北京工業大學

BEIJING UNIVERSITY OF TECHNOLOGY

嵌入式系统设计技术

课设项目设计报告

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

项目名称：　项目名称

完成日期：　2024.04.19

教师评语					
得分		教师签字		评阅日期	

一、 功能设计

本项目作品为便携式血压计。面向居家便携式测血压和心率，用户群体面向全年龄，主要为中老年人群体。相较于传统的血压仪，本作品具有便携性强（可制成腕表随时佩戴）、测量速度快、可操作、可联网、语音实时播报的特点，其核心为血压/心率模块，仅需贴近皮肤稍等半分钟，即可开始实时测量高低压和心率。用户可以根据自身实际情况，进行报警上下限的设置，还可以自行保存测量数据，便于日后对数据的查看追踪，同时，该作品还设置了自动保存功能。每次测量完血压和心率，作品的语音合成模块会自动播报数据，便于中老年人更好地使用。该作品还具备联网功能，在手机 APP 端也可进行报警阈值设置、查看测量数据结果和历史记录的功能，同时 APP 还会对报警详细信息进行显示（如“高血压”、“心率过低”等）。

1. 测血压心率功能

用户可以在主界面 MENU，按按键 0 进入测试状态，程序首先会检测用户是否将血压/心率模块佩戴在腕部，如果正确佩戴，就会开始检测血压/心率，首次佩戴需要等待 30-60 秒的时间，随后就可不断测出三相数值（高压、低压、心率）。约每 2s 检测一次，检测结果会显示在 OLED 屏幕和 APP 上，若检测结果异常，APP 界面会显示异常信息（如“高血压 心率高”），如果用户在进入测血压模式后，按照操作提示按按键 1 开启报警，蜂鸣器会在数据异常时持续鸣响。

每一次检测，软件会读取 25 次传感器测量结果，通过均值滤波的方法，取平均值后返回，作为最终结果。

具体检测方式：

1) 传感器稳定地佩戴在腕部
2) 首次佩戴后等待约 30-60 秒，可通过 OLED 显示屏或 APP 界面查看相关高压、低压和心率数值。
3) 2)步骤完成后，每隔约 2s，检测一次，更新数值，并显示。

2. 历史记录功能

在检测血压/心率时，每隔 10s 进行一次自动记录，用户也可以通过按按键 0 进行手动记录，记录下的数据会保存在 EEPROM 中。用户可以通过在主界面按按键 1 查看历史记录，也可以在 APP 界面底部查看。可以保留最近五条记录，旧数值将被覆盖掉。

历史记录形式：年/月/日 时/分/秒 高压 低压 心率

3. 报警与报警阈值设置功能

在检测血压/心率时，如果用户的高压、低压、心率中的任一数值超出了阈值，且用户开启了报警功能，则蜂鸣器会鸣响，APP 端会显示具体症状，如“高血压 心率高”、“低血压”等。报警功能的开关情况也会记录在 EEPROM 里，并不会重置。

用户可以在主界面按 2，选择设置高压、低压和心率的阈值，进而设置三者之中某一个的报警上限和下限。当且仅当用户设置的上限大于下限时，才可以保存设置，否则设置不合法，禁止保存，直到输入正确为止。阈值会实时保存在 EEPROM 中。

4. 时钟功能

本作品提供了本地时间显示的功能，在 OLED 屏的最上方会实时显示本地时间，基于 RTC 实现。

5. 语音播报功能

在检测血压/心率时，每当数值更新，语音合成模块就会进行“高压 XXX 低压 XXX 心率 XXX”的播报，便于中老年人的使用。

6. Wi-Fi 连接功能

本作品可以通过 Wi-Fi 连接，与移动设备端实现数据互通。若连接成功，则用户可以在 APP 端查看检测结果、报警信息和历史记录。APP 端还提供了报警阈值设置，且同样会检测用户设置的合法性，确保阈值上下限合法。若连接失败，其它功能不会受到影响，会正常执行，而 Wi-Fi 模块会一直尝试连接，直到连接成功。

二、 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 J7 接口与无源蜂鸣器连接，ESP32-C3 的 IO2 引脚控制蜂鸣器的 Sig 信号；通过开发板 J8 接口与血压/心率模块连接，借助 SPI 总线与该模块进行通信；通过开发板 J6 接口与语音合成模块连接，通过软串口的方式将 IO1 和 IO0 模拟为 RX 和 TX，实现语音合成。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 1 所示。

