EE 371 Lab 5 Report

Jiarong Qian

Introduction

In this lab, I worked on several systems that utilizes the CODEC (audio coder/decoder) on the FPGA board to take in and output audio signals. Each system has a different noise filtering algorithm and the resultant sound varies as well.

Procedure

I started by outputting the read data directly to the write data. The two signals, read and write, that tell the system to do reading and writing are assigned to the values of signals read_ready and write_ready, which tell the system that the two actions can be taken. By writing 4 simple assignments, I made the CODEC to work and the input sound could be played on the output speaker.

In order to reduce the noise coming from the speaker, I implemented an average filter for the read data to go through before outputting to write data. The averaging filter contains 7 internal flipflops to store the last 7 input values. It calculates and output the average of the 7 values and the input value. By calculating the average of the past values, the filter removes some small fluctuations in the sound, which is the noise we hear.

By taking the average of even more samples, we can further improve the sound quality. In order to implement a flexible module whose sample size can be changed, instead of using flipflops, I used a fifo module whose size could changed by passing in different parameters. The module also has a flipflop as an accumulator. Each time when a new value is inputted, the quotient of the value divided by the sample size is added to the accumulator and written into the fifo buffer. When the buffer becomes full, the system starts reading from it. The read value is subtracted from the accumulator so that the accumulator is the average of the latest samples.

Results

The simulations of filters show that they are correctly calculating the average of the samples. After downloading the designs to the FPGA board, I found that the sound quality is much better with an averaging filter. I also tried different size for the fifo and found that the size with the best effect was $2^8 = 256$.

Though the averaging filter can filter out some noise, it also filters out part of the original audio signal. It may be hard to hear the sound of the singer clearly with the fifo filter.

Additionally, the filters' performances vary with different songs played. The filters work well with soft music. However, when playing rock 'n' roll, the performances would degrade greatly.

Problem Faced & Feedback

I initially got a lot of noise because I did not use signed logic for the voice samples. I also had some trouble using the fifo. I first used non-blocking assignment to set read for fifo to 1 when full becomes 1, but that

introduced an extra delay. Thus I used blocking assignment instead. Another issue was that I could not write values to the fifo after it was full. And no new data could enter because I was only reading when the fifo was full and the fifo was set only to be written when not full. When I found that out, I disabled that feature so that I could just overwrite on the old values.

Additionally, after I first implementing the filters, the effect was not really obvious. I was suggested by the TA that this may be due to that my filter is collecting multiple values that are exactly the same during one read cycle. The solution was to use a finite state machine which has two states. The first state waits for the read signal to become true, at which we read in the new data and updates the new average value as well. Then the FSM goes to the next state, in which the system only waits for the read signal to go back to 0 so that it can return to state 1. The state diagram is given below:

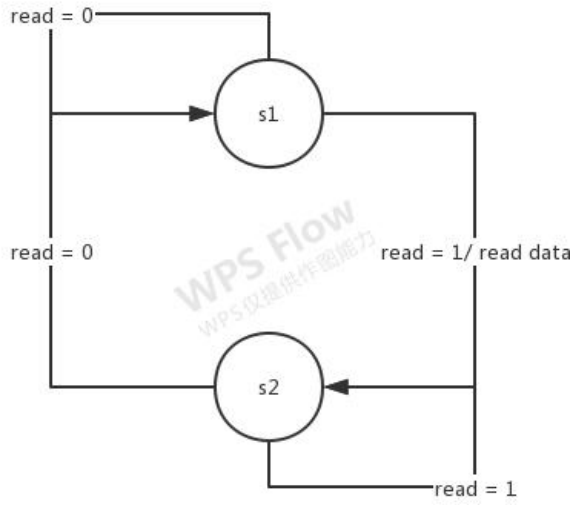

Figure 1. state diagram of FSM in filters

I spent more than 40 hours on this lab. Mostly because of being stuck on using shift and signed values and trying to reduce the noise after the initial implementation.

