Music DSP with a SHARC Processor

Justin Sconza Spencer Walters Mark Wudtke

December 11, 2017

1 Introduction

This project is a continuation from Spring 2017. At the end of last semester, we successfully configured the board for audio throughput but were unable to move beyond integer operations on the audio samples. Over the summer, we discovered how to properly format the input, enabling us to perform floating point computations on the samples. Once we figured this out, we implemented FIR and IIR filters. We also constructed a daughterboard for sampling multiple analog potentiometers through the use of a multiplexer and an ADC. The SHARC processor used for this project is the ADSP-21489. The code is written in `C`.

For the interested reader who wishes to explore the depths of the concepts discussed here, we advise viewing the companion files. The companion files contain the code to implement the project discussed below.

1.1 Delay

This semester, we spent the majority of our time developing a good delay algorithm. In the process, we discovered many things that did and did not work, which we will discuss below.

1.2 Chorus

By learning how to control the delay time with a low-frequency oscillator, we got a chorus unit for free since it is essentially a delay effect with faster but *modulating* delay time.

2 Final Results

In this section we will describe everything that worked.

2.1 Sampling a Potentiometer

The following schematic is of the daughterboard we built to sample analog voltage on rotary potentiometers.

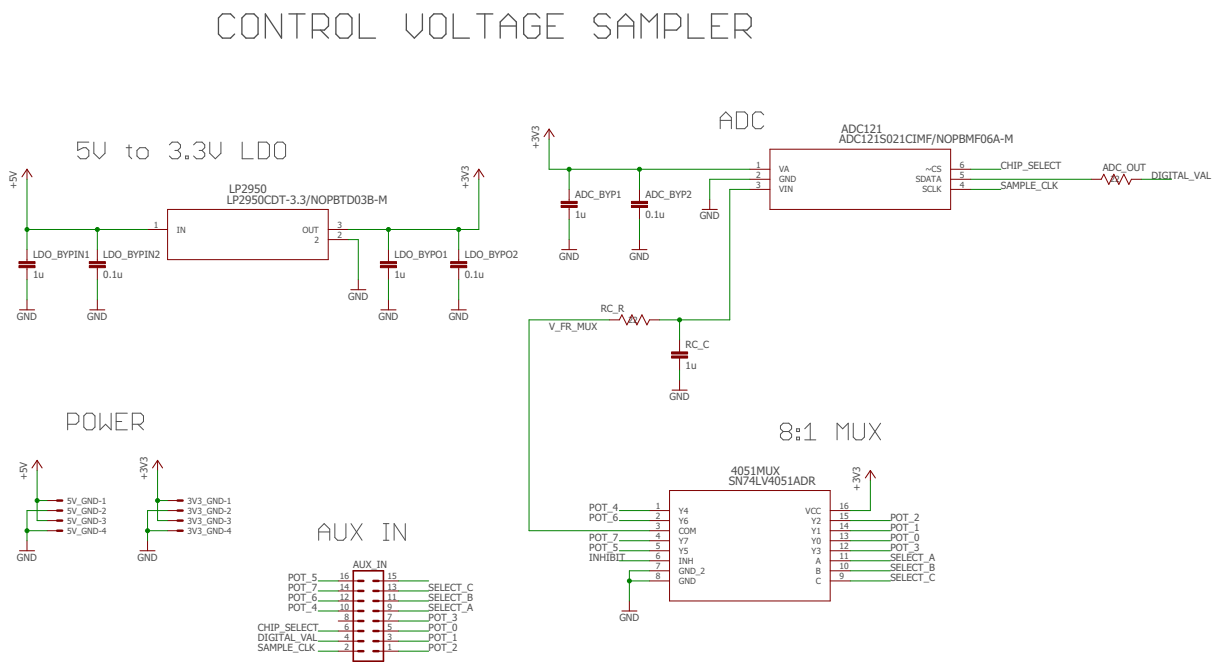

Figure 1: Daughterboard Schematic

As illustrated in the above schematic, there can be up to eight analog voltages input to the mux. The sampling of each input is perceived as being done in parallel by rapidly changing the channel select lines on the mux, a process commonly known as scanning. Not shown in the schematic is a counter, which performs the actual channel selection via a software based pulse. Each time the mux channel is changed, the ADC is instructed to begin sampling whatever voltage appears on the COM line.

