



本项目通过鸿蒙 App 控制 Hi3861 开发板,监测用户的心率,基于心跳变化,判断用户是否说谎。

3.1 总体设计

本部分包括系统架构和系统流程。

3.1.1 系统架构

系统架构如图 3-1 所示,Hi3861 开发板与外设引脚连线如表 3-1 所示。

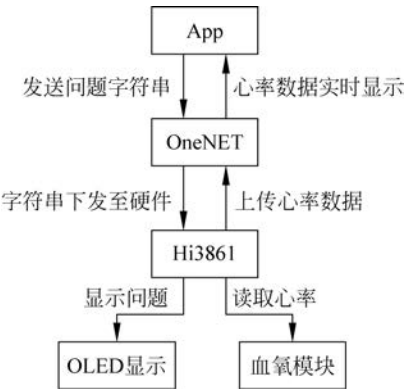


图 3-1 系统架构

表 3-1 Hi3861 开发板与外设引脚连线

Hi3861 开发板	OLED	MAX30102 血氧模块
GND	GND	GND
3.3V	VIN	VIN
GPIO13	SDA	SDA
GPIO14	SCL	SCL

3.1.2 系统流程

系统流程如图 3-2 所示。



图 3-2 系统流程

3.2 模块介绍

本部分包括 OLED 显示、WiFi 模块、血氧模块、OneNET 云平台 and 前端模块。下面分别给出各模块的功能介绍及相关代码。

3.2.1 OLED 显示

本模块将问题呈现到 OLED 显示屏上,重点在于实现 I2C 通信,调用其常用 API 接口,如表 3-2 所示。

表 3-2 API 接口与说明

API 接口	说 明
I2cInit(WifiIotI2cIdx id, unsigned int baudrate)	用指定的波特速率初始化 I2C 设备
I2cDeinit (WifiIotI2cIdx id)	取消初始化 I2C 设备
I2cWrite (WifiIotI2cIdx id, unsigned short deviceAddr, const WifiIotI2cData * i2cData)	将数据写入 I2C 设备
I2cRead (WifiIotI2cIdx id, unsigned short deviceAddr, const WifiIotI2cData * i2cData)	从 I2C 设备中读取数据
I2cWriteread (WifiIotI2cIdx id, unsigned short deviceAddr, const WifiIotI2cData * i2cData)	向 I2C 设备发送数据并接收数据响应
I2cRegisterResetBusFunc(WifiIotI2cIdx id, WifiIotI2cFunc pfn)	注册 I2C 设备回调
I2cSetBaudrate(WifiIotI2cIdx id, unsigned int baudrate)	设置 I2C 设备的波特率

OLED 显示的相关代码如下：

```

//显示字符串
//x,y:起点坐标
//size1:字体大小
//chr:字符串起始地址
void OLED_ShowString(u8 x,u8 y,char *chr,u8 size1)
{
    while((*chr>=' ')&&(*chr<='~'))    //判断是否为非法字符!
    {
        OLED_ShowChar(x,y,*chr,size1);
        x+=size1/2;
        if(x>128-size1)                //换行
        {
            x=0;
            y+=2;
        }
        chr++;
    }
}
//在指定位置显示一个字符,包括部分字符
//x:0~127
//y:0~63
//size:选择字体 12/16/24
//取模方式:逐列式
void OLED_ShowChar(u8 x,u8 y,u8 chr,u8 size1)
{
    u8 i,m,temp,size2,chr1;
    u8 y0=y;
    size2=(size1/8+((size1%8)?1:0))*(size1/2); //得到字体一个字符对应点阵集所占的字节数
    chr1=chr-' ';                               //计算偏移后的值
    for(i=0;i<size2;i++)
    {
        //temp=asc2_1206[chr1][i];
        if(size1==12)
        {temp=asc2_1206_1[chr1][i];} //调用 1206 字体
        else if(size1==16)
        {temp=asc2_1608_1[chr1][i];} //调用 1608 字体
        else return;
        for(m=0;m<8;m++)                //写入数据
        {
            if(temp&0x80) OLED_DrawPoint(x,y);
            else OLED_ClearPoint(x,y);
            temp<<=1;
            y++;
            if((y-y0)==size1)
            {
                y=y0;
                x++;
                break;
            }
        }
    }
}
//清屏函数
void OLED_Clear(void)
{