Appendix

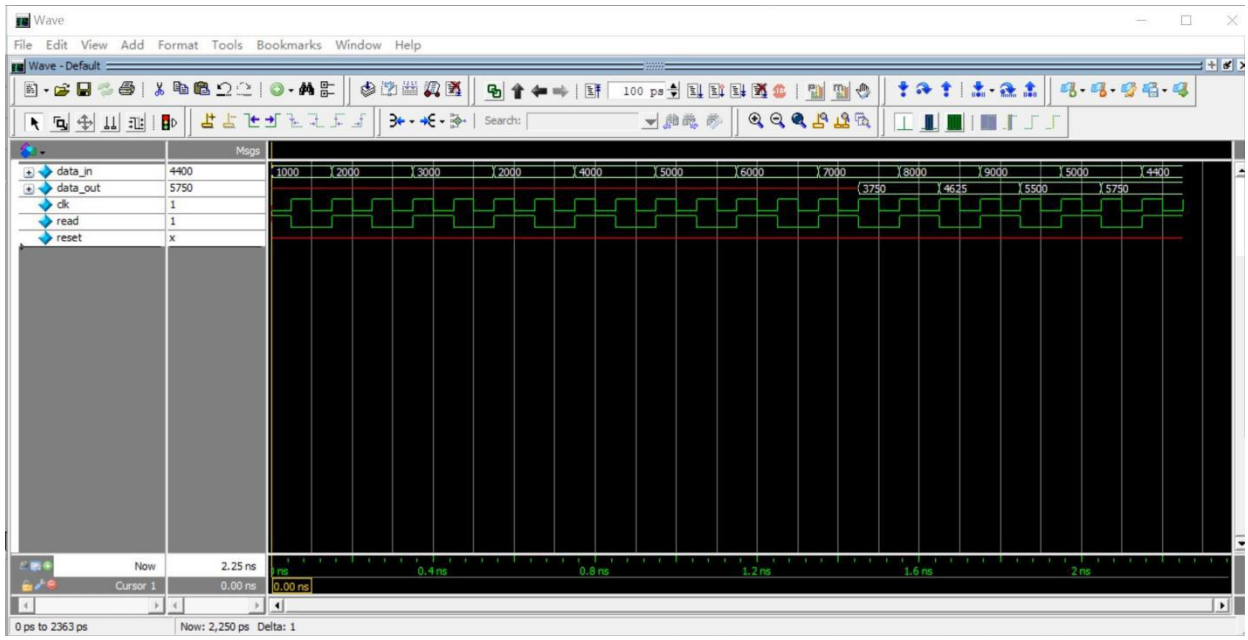

Figure 2. average filter simulation

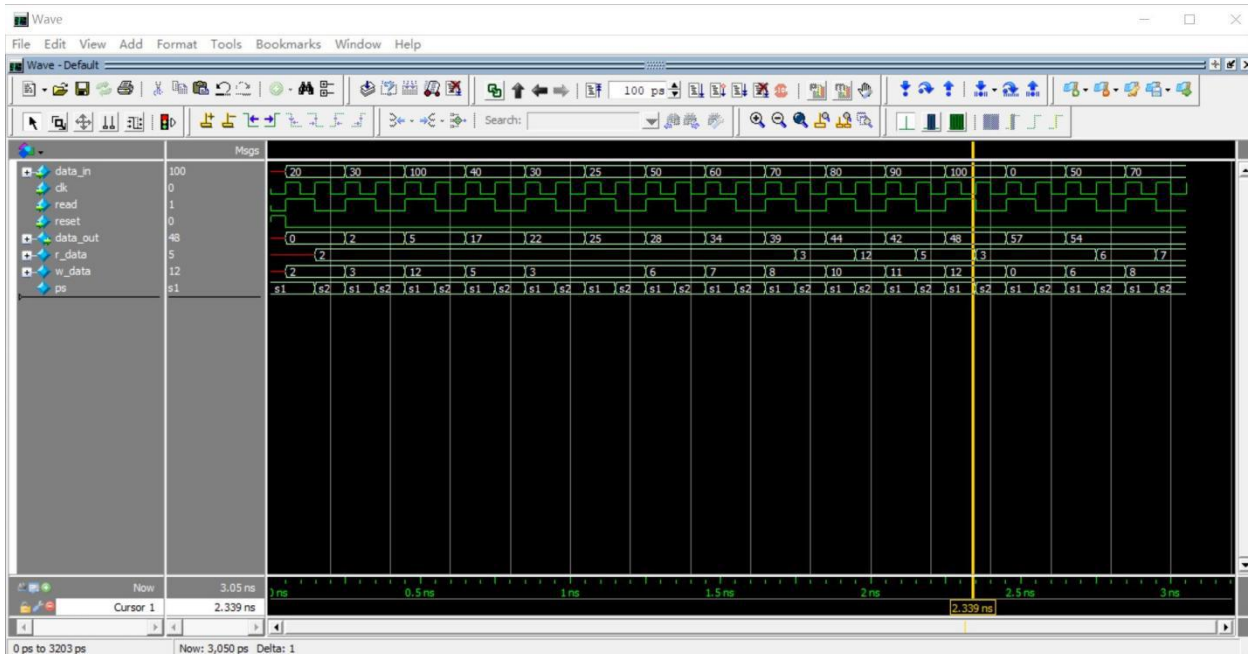

Figure 3. N average filter simulation