Emulating a Sound Chip in Software

David Glandon

April 28, 2005

Contents

List of Figures

Chapter 1

Introduction

Emulating a sound chip in software proved to be an interesting project. The emulation was to be pin-level accurate. This meant that the software could emulate the hardware using any method provided the external pins of the chip worked consistently with the hardware example. The main problems faced in emulating a sound chip were understanding how a sound chip works, learning what cycle rates are needed for a successful emulation, and evaluating the results of the emulation to ensure accuracy.

The software emulation was to be part of a larger hardware simulation project. The project goal was to create a simulation environment that served as a virtual breadboard. Through the user-interface, it would be possible to add multiple chips whose emulation was already implemented in a library. The simulation environment would then execute the simululated environment and use virtual interconnections to allow the chips to communicate with each other. The clock cycle created by the simulation environment would serve as the clock for all chips to be simulated.

Chapter 2

Background on Digital Audio

Digital audio, when created, is recorded at a certain sample rate. This rate, such as the $44.1KHz$ standard for audio CDs, is the number of samples per second of the analog audio signal that are captured into digital data. The device capturing the audio signal converts the analog to a digital Pulse Code Modulation (PCM) representation. PCM is one way that audio data can be represented digitally. Most digital audio systems use PCM data to represent the analog waveform. Compact discs and DAT tapes are common audio storage mediums that use PCM to represent audio. PCM is a digital represention of an analog sample. A 16-bit signed (2's complement) PCM sample will allow $65,536$ possible values to represent the analog sample. If the analog sample is at the minimum value possible, then the digital value will be $-32,768$. If the analog sample is at the maximum value possible, then the digital value will be $32,767$. Any analog sample in between these two extremes will be linearly interpolated into a 16-bit digital value.

The only data that is important for reconstructing the audio signal are the frequencies between $0Hz$ and half of the sample frequency. This is due to Nyquist's theorem which states that data must be sampled at twice the frequency that needs to be represented. Any frequencies above this "Nyquist Rate" are aliased and do not accurately represent the sampled data. Ideally, a low-pass filter would be used on the data to essentially throw out the data above the Nyquist rate. This ideal low-pass filter would have no attenuation below the Nyquist rate, but have full attenuation above

this rate. Unfortunately, an analog filter that satisfies this requirement is impossible to create. This is where oversampling improves audio quality by reducing the requirements of the analog low-pass filter. A digital filter, such as a Finite Impulse Response (FIR) filter can be used to execute a simple low-pass filter on the sampled data. The output of this digital filter can then be sampled at a higher rate. As this oversampled data is converted from Digital to Analog for output, a less precise analog low-pass filter can be used and have less of an impact on the audio quality of the output [3].

Chapter 3

Methods

Several chips were considered for the sound chip simulation. The Analog Devices' (www.analog.com) AD1851 chip was the basis for this software simulation. It was chosen because it is specifically designed for audio output and represents a simple audio digital to audio converter (DAC). Its data input is a 16-bit signed (2's complement) PCM, and it can support a $12.5MHz$ data rate. The audio data is input into the chip using a serial, MSB format. The Data, Clock, and Latch Enable pins facilitate the input of the data. The chip does not do any oversampling or filtering (analog or digital). The analog output of the AD1851 is $\pm 3V$ [1].

Another chip that was considered, although not as seriously, was the DAC7742 by Texas Instruments (www.ti.com). This chip was also a 16-bit DAC but was not designed for audio reproduction. Its manual was very descriptive, however, and aided in understanding how a DAC works. It helped explain the purposes of the feedback resistor and summing junction pins to allow better understanding of how to use a Digital to Analog chip [4].

The first chip to be considered for the simulation was the Analog Devices AD1858. This chip was considered because it is designed for stereo audio and filters the digital input before the analog output is generated. It supports many more data input modes than the AD1851. It accepts both right-justified and DSP serial port style 16-bit stereo input. In addition to the digital to analog conversion, the AD1858 uses a finite impulse response (FIR) filter to interpolate and oversample

the digital data 128 times. In addition to the oversampling, a sigma-delta modulator is also used to improve the digital data to reduce the load on the analog filtering step. After the oversampled data is converted to analog data, the AD1858 then uses an analog filter to reconstruct the audio data [2]. The complexity of the FIR interpolation and the sigma-delta modulator made this chip difficult to use for the simulation. The specifics of the digital filtering were not available in the data-sheet acquired from Analog Devices, and the proprietary data was not available through customer support. In addition, the analog filtering would be difficult to simulate in software.