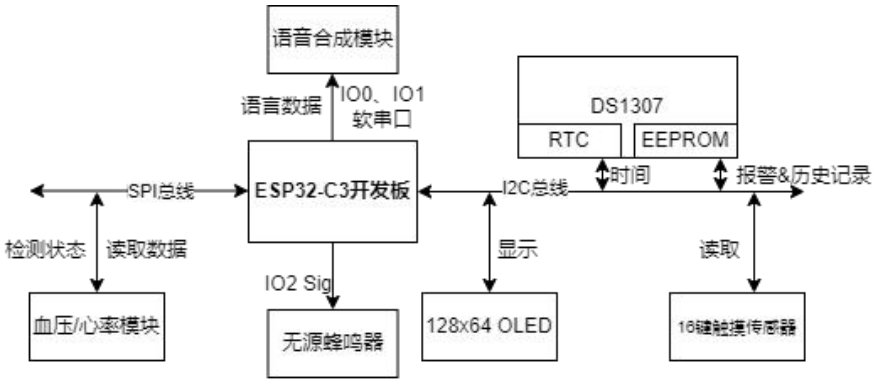

图 1 硬件电路示意图

三、 程序代码设计

本项目中使用 EspSoftwareSerial 库中的 SoftwareSerial 库(8.1.0)实现软串口模拟。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。使用 Arduino 自带的 Wire.h 实现 I2C 通信。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<string>头文件实现字符串相关处理。使用 Blinker 点灯科技的最新版库(0.3.10230510)，进行 APP 开发。

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

蜂鸣器和语音合成部分和前 4 个基础项目的代码相似，且非重要代码，故该报告中不再展示。语音合成模块有一控制变量 vMode，用于控制播报内容。

OLED 开发相关函数和常量如下，其中 Adafruit_SSD1306 对象实例化和 num2str 函数在之前的报告中已经展示过，不再赘述。OLED_为开头的宏定义，为 OLED 的五个显示模式，0-4 分别代表主界面、查看历史界面、测试模式界面、选择阈值界面、设置阈值界面。SET_为开头的宏定义，为设置阈值界面的三种显示，分别对应了设置高压、低压和心率的显示。disp 为开头的函数是屏幕显示函数。3 个 byte 类型的变量见注释。

```
#define OLED_MENU 0
#define OLED_HISTORY 1
#define OLED_TESTING 2
#define OLED_SELECT 3
#define OLED_SET 4
#define SET_SBP 0
#define SET_DBP 1
#define SET_HR 2
byte dispState = OLED_MENU;          // 当前 OLED 的显示模式，默认为主界面
byte setState = -1;                  // 当前设置谁的阈值，对应三个 SET_
byte hisPage = 0;                    // 历史记录翻页
void dispTime();
void dispHis();
void disp0();
void disp1();
void disp2();
void disp3();
void disp4();
```

RTC 模块和 EEPROM 的读写和基础项目中的相同，因此不再赘述。

血压/心率模块相关开发如下，详见注释。

```
byte BPstatueCMD[6] = {0xF8, 0x00, 0x00, 0x00, 0x00, 0x00};
// 血压传感器读取状态命令帧
byte BPstatue[6];                                    // 血压传感器返回的状态
byte BPdataCMD[6] = {0xFD, 0x00, 0x00, 0x00, 0x00, 0x00};
```

```
// 血压传感器读取数据命令帧
byte BPdata[6];                                          // 血压传感器返回的数据
int SBP = 0;                                             // 收缩压（高压）
int DBP = 0;                                             // 舒张压（低压）
int HR = 0;                                              // 心率
int BPnum = 0;                                           // 血压采集次数
int SBPsum = 0;                                          // 收缩压的和
int DBPsum = 0;                                          // 收缩压的和
int HRsum = 0;                                           // 心率的和
byte History[9][5];                                // 最近 5 条历史，存储时间和三数据
byte Range[6];          //maxSBP, minSBP, maxDBP, minDBP, maxHR, minHR;
#define MAXSBP 150      // 上下限宏定义
#define MINSBP 120
#define MAXDBP 90
#define MINDBP 60
#define MAXHR 95
#define MINHR 60
bool loading = false;                                    // 是否正在获取数值中
bool rOver = false;                                      // 是否请求完毕
bool first_load = false;                                 // 是否为本次第一次请求
byte autoSave = 0;                                       // 自动保存
void record();                                           // 记录历史函数
bool accept;                                             // 是否接受阈值设置
```

按键的读取采用了计时器，详细开发设计见下文。TTP229_开头的宏定义为键盘模块状态，KEY_开头的宏定义为按键状态。

```
#define TTP229_WAITING 0              // 等待读取
#define TTP229_GETTING 1              // 正在读取
#define TTP229_FINISHED 2             // 读取完毕
#define KEY_COMMON 0                  // 没按键被按
#define KEY_PRESS 1                   // 按键按下
#define KEY_RELEASE 2                 // 按键释放
uint16_t c = 0;                       // 暂存 TTP229 返回的数据
byte ttp229_state = TTP229_WAITING;   // TTP229 初始状态
int button = -1, last_button = -1;    // 本次输入和上一次的输入
byte button_mem = -1;                 // 键盘读取缓存变量
byte key_state = KEY_COMMON;          // 按键状态
bool clicked = false;                 // 是否有按键刚被按下
```

计时器的开发详见下文，该部分主要定义了 1ms 为最小单位的时间以及对应的计时器。时间有 15ms、1、2、3、10 秒。下面的函数绑定在计时器上。

```
void on_Timer();
```

