华南理工大学

South China University of Technology

课程设计报告书

IWatch 智能健康手环设计

教师评语	教师签名： 日期：
成绩评定	
备注	

目录

IWatch 智能健康手环设计

1 选题背景

1.1 设计背景

智能手环是一种穿戴式智能设备，可以帮助用户记录日常生活中的锻炼、睡眠、饮食等实时数据，并将这些数据与手机、平板同步，起到通过数据指导健康生活的作用。

手环一般采用医用橡胶材质，天然无毒，外观设计高档时尚、大方，有流线花环，颜色多样。当下我国智能手环具有能耗低、续航久、个头小功能强等特点。具体来看，智能手环内置低功耗蓝牙模块，同时内置了一颗锂聚合物电池，续航时间可达 10 天，而且智能手环作为一款兴起未多久的高科技可穿戴式智能设备，其功能愈发齐全。在早期有些智能手环实际上只有计步与显示时间等功能，并无健康监测功能，现如今智能手环一般具有健康监测（心电监测、睡眠监测、热量监测）以及计步器、显示时间、蓝牙、久坐提醒等功能，有些则还具有测量距离、计算消耗的卡路里和脂肪、天气预报、移动支付、NFC、测量脉搏、心率、皮肤温度，以及其它环境信息，如光照及环境噪音水平等功能。

而现阶段我国的人民生活质量在不断提高的同时，生活压力也不断增加，健康状况差已成为许多人关注的问题。人们对于包括血氧饱和度、心率、步数在内的健康指标的需求进一步提高。随着传感器术、微型化集成技术以及无线传输技术的发展，小型化、低成本的无线健康监护系统相继被应用到老年人、慢性病患者等群体的日常监护，成为新型的监护模式。

1.2 设计目标

本次课程设计希望能设计一款集成心率检测、血氧饱和度检测、步数统计与物联网功能的智能健康手环，可以基本满足一般群体的需求。鉴于开发时间比较短，此次课设主要把目光聚集在了如何对采集到的心率、血氧饱和度、步数数据的滤波上，并附加了通过 esp8266 模块将数据上传至阿里物联网平台的功能，最后基于该平台开发了一个 APP 界面，初步实现了预期功能。

2.方案论证

2.1 总体方案论证

设计时考虑到智能手表与手机交互、数据处理，提出下述两种方案：第一种使用 WIFI 模块连接阿里云 IOT 平台再传输给其它设备，第二中使用蓝牙模块直接将数据传

输手机再转发至电脑端。

方案一：阿里云 IOT 平台，使用 WIFI 模块传输数据。手机 APP 和电脑端调用 IOT 平台的 API 端口获得数据，电脑端将数据处理后的结果返回至手机。方案二：蓝牙调试助手将数据传至电脑端，以实现数据的交互。

经过对比选择，选择使用 WIFI 模块。首先，因为电脑端使用物联网平台对获取得到数据处理比较容易，而使用蓝牙模块则使得电脑端对数据处理比较困难。其次，物联网平台可以通过网络将数据传输至不同主机上，可以有效实现数据的交互功能。最后，云服务器的算力远远超过单台主机，可以对上传的数据实现复杂方式的处理。

所以，最终选择方案 1 作为而本次课设的总体框架。

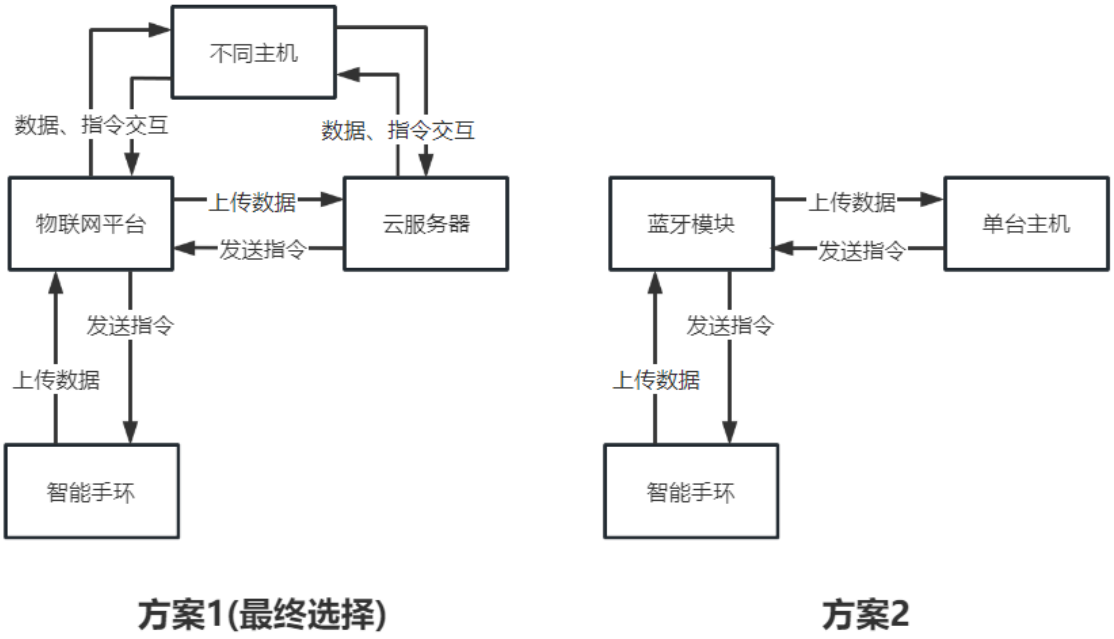


图 2.1 两方案总体框架图

2.2 芯片选择论证

单片机（Microcontrollers）是一种集成电路芯片，是采用超大规模集成电路技术把具有数据处理能力的中央处理器 CPU、随机存储器 RAM、只读存储器 ROM、多种 I/O 口和中断系统、定时器/计数器等功能（可能还包括显示驱动电路、脉宽调制电路、模拟多路转换器、A/D 转换器等电路）集成到一块硅片上构成的一个小而完善的微型计算机系统。由于 8 位单片机由于内部构造简单，体积小，成本低廉，在一些较简单的控制器中应用很广。常见的 8 位单片机主要有：Intel 的 51 系列，Atmel 的 AVR 系统，Microchip 公司的 PIC 系列，TI 的 MSP430 系列等。而 STM32 是一种功能比较强大的 32 位的单片机。它和 8 位单片机最大的不同是，它不仅可以使用寄存器进行编程，还

可以使用官方提供的库文件进行编程，这样不仅编程方便，而且更容易移植。

STM32 系列 32 位 Flash 微控制器基于 ARM Cortex™-M 处理器，旨在为 MCU 用户提供新的开发自由度。它包括一系列 32 位产品，集高性能、实时功能、数字信号处理、低功耗与低电压操作等特性于一身，同时还保持了集成度高和易于开发的特点。而 STM32 F1 系列主流 MCU 满足了工业、医疗和消费类市场的各种应用需求。凭借该产品系列，意法半导体在全球 ARM Cortex-M 微控制器领域处于领先地位，同时树立了嵌入式应用的里程碑。该系列利用一流的外设和低功耗、低压操作实现了高性能，同时还以可接受的价格、利用简单的架构和简便易用的工具实现了高集成度。

其中，STM32F103 器件采用 Cortex-M3 内核，CPU 最高速度达 72 MHz。该产品系列具有 16KB~ 1MB Flash、多种控制外设、USB 全速接口和 CAN。STM32F103C8T6 基于 ARM32 位 Cortex™-M3 内核，电压使用范围是 2.0V ~ 3.6V，工作频率最高可以达到 72MHz，内部采用 64K 或 128K 字节 Flash 程序存储器，以及高达 20K 字节的 SRAM 数据存储器；内置 CRC 循环冗余校验以及 96 位编码（24 位的十六进制数）的芯片唯一序列号。它的主系统由 4 个控制单元（DCode 总线 D-bus、系统总线 S-bus、通用 DMA1、通用 DMA2）以及 4 个受控单元（内部 SRAM、内部 Flash、FSMC、AHB 到 APB 的桥 AHB2APBx）组成，它们通过一个多级的 AHB 总线相互进行连接。

综上，我们选择 STM32F103C8T6 作为本次设计的主控芯片，既满足了低成本的经济要求，还能够满足我们需要的性能要求，并且拥有低功耗的优良特点。我们采用模块化硬件的方法，电路板只需绘制不同元件的连接次序，直接令 STM32F103C8T6 核心板插入到电路板中，方便我们进行开发与调试。

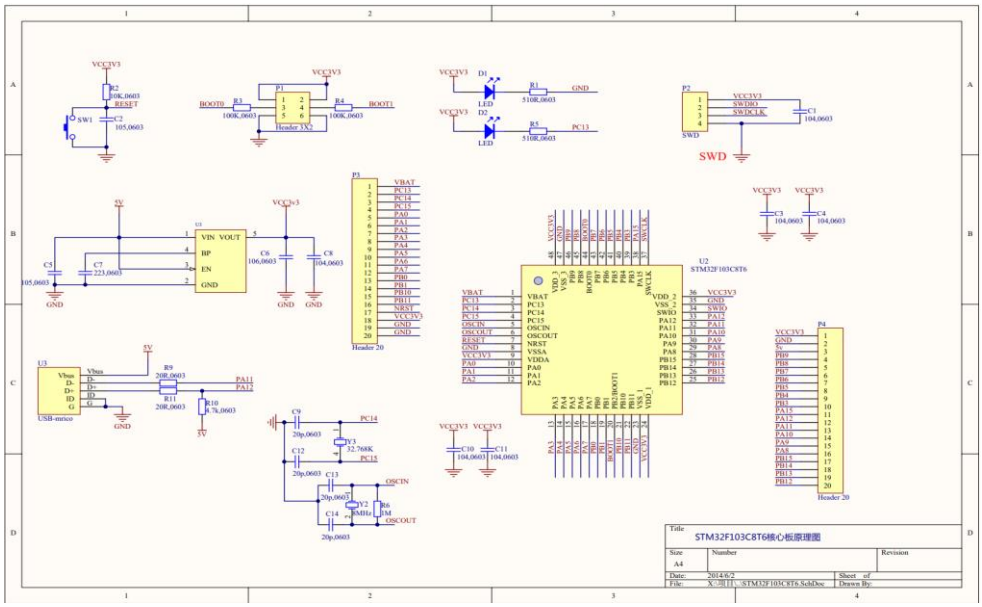


图 2.2 STM32F103C8T6 核心板原理图

3 硬件设计

3.1 显示器件选择

方案一：采用 LED 数码管动态扫描显示。 LED 数码管的价格适中，对于显示数字或者简单的字母会比较合适。但是采用动态扫描法与单片机连接时占用 CPU 的 I /0 口较多，并且由于单片机的 IO 口输出电流不够，所以需要一个驱动电路，通过驱动电路放大电流后控制数码管，还有就是采用数码管进行显示的话显示的内容多了对于电路的焊接机会增大难得容易焊接错误。

方案二：采用 LCD 液晶显示屏。液晶显示功能强大，可以同时显示出 16*2 即 32 个字符，可包括数字、字母、符号、或者自定义字符。LCD 液晶显示器中的每一个字符都是由 5*7 的点阵组成。LCD 采用并行数据传输也可以采用串行数据传输。

方案三：采用 OLED 显示屏。OLED ，即有机发光二极管（ Organic Light Emitting Diode)。 OLED 由于同时具备自发光，不需背光源、对比度高、厚度薄、视角广、反应速度快、可用于挠曲性面板、使用温度范围广、构造及制程较简单等优异之特性，被认为是下一代的平面显示器新兴应用技术。 LCD 都需要背光，而 OLED 不需要，因为它是自发光的。以目前的技术， OLED 的尺寸还难以大型化，但是分辨率确可以做到很高。 OLED 的分辨率为 128*64 它有多种接口方式： OLED 裸屏总共种接口包括：6800、8080 两种并行接口方式、3 线或 4 线的串行 SPI 接口方式、I2C 接口方式（只需要 2 根线就可以控制 OLED)，这五种接口是通过屏上的 BSO ~BS2 来配置的。

