摘要

随着数字技术的出现与应用，如今的人们生活在一个几乎已经完全数字化的世界中，而数字音频技术则称得上是应用最为广泛的数字技术之一。本文以一款基于STM32的数字音频播放器为例，研究了数字音频技术的基本原理。遵循嵌入式系统开发的基本过程，对播放器的功能进行需求分析，设计并实现了一款完整的WAV数字音频播放器。通过对FATfs文件系统的移植，实现对带有FAT文件系统的大容量SD卡中音频数据的解码播放，同时采用了双缓存的工作模式来解决从SD卡中读取并播放音频数据的连贯性问题。最后对已完成的数字音频播放器进行系统测试，结果表明已基本实现预期目标。

**关键词： 数字音频 STM32 WAV FATfs 双缓存**

ABSTRACT

With the appearance and application of digital technology, people now live in an almost complete digital world, and the digital audio technology is one of the most widely used digital technology. The basic principle of digital audio technology has been studied in this paper, through an example digital audio player based on STM32. Follow the basic process of the embedded system development, analyze the function of the player, design and implement a complete WAV digital audio player. Through the transplantation of FATfs file system, to achieve audio data in high-capacity SD card with the FAT file system to play. The detail of the WAV audio file format and the realization of the decoding algorithm has been analyze in the middle of the article, while using the work mode of double-buffer to solve this problem which read and play the audio data from the SD card continuously. Finally, the system has been tested and the results show that the target has been realized.

**Keywords: Digital audio STM32 WAV FATfs Double-buffer**

目录

第一章 绪论 1

1.1引言 1

1.2本文内容及章节安排 2

第二章 数字音频 3

2.1音频播放的基本原理 3

2.1.1数字音频的原理 3

2.1.2音频系统 4

2.2 WAV音频文件 6

2.2.1 WAV简介 6

2.2.2 WAV文件结构 6

第三章 硬件选型 11

3.1总体框架 11

3.2 STM32微控制器 12

3.3 SD卡模块 13

3.3.1 SD卡介绍 13

3.3.2 SD卡内部结构及引脚功能 14

3.4 外部按键模块 15

3.4.1按键控制介绍 15

3.4.2电路原理图 15

第四章 软件设计 17

4.1系统概述 17

4.1.1方案选择 17

4.1.2框架设计 18

4.2相关技术 19

4.2.1 SDIO驱动 19

4.2.2 FATfs文件系统 21

4.3 WAV音频解码算法 23

4.3.1主要函数介绍 23

4.3.2解码流程分析 25

4.4双缓存 26

4.4.1概述 26

4.4.2设计方案 27

4.5播放控制 29

4.5.1设计需求 29

4.5.2方案介绍 29

第五章 系统测试 33

5.1测试环境 33

5.1.1硬件环境 33

5.1.2软件环境 33

5.2测试过程及结果分析 33

5.2.1测试过程 33

5.2.2结果分析 36

第六章 总结与展望 37

6.1全文总结 37

6.2改进及下一步工作 37

6.2.1系统存在的不足 37

6.2.2下一步的工作 38

致谢 39

参考文献 41

第一章 绪论

1.1引言

音乐，作为人类最宝贵的精神食粮之一，几乎贯穿着人类的整个发展史。从最早中世纪时期的教堂圣咏开始，到以巴赫为代表的巴洛克时期，再到经过启蒙运动洗礼后诞生的古典与浪漫主义时期，音乐一直都是作为艺术的最高表现而被人们所敬仰。

正因为音乐的美好，人们对于音乐存储方式的探索从来没有停止过。早期的人们通过口口相传的方式记录音乐，而五线谱的发明，则让人们开始用纸笔等工具来长久的记录音乐；随着电学、电子学的发展，人们开始尝试把声的振动转换成电信号，最终电声技术获得了迅速发展。

传统的声音记录方式就是将模拟信号直接记录下来，存储到黑胶唱片或磁带中。随着计算机技术的发展，特别是海量存储设备和高性能处理器技术的突破，使得对音频媒体的数字化处理成为可能。1939年法国工程师Alec Reeves发明了将连续的模拟信号变换成时间和幅度都离散的二进制码代表的脉冲编码调制信号，但一直到1962年，美国Bell实验室才为AT&T制成了国际上第一套商用PCM电话系统（T1系统）。1982年秋，随着Sony在日本发布了世界上第一部商用CD光盘播放器CDP-101，数字音频革命的序幕被缓缓揭开。和磁带不同，CD是以数字化的形式储存音乐，因为更高的存储密度、更优质稳定的音质，CD迅速取代磁带成为追求品质人群的首选[4]。

生产力的解放带动着技术的革新，而技术的革新则推动着世界的进步。时至今日，随着电子信息技术的爆炸性增长，数字音频技术发生也着翻天覆地的变化。MP3等数字音频格式的出现加快了数字音乐的流行与传播，而苹果公司所推出的iPod与iTunes则改变了整个音乐产业。随着互联网的高速发展，曾经高昂的存储价格和流量费用现已不复存在，取而代之的是更大的存储空间，更快的网络，更低廉的价格，以及人们对HIFI音乐的追求。

1.2本文内容及章节安排

本文以一款基于STM32的数字音频播放器为例，研究了数字音频技术的的基本原理，遵循嵌入式系统开发的基本过程，对播放器的功能进行需求分析，设计并实现了一款完整的WAV数字音频播放器。通过对FATfs文件系统的移植，实现对带有FAT文件系统的大容量SD卡中音频数据的解码播放，同时采用了双缓存的工作模式来解决从SD卡中读取并播放音频数据的连贯性问题。最后对已完成的数字音频播放器进行系统测试。

第二章主要介绍了数字音频的基本知识，阐述了声电转换及声音数字化的基本原理及过程，详细分析了WAV音频文件的文件格式以及在程序中的存储方式。

第三章主要讲述了本项目的硬件模块，介绍了在项目中所使用的STM32F103微控制器，而作为音频文件的存储媒介，SD卡模块的电路设计及指令格式也在这一章有所涉及。在本章的最后还介绍了用于实现音乐播放控制的外部按键模块及电路结构。

第四章是本文的重点，主要叙述了相关的软件技术以及系统设计。在本章中，首先介绍了项目中所用到的SDIO驱动程序和FATfs文件系统的原理，并分析了其移植过程；紧接着根据第二章所分析的WAV音频文件格式，设计并实现了WAV音频文件的解码算法，使之可以清晰流畅的播放存储于装有FAT文件系统的SD卡中的WAV音频文件数据；最后利用已完成的驱动程序及外部按键，设计并实现了播放器的基本功能。

为了更好的介绍本项目的设计与开发，在第四章中分别以流程图的形式展示了各个模块的工作流程，而从所展示的流程图中可以看出，所有的模块在实际运行中都是相互嵌套，相互协作，共同实现各个功能的。

第五章完成了对已实现功能的测试工作，测试其是否实现最初的设计需求，并在第六章指出了本文现有的缺陷以及之后的改进。

第二章 数字音频

2.1音频播放的基本原理

#### 2.1.1数字音频的原理

声音在本质上是一种机械振动，它通过空气传播到人耳，刺激神经后使大脑产生一种感觉。在一些专业场合，声音通常被称为声波或音频。声音在物理学上称之为声波，是通过一定介质（如空气、水等）传播的一种连续振动的波，也称为声波。通常把频率范围为20Hz～20kHz的信号称为音频信号，低于20Hz的信号为次声信号，高于20kHz的信号称为超声信号。

数字音频是一种利用数字化手段对声音进行录制、存放、编辑、压缩或播放的技术，它是随着数字信号处理技术、计算机技术、多媒体技术的发展而形成的一种全新的声音处理手段。在现实生活中，人耳听到的声音是模拟信号，计数机中能只能存储数字信号，把模拟信号转换成数字信号的原理简单地说就是利用一个固定的频率对模拟信号进行采样，采样后的信号在波形上看就像一串连续的幅值不一的脉冲，把这些脉冲的幅值按一定的精度进行量化，这些量化后的数值被连续地输出、传输、处理或记录到存储介质中，所有这些组成了数字音频的产生过程。图2.1展示了这个过程：

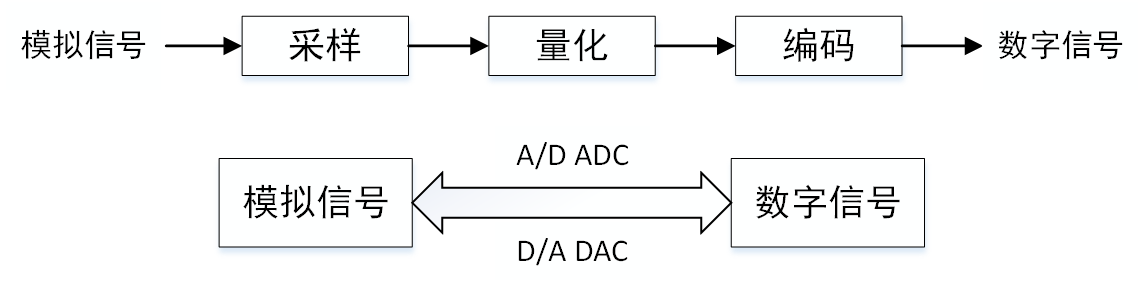


图2.1 转换过程

信号取样定理：对于上限频率为fm的限带信号f（t），若取样频率为：fs≥2fm，则用频率为fm的理想低通滤波器就可以完全恢复原信号[5]。

声音数字化三要素：

1. 采样频率：每秒钟抽取声波幅度样本的次数，通常来说，采样频率越高，声音质量越好，数据量也越大。常用采样频率：11.025kHz、22.05kHz、44.1kHz；
2. 量化位数：每个采样点用多少二进制位表示数据范围，量化位数越多，音质越好，数据量也越大。常用量化位数：8位＝256、16位＝65,536、24位＝16,777,216；
3. 声道数：使用声音通道的个数，立体声比单声道的表现力更丰富，但数据量翻倍。常用声道数：单声道、立体声[1]。

图2.2展示了一个信号量化的例子：

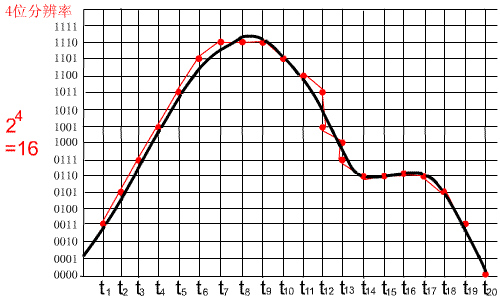


图 2.2 信号量化示意图

#### 2.1.2音频系统

针对不同的数字音频子系统，出现了几种微处理器或DSP与音频器件间用于数字转换的接口，如图2.3所示。其中最常用的有以下三种接口：

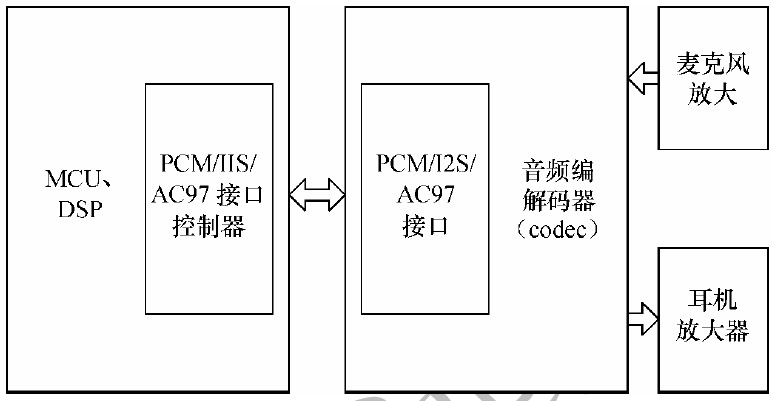


图 2.3 音频系统示意图

1. PCM接口

PCM（Pulse Code Modulation，脉冲编码调制）接口是最简单的音频接口，该接口由时钟脉冲（BCLK）、帧同步信号（FS）及接收数据（DR）和发送数据（DX）组成。在FS信号的上升沿，数据传输从MSB（Most Significant Bit）字开始，FS频率等于采样率。FS信号之后开始数据字的传输，单个的数据位按顺序进行传输，1个时钟周期传输1个数据字，请参考图2.4。

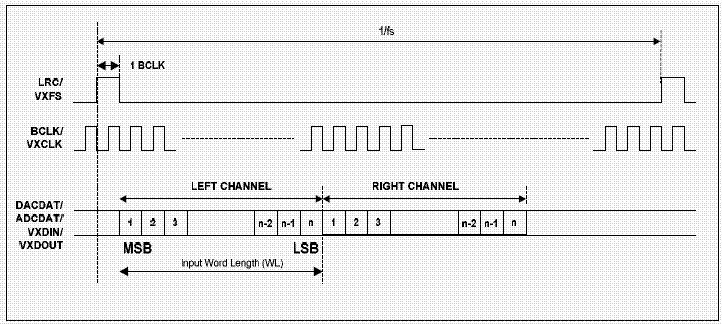


图 2.4 PCM传输示意图

1. I2S接口

IIS接口（Inter-IC Sound）在20世纪80年代首先被飞利浦用于消费音频，并在一个称为LRCLK（Left/Right CLOCK）的信号机制中经过多路转换，将两路音频信号变成单一的数据队列。当LRCLK为高时，左声道数据被传输；LRCLK为低时，右声道数据被传输。与PCM相比，IIS更适合于立体声系统（因为可以传送多个声道的数据，所以显然更加适合Stereo Single，其实所谓的立体声，也就是多个声道可以发出不一样的声音），请参考图2.5。

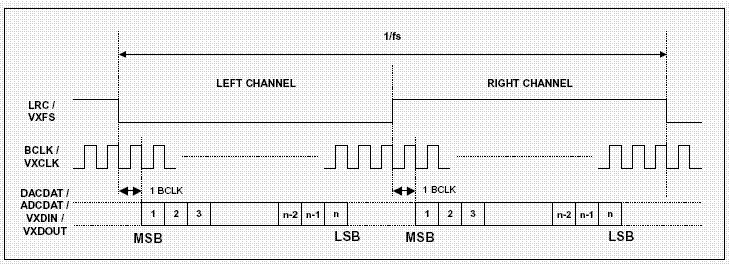


图 2.5 I2S传输示意图

1. AC97接口

AC'97（Audio Codec 1997）是以Intel为首的五个PC厂商Intel、Creative Labs、NS、Analog Device与Yamaha共同提出的规格标准。

