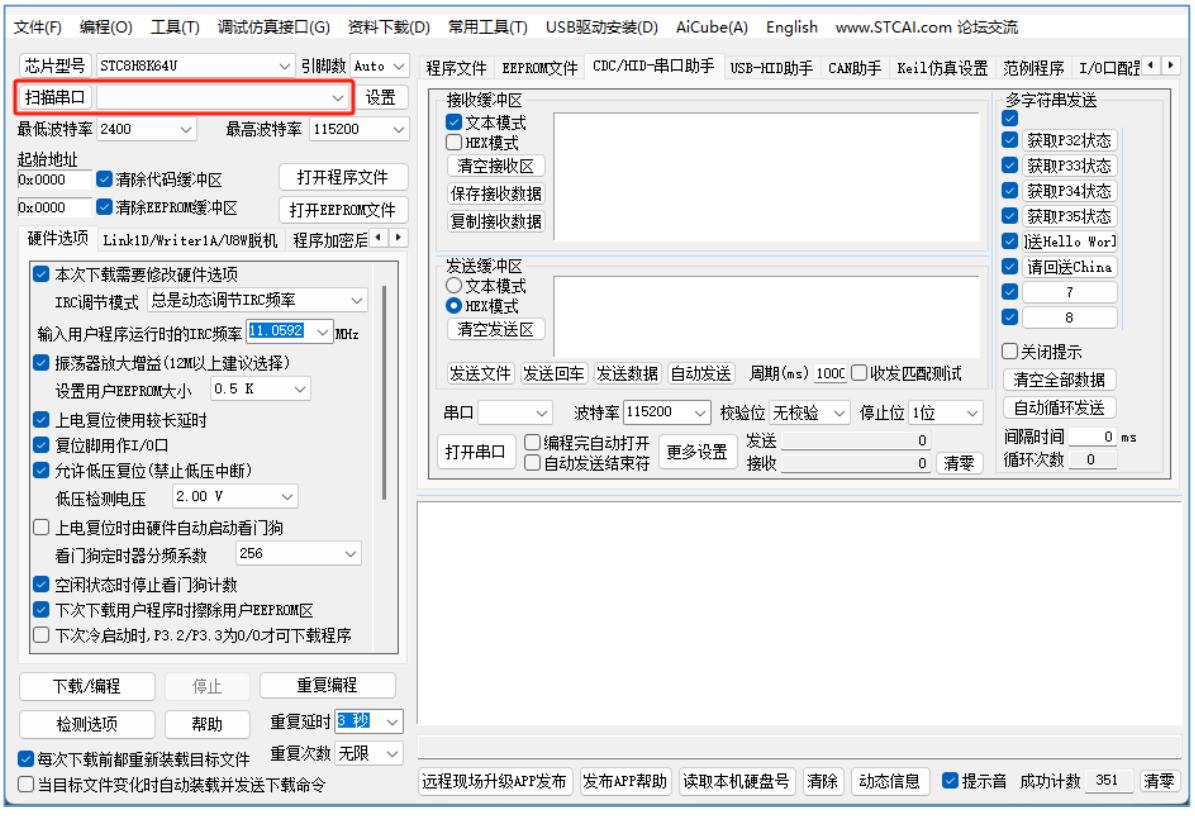
➲ STC8H8K64U 实验箱完整实物图（含外壳）：正面



➲ STC8H8K64U 实验箱与电脑连线实物图：USB-TypeA 到 USB-TypeA 线将实验箱与电脑正确连接

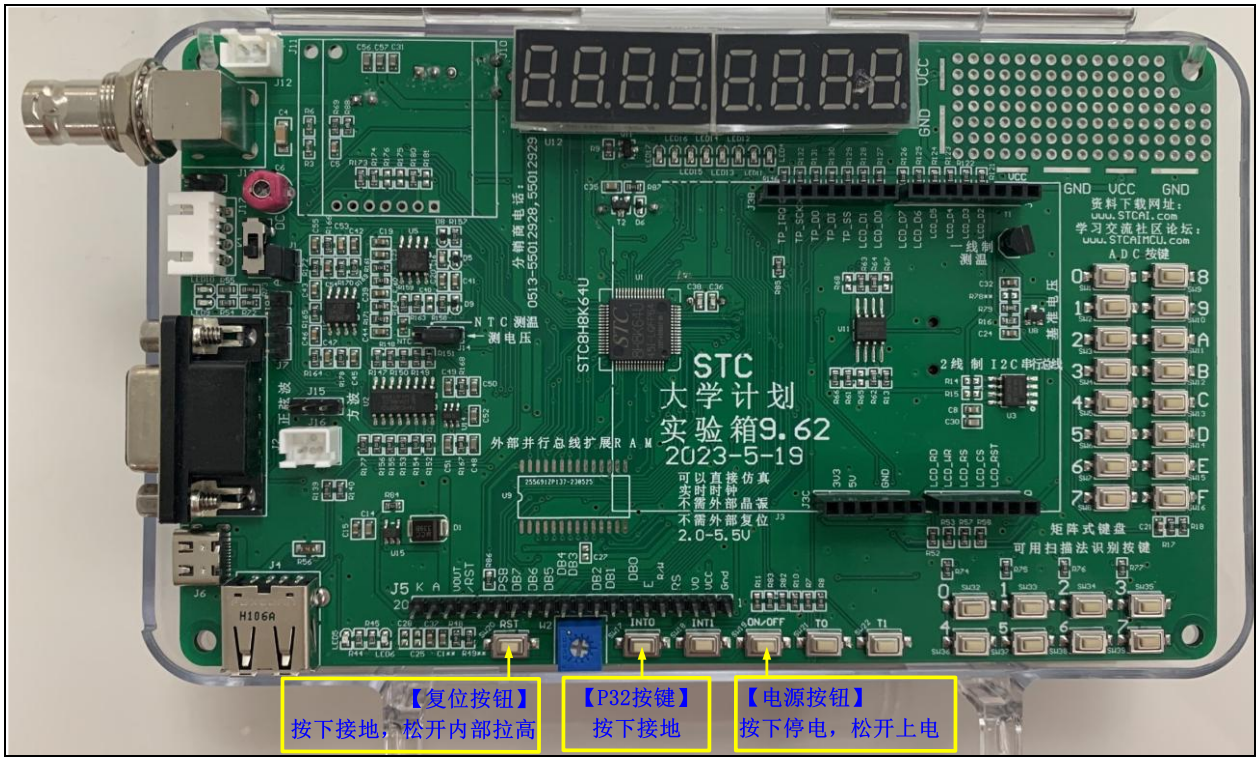


现在运行 AiCube-ISP-v6.96A 或以上版本。选择芯片型号：STC8H8K64U，如下图，我们看到“扫描串口”后面的端口列表框不是“（HID1）USB-Writer”，说明还不能下载程序到实验箱。

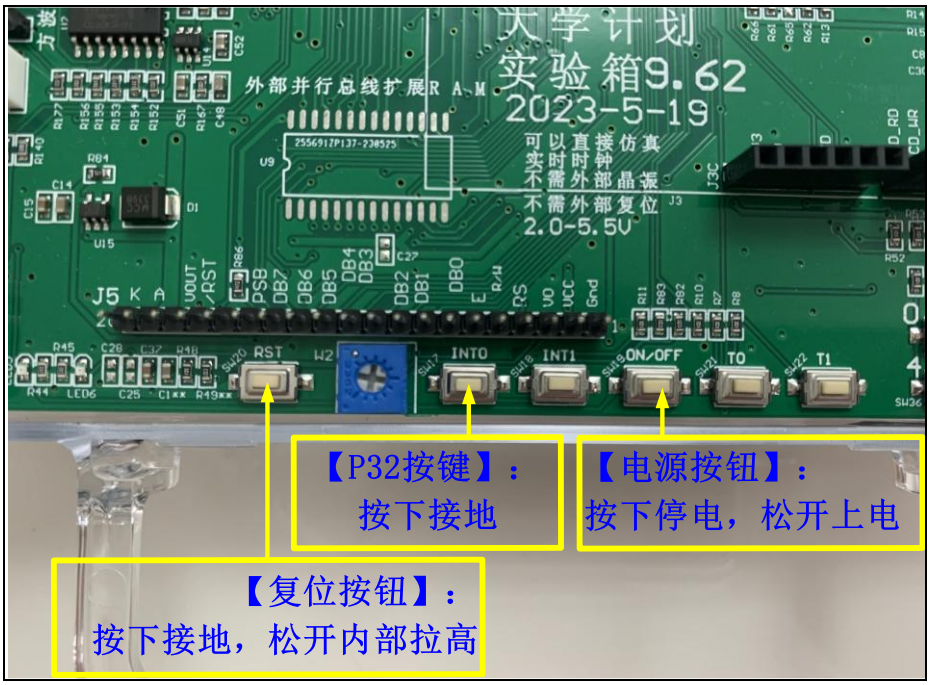


3.1.1.3　　硬件 USB 下载用户程序步骤

下图是 STC8H8K64U 实验箱打开盒盖的正面高清实物图：



局部放大实验箱图片，其正下方有一排按键，找到 P3.2/INT0 按键和 Power_SW 按键，如下图：



【方法一，P3.2 按键，再结合停电上电下载】

1、按下板子上的 P3.2 按键，就是 P3.2 接地

2、给目标芯片重新上电，不管之前是否已通电。

===本 STC8H8K64U 实验箱是：按下电源按钮停电，再松开电源按钮上电，

这时电脑端 ISP 下载软件会自动识别出“(HID1) USB Writer”，此时就与 P3.2 状态无关了，可以松开 P3.2 按键（P3.2 在用户程序区可做其它任意用途）

3、点击电脑端下载软件中的【下载/编程】按钮（注意：USB 下载与串口下载的操作顺序不同）
电脑端 ISP 下载软件提示：下载进行中…，几秒钟后，提示下载成功！

【方法二：复位管脚低电平复位下载】

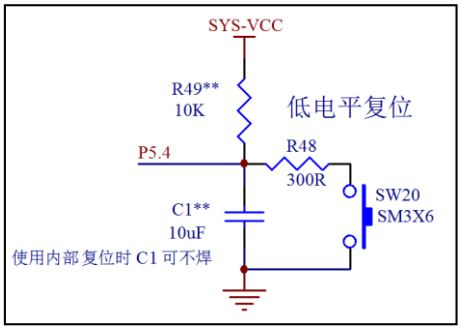
USB 连接好并已上电的情况下，外部按键复位也可进入 USB 下载模式，注意：

P5.4-nRST 出厂时默认是 P5.4-I/O 功能，要改为复位功能，需 ISP 烧录时取消设置复位脚用作 I/O 口，停电一次再上电才生效，程序区中用户程序也可改为复位脚或 I/O，这个立即生效。

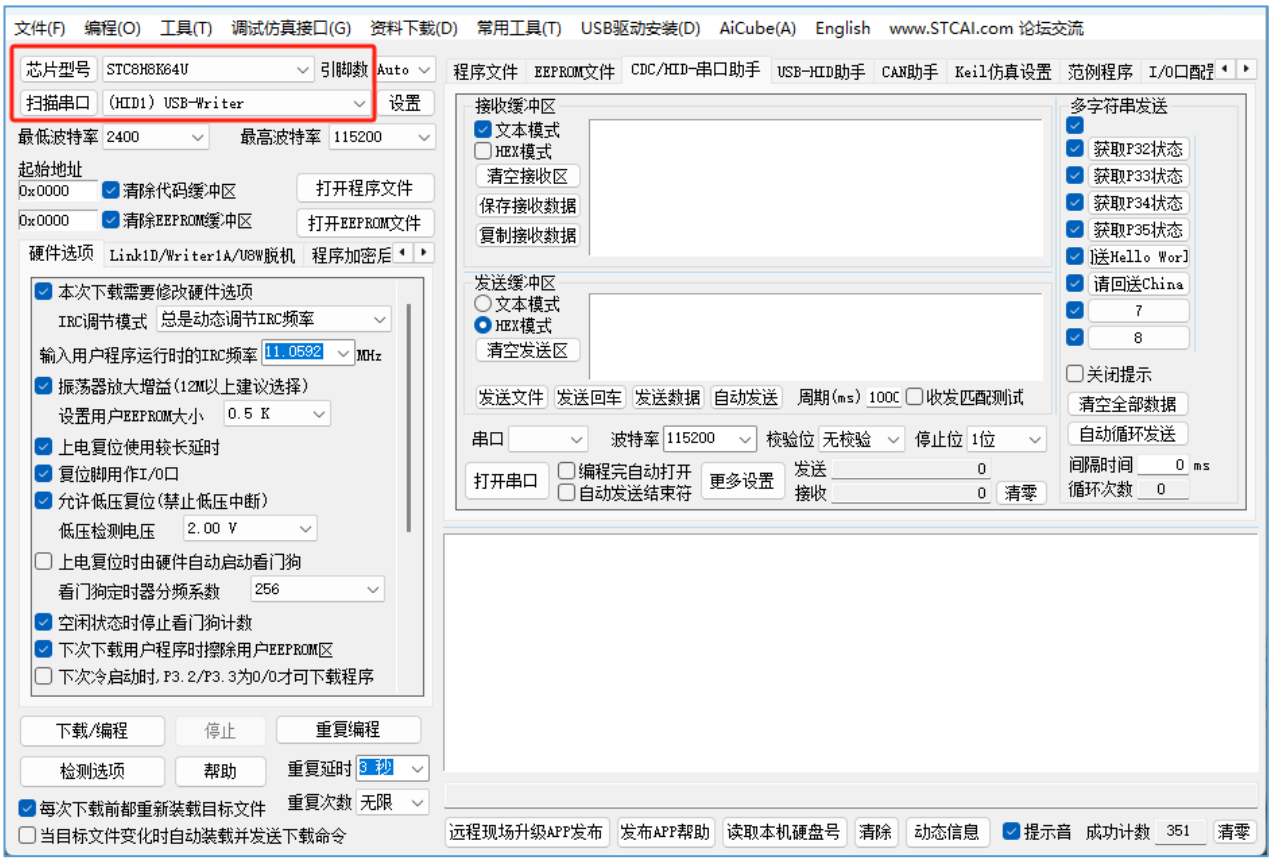
1、按下复位管脚/P5.4-nRST 外接的低电平复位按键，复位 MCU，再松开复位键，MCU 首先从系统程序区启动运行系统程序，判断是否要下载用户程序，

等待电脑端 ISP 下载软件中自动识别出“(HID1) USB Writer”后

2、点击电脑端下载软件中的【下载/编程】按钮
电脑端 ISP 下载软件提示：下载进行中…，下载成功！



备注：如果目标 MCU 的用户程序区已有 USB 通讯程序（含 CDC、HID 等），则通过复位脚复位进入 USB 下载模式不被支持，只能通过停电上电冷启动下载或者从用户程序区软复位到系统区下载



通过上述方法一或者方法二的操作步骤，我们看到 AiCube-ISP 系统软件界面的“扫描串口”的端口列表框显示“（HID1）USB-Writer”，表明 STC8H8K64U 实验箱已准备好，可以制作仿真芯片了。

