模拟电路实验 diy 项目结题报告——电子陶埙的制作

By 肖星辰 陈薇羽 应逸雯

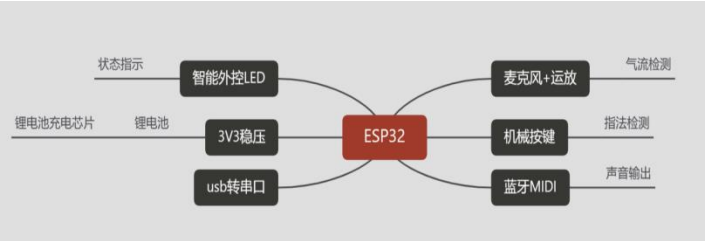
1.项目背景

埙的历史可以追溯到几千年前的中国古代。最早的埙可能是由竹子制成的，后来演变为使用陶瓷材料制作。其原始形态通常是一个圆筒形的管状乐器，有一个吹孔和若干音孔。这些早期的埙在古代中国的文化活动中扮演了重要角色，如祭祀、婚礼和宴会等。随着时间的推移，埙的设计和制作技术不断发展。不同地区和历史时期的埙可能具有不同的形状和音域。有些埙是单声道的，而有些则是多声道的，可以演奏出更丰富的音乐。将传统乐器与现代技术相结合，制作一个轻便有趣的可吹奏的电子陶埙。

2.项目组成（基础部分）

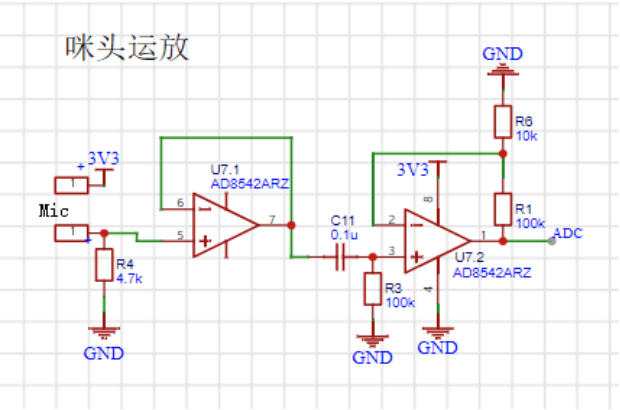
2.1 硬件基本部分

2.1.1 电路拓扑



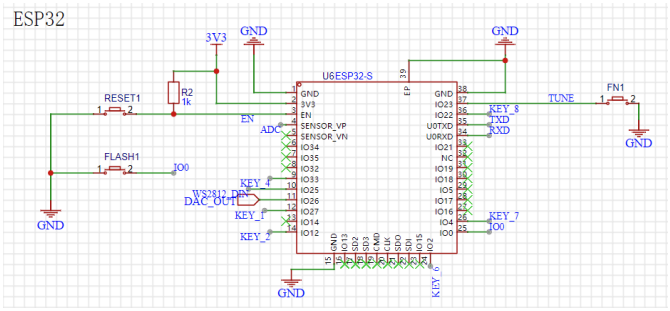
2.1.2 主体部分原理

1.麦克风与运算放大器



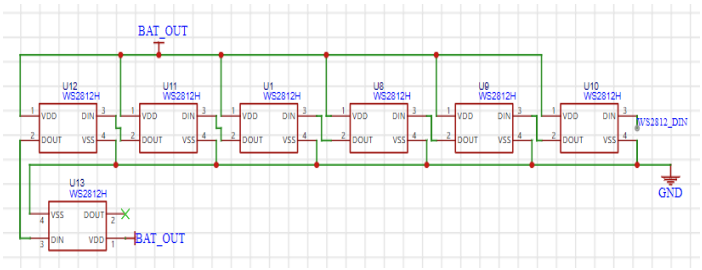
将气流信号转换成微弱的电压信号，第一个运算放大器用作电压跟随，改变线路的输出阻抗，经电容耦合后传输到第二个运算放大器的输入端，第二个运算放大器将信号放大 10 倍后，传输给单片机的 ADC。

