

ECE371 Design of Digital Circuits and Systems  
  
Winter 2019

**Lab 5 - FPGA Implementation of FIR Averaging Filter Hardware Accelerator Audio CODEX**

Student: Jeff Josephsen  
Professor: Dr R. Hussein

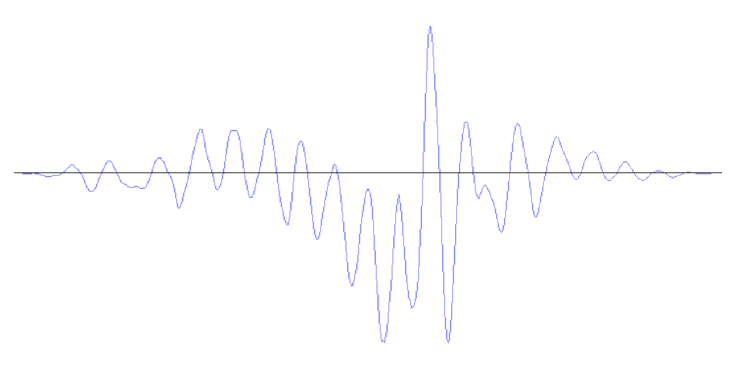
2-27-2019

**Abstract:**

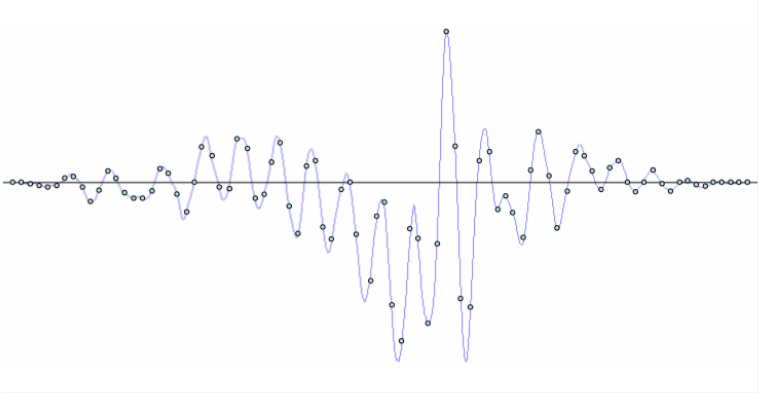
The purposes of this laboratory assignment was to investigate the audio coder/decoder (CODEC) on the DE1-SoC Development Board and to utilize the CODEC in the implementation of an Finite Impulse Response Averaging Filter of 8 and N samples. The implementation of these algorithms and their testing were completed both in software on Quartus Prime Lite Edition with data verification testing in ModelSim and in hardware on the Field Programmable Gate Array (FPGA) side of the Cyclone V chip of the Altera DE1-SOC development board with the data verification testing by SignalTap II Logic Analyzer. Significant amounts of debugging occurred during the completion of the lab. Significant amounts of time were spent in debugging efforts to make the software tools and SystemVerilog code work correctly. At the end of the laboratory assignment all parts of the software and hardware worked correctly. This was demonstrated to Teaching Assistant Staff.

**Introduction:**

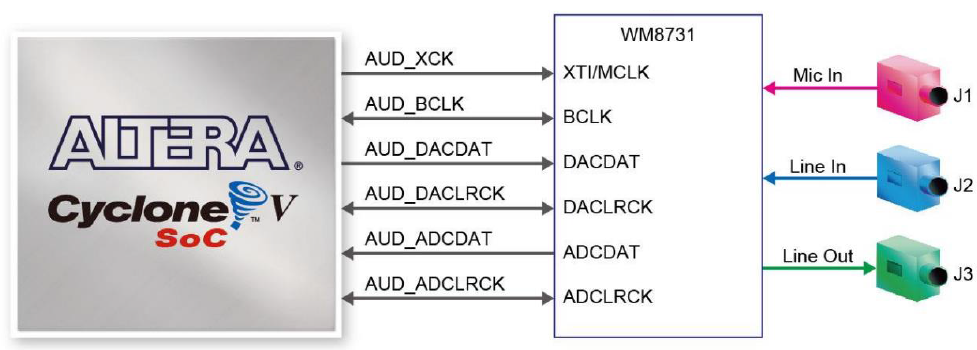
Sound waves, as they travel through air and other media, make initial contact with that media as an ongoing stream of pressure. The image below shows a sounds wave that is travelling through space. This specific sound wave could be a vocal sound such as the when someone says "Ah" or another sound.