与PCM和IIS不同，AC'97不只是一种数据格式，用于音频编码的内部架构规格，它还具有控制功能。 AC97 标准把音频设备中的数字部分和模拟部分分开, 并规定数字信号处理由CPU 来负责,或者采用专门的DSP 芯片; 而模拟部分, 即A/D、D/A 转换与Mix 混音操作, 则由CODEC 芯片(Coder2Decoder: 编码/解码器) 完成[1]。

PCM、IIS和AC97各有其优点和应用范围，例如在CD、MD、MP3随身听多采用IIS接口，移动电话会采用PCM接口，具有音频功能的PDA则多使用和PC一样的AC'97编码格式。

在本项目中，由于时间的原因，最终选择的是采用PCM接口来直接输出音频数据。

2.2 WAV音频文件

#### 2.2.1 WAV简介

WAV是一种微软公司开发的，用于存放声音文件的重要的文件格式，它符合RIFF(Resource Interchange File Format)文件规范。

由于WAV文件没有采用压缩技术，所以它的文件很庞大，一般都在几MB以上。但也正是因为没有采用压缩技术，声音的采样数据很容易被读出来，便于用作其他的处理。该格式支持多种量化位数，取样频率和声道，标准格式化的WAV文件和CD格式一样，也是44.1K的取样频率，16位量化位数，因此在声音文件质量和CD相差无几。

由于WAV格式的音频文件有着优秀的音质与相对简单的格式，因此在本项目中采用手动实现WAV解码算法来软解码播放。

#### 2.2.2 WAV文件结构

WAV格式符合 RIFF(Resource Interchange File Format)规范，每一个WAV文件都有一个文件头，这个头文件纪录着这个文件中音频流的编码参数。

数据块的记录方式是little-endian字节顺序，标志符并不是字符串而是单独的符号，数据本身的格式为PCM或压缩型，属于无损音乐格式的一种。

表2.1为WAV音频文件的文件头格式。

表2.1 WAV文件头文件

|  |  |  |  |
| --- | --- | --- | --- |
| 偏移地址 | 字节大小 | 数据块类型 | 内容 |
| 00H～03H | 4 | 4字符 | 资源交换文件标志（RIFF） |
| 04H～07H | 4 | 长整数 | 从下个地址开始到文件尾的总字节数 |
| 08H～0BH | 4 | 4字符 | WAV文件标志（WAVE） |
| 0CH～0FH | 4 | 4字符 | 波形格式标志（fmt ），最后一位为空格。 |
| 10H～13H | 4 | 整数 | 过滤字节（一般为00000010H） |
| 14H～15H | 2 | 整数 | 格式种类（为1时，表示数据为线性PCM编码） |
| 16H～17H | 2 | 整数 | 通道数，单声道为1，双声道为2 |
| 18H～1BH | 4 | 长整数 | 采样频率 |
| 1CH～1FH | 4 | 长整数 | 波形数据传输速率（每秒平均字节数） |
| 20H～21H | 2 | 整数 | DATA数据块长度，字节。 |
| 22H～23H | 2 | 整数 | PCM位宽 |
| 24H～27H | 4 | 4字符 | 数据标志符（data） |
| 28H～2BH | 4 | 长整型 | DATA总数据长度字节 |
| 2CH... | ... |  | DATA数据块 |

图2.6是通过Ultra Edit打开的一首WAV音乐的头文件部分，总共44个字节：

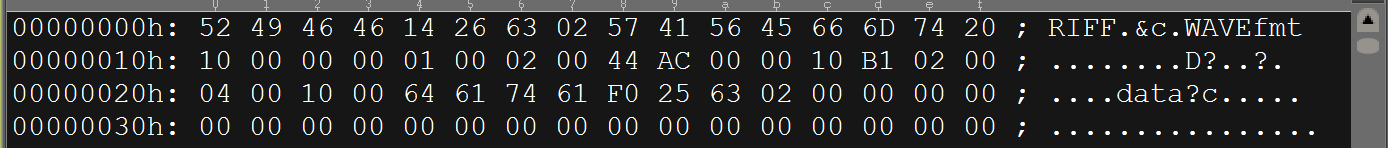


图 2.6 用Ultra Edit解析WAV

其中：

“52 49 46 46”这个是Ascii字符“RIFF”，这部分是固定格式，表明这是一个WAVE文件头。

“14 26 63 02”这个是的WAV文件的数据大小，这个大小包括除了前面4个字节的所有字节，得到40,052,244字节。在计算由多个字节表示的数字时，要按照从右到左的顺序组合之后再计算，以上数据即为02 63 26 14，转换为十进制就是40,052,244。这是由于CPU在读数据的时候是把数据按先后顺序一个字节一个字节的放入栈中，达到指定的位数以后再重新组合在一起，最后读出新组合的数据。

“57 41 56 45 66 6D 74 20”，也是Ascii字符“WAVEfmt ”，注意最后一个是一位空格。这部分是固定格式，以后是PCMWAVEFORMAT部分。

“10 00 00 00”，对应数字16，这个对应定义中的PCMWAVEFORMAT部分的大小，可以看到后面的这个段内容正好是16个字节。当为16时，最后是没有附加信息的，当为数字18时，最后多了两个字节的附加信息。

“01 00”，对应定义为编码格式（WAVE\_FORMAT\_PCM格式用的就是这个）。

“02 00”，对应数字2，表示声道数为2，是立体声WAV。

“44 AC 00 00”对应数字44,100，代表的是采样频率44,100次／秒，采样率（每秒样本数）表示每个通道的播放速度。

“10 B1 02 00”对应数字176,400，代表的是每秒的数据量，波形音频数据传送数率，其值为通道数×每秒样本数×每个样本的数据位数/8。

“04 00”对应数字是4，表示块对齐的内容。数据块的调整数（按字节算），其值为通道数×每个样本的数据位置/8。播放软件需要一次处理多个改值大小的字节数据，以便将其值用于缓冲区的调整。

“10 00”，此数值为16，采样大小为16bits，每样本数据位数，表示每个声道中各个样本的数据位数。如果有多个声道，对每个声道而言，样本大小都一样。

“64 61 74 61”，这个是Ascii字符“data”，表示头结束，开始数据区域。

“F0 25 63 02”，对应十进制40,052,208，是数据区的开头以后的数据总数[12]。

知道了WAV头文件的数据格式，就可以通过程序读取音频文件信息。在上述的基础上定义头文件结构体，如表2.2到表2.6。

表2.2 RiffHeader(RIFF部分)

|  |  |  |
| --- | --- | --- |
| 名称 | 含义 | 数据类型 |
| szRiffID | ‘R’‘I’‘F’‘F’ | uint8\_t(5) |
| dwRiffSize | 去头文件大小，总数据减8 | uint32\_t |
| szRiffFormat | ‘W’‘A’‘V’‘E’ | uint8\_t(5) |

表2.3 WaveFormat(波型格式)

|  |  |  |
| --- | --- | --- |
| 名称 | 含义 | 数据类型 |
| wFormatag | 编码格式，为1时表示数据为线性PCM编码 | uint16\_t |
| wChannls | 声道数，单声道为1，双声道为2 | uint16\_t |
| nSamplesPerSec | 采样频率，11,025/22,050 | uint32\_t |
| dwAvgBytesperSec | 每秒的数据量 | uint32\_t |
| wBlockAlign | 每个采样需要的字节数，1/2 | uint16\_t |
| wBitsPerSample | 每个采样所需要的bit数，8/16/32 | uint16\_t |

表2.4 PcmWaveFormat(PCMWAVEFORMAT部分)

|  |  |  |
| --- | --- | --- |
| 名称 | 含义 | 数据类型 |
| szFmtID | ‘f’‘m’‘t’‘ ’ | uint8\_t(5) |
| wBitsPerSample | WAVE文件的采样大小 | uint32\_t |
| wf | 波型格式 | WaveFormat |

表2.5 DataBlock(data部分)

|  |  |  |
| --- | --- | --- |
| 名称 | 含义 | 数据类型 |
| szDataID | ‘d’‘a’‘t’‘a’ | uint8\_t |
| dwDataSize | 数据段的大小 | uint32\_t |

表2.6 WavHeader(WAV头文件部分)

|  |  |  |
| --- | --- | --- |
| 名称 | 含义 | 数据类型 |
| riffheader | RIFF部分 | RiffHeader |
| pcmwaveformat | PCMWAVEFORMAT部分 | PcmWaveFormat |
| factblock | fact部分（可选） | FactBlock |
| datablock | DATA部分 | DataBlock |

图2.7为经过解码算法解出来的通过串口打印到桌面的头文件信息：



图 2.7 STM32读取WAV头文件信息

可以看到与之前分析得出的数据一致。头文件的分析到此为止，下面来分析WAV数据体。

常见的声音文件主要有两种，分别对应单声道和双声道。对于单声道量化位数为8bits的声音文件，采样数据为8位的短整数；而对于立体声声音文件，每次采样数据位一个16位的整数，高8为和低8位分别代表左右两个声道。

WAVE文件数据块包含以脉冲编码调制（PCM）格式表示样本。WAVE文件是由样本组织而成的。在单声道WAVE文件中，声道0代表左声道，声道1代表右声道。在多声道WAVE文件中，样本是交替出现的。PCM数据的存放方式如表2.7所示。

表 2.7 PCM数据存放方式

样本1

8位单声道

样本2

8位单声道

样本3

8位单声道

样本4

8位单声道

样本1

16位单声道

样本1

8位单声道

样本1

8位左声道

样本1

8位右声道声道

样本2

8位左声道声道

样本2

8位右声道声道

样本1

16位左声道

样本1

16位右声道

知道了PCM数据存放方式，以及WAV音频文件的采样频率、量化位数和声道数，就可以据此编写WAV音频解码算法了。由于解码模块与文件系统模块相关，因此在第四章另行讨论。

第三章 硬件选型

3.1总体框架

本文所设计的是一款基于STM32F103微控制器的WAV数字音频播放器，其主要功能包括从带有FAT文件系统的SD卡中播放WAV音频文件；通过摇杆及按键可以实现对音量的调节和基本的音乐播放控制（上一曲、下一曲、暂停／继续播放、停止等）；可实现播放列表循环播放功能。在硬件方面，本系统分为了六个主要模块，分别是控制模块、文件系统模块、固件下载模块、串口调试模块、播放控制模块和播放模块，系统结构如图3.1所示。

图 3.1系统总体框架图

MCU

SDIO

SD Card

Keys

Main board

Speaker/ earphone

J-Link

JTAG/SW

PC

Usart

为了便于使用及降低开发成本，在其中所使用的MCU主控制器、SDIO接口、JTAG/SW接口、外部按键及Usart串口接口均由Main board开发板所提供。MCU主控制模块负责整个系统的控制；SDIO接口模块负责从装有FAT文件系统的SD卡中读写数据；JTAG/SW通过与J-Link连接PC来实现程序的调试与烧录；Usart串口可以实现PC与开发板的通信，便于在开发与测试过程中调试程序；外部按键Keys提供了独立于PC的人机交互功能；MCU内置的DAC则负责输出音频信息，并通过扬声器输出。

下面着重介绍MCU控制模块、SDIO模块和播放控制模块。

3.2 STM32微控制器

STM32F103ZET6使用高性能的ARM Cortex-M3核心的32位微控制器，LQFP-144封装，512K片内FLASH（相当于硬盘），64K片内RAM（相当于内存），高达72M的时钟频率,丰富的增强I/O端口和联接到两条APB总线的外设，3路共16通道的12位AD输入，2路共2 通道的12位 DA 输出。4个通用定时器，2个高级定时器，2个基本定时器，使得处理速度大大提高，机器的功耗也大幅降低。32位的处理器使得控制更加的稳定、迅速，而窗口型的看门狗，则使得程序的运行更加高效。另外，通过连接J-Link仿真器可以方便实现在线逐步调试功能，大大方便了系统的开发与调试工作[8]。

芯片的管脚如图3.2下所示：



图 3.2 STM32F103ZET6管脚图

为了降低成本，在本项目中使用的是一款搭载了STM32F103ZET6芯片的多功能STM3210E-EVAL开发板，丰富的外设资源为此项目的升级与扩展提供了更多的选择。关于开发板的具体功能在这里就不再赘述，STM32F103芯片的各个管脚与开发板的连接如图3.3所示，根据此图来选择所需要的管脚。

