

毕 业 设 计（论文）

题目 基于i.MX283处理器的音乐播放器设计

学 院 机械工程学院

专 业 测控技术与仪器

班 级 113070303

学生姓名 张进科 学号 11307030328

指导教师 职称

时 间

目 录

[摘 要 I](#_Toc483520362)

[Abstract II](#_Toc483520363)

[1 绪论 1](#_Toc483520364)

[1.1 课题的研究意义 1](#_Toc483520365)

[1.2 国内外研究现状 1](#_Toc483520366)

[1.3 本课题的主要研究内容 3](#_Toc483520367)

[2 总体方案设计 4](#_Toc483520368)

[2.1 系统设计流程 4](#_Toc483520369)

[2.2 系统硬件框架 5](#_Toc483520370)

[2.3 系统软件框架 5](#_Toc483520371)

[3 系统硬件设计 7](#_Toc483520372)

[3.1.1 处理器简介 7](#_Toc483520373)

[3.1.2 i.MX283电源简介 7](#_Toc483520374)

[3.1.3 DDR2电路简介 8](#_Toc483520375)

[3.1.4 声卡的选择 8](#_Toc483520376)

[4 系统软件设计 10](#_Toc483520377)

[4.1 AWorks简介 10](#_Toc483520378)

[4.2 emWin简介 11](#_Toc483520379)

[4.3 系统任务设计 11](#_Toc483520380)

[4.3.1 GUI任务 12](#_Toc483520381)

[4.3.2 触摸屏任务 12](#_Toc483520382)

[4.3.3 音乐播放任务 13](#_Toc483520383)

[4.4 音频解码 14](#_Toc483520384)

[4.4.1 WAV格式解码 14](#_Toc483520385)

[4.4.2 MP3格式 15](#_Toc483520386)

[4.4.3 FLAC格式 17](#_Toc483520387)

[4.4.4 音乐播放流程 18](#_Toc483520388)

[4.5 LRC歌词解析 19](#_Toc483520389)

[4.5.1 LRC格式简介 19](#_Toc483520390)

[4.5.2 LRC歌词解析 20](#_Toc483520391)

[4.6 用户界面设计 20](#_Toc483520392)

[4.6.1 主界面设计 21](#_Toc483520393)

[4.6.2 播放列表界面设计 23](#_Toc483520394)

[4.6.3 歌曲信息界面 24](#_Toc483520395)

[4.6.4 系统设置界面 25](#_Toc483520396)

[5 测试与结果 26](#_Toc483520397)

[5.1 CPU使用情况 26](#_Toc483520398)

[5.2 设计完成情况 26](#_Toc483520399)

[5.2.1 要求 26](#_Toc483520400)

[5.2.2 完成情况 27](#_Toc483520401)

[6 体会与致谢 28](#_Toc483520402)

[参考文献 29](#_Toc483520403)

[附录 31](#_Toc483520404)

摘 要

现阶段的音乐播放器对于有损音乐的支持已经非常不错，但是能够支持无损格式音乐播放器却鲜有耳闻。所以迫切需求一种能够播放无损音乐，且体积小巧，性能强大的音乐播放器，占领随身设备的高端市场。

本系统所釆用的基于飞思卡尔i.MX283微处理器的EasyARM-i.MX283A开发套件作为目标平台开发板。系统开发环境的构建包括三个部分，首先在嵌入式开发板EasyARM-i.MX283A平台上搭建交叉编译环境，然后移植AWorks软件平台，最后编写相关驱动。软件设计部分包含四个模块，音频解码模块、LRC歌词解析模块、播放列表模块和人机交互模块。文中详细分析了音频解码流程，并通过开源解码器实现了软件设计中音频解码功能。LRC歌词解析模块是在详细分析LRC歌词格式后，设计并实现了歌词解析模块。播放列表模块实现了文件系统内音乐文件的显示与管理。最后在实现解码及人机交互后，釆用线程间通信方式设计并实现以触摸屏控制音乐播放功能。

经测试后此设备运行良好，播放功能完善，具有低能耗高性能的特点，尤其是采用ES9018K2M音频DAC芯片后，具有较高的音质，符合消费者对高品质音乐的追求。

关键词：音乐播放器；AWorks；emWin；WAV；MP3；FLAC

Abstract

This paper further divides incentive……

*小四号、Times New Roman字体*

**Key words:** Wage Incentive……

*小四号、Times New Roman字体、加粗 小四号、Times New Roman字体*

# 绪论

## 课题的研究意义

音乐是人们文娱活动重要的组成部分，音乐播放器的重要性显而易见，它的主要功能是播放音乐文件并显示歌词信息，而且支持多种多样的音频格式。随着电子技术的不断发展，音乐播放器不断的更新换代。

现在，音乐播放器在人们眼中是随处可见的电子设备，但是在没有音乐播放器的时代，能记录声音与播放声音无异于神迹。1898年，以带有磁性的材料为储存介质的钢丝录音机登上历史舞台，磁带录音机开始普及。随着社会的进步，老式录音机的音质已经不能满足用户的要求。至此，通过数字信号存储音频信息的CD被发明出来。1982年，SONY开发的世界首台CD音乐播放器问世。CD具有极好的音质，操作简单，功能强劲。流行了许多年，直到MP3播放器的出现。尽管MP3早在1995年就已经发布，然而受限于互联网的技术限制，MP3并没有普及，直到2001年APPLE公司推出IPOD，它拥有时尚的外观，巨大的容量，且配套服务非常完善，很快就推广到了世界范围内并拥有大量用户。现在，智能手机整合了手机、音视频播放器、相机、上网本等功能，方便人们的使用。但是从前随处可见的音乐播放器由于功能单一，储存空间小，播放效果一般。想要扩展更多的功能比较困难。导致市场占有率越来越低，需要升级换代来适应时代的发展。

现阶段的音乐播放器对于有损音乐的支持已经非常不错，但是能够支持无损格式音乐播放器却鲜有耳闻。所以迫切需求一种能够播放无损音乐，性能强大，且体积小巧的音乐播放器，占领随身设备的高端市场。

## 国内外研究现状

随着智能手机的出现，音乐播放器在市场上的占有率变低。但是在专业领域，音乐播放器有着手机无法比拟的优势，国内外对音乐播放器的研究较多，实现的方案各不相同。国内外有关音乐播放器的研究大致有以下方案：

* 1. 采用低成本单片机的方案

刘垣等[[[1]](#endnote-1)](2015)采用STC的8位单片机设计了WAVE音乐播放器，硬件电路简单，成本低廉，播放效果达到CD音质级别，但受限于8位单片机的性能，不能解码其它格式。

* 1. 使用高性能32位单片机软件解码的方案

袁卫[[[2]](#endnote-2)](2015)使用TI公司的微控制器LM3S9B96实现了mp3、wma、wav等不同的格式文件的播放，搭载有μC/GUI图形界面以及μC/OS-II实时操作系统，人机交互可通过触摸屏实现，系统具有播放流畅、易操作等特点。E.Kalpana等[[[3]](#endnote-3)](2012)基于ARM926处理器验证了MP3解码算法，可以解码44.1Khz，128Kbps的MP3文件并取得良好的效果。Wonchul Lee等[[[4]](#endnote-4)](2002)深入研究了基于32位RISC处理器的MP3软件解码算法，提高了解码的效率，减小了代码的体积，提高了音乐播放器的性能。Kiruthikamani Govardhanaraj等[[[5]](#endnote-5)](2015)使用ARM7处理器研发了一种新型的音乐播放界面，能让盲人与司机更加方便的在海量的音乐列表中选择需要的音乐，方便了盲人的使用，提高了驾驶车辆时切歌的安全性。

* 1. 基于Linux操作系统的方案

於少峰等[[[6]](#endnote-6)](2004)基于AC97标准，采用Linux操作系统解码音频文件，Codec芯片驱动喇叭播放音乐，降低了电磁干扰，获得较好的音效品质。陈自龙等[[[7]](#endnote-7)](2012)基于ARM芯片，搭载Linux操作系统，通过软件解码音频文件，能播放WAV、APE和FLAC无损音乐，但是限制于S3C6410处理器的音频输出外设，最高能播放48kHz采样率的歌曲，后续可以通过更换codec芯片提高系统性能。焦正才等[[[8]](#endnote-8)](2012)基于Phonon对MP3音频进行解算，使用QT介面库，成功做到了对本地MP3音乐文件的支持，由于采用了QT图形界面，系统移植性好，做到了一次编写，处处编译的目的。汪永好等[[[9]](#endnote-9)](2009)在分析MP3编解码原理的基础上，使用开源的音频解码器libmad与audiere实现MP3的播放，取得了较好的效果。王灵芝等[[[10]](#endnote-10)](2009)使用codec芯片UDA1341连接主频高达203MHz的S3C2410处理器，使用QT 进行人机交互，介面美观，功能较多。

* 1. 使用FPGA加速的方案

黄默等[[[11]](#endnote-11)](2010)使用FPGA连接音频解码芯片STMP3410的方式实现了音乐的播放，因为FPGA有高密度、可编程及有功能强大的EDA软件等特点, 所以具有功能强大、 可靠性高等特点。实现低音动态平衡功能，高性能，低成本。

* 1. 应用专用音频解码芯片的方案

何谐[[[12]](#endnote-12)](2013)使用STM32单片机连接VS1003专用音频解码芯片，通过STM32单片机将储存在SD内的音频文件发送到VS1003进行解码并播放，系统能稳定的运行。由于模拟电路与数字解码集成在一块芯片上，音质受到一定的影响。杨雪梅等[[[13]](#endnote-13)](2016)构建的音乐播放器系统使用了STM32作为处理器，VS1053解码芯片做音频解码，通过低功耗、微型化的设计理念，使系统成本低、体积小、使用时间长。李伟等[[[14]](#endnote-14)](2015)与杨雪梅的方案基本一致，可以实现开始、暂停、声音加减和上一首、下一首的功能。功能比较单一，如何在单一平台上实现更多的功能是一个发展方向。瞿兵等[[[15]](#endnote-15)](2015)于何谐的方案相似，但是显示部分采用了TFT彩屏，人机界面更加友好，操作更加简便。章坚武等[[[16]](#endnote-16)](2006)采用了数字多媒体处理芯片EM8510，能解码多种格式的音频文件，但由于使用的是专用芯片，后续升级比较困难。

