Khoa Tran and Ravi Sangani

EE 371

May 22, 2021

Lab 5 Report

**Procedure**

**Task #1**

In order to approach this task, we looked at the block diagram to understand the connection between the audio CODEC interface as well as the audio/video configuration and the noise generator. Looking at these modules and their connection, we were able to understand when read and write is enabled allowing for writedata\_left and writedata\_right to receive the read data with the addition of the noise. We were able to understand how KEY0 was implemented to add the noise in the given audio input. However, without clicking any KEY the audio will receive the input from the given piano file.

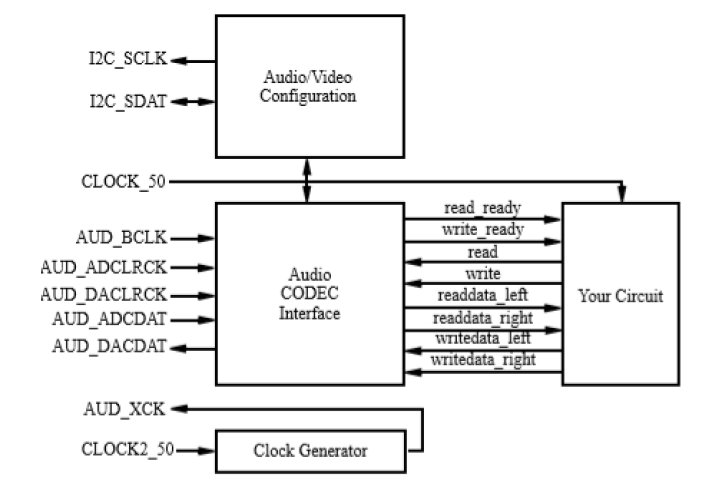


Figure 1: Block diagram for task 1

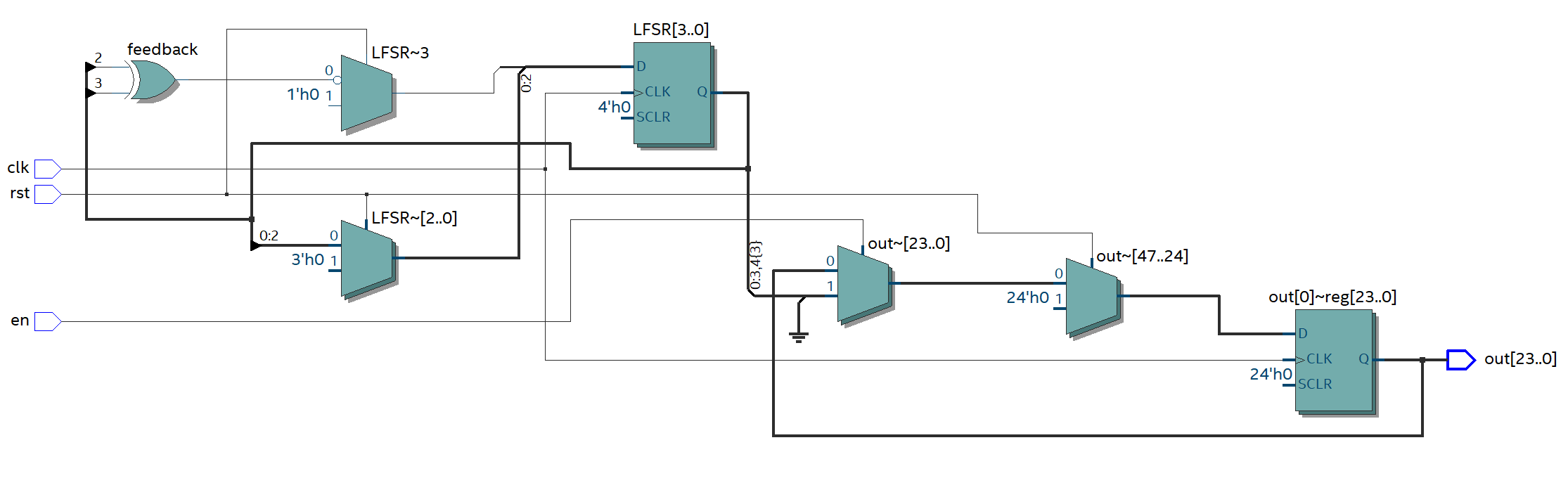


Figure 2: Block Diagram for noise generator

**Task #2**

Approaching this problem, we developed the block diagram for task 2, which is a fir filter that takes 8 samples and shifts between 7 different registers and from the output of each register, divide the value by 8 and add for the total in order to get the average and eliminate the noise through the average of 8 samples and continues onwards.