```

```

    u8 i, n;
    for (i=0; i<8; i++)
    {
        for (n=0; n<128; n++)
        {
            OLED_GRAM[n][i]=0;//清除所有数据
        }
    }
    OLED_Refresh(); //更新显示
}
//画点
//x:0~127
//y:0~63
void OLED_DrawPoint(u8 x, u8 y)
{
    u8 i, m, n;
    i=y/8;
    m=y%8;
    n=1<m;
    OLED_GRAM[x][i]|=n;
}
//清除一个点
//x:0~127
//y:0~63
void OLED_ClearPoint(u8 x, u8 y)
{
    u8 i, m, n;
    i=y/8;
    m=y%8;
    n=1<m;
    OLED_GRAM[x][i]=~OLED_GRAM[x][i];
    OLED_GRAM[x][i]|=n;
    OLED_GRAM[x][i]=~OLED_GRAM[x][i];
}
hi_u32 my_i2c_write(hi_i2c_idx id, hi_u16 device_addr, hi_u32 send_len)
{
    hi_u32 status;
    hi_i2c_data es8311_i2c_data = { 0 };
    es8311_i2c_data.send_buf = g_send_data;
    es8311_i2c_data.send_len = send_len;
    status = hi_i2c_write(id, device_addr, &es8311_i2c_data);
    if (status != HI_ERR_SUCCESS) {
        printf("==== Error: I2C write status = 0x%x! ====r\n", status);
        return status;
    }
    return HI_ERR_SUCCESS;
}
//I2C Write Command
void Write_IIC_Command(unsigned char IIC_Command)
{
    g_send_data[0] = 0x00;
    g_send_data[1] = IIC_Command;
    my_i2c_write(HI_I2C_IDX_0, 0x78, 2);
}
//I2C Write Data

```

```
void Write_IIC_Data(unsigned char IIC_Data)
{
    g_send_data[0] = 0x40;
    g_send_data[1] = IIC_Data;
    my_i2c_write(HI_I2C_IDX_0, 0x78, 2);
}
```

3.2.2 WiFi 模块

实现 WiFi 连接包括寻找可用热点和连接热点,相关代码请扫描二维码获取。

3.2.3 血氧模块



通过 MAX30102 模块读取信息并计算得到心率,MAX30102 FIFO 的深度是 32,每个 buf 是 6 字节(两通道数据,每通道 3 字节)。例如,提取的心率值,是第 3~5 个 buf 组合而成,如图 3-3 所示,相关代码如下:

FIFO(0x04-0x07)

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
FIFO Write Pointer				FIFO_WR_PTR[4:0]					0x04	0x00	R/W
Over Flow Counter				OVF_COUNTER[4:0]					0x05	0x00	R/W
FIFO Read Pointer				FIFO_RD_PTR[4:0]					0x06	0x00	R/W
FIFO Data Register	FIFO_DATA[7:0]								0x07	0x00	R/W

