毕 业 设 计 中 文 摘 要

音乐喷泉几乎已经成为了广场装饰的必选方案，其在商业、健康和观赏方面都有很高的价值。在对国内外已经投入应用的音乐喷泉和一些音乐喷泉设计的相关文献进行了研究的基础上，设计了一套基于 STM32 嵌入式实时操作系统的，能够播放音乐并且灯条和水柱高度能够随着音乐变化的系统。在以声音信号的采集和处理、分类为基本功能的基础上，完成了音乐播放、声音采集处理和灯条、水泵状态改变的功能。其硬件系统包括声音采集电路、MP3 音乐播放电路、电机驱动电路、供电电压转换电路、STM32 控制电路和显示控制电路。在软件上采用了 UCOS 嵌入式实时操作系统来提高整个系统的调度效率。该系统实现了播放音乐并对音乐进行采集然后处理、分类并将变化体现到灯条和水泵上的设计要求。整个设计采用 PCB 打板的电路设计，体积小巧，符合移动式音乐喷泉的移动设计要求。

关键词 音乐喷泉 STM32F103C8T6 UCOS 嵌入式实时操作系统 外部中断

毕 业 设 计 外 文 摘 要

Title Design of movable music fountain

Abstract

Musical fountain has almost become a must-chosen plan for square decoration, and it has high value in commercial, health and ornamental aspects. On the basis of researching the music fountain and some related literature on music fountain design that have been put into use at home and abroad, a set of STM32 embedded real-time operating system is designed, which can play music and the height of the light bar and the water column can follow The system of music changes. Based on the basic functions of sound signal collection, processing, and classification, the functions of music playback, sound collection and processing, and status changes of light bars and water pumps have been completed. Its hardware system includes sound collection circuit, MP3 music playback circuit, motor drive circuit, power supply voltage conversion circuit, STM32 control circuit and display control circuit. UCOS embedded real-time operating system is used in the software to improve the scheduling efficiency of the entire system. The system realizes the design requirements of playing music and collecting the music, then processing, classifying and reflecting the changes to the light bar and water pump. The whole design adopts the circuit design of PCB board, and the volume is small and exquisite, which meets the mobile design requirements of the mobile music fountain.

Key Words Music fountain STM32F103C8T6 UCOS embedded real-time operating system External Interrupt

目　录

1　绪论

1．1　课题的研究背景

作为一种具有很高观赏价值的水景艺术——喷泉，越来越多地出现在了我们的身边。例如：城市的广场、公园等公共场所。人们生活水平的提高，也提高了人们对生活环境的要求。这样，一种将音乐欣赏与喷泉水景的观赏合为一体的产物诞生了，它就是音乐喷泉。音乐喷泉的出现丰富了喷泉艺术的表现形式。喷泉与音乐结合，喷泉随着音乐变化。当响起昂扬的音乐时，喷泉“直上云霄”；当响起舒缓的音乐时，喷泉又悠然地流淌。这样观众们就可以借助喷泉的表现力，来进一步地感受音乐所表达的情感。因为音乐喷泉巨大的观赏价值，大型喷泉广场、人工瀑布、超高喷泉、激光喷泉和水幕电影等多种的应用形式都涌现了出来。音乐喷泉在作为人工景观和提供视觉享受上的公用已经得到了人民大众的认可。

喷泉不仅能湿润周围的空气、清除尘埃，还有其他作用。因为喷泉喷射的细小水珠与空气分子碰撞，能产生出大量对人体有益的负氧离子。而负氧离子在低浓度下则有保护环境、利于人的身心健康、让人感到空气清新和治疗一些疾病的作用。负氧离子能使人的大脑皮层抑制过程加强，调整大脑皮层的功能，因此能有镇静、催眠和降压的作用。

模数转换器和麦克风的搭配为我们将音频信号转化为数字信号提供了很大的便利。模数转换器，是把经过与标准量（或参考量）比较处理后的模拟量转换成以二进制数值表示的离散信号的转换器。模数转换器必须有足够的转换精度，才能保证数据处理结果的准确性。同时 AD 转换器还必须有足够快的转换速度，才能适应快速过程的控制和检测。因此，转换精度和转换速度是衡量 AD 转换器性能好坏的主要标志。

现在，在现实生活中应用的音乐喷泉多是坐落于大型广场商业街的大型设备。一般家庭并不能拥有这样的音乐喷泉。而家庭使用就要求音乐喷泉具有较小的体积和较低的成本。故需要设计一款便于移动的小型的移动式音乐喷泉。使用麦克风和模数转换器采集音乐信息，通过单片机分析处理后用水流和灯表现出来就成为了一个可行的方案。

1．2 课题的研究意义

音乐喷泉基本上存在三大意义：

（1） 美化环保功能

在音乐喷泉的各种应用上，首先体现出来的就是美化和环保的功能。一方面，音乐喷泉在设计和播放的音乐类型的方面，有一定程度上体现着建筑者或者是居民的气势和面貌。在商业中心或这是城市广场中，音乐喷泉都起着画龙点睛的装饰作用。另一方面，特别是一些大型的音乐喷泉表演，可以起到保护环境和调节身体健康的作用。空气中的尘埃可以被喷泉喷射的水雾降下，并且湿润周围的空气和降低温度。而且喷泉水经过处理后还可以循环利用，装饰环境的同时也节约了水资源。喷泉的水珠撞击空气产生的负氧离子，对人体健康也很有帮助。

（2）艺术功能

通过查阅相关资料，可以大致上发现，哪些具有生命力似的，工艺简介并且具有经典艺术造型的音乐喷泉能够像艺术品一样长青不衰，甚至可以写入教材作为经典案例供学生门学习和钻研。音乐喷泉那随着音乐的光芒和高潮，构成了由声、光、电、水、火结合的视觉盛宴。音乐喷泉的设计公司需要不断的推陈出新和积累扎实的景观施工实践经验才能获得良好的艺术效果，赢得用户的青睐，才能推出与景观协调统一，与音乐喷泉的功能和艺术感统一的作品。

（3）商业价值

音乐喷泉除了具有喷泉表演的美化环保功能和艺术效果功能外，还存在着不错的商业价值。水景喷泉不仅能靠一场水秀表演来愉悦本地市民，而且一个经典的音乐喷泉还能吸引来游客，给游客一种必须身临其境才能有所体验的体会。在音乐喷泉所展示的舞台艺术的同时，也提高了城市形象。音乐喷泉的表演还能够给前来城市投资的人们留下对城市鲜活的印象。音乐喷泉周边常常驻足的观景人流为其周边地带带来了潜在的消费能力。这里可以举两个例子，泉州浦西万达广场的激光音乐喷泉工程和广西全州县人民广场的音乐喷泉项目。这两个项目都是在城市举行重大的招商引资项目或者开展各种各样的展览会的同时，进行音乐喷泉水秀表演。

1．3 国内外研究现状

在中国 80 年代以前，喷泉只是建筑给排水和景观设计的一个技术细节，工程安装从设计到产品生产都没有形成规模。只有在展览馆等公共建筑和公园里，才能看到一些小喷泉，产品的层次简单而粗放。20 世纪 80 年代后，随着国民经济的恢复和发展，人民生活水平的提高，一些专业的喷泉设备加工厂和公司开始成立和发展，为喷泉的专业发展提供了力量和基础。20 世纪 90 年代以来，社会主义市场经济发展迅速，中国城市建设速度加快。喷泉已经走出公园、花园和广场中心的围墙，延伸到人们各种社会活动的场所。这一时期出现了许多专业喷泉制造商和专业公司，都得以发展迅速。专业技术人员将计算机技术、信息技术以及声、光、电、雾、火、激光、音乐、水幕电影等学科应用到喷泉中，将喷泉行业的技术水平推上了一个新的台阶，开辟了广阔的应用空间，喷泉行业发展迅速。2001 年，中国建筑金属结构协会给排水设备分会成立了水景喷泉专业委员会。一些大学、中专也开设水景、喷泉等课程，为喷泉行业未来的发展提供知识储备。喷泉产业的发展进入了一个新的阶段。

对于目前的喷泉控制系统，单片机控制继电器的接触控制和 PLC 控制是国内外多采用的控制技术。例如，使用 PLC 与可控硅实现的方案。这种方案以 PLC 的 CPU 模块作为程序运行中心，使用可控硅通过可控硅调速控制电机的转速。可控硅调速的原理是用改变可控硅导通角的方法来改变输出端电压的波形，从而改变输出端端电压的有效值，以达到调速的目的。再辅以麦克风、AD 转换器件和灯光设置来实现音乐喷泉的效果。

我国一直以来是一个缺水的国家，节约用水是基本国策。所以喷泉应该提倡循环用水。并且设计时应该尽量避免喷水的雾化与飘移的损失。循环用水系统也应当有过滤净化装置。老式的旱喷泉，喷出的水很容易把泥土脏物带入地下水池。搞好水池净化，能够延长池水的使用周期。池水还可以浇灌植物，既节约用水又有环保功能。

1．4　课题的技术构成

本设计以 STM32F103C8T6 作为系统的控制中心，使用 STM32 自带的逐次逼近型 AD 转换器来将模拟声音信号转变为适合 STM32 处理的数字电平信号。

STM32F103C8T6 是一款基于 ARM　Cortex-M3 内核的 32 位微控制器(ARM 公司以 ARM11 命名为 Cortex，分 a、r、m，m 系列有 M0、M0、M3、M4、M7)。它采用 LQFP48 封装，由意法半导体公司(ST)推出，属于 STM32 系列。其程序存储器的闪存容量为 64KB(64K　x　8 位)，随机存储器容量为 20KB(20K　x　8 位)，两个 12 位模数转换器共有 12 个通道(外部通道只有 PA0 至 PA7 和 PB0 至 PB1，不是 18 个通道)。共有 37 个通用 I/O 端口(PA0-PA15、PB0-PB15、PC13-PC15、PD0-PD1)、4 个 16 位定时器(TIM1(带死区时间插入的高级控制定时器，常用于产生 PWM 控制电机)、TIM2、TIM3 和 TIM4)、2*IIC 和 2*SPI 工作温度为-40C　～ 85C，系统时钟最高可达 72MHz(通常，8MHz 的外部时钟由锁相环 9 倍频至 72MHz）。

本设计使用 UCOSIII 来调度系统的代码。

UCOS 有两个版本，UCOSII 和 UCOSIII，他们都是 Micrium 公司推出的 RTOS 类实时操作系统。UCOSIII 是一个没有任务数限制的可裁剪、可剥夺型的多任务内核。UCOSIII 提供了实时操作系统所需的所有功能，包括同步、任务通信、资源管理等。UCOSIII 是用 C 和汇编来设计的，只有极少数的与处理器密切相关的部分代码才是用汇编写的，其中绝大部分都是用 C 语言编写的， UCOSIII 结构简洁，可读性很强。最主要的是非常适合初次接触嵌入式实时操作系统的学生、嵌入式系统开发人员和爱好者学习。

音乐喷泉的基本原理可以理解为对声音信号的采集和分类，然后通过灯条和水柱高度实现具有观赏价值的可视化。由此，模拟信号向数字信号的转换是这个系统设计的开始也是系统设计的关键。本设计采用 STM32F103C8T6 自带的逐次逼近型 AD 转换器实现模拟信号对数字信号的转化。逐次逼近型 AD 转换器的原理可以理解为“天平称量”。先使用质量最大的“砝码”称量，如果砝码的质量大于被测物体则记录 0，换用质量小一个等级的砝码重复步骤；如果砝码的质量小于待测物体的质量则记录 1，在已经加上砝码的基础上继续加砝码，重复测量步骤，直到记录满所有的记录位。如此一个模拟信号就转化为了只有高低电平的数字信号。

2 移动式音乐喷泉总体方案设计

2.1 音乐喷泉设计要求分析

本设计主要利用 STM32F103C8T6 作为核心控制器。通过麦克风采集设备播放的模拟声音信号，传输到 STM32 的 ADC 采集端口，经过 STM32 内置的逐次逼近型 AD 转换器转换为适合单片机处理的数字信号。单片机对数字信号的值进行分类，然后不同的数字信号分类对应不同的灯条状态和水泵功率。如此完成音乐喷泉的基本功能。另一方面，通过 STM32F103C8T6 检测外部按键的中断信号来控制 MP3 模块和显示模块。实现音乐的选择和播放。

本次毕业设计课题任务的内容和要求主要如下：

主要内容：

（1）对基于单片机的移动式音乐喷泉有总体认识；

（2）学习 STM32 的使用和音频处理的基本方法；

（3）熟悉使用 Keil 编程工具以及 Proteus 电路仿真软件；

（4）完成音频声音采集显示和喷泉高度变换功能；；

（5）调试并完善功能；

技术要求：

（1）能够完成喷泉水型随音乐的高低旋律发生变化；

（2）能够实现用彩灯表示音乐大小情况；

（3）能够实现彩灯、水型与音乐节奏的同步变化

另外，还有工作要求为设计出实物。

通过对以上要求的分析和对资料的调查汇总，得出了以下设计计划：

（1）选择以 STM32F103C8T6 作为整个设备的核心控制器。STM32F1 带有 ADC 转换功能，使用带有此功能的 STM32 单片机正好可以满足将模拟声音信号转换为数字信号的需求且不会增加外部电路的复杂程度，对于“移动式”的要求也有正面的效果。且进过查询相关资料，其他音乐喷泉的设计大多采用 STC89C52 之类的单片机进行设计，采用 STM32 进行音乐喷泉的设计也具有一定的新颖性。

（2）本次设计选择使用嵌入式操作系统 uCOSIII 作软件程序的调度。使用 uCOSIII

嵌入式实时操作系统不仅增加了设计的新颖程度，而且可以依靠 uCOS 的实时性优势来提高系统的工作效率，达到高效的目的 。

（3）本次设计还计划采用使用印制电路板（PCB）加功能模块的方式设计整个系统电路。PCB 便于使用贴片器件，功能模块的设计往往紧凑，将这两种情况结合将有利于压缩成品实物的体积，以达到体现“移动式”的要求。

（4）电源接口计划采用目前在国产 Android 移动设备上应用普遍的 microUSB 接口作为电源输入口。这种接口可以提供 5V 的电压、体积小巧且在生活中几乎随处可见。