Blinker 开发用到了四种组件，分别是 BlinkerSlider、BlinkerButton、BlinkerNumber、BlinkerText，前二者用于用户主动输入，后二者用于数据显示。组件绑定的函数和辅助开发的函数如下。以_cb 为结尾的函数是绑定在 Slider 上的，按顺序分别对应高压报警上下限、低压报警上下限和心率报警上下限的设置（共 6 个函数）；两个返回值类型为 String 的函数分别用于获取历史记录时间和数据字符串。

```
void sas_cb(int32_t value){
  if((byte)value <= Range[1]) {          // 如果设置的数值小于下限
    Slider_maxsbp.print(Range[1]);       // 组件数值设置为下限
    Range[0] = Range[1];                 // 实际上限设置为下限（同步）
  }
  else  Range[0] = (byte)value;          // 否则更新上限
  Number_maxsbp.print(value);            // 显示阈值的组件更新上限
  ewrite_b(Range[0], 0);                 // EEPROM 写入
}
// 逻辑同上
void sis_cb(int32_t value);
void sad_cb(int32_t value);
void sid_cb(int32_t value);
void sah_cb(int32_t value);
void sih_cb(int32_t value);

// 逻辑简单，不再展示
String getTimeStr();
String getValStr();

void up_call(const String & state){
  if(hisPage > 0)  hisPage--;                // 翻页
  String str = String(hisPage + 1) + "/5";  // APP 显示页码
  Text_his.print(str);                       // 相关组件更新信息
  Text_time.print(getTimeStr());
  Text_val.print(getValStr());
}
// 逻辑同上
void down_call(const String & state);
```

1. 程序总体结构设计

程序总体结构设计及流程图如下图 2 所示，图 2 主要展示了 setup 函数的工作，以及 loop 函数的流程。

setup 函数主要完成串口和组件等的初始化工作。loop 函数的主要工作为读取键盘、执行键盘操作、测血压/心率、以及 OLED 显示等逻辑判断与对应操作。其中键盘读取用到了计时器，因此本项目中除了串口 write 和 read 为了避免错误用到了 delay 函数之外，没有任何一处用到 delay，保证了 OLED 界面时间的显示不受任何操作的影响，正常流动。由于没有 delay，一些逻辑判断对应了若干 bool 型或整形变量，用于状态转换控制等，下文会详细展开说明。

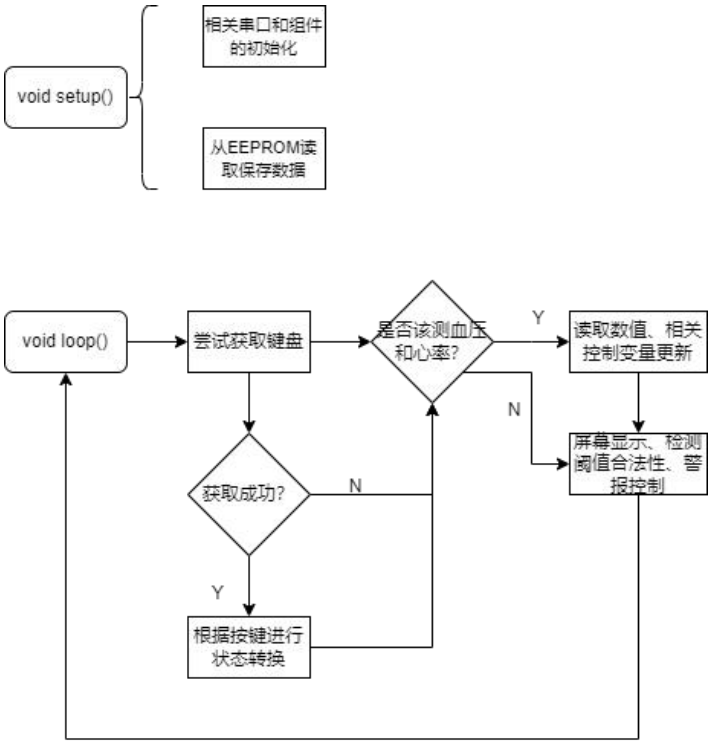

图 2 总体设计

2. 定时器模块设计

定时器模块设计包含了部分的键盘模块设计、血压/心率模块设计、蜂鸣器设计、语音播报设计。其主要功能有：控制键盘的读取（基于状态机）、血压/心率检测周期设置、佩戴情况检测、蜂鸣器鸣响周期设置、语音播报间隔设置。其中状态机为核心代码，状态机如下图 3 所示，对应的代码在图片正下方。当状态机完整地循环一个周期，才判定为某个按键被按下，这样的设计，确保了用户按下按键时，无论按下的时间有多久，最后都只能识别到一次按下，确保了软件的正常运转。