2.ESP32 单片机



接收气流产生的电压信号和按键输入的信号，整合成音频信号通过蓝牙通信输出。同时产生不同状态下灯带闪烁的信号。

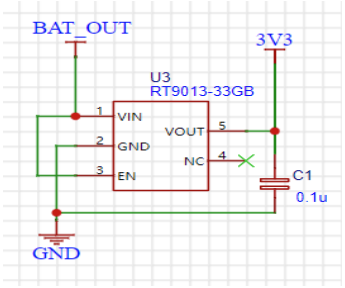
3.WS2812 灯珠



接收单片机产生的信号，将电子埙的工作状态可视化。

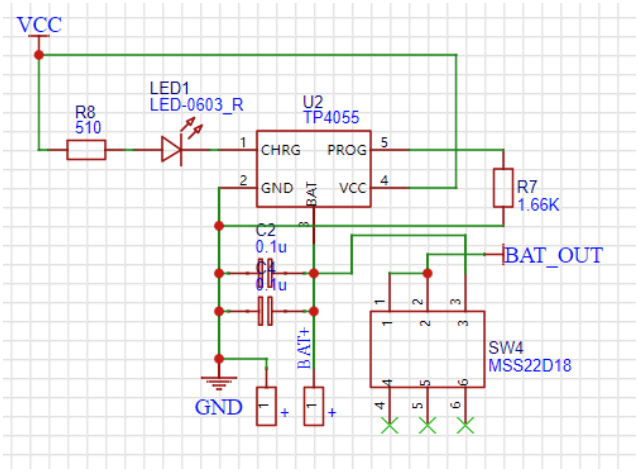
2.1.3 周边电路

1.3V3 稳压电路



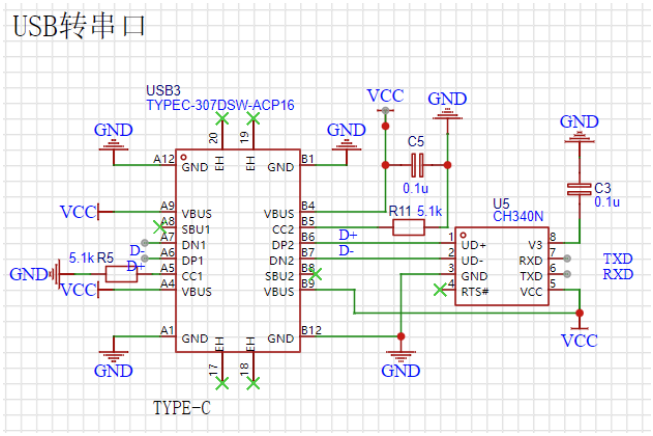
电池-运放，电池-ESP32 各需要一个稳压电路。

2.锂电池充电电路



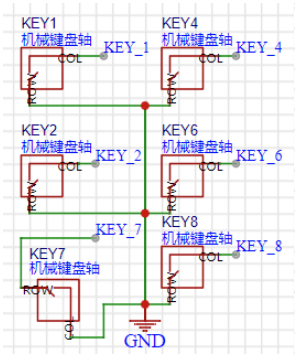
使用锂电池充电芯片进行稳压，使用一颗 LED 显示充电状态。

3.USB 转串口电路

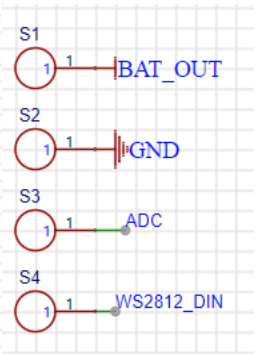


实现 USB（typec)线连接电脑后可以烧录代码。

4.机械按键



5.铜柱



固定上下板位置，与机械部分连接，并实现上下板的电气连接

2.2 软件基本部分-

2.2.1 工程环境

在 VSCode 中编写，使用了 C 语言编写，在 ESP-IDF 中编译

2.2.2 main.c

```
void app_main(void)
{
    esp_err_t ret;

    // Initialize NVS.
    ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK( ret );

    ble_midi_init(); //初始化蓝牙MIDI协议栈
    key_init(); //初始化6个音符按键及2个功能按键，读取设置数据
    adc_mic_init(); //初始化ADC及LED灯
}
```

初始化 NVS、MIDI 协议通信、按键数据、LED 数据、ADC，开启函数的监测循环

2.2.3 key.c

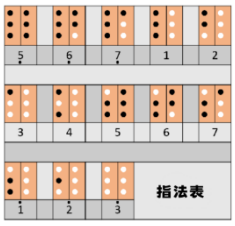
逻辑组合出按键数值，根据指法表匹配 MIDI 编码，得到对应的按键转换为 MIDI 编码的逻辑

