基于 Nios 处理器的个人多媒体助手

Personal Multimedia Aides Based On Nios Processor

学生姓名：　周　鼎

指导教师：　杨　越

校（院）：中国地质大学（武汉） 机电学院

原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名：　周鼎　　2008 年 6 月 15 日

摘　　要

近年来，市场上的电子数码产品更新换代很快，从当初的分立元件到现在的集成芯片，产品的体积越来越小，而所展现出来的功能则日益强大。它们为我们的日常生活提供了极大的帮助。

SOPC(System on a programmable Chip,片上可编程系统)是 Altera 公司提出来的一种灵活、高效的 SOC 解决方案。它将处理器、存储器、I/O 口，LVDS、CDR 等系统设计所需要的模块集成到一个 PLD 器件上，具有可裁剪、可扩充、可升级，并具备软硬件在系统可编程等优点。

本文提出一种基于 NIOS 处理器的个人电子多媒体助手,系统是基于 Altera 公司的 Cyclone II EP2C35 开发板来进行设计的，利用板上的 SD 卡来作为存储设备，在 NIOS 处理器上实现 MP3 的播放、TXT 文件的读取以及 LCD 液晶显示等功能。

设计中将需要用到的外围设备的 IP 核添加到自己定制的 SOPC 控制系统中，生成具有强大处理性能的 Nios Ⅱ软核，再结合 Quartus Ⅱ EDA 工具，将其嵌入到 FPGA 芯片中，获得恰好满足需求的定制了 CPU 和外设的处理器。

在 Nios II 的 IDE 环境中进行软件部分的设计，实现 SD 卡中存储的 TXT 和 MP3 文件的读取，以及 LCD 显示、MP3 音乐的播放以及按键的控制。

运用 SOPC 设计理念来设计数码产品，能在较短的时间内完成开发过程，还能满足技术不断更新换代的需要。将当前先进的 NIOS 软核技术应用于电子电路设计当中去，对于了解和掌握这种工具有重要的意义。

关键词

SOPC　Nios II Soft core　LCD　MP3　SD

Abstract

In recent years, digital electronic products on the market renew very fast. from the original discrete components to the current integrated chips,these products showing us with much powerful functions while decreasing their sizes, They provides us a great help for daily life.

SOPC(System on a Programmable Chip)is a flexible and efficient SOC solution proposed by Altera Corporation. It put modules that are necessary like processor、memory、input/output interface、LVDS and CDR together into a PLD device. as a result, the system can be cut、expand、upgraded at our will, hardware and software are programmable in-system at the same time.

This paper presents a NIOS processor-based multimedia personal electronic assistant, the system is based on Altera's Cyclone II EP2C35 development board , using the SD card on the board as store equipment to achieve functions like MP3 playing、TXT file reading and LCD displaying.

Through adding the IP cores of the peripherals to their own customized SOPC control system, a Nios Ⅱ software core with high performance was generated. Combined with Quartus Ⅱ EDA tools, we can precisely meet the demand of the customized CPU and peripherals processors after embedded the core into the FPGA chips.

Next, the design of the the software part was completed in the Nios II IDE environment,and functions like reading the TXT and MP3 files stored in the SD card、LCD displaying、MP3 music playing and buttons controlling can be achieved.

. Through applying the SOPC design concept into the process of designing digital products, period of the development process can be shortened. needs of the upgrading technology can be meeted at the same time. It's very importannt to apply the advanced NIOS softcore into electronic circuit design, which can help us to understand and master this kind of technology.

key words

SOPC Nios II Soft core LCD MP3 SD

目　　录

第一章 绪论

§1.1 课题研究背景

随着时代的发展、科学技术的进步，各种多媒体设备越来越多的走进了人们的日常生活，其中包括掌上音频播放设备、迷你视频播放器等等，它们不仅体积小巧，而且功能强大。这些电子产品丰富了我们的日常生活，已经成为生活中不可或缺的一部分。

由于核心器件 Flash 存储器等的成本居高不下，造成一些电子产品卖价不菲，让不少消费者望而生畏；从开发商的角度来讲，因为目前各种技术更新换代很快，他们惟有不断的更新自身产品的软硬件，才能满足消费者的更高的需要。但是这样无疑就延长了开发周期，更升高了产品成本。如何在较短的时间内开发出满足要求的产品，是当前亟待解决的难题。

§1.2 系统的意义与本论文的主要内容

目前市场上的较高端的微处理器以 ARM、DSP、FPGA 为主，这三者的应用领域各有千秋。

ARM（Advanced RISC Machines），1991 年该公司成立于英国剑桥，主要出售芯片设计技术的授权。目前，采用 ARM 技术知识产权（IP）核的微处理器，即我们通常所说的 ARM 微处理器，已遍及工业控制、消费类电子产品、通信系统、网络系统、无线系统等各类产品市场，基于 ARM 技术的微处理器应用约占据了 32 位 RISC 微处理器 75％以上的市场份额；

DSP(Digital Signal Processor) 是一种独特的微处理器，是以数字信号来处理大量信息的器件。其工作原理是接收模拟信号，转换为 0 或 1 的数字信号，再对数字信号进行修改、删除、强化，并在其他系统芯片中把数字数据解译回模拟数据或实际环境格式。它不仅具有可编程性，而且其实时运行速度可达每秒数以千万条复杂指令程序，远远超过通用微处理器，是数字化电子世界中日益重要的电脑芯片。它的强大数据处理能力和高运行速度，是最值得称道的两大特色；

FPGA (Field Programmable Gate Array) 即现场可编程门阵列,它是在PAL、GAL、EPLD等可编程器件的基础上进一步发展的产物。它是作为专用集成电路（ASIC）领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。FPGA采用了逻辑单元阵列LCA（Logic Cell Array）这样一个新概念，内部包括可配置逻辑模块CLB(Configurable Logic Block)、输出输入模块IOB(Input Output Block)和内部连线(Interconnect)三个部分。

FPGA 的基本特点主要有：

1）采用 FPGA 设计 ASIC 电路，用户不需要投片生产，就能得到合用的芯片。

2）FPGA 可做其它全定制或半定制 ASIC 电路的中试样片。

3）FPGA 内部有丰富的触发器和 I / O 引脚。

4）FPGA 是 ASIC 电路中设计周期最短、开发费用最低、风险最小的器件之一。

5）FPGA 采用高速 CHMOS 工艺，功耗低，可以与 CMOS、TTL 电平兼容[1]。

ARM具有比较强的事务管理功能，可以用来跑界面以及应用程序等，其优势主要体现在控制方面。

而DSP主要是用来计算的，比如进行加密解密、调制解调等，优势是强大的数据处理能力和较高的运行速度。

FPGA 可以用 VHDL 或 verilogHDL 来编程，灵活性强，由于能够进行编程、除错、再编程和重复操作，因此可以充分地进行设计开发和验证。当电路有少量改动时，更能显示出 FPGA 的优势，其现场编程能力可以延长产品在市场上的寿命，而这种能力可以用来进行系统升级或除错。FPGA 目前的趋势是有代替前两者的可能，在 FPGA 内部置入乘法器和 DSP 块，就具有高速的 DSP 处理能力。在 FPGA 内置入硬核 CPU 或软核 CPU（Xilinx 有 powerpc 硬核的产品，有 microblaze 软核。Altera 有 NIOS II 软核）就可以成为既有能实现数字逻辑有适应嵌入式开发的综合性器件了。

可以说，FPGA 芯片是小批量系统提高系统集成度、可靠性的最佳选择之一。

本系统是基于 Altera 公司的 Cyclone II EP2C35 开发板来进行设计的，主要分为 CPU 模块(包括外围配置器件)、存储模块(SD 卡)、MP3 解码播放模块、TXT 文件读取以及 LCD 显示模块等几个部分。

设计中利用 Quartus II EDA 工具，将自己定制的 SOPC 系统生成具有强大处理性能的 Nios II 软核，并将其嵌入到 FPGA 芯片，最后在 Nios II 的 IDE 环境中进行软件部分的设计，成功地实现了 SD 卡中存储的 TXT 和 MP3 文件的读取、LCD 显示、MP3 音乐的播放以及按键的控制等。

第二章 系统构造及设计方案

§2.1 系统结构框图

本设计是基于 Altera 公司提供的 DE2 开发板开发的，其核心芯片为 Cyclone II EP2C35F672C6，整个系统分为四个部分：CPU 模块（包括外围配置芯片、扩展芯片等）、存储器（SD 卡）模块、LCD 显示模块、MP3 播放模块（包解码电路、DA 转换电路、功率放大电路等）。其组成框图如下：

