Break the Xilence

Elliot Rosen | Jacob Nelson | Stephanie Mao

18-545: Advanced Digital Design Project

Fall 2014

Final Report

Table of Contents

Statement of Use

The members of this team, Elliot Rosen, Jacob Nelson, Stephanie Mao, hereby give permission for anyone to use the code they produced for this project in an academic, educational, or otherwise non-profit-generating manner as long as the original authors (the members of this team) are given credit for their work. If you have questions about the project, you can email us at:

Elliot Rosen: elliotr@andrew.cmu.edu
Jacob Nelson: jcnelson@andrew.cmu.edu
Stephanie Mao: stephanie.mym@gmail.com

The source code can be found at https://github.com/Lit0r/break-the-xilence

Introduction - Project Description

Our plan at the beginning

We proposed to make a hardware digital synthesizer. We wanted to build our own MIDI synthesizer, with all tone creation occurring in the FPGA fabric. This is in part due to the gratification of instant feedback when tweaking knobs and sliders along with a system that has been optimized for a single job. Ideally, we wanted to implement subtractive synthesis in order to create our tones, with envelopes controlling both amplitude and the cutoff frequency of a lowpass filter.

What we actually achieved

Though we did not accomplish all of our goals, we did succeed with many of them. We were, however, able to show off a fully-functional monophonic synthesizer for our public demo, as well as multiple sound manipulation features that could be adjusted with our parameter control unit. In order for this to happen, a number of pieces had to be successfully integrated. This included installing Xillinux, integrating our project with Xillybus, getting our audio codec working (the ADAU1761 is not an audio codec that anyone in the class could help us with), writing a functionally correct monophonic scheduler, designing and constructing our own parameter control unit, using PMODs to get data from the parameter control unit to the note banks, and reading MIDI input over a USB port. Additionally, since the envelopes and filter were unsuccessful, two additional audio effects were implemented as a substitute: the addition of a tremolo feature, with a knob controlling tremolo rate and/or disabling the effect; and replacing the square wave generator with a variable-width pulse wave generator, with the duty cycle controlled by a slider.

Development Tools Overview

Board

We used a Xilinx Zynq-7020 Zedboard for our project, primarily because the Zedboard has all the peripherals that we want, as well as hard processor cores built on the same chip. It also has a 3.5 mm stereo jack for sound output, and a USB-OTG port that we used to interface with the MIDI keyboard.

Xilinx Tools

We began the semester using Xilinx Vivado 2014.2 (and then 2014.3, briefly) to compile and synthesize our code. As the semester progressed we made the design choice to switch to Xilinx ISE 14.2 as Vivado 2014.3 (or 2014.2) was not supported by the Xillybus IP cores that we wanted to use to interface the on board ARM processor with the Zynq SoC. We also used the IP core generator provided by Xilinx quite frequently throughout the project.

Operating Systems

We used the lab machines with Red Hat Enterprise Linux installed as the host computers to communicate with the board. Half way through this semester, we discovered Xillinux, a complete, graphical, Ubuntu 12.04 LTS-based Linux distribution for the Zynq-7020 device as an alternative to the minimal linux distribution provided out of the box with the Zedboard. With Xillinux we were able to integrate between the device's FPGA logic fabric and plain user space applications running on the ARM processors using the Xillybus IP core, which simplifies our AXI connections.

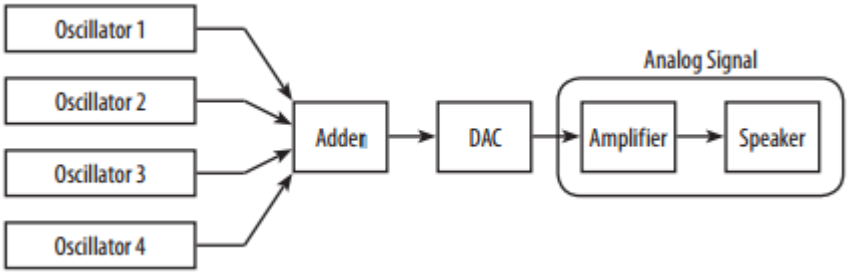
Version Control

We created a shared Git repository early on in the course and used to manage our code throughout the semester. We also spent a little bit of time saving old versions of code on people's desktops and reverting as need be. Manual version control like that is a bad idea, as often we had to keep one user logged in so we could work on that particular "branch", which got to be very annoying. We recommend *forcing* all team members to become familiar with the version control mechanism of choice (instead of assuming working knowledge), preventing confusion down the road when the codebase becomes complicated.

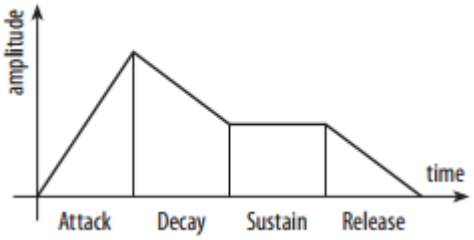
Background

Digital Sound Synthesis

A sound can be represented as a sum of elementary waveforms such as sine waves, sawtooth waves and square waves. The wave that we used in our sound synthesis is square wave, because a square wave is very easy to generate, and it only contains odd multiples of the fundamental frequency. (If we used low pass filters on a sine wave, they would only attenuate the sound without changing the timbre.)
In order to be able to reproduce multiple notes at once, we need multiple note generators, units that generate the desired waveforms. This involves a lot of computation that can be run in parallel, hence the hardware implementation. A general scheme of a synthesizer is presented in the following:



These waveforms can be added to form a single resulting sound.
For our digital synthesizer, we are modulating the amplitude by multiplying the signal with the envelope we designed. We are using one of the most used envelope model called ADSR - attack, decay, sustain, release, as shown in the following image.