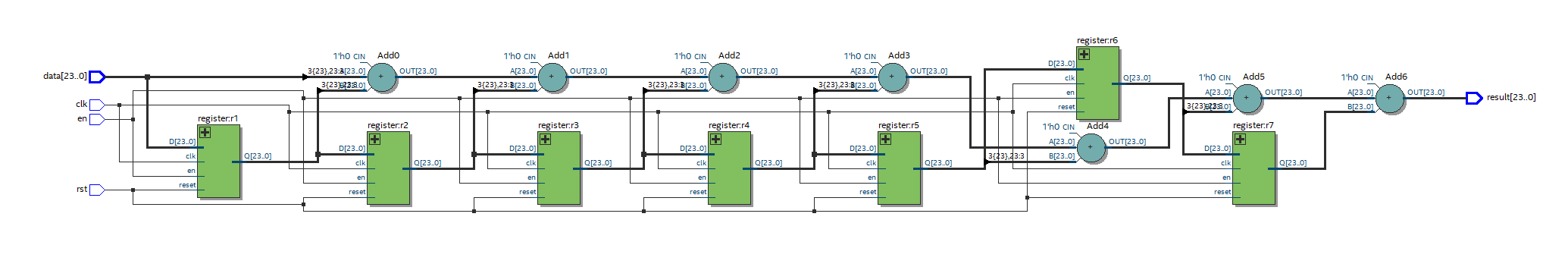


Figure 3: Block diagram for task 2

**Task #3**

In order to solve this problem, we developed the block diagram, which consist of 16 registers or n shift registers acting like an n size buffer. This was developed to parameterize the FIR filter in task 2, in order to increase the amount of samples to average. This parameterization allows for filtering of the noise to increase and eliminating some of the high pitch sounds that still exist in task 2. In this task, we used a generate statement to have a for loop, calling n number of registers and shifting them, storing the values in a 2D array. This way, we can keep track of the oldest value that was entered and subtract from the final output value. The accumalator also works like a register, adding divided values as well as subtracting the oldest value.

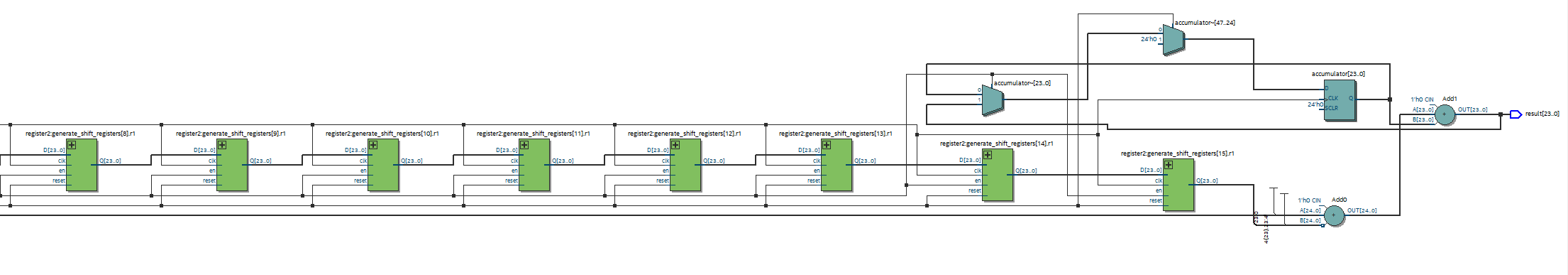
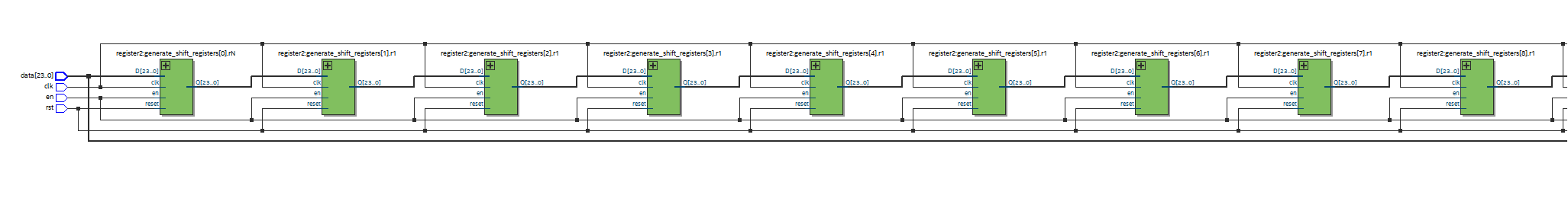


Figure 4: Block diagram for task 3 with n = 16

**Results**

**Task 1:**

No additional modules or testbench was needed for the completion of this task.

**Task 2:**

For the first part of task 2, we developed the registers to shift and store 8 consecutive samples. From these 8 samples, data is divided by 8 for each output of a register and added together for the resulting output. In Figure 6, the waveform simulation shows how Q outputs D on posedge clk. As well as Figure 5, showing the waveform simulation of the filter, averaging samples given, outputting to result.