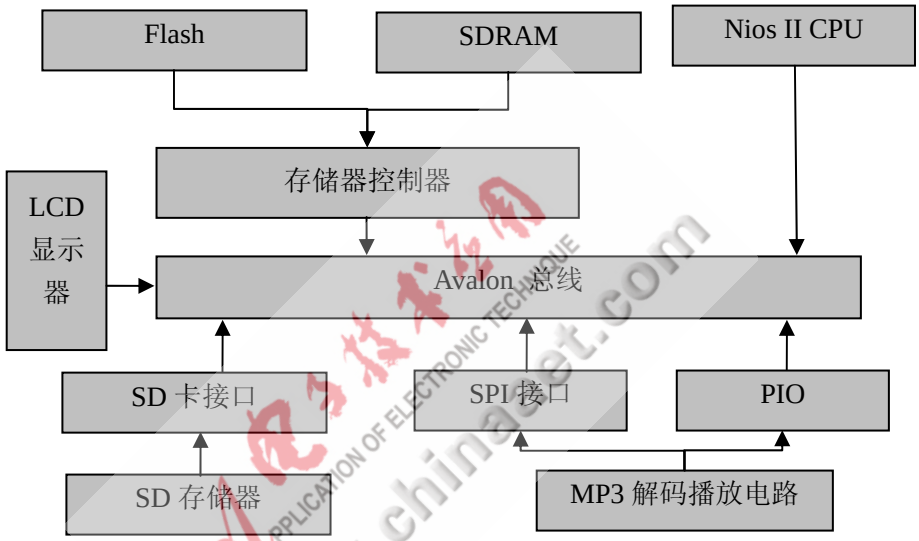


图 2-1 系统结构框图

§2.2 系统电路组成模块介绍

2.2.1 MP3 解码硬件电路方案设计及实现

1. 解码电路芯片选择以及与开发板连线

电路中的解码芯片选择的是 ST（意法半导体）公司的一款集成度较高、灵活性较强的 MP3G Layer III 解码芯片 STA013，。STA013 支持所有的 MPEG1、MPEG2 标准规范，还支持低采样频率的非标准的 MPEG2.5，且可以自动侦测到 MP3 的编码速率，支持数字音量、低音和高音控制。STA013 支持的信号采样率有 8、11.025、12、16、22.05、24、32、44.1 和 48KHz 等，且该解码芯片可以直接完成各种格式 MP3 数据流的解码操作，解码速度从 8kbit/s 到 320kbit/s,具有左右声道独立的音量控制，重低音、中音、高音均衡控制，有 14.31818MHz、14.7456MHz、10MHz 时钟频率可选，能够输出 PCM 或 US 格式的数字音频信号。

MP3 解码电路与开发板通过 6 个通用 PIO 口相连。其中 PIO_10 口和 PIO_11 口与 STA013

的 SDI、SCKR 连接，用来实现主机与从机 STA013 的 SPI 协议通讯，传输 MP3 数据；PIO_0 和 PIO_1 口分别与 STA013 中的 SCL、SDA 相连，实现与 STA013 的 I2C 通讯；PIO_2 和 PIO_3 分别与 STA013 的 DATA_REQ、RESET 引脚相连。MP3 解码电路通过 GPIO_0 与 FPGA 连接。MP3 解码电路原理图如下图所示：

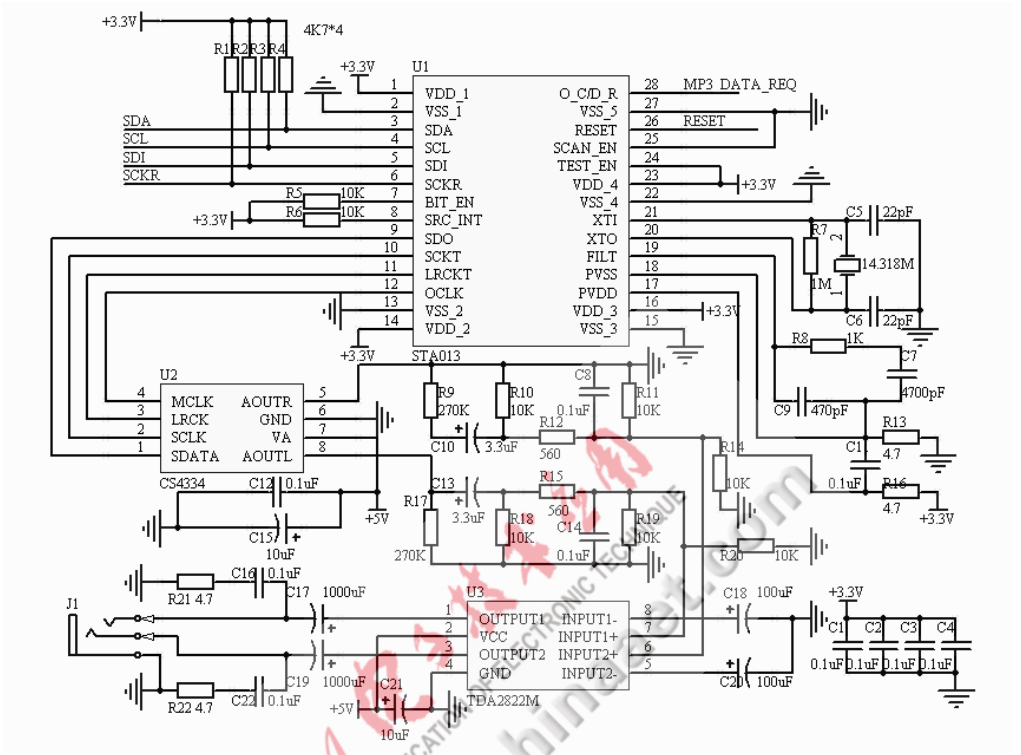


图 2-2　MP3 解码硬件电路图

STA013通过I²C接口接收输入数据，解码后的信号可以是立体声、单声道或者双声道的数字输出，可以通过PCM输出接口，直接送去D/A转换芯片处理。

这个输出接口可以软件编程，能兼容市场上的大部分通用的DAC芯片。其功能框图如图2-4所示：

D/A转换芯片采用的是24bit串行数模转换芯片CS4334，它支持的采样频率从2KHz～100KHz可变，能输出“录音线等级”（line-level)的高品质音频信号。CS4334包含了一个具有4倍内插(interpolation)和连续时间模拟输出的滤波器，有了它，就可以省略掉额外的外部放大器，以及复杂的输出滤波电路。

2. STA013的工作过程

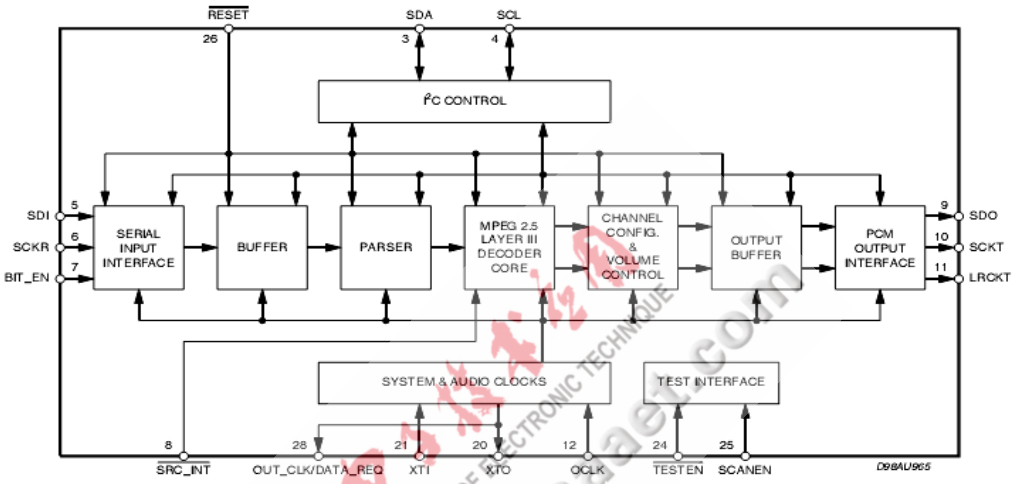
1)芯片初始化：检查STA013芯片是否存在；向STA013传送SST公司提供的“p02—0609．bin”配置文件。

2)传送MP3数据：传送MP3数据的基本思想就是在STA013需要数据的时候，给它传送。使用者不需要关心MP3的比特率问题，STA013会测定MP3的比特率，然后决定以合理的速度吸纳传送过来的数据，同时给出继续需要数据的信号。在传送数据的过程中，当STA013的

缓冲区将要满的时候，STA013停止给出继续需要数据的信号，对于易于变化的MP3比特流，STA013可以自动处理。它同时还可以自动探测MP3的采样频率(44.1、48KHz等)并合理调整DAC的时钟。所要做的就是以尽可能快的速度传送，只要它小于20Mbit／s。

3)解码：该过程由STA013的DSP核来进行，它先通过MP3文件头来识别歌曲的一些解码参数从而自动适应不同的MP3歌曲的解码,如通过识别信号的采样频率来自动调整其输出的时钟频率。这一切对用户来说都是透明的，可以随时查询这些解码参数。

4)输出数字音频信号：STA013解码后的数字音频信号由PIN9(SDO串行数据输出)、PIN10(SCKT串行时钟)、PINI1(LRCKT 左右声道时钟)、PIN12(OCLK采样时钟)4个引脚输出到D／A转换器CS4334。转换后由C54334的PIN5和PIN8输出模拟音频信号，经放大后，就可