组合按键数值：

```
int key_value_list[] = {1,2,4,8,16,32}; //按键的二进制组合

int key_value = 63; //按键全按下为 63

if(gpio_get_level(key_q[0]) == 0) key_value -= key_value_list[0];

if(gpio_get_level(key_q[1]) == 0) key_value -= key_value_list[1];

if(gpio_get_level(key_q[2]) == 0) key_value -= key_value_list[2];

if(gpio_get_level(key_q[3]) == 0) key_value -= key_value_list[3];

if(gpio_get_level(key_q[4]) == 0) key_value -= key_value_list[4];

if(gpio_get_level(key_q[5]) == 0) key_value -= key_value_list[5];
```

根据 MIDI 协议编码和指法表：

```
uint8_t tune_start = FIRST_TUNE;

int8_t tune_q[] = { 0, 2,-1, 4,-1,-1,-1, 5, //指法表
    -1,-1,-1,-1,-1,-1,-1, 7,
    -1,-1,-1,-1,-1,-1,-1,10,
    -1,-1,-1,-1,-1,-1,-1, 9,
    12,14,-1,16,-1,-1,-1,17,
    -1,-1,-1,-1,-1,-1,-1,19,
    -1,-1,-1,-1,-1,-1,-1,-1,
    -1,-1,-1,-1,-1,-1,-1,21}; //定义每一个按键现对于 FIRST_TUNE 的偏移
```



tune_start+tune_q[key_value]即可匹配出按键符合的 MIDI 协议编码

2.2.4 adc_mic.c

初始化 LED

将采集到的电压的值控制为 0-2800；200 以下认为是噪声，统一设置为 0，2800 以上统一设置为 2800；便于后续计算

由电压的值计算亮灯个数，

```
leds_on_count = adc_reading / 360 ;//2800/360 = 7.7 最多 7 个灯亮
```

由电压的值计算音量，

```
midi_vol[4] = adc_reading/22;//2800/22 = 127
```

2.2.5 ble_midi.c

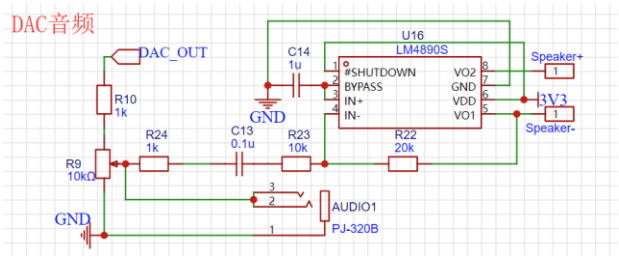
运行一个 BLE GATT 服务器，与 BLE GATT 客户端通信。主要涉及 GATT 服务配置、MIDI 协议配置、BLE 初始化、MIDI 数据发送。

```
//发送 MIDI 指令

void MiDi_Send(uint8_t *notify_data,uint8_t l)

{

  if(midi_gatts_if != 0xff )

   esp_ble_gatts_send_indicate(midi_gatts_if, gl_profile_tab[PROFILE_A_APP_ID].conn_id ,
gl_profile_tab[PROFILE_A_APP_ID].char_handle,

                l, notify_data, false);

}
```