Figure 5: Waveform simulation for fir Filter in task 1

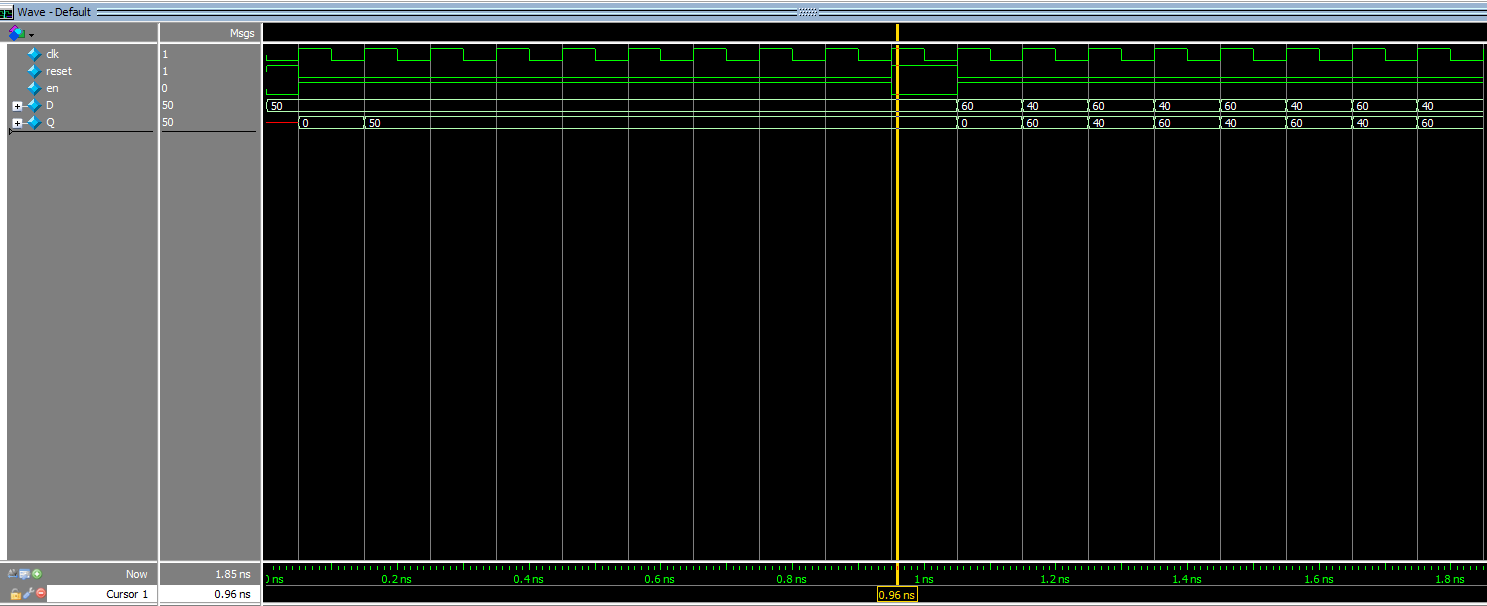


Figure 6: Waveform simulation for register in task 1

**Task 3:**

In this tasks, we performed simulation test on the register2 module, as well as the parameterized FIR filter that uses a buffer sized n, and an accumulator. This method allows for n samples to be average and filter out the noise. In Figure 8, the simulation for the register module can be seen as well as Figure 7, for the FIR filter with N = 16, the values are taken, shifted across registers, and averaged out for the output of result.