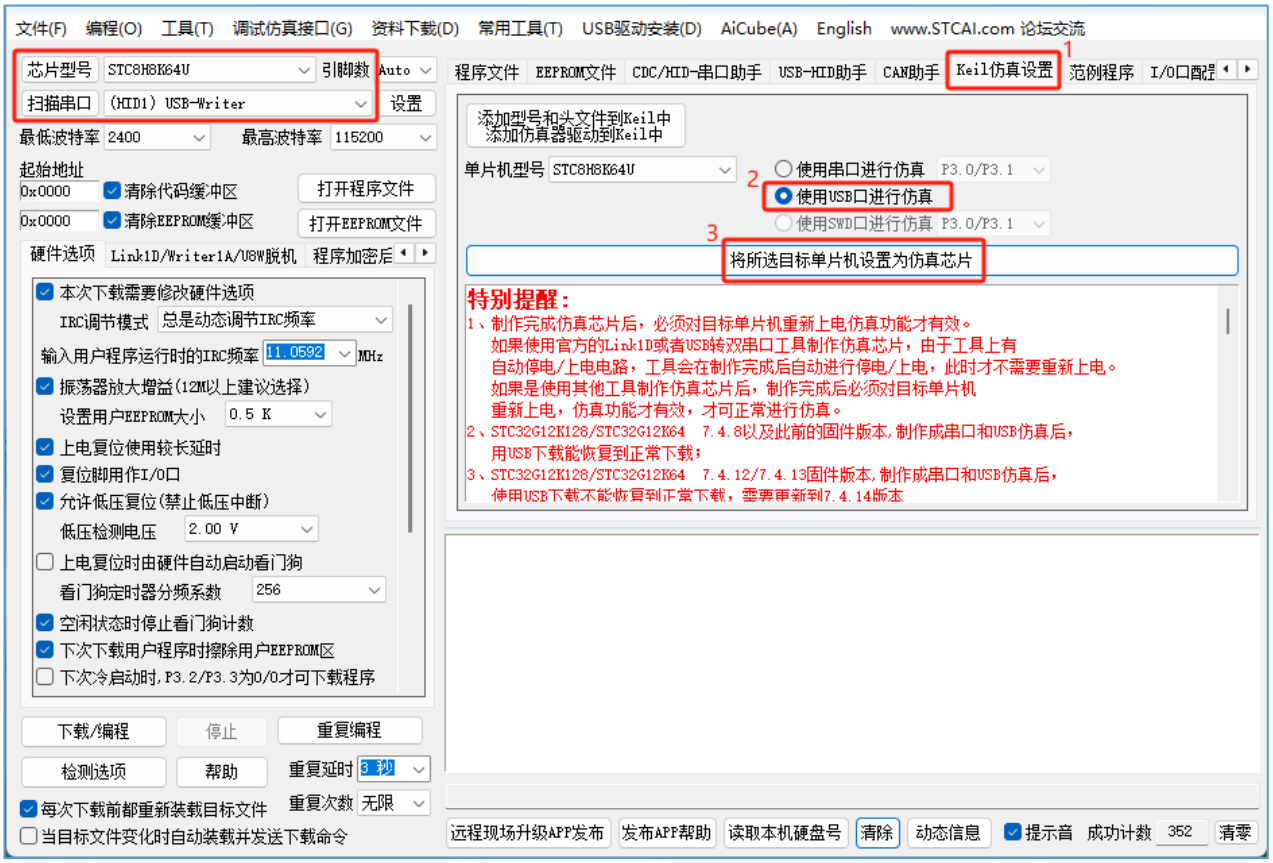
3.1.1.4　制作仿真芯片

在 AiCube-ISP 系统软件界面：

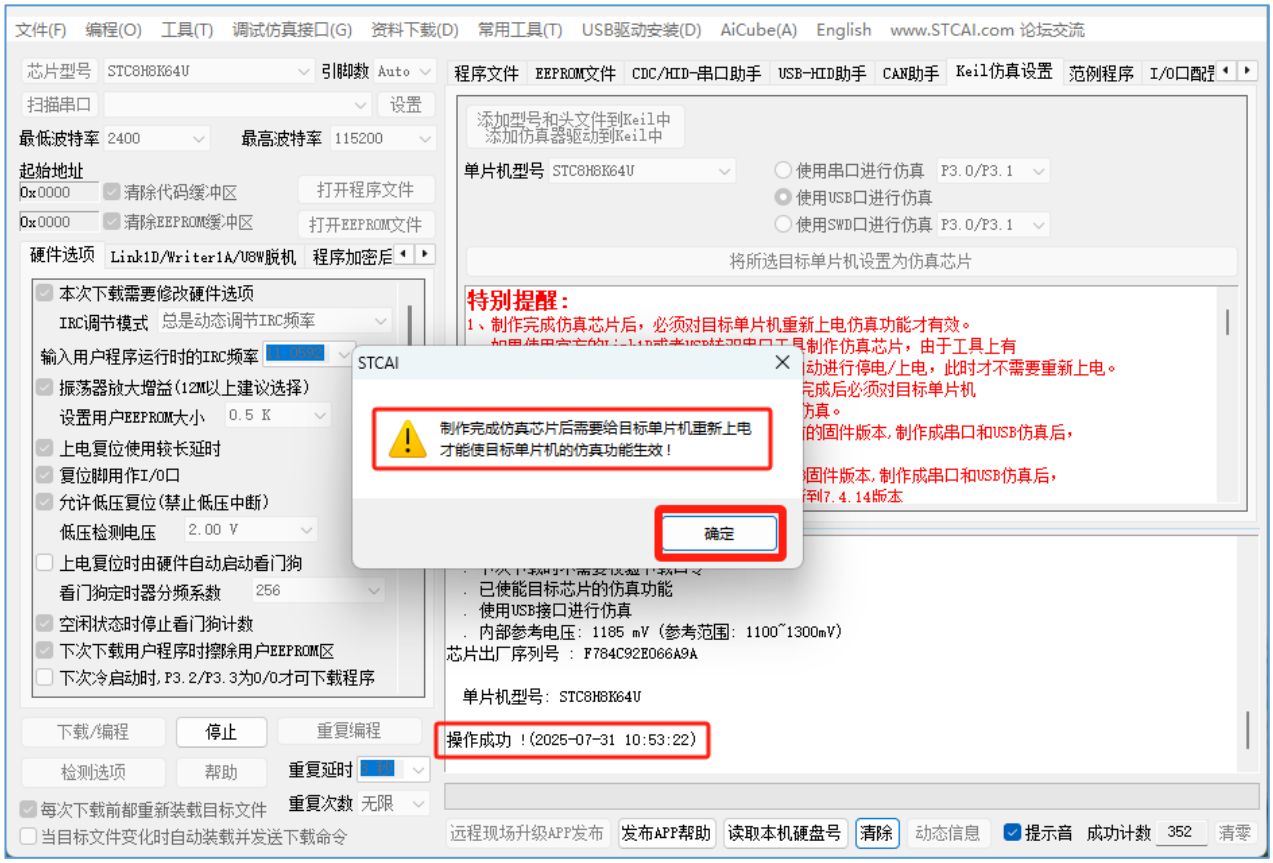
➲ 1、点击“Keil 仿真设置”标签，打开“Keil 仿真设置”选项卡界面

➲ 2、选择“使用 USB 口进行仿真”

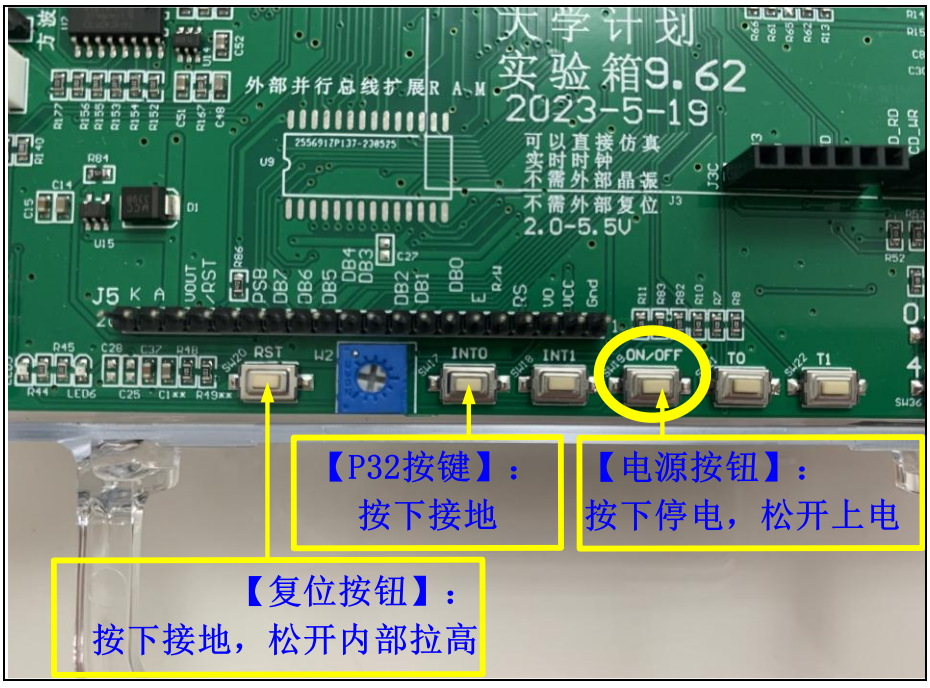
➲ 3、点击“将所选目标单片机设置为仿真芯片”



然后 AiCube-ISP 软件开始自动进行设置仿真芯片的操作，界面下面的进度条在推进，最后“操作成功!”界面出现特别提示：“制作完成仿真芯片后需要给目标单片机重新上电才能使目标单片机的仿真生效!”如下图：



所以，点击“确定”按钮后，按提示要求【按一下 STC8H8K64U 实验箱的“Power_SW”按钮】，给目标单片机重新上电才能使目标单片机的仿真生效！下图为 STC8H8K64U 实验箱正下方放大图：

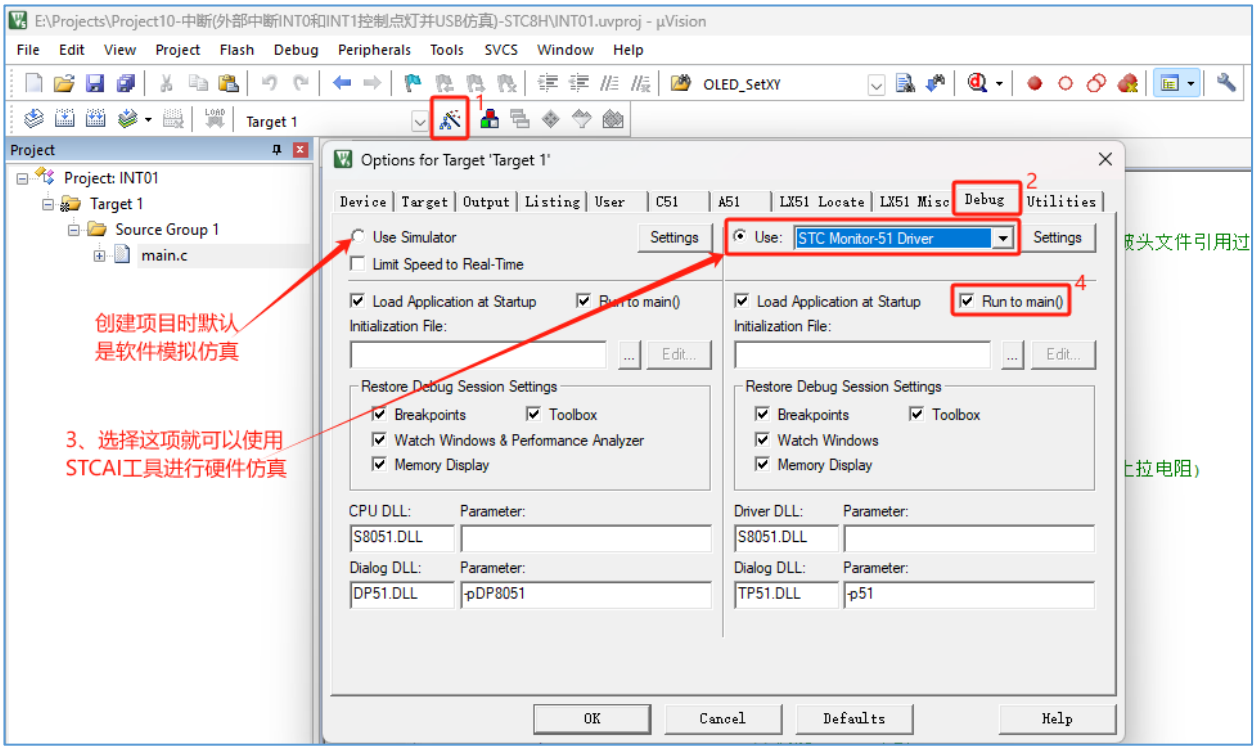


3.1.2　Debug 和 Utilities 选项设置--在 Keil 集成环境中

仿真芯片制作完成后，打开要仿真的项目 Keil 文件，在菜单栏下方找到一个类似魔术棒的工具，点击打开项目选项设置界面，在 Keil 集成环境中进行与 Debug 相关的必要设置。

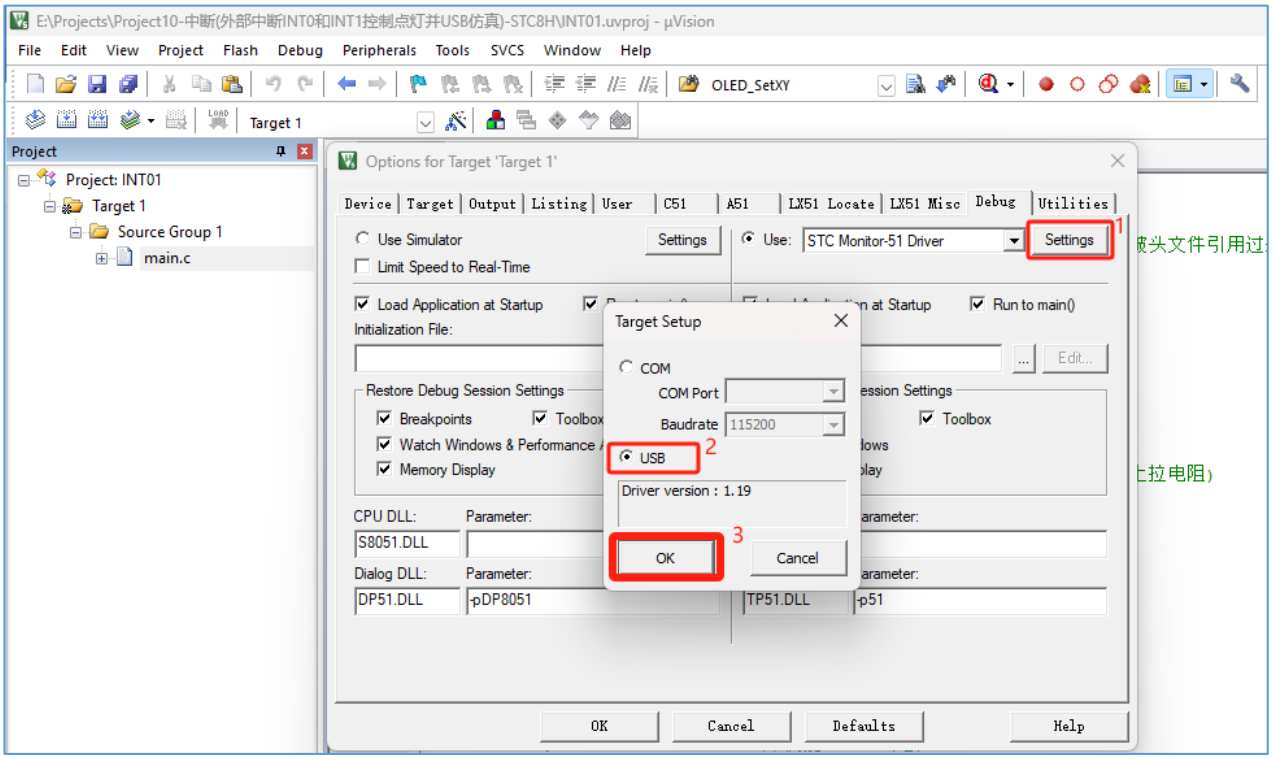
3.1.2.1　Debug 选项卡--使用 STCAI 工具进行硬件仿真设置

切换到 Debug 选项卡，创建项目时，Keil 默认是软件模拟仿真。我们可以借助 STCAI 的工具进行硬件仿真，如下图：



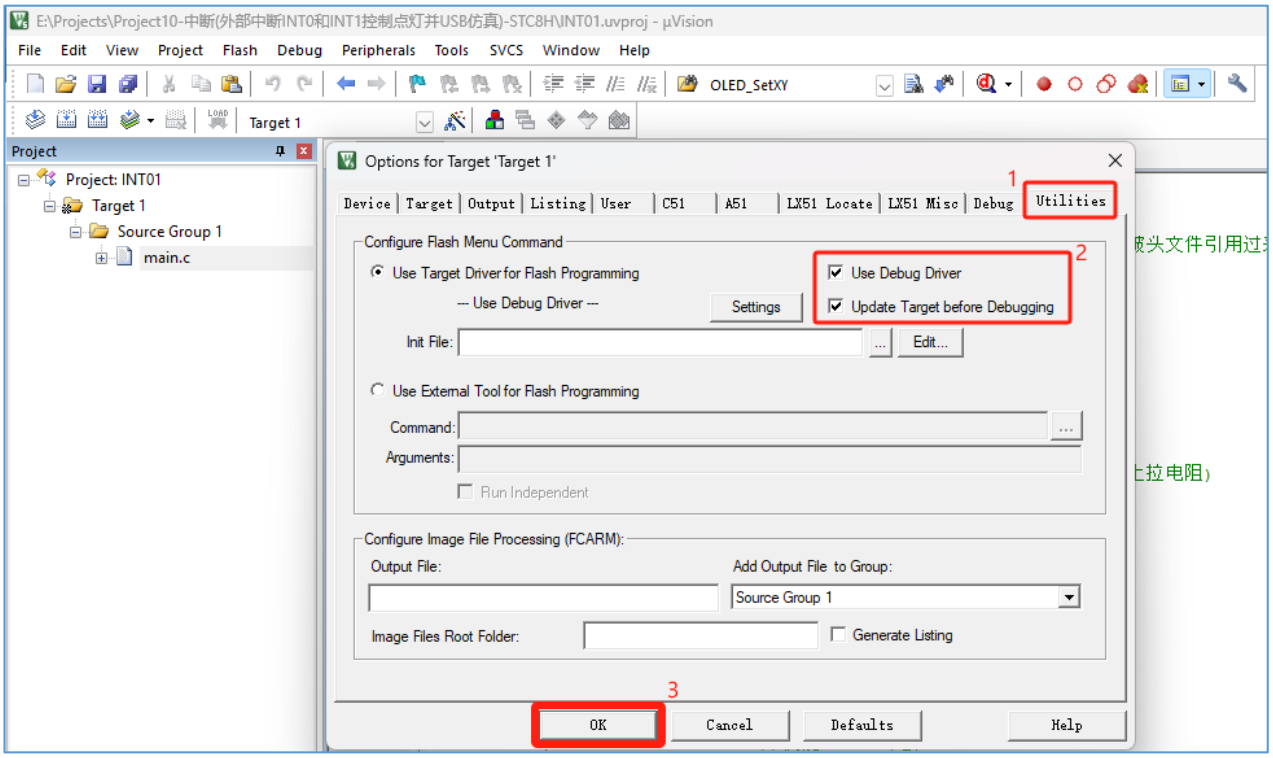
STC8H8K64U 既可以 USB 直接通信仿真，也可以 COM 串口通信仿真。

