19.11 大型综合实战--深大课程设计：硬件 UART1 发送按键码，接收端蜂鸣器根据收到的键码发声

19.11.1. 实验介绍

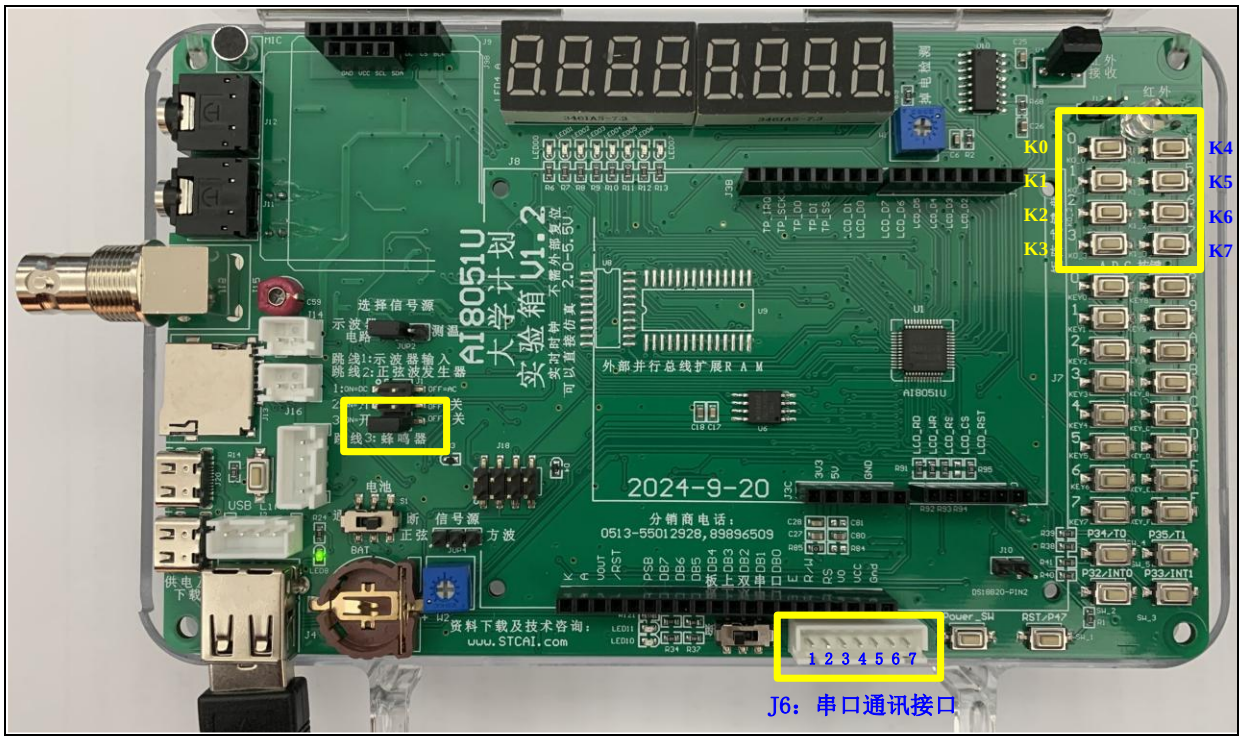
1、熟悉串口互联收发功能的使用
2、熟悉矩阵式按键扫描方法
3、I/O 控制无源蜂鸣器发声，Key0 ~ Key7 按键按下，会发出对应的 Do/Re/Mi/Fa/So/La/Si/Do 音符
4、熟悉如何管理多文件项目

a) 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。

b) 实验现象和操作说明：

Ai8051U 的实验箱正面图：



1、矩阵式键盘：在实验箱 A 按下按键 Key0 ~ Key7，实验箱 B 的蜂鸣器会发出对应的 Do/Re/Mi/Fa/So/La/Si/Do 音符
2、蜂鸣器跳线：需要将实验箱上“**跳线 3：蜂鸣器/原理图标号是 J1C**”的跳线连上，蜂鸣器才会发声
3、J6 串口通讯接口（见原理图 J6）：通过 3 根或 4 根杜邦线连接 2 个 Ai8051U 实验箱，实现串口通讯。
注意：本实验通过 UART1 串口通讯，