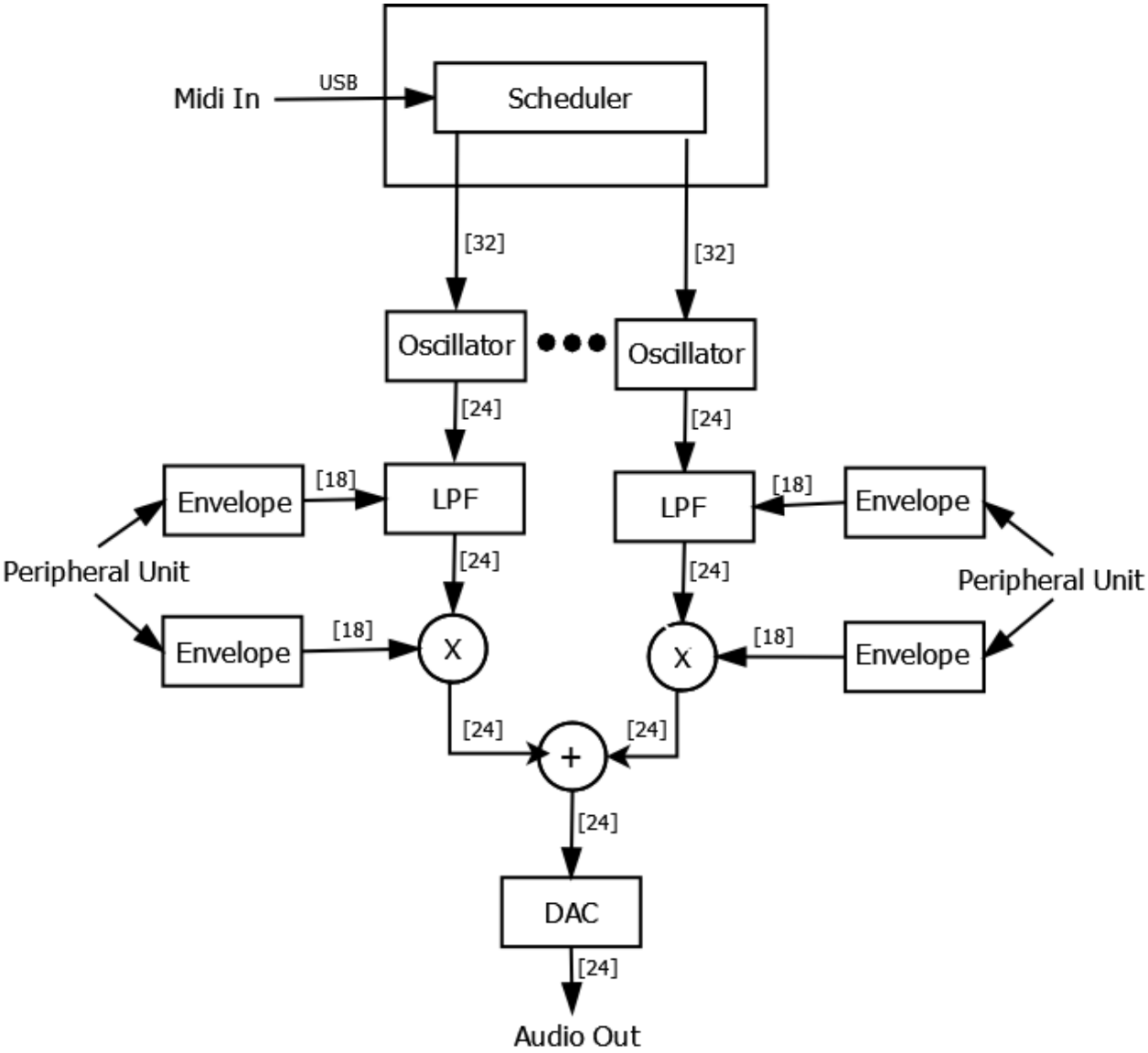
During the attack phase (corresponding to the MIDI “Note On” event), the amplitude of the sound is increased from zero to its maximum value. During decay, the amplitude is decreased from its maximum value to the value at which it will remain constant during the sustain phase until a MIDI “Note Off” event is received. During release the amplitude of the sound is decreased from the constant value to zero. The length and levels of phases are determined by peripheral control unit that we are going to build later during the project.

MIDI

MIDI (Musical Instrument Digital Interface) is a standard that allows communication between digital instruments, including synthesizers and computers, and other devices. Though it's been around since the early '80s, it's still widely used today. MIDI over USB, which our system uses, is popular as well. It allows for the transmission of MIDI protocol signals over a USB connection, allowing us to connect MIDI keyboards to our Zedboard.

Block Diagram

The scheduler in the block is written in Python and is run on the on board ARM Processor. Everything else not in the block is written in VHDL/Verilog on the programmable logic on the Zedboard.



Software Design

Scheduler

Overview

The scheduler runs on the on board ARM Processor. It takes in MIDI events from the keyboard over the USB OTG port on the board. It will then interpret those events and map them to the available note banks on the FPGA. Then it will send data containing the note to be played and the note bank to play it or the note bank to be turned off if it was a note on or note off even respectively.

Process

The scheduler was a very dynamic component of our project. Much of the reason for this lies in the fact that it was written in software and therefore easily testable and demonstrably correct. Also, it encountered a few roadblocks that required fairly large design changes along the way.

Our first problem was finding a way to actually interact with the board. We began the project with the goal of using the RTmidi library provided as open source for academic endeavours by McGill University to process the MIDI events and send them forward. We went to implement this and came to a really unfortunate roadblock that we spent much too much time trying to pound through. The out of the box OS that came on the SD card given with the ZedBoard would recognize our MIDI keyboard as a device but would not recognize it as an input or output device no matter what we would do. We spent almost three weeks playing with ALSA and Jackd libraries trying to find some way to get the processor to accept the MIDI controller as an input device. Halfway through this process we had decided that we were going to use Xillinux (a Linux distro specifically designed to be run on Xilinx FPGAs) and Xillybus to create easier AXI connections. After completing this transition we hoped that our problem would go away, but sadly it did not. After trying to hack the kernel for a bit we made the decision that it was probably not going to work and we needed to take a new look at the problem. At this point we managed to find an input bytecode stream that would display the bytecode output from the MIDI keyboard. It was located at /dev/snd/midiC1D0 and we decided to write our own software to interpret that bytecode. This ended up being the final version of our scheduler and worked fairly well for our final project. We never got to figure out what the problem was which was kind of frustration in the end.

Results

We got to write our own scheduler interpreting bytecode from an input stream which was pretty cool. It is capable of scheduling for both monophonic and polyphonic synthesizers but

unfortunately we never got the polyphonic hardware perfectly set up. It would read events from both the MIDI controller and the hardware and perform all of the necessary scheduling using those two streams.

