ECEN 449: Microprocessor System Design

Department of Electrical and Computer Engineering

Texas A&M University

Prof. Sunil Khatri

TA: Winfield Wen

Laboratory Report #10

Section 504

Audio Player Application with IR-remote and AC’97 Codec

YAYUN LIU

Dec 4, 2013

**Introduction**

This lab is to create an application which is capable of playing the wav files stored on the CF card. In order to implement the functionality, the custom device drivers for the IR-remote and AC’97 audio device are needed. The system hardware for this lab is depicted in Figure 1.



Figure 1: Hardware/Software System Diagram

**Procedures**

Part1

In the first part, there are two tasks. First, we are required to use a “rising edge” triggered interrupt. Additionally, the transferred data should be from user space to kernel space. We need to write the *write* routine so that the data movement is from user space to kernel.

1. Re-import the audio controller peripheral setting both the playback and record interrupts are rising-edge triggered.

1.1. In the "name and version" page, you need to manually input the name and make sure it's exactly the same as that in lab 9.  
1.2. In the "source file types" page, check both "HDL source files" and "Netlist files"  
1.3. In the "HDL Analysis Information" page, manually add the .vhd files under folder "fifo\_coregen"  
 1.4. In the netlist file page, manually add the files from folder "netlist"  
 1.5. When you select the interrupt sensitivity type, please click on the playback and record interrupts individually and make the selection individually.

1. The **device\_write** routine. In this routine, **get\_user()** method is used to get data from the user data segment to the kernel data segement. The device driver sleeps using ‘**wait\_event\_interruptible()**’ when the audio FIFO is full.
2. In the **interrupt\_handler**, it should only call ‘**wake\_up\_interruptible’** to wake up the *write* method so that it can continue to transfer data.
3. Crete a devtest application to test if the data from the predefined buffer in “audio\_samples.h” is successfully transferred to the audio codec.

Part 2

At this point, we should enable support for audio files which have either one channel or two channels.

1. In the **ioctl()** routine, **cmd** used to determine which command users give. When **cmd** is ENABLE\_DISABLE\_MONO, if val is false, then mono is disable. Otherwise, if val is true, the mono is set.
2. In the *write* method, when mono is set to true, duplicate samples are written to the playback FIFO.

Part 3

The next step is to create an application which can open an audio file, parse the header within the wav file to determine the format of the file, initialize the audio codec accordingly, and copy data from wav file to the audio device driver file.

1. Use *open* method to open our audio device file at the beginning of the main function. And then in the **open\_wav\_file()** routine, initially open wav file. Use *mmap* method to map file to region of memory.
2. Ensure we can play wav file, we should make the ‘format’ filed equal to 1 and ‘bits\_per\_sample’ equal to 8.
3. Next, we should set playback rate utilize the *ioctl* functionality of the audio device driver.

**ioctl(fd\_out, ADJUST\_PLAYBACK\_RATE, setting\_ptr2);**

1. Finally, we need to make the audio\_samples pointer point to beginning of data section and it points to current sample as well.

Part 4

Now we need to add the IR-remote to control the wav player application.

1. Create a 200 byte buffer, which is in user space, to read in all messages in queue.

2. After reading in the IR-remote messages, process every message, updating the system accordingly.

Part 5

The final capability the player application owns is the ability to select various wav files by using certain buttons on the remote.

1. In the *while* loop, determine if file selection changed load in a new file. Munmap and close the previous file using *munmap* and *close* method and then open the new file.
2. If ‘Channel Up’ is pressed, set the **channel\_up** flag and **new\_file** flag to 1. Similarly, when ‘Channel Down’ pressed, set the **channel\_down** flag and **new\_file** flag to 1.

**Results**

The seven wav files on the compact flash card could be played accordingly when I press the ‘Channel Up’ button. If the ‘muting’ button is pressed, no sound comes from the speaker. And the ‘pause’ can stop the sound and play at where it left off when I press ‘play’. And ‘Volume Up’ and ‘Volume Down’ buttons can be utilized to control the sound of wav files.

**Questions**

**[a] In your wav player application, we suggested the use of mmap for reading the contents of the wav file. What other ways are there to accomplished the same thing? Compare and contrast these methods. Some of the things to consider include ease of use, code readability, efficiency, and blocking time. Hint: You can find more information on mmap in the** *O’Reilly’s Linux System Programming* **book available for free on-line.**

Solution

An *lseek/write* implementation can also accomplish the same thing as *mmap* method does.

A big constraint exists when ISA devices are used on some non-x86 platforms, because their hardware view of ISA may not be contiguous. For example, some Alpha computers see ISA memory as a scattered set of 8-bit, 16-bit, or 32-bit items, with no direct mapping. In such cases, you can’t use *mmap* at all. However, there are sound advantages to using *mmap* when it’s feasible to do so. For example, there is a X server, which transfer a lot of data to and from video memory; mapping the graphic display to user space dramatically improves the throughput, as opposed to the *lseek/write* implementation.

**[b] What is the purpose of the call to ‘madvise’? What might be the effect of removing this call? You can test this with the ‘audio samples7.wav’ which is a full length song recorded at 44100 Hz with two channels (stereo)?**

Solution

The **madvise**() system call advises the kernel about how to handle paging input/output in the address range beginning at address start and with size length bytes. It allows an application to tell the kernel how it expects to use some mapped or shared memory areas, so that the kernel can choose appropriate read-ahead and caching techniques.

