北京工業大學

BEIJING UNIVERSITY OF TECHNOLOGY

嵌入式系统设计技术

实验项目设计报告

学　　期：　2023-2024 学年 第二学期

专　　业：　计算机科学与技术（实验班）

学号姓名：　21071003　高立扬

完成日期：　2024.04.06

教师评语					
得分		教师签字		评阅日期	

一、 测距仪

1. 功能设计

本项目设计并实现利用超声波测距技术的测距仪，其具有以下几项功能：

1) 距离测量，其有效测量范围约为 2cm-300cm，根据手册说明，其理论范围为 2cm-800cm；
2) 距离过近预警。当测量到物体到传感器距离为 5-10cm 时，LED 亮起，呈黄色；为 5cm 以内时，LED 变为红色，同时蜂鸣器持续鸣叫；
3) 滤波功能，其滤波方式为 EWMA，对传感器位姿突然转变或物体阻挡带来的测距影响具有一定的鲁棒性，且计算量非常小。

2. 硬件电路设计

本项目以 ESP32-C3 开发板为核心控制板。通过开发板 J3 接口与 RGB-LED 连接，ESP32-C3 的 SS 引脚控制 RGB-LED 的红灯、MISO 引脚控制 RGB-LED 的绿灯、SCK 引脚控制 RGB-LED 的蓝灯；通过开发板 J7 接口与超声波传感器模块连接,ESP32-C3 的 IO2 引脚控制超声波传感器的 Trig 信号,ESP32-C3 的 IO3 引脚读取超声波传感器的 Echo 信号，以 cm 为单位的测距数值传输至上位机的串口监视器中显示；通过开发板 J6 接口与无源蜂鸣器连接，ESP32-C3 的 IO0 引脚控制蜂鸣器的 Sig 信号。

该系统的硬件电路示意图如图 1 所示。

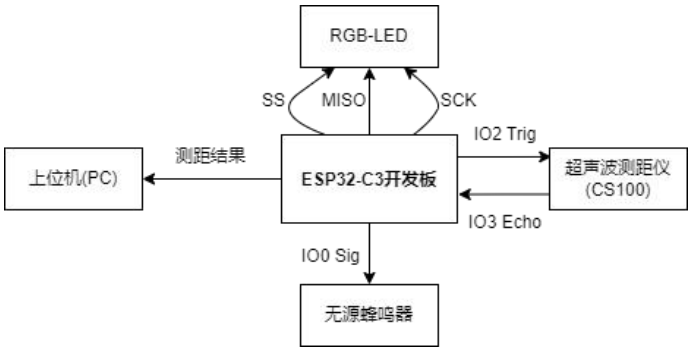

图 1 硬件电路示意图

3. 程序代码设计

本项目中使用到的一些全局变量和宏定义如下

```
const int TrigPin = 2;  // IO 接口定义
const int EchoPin = 3;
const int SigPin = 0;
double distance;  //定义全局变量，保存距离
double previous = -1; // 滤波过程中用到的变量
#define RLED 7 // IO 接口宏定义
#define GLED 5
#define BLED 4
#define ALPHA 0.1
```

setup 函数中，串口波特率初始化为 115200 字节每秒，随后是 pinMode 的初始化以及 LED 灯为“灭”的初始化，在此不进行代码展示

loop 函数是本程序主函数，首先向 CS100 的 Trig 管脚发送 10 微秒的高电平，使得 TP、TN 探头循环发出 8 个 40K HZ 的超声波脉冲，随后 RP、RN 检测并接收回波信号，从 Echo 管脚输出。软件负责检测 Echo

管脚的高电平持续时间（单位为微秒）数据类型为 float。，通过换算后得到单位为 cm 的距离，存放到数据类型为 double 的相关变量后，进行 EWMA 滤波，输出到串口上。LED 会根据接收到的距离，进行灯色调整。无源蜂鸣器会在距离达到 5cm 之内时，持续鸣响。测距时间间隔为 10 毫秒。

```
void loop() {
   // trigger 触发
  digitalWrite(TrigPin, LOW);
  delayMicroseconds(10);
  digitalWrite(TrigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(TrigPin, LOW);

  // echo 接收并计算 cm
  distance = pulseIn(EchoPin, HIGH) * 340.0 / (2.0 * 10000.0);
  // 滤波
  if(!previous) previous = distance;
  else          previous = previous * (1 - ALPHA) + ALPHA * distance;
  // 串口输出
  Serial.println(previous);

  // 指示灯判断，安全距离为 10cm 及以上，警告为 5-10cm，0-5 为危险
  if(distance > 10.0){  // 绿灯
    digitalWrite(RLED, LOW);
    digitalWrite(GLED, HIGH);
    digitalWrite(BLED, LOW);
  }
  else if(distance > 5.0 && distance <= 10.0){  // 黄灯
    digitalWrite(RLED, HIGH);
    digitalWrite(GLED, HIGH);
    digitalWrite(BLED, LOW);
  }
  else{  // 红灯 & 蜂鸣器
    digitalWrite(RLED, HIGH);
    digitalWrite(GLED, LOW);
    digitalWrite(BLED, LOW);

    digitalWrite(SigPin, HIGH);
    delay(1);
  }
  digitalWrite(SigPin, LOW);
  // 测距间隔
  delay(10);
}
```

4. 测试

待测物体正面垂直于超声波传感器 5cm、20cm 和 40cm 时，系统检测值相对稳定后，取得结果分别为 5.05cm、20.09cm 和 39.72cm，误差值较小。待测物体与超声波传感器形成 20°左右夹角时，在 5cm、20cm 处检测结果为 4.95cm、19.87cm，误差值较小。当距离 40cm 时，检测结果为 37.37cm，已产生一定误差。测试结果分别如表 1 和表 2 所示。

表 1 待测物体正面垂直于传感器时测试结果

测试条件	5cm	20cm	40cm
测试结果	5.05cm	20.09cm	39.72cm

表 2 待测物体与传感器形成 20°夹角时测试结果

测试条件	5cm	20cm	40cm
测试结果	4.95cm	19.87cm	37.37cm

二、 电子密码箱

1. 功能设计

本项目设计并实现基于 TTP229 芯片进行密码输入的电子密码箱，其具有以下几项功能：

1) 密码箱四种状态：上锁状态、关门状态、开锁状态、锁定状态，分别对应用户为密码箱设置密码、关闭密码箱、输入密码并解锁密码箱、输入错误次数过多联系管理员开启密码箱；
2) 密码设置/输入：用户可以通过 16 键触摸传感器模块进行密码设置/输入，输错重置可以按 K2 按钮，确认输入可以按 K1 按钮；
3) 交互功能：OLED 屏幕对密码箱的状态进行显示，并且可以对用户的密码输入进行实时展示，便于用户输入密码。RGB-LED 也会根据状态进行颜色变换，辅助提示当前密码箱的状态；
4) 管理员后台解锁：当用户连续输入密码错误达到五次，密码箱会强制锁定，此时需要管理员在串口监视器输入密码进行强制解锁。