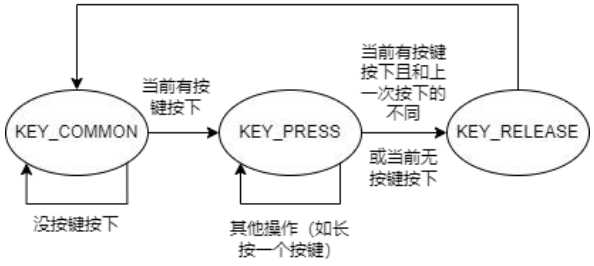

图 3 按键状态机

```
void ARDUINO_ISR_ATTR on_Timer(){
  // 按键状态机
  if (TTP229_FINISHED == ttp229_state) {
    if(button != -1){
      // 刚按下
      if(key_state == KEY_COMMON){
        key_state = KEY_PRESS;
      }
```

```
      else{
        key_state = key_state;
      }

      if(key_state == KEY_PRESS){
        if(last_button != button){
          key_state = KEY_RELEASE;
        }
        else key_state = key_state;;
      }

      if(key_state == KEY_RELEASE){
        key_state = KEY_COMMON;
        clicked = true;
      }
    }
    ttp229_state = TTP229_WAITING;
  }

  // 15ms 识别一次
  if (TTP229_WAITING == ttp229_state) {
    if (15 == ++counter_15ms) {
      counter_15ms = 0;
      ttp229_state = TTP229_GETTING;
    }
  }
  // 其他代码...
}
```

3. OLED 模块设计

OLED 模块的设计重点在于，OLED 显示屏每时每刻要显示什么内容。上文提到，OLED 的显示模式有五种，个别显示模式内部也进行了划分，具体状态转换如下图 4 所示，图中数字为用户在此状态下按下的按键。OLED_TESTING 状态分为了两部分，第一部分是传感器首次佩戴，还在获取数据中，屏幕显示“testing”的状态，等待 30-60 秒后，开始获取到数据，此时进入第二部分，屏幕显示数值。当检测到传感器没有贴近手腕，那么会返回第一部分。OLED_SET 分为了三种显示模式，分别是设置高压、低压和心率阈值的模式。

OLED 显示的转换，与键盘模块密不可分，代码详见 4.