本文介绍通过 STC8H8K64U 实验箱中 **STC8H8K64U 芯片自带的硬件 USB 通信仿真**，如下图设置：



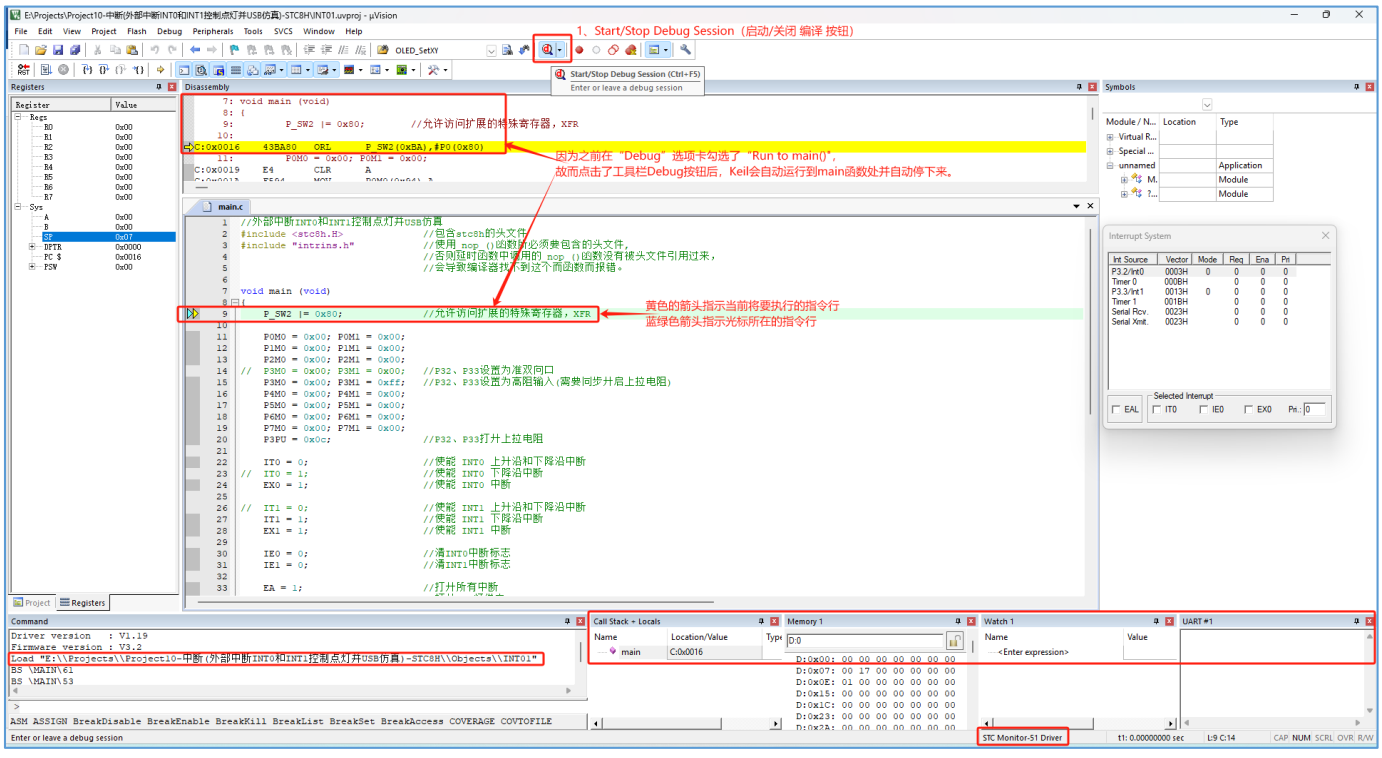
3.1.2.2 Utilities 选项卡--勾选“Use Debug Driver”

切换到“Utilities”选项卡，勾选“Use Debug Driver”，点击“OK”，完成设置。



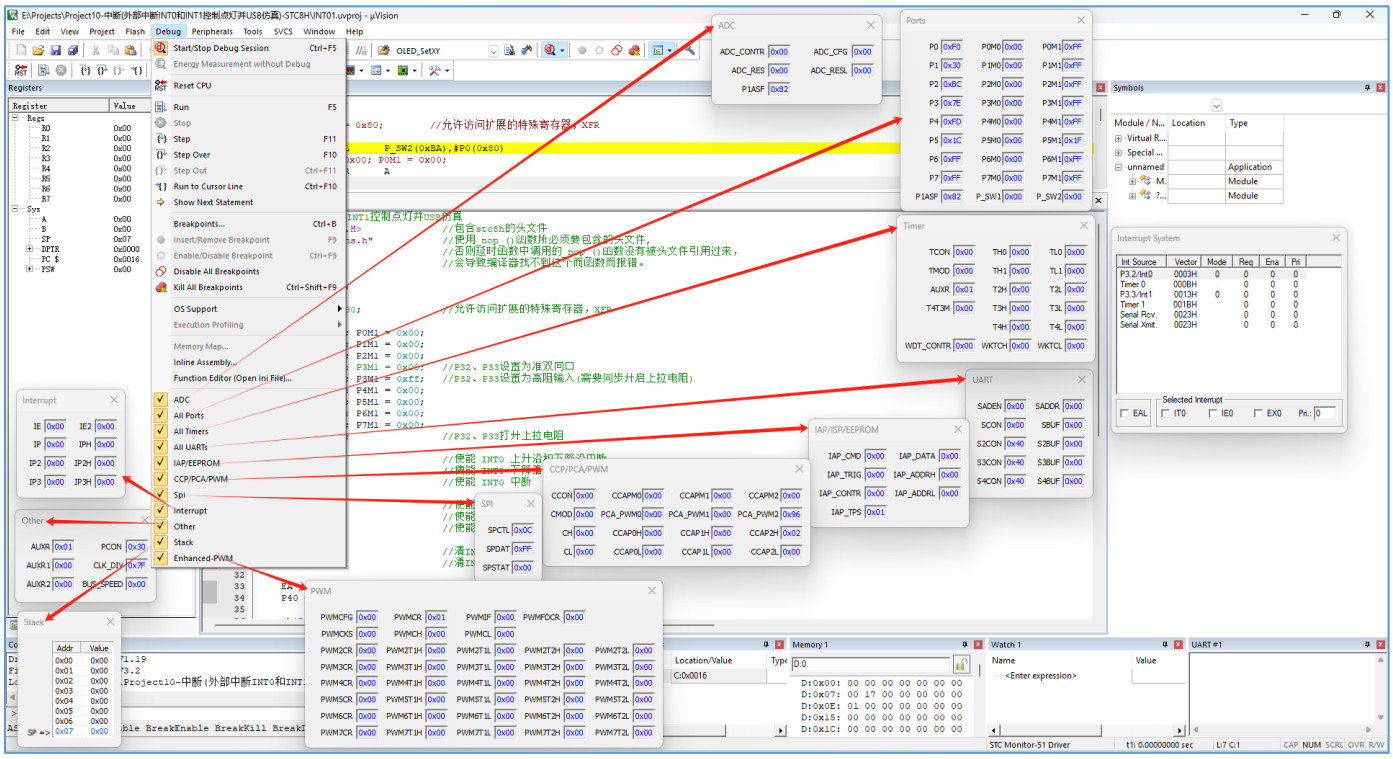
3.1.3 进入 Debug 界面，观察实验现象

设置完毕，返回 Keil 界面，代码程序编译无误后，点击工具栏的“Debug”（启动/关闭编译调试操作界面）按钮后，Keil 软件进入仿真界面。如下图：



➲ **步骤一：利用仿真，可以设置运行到断点处，捕捉当前程序运行位置，INT0/P3.2 是下降沿还是上升沿**

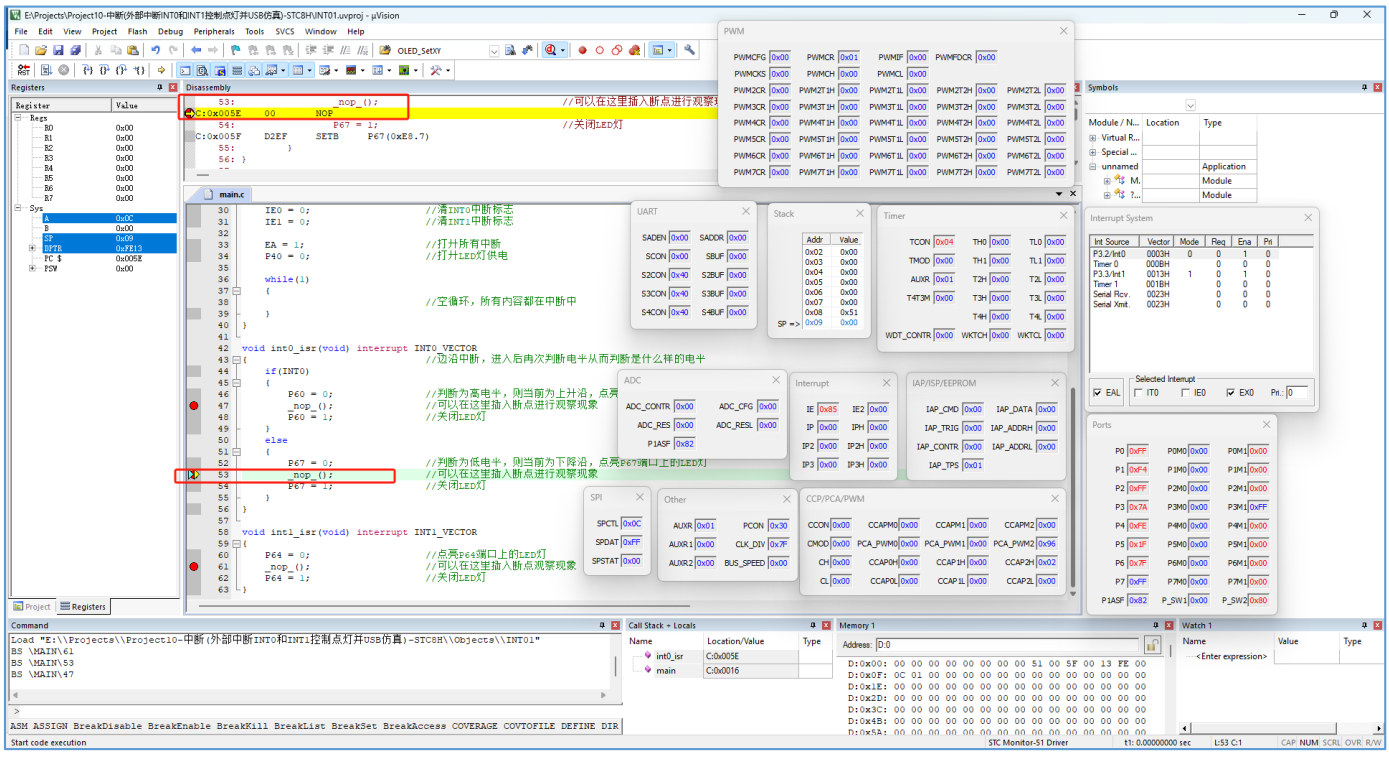
➢ **为方便观察实验现象，需在中断服务程序中设置如下 3 个断点。**当然，你也可以打开全部或部分外设窗口观察各外设数据在 Debug 过程中的变化：



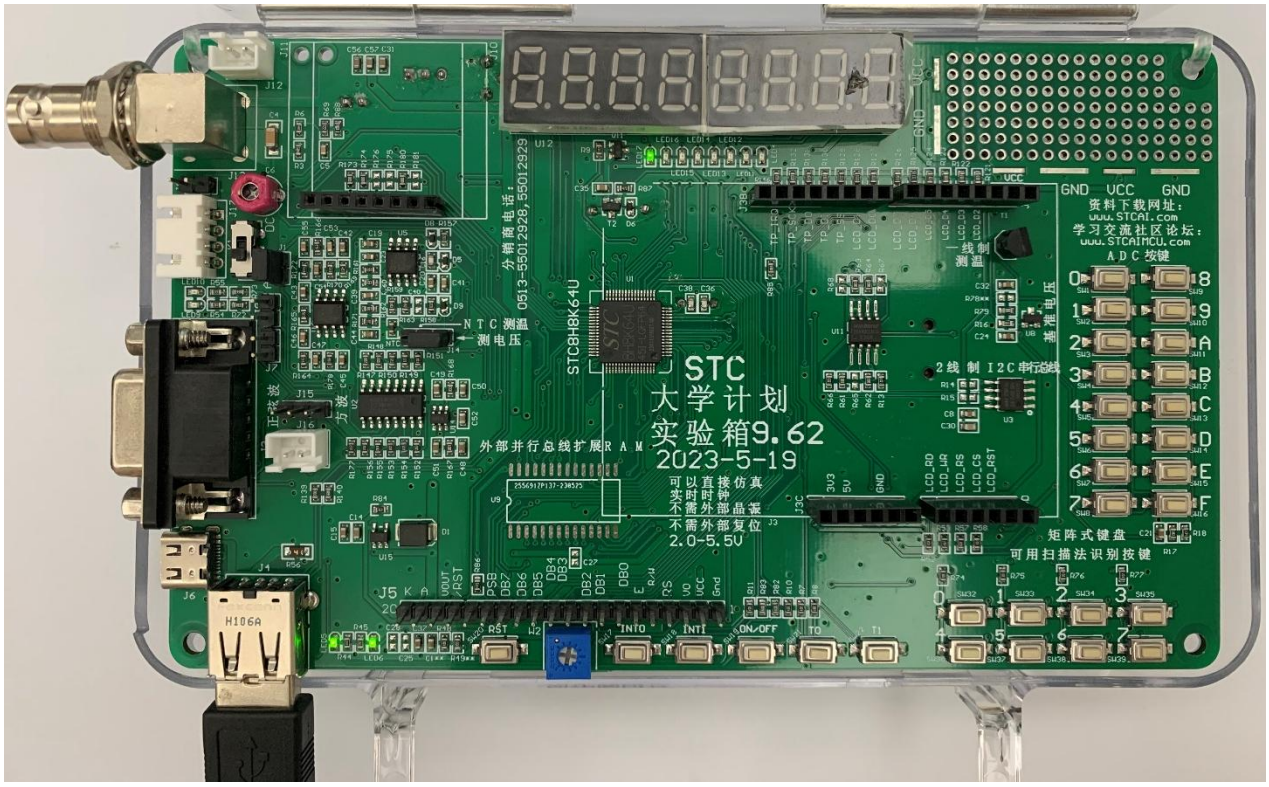
➢ 先点击 Run（运行）按钮，再按下 P3.2 按键，保持不松开（为了验证下一步的实验）：