2．2　音乐喷泉设计方案

该设计的流程图 2-1、硬件组成图 2-2 和软件设计图 2-3 如下图所示：

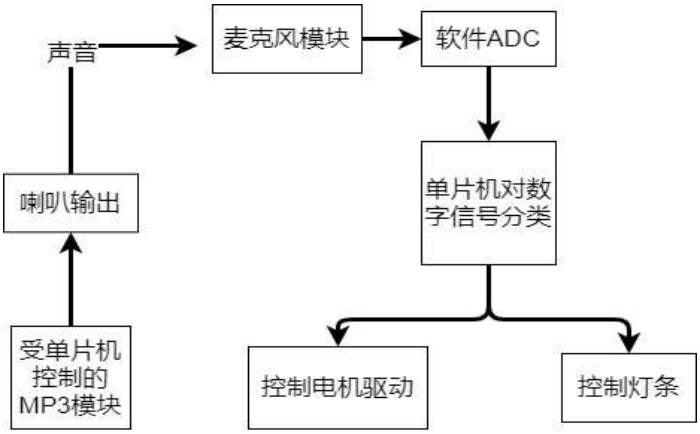


图 2-1 音乐喷泉流程图

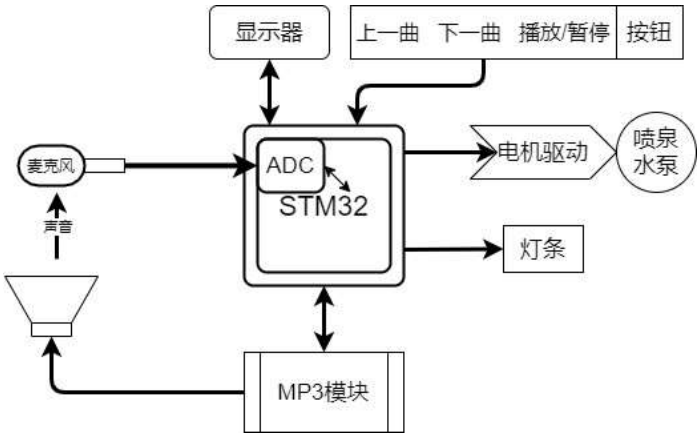


图 2-2 音乐喷泉硬件组成图

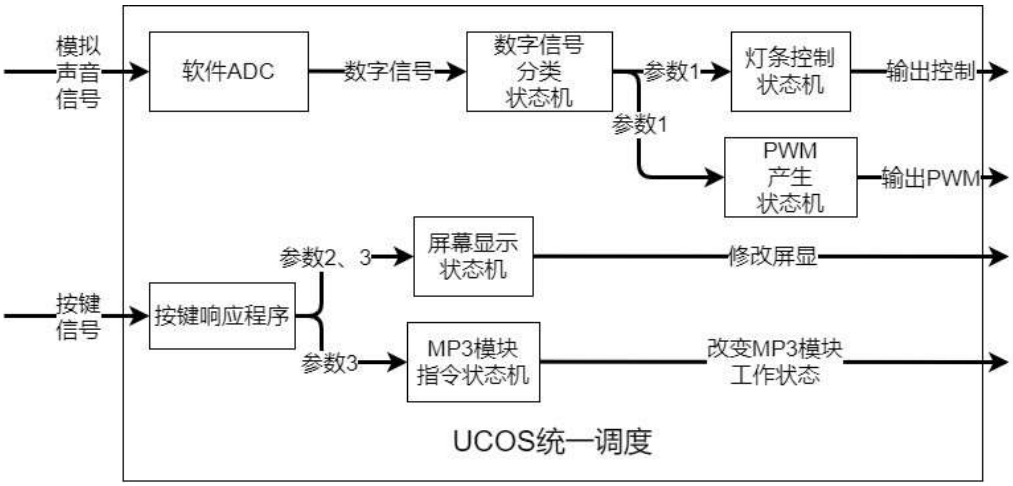


图 2-3 音乐喷泉软件设计图

本设计计划使用 STM32F103C8T6、麦克风模块、直流电机驱动模块、OLED 显示模块、喇叭、水泵和其他外围电路组成硬件实物。由 STM32 通过响应外部按键中断控制的MP3模块模仿音乐，再通过带有声音信号采集和基本处理功能的麦克风模块采集后，声音信号将传送给 STM32 进行模拟信号转变为数字信号的初步处理，然后 STM32 根据数字信号的强度将源源不断传来的数字信号分类，一边分类，一边修改驱动水泵的 PWM 波的占空比和外接灯条的 IO 口的状态。当外部按键被按下后，STM32 将按照约定的按键意义修改屏幕的显示和 MP3 模块的工作状态，并将状态的改变显示在屏幕上。

2．3　音乐喷泉设计理论依据

音乐喷泉的设计理论依据主要来自三个方面：一个是声音信号的传播，另一个是模拟信号向声音信号的转化，还有离心水泵的工作原理。

声音是以波的形式传播的。由于物体的振动，才能产生声音，声音是物体振动产生的波动，声音的波动要靠介质才能传播。在磁场中运动的线圈带动喇叭表面的膜振动，膜振动发出声音，喇叭附近的空气作为介质传播声波的振动。如此，声音才能够从喇叭传送到麦克风。麦克风的工作原理与喇叭相反。声波传来，空气介质的波动带动麦克风中线圈的运动，线圈在磁场中运动将振动的信号转化为变化的电信号。由于空气介质的存在，声音才能够从喇叭传递到麦克风。由此，声音从源传播到了探测器。

模拟信号向数字信号转化的技术是音乐喷泉能够实现的另一个依据。模拟信号向数字信号的转化技术有很多种，这里重点介绍逐次逼近型模数转换器。

离心水泵在工作时会将水泵的泵壳内注满水。水泵工作时，水泵的叶轮会高速旋转，

液体在离心力的作用下会产生很高的速度。泵壳内与叶轮之间的空间是逐渐变大的，高速的液体流过逐渐扩大的空间，便获得了喷出水泵出口的压力。如此一来，调节水泵的叶轮转速就可以达到调节水泵内液体压力的目的，液体的压力变化，液体喷出水泵的高度就会发生变化。

3　硬件设计

3．1　系统基本组成

本系统由电源供电和降压电路、STM32 最小核心电路、LED 灯条电路、按键控制电路、MP3 音乐播放模块、麦克风模块、电机驱动模块和显示模块组成。其电路整体原理见图 3-1 及 PCB 图 3-2 和图 3-3。以此设计并制作出了一种以 STM32F103C8T6 为核心控制器的音乐喷泉设计方案。STM32 控制 MP3 播放选定的音乐，通过麦克风模块采集模拟声音信号，然后交由 STM32 内部的 AD 转换电路将模拟信号转变为数字信号。STM32 对采集转化来的信号进行分类，然后依据分类来改变电机的转速和灯条的显示状态。

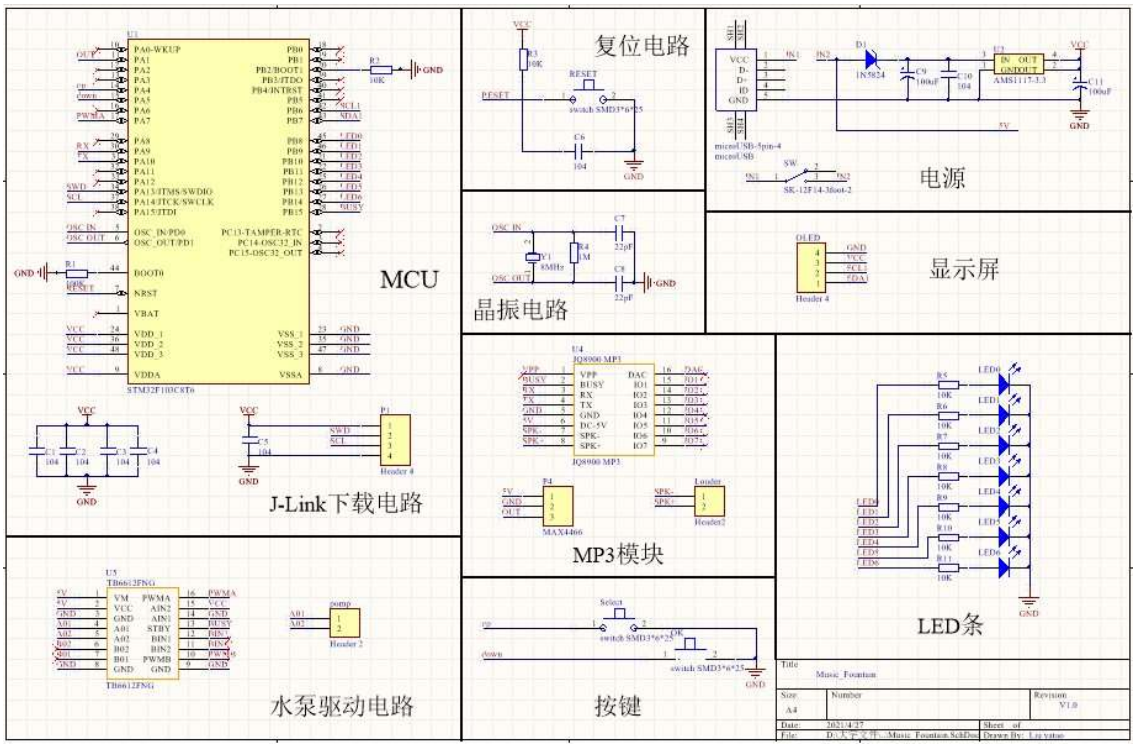


图 3-1 音乐喷泉整体电路原理图

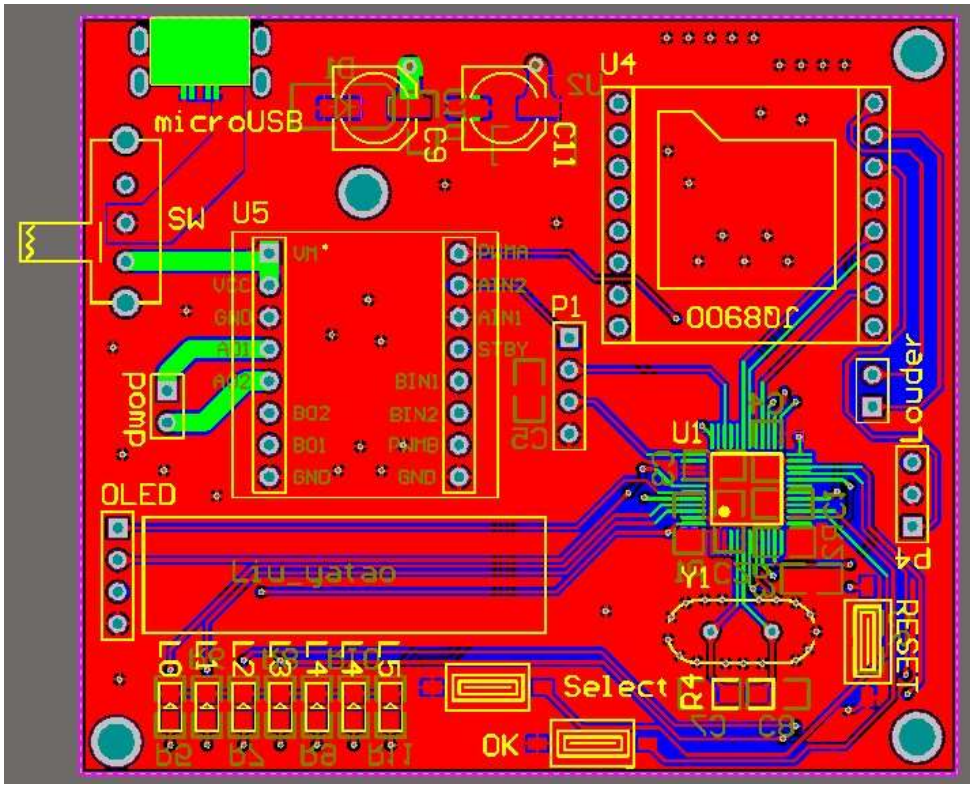


图 3-2 音乐喷泉 PCB 正面 2D 图

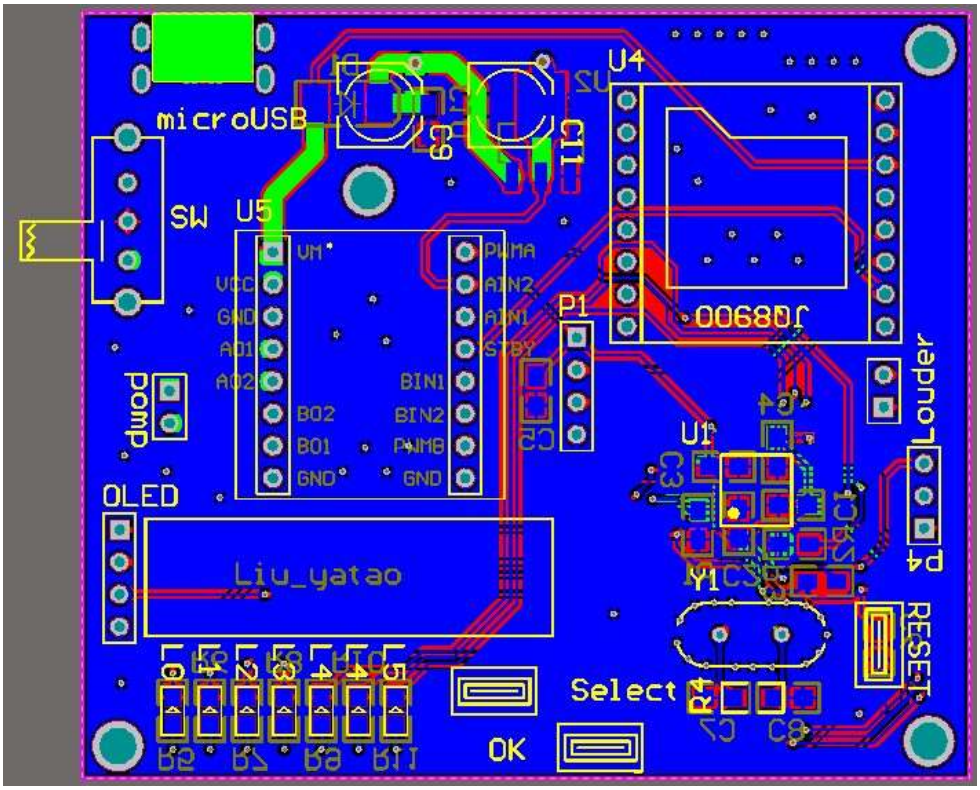


图 3-3 音乐喷泉 PCB 背面 2D 图

3．2　MCU 处理器的选择

微控制单元(Microcontroller Unit；MCU)，又称单片微型计算机(Single Chip Microcomputer)，是把中央处理器(Central Process Unit；CPU)的频率与规格做合理缩减，并将 USB、A/D 转换、UART、PLC、内存(memory)、计数器(Timer)、DMA 等周边接口都整合在单一芯片上，形成芯片级的计算机，可以对不同的应用场合做不同组合控制。例如智能手表、电脑外设、控制器，至汽车电子、工业上的步进电机、机器手臂的控制等等，都可见到 MCU 的应用。MCU 也是进行嵌入式开发的核心部件，而嵌入式系统是一个很广泛的概念，主要是相对于计算机而言，是一种功耗受限、尺寸受限的特殊类型计算机，小到智能手环、大到智能手机，都可以统称为是嵌入式开发系统。