Hardware Design

Peripheral Control Unit

Overview

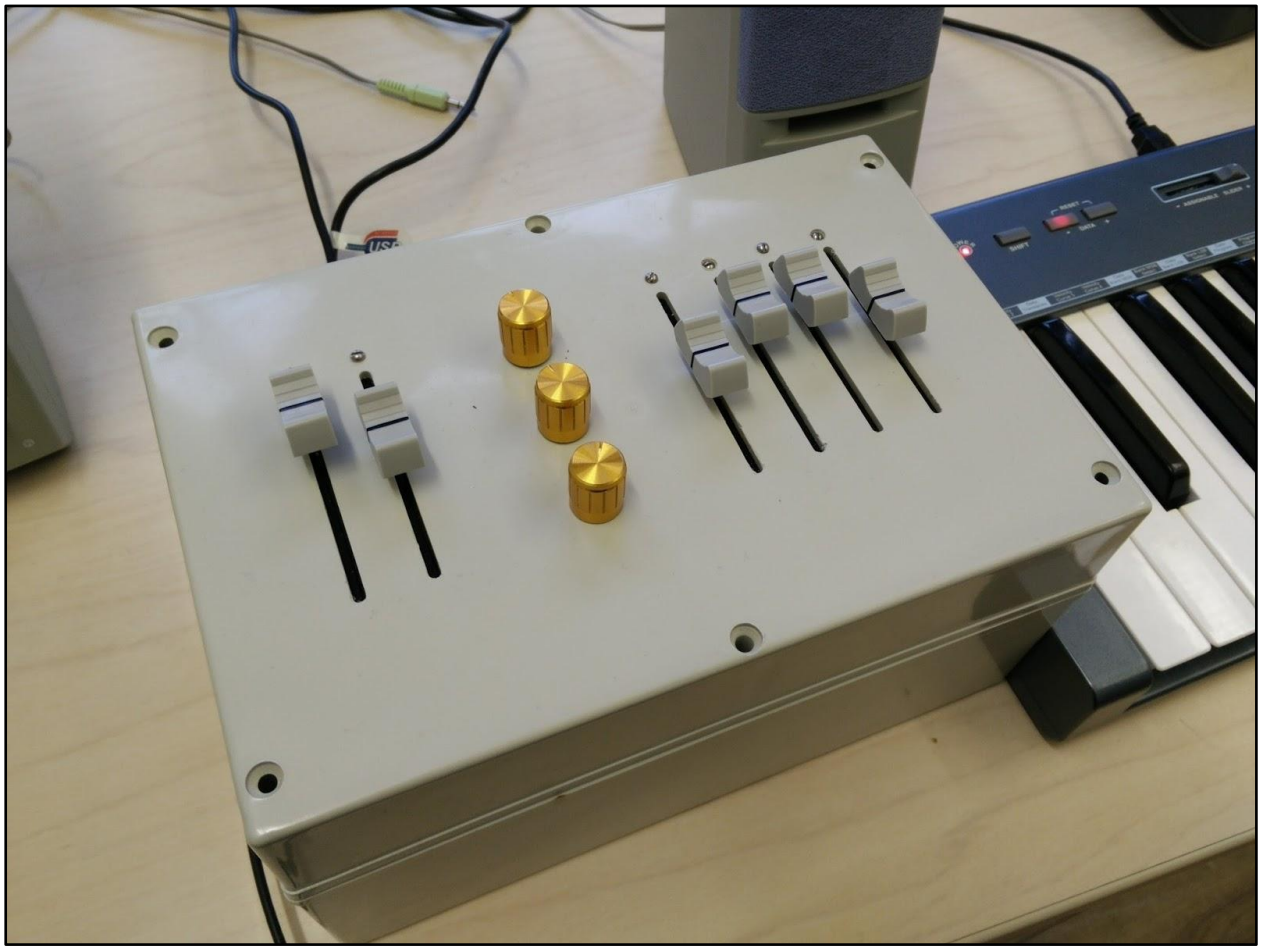
The peripheral control unit is designed to alternate the features of the envelope, controlling the length and the amplitude of each envelope phase.

Process

Initially, we decided that we wanted to be able to control the length of attack, decay and release phases, the maximum amplitude of the attack phase (the start of decay phase), and the amplitude of the sustain phase (the end of decay phase).
Since we used most of the I/O interface on the Zedboard except for PMODs interface, we decided to give it a try. We also realized that we would need some analog-to-digital converter in order for the control unit to communicate with the Zedboard. At one point, we were thinking about using the knobs and sliders that are already built-in on the keyboard instead of making our own control unit because we were not sure how complicated it would be to build the unit and interface it with the board. After talking to Professor Nace, we decided to stick with our original plan. We were able to find a Digilent PMODs called PmodAD2 that contains 4 analog to digital converter channels that has an I2C interface. We then obtained them online and some rotary/slider potentiometers to serve as analog input to the PMODs. We also found some sample demo project on the Digilent website, so we just modified the code provided and were able to make it work in the end.

Results

In the final project demo, although we were not able to use the control unit to alternate the features of the envelope, we were able to use the knobs and sliders to control the tremolo effect and the duty cycle of each note played.



Oscillator

Overview

Oscillators are a fairly straightforward piece of the project but also incredibly instrumental. They are the basis of our sound creation. The oscillator is the first piece of each note bank which is then manipulated by all the other elements until it is finally outputted. We began with implementing a simple square wave and then moved on to sawtooth and pulse waves. These waveforms were chosen due to their harmonic-rich properties that will make them ideal for use in subtractive synthesis. (square waves contain all odd integer harmonics, while the sawtooth wave contains all integer harmonics)

Process

We began by implementing a simple pulse square wave to make sure our implementation of an interface with the audio codec was functionally correct. After we found this to be working we created a couple of different square pulse waves with differing frequencies that you could control with the switches on the board. When we got this to work we created a dynamically reconfigurable frequency square wave using input from our scheduler. We then moved forward

and attempted to implement the more complex sawtooth wave. We got this implemented but never working perfectly. We never got enough time to entirely fix them as we ended up devoting all of our debugging time on the envelopes. This is something that given more time we would like to get working. At the end of the semester we implemented some effects that allowed us to manipulate the pulse width of the square wave to demonstrate our ability to use input from the peripheral control unit to manipulate the sound.