- 用 3 根杜邦线，使实验箱 A 的 J6.5（P3.7）、J6.6（P3.6）、J6.7（Gnd）连接实验箱 B 的 J6.6（P3.6）、J6.5（P3.7）、J6.7（Gnd），2 个实验箱都通过电脑供电；
- 用 4 根杜邦线，使实验箱 A 的 J6.4（SYS-Vcc）、J6.5（P3.7）、J6.6（P3.6）、J6.7（Gnd）连接实验箱 B 的 J6.4（SYS-Vcc）、J6.6（P3.6）、J6.5（P3.7）、J6.7（Gnd），这样连接电脑的实验箱就可以给另一个实验箱供电，不需要 2 个实验箱都通过电脑供电。

4、杜邦线：3 根/4 根

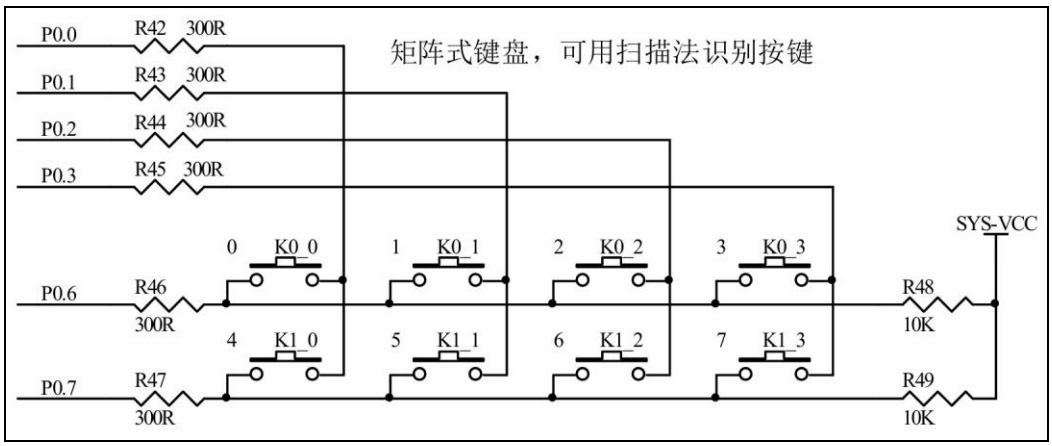
c) Keil 环境下多文件项目管理说明

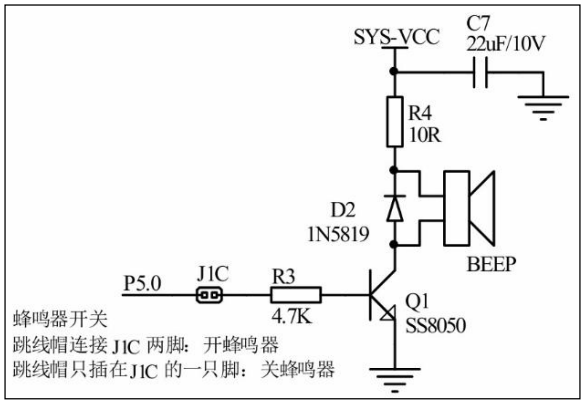
当项目的功能比较复杂时，就需要在 Keil 中建立多文件项目，以方便分工合作、代码复用、模块化管理、增强可读性和可维护性。

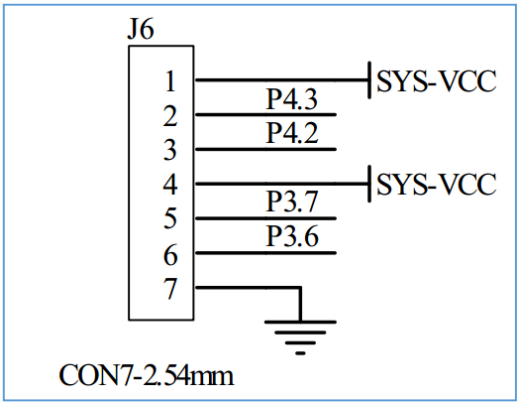
- 比较好的建议是将项目功能模块化，不同模块的实现代码放在不同的.c 文件中。
- 一般建议是一个模块对应一个.c 程序文件和一个.h 头文件
- 模块的初始化函数以及相关的数据处理函数都在.c 文件中实现
- 与模块相关的全局变量也必须在.c 文件中进行定义，一定不能在.h 文件中定义变量
- 如果有其他模块需要使用本模块定义的变量或函数，则这些函数和变量都需要在.h 文件中声明。
- 特别提醒：在.h 文件中声明外部变量必须使用 **extern** 关键字，否则就变成变量定义了，这样会出现变量重复定义的错误
- 为防止头文件被多次包含而产生错误或者警告，在头文件中使用类似如下的条件编译组合语句，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
```

19.11.2. 原理图







19.11.3. 实验程序代码

a) main.c -- 程序主函数