综合上述，为了压缩成本并获得更佳的显示效果，本次课程设计选用方案 3，即使用 0.96 英寸的 OLED 模块，它共有 4 个引脚，接口使用 IIC 协议。各引脚说明如下面表格所示。其接口信号如表所示，其实物图与在电路板中的排母原理图如图。

表 3.1 0.96 寸 OLED 屏幕引脚说明

编号	符号	引脚说明
1	GND	电源地引脚
2	VCC	电源正引脚(3~5.5V)
3	SCL	I2C 的时钟引脚
4	SDA	I2C 的数据引脚

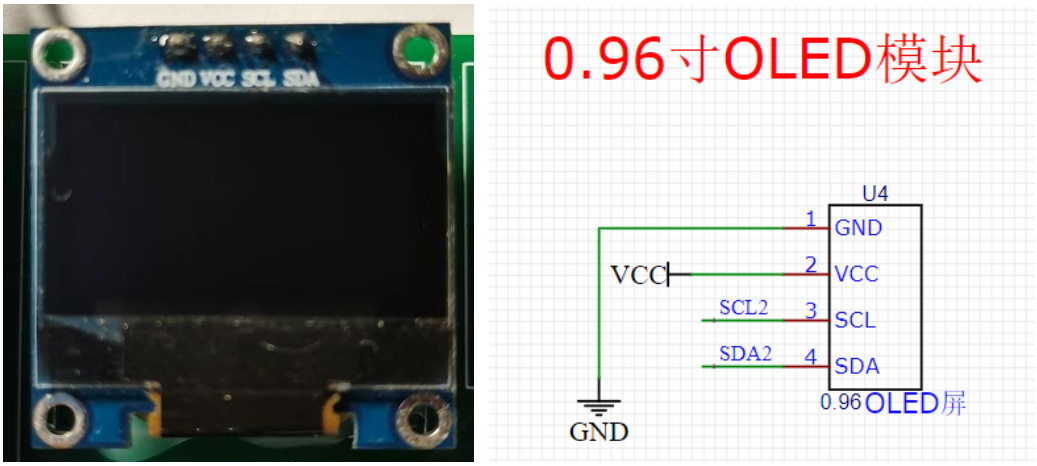


图 3.2 0.96 寸 OLED 屏幕实物图与排母原理图

3.2 MAX30102 血氧模块

MAX30102 是一个集成的脉搏血氧仪和心率监测仪生物传感器的模块（芯片）。它集成了一个 660nm 红光 LED、880nm 红外光 LED、光电检测器、光器件，以及带环境光抑制的低噪声电子电路。可通过软件关断模块，待机电流为零，实现电源始终维持供电状态,可运用于低功耗产品中。

MAX30102 采用一个 1.8V 电源和一个独立的 3.3V 用于内部 LED 的电源，标准的 I2C 兼容的通信接口。市面很多都将 MAX30102 芯片集成在一个 PCB 模块上，内部增加一个 1.8V 和 3.3V LDO 稳压电路，可对模块单独供 5.0V 电源，方便开发者进行开发。

本次设计选择的血氧采集与心电信号采集模块是 MAX30102 血氧模块，其模块内部电路图如图所示。

表 3.3 MAX30102 血氧模块引脚说明

编号	符号	功能
1, 7, 8, 14	N.C.	没有连接。连接 PCB 板以实现机械稳定性
2	SCL	I2C 时钟输入
3	SDA	I2C 数据双向
4	PGND	LED 驱动器块的电源接地。
5	R_DRV	红色 LED 驱动程序。
6	IR_DRV	IR LED 驱动程序。
9	VLED+	LED 电源（阳极连接）。使用旁路电容器到 PGND 以获得最佳效果
10	VDD	模拟电源输入。使用旁路电容器接地以获得最佳性能。
11	GND	模拟电源地

12	INT	有源低电平中断（漏极开路）。使用上拉电阻器连接到外部电压。

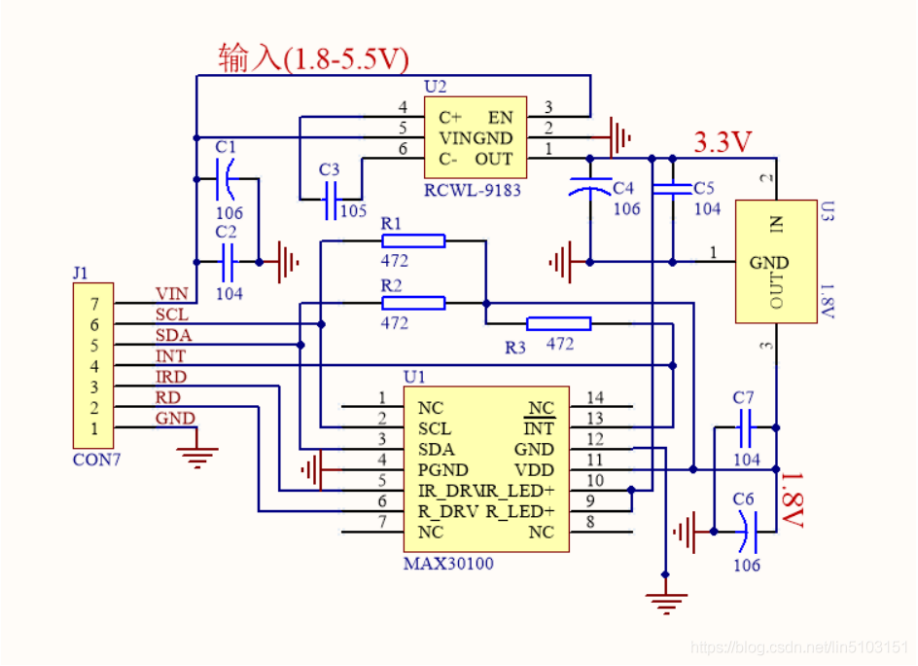


图 3.4 MAX30102 血氧模块内部电路图

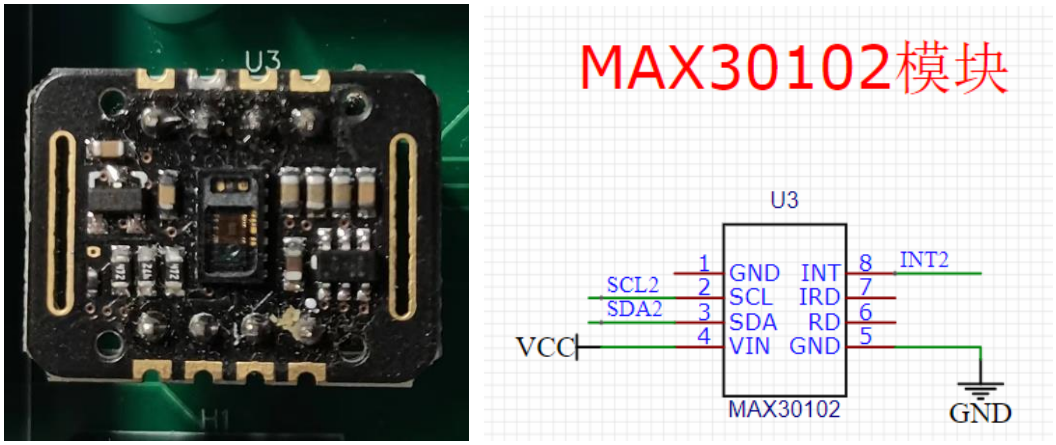


图 35 MAX30102 血氧模块实物图与排母原理图

3.3 MPU9250 9 轴姿态传感器

MPU9250 是一个 QFN 封装的复合芯片，由 2 部分组成。3 轴的陀螺仪、加速度与磁力计，芯片内置 16bit AD 转换器,16 位数据输出；支持 I2C 与 SPI 协议。陀螺仪范围：±250 500 1000 2000°/s；加速度范围：±2 ±4 ±8 ±16g；磁场范围：±4800uT；I2C 通信速度 400Khz，SPI 通信速度最高可达 1Mhz。可广泛应用于航模无人机，机器人，VR 等领域。

由于我们需要做到步数统计的功能，所以需要用到速度、加速度等数据的信息，所以选择了这款市面上最为常见、可靠性高的模块。

表 3.6 MAX30102 血氧模块引脚说明

编号	符号	功能
1	VCC	3-5V
2	GND	地
3	SCL	I2C 串行时钟线/SPI 串行时钟端口
4	SDA	I2C 串行数据线/SPI 串行数据输入
5	EDA	连接其他 I2C 设备主机数据口
6	ECL	给 I2C 设备提供时钟
7	AD0/SO0	I2C 器件地址选择位/SPI 串行数据输出
8	INT	中断引脚
9	NCS	片选
10	FSYNC	数字同步接入帧，不用时接地

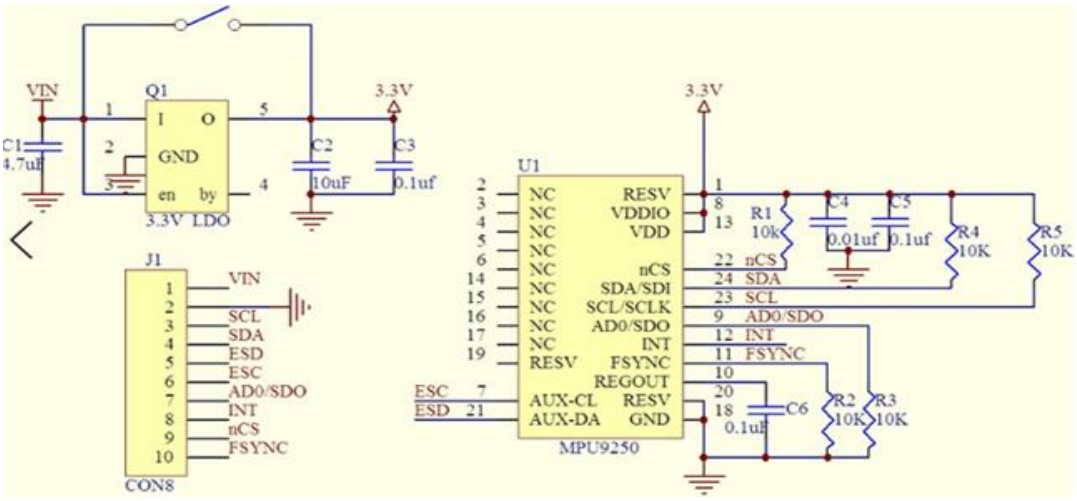


图 3.7 MPU9250 9 轴姿态传感器内部电路图

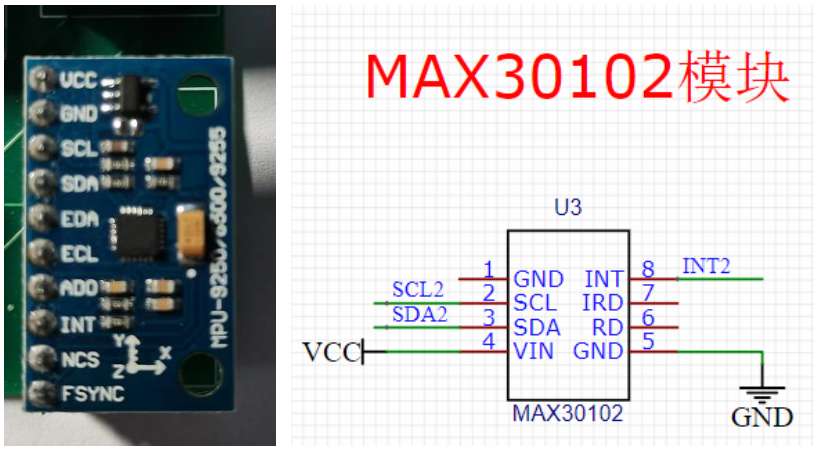


图 3.8 MAX30102 血氧模块实物图与排母原理图

3.4 ESP8266 WIFI 模块

ESP8266 是一个完整且自成体系的 WiFi 网络解决方案，能够独立运行， 也可以作为 slave 搭载于其他 Host 运行。

首先在搭载应用并作为设备中唯一的应用处理器时，其能够直接从外接闪存中启动。内置的高速缓冲存储器有利于提高系统性能，并减少内存需求。另外一种情况是，无线上网接入承担 WiFi 适配器的任务时，可以将其添加到任何基于微控制器的设计中，连接简单易行，只需通过 SPI/SDIO 接口或中央处理器 AHB 桥接口即可。