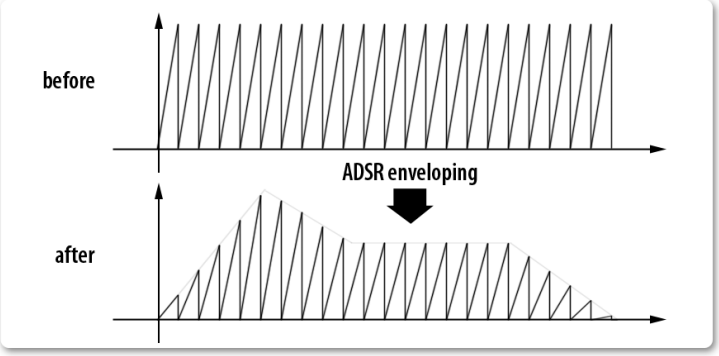
Results

At the end of the project we were able to demonstrate square pulse waves with manipulatable pulse widths. Additionally, square waves were successful, and saw waves were mostly successful, although not fully tested.

Envelope

Overview

The envelopes were arguably the most essential part of making our project a success. As explained in the background section, the ADSR envelopes will control the volumes and cutoff frequencies for the multiplier and low pass filter stages. Their timing will be determined by the parameter control unit, and they will be activated by the scheduler. Without these being implemented properly it is difficult to impossible to manipulate the sound the way we would have liked to. Below is an example of how the envelopes manipulate the oscillators to change the shape of the sound for anyone unfamiliar with signals. This operation is completed by multiplying the envelope by a sawtooth wave (in this illustration).



Process

We began by drafting up a complete design of our envelope generator state chart. After this one of our group members drafted up a rather simplistic system verilog implementation of this state chart. Unfortunately, when we switched to ISE from Vivado we found that ISE does not support system verilog despite claiming that it does. As a result we had to rewrite the module in verilog. This was not a huge effort, but system verilog offers many useful constructs that are not allowed

in verilog so it was rather unfortunate. Our first attempt was a rather straightforward mathematical approach at producing the correct shape for the envelope. When this was unsuccessful we thought that it might have been because we were unable to meet timing constraints for our design. Because of this we slowed our clock down and piplelined our envelope generator so that it would only perform one of its operations per cycle: a subtraction, division, multiplication, and addition. Eventually, when this didn't fully work we switched over to an envelope generator that used mathematical units entirely generated by the Xilinx IP core generator. We successfully got each of these implementations to behave perfectly in simulation in response to both note on and off events, but when it came time for it to be synthesized and placed on the board we were never capable of producing a perfect envelope. We believe that this was most likely due to the difficulties of implementing floating point divisions and multiplications in real life hardware.

Results

We were able to produce several versions of envelope generators that worked perfectly in simulation as well as being able to produce the attack, decay, and release stages properly but missing out on the sustain. However none of these worked in practice so we were unfortunately not able to demo our envelopes. This is definitely the spot where we wish we had started earlier and put more focus on. Some of our group did not realize the importance of them and it caused our final demo to be much more lackluster than it deserved to be.

Low Pass Filter

Overview

The low-pass filter stage is essential to altering the timbre of the generated tones. The lowpass filter is usually considered one of the simpler classes of filters. We wanted to take it a step further and implement a filter with a variable cutoff frequency, since filter sweeps sound really awesome, and can be helpful for us to give unique timbres to sounds. However, this means designing a variable-cutoff filter, which means a lot of math. We chose to implement a digital transformation of an analog second-order lowpass filter, having the transfer function in the frequency domain, for the sake of simplicity:

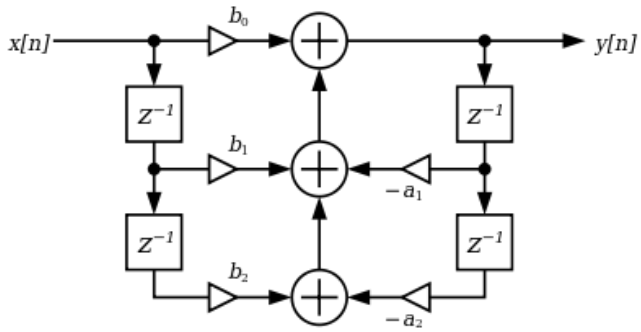
$$H(s) = \frac{\omega_0^2}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2}$$

Process

Analog filters are great, but we needed a digital implementation. Digital filter transfer functions look like this (with M and N at 2, since this is a second-order filter):

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}}$$

To implement that particular transfer function, we used a direct form 1 realization, which looks like this:



where z^{-1} represents a unit time delay, and the triangles are multiplication.
So what about those coefficients? Those have to be calculated on the fly as well. The below matlab code succinctly represents the coefficient calculation, as implemented in hardware.

```
Fs = 4800000 / 8;
Q = 4;
K = pi * f / Fs; % small-angle approximation for: tan(pi * f / Fs)
% actual coefficient calculator just gets K stright from parameter
% control unit, scaled by powers of 2.
K_div_Q = K/Q;
K_2 = K^2;
p = 1 / (1 + K_div_Q + K_2);

b0 = K_2 * p;
b1 = 2 * b0;
b2 = b0;

a1 = 2 * (K_2 - 1) * p;
a2 = (1 - K_div_Q + K_2) * p;


freqz(single([b0 b1 b2]), single([1 a1 a2]), 2^19, Fs)
ax = findall(gcf, 'Type', 'axes');
set(ax, 'XScale', 'log');
set(ax, 'XLim', [1 Fs/2]);
set(ax, 'YLim', [-100 50]);
```

Due to time pressure, this was implemented using floating point units, which took up a lot of space. In the future, a fixed-point representation would be preferable.

Results

This could not be debugged in time. Additionally, there might be other transformation methods that are more suitable. Future work will be exploring better ways to create adjustable filters in hardware.

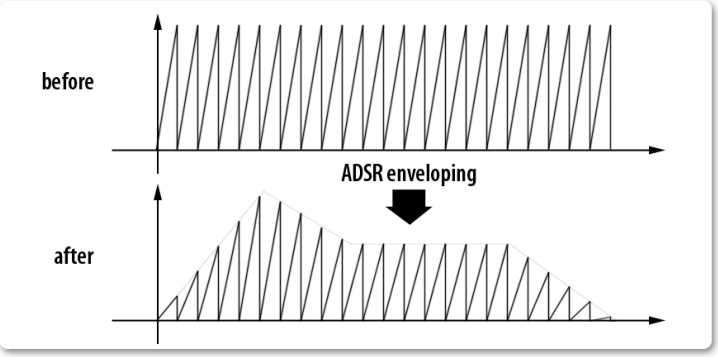
Amplitude Control

Overview

What sound device is complete without a volume control? The only thing better than a volume control is being able to control the volume with an envelope.