```
//main.c
//程序主函数
//短接跳线 3 蜂鸣器开关
//使用杜邦线连接两个实验箱 J6 接口的 P3.6, P3.7 与地线
//实验箱 A 的 P3.6 接实验箱 B 的 P3.7，实验箱 A 的 P3.7 接实验箱 B 的 P3.6，双方地线相连

#include "config.h"        //头文件中已包含 ai8051u.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();    //按键处理事件件
        }

        if (RI)                    //检测串口 1 接收完成标志
        {
            RI = 0;                //清除串口 1 接收完成标志
            BeepOn(SBUF);     //蜂鸣器根据收到的数据发声
        }
    }
}

void SYS_Init(void)      //系统初始化函数
```

```
{
    EAXFR = 1;          //使能访问扩展 XFR
    WTST = 0x00;        //设置最快速度访问程序代码
    CKCON = 0x00;       //设置最快速度访问内部 XDATA

    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 = 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

    S1_S1 = 0;          //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
    S1_S0 = 1;          //UART1 选择 P3.6 P3.7 作为串口收发通道

    Timer0_Init();      //定时器 0 初始化,用于产生 1ms 的时间基准
    Timer1_Init();      //定时器 1 初始化,用于驱动无源蜂鸣器,使蜂鸣器发声
    Uart1_Init();       //UART 接口初始化

    EA = 1;             //使能全局中断
}
```

b) 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 ai8051u.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 |= 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 = 0xA9;           //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 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
```

```
}
```

c) 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 ai8051u.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

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)
    }
    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
            }
        }
    }
}

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
        Uart1_Send(bKeyCode);           //串口发送键码
    }
}
```

d) beep.c -- 蜂鸣器发声控制程序

```
//beep.c
//蜂鸣器发声控制程序
//实验箱上的蜂鸣器为无源蜂鸣器
//通过蜂鸣器的控制口 P5.0 上输入不同频率的方波
//即可控制蜂鸣器发出不同频率的声音
//从而达到播放不同音调的效果

#include "config.h"             //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

//根据 Do/Re/Mi/Fa/So/La/Si/Do 这 8 个音符的频率定义定时器 1 的定时重载值
//由于定时器是翻转模式,所以需要将音符的频率值*2
#define   BEEP_DO     (WORD)(65536 - (float)FOSC / 12 / (261.6 * 2))
#define   BEEP_RE     (WORD)(65536 - (float)FOSC / 12 / (293.6 * 2))
#define   BEEP_MI     (WORD)(65536 - (float)FOSC / 12 / (329.6 * 2))
#define   BEEP_FA     (WORD)(65536 - (float)FOSC / 12 / (349.2 * 2))
#define   BEEP_SO     (WORD)(65536 - (float)FOSC / 12 / (392.0 * 2))
#define   BEEP_LA     (WORD)(65536 - (float)FOSC / 12 / (440.0 * 2))
#define   BEEP_SI     (WORD)(65536 - (float)FOSC / 12 / (493.8 * 2))
#define   BEEP_DOH    (WORD)(65536 - (float)FOSC / 12 / (523.2 * 2))