图 3-3 MAX30102 模块端口

```
/**
 * @brief Initialize MAX30102
 * @param redAddr the register address to Read or Writen.
 * @return Returns{@link IOT_SUCCESS} if the operation is successful
 */
void max30102_init(void)
{
    uint8_t max30102_info = 0;
    printf("\r\n come in MAX30102 init\r\n");
    max30102_reset();
    MAX_Read_Data(REG_PART_ID, &max30102_info, 1);
    printf("REG_PART_ID = %d \n", max30102_info); //测试是否输出 ID, I2C 是否正常
    //if(max30102_info == 1) //ID 不确定是多少测试
    //{
    //printf("\r\n MAX30102 init Faild \r\n");
    //return;
    //}
    printf("\r\n MAX30102 init Successful \r\n");
    MAX_Write_Data(REG_INTR_ENABLE_1, 0xc0, 1); //INTR setting
    MAX_Write_Data(REG_INTR_ENABLE_2, 0x00, 1);
    MAX_Write_Data(REG_FIFO_WR_PTR, 0x00, 1); //FIFO_WR_PTR[4:0]
    MAX_Write_Data(REG_OVF_COUNTER, 0x00, 1); //OVF_COUNTER[4:0]
    MAX_Write_Data(REG_FIFO_RD_PTR, 0x00, 1); //FIFO_RD_PTR[4:0]
    MAX_Write_Data(REG_FIFO_CONFIG, 0x0f, 1); //sample avg = 1, fifo rollover=false,
    fifo almost full = 17
}
```

```

    MAX_Write_Data(REG_MODE_CONFIG, 0x03,1); //0x02 for Red only, 0x03 for SpO2 mode
0x07 multimode LED
    MAX_Write_Data(REG_SPO2_CONFIG, 0x27,1); //SPO2_ADC range = 4096nA, SPO2 sample
rate (100 Hz), LED pulseWidth (400uS)
    MAX_Write_Data(REG_LED1_PA, 0x24,1);      //Choose value for ~7mA for LED1
    MAX_Write_Data(REG_LED2_PA, 0x24,1);      //Choose value for ~7mA for LED2
    MAX_Write_Data(REG_PILOT_PA, 0x7f,1);      //Choose value for ~25mA for Pilot LED
}
/**
 * @brief read FIFO data in max30102 FIFO register 0x07
 * @param RED_channel_data
 * @param IR_channel_data
 */
void max30102_FIFO_Read_Data(uint8_t *RED_channel_data, uint8_t *IR_channel_data)
{
    printf("begin to read\n");
    uint8_t buff[6];                          //LSB
    /*组合数据
    uint8_t H,M,L;
    H=buff[0]&0x03;                            //bit17-bit16
    M=buff[1];                                //bit8-bit15
    L=buff[2];                                //bit0-bit7
    *RED_channel_data = (H<<16) | (M<<8) | L; */
    int res;
    res=MAX_Read_Data(REG_FIFO_DATA, &buff,6);
    if(res == IOT_SUCCESS)
    {
        printf("read max30102 success\n");
        *RED_channel_data=((buff[0]<<16) | (buff[1]<<8) | (buff[2]) & 0x03ffff);
                                                //buff[0-2] 组合
        *IR_channel_data=((buff[3]<<16) | (buff[4]<<8) | (buff[5]) & 0x03ffff);
                                                //buff[3-5] 组合
    }
    else{
        printf("read max30102 failed\n");
    }
}

```

3.2.4 OneNET 云平台

本部分包括创建账号、创建产品、添加设备和获取信息。

1. 创建账号

登录网页 <https://open.iot.10086.cn/passport/reg/>，按要求填写注册信息后进行实名认证。

2. 创建产品

进入 Studio 平台后，在全部产品中选择多协议接入，单击“添加产品”按钮，在弹出页面中按照提示填写基本信息。本项目采用 MQTT 协议接入。

3. 添加设备

单击“创建产品”按钮，进入详情页面，单击菜单栏中的设备列表，按照提示添加设备。

4. 获取信息

在代码中，需要获取以下认证信息。

```
#define ONENET_INFO_DEVID "954036868"  
#define ONENET_INFO_AUTH "20220604"  
#define ONENET_INFO_APIKEY "Ee4chljcr7Cv1k2IOJ1ZZHZ36CY="  
#define ONENET_INFO_PROID "524506"  
#define ONENET_MASTER_APIKEY "01tIq=T30=4SiZAdw1VsVXgp7Sg="
```

(1) ONENET_INFO_DEVID 和 ONENET_INFO_AUTH。通过查看设备详情获取 ID 和鉴权信息,如图 3-4 所示。



图 3-4 获取设备 ID 和鉴权信息

(2) INFO_APIKEY, 添加 APIKey, 如图 3-5 所示。



图 3-5 获取 APIKey

(3) INFO_PROID 和 MASTER_APIKEY。获取产品 ID 和 Master-API key,如图 3-6 所示。



图 3-6 获取产品 ID 和 Master-API key

3.2.5 前端模块

本部分包括界面设计和程序逻辑。

1. 界面设计

在 ability_main.xml 文件中将心率图片、当前心率、心率值、问题输入文本框以及输入完毕按钮设计为垂直布局,如图 3-7 所示。

(1) 图片插入。将 heartbeat.jpg 图片保存在 entry/src/main/java/resource/base/media 文件夹下,通过 ability_main.xml 对其进行调用,并设置其 ID、长、宽、上边距、缩放类型等。