✓ 程序会停在如下图所示断点处。

✓ 同时，相较于上图，我们观察到：Ports / Timer / Interrupt / Interrupt System / Call Stack /Memory1 / Registers / Disassembly 这些窗口显示的数据都有可见的变化。



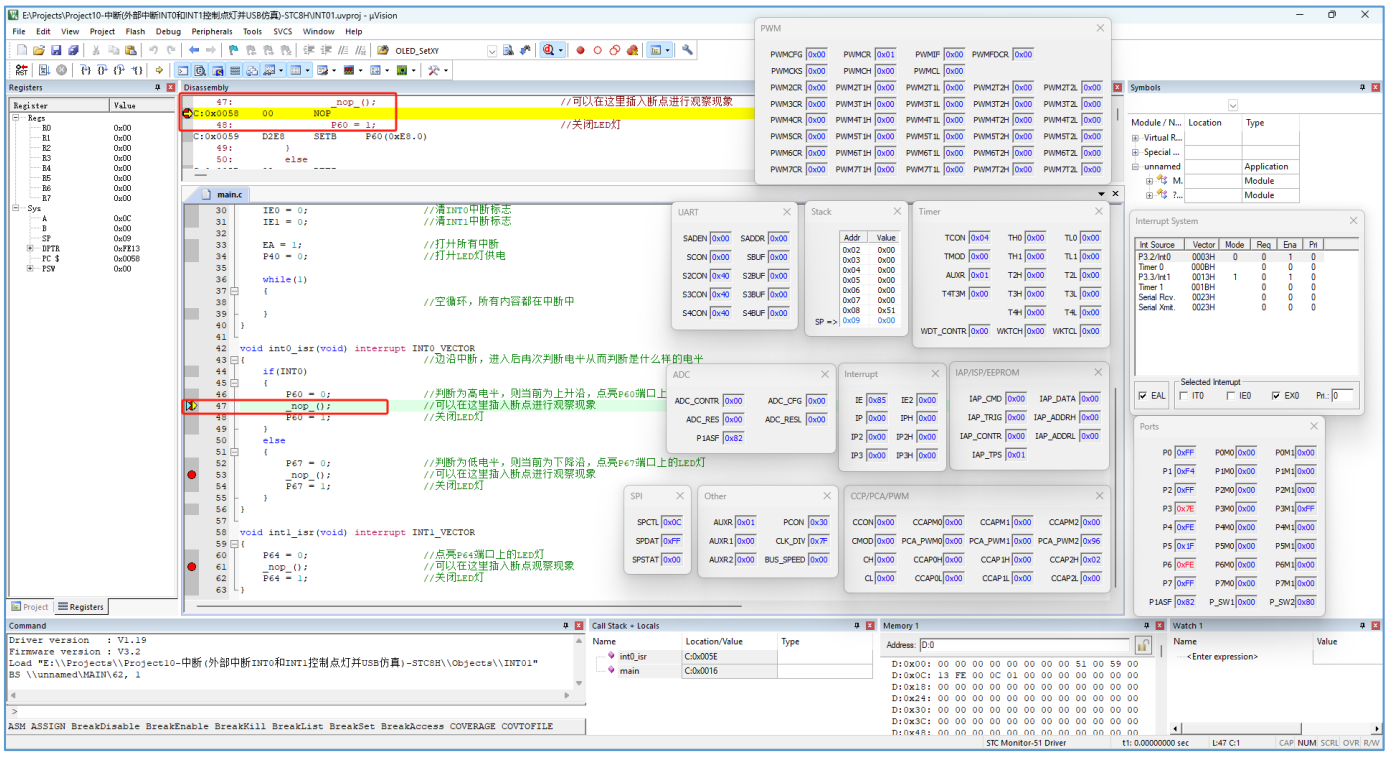
✓ 进 INT0 中断后，读到 INT0/P3.2 口是 低，下降沿，点亮 P6.7，STC8H8K64U 实验箱如下图：



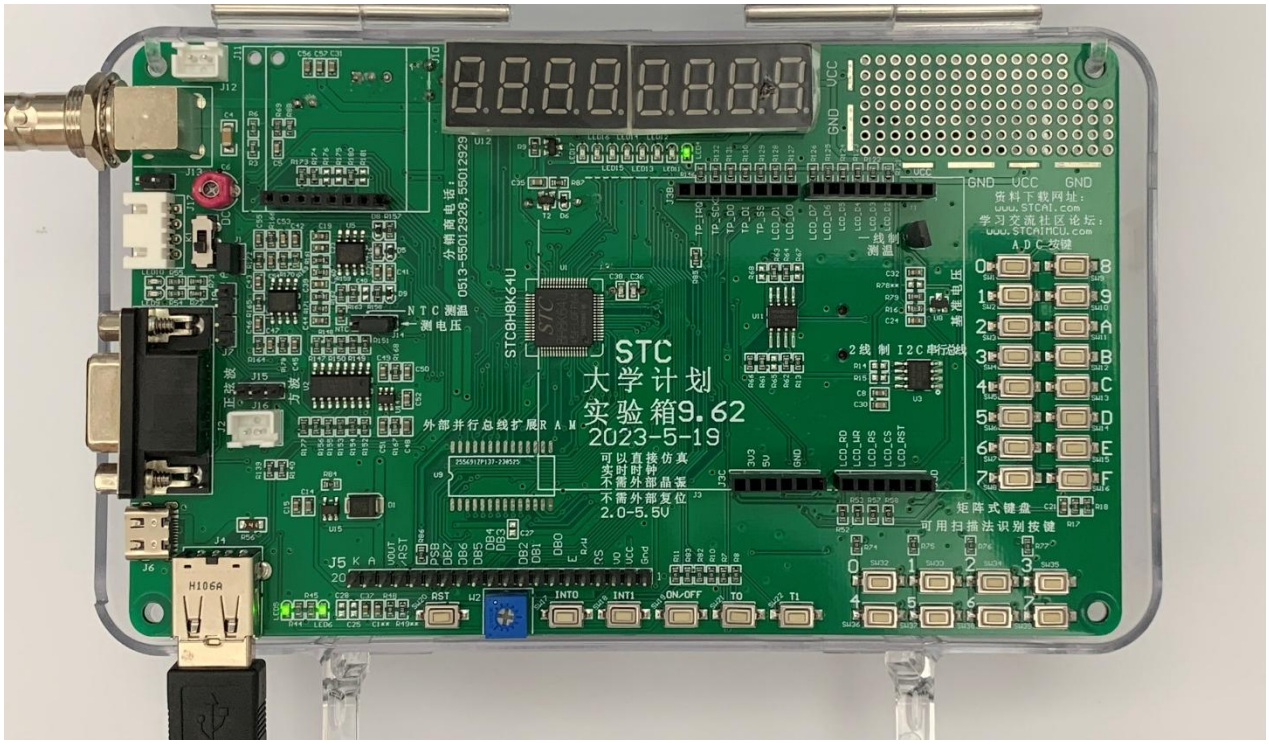
➢ 继续点击 Run（运行）按钮运行，退出中断服务程序。再松开 P3.2 按键：

✓ 程序会停在如下图所示断点处。

✓ 同时，相较于上图，我们观察到：Ports /Memory1 / Registers / Disassembly 这些窗口显示的数据都有可见的变化。



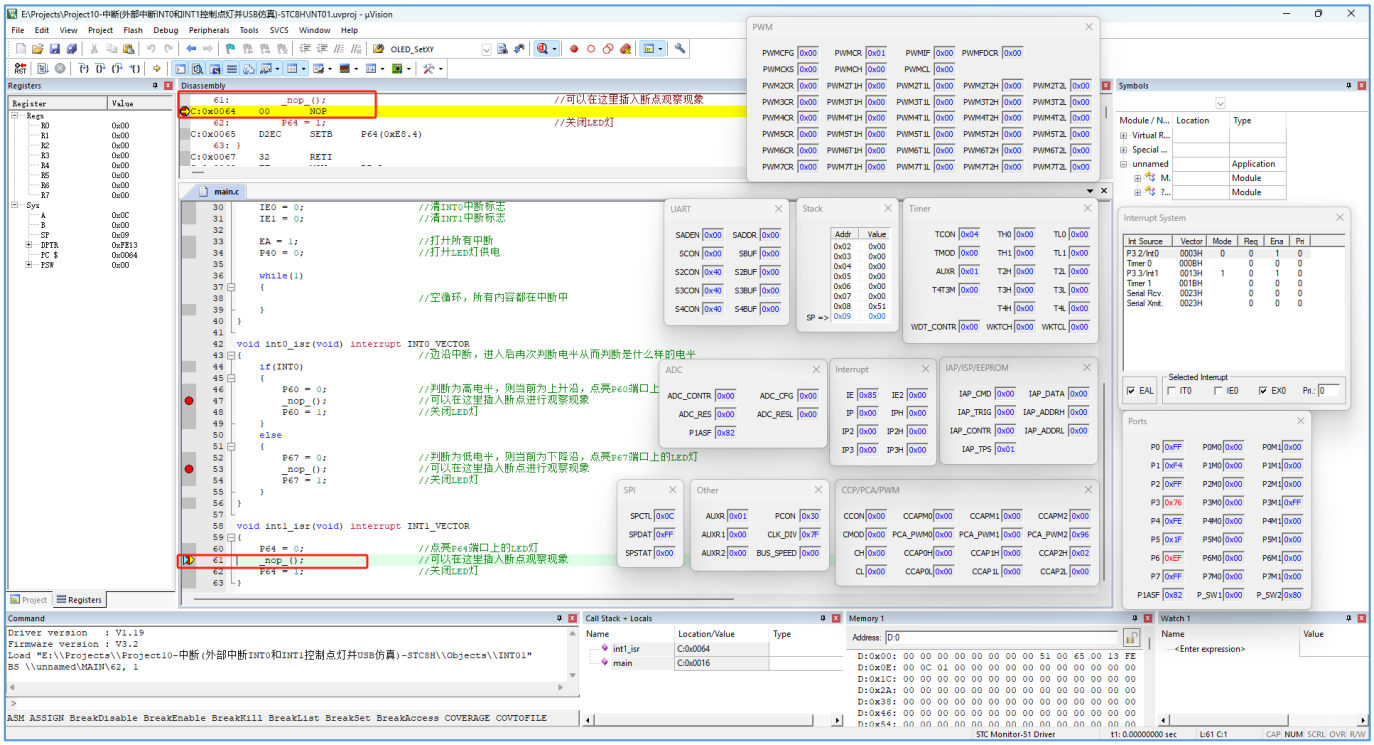
✓ 进 INT0 中断后，读到 INT0/P3.2 口是 高，上升沿，点亮 P6.0，STC8H8K64U 实验箱如下图：



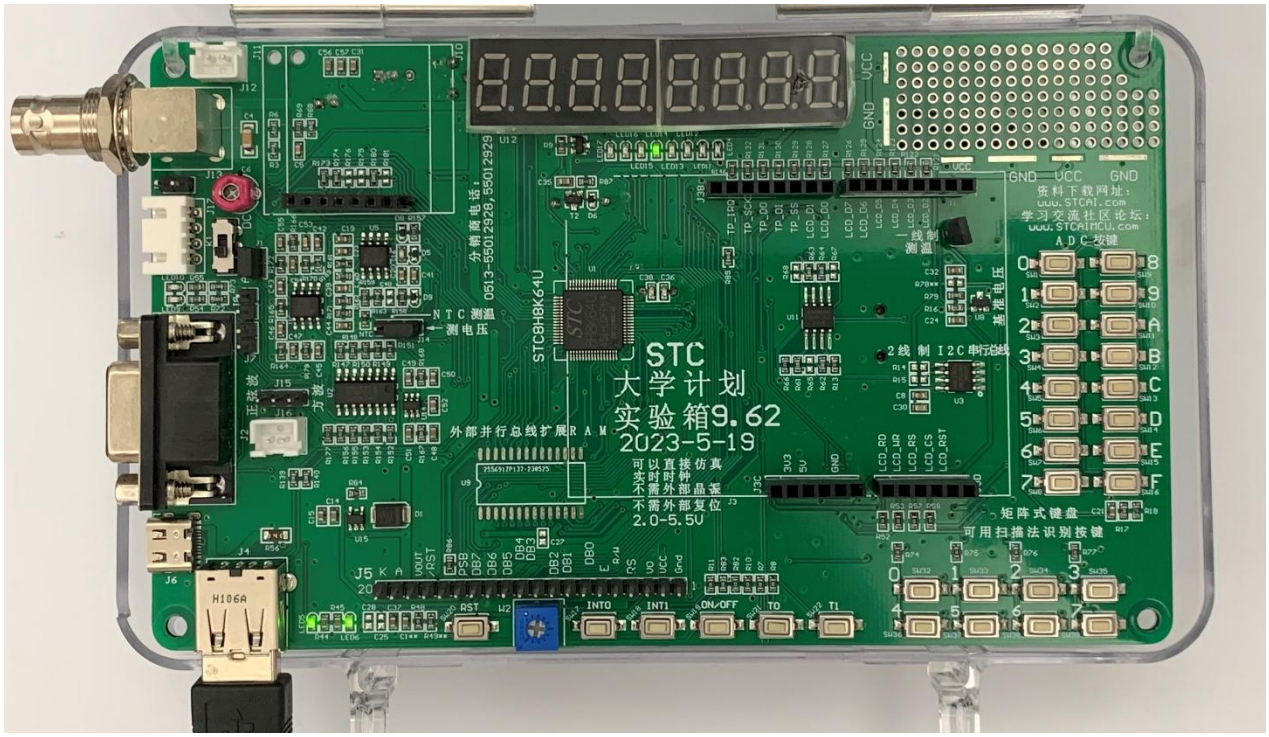
➢ 继续点击 Run（运行）按钮运行，退出中断服务程序。

➲ 步骤二：利用仿真，可以设置运行到 INT1/P3.3 外部中断的断点处

- ➢ 如果程序没有在运行，先点击 Run（运行）按钮，再按一下 P3.3 按键：
  - ✓ 程序会停在如下图所示断点处。同时，相较于上图，我们观察到：Ports / Call Stack /Memory1 / Registers / Disassembly 这些窗口显示的数据都有可见的变化。



  - ✓ 进 INT1 中断后，点亮 P6.4，STC8H8K64U 实验箱如下图：



- ➢ 继续点击 Run（运行）按钮运行，退出中断服务程序。

完成本实验。

3.2　串口通信仿真 -- STC8H8K64U 实验箱仿真

在编写较大的程序时，往往会遇到代码编译没问题，但运行异常的情况，或者要观察项目运行状况，这时就可以借助仿真来查找问题，进行程序调试，或者对项目实时监控，进行逻辑验证。

STC8H 可支持仿真功能。STC8H 芯片出厂时默认仿真功能是关闭的，需要使用 AiCube-ISP 软件将目标单片机设置为仿真芯片，才能开启芯片的仿真功能。

STC8H 均支持串口通信仿真，其中 STC8H2K12U、STC8H2K32U、STC8H8K64U 芯片既可 USB 直接通信仿真，也可串口通信仿真。

串口直接通信仿真，不需要任何仿真器，单片机本身就是仿真器，直接与电脑的串口相连接就可进行仿真。注意：由于目前的电脑均无硬件串口，所以需要使用 USB 转串口工具将目标芯片和电脑相连接。本文档使用的 STCAI 公司提供的“**USB 转双串口工具@Ai8H2K12U**”。

下面我们就以“中断实验 1(外部中断 INT0/P3.2，INT1/P3.3 控制点灯并 USB 仿真)”实验为例，使用 **STC8H8K64U 实验箱仿真 -- 串口通信仿真**，并观察实验现象。