This call does not influence the semantics of the application (except in the case of **MADV\_DONTNEED**), but may influence its performance.

In this lab, if we remove the call to ‘madvise’, then it can’t play the song continuously. Its performance is not as good as the one when the ‘madvise’ is added.

**[c] Why are wav files stored using little-endian byte representation even though the MicroBlaze is a big-endian processor? What are the costs associated with the conversion from little-endian to big-endian in your current application?**

Solution

WAV files are little-endian(least significant bytes first) because the format originated for operating system running on intel processor based machines which use the little-endian format to store numbers.

Obviously, the costs associated with the conversion should be time and memory space.

**[d] If you wanted to support 16-bit audio in your wav player application, what changes to your existing code would you have to make? Would your audio device driver have to change? If so, how? Hint: Research how 16-bit audio samples are stored in wav files.**

Solution

If we want to support 16-bit audio, then our audio device driver have to be modified. In the current *write* routine of the audio device driver, we have a char variable *sample\_c*, and *get\_user(sample\_c, buff)* method is to get data from the user data segment to the kernel data segment. Now, if we want to support 16-bit, we should make *sample\_c* a *u16* type variable.

**Conclusion**

This lab exercise familiarized me with the device driver creation in an embedded Linux

environment. What is more, this lab provides me with experience developing drivers that handle buffering of real-time data.

**Appendix: C files and Head files**

**Appendix 1: audio\_buffer.c**

/\* Moved all prototypes and includes into the headerfile \*/

#include "audio\_buffer.h"

/\* This structure defines the function pointers to our functions for

opening, closing, reading and writing the device file. There are

lots of other pointers in this structure which we are not using,

see the whole definition in linux/fs.h \*/

// file operations structure

static struct file\_operations fops **=** **{**

**.**read **=** device\_read**,**

**.**write **=** device\_write**,**

**.**open **=** device\_open**,**

**.**release **=** device\_release**,**

**.**ioctl **=** device\_ioctl**,**

**};**

/\*

\* This function is called when the module is loaded and registers a

\* device for the driver to use.

\*/

int my\_init**(**void**)**

**{**

init\_waitqueue\_head**(&**queue**);** /\* initialize the wait queue \*/

/\* Initialize the semaphore we will use to protect against multiple

users opening the device \*/

sema\_init**(&**sem**,** 1**);**

/\* Map virtual address to multiplier physical address \*/

printk**(**KERN\_INFO "Mapping virtual address....\n"**);**

virt\_addr **=** ioremap**(**PHY\_ADDR**,** MEMSIZE**);**

baseaddr **=** **(**u32**)**virt\_addr**;**

printk**(**"physical address is: 0x%x\nvirtual address is: 0x%x\n"**,** PHY\_ADDR**,** baseaddr**);**

Major **=** register\_chrdev**(**0**,** DEVICE\_NAME**,** **&**fops**);**

**if** **(**Major **<** 0**)** **{**

printk**(**KERN\_ALERT "Registering char device failed with %d\n"**,** Major**);**

**return** Major**;**

**}**

printk**(**KERN\_INFO "Registered a device with dynamic Major number of %d\n"**,** Major**);**

printk**(**KERN\_INFO "Create a device file for this device with this command:\n'mknod /dev/%s c %d 0'.\n"**,** DEVICE\_NAME**,** Major**);**

**return** 0**;** /\* success \*/

**}**

/\*

\* This function is called when the module is unloaded, it releases

\* the device file.

\*/

void my\_cleanup**(**void**)**

**{**

/\*

\* Unregister the device

\*/

unregister\_chrdev**(**Major**,** DEVICE\_NAME**);**

/\* Unmapping virtual address space \*/

printk**(**KERN\_ALERT "unmapping virtual address space....\n"**);**

iounmap**((**void**\*)**virt\_addr**);**

**}**

/\*

\* Called when a process tries to open the device file, like "cat

\* /dev/irq\_test". Link to this function placed in file operations

\* structure for our device file.

\*/

static int device\_open**(**struct inode **\***inode**,** struct file **\***file**)**

**{**

int irq\_ret**;**

**if** **(**down\_interruptible **(&**sem**))**

**return** **-**ERESTARTSYS**;**

// We are only allowing one process to hold the device file open at

// a time.

**if** **(**Device\_Open**){**

up**(&**sem**);**

**return** **-**EBUSY**;**

**}**

Device\_Open**++;**

// OK we are now past the critical section, we can release the

// semaphore and all will be well

up**(&**sem**);**

// request a fast IRQ and set handler

irq\_ret **=** request\_irq**(**IRQ\_NUM**,** irq\_handler**,** 0 /\*flags\*/ **,** DEVICE\_NAME**,** **NULL);**

**if** **(**irq\_ret **<** 0**)** **{** /\* handle errors \*/

printk**(**KERN\_ALERT "Registering IRQ failed with %d\n"**,** irq\_ret**);**

**return** irq\_ret**;**

**}**

//initialize AC97

/\*Initialize the codec with or without loopback, this statement could be commented out\*/