Process
This consists of a simple fixed-point multiplier. You have to hold onto the high bits and discard the low ones. This can be controlled by an ADSR envelope to shape the attack and decay of a note.

Results
The multipliers will multiply the envelope generated by the peripheral unit by the filtered square wave. This will create a varying, more realistic, and more pleasing sound out of a simple square wave. Here is an example of what that will look like on a sawtooth wave, the amplitude effects on a square wave will be the same:



Adder

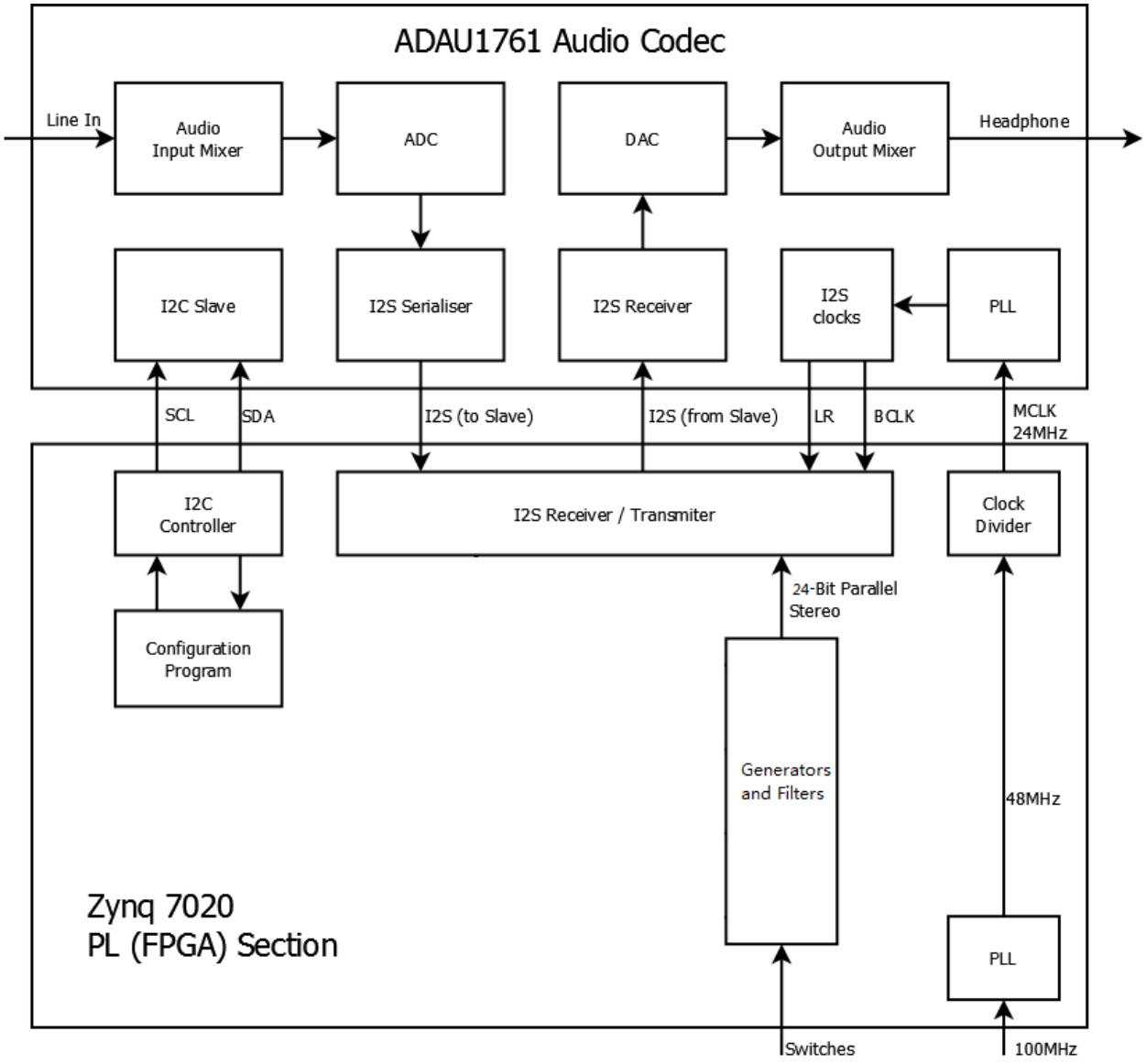
Overview
Not much to say here. We needed to sum up the outputs of each note bank in order to get all of the notes into a single audio channel.

Process
The particular hardware generated summed the note banks by accumulating each bank's output, which is less efficient than a tree-based approach for summing them. However, this was more difficult to generate in a parameterizable manner, which made it easier to alter the number of note banks generated.

Results
Audio generated from multiple sources gets combined into a single audio channel.

DAC

Overview
The Zedboard has ADAU1761 codec which uses an I2S interface. ADAU1761 is a stereo audio with integrated digital audio processing that supports stereo 48 kHz record and playback. The DAC we used is a built-in part on the chip. Please see the following for the block design of a simple interaction between the ADAU1761 Audio Codec and the Zedboard.



The generated signals from other programmable logic will be passed from the I2S transmitter to the I2S receiver on the codec. Then it will be feed to the DAC on chip and an audio output mixer will output the sound through the speakers connected to the Zedboard.

Process

For lab three we were asked to mess around with the Codec 97 on the FPGA. Unfortunately the Zeboard doesn't have AC97, so at the beginning we had some trouble figuring out how to get the audio working. However, we were lucky enough to find a project online that helped us understand and use the audio side of the Zedboard without using the Linux stack, since we wanted to connect directly to it from the programmable logic. The project that we found was able

to take audio in from Line In and apply some basic filters, then output through Headphone Out. We used the code from this project for configuring the ADAU1761 and modified it to take the input from other parts of the programmable logic and then output through Headphone Out.

Results

We were able to configured the ADAU1761 and output sound from the Zedboard through speakers connected to Headphone Out.

Clocking

The top level interface with the peripheral control unit was done through three pmod controllers, each one instantiated in the top level module of our design. It used a 100 MHz clock and two dividers to interface with the pmods. Internally it had a 100 KHz clock to send and receive signals over the sda and scl ports at the right speed. It also had a 100 Hz clock to switch the potentiometer that it was reading from as each pmod had four potentiometers to read from.

