重慶理工大學

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

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的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。

本系统采用基于飞思卡尔 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，最后编写相关驱动程序。软件部分包含四个模块，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的音频解码功能。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，采用线程间通信方式设计并实现了以触摸屏控制的音乐播放功能。

经测试后此设备运行良好，播放功能完善，且具有低能耗、高性能的优点，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

At this stage of the music player for the loss of music support has been very good, but can support lossless format music player is rarely heard. So the urgent need to play a lossless music, and small size, powerful music player, the occupation of portable devices of high-end market.

The system uses the Freescale i.MX283 microprocessor EasyARM-i.MX283A development kit as the target platform development board. The construction of the system development environment consists of three parts. First, we build the cross compiler environment on the embedded development board EasyARM-i.MX283A platform, then transplant the AWorks software platform, and finally write the related driver. The software part contains four modules, audio decoding module, LRC lyrics analysis module, playlist module and human-computer interaction module. In this paper, the audio decoding process is analyzed in detail, and the audio decoding function in software design is realized by open source decoder. LRC lyrics analysis module is a detailed analysis of the LRC lyrics format, the design and implementation of the lyrics analysis module. Playlist module to achieve the file system within the music file display and management. Finally, after the realization of decoding and human-computer interaction, the use of inter-thread communication design and implementation of the touch screen control of the music player.

After testing this equipment is running well, playback features perfect, and has low power consumption, high performance advantages, especially with ES9018K2M audio DAC chip, with high sound quality, in line with the pursuit of high quality music.

Key words: Music player Aworks emWin WAV MP3 FLAC

1 绪 论

1.1 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，智能手机集成了手机、相机、上网本、音视频播放器等功能，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。

1.2 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]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。

2) 使用高性能 32 位单片机软件解码的方案

袁卫[2]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]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]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]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。

3) 基于 Linux 操作系统的方案

於少峰等[6]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]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]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]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]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。

4) 使用 FPGA 加速的方案

黄默等[11]使用 FPGA 连接音频解码芯片 STMP3410 的方式实现了音乐的播放，由于 FPGA 具有大容量、可重构及有性能强劲的 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。

5) 应用专用音频解码芯片的方案

何谐[12]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]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]与杨雪梅的方案基本一致，能够实现播放、停止、音量设置和切歌功能。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]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]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。

1.3 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 储存器以及内存卡内的音乐文件，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。

2 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。

2.1 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。

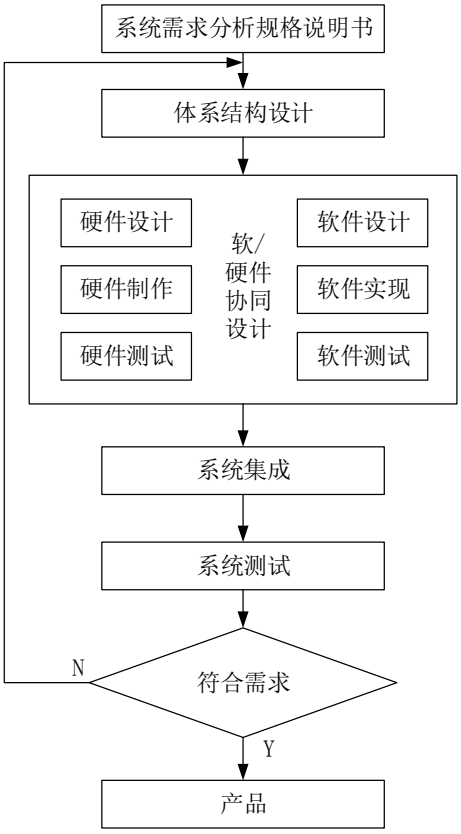

图 2.1 嵌入式系统开发流程

如图 2.1 所示，嵌入式开发流程分为以下 5 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，硬件部分主要由处理器以及配套的外围电路和将数字音频信号转换为声音信号的声卡组成。软件有最底层的驱动程序，中间层的操作系统和最上面的应用层。