本次设计选择了STM32F103C8T6作为核心控制器来设计音乐喷泉。其与STC89C52对比具有很多优势，如表 3-1 整理的对比。如下表所示，为了达到搭载操作系统和运行速度快的要求，选择使用 STM32 作为控制中心。

表 3-1 STM32 与 89C51 对比

	STM32	89C51
内核	Cortex-M3,32Bit@72MHz	8Bit@2MHz Max(分频后)
整数计算能力	1.25DMIPS	0.06DMIPS
地址空间	4GB	64KB
片上存储器(ROM)	20K-1MB	2K-64K
RAM	8K-256K	128B-1K
外设	AD,DA,Timer,WWDG 等	Timer,USART
经典开发工具	UV4(标准 C 编译器)	UV2
操作系统	uClinux,uCOS 等	基本不运行系统

另外，经过前期的材料查询了解到，部分音乐喷泉采用 PLC 的控制实现。因为设计要求对体积有要求和对 PLC 不够熟悉，所以在 PLC 与 STM32 之间，选择了 STM32 进行控制的设计方案。STM32 核心电路对应的 PCB 位置图 3-4 和设计原理图 3-5 及如下所示。其中 C1, C2, C3, C4 作为滤波电容分布在距离 STM32 管脚最近的位置。

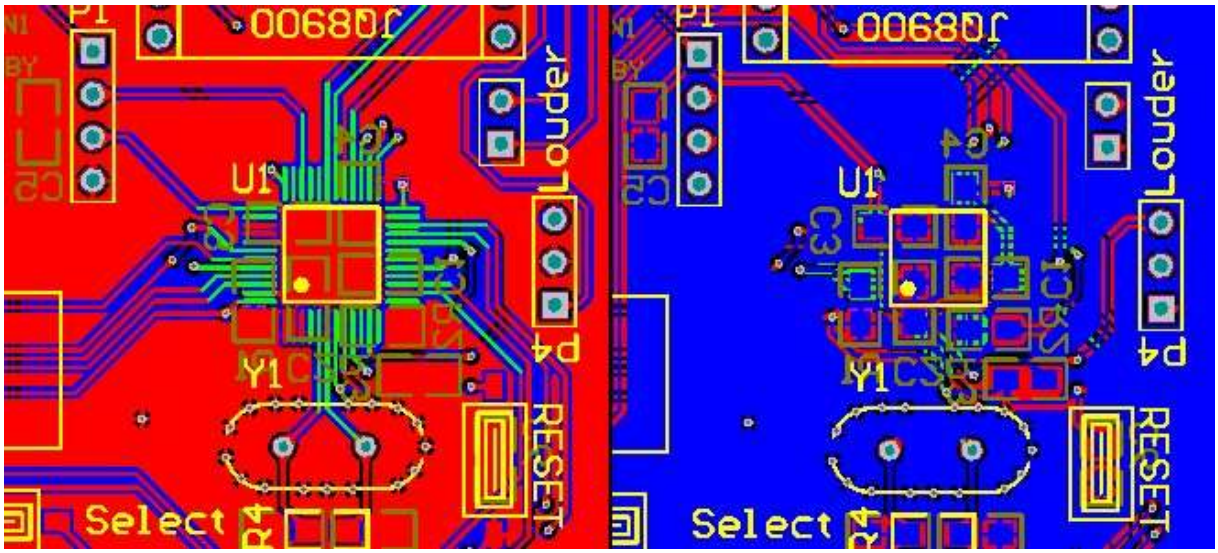


图 3-4 STM32 核心部分 PCB 图

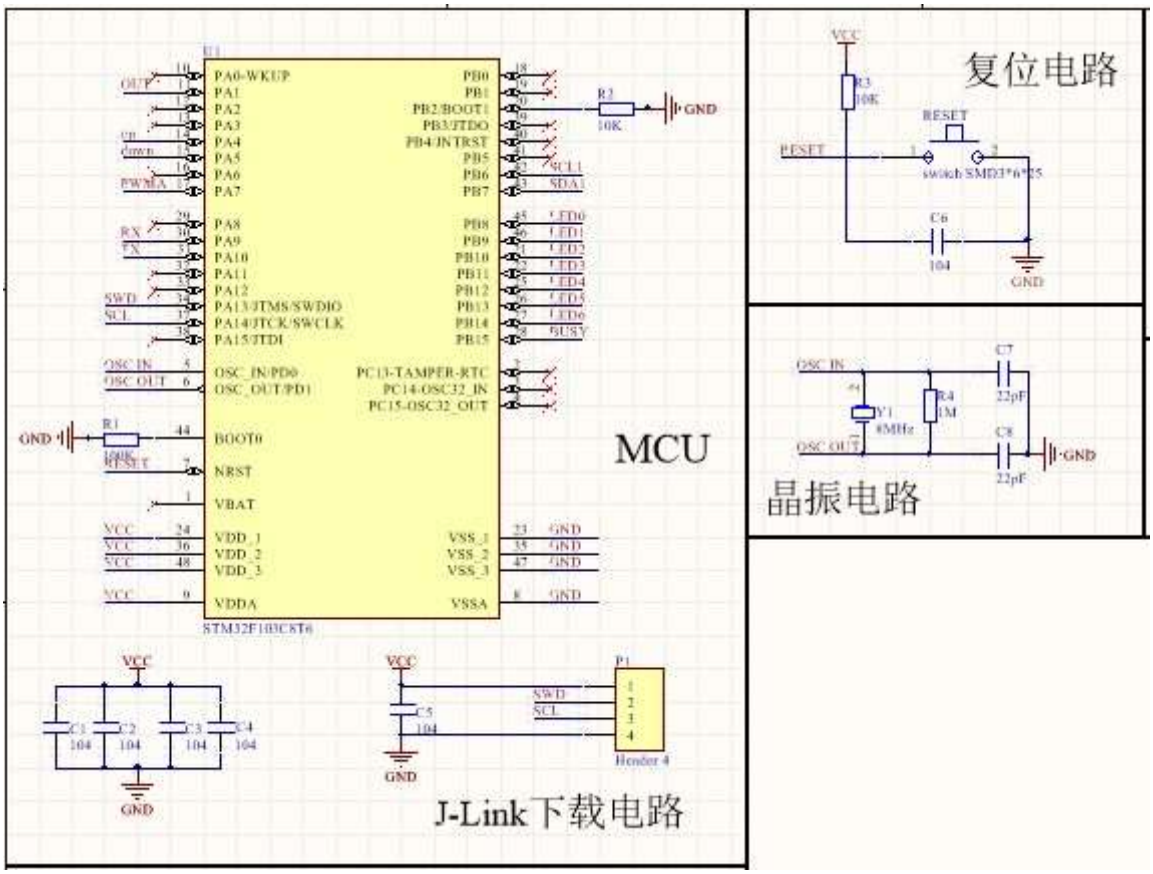


图 3-5 STM32 核心部分原理图

3．3　声音采样电路模块的设计选择

声音的采集是整个设计电路信号部分的开始，也是电路搭建中最重要的部分之一。电路能否采集到波形正确的信号决定了后续电路所处理的信息的价值。因为手动焊接的电路极有可能在元件的连接处出现尖端等不利于信号传递的缺陷，所以本次设计使用了基于 MAX4466 的设计好的麦克风模块。

MAX4466 可以说是微功率运放的最佳选择。MAX4466 可用于麦克风前置放大器。MAX4466 以超小的封装提供了优化的增益带宽与电源电流的理想组合。MAX4466 还具有优异的达到 112dB 的电源抑制比和 126dB 的共模抑制比。MAX4466 的增益稳定性，仅需 24uA 的电流即可提供 200KHz 的增益带宽。此外，其还有轨到轨输出，高 AVOL，适合在嘈杂的环境中工作。MAX4466 有三个输出接口，分别为 Vcc、GND 和 OUT，使用简单。

表 3-2 MAX4466 引脚功能图

名称	作用
Vcc	模块电源正极（支持 3.3V 到 5V）
GND	电源模块地
OUT	模拟信号输出

MAX4466 的经典应用电路图 3-6 和麦克风前置放大器模块图 3-7 如下所示。

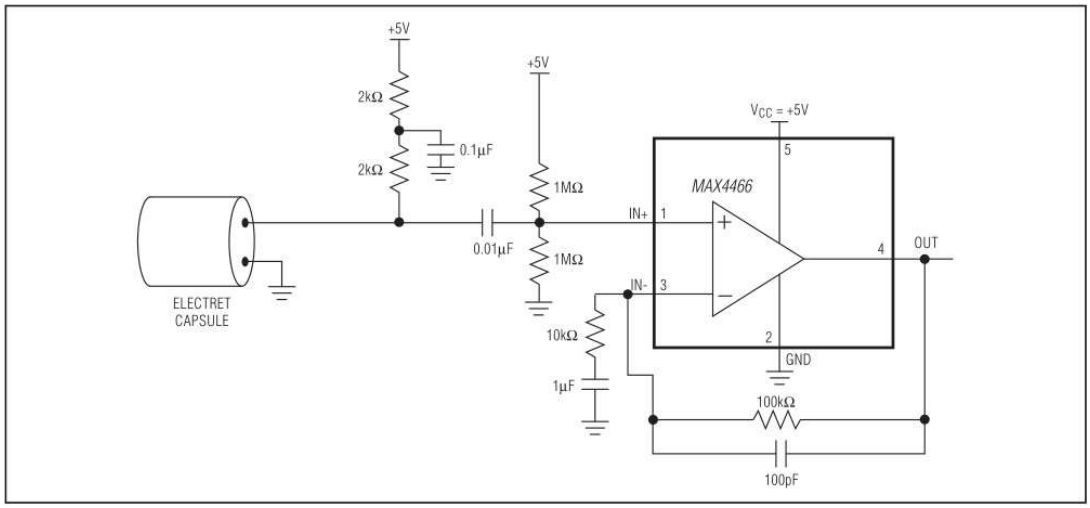


图 3-6 MAX4466 经典应用电路

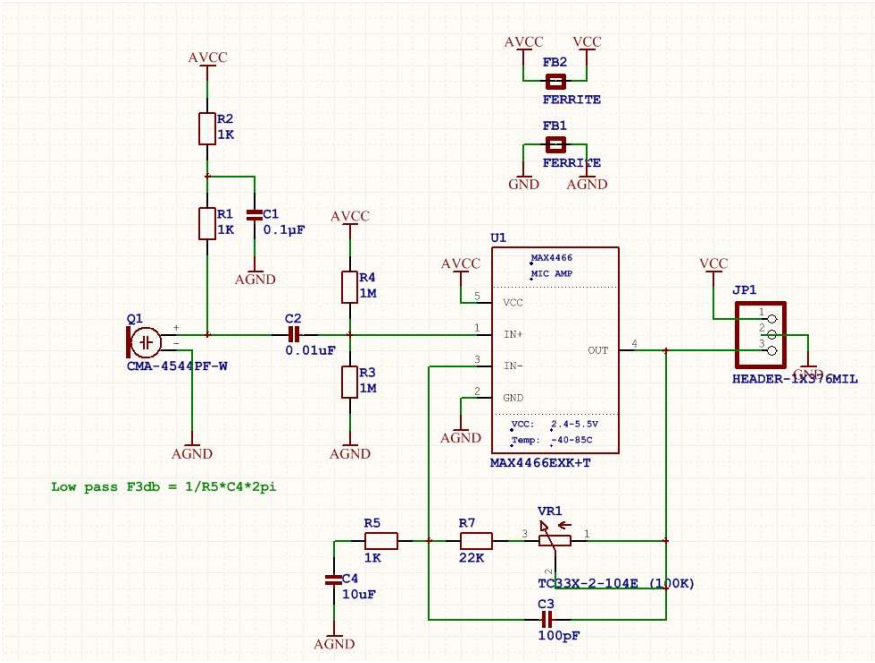


图 3-7 MAX4466 麦克风前置放大器模块原理图

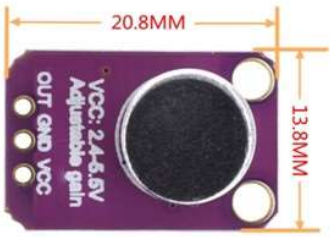


图 3-8 MAX4466 麦克风前置放大模块

3．4　MP3 模块的设计选择

音乐播放是音乐喷泉的必备功能。虽然可以借助 STM32 搭载文件系统与外部电路结合自己设计 MP3 电路，但是考虑到要保证 STM32 的性能主要分配到 ADC 和自制焊接电路的焊接处的瑕疵对交流信号的影响，最终还是选择了 MP3 模块。

本设计最终选择 JQ8900-TF 语音模块作为 MP3 模块。JQ8900-TF 支持 MP3 硬件解码、多种控制模式（两线串口、一线串口和按键）、支持 TF 卡、HID、模拟 U 盘、支持“上一曲”“下一曲”“播放”“暂停”等常用指令功能、音量可调和支持专用的 BUSY 信号输出，BUSY 信号有利于用户判断模块的工作状态。其可以用于：车载导航语音播报、公路运输稽查、收费站语音提示、车站安全检查语提示、营业厅语音提示、机电设备自动报告故障和定时广播等场景。