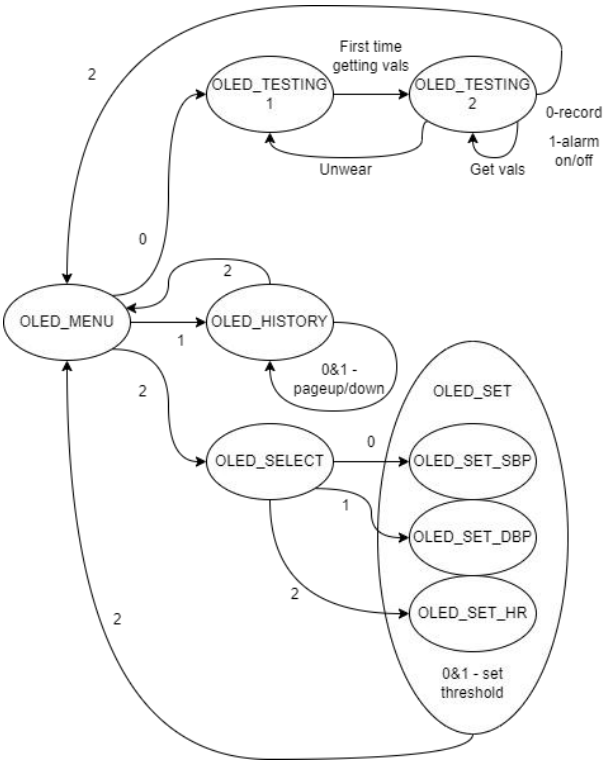

图 4 OLED 状态转换

4. 键盘模块设计

键盘模块主要功能为读取用户从 TTP229 输入的键值。键盘状态机封装在了定时器模块内部，根据键盘反馈进行 OLED 状态转换的代码如下。在读取到键值执行相关 OLED 状态转换时，还会涉及到其他模块的控制：OLED_SET 状态，按按键 2 进行保存设置返回 MENU 的过程，涉及到数据写入 EEPROM；OLED_TESTING 状态，按按键 1 开关蜂鸣器，涉及到了蜂鸣器鸣响的控制，按按键 0 保存历史记录，涉及到了历史数据写入 EEPROM。

```
if(ttp229_state == TTP229_GETTING){
    c = 0;
    Wire.requestFrom(TTP229_ADDR, 2);
    c += (Wire.available()) ? (Wire.read() << 8) : 0;
    c += (Wire.available()) ? (Wire.read()) : 0;
    while (Wire.available()) Wire.read();
    if(c) {
      button = 15 - log2(c);
      ttp229_state = TTP229_FINISHED;
      // 识别到了
      if(clicked) {
        Serial.println(button, HEX);
        // 开始操作
        if(dispState == OLED_MENU){
          if(button == 0) {
            dispState =  OLED_TESTING;
```

```
      loading = true;
      rOver = false;
      first_load = true;
    }
    if(button == 1) dispState = OLED_HISTORY;
    if(button == 2) dispState = OLED_SELECT;
  }
  else if(dispState == OLED_HISTORY){
    if(button == 0)
      if(hisPage > 0)  hisPage--;
    if(button == 1)
      if(hisPage < 4)  hisPage++;
    if(button == 2) dispState = OLED_MENU;
  }
  else if(dispState == OLED_TESTING){
    if(loading && first_load) {
      if(button == 2) {
        dispState = OLED_MENU;  // 读取的时候只能退出
        loading = false;
        rOver = false;
        first_load = false;
        BPnum = 0;       //血压采集次数
        SBPsum = 0;      //收缩压的和
        DBPsum = 0;      //收缩压的和
        HRsum = 0;       //心率的和
        beep = false;
      }
    }
    else if(!first_load){ // 读取完毕就能正常操作了
      if(button == 0) record();   // 记录
      if(button == 1) {          // 警报
        alarm_on = !alarm_on;
        ewrite_b(alarm_on, 51);
        if(!alarm_on) beep = false;
      }
      if(button == 2) {          // 退出
        dispState = OLED_MENU;
        loading = false;
        rOver = false;
        first_load = false;
        BPnum = 0;       //血压采集次数
        SBPsum = 0;      //收缩压的和
        DBPsum = 0;      //收缩压的和
        HRsum = 0;       //心率的和
```

```
              beep = false;
            }
          }
        }
        else if(dispState == OLED_SELECT){
          if(button == 0){
            dispState = OLED_SET;
            setState = SET_SBP;
          }
          if(button == 1){
            dispState = OLED_SET;
            setState = SET_DBP;
          }
          if(button == 2){
            dispState = OLED_SET;
            setState = SET_HR;
          }
        }
        else if(dispState == OLED_SET){
          if(button == 0){
            if(setState == SET_SBP) Range[0] = (Range[0] + 1 > MAXSBP) ? MINSBP :
Range[0] + 1;
            if(setState == SET_DBP) Range[2] = (Range[2] + 1 > MAXDBP) ? MINDBP :
Range[2] + 1;
            if(setState == SET_HR)  Range[4] = (Range[4] + 1 > MAXHR) ? MINHR :
Range[4] + 1;
          }
          if(button == 1){
            if(setState == SET_SBP) Range[1] = (Range[1] - 1 < MINSBP) ? MAXSBP :
Range[1] - 1;
            if(setState == SET_DBP) Range[3] = (Range[1] - 1 < MINDBP) ? MAXDBP :
Range[3] - 1;
            if(setState == SET_HR)  Range[5] = (Range[5] - 1 < MINHR) ? MAXHR :
Range[5] - 1;
          }
          if(button == 2 && accept){
            for(int i = 0; i < 6; i++)  ewrite_b(Range[i], i);
            dispState = OLED_MENU;
          }
        }

        // 重置
        clicked = false;
      }
```

```
// 其他代码...
    }
// 锁存
    button_mem = button;
}
```

5. 语音播报模块设计

语音播报通过软串口实现，用 write 函数将语音播报数据输入串口即可实现播报，而播报的重点问题在于，相邻两次 write 需要有一定的时间间隔，否则第一次 write 的语音内容还未播报完毕，第二次 write 的语音内容就覆盖了第一次，造成语音播报不完整的问题。若在 loop 函数中，通过 write 与 delay 配合的方式进行语音播报，虽然能够避免上述问题，但是 delay 过程中，无法进行键盘读取、时钟实时显示、蜂鸣器持续鸣响等操作，因此本项目选择用定时器实现语音延时功能——每当血压心率的数值刷新，bool 类型变量 bo 由 false 变为 true，定时器中如下代码激活，实现 1s 延时，保证了每一组词都能完整播报。播报结束将 bo 变量重置为 false，等待下一次数值刷新，实现循环。

```
// 播放间隔
  if(dispState == OLED_TESTING && bo){
    if(0 == counter_1s_1){
      if(vMode == 0) myS.write(vSBP, 9);
      else if(vMode == 1) voice(SBP);
      else if(vMode == 2)  myS.write(vDBP, 9);
      else if(vMode == 3) voice(DBP);
      else if(vMode == 4)  myS.write(vHR, 9);
      else if(vMode == 5) voice(HR);
      counter_1s_1++;
    }
    else if(timer_1s == ++counter_1s_1){
      counter_1s_1 = 0;
      vMode++;
    }
  }

  if(bo && vMode == 6){
    bo = false;
    vMode = 0;
  }
```

