支持用户应用程序的
STM32 主控智能腕表设备

ZeptoWatch v3.0

2022 Fall 模拟电路实验
DIY 项目报告

王卓扬 李稚凌 杜凡 卢鸿宇

2022.12.29

目录

1 项目简介

1.1 背景重述

> *随着物联网技术飞速发展，智能设备小型化进程加速，可穿戴设备逐渐成为继智能手机后又一为大众广泛接受的智能设备形式。腕表作为经典的示时工具，其便携性和实用性经得起时间的考验；加之集成电路技术高速发展，赋能传统工具向智能设备转型，可使其具备交互操作、智能传感、无线通信等功能，深度对接现代智慧生活。*
>
> *近年市面上已有大量智能穿戴设备涌现，有聚焦运动检测的运动手环，亦有完整的搭载操作系统的智能手表。而与传统腕表产业相同，量产产品往往激起用户个性化定制的需求，这使得智能手表的产品形式有较大创新的空间。*

以上是我们在开题报告中撰写的项目背景，而在项目实际进行过程中，随着对智能手表市场概况了解更加全面，我们的目标也发生了一定程度上的修正。

我们了解到，消费者多将当前市面上的智能腕表设备大致划分为两类。一是真正的智能手表，基于高性能 CPU，运行 WearOS 等通用操作系统的“小手机”；二是所谓的“大号手环”，多以单片机为主控，运行实时操作系统（RTOS），仅实现特定的功能，但固件缺乏可拓展性，出厂后偶有软件更新，但原理和性能与智能手表完全不同。二者在功能、功耗方面各自占据市场，彼此间没有太多交集。

那么仅用普通单片机能否实现突破“大号手环”框架限制的产品呢？实际上 2021 年各厂商都进行了类似的实践，主要以降低智能手表的功耗以及提高“大号手环”的可扩展性两条思路为主。

于是我们也想进行一些尝试，开发一款可以“安装用户 APP”的智能手环。我们的思路有部分来自 Casio 和 TI 的可编程图形计算器：它们分别在一些机型上实现了对类 BASIC、Lua、MicroPython 等的支持。我们发现了一项尚在起步期的开源项目 PikaScript，作者重写了 Python 解释器并做到了高度轻量化，这使得我们的努力有了方向。

1.2 成果特性简述

我们先简单介绍本项目的一些特性：

- **灵敏的触摸控制以及较流畅的动画**
  - CST816 触摸芯片，IIC 协议
  - GC9A01 显示芯片，SPI 协议
  - 广泛应用的 LVGL 嵌入式 GUI 框架
  - 亮度无极调节
- **丰富的硬件外设及对应封装**
  - 板设 EEPROM、LCD、IMU、麦克风、振动马达、蓝牙等资源
  - 编写好的固件驱动与脚本开发 API
- **内置脚本解释器**
  - 使用 PikaScript 作为内建 Python 脚本解释器

- 将用户存储的 Python 脚本作为应用程序执行

- **Type-C 接口**
  - 锂电池充电管理供电接口
  - 实现 Full-speed USB 2.0 通讯
- **可识别为 USB 大容量存储设备**
  - 插上电脑即可像 U 盘一样读写文件
  - 实测 Windows / Ubuntu 均可识别
- **FAT 文件系统**
  - 使用 FatFs 实现
  - 基于 64KB EEPROM 空间建立
  - 存储用户应用程序与资源
- **GPL3.0 开源协议**
  - 项目地址：https://github.com/Gralerfics/ZeptoWatch

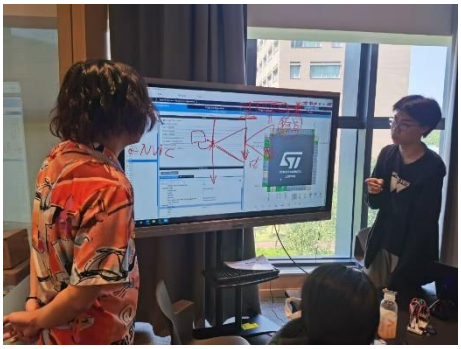
……还有其它尚待完善的功能与设计。

相较开题报告提出的目标，我们加入了新的想法，也不可避免地因时间原因放弃了部分功能。于此附上整体的图片展示，细节见后文：

2 过程报告

2.1 讨论交流

2.1.1 组内分享会




分享中（杜凡同学在拍照）

在项目开始前，我们择空于讨论间等场所进行了分享会，分享一些关于单片机与硬件设计的基础应用知识，帮助大家做好准备。

2.1.2 专业人士指点

一位组员的父亲是相关领域的专业人士，为我们在硬件设计上提供了大量改进建议，如指出初版设计中电源芯片开启电压不足的问题、提议使用恒流芯片给屏幕背光供电、使用软件开关、使用 USB 软上拉等，弥补了我们对工业界各种主流实践了解上的不全面性。