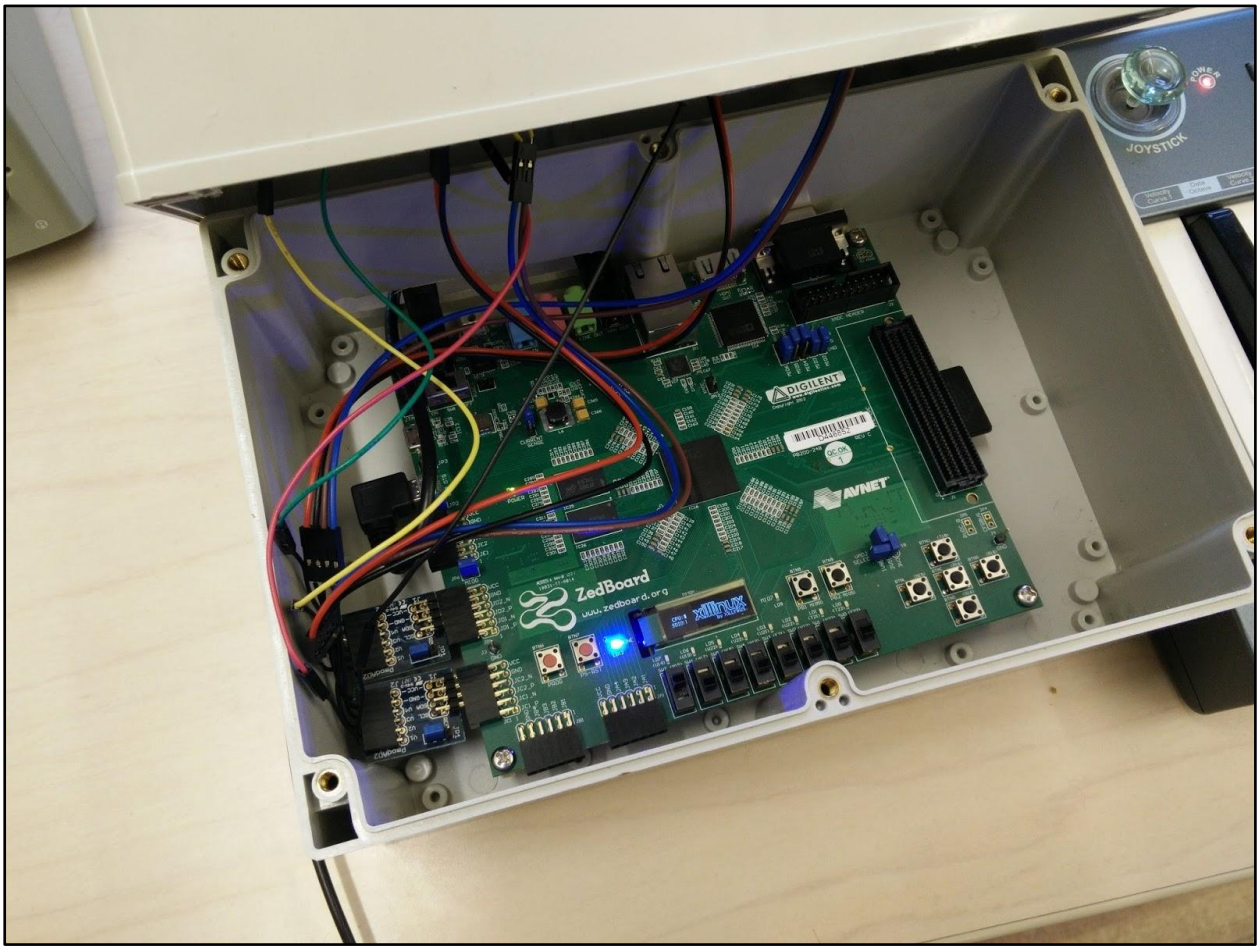
According to the HamsterWorks wiki, the hardware related to the ADAU1761 was clocked as follows:
“The internal logic of the ADAU1761 needs a core clock of 1024 times the sampling frequency. For 48kHz audio this work out to be 49.152MHz. To help with implementing this the codec chip has an PLL that can be used to generate a core clock from a external reference (MCLK) which is between 8MHz and 27MHz.
For this design I'm going to go with MCLK of 24MHz. To generate this 24MHz clock I'll need to take the PL clock of 100MHz, and then within the FPGA use that to generate 48MHz for the FPGA design to use. I'll divide this by two and send it out to the Codec.
By setting the MS bit of R15 to '1' the ADAU1761 can be told to act as master, so it will supply the I2S_CLK and I2S_LR signals back to the design running in the FPGA.
The pin that receives the I2S_CLK signal is collected to is not one of the Zynq's preferred clock inputs, I'm oversampling it at 48kHz, and then look for the edges where I need to process the I2S_Data signals. This has the benefit of keeping the FPGA logic all in one clocking domain.”

We utilized the 48 MHz clock for clocking our tone generation hardware, as well as a 4.8 MHz clock to use for when arithmetic logic generated outside of IP cores lengthened our critical paths. We moved the 48 MHz clock out of his code and into our own design, redoing the clock division for our 4.8MHz clock.

Testing/Verification Methodology

Testing was a very difficult thing for our project. Most groups in this class will first get their designs to work in simulation and then debug them on the board using Chipscope, which reads in signals over the JTAG port. Unfortunately due to the nature of our project we required the ability of booting from the SD card to run our project and there were three jumpers on the Zedboard that need to be placed one way to boot from the SD card and a different way to enable the JTAG port. As such we were forced to choose between having a runnable project that wasn't runnable and having a testable project that wasn't runnable. This put a serious hurdle in our path as far as testing was concerned. To further hinder our testing abilities, the OS that we chose took for itself all of the switches and LEDs on the board so we could not use them either. As a result of these two design decisions we were required to test in rather creative ways. The bulk of our tests therefore came in manipulating the wave that would be output in different ways depending on the state of different variables and then viewing the output on the oscilloscope. As silly as this testing methodology sounds it actually allowed us to find and fix a good number of bugs in our system. Overall, however we were not as able as we would have liked to create a good testing harness for our system.