在众多研究中，学者们就音乐播放器为主题进行了一系列研究。现阶段的研究，或多或少都有一些缺点，研究的路还很长，例如，采用低成本8位单片机的方案虽然成本低廉，但是性能受到限制。使用FPGA加速的方案性能高的同时成本也高，且开发不是很灵活。使用高性能32位单片机软件解码的方案软件灵活，成本适中，但是想要扩展更多的功能比较困难。采用高性能处理器搭载嵌入式操作系统的方案软件灵活，性能强大，升级便利，且有足够的性能扩展更多的功能，虽然成本比较高，但是在高端播放器领域优势比较明显。

## 本课题的主要研究内容

本课题拟设计一种基于AWorks嵌入式系统软件开发平台，使用i.MX283处理器，搭载emWin界面库的高性能音乐播放器。

该播放器通过软件解码音频文件，codec芯片做模数转换实现音乐的播放，除支持常见格式的播放外，还能播放USB设备以及SD卡内的音乐文件，且有美观的界面以及优良的人机交互性能，如播放、暂停、快退、快进、增加音量、减小音量和循环播放等操作的支持。

除以上功能外，还支持歌词同步显示以及多种语言支持。

# 总体方案设计

本设计是基于i.MX283处理器的音乐播放器，采用32位的i.MX283处理器，把数字电路和模拟电路分离开来，从而降低电磁干扰，获得较好的音效品质。开发环境为开源的集成开发环境eclipse cdt，程序采用GNU C语言编程。

## 系统设计流程

音乐播放器也属于嵌入式系统中的一种，所以音乐播放器的开发流程与一般的嵌入式系统的开发流程是相同的。



图1 嵌入式系统开发流程

如图1所示，嵌入式开发流程分为以下五个步骤：

1. 说明书的编写要确定功能和性能要求。然后作为开发产品的参考文件和检验的依据。
2. 体系架构的设计要从生产、成本与使用详情多方考虑，设计总体方案。
3. 软、硬件的协同设计，嵌入式系统的硬件和软件关系密切，软件与硬件的密切相关，所以软硬件设计要同时进行。一般分为各个模块设计，可以减小产品开发周期，便于调试。
4. 系统集成是将开发完成的软件下载到硬件电路中，进行联调。最后将软件固化在硬件电路中。
5. 系统的测试及可靠性报告，测试开发完成的系统能否达到设计书的要求。若满足，则将正确的软件固化在硬件电路中。若不能满足，在情况最坏时，需要重新制定设计方案。

音乐播放器作为一个嵌入式音频播放系统，它的硬件主要包括以嵌入式微处理器为核心的最小系统和声卡。软件包括运行于微处理器之上的嵌入式操作系统，底层的硬件设备驱动及上层应用程序。

## 系统硬件框架

本系统主要由i.MX283处理器、DDR2存储器、TFT液晶屏、电阻式触摸屏、串口、USB主机、音频接口、储存接口、时钟源、调试与下载接口和ES9018K2M声卡组成。硬件系统框架如图2所示。

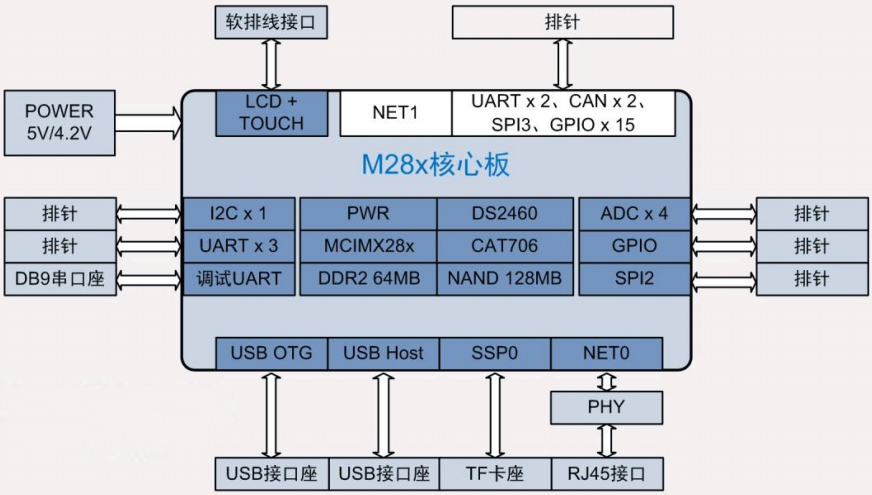


图2 硬件系统框架图

## 系统软件框架

本设计的软件框架分为音频解码、歌词解析、播放列表和人机交互四大部分。其中音频解码分为WAV解码、MP3解码和FLAC解码三种解码支持。WAV和FLAC文件主要由文件头、音频信息以及音频数据组成，MP3文件则由TAGV2标签、音频信息与数据和TAGV1标签组成。

歌词解析部分分为标识标签和时间标签解析，解析出的时间标签存存放到全局变量中供歌词同步显示。

播放列表部分的功能较简洁，只有基本的添加歌曲、删除歌曲和搜索文件系统内的歌曲并添加到播放列表中的功能。

人机交互部分是本系统与用户之间进行信息交流的主要途径，基本上能满足音乐播放器的使用需求。主要分为主界面、播放列表界面、歌曲信息界面和系统设置界面几大界面。主界面主要有播放/暂停、上一曲/下一曲、播放模式切换、歌词显示、歌手图片显示、播放进度调整、音量调整和打开其它界面的功能。播放列表界面主要有显示歌曲列表、切换指定歌曲、删除歌曲和刷新列表几大功能。系统设置界面主要能进行歌词是否显示的设置、歌手图片是否显示的设置以及作者信息显示。歌曲信息界面的功能较少，只是显示当前正在播放音乐的一些基本信息，比如歌曲名称、歌手名称、时长、格式等信息。系统软件框架如图3所示。

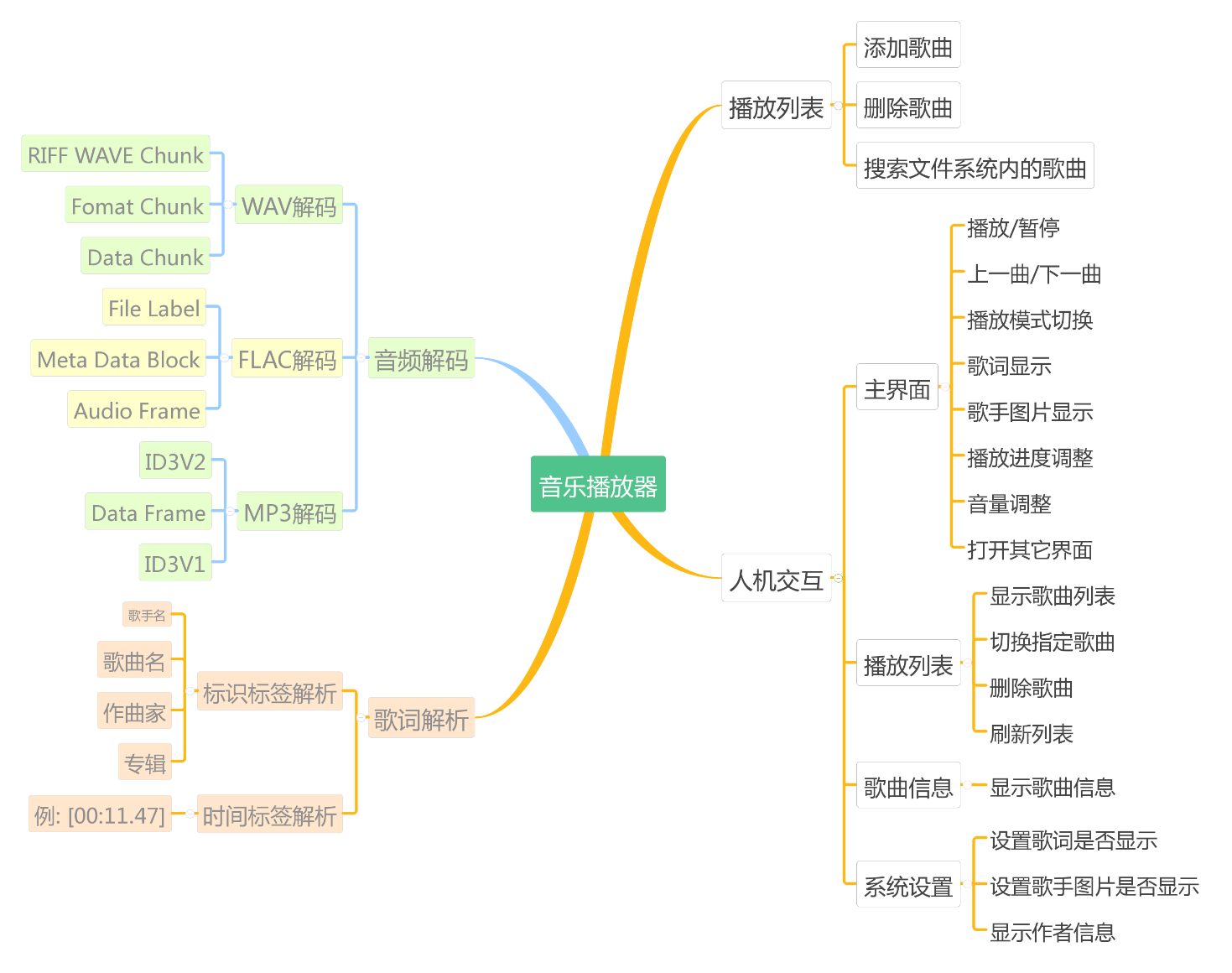


图3 系统软件框架思维导图

# 系统硬件设计

本系统硬件部分主要包含处理数字信号的处理器与处理器相关的外围电路和将处理器处理好的数字音频信号转换为模拟音频信号并输出的声卡。处理器的简介与声卡的选择如后文所示。