If that soundwave were to come into contact with a microphone, then the vibrating drum of the speaker head could be translated into electrical signals. These signals could be measured at regular intevals in time. The following image shows that those sample, happening in regular intervals, would measure a variety of of values over time**.**

****

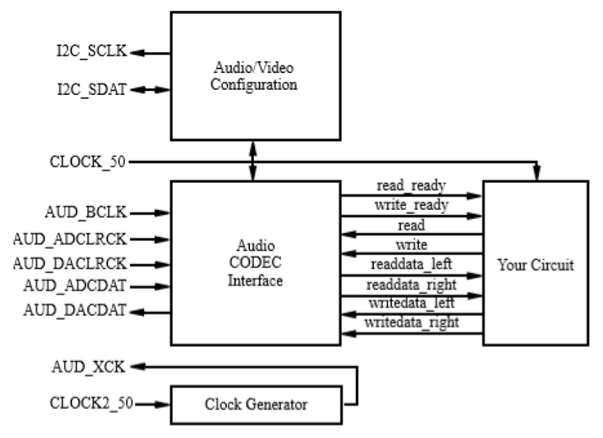
There is a system built into the Altera DE1-SoC Development Board which has audio coder and decoder, known as a CODEC, which is able to create signals for a speaker system or record signals from an input device like a microphone. The following image shows a block diagram, connection points, and variable names used in this codec system.



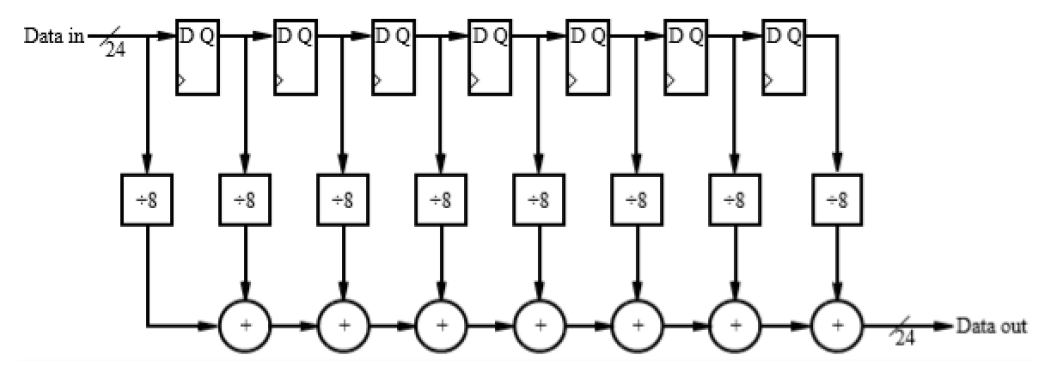
This codec system was utilized in the following laboratory assignment.

**Procedure:**

The laboratory system was broken up into three tasks. In the first task we were directed to use the following image and the variable names in the image to update a starter code such that the code would compile and synthesize appropriately. for use on the Hardware of the development board.