如若缺少这些经验，可以预见到本项目的硬件设计将会非常不实用且不稳定，甚至可能无法支持后续的固件开发。

在此感谢他的指点（orz）。

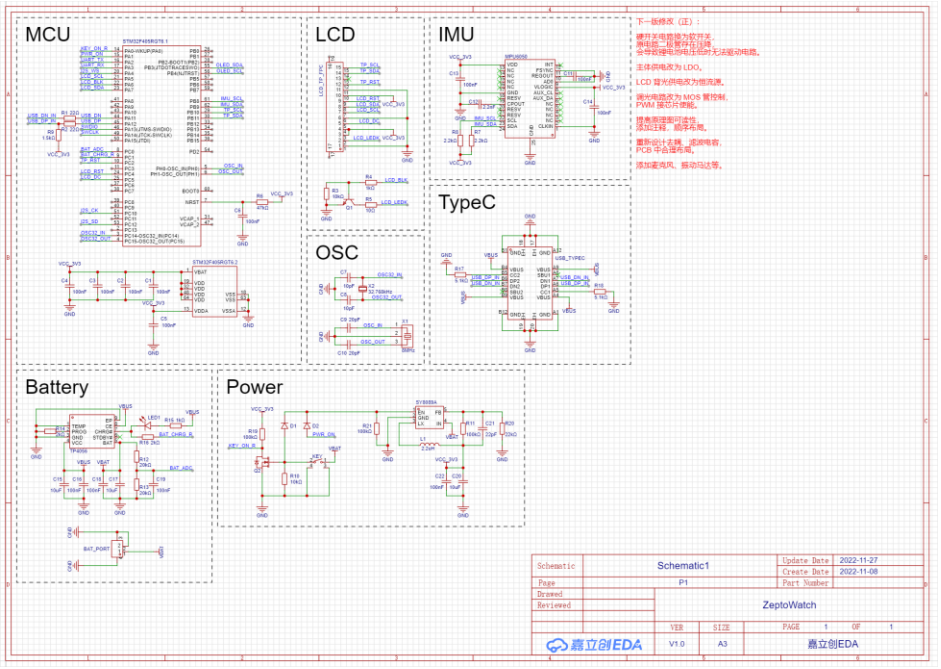
2.2 硬件制作

2.2.1 原理图与 PCB 设计

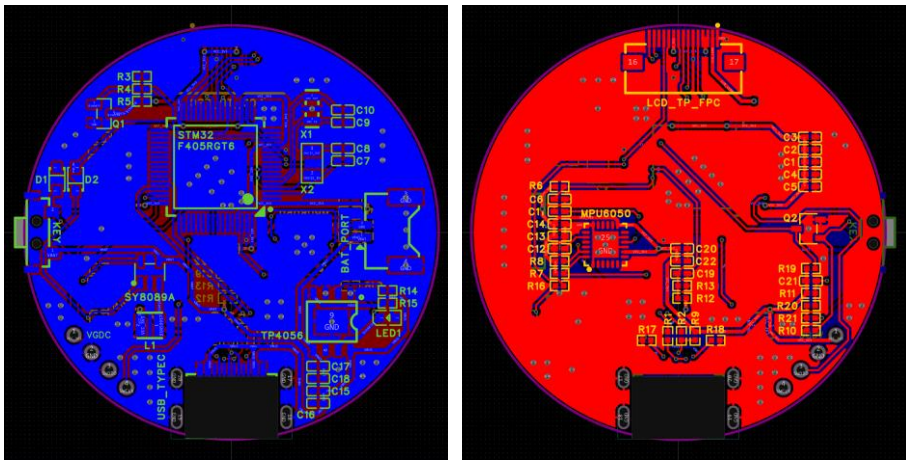
除去开始插面包板的原型验证阶段，我们总共进行了四版原理图设计，和其中三版的 PCB 布线设计。