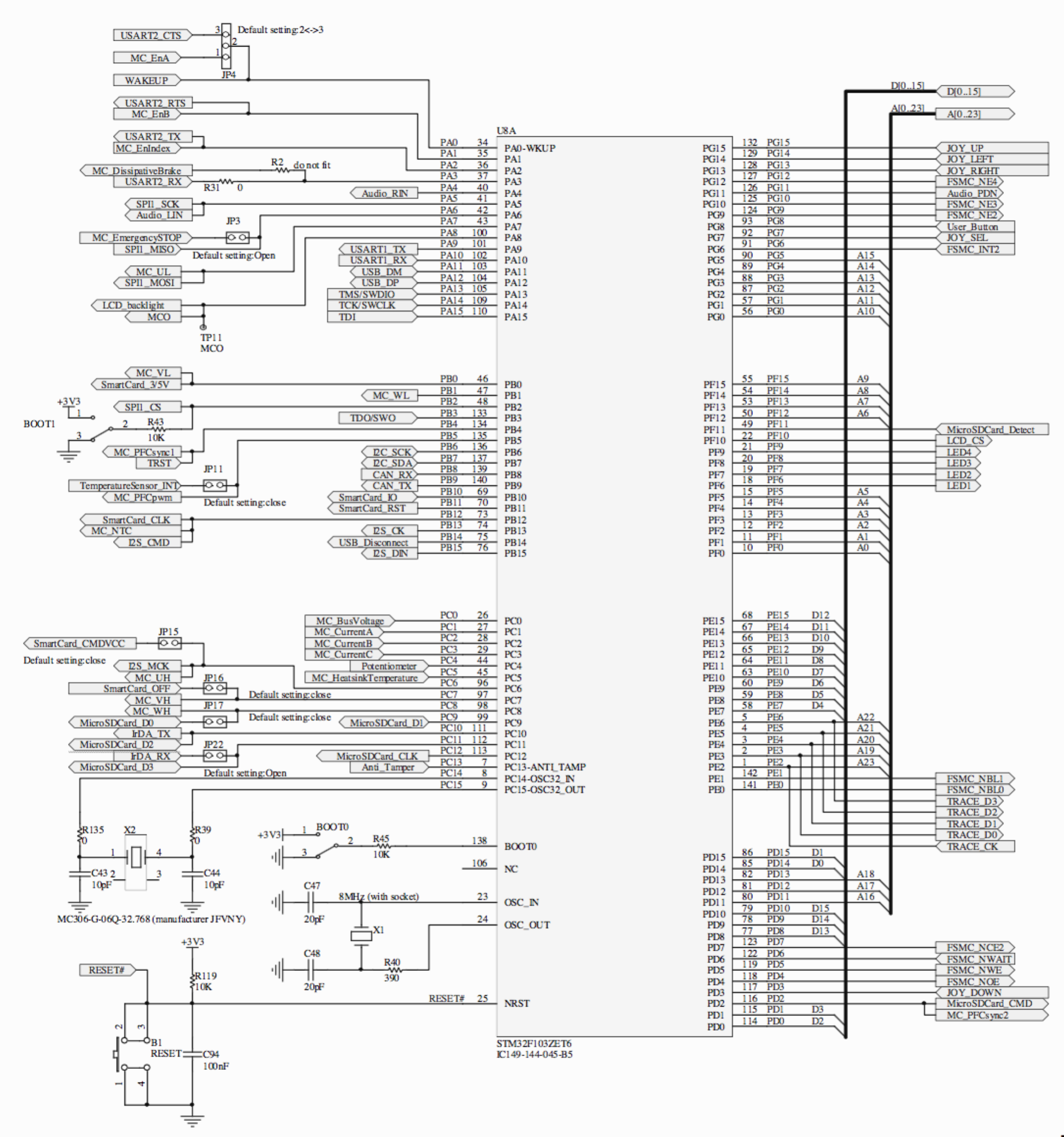


图 3.3 STM32F103ZET6管脚引出电路图

3.3 SD卡模块

#### 3.3.1 SD卡介绍

Secure Digital缩写SD，全名Secure Digital Memory Card，中文翻译为安全数位卡，是一种基于半导体快闪记忆器的新一代记忆设备，它被广泛地于便携式装置上使用。SD卡是一种基于半导体闪存工艺的存储卡，1999年由日本松下主导概念，参与者东芝和美国SanDisk公司进行实质研发而完成，目前SD卡已成为消费数码设备中应用最广泛的一种存储卡，而作为SD卡的衍生品，miniSD卡和microSD卡虽然出现的比较晚，但由于有着更小的体积，因而再电子市场上有着更为广泛的应用。在此项目中使用的是microSD作为音频文件的存储媒介。图3.4为开发板上的microSD管脚电路图。

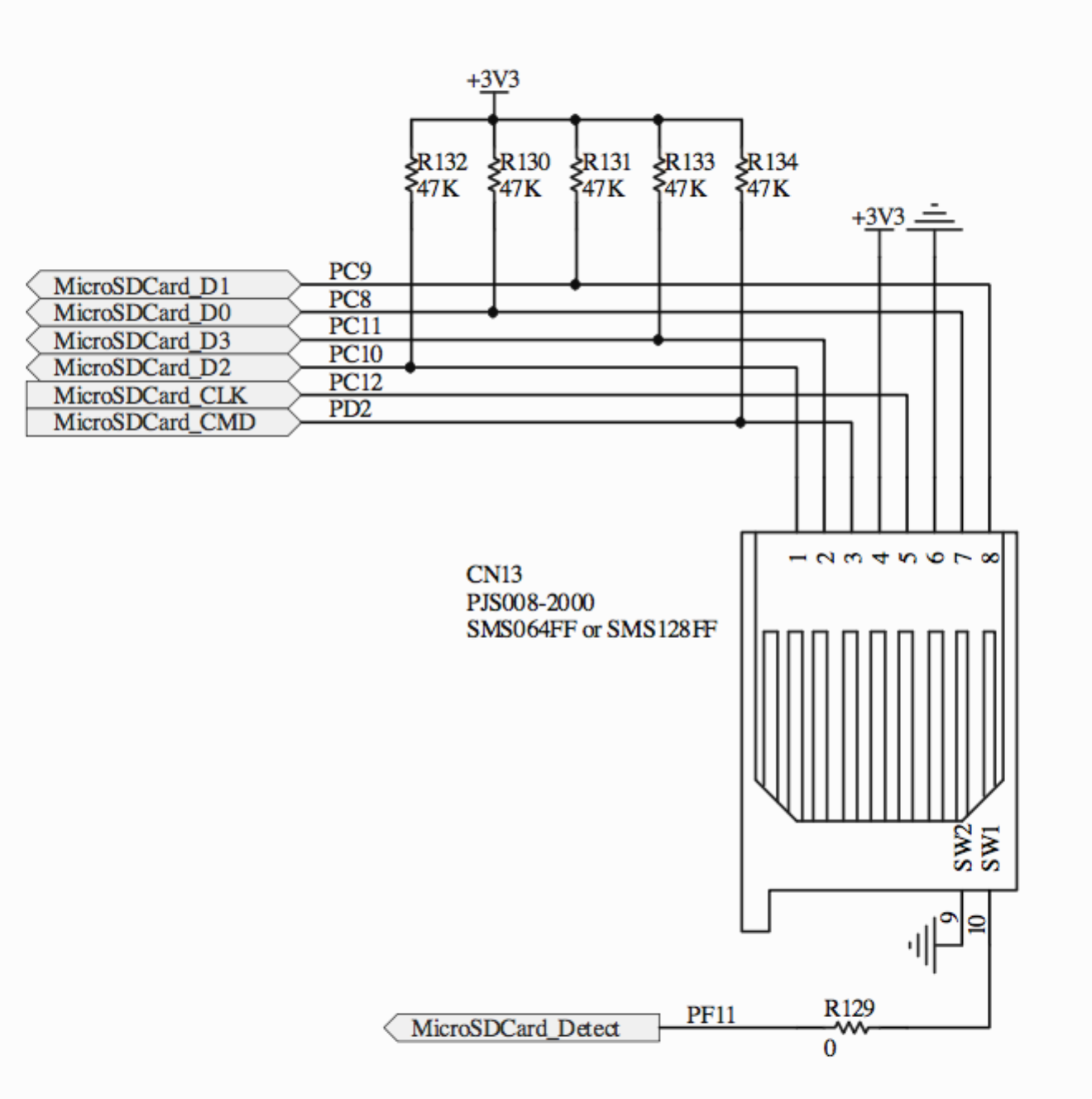


图 3.4 microSD管脚引出电路图

#### 3.3.2 SD卡内部结构及引脚功能

SD卡允许在两种模式下工作，即SD模式和SPI模式，由于本项目使用的是微处理器中的SDIO通信接口来连接SD卡，因此采用的是SD模式。SD卡内部结构及引脚如图3.5所示：

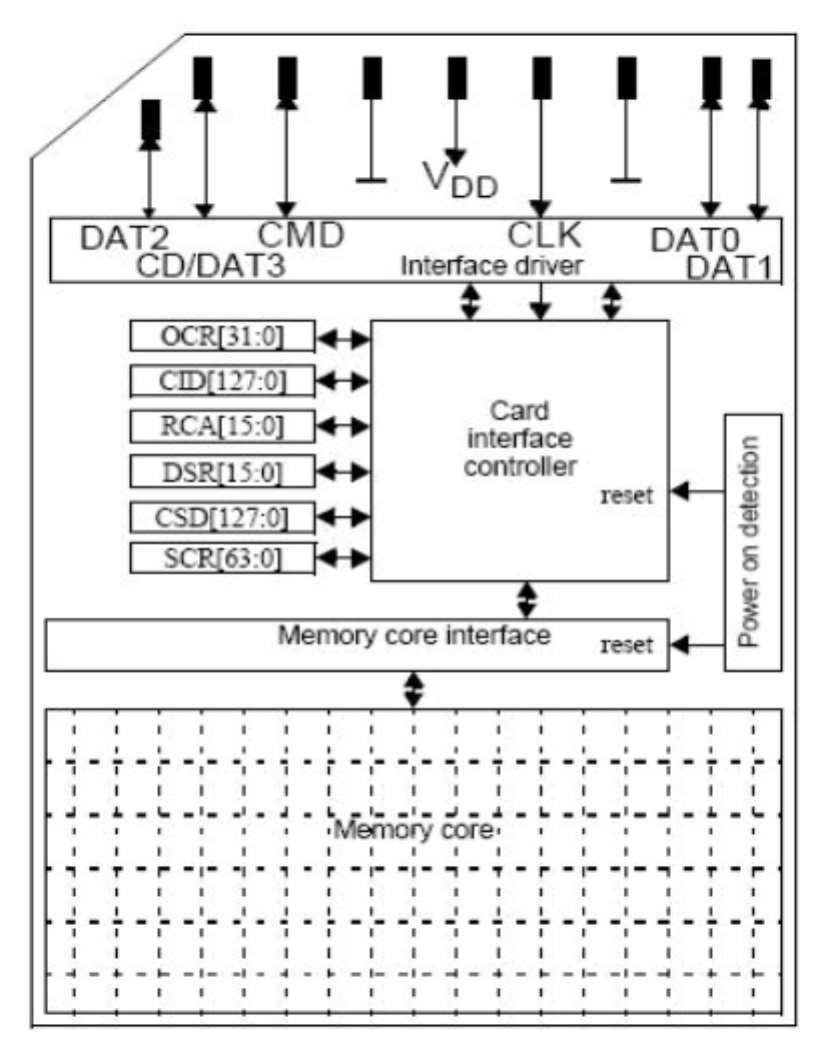


图 3.5 SD卡内部结构及引脚

SD卡的主要引脚和功能：

1. CLK

CLK是卡的时钟信号线，由主机产生时钟信号，在命令和数据线上，每个时钟周期传输一个命令或数据位，频率可在0～25MHz之间变化。

1. CMD

CMD为双向命令和回复线，SDIO的所有主机发出的命令及从机对命令的响应都是在这条信号线上传输的。

1. DAT［3:0］

DAT［3:0］表示四条数据线，主从机的数据信号在这四条线上传输。

3.4外部按键模块

#### 3.4.1按键控制介绍

本次项目使用了八个外部按键来实现对音频播放的控制，其中包括一个操纵杆及三个按键Wakeup、Tam和Key。操纵杆包含上、下、左、右、确认五个按键，两个独立按键分别为。在本项目中使用按键触发外部中断来实现对播放系统的控制及音量的调节。

#### 3.4.2电路原理图

图3.6为操纵杆的电路图，DOWN和UP与MCU的PG3和PG15相连，分别控制着音量的增减；LEFT和RIGHT与MCU的PG14和PG13相连，分别控制着上一曲、下一曲功能；Selection与MCU的PG7相连，在音乐播放时控制播放的暂停与继续播放功能。