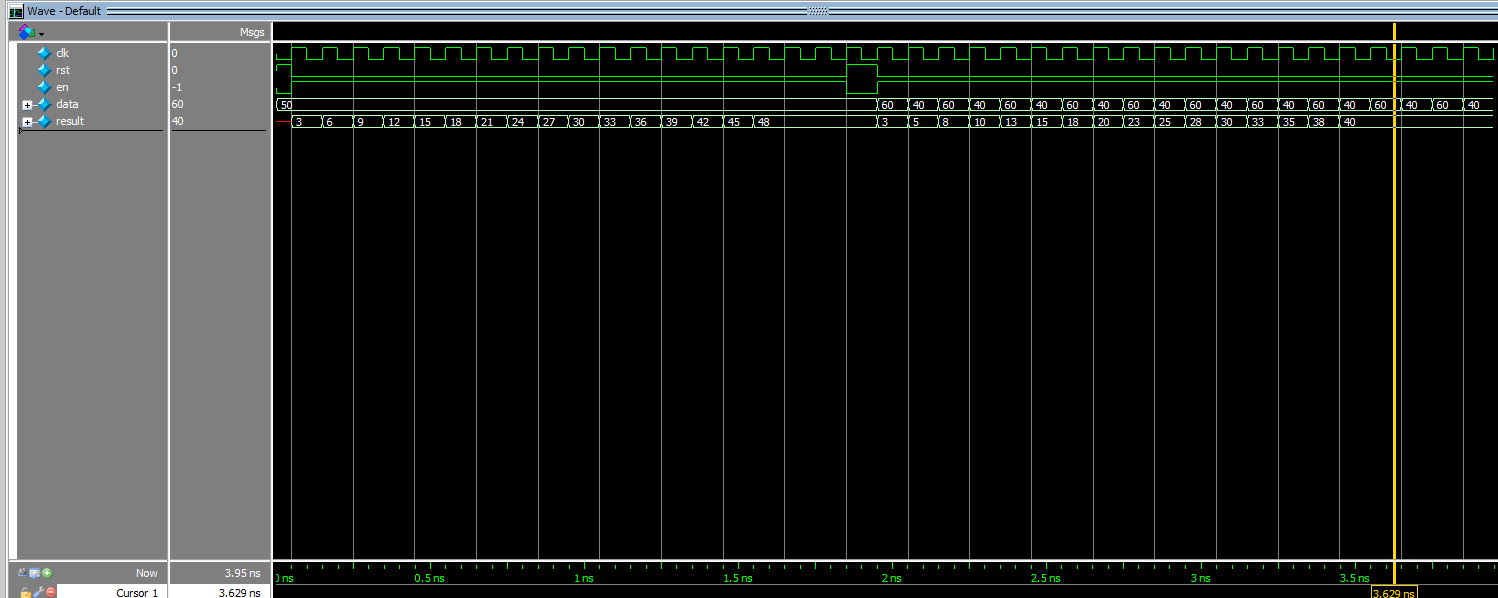


Figure 7: Waveform simulation for parameterized fir filter for task 2

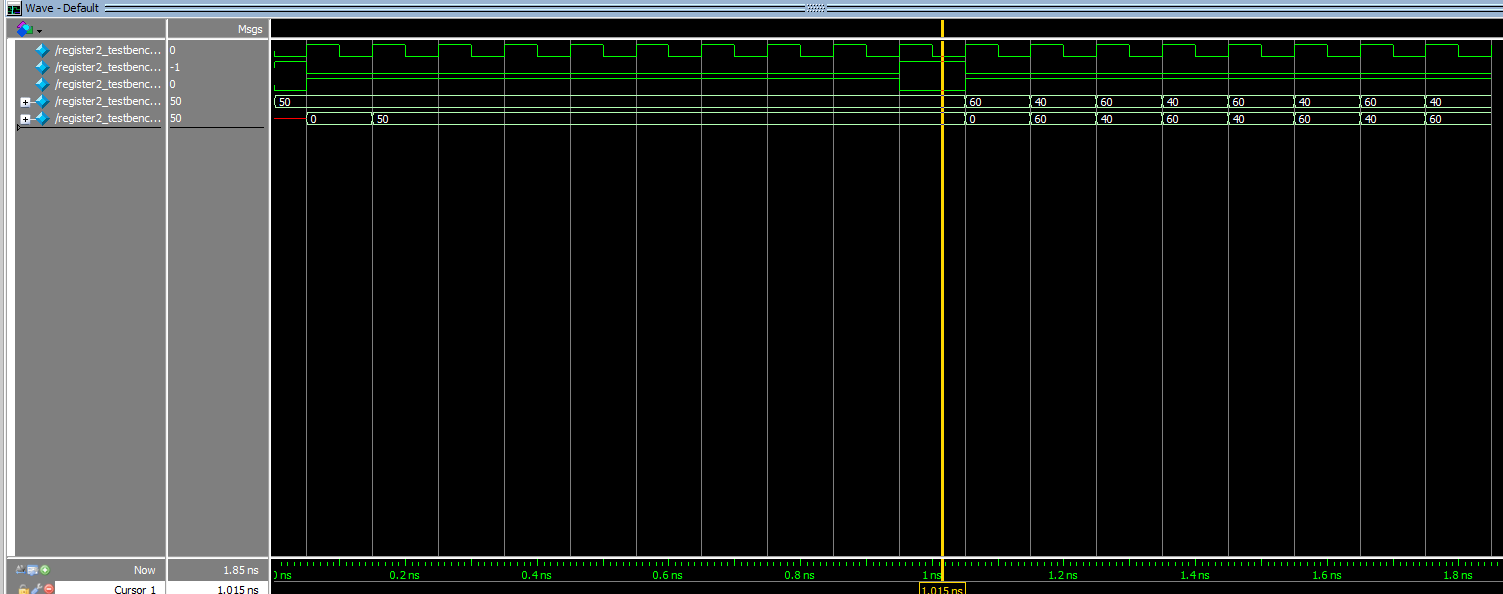


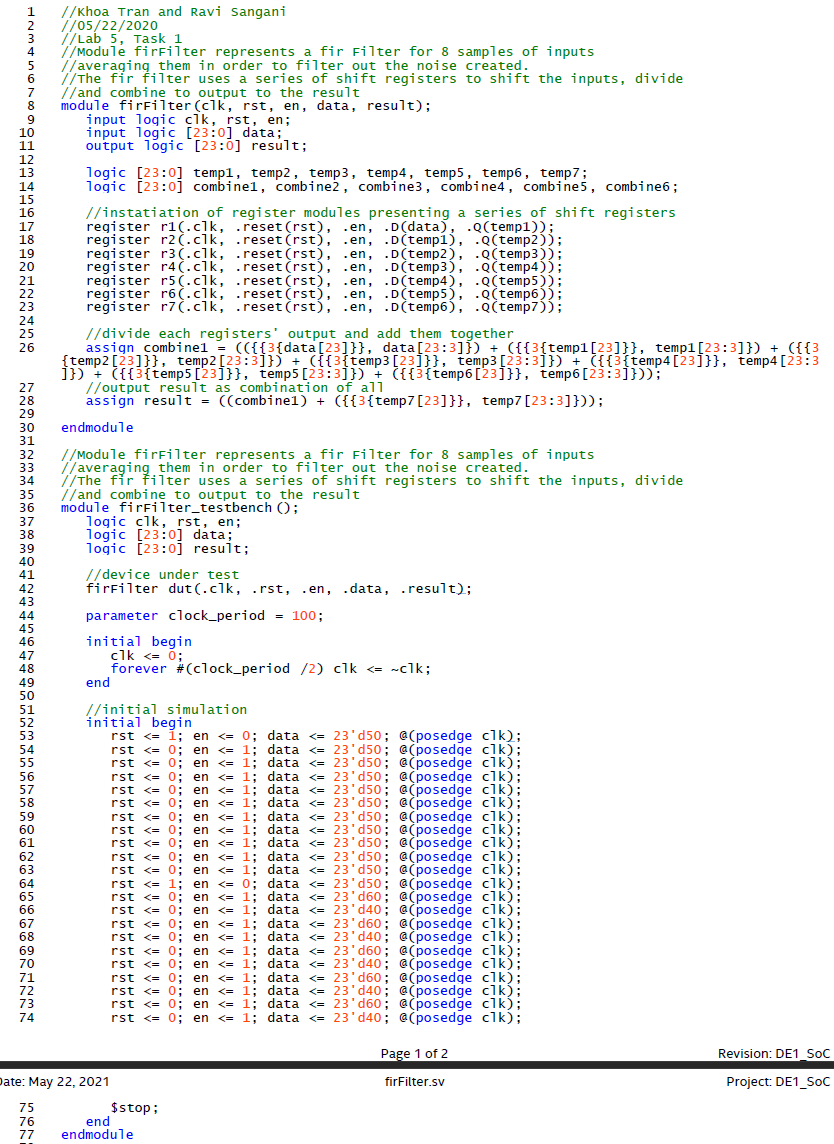
Figure 8: Waveform simulation for register2 module for task 2