XAC97\_InitAudio**(**baseaddr**,** 0**);**

//enables variable rate audio mode (enables sample rate registers and SLOTREQ signaling

XAC97\_WriteReg**(**baseaddr**,** AC97\_ExtendedAudioStat**,** 1**);**

//set master volume to max

//XAC97\_WriteReg(baseaddr, AC97\_MasterVol, AC97\_VOL\_MAX);

//set headphone volume to max

XAC97\_WriteReg**(**baseaddr**,** AC97\_AuxOutVol**,** AC97\_VOL\_MAX**);**

//set the PCM playback rate at 11025Hz

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_DAC\_Rate**,** AC97\_PCM\_RATE\_11025\_HZ**);**

//XAC97\_WriteReg(baseaddr, AC97\_PCM\_ADC\_Rate, AC97\_PCM\_RATE\_11025\_HZ);

mono **=** 0**;**

try\_module\_get**(**THIS\_MODULE**);** /\* increment the module use count

(make sure this is accurate or you

won't be able to remove the module

later. \*/

printk**(**KERN\_ALERT "Device opened\n"**);**

printk**(**KERN\_ALERT"IRQ Num %d called\n"**,** IRQ\_NUM**);**

**return** 0**;**

**}**

/\*

\* Called when a process closes the device file.

\*/

static int device\_release**(**struct inode **\***inode**,** struct file **\***file**)**

**{**

Device\_Open**--;** /\* We're now ready for our next caller \*/

// perform a soft reset on the audio codec

XAC97\_SoftReset**(**baseaddr**);**

// unregister the interrupt handler

free\_irq**(**IRQ\_NUM**,** **NULL);**

/\*

\* Decrement the usage count, or else once you opened the file,

\* you'll never get get rid of the module.

\*/

module\_put**(**THIS\_MODULE**);**

printk**(**KERN\_ALERT "Device closed\n"**);**

**return** 0**;**

**}**

/\*

\* Called when a process, which already opened the dev file, attempts to

\* read from it.

\*/

static ssize\_t device\_read**(**struct file **\***filp**,** /\* see include/linux/fs.h \*/

char **\***buffer**,** /\* buffer to fill with data \*/

size\_t length**,** /\* length of the buffer \*/

loff\_t **\*** offset**)**

**{**

/\* Fail \*/

**}**

/\*

\* Called when a process writes to dev file

\*/

static ssize\_t device\_write**(**struct file **\***filp**,** const char **\***buff**,** size\_t len**,** loff\_t **\*** off**)**

**{**

u32 sample**;**

char sample\_c**;**

**while(**len**)**

**{**

/\*

\* The buffer is in user data segment, not the kernel segment.

\* We use get\_user which gets data from the user data segment

\* to the kernel data segment.

\*/

get\_user**(**sample\_c**,** buff**);**

sample **=** 10**\*(**u32**)((**unsigned char**)**sample\_c**);**

buff**++;**

/\* sleep when the audio FIFO is full \*/

//wait\_event\_interruptible(queue, XAC97\_isInFIFOFull(baseaddr) );

**if(**XAC97\_isInFIFOFull**(**baseaddr**))**

wait\_event\_interruptible**(**queue**,** 1**);**

//XAC97\_WriteFifo(baseaddr, sample);

//XAC97\_WriteFifo(baseaddr, sample);

**if(!**mono**)**

**{**

// stereo

XAC97\_WriteFifo**(**baseaddr**,** sample**);**

XAC97\_WriteFifo**(**baseaddr**,** sample**);**

**}**

**else**

// mono

XAC97\_WriteFifo**(**baseaddr**,** sample**);**

len**--;**

**}**

**return** 0**;** /\* Fail \*/

**}**

/\* Just wake up anything waiting for the device \*/

irqreturn\_t irq\_handler**(**int irq**,** void **\***dev\_id**)**

**{**

wake\_up\_interruptible**(&**queue**);**

**return** 0**;**

**}**

/\* This function allows the user process to provide control

command to our device driver and read status from the device \*/

static int device\_ioctl**(**struct inode **\***inode**,**

struct file **\***file**,**

unsigned int cmd**,**

unsigned int **\***val\_ptr**)**

**{**

u16 val**;** //temportary value

get\_user**(**val**,** val\_ptr**);** //grab value from user space

//printk(KERN\_ALERT "cmd = %d, val = %d\n", cmd, val);

// switch statement to execute command

**switch(**cmd**)**

**{**

//enable or disenable mono

**case** ENABLE\_DISABLE\_MONO**:**

**if(**val**==**0**)**

mono**=**0**;**

**else** **if(**val**==**1**)**

mono**=**1**;**

**break;**

// adjust aux volume

**case** ADJUST\_AUX\_VOL**:**

XAC97\_WriteReg**(**baseaddr**,** AC97\_AuxOutVol**,** val**);**

**break;**

// adjust playback rate

**case** ADJUST\_PLAYBACK\_RATE**:**

**if(**val**==**0**){**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_DAC\_Rate**,** AC97\_PCM\_RATE\_8000\_HZ**);**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_ADC\_Rate**,** AC97\_PCM\_RATE\_8000\_HZ**);**

**}**

**else** **if(**val**==**1**){**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_DAC\_Rate**,** AC97\_PCM\_RATE\_11025\_HZ**);**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_ADC\_Rate**,** AC97\_PCM\_RATE\_11025\_HZ**);**