以听到解码后的MP3音乐了。

图 2-3 MP3解码芯片STA013的功能框图

2.2.2 I²C 总线协议及应用

I²C总线支持任何IC生产过程（NMOS 、CMOS 、双极性）。两线——串行数据（SDA）和串行时钟（SCL）线在连接到总线的器件间传递信息。每个器件都有一个唯一的地址识别（无论是微控制器、LCD驱动器、存储器或键盘接口），而且都可以作为一个发送器或接收器（由器件的功能决定）。除了发送器和接收器外器件在执行数据传输时也可以被看作是主机或从机。主机是初始化总线的数据传输并产生允许传输的时钟信号的器件。此时任何被寻址的器件都被认为是从机，本设计中FPGA被作为主机，STA013被作为从机通讯。

在I²C 总线上产生时钟信号通常是主机器件的责任当在总线上传输数据时每个主机生自己的时钟信号。SDA 和SCL 都是双向线路，都通过一个电流源或上拉电阻连接到正的电源电压。 当总线空闲时，这两条线路都是高电平。连接到总线的器件输出级必须是漏极开路或集电极开路才能执行线与的功能。I²C 总线上数据的传输速率在标准模式下可达100kbit/s，在快速模式下可达400kbit/s，在高速模式下可达3.4Mbit/s。本设计中用到的是快速模式最快可达400kbit/s，完全可以达到STA013对MP3数据传输的要求。

另外，SDA 线上的数据必须在时钟的高电平周期保持稳定。数据线的高或低电平状态只有在SCL 线的时钟信号是低电平时才能改变，其时序见下图所示：

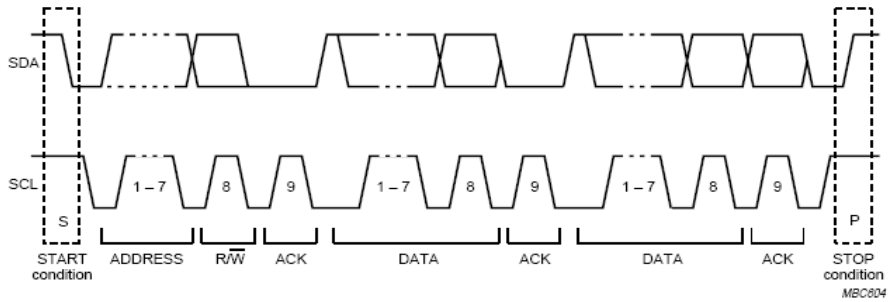


图 2-4 I²C总线的位传输时序图

输出到 SDA 线上的每个字节必须是 8 位，每次传输的字节不受限制，每个字节必须有一个应答为 ACK。如果一接收器件在完成其他功能（如一内部中断）前不能接收另一数据的完整字节时，它可以保持时钟线 SCL 为低，以促使发送器进入等待状态，当接收器械准备好接受数据的其它字节并释放时钟 SCL 后，数据传输继续进行。I²C 数据总线传送时序如图 2-5 所示。

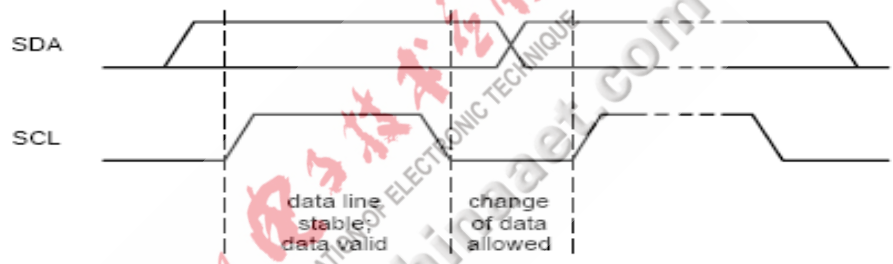


图 2-5 I²C 总线数据传输时序图

数据传送具有应答是必须的。与应答对应的时钟脉冲由主控器产生，发送器在应答期间必须下拉 SDA 线。当寻址的被控器件不能应答时，数据保持为高，接着主控器产生停止条件终止传输。在传输的过程中，当用到主控接收器的情况下，主控接收器必须发出一数据结束信号给被控发送器，被控发送器必须释放数据线，以允许主控器产生停止条件。

I²C 总线有如下四种基本操作：

1）开始条件：SDA由高到低跃变，SCL为高。表示一个通讯过程的开始或者停止，而不是在传送数据。

2）写字节，得到确认信息：此时SCL为低，FPGA传出8位数据，当第9个时钟到来的时候，FPGA收到一个来自STA013的确认信息。

3）读字节，发送确认信息：FPGA在SCL上传出8个时钟周期，在每个时钟周期上升沿，FPGA从STA013读出一位数据，在第9时钟上升沿，FPGA使SDA变低，表示已经成功读出STA013的信息。

4）停止条件：SDA的上升沿，此时，SCL为高。在结束时，I2C的两条线都保持高电平，这就是I2C总线的禁止状态。

开始和停止条件都由主控器产生。使用硬件接口可以很容易地检测开始和停止条件，没有这种接口的微机必须以每时钟周期至少两次对SDA取样以使检测这种变化。

2.2.3 SD卡简介及FAT16 文件系统构成介绍

图 2-6 SD 卡

SD卡（Secure Digital Memory Card）如图 2-6，是一种基于半导体快闪记忆器的新一代记忆设备。SD卡由日本松下、东芝及美国SanDisk公司于 1999 年 8 月共同开发研制[10]。

由于 SD 卡具有容量大、体积小、高性能、读/写速度快以及可与多种计算机操作系统平台兼容等优点，并且在 DE2 开发板上自带了 SD 卡接口，所以在本设计中，采用 SanDisk 公司的 256MB 的 SD 卡来存储数据文件和 MP3 文件，并且采用的是 SD 卡的 SPI 总线协议。

1）SD卡的SPI总线接口规范

SD 卡有两种总线访问方式：SPI 总线和 SD 总线。不同的总线访问方式其引脚功能定义不同。

表 2-1 SD 卡 SPI 模式的针脚定义

针脚	名称	类型	描述
1	CS	I	片选（负有效）
2	DI	I	数据输入
3	VSS	S	地
4	VCC	S	供电电压
5	CLK	I	时钟
6	VSS	S	地
7	DO	0	数据输出
8	RSV	--	
9	RSV	--	

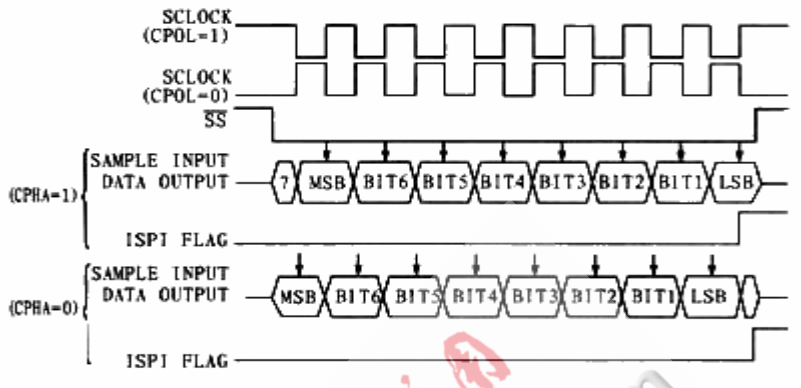
2）SPI协议

SPI，是英语 Serial Peripheral interface 的缩写，顾名思义就是串行外围设备接口。SPI 总线在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为 PCB 的布局上节省空间，提供方便，正是出于这种简单易用的特性，现在越来越多的芯片集成了这种通信协议。SPI 主要应用在 EEPROM，FLASH，实时时钟，AD 转换器，还有数字信号处理器和数字信号解码器之间，是一种高速的，全双工，同步的通信总线[11]。

跟 SPI 密切相关的两个概念是时钟极性和时钟相位。时钟极性：表示时钟信号在空闲时是高电平还是低电平。时钟相位：决定数据是在 SCK 的上升沿采样还是在 SCK 的结束沿采样。时钟极性和时钟相位相配合可以组合出不同的 SPI 总线时序，本设计中对 SD 卡的操作采用的是，时钟相位为正，即 SCK 上升沿采样。SPI 的读写时序见下图。

图 2-7　SPI 读写时序图

SPI 的通信原理比较简单，它以主从方式工作，这种模式通常有一个主设备和一个或



多个从设备，需要至少 4 根线，事实上 3 根也可以（单向传输时）。也是所有基于 SPI 的设备共有的，它们是 SDI（数据输入），SDO（数据输出），SCK（时钟），CS（片选），具体功能见下表。

表 2-2 SD 卡 SPI 模式下引脚功能

引脚名称	功能和作用
SDO	主设备数据输出，从设备数据输入
SDI	主设备数据输入，从设备数据输出
SCLK	时钟信号，由主设备产生
CS	从设备使能信号，由主设备控制

CS 为片选信号，剩下的 3 根线负责通讯。通讯是通过数据交换完成的，这里先要知道 SPI 是串行通讯协议，也就是说数据是一位一位的传输的。这就是 SCK 时钟线存在的原因，由 SCK 提供时钟脉冲，SDI，SDO 则基于此脉冲完成数据传输。数据输出通过 SDO 线，数据在时钟上升沿或下降沿时改变，在紧接着的下降沿或上升沿被读取。完成一位数据传输，输入也使用同样原理。这样，在至少 8 次时钟信号的改变（上沿和下沿为一次），就可以完成 8 位数据的传输。