**Final Product**

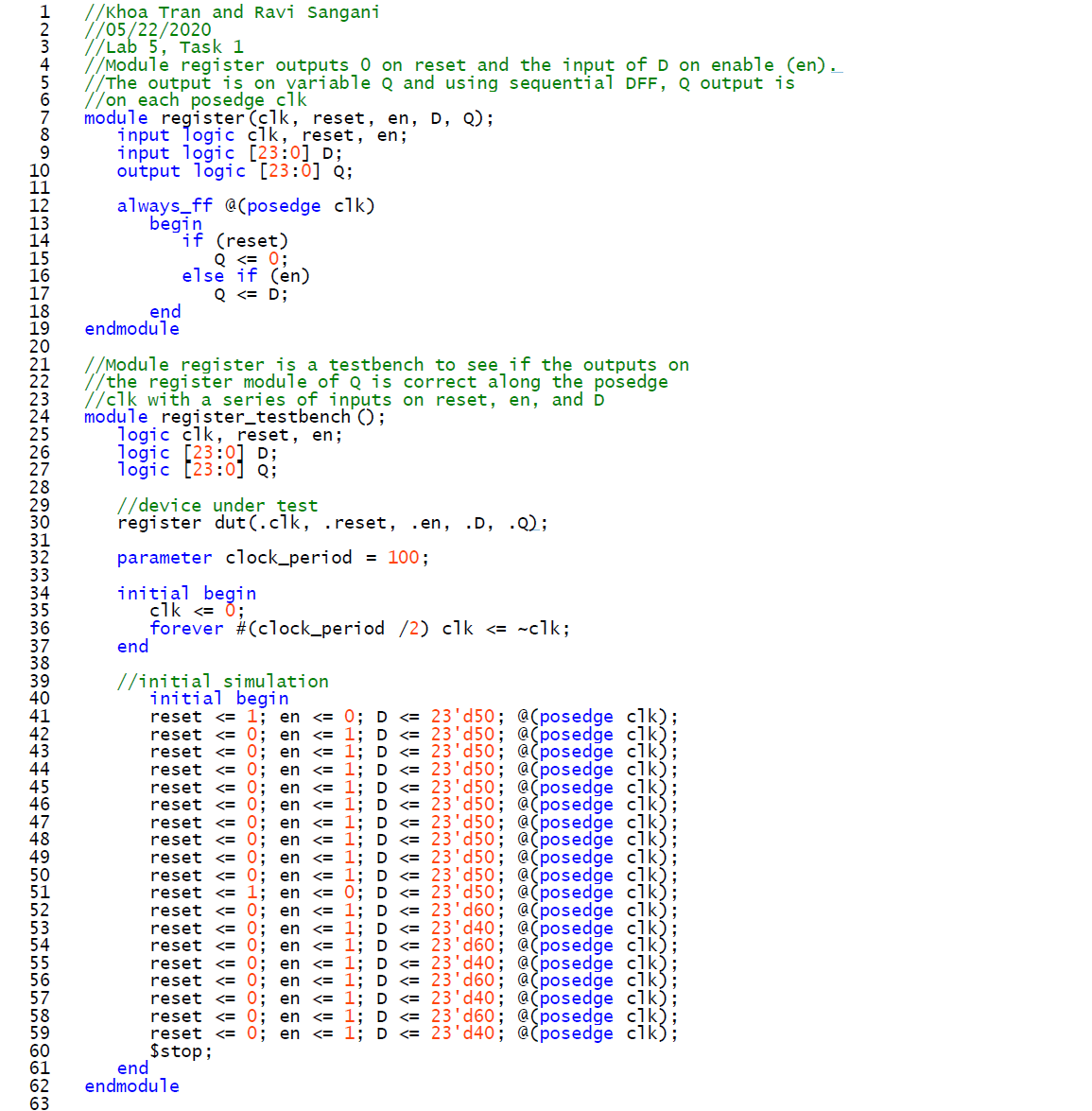
The overarching goal of this project was to design a FIR filter to filter out the noise created in the task 1 modules as well as parameterizing the FIR filter to be able to average a given number of samples. In task 1, we were able to comprehend the connections between the AUDIO CODEC and our interface, how noise is added, and how to connect our filter to eliminate the noise. From task 2, we understood how the filter operates and was able to create a buffer that is sized n, using shift registers to move the data through n registers. This way, we were able to keep track of the oldest data value and compute the average with the values in all of the registers. We didn’t have many issues implementing the FIR filter and we enjoyed the aspect of using frequency and audio in this lab as an introduction to how FPGA systems record, taken in, and output audio.