**}**

**else** **if(**val**==**2**){**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_DAC\_Rate**,** AC97\_PCM\_RATE\_16000\_HZ**);**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_ADC\_Rate**,** AC97\_PCM\_RATE\_16000\_HZ**);**

**}**

**else** **if(**val**==**3**){**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_DAC\_Rate**,** AC97\_PCM\_RATE\_22050\_HZ**);**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_ADC\_Rate**,** AC97\_PCM\_RATE\_22050\_HZ**);**

**}**

**else** **if(**val**==**4**){**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_DAC\_Rate**,** AC97\_PCM\_RATE\_44100\_HZ**);**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_ADC\_Rate**,** AC97\_PCM\_RATE\_44100\_HZ**);**

**}**

**else** **if(**val**==**5**){**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_DAC\_Rate**,** AC97\_PCM\_RATE\_48000\_HZ**);**

XAC97\_WriteReg**(**baseaddr**,** AC97\_PCM\_ADC\_Rate**,** AC97\_PCM\_RATE\_48000\_HZ**);**

**}**

**break;**

// if unknown command, error out

**default:**

printk**(**KERN\_INFO "Unsupported control command!\n"**);**

**return** **-**EINVAL**;**

**}**

**return** 0**;**

**}**

/\* These define info that can be displayed by modinfo \*/

MODULE\_LICENSE**(**"GPL"**);**

MODULE\_AUTHOR**(**"Yayun Liu"**);**

MODULE\_DESCRIPTION**(**"xac97 driver"**);**

/\* Here we define which functions we want to use for initialization

and cleanup \*/

module\_init**(**my\_init**);**

module\_exit**(**my\_cleanup**);**

**Appendix 2: wav\_play.c**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <unistd.h>

#include <sys/dir.h>

#include <sys/param.h>

#include <dirent.h>

#include <string.h>

#include "parse\_wav.c"

/\*user defined header files\*/

#include "sound.h"

#define SMAPLE\_SIZE 4000

#define ENABLE\_MONO 1

#define DISABLE\_MONO 0

#define AC97\_VOL\_ATTN\_DELTA\_DB 0x0101

#define ENABLE\_PLAY 1

#define DISABLE\_PLAY 0

/\*function prototypes\*/

static int open\_wav\_file**(**char**\*** filename**);**

static int wav\_file\_filter**(**const struct direct **\***entry**);**

/\*global variables shared by all functions in this source\*/

static int fd\_out**,** fd\_in**;**//file designators for audio device and wav file

static int fd\_ir**;** // file designator for ir device

static struct stat file\_info**;**//used for getting file length needed by mmap

int val**;**//temp variable used for ioctl calls

unsigned int **\***val\_ptr**;**

static void **\***contents**;**//points to the beginning of the mmap region

static unsigned char **\***audio\_ptr**;**//points to current audio sample

static unsigned char **\***samples**;**//points to the beginning of the audio data

static int num\_samples**;**//number of samples in current wav file

//static unsigned short msg\_buff[200];//used to hold read IR messages

char msg\_buff**[**200**];**

/\*audio playback mode settings\*/

int vol\_setting**,** playback\_rate**,** mono\_setting**,** pr\_val**;** //hold current audio settings

unsigned int **\***setting\_ptr1**,** **\***setting\_ptr2**;**

static int play\_audio **=** 0**;**//play flag

//static int stop\_audio = 1;//stop flag

static int completion\_mode **=** 1**;** //0=stop,1=repeat,2=forward,3=reverse

static int mute\_audio **=** 0**;**//mute flag

static int channel\_up **=** 0**;**//channel up flag

static int channel\_down **=** 0**;**//channel down flag

static int volume\_up **=** 0**;**//volume up flag

static int volume\_down **=** 0**;**//volume down flag

static int num**=**0**;**

static int new\_file **=** 0**;**

static unsigned short current\_volume **=** AC97\_VOL\_ATTN\_0\_DB**;**

int main**()**

**{**

int i**;** //loop variable

unsigned short user\_input**;**

int ir\_read\_bytes**,** audio\_write\_bytes**;**

int quit **=** 0**;** //exit flag

//counters which handle the multple message problem

int wait\_count **=** 0**;**

struct dirent **\*\***namelist**;**//structure pointer for reading directory

int num\_files**;** //number of wav files contained in directory

int cur\_file**=**0**;** //index points to wav file currently open

int nxt\_file**=**0**;**

unsigned int cmd**;**

char buffer**[**4000**];**

//char buffer\_ir[200]={0};//a 200 byte buffer in user space

int result**;**

/\*scan current directory for wav files and provide them in alphabetical order\*/

num\_files **=** scandir**(**"."**,** **&**namelist**,** wav\_file\_filter**,** alphasort**);**