(2) 文本显示。通过 Text 组件创建文本内容,并设置其 ID、长、宽、上边距、对齐方式、文本大小等,相关代码如下:

```
<Text
    ohos:id="$+id:text_tem"           //文本 ID 为 text_tem
    ohos:below="$id:text_title"       //文本在 text_title 组件下方
    ohos:height="match_content"       //文本高度
    ohos:width="match_parent"         //文本宽度
    ohos:text_alignment="center"      //对齐方式为中心对齐
    ohos:top_margin="30vp"            //上边距
    ohos:text="1"                     //文本内容
    ohos:text_size="30vp"             //文本大小
/>
```

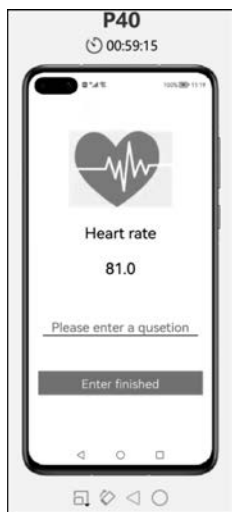


图 3-7 界面布局

(3) 文本输入框。使用 TextField 组件实现文本输入框,TextField 组件与 Text 组件属性类似,独有属性为 basement(输入框基线)以及 Bubble(文本气泡),TextField 组件相关代码如下:

```
<TextField
    ohos:padding="10vp"                //设置内边距
    ohos:text_font="# 000099"          //设置文本字体颜色
    ohos:hint="请输入问题..."        //设置提示文字
    ohos:element_cursor_bubble="$graphic:bubble" //设置气泡 Bubble
    ohos:multiple_lines="true"          //多行显示
    ohos:enabled="true"                //设置可用状态,true 可用,false 不可用
    ohos:basement="#000099"
    ohos:input_enter_key_type="enter_key_type_go" //输入键类型
    ohos:text_input_type="pattern_text" //输入内容类型(数字、密码、文本等)
/>
```

(4) 使用 Button 组件实现一个按键功能,表示文本已输入完毕。后续逻辑设计需要通过该按键的单击事件获取文本输入框中的内容,并设置其 ID、长、宽、按键背景颜色、上边距、对齐方式、文本大小等信息。

2. 程序逻辑

鸿蒙 App 软件部分的核心是与 OneNET 云平台进行交互,分为查询数据流和获取数据流两部分。

(1) 查询数据流详情。

请求方式: GET。

URL 构成: http://api.heclouds.com/devices/device_id/datastreams/datastream_id。

其中,device_id 需要替换为设备 ID,datastream_id 需要替换为数据流 ID,利用 OneNET 云平台提供的 API 调试工具执行请求,观察返回内容,发现其为一个 Json 对象,如表 3-3 和

图 3-8 所示。

表 3-3 返回参数

参 数 名 称	格 式	说 明
errno	int	调用错误码，0 表示调用成功
error	string	错误描述,succ 表示调用成功
data	json	接口调用成功后返回的设备信息
id	string	数据流 ID
create_time	string	数据流创建时间
update_at	string	最新数据上传时间
current_value	string/int/json	最新数据点