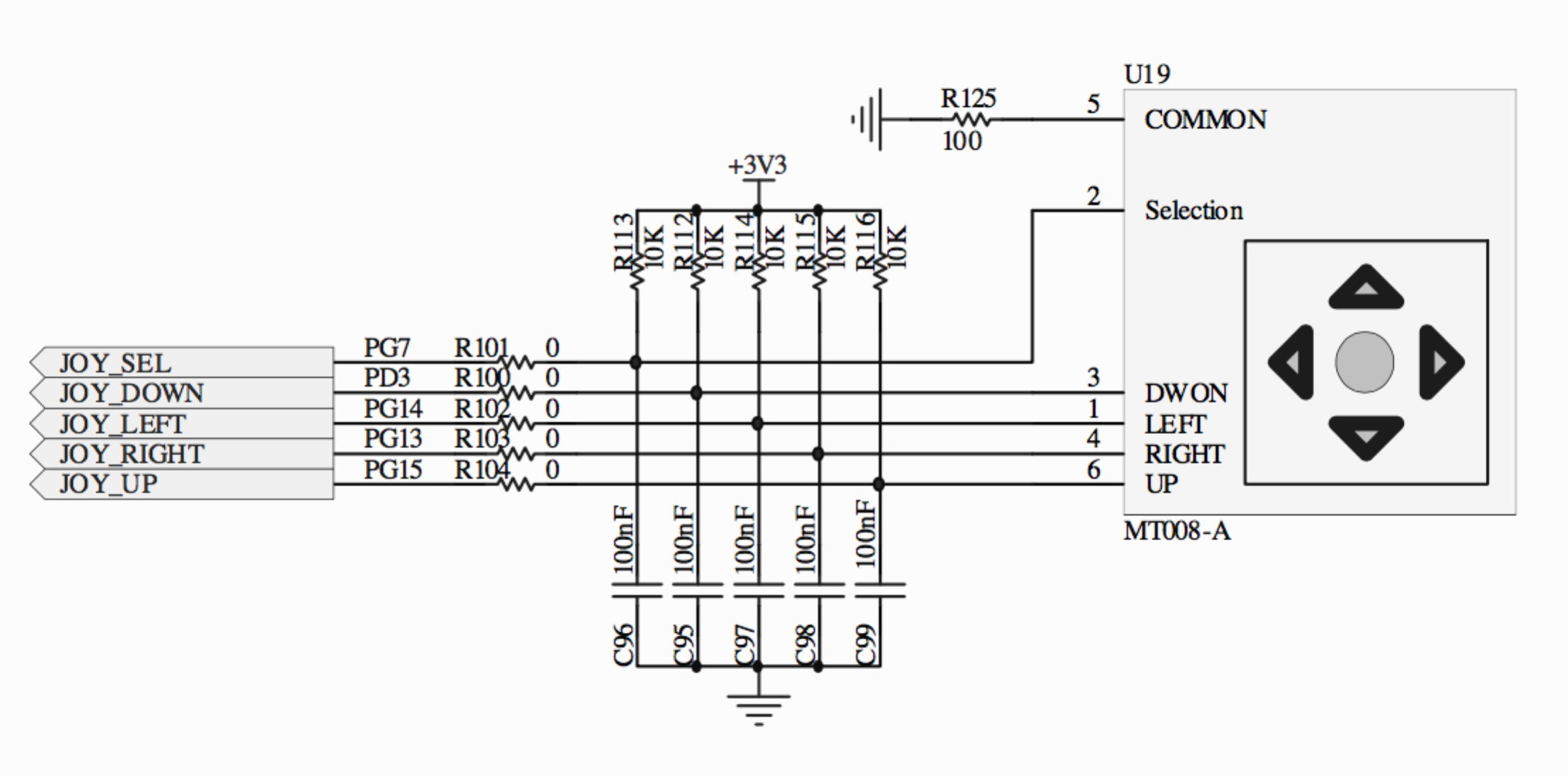


图 3.6 操纵杆管脚引出电路图

图3.7是名为Key按键的电路图，与MCU的PG8相连，负责停止音乐播放。

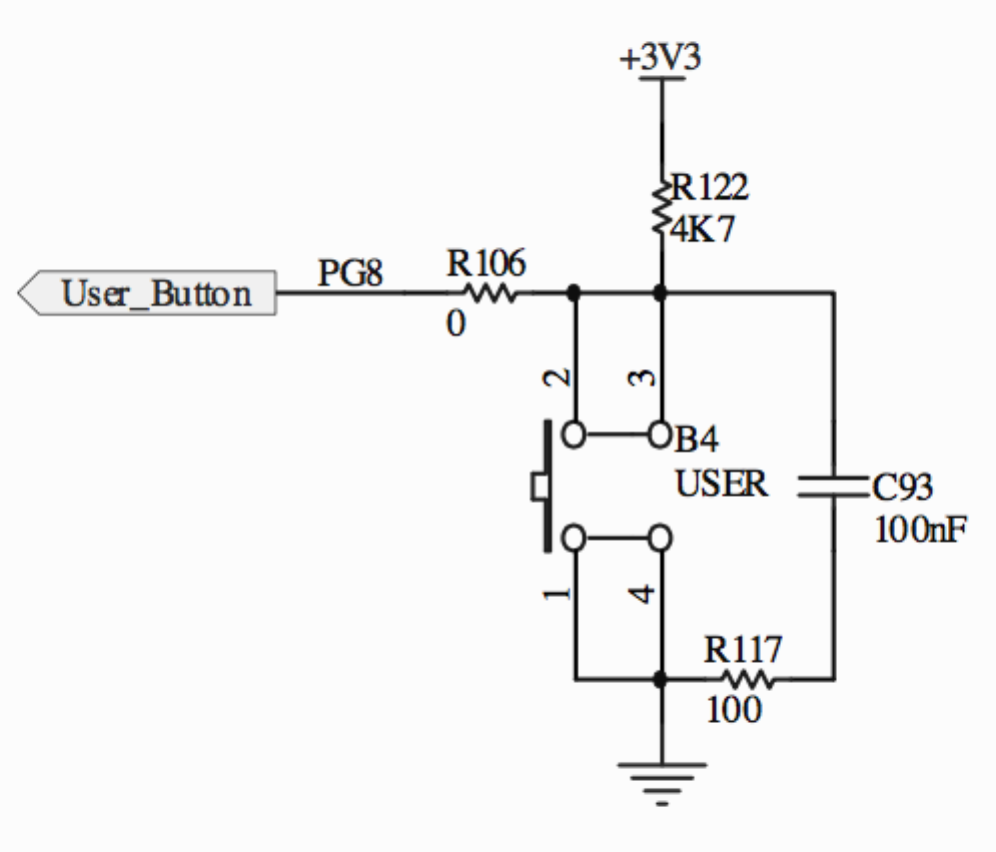


图3.7 Key按键管脚引出电路图

图3.8是名为WAKEUP按键的电路图，与MCU的PA0相连，负责在系统启动后和停止后重新播放音乐。

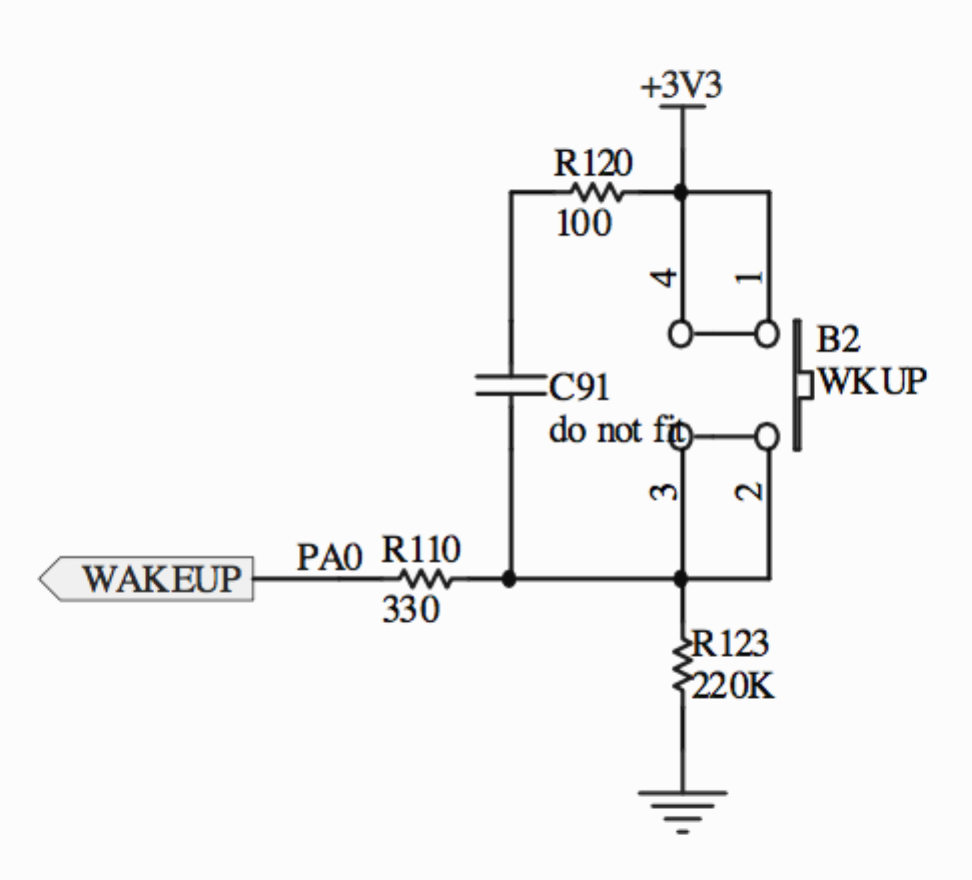


图 3.8 Wakeup按键管脚引出电路图

图3.9是名为Tamper按键的电路图，与MCU的PC13相连，负责退出程序。

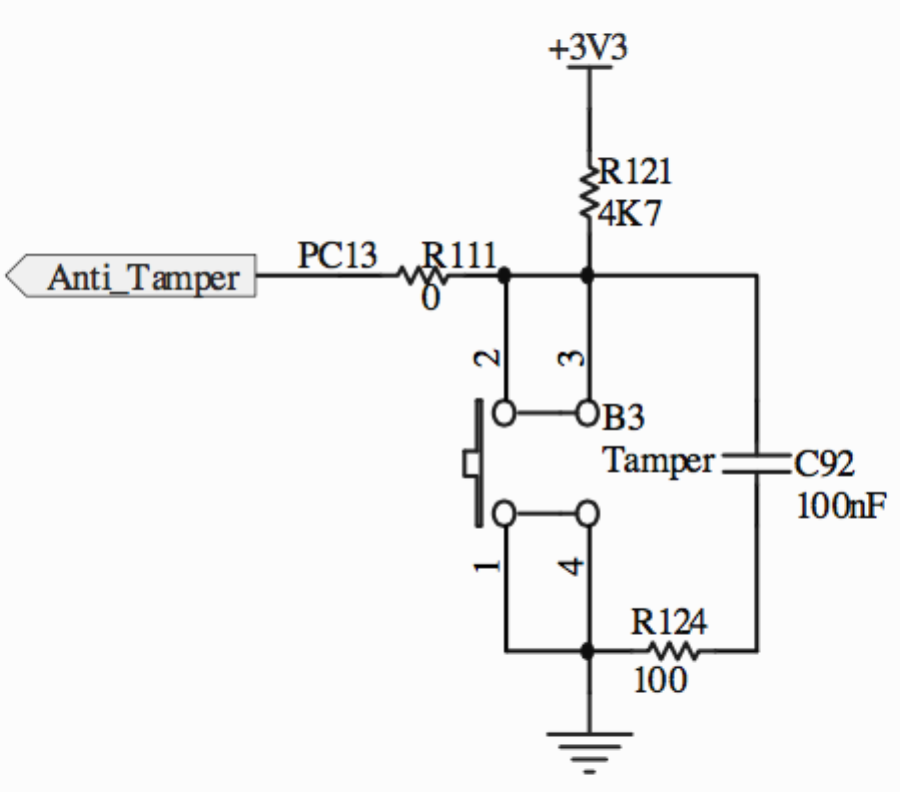


图 3.9 Tamper按键管脚引出电路图

良好的硬件设计为软件的正常运行提供了一个可靠平台，下面将介绍本项目的软件设计模块。

第四章 软件设计

在嵌入式系统开发中，软件设计是最为重要的一步，对于此项目来说，是否拥有足够的功能与优异的性能，取决于软件设计的好与坏。在本章中，首先介绍了在此项目中所使用的相关技术及知识背景，分析它们在此项目中的使用方式；然后依据第二章所分析的WAV文件格式，完成WAV音频解码算法；最后利用已完成的驱动程序及外部按键，设计并实现了播放器的基本功能。

4.1系统概述

#### 4.1.1方案选择

数字音频播放器是一个数据流型的软件，在它运行的每时每刻都会有大量的数据被处理。这些数据首先从SD卡中读取出来，然后经过MCU的处理，通过DAC输出到扬声器，最后形成所听到的音乐。而如何能够利用现有的技术，协调有序的实现这一过程，同时还要实现相应的控制功能，则是本章重点讨论的内容。

在音频文件的解码过程中，通常会有两种选择：软解码和硬解码。软解码直接使用CPU来处理数据，而硬解码则通过使用特定的解码芯片来处理数据。在查阅了相关资料后，提出了以下两种方案：

1. 在查阅资料的过程中，发现使用专门的音频解码芯片——也就是硬解码——成为了大多数人的选择。使用音频解码芯片的好处是可以不必了解复杂的音频解码算法，只需把从文件系统中读取出来的数据直接按顺序输送给芯片即可，后续工作则全部都交给了芯片来完成。而坏处则是由于使用了额外的芯片，成本会有相当幅度的增加，并且还需要重新设计电路主板，增加了开发的难度。
2. 经过与导师的深入交流以及对WAV音频文件的分析后，发现可以通过编写解码算法来实现音频文件的软解码。由于WAV采用PCM编码，因此数据很容易被读取出来，同时播放的音质也非常不错。由于采用的是软解码，因而主要的难点在于对音频文件的解码算法上。

经过对两种方案的考虑，最终选择采用方案二来完成对音频文件的软解码，因此，读取并解码音频文件这一部分将会成为整个项目的设计重点。下面，通过逐一的介绍，来一步步地揭开它神秘的面纱。

下面是此次项目中所使用的开发环境：

（1）硬件环境：

* + STM3210E-EVAL开发板；
  + J-Link仿真器；
  + 万用表；
  + 带功放的扬声器及耳机。

（2）软件环境：

* + Keil MDK-ARM 开发套件；
  + 串口调试软件：串口调试助手V1.exe；
  + GoldWave；
  + UltraEdit。

#### 4.1.2框架设计

根据之前所做的需求分析，本项目把软件设计模块分为了五个主要的部分，如图4.1所示。

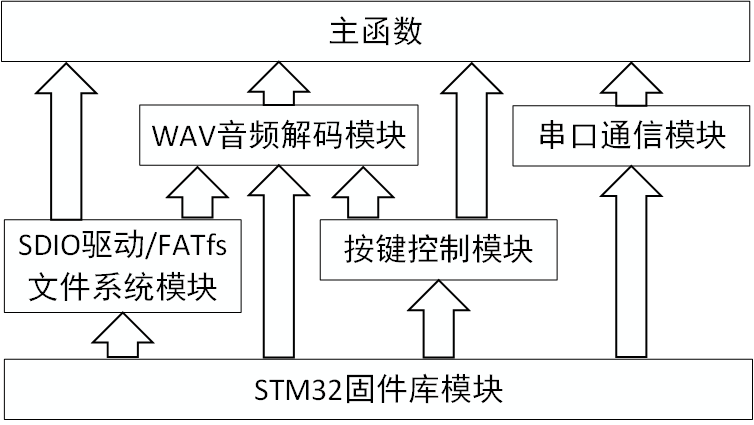


图 4.1软件总体框架图

其中，串口通信模块主要负责接收和发送来自PC端的消息，方便在开发过程中调试程序；STM32固件库是由S公司针对STM32芯片提供的函数接口，开发者可以调用这些函数接口来配置STM32的寄存器，使开发人员得以脱离最底层的寄存器操作，极大的提高了开发效率。外部按键模块为用户提供了人机交互的方式，SDIO驱动和FATfs文件系统为音乐文件的存储与操作提供了良好的技术基础；WAV音频文件解码模块则实现了WAV解码算法。

4.2相关技术

#### 4.2.1 SDIO驱动

1. SDIO介绍

SDIO（Secure Digital Input and Output Card），中文名安全数字输入输出卡，是在SD标准上定义的一种外设接口。目前，SDIO主要有两类应用——可移动和不可移动。可移动设备作为Palm和Windows Mobile的扩展设备，用来增加蓝牙、照相机、GPS和802.11b功能。

