EE 371 Lab2 Report: Memory Blocks

Ziyi Huang

ECE, Embedded System

University of Washington

April 16, 2019

Introduction

The purpose of this lab is to model the memory implementation on an FPGA and examine the general issues involved in such implementation. In this lab, a 32x4 RAM and a dual-port 32x4 RAM are implemented using Quartus IP catalog or self-written SystemVerilog modules. Four procedures were performed. The rest of report will describe in details about the procedures, the results, and the problems faced & feedback of the lab.

Procedure

Task 1

In this task, a single-port 32x4 RAM was implemented via Quartus IP catalog and simulated in ModelSim. The block diagram of the 32x4 is shown if Figure 3. Not a lot of things are done in this task. All I had to do was to follow the instruction of using Quartus IP catalog. It was shown that the tool was very powerful in implementing memory for an FPGA.

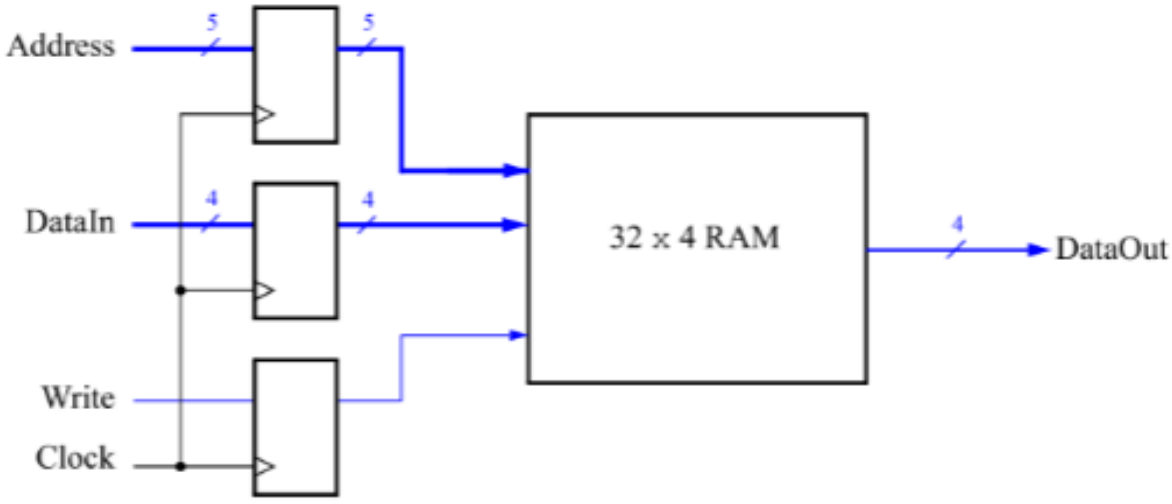

Figure 1: 32x4 RAM Block Diagram

The compilation report is shown in Figure 2.