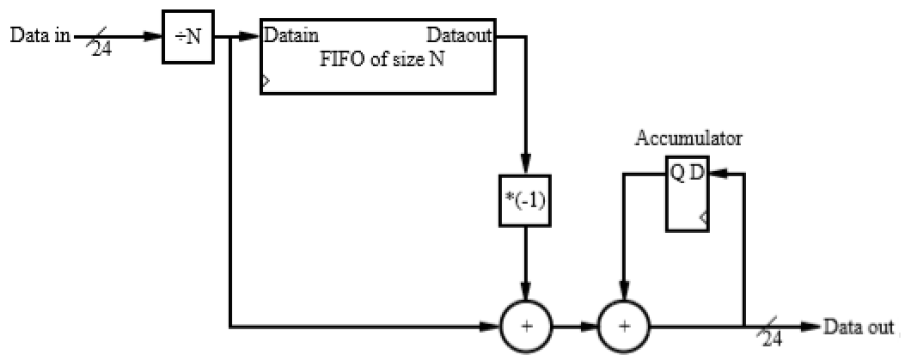


In Task 2 the objective was to write a module a testbench module such that an incoming signal would be fed unto an array of eight flip flops and averaged by the algorithm laid out in the following block diagram.



Next, the eight sample averaging algorithm was tested via a SystemVerilog test bench and ModelSim software. After testing, debugging, and retesting the project with its modules was synthesized in HDL machine language and uploaded to the hardware. The smoothed signal was audibly compared with the previous sample that had not been tested.

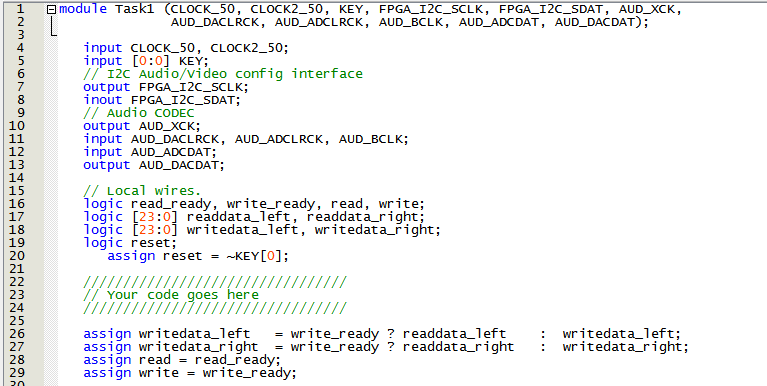
Lastly, in Task 3 the objective was to write a module a testbench module such that an incoming signal would be fed unto an array of and unknown number of flip flops and averaged by the algorithm laid out in the following block diagram.



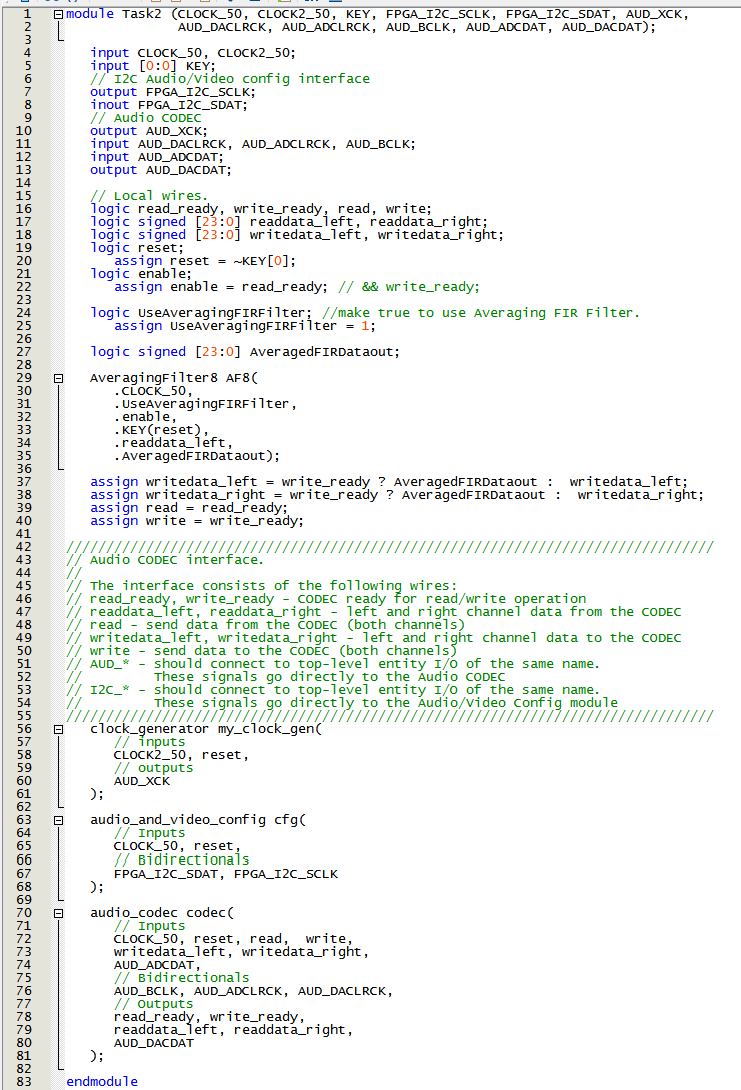
The N-sample averaging algorithm was tested via a SystemVerilog test bench and ModelSim software. After verifying operation through testing the project with its modules was synthesized and uploaded to the hardware. Again, the smoothed signal was audibly compared with the previous sample that had not been tested.