需要注意的是，SCK 信号线只由主设备控制，从设备不能控制信号线。同样，在一个基于 SPI 的设备中，至少有一个主控设备。这样的传输方式有一个优点，与普通的串行通讯不同，普通的串行通讯一次连续传送至少 8 位数据，而 SPI 允许数据一位一位的传送，甚至允许暂停，因为 SCK 时钟线由主控设备控制，当没有时钟跳变时，从设备不采集或传送数据。也就是说，主设备通过对 SCK 时钟线的控制可以完成对通讯的控制。SPI 还是一个数据交换协议：因为 SPI 的数据输入和输出线独立，所以允许同时完成数据的输入和输出。不同的 SPI 设备的实现方式不尽相同，主要是数据改变和采集的时间不同，在时钟信

号上沿或下沿采集有不同定义。

SPI 接口的缺点：没有指定的流控制，没有应答机制确认是否接收到数据，数据传输速度较慢，最高只能达到：4M 比特每秒。

3）FAT16 文件系统构成

SD 卡与硬盘的结构类似，文件格式为 FAT16，所以在格式化时，系统采用了 FAT16 文件格式。下面结合所使用的 SD 卡介绍文件系统原理。

SD 卡逻辑结构如下表所示：

表 2-3 SD 卡逻辑结构

主引导区	引导区	FAT 表 1	FAT 表 2	目录区	数据区
32 个扇区	1 个扇区	按容量可变	按容量可变	32 个扇区	按容量可变，以簇为单位

主引导区位于整个 SD 卡的 0 柱面 0 磁头，1 扇区存放引导程序用于启动和引导操作系统，同时存放 SD 卡的主分区表，记录卡的分区信息。在总共 512 字节的主引导记录中，MBR 的引导程序占了其中的 446 字节（相对与扇区首地址的偏移量 0H～1BDH），随后的 64 字节（偏移量 1BEH～1FDH）为 DPT（Disk Partition Table,硬盘分区表），最后的两个字节“55 AA”（偏移量 1FEH～1FFH）是分区有效结束标志。

FAT16文件系统最多支持4个分区项，对应于4个分区表。在DPT部分共64字节中，以16字节为分区表项单位描述一个分区的属性。在本系统能够SD卡只作为一种存储介质使用，且整个系统作为一个分区处理。表2-4对这个分区的DPT数据进行了解释。

表 2-4 磁盘分区表

便移量	字段长度	值	字段名和值定义
0x01BE	BYTE	0x80	引导指示符，0x80 表示该分区为活动分区
0x01BF	BYTE	0x01	开始磁头
0x01C0	6 位	0x01	开始扇区，只用低 6 位，高两位被开始柱面字段使用
0x01C1	10 位	0x00	开始柱面
0x01C2	BYTE	0x04	系统 ID 定义了分区类型，0x04 表示为 FAT16
0x01C3	BYTE	0x07	结束磁头
0x01C4	6 位	0xE0	结束扇区，只用低 6 位，高两位被结束柱面字段使用
0x01C5	10 位	0xD2	结束柱面
0x01C6	DWORD	0x00000020	相对扇区数，从该磁盘的开始到该分区的开始位移量
0x01CA	DWORD	0x0003D2E0	总扇区数，该分区中的扇区总数

SD卡的引导扇区DBR位于文件系统开头，占用1扇区，在这512字节中，其实又是由跳转指令（0x00～0x02）、厂商标志和操作系统版本号（0x03～0x0A）、BPB（BIOS Parameter

Block）（0x0B～0x23）、扩展BPB（0x24～0x3D）、OS引导程序（0x3E～0x01FD）和结束标志几部分组成[12]。下表对BPB字段数据进行了分析：

表 2-5 BPB字段结构

偏移量	长度	名称和定义
0x0B	2	扇区字节数（Bytes Per Sector），本字段的值为 512
0x0D	1	每簇扇区数（Sectors Per Cluster）
0x0E	2	保留扇区数，第一个 FAT 开始之前的扇区数，包括引导扇区
0x10	1	FAT 书（Number of FAT），本字段的值一般为 2
0x11	2	根目录项数（Root Entries）
0x13	2	小扇区数（Small Sector），本字段的值为 0，而使用大扇区数来取代
0x15	1	媒体描述符（Media Descriptor），提供有关媒体被使用的信息
0x16	2	每 FAT 扇区数（Sectors Per FAT），该分区上每个 FAT 所占用的扇区数
0x18	2	每道扇区数（Sectors Per Track）
0x1A	2	磁头数（Number of Head）
0x1C	4	隐藏扇区数（Hidden Sector），该分区上引导扇区之前的扇区数
0x20	4	大扇区数（Large Sector）

FAT 表（File Allocation Table，文件分配表）是 Microsoft 在 FAT 文件系统中用于磁盘数据（文件）索引和定位引进的一种链式结构。簇（Cluster）是文件数据区被划分成的具有大小相等的区域用于磁盘文件的计量分配单位。簇的大小必须是 2 的幂，也就是由 2 的幂个扇区构成，对于特定磁盘其值是一定的。在 FAT 文件系统中，文件的存储依照 FAT 表制定的簇链式数据结构来进行。

FAT16 每个 FAT 项占两个字节，不同的表项值有不同的含义，下面是 FAT16 的表项值对应表。

表 2-6 FAT文件分配表

FAT16 表项值	0000H	0001～FFEFH	FFF0～FFF6H	FFF7H	FFF8～FFFFH
含义	未用的空簇	文件已使用的簇	系统保留簇	坏掉的簇	文件的最后一簇

FAT表和簇是一一对应的关系，对于FAT16的FAT每2个字节为一个FAT项，从0～N个FAT项分别对应0～N个簇，在我们对文件进行访问时，总是先访问文件的目录项，找到首簇簇号，再找到该簇号对应的FAT项，在其中找到下一簇的簇号，再在下一簇对应的FAT项中找到再下一簇的簇号，一直到在FAT项中找到有文件最后一簇的标志，我们对该文件的查找才结束这样就形成了一个链，我们把它称为磁盘簇链。

文件目录区在第2个FAT表后， FAT16的目录项由32个字节构成，各个字节的含义见下表：

表 2-7 FAT16 的目录项

偏移量	长度	定义				
0x0～0x7	8	文件名，其中 00H 为以下值时有些特定含义： 00H 表项为空表项 ， E5H 文件已被删除 ， 05H 实际该字节为的值为 E5H ， 08H~0AH 文件的扩展名				
0x8～0xA	3	文件扩展名				
0xB	1	属性字节	00000000	读/写	00001000	卷标
			00000001	只读	00010000	子目录
			00000010	隐藏	00100000	归档
			00000100	系统文件		
0xC～0x15	10	系统保留				
0x16～0x17	2	文件最近修改时间				
0x18～0x19	2	文件最近修改日期				
0x1A～0x1B	2	表示文件的首簇号				
0x1C～0x1F	4	表示文件的长度				

2.2.4 读SD卡（FAT16 文件系统）的操作

1. SD 卡的命令

SD 卡的命令有很多种，本设计中只用到少数几条，下面将本设计中用到的命令介绍如下：

1)：CMD0：复位和模式选择：

SD 卡被唤醒后进入 SD 模式，在接收复位命令的时候如果主机把 CS 线拉低，SD 卡就会进入 SPI 模式。

2)：CMD1：激活 SD 卡：

SD 卡被复位后处于 BUSY 状态，主机需要不断发 Active 命令给 SD 卡直到 SD 卡处于 IDLE 状态。

3)：CMD16：设置读写块长度命令

对 SD 读写必须以块为单位读写，在进行读写之前要通过此命令来设置块长度。

4)：CMD17：读单块命令

读给定 SD 卡地址下的一块内容，块的长度可以为 1 到 512 字节（包括 1 和 512 字节）

数据由 16 字节的 CRC 校验保护。

2. SD 卡的命令时序

SD卡命令总长6字节，包括：一字节的命令代号，四字节的命令参数和一字节的校验码，这种格式固定即使有些命令没有参数字节,主机向SD卡发送命令的时候也必须发送4字节的参数，这种情况下参数值可以为任何的的数值，SD卡会自动忽略它。其命令格式如下：

表 2-8 SD卡的命令格式

Byte1			Byte2-Byte5	Byte6	
7	6	5 0	31 0	7 1	0
0	1	Command	Command Argument	CRC	1

主机在向 SD 卡发送命令时，应先发高字节，在上述格式中命令码为最高字节，校验码为最低字节。主机依次向 SD 卡发完上述 6 字节后，SD 卡会响应主机的命令，响应消息根据不同的命令长度也不同，但第一个字节 R1 是公共的。R1 的格式和含义如下：