2.2 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 2.2 所示。

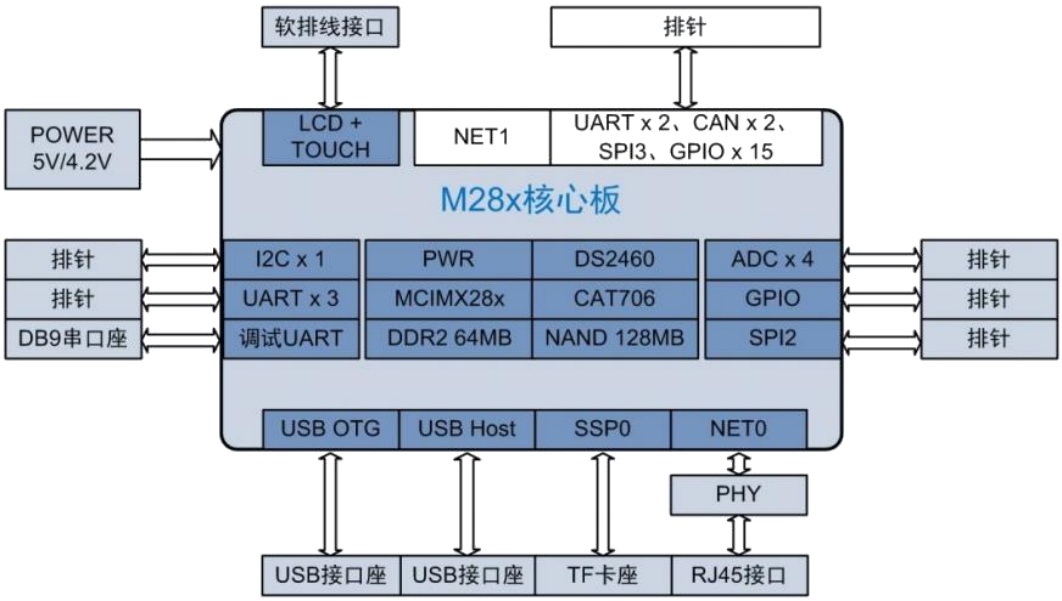

图 2.2 硬件系统框架图

2.3 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 2.3 所示。

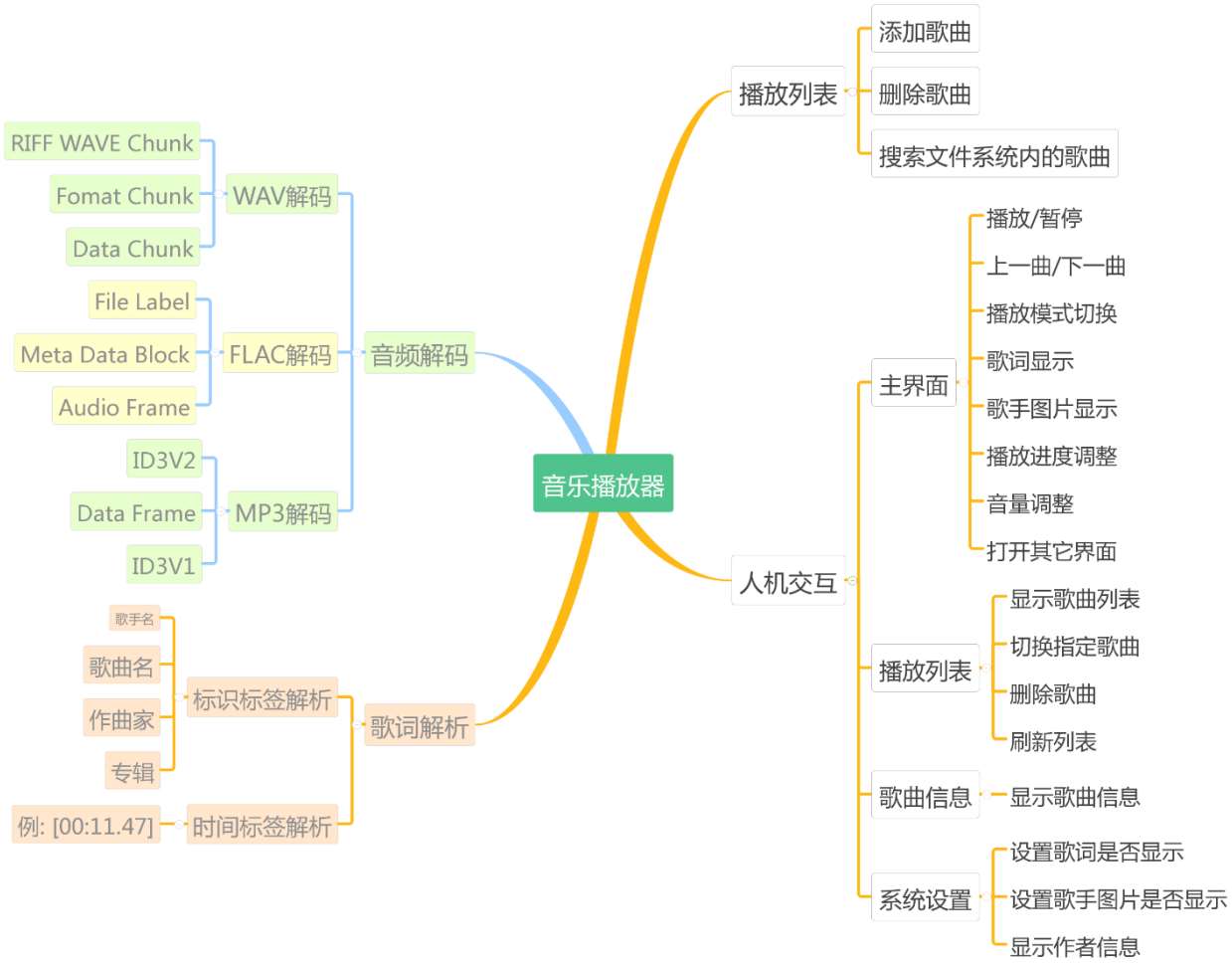

图 2.3 系统软件框架思维导图

3 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。

3.1 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。

3.2 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。

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。

3.3 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]。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 3.1。

