Design and Implementation of Embedded Audio System Based on Audio Codec '97

YU Shao-feng, YAN Ju-ming, HU Chen
(*National Engineering Research Center for Application Specific Integrated Circuit System Southeast University, Nanjing* 210096, *China*)

Abstract AC97(Audio Codec '97) is an industry standard of audio device brought forward by Intel Corporation. This paper describes an approach about implementation of AC97 audio system based upon embedded system. The basic frame about AC97 audio system is firstly introduced. Then this paper discusses the concrete implementation of AC97 audio system in Intel XScale PXA255 system from analog and digital aspects. Finally we describe the device driver based on the embedded Linux system, which focuses on the implementation of the audio buffer.

Key words AC97; audio; CODEC; Intel XScale PXA255; embedded linux
EEACC: 6150M

基于 AC97标准的嵌入式音频系统设计与实现

於少峰,严菊明,胡 晨
(东南大学国家专用集成电路系统工程技术研究中心,南京 210096)

摘 要: AC97是 Intel公司提出的针对音频设备的业界标准。本文介绍了其在嵌入式系统中的一种实现方案。首先介绍了 AC97音频系统的基本框架,然后从模拟和数字两个方面讲述了基于 Intel XScale PXA255的音频系统的实现,以及基于嵌入式 Linux的软件驱动的实现。其中重点讲述了驱动中音频缓冲区的实现。

关键词: AC97;音频; CODEC; Intel XScale PXA255;嵌入式 Linux
中图分类号: TP391 文献标识码: A 文章编号: 1005-9490(2004)04-0733-04

随着移动终端设备在人们的生活中不断普及,人们对于移动终端的要求已不限于通话和文字处理等简单的应用了。特别是对于 Smart Phone, PDA等多功能的手持设备,人们追求越来越好的画面和音乐效果。于是在手持终端中提供优秀的音乐效果是当前手持终端的一个发展趋势。

AC(Audio Codec)97是 Intel公司在 1997年推出的专门针对音频设备的业界标准与设计规范。它把一般音频设备中的数字部分(DSP)和模拟部分(CODEC)分离开来,从而降低了电磁干扰,获得较好的音效品质[1]。目前 AC97标准的音频设备在 PC上被广泛的采用。随着手持设备的功能越来越强大,AC97标准也逐渐步入嵌入式系统中。

嵌入式系统目前有好多种,如 WinCE, vxworks等,但他们要收取不菲的权利金。而嵌入式 Linux是一个完全开放的免费的操作系统。它支持多种硬件体系结构,内核运行高效稳定,而且源代码开放,有着完善的开发工具,为开发人员提供了不逊于商用系统的开发环境。本文针对 Intel公司的 XScale PXA255处理器构造了基于 AC97

收稿日期: 2004-07-23
作者简介: 於少峰(1981-),男,东南大学国家专用集成电路系统工程技术研究中心研究生,研究方向为嵌入式系统开发, Nand Flash应用, iroifirst@hotmail.com

标准的音频系统,并介绍了该音频系统基于 Linux2.4.19内核的驱动程序实现。

1 AC97音频系统介绍

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(Coder-Decoder:编码/解码器)完成。AC97音频系统的总体结构如图 1所示。

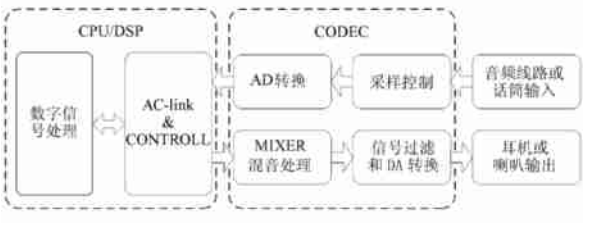

图 1 AC97音频系统总体框架

音频处理的流程如下:

① CODEC采样音频数据,把模拟信号转换成数字信号通过 AC-link和 AC控制单元传送给 CPU或 DSP处理。

② CPU对数字信号处理后交由 AC控制单元通过 AC-link传输给 CODEC进行混音处理,再转换成模拟信号输出。

其中 AC控制单元是 AC97 CODEC的控制器。AC-link是 AC97标准定义的全双工的串行接口,负责传输音频数据、寄存器控制命令和状态信息。详见 3.1

2 硬件框架实现

由图 1可知,AC97音频系统有两种应用方案,主要区别就是是否采用专门的 DSP芯片。目前嵌入式处理器的能力越来越强大,CPU已经可以代替 DSP处理数字信号,所以 AC97音频系统的硬件实现主要就是 CODEC的连接和控制,即模拟部分的实现。

2.1 CODEC连接的实现

本系统采用的 CODEC芯片是 Philips公司出品的一款符合 AC97标准的多功能 CODEC芯片 UCB1400。它不仅是一枚 CODEC芯片,还集成了触摸和电池管理两个功能模块[3],在嵌入式系统中应用广泛。