3.自研历程

3.1 硬件

3.1.1 音频播放（未完成）



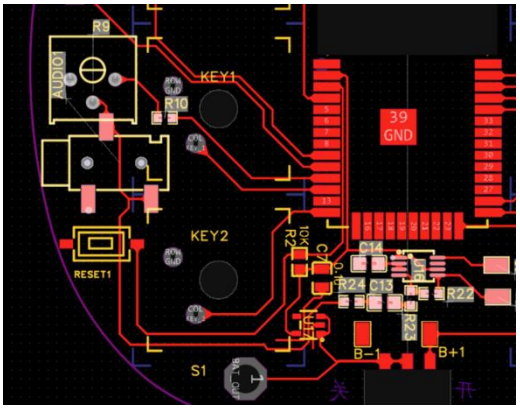
使用 LM4890S 音频功放芯片，输入端连接单片机完成数模转换后的输出口。

进行低通滤波（去除高频噪音）与直流分量的阻隔。

使用电位器调节分压，从而调节声音的输出。

使用音频输入插座，可以外接音频，使音频播放的硬件部分和软件部分可以分开调试。

PCB



图中高亮部分为音频播放电路，已完成布局与布线。但还未完成焊接与代码编写。

3.2 软件

3.2.1 升降调操作

通过更改 first_tune，可以实现从 C 大调到 D 大调或其他声调的转换

tune_start+=1/-=1

操作两次+1 就是 C 大调移动到 D 大调

功能键被按下+第一个指法键被按下——降调

功能键被按下+第三个指法键被按下——升调

3.2.2 升降八度操作

初始设置为 G2-E4，只能弹奏音域有限的歌曲，而整体移动八度即可便捷地适应演奏需要随时扩展音域。通过更改 first_tune，可以整体移动音域。

tune_start+=12/-=12

功能键被按下+第四个指法键被按下——降八度

功能键被按下+第六个指法键被按下——升八度

3.2.3 升降调的灯带显示

为便于观察升降掉命令是否成功执行，增加升降调时的灯带显示，即每一个升降调操作的代码段修改为：

tune_start += 升降数;

ESP_ERROR_CHECK(strip->set_pixel(strip, 第几颗灯, R 亮度, G 亮度, B 亮度));

ESP_ERROR_CHECK(strip->refresh(strip, 100));//刷新 LED

nvs_set_u8(my_handle, "tune_start", tune_start); //保存设置

外置两个 if 函数，检测功能键被按下及基础按键被按下，控制升降调操作

3.2.4 演奏的灯带显示

为演奏时便于演奏者辨析，增加了以不同颜色展示原本的音量显示的功能。

in key.c:

```
if(tune_start == FIRST_TUNE) tune = 0;

if(tune_start > FIRST_TUNE) tune = 1;

if(tune_start < FIRST_TUNE) tune = -1;
```

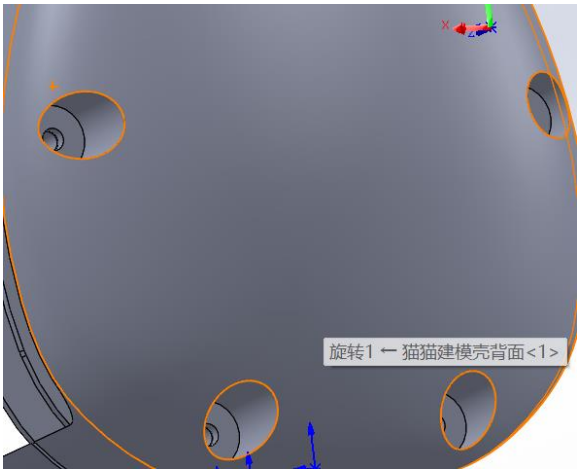
in adc_mic.c:

```
while(leds_on_count --) //吹气气流指示

    {

        if(tune == 0)
ESP_ERROR_CHECK(strip->set_pixel(strip, 6-
leds_on_count, 0, 50, 0)); //标准，绿色

        if(tune > 0)
ESP_ERROR_CHECK(strip->set_pixel(strip, 6-
leds_on_count, 50, 10, 0));//升调，橙色

        if(tune < 0)
ESP_ERROR_CHECK(strip->set_pixel(strip, 6-
leds_on_count, 32, 32, 32));//降调，白色

    }
```