其次，ESP8266 强大的片上处理和存储能力，使其可通过 GPIO 口集成传感器及其他应用的特定设备，实现了最低前期的开发和运行中最少地占用系统资源。ESP8266 高度片内集成，包括天线开关 balun、电源管理转换器，因此仅需极少的外部电路，且包括前端模块在内的整个解决方案在设计时将所占 PCB 空间降到最低。

最后，装有 ESP8266 的系统表现出来的领先特征有：节能 VoIP 在睡眠/唤醒模式之间的快速切换、配合低功率操作的自适应无线电偏置、前端信号的处理功能、故障排除和无线电系统共存特性为消除蜂窝/蓝牙/DDR/LVDS/LCD 干扰。

所以，使用 ESP8266 模块作为 WIFI 模块使得物联网功能的实现变得简单可行，故作此选择。

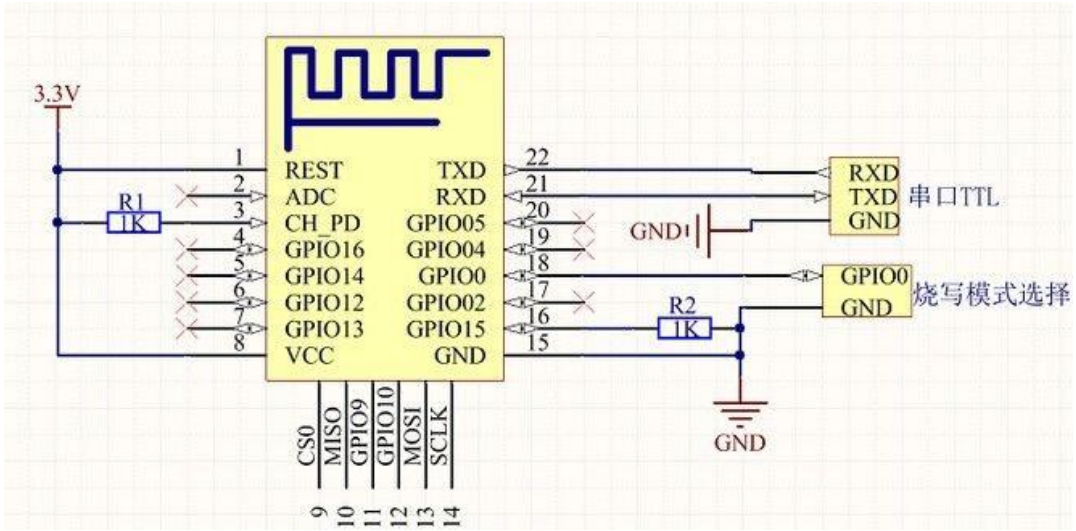


图 3.9 ESP8266 最小系统板原理图

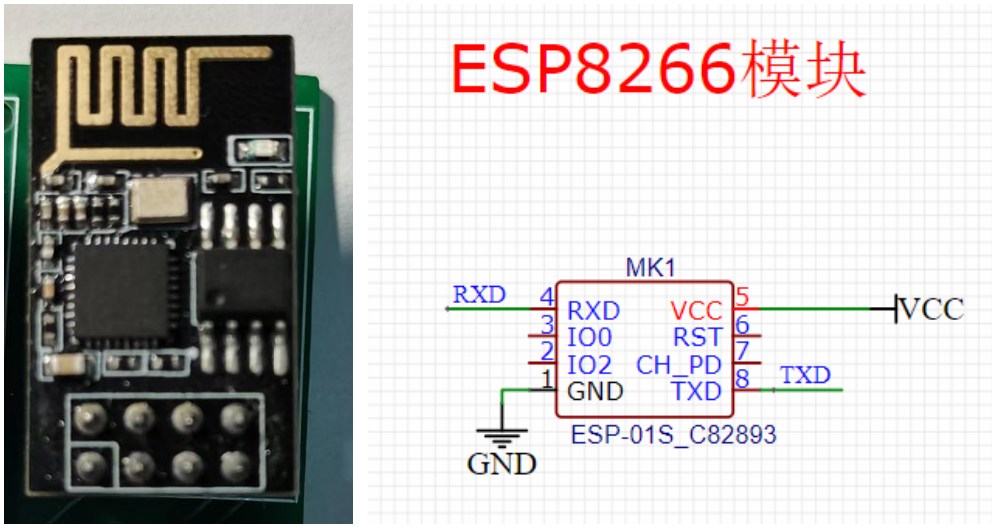


图 3.10 ESP8266 WIFI 模块实物图与排母原理图

3.5 整体电路设计

通过上述方案比较，我们最终选择了 STM32F103C8T6 作为本次设计的主控芯片，通过 MAX30102 血氧模块实现对血氧浓度、心率信息的获取，同时通过 MPU9250 9 轴姿态传感器实现对各方向速度、加速度的获取并最终得到步数信息，最后通过 ESP8266 WIFI 模块实现与阿里云 IOT 平台的数据交互，在手机端使用 APP 完成上述信息的获取与展示。整体电路设计框图、电路原理图与 PCB 板图如图所示。

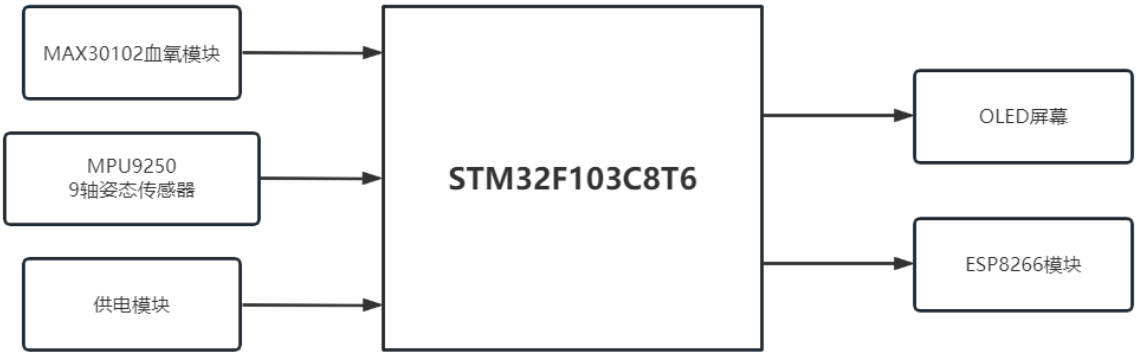


图 3.11 整体电路设计框图

stm32 引脚		元件	
USART1	PA9	Esp8366-01s	RX
	PA10		TX
	3.3v		3.3v
	GND		GND
USART2	PA2	HC05	RX
	PA3		TX
	3.3v		VCC
	GND		GND
	5V		EN
IIC	PB6	MAX30102	SCL
	PB7		SDA
	PB9		INT
	3.3v		VIN
	GND		GND
IIC	PB6	OLED 屏幕	SCL
	PB7		SDA
	3.3v		VCC
	GND		GND
IIC	PB10	MPU9250	SCL
	PB11		SDA
	PB5		INT
	3.3v		VCC
	GND		GND

图 3.12 整体引脚分配图

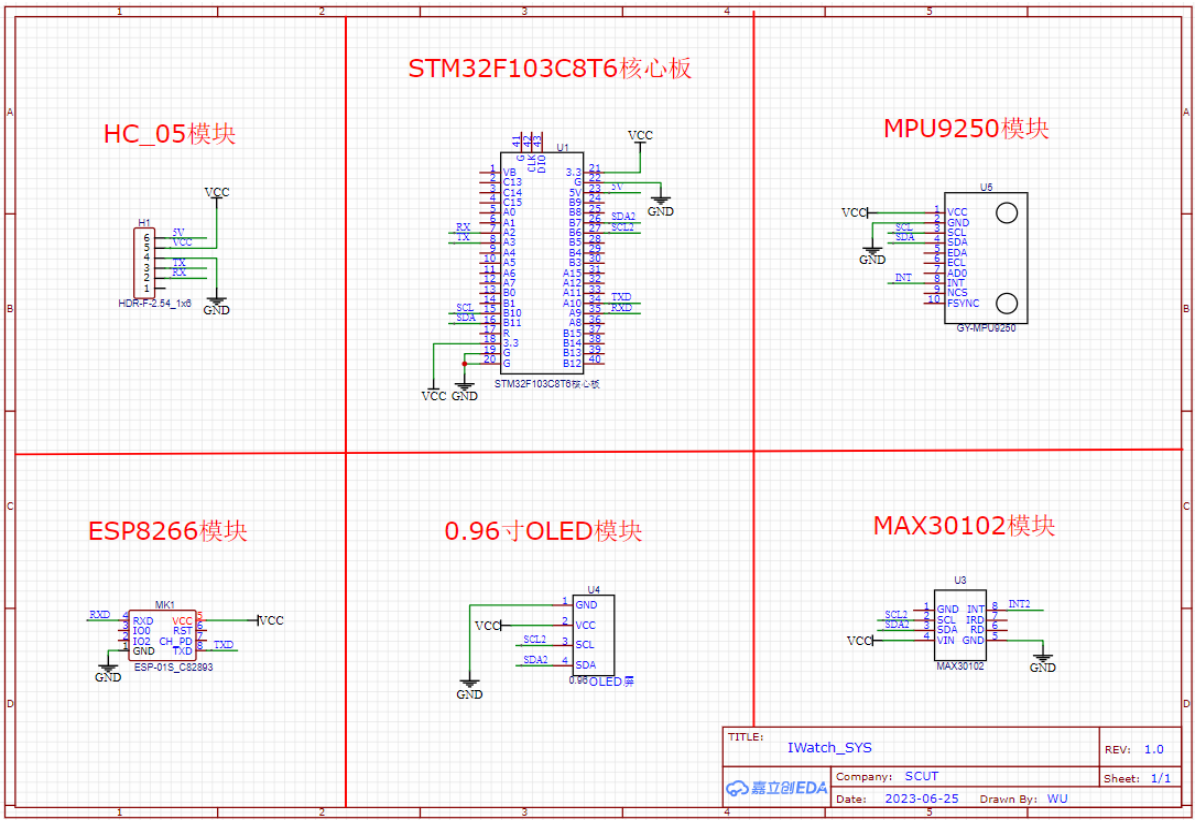


图 3.13 整体电路原理图

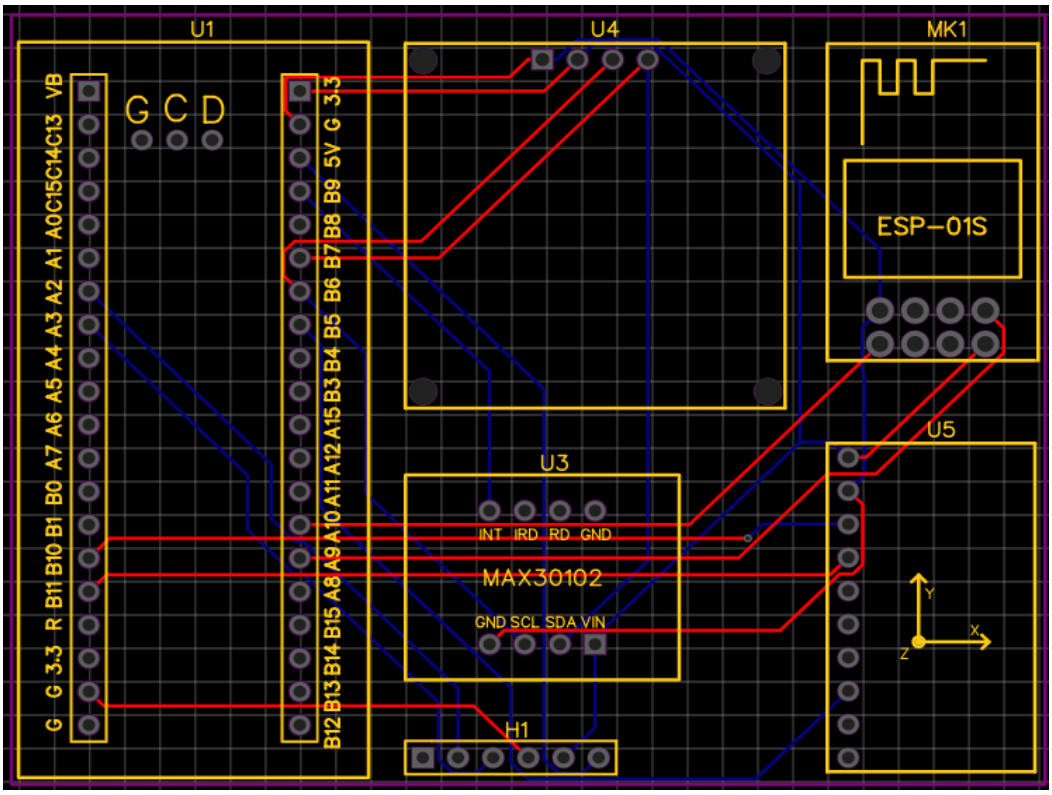


图 3.14 整体电路 PCB 板图

4 软件设计

4.1 总体程序流程图

系统上电后，首先进行系统与外设初始化，包括：软件 I2C 初始化、RTC 时钟初始化、OLED 初始化、MAX30102 初始化、ESP8266 初始化以及定时器初始化，然后判断初始化是否成功从而选择是否进入主循环。主要采用中断方式进行信息的采集、运算、数据上传以及屏幕的刷新。总体流程图如图所示。