SDIO和SD卡规范间的一个重要区别是增加了低速标准。SDIO卡只需要SPI和1位SD传输模式。低速卡的目标应用是以最小的硬件开支支持低速I/ O能力。低速卡支持类似调制解调器、条码扫描仪和GPS接受器等应用。对“组合”卡(存储器+ SDIO)而言，全速和4位操作对卡内存储器和SDIO部分都是强制要求的。

STM32的SDIO接口与标准SDIO还有少许区别，区别主要在于数据线的数量。如图4.2，它具有8条数据线——SDIO\_D[7:0]，STM32这样设计是为了支持部分MMC存储卡，而这些MMC卡使用的是8条数据线。

控制单元

命令通道

数据通道

卡总线

适配器寄存器

FIFO

SDIO\_CK

SDIO\_CMD

SDIO\_D [7:0]

SDIOCLK

HCLK/2

至AHB接口

SDIO适配器

1. SDIO的命令格式

图4.2 STM32的SDIO接口

SDIO的所有命令及命令响应，都是通过SDIO\_CMD引脚来传输的，并且命令只能由主机即STM32的SDIO控制器发出。

SDIO协议规定了非常多的命令，把这些命令分类整理也有11种之多。参照命令格式表，见表4.1。一个命令包含了6个段，分别为起始位、传输位、命令索引、参数、CRC7和结束位，其中除了命令索引段和参数段是需要在软件配置的时候设置的，其他段都由硬件完成。命令索引段是指SD协议规定的命令编码，如命令CMD0、CMD1…的编码为0、1…。有的命令会包含参数，如读命令的参数为要读取数据的地址，这些命令参数存放在参数段。

表 4.1 SDIO命令格式

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 位 | 47 | 46 | ［45:40］ | ［39:8］ | [7:1] | 0 |
| 宽度 | 1 | 1 | 6 | 32 | 7 | 1 |
| 数值 | ‘0’ | ‘1’ | X | X | X | ‘1’ |
| 描述 | 起始位 | 传输位 | 命令索引 | 参数 | CRC7 | 结束位 |

除了CMD0命令外，SD卡在接收到命令后都会返回一个响应。对于不同的命令，会有不同的响应格式，一共有七种，简称R1～R7.按响应的字节长度又分为长响应型（136bit）和短响应型（48bit）。

1. SDIO驱动移植

由于SDIO的通信协议相当庞大，要自行编写完整而严谨的SDIO驱动不仅费时费力，而且极易出错。因此在本项目中，通过移植官方例程中的驱动文件来实现对SD卡的操作。

官方的SDIO驱动文件提供了许多有用的API，通过使用这些API，开发人员可以很方便的读写SD卡上的数据。下面介绍几个主要的函数：

1. SD\_Init()

在开始阶段调用SD\_Init()来进行SDIO的初始化，同时把返回值附给Status变量，若返回值等于枚举元素SD\_OK，则初始化成功。在SD\_Init()函数中还对纪录了SD卡信息的全局结构体变量SDCardInfo进行了赋值。

1. SD\_WriteMultiBlocks()

调用此函数，可以实现向SD卡中写入多个数据块。同理，SD\_WriteBlock()为写入单个数据块。

1. SD\_ReadMultiBlocks()

调用此函数，可以实现从SD卡中读出多个数据块。同理，SD\_ReadBlock()为读出单个数据块。

使用官方提供的函数，很大的提高了项目的开发速度和可靠性。在接下来的FATfs文件系统移植中，就是通过调用这些函数来连接文件系统模块和SDIO驱动，最后实现对装有FAT文件系统的SD卡进行数据读写。

#### 4.2.2 FATfs文件系统

1. FATfs简介

FATfs是一个通用的文件系统模块，常用于在小型嵌入式系统中。由于它是由ANSI C语言编写并且完全独立于底层的I/O介质，因此可以很容易地不加修改地移植到其他处理器当中，例如8051、PIC、ARM等。文件系统庞大而复杂，它需要根据应用的文件系统格式而编写，并与驱动层分离开来，以方便移植，所以在工程应用中一般是移植现成的文件系统源码。FATfs文件系统支持FAT12、FAT16、FAT32等格式，本项目利用前面写好的SDIO驱动，配合FATfs文件系统，就可以利用文件系统的各种函数，对装有FAT文件系统的SD卡的文件进行读写了[9]。

1. FATfs目录结构

在移植FATfs文件系统前，要先到FAT的官网来下载源码，为了稳定起见，此项目并没有使用最新的版本，而是采用了比较常用的R0.09版本。

移植后的头文件和源代码如图4.3所示。

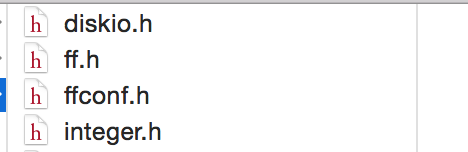




图 4.3 FATfs头文件和源代码

其中：

1. integer.h：文件中包含了一些数值类型定义；
2. diskio.c：包含底层存储介质的操作函数。这些函数需要用户自己实现，主要添加底层驱动函数，用来连接FATfs文件系统与SDIO驱动；
3. ff.c：独立于底层介质操作文件的函数，开发人员可以利用这些函数实现文件的读写；
4. cc936.c：中文字库，时简体中文支持所需要添加的文件，包含了简体中文的GBK和转换函数；
5. ffconf.h：这个头文件包含了对文件系统的各种配置，开发人员可以在此配置所支持的语言及是否支持长字节文件名的地方。
6. 为文件系统添加底层驱动

FATfs文件系统与底层介质的驱动分离开来，对底层介质的操作都要交给用户去实现，它仅仅提供了一个接口函数，函数为空，要用户添加代码。因此要把diskio.c中的函数接口与前面的SDIO驱动连接起来。

根据FATfs帮助文档的说明，需要由用户实现存储介质初始化、存储介质状态函数、扇区读取函数、扇区写入函数、和控制功能等五个函数原型，之后FATfs的应用函数就可以调用这些函数来操作SD卡。下面来介绍几个主要的函数实现：

1. 初始化函数接口

disk\_initialize()为文件系统初始化函数接口，文件系统的一些函数会调用这个函数接口来进行底层存储介质的初始化。这个初始化函数接口调用了SDIO的SD\_Init()函数，返回初始化成功或失败的参数，当文件系统的上层函数调用此函数时，实质调用了SD\_Init()函数对SD卡进行初始化。

1. 扇区读取函数

接下来是disk\_read()函数，他是文件系统读取SD卡数据会调用的一个函数，所以在此加入SDIO驱动的读取函数的接口。

1. 扇区写入函数

扇区写入函数disk\_write()与扇区读取函数十分类似，也是根据写入的扇区数目是一个还是多个来分别调用不同的SD数据块写入函数。在调用了写入函数之后，也要等待SDIO发送完毕及DMA传输结束。

1. 添加简体中文和长文件名支持

在使用音乐播放器的时候，经常会遇到有些音乐的文件名很长，比如这个：《Janos Starker - Suite No. 1, BWV 1007, In G： Prelude.wav》，而且许多歌曲都是中文名，比如这个：《久石让 - 太阳照常升起.wav》，因此，为了让系统支持简体中文和长文件名，就需要对文件进行配置。

在FATfs文件系统里，可以在ffconf.h头文件中对文件系统进行剪裁。在这个文件中，修改如下的宏定义，如图4.4。



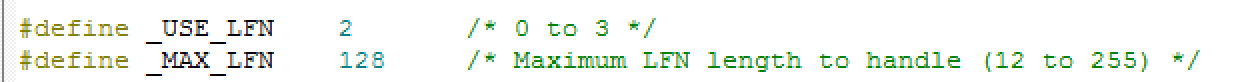


图 4.4 ffconf.h文件中的宏

三个宏定义分别表示：

* 把\_CODE\_PAGE宏配置成简体中文的code\_page，936。
* 把\_USE\_LFN宏配置成2，表示开启长文件名支持。
* 把\_MAX\_LFN宏配置成128，表示可支持的长文件名的最大字节数。

第一个和第三个很好理解，这里要对第二个，也就是\_USE\_LFN宏做一个详细的说明。

图4.5是关于宏\_USE\_LFN各个值的官方定义：

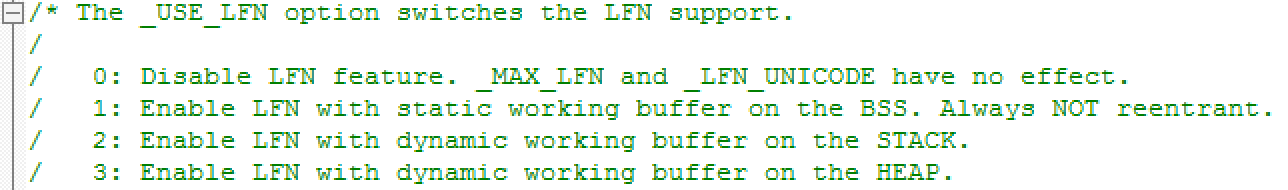


图 4.5 宏\_USE\_LFN各个值官方定义

由于栈区存放的内容在子函数调用后会被释放，而且少用全局变量会让代码的移植性更好，因此在本项目中把文件系统长文件名的存储空间放到栈区。

关于软件设计模块的介绍先到此为止，接下来将通过对数字音频播放器系统的分析，由内而外的揭示出它的工作原理。

4.3 WAV音频解码算法

根据对WAV音频文件格式的分析，这一节主要来介绍解码算法的实现过程[7]。

#### 4.3.1主要函数介绍

本文所实现的解码算法，主要由以下几个函数组成：

1. void WavPlay(WavHeader \*wavheader, char \*Apath)

音频播放的起点，在main函数中调用，执行一首音乐的完整播放，返回值为零。在这个函数中首先对头文件进行初始化，然后初始化DAC和中断定时器；紧接着进入到正式播放的for循环中，在把数据从SD卡中读入缓存的同时，通过定时器中断完成DAC音频输出，直到整首音乐播放完毕为止。参数\*wavheader是一个由main函数声明的空的结构体指针，为3.3节中所定义的WAV头文件类型，用来存放即将播放的WAV音频文件的头文件信息；\*Apath是一个字符型指针，存放着即将播放的音频文件的绝对路径，通过这个绝对路径，可以从装有FAT文件系统的SD卡中读取选定文件里的数据。

1. void WavPlayConfig(WavHeader \*wavheader, char \*Apath)

音频播放的初始化函数，在WavPlay函数的起始处调用，主要用来完成对WAV头文件的读取及检测工作，同时完成播放控制和双缓存的初始化工作。在这个函数中，首先调用了f\_open()和f\_read()两个文件操作函数来读取文件的前44个字节的头文件，并存储到header数组中以供头文件解析函数使用；在完成解析函数后， 逐一检测头文件中的RIFF位、WAVE位、fmt位和data位，若符合要求，则头文件解析成功，反之则失败。

1. void WavHeaderInit(WavHeader \*wavheader)

WAV头文件的解析函数，在WavPlayConfig函数中调用。在此函数中，按照WAV头文件的格式，调用WAV\_Info和WAV\_Num两个函数来依次读取存放在header数组中的数据，并逐一填充到\*wavheader结构体中。

1. void WAV\_Info(uint8\_t \*Info, uint32\_t Addr, uint32\_t Num)

从Addr位开始，以一字节为一单位，读取header 数组Num次，并依次放入到\*Info成员中，最后还需要添加一个‘\0’位作为结尾标志。\*Info为被填充的头文件结构体成员；Addr为全局变量，表示当前读到头文件成员应该在header数组中的起始地址位，初始化为0，每读完一个头文件的成员，Addr就要增加到下一个成员的起始位，直到读完全部44个字节为止。在这里\*Info指的是‘RIFF’、‘WAVE’、‘fmt ’和‘data’四个成员，因此Num也为4。

1. uint32\_t WAV\_Num(uint32\_t Addr, uint32\_t Byte)

从Addr位开始，以Byte个字节为单位，读取header数组一次，并以整型的类型返回给相应的结构体成员。在这里需要注意的是，由于一次只能读取一个字节，即8bit，因此当Byte大于一个字节时需要同时读取多个字节，然后重新组合读取的数据。

1. void Time\_Init(WavHeader wavheader)

中断定时器初始化函数，用来把wavheader头文件中重要的数据赋值给定时器结构体，初始化并使能中断。在这个函数中主要是利用wavheader中的采样频率及晶振频率来配置定时周期。配置完成之后，定时器就会每隔一段时间出发一次定时器中断，而每一次定时器中断就会执行一次中断处理函数。

1. void TIM3\_IRQHandler()

定时器三的中断处理函数，由定时器三触发，实现DAC的音频输出，这个函数也是数字音频播放器的核心函数之一。根据第三章所讲到的数据存储格式，需要根据声道数和量化位数的不同来对读入的数据分别进行处理，一次只处理一个音频数据。声音是由震动产生的，对于扬声器而言，每一次的震动都是由两个不同的电压切换所产生的，而音频文件中的每个数据都代表着一个电压的相对值，把这些相对值按照一定的频率来切换时，就可以重现音频文件中所记录的音乐信息。

需要注意的是，由于每一位的缓存都只保存了一个字节——即8bit——的数据，因此在播放16bit的音频文件时需要重新组合数据；同时由于数据是16bit，且有正负之分，而内置DAC只有12位，因此还需要对此进行处理。