3.2.1　安装 AiCube-ISP 软件，制作仿真芯片

3.2.1.1　安装 AiCube-ISP 下载/编程/烧录 工具软件

安装 STCAI 公司研发的 AiCube-ISP 下载 / 编程 / 烧录 / 软件，含强大的辅助开发工具包。登录 STCAI 官网，下载最新版的 AiCube-ISP 安装包，下载链接如下：https://www.stcai.com/gjrj

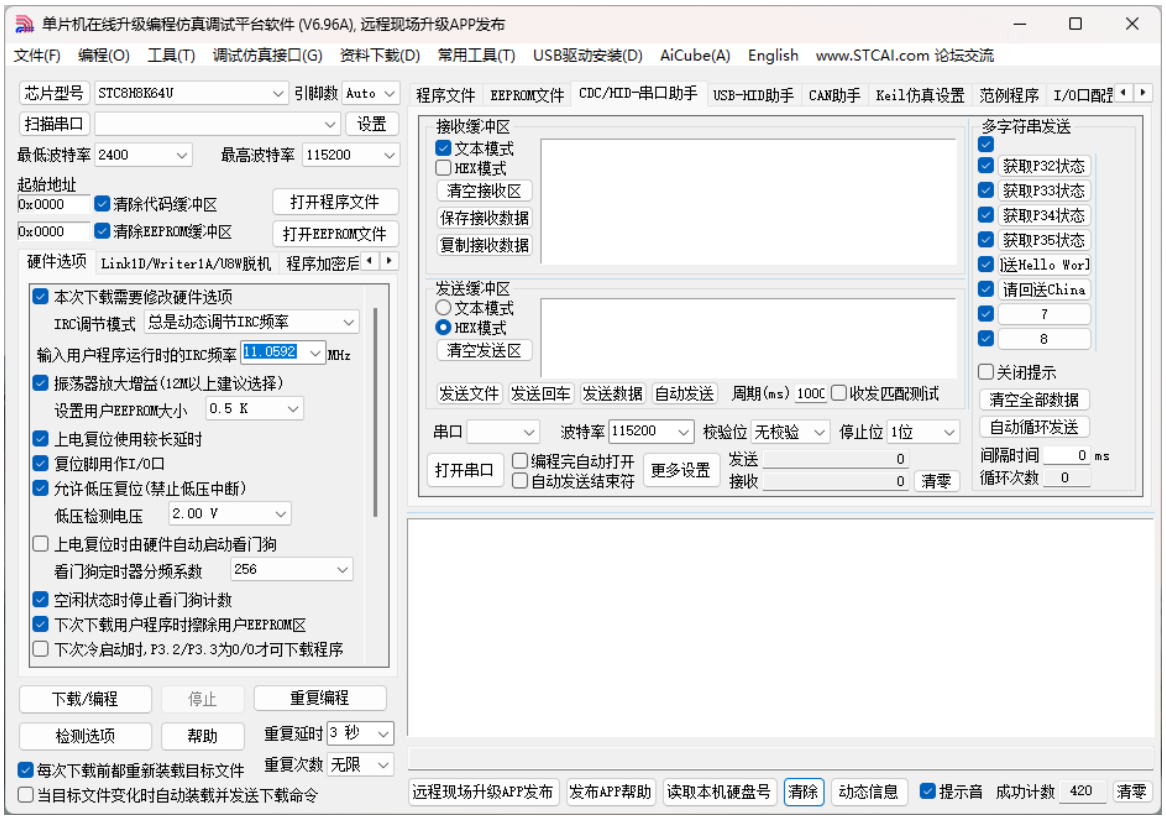


下载下来的是压缩包文件图标和解压缩后的 AiCube-ISP 文件图标，如下图：

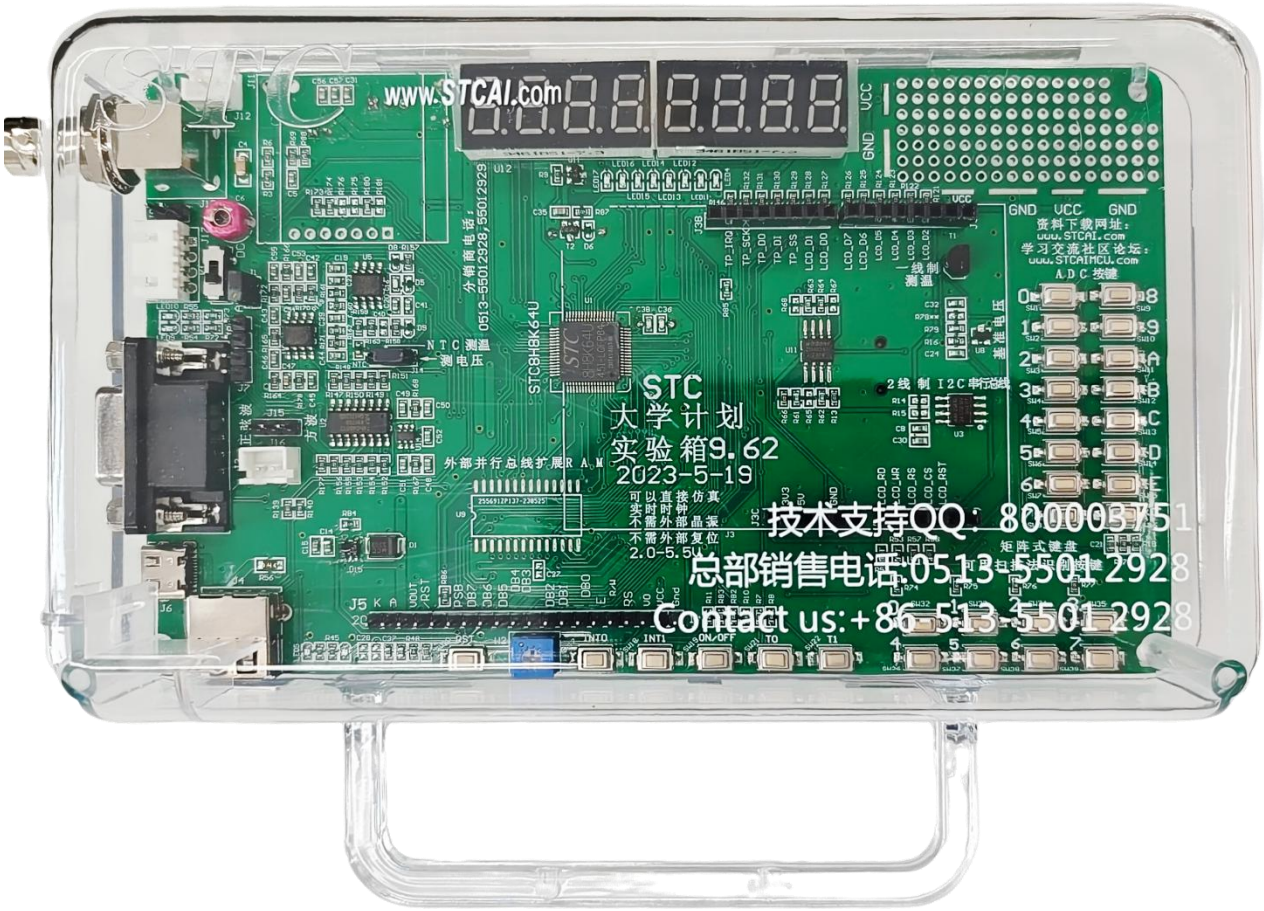


如果是解压缩到非桌面的其他目录，如 D:\STCAI-ISP，建议创建快捷方式到桌面方便打开。这实际是一个免安装的绿色软件。

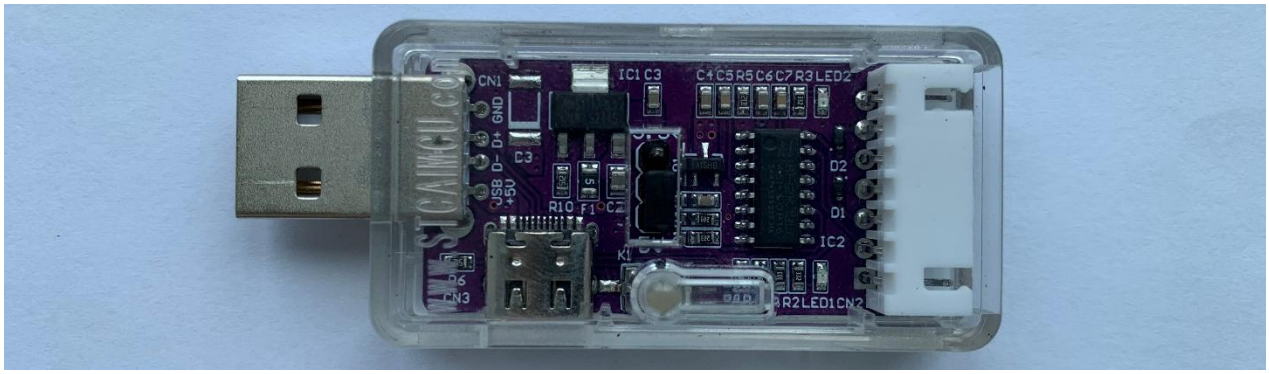
双击 AiCube-ISP 系统软件图标，打开 AiCube-ISP 软件界面，即可直接使用，如下图：