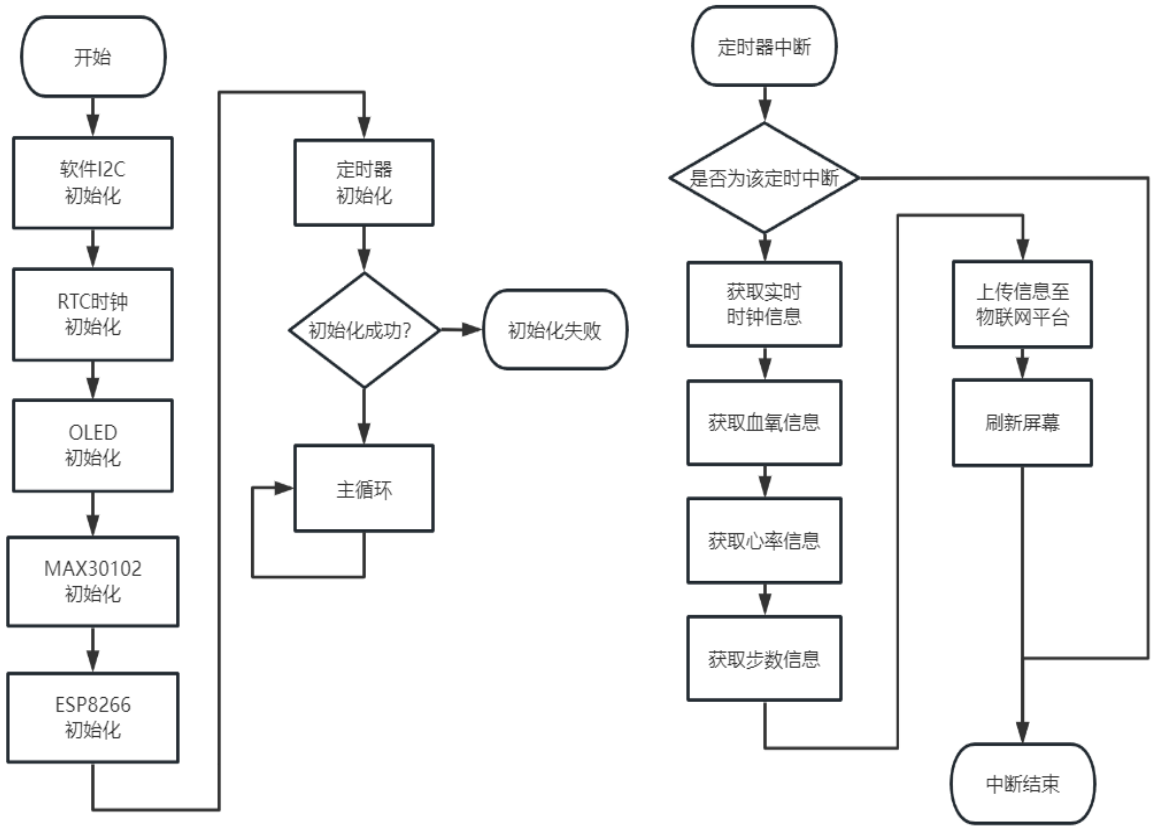


图 4.1 总体程序流程图

4.2 ESP8266 WIFI 模块联网程序流程图

首先，创建阿里云 IOT 平台账号，并且将阿里云账号的 clientId、username、mqttHostUrl、passwd、port 等信息加入 ESP8266 初始化代码中。然后，进行 ESP8266 的初始化配置，关闭其回显功能，开启 STA 模式并使其与 WIFI 进行连接。连接上 WIFI 后，配置 MQTT 用户信息，并通过先前填写的信息连接至阿里云 IOT 平台，等进入定时器中断后，上传信息至平台，至此完成 WIFI 模块的基本功能。

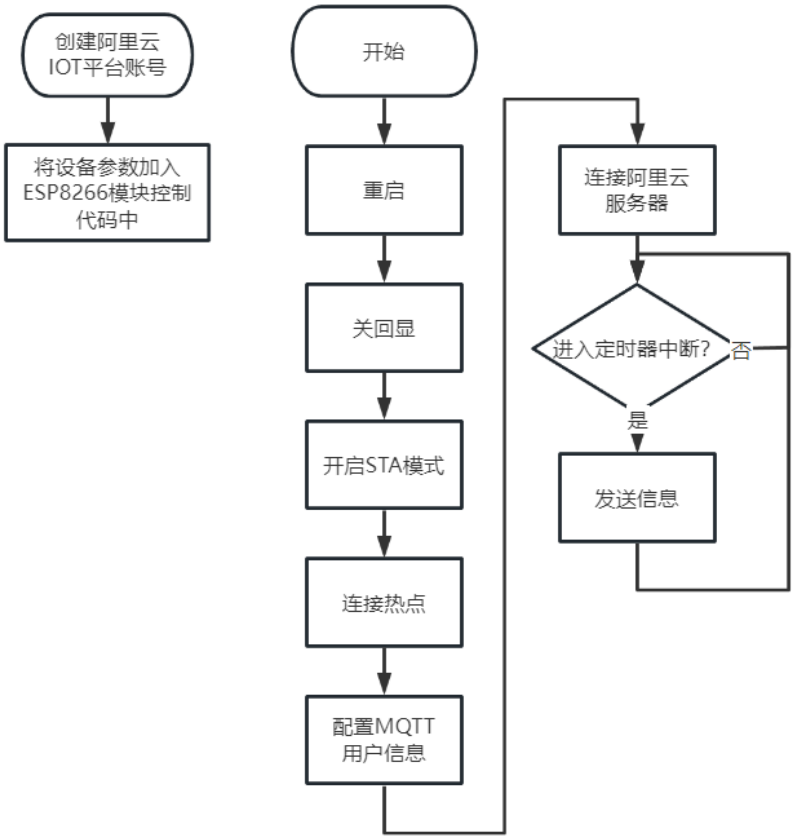


图 4.2 ESP8266 WIFI 模块联网程序流程图

4.3 血氧与心率采集程序流程图

首先，对 MAX30102 血氧模块采集到的数据进行获取，通过使能 g_fft_index，当 g_fft_index<FFT_N(FFT 变换所需点数)时，持续从 MAX30102 血氧模块的 FIFO 队列中获取得到心率信息以及血氧信息，并将其分别存入 s1、s2 数组中进行存储。

然后，当数据采集完成后，我们进行血液信息的转换。我们先进行了直流滤波，将数组中的数据减去均值，并将其直流成分保留在 dc_red、dc_ir 中存储。接着，我们将进行移动平均滤波与八点平均滤波，分别滤除了高频分量与周期性干扰。最后，我们使用 FFT(快速傅里叶变换)得到了频率域的数据，转换完成后存储在 s1、s2 数组中的数据就变成了对应每一个频率点上的数据。在我们通过对频率域中的数据进行下列三式子中的运算，即可得到相应的心率、血氧饱和度信息。

最后将数据上传至阿里云 IOT 平台，并对屏幕进行刷新。

$$HeartRate = \frac{60 \times X \times f_{\max}}{N_{FFT}}$$

$$R = \frac{AC_{red} / DC_{red}}{AC_{ired} / DC_{ired}}$$

$$SpO_2 = -45.060 \times R \times R + 30.354 \times R + 94.845$$

其中，X 代表每秒的采样次数；f_{max} 代表 s1 数组中对应的最大频率；N_{FFT} 代表快速傅里叶变换的采样点数；AC_{red} 代表红光的交流分量；DC_{red} 代表红光的直流分量；AC_{ired} 代表红外的交流分量；DC_{ired} 代表红外光的直流分量。

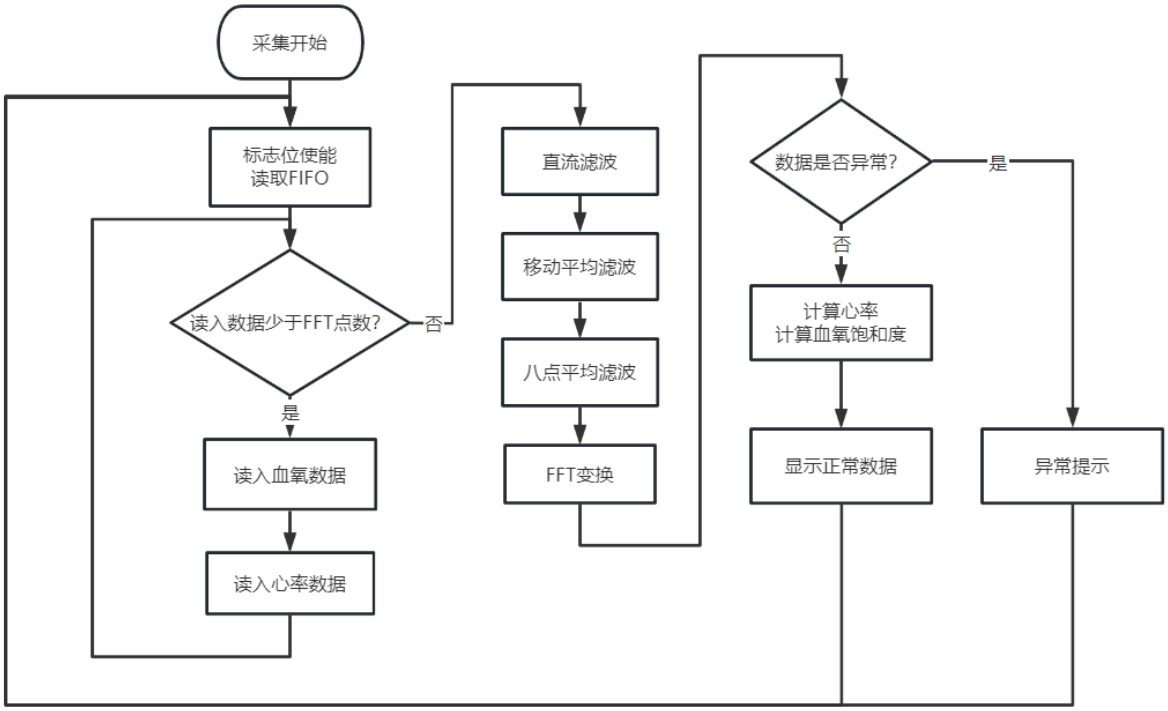


图 4.3 血氧与心率采集程序流程图

4.4 步数采集流程图

步数统计原理：在行走过程中，腿部垂直于人体矢状面方向的角速度变化最为明显，且具有一定的规律性，可以通过这一规律性来判断步数。角速度在行走过程之中的大致变化曲线如图所示。 其中摆动中相就是走路过程中两腿并排的那一刻，此时角速度最大。由图可以看出在每一步中，角速度有一个骤增和一个骤减的过程最具有标志性，若是能够通过算法捕获到这个现象，就可以实现计步功能了。