嘉立创于今年更改了规则，要求普通的优惠券必须使用立创 EDA 才能免单，所以半途改用立创 EDA。

【Ver 0.0】比较想当然的设计，主要接受指导的也是该版。不过因为后来才接受了指导，已经设计完了 PCB 生产出来了，所以浪费了一版，有些可惜。



Ver 0.0 原理图设计



Ver 0.0 PCB 设计

可以看到首先硬件资源不多，布线也相对简单。

这一版中存在不少明显的问题，一是电路设计和器件选用不合理：

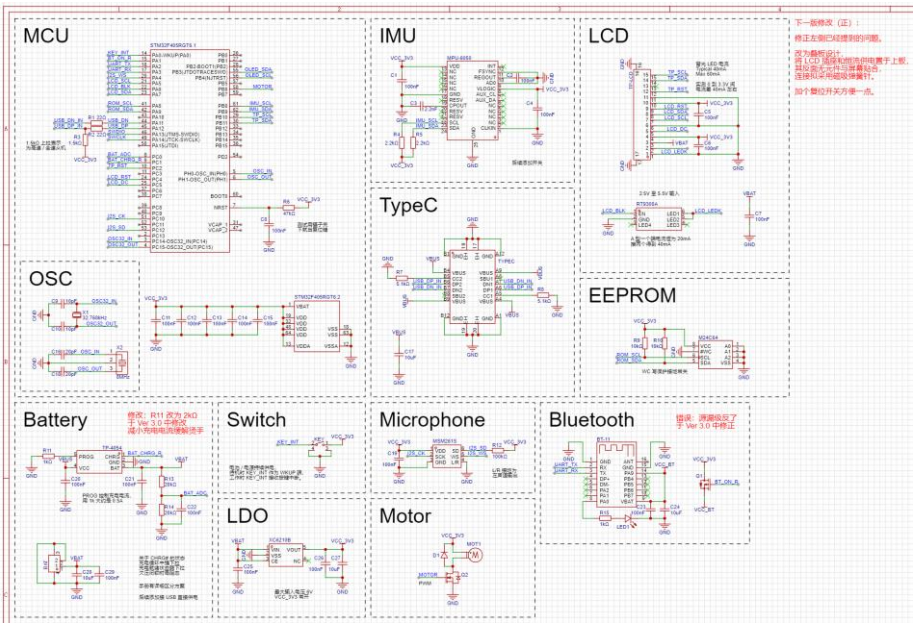
- 电源开关用硬件电路设计，可靠性差；
- 未考虑二极管压降，可能导致锂电池电压略有下降后就不满足电源芯片的输入电压最低值，甚至直接无法工作；
- 充电指示灯供电、信号线耦合；
- LCD 背光供电未使用恒流源，且三极管功耗更大；
- 原理图缺少标注；
- 不少地方仅为美观没有遵守信号从上至下的惯例约定。
- ……

二是 PCB 设计不合理：

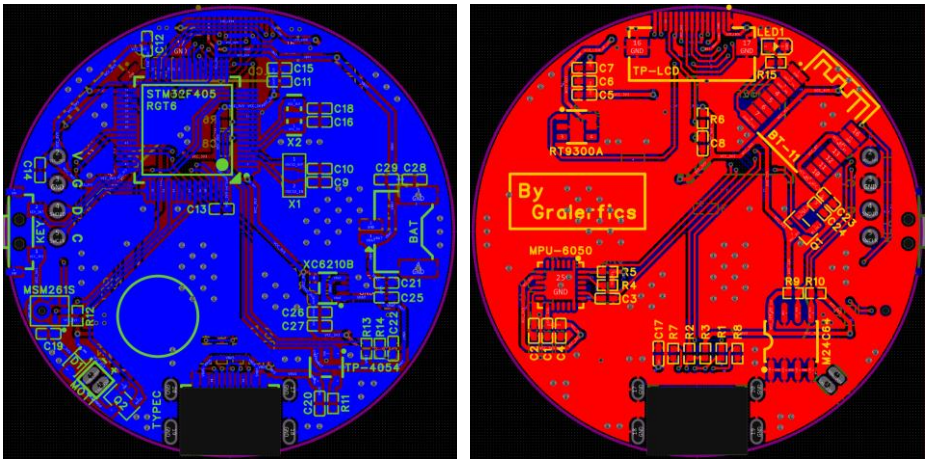
- 没有理解去耦电容、滤波电容的实际作用，只是依葫芦画瓢设计了原理图，未能在 PCB 设计中体现（排成一排并不会有什么作用）；
- 屏幕位置尴尬，且 FPC 座偏的方向反了导致会有点扭排线；
- 乱改电气规则，嘉立创精度不高，容易出事；
- 通 GND 过孔打得不均匀；
- ……

【Ver 1.0】经指点后改进的设计，没有进行布线，直接进入下一版，作了更多修改。

【Ver 2.0】加入杂七杂八的东西，修改了前述各种问题。



Ver 2.0 原理图设计



Ver 2.0 PCB 设计

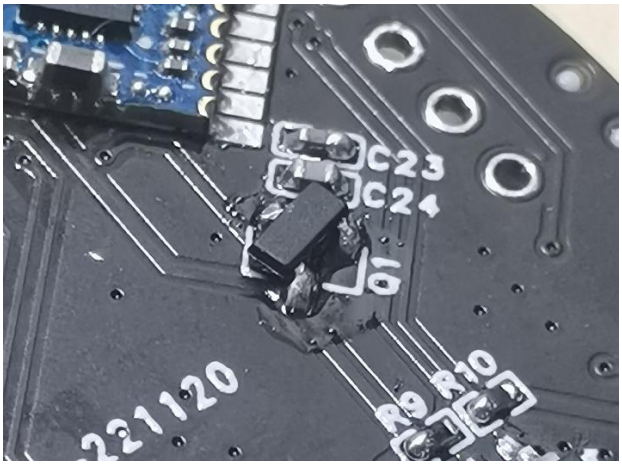
可以看到稍微像样了点，不过其实布局还可以优化，有很多空余空间可以使用。同时该版并没有解决屏幕安装尴尬的问题——屏幕最好是紧贴一面无元器件的 PCB（因为我们暂时没有外壳固定）。

由于第零版知道有问题所以直接没有做出来，上述第二版实际上是第一个打板并焊接好的版本——即对改版进行实际测试就可以得知**原理图设计是否正确可用**。

经过测试，大部分设计无误，但还是存在一些问题：

- 上述屏幕问题；
- 蓝牙天线下最好不铺铜、无平行导体；
- 电池插座可以再往里收收；
- 所用麦克风型号缺货（后来淘宝买到了，所以第三版没有更换）；
- 以及最重要的：控制蓝牙供电的开关无效。