### 处理器简介

考虑到对音频文件进行解码时，需要进行大量的数据运算，这就对处理器的性能有较高的要求。本设计采用的飞思卡尔公司的i.MX283处理器是一款高性价比的处理器，具有丰富的硬件资源。专门为嵌入式应用设计。i.MX283使用Freescale高速、稳定、低功耗的ARM926EJ-S内核，主频高达454MHz。为音频解码以及图形界面显示提供了足够的处理速度。并且飞思卡尔为i.MX283处理器提供了完备的驱动、优秀的开发环境与大量的参考程序，极大的方便了用户学习与使用。

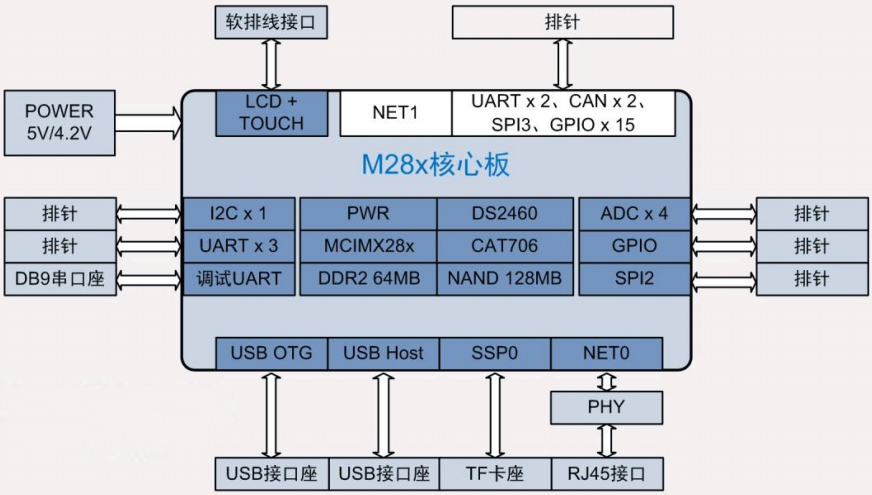


图4 i.MX283系统板功能框图

### i.MX283电源简介

i.MX283嵌入式处理器自带有高效率的电源模块。电源管理模块内部有多个开关电源与线性电源。只要提供一路电源，电源管理模块就能输出CPU所需要的所有电压。这方便了用户的设计，并且提高了系统的性价。所以能够很简单的应用于电池供电的系统。i.MX283电源管理模块框图如图5所示。

电源管理模块有如下三种不同的供电选择：

* 1. 独立的5V电源，比如使用电脑USB接口供电；
  2. 独立的4.2V电源，比如锂电池或降压模块供电；
  3. 只使用电池供电，其它输入接口用来给电池充电；

本设计中，为了使用方便，直接通过随处可见的USB接口给系统板供电。

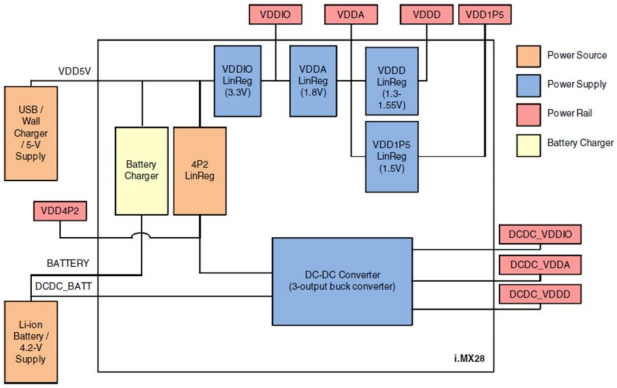


图5 PMU内部结构框图

### DDR2电路简介

i.MX283能够使用多种接口的存储芯片，接口频率最高为205MHz[[[17]](#endnote-17)]。本设计的使用DDR2存储芯片作为系统的内存，大小为64兆字节。DDR2参考电路详见图6。

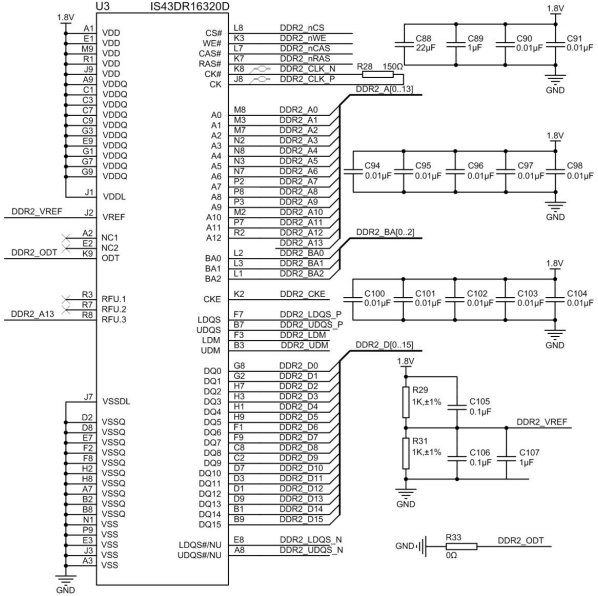


图6 DDR2典型应用电路

### 声卡的选择

当前嵌入式音乐播放系统普遍采用专用解码芯片进行音频的解码与播放，但是由于数字电路与模拟电路集成在一块芯片上，音质受到一定的影响。且后期功能升级需要更换解码芯片，不利于更新功能。所以本设计使用的声卡为音频专用DAC，将数字信号处理与模拟电路部分分开，减小两部分电路之间的相互影响，提高音质。

ES9018K2M是一款高性能的32位2通道音频DAC解决方案，设计用于手机和数字音乐播放器等发烧级便携式应用，消费类应用如蓝光播放器，音频前置放大器和A/V接收器，以及专业应用如录音系统，调音控制台和DAW。

ES9018K2M采用专利的32位Hyperstream DAC架构和时域抖动消除器,可提供高达127dB的信噪比(DNR)和-120dB的THD+N，音质能满足绝大部分用户。ES9018K2M通过I2S接口可以输入多达32位384kHz的PCM数据，DSD接口则能够输入11.2MHz的数据。而且支持最高性能应用的单声道模式。支持同步和异步采样率转换模式。正常工作模式下的功耗低于40mW。

所以ES9018K2M能够满足用户对音质的要求。与处理器连接则通过IIS接口向ES9018K2M传输音频数据，并使用I2C接口写入配置信息。

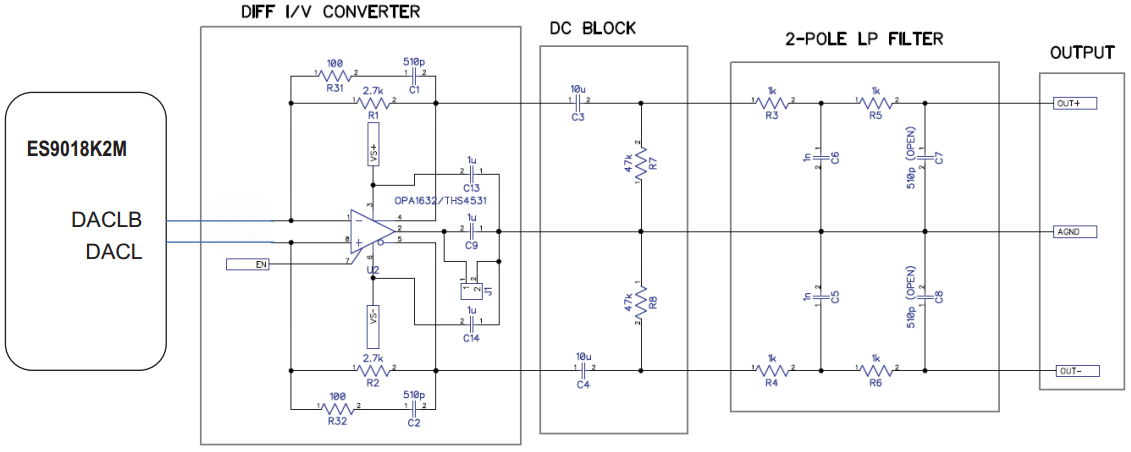


图7 ES9018K2M应用电路

# 系统软件设计

本音乐播放器的软件体系结构采用分层模式，总共包含四层：应用层、系统层、驱动层及硬件层 [[[18]](#endnote-18)]。应用层使用emWin设计人机交互介面，通过解码器与控制器实现对音乐文件进行解码、播放、音量增减的控制。操作系统层为Aworks实时嵌入式操作系统。设备驱动层为系统内各种硬件的驱动代码。硬件层包括各种硬件如声卡ES9018K2M芯片、4.3寸的液晶显示屏、内存卡接口、电阻触摸屏等。

## AWorks简介

AWorks是广州致远电子有限公司开发的一个创新的嵌入式软件平台，它把各种软件组件集成在一起，提供了数量庞大且高质量的服务。大量现成的软件不仅节省了用户的研发投入，还能提高产品的质量。通过简单的剪裁和配置之后，它甚至能够在只有几K内存的小资源平台上运行。尽管AWorks本身已经提供了众多高品质的可复用组件，AWorks同样也支持用户将自己的软件组件集成到平台中。AWorks对底层硬件做了良好的抽象和封装，最大程度上降低了上层应用与底层硬件的耦合。上层应用不再绑死在某款MCU上，有利于产品的升级和维护。

其关键特性如下：

* 1. 跨平台的应用编程接口；
  2. 物联网关键协议栈支持；
  3. 传感器支持；
  4. 电源管理；
  5. 网络通信；
  6. 丰富的扩展接口；
  7. 轻量级实时内核(Real-Time Kernel)；
  8. 图形配置工具；
  9. 多媒体支持；