**Results, and Analysis:**

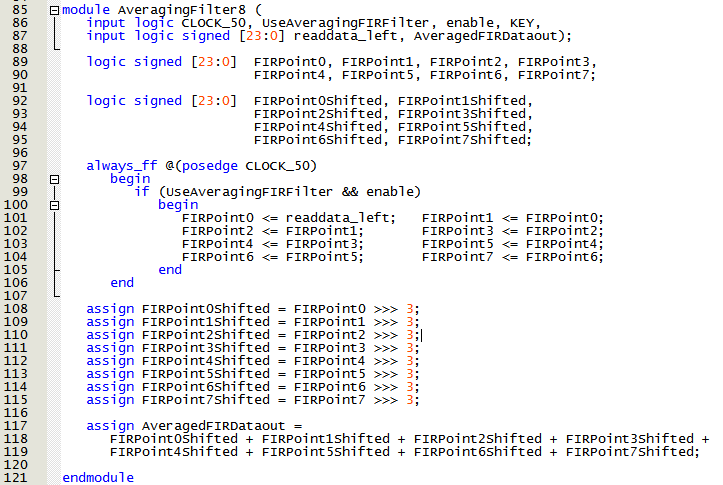
In Task 1 the following module was, with the exception of lines 26-30, the module that was given to the students. Those lines 26-30 were updated to allow for an unfiltered signal to pass from the microphone through the CODEC to the speaker system.



In Task 2, and 8-Sample Averaging Algorithm was implemented in software and hardware. The following image shows the code of the top level module and includes the instantiation of the AveragingFilter8.

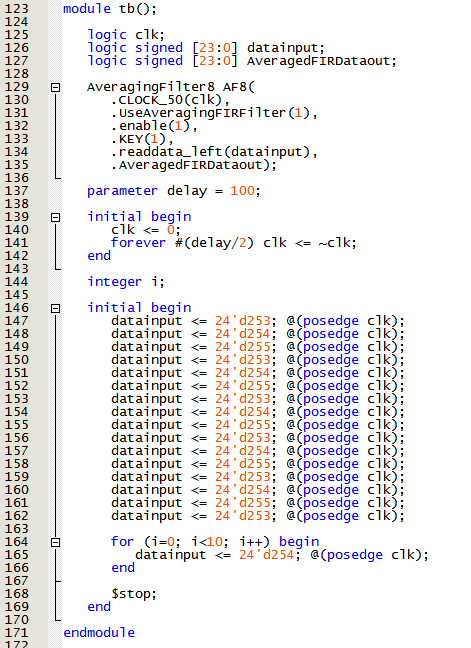


The following image shows the AveragingFilter8. You may notice that the signals of the flip flops were manually coded. This was appropriate for module made to instantiates a filter of *known* signal length. You may notice that in the upcoming task 3 that these signals were by necessity created by use of a generate-variables loop.



Another important item to notice in the above image of the module code is the "if ... && enable)" test. This is necessary as the development board runs on a 50MHz clock and the audio CODEC runs on a 48KHz clock. The "enable" signal is only high when the audio codec has completed and analog to digital conversion each 1/48,000 of a second. Without this constraint the eight flip flops would fill will the same signal approximate 1000 times and would produce an average of eight equal samples.