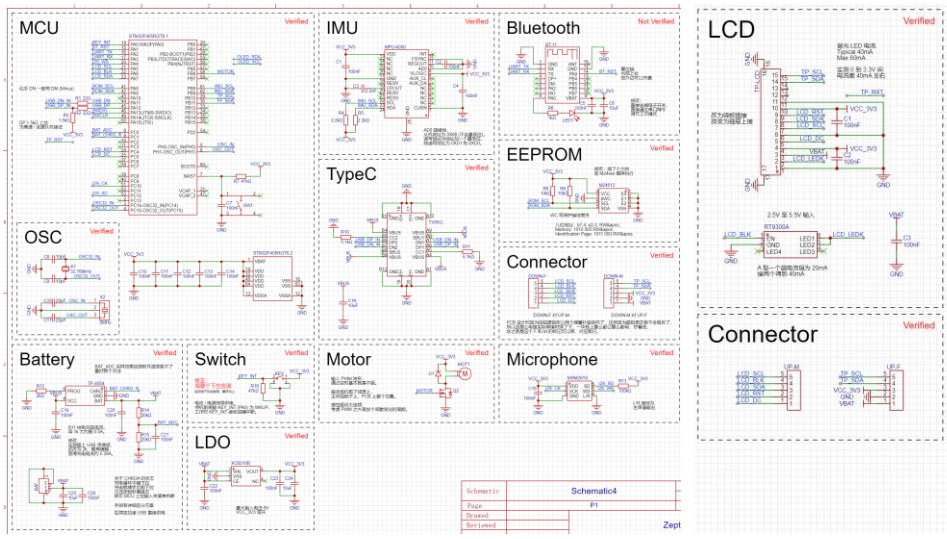
关于最后一个问题，最初发现是 PMOS 管源极和漏极直接设计反了，导致内部的二极管一直导通，相当于不存在。不重新打板前提下的暂时性解决方法：把 MOS 管翻个面并且引脚掰反过来，重新焊回去，就调换了两极。



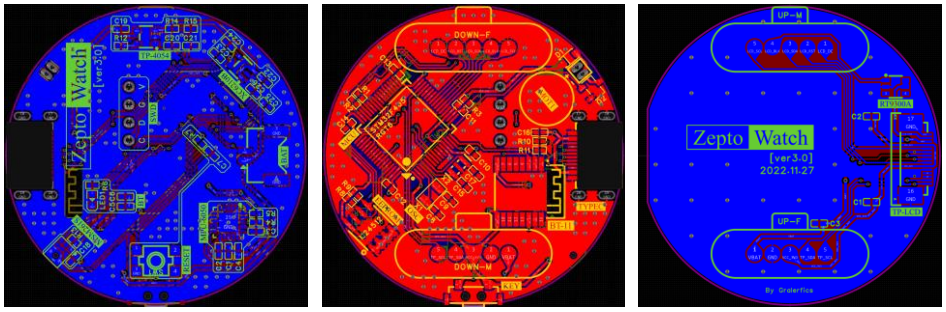
给 MOS 管翻身

……令人惊喜的是如此操作后问题仍然没有解决，并且其供电变得更加玄学。后来经过考察，没有找到具体的原因，但猜测是从通讯用的串口给蓝牙模块供电了，所以打电表测得的电压约在 2.7V 左右，处于不稳定但确实在供电的状态。所以下一版的解决方案是直接去掉，改软件控制其休眠。（不过这里感觉应该是有硬件上的解决方法的）

【Ver 3.0】综合前面所有的考量，第二版验证后马上就开始设计第三版。并且为了解决屏幕的问题，采取了两层板叠放的设计，使上板有足够的空闲留一面贴放 LCD。



Ver 3.0 原理图设计

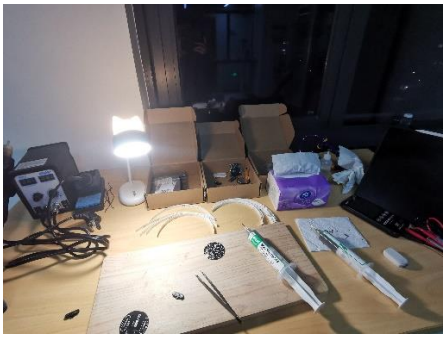


Ver 3.0 PCB 设计

由于连连看水平见长，可以看到第三版的布线设计更加紧凑了一些。该版基本解决了前述所有的大问题，也是当前正在使用的一版，还闲得没事加了很多丝印。两板间采用磁吸弹簧针连接，手感很好，但打算在下一版去掉，主要因为连接不稳定、高度太高、类似 BGA 的封装焊起来比较麻烦——该版的其它修改计划见后改进措施一节。

2.2.2 焊接与测试

首先经历本项目后深感身边有工具的必要性。第三版焊接时没有做好，导致软件调试时隔三岔五就出现硬件问题，一般都是虚焊等等。一旦出问题就搬工具出去解决，不断体味“工欲善其事，必先利其器”的深刻内涵。