2. 硬件电路设计

本项目以 ESP32-C3 开发板为核心控制板。通过开发板 J3 接口与 RGB-LED 连接，ESP32-C3 的 SS 引脚控制 RGB-LED 的红灯、MISO 引脚控制 RGB-LED 的绿灯、SCK 引脚控制 RGB-LED 的蓝灯；通过开发板 J6 接口与 KEY Card 数字量输入 J3 接口连接，K1 和 K2 引脚分别接收两个按钮的状态并输出到开发板；通过开发板 J4 接口借助 I2C 扩展板与 16 键触摸传感器模块和 128x64OLED 显示模块连接，软件内利用 I2C 总线函数库实现相关控制。

该系统的硬件电路示意图如图 2 所示。

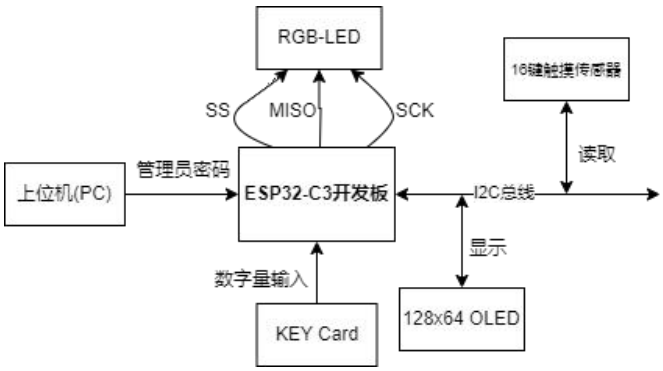

图 2 硬件电路示意图

3. 程序代码设计

本项目中使用 Adafruit 公司的函数库 Adafruit SSD1306 (2.5.9)和 Adafruit GFX(1.11.9)实现 OLED 显示屏相关功能。使用 Wire 库进行 I2C 通信。

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Wire.h>
```

本项目宏定义和全局变量声明如下

```
#define RLED 7 // 接口宏定义
#define GLED 5
#define BLED 4
#define K2 1
#define K1 0
```

```
#define TTP229_ADDR 0x57  // TTP229 地址
#define OLED_RESET    -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C  // OLED 地址
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
// OLED 实现
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
/*上锁状态 0，关门 1，开锁 2，超过次数 3*/
int state;
// 密码错误次数累计
int failNum=0;
// 显示在屏幕上的输入
char code[15] = {'_', '_', '_', '_', '_', '_', '_', '_', '_', '_', '_', '_',
'_', '_', '_'};
// 输入压栈指针
int codeFull=0;
// 显示在屏幕上要设置的密码输入
char pwd[15] = {'_', '_', '_', '_', '_', '_', '_', '_', '_', '_', '_', '_',
'_', '_', '_'};
// 密码指针
int pwdFull=0;
// 按键数据接收变量
unsigned int c = 0;
// 记录当前按钮
int button = -1;
// 按钮数据转换为 char
char cButton;
```

setup 函数中，参照了 Adafruit 官方的示例进行初始化，以及其他初始化，如下

```
void setup() {
  Wire.begin();
  Serial.begin(115200);
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
  display.display();
  delay(2000);
  // Clear the buffer
  display.clearDisplay();
  // LEDs
  pinMode(RLED, OUTPUT);
  pinMode(GLED, OUTPUT);
  pinMode(BLED, OUTPUT);
  state = 0;
```

```
  digitalWrite(RLED, HIGH); // 初始状态下是上锁状态，所以是黄灯
  digitalWrite(GLED, HIGH);
  digitalWrite(BLED, LOW);
  // button
  pinMode(K1, INPUT); // K1 for accept
  pinMode(K2, INPUT); // K2 for reset
}
```

rstCode 函数和 rstPwd 函数负责重置用户输入，使得屏幕上已输入的密码重新变为待输入符“_”

```
void rstCode(){
  for(int i = 0; i < 15; i++)  code[i] = '_';
}
void rstPwd(){
  for(int i = 0; i < 15; i++)  pwd[i] = '_';
}
```

YFlash 函数和 RFlash 函数负责控制 LED 进行黄色或红色的灯光闪烁，下面展示其一

```
void YFlash(){
  digitalWrite(RLED, LOW);
  digitalWrite(GLED, LOW);
  digitalWrite(BLED, LOW);
  delay(50);
  digitalWrite(RLED, HIGH);
  digitalWrite(GLED, HIGH);
  digitalWrite(BLED, LOW);
  delay(50);
}
```

state0print 函数、state1print 函数、state3print 函数和 gly 函数负责在密码箱不同状态下，在 OLED 屏输出相关内容，下面展示其一，具体实现见 4.

```
void state0print(){
  display.clearDisplay();
  display.setTextSize(1);           // Normal 1:1 pixel scale
  display.setCursor(5,5);           // Start at top-left corner
  display.setTextColor(1, 0); // 黑色背景白字
  display.println("State:SETTING");
  display.setTextSize(2);
  display.setCursor(5,20);
  for(int i = 0; i < 15; i++){
    display.print(pwd[i]); // 在屏幕上显示十六进制数字
    Serial.print(pwd[i]);
  }
  Serial.println();
  display.display(); // 更新屏幕显示
}
```

setPwd 函数和 enterCode 函数对应设置密码和输入密码，原理相同，下面展示其一，具体实现见 4.