**if(**num\_files **<** 0**)** //handle errors

**{**

perror**(**"scandir"**);**

**return** **-**1**;**

**}**

/\*open ir device file\*/

fd\_ir **=** open**(**"/dev/irq\_test"**,** O\_RDWR**);**

/\*handle error opening file\*/

**if(**fd\_ir**<**0**)**

**{**

printf**(**"Failed to open ir device!\n"**);**

**return** **-**1**;**

**}**

/\*open audio device file \*/

fd\_out **=** open**(**"/dev/xac97"**,** O\_RDWR**);**

/\* handle error opening file \*/

**if** **(**fd\_out **<** 0**)**

**{**

printf**(**"Failed to open ac97 device file!\n"**);**

**return** **-**1**;**

**}**

// for test

printf**(**"Before opening\n"**);**

/\*initially open first wav file\*/

/\*open and mmap file, check for valid file, and adjust codec\*/

printf**(**"Opening %s\n"**,** namelist**[**cur\_file**]->**d\_name**);**

**if(**open\_wav\_file**(**namelist**[**cur\_file**]->**d\_name**)<**0**)**

**return** **-**1**;**//if error opening

**while(**quit **!=** 1**)** //continue until killed

**{**

/\*if file selection changed load in a new file\*/

**if(**nxt\_file **!=** cur\_file **&&** new\_file**==**1**)**

//if( nxt\_file != cur\_file)

**{**

new\_file **=** 0**;**

num **=** 0**;**

//\*munmap current mapping\*/

munmap**(**contents**,** file\_info**.**st\_size**);**

/\*close currently opened file\*/

close**(**fd\_in**);**

printf**(**"Opening %s\n"**,** namelist**[**nxt\_file**]->**d\_name**);**

/\*open and mmap file, check for valid file, and adjust codec\*/

**if(**open\_wav\_file**(**namelist**[**nxt\_file**]->**d\_name**)<**0**)**

**return** **-**1**;**//if error opening

/\*clear contents already written to audio device\*/

/\*update current file index\*/

cur\_file **=** nxt\_file**;**

**}**

printf**(**"play\_audio is %d.\n"**,** play\_audio**);**

/\*copy num\_samples worth of samples from wav file to

audio device in repeat mode\*/

//if(play\_audio == 1 && stop\_audio == 0)

**if(**play\_audio **==** ENABLE\_PLAY**)**

**{**

printf**(**"It's playing!\n"**);**

**for(**i**=**0**;** i**<** 4000**;** i**++)**

**{**

buffer**[**i**]** **=** **\***audio\_ptr**;**

num**++;**

audio\_ptr**++;**

**if(**num**==**num\_samples**)**

**{**

num**=**0**;**

audio\_ptr **=** samples**;**

**break;**

**}**

**}**

//for testing

write**(**fd\_out**,** buffer**,** 4000**);**

**}**

**else** // if not writting audio data

usleep**(**100000**);** // sleep for 100ms

/\*read in all messages in queue\*/

read**(**fd\_ir**,** msg\_buff**,** 4**);**

//convert char to int type

result **=** **(**unsigned int**)((**unsigned char**)**msg\_buff**[**3**])\***256**\***256**\***256 **+**