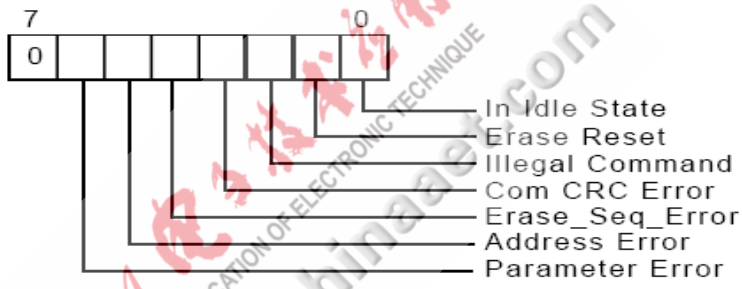


图 2-8 R1 字节的格式和含义

最高位固定为 0，其他位为错误标志位，为 0 表示没有错误，为 1 表示有相应的错误。由于最高位固定为 0，因此在接收 SD 卡的回应的时候，可以根据最高位是否为 0 来判断接收的的字节是否为合法的回应。

SD 卡的读写以块为单位,读操作的块长度可以在 1 到 512 字节间(包括 1 和 512 字节)，而写操作的块长度是固定的其值为 512 字节。读写操作又分为单块读写和多块连续读写。在本设计中主要涉及 SD 卡的读操作，即从 SD 卡中读取数据文件和 MP3 文件到 SDRAM 中，通过 SD 读卡器从 PC 机把数据文件和 MP3 文件下载到 SD 卡中。单块读命令其时序图如下：

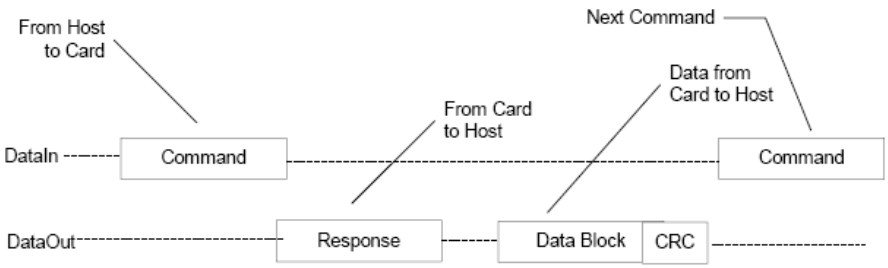


图 2-9 SD 卡单块读时序图

向 SD 卡发送读命令和块操作长度命令成功后，SD 卡会把一块数据发给主机，数据块由数据头，数据体和校验和组成（数据体的长度要先通过设置块长度命令设置），数据体前面有一个数据头标志数据的到达，数据头为 0xfe。如果检测到数据头就可以开始接收数据。数据块的最后两个字节为 16 位校验和。

在对 SD 卡进行操作的过程中，SD 卡的 CS 引脚要置为低，不对 SD 卡操作时，SD 卡的引脚要置高。

第三章 系统软件设计

§3.1 系统整体程序框图

系统的整体程序框图如下图所示，主函数在执行了 LCD 与 SD 卡初始化之后就一直在等待按键中断，接下来，在中断函数中进行按键值的判断，并执行相应的操作。

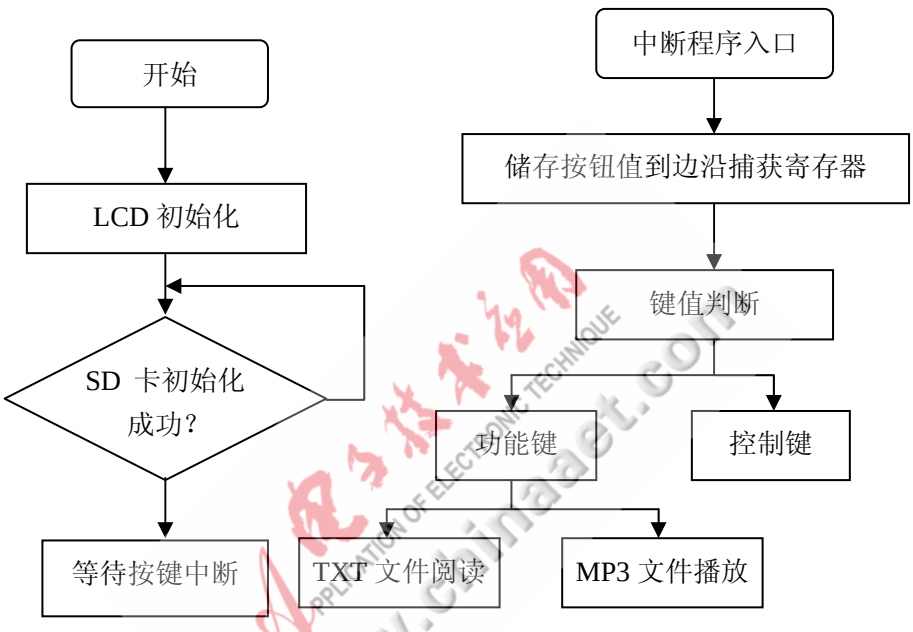


图 3-1 系统整体程序框图

§3.2 LCD16207 液晶模块显示程序

系统中的 LCD 液晶显示模块是 DE2 板上集成的，电路连接如图 3-1 所示，其中 LCD 上午 2 脚和背光引脚 15 分别连到 FPGA 的 IO 口上，其工作与否由编程来控制：

LCD16027 的读写时序要求比较严格，如图 5-1 所示，由于 Altera 公司事先做好了控制显示的 IP 核，只要将这个显示核添加进 Nios 系统当中后，调用 Nios 基本输出函数 IOWR（BASE，OFFSET，DATA），就可以自行按照时序发送和接收数据了。

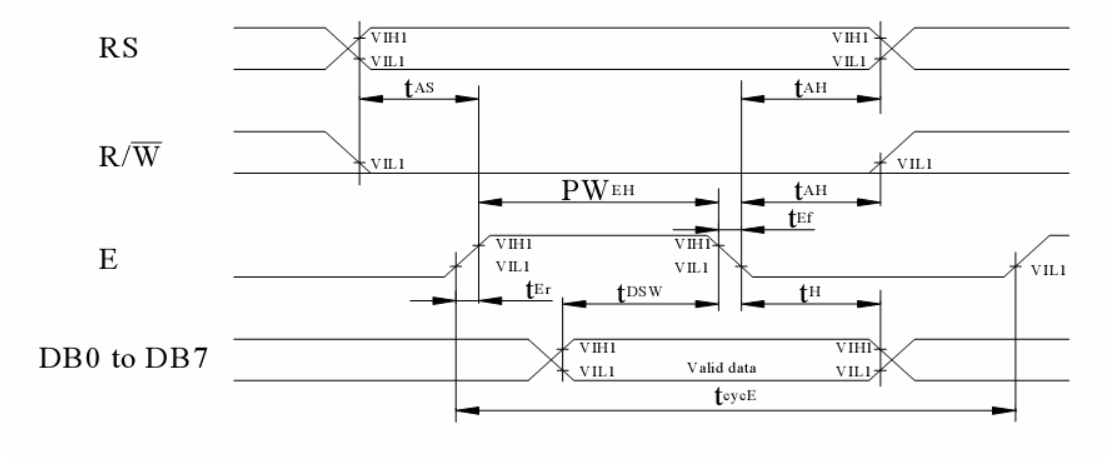


图 3-1　LCD16207 的写时序

§3.3 读取 SD 卡软件设计

该部分的程序主要负责把 MP3 文件从 SD 卡中读取到 SDRAM 中，供 STA013 解码用。系统设计了一个小型的 FAT16 文件系统来实现 SD 卡的读取，该文件系统不支持长文件名；不支持 FAT32 文件格式；可支持对 SD 卡进行写操作，SD 卡里面的 MP3 文件可通过 SD 卡读卡器从 PC 机上复制,附录程序 2 给出了获取磁盘参数子程序 GetDiskPara()。

在文件系统初始化开始，首先调用InitSDToSpi()函数把SD卡配置为SPI模式和判断SD卡是否存在，若存在则激活SD卡，读取FAT文件系统的一些基本信息，获取磁盘参数，如根目录开始扇区以及数据区开始扇区等。通过调用fFind(int1 *pname)来查找需要读取的文件是否存在，确定存在后用指针为文件分配一个缓冲区，由于SDRAM空间足够大，因此可以把文件数据全部读取到SDRAM中。读取时每次读取一个扇区，直到把数据全部读取到SDRAM中。详细流程图见图3-3所示：