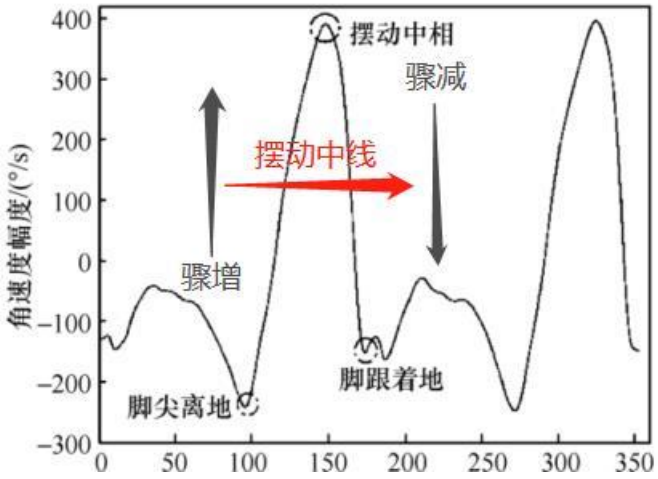


图 4.4 行走过程中角速度变换曲线

程序上，首先，进行均值滤波采样，当变换范围在可信赖变化量内，则记录 MAX 和 MIN，因为单次采样数据必然不可信，所以要多次采样取平均值。同时，取完平均值后的数据依然不一定是可信的数据，如果本次采样的数据和上一个采样的数据差值过小或者过大，应当不予采纳，需要设置可信赖变化量的下限和上限。

然后，我们保存上一次采样数据 → 均值采样本次数据 → 计算本次和上次的差值 → 检验差值大小(若超限则将本次数据回退到上一次的大小) → 保存最大值和最小值 。这个函数固定时间会被调用一次。

接着，进行最活跃轴的判断，也就是 MPU9250 哪个轴是更接近垂直于人体矢状面，我们隔一段时间计算每个轴角速度上一次和这一次原始数据的差值，然后比较这三个差值的大小，增加最大差值轴的活跃度权重。比较一次权重值，权重最大的轴就是最活跃轴，然后把权重都清零，下一个 1.5 秒再重新判断一次。

最后，就是捕获原始数据骤增和骤减现象了。取最大值和最小值的均值 mid，每当出现图中的情况，即先后出现骤增与骤减，则算作一次有效走步。把 step 数据++，并上传至阿里云 IOT 平台，对屏幕进行刷新。

本次项目中步数采集过程主要使用 MPU6050 官方提供的 DMP 库，滤波效果良好，基本满足设计需求。

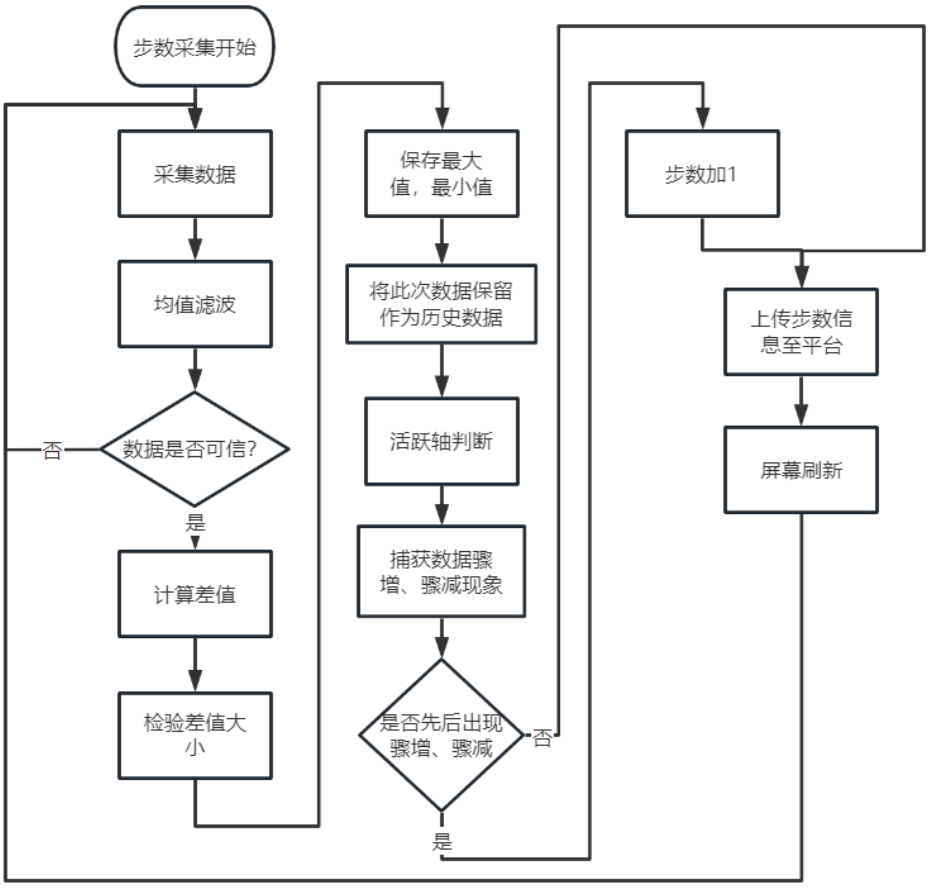

步数采集开始
采集数据
均值滤波
数据是否可信?
否
是
计算差值
检验差值大小
保存最大值，最小值
将此次数据保留作为历史数据
活跃轴判断
捕获数据骤增、骤减现象
是否先后出现骤增、骤减
否
是
步数加1
上传步数信息至平台
屏幕刷新