#### 4.3.2解码流程分析

图4.6为音频解码流程图：

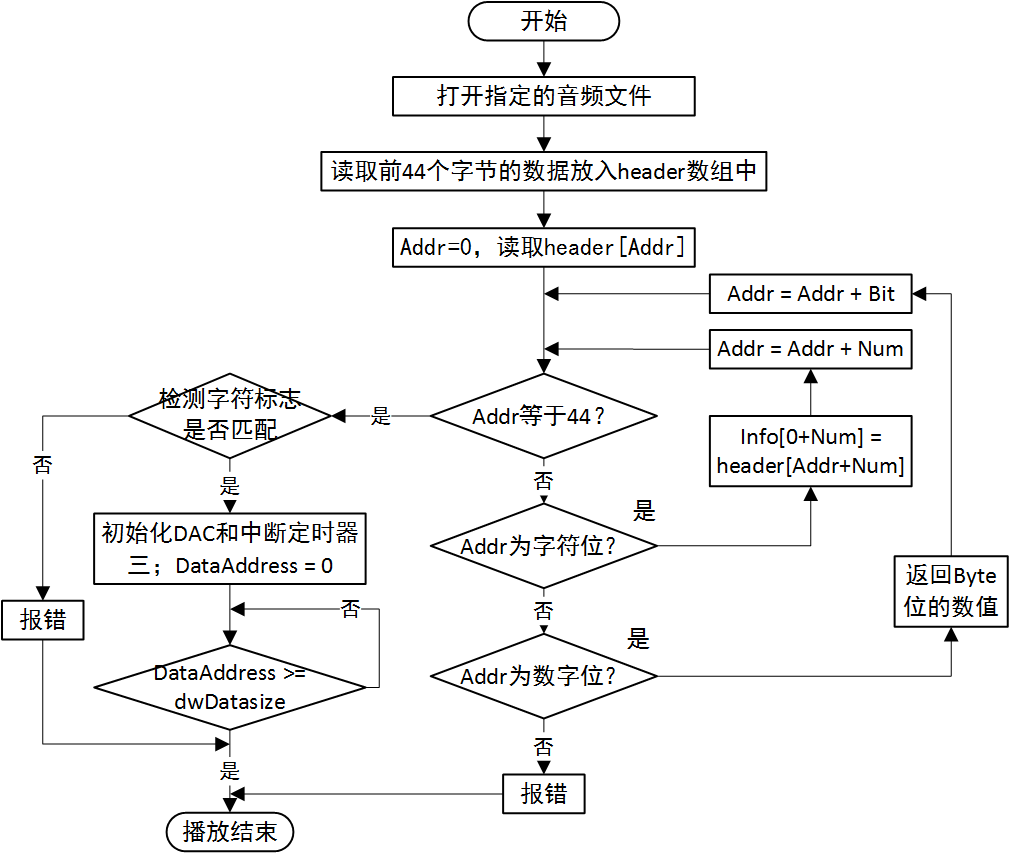


图 4.6 头文件初始化流程图

当完成对DAC和中断定时器三的初始化后，定时器三开始不断触发中断，每触发中断就执行一次中断服务函数，通过中断服务函数来实现DA输出。中断服务函数流程图如图4.7:

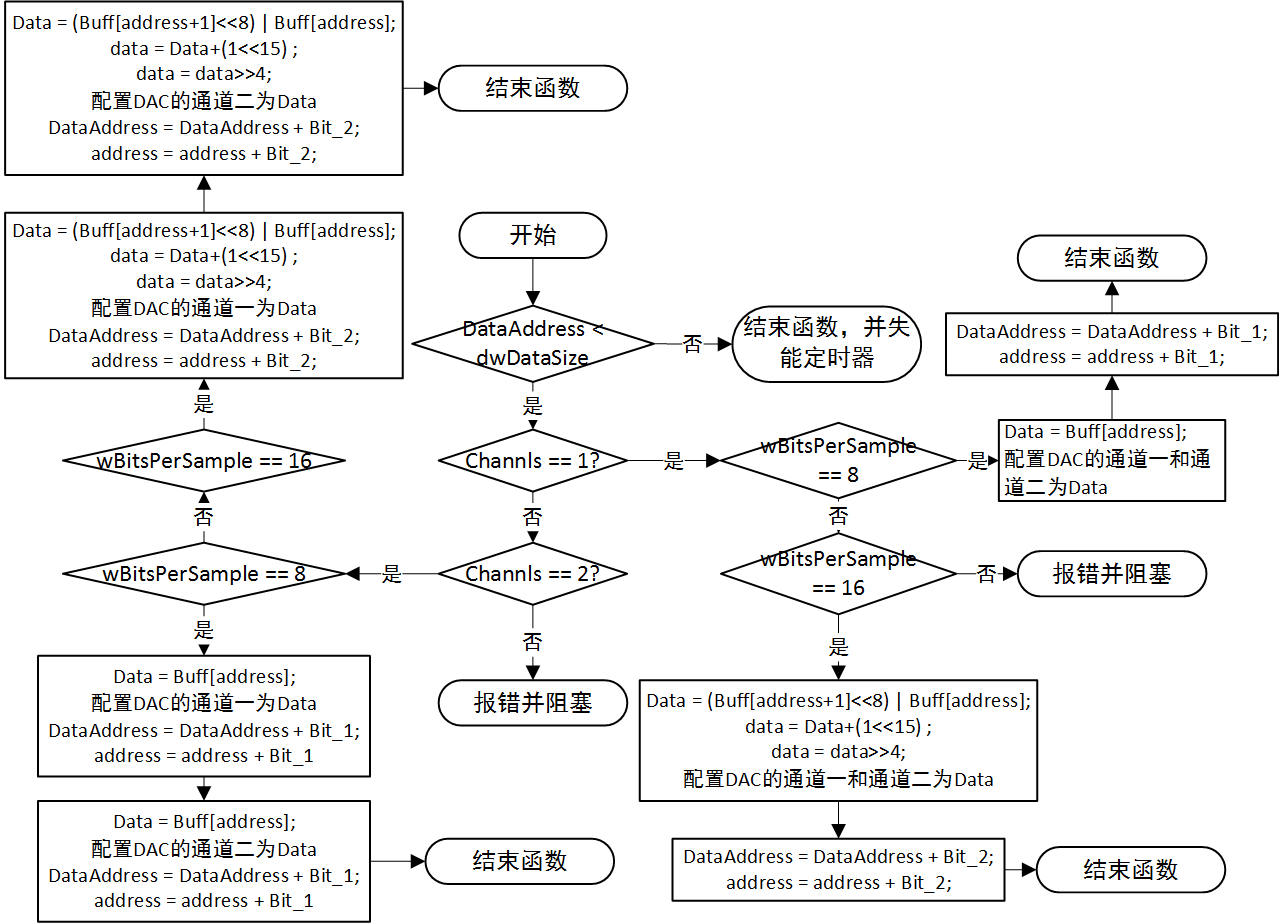


图 4.7 中断服务函数流程图

当DataAddress >= dwDataSize时，当前数据相对地址大于数据大小，失能定时器三，结束定时器中断，同时结束主函数中的for循环，一首歌就这样播放结束了。

4.4双缓存

#### 4.4.1概述

在上面的WAV音频解码算法中，当触发定时器中断时，相应的中断服务函数开始以特定的格式从Buff中读取音频数据，经处理后送到DA并输出到扬声器，但由于音频文件数据是存放在具有FAT文件系统的SD卡中，而Buff则是一块在内存中开辟出来的地址空间，因此，在播放之前就需要把SD中的数据先读入到Buff中，然后才能完成后续的播放。

有些人会问了，为什么不直接把SD卡中的数据读入到DAC中呢，那样岂不是效率更高？之所以没有采用这个方法，主要是因为FAT文件系统所致，在使用文件系统的时候，通常是通过调用f\_read函数来实现对文件数据的读取，但是，经过测试通过f\_read函数来读取数据的速度远低于直接读取SD卡的速度，而且f\_read中的size的值越大，读取的速度越快，这又是为什么呢？原来在调用f\_read后，首先是一大串需要CPU处理的必要的FAT指令操作，之后才是数据的传输，而由于数据传输是由DMA来完成的，因此整个传输过程的瓶颈就在指令操作，当读取的数据流速度低于音频播放的数据流速度时，播放就无法正常进行。因此，就需要调用f\_read读取适量size的数据到Buff中，然后在从Buff中播放，而由于内存的限制，size的值不可能去无限大，因此在本项目中size的值为10\*1024字节。

通过这样的设计，已经可以实现从SD卡中播放音频数据，然而，在调用f\_read从SD卡中读取数据到Buff时，由于读数据需要时间，因此会在整个播放过程中形成一个真空期，Buff中没有正确的数据可被读，这样导致的结果就是在播放的过程中会有明显的卡顿现象。而双缓存的方案，就是为了解决这个问题而出现的。

#### 4.4.2设计方案

双缓存，顾名思义，就是通过设置两个缓存来实现一边播放一边读取数据。当定时器中断正在播放缓存一中的数据时，调用f\_read往缓存二里写数据，当完成对缓存一中数据播放时，定时器中断就开始播放缓存二中数据，由于缓存二中的数据是紧连着缓存一中的数据，因此音频数据几乎没有受到损失；在播放缓存二的同时，开始调用f\_read往缓存一中写数据。重复此过程，直到音频文件中所有的数据全部被读取为止。

要实现这样的这样的过程，首先需要为每个缓存配置一个状态机，以确定每个缓存所处的状态。请参考表4.2。

表4.2 Buffer(缓存结构体)

|  |  |  |
| --- | --- | --- |
| 名称 | 含义 | 数据类型 |
| Data | 缓存数据存放处 | BYTE(BuffMax) |
| Read | 读状态，可设为Noset/Set/Setting | uint32\_t |
| Write | 写状态，可设为Noset/Set/Setting | uint32\_t |

其中Data为缓存空间，char类型，容量BuffMax为之前所定义的的10K字节；状态Read为此缓存当前读属性的状态，分为Setting、Set和Noset三种，其中Setting表示此缓存正在被播放，Set表示此缓存已被填满正确的数据，Noset表示此缓存中的数据已被播放完成，没有可读数据。图4.8为三种状态间的关系：

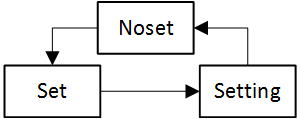


图 4.8 状态转换图

Buff为一个char型指针，它指向了当前正在播放的缓存的头地址，这样做的好处是在每次切换需要读取的缓存的时候只需改变Buff指针的地址位即可。

图4.9清晰的展示出了双缓存的工作流程图：

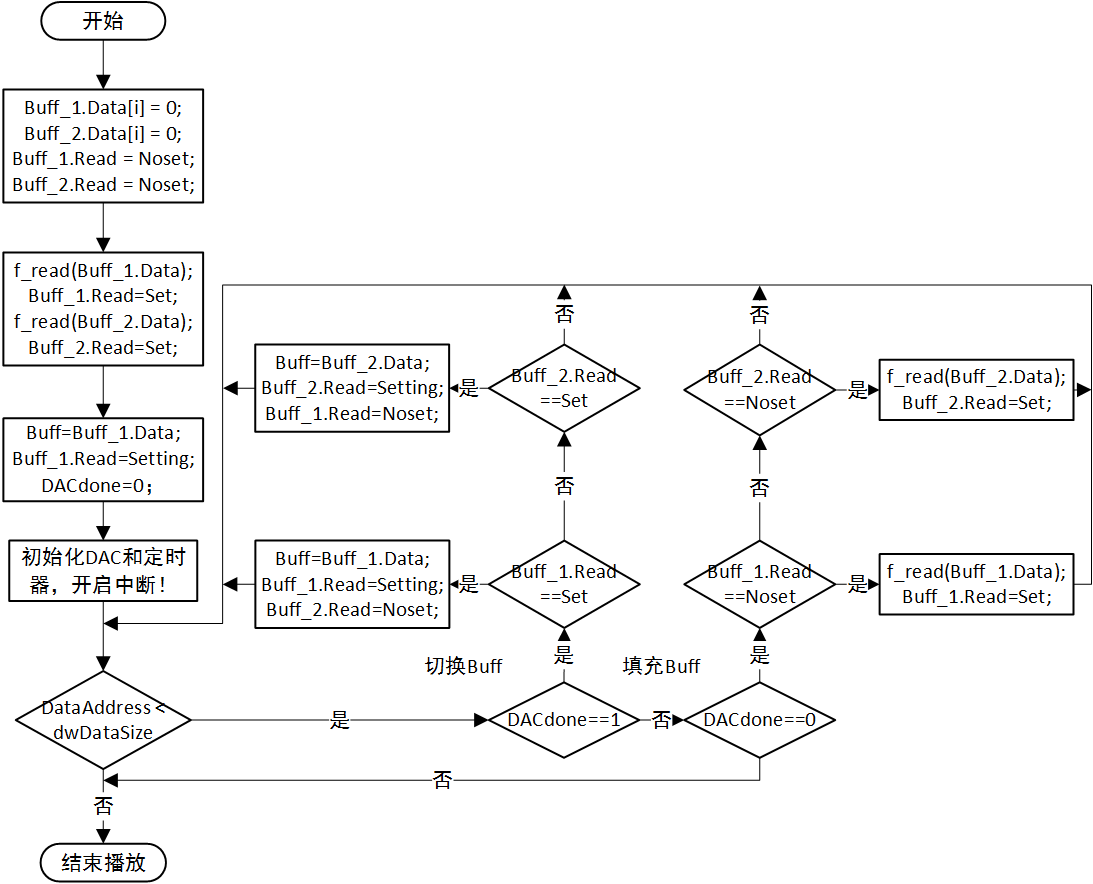


图 4.9 双缓存流程图

从图4.6，图4.7和图4.9可以看到，双缓存的工作流程与之前的解码流程及播放流程是紧密结合在一起的，这几者相互配合，共同实现了WAV音频文件的读取、解码和播放功能。

4.5播放控制

#### 4.5.1设计需求

在完成了针对WAV文件的音频解码算法后，已经可以成功的解码并播放FLASH里的WAV音频文件；在把双缓存模式与带有FAT文件系统的SD卡一同融入到音频解码算法中以后，播放器已经具有完整播放一首选定的WAV格式的高品质音乐的能力，这也就意味着本项目的核心部分即驱动层已经基本完成。然而，截止到目前为止，针对整个系统的控制是通过串口通信的方式来进行的，而要让这个数字音乐播放器可以独立于PC来运行和控制，则需要使用其他的交互方式。因此，下一步的工作，就是找到一种新的交互方式来代替串口通信，实现其对整个系统及音乐播放的控制，并优化其内部逻辑，使之可以实现一个播放器应有的基本功能，而最容易实现的，则是使用外部按键来作为交互方式。

#### 4.5.2方案介绍

STM3210E-EVAL提供了丰富的外部按键，包括一个重启按键Reset；三个自定义按键Wakeup、Tamper、Key；以及一个五向摇杆Joystick，共计九个物理按键，其中Reset无法定义，因此在这里使用的是Wakeup、Key、Tamper、Joystick，共计八个按键来实现操作功能。在上面的硬件设计中已经详细介绍了各个按键的工作原理，因此这里的重点将放在如何通过编写程序使用这些按键来实现对系统及音乐播放可控制[11]。

不同于单独播放一首歌曲，一个完整的音频播放器应该能够实现音乐的播放、停止、暂停和继续播放这四个基本功能，同时音量的可控是每个播放器都应该具有的功能，而对播放列表中的所有音乐的上一曲／下一曲操作则更是一个可用的音乐播放器的基本功能。

为了实现这些功能，首先需要在播放开始前建立一个播放列表，用来存储指定文件夹下每个wav音频文件的编号、文件名、大小和绝对路径，以及音频文件的数量，在这里通过建立一个结构体数组来实现这一点。

请参考表4.3和表4.4.