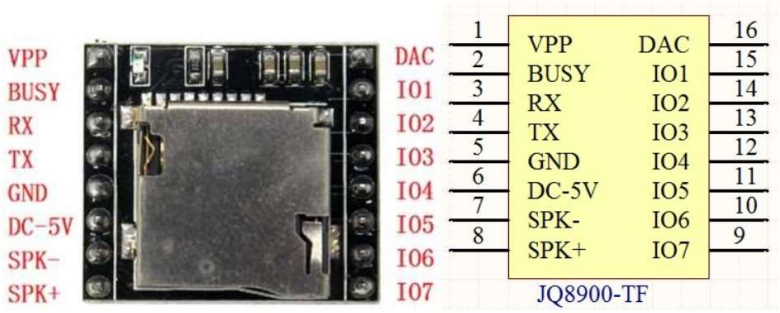


图 3-9 JQ8900-TF 模块及引脚图

本次设计并不需要 JQ8900 的所有外设引脚，设计用到的引脚及功能见表 3-3。

表 3-3 JQ8900-TF 引脚定义表

名称	功能
BUSY	音乐播放状态输出
RX	串口信息接收引脚
TX	串口信息发送引脚
GND	模块电路低电平引脚
DC-5V	模块 5V 供电引脚
SPK-	接 3W 喇叭负极
SPK+	接 3W 喇叭正极

为了避免外界的干扰，可以使用 JQ8900-TF 自带的 BUSY 引脚的电平信号来控制其他模块的工作状态。设计中将 JQ8900 的 BUSY 引脚接到电机驱动模块的使能端，达到 MP3 播放音乐水泵才能够工作的目的。JQ8900-TF 在 PCB 上的电路设计如图 3-10。

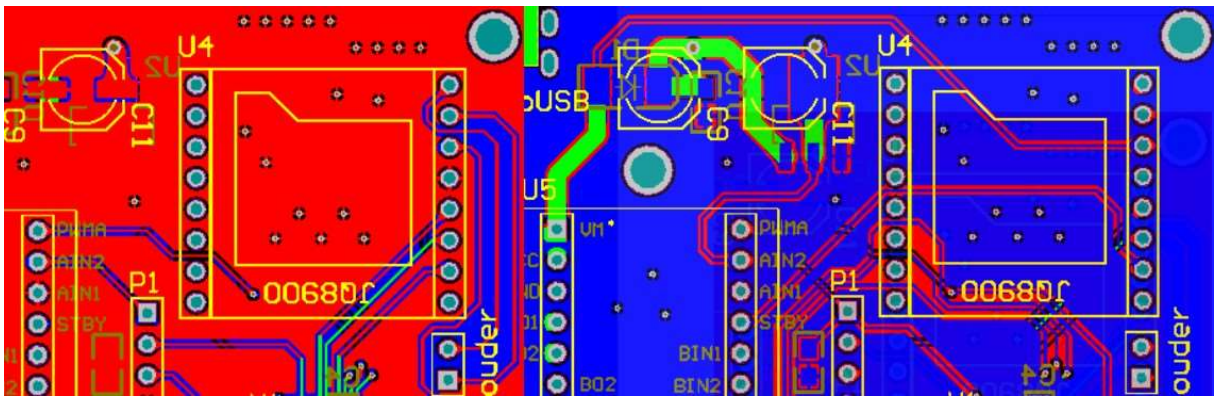


图 3-10 JQ8900PCB 设计图

3．5　水泵驱动电路模块设计选择

水泵驱动电路的作用是在 STM32 的控制下改变输出的功率以实现改变水泵的功率进而改变水柱高度。本次设计使用了 TB6612FNG 电机驱动模块作为水泵的驱动电路。TB6612 体积扁平，小巧却有很高的性能，这正好符合移动式音乐喷泉的“移动”要求。

TB6612FNG 拥有两路电机驱动，每路驱动可以单独设置电机转动方向。STM32 输出的占空比可调的 PWM 波作为 TB6612FNG 的输入，TB6612 的驱动能力与 PWM 波的占空比成正比。TB6612FNG 还具有 STBY 这一使能端（高电平使能），通过连接 JQ8900 的 BUSY 引脚到 TB6612 的 STBY 引脚以达到 TB6612 的驱动受 JQ8900 控制的目的。本次设计只使用了 TB5512FNG 的部分引脚，使用到的引脚及功能如表 3-4 所示。

表 3-4 TB6612FNG 引脚及定义

名称	功能
VM	10V 以内供电引脚
VCC	5V 供电引脚
A01/A02	驱动输出引脚
PWMA	PWM 输入引脚
STBY	高电平使能驱动引脚
AIN1/AIN2	决定正反转引脚

TB6612FNG 的电路设计原理图 3-11 及 PCB 图 3-12 如下所示。

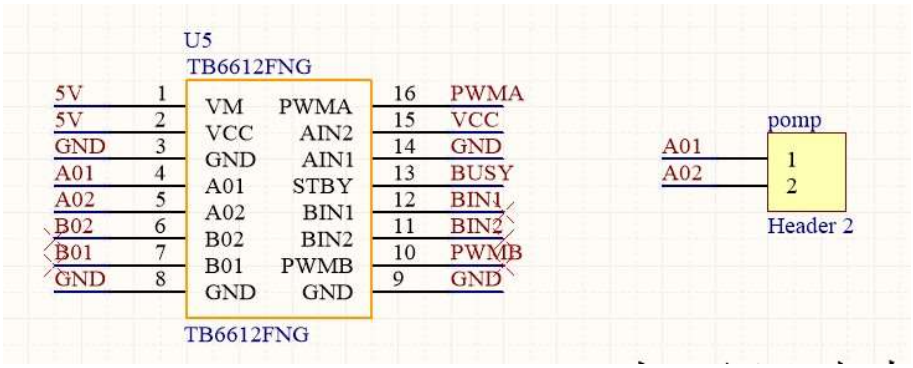


图 3-11 TB6612FNG 水泵驱动电路原理图

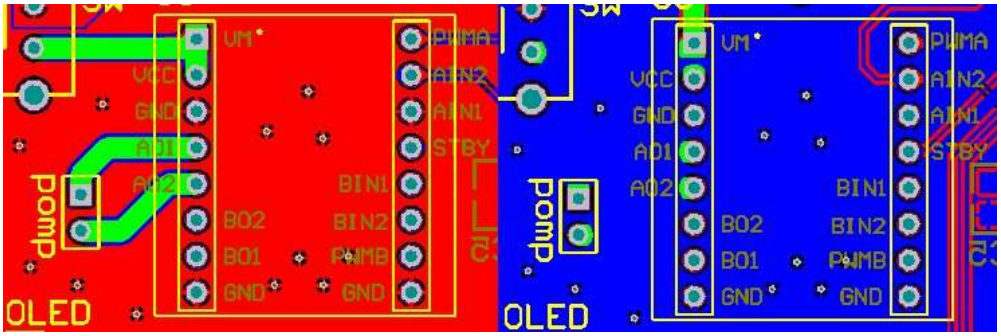


图 3-12 TB6612FNGPCB 设计图

3．6　显示电路模块设计选择

本次设计的设计要求有播放音乐的条目，为了使用的友好性，本次设计应该具备一个简单的用户界面来完成与用户的互动。所以本次设计采用了两个按钮加一个显示屏的设计来完成与用户的交互功能。两个按键分别有不同的功能：一个按键负责轮流选中不同的菜单条目，另一个按键负责发出确定执行此功能的信号。按照以上的设计思路，本次设计采用了 0.91 寸 4 针 OLED 显示屏。该显示屏体积小巧且足够显示本次设计所需的全部内容，仍然十分匹配本次设计的“移动”要求。

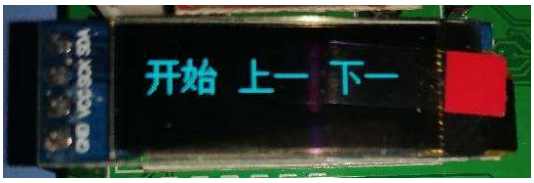


图 3-13 OLED 显示模块实物图

OLED 显示模块的原理图 3-14 及 PCB 图 3-15 如下所示。

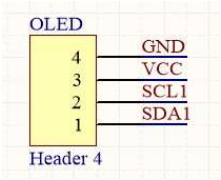


图 3-14 OLED 显示屏原理图

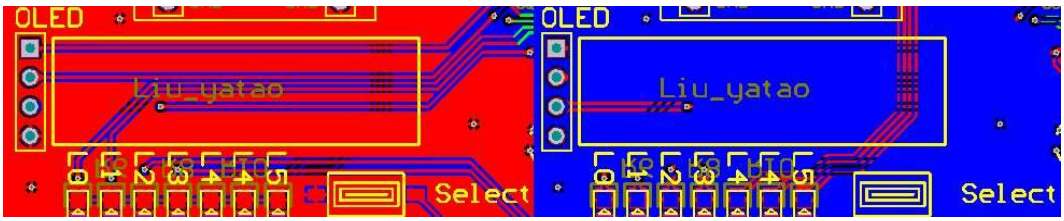


图 3-15 OLEDPCB 设计图

经过以上设计得出的 PCB 实物图 3-16 和硬件装配后的实物图 3-17 如下图所示：

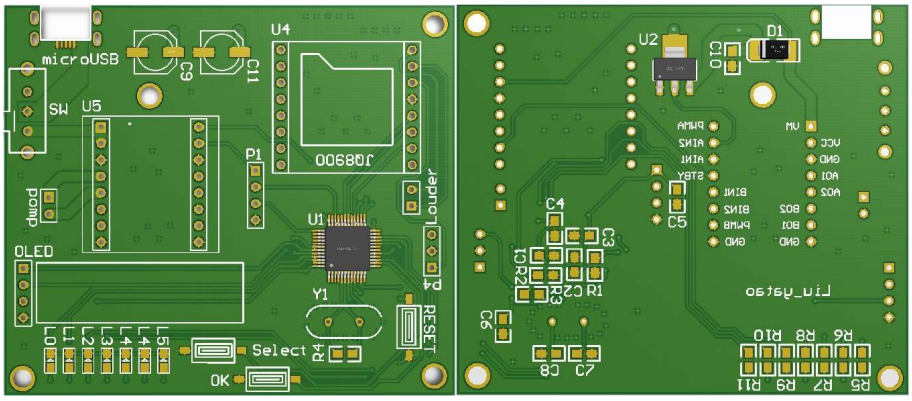


图 3-16 PCB 仿真实物图

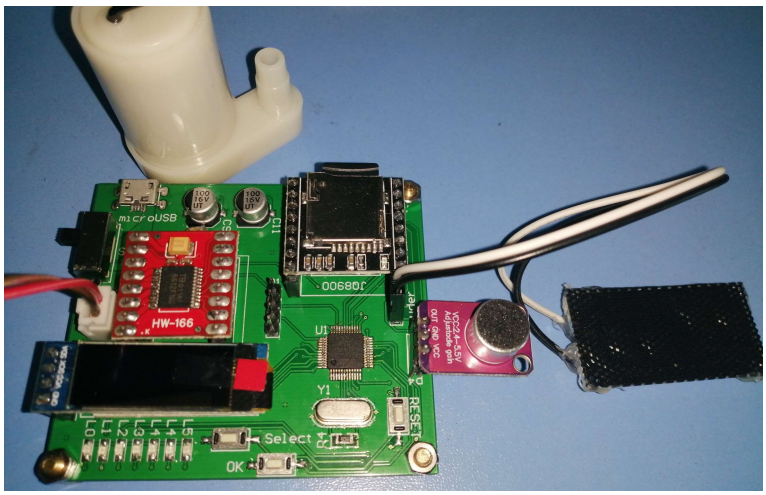


图 3-17 硬件装配图

4 软件设计

4．1 软件设计方法

如今单片机的种类更加的丰富，使用单片机的范围也更加广阔，几乎很难找到哪个没有单片机身影的领域。单片机可以在智能仪表、工业控制、家用电器、计算机网络通信、医疗设备和大型电器内应用，几乎涵盖了生活的方方面面。当然，只有单片机是不能够彻底完成以上应用的，单片机还要和软件程序一起才能够解决实际问题。在单片机的编程语言中应用最广泛的应当是 C 语言，其次是汇编语言，近年来使用 Python 编程单片机的方法也发展了起来。机器语言是最底层的语言，机器语言指挥着控制器工作，汇编语言作为机器语言的助记符，让代码的编写变得更加直观。C 语言如今已经是很成

熟的语言了。它的各种语法规则、思想都已经确立了起来。C 语言也对其他语言有着很大的影响。

C 语言具有三大优点。第一，C 语言的代码量小。这里以 Microsoft 的 office 和 WPS 做对比。Microsoft office 的安装文件大小往往在 1GB 以上，而 WPS 的安装文件只有大概 50MB。这里的原因在很大一部分上是因为 WPS 的内核是使用 C 语言写的。第二，C 语言的运行速度块。世界上的三大操作系统 UNIX、Linux 和 Windows 的内核都是 C 语言写的，UNIX 和 Linux 更是纯 C 语言开发的。这是因为我们使用的软件就是在操作系统上运行的，如果操作系统的速度很慢，那么在其上搭载的应用程序也就会更慢。而且操作系统还控制着计算机上的所有硬件。正事 C 语言快速的运行速度，使它成为了操作系统的内核语言。第三，C 语言功能强大。这个也可以从其在操作系统中的作用上体现出来。

本次设计即是使用 C 语言来编写 STM32 的程序，并且使用 uCOS 操作系统。进行设计需要的软件为 Keil MDK v5。Keil 公司于 2013 年 10 月正式推出 Keil MDK v5，这个版本使用 uVision5 IDE 集成开发环境，是 ARM Cortex-M 内核微控制器最佳的一款集成开发工具（针对 ARM 微控制器）。MDK5 对传统的开发模式和界面进行了升级并且向后兼容 Keil MDK-ARM uVision4。