6. 血压/心率模块设计

血压/心率模块的代码主要写在 loop 函数内，位于 4.中展示的代码中的“其他代码”位置，同时有若干变量进行控制，确保其实时性。同时，该模块还涉及了自动保存、APP 界面数值显示。具体代码和注释如下

```
// 随时读取血压计
    if(dispState == OLED_TESTING){
```

```
    if(!rOver && loading){                    // 阶段1：开始读取，且未完成状态查询
      Serial1.write(BPstatueCMD,6);          //发送查询传感器状态命令帧
      delay(10);
      if (Serial1.available() > 0)          //读取传感器返回的状态帧
          for (int i = 0; i <= 5; i++)
            BPstatue[i] = Serial1.read();
      if((BPstatue[0]== 0xF8) || (BPstatue[3]== 0x07)) {    // 读取成功,转换
        BPnum = 0;        //血压采集次数
        SBPsum = 0;       //收缩压的和
        DBPsum = 0;       //收缩压的和
        HRsum = 0;        //心率的和
        rOver = true;                      // 状态转换控制
      }
    }
    else if(rOver && loading){            // 阶段2：完成状态查询，未完成读取
      Serial1.write(BPdataCMD,6);          //发送读取数据（血压/心率）命令帧
      delay(10);

      if (Serial1.available() > 0)        //读取传感器返回的数据
        for (int i = 0; i <= 5; i++)
          BPdata[i] = Serial1.read();
      if((BPdata[0] == 0xFD) && BPdata[1] != 0xFF && BPdata[1]!=0x00){
        SBPsum += BPdata[1];         //收缩压求和
        DBPsum += BPdata[2];         //舒张压求和
        HRsum += BPdata[3];          //心率求和
        BPnum++;
      }
      if(BPnum >= 25)  {            // 读取完毕
        loading = false;            // 状态重置
        first_load = false;
        rOver = false;
        BPnum = 0;
        SBP = SBPsum/25;         //求收缩压平均值
        DBP = DBPsum/25;         //求舒张压平均值
        HR = HRsum/25;           //求心率平均值
        Serial.print("SBP = "); Serial.print(SBP, DEC);
        Serial.print("\tDBP = "); Serial.print( DBP, DEC);
        Serial.print("\theart_rate = "); Serial.println( HR, DEC);
        if(Blinker.connected()){
          Number_DBP.print(DBP);
          Number_SBP.print(SBP);
          Number_HR.print(HR);
        }
        bo = true;              // 开启播报
```

```
          autoSave++;
          if(autoSave == 5){
            record();        // 自动保存
            autoSave = 0;
          }
        }
      }
    }else{
      if(Blinker.connected()){
        Number_DBP.print(0);
        Number_SBP.print(0);
        Number_HR.print(0);
      }
    }
```

7. EEPROM 模块设计

本项目中，EEPROM 模块主要完成阈值、历史数据和蜂鸣器开启状态的读取和写入，读取操作在 setup 中完成，不再赘述。阈值和蜂鸣器开启状态的写入在键盘模块中已全部展示。历史数据的写入由 record 函数完成，在键盘模块代码中，由用户在 OLED_TESTING 状态下按按键 0 调用；在血压/心率模块代码中，由自动保存调用。record 代码如下

```
void record(){
  for(int i = 4; i >= 1; i--){           // 数组后移
    for(int j = 0; j < 9; j++){
      History[j][i] = History[j][i-1];
    }
  }
  History[0][0] = now.year() % 100;
  History[1][0] = now.month();
  History[2][0] = now.day();
  History[3][0] = now.hour();
  History[4][0] = now.minute();
  History[5][0] = now.second();
  History[6][0] = SBP;
  History[7][0] = DBP;
  History[8][0] = HR;
  for(int i = 0; i < 5; i++)
    for(int j = 0; j < 9; j++)
      ewrite_b(History[j][i], 6 + i * 9 + j);
  Serial.println("recored!");
}
```

对于阈值的写入，需要特别判定用户设置的上限是否大于下限，否则不允许保存，由 accept 变量进行控制，控制代码在 loop 函数中，代码如下

```
// accept
```

```
    if((Range[0] <= Range[1]) || (Range[2] <= Range[3]) || (Range[4] <= Range[5]))
accept = false;
    else  accept = true;
```

8. 蜂鸣器模块设计

蜂鸣器模块的设计较为简单，其鸣响间隔由定时器模块实现，代码如下。其鸣响于 loop 函数实现。

```
  // 定时器中的 alarm
  // 1s 刷新一次蜂鸣器间隔
    if(alarm_on && dispState == OLED_TESTING){
      if(timer_1s == ++counter_1s){
        counter_1s = 0;
        beep = !beep;
      }
    }

  // loop 中的 alarm
    if(alarm_on && dispState == OLED_TESTING)
      if(SBP > Range[0] || SBP < Range[1] || DBP > Range[2] || DBP < Range[3]
|| HR > Range[4] || HR < Range[5])
        if(beep)  tone(SigPin, 1000);
        else  noTone(SigPin);
    else if(!alarm_on)  beep = false;
```