图 4.5 步数采集流程图

5 结果分析

5.1 心率采集与血氧饱和度采集分析

上电后，系统心率采集与血氧饱和度采集正常，屏幕正常显示。

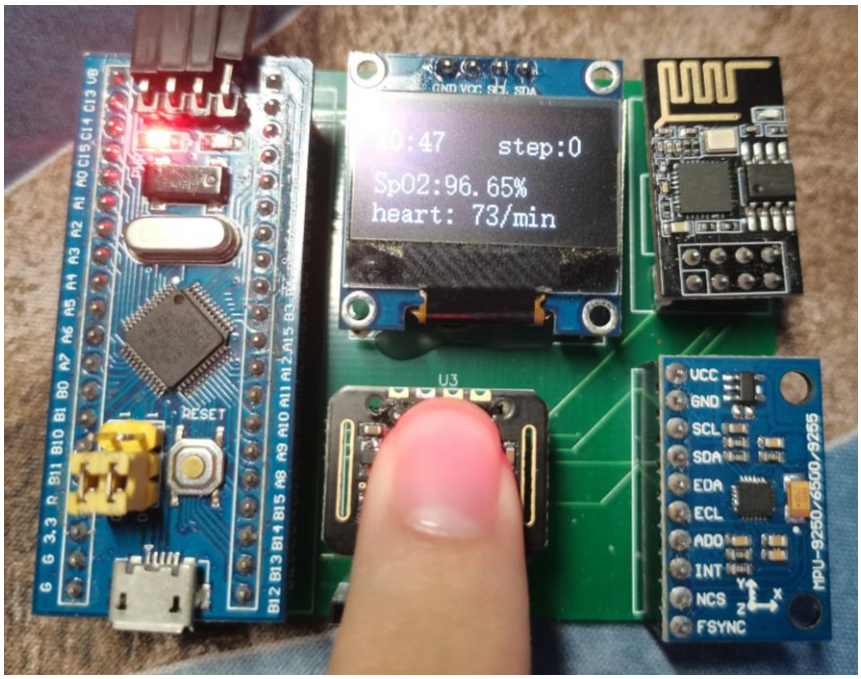


图 5.1 心率采集与血氧饱和度采集实物图

通过一段时间的测试，我们发现数据采集正常，且中途松手了两次发现系统均检测出了异常并把相关数据置零。在阿里物联网平台观测相关数据如下图。

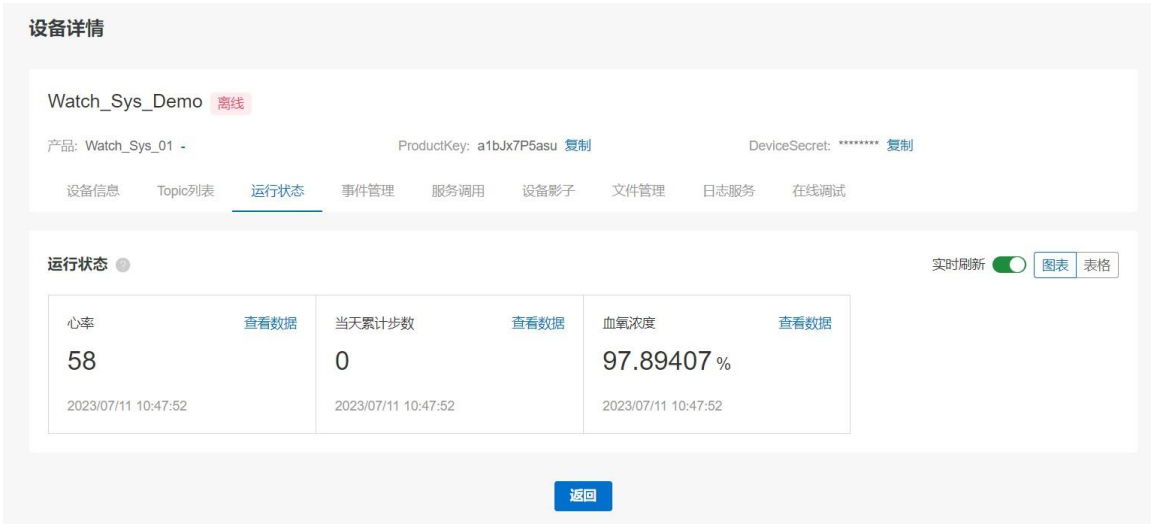


图 5.2 阿里物联网平台观测相关数据

5.2 步数采集分析

上电后，佩戴设备匀速走了一段路，发现步数显示正常。

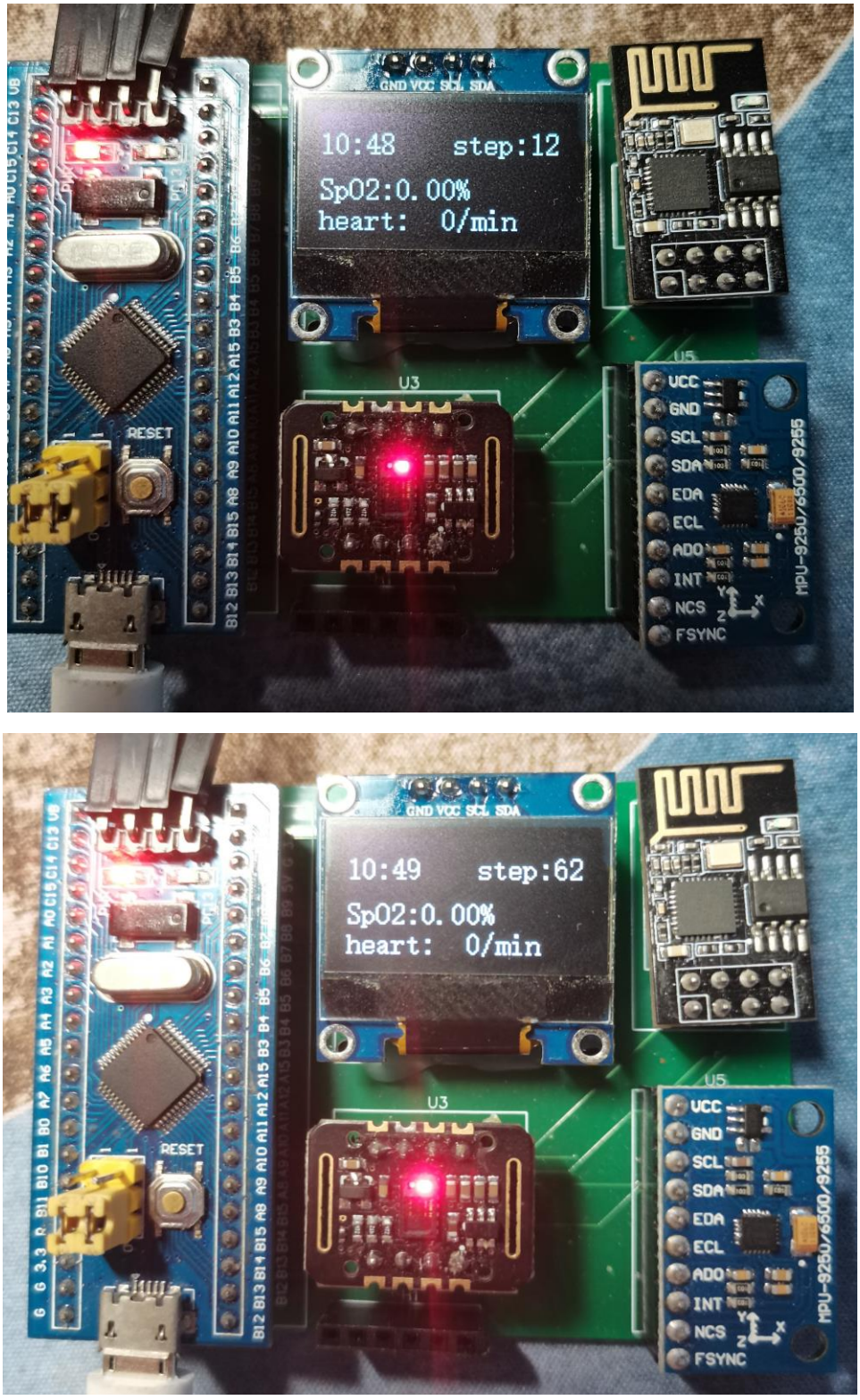


图 5.3 步数采集实物图

佩戴匀速行走一段时间后，阿里物联网平台观测相关数据如下图(前面恒等于 0 是因为在测试心率数据)。

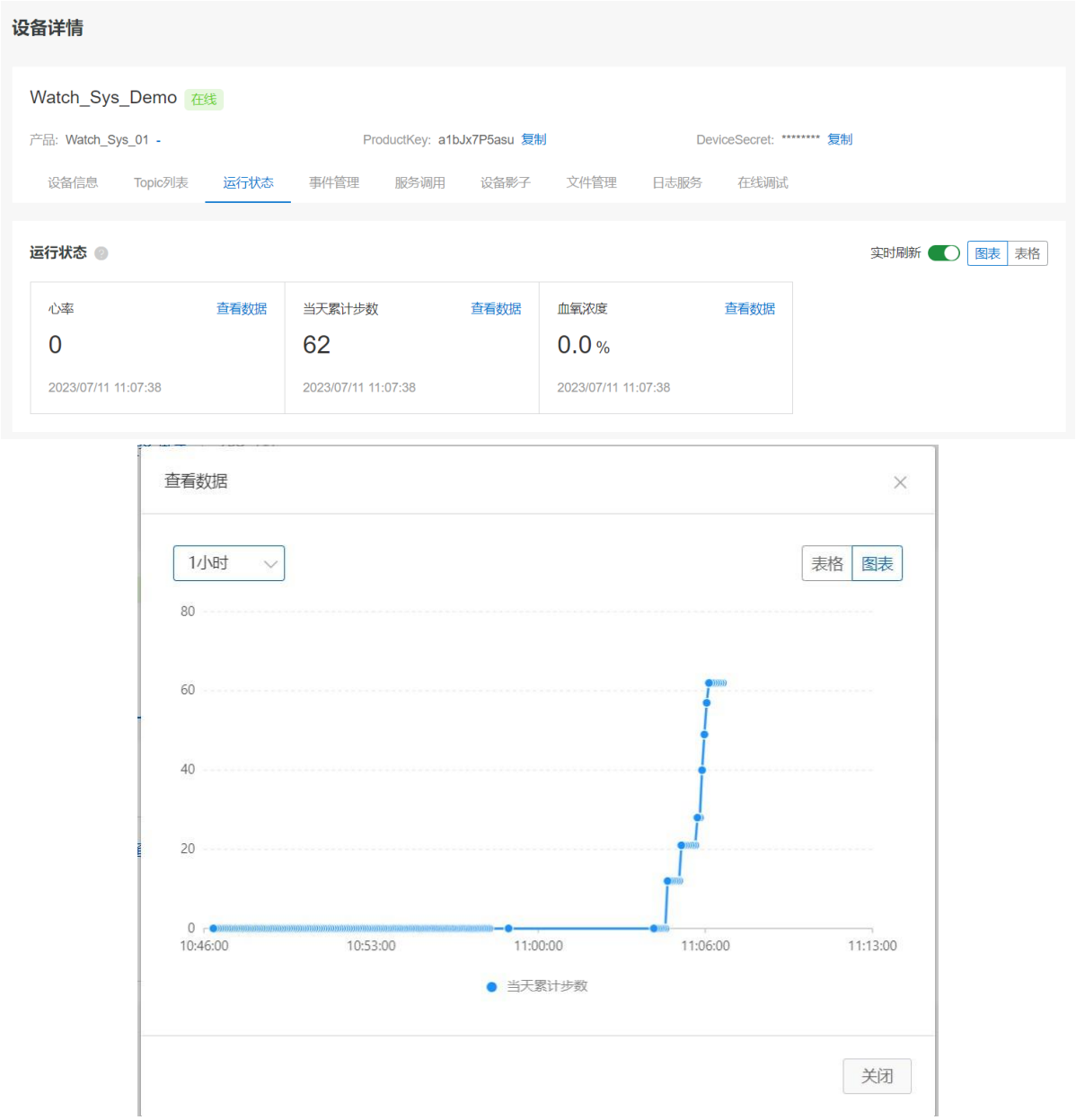
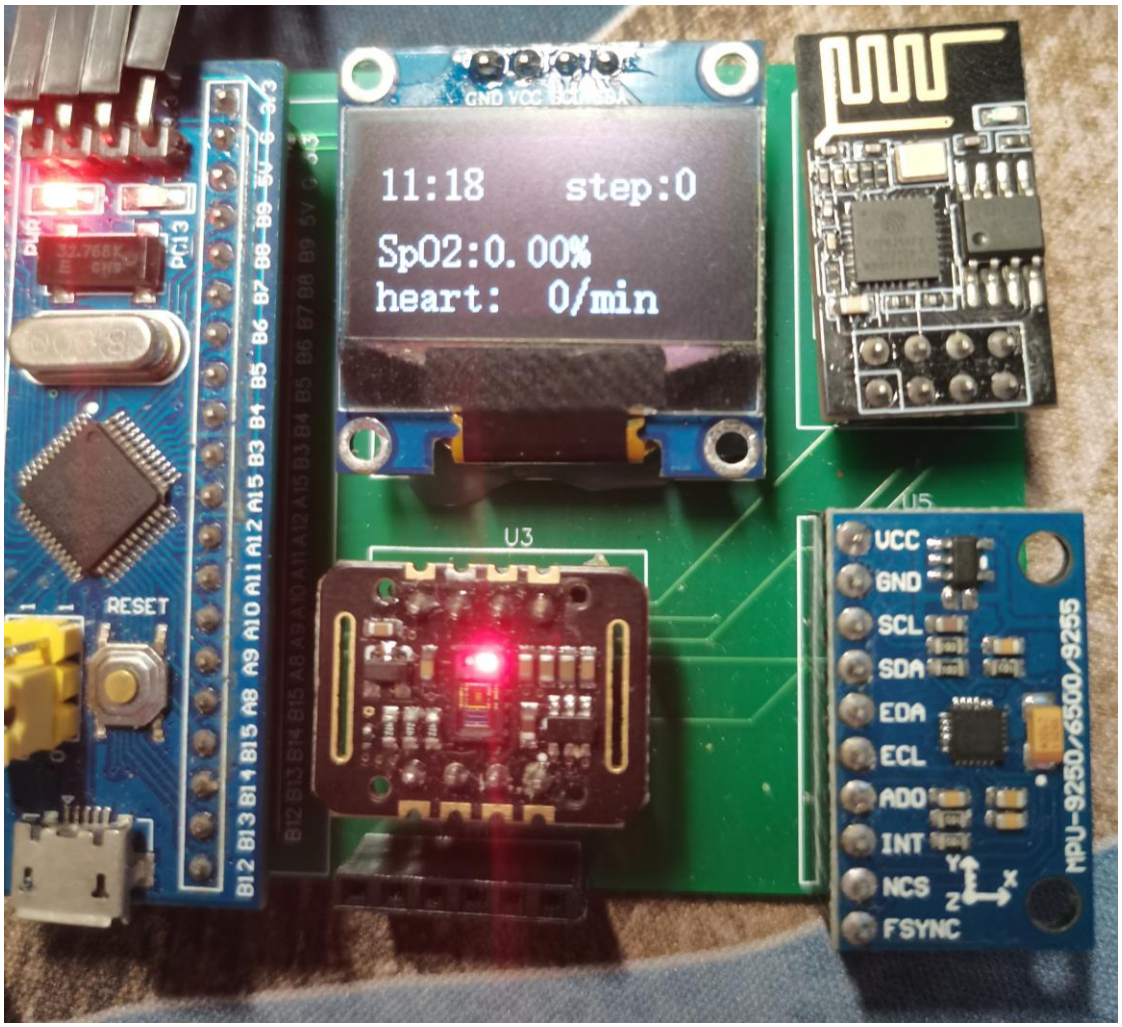


图 5.3 匀速行走时步数采集相关数据

5.3 实时时钟分析

RTC 时间与世界时钟保持一致，且掉电后不丢失。



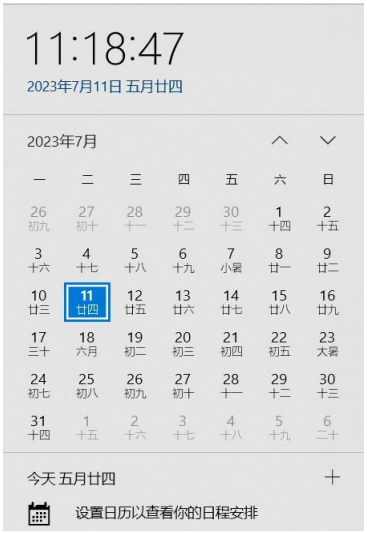


图 5.4 实时时钟分析

5.4 物联网相关分析

上电后，完成 ESP8266 初始化后，ESP8266 配置为 STA 模式并连接上了电脑热点。然后，设备通过 AT 指令顺利连接上了阿里物联网平台，实现了预取功能。

已连接的设备: 1 台(共 8 台)

设备名称	IP 地址	物理地址(MAC)
espressif	192.168.137.146	c8:c9:a3:54:aa:65

图 5.5 物联网相关分析

其次，基于阿里云平台开发了一个公版的 APP 界面，不过现阶段 APP 的数据显示还存在问题，后续希望能够解决。

图 5.6 物联网 APP 界面展示

6 心得体会

在本次课设过程中，我获得了很多宝贵的经验和体会。下面是我对这个课设的心得体会。

首先，我明白了嵌入式开发是一个硬件与软件结合的过程。这个课设要求我们不仅仅掌握 STM32 单片机的编程，还需要与传感器、WIFI 模块等硬件设备进行交互。我深刻认识到了硬件与软件的结合对于实现完整功能的重要性。在开发过程中，我需要熟悉硬件的接口和通信协议，并将其与编写的软件代码进行有效的配合，确保整个系统的稳定性和可靠性。