3.4 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，应用场景广泛，由于其有非常低

的功耗，很适合在便携式设备中使用，而且优秀的音质也让 ES9018K2M 非常适合在专业系统中使用。

ES9018K2M 使用时域抖动消除器和 32 位的 Hyperstream DAC 架构。信噪比(DNR)高达 127dB，THD+N 则低至-120dB。所以音质能满足大部分要求严格的用户。ES9018K2M 通过 DSD 接口能够输入 11.2MHz 的数据，I2S 接口则可以输入多达 32 位 384kHz 的 PCM 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。ES9018K2M 应用电路如图 3.2 所示。

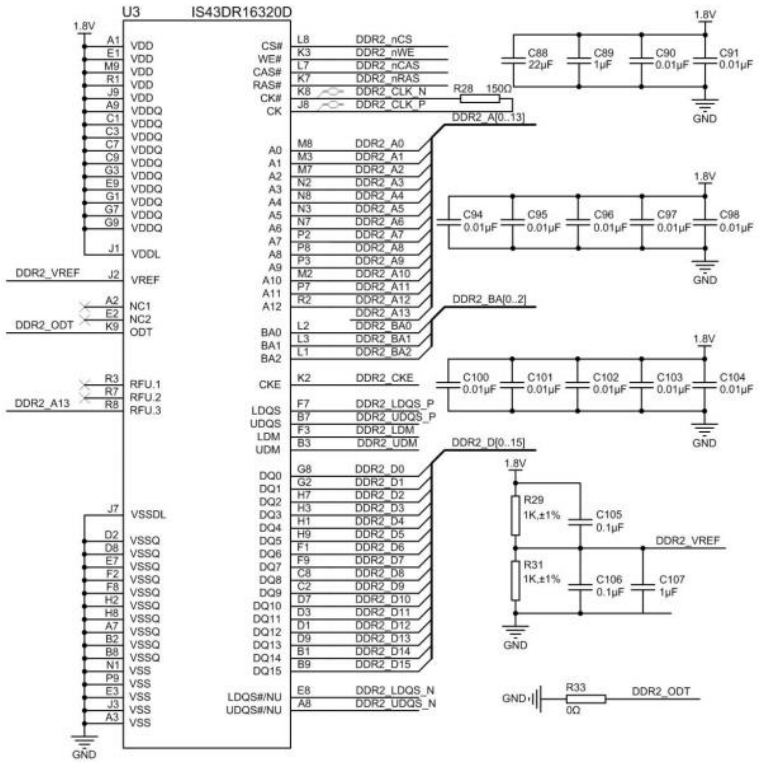

图 3.1 DDR2 典型应用电路

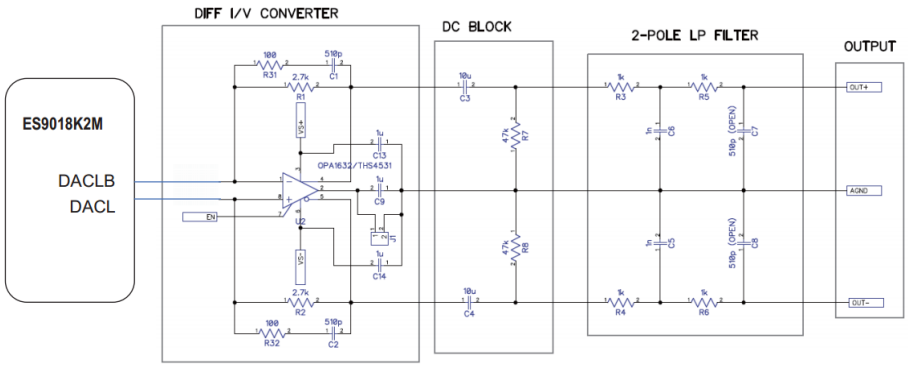

图 3.2 ES9018K2M 应用电路

4 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]。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与播放，通过声卡驱动进行音量增减的控制。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。

4.1 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。AWorks 架构如图 4.1 所示。

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。

4.2 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]。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]。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。