本次软件设计首先需要移植 UCOSIII 到 STM32F103C8T6。移大致可以分为以下几步：

（1） 创建一个最简单的无系统项目工程。

（2） 从官方渠道获取 UCOSIII 源码以及移植好的与 STM32F103C8T6 最接近芯片型号的工程。

（3） 将操作系统的代码移植到之前创建的裸机工程中。

（4） 根据报错修改代码，直到没有错误。并且运行成功。

首先，打开 MDK5 的软件界面。选择顶部菜单上的“Project”, 选择弹出菜单的第一个选择项“New uVision Project...”。之后会弹出一个名为“Create New Project”的界面，在这个界面选择一个文件保存的位置，并命名工程。之后会显示选择 CPU 的界面，这里选择 STM32F103C8。具体界面如图 4-1 所示。

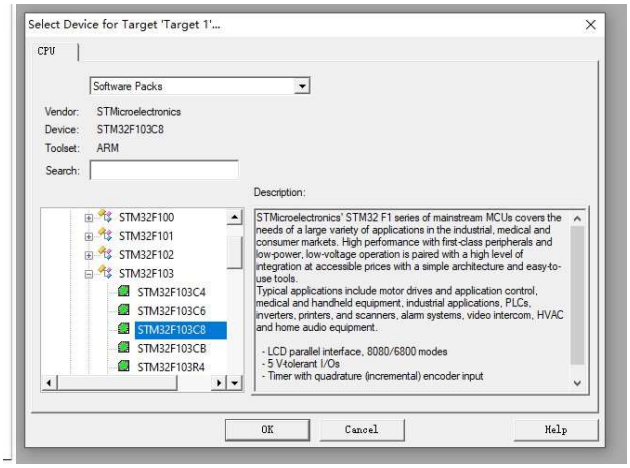


图 4-1 CPU 选择界面

上一步结束后，会弹出一个“Manage Run-Time Environment”的界面，直接点击 OK。然后找到之前设置的存储位置创建如下的文件夹：“CORE”“HARDWARE”“OBJ”“STM32F10X_FWILB”“SYSTEM”“user”。其中“CORE”中用来存放“core_cm3”的“.c”和“.h”的文件以及 STM32 对应的启动文件。“HARDWARE”用来存放我们自己编写的外设程序的文件。“STM32F10X_FWILB”中存放相关的库文件。“SYSTEM”中存放一些几乎所有工程都会涉及的公共配置文件。“user”为用户的文件，主函数就存放在这里。将对应的文件放到文件夹后回到 MDK5 页面。对建立的工程中 Source Group1 右键，这里需要添加一个源文件，命名为 main。然后，选择上面第三行的魔术棒按钮“Options for target…”对其中的 C/C++目录下的 Define 和 Include Paths 进行操作。

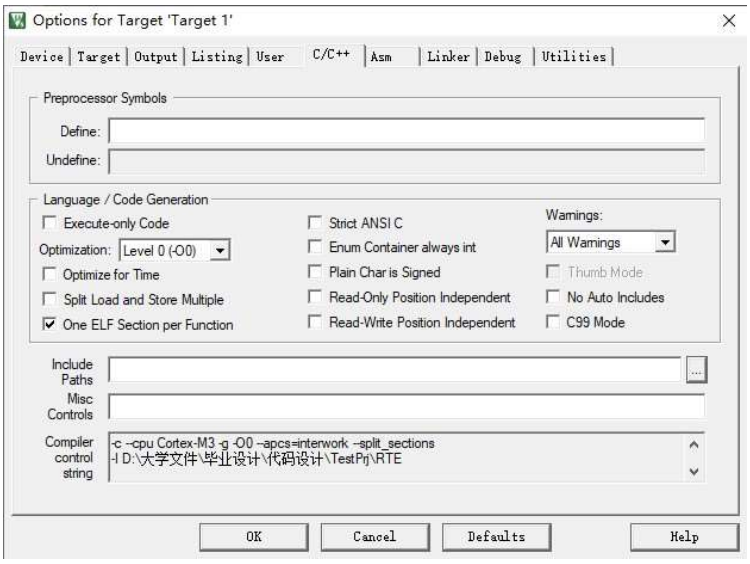


图 4-2 Options for target 界面

之后在 Define 中填写“STM32F10X_MD,USE_STDPERIPH_DRIVER”，在 Include Paths 中添加所需的文件，这里主要是“.h”文件。然后在 main.c 文件中编写一些基本的语句，当出现“0 Error(s),0 Warning(s)”时，表示初次编译成功。可以进行下一步了。

其次，来到官方网站下载 UCOSIII 的代码：http://micrium.com/downloadcenter/download-results/?searchterm=mp-uc-os-iii-1&supported=true。打开下载好的文件后，UCOSIII 的源码就在 Source 文件夹中。关于移植好的工程可以参考下载地址：http://micrium.com/downloadcenter/download-results/?searchterm=mi-stm32f107&supported=true。在这里的移植例子是在 STM32F107 上实现的，它和 STM32F103 同属于 STM32F1 系列。所以这里使用官方移植好的文件来减少自己不必要的负担。打开工程之后会发现四个文件夹，它们分别为“EvalBoards”、“uC-CPU”、“uC-LIB”和“uCOS-III”。EvalBoards 文件夹中只需要 app_cfg.h、cpu_cfg.h、includes.h、lib_cfg.h、os_app_hooks.c、os_app_hooks.h、os_cfg.h 和 os_cfg_app.h。其他三个文件下一步要使用。

再次，来到第一步新建的文件夹，在文件夹中新建名为“UCOSIII”的文件夹，将上一步的“uC-CPU”“uC-LIB”“uCOS-III”文件夹复制到“UCOSIII”里面。后面再另外创建两个“UCOS_BSP”和“UCOS_CONFIG”。UCOS_CONFIG 中存放上一步中 EvalBoards 中需要的 8 个文件。UCOS_BSP 中存放官方移植工程中位于 Micrium/Software/EvalBoards/Micrium/uC-Eval-STM32F107/BSP 的“bsp.c”和“bsp.h”。再回到工程中创建“UCOSIII_BSP”“UCOSIII_CPU”“UCOSIII_LIB”“UCOSIII_CORE”“UCOSIII_PORT”“UCOSIII_CONFIG”。这六个文件夹所需文件如图 4-3 所示，然后添加相应的“.h”文件到 Include Paths。

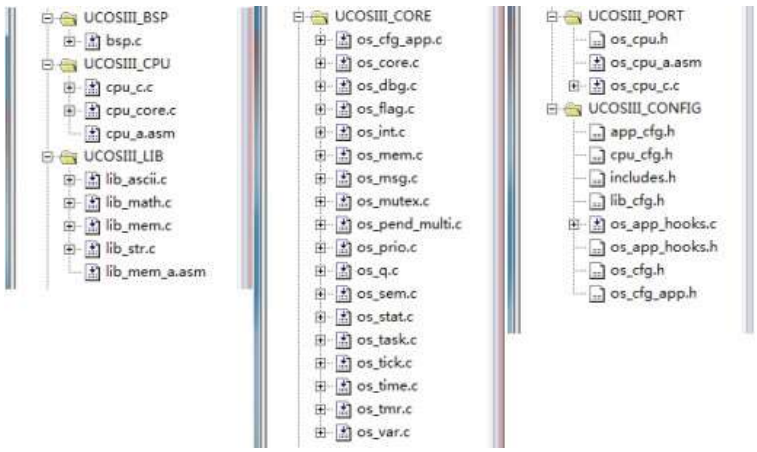


图 4-3 各文件夹所需文件内容

最后，进行工程的编译。这里的编译几乎都会报错，至少有两个。接下来进行一些文件的修改，修改“bsp.c”、“bsp.h”、“os_cpu_a.asm”、“os_cpu_c.c”、“os_cfg_app.h”和“sys.h”文件。具体的修改内容不在这里赘述了，这样基本上就没有问题了。

以上 UCOSIII 的大致移植过程就完成了，接下来就是修改 main.c 文件内容成为符合 UCOSIII 格式的代码样式。UCOSIII 的主函数代码样式的一开始仍然是要去包含头文件，之后是分别为每个任务定义“任务优先级”、“任务堆栈大小”、“任务控制块”、“任务堆栈”和“任务函数”。这里至少要包含两个任务，一个是系统用来启动其他任务的启动任务，另一个就是用户程序。接下来就是编写主函数。之后再从主函数外编写任务启动任务和编写用户函数的代码。如果之后的编译通过了，则移植 UCOSIII 完成。

4．2　主程序流程图

整个程序在 UCOSIII 的统一调度下进行。上电后 STM32 先初始化，之后等待外部按键中断。当外部按键中断出现决定播放的音乐后，STM32 指挥 JQ8900 播放音乐。JQ8900 播放音乐后其 BUSY 引脚高电平使能 TB6612FNG 模块。STM32 中的逐次逼近型 ADC 收到麦克风模块收来的信号后，将模拟信号转化为数字信号，STM32 对数字信号分类，不同的信号分类对应不同的外设状态。STM32 根据分类改变连接 LED 灯条的 IO 口状态和连接 TB6612 的 PWMA 接口的 PWM 的输出占空比。整个过程中的音乐播放可以随时暂停和继续播放，以及中途切换不同的音乐。主程序流程图如图 4-4 所示。

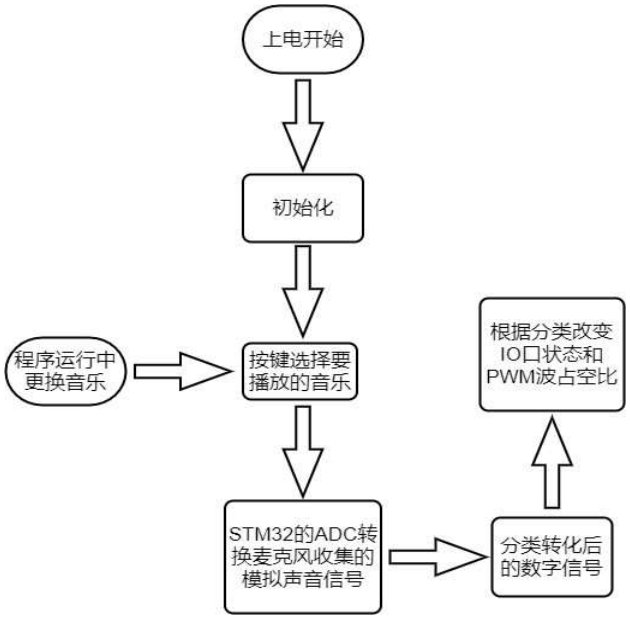


图 4-4 音乐喷泉主程序流程图

关于按键的外部触发部分，STM32 的每一个 GPIO 都能配置成为一个外部中断的触发源。但是 STM32 的外部中断引脚是分组的，每一组中只能同时有一个工作在外部触发状态。也就是说最多可以同时工作的引脚有 16 个。这个数量的引脚对于本项目是完全足够的。引脚与外部中断的对应表如表 4-1 所示。本次设计使用了三个外部中断，分别是 PA4、PA5 和 PB15。他们对应的中断标志位为 EXTI4、EXTI5 和 EXTI15，对应的中断处理函数为 EXTI4_IRQHandler、EXTI9_5_IRQHandler 和 EXTI15_10_IRQHandler。

表 4-1 引脚与外部中断对照表

GPIO 引脚	中断标志位	中断处理函数
PA0~PG0	EXTI0	EXTI0_IRQHandler
PA1~PG1	EXTI1	EXTI1_IRQHandler
PA2~PG2	EXTI2	EXTI2_IRQHandler
PA3~PG3	EXTI3	EXTI3_IRQHandler
PA4~PG4	EXTI4	EXTI4_IRQHandler
PA5~PG5	EXTI5	EXTI9_5_IRQHandler
PA6~PG6	EXTI6	
PA7~PG7	EXTI7	
PA8~PG8	EXTI8	
PA9~PG9	EXTI9	
PA10~PG10	EXTI10	EXTI15_10_IRQHandler
PA11~PG11	EXTI11	
PA12~PG12	EXTI12	
PA13~PG13	EXTI13	
PA14~PG14	EXTI14	
PA15~PG15	EXTI15	

4．3　显示屏模块程序流程图

OLED 显示屏模块的流程图如图 4-6 所示。其中显示器初始化为清空显示区域再将“播放”、“上一”和“下一”显示在屏幕上。此时屏幕等待从按键中断发来的内建信号量。当显示器程序没有收到信号量时，程序就会一直在此处等待，等待会直到信号量到来。按键按下后，显示器程序从等待中恢复。首先，显示器程序会检查按下的是哪一个按键，如果此处是按键 1，则循环选中显示器的四个选项（包括一个空选项）。如果此处按下的为按键 2，则会首先检查按键 1 是否空选，如果空选则不执行任何程序；如果此时按键 1 有选中选项则发送对应的指令给 JQ8900。发送过指令后会重置显示器的状态。

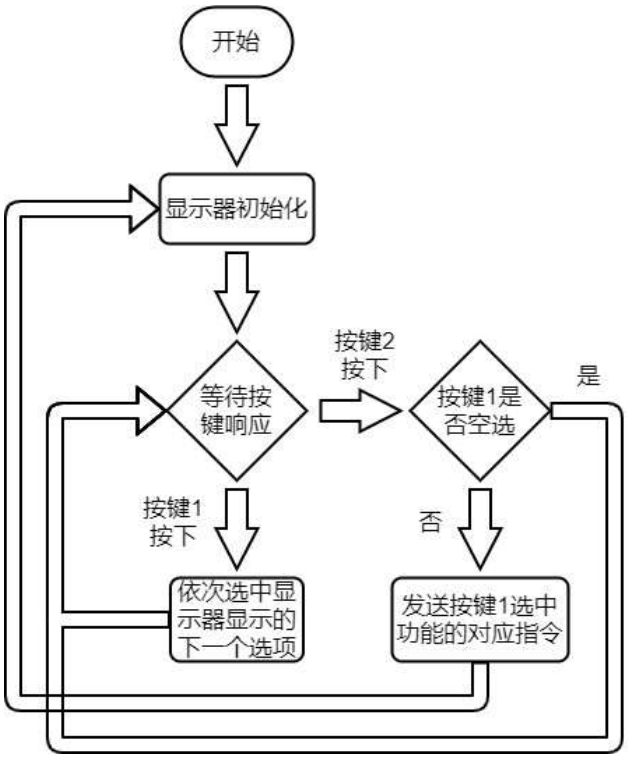


图 4-6 显示屏模块流程图

按键与 OLED 显示屏模块共同组成的显示状态机示意图如图 4-7 所示。其中按键 1 负责切换显示器上显示选中的选项，按键 2 负责发送对应选项的指令。第三个按键则没有使用实体按键，而是将触发接口与 JQ8900 的 BUSY 接口相连。JQ8900 播放完毕一个音乐后 BUSY 引脚会从高电平置向低电平，这个下降沿正好出发了外部中断。按键 3 的触发负责在一个音乐自然播放完毕后，将显示器恢复初始化后的状态。