Prior to pushing the algorithm to the board a testbench was written and signals were visually observed in ModelSim. The following image shows that testbench as written in SystemVerilog.



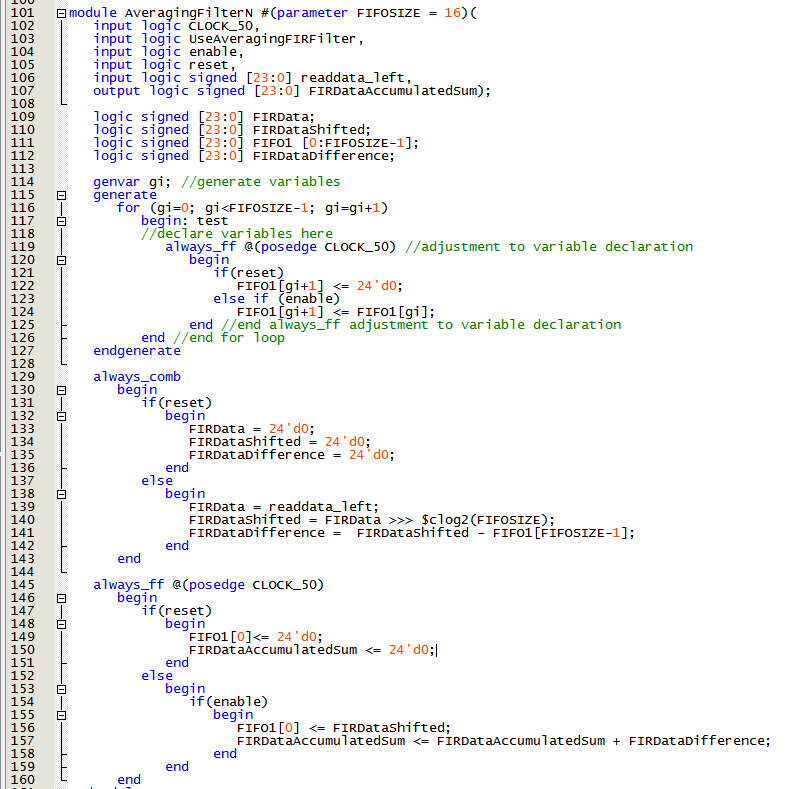
This code was compiled and simulated at the register transfer level in the ModelSim software environment. This allowed for debugging and enabled the code to be updated effectively. The following image is a screenshot of the simulation working correctly. It is a masterpiece.



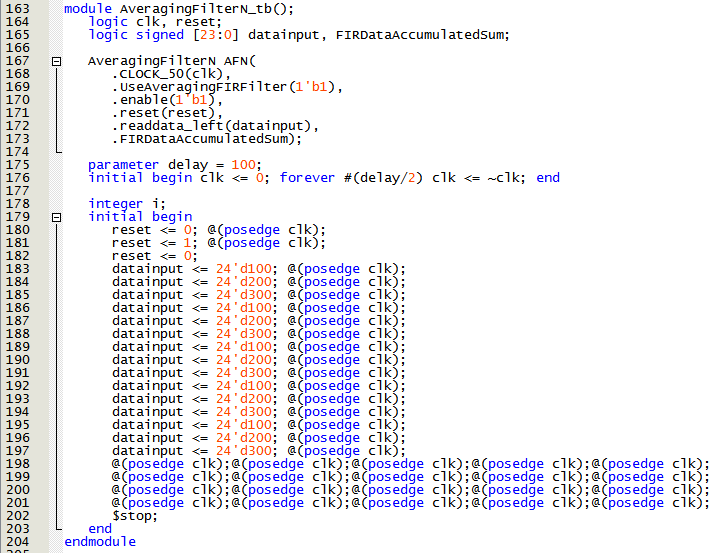
After correctly working in software simulations the software was synthesized and uploaded to the development board FPGA. The system, after more debugging and recompiling worked correctly and was inspected by Teaching Assistant Staff.