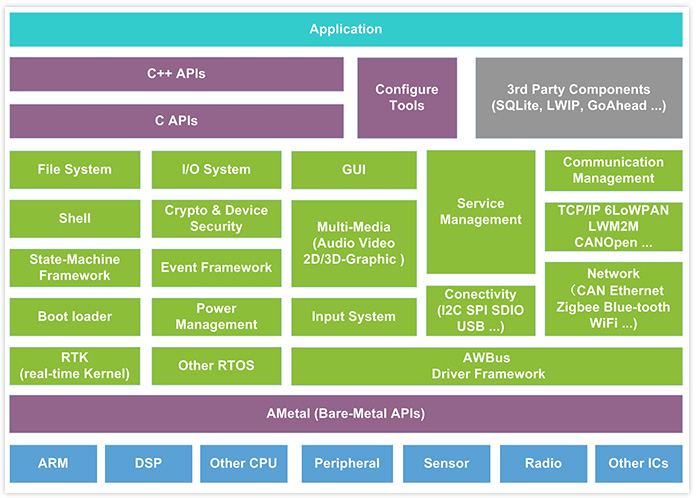


图8 AWorks架构

## emWin简介

emWin是新一代的图形界面，通过模拟器设计的界面与处理器和显示器无关，移植性好。适用于所有需要进行图像显示的设计 [[[19]](#endnote-19)]。emWin兼容性极好，能移植到任何操作系统或者裸机系统并稳定运行。且支持所有点阵型显示设备[[[20]](#endnote-20)]。其特性包括：

* 1. 支持基于所有显示控制器显示设备；
  2. 无显示控制器也可运行；
  3. 使用配置宏可支持任意接口；
  4. 显示尺寸可配置；
  5. 可在任何点上写入内容；
  6. 针对尺寸和速度优化；
  7. 利用编译时间切换进行优化；
  8. 显示控制器速度不高时，可在显存中绘制，加速显示；
  9. 结构清晰；
  10. 支持大于实际的虚拟显示；

## 系统任务设计

在AWorks平台中的任务和其它操作系统中的线程是一样的，只是名称不一样，所以系统任务的设计即是系统线程的设计。

由于本系统需要完成音频播放，且有比较丰富的人机交互，用户通过触摸屏进行操作的时候不能影响音乐的播放，而音频的解码也不能影响用户的操作。所以设计系统中任务如下：

### GUI任务

本任务中主要进行图形界面的初始化、处理界面相关的事件、轮询SD卡，U盘的插入与拔出。



图9 GUI任务流程图

### 触摸屏任务

本任务中处理的事情比较少，只是将触摸屏状态保存进emWin。



图10 触摸屏任务流程图

### 音乐播放任务

本任务完成音频的解码并播放以及音乐的切换。如图11所示，开始任务后，首先等待GUI初始化完成，然后使用默认参数打开声卡，经过一系列的初始化之后，进入音乐播放任务的大循环。在循环中检查是否需要暂停播放，如果需要暂停播放，则放弃CPU占用，直到需要播放音乐为止。之后判断当前需要播放的音乐格式，不同格式调用不同的播放器。一首歌播放完成或者中途切歌后，判断是否是切歌退出的，如是切歌退出，证明用户已经选定了下一曲需要播放的音乐，直接开始播放即可。否则是歌曲播放完成，那么就根据不同的播放模式切换下一首歌。



图11 音乐播放任务流程图

## 音频解码

本设计支持解码WAV、MP3、FLAC三种格式的音频。其中WAV和FLAC为无损音频，而MP3为有损压缩音频。

### WAV格式解码

在Windows操作系统中，微软为多数多媒体文件指定了通用的框架来储存内容。这种框架命名为资源互换文件格式，简称RIFF。比如WAV音频文件，RDI图片文件等都是属于RIFF文件。RIFF框架类似于树状，以块为基本结构。每个块由数据、数据长度和特征字符串构成。

表1 块的结构示意

|  |
| --- |
| 块的标志符（4字节） |
| 数据大小（4字节） |
| 数据 |

特征字符串为长度为四的字符串，数据长度是跟在之后的以字节为单位的数据的长度。但是数据长度自身也需要占用四字节的长度，因此一个块的长度为数据长度加八。通常来说，块与块之间级别相同，不能嵌套。但是RIFF和L1ST块除外。

表2 RIFF/LIST块框架

|  |  |
| --- | --- |
| RIFF/LIST标志符 | |
| 数据大小 | |
| 数据 | 格式/列表类型 |
| 子数据 |

所有按照这种结构存储数据的文件都是RIFF文件。

WAV文件也叫波形文件，是最早的数字音频文件格式。WAV格式基于RIFF(Resource Inter⁃change File Format)框架制定[[[21]](#endnote-21)]。用于储存声音的波形数据，被绝大部分的多媒体软件所支持。WAV文件直接保存声音的采样数据，因此对声音的还原度极高。但是WAV格式的音频文件大小极大，不利于在互联网上进行传播。WAV格式能够保存各种采样频率与位数的声音，标准的WAV音频文件格式与CD相同。采样频率为44.1KHz，量化位数为16位。所以音质极好，和CD差不多。

由表3可知，RIFF块由fmt块和data块组成。fmt块中保存了音频的格式信息。

表3 WAVE文件结构

|  |
| --- |
| 标志符（RIFF） |
| 数据大小 |
| 格式类型（"WAVE"） |
| "fmt" |
| sizeof(PCMWAVEFORMAT) |
| PCMWAVEFORMAT |
| "data" |
| 声音数据大小 |
| 声音数据 |

data块内保存了音频采样数据，这些音频数据按照fmt块内指定的格式储存。在单声道WAV文件中，音频数据是连续保存的，在多声道WAV文件中，音频数据是交替保存的。如采样位数为16位的单声道音频文件和多声道音频文件采样数据储存格式分别如表4和表5所示：

表4 16位单声道

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 采样一 | | 采样二 | | …… |
| 低字节 | 高字节 | 低字节 | 高字节 | …… |

表5 16位双声道

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 采样一 | | | |  |
| 左声道 | | 右声道 | | …… |
| 低字节 | 高字节 | 低字节 | 高字节 | …… |

本设计中，最高能播放采样率为192KHz，量化位数为32位的立体声WAV音频。由于网络上下载到的WAV格式歌曲通常都没有压缩，从文件内读取出的数据直接就是声卡所需要的PCM数据，所以从文件内读取的数据不需要送入解码器，直接发送给声卡播放即可。

### MP3格式

MP3全称为MPEG audio player 3。MPEG标准内包含音频标准与视频标准。音频编码的等级越高，编码算法越复杂。MPEG Layer3相比于Layer1和Layer2添加了MDCT变换，所以Layer3的频率分辨力是Layer2的十八倍。Layer3还增加了平均信息量编码算法，冗余信息量降低了[[[22]](#endnote-22)]。MP3的采样率、位数率和编码器决定了音质。MP3为了减少声音失真，使用了感官编码技术，舍弃掉了声音信息中人耳不敏感的部分，压缩比因此变得很大[[[23]](#endnote-23)]。编码时将音频数据转换到频域，然后进行分析，通过滤波器滤除不需要的部分，再打乱每位数据的排列顺序。经过以上步骤生成的MP3文件压缩比极高，并且解压之后的音频数据与原始音频数据差别不明显[[[24]](#endnote-24)]。

MP3文件由一系列的帧组成，每一帧都有帧头。文件主要分为3个部分，音频数据帧、TAGV2和TAGV1帧。TAGV2不是必须存在的，音频数据帧和TAGV1帧则必须有。TAGV1在音频文件的末尾，固定的128个byte的长度。存在歌手、作曲家、音乐专辑、时间等标签。TAGV2在音频文件的头部，包含了革命、歌手、作曲家、音乐专辑等标签，长度随标签的数量不同而相应的发生改变。扩展了TAGV1的数据量。

表6 MP3文件结构

|  |  |
| --- | --- |
| ID3V2 | 包含了作者，作曲，专辑等信息，长度不固定，扩展了ID3V1的信息量 |
| FRAME | 一系列的帧，个数由文件大小和帧长决定  每个FRAME的长度可能不固定，也可能固定，由位率bitrate决定  每个FRAME又分为帧头和数据实体两部分  帧头记录了mp3的位率，采样率，版本等信息，每个帧之间相互独立 |
| ID3V1 | 包含了作者，作曲，专辑等信息，长度为128BYTE |

每一个数据帧都有一个大小为4字节的帧头，紧跟着帧头之后有一定几率存在2个byte的CRC数据。CRC数据是否有由帧头信息的第16位决定。接下来就是音频采样数据。Header格式如表7所示。

表7 MP3帧头字节说明

| 名称 | 位长 | | 说明 |
| --- | --- | --- | --- |
| 同步信息 | 11 | 第1、2字节 | 所有位均为1，第1字节恒为FF |
| 版本 | 2 | 00-MPEG2.5 01-未定义10-MPEG2 11-MPEG1 |
| 层 | 2 | 00-未定义 01-Layer3 10-Layer2 11-Layer1 |
| CRC校验 | 1 | 0-校验 1-不校验 |
| 位率 | 4 | 第3字节 | 取样率，单位是kbps  "free"表示位率可变"bad"表示不允许值 |
| 采样频率 | 2 | 采样频率，对于MPEG-1：00-44.1kHz01-48kHz10-32kHz11-未定义  对于MPEG-2：00-22.05kHz01-24kHz10-16kHz11-未定义  对于MPEG-2.5：00-11.025kHz01-12kHz10-8kHz11-未定义 |
| 帧长调节 | 1 | 用来调整文件头长度，0-无需调整，1-调整 |
| 保留字 | 1 | 没有使用 |
| 声道模式 | 2 | 第4字节 | 表示声道，00-立体声Stereo 01-Joint Stereo 10-双声道 11-单声道 |
| 扩充模式 | 2 | 当声道模式为01时才使用。 |
| 版权 | 1 | 文件是否合法，0-不合法 1-合法 |
| 原版标志 | 1 | 是否原版，0-非原版 1-原版 |
| 强调方式 | 2 | 声音降噪压缩后再补偿分类。  00-未定义 01-50/15ms 10-保留 11-CCITTJ.17 |