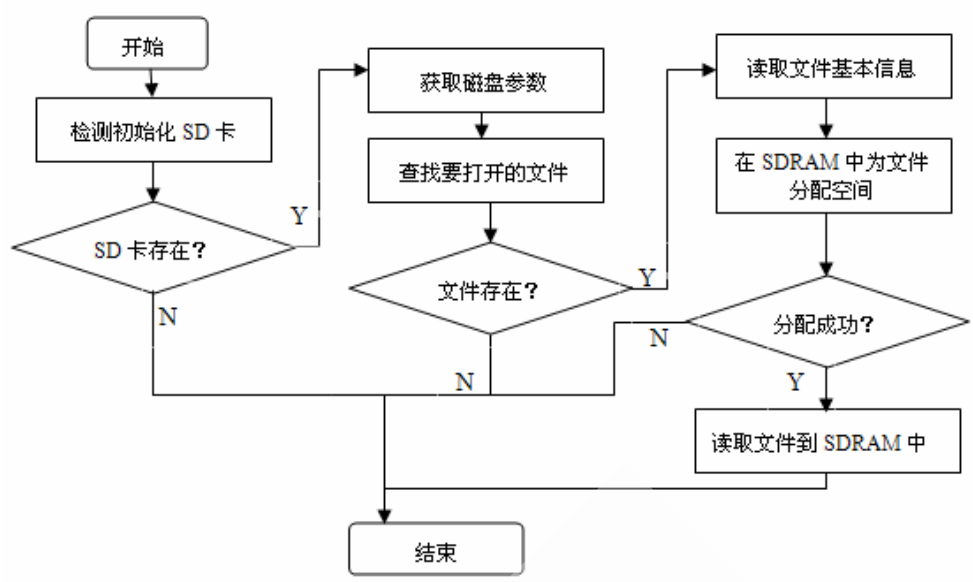


图 3-3 读取SD卡程序流程图

§3.4 播放MP3 音乐的程序设计

这部分程序的主要任务是控制 STA013 对数据进行解码以及 D/A 转换芯片工作，从而实现播放 MP3 音乐的功能。

这部分程序了比较“安全”的硬件解码方案。程序开始时，先初始化 I^2C 接口，然后调用 STA013_Init（）函数完成 STA013 的初始化，初始化包括复位 STA013、识别 STA013、写入配置文件等操作。配置文件放在 STA013_UpdateData 数组中。随后就是配置 STA013 和设置音调、音量、准备解压。接下来调用解码控制函数 sta013_SendToDecode（）进行解码，当 STA013 的 DATA_REQ 引脚为 1 时，表明 STA013 需要新的 MP3 数据进行解压播放，此时连续往 STA013 中发送数据，直到 STA013 的缓冲区满停止，到下一个数据请求信号继续发送。STA013 接收到数据后开始解码播放。其流程图如图 3-4 所示：

§3.5 通过I^2C总线读写STA013

通过对I^2C四种基本操作的认识，可以理解通过I^2C总线读写STA013数据流程图如图3-5所示，通知准备写 / 读时即发送一字节，值为0x86 / 0x87，前面最重要的7位表示STA013准备接收，因为总线上可能还有其他设备，最低位清零表示STA013将要写数据、置1表示将要读出下一地址内容,附录中列出了sta013MP3文件解码程序sta013StartDecoder()。

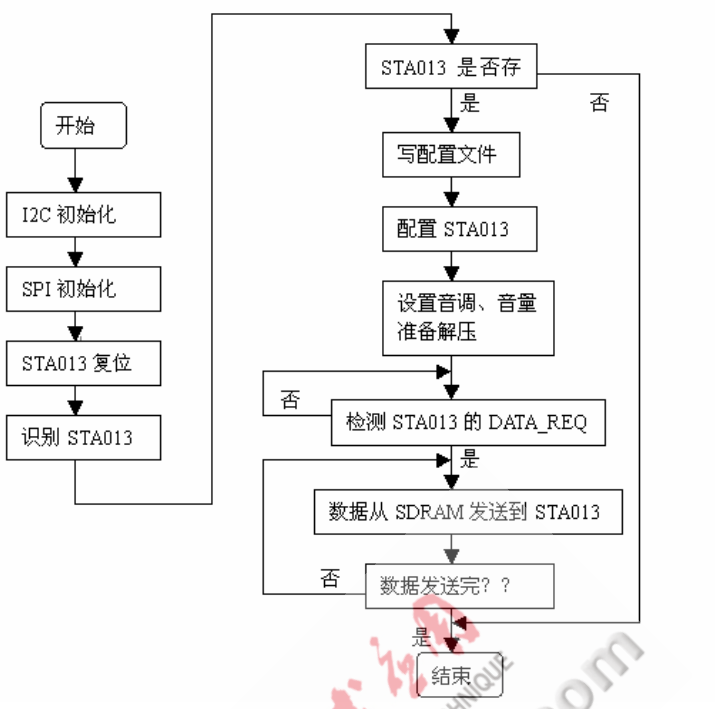


图 3-4 播放MP3音乐程序流程图

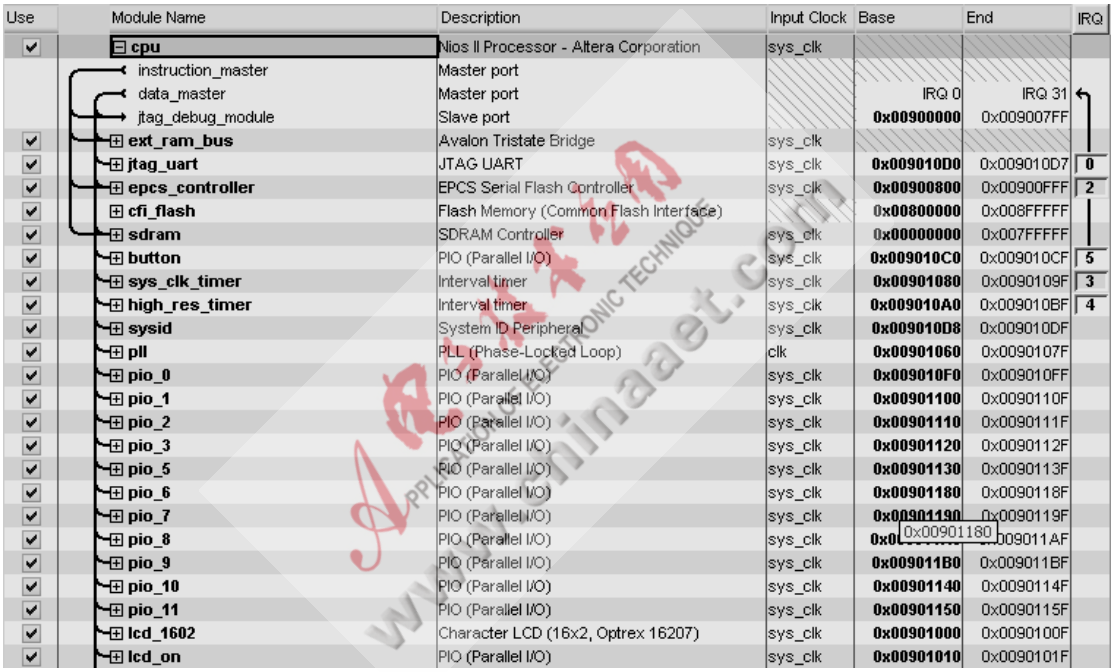
图 3-5 I²C 总线读（右）和写（左）STA013 程序流程图

第四章 系统已经实现的功能介绍及总结

§4.1 系统实现的功能介绍

4.1.1 SOPC Builder 组合

如下图所示，在 Quartus II 的 SOPC Builder 中将 Nios II processor 和外围器件的 IP 核进行进行组合



Use	Module Name	Description	Input Clock	Base	End	IRQ
☑	cpu	Nios II Processor - Altera Corporation	sys_clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		0x00900000	0x009007FF	
☑	ext_ram_bus	Avalon Tristate Bridge	sys_clk			
☑	jtag_uart	JTAG UART	sys_clk	0x009010D0	0x009010D7	0
☑	epcs_controller	EPCS Serial Flash Controller	sys_clk	0x00900800	0x00900FFF	2
☑	cfi_flash	Flash Memory (Common Flash Interface)		0x00800000	0x008FFFFF	
☑	sdram	SDRAM Controller	sys_clk	0x00000000	0x007FFFFF	
☑	button	PIO (Parallel I/O)	sys_clk	0x009010C0	0x009010CF	5
☑	sys_clk_timer	Interval timer	sys_clk	0x00901080	0x0090109F	3
☑	high_res_timer	Interval timer	sys_clk	0x009010A0	0x009010BF	4
☑	sysid	System ID Peripheral	sys_clk	0x009010D8	0x009010DF	
☑	pll	PLL (Phase-Locked Loop)	clk	0x00901060	0x0090107F	
☑	pio_0	PIO (Parallel I/O)	sys_clk	0x009010F0	0x009010FF	
☑	pio_1	PIO (Parallel I/O)	sys_clk	0x00901100	0x0090110F	
☑	pio_2	PIO (Parallel I/O)	sys_clk	0x00901110	0x0090111F	
☑	pio_3	PIO (Parallel I/O)	sys_clk	0x00901120	0x0090112F	
☑	pio_5	PIO (Parallel I/O)	sys_clk	0x00901130	0x0090113F	
☑	pio_6	PIO (Parallel I/O)	sys_clk	0x00901180	0x0090118F	
☑	pio_7	PIO (Parallel I/O)	sys_clk	0x00901190	0x0090119F	
☑	pio_8	PIO (Parallel I/O)	sys_clk	0x0	009011AF	
☑	pio_9	PIO (Parallel I/O)	sys_clk	0x009011B0	0x009011BF	
☑	pio_10	PIO (Parallel I/O)	sys_clk	0x00901140	0x0090114F	
☑	pio_11	PIO (Parallel I/O)	sys_clk	0x00901150	0x0090115F	
☑	lcd_1602	Character LCD (16x2, Optrex 16207)	sys_clk	0x00901000	0x0090100F	
☑	lcd_on	PIO (Parallel I/O)	sys_clk	0x00901010	0x0090101F	