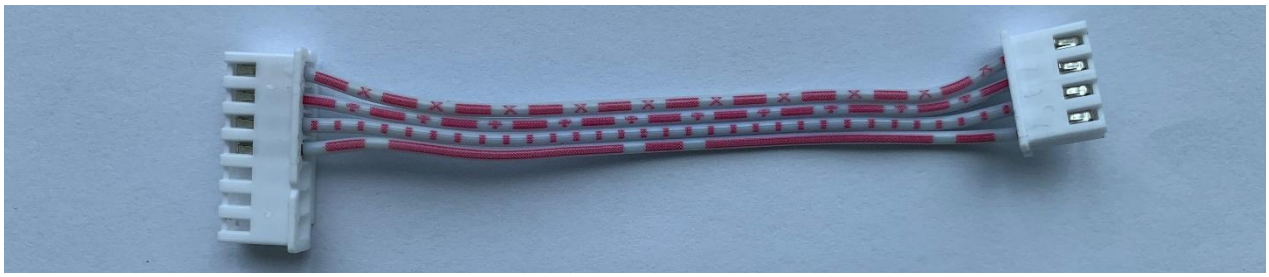
3.2.1.2　　STC8H8K64U 实验箱完整实物图（含外壳）：正面



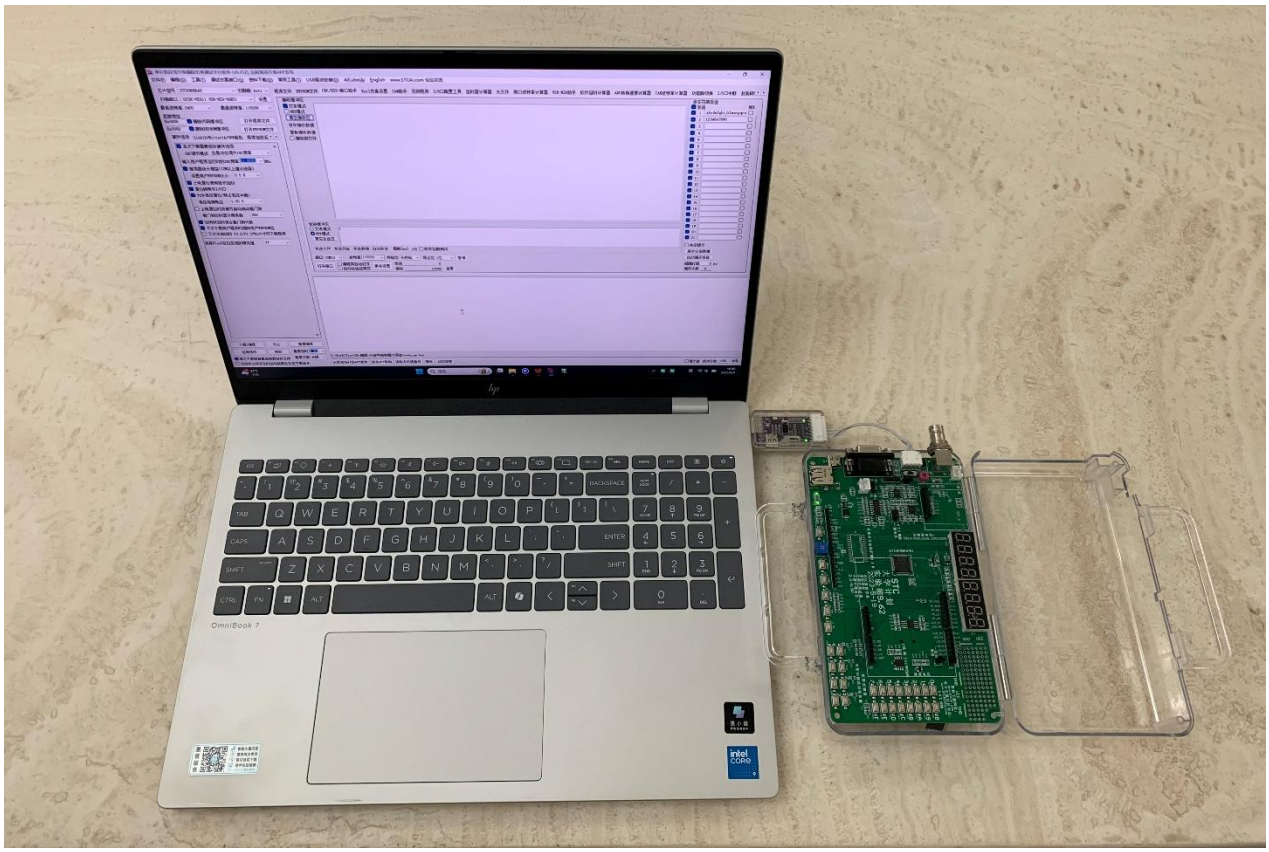
3.2.1.3 USB 转双串口工具@Ai8H2K12U：正面



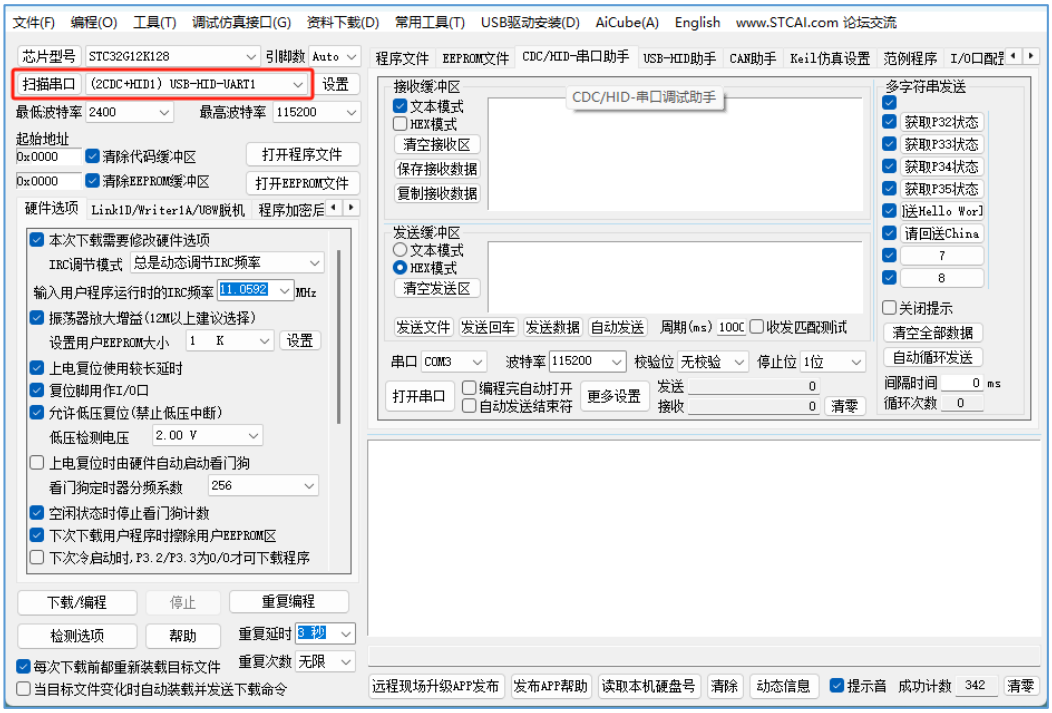
3.2.1.4 USB 转双串口/TTL 连接线：正面



3.2.1.5 STC8H8K64U 实验箱通过 USB 转双串口工具与电脑连接实物图



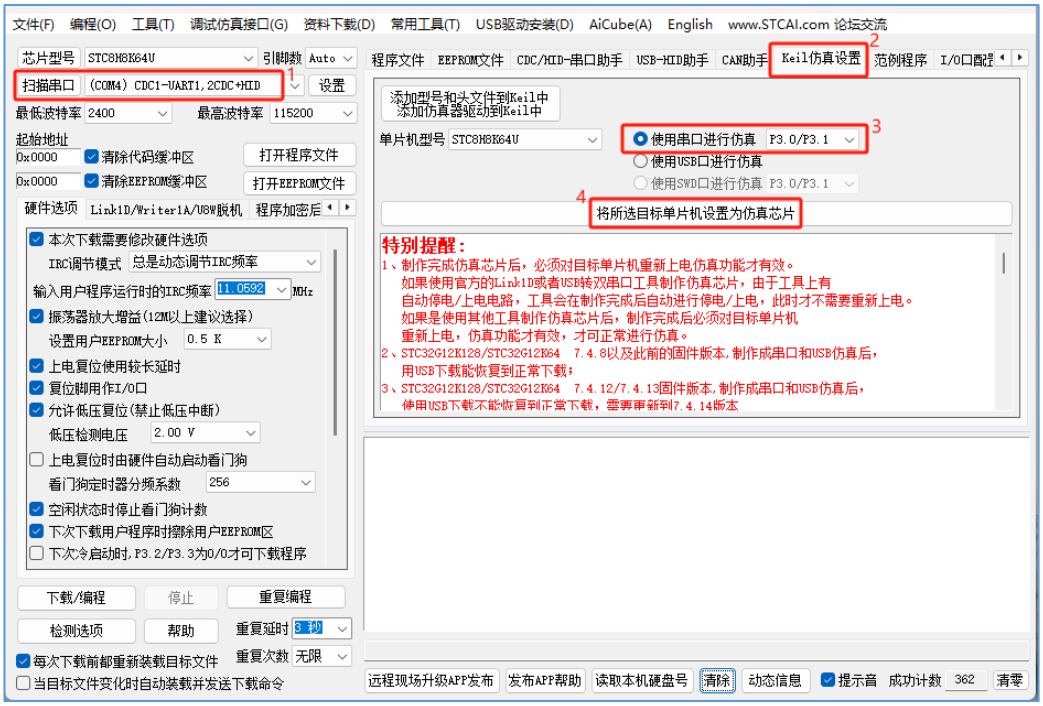
现在运行 AiCube-ISP-v6.95Z 或以上版本。选择单片机型号：STC8H8K64U，如下图，我们看到“扫描串口”后面的端口列表框显示：（2CDC+HID1）USB-HID-UART1”，如下图。



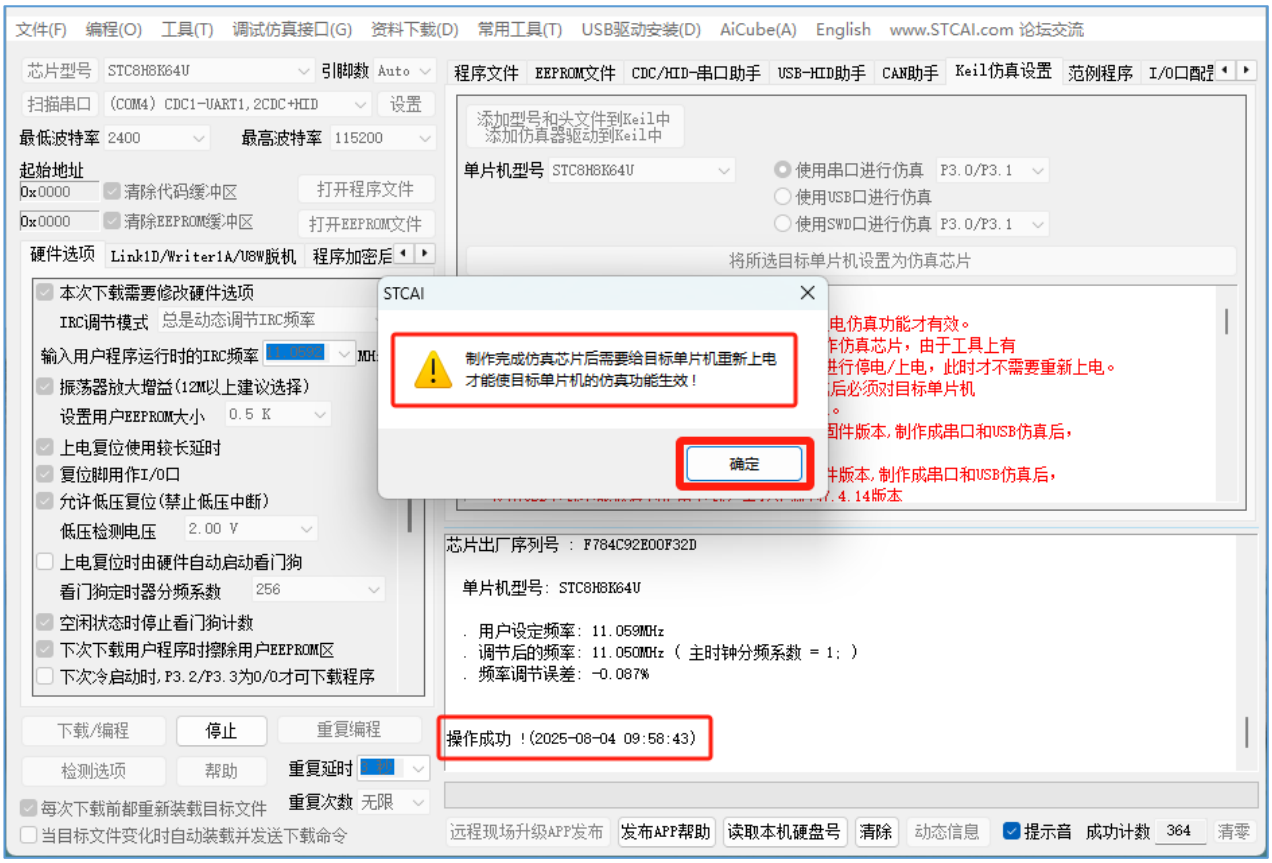
3.2.1.6　制作仿真芯片

在 AiCube-ISP 系统软件界面：

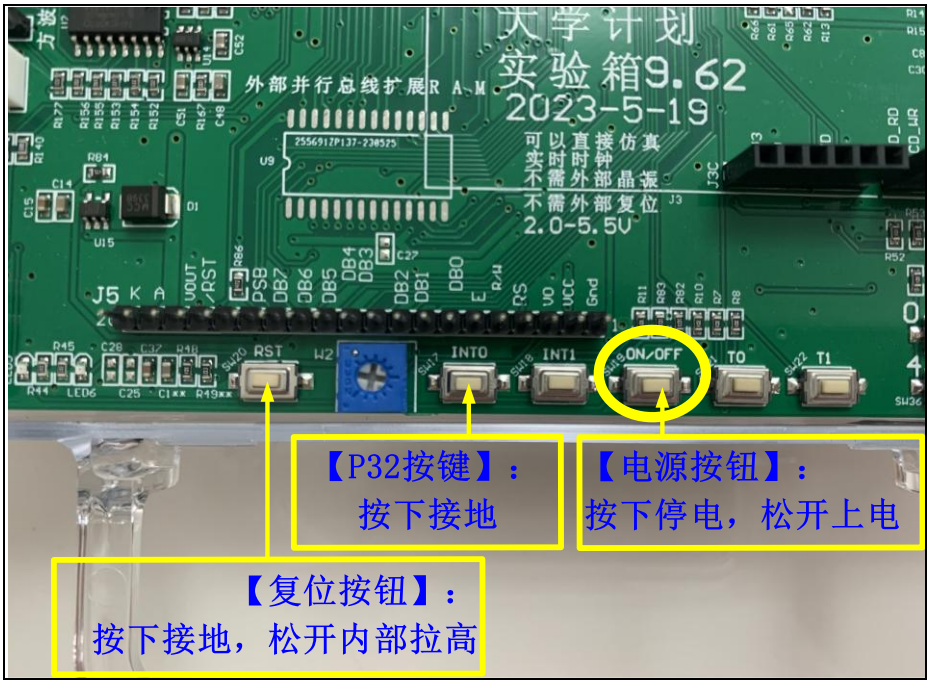
- 在“扫描串口”后面的端口列表框选择“（COM4）**CDC1-UART1**，2CDC+HID”。COM 编号会因电脑不同而分配不同的串口号，但必须选择 CDC1-UART1 对应的串口号。
- 点击“Keil 仿真设置”标签，在“Keil 仿真设置”选项卡：
  选择“使用串口进行仿真”：默认为“P3.0/P3.1”（那么程序里就不能出现任何占用 P3.0/P3.1 引脚的功能，比如：USB-CDC 功能）
  点击“将所选目标单片机设置为仿真芯片”



然后 AiCube-ISP 软件自动进行设置仿真芯片的操作，界面下面的进度条在推进，最后“操作成功!”界面出现特别提示：“制作完成仿真芯片后需要给目标单片机重新上电才能使目标单片机的仿真生效!”如下图：



所以，点击“确定”按钮后，按提示要求【按一下 STC8H8K64U 实验箱上的“Power_SW”按钮】，给目标单片机重新上电才能使目标单片机的仿真生效！下图为 STC8H8K64U 实验箱正下方放大图：

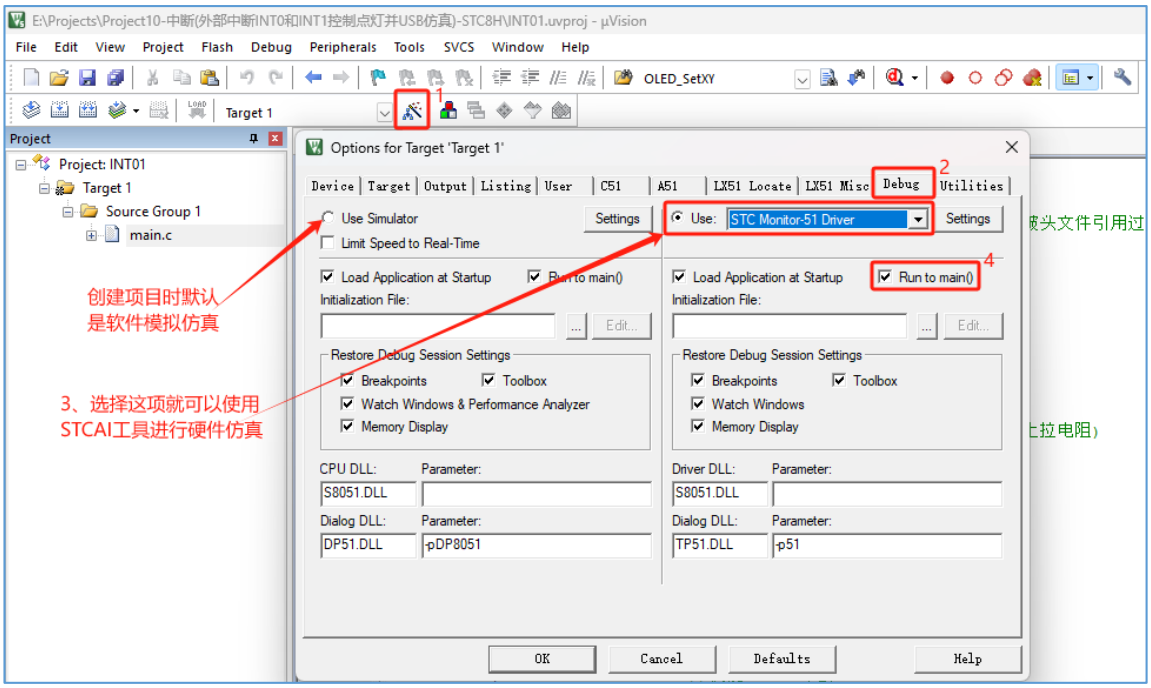


3.2.2 Debug 和 Utilities 选项设置--在 Keil 集成环境中

仿真芯片制作完成后，打开要仿真的项目 Keil 文件，在菜单栏下方找到一个**类似魔术棒的工具**，点击打开项目选项设置界面，在 Keil 集成环境中进行与 Debug 相关的必要设置。

3.2.2.1 Debug 选项卡--使用 STCAI 工具进行硬件仿真设置

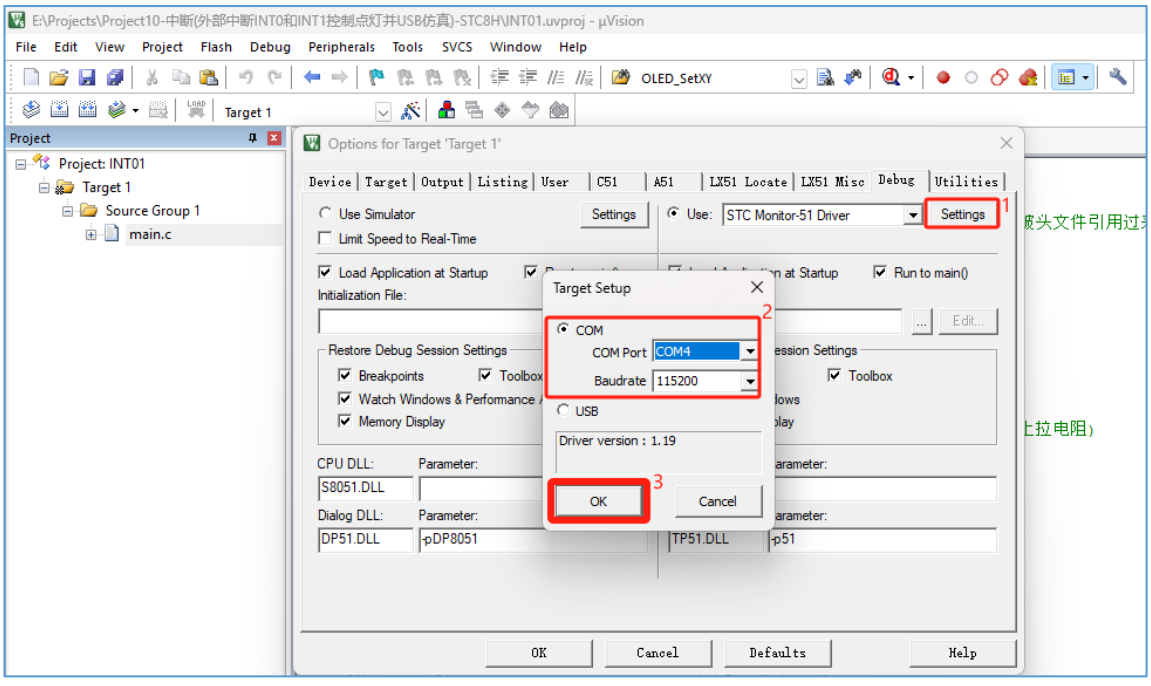
切换到 Debug 选项卡，创建项目时，Keil 默认是软件模拟仿真。我们可以借助 STCAI 的工具进行硬件仿真，如下图：