bit   fBeepEn;                  //蜂鸣器发声使能控制

void BeepOn(BYTE bBeep)         //蜂鸣器发声控制程序
```

```
{
    switch (bBeep)                  //根据函数的入口参数,控制蜂鸣器发不同的声音
    {
        case 0:                     //入口参数为 0 时,蜂鸣器发 Do 的声音
            TL1 = BEEP_DO;          //将 Do 的定时器重装值低 8 位加载到定时器 1 低位寄存器
            TH1 = BEEP_DO >> 8;     //将 Do 的定时器重装值高 8 位加载到定时器 1 高位寄存器
            break;
        case 1:                     //入口参数为 1 时,蜂鸣器发 Re 的声音
            TL1 = BEEP_RE;          //将 Re 的定时器重装值低 8 位加载到定时器 1 低位寄存器
            TH1 = BEEP_RE >> 8;     //将 Re 的定时器重装值高 8 位加载到定时器 1 高位寄存器
            break;
        case 2:                     //入口参数为 2 时,蜂鸣器发 Mi 的声音
            TL1 = BEEP_MI;          //将 Mi 的定时器重装值低 8 位加载到定时器 1 低位寄存器
            TH1 = BEEP_MI >> 8;     //将 Mi 的定时器重装值高 8 位加载到定时器 1 高位寄存器
            break;
        case 3:                     //入口参数为 3 时,蜂鸣器发 Fa 的声音
            TL1 = BEEP_FA;          //将 Fa 的定时器重装值低 8 位加载到定时器 1 低位寄存器
            TH1 = BEEP_FA >> 8;     //将 Fa 的定时器重装值高 8 位加载到定时器 1 高位寄存器
            break;
        case 4:                     //入口参数为 4 时,蜂鸣器发 So 的声音
            TL1 = BEEP_SO;          //将 So 的定时器重装值低 8 位加载到定时器 1 低位寄存器
            TH1 = BEEP_SO >> 8;     //将 So 的定时器重装值高 8 位加载到定时器 1 高位寄存器
            break;
        case 5:                     //入口参数为 5 时,蜂鸣器发 La 的声音
            TL1 = BEEP_LA;          //将 La 的定时器重装值低 8 位加载到定时器 1 低位寄存器
            TH1 = BEEP_LA >> 8;     //将 La 的定时器重装值高 8 位加载到定时器 1 高位寄存器
            break;
        case 6:                     //入口参数为 6 时,蜂鸣器发 Si 的声音
            TL1 = BEEP_SI;          //将 Si 的定时器重装值低 8 位加载到定时器 1 低位寄存器
            TH1 = BEEP_SI >> 8;     //将 Si 的定时器重装值高 8 位加载到定时器 1 高位寄存器
            break;
        case 7:                     //入口参数为 70 时,蜂鸣器发 DoH 的声音
            TL1 = BEEP_DOH;         //将 DoH 的定时器重装值低 8 位加载到定时器 1 低位寄存器
            TH1 = BEEP_DOH >> 8;    //将 DoH 的定时器重装值高 8 位加载到定时器 1 高位寄存器
            break;
        default:                    //入口参数为其他值时
            BeepOff();              //关闭蜂鸣器
            return;                 //直接返回
    }
    fBeepEn = 1;                    //是蜂鸣器发声
}

void BeepOff()                      //关闭蜂鸣器发声
{
    fBeepEn = 0;                    //关闭蜂鸣器发声控制位
}
```

e) 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
//定时器 1:用于翻转蜂鸣器的控制口 P5.0

#include "config.h"                 //头文件中已包含 ai8051u.h, ai_usb.h 以及其他头文件

bit    f1ms;                        //1ms 标志位

void Timer0_Init(void)              //1 毫秒@40.000MHz
{
    AUXR |= 0x80;                   //定时器 0 时钟 1T 模式
    TMOD &= 0xF0;                   //设置定时器 0 为模式 0
    TL0 = 0xC0;                     //设置定时器 0 初始值
    TH0 = 0x63;                     //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 1ms 标志位
}

void Timer1_Init(void)              //@40.000MHz
{
    AUXR &= ~0x40;                  //定时器 1 时钟 12T 模式
    TMOD &= 0x0F;                   //设置定时器 1 为模式 0
    TL1 = 0x00;                     //设置定时器 1 初始值
    TH1 = 0x00;                     //设置定时器 1 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

    fBeepEn = 0;                    //初始化蜂鸣器发声使能位
}

void Timer0_Isr(void) interrupt 1   //定时器 0 中断服务程序
{
    f1ms = 1;                       //设置 1ms 标志位
}

void Timer1_Isr(void) interrupt 3   //定时器 1 中断服务程序
{
```

```
    if (fBeepEn)                //判断是否使能蜂鸣器发声
        P50 = !P50;             //如果需要则翻转蜂鸣器控制端口 P5.0
    else
        P50 = 0;                //如果不需要,则关闭蜂鸣器控制端口 P5.0
}
```

f) 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            40000000UL
#define FOSC            40000000UL
#define SYSCLK          FOSC
#define MAIN_Fosc       FOSC