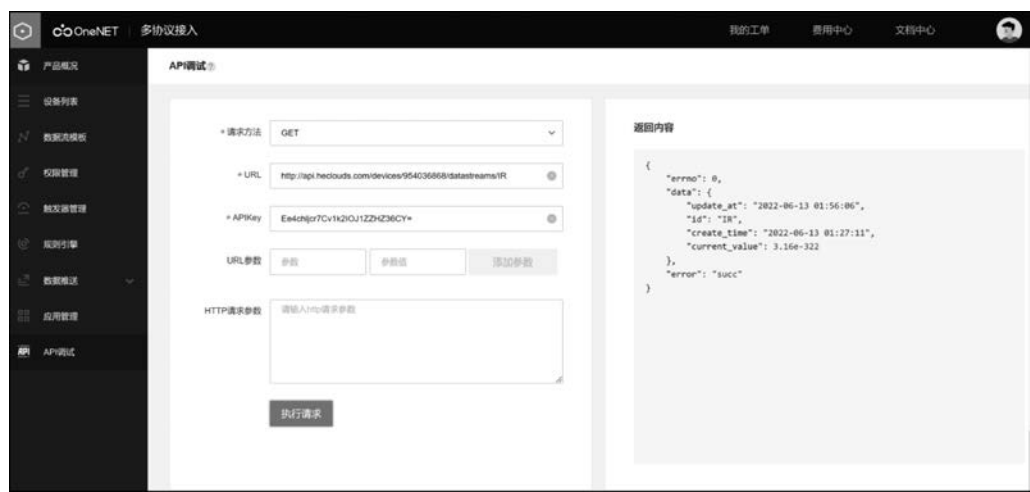


图 3-8 获取数据的 API 调试及返回内容

由查询到的数据流可知,返回内容为 Json 对象,根据其结构,定义一个 JsonBean 类,类的成员为参数 errno、data、error 与 Json 结构对应。

```
public static class JsonBean{
    public int errno;
    public Data data;
    public String error;
    ...
}
```

定义一个内部类 Data 进行解析。

```
static class Data{
    public String update_at;
    public String id;
    public String create_time;
    public float current_value;
    ...
}
```

对每个对象定义 GET 方法,用于在外部获取 JsonBean 内部对象的值。

```
public int getErrno() {
    return errno;
}
public Data getData() {
    return data;
}
public String getError() {
```

```
return error; }
...
```

JsonBean 类定义完成后,通过 Json 库对其进行解析,在鸿蒙操作系统中实现这个操作需要用到外部库,本项目使用 Google 开发的 Gson 库。使用前,需要在 build.gradle 文件中导入依赖,输入命令“implementation group: 'com.google.code.gson', name: 'gson', version: '2.9.0'”按钮后单击“立即同步”按钮。

在发起 GET 请求获取数据的函数中,利用 JsonBean 类和 Gson 获取键值对,从而取得心率等关键信息。

```
JsonBean jsonBean = new Gson().fromJson(result, JsonBean.class);
```

在 App 界面中,实时呈现检测者的心率数据。实现方法是定义线程类 myEventHandler,并通过 sendEvent 方法将数据投递到新线程。

```
myEventHandler.sendEvent(InnerEvent.get(1002, jsonBean));
if(event.eventId==1002) {
    JsonBean jsonBean = (JsonBean) event.object;
    float tem = jsonBean.getData().getCurrent_value(); //获取最新数据点值
    String temText = Float.toString(tem); //将其呈现在界面的 temText 位置
    textTem.setText(temText); }
```

定义 timer 类,重复执行任务,使心率数据自动更新。

```
timer.schedule(new TimerTask() {
    @Override
    public void run() {
        doGet(getUrl); }
},delay:0,period:500); //每 0.5s 执行一次任务后更新界面
```

(2) 获取数据流。

请求方式: POST。

URL: http://api.heclouds.com/cmds? device_id=/device_id。

利用 OneNET 云平台提供的 API 调试工具执行请求,由图 3-9 可以看出,返回内容为“{“errno”: 10, “error”: “device not online: 954036868”}”,表示设备不在线。在此不需要关注返回内容,重点在于下达命令的数据类型。OneNET 云平台可接收 POST 请求的 body 内容为: Json、string 或二进制数据(小于 64KB)。

通过单击事件获取文本内容。在进行界面设计时,应实现按钮组件的实体,对按钮绑定单击事件,使用当前类作为接口的实现类: but.setOnClickListener(this)。

事件绑定完毕后,通过 onClick 函数实现单击按钮并获取文本框中内容的操作:“String message=tf.getText();”。其中,message 为获取的文本框内容,后续下达命令可以直接调用它。