帧长为每一帧音频数据压缩之后加上帧头大小的长度。而且用来占位的空数据也是被包含在内的。Layer I有4 byte的空数据，Layer II与Layer III则只有1 byte的空数据。当解码MP3文件的时候，必须计算帧长才能正确的读取到下一帧。

从帧头中解析出采样率、比特率和空数据填充长度之后可以计算出帧长度。

Lyaer I使用公式：

(4.1)

Lyer II和Lyaer III使用公式：

(4.2)

每一帧音频数据能够播放多少时间也需要计算，公式如下所示：

(4.3)

本设计中，MP3的播放做到了全参数支持，也就是说本播放器能播放MP3格式能支持到的最高格式(采样率48KHz，比特率320K)。主要采用libmad解码库。libmad为开源音频解码库。能够解码并生成24位采样的音频数据。进行MP3解码时解码精度高，效果好。但是libmad提供的接口函数比较少，只有顺序播放音乐的接口。所以如果需要快退快进功能需要自己实现。

### FLAC格式

FLAC为无损音频压缩编码。FLAC是一个非常流行的开源无损压缩音频编码格式。与那些有损压缩算法不同，FLAC格式不会破坏任何音频信息。所以使用FLAC格式播放音乐的效果与CD效果相近。

FLAC格式和MP3格式区别非常大，MP3格式在进行音频压缩的时候会删除某些数据，但FLAC格式不会。将FLAC文件解压缩为音频数据后，通过对比可以发现与压缩前的音频数据一模一样。FLAC格式的压缩与RAR的压缩的结果差不多。由于FLAC格式是为音频数据设计的压缩格式，所以压缩率远远大于一般的压缩文件。而且FLAC的使用也特别方便，直接用播放器打开就可以播放。

FLAC文件的结果和帧头结构如表8和表9所示。

表8 FLAC文件结构

|  |
| --- |
| 字符串“flaC” |
| 文件信息描述块 |
| 可选的其它描述信息块 |
| 一个以上的音频帧 |

表9 Header结构

|  |  |  |
| --- | --- | --- |
| 长度 | 功能 | 说明 |
| 14 | 同步字 | '11111111111110' |
| 1 | 保留 | 0：强制值 1：保留未来使用 |
| 1 | 分块策略 | 0：固定块大小，帧头包含帧的序号  1：块大小可变，帧头包含采样点序号 |
| 4 | 块内的采样数 | 块内的采样数 |
| 4 | 采样率 | 采样率 |
| 4 | 声道分配 | 声道分配 |
| 3 | 采样深度 | 采样深度 |

本设计中，FLAC格式最高能支持到采样率192KHz，量化位数24位。

### 音乐播放流程

本设计在音乐播放任务中需要调用音乐播放函数进行一首曲目的播放，所以需要一个能够解码并播放音频文件的函数。在该函数中，首先打开音频文件并获取文件大小，然后读取文件内的音频信息，如采样率、量化位数、比特率等。获取到音频的相关信息后，通过这些参数初始化声卡。接下来读取到的就是音频数据，将音频数据送入解码器之后得到PCM数据，然后输入到声卡播放，直到当前文件播放完成退出播放。

解码并播放一首音乐的程序流程如图12所示。



图12 音频文件播放流程图

## LRC歌词解析

### LRC格式简介

LRC是一个可以跟音乐文件做同步的文件格式。当一个音乐文件（如MP3、Vorbis或WMA等）被电脑音乐播放程序或现代的MP3随身听以及DVD播放机等设备播放时，歌词可以被同步显示出来。歌词文件通常和音乐文件有同样的文件名称，但是扩展名不同。例如song.mp3和song.lrc。LRC格式是一种文字格式，与电视和电影的字幕档很相似。由于中、日、韩文歌词在ANSI格式里可能产生乱码，可以使用UTF-8或Unicode文字编码避免。

LRC歌词标签主要分为两种：

第一种是形如[标识名:参数]的标识标签。标示标签主要分为以下几种：

[by: lrc歌词作者]、[ar:歌星名称]、 [al:专辑名称]、[ti:歌曲名称]、[offset:时间调整参数]（时间调整参数以ms为单位。用来调整整个歌词的时间误差，为有符号数，负值为整体延后，正值与负值相反，为整体提前）。

第二种是形如[mm:ss.ff]或者[mm:ss]的时间标签。时间标签从一行的头部开始，但是一行可以存在多个时间标签[[[25]](#endnote-25)]。当音乐播放到歌词指定的时间后，播放器根据相应算法同步显示歌词。

### LRC歌词解析

LRC歌词是以文本的形式存储在文件中的，解析LRC歌词文件时，通过C语言标准库函数内的标准输入函数可以按行读取文本数据，读取到一行数据后，判断该行内标示标签和时间标签的数量，然后分别解析标识标签和时间标签，然后存储到数组中，供其它函数使用。解析LRC歌词流程如图13所示。



图13 歌词解析流程图

## 用户界面设计

用户界面是人和机器之间进行信息传递与交换的桥梁。用户界面将机器内部的数据转化为人类能够理解的内容。用户界面存在的主要是为了用户能简单快捷的了解机器的信息以及操作机器。

在本设计中，用户界面是用户和系统实现信息交流的唯一途径，用户界面的设计直接关系到用户的体验。本设计的用户界面简单明了，通俗易懂，操作方便。

为了加快开发进度，UI部分使用是SEGGER提供的运行于Windows上的模拟器开发的。在模拟器中开发好的代码能够很容易的移植到对应的平台。

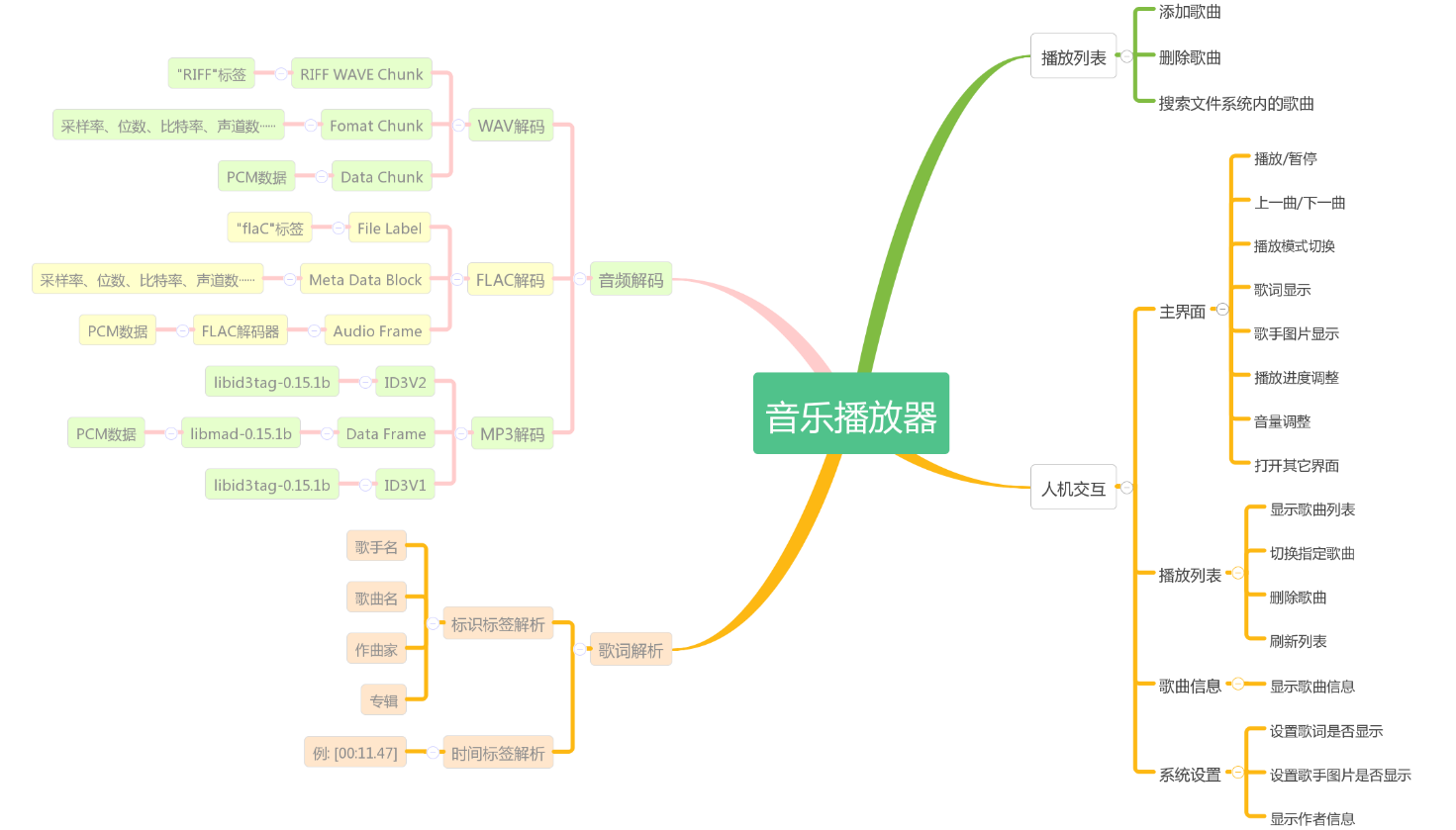


图14 用户界面设计思维导图

如图14所示，本设计拥有主界面、播放列表界面、歌曲信息界面和系统设置界面四大界面，系统开机时默认显示主界面，其它三大界面可以通过点击主界面上的相关按钮呼出。

### 主界面设计



图15 主界面

图15是在模拟器中开发的主界面的显示效果，可以看到在界面的左下方有三个按键，分别是播放键、上一曲、下一曲。点击播放键之后播放键变为暂停键，如图16所示，此时系统开始播放音乐，进度条开始滚动，且歌词也会同步显示。当需要暂停播放的时候，点击暂停键，暂停键会变为播放键，同时暂停播放音乐。播放键右边分别是上一曲和下一曲，点击之后系统根据当前的播放模式自动切换到相应歌曲。