STC8H8K64U 既可以 USB 直接通信仿真，也可以 COM 串口通信仿真。

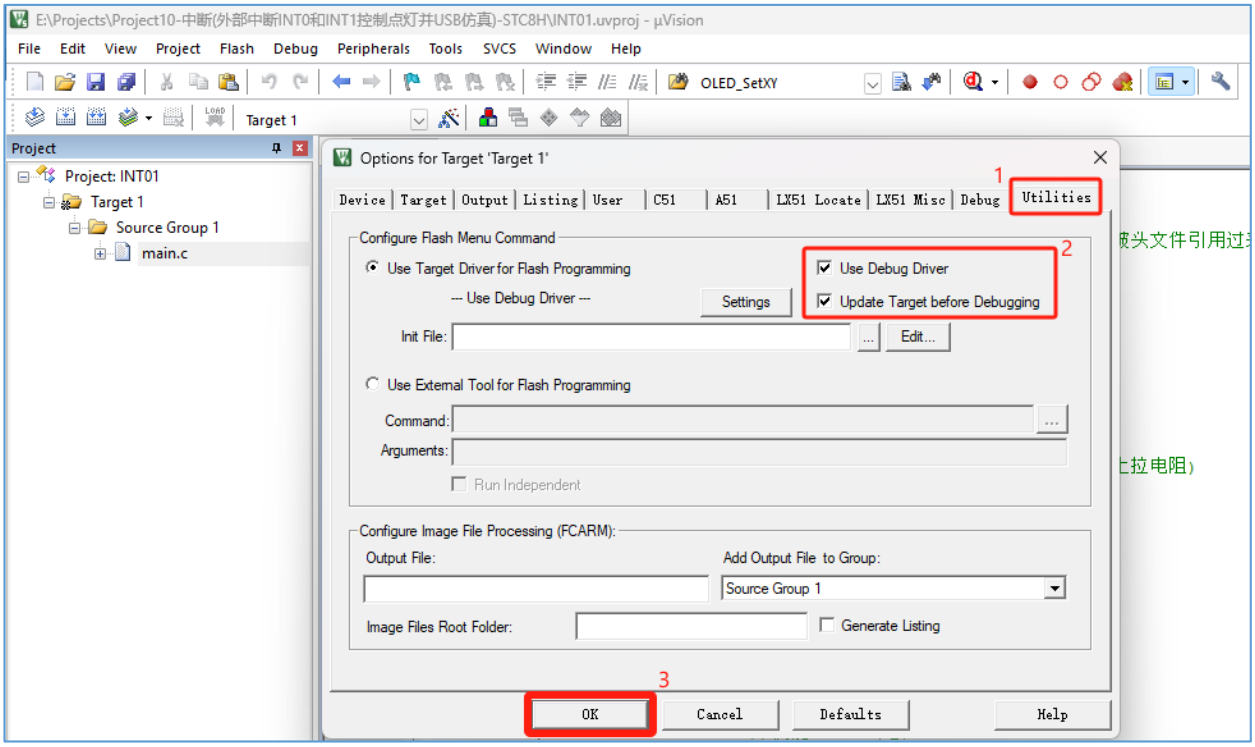
本文介绍 STC8H8K64U 实验箱通过 USB 转双串口工具进行**串口通信仿真**，如下图设置，点选“COM”：

- COM Port：COM4（与 AiCube-ISP 界面中“扫描串口”端口列表框中 CDC1 对应的 COM4 一致）
- Baudrate：115200



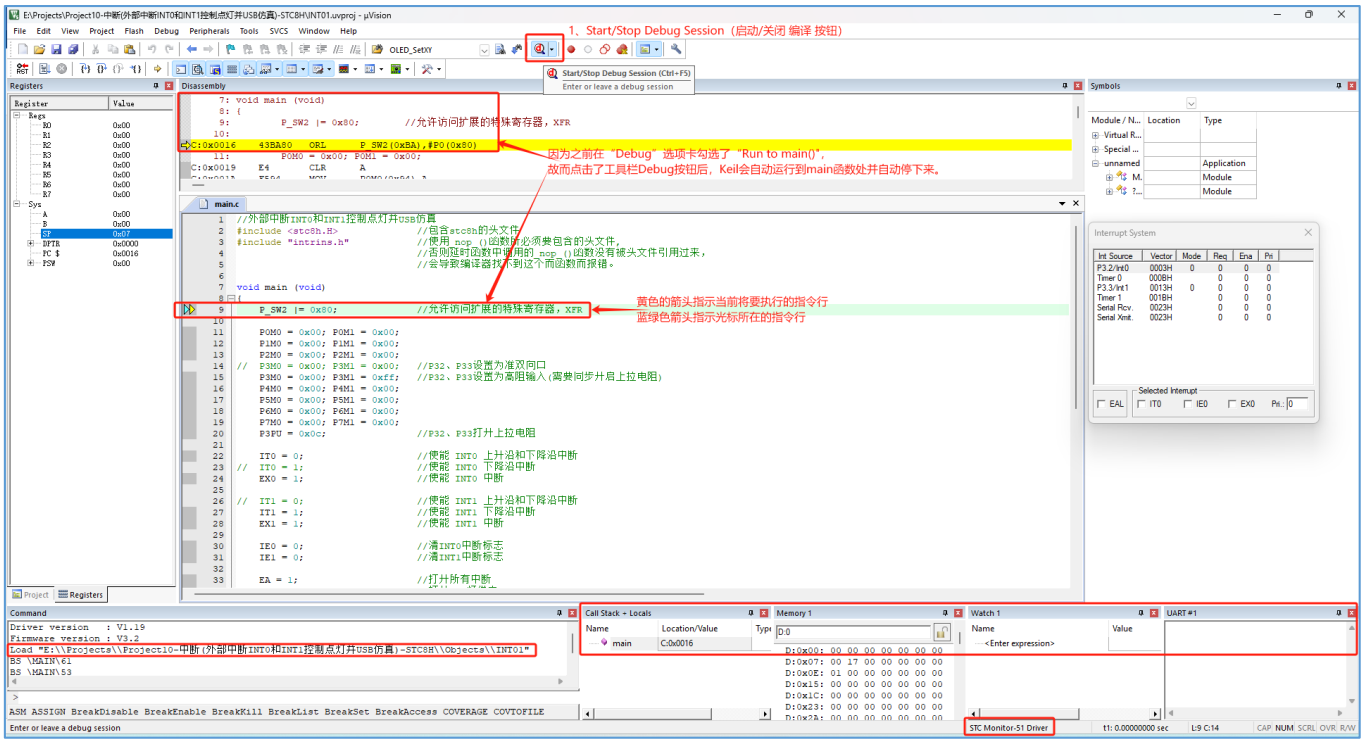
3.2.2.2　Utilities 选项卡--勾选“Use Debug Driver”

切换到“Utilities”选项卡，勾选“Use Debug Driver”，点击“OK”，完成设置。



3.2.3　进入 Debug 界面，观察实验现象

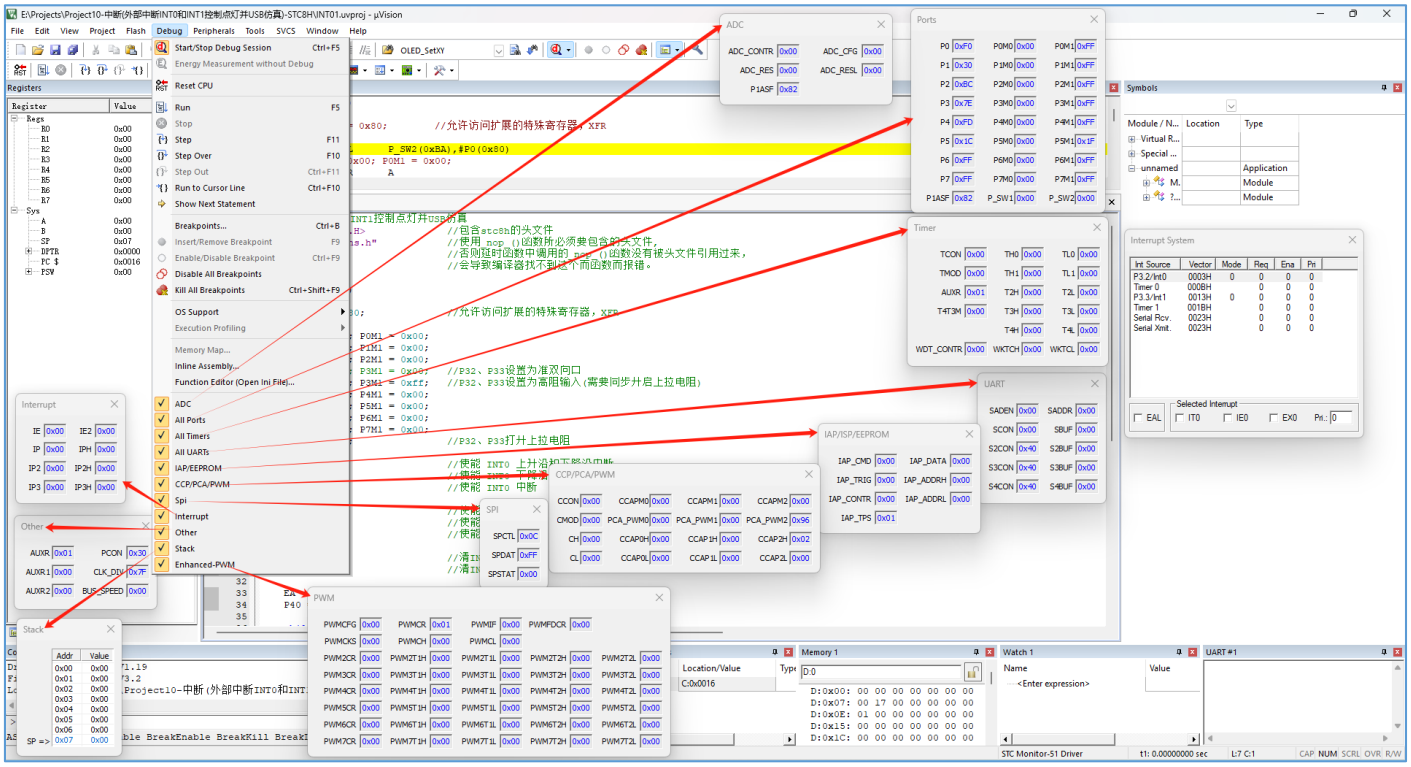
设置完毕，返回 Keil 界面，代码程序编译无误后，点击工具栏的“Debug”（启动/关闭编译调试操作界面）按钮后，Keil 软件进入仿真界面，如下图：



提示：退出 Debug 调试界面，同样点击工具栏的“Debug”按钮。

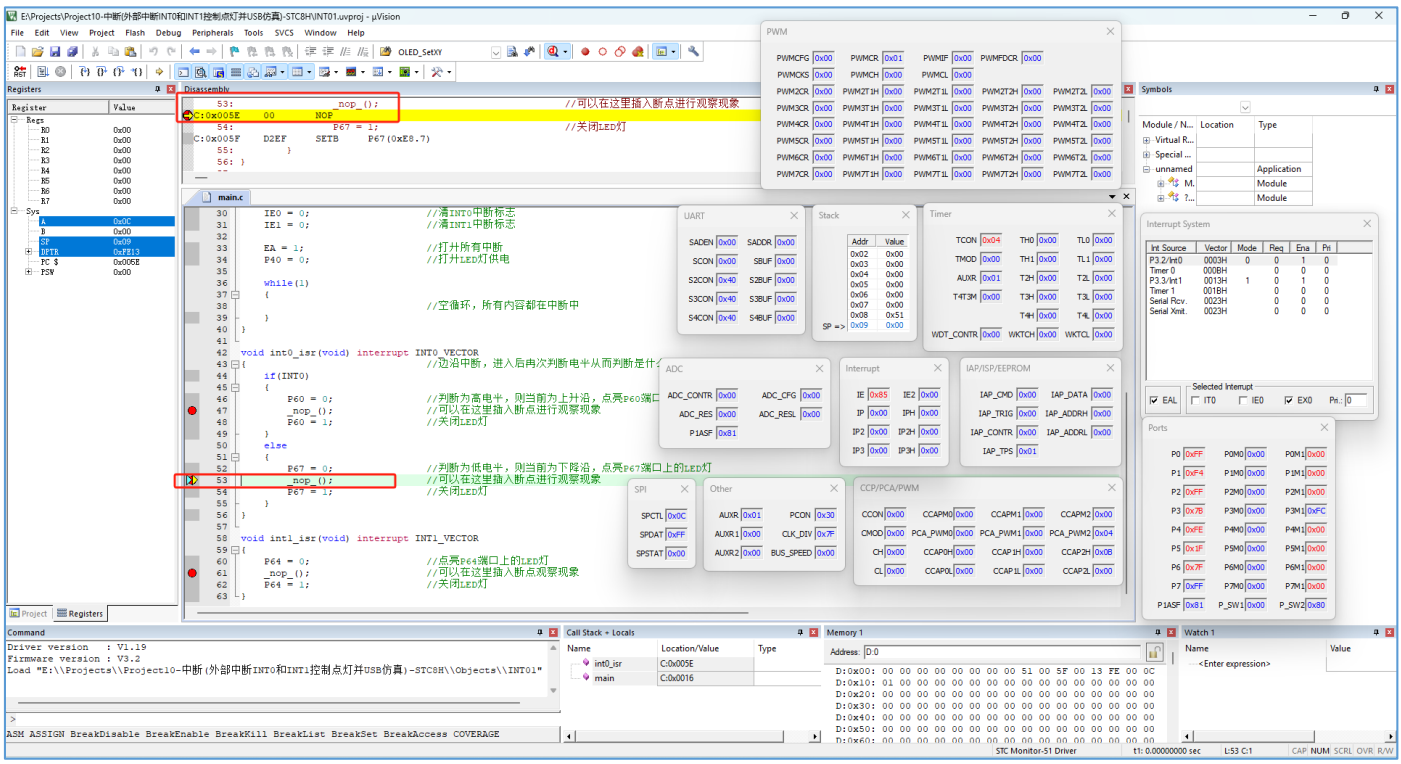
➲ 【步骤一】: 利用仿真，可以设置运行到断点处，捕捉当前程序运行位置，INT0/P3.2 是下降沿还是上升沿

➢ **为方便观察实验现象，需在中断服务程序中设置如下 3 个断点。**当然，你也可以打开全部外设窗口观察各外设数据在 Debug 过程中的变化：

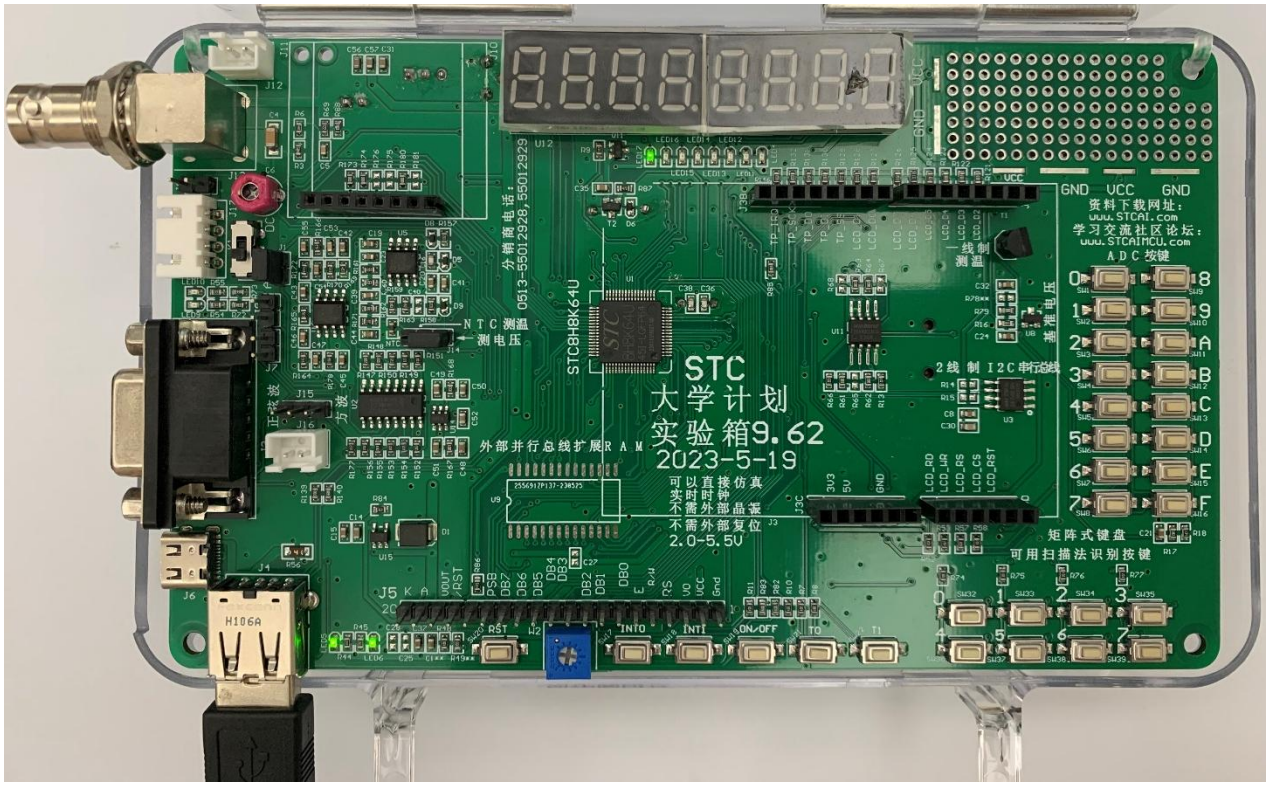


➢ 先点击 Run（运行）按钮，再按下 P3.2 按键，保持不松开（为了验证下一步的实验）：

✓ 程序会停在如下图所示断点处。同时，相较于上图，我们观察到：Ports / Timer / Interrupt / Interrupt System / Call Stack /Memory1 / Registers / Disassembly 这些窗口显示的数据都有可见的变化。