**Appendix: SystemVerilog Code**

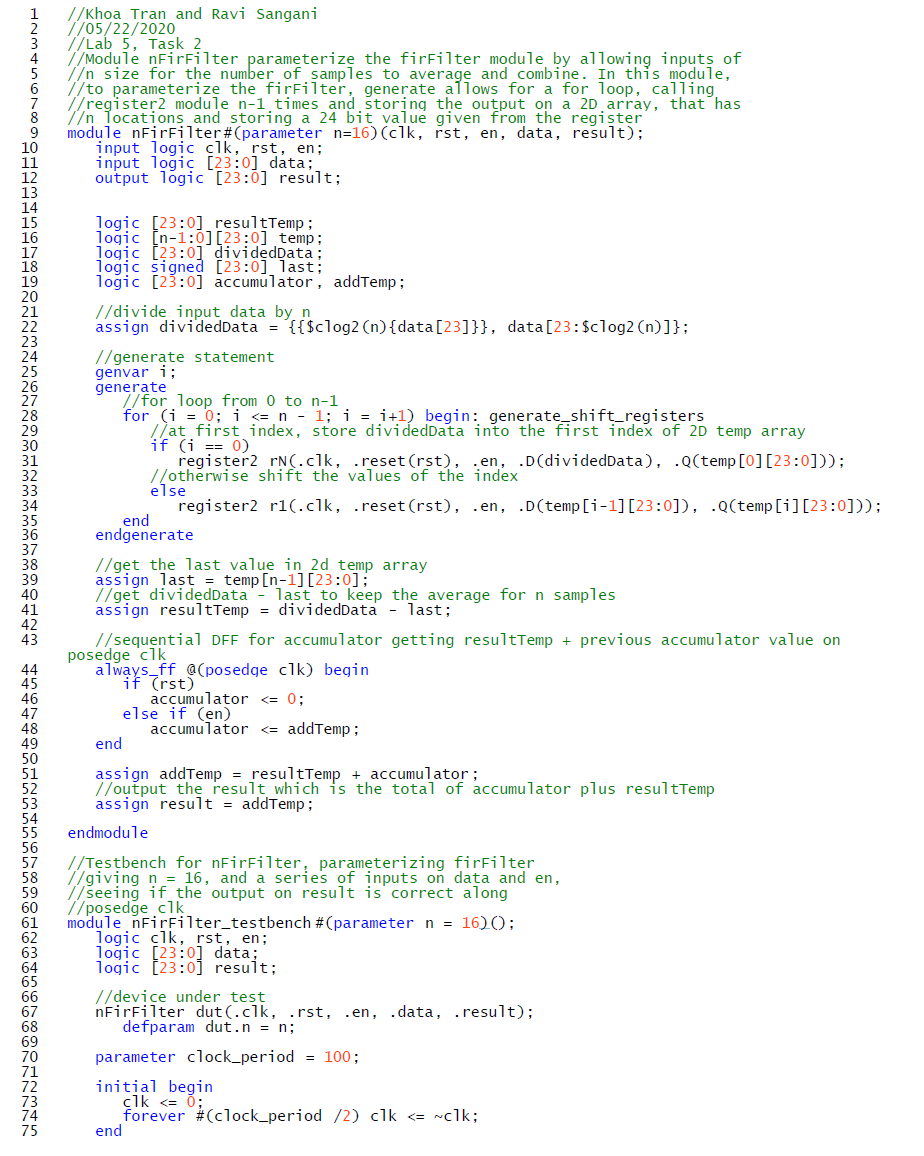
1. **firFilter.sv (task 1)**

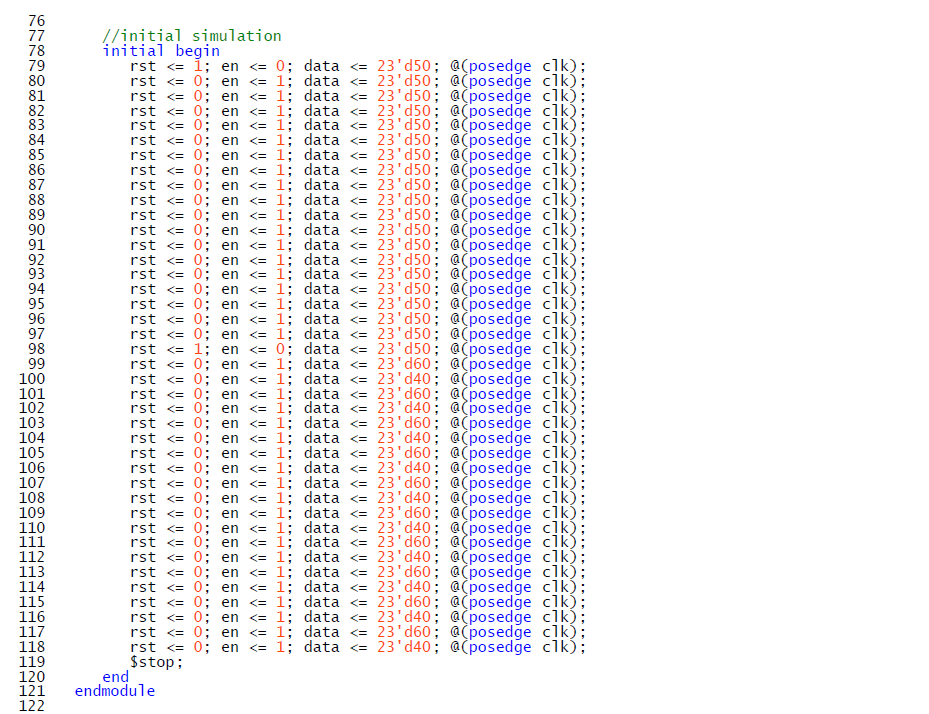