```
void setPwd(){
```

```
    if(button != -1)
      if(pwdFull != 15)
        pwd[pwdFull++] = cButton;    // 字符压栈
      else
        for(int i = 0; i < 3; i++) YFlash();   // 输入超出了界限，黄灯闪烁
      button = -1;
    state0print(); // 状态 1 的 OLED 显示
  }
```

loop 函数为核心。首先进行键盘输入字节请求，每次请求两个字节，将读取的字节通过移位相加存入变量 c，得到一个二进制串，仅其中一位为 1，其余位都为 0，为 1 的那一位就是按下的那一个按钮，因此取 log 运算后就可以获取实际输入按键，进行 char 型转换后存入临时变量，读取完毕。有了读取按键信息，就可以根据密码箱的状态进行对应的操作。

在编写代码时，发现判断密码箱状态进行对应操作的语句，如果封装为一个函数，会导致程序不按照预期执行，但放在 loop 函数内就会正常运转，所以放弃了封装（此部分代码见// core 注释之后）

```
  void loop() {
    // 每次请求两个字节（16output 模式）
    Wire.requestFrom(TTP229_ADDR, 2);
    while(Wire.available()){
      c = 0;
      // read 每次读取一个字节
      for(int i = 0; i < 2; i++)
        if(i) c += Wire.read();
        else  c += Wire.read() << 8;
      if(c)   button = 15 - log2(c);
      // 转换
      if(button >= 0 && button <= 9)  cButton = button + 48;
      else if(button >= 10 && button <= 15) cButton = button + 55;

      // display
      if(state == 0)       setPwd();
      else if(state == 1)  enterCode();
      else if(state == 2)  state2print();
      else if(state == 3)  gly();

      // core
      if(state == 0){  // 设置密码
        // pwd rst
        if(digitalRead(K2) == LOW){
          pwdFull = 0;
          rstPwd();
          state0print();
        }
        // pwd accept
        if(digitalRead(K1) == LOW && pwdFull >= 4){  // 密码位数大于等于 4，接受
```

```
      state = 1;                                    // 状态转换为关闭
      digitalWrite(RLED, HIGH);                     // 变灯
      digitalWrite(GLED, LOW);
      digitalWrite(BLED, LOW);
      continue;
    }
    else if(digitalRead(K1) == LOW && pwdFull < 4) YFlash(); // 错误输入
  }
  else if(state == 1){  // 关闭状态，输入密码
    if(failNum == 5){  // 输入错误次数过多
      state = 3;
      digitalWrite(RLED, HIGH);
      digitalWrite(GLED, HIGH);
      digitalWrite(BLED, HIGH);
      continue;
    }
    // code rst   以下原理同上
    if(digitalRead(K2) == LOW){
      codeFull = 0;
      rstCode();
      state1print();
    }
    // code accept
    if(digitalRead(K1) == LOW && !strcmp(pwd, code)){
      state = 2;
      digitalWrite(RLED, LOW);
      digitalWrite(GLED, HIGH);
      digitalWrite(BLED, LOW);
      continue;
    }
    else if(digitalRead(K1) == LOW && strcmp(pwd, code)){
      RFlash();
      failNum++;  // 错误次数积累
    }
  }
  else if(state == 2){   // 开锁状态，用户仅需按按钮进行关闭
    if(digitalRead(K2) == LOW){ // 按 K2 关闭，进行变量重置
      state = 0;
      rstPwd();
      rstCode();
      pwdFull=0;
      codeFull=0;
      digitalWrite(RLED, HIGH);
      digitalWrite(GLED, HIGH);
```