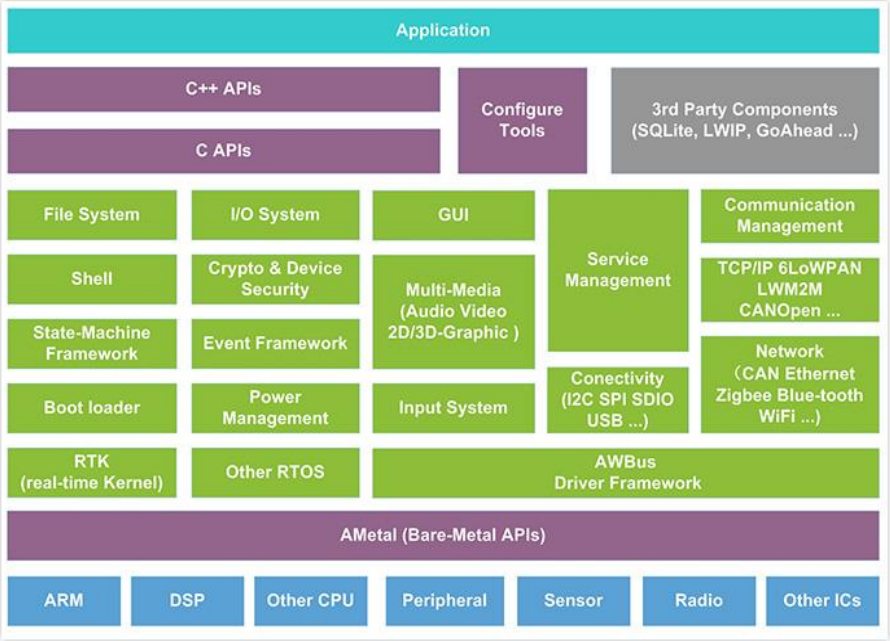

图 4.1 AWorks 架构

4.3 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分为 GUI 任务、触摸屏任务和音乐播放任务。

4.3.1 GUI 任务

本任务流程如图 4.2 所，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。

4.3.2 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。流程如图 4.3 所示。

4.3.3 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 4.4 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。