1. **register.sv (task 1)**

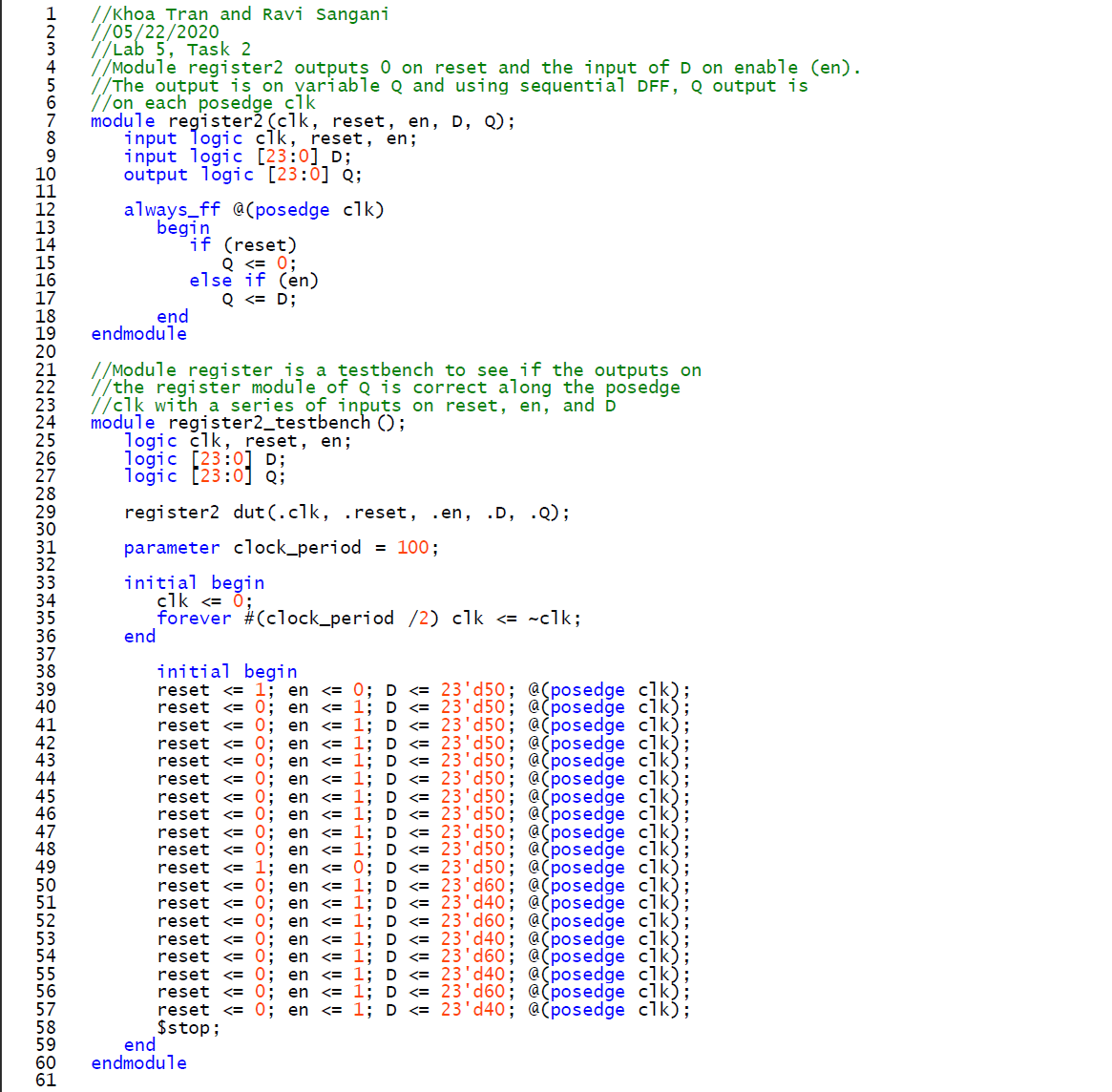


1. **nFirFilter.sv (task 2)**

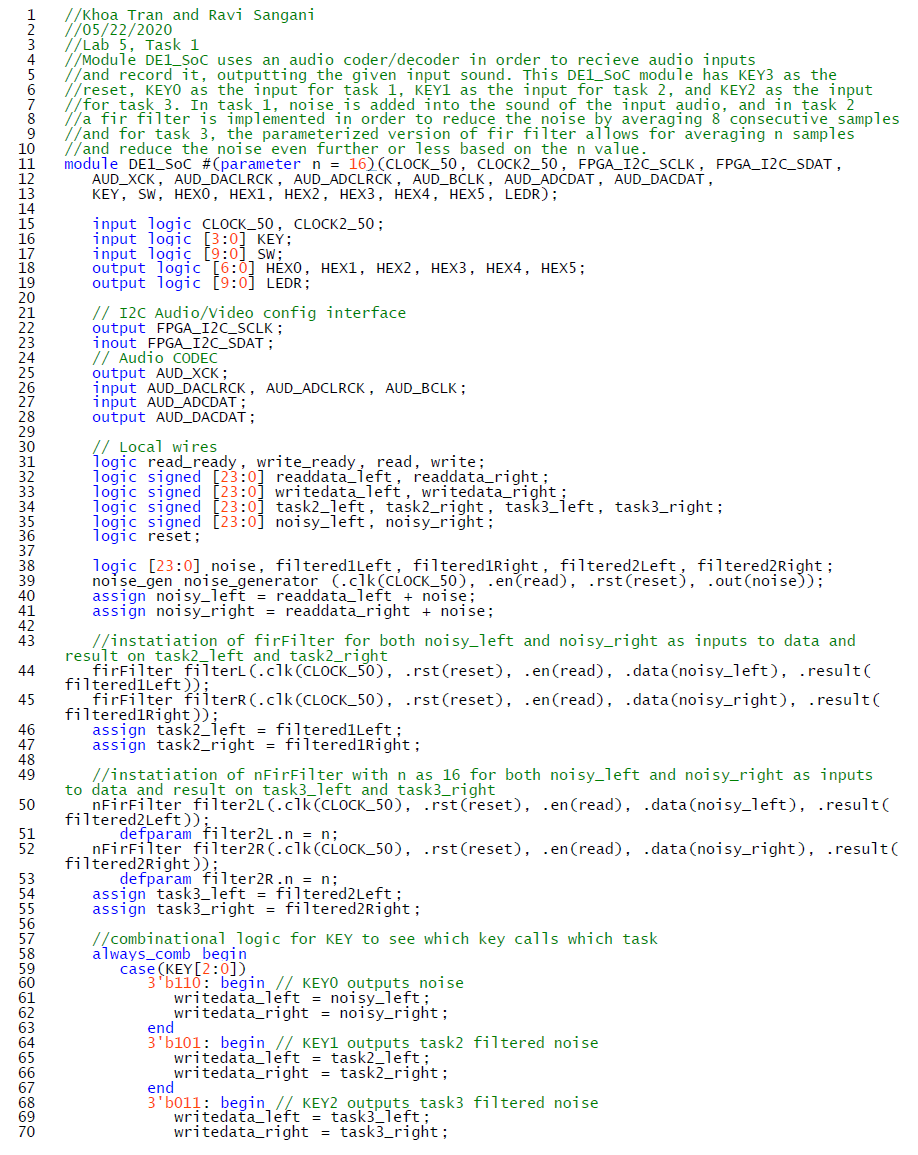