图 4-1 系统的 SOPC Builder 综合图

4.1.2 硬件系统编译

在 SOPC Builder 中把系统的外围组件配置完成之后，用 Quartus5.1 进行各个引脚的分配，最后进行编译，编译完成如 4-2 所示

Quartus II Version	5.1 Build 176 10/26/2005 SJ Full Version
Revision Name	led_project
Top-level Entity Name	led_project
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Preliminary
Met timing requirements	Yes
Total logic elements	4,090 / 33,216 (12 %)
Total registers	2662
Total pins	124 / 475 (26 %)
Total virtual pins	0
Total memory bits	50,688 / 483,840 (10 %)
Embedded Multiplier 9-bit elements	5 / 70 (7 %)
Total PLLs	1 / 4 (25 %)

图 4-2 系统 Quartus 编译结果图

4.1.3 下载运行程序

编译通过以后，生成的工程文件配置信息以.sof 和.pof 为后缀。下载调试时把.sof 配置信息文件通过 Quartus5.1 专有工具 JTAG 口下载到 Cyclone II EP2C35 板上进行测试，NIOS II IDE 开发环境中生成的应用程序同时也下载到外部的 SDRAM 中执行。

经以上实现步骤后，本设计很好的实现了 TXT 文件的读取和 MP3 音乐的流畅播放。

4.1.4 TXT文件读取及显示

系统上电运行后，能正确的找出 SD 卡中存储的 TXT 文件的数目，并在 LCD 显示屏上显示出文件名，TXT 文件的阅读功能由板上的按键来控制，可以执行选择文件、上下翻页等操作，到达文件末尾时显示“end of the file!”,见附录图 3、4 所示。

4.1.5 MP3 文件读取及播放

程序下载运行后，系统能正确的初始化 SD 卡以及解码芯片 STA013，找到存储的 MP3 文件，并在 LCD 显示屏上显示出要播放的文件名、总文件数、当前文件序号，MP3 播放由板上的按键来控制，可以执行选择文件、上下翻页等操作。MP3 播放流畅，外接扬声器声音效果理想，见附录图 1、2。

§4.2 结论及建议

结合 Altera 公司的 DE2 开发板，通过在 SOPC Builder 中搭建系统，并在 Nios II 编程

环境中完成程序的调试与下载，本设计较好的完成了 TXT 文档的读取显示以及 MP3 文件播放功能。

MP3 部分的控制还不够完善，仍需做大量的工作来进一步的完善和对整个系统综合优化。其改进建议如下：

1. 系统中采用的 LCD 显示屏系板上集成，显示字符数量较少，且不能显示汉字，可以通过外扩较大的显示设备来弥补这一不足，进一步完善其阅读功能。

2. 目前系统是将文件完整读出后再进行操作，中间过程相对而言较慢，可通过中断来解决，在 STA013 的 Data_REQ 有电平中断到来时，转中断服务程序发送 MP3 数据到 STA013 解码，主程序中可一直读 SD 卡中的 MP3 文件到缓存中，这样可以节省读 MP3 文件的时间，使播放上下两首歌间的停顿时间减小。

3. 进一步完善 MP3 播放的控制程序，例如播放、暂停等等，还可以加上液晶显示歌名、显示播放状态、歌词同步显示等等，播放器的功能更加完美。

参考文献

[1] 叶淦华.FPGA 嵌入式应用系统开发典型实例.中国电力出版社
[2] 董代洁、郭怀理等编著.基于 FPGA 的可编程 SOC 设计.北京航空航天大学出版社
[3]李兰英等编著.Nios II 嵌入式软核 SOPC 设计原理及应用.北京航空航天大学出版社
[4]杨春玲、张辉主编.现代可编程逻辑器件及 SOPC 应用设计.哈尔滨工业大学出版社
[5]湖北省大学生电子设计竞赛——SOPC 专题竞赛培训教材.2006 年
[6]王诚.Altera FPGA/CPLD 设计（初级篇）.人民邮电出版社
[7]Quartus II Handbook . Version 6.1 . http://www.altera.com
[8]Nios II Software Developer’s Handbook .Version 5.1 . http://www.altera.com
[9] 彭澄廉主编.挑战 SOC——基于 NIOS 的 SOPC 设计与实践.清华大学出版社
[10]SanDisk Secure Digital Card Product Manual . Version 2.2 . www.sandisk.com
[11]SD Memory Card Specifications . Version 1.0.1 . SanDisk Corporation
[12]硬盘 FAT 文件系统原理的详细分析 . 存储论坛网友/sjhf 2004-04-02
[13]王金明编著.数字系统设计与 verilog HDL.第二版.电子工业出版社
[14]verilog HDL 数字设计与综合.第二版.电子工业出版社
[15]郭书军、王玉花、葛纫秋.嵌入式处理器原理及应用——Nios 系统设计和 C 语言编程.清华大学出版社
[16]王诚. Altera FPGA/CPLD 设计（高级篇）.人民邮电出版社
[17]I2C 总线规范 .版本 2.1-2000 . http://www.zlgmcu.com
[18]周立功等著.SOPC 嵌入式系统教程.北京航空航天大学出版社
[19]郭书军等编.嵌入式处理器原理及应用——Nios 系统设计和 C 语言编程.清华大学出版社
[20]汪国强编著.SOPC 技术与应用.机械工业出版社