9. APP 模块设计

APP 模块基于 Blinker 实现，主要涉及了组件的交互和信息实时更新。在血压/心率模块中，一旦三项数值更新，若此时与 APP 连接成功，则会实时更新 APP 端数值。在 loop 函数中，有如下代码进行报警信息的显示。

```
    if(Blinker.connected()){
      Slider_maxsbp.print(Range[0]);
      Slider_minsbp.print(Range[1]);
      Slider_maxdbp.print(Range[2]);
      Slider_mindbp.print(Range[3]);
      Slider_maxhr.print(Range[4]);
      Slider_minhr.print(Range[5]);
      Number_maxsbp.print(Range[0]);
      Number_minsbp.print(Range[1]);
      Number_maxdbp.print(Range[2]);
      Number_mindbp.print(Range[3]);
      Number_maxhr.print(Range[4]);
      Number_minhr.print(Range[5]);
      String temp = "";
      int isA = 0;
```

```
    if(SBP > Range[0] || DBP > Range[2]){
      temp += "高血压";
      isA++;
    }
    if(SBP < Range[1] || DBP < Range[3]){
      temp += "低血压";
      isA++;
    }
    if(HR > Range[4])
      if(isA) temp += " 心率高";
      else temp += "心率高";
    else if(HR < Range[5])
      if(isA) temp += " 心率低";
      else temp += "心率低";
    else if(!isA)  temp += "正常";

    if(dispState != OLED_TESTING )   Text_test.print("待测试");
    else  Text_test.print(temp);
  }
```

APP 组件绑定的函数已在 1.中展示，故不再赘述。

四、 测试

内容重点：测试方法和对应结果或现象的说明，可使用表格或实物照片等方式辅助文字说明。

如上文所述，本项目功能分别为：测血压心率、历史记录、报警与报警阈值设置、时钟、语音播报、Wi-Fi 连接。其中报警功能、时钟的实时显示和语音播报已在验收时展示，其动态测试过程不便于在报告中撰写，因此只给出字面测试结果。实物连接图如图 5 所示

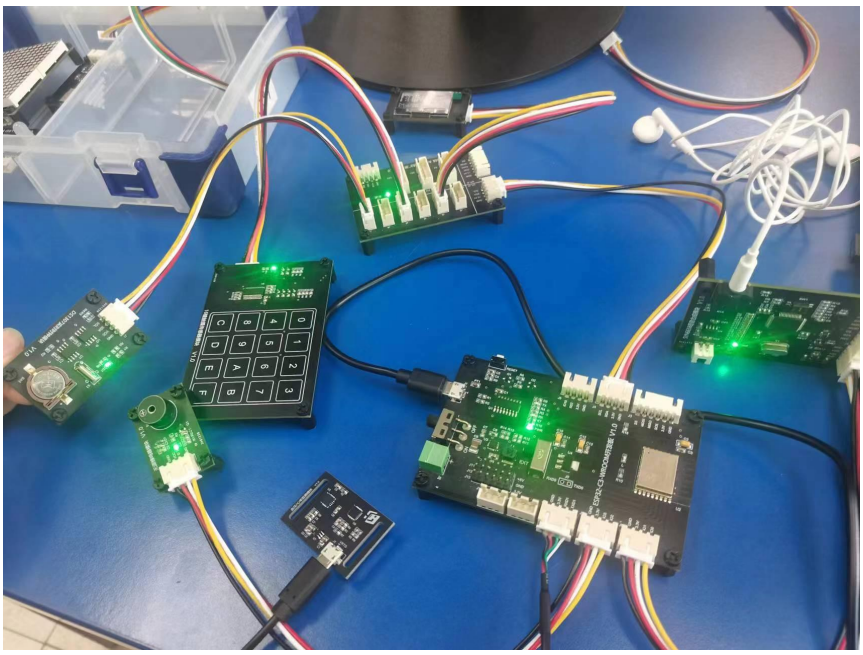

图 5 实物连接图

1. 测血压心率功能测试

测血压心率功能重点在于数值的稳定性，在静坐状态下，连续测试 10 次三项数值，得到结果如表 1 所示。如表 1 可看出，只有第一次测试出的高低压有些偏低，而之后的测试数据非常稳定。根据分析发现，第一次测试的数据不准确，可能与本日刚刚佩戴血压/心率模块有关，在后续的测试中，在同一块皮肤上反复贴近和远离模块，发现数值稳定。

表 1 血压/心率测试结果

测试数据	高压	低压	心率
测试结果 1	118	65	83
测试结果 2	138	75	80
测试结果 3	138	76	85
测试结果 4	138	76	83
测试结果 5	138	76	83
测试结果 6	138	75	84
测试结果 7	138	76	81
测试结果 8	140	76	80
测试结果 9	140	75	81
测试结果 10	140	78	83

2. 历史记录功能测试

历史记录功能测试重点在于，用户能否在历史记录查看界面上下翻页，且翻页时软件能够正确地读取相关数据并显示在屏幕上。在验收时经测试，本功能正常运作，没有发生错误，界面如图 6 所示