2.1.1 Slewing

Nature is continuous; the ADC is not. The problem is how to emulate the continuous behavior of changing values on an analog potentiometer in the discrete world. For example, if at one ping to the ADC, a voltage corresponding to a discrete value of 7 is read by the ADC and one ping later, a value of 22 is read, in order to recover the data between "snapshots," we must slew.

In order to accomplish this, we take those two values of 7 and 22, divide the length of time between them into a discrete number of steps, such that the destination is reached in an integer number of audio sample clocks. We have empirically discovered that 20 ms is a good length of time for this set duration, which we will call T_{read}. From this, using a unit conversion to samples, the number of steps is $N_{read} = F_s \times T_{read}$. In mathematical terms, this is expressed as follows:

$$d[n] = \frac{d_{target} - d_{old}}{N_{read}}\ n + d_{old}\ , \tag{1}$$

where using the above example, d_{target} is 22, d_{old} is 7, $n \in [0, N_{read}]$ and $d[n]$ is the value approaching 22 at time n.

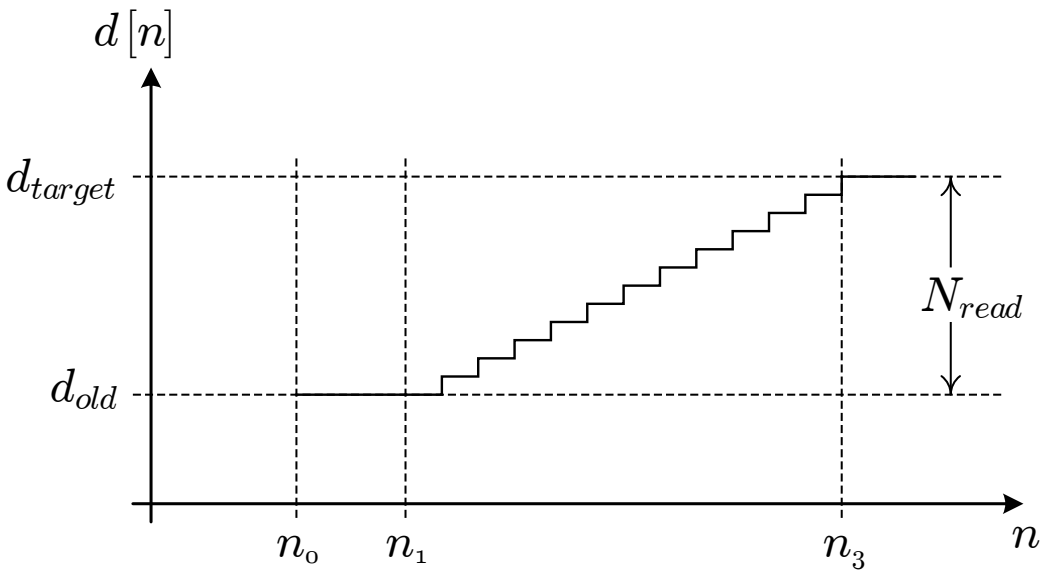

Figure 2: Slewing Visualized

2.1.2 Reading More Mux Channels

Even if we utilize more channels on our mux, we still want the time between ADC reads for a given channel to be the same. This leads to a new variable corresponding to the time between adjacent mux channel reads by the ADC, t_{toggle}.

Mathematically speaking, this relationship is as follows:

$$t_{toggle} = \frac{T_{read}}{N_{pots}}, \tag{2}$$

where $N_{pots} \in [1, \ldots, 8]$.

Whenever the mux channel is changed, the ADC is pinged for a new value. This pinging occurs every N_{toggle} sample clocks, where $N_{toggle} = \lfloor F_s \times t_{toggle} \rfloor$.