为了验证文本框内容是否成功获取,使用 Toast 弹框将信息显示在按键下方,此部分起验证作用,成功后可以删除。

```
ToastDialog td=new ToastDialog(context:this); //补充上下文信息,为当前页面
td.setTransparent(true); //设置 Toast 的背景
td.setAlignment(Layout.Alignment.BOTTOM); //位置(默认居中)
td.setOffset(0,200); //设置一个偏移
td.setText(message); //设置 Toast 的内容为获取的文本框信息
```

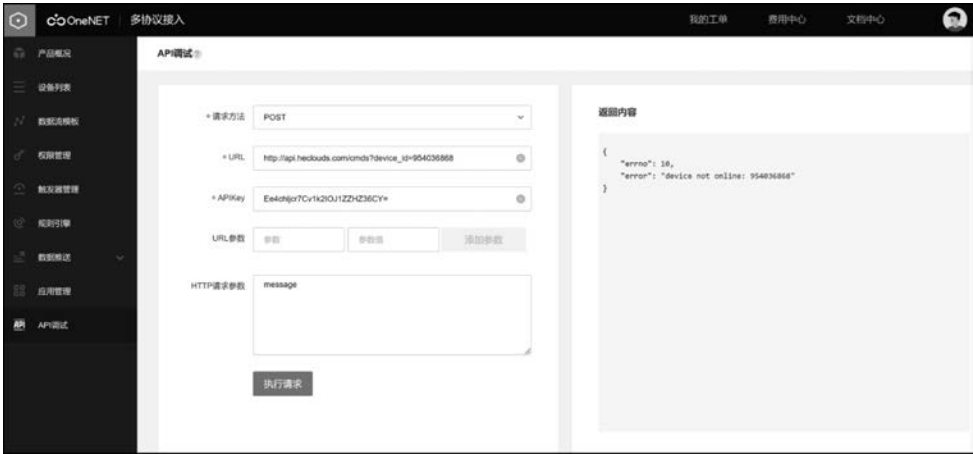


图 3-9 下达命令的 API 调试及返回内容

弹框出现于按键下方,应正确显示输入的文本信息,如图 3-10 所示。

为防阻塞主线程启用新线程发起 POST 请求,实时呈现 OLED 界面。在 dopost 函数中,将 string 类型的命令放入 POST 请求的 body 中访问 API。

```
postConnection.setRequestProperty ( " Content - Type ",
application/octet-stream");
postConnection.setRequestProperty("api-key", api_key);
new Thread(new Runnable() {
    @Override
    public void run() {
        doPost(postUrl, message, 1102);
    }
}).start();
```

首先,单击“输入完毕”按钮后,下达发送 message 的命令,提问者所输入的问题将显示在 OLED 屏幕上;然后,借助 MyEventHandler 类更新界面,将当前的屏幕显示同步到页面中。



图 3-10 文本信息

3.3 成果展示

计算机屏幕左侧为 OneNET 云平台的数据波动,右侧为 App 端的界面,下方为 Hi3861 开发板实时刷新的展示效果,如图 3-11 所示;单击按键后的 OLED 显示相应数据,如图 3-12 所示。



图 3-11 开发板实时刷新结果

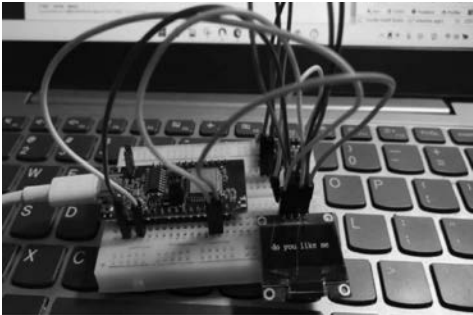


图 3-12 单击按键后的 OLED

3.4 元件清单

完成本项目所需的元件及数量如表 3-4 所示。

表 3-4 元件清单

元件/测试仪表	数 量	元件/测试仪表	数 量
面包板	1 个	OLED 显示屏	1 个
Hi3861	1 个	MAX30102 血氧模块	1 个