#include <ai8051u.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 "key.h"
#include "uart.h"
#include "beep.h"
#include "timer.h"

void SYS_Init(void);            //函数声明

#endif
```

g) 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
```

h) 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
```

i) beep.h -- 蜂鸣器发声控制程序的头文件

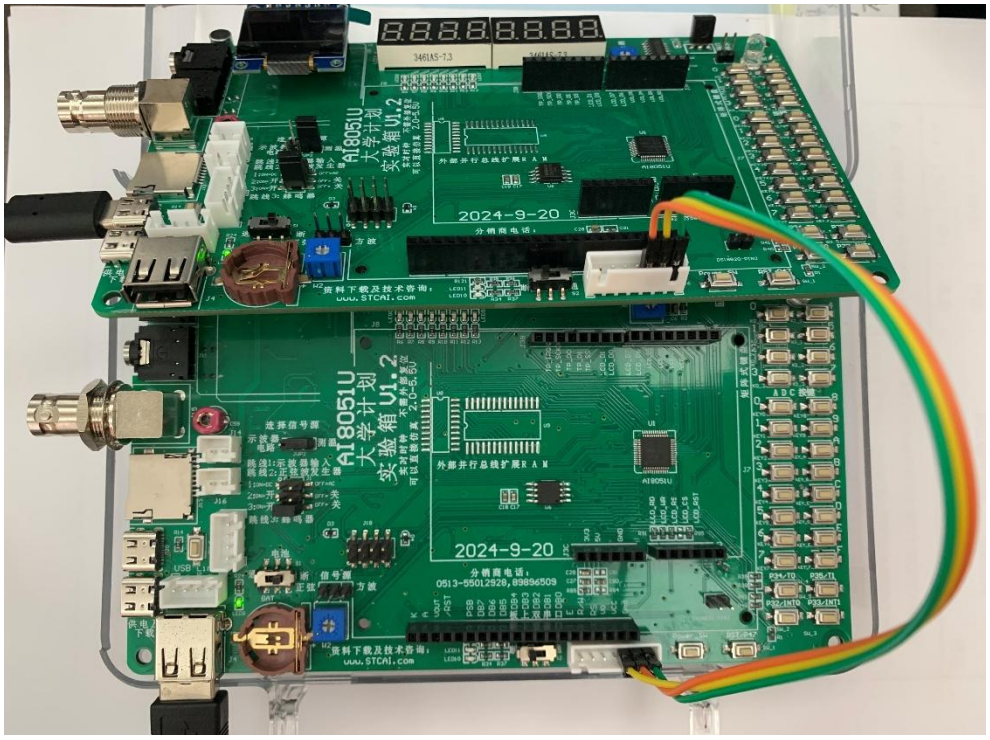
```
#ifndef __BEEP_H__
#define __BEEP_H__

