**Music File Format**

I chose Wav files because they are uncompressed and very easy to work with. I wrote a short Matlab script to read in a file and truncate it down to the necessary mono channel, with eight bits per sample. The primary purpose of this is to make the file easier to work with using pulse width modulation, where the number of bits you need to convert has a huge impact on your final noise. It also reduces the file size, allowing you to get six and a half minutes of music into the 16 Mb ram chip, or thirteen minutes if you use both the cellular ram and the flash memory. Note that 8 bit wavs use unsigned numbers to represent data, rather than signed like wav files usually do.

**Memory**

Memory is pretty easy, Adept has the capabilities to load files into the chip memory, and read them back out again. It starts at address 0 and goes from there. The identifying information at the start of the file is short, and it’s not being interpreted anyway, so the memory controller starts at address 0 and doesn’t treat it any different from the normal data. Memory control simply uses the program code from the memory lab to read the memory 44100 times a second and load the samples into a register for PWM.

**Pulse Width Modulation**

This one is more a timing issue than anything else. I have an unsigned counter going from 0 to 255 and back down again, with a comparator outputting a 1 when the counter is lower than the sample, or 0 when higher. This produces a set of pulses with width proportional to the value at the input. At this point you send the pulses out of the fpga and to the filter. Note that the wave will be centered around (Vcc – Vground) /2, which leads to some design considerations for the final filter. Eight bit audio is used Out of sheer necessity. Sixteen bit would require a 2.8 GHz clock at the bare minimum, eight bit requires a minimum of 11 MHz, with higher frequencies leading to less noise.

**Filtering**

This is the most difficult part of the circuit to get working right. You can use a capacitor to average out the results of pwm, and you get a rough waveform out of it, with noise. With a good filter you can get a signal which closely duplicates the target waveform. A filter with hefty rolloff is a necessity. The filter is powered by the FPGA, and only has a voltage range from 0 – 3.3V. This necessitates the use of a virtual ground, which adds its own challenges, but has been as major an obstacle as inherent noise from the op-amp. I have attempted to use three stages for a low-pass filter, but it needs some serious math and optimization. The filter I used was pretty much the example filter on the Sallen-Key page on Wikipedia. The low voltage isn’t an issue for volume; when I was testing it off of a normal computer headphone out port and cheap earbuds, it was getting some pretty decent volume. Even with the noise, you can hook up a set of headphones and hear recognizable music coming out.

**Current issues**

***Wav file Noise***. There is more noise coming out of the headphones than speakers right now, and a great deal of it is coming from the wav file itself. Playing the eight bit wav file on a computer reveals a huge amount of background static, but it is strangely reduced in periods of the song with lots of new notes playing, and is greatest at points where notes are simply being held. Using sixteen bits for PWM is simply not practical, but it may be feasible to

***Volume and waveform amplitude***. There is a large amount of capacitance in the filter right now and the output waveform is not running through its full range. Reducing the capacitance on the filters would allow for a wider voltage range for signal processing and before boosting, which would help reduce noise. Boosting the volume on a clean signal should be fairly straightforward, but will be easier with a clean signal.

***Distortion***. The piano sounds more like electric guitar at this point. The problem seems to be coming from too low a corner frequency on the filter.

***Noise dependent on FPGA***. I tested the circuit using two different FPGA’s, and they both had slightly different noise. Part of it might have been the power source, as one was powered from a wall adapter and the other by USB port. Need to see what sort of effects it has. It seems that running the seven segment display at the same time as audio increases noise.

***FPGA output distortion***. The signal coming out of the FPGA often has some sine-wave like noise on the tops of the square waves. It may be a result of an older FPGA, it might be a transmission line effect, it might be something else. The waves themselves are a high enough frequency that the low-pass filter should remove them fairly easily, but the effects of this distortion on the final output wave need to be determined.

***Clock mismatch***. This may or may not be contributing to noise. Right now the memory is pulling in 44100 samples a second, but the pwm is making 98039 complete strobes up and down per second, leading to new information being inserted in the middle of a wave, which might be causing some distortion. If the memory read 44,444.4̅ sample per second, and the pwm counter adjusted to use 250 clock cycles on each up strobe and the same on the down strobe, it would allow the two to stay in sync with each other. There would still be a problem of a changing value in the middle of a rising cycle, but if the pwm counter incremented on both the rising edge and falling edge, it would address that issue.