Status and Future Work



The current status of this project can be seen in the following youtube clip which was taken on the day of the final public demo. The parameter control unit, monophonic scheduler, MIDI over USB interface, processor-logic connection, tone generators, adjustable effects (tremolo and duty cycle), and audio codec were all functional for our public demo.

Video link:

http://www.youtube.com/watch?v=6N6woWF4siU

Future Work

We were rather dissatisfied with the tools that we were given to do this project which will most likely stop us from continuing to work on it. Although a few of the group members are still very interested in fielding a complete result from this experience the effort that is required to port it over to an environment that we would like more may prove too great for us. It is also possible that we will continue our work in ISE but it is more likely that the interested members will stay as far away from ISE as they can in the future, and consider other hardware options as well.

Group Thoughts

What we wish we had known

- The board you select and the research that you do in the first week is incredibly important to the successes and failures of your project down the road. We were told this but did not pay enough attention to it.

- While talking to the TAs after the final demo we found out that one of their specialties was DSP work. Had we known that earlier in the semester we would have been able to ask many more questions instead of assuming that since our problems were often outside the scope of using the hardware the TA would have still known a lot about it. Ultimately it would be nice to have an introduction to the TAs early in the course, especially regarding their particular skill sets.

- A well defined project makes it easier to get started.

Good decisions

We believe that we made very few decisions that were inherently and totally good, but here are a few of the decisions that we believe were most helpful to us:

- Choosing an out of the box, fun project that we could get excited about. This was something that can go really well or really poorly. We think it was one of the better decisions that we made as none of us got very excited at the thought of making a video game.

- Ditching RTmidi and writing our own bytecode interpreter. Sometimes you have to not be afraid to throw away something that is holding you back and learn to do it another way. We held on to RTmidi for way too long and finally throwing it away was a good thing.

- Making a visually appealing case for our project that showcased the real hardware that we built. This made our demo much better. It allowed us to show that our only real problem with the project was buggy envelopes and that everything else was actually really great.

Bad decisions

Bad decisions were much easier for our group to come up with. Here are a couple things to avoid:

- Our board choice and high level design decisions basically hamstrung our testing efforts. We were unable to do anything besides behavioral simulation and full bitstream generation (without being able to use tools like chipscope at the same time) which created big problems for us in the debugging phase.

- We did not have our entire toolchain lined up before we started to work. As a result we had to do things like switch from Vivado to ISE which slowed down our project.

- We did not always hold all group members to deadlines and standards of results. As a result some group members may have gotten by with doing little to no actual work for the project. This was not addressed early enough.

- We picked a project very math intensive without having the necessary knowledge for it. This is by no means innately bad. It just requires a lot of communication with the instructor that our project requires a lot of research and learning that may not be easily evident in the demo.

- Not expecting and leaving enough time for problems. Around the third week we were really ahead of the schedule that we had set for ourselves and then eased off the throttle a little bit. Had we continued that pace we might have had more to demo. We also may have burned out, but now we have no way of knowing.

Words of wisdom

- If you're going to go off the beaten path, have a well-defined, fairly closed-ended idea. It makes it easier to get started, and to exit the concept phase sooner.

- Lay out your entire toolchain and design before you begin the actual coding. It will make integration and coding so much more seamless.

- Be sure to set a good schedule and make sure to follow it. Do not be afraid to communicate with the TAs or instructor about how that is going, including your own or others failures.

- Don't be afraid to throw something away and take a different approach at it if it is causing you way more problems than it should. This was a really bad thing for our project that cost us valuable debugging time.

Sources

This section contains a list of sources that we found useful throughout the implementation of our project.

Zedboard Audio Project by Hamster (Michael Field)

This is a project that we found online which is written by Michael Field. Basically, the project helps us understand and use the audio side of the Zedboard without using the full Linux stack, since we want to connect directly to it from the programmable logic. Since the Zedboard doesn't have AC97, we had some trouble figuring out how to get the audio working on Zedboard. The project that Michael wrote is able to take audio in from Line In and apply some basic filters, then output through Headphone Out. We used his code for configuring the ADAU1761.
http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_Audio

Digital Synth Report

This is a similar project to create a digital synthesizer on a FPGA. They had slightly different goals and designs than us but many of the overlaps were incredibly helpful for us.
http://www.digilentinc.com/events/Common/Report%20Sample%20~%20DigitalSynth%20Report.pdf

PMODs demo

This is the demo project that we found online that helped us configured the PmodAD2. The project itself is made for Xilinx ISE 13.4, targets the Nexys3, and is written in VHDL. We used the code in this project and did some modification to make it work on the Zedboard. The whole project can be found in the following link:
http://www.digilentinc.com/Data/Documents/Demonstration%20Project/PmodAD2_DA1_CSE_Demo.zip

Individual Comments

Elliot Rosen

What I did and time spent

I functioned primarily as the project manager for this, though I had my hands in pretty much every aspect of the project *except* for the PMOD interface.

I hate to give myself this much credit, but I was the driving force behind the idea of this project, and took it upon myself to convince my (rightfully) skeptical teammates that we should stray off of the beaten path with regards to our project. This meant conceiving much of the plan myself, and pointing the others to the resources they needed to learn about what we were going to do. This was a challenge, as this project had the potential to be far too open-ended for the scope of this class. This meant that most of the concept phase was my responsibility, and that exiting this phase was my responsibility as well. My other tasks related to this were designing the parameter control unit, determining the feature set of the synthesizer, and laying out (somewhat informal) functional specifications for the behavior of various units, such as the schedulers and envelope parameters, and explaining exactly why they were to be that way.

Another contribution was implementing, reimplementing, and reimplementing the envelopes again. I revised the state machine, adjusted the arithmetic, pipelined the design, integrated fixed point arithmetic units, and helped test the unit in its many phases. I rewrote most of the design at least twice in order to convince ISE to correctly synthesize the design, despite my behavioral simulations working exactly as intended.

My other main contribution was my work on the filter section, which meant both designing it and implementing it, as well as debugging it, which I was unable to complete. There were a lot of design decisions made that no one else could help me with, since none of us have a background in signal processing, but I was pushing for this project, so I took on the filter design on my own.

Lastly, I conceived of and implemented the tremolo and adjustable pulse wave features which we used to demonstrate the functionality of our parameter control unit. They were written when it became clear that the envelopes and filters would not be working by demo day.