void BeepOn(BYTE bBeep);    //函数声明
void BeepOff();             //函数声明

extern bit fBeepEn;         //外部变量声明

#endif
```

j) timer.h -- 项目配置文件的头文件

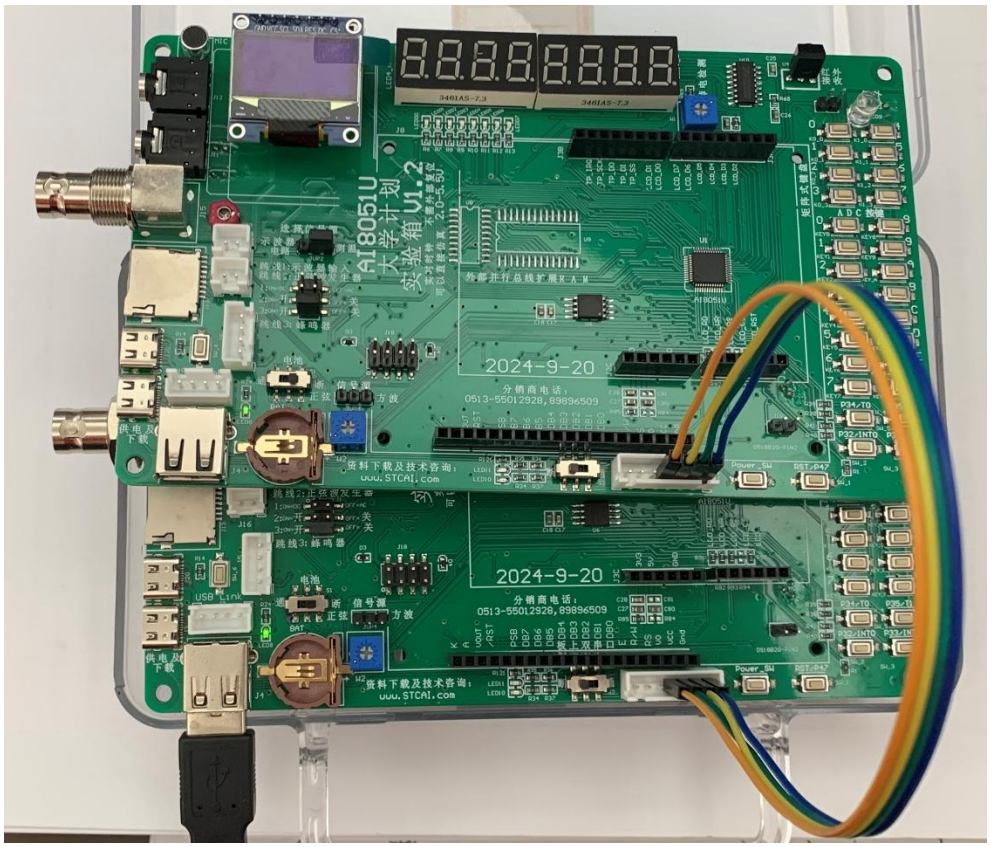
```
#ifndef __TIMER_H__         //防止头文件被重复包含
#define __TIMER_H__