其次，我对于项目规划与时间管理有了更清晰的认识。由于一些个人事务使得课设时间较紧。本次课设主要包括心率采集、血氧饱和度采集、步数计数和物联网功能。为了能够高效地完成任务，我需要在开始项目之前进行详细的项目规划和时间管理。我将整个项目划分为不同的阶段和任务，为每个任务设置合理的时间和优先级，这样能够更好地控制进度，确保按时完成。

再者，我实际进行了对传感器数据的处理，课设中使用了心率传感器和血氧传感器来采集数据。在处理这些传感器数据时，我学会了使用适当的算法和滤波技术来消除噪声并提取有用的信息，并且将快速傅里叶变换 FFT 这个方法实际应用于单片机上，取得了还不错的效果。同时，我还学会了处理不同传感器的数据格式和通信协议，确保数据的准确性和可靠性。在步数计数算法方面，由于步数计数是课设中的一个重要功能。为了实现准确的步数计数，我研究了多种步数计数算法，并根据实际情况选择了适合的算法。经过千辛万苦将 MPU6050 的标准库 DMP 移植到 STM32F103 芯片上，最终取得了较为满意的效果。

最后，我手把手实现了物联网功能。在这个课设中，我还学习了如何将 STM32 与物联网平台进行连接，并实现数据的传输和远程控制。这次课设选择的是阿里 IOT 平台，我使用了 MQTT 协议和云平台进行通信，了解了物联网架构和通信方式。通过这个课设，我深入了解了物联网的概念和应用，为以后的物联网开发打下了坚实的基础。

总的来说，这次课设给我提供了一个很好的实践机会，让我将在课堂上学到的知识应用到实际项目中。通过这个课设，我不仅提高了对 STM32 的理解和掌握，还学到了很多与硬件和软件结合的技能。这对我今后的学习和职业发展都具有重要意义，让我更加熟悉嵌入式系统开发和物联网应用。我相信这些经验和体会将对我的未来工作产生积极的影响。

参考文献

[1]何智慧,王锦航,刘易婷等.基于 STM32 的便携式低功耗血氧仪的设计与实现[J].延安大学学报(自然科学版),2021,40(03):79-82.DOI:10.13876/J.cnki.ydnse.2021.03.079.

[2] 黎圣峰,庞宇,高小鹏等.便携式血氧信号检测装置设计[J].传感器与微系统,2017,36(03):110-112+119.DOI:10.13873/J.1000-9787(2017)03-0110-03.

[3]刘俊微,庞春颖,徐伯鸾.光电脉搏血氧仪的设计与实现[J].激光与红外,2014,44(01):50-55.

[4] 徐斌,裴晓芳,李太云.穿戴式智能计步器设计[J].电子科技,2016,29(03):178-182.DOI:10.16180/j.cnki.issn1007-7820.2016.03.047.

[5]程学农,李学明.基于阿里云平台的智能门锁系统[J].单片机与嵌入式系统应用,2020,20(09):75-78.

附录(部分源代码)

main.c 文件

```
#include "stm32f10x.h"
#include "systick.h"
//#include "usart.h"
#include "iic.h"
#include "Font.h"
#include "OLED_IIC.h"
#include "MAX30102.h"
#include "algorithm.h"
#include "blood.h"
#include "RTC.h"
#include "delay.h"
#include "inv_mpu_dmp_motion_driver.h"
#include "inv_mpu.h"
#include "math.h"
#include "usart2.h"
#include "AT.h"
#include "stm32_iic.h"
#include "stdio.h"
#include "mpu_user_api.h"
#include "Timer.h"

extern BloodData g_blooddata;
extern const char* pubtopic;  //配置参数，发布消息


int main(void)
{
	IIC_GPIO_INIT();
	i2cInit();		//IIC 总线的初始化

	RTC_Init();
//	RTC_Set(2023,7,11,11,18,00);

	OLED_Init();
	fill_picture(0x00);

	Serial_Init();

	OLED_Printf_EN(1,0,"IWatch is initializing.");
```

```
    OLED_Printf_EN(6,0,"       Please wait");
    Mpu_Init(1);
    esp_Init();

    MAX30102_GPIO();
    Max30102_reset();
    MAX30102_Config();

    fill_picture(0x00);

    Timer_Init();

    while(1)
    {

    }
}
```

blood.c 文件

```
#include "blood.h"
#include "usart.h"

uint16_t g_fft_index = 0;                    //fft 输入输出下标

struct compx s1[FFT_N+16];                   //FFT 输入和输出：从 S[1]开始存放，根据大小自己定义
struct compx s2[FFT_N+16];                   //FFT 输入和输出：从 S[1]开始存放，根据大小自己定义

struct
{
    float    Hp  ;              //血红蛋白
    float    HpO2;              //氧合血红蛋白

}g_BloodWave;//血液波形数据

BloodData g_blooddata = {0};                        //血液数据存储

#define CORRECTED_VALUE          47                 //标定血液氧气含量

/*funcation start ------------------------------------------------------------*/
//血液检测信息更新
```

```
void blood_data_update(void)
{
    //标志位被使能时  读取 FIFO
    g_fft_index=0;
    while(g_fft_index < FFT_N)
    {
        while(MAX30102_INTPin_Read()==0)
        {
            //读取 FIFO
            max30102_read_fifo();    //read from MAX30102 FIFO2
            //将数据写入 fft 输入并清除输出
            if(g_fft_index < FFT_N)
            {
                //将数据写入 fft 输入并清除输出
                s1[g_fft_index].real = fifo_red;
                s1[g_fft_index].imag= 0;
                s2[g_fft_index].real = fifo_ir;
                s2[g_fft_index].imag= 0;
                g_fft_index++;
            }
        }
    }
}

//血液信息转换
void blood_data_translate(void)
{
    float n_denom;
    uint16_t i;

    //直流滤波
    float dc_red =0;
    float dc_ir =0;
    float ac_red =0;
    float ac_ir =0;

    for (i=0 ; i<FFT_N ; i++ )
    {
        dc_red += s1[i].real ;
        dc_ir +=    s2[i].real ;
    }
```

```
        dc_red =dc_red/FFT_N ;
        dc_ir =dc_ir/FFT_N ;
    for (i=0 ; i<FFT_N ; i++ )
    {
        s1[i].real =    s1[i].real - dc_red ;
        s2[i].real =    s2[i].real - dc_ir ;
    }

    //移动平均滤波
    for(i = 1;i < FFT_N-1;i++)
    {
            n_denom= ( s1[i-1].real + 2*s1[i].real + s1[i+1].real);
            s1[i].real=    n_denom/4.00;

            n_denom= ( s2[i-1].real + 2*s2[i].real + s2[i+1].real);
            s2[i].real=    n_denom/4.00;
    }

    //八点平均滤波
    for(i = 0;i < FFT_N-8;i++)
    {
            n_denom= ( s1[i].real+s1[i+1].real+ s1[i+2].real+ s1[i+3].real+ s1[i+4].real+ s1[i+5].real+
s1[i+6].real+ s1[i+7].real);
            s1[i].real=    n_denom/8.00;

            n_denom= ( s2[i].real+s2[i+1].real+ s2[i+2].real+ s2[i+3].real+ s2[i+4].real+ s2[i+5].real+
s2[i+6].real+ s2[i+7].real);
            s2[i].real=    n_denom/8.00;
    }

    //开始变换显示
    g_fft_index = 0;
    //快速傅里叶变换
    FFT(s1);
    FFT(s2);

    //开始 FFT 算法
    for(i = 0;i < FFT_N;i++)
    {
        s1[i].real=sqrtf(s1[i].real*s1[i].real+s1[i].imag*s1[i].imag);
        s1[i].real=sqrtf(s2[i].real*s2[i].real+s2[i].imag*s2[i].imag);
```

```
    }

    //计算交流分量
    for (i=1 ; i<FFT_N ; i++ )
    {
        ac_red += s1[i].real ;
        ac_ir +=    s2[i].real ;
    }

    for(i = 0;i < 50;i++)
    {
        if(s1[i].real<=10)
            break;
    }

    //读取峰值点的横坐标   结果的物理意义为心率对应的频率
    int s1_max_index = find_max_num_index(s1, 30);
    int s2_max_index = find_max_num_index(s2, 30);

    //检查 HbO2 和 Hb 的变化频率是否一致
    if(i>=45) //状态正常
    {
        //心率计算
        uint16_t Heart_Rate = 60.00 * SAMPLES_PER_SECOND * s1_max_index / FFT_N;
        g_blooddata.heart = Heart_Rate;

        //血氧计算
        float R = (ac_ir*dc_red)/(ac_red*dc_ir);
        float sp02_num =-45.060*R*R+ 30.354 *R + 94.845;
        g_blooddata.SpO2 = sp02_num;

    }
    else //数据发生异常
    {
        g_blooddata.heart = 0;
        g_blooddata.SpO2  = 0;
    }

}

void blood_Loop(void)
```

```
{

    //血液信息获取
    blood_data_update();
    //血液信息转换
    blood_data_translate();

    //显示血液状态信息
    OLED_Printf_EN(6,0,"heart:%3d/min    ",g_blooddata.heart);
    g_blooddata.SpO2 = (g_blooddata.SpO2 > 99.99) ? 99.99:g_blooddata.SpO2;
    OLED_Printf_EN(4,0,"SpO2:%2.2f%%    ",g_blooddata.SpO2);

}
```

Timer.c 文件

```
#include "stm32f10x.h"                  // Device header
#include "blood.h"
#include "RTC.h"
#include "mpu_user_api.h"
#include "AT.h"

unsigned long    STEPS;
extern BloodData g_blooddata;
extern const char* pubtopic;  //配置参数，发布消息

void mpu_Loop(void)
{
mpu_set_dmp_state(1);
dmp_get_pedometer_step_count(&STEPS);
}

void Timer_Init(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    TIM_InternalClockConfig(TIM2);

    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
    TIM_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
```

```
    TIM_TimeBaseInitStructure.TIM_Period = 5000 - 1;
    TIM_TimeBaseInitStructure.TIM_Prescaler = 7200 - 1;
    TIM_TimeBaseInitStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStructure);

    TIM_ClearFlag(TIM2, TIM_FLAG_Update);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_Init(&NVIC_InitStructure);

    TIM_Cmd(TIM2, ENABLE);
}

void TIM2_IRQHandler(void)
{
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) == SET)
    {
        blood_Loop();

        RTC_Get();
        OLED_Printf_EN(1,0,"%d:%d",calendar.hour,calendar.min);

        mpu_Loop();
        OLED_Printf_EN(1,70,"step:%d",STEPS);


    printf("AT+MQTTPUB=0,\"%s\",\"{\\\"method\\\":\\\"thing.event.property.post\\\"\\,\\\"params\\\":{\\
\"%s\\\":%d\\,\\\"%s\\\":%f\\,\\\"%s\\\":%d\\}}\",0,0\r\n",pubtopic,"totalStep",(int)STEPS,"SpO2",g_blood
data.SpO2,"heartRate",g_blooddata.heart);


        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
}
```