Before beginning a software emulation of a sound chip, it was necessary to determine the method that a sound chip uses to convert digital audio data into an analog output. The digital input of the sound chip was to be emulated using PCM digital data. To test the use of PCM data to represent an analog signal, an experiment was constructed. A digital audio file would be output through the computer sound card. The analog data would then be captured through a PCI A/D card on the same computer. This A/D card would capture the voltage being input from the sound card and digitally represent that value. The captured digital signal could then be played on the computer to assure the sound capture and its linear representation accurately represented the original digital audio. Audacity (http://audacity.sourceforge.net) was used to compare the waveforms of the original digital audio to the captured audio.

Chapter 4

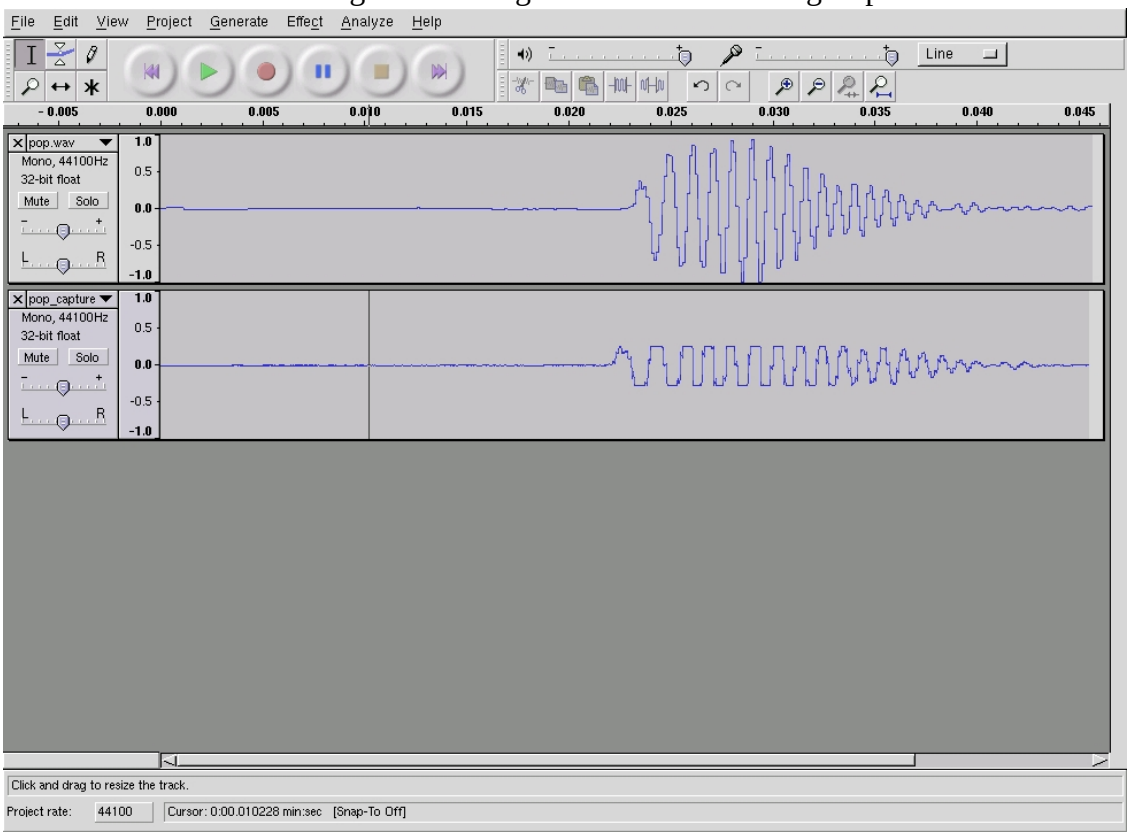
Results

The AD1851 supports an input rate of up to 12.5MHz or 80ns. The data must be clocked in at a rate of at least 705.6KHz or 1.41us for a 44.1KHz audio signal. The embedded system environment can support 12us or 83.3KHz. This is limited by the time required to synchronize threads in the embedded system environment.