表4.3 WavFile(文件信息)

|  |  |  |
| --- | --- | --- |
| 名称 | 含义 | 数据类型 |
| ID | 播放编号 | int |
| Name | 音频文件名 | char(NumStrMax) |
| Apath | 音频文件在SD卡中的绝对路径 | char(NumStrMax] |
| Size | 文件大小 | long int |

表4.4 PlayList(播放列表)

|  |  |  |
| --- | --- | --- |
| 名称 | 含义 | 数据类型 |
| File | 文件信息结构体 | WavFile(NumFileMax) |
| Number | 播放列表中的音乐数量 | int |

通过调用ShowAllFile()函数来从SD卡中读取指定目录下的文件列表，并放入在主函数中声明的Playlist播放列表中，由于ID编号是从0开始的，因此设置Chose为0来默认选择播放第一首歌。

紧接着便进入主循环中。

为了执行主函数以外的程序，中断是最好的选择，因此，这里的七个按键全部采用外部中断来触发相应的中断服务程序。中断服务程序会根据自己的优先级而先后执行，直到执行完成后才把CPU的使用权限交给下一个程序，而当一个中断服务程序占用了太多的CPU资源的时候，整个系统就会因为资源不足而发生卡顿，甚至是死机，俗称跑飞了。因此，为了不妨碍整个系统的运行，中断服务函数所耗时间应该尽可能的短[3]，如图4.10所示：

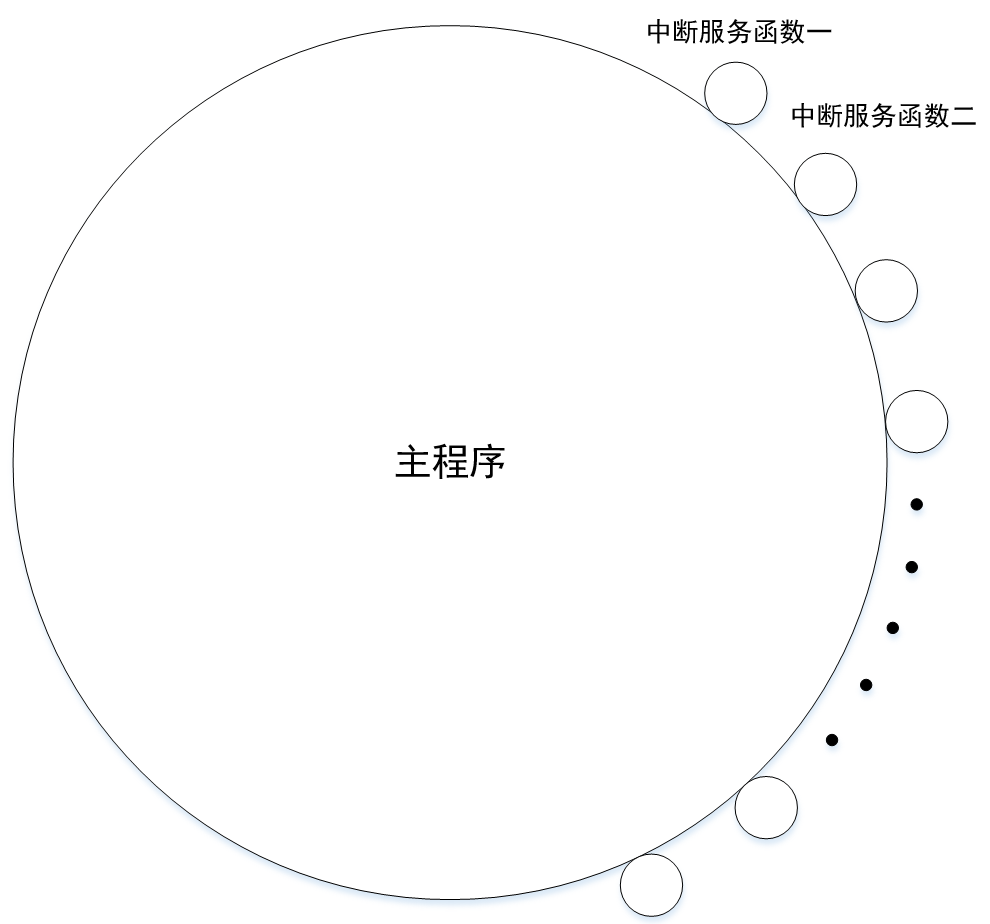


图 4.10 主函数与中断服务函数

由于中断服务函数应该尽可能的短，因此除了那些对时序要求非常高的执行程序外，中断服务函数里的主要过程都应放入主程序中执行，这就需要为每一个中断设置一个缓存，当其中一个中断被触发时，相应的缓存即发生变化，主程序在循环中发现该缓存发生变化后，进入并执行该缓存所对应的中断服务函数，而对时序要求非常高的DA输出部分则全部在中断服务函数中执行。整个播放控制模块便是以这样的模式来运行。

介绍完了播放列表和中断，接下来将依次介绍各个功能的实现过程：

1. 开始／停止播放

在此功能中，通过设置PlayIRQBuffer作为开始／停止播放的中断缓存。启动程序后，PlayIRQBuffer的值被初始化为Noset，在初始化结束后进入主循环不断检测PlayIRQBuffer的值，当PlayIRQBuffer的值被由Wakeup按键所触发的中断服务函数修改为Set时，开始播放播放列表中的第一首音乐；当在播放过程中PlayIRQBuffer被由Key按键所触发的中断服务函数修改为Noset，同时DACdone被置为2时，程序跳出播放执行函数，播放停止。

1. 暂停／继续播放

在此功能中，通过设置PlayWait作为暂停／继续播放的中断缓存。启动程序后，PlayWait的值被初始化为Noset，在音乐播放的过程中，当PlayWait的值被由Joystick中的Selection按键所触发的中断服务函数修改为Set时，播放中断定时器失能，播放暂停，PlayWait的值重新置为Noset；当在此触发Selection按键时，PlayWait被修改为Set，播放中断定时器使能，继续播放当前音乐，PlayWait的值又重新置为Noset。

1. 音量加／音量减

在此功能中，通过设置VolumeSub和VolumeAdd作为音量加和音量减的中断缓存，同时设置全局变量volume作为音量的级别，由一到八分为八个等级，初始化为三。启动程序后，VolumeSub和VolumeAdd均被初始化为Noset，当VolumeSub或VolumeAdd的值分别被由Joystick中的UP或DOWN按键所触发的中断服务函数修改为Set时，volume的值便进行相应的增减，然后VolumeSub或VolumeAdd重新被置为Noset，从而达到调节音量的作用。

1. 上一首／下一首

在此功能中，通过设置LastMusic和NextMusic作为上一首和下一首的中断缓存。启动程序后，LastMusic和NextMusic均被初始化为Noset，当LastMusic或NextMusic的值分别被由Joystick中的LEFT或RIGHT按键所触发的中断服务函数修改为Set时，首先停止播放，然后递增或递减全局变量Chose的值，以达到上一曲／下一曲的选择，最后重新进入播放函数，播放新的音乐。当播放到播放列表中的最后一首音乐时，再触发下一曲中断将会跳转到第一曲，同样的，当播放到播放列表中的第一首音乐时，再触发上一曲中断将会跳转到最后一曲。当前歌曲播放完成时，也会调用下一曲的执行函数，自动播放下一曲。

1. 结束播放

在此功能中，通过设置ShutDown作为关机功能的中断缓存。当Tamper按键被触发时，程序将首先将调用停止播放函数停止当前播放，之后跳出播放主循环来结束程序。

下面将通过流程图的方式，更加通俗易懂的介绍播放流程，如图4.11。

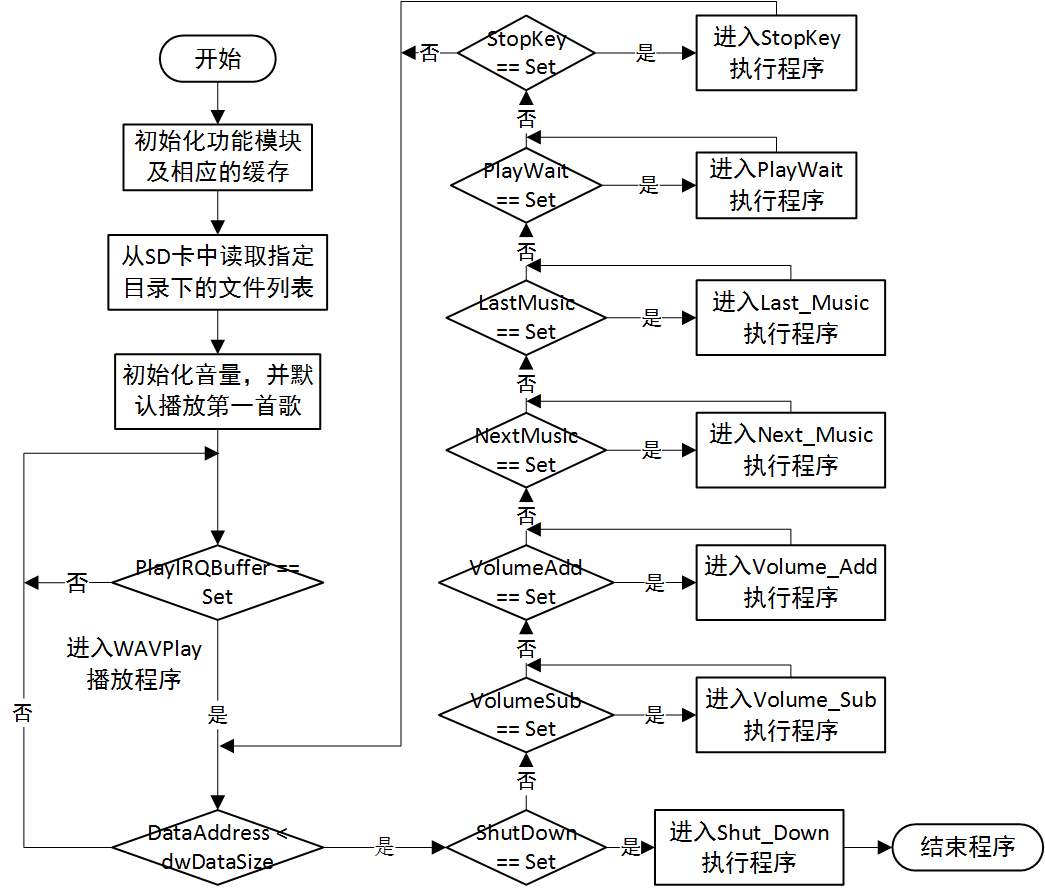


图 4.11 播放控制流程图

在执行完初始化程序之后，主函数便不断循环等待播放键的触发；进入播放程序后，使能除播放功能以外的所有功能，并在播放循环中等待各个功能键的触发，当某个功能被触发后，主循环进入相应的功能执行函数，之后重新返回主循环中，这样就实现了一个基本的音频播放器。

第五章 系统测试

在完成了项目的第一个版本的开发后，就要进入系统测试环节。通过系统测试来验证之前所设计的功能是否已全部完成，系统中是否有严重的BUG未解决，系统中的每个模块是否可以协作运行。由于本项目未使用独立的可视化模块来显示运行状态，并且输出的声音信息很难用文字和图片来展示，因此在本章的系统测试中主要是通过串口的输出信息来判断程序运行状态。

5.1测试环境

#### 5.1.1硬件环境

PC：MacBook pro；

开发板：STM3210E-EVAL；

其他配件：耳机、电源线、杜邦线，带功放的扬声器、读卡器。

#### 5.1.2软件环境

PC：Windows 8.1虚拟机、串口调试助手V1.exe；

开发板：测试固件v1.2。

5.2测试过程及结果分析

#### 5.2.1测试过程

连接电源，开发板上电后，自动开始运行写入板中的固件。首先完成串口初始化和外部中断初始化，使得串口通信和外部按键可以使用，完成后通过串口在PC端打印“Usart init!”和“EXTI init!”两条信息；当打印出SD init OK这条信息时，说明SDIO已经初始化完毕，并且系统已经正确到装有FAT文件系统的SD卡，FATfs文件系统所提供的文件操作API已可以正常使用。紧接着，在SD卡中指定的目录中读取此文件夹中所有的非隐藏的文件，存储到播放列表中，并以ID，名称，大小的顺序通过串口打印出来，最后打印出文件的数量，并把第一首歌作为默认播放音乐。串口打印信息如图5.1和图5.2所示。