Lastly in Task 3 the code found in the following image was written. This code allowed for the system to keep a running average of the last N number of signals flowing into the CODEC.

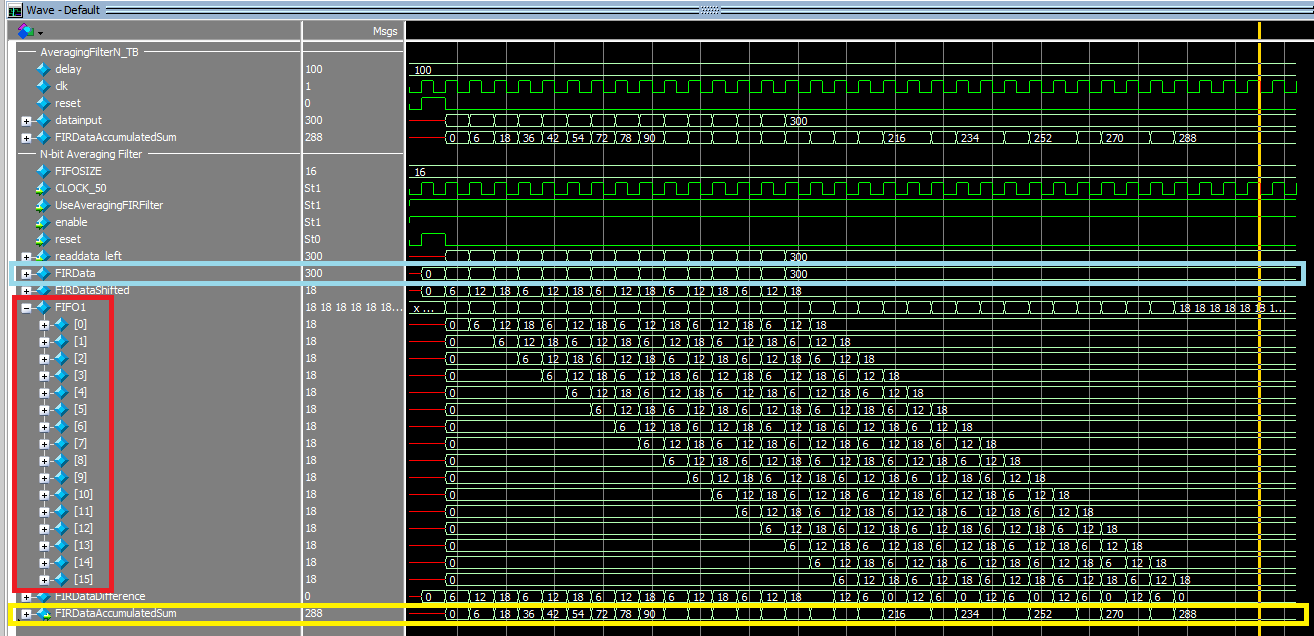
Please notice the Generate coding block. This block allows for the creation of a number of internal signals. These signals are created with the attributes of the alway\_ff @(posedge clock) and if(enable), among other, constraints. This is very useful in the creation of multiple variables.



Again, a testbench was written, debugged, and used in multiple tests in ModelSim RTL simulation. The code of the testbench can be seen in the following image.