AT.c 文件

```
#include "stm32f10x.h"                          // Device header
#include <stdio.h>
#include <stdarg.h>
#include "usart2.h"
#include "delay.h"

extern char receive_cmd[256];                   //定义字符串缓冲区
const char* WIFISSID = "WWW";                   //配置 ssid
const char* WIFIPASSWORD = "wyn62399";          //配置 password

const                                                                    char*
ClintID="a1bJx7P5asu.Watch_Sys_Demo|securemode=2\\,signmethod=hmacsha256\\,timestamp=168888
0712615|";
const char* username="Watch_Sys_Demo&a1bJx7P5asu"; //根据云平台账号配置 username
const char* password="7ec58f77844de0175fbadaab6d5106a2048a1bd18c52601286ce81d867730e8b";
        //根据云平台账号配置 passwod
const char* Url="a1bJx7P5asu.iot-as-mqtt.cn-shanghai.aliyuncs.com";     //配置 api

const char* pubtopic="/sys/a1bJx7P5asu/Watch_Sys_Demo/thing/event/property/post";  //配置参数，发
布消息

/*
************************************************************

"product_key": "a1bJx7P5asu",
"device_name": "Watch_Sys_Demo",
"device_secret": "6f8ef89259006691721c35a16c2c1775"

{"clientId":"a1bJx7P5asu.Watch_Sys_Demo|securemode=2,signmethod=hmacsha256,timestamp=168888
0712615|",
"username":"Watch_Sys_Demo&a1bJx7P5asu",
"mqttHostUrl":"a1bJx7P5asu.iot-as-mqtt.cn-shanghai.aliyuncs.com",
"passwd":"7ec58f77844de0175fbadaab6d5106a2048a1bd18c52601286ce81d867730e8b",
"port":1883}

************************************************************
*/


/*
```

```
*************************************************************
*函数名  fputc
*功能  重定向
*形参  无
*返回值  无
*************************************************************
*/
int fputc(int ch,FILE *f)
{
      Serial_SendByte(ch);
      while(USART_GetFlagStatus (USART1,USART_FLAG_TC) == RESET)

      return ch;
}                                                    //重定向
/*
*************************************************************
*函数名  esp_Init
*功能     esp 初始化，连接服务器
*形参  无
*返回值  无
*************************************************************
*/
char esp_Init(void)
{

      memset(receive_cmd,0,sizeof(receive_cmd));
      printf("AT+RST\r\n");                                  //重启
      delay_ms(200);

      memset(receive_cmd,0,sizeof(receive_cmd));           //关闭回显
      printf("ATE0\r\n");
      delay_ms(10);
      if(strcmp(receive_cmd,"OK"))
//          return 1;

      printf("AT+CWMODE=1\r\n");                               //开启 sta 模式
      delay_ms(200);
      if(strcmp(receive_cmd,"OK"))
          //return 2;

      memset(receive_cmd,0,sizeof(receive_cmd));           //连接热点
```

```
    printf("AT+CWJAP=\"%s\",\"%s\"\r\n",WIFISSID,WIFIPASSWORD);
    delay_ms(5000);
    if(strcmp(receive_cmd,"OK"))
        return 3;

    memset(receive_cmd,0,sizeof(receive_cmd));          //配置 mqtt 用户信息
    printf("AT+MQTTUSERCFG=0,1,\"%s\",\"%s\",\"%s\",0,0,\"\"\r\n",ClintID,username,password);
    delay_ms(10);
    if(strcmp(receive_cmd,"OK"))
        return 4 ;

    memset(receive_cmd,0,sizeof(receive_cmd));          //连接服务器
    printf("AT+MQTTCONN=0,\"%s\",1883,1\r\n",Url);
    delay_ms(5000);
    if(strcmp(receive_cmd,"OK"))
        return 5 ;

    memset(receive_cmd,0,sizeof(receive_cmd));
    return 0;                                           //缓冲区清零
}
```

algorithm.c

```
#include "algorithm.h"
#include "stm32f10x.h"

/*base value define----------------------------------------------------------*/
#define XPI                 (3.14159265358979323846264338327950288419 71)
#define XENTRY              (100)
#define XINCL                   (XPI/2/XENTRY)
#define PI                              3.14159265358979323846264338327950288419 71
//定义圆周率值

/*Global data space ----------------------------------------------------------*/
//正弦值对应表
static const double XSinTbl[] = {
            0.00000000000000000     ,   0.015707317311820675    ,   0.031410759078128292    ,
0.047106450709642665 , 0.062790519529313374 ,
            0.078459095727844944    ,   0.094108313318514325    ,   0.10973431109104528     ,
0.12533323356430426   , 0.14090123193758267   ,
            0.15643446504023087     ,   0.17192910027940955     ,   0.18738131458572463     ,
0.20278729535651249   , 0.21814324139654256   ,
```

```
                0.23344536385590542    ,  0.24868988716485479    ,  0.26387304996537292    ,
0.27899110603922928   , 0.29404032523230400   ,
                0.30901699437494740    ,  0.32391741819814940    ,  0.33873792024529142    ,
0.35347484377925714   , 0.36812455268467797   ,
                0.38268343236508978    ,  0.39714789063478062    ,  0.41151435860510882    ,
0.42577929156507272   , 0.43993916985591514   ,
                0.45399049973954680    ,  0.46792981426057340    ,  0.48175367410171532    ,
0.49545866843240760   , 0.50904141575037132   ,
                0.52249856471594880    ,  0.53582679497899666    ,  0.54902281799813180    ,
0.56208337785213058   , 0.57500525204327857   ,
                0.58778525229247314    ,  0.60042022532588402    ,  0.61290705365297649    ,
0.62524265633570519   , 0.63742398974868975   ,
                0.64944804833018377    ,  0.66131186532365183    ,  0.67301251350977331    ,
0.68454710592868873   , 0.69591279659231442   ,
                0.70710678118654757    ,  0.71812629776318881    ,  0.72896862742141155    ,
0.73963109497860968   , 0.75011106963045959   ,
                0.76040596560003104    ,  0.77051324277578925    ,  0.78043040733832969    ,
0.79015501237569041   , 0.79968465848709058   ,
                0.80901699437494745    ,  0.81814971742502351    ,  0.82708057427456183    ,
0.83580736136827027   , 0.84432792550201508   ,
                0.85264016435409218    ,  0.86074202700394364    ,  0.86863151443819120    ,
0.87630668004386369   , 0.88376563008869347   ,
                0.89100652418836779    ,  0.89802757576061565    ,  0.90482705246601958    ,
0.91140327663544529   , 0.91775462568398114   ,
                0.92387953251128674    ,  0.92977648588825146    ,  0.93544403082986738    ,
0.94088076895422557   , 0.94608535882754530   ,
                0.95105651629515353    ,  0.95579301479833012    ,  0.96029368567694307    ,
0.96455741845779808   , 0.96858316112863108   ,
                0.97236992039767667    ,  0.97591676193874743    ,  0.97922281062176575    ,
0.98228725072868872   , 0.98510932615477398   ,
                0.98768834059513777    ,  0.99002365771655754    ,  0.99211470131447788    ,
0.99396095545517971   , 0.99556196460308000   ,
                0.99691733373312796    ,  0.99802672842827156    ,  0.99888987496197001    ,
0.99950656036573160   , 0.99987663248166059   ,
                1.00000000000000000   };

//向下取整
double my_floor(double x)
{
    double y=x;
     if( (*( ( (int *) &y)+1) & 0x80000000)    != 0) //或者 if(x<0)
```

```
            return (float)((int)x)-1;
        else
            return (float)((int)x);
}

//求余运算
double my_fmod(double x, double y)
{
    double temp, ret;

    if (y == 0.0)
        return 0.0;
    temp = my_floor(x/y);
    ret = x - temp * y;
    if ((x < 0.0) != (y < 0.0))
        ret = ret - y;
    return ret;
}

//正弦函数
double XSin( double x )
{
      int s = 0 , n;
      double dx , sx , cx;
      if( x < 0 )
            s = 1 , x = -x;
      x = my_fmod( x , 2 * XPI );
      if( x > XPI )
            s = !s , x -= XPI;
      if( x > XPI / 2 )
            x = XPI - x;
      n = (int)( x / XINCL );
      dx = x - n * XINCL;
      if( dx > XINCL / 2 )
            ++n , dx -= XINCL;
      sx = XSinTbl[n];
      cx = XSinTbl[XENTRY-n];
      x = sx + dx*cx - (dx*dx)*sx/2
            - (dx*dx*dx)*cx/6
            + (dx*dx*dx*dx)*sx/24;
```

```
    return s ? -x : x;
}

//余弦函数
double XCos( double x )
{
    return XSin( x + XPI/2 );
}

//开平方
int qsqrt(int a)
{
  uint32_t rem = 0, root = 0, divisor = 0;
  uint16_t i;
  for(i=0; i<16; i++)
  {
    root <<= 1;
    rem = ((rem << 2) + (a>>30));
    a <<= 2;
    divisor = (root << 1) + 1;
    if(divisor <= rem)
    {
      rem -= divisor;
      root++;
    }
  }
  return root;
}

/*********************************************************************
函数原型：struct compx EE(struct compx b1,struct compx b2)
函数功能：对两个复数进行乘法运算
输入参数：两个以联合体定义的复数 a,b
输出参数：a 和 b 的乘积，以联合体的形式输出
*********************************************************************/
struct compx EE(struct compx a,struct compx b)
{
      struct compx c;
      c.real=a.real*b.real-a.imag*b.imag;
      c.imag=a.real*b.imag+a.imag*b.real;
      return(c);
```

```
}

/*****************************************************************
函数原型：void FFT(struct compx *xin,int N)
函数功能：对输入的复数组进行快速傅里叶变换（FFT）
输入参数：*xin 复数结构体组的首地址指针，struct 型
*****************************************************************/
void FFT(struct compx *xin)
{
    int f,m,nv2,nm1,i,k,l,j=0;
    struct compx u,w,t;

    nv2=FFT_N/2;                    //变址运算，即把自然顺序变成倒位序，采用雷德算法
    nm1=FFT_N-1;
    for(i=0;i<nm1;i++)
    {
        if(i<j)                     //如果 i<j,即进行变址
        {
            t=xin[j];
            xin[j]=xin[i];
            xin[i]=t;
        }
        k=nv2;                      //求 j 的下一个倒位序

        while(k<=j)                 //如果 k<=j,表示 j 的最高位为 1
        {
            j=j-k;                  //把最高位变成 0
            k=k/2;                  //k/2，比较次高位，依次类推，逐个比较，直到某个位为 0
        }

        j=j+k;                      //把 0 改为 1
    }

    {   //FFT 运算核，使用蝶形运算完成 FFT 运算
        int le,lei,ip;
        f=FFT_N;
        for(l=1;(f=f/2)!=1;l++)             //计算 l 的值，即计算蝶形级数
            ;
        for(m=1;m<=l;m++)                   // 控制蝶形结级数
        {                                   //m 表示第 m 级蝶形，l 为蝶形级总数
l=log（2）N
```

```
            le=2<<(m-1);                                    //le 蝶形结距离，即第 m 级蝶形的蝶形结相距 le 点
            lei=le/2;                                       //同一蝶形结中参加运算的两点的距离
            u.real=1.0;                                     //u 为蝶形结运算系数，初始值为 1
            u.imag=0.0;
            w.real=XCos(PI/lei);                            //w 为系数商，即当前系数与前一个系数的商
            w.imag=-XSin(PI/lei);
            for(j=0;j<=lei-1;j++)                           //控制计算不同种蝶形结，即计算系数不同的蝶形结
            {
                for(i=j;i<=FFT_N-1;i=i+le)                  //控制同一蝶形结运算，即计算系数相同蝶形结
                {
                    ip=i+lei;                               //i，ip 分别表示参加蝶形运算的两个节点
                    t=EE(xin[ip],u);                        //蝶形运算，详见公式
                    xin[ip].real=xin[i].real-t.real;
                    xin[ip].imag=xin[i].imag-t.imag;
                    xin[i].real=xin[i].real+t.real;
                    xin[i].imag=xin[i].imag+t.imag;
                }
                u=EE(u,w);                                  //改变系数，进行下一个蝶形运算
            }
        }
    }
}

//读取峰值
int find_max_num_index(struct compx *data,int count)
{
    int i=START_INDEX;
    int max_num_index = i;
    //struct compx temp=data[i];
    float temp = data[i].real;
    for(i=START_INDEX;i<count;i++)
    {
        if(temp < data[i].real)
        {
            temp = data[i].real;
            max_num_index = i;
```

```
            }
        }
        //printf("max_num_index=%d\r\n",max_num_index);
        return max_num_index;
}


//直流滤波器
int dc_filter(int input,DC_FilterData * df)
{

        float new_w    = input + df->w * df->a;
        int16_t result = 5*(new_w - df->w);
        df->w = new_w;

        return result;
}


int bw_filter(int input,BW_FilterData * bw) {
        bw->v0 = bw->v1;

      // v1 = (3.04687470e-2 * input) + (0.9390625058 * v0);
        bw->v1 = (1.2411061909675448 82e-2*input)+(0.97517787618064910582 * bw->v0);
        return bw->v0 + bw->v1;
}
```