Intel公司的 XScale PXA255是一款基于 ARM 5TE内核技术的嵌入式处理器[2]。它提供了符合 AC97 rev2.0标准的 AC97控制单元(ACUNIT)和音频控制连接(AC-link)。ACUNIT就是 CODEC控制器,它通过 AC-link连接和控制 CODEC芯片 UCB1400。如图 2所示:

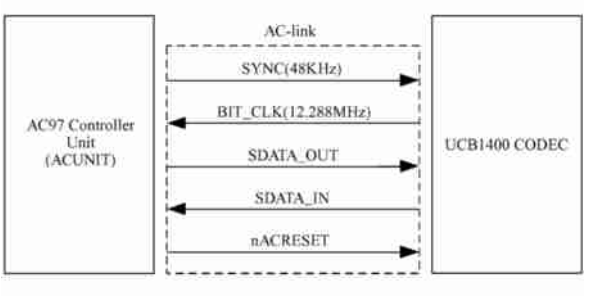

图 2 AC97控制单元和 UCB1400的连接

AC-link由 4根串行总线和一个 nRESET信号线组成。串行总线分别是:位时钟信号线(BIT__CLK),帧同步信号线(SYNC),数据输出(SDATA__OUT)和数据输入(SDATA__IN)信号线。它按照 AC97 rev2.1标准规定的 AC-link数字串口接口协议[1]进行数据的传输。AC-link在 BIT__CLK的上升沿发送数据,在下降沿接收数据,实现全双工的数据传输。

AC-link连接了 ACUNIT和 UCB1400 Codec。本系统只要对 ACUNIT寄存器操作就可以实现同 UCB1400之间的数据传输。另外 Xscale PXA255还提供了 CODEC内部寄存器的地址映射,这样可以直接读写 CODEC内部寄存器,实现对音频采样和混音处理的控制。当然这些读写操作也是经由 AC-link传输的。

2.2 数字部分的处理

数字部分主要是进行音频数据的处理,然后发送到 ACUNIT;或者是从 ACUNIT接收音频数据进行处理。由于音频数据量通常较大,为了减轻 CPU的负荷,本系统采用 DMA负责音频数据的传输,CPU只负责数据的处理,如采样率的转换、数字音频的混合和特效处理等。

Intel Xscale PXA255提供了 16个 DMA通道,可以很方便的为外围设备提供数据传送。ACUNIT也为 CODEC的立体声输入输出和话筒输入提供了独立的 16bit数据通道。每个通道都有一个专门的 FIFO。本系统中为各个 FIFO申请了独立的 DMA通道。当负责接收的 FIFO(Receive FIFO)中数据超过一半时触发 DMA进行数据传输。同样的当负责发送的 FIFO

(Transmit FIFO)中数据少于一半时,也会触发 DMA传输。这样数据可以很快捷的传输到内存中,而且 CPU也可以专心的处理数据,大大提高了效率。

Receive FIFO和 Transmit FIFO都连接到 PCDR寄存器,所以 DMA读 PCDR寄存器就是读取 Receive FIFO,写 PCDR寄存器就是写入 Transmit FIFO,非常便捷。图 3描述了 DMA读写 PCM FIFO的过程:

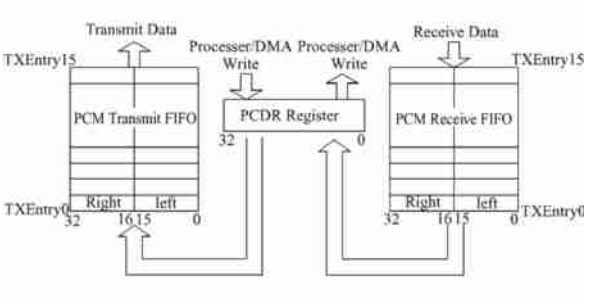

图 3　DMA-FIFO传输

3　软件驱动的实现

嵌入式 Linux把硬件设备都看成是文件,通过打开、读取或写入对应的设备文件实现对硬件设备的读写操作,另外还提供 ioctl接口给上层用户来设置设备参数[5]。

本驱动程序主要是通过对硬件的控制实现音频流的传输,同时向上层提供了标准的音频接口。本音频系统驱动提供了两个标准接口:

① 数字音频处理,即 Dsp,负责音频数据的传输即播放或录音等操作。

② 混频器,即 Mixer,负责对输出音频进行混音处理。如音量调节,高低音控制等。

这两部分分别对应设备文件 /dev/dsp和 /dev/mixer

3.1　设备的初始化

设备的初始化主要就是 UCB1400 Codec的初始化和音频设备的注册,步骤如下:

① 初始化 PXA255的 GPIO口为 AC-link功能口;

② 初始化 ACUNIT的 GCR寄存器,并重启 UCB1400;

③ 读取 UCB1400的 ID号,并初始化 Codec中的各个混音控制寄存器;

④ 根据我们系统的需求调整 Codec混音的各项参数;