图16 主界面暂停键

在主界面的中间靠下方为播放进度条，该进图条左边显示当前播放的时间，右边显示当前歌曲总时间，在播放音乐的时候，可以通过拖动该进度条来调整音乐的播放进度。

主界面的右下角为音量调整区，有一小喇叭图标指示该区域为调整音量的功能，小喇叭右侧为音量调整进度条，拖动该进度条可以调整音乐播放的音量。

主界面的背景为当前播放音乐的歌手图片，在歌手图片的上方重叠显示歌词。歌词会根据当前播放的音乐自动滑动，方便用户观看。当用户需要调整到指定歌词行播放的时候，可以直接拖动歌词，此时界面上会自动显示歌词调整控件，如果两秒内无下一步操作的话，取消本次调整并隐藏歌词调整控件。或者直接点击界面中部靠右的确定按钮切换到指定歌词行播放，歌词调整界面如图17所示。



图17 歌词调整界面

主界面靠下方有一灰黑色半透明横条，横条上方有四个按钮，分别为播放模式切换按钮，歌曲信息界面呼出按钮，系统设置界面呼出按钮，播放列表界面呼出按钮。本设计支持三种音乐播放模式，点击模式切换按钮时，播放模式在如图18所示中的三种模式循环切换。



图18 三种播放模式

### 播放列表界面设计



图19 播放列表界面

当用户点击主界面中的播放列表按键时，屏幕右侧将显示播放列表，播放列表上部为艺术字”PlayList”，提示用户当前窗体的功能与作用，中部为播放列表中的歌曲，每一行显示一首歌的信息，左方是歌曲序号，中部为歌曲名称，右侧为歌曲总时长。该部分可以上下滑动，查看系统内所有的歌曲，需要选定歌曲时，直接点击歌曲名称即可，如果点击的是现在正在播放的音乐，则播放状态在播放和暂停之间相互切换。

在播放列表的下部有三个按钮，分别为删除当前歌曲，刷新播放列表和关闭播放列表。点击删除播放当前歌曲按钮时，会弹出确认对话框，如果用户确认删除，则删除当前歌曲。点击刷新按钮时，系统自动扫描SD卡和U盘内的歌曲并添加到播放列表中。最后，可以通过关闭播放列表按钮来关闭播放列表。

### 歌曲信息界面

歌曲信息界面如图20所示，分别显示了歌曲名称、歌手名称、时长、格式、采样率、比特率、通道数、位数和文件大小信息，方便用户查看。



图20 歌曲信息界面

### 系统设置界面

系统设置界面如图21所示，需要设置的地方比较少，所以界面比较简介，可以设置歌词和歌手图片分别能不能显示，然后还显示了一些作者信息。



图21 系统设置界面

# 测试与结果

本设计完成后实物如图22所示，界面的显示效果与模拟器基本相同。能流畅的播放音乐，且音质不错。

## CPU使用情况



图22 作品实物图

音频解码都是由软件进行解码，解码不同格式时CPU占用率如表10所示。可以看到不需要解码的WAV格式CPU占用率最低，只有2-3%。MP3格式CPU占用率最高，为9-12%，这是因为MP3格式需要删除大量的数据，压缩和解压算法比较复杂，需要较多的计算资源解码。FLAC格式CPU占用率比MP3格式稍低，为8-11%。

表10 音频解码CPU占用率

|  |  |
| --- | --- |
| 格式 | CPU占用率 |
| WAV | 2-3% |
| MP3 | 9-12% |
| FLAC | 8-11% |

## 设计完成情况

### 要求

1.使用i.MX283处理器设计；

2.基于AWorks软件平台；

3.使用emWin界面库；

4.支持多语言显示；

5.支持SD卡播放；

6.支持U盘播放；

7.支持WAV格式播放；

8.TFT液晶屏显示；

9.简单的人机交互；

扩展：支持MP3格式、支持歌词同步显示。

### 完成情况

经过漫长的设计、制作过程，本设计完成了所有的设计要求，相关功能一切正常。

# 体会与致谢

因为公司也要求做毕业设计且参加公司的答辩作为是否转正的依据，所以本次毕业设计是将公司给的题目自命题作为学校的毕业设计题目的。并且在本学期到公司去实习了接近三个月，在实习期间完成了本次毕业设计的实物部分。

从2月13号到公司就开始了本次毕业设计的制作，由于相关资料没有到位，本人是从设计用户界面开始制作的，通过在电脑的模拟器上开发用户界面，大大的加快了开发的进度，因为这免去了在实物上开发耗时颇多的下载步奏。通过两个周的开发，本人更加深刻的了解了音乐播放器所需要的功能和很多细节，熟悉了emWIn的使用，为后续的开发打下了良好的基础。

用户界面开发完成大部分时，公司将需要的嵌入式操作系统AWorks下发，熟悉几天的之后，正式开始了代码的编写。最后通过2个月的努力，终于将实物制作完成，通过了公司的答辩。

经过这几个月的制作，本人了解了几种常用的音频格式的解码流程，也对数字音频的播放流程有了清晰的认识。锻炼了动手能力和查阅资料的能力。在此期间遇到了各种困难，但是还是努力的解决了这些问题，在此过程中，离不开那些对本人有帮助的同学，也离不开指导老师对本人的帮助。在此，本人对这些同学和指导老师表示衷心的感谢！