```
          digitalWrite(BLED, LOW);
          continue;
        }
      }
      else if(state == 3){   // 输入错误过多后的锁定状态
        String instring = "";
        bool StringComplete = false;
        while(Serial.available()){   // 接收管理员在串口监视器的输入
          char inchar = Serial.read();
          delay(10);
          if(inchar != '\r')  instring += inchar;
          else if(inchar == '\r') StringComplete = true;
        }
        if(StringComplete){
          if(instring == "Gly2003"){  // 密码匹配
            state = 2;
            rstPwd();
            rstCode();
            pwdFull=0;
            codeFull=0;
            failNum=0;
            digitalWrite(RLED, LOW);
            digitalWrite(GLED, HIGH);
            digitalWrite(BLED, LOW);
            continue;
          }
        }
      }
      delay(250); // 键盘读取间隔，以免按键数据过量读入影响使用
    }
  }
```

4. 测试

测试重点为密码正确的输入，OLED 的正确显示，以及密码箱状态正确切换。一些动态测试结果（例如 LED 闪烁）已在第五周检查完毕，不便展示在报告中，因此下面只展示静态实物图。

初始状态下密码箱为“上锁状态”，黄灯亮起，密码输入可以正确显示在 OLED 上。当密码小于 4 位时，用户按 K1 按钮会导致黄灯闪烁，密码设置不成功；当密码达到最大的 15 位且用户还在输入时，黄灯同样闪烁，提示用户输入超出界限。用户输错时，按下 K2 按钮会清空已输入的密码。下图 3 为“上锁状态”静态实物图。

密码设置成功，密码箱转换为“关闭状态”，红灯亮起。用户输入密码错误会导致红灯闪烁，用户同样可以通过按钮 K2 进行重新输入，按 K1 才会使得密码箱读入已输入的密码。下图 4.为静态实物图。

密码输入正确，密码箱开启，转换为“开锁状态”，绿灯亮起，OLED 提示用户按 K2 关闭。关闭后返回“上锁状态”，实现循环。下图 5 为静态实物图。

若密码输入错误次数到达五次，则白灯亮起，密码箱进入“锁定状态”，等待管理员串口正确输入后才会切换为“开锁状态”。下图 6 为静态实物图。

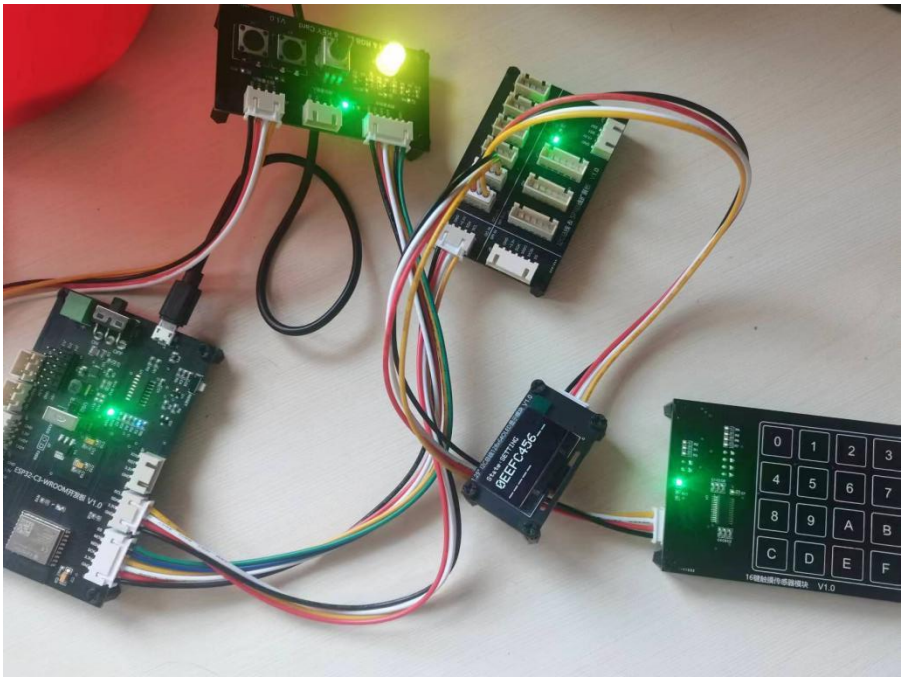

图 3 电子密码箱“上锁状态”示意图

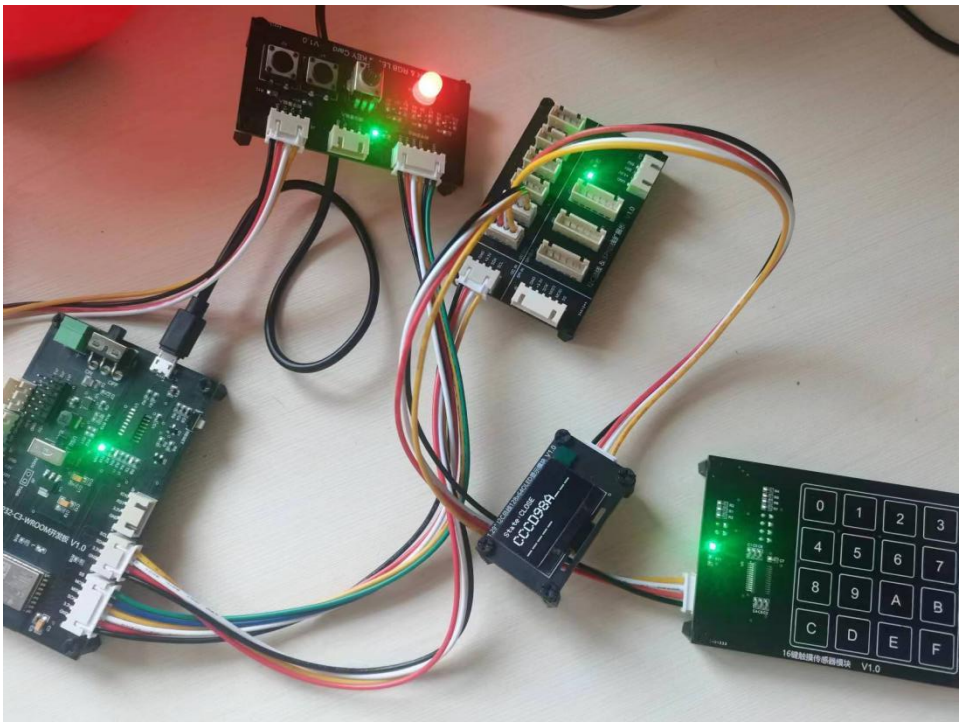

图 4 电子密码箱“关闭状态”示意图

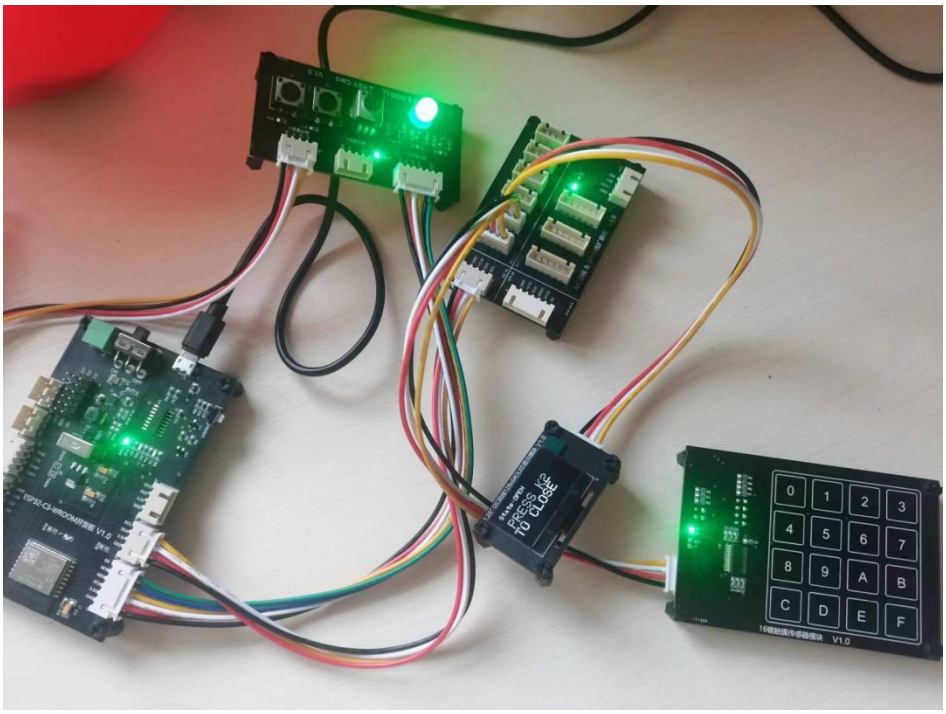

图 5 电子密码箱“开锁状态”示意图

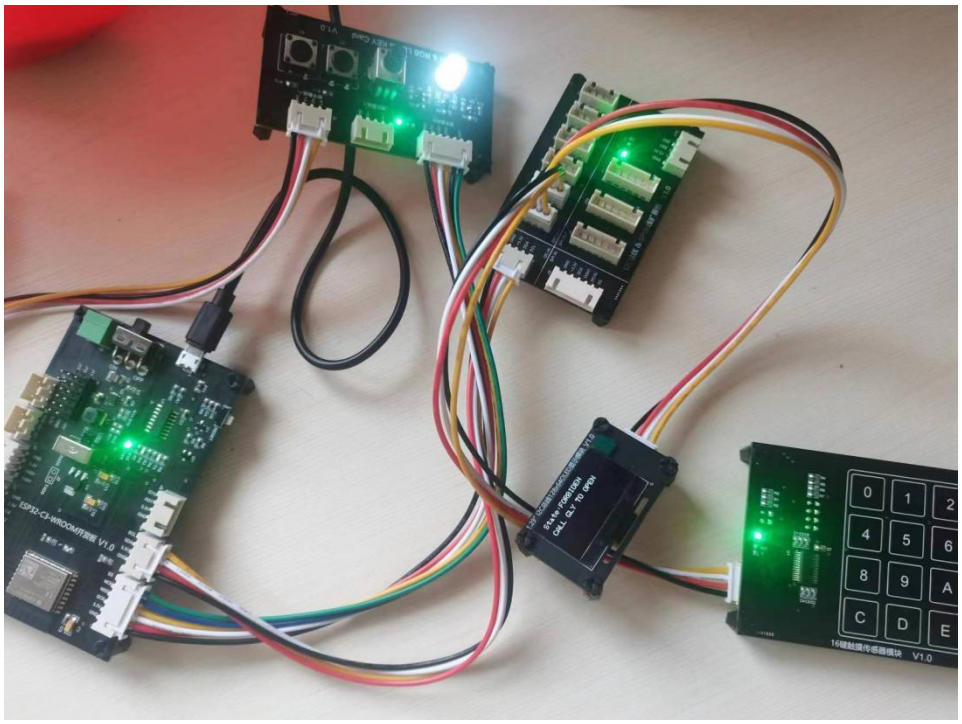

图 6 电子密码箱“锁定状态”示意图

三、 电子时钟

1. 功能设计

本项目设计并实现利用 RTC 芯片计时技术和 EEPROM 存储技术的电子时钟，其具有以下几项功能：

1) 显示时间：从 RTC 芯片获取时间后，显示在 OLED 上；
2) 更改时间：借助 16 键触摸传感器，可以对当前时间的“时”和“分”进行调整；
3) 设置闹钟：借助 16 键触摸传感器，可以设置闹钟时间的“时”和“分”；
4) 开关闹钟：借助 16 键触摸传感器，可以开关闹钟；
5) 人性化交互：OLED 会根据用户操作进行不同的显示，正在调整的时间会进行“数字”和“__”的频闪，提示用户当前更改的是“时”还是“分”。由于项目要求至多三个按钮，因此“时”和“分”会交替切换更改，无需用户操作，十分便捷；
6) 闹钟：若当前时间为闹钟时间，且闹钟开启，蜂鸣器会间歇地响起；
7) 闹钟时间存储：用户设置好闹铃时间后，它会存储到 EEPROM 内，开发板重启后数据不会消失。

2. 硬件电路设计

本项目以 ESP32-C3 开发板为核心控制板。通过开发板 J6 接口与无源蜂鸣器连接，ESP32-C3 的 IO0 引脚控制蜂鸣器的 Sig 信号；通过开发板 J4 接口借助 I2C 扩展板，与 16 键触摸传感器模块、128x64OLED 显示模块和 DS1307 实时时钟模块连接，软件内利用 I2C 总线函数库实现相关控制。

该系统的硬件电路示意图如图 7 所示。

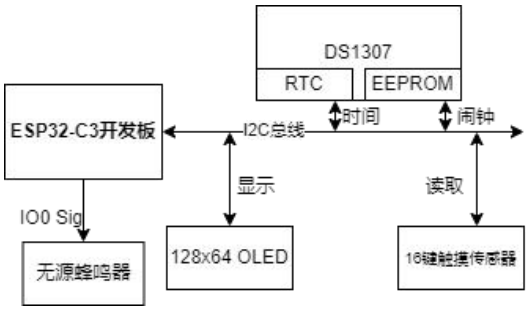

图 7 硬件电路示意图

3. 程序代码设计

本项目中使用 Adafruit 公司的函数库 Adafruit SSD1306 (2.5.9)和 Adafruit GFX(1.11.9)实现 OLED 显示屏相关功能。使用 Wire 库进行 I2C 通信。使用 RTClib 库（2.1.3）实现 RTC 时间读取、更改等功能。

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "RTClib.h"
#include <string>
```