⑤ 向 Linux系统注册 dsp设备和 mixer设备。

这样 UCB1400 Codec基本上就处于可用状态了。

3.2　Mixer驱动

Mixer驱动只是控制混音效果,并不执行读写操作,所以 Mixer的文件操作结构只实现了一个 ioctl调用,提供给上层设置 Codec的混音效果。

驱动中主要实现了一个结构 struct ac97_codec pxa_ac97_codec。该结构描述了 Codec的基本信息,主要是实现了 Codec寄存器的读写函数和混音的控制函数。Mixer文件操作结构中的 ioctl就是调用 pxa_ac97_codec中的混音控制函数来实现的。

3.3　Dsp驱动

Dsp驱动实现了音频数据的传输,即播放和录音的数据传输。同时还提供 ioctl对 UCB1400 Codec中 DAC和 ADC采样率进行控制。采样率的控制主要就是对照 UCB1400手册读写 UCB1400 Code中的采样率控制寄存器。所以驱动的主要部分是音频数据传输的控制。

驱动中通过一个结构 static audio_state_t ac97_audio_state来描述整个音频系统的状态,其中最主要的就是两个数据流结构 ac97_audio_in和 ac97_audio_out。这两个结构分别描述输入音频流和输出音频流的信息。

通过对 ac97_audio_in和 ac97_audio_out的操作,分别实现了音频的输入和输出,即音频的播放和录音,本驱动的主要内容就是数据流结构的设计和实现。

该结构应该包含音频缓冲区的信息、DMA的相关信息和所用到的信号量。还有就是 FIFO的入口寄存器的地址,即图 3中所示的 PCDR。

其中在传输时缓冲区的设计是音频传输的重点。以 Write函数为例,因为音频数据量通常较大,缓存太小容易造成缓存溢出,所以要采用较大的缓冲区。而要填充大的缓冲区,CPU就要一次处理大量的数据,这样处理数据时间较长,容易造成延迟。在这种情况下,我们采用多个缓存的机制,将缓冲区分为多个数据段。数据段的个数和大小分别在数据流结构中指定。这样把大的数据段分为几个小段处理,每处理一小段数据就可以通过 DMA发送出去。Read函数也是如此,DMA每

发来一小段数据就可以处理了,而不用等到大缓冲区都填满才处理数据。这里还提供了 ioctl接口给上层调用,这样上层还可以根据音频数据的精度,即数据流量,来调整缓冲区数据段的大小和个数,来取得最好的传输效果。

数据流结构的实现框图如图 4

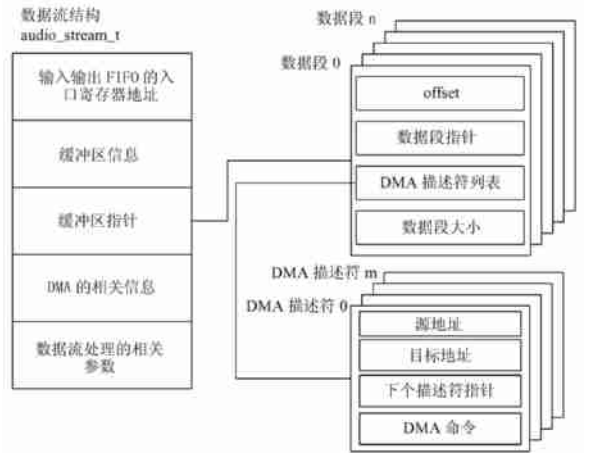

图 4 数据流结构的实现框图

audio__stream__t中的缓冲区指针指向数据段结构列表,每个数据段结构中指明数据段的地址以及对应该数据段的 DMA描述符列表。这样通过循环的 DMA发送就可以实现音频数据流的传输。

另外驱动还提供了另一个函数 mmap,即内存映射。因为 Linux中的用户空间和内核空间是相互独立的,不能直接相互访问的。该函数通过调用 remap__page__range(unsigned long virt__add, unsigned long phys__add, unsigned long size, pgprot__t prot)函数,建立内核缓冲区和用户空间的映射。这样就省去了将数据从用户空间拷贝到内核空间的操作,加快了上层对数据的访问。

4 总结

本文介绍了在嵌入式系统中构架基于 AC97标准的音频系统,实现音频的播放和录音采集。首先讲述了 AC97音频系统的基本构架,然后介绍了基于 Intel Xscale PXA255平台的 AC97 Codec硬件连接的实现和基于嵌入式 Linux的音频驱动程序的实现。该系统已经在基于 Intel XScale PXA255的平台上得到了实现,可以顺利的进行音频的播放和采集,并取得良好的效果。

目前音频系统在手持设备中的使用越来越广泛,但手持设备对于功耗的方面要求较高,所以本系统在电源管理方面仍需优化。

参考文献:

[1] 《Audio Codec '97 Revision 2.1》[S]. Intel Corporation, May 22 1998.

[2] 《Intel(r) PXA255 Processor Developer's Manual》[S]. Intel Corporation, March 2003.

[3] 《UCB1400 Audio codec with touch screen controller and power management monitor Rev. 02》[S]. Philips Semiconductors, June 21 2002.

[4] 肖文鹏.《Linux音频编程指南》[S]. Feb 2004, http://www-900.ibm.com/.

[5] 《Linux Device Drivers 2nd》[S]. Alessandro Rubini & Jonathan Corbet, 2001.