The steps for testing the playback and capture of audio to test the hypothesis are as follows. To capture analog data from the A/D card, a program was written to continually capture the analog data. The code executed to capture the analog data uses the comedi libraries and kernel modules (http://www.comedi.org). The exact sample rate is not able to be specified because the function call to make an analog capture is a blocking call. The analog capture card also only had a resolution of 12 bits. Each 12-bit value then had to be normalized to a 16-bit value. This is achieved by multiplying the original value by 16 or $2^{16}/2^{12}$. The waveforms of the original digital audio file and the captured PCM data are very similar as shown in the screenshot from audacity in Figure 4.1. The top waveform is from pop.wav included with KDE and the other waveform is the PCM data captured through the A/D card. The amplitude of the captured waveform is less because the output maximum was less than the range of the A/D card.

To allow the A/D card (Measurement Computing's PCI-DAS08) to capture the audio, the comedi module and libraries were needed. Since the system being used is based on Redhat 9.0, the

Figure 4.1: Original Waveform/Analog Capture



```
#include <stdio.h>                    /* for printf() */
#include <comedilib.h>

int subdev = 0;
int chan = 0;
int range = 0;
int aref = AREF_GROUND;

int main(int argc, char *argv[])
{
    comedi_t *cf;
    lsampl_t data;
    int maxdata;
    double volts;
    comedi_range *cr;

    cf = comedi_open("/dev/comedi0");
    maxdata = comedi_get_maxdata(cf, subdev, chan);
    cr = comedi_get_range(cf, subdev, chan, range);
    while(1)
    {
        comedi_data_read(cf, subdev, chan, range, aref, &data);
        volts = comedi_to_phys(data, cr, maxdata);
        printf("%d %g\n", data, volts);
    }

    return 0;
}
```

Figure 4.2: Analog Capture Code

kernel had to be changed to a “vanilla” (unpatched) kernel. The system being used did not have a master hard drive for some unknown reason. This required that the new kernel be built and an initial ram disk (initrd) be created to tell the kernel that the root filesystem was on the slave device (hdb). Once the new kernel was loaded, the pci-das08 module had to be loaded. This automatically loaded the comedi module needed as a dependency.

The comedilib (libraries) package included several demos. One of these demos, tut2, was altered to allow the analog to be captured continuously through a loop. The code used to capture the data can be seen in Figure 4.2.

```
#include <stdlib.h>
#include <stdio.h>

int main( int argc, void **argv )
{
    short value;
    double throwout;
    char result = 1;

    while( result != EOF )
    {
        result = fscanf( stdin, "%d %lf\n", &value, &throwout );
        value = value - 2048;
        value = value * 16;
        fwrite( &value, sizeof(short), 1, stdout );
    }
}
```

Figure 4.3: Analog Capture to 16-bit signed PCM Conversion Code

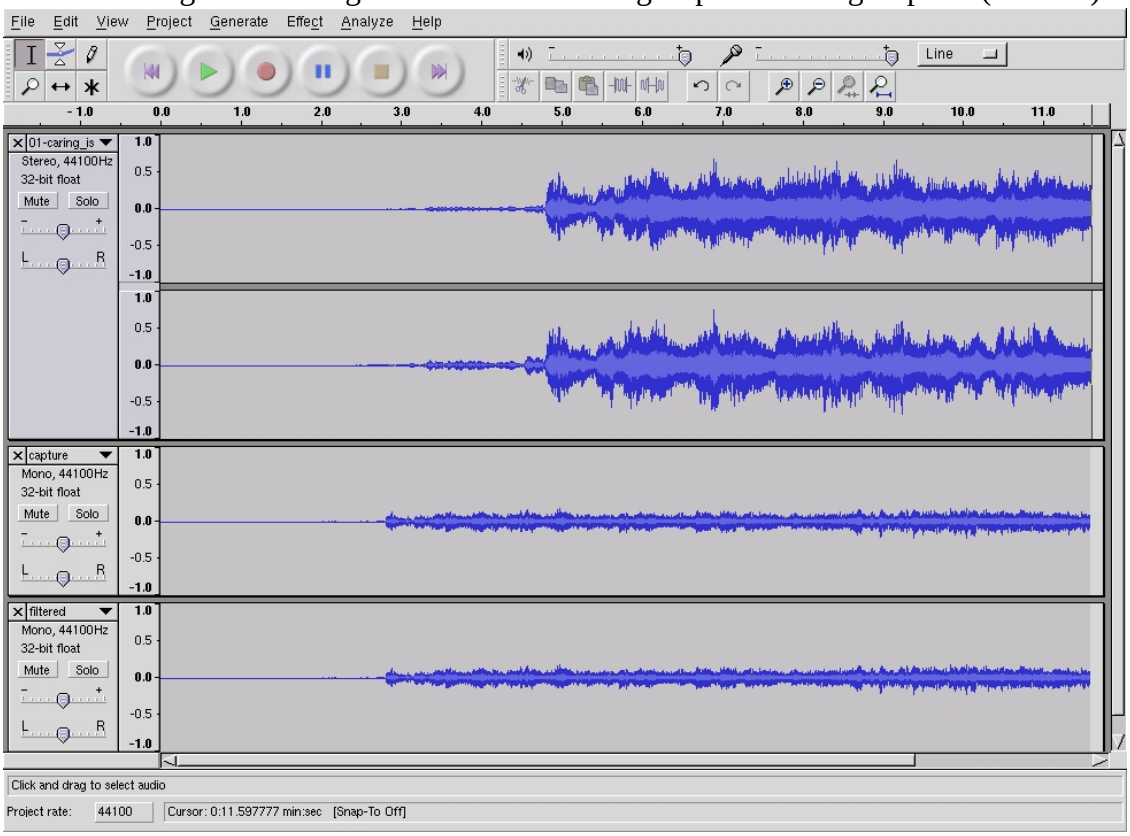
The output from the analog capture was piped into a file to be modified into a PCM data file. The analog capture was then piped into a program that would scan the ASCII analog capture file and convert the 12-bit data to signed 16-bit data. The code used to convert this analog capture file to a 16-bit signed PCM file is shown in Figure 4.3.