半夜三点的活动室

好在备齐了工具：包括焊台、恒温加热台、电表、夹具、台灯、橡皮、镊子等工具，以及焊锡、锡浆、焊膏、洗板水、胶带等耗材。

设计时考虑板子大小，电容电阻都采用 0402 封装，下一版可以尝试 0201 画单板。

附生产的几版 PCB 和一些过程图：

各版本实物及某次上锡

2.3 固件编写

2.3.1 环境与工具准备

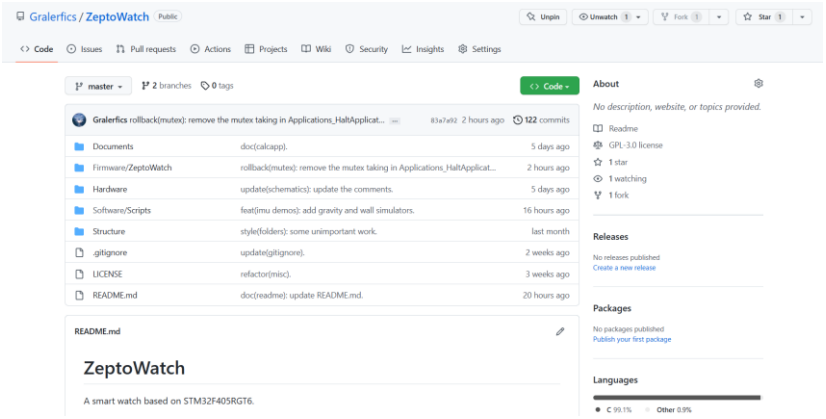
主力 IDE 使用 CLion，编译器使用 MinGW-GCC，固件下载使用 OpenOCD，项目用 CMake 管理，C 与 C++混编。

其它工具、软件还有：

- STM32CubeMX 用于生成代码和参考官方实现；
- CH340 模块及 Serial Port Utility、MobaXterm 等用于串口调试；
- ST-Link-v2 配合 4-Pin 钻石头和板上设计的烧录孔用于固件下载调试；
- 逻辑分析仪及其上位机软件在调试 SPI 和 USB 方面有显著作用；
- STM32 ST-Link Utility 在检查芯片是否安好方面有奇效；
- Squareline Studio 用于设计参考 LVGL 界面；
- Illustrator 和 Photoshop 用于处理部分图片素材；
- ……

2.3.2 版本管理

上传至 Github 进行托管，使用 CLion 自带的 Git 工具进行版本管理，非常方便。截至目前共有 122 Commits，且描述较为规范。



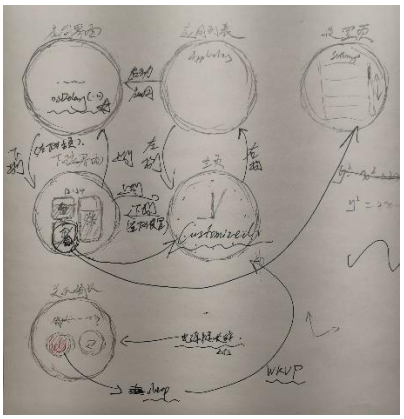
项目仓库

2.3.3 架构设计

整体的固件工程结构设计简写如下：

- 板级资源驱动
  - LCD 显示与触摸
    - 触摸芯片 CST816 驱动采用 GPIO 模拟 IIC 协议实现触点获取
    - 显示芯片 GC9A01 驱动使用硬件 SPI 配合 DMA 实现高速搬运
    - 还可以借此实现屏幕拔掉重插的复位操作（注释掉了）
  - 实时时钟

        - 封装了安全获取 Date 和 Time 的驱动
    - EEPROM
        - 驱动使用 GPIO 模拟 IIC 信号与 M24512 芯片通讯
    - IMU
        - 驱动使用 GPIO 模拟 IIC 信号与 MPU6050 通讯
    - 电源（键）管理
        - 按键中断服务函数定义
        - 息屏、进入 StandBy 模式（关机）等
    - 麦克风
        - 使用硬件 I^2S 配合 DMA 以 8k 采样率获取麦克风采样结果
    - 振动马达
    - 电池电压采样
        - 使用 ADC 配合 DMA 获取电池电压分压的采样结果
    - 蓝牙
    - ……
- FreeRTOS（实时操作系统）
    - 实现多任务并行
- LVGL（通用嵌入式 GUI 框架）
    - 独立 UI 界面

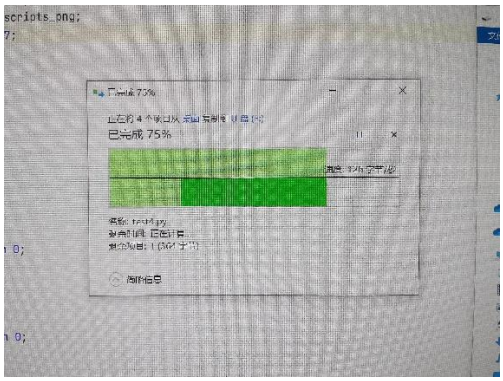


UI 设计草图

    - 完善 pika_lvgl 开放 API 给脚本应用程序
    - 系统内置界面的设计（日期时间设置、主页钟表、主页下拉、APP 界面下拉菜单）

主页左滑切入日期时间设置

- FatFs（文件系统）
  - 编写工具函数实现安全的文件读写等



连接电脑直接拖入脚本

- PikaScript（开源轻量级 Python 解释器）
  - 解释器核心组件
  - 自定义扩展包（见附录）
- 内存管理
  - CCMRAM
    - 所用单片机具有 64K CCMRAM，可以存放非外设使用的资源
    - 修改链接脚本将部分地址改到 CCMRAM，空出的空间分给实时操作系统和 Python 虚拟机
  - LVGL 内存释放
    - 应用执行完毕后及时清理 UI 对象
  - 虚拟机内存释放
    - 应用执行完毕后释放根对象内存
    - 编写扩展库时释放参数所占内存
- ……