图 6 历史记录

3. 报警阈值设置功能测试

报警阈值设置功能重点在于，用户设置的数值的合法性检测，以及数据是否能成功被保存。如图 7 所示，如果用户设置的阈值上限小于下限，那么 OLED 下方会出现提示，此时按 2 进行保存操作无效，当且仅当上限大于下限时，操作才有效。设置 HR 上限 95 下限 94，此时保存回到 MENU，发现 HR 的上下限已经保存成功，如图 8

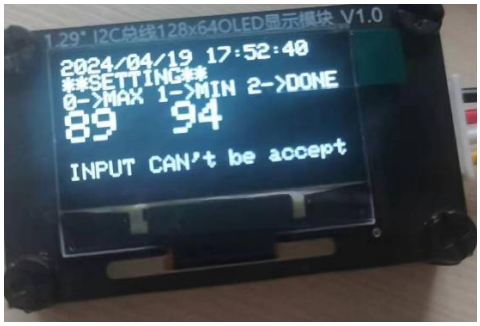

图 7 无法保存

图 8 保存成功

4. 报警功能测试

在验收时，将阈值设定为极端数据，开启 alarm，OLED 上可以看见 alarm 状态显示为 on，同时蜂鸣器开始间断鸣响，关闭 alarm 后蜂鸣器也停止鸣响。在验收时，alarm 在开启状态下进行结束测试的操作后，蜂鸣器并未停止鸣响，这不符合本项目最初预期，已在验收之后修改。

5. 时钟功能测试

时钟功能测试的重点为，无论用户进行任何操作，都不影响时钟一秒一秒地流动和现实。在验收时，该功能已得以验证。

6. 语音播报功能测试

语音播报的测试重点为，每当血压和心率数值更新时，语音能否准时播报，且播报内容能否完整播出。在验收时，该功能已得以验证。

7. Wi-Fi 连接功能测试

Wi-Fi 连接功能测试的重点在于，APP 端能否与硬件端进行实时的数据交互。如图 9 所示，此时正处于测量血压/心率状态，能够查看到三项数值以及报警提示，页面最下方能够正确地查看到历史记录。

在 3.中测试阈值设置时，将心率设置为上限 95 下限 94，APP 端如图 10 所示，可见硬件端更改的数据可以实时在 APP 端更新。现在在 APP 端重新设置心率下限如图 11，如图 12 可见在 APP 端更改的数据可以实时在硬件端更新。

图 9 APP 端更新

图 10 APP 端设置

图 11 硬件端更新

五、 小结

在课设项目设计过程中，我遇到最大的问题就是定时器内部代码编写时，实际测试结果与预期不符。比如按键状态机的编写，我按照自己草稿上的状态机，进行初步编写后，发现按键无论怎么按也没有反应。通过 print 输出状态，发现是初始状态的转换出了问题。更改之后，又出现了按键能够进入 PRESS 状态，进入不了 RELEASE 状态，稍作修改后，又出现了长按一直输出的非预期结果。通过分析发现，这一部分的代码需要判断本次与上一次输入是否相同，由于被按下的按键默认为-1，那么如果本次按键被按下，则相邻两次输入不同，当用户释放按键，则上一次输入为按下的按键，本次输入变回-1，如此状态才会成功且正确地转换。通过设置一个缓存变量，在 loop 函数中进行按键记录，最后配合上述分析结果进行状态转换代码更改，最后我成功解决了问题。

另一个问题就是语音播报延时问题。一开始，我一直在 loop 函数中思考如何保证语音完整播报，且不影响其他代码正常运作。一开始我写了一些控制变量，让语音播报在一次循环内只播放两三个字，这样延时不到 1 秒，似乎不影响时钟的正常流动，可是我却忽略了时钟以外，例如键盘读取操作的正常运转。在思考之后，我决定在定时器中实现语音播报，最后解决了问题。

随着项目的验收，嵌入式课接近了尾声。回首这八周的学习，我最大的收获就是“如何更好地去写代码”，正如老师在第七周课上所说，如果在构思环节偷了懒，那么在写代码环节就会把偷懒的时间全部补回来，甚至还会浪费多余的时间。在本学期课余时间，我在玩的一款游戏，也涉及到了一些复杂的流程以及多种产线的搭建和配合，这和写课设代码有着异曲同工之处——在开始动工之前，都需要建立相应的方案，构思出一个大框架，有初步的预期等，需要打草稿，画流程，甚至提前设想出一些可能出的错误以及解决方案。

在第七周听课之前，我在课余时间玩游戏的时候，在制作某化工材料产线时，就打了慢慢一张纸的草稿，详细地画出了制作该产线的能源需求，材料需求，以及附属的产线，这些草稿和方案对于我在游戏中的产线搭建给予了非常大的便利，而第七周听课之后，我恍然大悟，写代码做工程，都应当如此。

于是我这么做了，也便成为了第八周唯一一个按时验收完全部内容的学生，并且没有耽误了第八周早晨选修课的复习，更没有耽误了每天至少 1 小时的娱乐时间。

感谢老师能教我超脱于课堂知识之外的人生经验，我也感谢自己能三省吾身，获得了小小的成功。