本项目宏定义和全局变量声明如下

```
#define OLED_RESET    -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C  // OLED 地址
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define EEPROM_ADDR 0x50 // EEPROM 地址
```

```
#define TTP229_ADDR 0x57 // TTP229 地址
#define SigPin 0 // 无源蜂鸣器
// 同二.
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
RTC_DS1307 rtc;                        // rtc 实体
int isAlarm = 0;                       // 闹钟是否开启
int isWrite = 0, isChange = 0;         // 写入闹钟状态 和 更改当前时间状态
int pos = 0, editTimer = 0;            // “时”“分”设置的切换时间 和 总设置的时间
int alarmTime[2];                      // 闹钟时间暂存中间变量
bool flash = false;                    // 控制屏幕时间闪烁
DateTime now;                          // 存放读取的时间
```

EEPROM 的存和读操作均为字节存和字节读，即每次只从 EEPROM 存/读一个字节，需要注意的是，每次存/读完毕需要 delay(5)，否则会导致出错。

```
void ewrite_b(int data, int addr){
  Wire.beginTransmission(EEPROM_ADDR);   // slave device
  Wire.write(addr>>8);                   // 发送地址高八位
  Wire.write(addr & 0xFF);               // 发送地址第八位
  Wire.write(data);                      // 发送要存的一字节数据
  Wire.endTransmission();                // 结束通讯
  delay(5);                              // 延时
}

int eread_b(int addr){
  Wire.beginTransmission(EEPROM_ADDR);
  Wire.write(addr>>8);
  Wire.write(addr & 0xFF);
  Wire.endTransmission();
  Wire.requestFrom(EEPROM_ADDR, 1); // 请求从 AT24C32D 读取一个字节的数据
  int data;
  if (Wire.available())
    data = Wire.read();
  delay(5);
  return data;
}
```

setup 函数中对 rtc 的初始化参考了 RTClib 的官方示例，对 OLED 的初始化参考了 Adafruit 的官方示例

```
void setup() {
  Serial.begin(115200);
  Wire.begin();
  if (! rtc.begin()) {
    Serial.println("Couldn't find RTC");
    Serial.flush();
    while (1) delay(10);
  }
  if (! rtc.isrunning()) {
```

```
    Serial.println("RTC is NOT running, let's set the time!");
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }
  display.display();
  delay(2000);
  // Clear the buffer
  display.clearDisplay();
  alarmTime[0] = eread_b(0);  // 读取闹钟 hour
  alarmTime[1] = eread_b(1);  // 读取闹钟 minute
  pinMode(SigPin, OUTPUT);    // 蜂鸣器
}
```