参考文献

附录

GUI任务源代码：

1. /\*\*
2. \* \brief GUI任务入口
3. \*
4. \* \param[in] p\_arg 参数
5. \*
6. \* \return 无
7. \*/
8. **void** player\_gui\_task\_entry (**void** \*p\_arg)
9. {
10. uint8\_t sd\_unmount\_test = 0;
11. unsigned **char**  temp = 0;
12. dialog\_info\_t  info;
13. **struct** aw\_dir \*p\_dirp;
14. **int**            recv = 0;
16. WM\_SetCreateFlags(WM\_CF\_MEMDEV); /\* 窗口使用存储设备 \*/
17. GUI\_Init();                      /\* 初始化emwin \*/
19. aw\_touch\_gui\_calibrate(AW\_TS\_LIB\_FIVE\_POINT, gui\_calibrate\_cb);
21. /\* 初始化xbf字体 \*/
22. #if 1
23. xbf\_init((**void** \*)g\_xbf\_adobe\_hei\_std);
24. #else
25. xbf\_init((**void** \*)g\_ttf\_res);
26. #endif
27. GUI\_SetBkColor(GUI\_BLACK);
28. GUI\_Clear();
30. GUI\_UC\_SetEncodeUTF8();
31. GUI\_SetTextMode(GUI\_TEXTMODE\_TRANS);  /\* 透明文本 \*/
33. WM\_MOTION\_Enable(1);
34. BUTTON\_SetReactOnLevel();
36. WIDGET\_SetDefaultEffect(&WIDGET\_Effect\_None); /\* 设置控件风格 \*/
38. /\* 创建主菜单 \*/
39. g\_h\_winmain\_0 = GUI\_CreateDialogBox(g\_dialog\_create, /\* 控件资源表 \*/
40. GUI\_COUNTOF(g\_dialog\_create),    /\* 控件数量 \*/
41. main\_dlg\_callback, /\* 对话框过程函数的回调函数 \*/
42. WM\_HBKWIN,         /\* 父窗口的句柄\*/
43. 0,                 /\* 相对于父窗口的 X 轴位置 \*/
44. 0);                /\* 相对于父窗口的 Y 轴位置 \*/
46. /\* 创建主菜单定时器 \*/
47. g\_h\_timer\_0 = WM\_CreateTimer(g\_h\_winmain\_0, 0, 500, 0);
49. /\* 歌词窗体容器 \*/
50. g\_h\_winhead = WM\_CreateWindowAsChild(
51. 0,              /\* 父窗口在窗口坐标中的左上 X 位置 \*/
52. 0,              /\* 父窗口在窗口坐标中的左上 Y 位置 \*/
53. 480,            /\* 窗口的 X 尺寸 \*/
54. 205,            /\* 窗口的 Y 尺寸 \*/
55. g\_h\_winmain\_0,  /\* 父窗口的句柄 \*/
56. WM\_CF\_SHOW | WM\_CF\_HASTRANS , /\* 窗口创建标识 \*/
57. dummy\_callback, /\* 回调例程的指针 \*/
58. 0);             /\* 要分配的额外字节数 \*/
60. /\* 创建歌词窗体 \*/
61. g\_h\_winlyrics = WM\_CreateWindowAsChild(
62. 0,               /\* 父窗口在窗口坐标中的左上 X 位置 \*/
63. 0,               /\* 父窗口在窗口坐标中的左上 Y 位置 \*/
64. 480,             /\* 窗口的 X 尺寸 \*/
65. 0,               /\* 窗口的 Y 尺寸 \*/
66. g\_h\_winhead,     /\* 父窗口的句柄 \*/
67. WM\_CF\_SHOW | WM\_CF\_HASTRANS | WM\_CF\_MOTION\_Y, /\* 标识 \*/
68. lyrics\_callback, /\* 回调例程的指针 \*/
69. 0);              /\* 要分配的额外字节数 \*/
71. /\* 创建歌词定时器 \*/
72. g\_h\_timer\_1 = WM\_CreateTimer(g\_h\_winlyrics, 1, 0, 0);
73. g\_h\_timer\_2 = WM\_CreateTimer(g\_h\_winlyrics, 2, 0, 0);
75. /\* 创建播放列表窗体 \*/
76. g\_h\_winlist\_0 = music\_list\_create(g\_h\_winmain\_0); /\* 创建播放列表窗口 \*/
77. WM\_HideWindow(g\_h\_winlist\_0);                     /\* 隐藏播放列表窗口 \*/
79. g\_h\_setting\_0 = setting\_create(g\_h\_winmain\_0);
80. WM\_HideWindow(g\_h\_setting\_0);                     /\* 隐藏系统设置窗口 \*/
82. IMAGE\_SetJPEG(g\_h\_image\_0,
83. g\_res\_background\_2,
84. **sizeof**(g\_res\_background\_2));
86. /\* 将之前创建的窗口绘制出来 \*/
87. GUI\_Exec();
89. /\* 释放GUI初始化完成信号量 \*/
90. AW\_SEMB\_GIVE(g\_play\_ctrl.gui\_ok);
92. dialog\_info\_init(&info);
93. info.font\_title  = &g\_font14;
94. info.font\_text   = &g\_font14;
95. info.font\_btn[0] = &g\_font14;
96. info.font\_btn[1] = &g\_font14;
97. info.btn\_color\_text[1] = 0xD4881E;
99. AW\_FOREVER {
100. **if** ((p\_dirp = aw\_opendir("/udisk")) == NULL) {
101. **if** (1 == g\_is\_mount\_udisk) {
102. aw\_umount("/udisk", 0);
103. g\_is\_mount\_udisk = 0;
105. play\_list\_init(g\_play\_list,
106. **sizeof**(g\_play\_list) / **sizeof**(g\_play\_list[0]),
107. 1,
108. SEARCH\_RANGE\_USB);
110. winlist\_recreate();
112. strcpy(info.title, "");
113. strcpy(info.text, "USB存储器已移除");
114. strcpy(info.btn\_text[0], "确定");
115. info.btn\_num = 1;
116. info.blocking = 0;
117. dialog\_show(130,
118. 76,
119. info.rect.x1 + 1,
120. info.rect.y1 + 1,
121. g\_h\_winmain\_0,
122. WM\_CF\_SHOW | WM\_CF\_HASTRANS | WM\_CF\_STAYONTOP,
123. &info);
124. }
125. } **else** {
126. g\_is\_mount\_udisk = 1;
127. aw\_closedir(p\_dirp);
128. }
130. **if** (!g\_is\_mount\_udisk) {
132. /\* 文件系统挂载 \*/
133. recv = aw\_mount("/udisk", "/dev/ms2-ud0", "vfat", 0);
134. **if** (AW\_OK == recv) {
135. g\_is\_mount\_udisk = !g\_is\_mount\_udisk;
136. AW\_INFOF(("Find a new disk\n"));
138. strcpy(info.title, "发现新的USB存储器");
139. strcpy(info.text, "是否搜索歌曲");
140. strcpy(info.btn\_text[0], "取消");
141. strcpy(info.btn\_text[1], "确定");
142. info.btn\_num = 2;
143. info.blocking = 1;
144. **if** (dialog\_show(130,
145. 76,
146. info.rect.x1 + 1,
147. info.rect.y1 + 1,
148. g\_h\_winmain\_0,
149. WM\_CF\_SHOW | WM\_CF\_HASTRANS | WM\_CF\_STAYONTOP,
150. &info) == 2)
151. {
152. play\_list\_init(g\_play\_list,
153. **sizeof**(g\_play\_list) / **sizeof**(g\_play\_list[0]),
154. 1,
155. SEARCH\_RANGE\_USB);
157. winlist\_recreate();
158. play\_list\_write(g\_play\_list,
159. &g\_list\_state,
160. PLAY\_LIST\_MAX\_SIZE);
161. }
162. }
163. }
164. **if** (((p\_dirp = aw\_opendir("/sd0")) == NULL) || (1 == sd\_unmount\_test)) {
165. **if** (1 == g\_is\_mount\_sdcard) {
166. aw\_umount("/sd0", 0);
167. g\_is\_mount\_sdcard = 0;
169. play\_list\_init(g\_play\_list,
170. **sizeof**(g\_play\_list) / **sizeof**(g\_play\_list[0]),
171. 0,
172. SEARCH\_RANGE\_SD);
174. winlist\_recreate();
176. strcpy(info.title, "");
177. strcpy(info.text, "SD卡已移除");
178. strcpy(info.btn\_text[0], "确定");
179. info.btn\_num = 1;
180. info.blocking = 0;
181. dialog\_show(130,
182. 76,
183. info.rect.x1 + 1,
184. info.rect.y1 + 1,
185. g\_h\_winmain\_0,
186. WM\_CF\_SHOW | WM\_CF\_HASTRANS | WM\_CF\_STAYONTOP,
187. &info);
188. }
189. } **else** {
190. g\_is\_mount\_sdcard = 1;
191. aw\_closedir(p\_dirp);
192. }
194. **if** ((!g\_is\_mount\_sdcard) && (0 == sd\_unmount\_test)) {
196. /\* 文件系统挂载 \*/
197. recv = aw\_mount("/sd0", "/dev/sd0", "vfat", 0);
198. **if** (AW\_OK == recv) {
199. g\_is\_mount\_sdcard = !g\_is\_mount\_sdcard;
200. AW\_INFOF(("Find a new sdcard\n"));
202. /\* 初始化播放列表 \*/
203. recv = play\_list\_read(g\_play\_list,
204. &g\_list\_state,
205. PLAY\_LIST\_MAX\_SIZE);
206. **if** (0 == g\_is\_mount\_udisk) {
207. usb\_play\_list\_clean(g\_play\_list, PLAY\_LIST\_MAX\_SIZE);
208. }
209. **if** (AW\_OK == recv) {
210. winlist\_recreate();
211. strcpy(info.title, "发现新的SD卡");
212. strcpy(info.text, "播放列表读取完成");
213. strcpy(info.btn\_text[0], "确定");
214. info.btn\_num = 1;
215. info.blocking = 0;
216. dialog\_show(130,
217. 76,
218. info.rect.x1 + 1,
219. info.rect.y1 + 1,
220. g\_h\_winmain\_0,
221. WM\_CF\_SHOW | WM\_CF\_HASTRANS | WM\_CF\_STAYONTOP,
222. &info);
223. GUI\_Delay(1500);
224. dialog\_close();
225. } **else** {
226. dialog\_info\_init(&info);
227. info.font\_title  = &g\_font14;
228. info.font\_text   = &g\_font14;
229. info.font\_btn[0] = &g\_font14;
230. info.font\_btn[1] = &g\_font14;
231. info.btn\_color\_text[1] = 0xD4881E;
232. strcpy(info.title, "发现新的SD卡");
233. strcpy(info.text, "是否搜索歌曲");
234. strcpy(info.btn\_text[0], "取消");
235. strcpy(info.btn\_text[1], "确定");
236. info.btn\_num = 2;
237. info.blocking = 1;
238. **if** (dialog\_show(
239. 130,
240. 76,
241. info.rect.x1 + 1,
242. info.rect.y1 + 1,
243. g\_h\_winmain\_0,
244. WM\_CF\_SHOW | WM\_CF\_HASTRANS | WM\_CF\_STAYONTOP,
245. &info) == 2)
246. {
247. play\_list\_init(g\_play\_list,
248. **sizeof**(g\_play\_list) /
249. **sizeof**(g\_play\_list[0]),
250. 1,
251. SEARCH\_RANGE\_SD);
253. winlist\_recreate();
254. play\_list\_write(g\_play\_list,
255. &g\_list\_state,
256. PLAY\_LIST\_MAX\_SIZE);
257. }
259. }
261. /\* 获取配置文件 \*/
262. config\_read(&g\_config\_info);
263. **if** ((1 == g\_config\_info.is\_effect) &&
264. (g\_config\_info.play\_idx <
265. g\_list\_state.play\_list\_count)) {
266. volume\_set(g\_config\_info.volume);
267. g\_play\_ctrl.play\_idx = g\_config\_info.play\_idx;
269. } **else** {
270. volume\_set(50);
271. g\_play\_ctrl.play\_idx = 0;
272. g\_config\_info.lrc\_show = 1;
273. g\_config\_info.singer\_show = 1;
274. g\_config\_info.is\_effect = 1;
275. }
276. }
277. }
279. GUI\_Delay(500);
280. }
281. }

触摸屏任务源代码：

1. /\*\*
2. \* \brief 触摸屏处理任务入口
3. \*
4. \* \param[in] p\_arg 参数
5. \*
6. \* \return 无
7. \*/
8. **void** touch\_screen\_task\_entry (**void** \*p\_arg)
9. {
10. GUI\_PID\_STATE         th;
11. **struct** aw\_touch\_state ts\_state;
13. th.Layer = 0;
15. AW\_FOREVER {
16. aw\_touch\_get\_log\_state(&ts\_state);  /\* 获取抽象触摸设备当前逻辑坐标状态 \*/
18. th.x = ts\_state.x;
19. th.y = ts\_state.y;
20. th.Pressed = ts\_state.pressed;
22. GUI\_TOUCH\_StoreStateEx(&th);        /\* 把结果存储进emwin \*/
24. aw\_touch\_exec();                    /\* 定期对触摸设备进行轮询 \*/
26. aw\_mdelay(10);
27. }
28. }