2.3.4 脚本应用程序开发

我们内置了 PikaScript 用于解释运行 Python 程序，并且设计了扫描应用的功能。开机或者在应用界面再左滑就会自动扫描文件系统根目录下的所有 Python 脚本（暂时没有作太多判断，最好别放其他类型文件）。

在应用列表左滑主动刷新，扫描中

由此用户可以自己开发脚本应用，通过 USB 放入 ZeptoWatch，然后刷新列表，就可以看到自己的应用了。顺带一提，可以通过修改脚本开头的注释内容修改列表中应用图标的名称、图标（有限选择）和底色，具体规范详见附录 A。

关于 APP，组员提出了不少有意思的想法，于是我们提供了一些示例程序，一些是测试用，一些确实有实用性：

- BATUTIL.PY：电源插拔检测和电池电压采样测试，字体大小变更测试。
- CALCAPP.PY：简单的**计算器**，按钮阵列测试。
- FREQ.PY：**声音频谱**显示，麦克风、LVGL 样式、FFT 测试。
- LVTEST.PY：LVGL 测试，单个按钮与点击事件回调。
- MERITS.PY：**电子木鱼**，内置图片资源获取、振动马达测试。
- SETTINGS.PY：时间设置，roller 和 rtc 接口测试（设置已弃用改内置）。
- TEST.PY: 屏幕亮度变更、温度传感器测试。
- BALANCE.PY：运动传感器测试，**模拟重力**施加在一个对象上使其运动，碰到边缘会反弹。
- GRAVITY.PY：将前一个程序中的对象复制五份，五个物体在屏幕上随用户倾斜而滑动。

这里有一些图片展示：

计算器、声谱仪、电子木鱼、重力物体物理模拟

2.4 外壳设计

只有概念设计，尚未进行实际设计。

早先的外壳概念设计

2.5 典型问题记录

开发过程中我们遇到极多技术问题（各种意义上来说运气都非常差），下面挑一些印象深刻的进行记录和说明。

2.5.1 开发环境问题

使用 CMake 构建在某些情况下不如 Keil 方便，引入库时需要自己写。开始时对这块并不熟悉，出现了很多无法构建成功的问题，后期遇到问题大多都能一眼解决了。

C 和 C++混编时也出现大量问题，其中主要都是 extern “C” {}的使用问题。C++部分需要时刻注意有些方法不框在上述声明中，在构建中是无法被编译器找到的。

2.5.2 硬件问题

虚焊时有发生，购置了电表之后就一切安好无虞了。第二版焊接芯片时非常完美，第三版不知道出了什么问题（疑是中、低温锡混杂了），很多焊点都变成白而僵硬的死锡，好在也能用。特别是近似 BGA 封装的弹簧针，焊盘对焊盘压在下面看不见，因为最初上锡没把握上少了，导致十天半个月后终于断裂，卸下重焊不表。

最令人气愤的是淘宝买的芯片某一块有一天突然就坏了，各种读取和检查确认应该真的是坏了，无奈把上一版的芯片拆下来换上。

2.5.3 USB 问题

CubeMX 生成的 USB 代码始终不能正确工作，我们被该问题困扰长达一周半，近乎放弃，最终更新了一下 CubeMX 就解决了。

从此吸取教训，官方代码的 Bug 也是时常会有的，须及时更新。（更新后会导致工程结构出现问题，好在此时已经具备了解决的能力，没有其它影响）

2.5.4 文件系统问题

首先是文件系统一直无法通过电脑格式化，考虑直接调用 f_mkfs 在制作文件系统，一直报错。后来查询相关资料了解到 FatFs 针对太小容量的存储器作了限制，需要人工更改其源码，注释掉两句才能用。

第二是创建后的 FAT 文件系统迟迟无法被 Windows 识别，后来抱着试一试的心态用 Ubuntu 读取，成功识别了。又某一天 Windows 突然能识别了，至今不明原因。不过当时无法识别大概是因为 Windows 所谓“智能”设计，把太小容量的设备自动识别为了 FAT16 等其它格式。归根究底应该还是 EEPROM 容量太小的原因，所以下一版提出换用更大更快的 SPI Flash ROM。

2.5.5 解释器不完善问题

PikaScript 作为去年刚起步的开源项目，还有诸多不完善之处。

比如第一是应用程序崩溃后虚拟机会直接进入设计的死循环，导致无法通过硬件复位以外的手段复原。在向作者建议后加入了__platform_panic 声明，用以覆写原先缺失的崩溃处理。在其中进行内存释放和重置虚拟机即可解决应用崩溃后整个系统死机的问题。