附录

图 1 MP3 工作图片

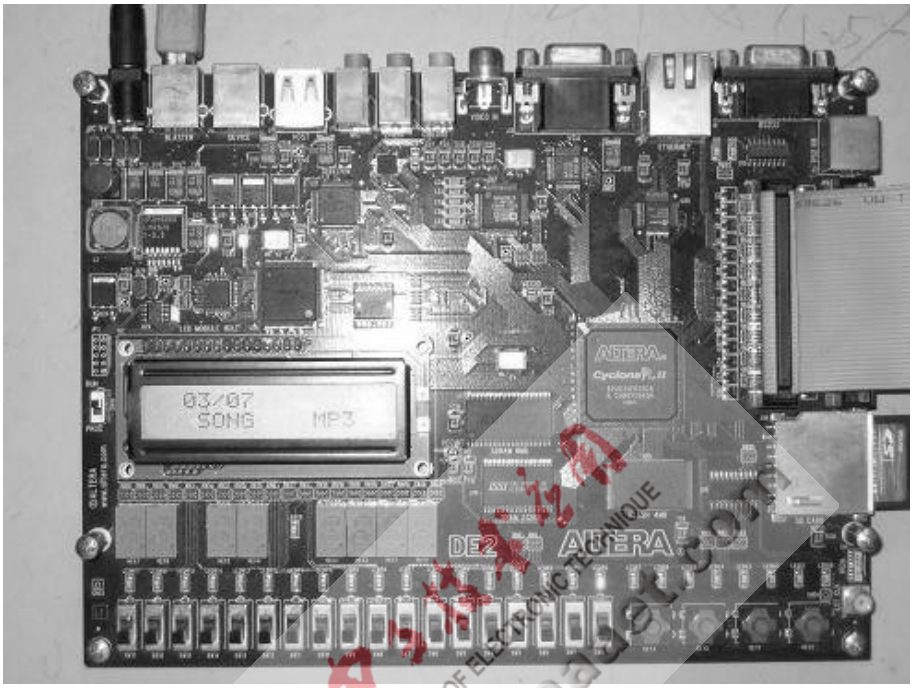


图 2 TXT 文件阅读及显示

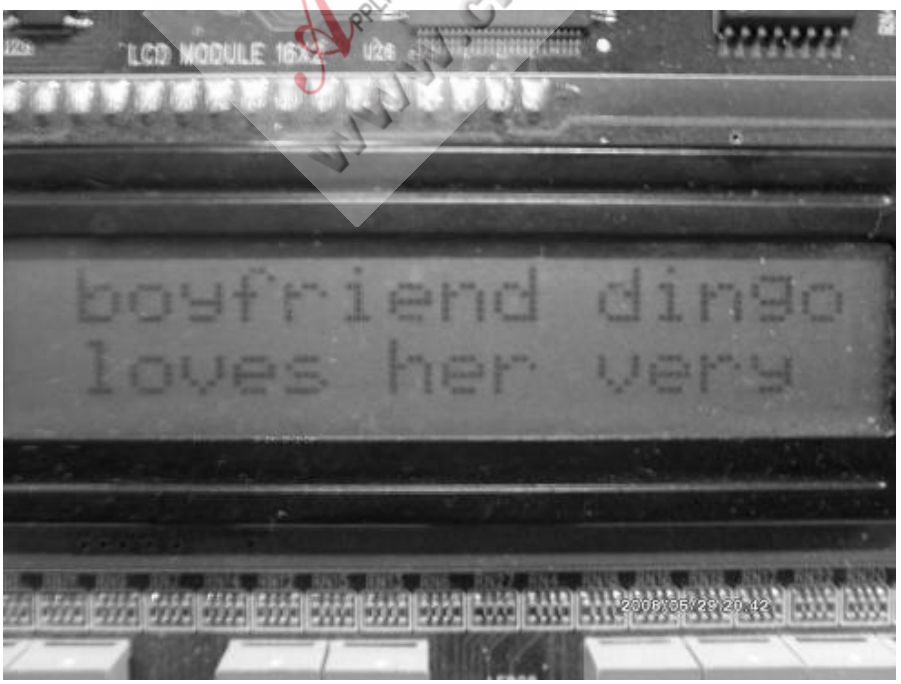
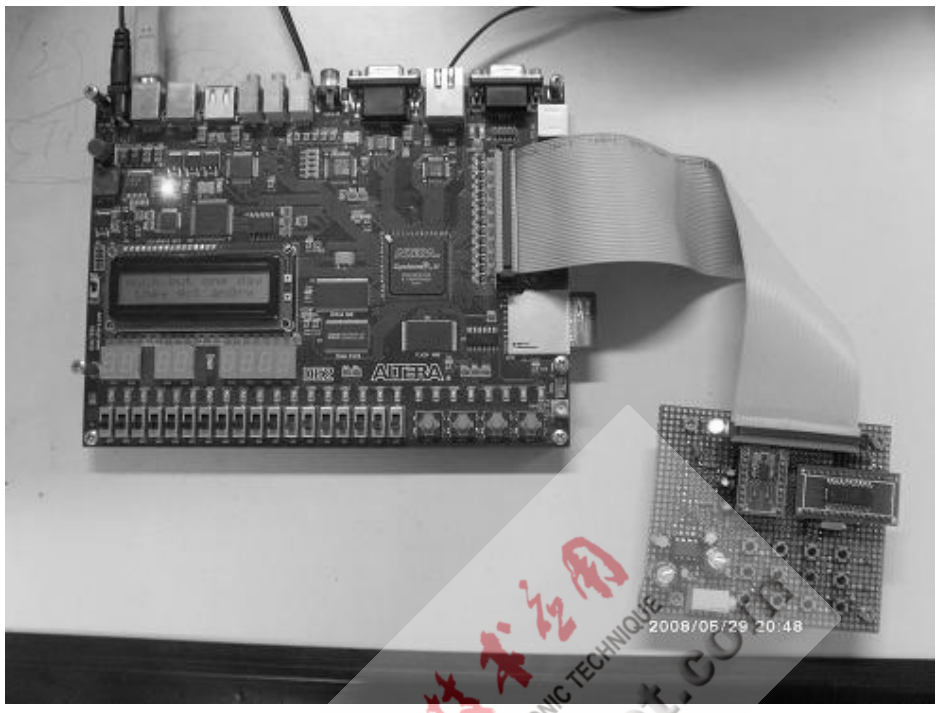


图 3 系统整体图



程序部分

1：Nios 键盘中断处理部分 C 语言代码

```
volatile    int    edge_capture;

static    void    handle_button_interrupts(void *context,alt_u32 id)
 {
    volatile int* edge_capture_ptr = (volatile int*)context;
    /*储存按钮的值到边沿捕获寄存器中.*/
    *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(PIO_0_BASE);

    usleep(30000);
    usleep(30000);            /*延时，去键盘抖动*/

    *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(PIO_0_BASE);

    /*  复位边沿捕获寄存器  */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PIO_0_BASE,0);
```

```
}

/* 初始化 PIO_0,即开发板上的四个按钮*/

static void init_PIO_0()
{
  void* edge_capture_ptr = (void*)&edge_capture;
  /*开放全部 4 个按钮的中断.*/
  IOWR_ALTERA_AVALON_PIO_IRQ_MASK(PIO_0_BASE,0xf);
  /*复位边沿捕获寄存器*/
  IOWR_ALTERA_AVALON_PIO_EDGE_CAP(PIO_0_BASE,0x0);
  /*登记中断源*/
  alt_irq_register(PIO_0_IRQ,edge_capture_ptr,handle_button_interrupts);

}
```

2：获取磁盘参数子程序 GetDiskPara()

```
char GetDiskPara()
{
  uint1 datbuf[512];
  uint2 temp,temp1,i;
  uint4 fatsize;
  if(RDBlock(0,datbuf,512) != NO_ERROR)
  {
    printf("read 0 addr error\r\n");
    return ERROR;
  }
  if(datbuf[0] == 0xeb || datbuf[0] == 0xe9)
  {
    Disk_Para.BootSecAddr = 0;
  }
  else
  {
    ConvertChar2ToInt2(&temp,&datbuf[0x1be + 0x8]);      //读取启动扇区的开始扇区号存放在 08H~0BH 的 4 字节 由高到低排列
    ConvertChar4ToInt4(&(Disk_Para.DiskSize),&datbuf[0x1be + 0xc]);      //读取磁盘的
```

```
容量   存放在 0CH~0FH 的 4 字节  由高到低排列
      Disk_Para.BootSecAddr = temp;
   }
   if(RDBlock(Disk_Para.BootSecAddr << 9,datbuf,512) != NO_ERROR)
   {
      printf("read boot sector addr error\r\n");
      return ERROR;
   }
   Disk_Para.BytesPerSec = datbuf[12];                          //读取每扇区的字节数   高字节
   temp =    datbuf[11];
   Disk_Para.BytesPerSec = (Disk_Para.BytesPerSec << 8) | temp;
   Disk_Para.SecPerClu = datbuf[13];                                    //读取每簇的扇区数
   ConvertChar2ToInt2(&temp,&datbuf[17]);//temp 为根目录项数
   Disk_Para.RootDirSize =    ((temp<<5) + (Disk_Para.BytesPerSec-1))/Disk_Para.BytesPerSec;
   //计算根目录的大小（以扇区为单位）
   ConvertChar2ToInt2(&temp,&datbuf[22]);
   fatsize =    datbuf[16]*temp;     //datbuf[16]fat 表的分数
   Disk_Para.FatSize = fatsize;   //fat 表大小
   ConvertChar2ToInt2(&temp,&datbuf[14]);     //保留区中保留扇区数目
   Disk_Para.FatAddr = Disk_Para.BootSecAddr +    temp;                 //fat 表开始扇区号
   Disk_Para.RootDirAddr = Disk_Para.FatAddr + Disk_Para.FatSize;   //根目录开始扇区号
   Disk_Para.DataSecAddr = temp + Disk_Para.RootDirSize + fatsize;//计算数据区开始扇区号
   Disk_Para.DataSecAddr += Disk_Para.BootSecAddr ;
   ConvertChar2ToInt2(&temp,&datbuf[19]);
   if(temp != 0) {Disk_Para.DiskSize = temp;}
   else
   {
    ConvertChar4ToInt4(&(Disk_Para.DiskSize),&datbuf[32]);          // 读取磁盘的容量
0CH~0FH 的 4 字节  由高到低排列

   }
   Disk_Para.DiskSize <<=9;
   if(datbuf[17] == 0) Disk_Para.FatType = 16;
   switch(Disk_Para.SecPerClu)
   {
      case 1:
         Disk_Para.SecPerClu_2n = 0;
```

```
            break;
        case 2:
            Disk_Para.SecPerClu_2n = 1;
            break;
        case 8:
            Disk_Para.SecPerClu_2n = 3;
            break;
        case 16:
            Disk_Para.SecPerClu_2n = 4;
            break;
        case 32:
            Disk_Para.SecPerClu_2n = 5;
            break;
        case 64:
            Disk_Para.SecPerClu_2n = 6;
            break;
        case 128:
            Disk_Para.SecPerClu_2n = 7;
            break;
    }

    return NO_ERROR;
}
```

3：sta013MP3 文件解码程序 sta013StartDecoder()

```
void sta013StartDecoder(void)
{

    //软件复位
    sta013WriteReg(STA_REG_SOFT_RESET, 0x01);
    sta013WriteReg(STA_REG_SOFT_RESET, 0x00);

    //静音与配置 DAC 输出
    sta013WriteReg(STA_REG_MUTE,          0x01);
    sta013WriteReg(STA_REG_PCMDIVIDER,    0x01);   // 32-bit mode, O_FAC = 256
    sta013WriteReg(STA_REG_PCMCONF,       0x21);   //24-bit mode & more
```

```
  //配置 PLL    14.7456      256
  sta013WriteReg(STA_REG_PLLFRAC_441_H,      0x04);
  sta013WriteReg(STA_REG_PLLFRAC_441_L,      0x00);
  sta013WriteReg(STA_REG_PLLFRAC_H,          0x55);
  sta013WriteReg(STA_REG_PLLFRAC_L,          0x55);
  sta013WriteReg(STA_REG_MFSDF_441,          0x10);
  sta013WriteReg(STA_REG_MFSDF,              0x0f);

  //配置界面等
  sta013WriteReg(STA_REG_DATA_REQ_ENABLE, 0x04); //使能 data request  信号
  sta013WriteReg(STA_REG_PLLCTL_2,           0x0c);
  sta013WriteReg(STA_REG_PLLCTL_3,           0x00);
  sta013WriteReg(STA_REG_PLLCTL_1,           0xa1);
  sta013WriteReg(STA_REG_SCLK_POL,           0x04);     //  上升沿进行数据采样,下降沿
发送数据
  sta013WriteReg(STA_REG_REQ_POL,            0x01);     // REQ line active high

  //设置音调
 sta_SetTone(0,0,0,8000);///////////////
 //STA013 开始运行
 sta013WriteReg(STA_REG_RUN,    0x01);
 sta013WriteReg(STA_REG_PLAY, 0x01);
 sta013WriteReg(STA_REG_MUTE, 0x00);
 }
```