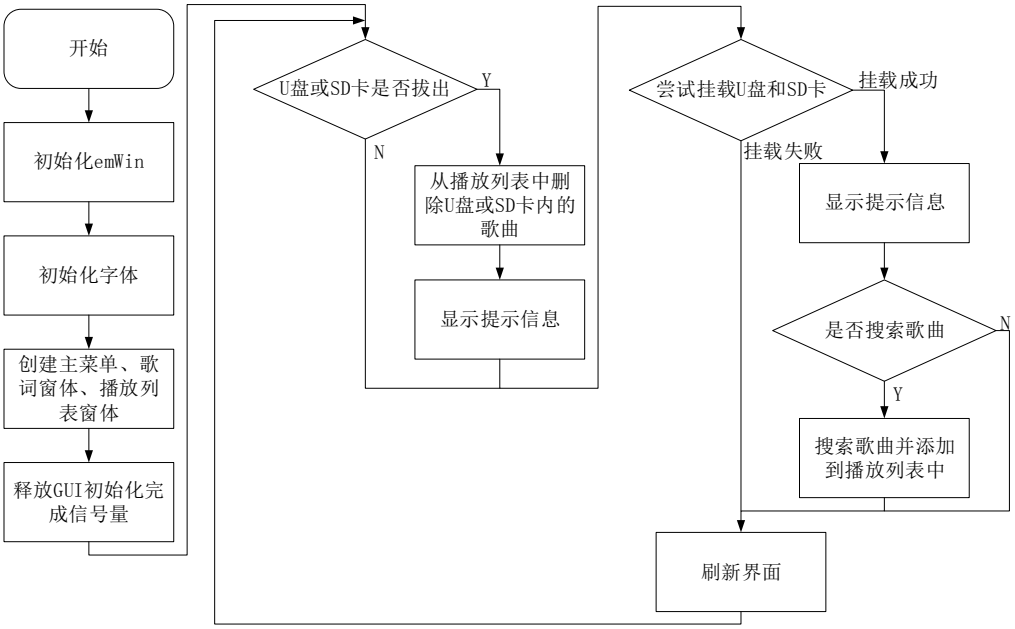

图 4.2 GUI 任务流程图

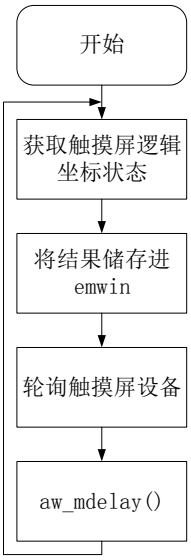

图 4.3 触摸屏任务流程图

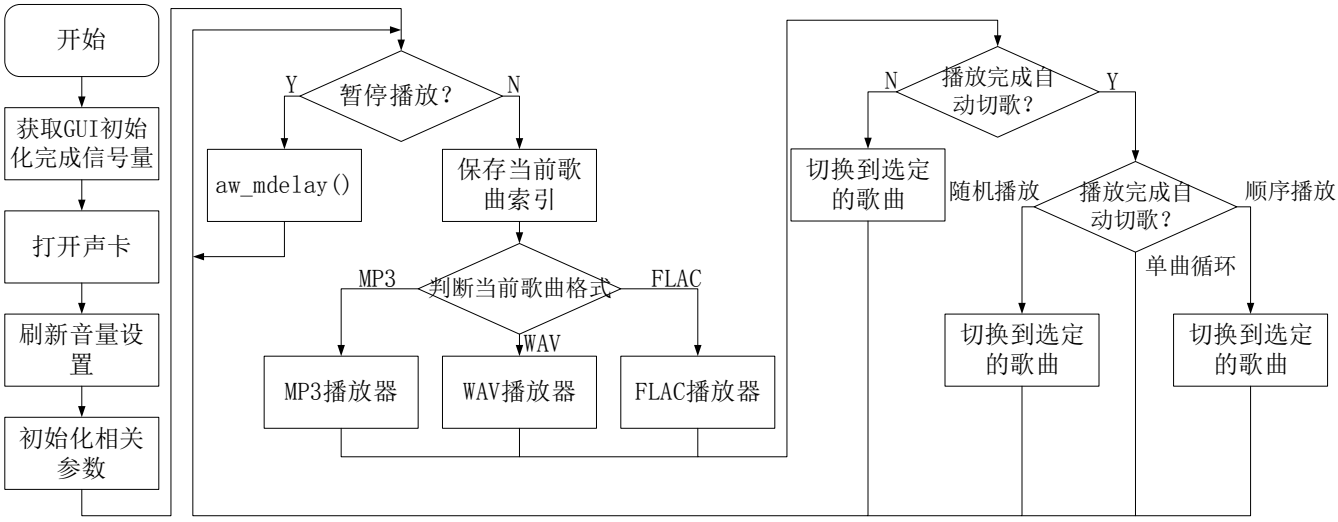

图 4.4 音乐播放任务流程图

4.4 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。

4.4.1 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。如表 4.1 所示，每个块由数据、数据长度和特征字符串构成。

表 4.1 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。RIFF/LIST 块结构如表 4.2 所示。

表 4.2 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]。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 4.3 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 4.3 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 4.4 和表 4.5 所示。

表 4.4 16 位单声道

采样一		采样二		……
低字节	高字节	低字节	高字节	……

表 4.5 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。

4.4.2 MP3 格式解码

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]。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]。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]。

MP3 文件由一系列的帧组成，所有帧都有自己的帧头。文件如表 4.6 可知，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 4.6 MP3 文件结构

ID3V2	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 每个帧的长度可能固定，也可能不固定，由位率决定 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	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 4.7 所示。

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：

$$帧长度(字节) = 每帧采样数/(8 * 比特率/采样频率) + 填充 * 4 \quad (4.1)$$

Lyer II 和 Lyaer III 使用公式：

$$帧长度(字节) = 每帧采样数/(8 * 比特率/采样频率) + 填充 \quad (4.2)$$

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：

$$每帧持续时间(毫秒) = 每帧采样数/(采样频率 * 1000) \quad (4.3)$$

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 320Kbps)。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。

表 4.7 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-1：00-44.1kHz 01-48kHz 10-32kHz11-未定义 对于 MPEG-2：00-22.05kHz 01-24kHz 10-16kHz11-未定义 对于 MPEG-2.5：00-11.025kHz 01-12kHz 10-8kHz11-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

4.4.3 FLAC 格式解码

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。因为 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 4.8 和表 4.9 所示。

本设计中，FLAC 格式最高能支持到采样率 192KHz，量化位数 24 位。

4.4.4 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 4.5 所示。

表 4.8 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 4.9 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

4.5 LRC 歌词解析

4.5.1 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。标示标签主要有[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]。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。