第二是提供的包数量不多。我们并不需要太多功能，主要需要使用到作者写了一半的 pika_lvgl——这是面向 PikaScript 对 LVGL 的封装，如果要运行用户创建 GUI 界面就需要它。而 pika_lvgl 尚不完善，故我们自己继续编写了部分所需功能，之后也可考虑并入主仓库。

2.5.6 软件工具问题

软件工具经常出问题。其中比较麻烦的是 LVGL 官方的 Squareline Studio 设计器，其免费版只能添加 5 个页面，而我们需要 6~7 个。付费一个月也无不可，但按官方提供的渠道，没有信用卡连付款购买都做不到。

于是尝试旁门左道，一是可以打开它的示例项目，其页面数多于 5 个，清空另存为即可。

二则更普适一点，打开其工程文件发现官方没有对工程文件进行加密，里面是以 json 格式组织的。网上找 json 格式化工具对其进行整理，找到页面相关的对象，复制，把 Hash ID 全部改掉即可，打开后正常工作。

2.5.7 死机原因合集

死机情况的出现几乎已经被我们习惯，直到最近才基本完全解决。总结来说死机出现的情况大致分三个原因，以及除此之外的玄学原因：

- **解释器堆栈溢出**：Python 虚拟机在执行应用时堆栈溢出。这主要是脚本编写和库编写上出了问题。除此之外只能开大内存解决，好在之前对 CCMRAM 的配置使得主 RAM 腾出不少空间可以分给 Python VM。
- **RTOS 堆栈溢出**：FreeRTOS 堆栈溢出，临床表现就是直接死机，非常令人无语。
- **LVGL 线程安全问题**：这是此前绝大部分莫名其妙死机产生的原因，直到最后这两天才知道原因并解决。LVGL 默认没有保护线程安全，这意味着在 FreeRTOS 不同任务中访问 LVGL 资源可能会导致冲突后宕机。解决方案是非常传统的，添加 Mutex 互斥量——但不看文档根本不知道这回事。

3 改进方案

3.1 硬件设计方面

3.1.1 减薄设计

回归单板设计，去掉磁吸弹簧针，单面贴屏，尝试能否在一面上布通所有器件。

3.1.2 升级 IMU

换用 MPU9250，带有磁力计，通过卡尔曼滤波可以获取完整的欧拉角，否则无法获取偏航角（yaw）。

3.1.3 USB 改进

添加 USB 软上拉，便于提醒计算机重新枚举识别；差分线做等长，或者也可以改 High-speed 设计。

3.1.4 供电设计改良

可以换 LDO 为 DCDC；添加 USB 直接供电，否则充电管理 IC 不插电池就不停啸叫，并且就这样冒烟烧掉过两次；锂电池充电 IC 考虑换回 TP4056，带有温敏保护；电池采样结构需要改进，当前的采样计算波动非常大；各外设独立供电电子开关；以及需要添加 RTC 单独供电（考虑纽扣电池）。

3.1.5 电源按键

换用更合适的开关器件，可以加一点硬件消抖，比如加个电容。

3.1.6 布局考虑更全面

例如各功能电容的位置布局考虑。还有例如当前设计中有两个电容设计时没考虑到，有点挡到抽屉 FPC 的抽屉。

3.1.7 更换存储器

EEPROM 换用 SPI Flash，速度更快，容量更大。现在的 IIC 暴力读写非常慢，修改、存储脚本在 Windows 上往往花掉 15 秒左右的时间。

3.1.8 调试接口

除了 SWD 烧录孔以外，把 USART6 串口引出来方便调试。

3.1.9 添加外设

更换蓝牙模块为更小更便宜的集成芯片，或直接添加 WIFI 功能。添加扬声器（附带还需要放大电路）。添加 NFC（如果还是叠板设计，上板已经预留了天线的位置）。

3.2 软件设计方面

3.2.1 改进架构设计

略。

3.2.2 改进内存管理

更深入地学习 C 与 C++以达成目标。

3.2.3 统一复用代码

外设中大量使用了 IIC 通讯。我们为了省事有些用了现成的驱动，导致例如 GPIO 模拟 IIC 等代码重复写了很多次，而这些都是可以复用的。

3.2.4 不使用 HAL 开发

HAL 库资源占用大且不适合底层开发，容易打乱项目结构。如果可以的话以后可以自己封装需要的 MSP，预计能大幅节约资源占用。

3.2.5 更换 FreeRTOS

可能是我们对 FreeRTOS 还不够熟悉的原因，感觉它的功能相对有些弱，后续可以多了解其它如 RT-Thread 等。

3.2.6 裁剪 LVGL 等

对大型库如 LVGL、PikaScript 等可以进行裁剪，虽然不指望能裁剪到 200kB 以内，但总会节约一些资源占用。

4 经验收获

本次项目的制作对于我们组而言可谓收获颇丰，体现于应用需求的分析与规划、产品的设计制造、通用调试测试技能以及团队通力合作等多个维度上。

在开始制作智能手表之前，我们进行了产品应用需求的分析，大致确定想要制作的智能手表的功能以及特性。针对应用需求，我们提出了样式繁多的猜想，并分别寻找可能的解决方案——这一过程锤炼了我们的思维创新能力，并对我们查阅相关资料文献，以及对现有同类产品进行调研与参考的能力提出了高要求。相应地，我们也从中获得了对实际产品市场更深入的理解，体验了分析功能特性之需求与创新方案设计的全流程。