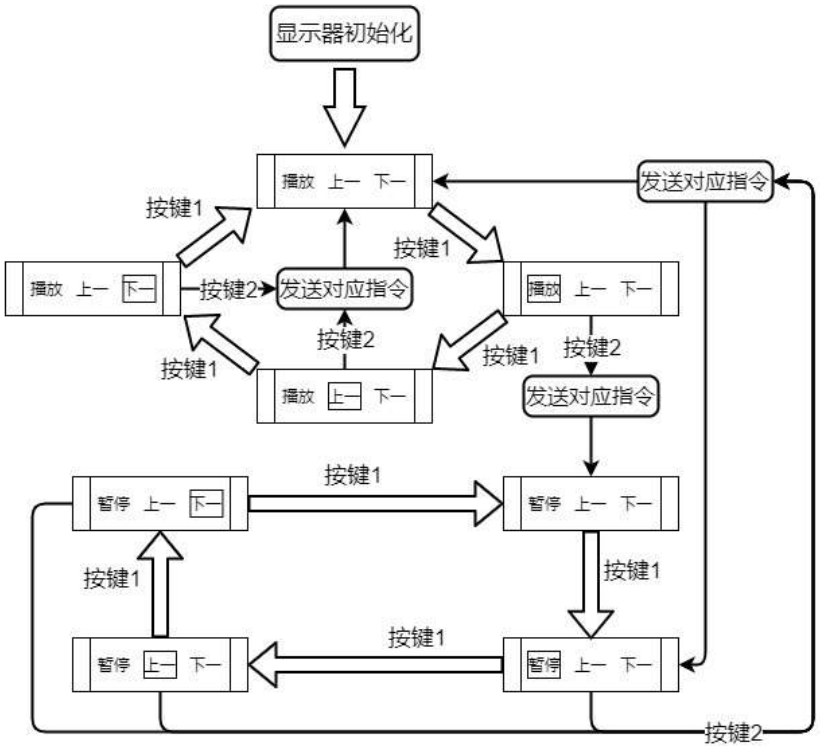


图 4-7 按键与显示器状态机示意图

4．4 程序调试

本次设计的程序初次编译后和与硬件结合后出现了以下两个主要的问题。这两个问题在编写报告前已经全部解决。他们分别为：

（1）Keil MDK v5 提示 STM32F103C8T6 的存储空间不足以存储该程序代码；

（2）硬件设备响应的速度很慢，对使用体验有可见的影响。

针对问题一的情况，从网络及相关材料上可以查询到很多在 STM32F103C8T6 上运行 UCOSIII 的例程。这些工程的代码量仅仅从对工程的估计上就应当比本工程的代码量大。但是可以查到这些资料均可以成功运行基于 UCOSIII 的软件代码。如此便可以了解到该问题极有可能是出在了个人对于 UCOSIII 的配置上。

首先想到的便是每个任务的任务堆栈可能占用了过大的空间，任务实际使用的空间小于分配的堆栈空间，造成了堆栈空间的浪费。为了解决这个问题就需要调查每个任务在实际运行时所占用的堆栈大小。UCOSIII 存在可用的函数来解决这一问题。UCOSIII 提供了堆栈使用情况统计的函数：void OSTaskStkChk()。可以封装该函数再从串口中打印出来堆栈的使用情况来实时确定各任务的堆栈使用情况。具体函数如图 4-8。

```
394  void CheckStack(OS_TCB *p_tcb,void *p_arg, char *tcb_title) //检查堆栈函数
395 {
396    OS_ERR err;
397    p_arg = p_arg;
398    CPU_STK_SIZE nuse;
399    CPU_STK_SIZE use;
400
401    OSTaskStkChk(p_tcb,&nuse,&use,&err);
402    printf("%s:tcb_free:%d\ttcb_use:%d\r\n",tcb_title, nuse, use); //nuse剩余堆栈空间，use已用堆栈空间，tcb_title哪个堆栈任务空间
403 }
```

图 4-8 堆栈检查函数

经过函数检查得出具有代表性的堆栈使用数据，数据表如下表 4-2 所示：

表 4-2 各任务堆栈使用情况统计

任务名称	已使用堆栈空间	未使用堆栈空间
ADC 采集任务	41	87
菜单显示任务	40	88
LED 与 PWM 任务	32	96
按键监测任务	62	66

由表格可见四个子任务都有很大的堆栈浪费。经过查阅资料和代码修改后做出了以下修改方案：取消按键监测任务，改为外部中断触发。然后剩余的三个任务的堆栈大小改为 64。编译后发现容量不足的问题仍然存在，但是已经得到缓解。

经过后续的查阅资料找到了另一条解决该问题的路径。在 UCOSIII 的 lib_cfg.h 的文件里的一行代码为：

#define LIB_MEM_CFG_HEAP_SIZE 27u * 1024u /* Configure heap memory size [see Note #2a]. */（这里将堆空间的大小设置到了 27K）。这里的参数把堆空间的大小设置在了 27K 的尺寸，但是 C8T6 的 RAM 的空间大小都小于这个值。所以现在需要将这个值减小到合适的大小。经过一系列的实验，最终将 LIB_MEM_CFG_HEAP_SIZE 的值设置在了 5u * 1024u 上。然后编译通过，问题一得到解决。

问题一解决后，将代码烧录进入单片机发现：按键反应过于迅速，出现跳过屏幕选项的问题；LED 灯条反应过慢，甚至会出现没有反应的情况；或者出现长期按压按键显示屏没有刷新的问题。

针对按键反应过快提出了以下解决方案。按键反应迅速主要是按键工作在外部中断的端口导致的。当按键触发，正常的程序被中断优先执行按键。如果按键中断程序完成后，按键还没有松开，就会发生多次触发的情况。这个问题的解决方法是将按键消抖操

作与 UCOSIII 系统延时结合起来，这样既不耽误按键消抖的执行又为系统可能存在的调度留下了时间。

LED灯条和PWM波状态改变反应慢和显示屏刷新速度慢的问题在很大程度上是由于任务优先级不够高，ADC 转换任务执行过于频繁导致其他任务执行频率过低的缘故。如此需要再次精简任务。经过多次的尝试得出了如下的分配方案：

（1）按键任务保持使用外部中断的方式触发。

（2)将 LED 灯条状态的改变和 PWM 波状态改变的任务精简到只剩下必要的部分，然后运用 switch 语句将这些函数添加到 ADC 分类的程序中。做到及时分类，及时处理。

（3）使用 UCOSIII 的任务内建信号量来联系按键和显示程序。当按键中断程序触发，并且相关状态的参数修改后再将信号量传送给屏幕显示任务。在屏幕显示任务的一开始便设计一个等待信号量的代码。当信号量还没有传递给显示任务时，显示任务的执行将一直等待信号量。

经过以上的软件调试后形成了三个外部中断，两个用户任务的软件设计方案。经过烧录实际运行后达到了不错的效果。

5 系统测试结果

5．1 硬件调试

硬件调试的方法是要将设计的 PCB 电路使用连接线连接到计算机上，然后将设计好的程序烧录到设计的PCB 设备中。此次设计使用的连接方法是 J-Link 下载调试器。J-Link 是由 SEGGER 公司推出的 JTAG 仿真器。能够配合 KEIL 集成开发环境支持 Cortex 内核芯片的仿真，可以和 Keil 编译环境无缝衔接，J-Link 操作方便、连接方便、简单易学，是学习开发 STM32 的最好最实用的开发工具。

使用 J-Link 进行硬件调试首先需要安装 J-Link 驱动软件到计算机。然后设置 Keil 软件。首先选择 MDK 软件界面上方的魔术棒按钮（Options for Target...），之后选择 Debug 选项卡，在右边一栏选中 Use 然后在下拉栏中选择 J-LINK/J-TRACG Cortex，并勾选 Run to main()选项。

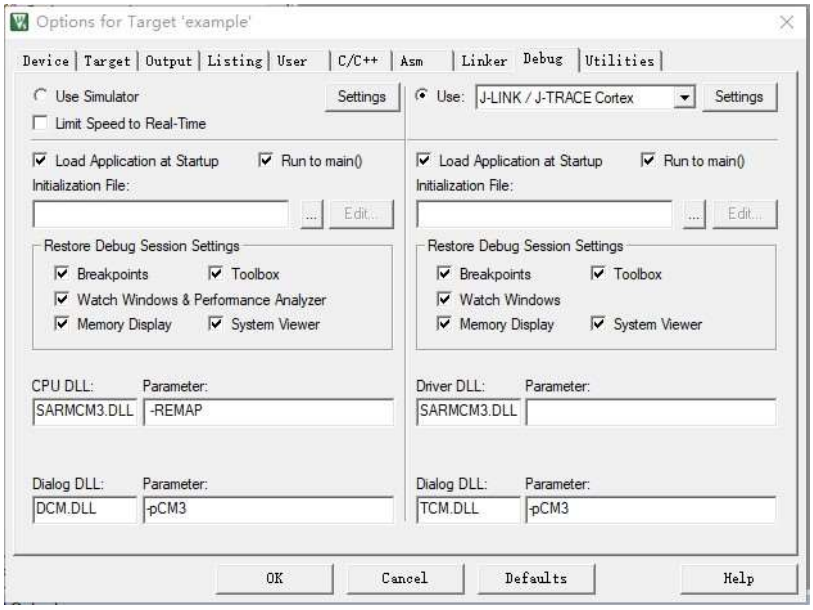


图 5-1 Debug 选项卡设置

之后点击 Settings，进入 Setup 选项卡，选择第一个 Debug 选项卡，将 Port 参数改为 SW。之后再进入上方的 Flash Dowanload 选项卡，在 Programming Algorithm 项目中点击 Add 按钮。从弹出的界面中选择第一项 128k 的选项。最后一直选择“应用”或“确定”按钮，退出所有弹出的界面。之后再用 J-Link 连接计算机的 USB 接口和 PCB 的 J-Link 调试接口。当打开 Debug 选项卡的 Settings 按钮后，看到 SWDIO 右侧出现一串数字，则说明 J-Link 连接计算机和 PCB 设备成功。设备连接成功后，回到 Keil 的主界面。点击界面上方第三行的第三个“Rebuild”按钮再次编译工程，保证编译的代码为最新版本。之后保持 J-Link 的正常连接，按下界面上方第三行的第六个按钮“Download”。之后程序会开始下载到硬件设备。另外，使用 J-Link 还可以仿真程序。在 J-Link 连接的状态下，点击“Start/Stop Debug Session”按钮可以实时仿真程序在设备上的运行情况。以上即为硬件仿真的方法。

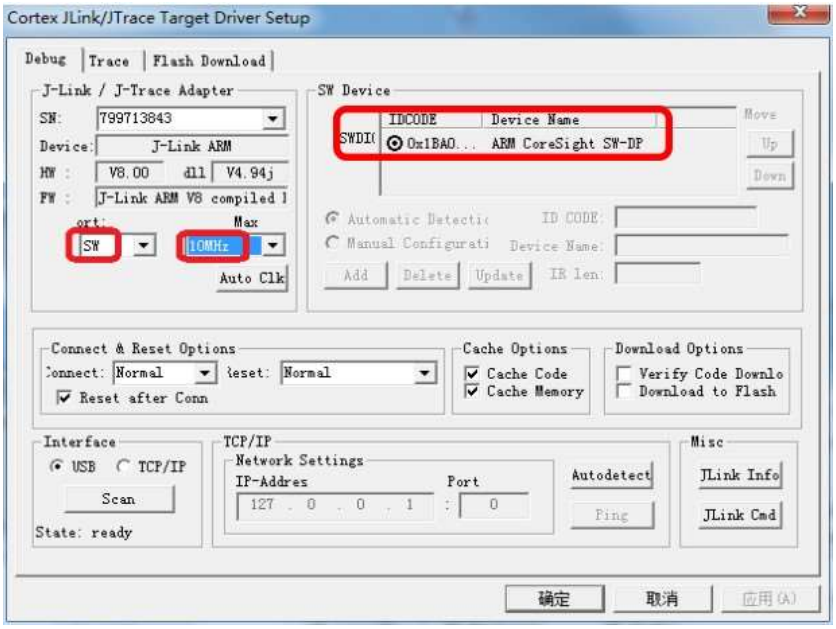


图 5-2 J-Link 连接成功

5．2　硬件调试结果

使用与 5.1 节相同的连接方式，连接计算机和 PCB 设备并烧录程序后的设备运行状态如图 5-3 所示：

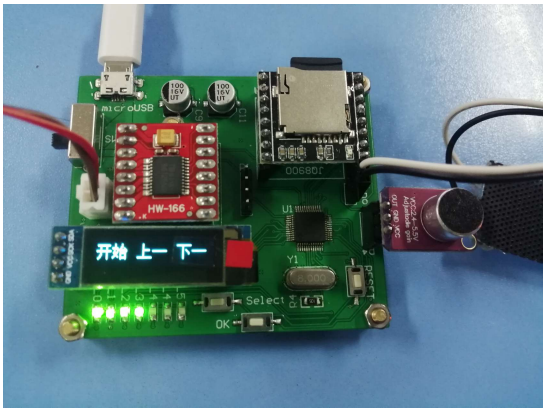


图 5-3　实物运行状态图

如图 5-3 所示，位于 PCB 右下角的 STM32F103C8T6 的核心电路接收到了其右侧的麦克风模块采集来的模拟声音信号，STM32 处理并分类后修改位于 PCB 下方的 LED 灯条的亮起状态和位于 OLED 模块上方的电机驱动模块的驱动功率。OLED 显示模块的显示状态由位于灯条旁边的按键来选择。声音信号则来自于 PCB 右上方的 JQ8900 模块。全电路的供电由 PCB 左上方的 microUSB 接口提供，microUCB 可以直接提供 5V 供电电压或者经过降压电路降压到 3.3V 后使用。