**(**unsigned int**)((**unsigned char**)**msg\_buff**[**2**])\***256**\***256 **+**

**(**unsigned int**)((**unsigned char**)**msg\_buff**[**1**])\***256 **+**

**(**unsigned int**)((**unsigned char**)**msg\_buff**[**0**]);**

printf**(**"Decoded message : 0x%x\n"**,** result**);**

**switch(**result**)**

**{**

/\*change the volume\*/

//when volume up is pressed

**case** 0x92000000**:**

printf**(**"volume up\n"**);**

**if(**volume\_up**==**0**)**

**{**

**if(**current\_volume**==**AC97\_VOL\_ATTN\_0\_DB**)**

current\_volume **=** AC97\_VOL\_ATTN\_0\_DB**;**

**else**

current\_volume **=** current\_volume **-** AC97\_VOL\_ATTN\_DELTA\_DB**;**

volume\_up **=** 1**;**

**}**

**else** **if(**wait\_count **==** 0**)**

**{**

volume\_up **=** 0**;**

**}**

val **=** current\_volume**;**

val\_ptr **=** **&**val**;**

ioctl**(**fd\_out**,**ADJUST\_AUX\_VOL**,**val\_ptr**);**

**break;**

//when volume down is pressed

**case** 0x93000000**:**

printf**(**"volume down\n"**);**

**if(**volume\_down**==**0**)**

**{**

**if(**current\_volume**==**AC97\_VOL\_ATTN\_46\_0\_DB**)**// the lowest sound

current\_volume **=** AC97\_VOL\_ATTN\_46\_0\_DB **;**

**else**

current\_volume **=** current\_volume **+** AC97\_VOL\_ATTN\_DELTA\_DB**;**

volume\_down **=** 1**;**

**}**

**else** **if(**wait\_count **==** 0**)**

**{**

volume\_down **=** 0**;**

**}**

val **=** current\_volume**;**

val\_ptr **=** **&**val**;**

ioctl**(**fd\_out**,**ADJUST\_AUX\_VOL**,**val\_ptr**);**

**break;**

/\*change the channel to switch wav files\*/

//if channel up pressed

**case** 0x90000000**:**

printf**(**"channel up\n"**);**

**if(**channel\_up**==**0**)**

**{**

/\*open next wav file in directory\*/

nxt\_file**=**cur\_file**+**1**;**

new\_file **=** 1**;**

**if(**nxt\_file**==**num\_files**)**

nxt\_file**=**0**;**//circular nature

channel\_up **=** 1**;**

**}**

**else** **if(**wait\_count **==** 0**)**

**{**

channel\_up **=** 0**;**

**}**

**break;**

//if channel down pressed

**case** 0x91000000**:**

printf**(**"channel down\n"**);**

**if(**channel\_down**==**0**)**

**{**

/\*open next wav file in directory\*/

nxt\_file**=**cur\_file**-**1**;**

new\_file **=** 1**;**

**if(**cur\_file**==**0**)**

nxt\_file**=**num\_files**-**1**;**//circular nature

**break;**

channel\_down **=** 1**;**

**}**

**else** **if(**wait\_count **==** 0**)**

**{**

channel\_down **=** 0**;**

**}**

//when play is pressed

**case** 0xdf000000**:**

printf**(**"play\n"**);**

play\_audio **=** ENABLE\_PLAY**;**

// stop\_audio = 0;

**break;**

//when pause is pressed

**case** 0xdc000000**:**

printf**(**"pause\n"**);**

play\_audio **=** DISABLE\_PLAY**;**

**break;**

//when stop is pressed

**case** 0xde000000**:**

// stop\_audio = 1;

printf**(**"stop\n"**);**

play\_audio **=** DISABLE\_PLAY**;**

num**=**0**;**

audio\_ptr **=** samples**;**

**break;**

/\*when mute pressed\*/

**case** 0x94000000**:**

/\*update codec\*/

printf**(**"muting\n"**);**

val **=** AC97\_VOL\_MUTE**;**

val\_ptr **=** **&**val**;**

ioctl**(**fd\_out**,**ADJUST\_AUX\_VOL**,**val\_ptr**);**

**break;**

//when exit is pressed

**case** 0xe3000000**:**

printf**(**"exit"**);**

quit **=** 1**;**

**break;**

**default:**

**;**

**}**

/\*maintain wait scheme used to ignore multiple messages\*/

**if(**wait\_count**>**3**)**

wait\_count **=** 0**;**

**else** **if(**wait\_count**!=**0**)**

wait\_count**++;**

**for(**i**=**0**;** i**<**4**;** i**++)**

msg\_buff**[**i**]** **=** 0**;**

**}**

close**(**fd\_in**);**

close**(**fd\_out**);**

**}**

/\*this function opens a wav file,

maps its contents to memory, parses

the wav header, and adjusts the audio

codec accordingly

\*/

static int open\_wav\_file**(**char**\*** filename**)**

**{**

wav\_properties\_t wav\_info**;**

/\*open wave file\*/

fd\_in**=**open**(**filename**,**O\_RDONLY**);**

/\*handle error opening file\*/

**if(**fd\_in **<**0**){**

printf**(**"Failed to wave file!\n"**);**

**return** **-**1**;**

**}**

/\*get file info\*/

fstat**(**fd\_in**,&**file\_info**);**

/\*use mmap to map file to region of memory\*/

//mmap file to contents use flags for read only and shared mapping

//printf("Mapping wav files\n");

contents **=** mmap**(**0**,** file\_info**.**st\_size**,** PROT\_READ**,** MAP\_SHARED**,** fd\_in**,** 0**);**

//ensure mmap worked properly

**if(**contents **==** MAP\_FAILED**)**

**{**

printf**(**"Error with mmap!\n"**);**

**return** **-**1**;**

**}**

/\*advise the kernel to do read ahead\*/

madvise**(**contents**,**file\_info**.**st\_size**,**MADV\_WILLNEED**);**

//printf("Doing madvise success\n");

/\*get wave file info\*/

**if(** parse\_wav**((**char**\*)**contents**,&**wav\_info**)** **<**0 **)**

**{**

printf**(**"Invalid file type!\n"**);**

**return** **-**1**;**

**}**

/\*ensure we can play wav file\*/

**if(**wav\_info**.**format **!=** 1**)**

**{**

printf**(**"Compressed wav files are not supported!\n"**);**

**return** **-**1**;**

**}**

**if(**wav\_info**.**bits\_per\_sample **!=** 8**)**

**{**

printf**(**"Only 8-bit audio is currently supported!\n"**);**

**return** **-**1**;**

**}**

/\*calculate the number of samples in file\*/

num\_samples **=** wav\_info**.**num\_bytes**;**//only supporting 8-bit audio

/\*adjust audio properties based on wav file\*/

/\*set play mode \*/

**if(**wav\_info**.**num\_channels **==** 1**)**//if mono

mono\_setting**=**0**;**//set mono

**else** //if stereo

mono\_setting**=**1**;**//reset mono

//use ioctl to enable or disable mono

setting\_ptr1 **=** **&**mono\_setting**;**

ioctl**(**fd\_out**,** ENABLE\_DISABLE\_MONO**,** setting\_ptr1**);**

/\*set playback rate accordingly\*/

playback\_rate **=** wav\_info**.**sample\_rate**;**

**if(**playback\_rate**==**8000**)**

pr\_val **=** 0**;**

**if(**playback\_rate**==**11025**)**

pr\_val **=** 1**;**

**if(**playback\_rate**==**16000**)**

pr\_val **=** 2**;**

**if(**playback\_rate**==**22050**)**

pr\_val **=** 3**;**

**if(**playback\_rate**==**44100**)**

pr\_val **=** 4**;**

**if(**playback\_rate**==**48000**)**

pr\_val **=** 5**;**

setting\_ptr2 **=** **&**pr\_val**;**

//set playback rate

ioctl**(**fd\_out**,** ADJUST\_PLAYBACK\_RATE**,** setting\_ptr2**);**

/\* part 2(i) initialize audio buffer pointers\*/

samples **=** wav\_info**.**audio\_samples**;**//point to beginning of data section