图 5.1 播放列表

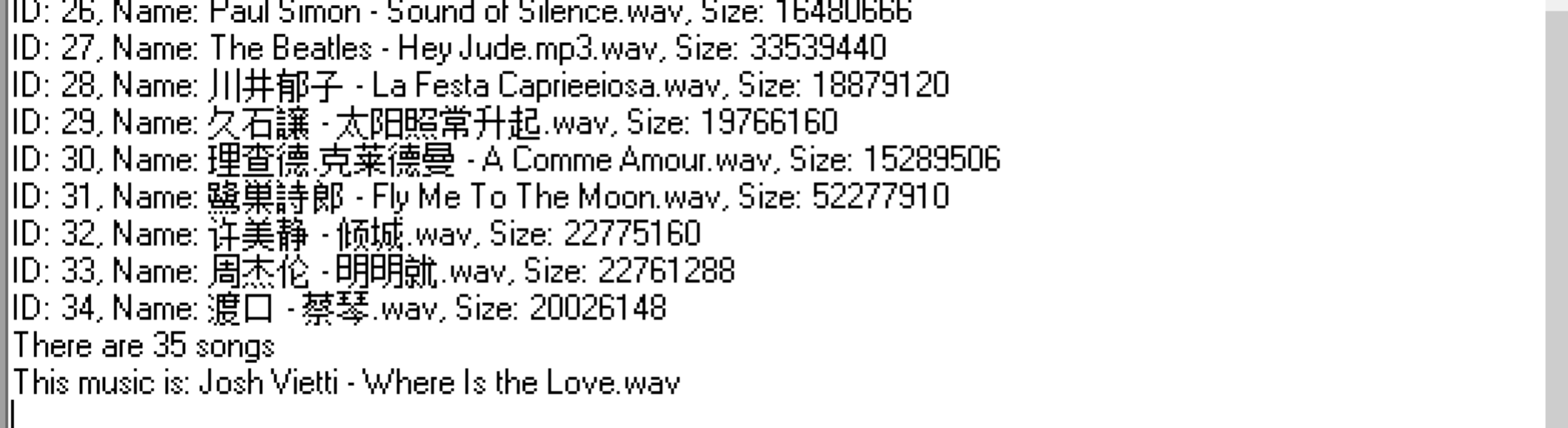


图 5.2 列表中音乐数量

接下来按下Wakeup键，当前选中歌曲开始播放，这时候可以通过耳机或扬声器听到音乐播放。串口输出如图5.3所示。

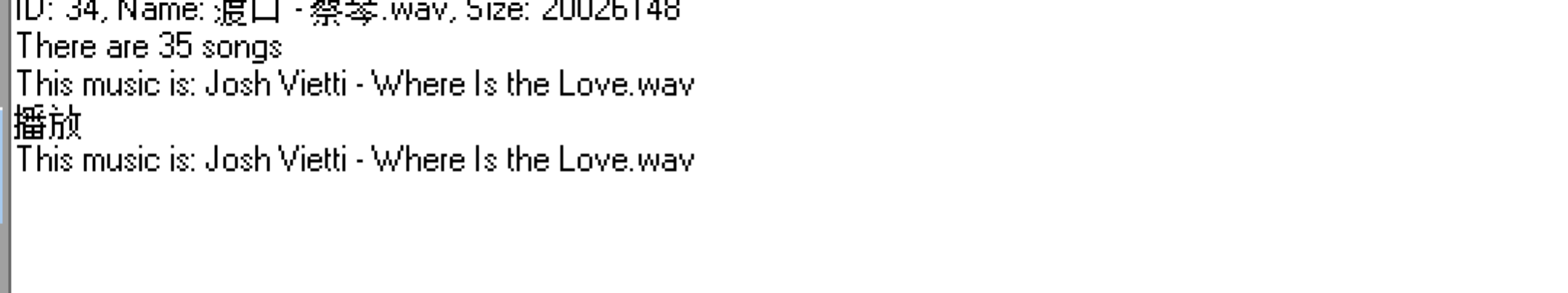


图 5.3 播放功能

在播放的过程中按下Joystick中的Selection键，音乐播放暂停。之后再次按下Joystick中的Selection键，音乐接着之前中断的部分继续播放。串口输出如图5.4所示。



图 5.4 暂停与继续播放功能

在播放的过程中，通过按下Joystick中的Up键和Down键，可以控制音量的增减。串口输出如图5.5所示。



图 5.5 音量加／音量减功能

在程序运行过程中按下Joystick中的Right键和Left键，可以实现下一曲和上一曲的播放控制，通过扬声器可以听到音乐发生了切换。串口输出如图5.6所示：

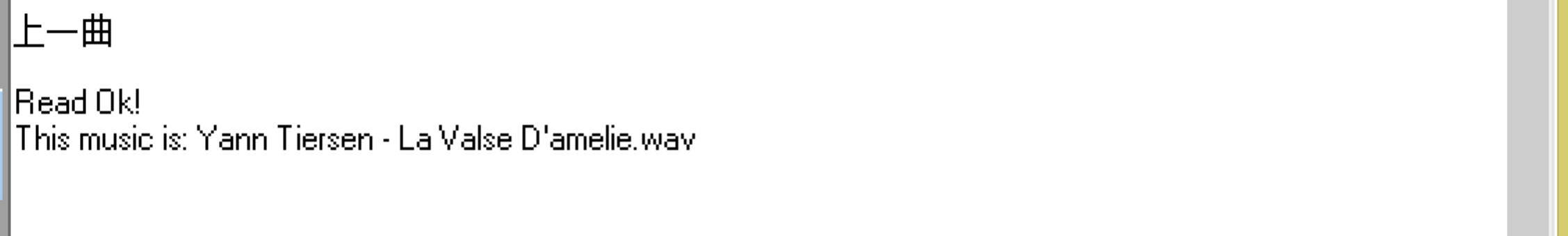


图 5.6 上一曲／下一曲功能

当播放到播放列表中的最后一曲时，再按下下一曲，则会跳到第一首音乐；同样的，当在播放第一曲时按下上一曲，则会跳到最后一首音乐，如图5.7所示；而当一首音乐播放完毕后，会自动播放下一首音乐。通过这样的设置，可以实现播放列表的循环播放功能。



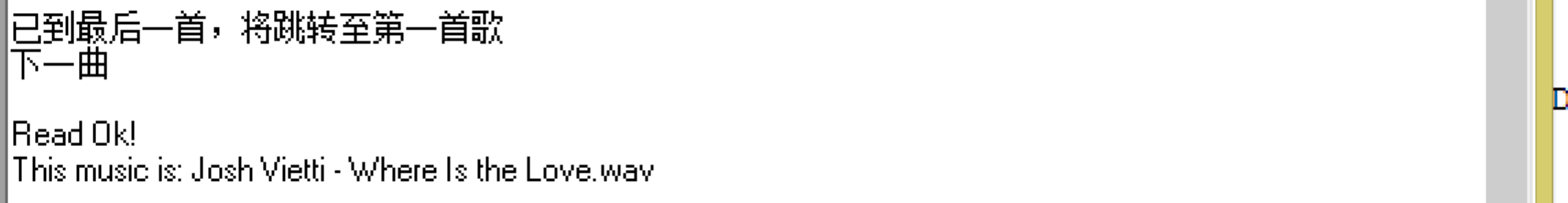


图 5.7 循环播放功能

在播放的过程中按下Key键，音乐停止播放。如图5.8所示。

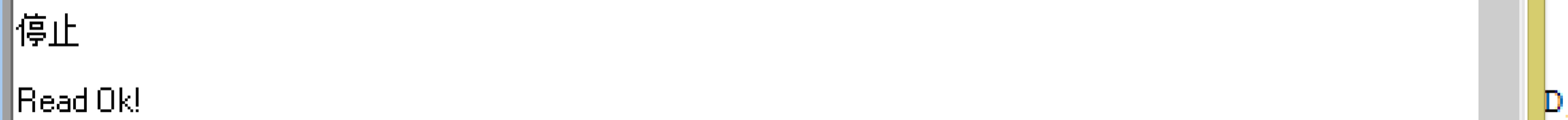


图 5.8 停止播放功能

在停止播放的时候按下Wakeup播放键，播放器开始重新播放之前停止那首歌。如图5.9所示。



图 5.9 重新播放功能

#### 5.2.2结果分析

经过测试，已完成的程序可以实现播放、停止、暂停／继续播放、音量加／音量减、上一曲、下一曲等基本功能，并且可以实现播放列表的循环播放；程序可以识别音频文件的长文件名及中文文件名，说明中文字库已经成功移植到程序中；在运行过程中，程序可以流畅而较清晰的播放WAV格式的音乐。到此为止，此项目所设计的功能已基本实现。

在测试的过程中也发现了一些问题，虽然可以播放大部分样式的WAV音频文件，但在播放最高码率——即码率为1411kbps——的WAV音乐时发生了卡顿，初步认为是由于FATfs文件系统从SD卡中读取数据的速度低于音频文件的播放速度所导致的，而由于内存的限制而无法扩大缓存来解决这个问题。同时，当文件夹中的音乐文件过多时，会出现无法完全读取所有音乐文件的情况。

第六章 总结与展望

6.1全文总结

本项目作为公司所给出的一个产品研究性题目，从一开始，便遵循公司导师的要求，严格按照嵌入式系统的开发模式，并以做产品的思路来作为开发导向，以实时操作系统RTOS的设计思想作为设计规范，来进行项目的设计与开发。

在文章中，不仅介绍了声音的基本原理以及把声音数字化的方法，还介绍了用来存储数字音频的WAV文件格式，以及针对WAV文件格式的解码算法；STM32F103作为意法半导体公司所推出的一款基于ARM Cortex-M3的32位微控制器芯片，在此项目中占据了核心地位；而SDIO + FATfs的组合，则为播放器提供了大容量的存储空间以及便捷的文件操作方式。

本项目在设计播放过程的时候，采用了在操作系统中常用的双缓存工作模式，通过对两个缓存分别进行数据的写入与数据的播放，有效的解决无法直接从SD中连续播放音频文件这个问题。

为了更好的介绍本项目的设计与开发，在本文的后半部分，分别以流程图的形式展示了不同模块的工作流程，而从所展示的流程图中可以看出，所有的模块在实际运行中都是相互嵌套，相互协作，共同实现各个功能的。

6.2改进及下一步工作

#### 6.2.1系统存在的不足

由于时间的关系，以及对嵌入式系统开发的认识不够，导致此项目还存在着诸多不足之处，具体表现为以下几个方面：

1. 无法正常播放比特率为1411kbps的WAV音频文件。主要原因在于此款MCU，即STM32F103芯片的内存过小，因此所设置的双缓存最大只能为10K，同时由于处理器的时钟频率只有72MHz，在缓存为10K的情况下，从带有FAT文件系统的SD卡中读取数据的速度最快仅为1Mb/s，而数据流小于1Mb/s的且质量最高的只有采样频率为22,050Hz的十六位双声道WAV音频文件，这也就导致目前无法正常播放采样频率为44,100Hz的十六位双声道WAV音频文件；
2. 在播放十六位WAV音频文件时失真率过高。主要原因是由于文件数据是十六位的，而在项目中使用的集成DAC是十二位，因此数据的最后四位在播放过程中丢失，从而导致这个问题。
3. 交互方式不够人性化。由于时间的关系，仅仅采用了七个外部按键来作为唯一交互方式，而用户除了通过听扬声器的方式外并没有其他方式来获取设备信息。

#### 6.2.2下一步的工作

针对上述已经存在的问题以及公司的要求，在后续的开发过程中将主要集中在以下几个方面：

1. 使用DMA模式来完成DAC的播放功能，降低对CPU资源的使用；
2. 使用开发板上来自AKM的十六位声卡芯片AK4343来代替内置的DAC，这样可以极大的提高声音播放的质量；
3. 移植libmad库到程序中，使之可以解码并播放MP3格式的音频文件。
4. 在项目中添加LCD模块来显示播放的状态，使其交互方式更加方便。

致谢

时光飞逝，随着毕业设计慢慢接近尾声，那看似漫长的四年大学时光也即将结束。回首曾经走过的青葱岁月，回望四年里给予过我帮助的人们，不禁感慨万千。

从论文选题到搜集资料，从项目设计到作品完成，从初写底稿到反复修改，期间经历了痛苦、喜悦、焦躁、激动、彷徨和兴奋，而如今，伴随着毕业设计作品的完成以及这篇毕业论文的最终成稿，复杂的心情烟消云散。在此次毕业设计完成的过程中，我首先要感谢我在公司的指导老师王鹏，感谢他能从忙碌的工作中抽出时间来指导我的毕业设计，其次要感谢我的论文指导老师覃桂敏，感谢她不辞辛劳的为我的论文提出指导意见，我还要感谢我的辅导员何良，感谢他那细致严谨、一丝不苟的工作作风，为了不让我们一个人掉队，可谓是用心良苦。我还要感谢在这四年里所有曾经带过我的老师们，感谢各位恩师的倾囊相授，你们的谆谆教诲我永记在心！

最后，我要感谢四年里陪伴在我身边的同学、朋友，感谢他们在我遇到困难时的雪中送炭，有了他们的支持、鼓励和帮助，我才能充实的度过了四年的学习生活，愿同窗之间的友谊地久天长！

参考文献

1. 苏菲. 数字音频原理与应用. 北京：电子工业出版社. 2002.
2. 刘火良，杨森. STM32库开发实战指南. 北京：机械工业出版社. 2013.
3. 李云. 嵌入式系统开发之道－－菜鸟成长日志与项目经理的私房菜. 人民邮电出版社. 2011.
4. 数字音乐发展历史及前景. http://danan.baijia.baidu.com/article/25434.
5. Dan Lavry. Sampling Theory For Digital Audio. Lavry Engineering, Inc. 2004.
6. 周立功. Keil C51使用技巧及实战. 广州：广州致远电子有限公司.
7. 周立功. 基于Stellaris系列ARM的PWM 语音播放器. 广州：广州致远电子有限公司. 2008.
8. STMicroelectronics group of companies. STM32F10xxx参考手册(参照RM0008 Reference Manual Dec. 2009 Rev 10翻译). 意法半导体(中国)投资有限公司. 2010.
9. FATfs文件系统指南. http://elm-chan.org/fsw/ff/00index\_e.html.
10. STMicroelectronics group of companies. STM32F10xxx Cortex-M3 programming manual Rev 3. STMicroelectronics. April 2010.
11. STMicroelectronics group of companies. How to use the high-density STM32F103xx microcontroller to play audio files with an external I2S audio codec. STMicroelectronics. May 2008.
12. 马志强，王建刚，孙少林等. 基于STM32的PWM音乐播放器应用设计. 单片机与嵌入式系统应用. 2012年，第12期.