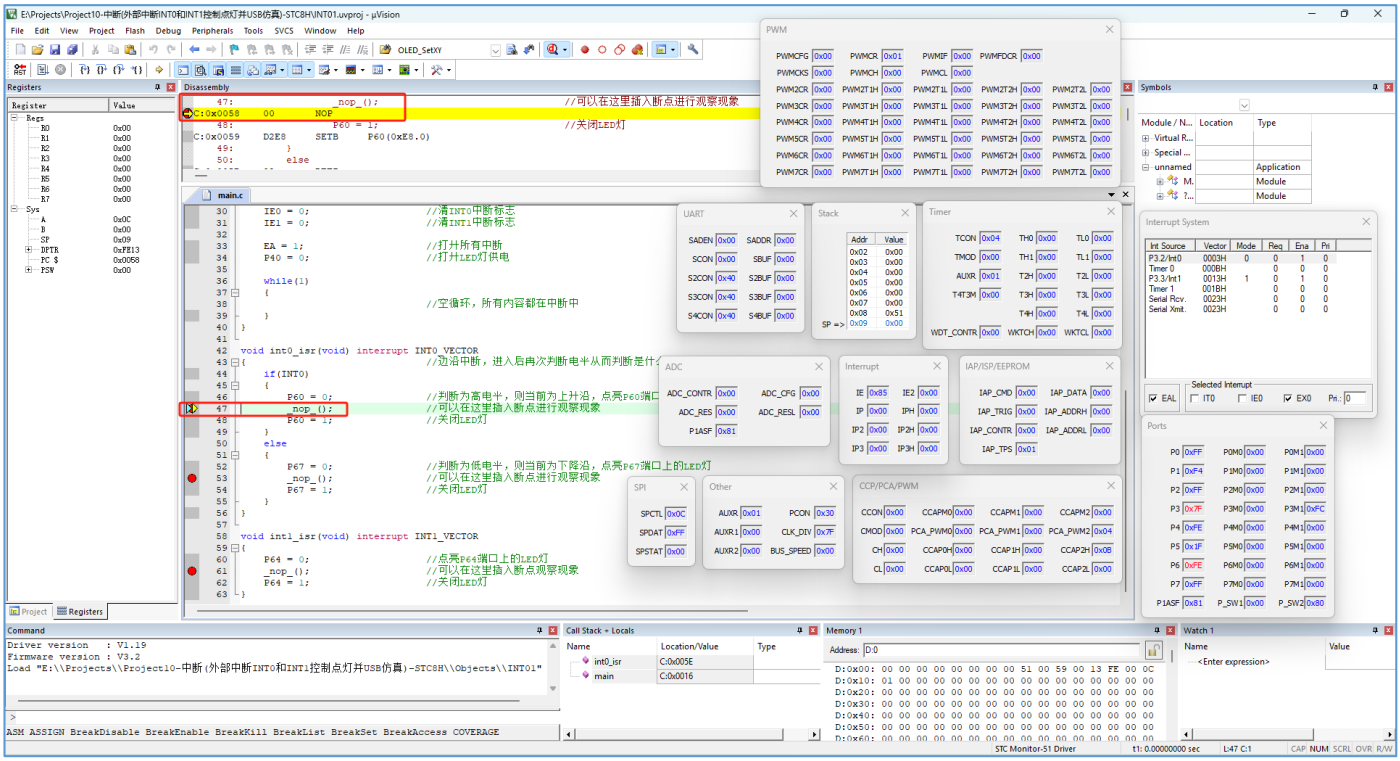


✓ 进 INT0 中断后，读到 INT0/P3.2 口是 低，下降沿，点亮 P6.7，STC8H8K64U 实验箱如下图：

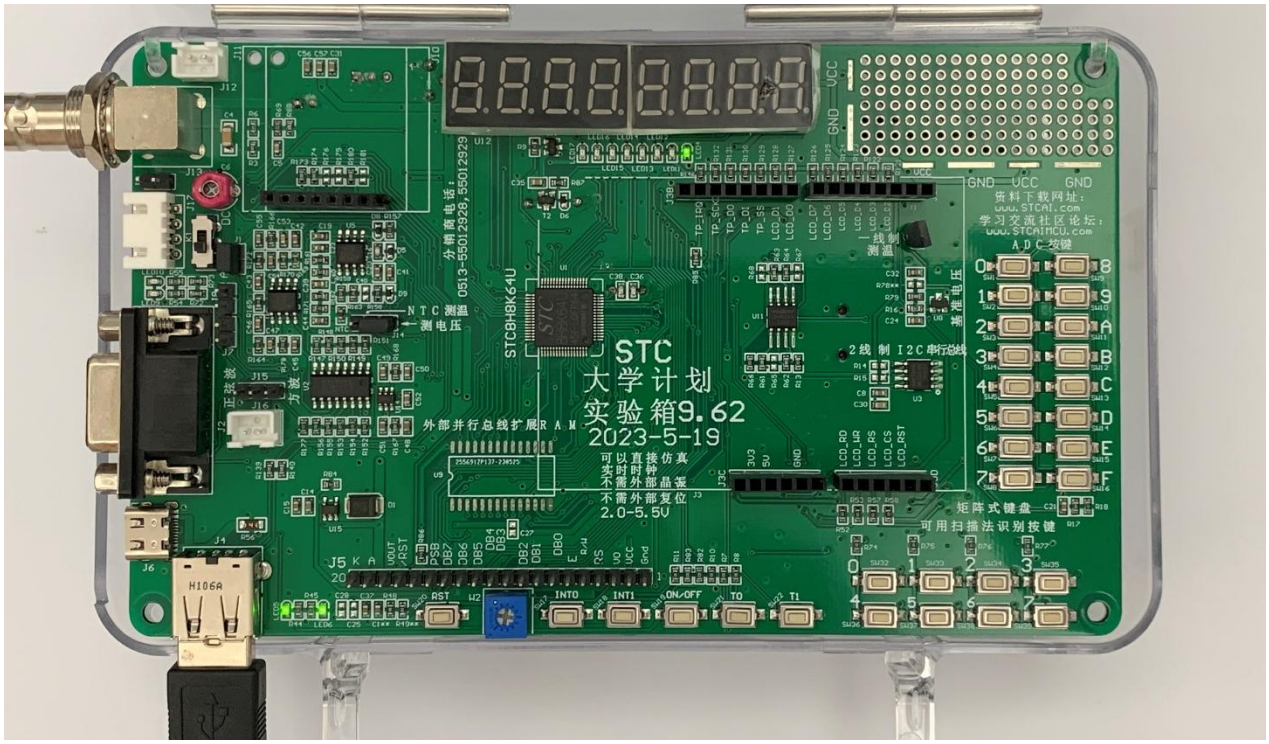


➢ 继续点击 Run（运行）按钮运行，退出中断服务程序。再松开 P3.2 按键：

✓ 程序会停在如下图所示断点处。同时，相较于上图，我们观察到：Ports /Memory1 / Registers / Disassembly 这些窗口显示的数据都有可见的变化。



✓ 进 INT0 中断后，读到 INT0/P3.2 口是 高，上升沿，点亮 P6.0，STC8H8K64U 实验箱如下图：

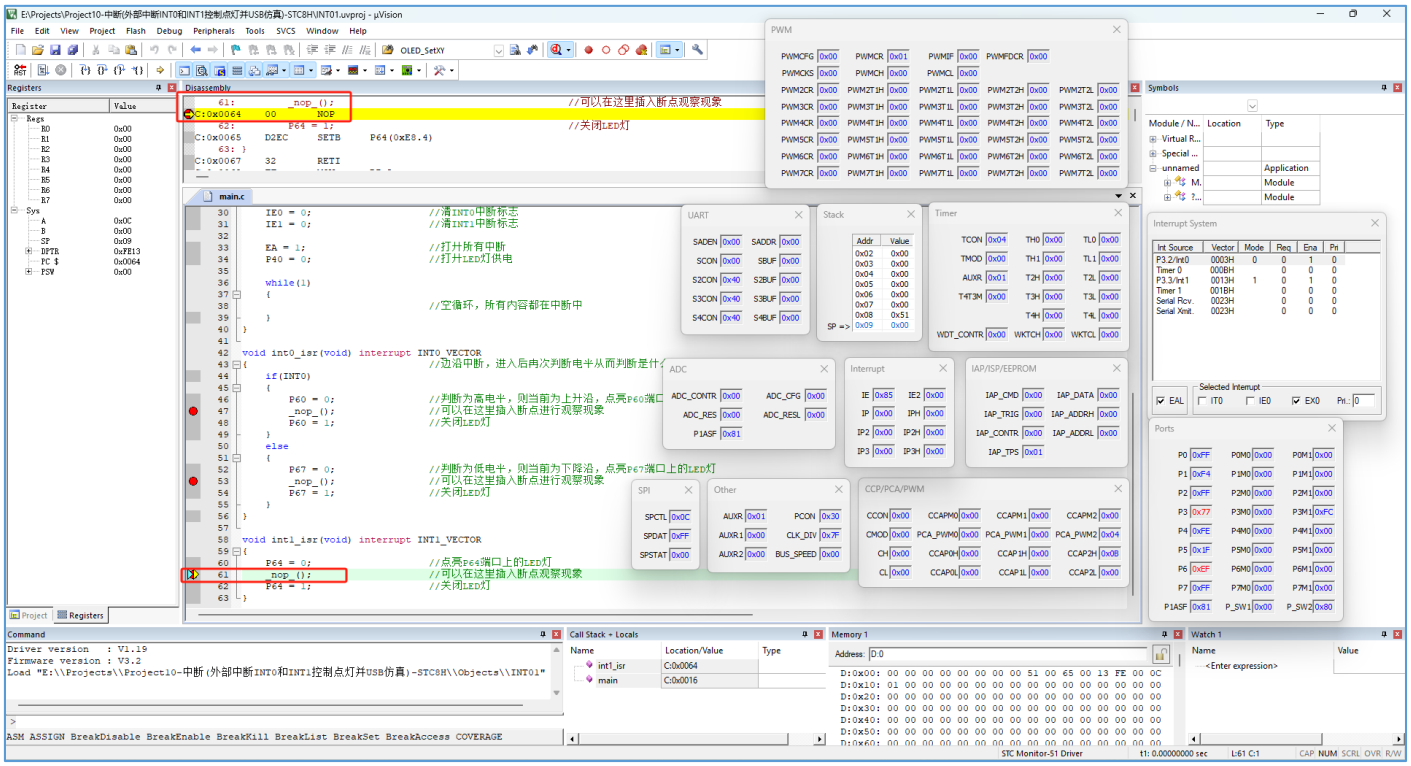


➢ 继续点击 Run（运行）按钮运行，退出中断服务程序。

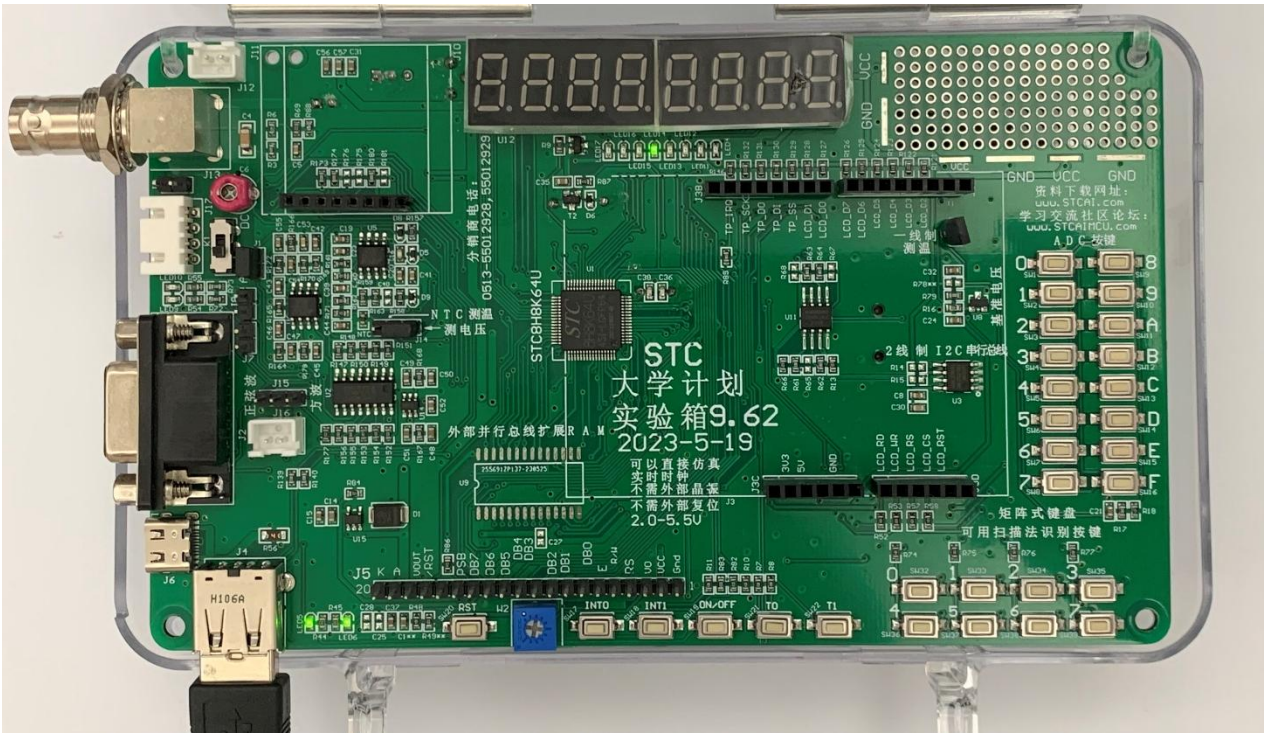
➲ 【步骤二】：利用仿真，可以设置运行到 INT1/P3.3 外部中断的断点处

➢ 如果程序没有在运行，需先点击 Run（运行）按钮，再按一下 P3.3 按键：

✓ 程序会停在如下图所示断点处。同时，相较于上图，我们观察到：Ports / Call Stack /Memory1 / Registers / Disassembly 这些窗口显示的数据都有可见的变化。



✓ 进 INT1 中断后，点亮 P6.4，STC8H8K64U 实验箱如下图：



继续点击 Run（运行）按钮运行，退出中断服务程序。

完成本实验。

更新记录

● 2026/06/30

1、 增加章节：使用【AiCube】工具新建一个 8 位 8051 项目，在 AiCube-ISP 中

● 2025/09/29

1、 修正个别笔误和图片

● 2025/09/12

1、 增加实验：2.18 串口实验 -- 串口通讯(串口通讯，串口绘图，ADC 检测按键，ADC 热敏电阻测温)
2、 增加实验：2.19 大型综合实战 -- 模拟 RTC 时钟+数码管+ADC 检测按键(查询)
3、 增加实验：2.20 ADC 实验 -- 数码管+ NTC 热敏电阻+ADC 测温(中断方式)
4、 增加实验：2.21 大型综合实战 -- 课程设计:硬件 SPI 读写串行 Flash
5、 增加实验：2.22 SPI 实验 -- 硬件 SPI 驱动 OLED 显示图片、图形、字符 R

● 2025/09/02

1、 STC8H-实验指导书 初版