audio\_ptr **=** wav\_info**.**audio\_samples**;**//points to current sample

/\*set play flag to start playing file\*/

play\_audio **=** ENABLE\_PLAY**;**

printf**(**"success!\n"**);**

**return** 0**;** //success!

**}**

/\*function used by scandir to filter out all files except

those with the .wav extension\*/

static int wav\_file\_filter**(**const struct direct **\***entry**)**

**{**

**if(**strstr**(**entry**->**d\_name**,**".wav"**))** //if .wav in filename

**return** 1**;**//true

**else**

**return** 0**;**//false

**}**

**Appendix 3: devtest.c**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include "sound.h"

#include "audio\_samples.h"

static unsigned short**\*** sample\_ptr **=** **&**audio\_samples**[**0**];**

int main**()**

**{**

int fd**;**

int i**;**

int input**;**

unsigned int cmd**;**

unsigned int **\***val\_ptr**;**

static int num**=**0**;**

char buffer**[**NUM\_SAMPLES**];**

sample\_ptr **=** **&**audio\_samples**[**0**];**

/\* open device file for reading and writing \*/

/\* use "open" to open '/dev/xac97' \*/

fd **=** open**(**"/dev/xac97"**,** O\_RDWR**);**

/\* handle error opening file \*/

**if(**fd **==** **-**1**)**

**{**

printf**(**"Failed to open device file!\n"**);**

**return** **-**1**;**

**}**

printf**(**"Enter 'v' for adjusting volume, 'f' for adjusting frequency and 'q' to exit.\n"**);**

**while(**input **!=** 'q'**)**

**{**

**if(** input **==** 'v'**){**

input **=** input**;**

printf**(**"Please select the voice level. 0-mute, 1-mid, 2-max.\n"**);**

scanf**(**"%d"**,&**num**);**

val\_ptr **=** **&**num**;**

**if(\***val\_ptr**==** 0 **||** **\***val\_ptr**==**1 **||** **\***val\_ptr**==**2**)**

cmd **=** ADJUST\_AUX\_VOL**;**

ioctl**(**fd**,** cmd**,** val\_ptr**);**

**}**

**else** **if(** input **==** 'f'**){**

printf**(**"Please select the frequency. 0-5--> slow-fast.\n"**);**

scanf**(**"%d"**,&**num**);**

val\_ptr **=** **&**num**;**

**if(\***val\_ptr**==**0 **||** **\***val\_ptr**==**1 **||** **\***val\_ptr**==**2 **||** **\***val\_ptr**==**3 **||** **\***val\_ptr**==**4 **||** **\***val\_ptr**==**5**)**

cmd **=** ADJUST\_PLAYBACK\_RATE**;**

ioctl**(**fd**,** cmd**,** val\_ptr**);**

**}**

**else** **if(** input **==** 'm'**){**

printf**(**"Please select the track. 0-stereo, 1-mono.\n"**);**

scanf**(**"%d"**,&**num**);**

val\_ptr **=** **&**num**;**

**if(\***val\_ptr**==**0 **||** **\***val\_ptr**==**1**)**

cmd **=** ENABLE\_DISABLE\_MONO**;**

ioctl**(**fd**,** cmd**,** val\_ptr**);**

**}**

**for(**i**=**0**;** i**<** NUM\_SAMPLES**;** i**++)**

**{**

buffer**[**i**]** **=** **(**char**)** audio\_samples**[**num**];**

num**++;**

**if(**num**==**NUM\_SAMPLES**)**

**{**

num**=**0**;**

**break;**

**}**

**}**

write**(**fd**,** buffer**,** NUM\_SAMPLES**);**

input **=** getchar**();**

**}**

close**(**fd**);**

**return** 0**;**

**}**

**Appendix 4: parse\_wav.c**

#include "parse\_wav.h"

/\*microblaze is big endian but wave files are strore

using little endian so we must reverse byte order\*/

void reverse\_endian**(**void **\***in\_ptr**,** void **\***out\_ptr**,** int num\_bytes**)**

**{**

int i**;** //loop variable

unsigned char**\*** in **=** in\_ptr**;**

unsigned char**\*** out **=** out\_ptr**;**

/\*loop over number of bytes\*/

**for(**i**=**0**;** i**<**num\_bytes**;** i**++)**

out**[**i**]** **=** in**[(**num\_bytes**-**1**)-**i**];**

**}**

/\*Extract wave file properties given by \*buff.