num2str 函数将数字时间转换为字符串，便于 OLED 的显示

```
String num2str(uint16_t num){
  String str = "";
  if(num < 10){
    str += '0';
    str += num;
  }
  else{
    str += (num / 10);
    str += (num % 10);
  }
  return str;
}
```

dispTime 将时间显示在 OLED 上，当更改时间或设置闹钟时间操作时，会进行“时”或“分”的闪烁，提示用户当前设置的时间。此外，还会判断当前时钟状态（“时钟”、“改变时间”、“设置闹钟”）并显示，且显示当前的闹铃开启状态以及闹铃时间。

```
void dispTime(){
  display.clearDisplay();
  display.setTextSize(1);           // Normal 1:1 pixel scale
  display.setCursor(2,2);           // Start at top-left corner
  display.setTextColor(1, 0); // 黑色背景白字
  display.print(num2str(now.year()));
  display.print("/");
  display.print(num2str(now.month()));
  display.print("/");
  display.println(num2str(now.day()));

  if((isChange | isWrite) && (pos / 10) & flash){
    if(isWrite) display.print(num2str(alarmTime[0]));
```

```
  else display.print(num2str(now.hour()));
  flash = !flash;
}
else if((isChange | isWrite) && (pos / 10) & !flash){
  display.print("__");
  flash = !flash;
}
else
  if(isWrite) display.print(num2str(alarmTime[0]));
  else display.print(num2str(now.hour()));

display.print(":");

if((isChange | isWrite) && !(pos / 10) & flash){
  if(isWrite) display.print(num2str(alarmTime[1]));
  else display.print(num2str(now.minute()));
  flash = !flash;
}
else if((isChange | isWrite) && !(pos / 10) & !flash){
  display.print("__");
  flash = !flash;
}
else
  if(isWrite) display.print(num2str(alarmTime[1]));
  else display.print(num2str(now.minute()));

if(isWrite) display.println();
else  {
  display.print(":");
  display.println(num2str(now.second()));
}


if(isChange)     display.println("**Editing Time**");
else if(isWrite)  display.println("**Writing alarm**");
else            display.println("**Running**");

display.print("AlarmState:");
display.println(isAlarm?"On":"Off");

display.print("AlarmTime:");
display.print(num2str(eread_b(0)));
display.print(":");
display.println(num2str(eread_b(1)));
```

```
    display.display();
  }
```

loop 函数为主函数。从 rtc 读取时间要重复执行，紧接着是对用户键盘输入的读取（仅读取 0、1、2，其余忽略），之后程序会根据用户的输入分别进行更改时间、设置闹钟和开关闹铃的操作。

```
  void loop() {
    now = rtc.now();
    Wire.requestFrom(TTP229_ADDR, 2);
    if(Wire.available()){                                  // 键盘读取
      unsigned int c = 0;
      int button = -1;
      // read 每次读取一个字节
      for(int i = 0; i < 2; i++)
        if(i) c += Wire.read();
        else  c += Wire.read() << 8;
      if(c) button = 15 - log2(c);

      if(button == 0){                                     // 更改时间
        isChange = 1;
        while(1){
          uint8_t hour = now.hour();
          uint8_t minute = now.minute();
          Wire.requestFrom(TTP229_ADDR, 2);                // 继续读取键盘
          c = 0;
          button = -1;
          for(int i = 0; i < 2; i++)
            if(i) c += Wire.read();
            else  c += Wire.read() << 8;
          if(c) button = 15 - log2(c);

          if(button == 0){
            if(!(pos / 10)){
              pos = 0;                                     // 重置“分”设置时间
              editTimer = 0;                               // 重置总设置时间

rtc.adjust(DateTime(now.year(),now.month(),now.day(),now.hour(),(minute + 1) %
60, 0));                                                   // 改变 rtc 的“分”
            }
            else if(pos / 10){
              pos = 10;                                    // 重置“时”设置时间
              editTimer = 0;                               // 重置总设置时间
              rtc.adjust(DateTime(now.year(),now.month(),now.day(),(hour + 1) %
24,now.minute(), 0));                                      // 改变 rtc 的“时”
            }
```

```
      }
      else{            // 输入非 0 不进行操作，且积累更改位置和总设置时间
        editTimer ++;
        pos++;
        if(pos >= 20)  pos = 0;
        if(editTimer == 40){                        // 设置出口，相关变量重置
          editTimer = 0;
          isChange = 0;
          button = -1;
          break;
        }
      }
      now = rtc.now();
      dispTime();
      delay(150);
    }
  }
  if(button == 1){                                  // 设置闹钟时间
    isWrite = 1;
    while(1){
      uint8_t hour = now.hour();
      uint8_t minute = now.minute();
      Wire.requestFrom(TTP229_ADDR, 2);             // 二次读取
      c = 0;
      button = -1;
      for(int i = 0; i < 2; i++)
        if(i) c += Wire.read();
        else  c += Wire.read() << 8;
      if(c) button = 15 - log2(c);

      if(button == 1){                              // 闹钟动态设置
        if(!(pos / 10)){
          pos = 0;
          editTimer = 0;
          alarmTime[1] = (alarmTime[1] + 1) % 60;
        }
        else if(pos / 10){
          pos = 10;
          editTimer = 0;
          alarmTime[0] = (alarmTime[0] + 1) % 24;
        }
      }
      else{                                         // 循环出口，原理同上
        editTimer ++;
```

```
          pos++;
          if(pos >= 20)  pos = 0;
          if(editTimer == 40){
            editTimer = 0;
            isWrite = 0;
            button = -1;
            ewrite_b(alarmTime[0], 0);
            ewrite_b(alarmTime[1], 1);
            break;
          }
        }
        dispTime();
        delay(150);
      }
    }
    if(button == 2){                                    // 开关闹铃
      isAlarm = !isAlarm;
    }
    if(isAlarm){                                        // 闹铃
      int hour = (int)now.hour();
      int minute = (int)now.minute();
      if(hour == alarmTime[0] && minute == alarmTime[1]){
        tone(SigPin, 1000);
        delay(500);
        noTone(SigPin);
      }
    }
    dispTime();       // 相关展示
    delay(250);       // 键盘读取间隔
  }
}
```

4. 测试

测试重点为时间的正确设置，以及验证闹钟时间是否存入了 EEPROM。一些动态测试结果（例如 OLED 时间闪烁）已在第六周检查完毕，不便展示在报告中，因此下面只展示静态实物图。