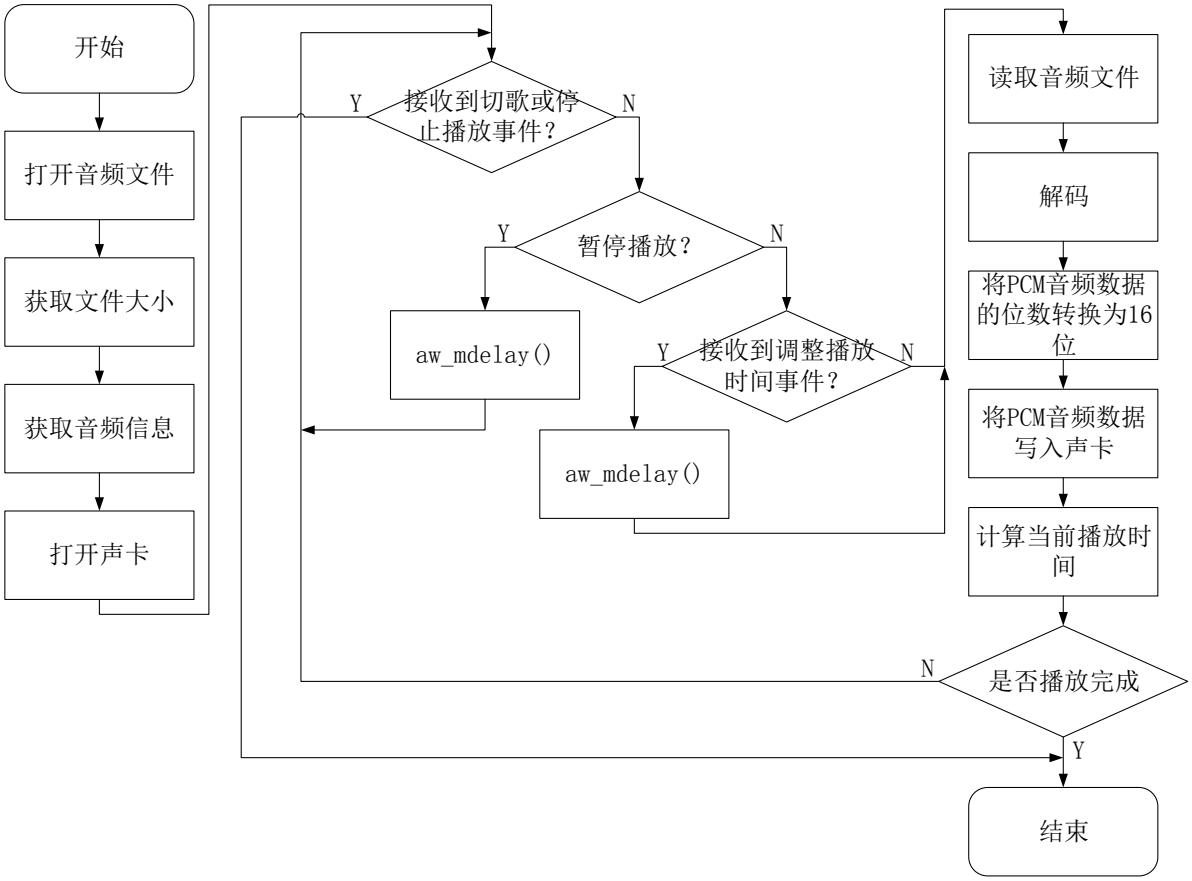

图 4.5 音频文件播放流程图

4.5.2 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 4.6 所示。

4.6 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。

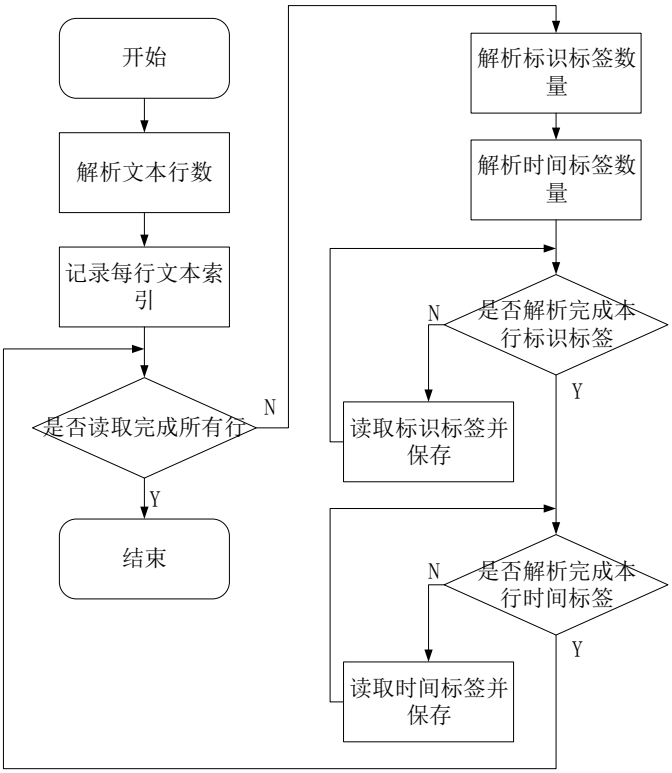

图 4.6 歌词解析流程图

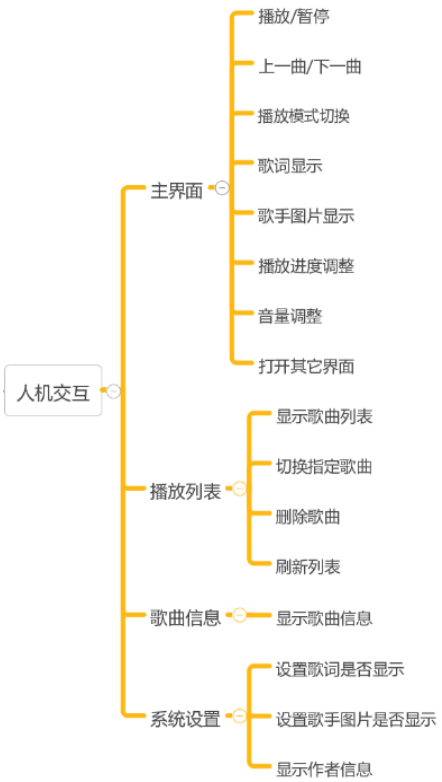

图 4.7 用户界面设计思维导图

如图 4.7 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。

4.6.1 主界面设计

图 4.8 主界面

图 4.8 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 4.9 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 4.9 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 4.11 所示。

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 4.10 所示中的三种模式循环切换。

图 4.10 三种播放模式

图 4.11 歌词调整界面

4.6.2 播放列表界面设计

如图 4.12 所示，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。

图 4.12 播放列表界面

4.6.3 歌曲信息界面

歌曲信息界面如图 4.13 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 4.13 歌曲信息界面

4.6.4 系统设置界面

系统设置界面如图 4.14 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。


系统设置
基于i.MX283处理器的音乐播放器设计
重庆理工大学
机械工程学院 测控技术与仪器
张进科 11307030328
显示歌词 ON
显示歌手图片 ON
取消
确定
00:00
03:26

图 4.14 系统设置界面

5 测试与结果

本设计完成后实物如图 5.1 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。

5.1 CPU 使用情况

图 5.1 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 5.1 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 5.1 音频解码 CPU 占用率

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

5.2 设计完成情况

经过漫长的设计、制作过程。本设计实现了软件设计中音频解码功能、歌词解析功能。采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。

6 体　会

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

[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.