For the sake of my mental health, I am not going to attempt to calculate the number of hours I spent on this project, but I'd estimate it's easily in excess of 300, probably closer to (or beyond) 400.

Class impressions/improvements/complaints

I liked the availability of resources and help, and felt the course went pretty smoothly (in terms of the overall course management. Our project was not smooth.) I felt our group, since we had such a different project, was difficult to hold accountable for our progress, since our team (or just myself) was the only one with a clear idea of what our project entailed. It'll be better for nontraditional 545 project teams to be held to a higher standard of clarity, which could help avoid some of the pitfalls we encountered. My only real complaint about the class itself was that the labs felt somewhat vaguely specified. Additionally, acting as a team leader can be difficult. I tried to lead by example when it came to putting in hours, but unfortunately Compiler Design consumed so much of my time during most of the semester. I'll admit I set a bad example for the rest of the group. With great freedom of project choice comes great responsibility. I also felt

like I was cleaning up after others at various points throughout the semester, whether proofreading presentation/report materials or code. This, I felt, was often in spite of repeated hand-holding. On an unrelated note, I really like those new oscilloscopes! I learned a lot from this class, although they weren't the lessons I wanted to learn, and this wasn't quite the matter in which I wanted to learn them. This was a valuable experience in terms of learning how to manage a team, and just work with a team in general. I wish I had more positive lessons to take away from this, although I did enjoy our final product. In any case, I know what mistakes not to make down the line.

Jacob Nelson

What I did and time spent

My contributions to this project were very diverse. I touched everything but the filtering in a meaningful way. My main two responsibilities, which I finished early, were the interfaces, both at the peripheral control unit and the keyboard to software interface. I implemented and debugged the interface over the I2C bus with the pmod controllers so that we could get our peripheral control unit running. I also took the layout design from Elliot and brought the enclosure to the machine shop to be machined with the help of a MechE friend of mine. I also worked on the interface with the MIDI controller. I did all of the debugging and design on the software side. In the end I wrote the scheduler that interpreted the bytecode given to us by the controller. After completing these two interfaces I worked on debugging the envelopes and the interface between the hardware and the software. The final two weeks of the course almost entirely consisted of Elliot and I sitting in lab working through different ways to try and fix the envelopes for about fifteen hours a day. I also spent the first few weeks working closely with the rest of the team on getting an interface with the codec set up so that we could produce sound for labs two and three. All in all I spent between, 200 and 250 hours on this project. Those hours are a semi-rough estimate though as I was not really keeping track until Thanksgiving break.

Class impressions/improvements/complaints

Overall I really enjoyed this class and it was an incredibly rewarding experience for me. I especially appreciated the freedom that we were given by the instructors to do a project that combined not only the discipline of digital design but also that of signal processing and embedded system design. It would have been easy for our project to have been denied on the grounds that its digital design concepts were not as complex as some of the other groups but we were encouraged to pursue something interdisciplinary which we greatly appreciated. One thing that can definitely be improved about this class is the labs. From my discussions with the other teams, none of us benefited too greatly from them. It would be great if we were offered a little clearer direction or instruction at that specific point. Other than that this semester was an intensely valuable experience for me and I greatly appreciate the opportunity to have participated with this team.

Stephanie Mao

What I did and time spent

My responsibilities were really diverse. At the beginning of the semester, I was responsible for understanding the Hamster project that we found online, since none of us knew VHDL before, I took on the task to learn VHDL myself so that we could better understand the code and how the clocking works in that project. After that, I spent some time figuring out how the file system work on Zedboard. At that time we were still thinking about having RTmidi run on the ARM core. I was able to connect the board through Ethernet, but after that we decided to switch to another approach which was getting Xillybus working on the board, so I switched task to work on getting Xillybus on the board. At the middle of the semester, I was also responsible for making the design review presentation and the design review report. After mid-semester, I was responsible for designing and implementing the external peripheral control unit. Although my original design was discarded at first, we eventually still took my research results and ordered the parts that I found online. I was also responsible for writing the first draft of the ADSR envelope in SystemVerilog, which was fully simulated and tested using VCS. After the parts that we ordered arrived, I started working on the interface between the PMODs unit and the board. I built the first version of the control unit using 4 rotary potentiometers, and then over the thanksgiving break me and Jacob were able to successfully implemented and debugged the interface over the I2C bus with the pmod controllers so that we could get our peripheral control unit running. After thanksgiving, I had some mental breakdown, so I wasn't able to contribute to the project as much as I could. My last responsibility was to make the poster for our public demo.
I definitely spent more time time on the project before and during Thanksgiving break, but not as much as my other team members after the break. I logged most of my hours and they came to a total of > 180 hours. I only started logging my hours at the end of September, so for the first month I only counted the class time.

Class impressions/improvements/complaints

Overall this was a great class. It was my first time experiencing building a project right from scratch that did not involve too much "hand-holding". I think I learned a lot about how to work in teams and how to stay motivated and keep pushing my limit when I didn't feel like doing so. I also learned and got to experience the project development process, especially the concept phase and refinement phase.

I was very worried at the beginning of the course when our team decided to implement something out of the box on the Zedboard. I also knew that I'm not really good at signal stuff so I doubted that I would be able to contribute much to this project. However, I didn't speak up. Later on, this became a problem for me because I realized it was extremely hard for me to work on something I don't really have confidence in. Later on during this semester, after talking to my advisor and several other friends in this field, I learned that this kind of situation actually happens a lot in the real world. So I convinced myself to stick with this project and tried my very best to help my teammates. However, something pretty drastic (personal) happened to me in the middle of November, which made me feel like I was a helpless member in this team. I really appreciated the help and understanding from Processor Nace at that time. I wish that I had communicated with my teammates and course staff earlier in the course about my concerns. I

also wish that I had spent time more wisely. If there was one thing I'd change, it would be spending more time on the project earlier in the course, and really develop a passion for the idea that our team was working on.

Acknowledgments

There are many people who contributed to the (somewhat) success of this project. We would like to thank Professor Nace as well as the TAs, Joe and Mark, for all of the help and guidance that they gave us throughout the semester. We would also like to thank Professor Don Thomas for providing us with an extra ZedBoard when it seemed as if all was lost. Also, Professor Tom Sullivan for his guidance in the realm of audio and signal processing. Lastly, we would like to thank the rest of the students in the course, without whom the many hours working in lab would have driven us entirely insane instead of mostly insane.