经过实物验证后发现，本次设计的 PCB 实物设备对声音信号的反应速度适宜，能够明显的感受到 LED 灯条和喷泉的高度随音乐的变化。按键反应和屏幕刷新速度良好。整体 PCB 外观扁平体积小巧，基本符合了“移动式”的要求。

5．3 Proteus 验证

Proteus 是一个集原理布图、PCB 设计、SPICE 电路仿真、互动电路仿真、仿真处理器和仿真外围电路功能于一体的软件。Proteus 是由 Lab Center Electronics 公司推出的工具软件。Proteus 具有丰富的器件库，其器件库中保存着超过 27000 种元器件；支持诸如 ARM7、8051/52、AVR、PIC10/12、PIC16、PIC18、PIC24、dsPIC33、HC11、BasicStamp、8086、MSP430 等的主流 CPU 类型；支持通用外设模型，如字符 LCD 模块、图形 LCD 模块、LED 点阵、多位数码管模块、键盘/按键、直流/步进/伺服电机、RS232 虚拟终端、电子温度传感器等。Proteus 的多功能使得它在互联网和文献上都有很多资料可查，使用便利，入门速度快。

由于 Proteus 内部的外设模型中不含 JQ8900 模块的缘故，本次设计的 Proteus 验证部分不能加入音乐播放部分。在 Proteus 的主界面选择新建原理图，然后选择左侧工具栏的第二个“原件模式”，电机第二列的“P”按钮，搜索所需的元器件。然后连接各器件组成如图 5-4 所示的仿真验证电路。仍然因为 Proteus 的仿真 CPU 中不含 STM32F103C8T6 的单片机，所以使用 STM32F103C6 系列代替。整个仿真电路的上方的正弦波发生器代表模拟声音信号，通过其左下方的示波器来观察正弦波的波形，和 STM32 输出的 PWM 波信号。其他电路按照从左向右、从上到下的顺序依次为：TB6612 电机驱动电路、LED 灯条和 STM32 核心电路。

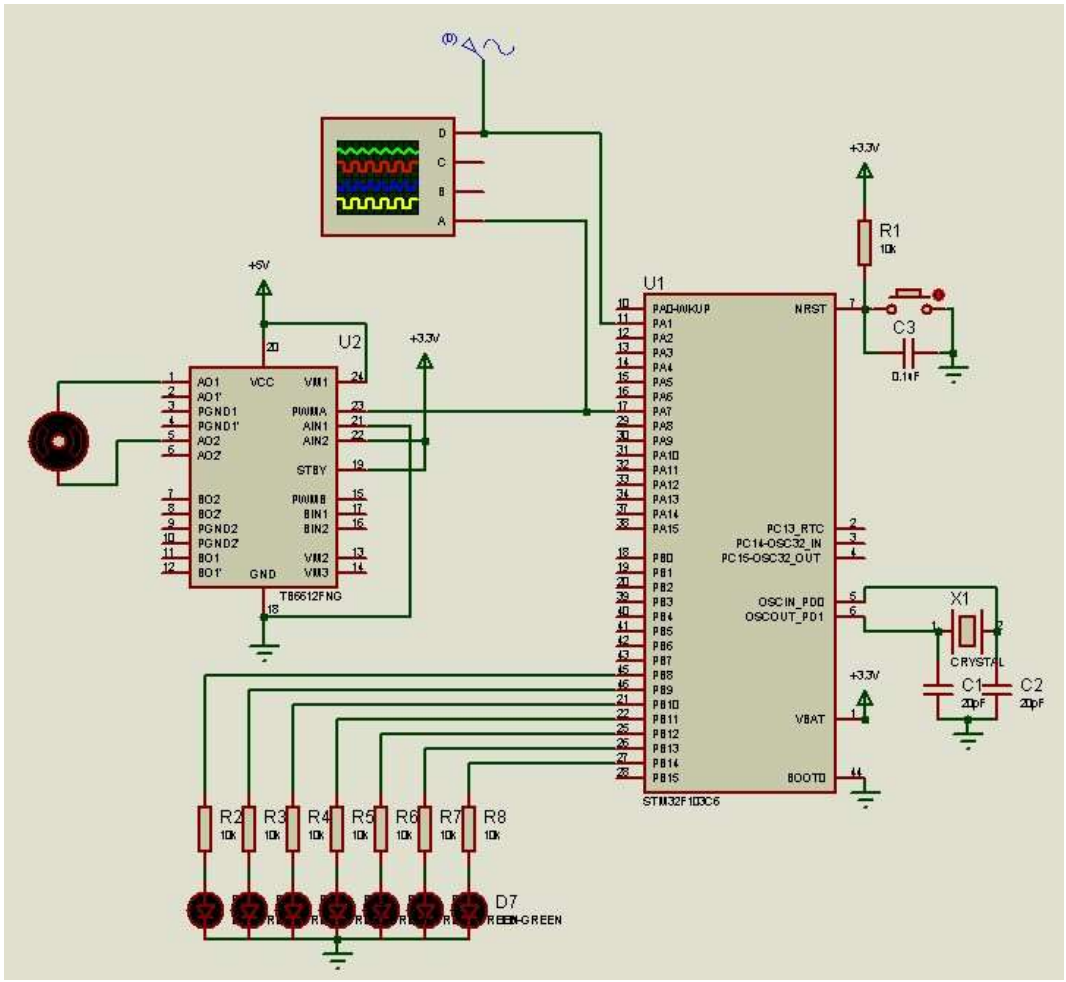


图 5-4 仿真验证电路

下面需要将 STM32C8 的程序移植到 C6 上。大致的修改方式为首先在 Keil 软件中修改芯片型号并且添加适合 md 的启动文件 md.s。之后查看启动文件和 C/C++选项卡中的宏定义是否与 STM32F103C6 相匹配。再查看 J-Link 相关选项卡中的 Flash Download 所选的容量大小是否和 STM32F103C6 相匹配。最后尝试编译工程，修改报错的内容。软件程序修改成功后，只要回到 Proteus 软件双击 STM32F103C6 的电路原理图，通过选择编译出的 hex 文件，即可将程序送入 Proteus 仿真中。

由此以上，本次设计达到了所有的拟定指标，并且通过了 Proteus 仿真和实物验证。LED 灯条的点亮状态和喷泉的高度明显地随着音乐改变。对于其他可能存在的软件和硬件问题，需要之后的长期运行加以暴露。

结 论

本次毕业设计顺利完成了任务书规定的各项任务。通过本次毕业设计我已经能够较为熟练的使用 Keil 编程工具、Proteus 电路仿真软件和 Altium Designer17。了解到了音频信号的发出和采集的原理以及如何改变喷泉的高度。再次熟悉了解了 STM32 内置的 ADC 功能，了解了逐次逼近型 AD 转换器的原理。并且使用 STM32 成功进行了模拟信号向数字信号的转化。

本次设计顺利的完成了音乐喷泉的设计。整体 PCB 大小在 7cm*7cm 以内，基本满足了移动式的要求。喷泉水的高低与 LED 灯条的明暗状态能够随音乐的播放改变，灯条和水型的变化与音乐的变化达到了适宜的同步。本次设计的成品体积小巧，功能明确。具有比较良好的用户交互界面允许用户切换不同的音乐。另外，成品的喷泉功能是与音乐播放功能在一定程度上绑定的，这样避免了其他声音的干扰。

本次设计仍然有可改进之处。首先，PCB 的面积大小在理论上可以继续缩小。因为原件是固定在插座上的，所以原件可以在 Z 轴方向上空闲的缝隙安置。另外，电子元件的封装可以选择更小的封装形式。其次，虽然本次设计的实物在整个调试过程中并未出现程序跑飞和失去响应的情况，如果将此设备用于实际生活中可能仍然需要加入看门狗程序，去监测软件的运行情况。

参 考 文 献

[1]任哲, 房红征, 曹靖. 嵌入式实时操作系统 UC/OS-Ⅱ原理及应用[M]. 北京航空航天大学出版社, 2014.

[2]王彤. 基于 51 单片机的音乐喷泉设计[J]. 中国新通信, 2020,22(01):143.

[3]叶春德, 冯友强. 基于 PLC 的音乐喷泉控制系统改造[J]. 科技视界, 2019(22):26-28+23.

[4]韩国荣, 石志孝. 音乐喷泉的工作原理及控制[J]. 演艺科技, 2018(08):60-62.

[5]王彬, 吴明晖, 周围. 音乐烟花喷泉智能控制系统设计[J]. 软件, 2020,41(07):197-200.

[6]聂东, 陈佩钦, 马嘉懿. 基于组态模式的喷泉电路测试控制系统[J]. 数字通信世界, 2019(12):144+178.

[7]王志远. 基于单片机输出 PWM 波控制音乐喷泉系统设计[J].国外电子测量技术，2018

[8]贺力克. PLC 在音乐喷泉系统控制中的一题多解[J]. 电气电子教学学报, 2019, 41(01):132-137.

[9]杨蕴哲,李昌禄，杨业昆，张倪，晁志腾. 基于单片机的音乐喷泉设计[J].实验科学与技术,2020,18(03):43-49.

[10]李飞,张冬梅,常弘煜,苏言. 基于单片机的音乐喷泉控制系统设计[J].智能城市,2018,4(10):162-163.

[11]正点原子. STM32F1 开发指南-库函数版本_V3.1.电子版手册，2012：329~341

[12]正点原子. 战舰 STM32F1 UCOS 开发手册_V2.0.电子版手册，2015

[13]Wang Feiyu,Zhang Ying,Sun Anan,Li Jinze,Liao Rong. Research on Music Fountain Design based on Intelligent Water System[J]. IOP Conference Series: Earth and Environmental Science,2021,690(1).

[14]Biqing Li,Zhao Li,Suping Jiang. Realization of Music Fountain Control system based on single Chip Microcomputer[A]. Wuhan Zhicheng Times Cultural Development Co., Ltd.Proceedings of 2018 International Conference on Network, Communication, Computer Engineering (NCCE 2018)[C].Wuhan Zhicheng Times Cultural Development Co., Ltd:武汉志诚时代文化发展有限公司,2018:7.

[15]Biqing Li,Zhao Li,Suping Jiang. Realization of Music Fountain Control system based on single Chip Microcomputer[P]. Proceedings of the 2018 International Conference on Network, Communication, Computer Engineering (NCCE 2018),2018.

致 谢

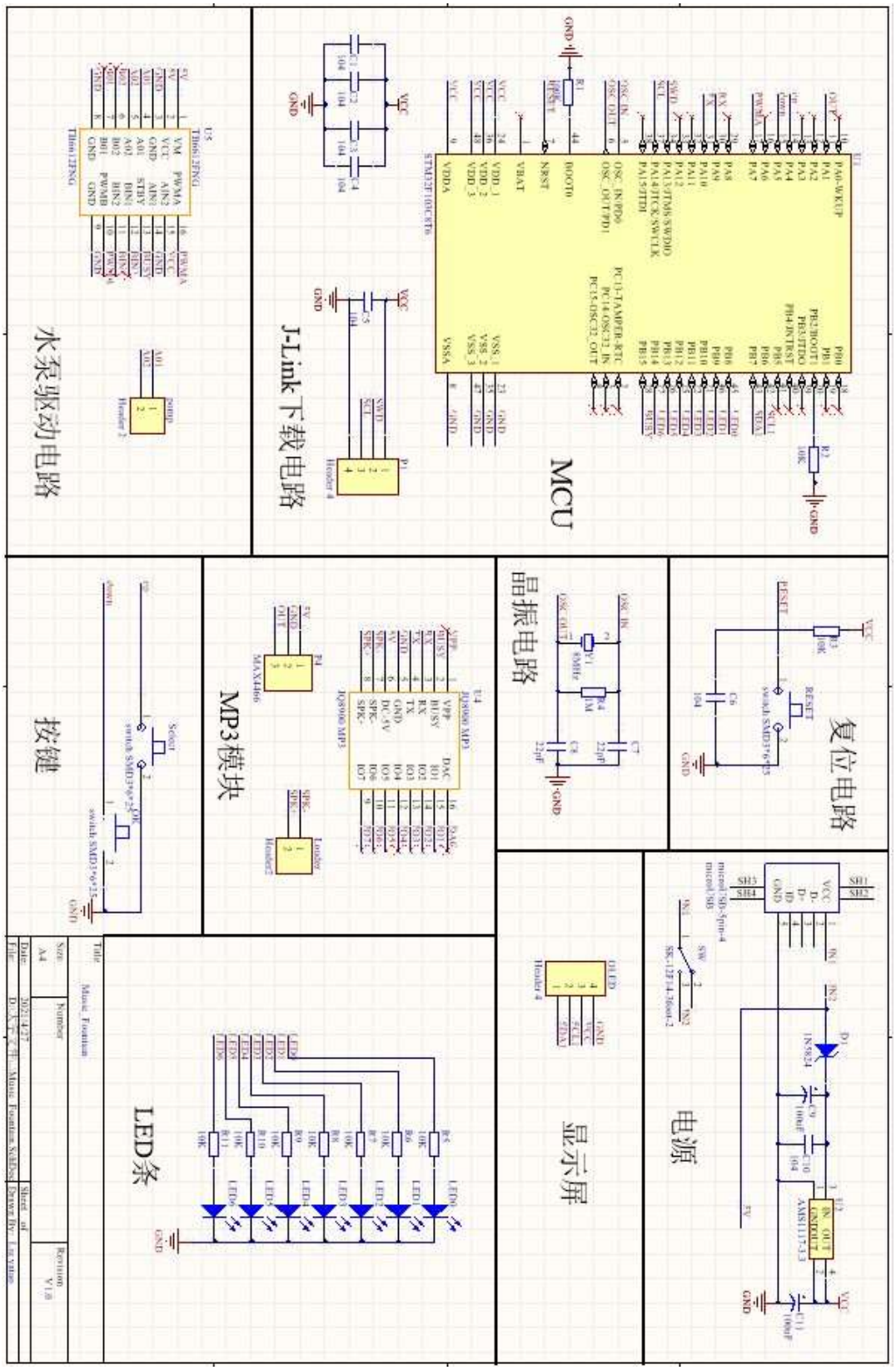
四年的时光如白驹过隙，充实美好的大学生活即将画上句号。回首四年的学习生活，我得到了很多老师、同学的帮助。在毕业论文即将完成之际，首先向陪伴并且帮助我度过了大学最后时光的毕设指导老师致以真诚的谢意，以及在大学期间所有给予过我帮助、支持和肯定的人表示感谢。同时也感谢我的母校，为我提供了我大学四年的学习环境，给了我一个发展自己的平台。感谢本专业的老师们，让我对自己专业的了解更进一步。