[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.

[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.

[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.

[5]K Govardhanaraj, D Nagaraj. Intelligent music player with ARM7 [J]. Global Conference on Communication Technologies, 2015:323-326.

[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.

[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.

[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.

[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.

[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.

[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.

[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.

[13]杨雪梅,张慧.基于 STM32 的音乐播放器[J].信息工程.2016(3):136-137.

[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.

[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.

[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.

[17]Davis B, Mudge T, Jacob B. DDR2 and Low Latency Variants[J]. Solving the Memory Wall Problem

Workshop, 2000.

[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.

[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.

[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.

[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.

[22]Liu Q, Sung A H, Qiao M. Detecting information-hiding in WAV audios[C]. International Conference on Pattern Recognition. IEEE, 2009:1-4.

[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.

[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.

[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.

附 录

GUI 任务源代码：

```
/**
 * \brief GUI 任务入口
 *
 * \param[in] p_arg 参数
 *
 * \return 无
 */
void player_gui_task_entry (void *p_arg)
{
    uint8_t sd_unmount_test = 0;
    unsigned char  temp = 0;
    dialog_info_t  info;
    struct aw_dir *p_dirp;
    int            recv = 0;

    WM_SetCreateFlags(WM_CF_MEMDEV); /* 窗口使用存储设备 */
    GUI_Init();                      /* 初始化 emwin */

    aw_touch_gui_calibrate(AW_TS_LIB_FIVE_POINT, gui_calibrate_cb);

    /* 初始化 xbf 字体 */
#if 1
    xbf_init((void *)g_xbf_adobe_hei_std);
#else
    xbf_init((void *)g_ttf_res);
#endif
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();

    GUI_UC_SetEncodeUTF8();
    GUI_SetTextMode(GUI_TEXTMODE_TRANS);  /* 透明文本 */

    WM_MOTION_Enable(1);
    BUTTON_SetReactOnLevel();

    WIDGET_SetDefaultEffect(&WIDGET_Effect_None); /* 设置控件风格 */

    /* 创建主菜单 */
    g_h_winmain_0 = GUI_CreateDialogBox(g_dialog_create, /* 控件资源表 */
                        GUI_COUNTOF(g_dialog_create),    /* 控件数量 */
```

```
                        main_dlg_callback, /* 对话框过程函数的回调函数 */
                        WM_HBKWIN,         /* 父窗口的句柄*/
                        0,                 /* 相对于父窗口的 X 轴位置 */
                        0);                /* 相对于父窗口的 Y 轴位置 */

    /* 创建主菜单定时器 */
    g_h_timer_0 = WM_CreateTimer(g_h_winmain_0, 0, 500, 0);

    /* 歌词窗体容器 */
    g_h_winhead = WM_CreateWindowAsChild(
                        0,              /* 父窗口在窗口坐标中的左上 X 位置 */
                        0,              /* 父窗口在窗口坐标中的左上 Y 位置 */
                        480,            /* 窗口的 X 尺寸 */
                        205,            /* 窗口的 Y 尺寸 */
                        g_h_winmain_0,  /* 父窗口的句柄 */
                        WM_CF_SHOW | WM_CF_HASTRANS , /* 窗口创建标识 */
                        dummy_callback, /* 回调例程的指针 */
                        0);             /* 要分配的额外字节数 */

    /* 创建歌词窗体 */
    g_h_winlyrics = WM_CreateWindowAsChild(
                          0,               /* 父窗口在窗口坐标中的左上 X 位置 */
                          0,               /* 父窗口在窗口坐标中的左上 Y 位置 */
                          480,             /* 窗口的 X 尺寸 */
                          0,               /* 窗口的 Y 尺寸 */
                          g_h_winhead,     /* 父窗口的句柄 */
                          WM_CF_SHOW | WM_CF_HASTRANS | WM_CF_MOTION_Y, /* 标识 */
                          lyrics_callback, /* 回调例程的指针 */
                          0);              /* 要分配的额外字节数 */

    /* 创建歌词定时器 */
    g_h_timer_1 = WM_CreateTimer(g_h_winlyrics, 1, 0, 0);
    g_h_timer_2 = WM_CreateTimer(g_h_winlyrics, 2, 0, 0);

    /* 创建播放列表窗体 */
    g_h_winlist_0 = music_list_create(g_h_winmain_0); /* 创建播放列表窗口 */
    WM_HideWindow(g_h_winlist_0);                     /* 隐藏播放列表窗口 */

    g_h_setting_0 = setting_create(g_h_winmain_0);
    WM_HideWindow(g_h_setting_0);                     /* 隐藏系统设置窗口 */

    IMAGE_SetJPEG(g_h_image_0,
                  g_res_background_2,
                  sizeof(g_res_background_2));

    /* 将之前创建的窗口绘制出来 */
```

```
87.     GUI_Exec();
88.
89.     /* 释放 GUI 初始化完成信号量 */
90.     AW_SEMB_GIVE(g_play_ctrl.gui_ok);
91.
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;
98.
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;
104.
105.                 play_list_init(g_play_list,
106.                                sizeof(g_play_list) / sizeof(g_play_list[0]),
107.                                1,
108.                                SEARCH_RANGE_USB);
109.
110.                 winlist_recreate();
111.
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.         }
129.
130.         if (!g_is_mount_udisk) {
131.
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"));
137.
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);
156.
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;
168.
169.                 play_list_init(g_play_list,
170.                                sizeof(g_play_list) / sizeof(g_play_list[0]),
171.                                0,
172.                                SEARCH_RANGE_SD);
173.
174.                 winlist_recreate();
175.
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.         }
193.
194.         if ((!g_is_mount_sdcard) && (0 == sd_unmount_test)) {
195.
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"));
201.
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);
252.
253.                        winlist_recreate();
254.                        play_list_write(g_play_list,
255.                                       &g_list_state,
256.                                        PLAY_LIST_MAX_SIZE);
257.                    }
258.
259.                }
260.
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;
268.
269.                } else {
270.                    volume_set(50);
```

```
                    g_play_ctrl.play_idx = 0;
                    g_config_info.lrc_show = 1;
                    g_config_info.singer_show = 1;
                    g_config_info.is_effect = 1;
                }
            }
        }

        GUI_Delay(500);
    }
}
```