2.2 Limiter & IIR Analog Distortion Emulation

When implementing feedback for our delay, we noticed some digital distortion artifacts. To mitigate this, we implemented a limiter function coupled with an IIR filter, and in the process achieved a convincing emulation of analog distortion. The details can be found in [2].

2.3 Low Frequency Oscillator

In order to implement real-time modulation in the delay time parameter, we designed a simple LFO.

2.3.1 Range Compression

We designed our delay function to correctly handle "safe" values for delay buffer indexing, but did not account for the LFO modulation adding to that index. As a small example, imagine we have a delay buffer of length 8 (indexed from 0 to 7). The current delay index determined by the delay function is 6. If the amplitude of the LFO is 3, after being applied to the current value, the new delay index will be 9, which causes a runtime error.

To get around this, we make room at the top and bottom of the delay buffer corresponding to the LFO amplitude.

2.3.2 Sine, Cosine Assembly Code

At first, we used the `math.h` sine function in computing our LFO. Professor Haken suggested that we use the `SinCos.asm` assembly routine he wrote for the SHARC processor, since it is significantly quicker than `math.h`. Although not entirely necessary at this stage in our project, it leaves more processing power free for other DSP in the future.

2.4 Button

Our button is for tap tempo delay time. Dealing with button information is fundamentally different from continuous control signals like potentiometer input.

2.4.1 Handling Button Events

The button is active low, so we must detect falling edges. In order to do this, we need memory of the previous sample. If it is low now and used to be high, we have detected a falling edge. In any other situation, we have not.

We keep track of the running average of times between pushes using the system clock. This is weighted more heavily towards the most recently read time difference.

For more information, please see the `checkButton` function in `delay.c`.

2.5 Two Modes of Control

It is important to be able to control the delay time via the knob or the button, and switch between them seamlessly. To accomplish this, we implemented a basic state machine.

2.5.1 State Machine

The state machine has two states, one for the knob and one for the button. Either of these can be in control of the delay time parameter. Adding the button as a second mode of control led us to the problem of the delay time switching immediately. For example, if the knob is set to a value corresponding to a slow delay time, and the user enters a delay time that is much faster using the button, with no modification the code will jump immediately to the button value. We found this behavior to be musically undesirable. This was similar to the slewing problem from before, so we followed a similar model to alleviate the issue.

To ensure a continuous change between delay times on either mode of control, we set a threshold corresponding to the difference between the previous delay time and the new delay time. If the threshold is met, we slew between the two values. The amount of time that we slew is proportional to the difference between the new and previous delay times. The range of possible time values is from 20 ms to 2000 ms.

2.6 Flash

We followed the steps from a previous ECE395 semester to use SPI flash as the boot method for the SHARC processor.

3 Road Bumps

Some issues we encountered bear discussion.

3.1 Delay Dead-end

Our first delay algorithm was a creation of our own. We heard undesirable artifacts when changing the delay reach in real-time with feedback enabled. We tried to solve this issue by inventing a new delay algorithm, but we found that we kept reinventing the same algorithm in a different form. We found an established algorithm from an IEEE paper [1].

3.2 Slewing in the Wrong Place

In our original implementation of slewing, there were three problems with our slope updates. (1) We recomputed the slope every sample. (2) The new slope was dependent on the previous slope. (3) We were doing all of our slewing in the delay function instead of `main`.

This is how we resolved the issues.

1. Reading the potentiometers every 20 ms with t_{toggle} as in 2.1.1.
2. Using equation (1) as in 2.1.1.
3. Implementing slewing in `main`.

3.3 Capacitance Crosstalk

While sampling our potentiometers, we had a high amount of capacitance from the COM (ADC IN) to ground. Because of this, while switching between MUX input lines, the value would not settle between adjacent reads. We removed the capacitor and all was well.

References

[1] P. Anghelescu, S. Angelescu, and S. Ionita. Real-time audio effects with dsp algorithms and direct-sound. In *Proceedings of the 2014 6th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 35–40, Oct 2014.

[2] Bastian Bechtold. Real time signal processing in python.