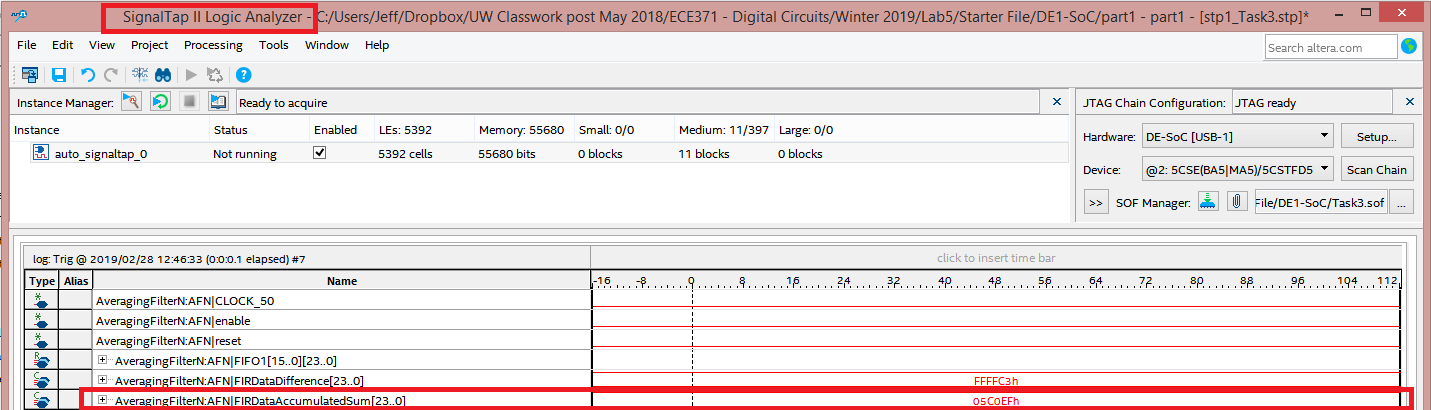


The RTL simulation on ModelSim for a working n-sample running average can be seen in the image below.

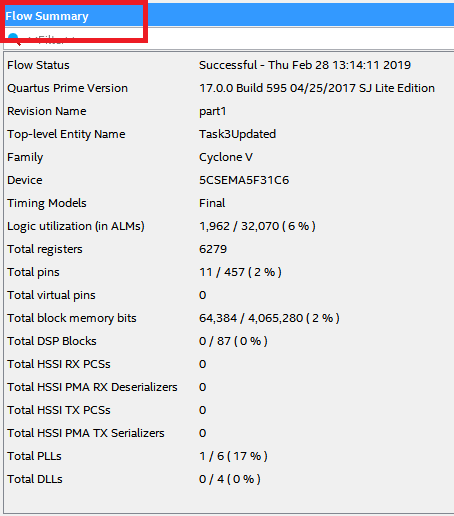


You may notice the genvar created FIFO in the red rectangle. The light blue rectangle shows the data input and the yellow rectangle shows the running average. The calculated average is correct enough for the application, but does suffer from a rounding-down error for no more than the number of elements in the FIFO.

The code was then synthesized for uploading to FPGA hardware on the development board. In that compilation and synthesis communication lines were created for internal chip signal viewing on Signal Tap software. The following image was taken after the system was working correctly.



The following image shows the Flow Summary of the system in its entirety. The system used approximately 2000 Arithmetic Logic Units and about 8 Kbytes of memory.

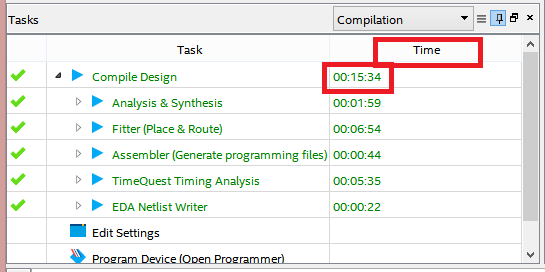


**Conclusion:**

I think that this lab, like the four before it, was invaluable. Specifically, I appreciated seeing a couple of algorithms built in hardware from a block diagram. There are many algorithms that could be implemented. This lab caused me to stop and go googling for "examples of HDL algorithms in FPGAs" I noticed that most any math problem could be implemented in hardware. I can see that this could be extraordinarily useful.

I ran into a number of bugs in this lab. I am learning more and more what the compiling codes are and I am learning to write cleaner code.

On a funny note, I have noticed that writing code that compiles well as early as possible is extraordinarily helpful prior to synthesizing for a push to the hardware. The following image shows the 15 minutes required to compile and synthesize. re-synthesizing 3 or 4 times for simple errors can quickly ruin plans to eat a healthy lunch.



This lab work took approximately 15 hours to complete.