触摸屏任务源代码：

```
/**
 * \brief 触摸屏处理任务入口
 *
 * \param[in] p_arg 参数
 *
 * \return 无
 */
void touch_screen_task_entry (void *p_arg)
{
    GUI_PID_STATE         th;
    struct aw_touch_state ts_state;

    th.Layer = 0;

    AW_FOREVER {
        aw_touch_get_log_state(&ts_state);  /* 获取抽象触摸设备当前逻辑坐标状态 */

        th.x = ts_state.x;
        th.y = ts_state.y;
        th.Pressed = ts_state.pressed;

        GUI_TOUCH_StoreStateEx(&th);        /* 把结果存储进 emwin */

        aw_touch_exec();                    /* 定期对触摸设备进行轮询 */

        aw_mdelay(10);
    }
}
```

音乐播放器任务源代码：

```
/**
```

```
 * \brief 音乐播放任务入口
 */
void play_task_entry (void *p_arg)
{
    aw_time_t time;

    /* 等待 GUI 初始化完成 */
    AW_SEMB_TAKE(g_play_ctrl.gui_ok, AW_SEM_WAIT_FOREVER);

    /* 构建音频设备 */
    aw_imx28_snd_mkdev(&g_snd_dev,                      /* 音频设备 */
                       0,                               /* SAIF0 */
                       SOUND_DEVICE,                    /* 声卡 */
                       256,                             /* 一次中断传输数据大小 */
                       128,                             /* 内部环形缓冲区大小 */
                       16,                              /* 位数 */
                       2,                               /* 通道数 */
                       44100,                           /* 采样率 */
                       AW_IMX28_SND_STREAM_PLAYBACK);   /* 播放音频 */

    /* 打开声卡 */
    if (aw_imx28_snd_open(&g_snd_dev) != AW_OK) {
        AW_ERRF(("aw_imx28_snd_open failed\n"));
    }

    AW_FOREVER {

        /* 播放索引合法性检查 */
        if ((0 == g_list_state.play_list_count) ||
            (g_play_ctrl.play_idx >= g_list_state.play_list_count)) {
            g_play_ctrl.play_idx = 0;
            g_play_ctrl.paly_state = PALY_STATE_PAUSE;
        }

        if (g_play_ctrl.event & PLAY_EVENT_STOP) {
            g_play_ctrl.paly_state = PALY_STATE_STOP;
            g_play_ctrl.event &= ~PLAY_EVENT_STOP;
        }

        if (g_play_ctrl.paly_state != PALY_STATE_PLAY) {
            aw_mdelay(100);
        } else {
            refresh_volume();

            /* 保存当前播放索引以及音量配置 */
            g_config_info.volume = volume_get();
```

```
48.             g_config_info.play_idx = g_play_ctrl.play_idx;
49.             config_write(&g_config_info);
50.
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];
55.
56.             AW_INFOF(("play music %s\n", g_play_ctrl.info.name));
57.
58.             /* 根据歌曲的格式调用不同的解码器 */
59.             switch (g_play_ctrl.info.type) {
60.
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;
67.
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.                 }
73.
74.                 break;
75.
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;
82.
83.             default:
84.                 ; /* VOID */
85.             }
86.
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) {
92.
93.                 case PALY_MODE_RANDOM: /* 随机播放 */
```

```
94.
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;
101.
102.                 case PALY_MODE_ORDER:  /* 顺序播放 */
103.
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.                         }
111.
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;
121.
122.                 case PALY_MODE_SINGLE: /* 单曲循环 */
123.
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.                         }
131.
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.                         }
139.
```

```
140.                     /* 单曲循环 */
141.                     } else {
142.                         ; /* VOID */
143.                     }
144.                     break;
145.
146.                 default:
147.                     ; /* VOID */
148.                 }
149.             }
150.         }
151.     }
152. }
```