图 8 为显示时间状态。

图 9 为更改时间状态，由用户按按键 0 进入，进入后，“分”和“时”会交替闪烁各两次，闪烁的部分是用户可以通过继续按按键 0 进行修改（加一）操作的。当用户不按下按键 0，闪烁才会切换，一旦用户进行了修改操作，闪烁计时会清零，确保良好的交互性。若用户不做任何操作，交替闪烁完毕后，退出此状态，切换到正常时钟显示状态。

图 10 为设置闹钟状态，图片拍摄到了“分”闪烁为“__”的情景。基本流程同上。本图中，闹钟开启，若此时时间和闹钟时间的分和时相同，且目前状态为显示时间状态，则蜂鸣器也会响起。

图 8 显示时间状态

图 9 更改时间状态

图 10 设置闹钟状态

四、 公交车报站器

1. 功能设计

本项目设计并实现利用汉字库和语音合成技术的公交车报站器，其具有以下几项功能：

1) 站点播报：语音合成模块根据当前站点进行语音播报；
2) 站点显示：LED 点阵显示当前站点；
3) 站点切换：每个站点播报完毕之后，可以按 K1 按钮，模拟公交车司机切换站点。

2. 硬件电路设计

本项目以 ESP32-C3 开发板为核心控制板。通过开发板 J6 接口与 KEY Card 数字量输入 J3 接口连接，K1 引脚接收 K1 按钮的状态并输出到开发板；通过开发板 J4 接口与 16x16LED 点阵连接，通过 I2C 总线进行点阵显示；通过开发板 J8 接口与语音合成模块相连，借助开发板自带的额外的串口控制其发声；通过开发板 J3 接口与 SPI 总线和汉字库芯片 GT30L32S4W 连接，实现借助 SPI 通信获取汉字点阵。

该系统的硬件电路示意图如图 11 所示。

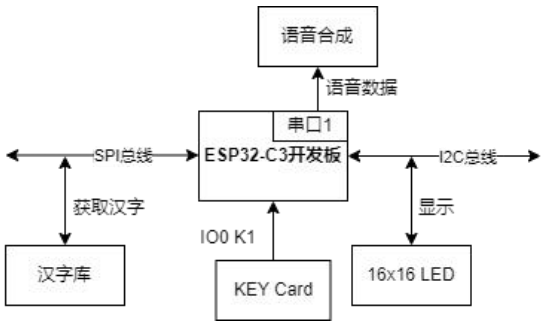

图 11 硬件电路示意图

3. 程序代码设计

本项目中使用 Adafruit 公司的函数库 Adafruit_LEDBackpack（1.5.1）实现 LED 点阵相关功能。使用 Wire 库进行 I2C 通信。使用 SPI 库进行 SPI 通信。

```
#include <Wire.h>
#include <SPI.h>
#include "Adafruit_LEDBackpack.h"
```

本项目宏定义和全局变量声明如下

```
#define K1 0                // 按钮接口
#define LEFT 0x70           // LED 左半边地址
#define RIGHT 0x71          // LED 右半边地址
#define HZK_CS 7            // 汉字库片选
#define BASE_ADDR 0x2C9D0 // GB2312 基地址
#define TAO 0xCCD5          // 汉字偏移
#define RAN 0xC8BB
#define QIAO 0xC7C5
#define BEI 0xB1B1
#define TAI 0xCCAB
#define PING 0xC6BD
#define JIE 0xBDD6
```

```
#define HU 0xBBA2
#define FANG 0xB7BB

hw_timer_t * timer = NULL; // timer, 用于站点切换
bool bo;                   // 控制站点仅播放一次用
// LED 点阵实体
Adafruit_8x16matrix matrixl = Adafruit_8x16matrix();
Adafruit_8x16matrix matrixr = Adafruit_8x16matrix();
int lop;                   // 站点
byte led_arr[32];          // 存储汉字点阵
byte ledl[16], ledr[16], zeros[16];  // 存储左右半边 LED 的点阵数据, zeros 清屏用
```

获取汉字地址用到了 msb 函数、lsb 函数以及 findByteAddr 函数，转换公式来自于数据手册

```
int msb(int addr){                        // 获取高八位地址
  return (addr & 0xFF00) >> 8;
}

int lsb(int addr){                        // 获取低八位地址
  return addr & 0x00FF;
}

int findByteAddr(int addr){               // 获取汉字点阵地址
  int MSB = msb(addr);
  int LSB = lsb(addr);
  if(MSB >= 0xA1 && MSB <= 0Xa9 && LSB >= 0xA1)
    return ((MSB - 0xA1) * 94 + (LSB - 0xA1)) * 32 + BASE_ADDR;
  else if (MSB >= 0xB0 && MSB <= 0xF7 && LSB >= 0xA1)
  return ((MSB - 0xB0) * 94 + (LSB - 0xA1) + 846) * 32 + BASE_ADDR;
}
```

存放语音合成数据的数组声明，输入格式同样来源于数据手册

```
uint8_t TRQB[13] = {0xFD, 0x00, 0x0A, 0x01, 0x00, msb(TAO), lsb(TAO), msb(RAN), lsb(RAN), msb(QIAO), lsb(QIAO), msb(BEI), lsb(BEI)};
uint8_t TPJ[11] = {0xFD, 0x00, 0x08, 0x01, 0x00, msb(TAI), lsb(TAI), msb(PING), lsb(PING), msb(JIE), lsb(JIE)};
uint8_t HFQ[11] = {0xFD, 0x00, 0x08, 0x01, 0x00, msb(HU), lsb(HU), msb(FANG), lsb(FANG), msb(QIAO), lsb(QIAO)};
```

setup 函数中，对串口、串口 1 和 SPI 总线进行初始化，对汉字库、LED 点阵、K1 按钮引脚进行初始化，对 timer 进行初始化，对相关变量进行初始化

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  Serial1.begin(115200);
  SPI.begin();
  // 汉字库
  pinMode(HZK_CS, OUTPUT);
```

```
  digitalWrite(HZK_CS, HIGH);
  // LED 点阵初始化
  matrixl.begin(LEFT);
  matrixr.begin(RIGHT);
  lop = 0;
  bo=true;
  // button
  pinMode(K1, INPUT);
  timer = timerBegin(0, 80, true);
  timerAttachInterrupt(timer, &onTimer, true);
  timerAlarmWrite(timer, 200000, true);            // 周期为 200ms
  timerAlarmEnable(timer);
}
```

timer 绑定了 onTimer 函数，该函数会检测 K1 按钮是否按下，如果是，则切换站点并开启播报