达到效果：演奏时，若存在任意形式的升调，以橙色灯展示音量；若存在任意形式的降调，以白色灯展示音量；若不存在任意形式的基础声调调整，以绿色灯展示音量

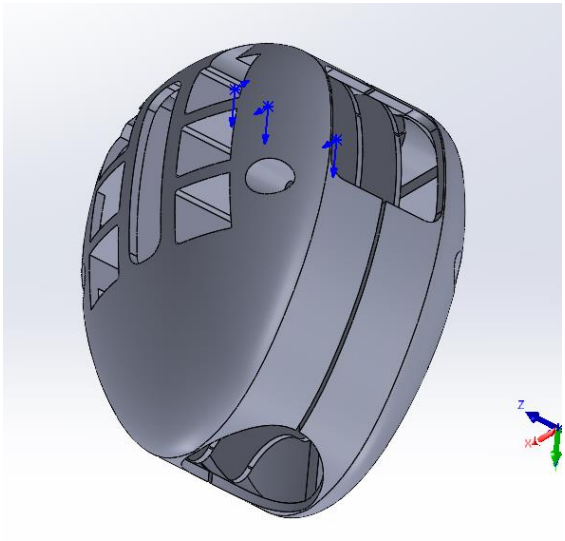
3.3 机械

1.沉孔



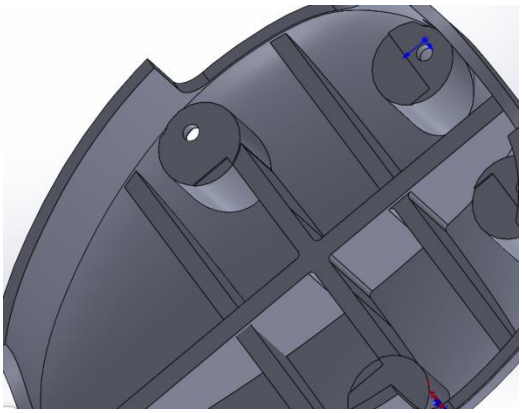
将 VCC 与 GND 对应的铜柱螺帽下沉，防止 VCC 与 GND 短路

2.外壳形状



外壳形状为蛋形，模仿了古代埙的造型，同时提升操作手感

3.结构强化



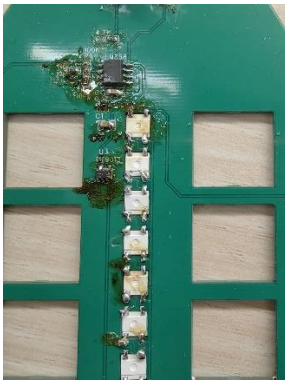
在壳体内加筋，提高整体强度，保证对 pcb 上器件的保护效果

4.主要工作内容及难点

4.1 pcb 绘制

过程：由于本作品没有太高的精度要求，所以在布局与布线时主要考虑元件位置相对合理与不违反 drc。

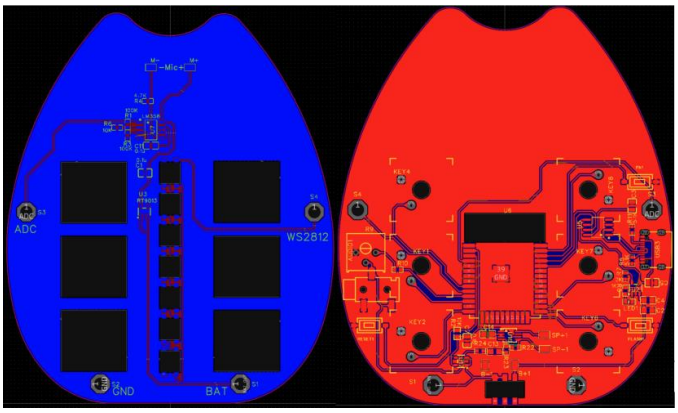
反思：键盘与其他元件（尤其是 ESP32)画在同一面会使键盘焊接完成后难以拆焊其他元件，所以在有外壳保护的情况下可以把位于电路板中间的元件放到另一面。



4.2 焊接

正面板第一版，可能由于虚焊、短路或元件损坏等原因，这一版在进行长时间维修（几乎所有元件换了一遍）之后依然无法正常使用

最终正面板与背面板的效果：



4.3 拓展部分调研与原理图、pcb 绘制

4.4 环境搭建

在 vscode 编译遇到的支持包缺失问题无法解决，最后采用了 ESP-IDF 原生软件编译，由于对 ESP32 不了解，不清楚完整的项目搭建流程，可能项目中存在配置问题，在找到解决方案后即放弃 vscode 编译模式

4.5 代码编写

由于工程代码量大，拆分为了 ble_midi.c、adc_mic.c、key.c 三个文件。基本做到了代码逻辑清晰，注释完整。MIDI 相关配置和蓝牙相关配置看不懂，但复制开源文件和网上参考代码能够运行。除了实现基本发声外，还增加了变调功能和灯带显示功能，使项目更为完整。烧录 bin 文件后，根据效果再反复调整了参数，使灯带显示更为和谐，音高符合调音器。

4.6.保护壳建模

将 pcb 图纸导出为 dxf，然后在 solidworks 中导入为实体，使用检验草图合法性进行连线，查询各零件参数后对 pcb 板进行整体建模转换实体引用确定孔位及相对位置，进行进一步的设计和建模

5.收获

陈薇羽：增加了对电路知识的理解，掌握了用立创 EDA 绘制原理图与 PCB 的基础方法，提升了焊接与电路调试能力

应逸雯：熟悉 ESP32 相关编译知识、MIDI 协议编码、LED 灯带控制（WS2812）

肖星辰：熟悉了 solidworks 的使用，提升了设计、绘图能力，了解了立创 EDA 绘制原理图与 PCB 的基本方法，熟悉了相关的购买和发加工流程