经过几个月的努力，本次毕业设计即将接近尾声。毕业设计也给了我很多的启示。仔细总结这些从毕业设计中得来的经验，想必也一定会对我未来的生活提供很多帮助。由于对于毕设所需专业知识的不牢固，在设计的过程中出现了一些本可以避免的错误。这些错误提醒着我要多多总结自己的开发经历，用这些经历为以后的具体工作加速。通过本次设计我还历练了文献查阅的能力，了解到了如何寻找有用的材料，如何使用合适的工具从浩如烟海的文献中，找出适合我的文章。资料的查阅能力与信息检索能力也必然能够在未来给予我有力的帮助。

再次回首大学四年的生活，给予我帮助的人有很多。他们和我一起生活，一起学习，帮助我补习课程，在闲聊中扩展我的事业。另外还要感谢我当初决定加入学校实验室的决定，在实验室学习的几年里为我的毕业设计活动打下了可观的基础。还要感谢我加入的实验室的两位指导老师和实验室的大家，实验良好的学习氛围使我可以醉心于我感兴趣的方面，然后在反补到我的在校活动中，既创造了应有的价值又愉悦了自我。最后，还是要感谢毕业设计的指导老师。多谢老师的细心指导，使我的毕业设计能够提前避开误区顺利的进行下去。

至此，对所有帮助过我的人表达真挚的谢意。

附录 A　移动式音乐喷泉电路原理图



附录 B　移动式音乐喷泉软件程序

```
#include "led.h"
#include "adc.h"
#include "oled.h"
#include "bmp.h"
#include "key.h"
#include "timer.h"
#include "delay.h"
#include "sys.h"
#include "usart.h"
#include "includes.h"
#include "os_app_hooks.h"
#include "exti.h"

//这些优先级分配给了 UCOSIII 的 5 个系统内部任务，用户不可用。
//优先级 0：中断服务服务管理任务 OS_IntQTask()
//优先级 1：时钟节拍任务 OS_TickTask()
//优先级 2：定时任务 OS_TmrTask()
//优先级 OS_CFG_PRIO_MAX-2：统计任务 OS_StatTask()
//优先级 OS_CFG_PRIO_MAX-1：空闲任务 OS_IdleTask()
/*针对空间不足：修改了文件 os_cfg.h 的服务，能显著降低空间使用
//修改了 lib_cfg.h 的 LIB_MEM_CFG_HEAP_SIZE 为 5<-------此参数应该是重点
*/

//任务优先级
#define START_TASK_PRIO      3
//任务堆栈大小
#define START_STK_SIZE       64
```

```
//任务控制块
OS_TCB StartTaskTCB;
//任务堆栈
CPU_STK START_TASK_STK[START_STK_SIZE];
//任务函数
void start_task(void *p_arg);

//任务优先级（ADC）
#define TASK1_TASK_PRIO     4
//任务堆栈大小
#define TASK1_STK_SIZE      64
//任务控制块
OS_TCB Task1TaskTCB;
//任务堆栈
CPU_STK TASK1_TASK_STK[TASK1_STK_SIZE];
void task1_task(void *p_arg);

//任务优先级（菜单）
#define TASK2_TASK_PRIO     5
//任务堆栈大小
#define TASK2_STK_SIZE      64
//任务控制块
OS_TCB Task2TaskTCB;
//任务堆栈
CPU_STK TASK2_TASK_STK[TASK2_STK_SIZE];
//任务函数
void task2_task(void *p_arg);
```

```
u8 flag0,flag1,flag2,flag3;
u16 adcx;

int main(void)
{
    OS_ERR err;
    CPU_SR_ALLOC();

    delay_init();       //延时初始化
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //中断分组配置
    uart_init(9600);    //串口波特率设置
    LED_Init();         //LED 初始化
    KEY_Init();
    EXTIX_Init();
    Adc_Init();
    OLED_Init();
    TIM3_PWM_Init(899,0);

    OSInit(&err);     //初始化 UCOSIII
    OS_CRITICAL_ENTER();//进入临界区
    //创建开始任务
    OSTaskCreate((OS_TCB     * )&StartTaskTCB,     //任务控制块
                 (CPU_CHAR * )"start task",      //任务名字
                   (OS_TASK_PTR )start_task,           //任务函数
                   (void       * )0,                   //传递给任务函数的参数
                   (OS_PRIO      )START_TASK_PRIO,      //任务优先级
                   (CPU_STK    * )&START_TASK_STK[0],   //任务堆栈基地址
```

```
                (CPU_STK_SIZE)START_STK_SIZE/10,     //任务堆栈深度限位
                (CPU_STK_SIZE)START_STK_SIZE,        //任务堆栈大小
                (OS_MSG_QTY  )0,                     //任务内部消息队列能够接收的最大消息数目,为0时禁止接收消息
                (OS_TICK     )0,                     //当使能时间片轮转时的时间片长度，为0时为默认长度，
                (void      * )0,                     //用户补充的存储区
                (OS_OPT      )OS_OPT_TASK_STK_CHK|OS_OPT_TASK_STK_CLR, //任务选项
                (OS_ERR    * )&err);                 //存放该函数错误时的返回值
    OS_CRITICAL_EXIT();  //退出临界区
    OSStart(&err);  //开启UCOSIII
    while(1);
}

//开始任务函数
void start_task(void *p_arg)
{
    OS_ERR err;
    CPU_SR_ALLOC();
    p_arg = p_arg;

    CPU_Init();
#if OS_CFG_STAT_TASK_EN > 0u
   OSStatTaskCPUUsageInit(&err);    //统计任务
#endif

#ifdef CPU_CFG_INT_DIS_MEAS_EN      //如果使能了测量中断关闭时间
    CPU_IntDisMeasMaxCurReset();
```

```
#endif

    OS_CRITICAL_ENTER(); //进入临界区
    //创建 TASK1 任务
    OSTaskCreate((OS_TCB     * )&Task1TaskTCB,
                 (CPU_CHAR * )"task1 task",
                   (OS_TASK_PTR )task1_task,
                   (void       * )0,
                   (OS_PRIO      )TASK1_TASK_PRIO,
                   (CPU_STK    * )&TASK1_TASK_STK[0],
                   (CPU_STK_SIZE)TASK1_STK_SIZE/10,
                   (CPU_STK_SIZE)TASK1_STK_SIZE,
                   (OS_MSG_QTY   )0,
                   (OS_TICK      )0,
                   (void       * )0,
                   (OS_OPT       )OS_OPT_TASK_STK_CHK|OS_OPT_TASK_STK_CLR,
                   (OS_ERR     * )&err);

    //创建 TASK2 任务
    OSTaskCreate((OS_TCB     * )&Task2TaskTCB,
                 (CPU_CHAR * )"task2 task",
                   (OS_TASK_PTR )task2_task,
                   (void       * )0,
                   (OS_PRIO      )TASK2_TASK_PRIO,
                   (CPU_STK    * )&TASK2_TASK_STK[0],
                   (CPU_STK_SIZE)TASK2_STK_SIZE/10,
                   (CPU_STK_SIZE)TASK2_STK_SIZE,
                   (OS_MSG_QTY   )0,
```

```
                (OS_TICK     )0,
                (void      * )0,
                (OS_OPT      )OS_OPT_TASK_STK_CHK|OS_OPT_TASK_STK_CLR,
                (OS_ERR    * )&err);

    OS_TaskSuspend((OS_TCB*)&StartTaskTCB,&err);      //挂起开始任务

    OS_CRITICAL_EXIT();  //进入临界区
}

//task1 任务函数
void task1_task(void *p_arg)//ADC
{
    u16 value;
    OS_ERR err;
    CPU_SR_ALLOC();
    p_arg=p_arg;

    while(1)
    {
        adcx=Get_Adc_Average(ADC_Channel_1,10);
        value=adcx;

        if(value<2638)
        {
            flag0=1;
        }
        else if(value>=2538&&value<2891)
        {
```

```
        flag0=2;
    }
    else if(value>=2891&&value<3144)
    {
        flag0=3;
    }
    else if(value>=3144&&value<3397)
    {
        flag0=4;
    }
    else if(value>=3397&&value<3650)
    {
        flag0=5;
    }
    else if(value>=3650&&value<3903)
    {
        flag0=6;
    }
    else if(value>=3903)
    {
        flag0=7;
    }

    switch(flag0)
    {
        case 1:LED_State(1),TIM_SetCompare2(TIM3,700);break;//通道 2 比较
值;
        case 2:LED_State(2),TIM_SetCompare2(TIM3,600);break;
        case 3:LED_State(3),TIM_SetCompare2(TIM3,500);break;
```

```
            case 4:LED_State(4),TIM_SetCompare2(TIM3,400);break;
            case 5:LED_State(5),TIM_SetCompare2(TIM3,300);break;
            case 6:LED_State(6),TIM_SetCompare2(TIM3,200);break;
            case 7:LED_State(7),TIM_SetCompare2(TIM3,100);break;
            default:LED_State(0),TIM_SetCompare2(TIM3,50);
        }

        OSTimeDlyHMSM(0,0,0,50,OS_OPT_TIME_HMSM_STRICT,&err);//延时
    }
}

//task2 任务函数
void task2_task(void *p_arg)//MENU
{
    u8 i=0;
    OS_ERR err;
    CPU_SR_ALLOC();
    p_arg=p_arg;

    OLED_Clear();
    OLED_SelectNull();

    while(1)
    {
        OSTaskSemPend(0,OS_OPT_PEND_BLOCKING,0,&err);//请求任务内建的信号量
        if(flag1!=0)
        {
            if(flag1==1)
```

```
{
    i++;
    if(i==4)
    {
        i=0;
    }
    if(flag3==0)
    {
        switch(i)
        {
            case 1:OLED_SelectPlay(),flag2=1,flag1=0;break;
            case 2:OLED_SelectLast(),flag2=2,flag1=0;break;
            case 3:OLED_SelectNext(),flag2=3,flag1=0;break;
            default:OLED_SelectNull(),flag2=0,flag1=0;
        }
    }
    else if(flag3==1)
    {
        switch(i)
        {
            case 1:OLED_PlayingSelectStop(),flag2=4,flag1=0;break;
            case 2:OLED_PlayingSelectLast(),flag2=2,flag1=0;break;
            case 3:OLED_PlayingSelectNext(),flag2=3,flag1=0;break;
            default:OLED_PlayingSelectNull(),flag2=0,flag1=0;
        }
    }
}
else if(flag1==2)
```

```
{
    switch(flag2)
    {
        case 1://播放情况指令
        {
            USART_SendData(USART1,0xAA),delay_ms(1),
            USART_SendData(USART1,0x02),delay_ms(1),
            USART_SendData(USART1,0x00),delay_ms(1),
            USART_SendData(USART1,0xAC);
            flag2=0,flag3=1,i=0,OLED_PlayingSelectNull();}break;//
正在播放模式
        case 2://上一曲模式
        {
            USART_SendData(USART1,0xAA),delay_ms(1),
            USART_SendData(USART1,0x05),delay_ms(1),
            USART_SendData(USART1,0x00),delay_ms(1),
            USART_SendData(USART1,0xAF);
            flag2=0,flag3=1,i=0,OLED_PlayingSelectNull();}break;
        case 3://下一曲模式
        {
            USART_SendData(USART1,0xAA),delay_ms(1),
            USART_SendData(USART1,0x06),delay_ms(1),
            USART_SendData(USART1,0x00),delay_ms(1),
            USART_SendData(USART1,0xB0);
            flag2=0,flag3=1,i=0,OLED_PlayingSelectNull();}break;
        case 4://暂停模式
        {
            USART_SendData(USART1,0xAA),delay_ms(1),
            USART_SendData(USART1,0x03),delay_ms(1),
```

```
                    USART_SendData(USART1,0x00),delay_ms(1),
                    USART_SendData(USART1,0xAD);
                    flag2=0,flag3=0,i=0,OLED_SelectNull();}break;
                default:flag2=0,i=0;
            }
        }
    }
//  OSTimeDlyHMSM(0,0,1,0,OS_OPT_TIME_HMSM_STRICT,&err);//延时
    }
}

void EXTI15_10_IRQHandler()//中断执行函数   KEY0
{
    OSIntEnter();
    if(KEY0==0)//这里的 KEY0、1 为位操作
    {
        OLED_SelectNull(),flag3=0,flag2=0,flag1=0;
    }
    EXTI_ClearITPendingBit(EXTI_Line15);//清除线四的中断标志位。防止下一次中断因为标志位未清零引起的中断不响应。
    OSIntExit();
}

void EXTI4_IRQHandler()//中断执行函数    KEY1
{
    OS_ERR err;

    OSIntEnter();
    OSTimeDlyHMSM(0,0,0,100,OS_OPT_TIME_HMSM_STRICT,&err);//延时
```

```
    if(KEY1==0)
        {
            flag1=1;
            EXTI_ClearITPendingBit(EXTI_Line4);//清除线四的中断标志位。防止下一次中断因为标志位未清零引起的中断不响应。
            OSTaskSemPost(&Task2TaskTCB,OS_OPT_POST_NONE,&err); //使用系统内建信号量向任务 task2 发送信号量
        }
    OSIntExit();
}

void EXTI9_5_IRQHandler()//中断执行函数     KEY2
{
    OS_ERR err;

    OSIntEnter();
    OSTimeDlyHMSM(0,0,0,100,OS_OPT_TIME_HMSM_STRICT,&err);//延时
    if(KEY2==0)
        {
            flag1=2;
            EXTI_ClearITPendingBit(EXTI_Line5);//清除线四的中断标志位。防止下一次中断因为标志位未清零引起的中断不响应。
            OSTaskSemPost(&Task2TaskTCB,OS_OPT_POST_NONE,&err); //使用系统内建信号量向任务 task2 发送信号量
        }
    OSIntExit();
}
```