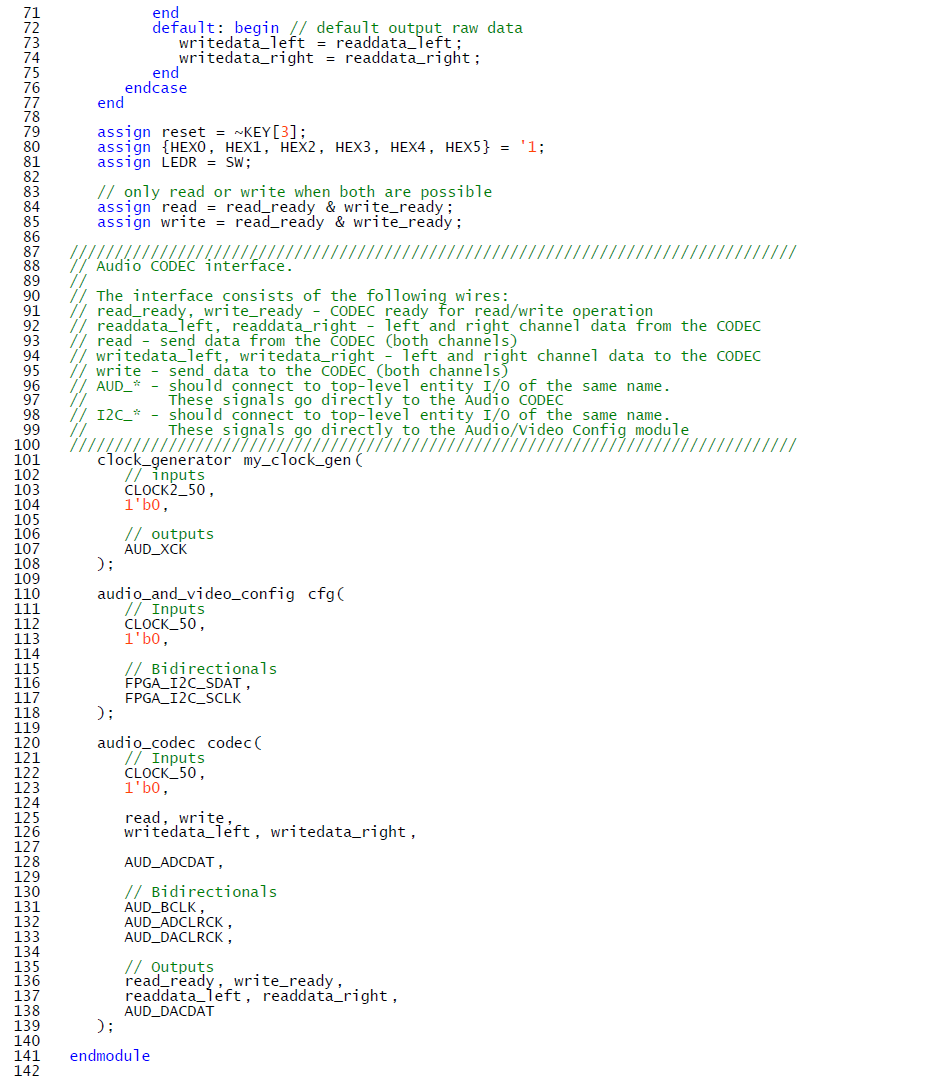


1. **register2.sv (task 2)**

****

1. **DE1\_SoC.sv (task 1,2,3)**

****

****