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

#endif
```

19.11.4. 程序下载

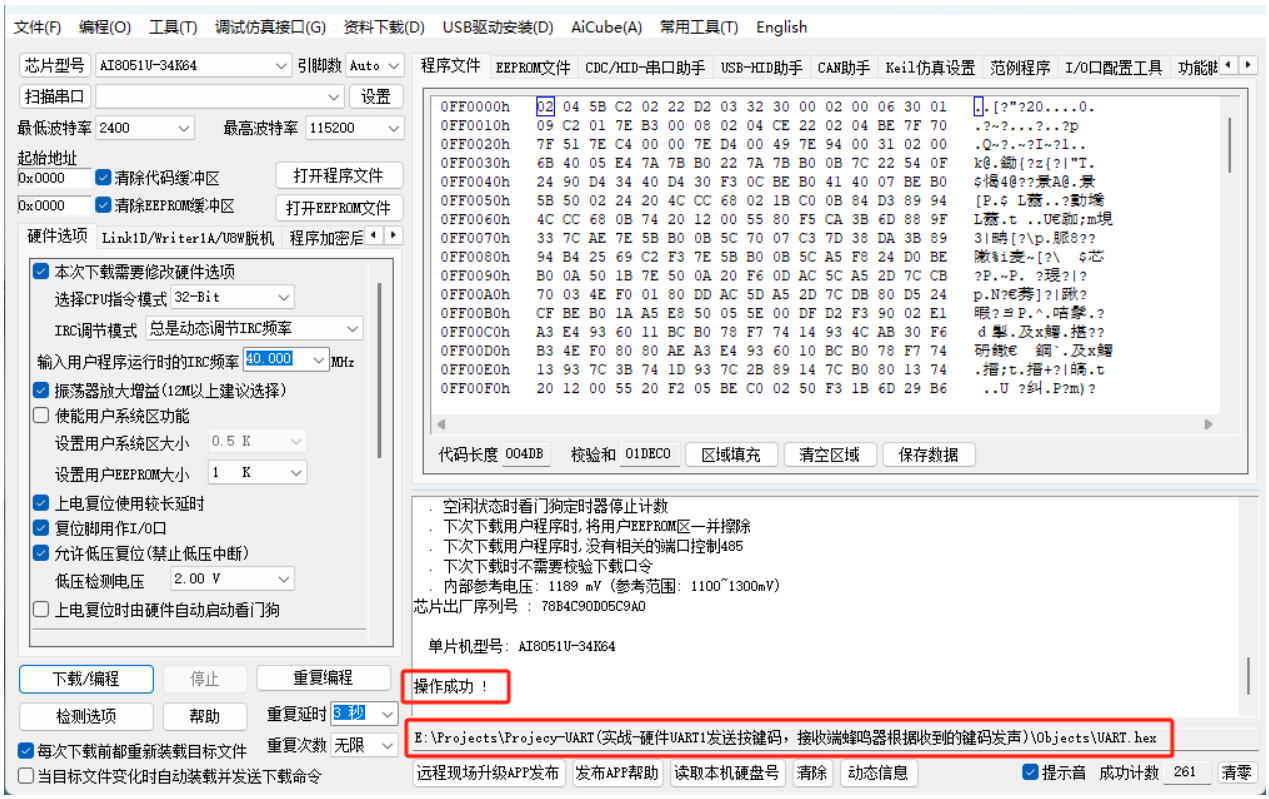
录入代码，保存，编译。连接好 Ai8051U 实验箱，下面两种连接方式皆可实现实验要求：
1、2 个实验箱分别和电脑相连，3 根杜邦线互连：



2、实验箱 A 和电脑相连，4 根杜邦线连接实验箱 B（实验箱 A 给实验箱 B 供电）：



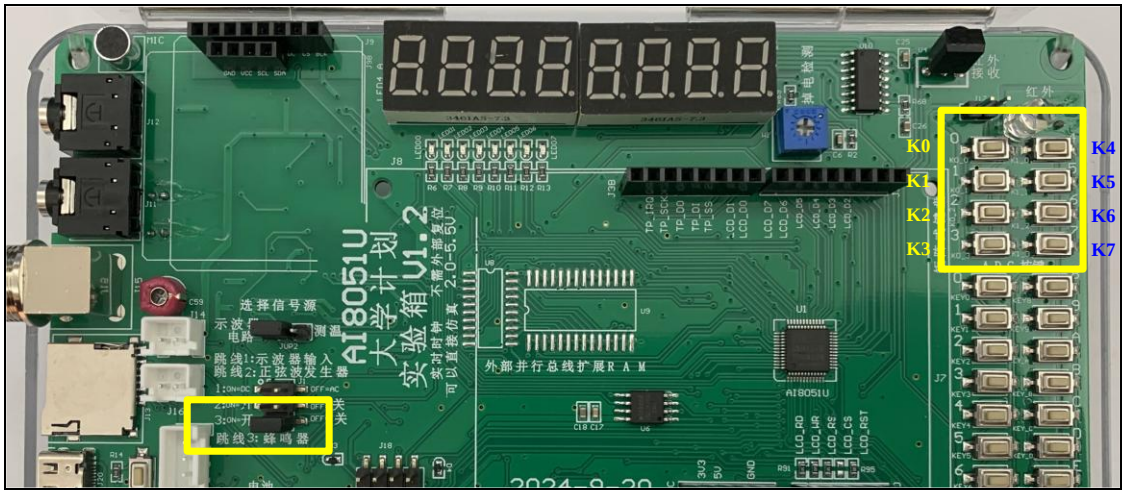
运行 AIapp-ISP 软件系统，按步骤打开并下载程序“uart.hex”，如下图：



19.11.5. 验证实验效果

将 2 个 Ai8051U 实验箱上“跳线 3：蜂鸣器”的跳线连上。

找到 K0_0 ~K0_3、K1_0 ~ K1_3 这 8 个按键（如下图）：



- 在实验箱 A 上分别按下 K0_0 ~K0_3、K1_0 ~ K1_3 这 8 个按键：实验箱 B 的蜂鸣器会发出对应的“Do、Re、Mi、Fa、So、La、Si、Do”的音符声
- 在实验箱 B 上分别按下 K0_0 ~K0_3、K1_0 ~ K1_3 这 8 个按键：实验箱 A 的蜂鸣器会发出对应的“Do、Re、Mi、Fa、So、La、Si、Do”的音符声

完成实验验证。