音乐播放器任务源代码：

1. /\*\*
2. \* \brief 音乐播放任务入口
3. \*/
4. **void** play\_task\_entry (**void** \*p\_arg)
5. {
6. aw\_time\_t time;
8. /\* 等待GUI初始化完成 \*/
9. AW\_SEMB\_TAKE(g\_play\_ctrl.gui\_ok, AW\_SEM\_WAIT\_FOREVER);
11. /\* 构建音频设备 \*/
12. aw\_imx28\_snd\_mkdev(&g\_snd\_dev,                     /\* 音频设备 \*/
13. 0,                              /\* SAIF0 \*/
14. SOUND\_DEVICE,                   /\* 声卡 \*/
15. 256,                            /\* 一次中断传输数据大小 \*/
16. 128,                            /\* 内部环形缓冲区大小 \*/
17. 16,                             /\* 位数 \*/
18. 2,                              /\* 通道数 \*/
19. 44100,                          /\* 采样率 \*/
20. AW\_IMX28\_SND\_STREAM\_PLAYBACK);  /\* 播放音频 \*/
22. /\* 打开声卡 \*/
23. **if** (aw\_imx28\_snd\_open(&g\_snd\_dev) != AW\_OK) {
24. AW\_ERRF(("aw\_imx28\_snd\_open failed\n"));
25. }
27. AW\_FOREVER {
29. /\* 播放索引合法性检查 \*/
30. **if** ((0 == g\_list\_state.play\_list\_count) ||
31. (g\_play\_ctrl.play\_idx >= g\_list\_state.play\_list\_count)) {
32. g\_play\_ctrl.play\_idx = 0;
33. g\_play\_ctrl.paly\_state = PALY\_STATE\_PAUSE;
34. }
36. **if** (g\_play\_ctrl.event & PLAY\_EVENT\_STOP) {
37. g\_play\_ctrl.paly\_state = PALY\_STATE\_STOP;
38. g\_play\_ctrl.event &= ~PLAY\_EVENT\_STOP;
39. }
41. **if** (g\_play\_ctrl.paly\_state != PALY\_STATE\_PLAY) {
42. aw\_mdelay(100);
43. } **else** {
44. refresh\_volume();
46. /\* 保存当前播放索引以及音量配置 \*/
47. g\_config\_info.volume = volume\_get();
48. g\_config\_info.play\_idx = g\_play\_ctrl.play\_idx;
49. config\_write(&g\_config\_info);
51. /\* 初始化参数 \*/
52. g\_play\_ctrl.event = PLAY\_EVENT\_NULL;
53. g\_play\_ctrl.current\_time = 0;
54. g\_play\_ctrl.info = g\_play\_list[g\_play\_ctrl.play\_idx];
56. AW\_INFOF(("play music %s\n", g\_play\_ctrl.info.name));
58. /\* 根据歌曲的格式调用不同的解码器 \*/
59. **switch** (g\_play\_ctrl.info.type) {
61. **case** MUSIC\_TYPE\_WAV:  /\* WAV播放 \*/
62. **if** (AW\_OK != wav\_play(g\_play\_ctrl.info.file\_name)) {
63. g\_play\_ctrl.play\_idx++;
64. **continue**;
65. }
66. **break**;
68. **case** MUSIC\_TYPE\_MP3:  /\* MP3播放 \*/
69. **if** (AW\_OK != mp3\_play(g\_play\_ctrl.info.file\_name)) {
70. g\_play\_ctrl.play\_idx++;
71. **continue**;
72. }
74. **break**;
76. **case** MUSIC\_TYPE\_FLAC: /\* FLAC播放 \*/
77. **if** (AW\_OK != flac\_play(g\_play\_ctrl.info.file\_name)) {
78. g\_play\_ctrl.play\_idx++;
79. **continue**;
80. }
81. **break**;
83. **default**:
84. ; /\* VOID \*/
85. }
87. /\* 计算下一曲索引 \*/
88. **if** (g\_play\_ctrl.event & PLAY\_EVENT\_CHOOSE) {
89. ; /\* VOID \*/
90. } **else** {
91. **switch** (g\_play\_ctrl.paly\_mode) {
93. **case** PALY\_MODE\_RANDOM: /\* 随机播放 \*/
95. /\* 以当前时间为种子产生随机数 \*/
96. aw\_time(&time);
97. srandom(time);
98. g\_play\_ctrl.play\_idx = (**double**)random() / RAND\_MAX \*
99. (g\_list\_state.play\_list\_count - 1);
100. **break**;
102. **case** PALY\_MODE\_ORDER:  /\* 顺序播放 \*/
104. /\* 上一曲 \*/
105. **if** (g\_play\_ctrl.event & PLAY\_EVENT\_PREV) {
106. g\_play\_ctrl.play\_idx--;
107. **if** (g\_play\_ctrl.play\_idx < 0) {
108. g\_play\_ctrl.play\_idx =
109. g\_list\_state.play\_list\_count - 1;
110. }
112. /\* 下一曲 \*/
113. } **else** {
114. g\_play\_ctrl.play\_idx++;
115. **if** (g\_play\_ctrl.play\_idx >=
116. g\_list\_state.play\_list\_count) {
117. g\_play\_ctrl.play\_idx = 0;
118. }
119. }
120. **break**;
122. **case** PALY\_MODE\_SINGLE: /\* 单曲循环 \*/
124. /\* 上一曲 \*/
125. **if** (g\_play\_ctrl.event & PLAY\_EVENT\_PREV) {
126. g\_play\_ctrl.play\_idx--;
127. **if** (g\_play\_ctrl.play\_idx < 0) {
128. g\_play\_ctrl.play\_idx =
129. g\_list\_state.play\_list\_count - 1;
130. }
132. /\* 下一曲 \*/
133. } **else** **if** (g\_play\_ctrl.event & PLAY\_EVENT\_NEXT) {
134. g\_play\_ctrl.play\_idx++;
135. **if** (g\_play\_ctrl.play\_idx >=
136. g\_list\_state.play\_list\_count) {
137. g\_play\_ctrl.play\_idx = 0;
138. }
140. /\* 单曲循环 \*/
141. } **else** {
142. ; /\* VOID \*/
143. }
144. **break**;
146. **default**:
147. ; /\* VOID \*/
148. }
149. }
150. }
151. }
152. }

1. []刘垣,李外云,赵嘉怡.基于STC单片机WAVE音乐播放器的设计与实现[J].科技创新与应用,2015(34):50-51. [↑](#endnote-ref-1)
2. []袁卫.基于CortexM3的音频播放器的设计[J].电子设计工程.2015(3):169-171. [↑](#endnote-ref-2)
3. []E Kalpana, V Sridhar,MR Prasad. MPEG-1/2 audio layer-3(MP3) ON THE RISC based ARM PROCESSOR(ARM92SAM9263)[J]. International Journal of Computer Science Engineering, 2012:79-107. [↑](#endnote-ref-3)
4. []Wonchul Lee, Kisun You, Wonyong Sung. Software optimization of MPEG audio layer-III for a 32 bit RISC processor [J]. Conference on Circuits & Systems, 2002, 1:435-438. [↑](#endnote-ref-4)
5. []K Govardhanaraj, D Nagaraj. Intelligent music player with ARM7 [J]. Global Conference on Communication Technologies, 2015:323-326. [↑](#endnote-ref-5)
6. []於少峰,严菊明,胡晨.基于AC97标准的嵌入式音频系统设计与实现[J].电子器件.2004,27(4):733-736. [↑](#endnote-ref-6)
7. []陈自龙,周书杰,汤勇明.基于ARM嵌入式系统的高保真无损音乐播放器设计[J].电子器件.2012,36(6):692-698. [↑](#endnote-ref-7)
8. []焦正才,樊文侠.基于Qt/Embedded的MP3音乐播放器的设计与实现[J].电子设计工程,2012,20(7):148-150. [↑](#endnote-ref-8)
9. []汪永好,周延森.基于嵌入式Linux的MP3播放器的设计与实现[J].计算机工程与设计,2009,30(17):3948-3949. [↑](#endnote-ref-9)
10. []王灵芝,陈雷松.基于嵌入式Linux与Qt的MP3播放器的设计[J].漳州师范学院学报:自然科学版,2009(1):39-43. [↑](#endnote-ref-10)
11. []黄默,陈云.基于现场可编程FPGA的随身听设计[J].九江职业技术学院学报.2010(4):22-24. [↑](#endnote-ref-11)
12. []何谐.FAT32文件系统在Cortex\_M3音乐播放器中的应用[J].单片机与嵌入式系统应用.2013,13(6):71-73. [↑](#endnote-ref-12)
13. []杨雪梅,张慧.基于STM32的音乐播放器[J].信息工程.2016(3):136-137. [↑](#endnote-ref-13)
14. []李伟,张真,范文豪.基于STM32微控制器的mp3播放器设计[J].现代电子技术,2015(4):118-120. [↑](#endnote-ref-14)
15. []瞿兵,阳泳,胡湘娟.基于嵌入式的音乐播放器设计与论述[J].电子世界.2015(21):174-174. [↑](#endnote-ref-15)
16. []章坚武,董平,马勇.一种嵌入式多媒体播放器的硬件设计与实现[J].电子器件,2006,29(4):1123-1125. [↑](#endnote-ref-16)
17. []Davis B, Mudge T, Jacob B. DDR2 and Low Latency Variants[J]. Solving the Memory Wall Problem Workshop, 2000. [↑](#endnote-ref-17)
18. []刘缶. 基于Linux的嵌入式音乐播放器设计与实现[D].电子科技大学,2014. [↑](#endnote-ref-18)
19. []钟涛,祝玲. 基于STM32单片机的emWin系统设计[J]. 中国新通信,2017,07:53-54. [↑](#endnote-ref-19)
20. []肖林京,于鹏杰,于志豪.基于STM32和emWin图形库的液晶显示系统设计[J].电视技术,2015,39(1):39-42. [↑](#endnote-ref-20)
21. []罗海涛.wav音频文件格式分析与数据获取[J].计算机工程应用技术.2016,12(27):211-213. [↑](#endnote-ref-21)
22. []Liu Q, Sung A H, Qiao M. Detecting information-hiding in WAV audios[C]. International Conference on Pattern Recognition. IEEE, 2009:1-4. [↑](#endnote-ref-22)
23. []张宁,李方圆,鹿珂珂.基于FPGA的MP3编码系统设计[J].仪表技术.2015(11):27-31. [↑](#endnote-ref-23)
24. []汪勇,熊前兴.MP3文件格式解析[J].计算机应用与软件.2004,21(12):126-128. [↑](#endnote-ref-24)
25. []张公礼,闫海燕,王东明.基于嵌入式Linux的MP3歌词同步显示[J].电子器件,2008,31(5):1463-1465. [↑](#endnote-ref-25)