在确定了需求和设计目标之后，我们开始设计实际的硬件结构。包括制作原型、绘制原理图、测试不同的元器件等。在这一过程中，我们更加熟练地应用到课堂中所学习的模拟电路、数字电路等相关的知识，以更好的完成电路功能的实现。自然而然地，在开发过程中我们遇到了不可胜计的技术难题，甚至于一度停滞不前，但瓶颈最终仍一一为我们所突破。在制作的过程中，我们对产品设计有了更深刻的理解，并且也学到了一些解决各种技术难题的方法。

伴随着软硬件制作的是日夜相继、反反复复的调试过程，我们曾多次走进玄学 Bug 的胡同举目无出路，也一次又一次抬头再见柳暗花明。软硬件结合的调试相比以往参与过的任何项目都要更复杂上一个维度，考虑问题的全面性要求前所未有地高，这一度让我们山穷水尽——也许在其中收获最大的，便是不抛弃、不放弃的精神。

在组队进行 DIY 项目实践时，我们需要管理对本项目的整体进度进行充分的管理，合理有序地分配任务，并确保项目按时而优质的完成。由此，我们的项目管理能力得到了很好的提升。在适当合理的管理分配下，我们的合作井然有序，最终成品点亮的那一刻令组员们心神激荡。

此外，在实践过程中，需要用到许多课堂上尚未出现的技术和知识，这要求我们不断学习和掌握新的技能，这一过程帮助我们提升了学习能力，并使我们在日后的学习和科研中更加具有优势。

总而言之，我们收获了宝贵的知识和精神力量（x）。

附录 A 应用开发简明手册

A.1 开发流程

脚本应用程序的开发流程大致如下：

- 编写脚本头尾注释（见下应用信息）
- 引入需要的包
- 编写主循环与非阻塞延时部分
- 编写脚本主体
- 保存为文件名不超过 8 字符、扩展名为.py 的脚本文件
- 安装 APP（拖进根目录）
- 刷新 APP 列表，运行 APP

A.2 应用信息

脚本头注释包含了当前应用将会显示的名称、图标、图标底色。例如首行写上：

```
###ICON=calc;NAME=Calculator;COLOR=13726207;###
```

将代表一个图标为 calc（内置的计算器图标）、显示名称为 Calculator、底色为紫色的应用程序。具体不再展开，详细可见**对计算器 APP 示例的解析**：ZeptoWatch/calcapp.md at master · Gralerfics/ZeptoWatch (github.com)

A.3 基本语法

详见 PikaScript 官方文档，是 Python3 语法的子集。

A.4 封装库

具体不再展开，详见简易的文档：ZeptoWatch/api.md at master · Gralerfics/ZeptoWatch (github.com)。

A.5 安装应用

直接连接电脑拖入根目录即可，等待传输完成再断开连接，否则可能损坏文件系统。

附录 B 封装库简要说明

部分封装库的略详细一些的说明见：ZeptoWatch/api.md at master · Gralerfics/ZeptoWatch (github.com)。

B.1 import PikaStdLib as std

详见官方文档：5.1. PikaStdLib 标准库 — PikaScript 0.1 文档 (pikadoc.readthedocs.io)

B.2 import ZeptoWatchStdLib as zws

提供阻塞与非阻塞延时、LVGL 手动更新、屏幕亮度调节、应用界面根对象获取、实时时钟获取与设置等 API，具体暂略。

详见：ZeptoWatch/ZeptoWatchStdLib.pyi at master · Gralerfics/ZeptoWatch (github.com)，以及脚本应用程序示例。

B.3 import ZeptoWatchPeriphLib as zwp

提供 IMU、电池采样、麦克风、振动马达等相关 API，具体暂略。

详见：ZeptoWatch/ZeptoWatchPeriphLib.pyi at master · Gralerfics/ZeptoWatch (github.com)，以及脚本应用程序示例。

B.4 import ZeptoWatchAssets as assets

只是提供了一张木鱼图片内置资源供电子木鱼应用使用，具体暂略。实际上是未完成下的临时解决方案，计划中需要封装文件系统 API，允许用户获取存储器中的美术资源。当前可以仿照木鱼图片的引用方式修改固件来添加用户自己的图片资源。

详见：ZeptoWatch/ZeptoWatchAssets.pyi at master · Gralerfics/ZeptoWatch (github.com)，以及脚本应用程序示例。

B.5 import pika_lvgl as lv

对 LVGL 的一部分封装，常用功能除了动画都比较齐全了，具体暂略。

详见：ZeptoWatch/pika_lvgl.pyi at master · Gralerfics/ZeptoWatch (github.com)，以及脚本应用程序示例。

B.6 import arm_math_dsp as dsp

引入了 ARM 的 arm_math（DSP 库），封装实数 FFT 等函数，供频谱仪使用，可继续扩充，具体暂略。

详见：ZeptoWatch/arm_math_dsp.pyi at master · Gralerfics/ZeptoWatch (github.com)，以及脚本应用程序示例。