In addition to the capture and playback of the captured audio. Several of the chips considered also include a low-pass FIR interpolation filter and additional analog filtering. Example code that could be used as an FIR interpolation filter is shown in Figure 4.4. A low-pass filter offers the capability to limit the audio to only the frequencies below the Nyquist rate. The FIR filter requires coefficients to operate with its desired functionality. These coefficients were sought from Analog devices but were not available because they were considered proprietary. A comparison between original data and filtered, captured data can be seen in Figure 4.5 This sample shows a waveform from an mp3, its captured waveform, and the resulting waveform after a sample FIR filter was executed on it.

```
#define SAMPLE  short       /* define the type used for data
   samples */

SAMPLE fir(SAMPLE input, int ntaps, const double h[], SAMPLE z[])
{
    int ii;
    SAMPLE accum;

    /* store input at the beginning of the delay line */
    z[0] = input;

    /* calc FIR */
    accum = 0;
    for (ii = 0; ii < ntaps; ii++) {
        accum += h[ii] * z[ii];
    }

    /* shift delay line */
    for (ii = ntaps - 2; ii >= 0; ii--) {
        z[ii + 1] = z[ii];
    }

    return accum;
}
```

Figure 4.4: FIR Filter Code

Figure 4.5: Original Waveform/Analog Capture/Analog Capture (Filtered)



Chapter 5

Conclusion

The selected chip, Analog Devices' AD1851, could be successfully emulated in the specified embedded simulation environment. However, the input rate would be limited by the cycle speed of the environment. Traditional 44.1KHz audio data could be simulated but oversampled data rates of 8X and more of this data would not be possible due to the thread synchronization time constraints.

The AD1858, also by Analog Devices, could also be simulated using the embedded environment provided that more information could be found about the coefficients of the FIR filter, the sigma-delta modulator and leaving out the analog filtering. This emulation would be much more complex than emulation of the AD1851 due to the multiple input formats. It would also be more limited in input rate because it allows the input of stereo data. Oversampling rate would not affect the input rate of the AD1858 and therefore would be possible in a pin-level accurate emulation of the sound chip.

In addition to learning about how sound chips work internally and externally, a greater knowledge of following documentation and using scientific libraries in linux was gained.

Bibliography

[1] Analog Devices, Norwood, MA. *Datasheet for ad1851/ad1861*. URL http://www.analog.com/UploadedFiles/Data_Sheets/5722336011851_61.pdf.

[2] Analog Devices, Norwood, MA. *Datasheet for ad1858*. URL http://www.analog.com/UploadedFiles/Data_Sheets/200281268AD1857_8_0.pdf.

[3] Nigel Redmon. Digital audio: Oversampling. URL http://www.earlevel.com/Digital Audio/Oversampling.html.

[4] Texas Instruments. *Datasheet for DAC7742*. URL http://www-s.ti.com/sc/ds/dac7742.pdf.