Return -1 if the file is not of proper

format.\*/

int parse\_wav**(**unsigned char **\***buff**,** wav\_properties\_t **\***wav**)**

**{**

/\*strings used for file format checks\*/

char riff\_str**[**5**]** **=** "RIFF"**;**

char wave\_str**[**5**]** **=** "WAVE"**;**

char fmt\_str**[**5**]** **=** "fmt "**;**

char data\_str**[**5**]** **=** "data"**;**

/\*points to the next byte in the array\*/

unsigned char**\*** cur\_ptr **=** buff**;**

/\*temporary variables used to hold fields within file\*/

unsigned int length**,** byte\_rate**,** tmp**;**

unsigned int sample\_rate**,** num\_bytes**;**

unsigned short format**,** num\_channels**,**bits\_per\_sample**;**

/\*ensure curid wave file\*/

/\*the first 4 bytes of wave file should

contain "RIFF"\*/

**if(**strncmp**((**char**\*)**cur\_ptr**,**riff\_str**,**4**)!=**0**)**

**return** **-**1**;**

cur\_ptr **=** cur\_ptr **+** 4**;**

/\*the next 4 bytes contains the length

of the rest of the file\*/

#ifdef isBigEndian

reverse\_endian**(**cur\_ptr**,&**length**,** 4**);**

#else

length **=\*(**unsigned int**\*)** cur\_ptr**;**

#endif

#ifdef printInfo

printf**(**"length of file %d\n"**,**length**);**

#endif

cur\_ptr**=**cur\_ptr**+**4**;**

/\*next 4 bytes should contain "WAVE"\*/

**if(**strncmp**((**char**\*)**cur\_ptr**,**wave\_str**,**4**)!=**0**)**

**return** **-**1**;**

cur\_ptr **=** cur\_ptr **+** 4**;**

/\*next 4 bytes should contain "fmt "\*/

**if(**strncmp**((**char**\*)**cur\_ptr**,**fmt\_str**,**4**)!=**0**)**

**return** **-**1**;**

cur\_ptr **=** cur\_ptr **+** 8**;**

/\*format check complete\*/

/\*format field\*/

#ifdef isBigEndian

reverse\_endian**(**cur\_ptr**,&**format**,**2**);**

#else

format **=\*(**unsigned short**\*)**cur\_ptr**;**

#endif

#ifdef printInfo

printf**(**"format = %d\n"**,**format**);**

#endif

cur\_ptr**=**cur\_ptr**+**2**;**

/\*number of channels field\*/

#ifdef isBigEndian

reverse\_endian**(**cur\_ptr**,&**num\_channels**,**2**);**

#else

num\_channels **=** **\*(**unsigned short**\*)**cur\_ptr**;**

#endif

//printf("Number of channels = %d\n",num\_channels);

cur\_ptr**=**cur\_ptr**+**2**;**

/\*sampling rate field\*/

#ifdef isBigEndian

reverse\_endian**(**cur\_ptr**,&**sample\_rate**,**4**);**

#else

sample\_rate **=** **\*(**unsigned int**\*)**cur\_ptr**;**

#endif

#ifdef printInfo

printf**(**"Sampling Rate = %d\n"**,**sample\_rate**);**

#endif

cur\_ptr**=**cur\_ptr**+**4**;**

/\*byte rate field\*/

#ifdef isBigEndian

reverse\_endian**(**cur\_ptr**,&**byte\_rate**,**4**);**

#else

byte\_rate **=** **\*(**unsigned int**\*)**cur\_ptr**;**

#endif

#ifdef printInfo

printf**(**"Byte Rate = %d\n"**,**byte\_rate**);**

#endif

cur\_ptr**=**cur\_ptr**+**6**;**//skip block align for now

/\*bits per sample field (2 bytes)\*/

#ifdef isBigEndian

reverse\_endian**(**cur\_ptr**,&**bits\_per\_sample**,**2**);**

#else

bits\_per\_sample **=** **\*(**unsigned short**\*)**cur\_ptr**;**

#endif

#ifdef printInfo

printf**(**"Bits per Sample = %d\n"**,** bits\_per\_sample**);**

#endif

cur\_ptr**=**cur\_ptr**+**2**;**

/\*sanity check\*/

tmp **=** sample\_rate**\***num\_channels**\***bits\_per\_sample**/**8**;**

**if(**byte\_rate **!=** tmp**)**

**return** **-**1**;**

/\*more sanity checks\*/

/\*next 4 bytes should contain "data"\*/

**if(**strncmp**((**char**\*)**cur\_ptr**,**data\_str**,**4**)!=**0**)**

**return** **-**1**;**

cur\_ptr **=** cur\_ptr **+** 4**;**

/\*next 4 bytes contains the number of data bytes\*/

#ifdef isBigEndian

reverse\_endian**(**cur\_ptr**,&**num\_bytes**,**4**);**

#else

num\_bytes **=** **\*(**unsigned int**\*)**cur\_ptr**;**

#endif

#ifdef printInfo

printf**(**"Number of data bytes = %d\n"**,**num\_bytes**);**

#endif

cur\_ptr **=** cur\_ptr **+** 4**;** //point to beginning of data

/\*place parsed values into structure\*/

wav**->**format **=** format**;**

wav**->**num\_channels **=** num\_channels**;**

wav**->**sample\_rate **=** sample\_rate**;**

wav**->**bits\_per\_sample **=** bits\_per\_sample**;**

wav**->**num\_bytes **=** num\_bytes**;**

wav**->**audio\_samples **=** cur\_ptr**;**

/\*if we got to this point, the file has been successfully

parsed!\*/

**return(**0**);**

**}**