```
void onTimer(){
  if(digitalRead(K1) == LOW){
    bo = true;
    lop = (lop + 1) % 3;
  }
}
```

led 函数接收单个汉字的地址，并通过 SPI.transfer 进行通信，根据手册提示，传输设备地址和 24 位汉字地址，获取汉字库中的 LED 点阵，并进行转换，分别存放到 ledl 和 ledr 数组内，供 LED 两个半屏的正确显示

```
void led(int addr){
  int l = 0, r = 0;                  // 左右数组指针
  digitalWrite(HZK_CS, LOW);         // 开始通信
  SPI.transfer(0x03);                // 0x03 和 24 位地址输入
  SPI.transfer(addr >> 16);
  SPI.transfer(addr >> 8);
  SPI.transfer(addr & 0xFF);
  for(int i = 0; i < 32; i++)
    led_arr[i] = SPI.transfer(0x00); // 不断地读取数据
  digitalWrite(HZK_CS, HIGH);        // 结束通信
  for(int i = 0; i < 32; i++)        // 点阵信息转换
    if(i % 2) ledr[r++] = led_arr[i];
    else if(!(i % 2)) ledl[l++] = led_arr[i];
}
```

disp 函数负责点阵的显示，接收 clean 布尔值，从而决定是清屏还是显示汉字

```
void disp(bool clean){
  matrixl.clear();
  matrixr.clear();
  if(clean){
    matrixl.drawBitmap(0, 0, zeros, 8, 16, LED_ON);
    matrixr.drawBitmap(0, 0, zeros, 8, 16, LED_ON);
```

```
    }
    else{
      matrixl.drawBitmap(0, 0, ledl, 8, 16, LED_ON);
      matrixr.drawBitmap(0, 0, ledr, 8, 16, LED_ON);
    }
    matrixl.writeDisplay();
    matrixr.writeDisplay();
    delay(300);                          // 和语速基本相同
  }
```

loop 函数进行站点播报，并且播报完毕将变量 bo 置为 false，防止一个站点无限播报。同时配合 timer 绑定的 onTimer 函数，实现站点切换播报

```
  void loop() {
    if(bo){
      if(lop == 0){               // 陶然桥北
        Serial1.write(TRQB, 13); // 串口 1 通信（写入语音数据并实现播报）
        led(findByteAddr(TAO));
        disp(false);              // 显示
        led(findByteAddr(RAN));
        disp(false);
        led(findByteAddr(QIAO));
        disp(false);
        led(findByteAddr(BEI));
        disp(false);
        disp(true);               // 清屏
      }
      else if(lop == 1){          // 太平街
        Serial1.write(TPJ, 11);
        led(findByteAddr(TAI));
        disp(false);
        led(findByteAddr(PING));
        disp(false);
        led(findByteAddr(JIE));
        disp(false);
        disp(true);
      }
      else if(lop == 2){          // 虎坊桥
        Serial1.write(HFQ, 11);
        led(findByteAddr(HU));
        disp(false);
        led(findByteAddr(FANG));
        disp(false);
        led(findByteAddr(QIAO));
        disp(false);
        disp(true);
```

```
      }
      bo = false;                  // 置 false
    }
  }
```

4. 测试

本程序的测试重点在于每次进行站点切换时，按下 K1 按钮后，站点是否能正确地按顺序切换并正常播报。因此连续进行 10 次全部 3 个站点的切换，记录每个站点是否显示并播报的情况，若正确显示并播报，用√表示，否则×表示，测试结果证明切换基本上很稳定，如下表 3 所示。通过分析代码发现，站点切换不正确的两次，是由于在播放还未结束时按下 K1 按钮导致，因为 timer 的周期是 200ms，而播放三个字需要 900ms，因此出现了按两次才能播报的情况，此时 lop 变量已经+2 了，表面上来看就是一个站点被跳过了。经过后续测试，发现在每个站点完全播报完毕后，稍微停顿 0.5 秒即可稳定地正确地切换站点。

表 3 连续切换 10 轮测试结果

测试站点	陶然桥北	太平街	虎坊桥
第一次	√	√	√
第二次	√	√	√
第三次	√	√	√
第四次	√	√	√
第五次	√	√	√
第六次	√	×	√
第七次	√	√	√
第八次	√	√	√
第九次	√	√	×
第十次	√	√	√

五、 小结

在实验 1 中，我快速上手并熟悉了 Arduino，对 ESP32-C3 开发逐渐熟悉，并开始学会了看数据手册和指导书来辅助完成程序代码。

在实验 2 中，我遇到的主要问题有：不会调用 Adafruit 的库函数，不会 I2C 通信代码编写。一开始，我即使阅读了数据手册，也无从入手，后来我意识到，我应该去看一看课件和官方的实例，在运行和分析后，我成功解决了不会写代码的问题，在草稿纸上画出状态图并写出了代码。

在实验 3 中，我遇到的主要问题有：①RTC 只能显示 upload 时的时间②RTC 无法实时改变时间。对于问题①我通过阅读数据手册和示例发现，我的 Datetime now = rtc.now()写在了 setup 函数中，而没有写在 loop 函数中，导致时间无法正确流动，通过修改，我解决了问题。对于问题②，我发现在更改当前时间的循环内，虽然调用了 rtc.adjust，但是并没有在循环体内写 now = rtc.now();仅在循环体之前写了此句话，这就导致当前的时间只有在更改时间操作结束的时候，才能改变并显示出来，通过添加语句，我解决了问题②。

在实验 4 中，我遇到的问题是不会 SPI 通信代码编写，通过看实例并和同学交流探讨，经过几次尝试后，我成功听到了语音。

一周的时间很短暂，而且在第六周周末我还要参加 CCF CSP 认证考试，一边复习一边做程序，还有其他课的作业，使得时间很紧张。感谢老师的提供的指导书和数据手册给予了我很大的帮助，感谢一些官方示例给予了我代码参考，感谢完成代码过程中互帮互助的同学们，使我按时完成了程序。在检查的过程中，老师的提问暴露了我追求结果，但是忽视细节和原理的问题，我也认识到了我的问题所在，要努力在剩下八周的硬件课设中注意这个问题，并在以后的学习生活中时刻提醒自己，要打好基础，不要急功近利。