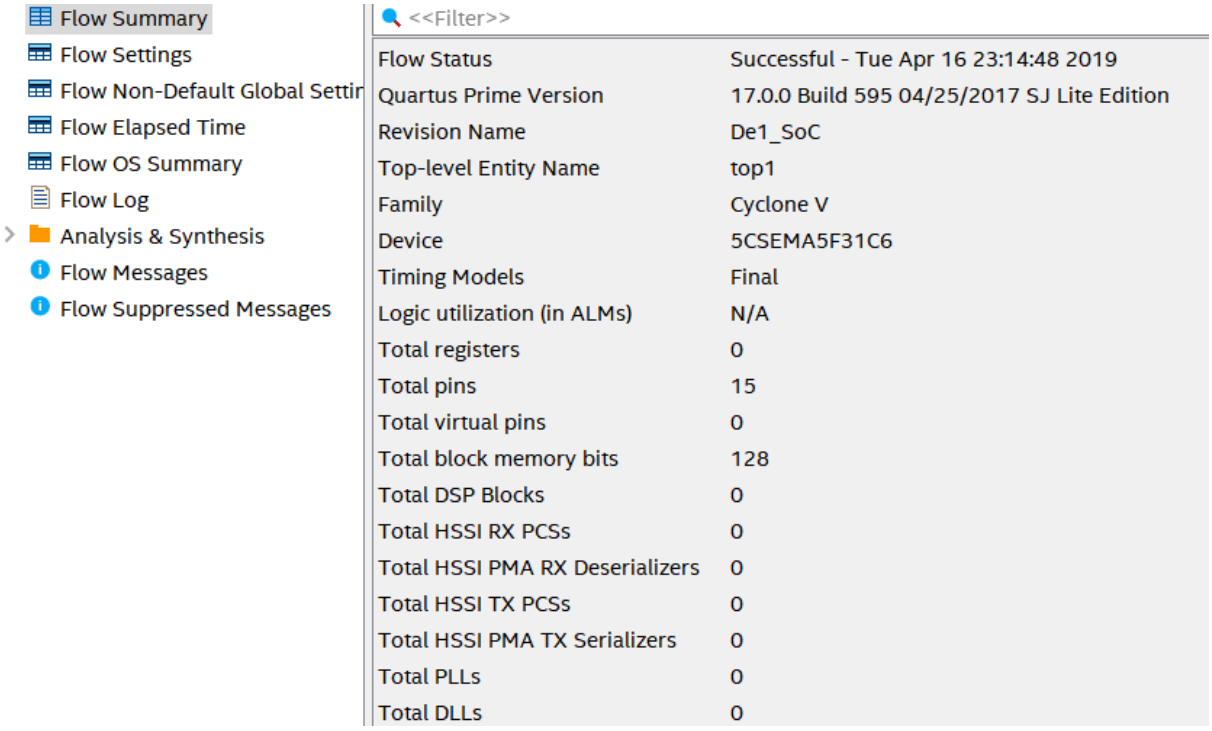

Figure 2: task1 top compilation flow summary

Task 2

In this task, a top-level module that interfaced the FPGA and the 32x4 RAM implemented in the previous task was developed so that I could interact with the RAM and view its contents. Not a lot of things are done in this task. All I had to do were to map the on-board elements to the RAM elements, implement display rules for the HEXes to display numbers and letters, and develop a testbench to simulate the driver on ModelSim. The on-board elements and their corresponding memory functions are listed below:

- SW[3:0] - input data for the RAM
- SW[8:4] - input data address
- SW[9] - write enable
- KEY[0] - clock input
- HEX[5:4] - address value
- HEX[2] - input data
- HEX[0] - data at the address

It was shown that the RAM generated by the Quartus IP catalog could integrate with the FPGA. Together, they could correctly model behaviors of the computer memory.

Task 3

In this task, a 32x4 RAM module and a top-level module that interfaced the FPGA and the RAM were developed so that I could interact with the RAM and examine the difference between this RAM and the RAM implemented in the previous tasks. The block diagram for the RAM module is the same as the one in Task 1, depicted in Figure 3. The RAM was implemented as a 2D array of size 32 and data width 4, as specified in the dimension. This implementation made sense because memory was essentially a large array of bytes. The writing was implemented as a D-flip-flops while the read was not, making writing synchronized and reading asynchronized. I chose to do this to reduce the waiting time between valid writing and getting the results through reading. I developed two testbenches, one for the top-level driver and one for the RAM module. The on-board elements and their corresponding memory functions are listed below:

- SW[3:0] - input data for the RAM
- SW[8:4] - input data address
- SW[9] - write enable
- KEY[0] - clock input
- HEX[5:4] - address value
- HEX[2] - input data
- HEX[0] - data at the address

It was shown that memory could be implemented as an array and the write as a D-flip-flop while reading was asynchronized.

Task 4

In this task, a dual-port 32x4 RAM module was developed via Quartus IP catalog. In addition, a top-level module that interfaced the FPGA and the RAM was developed so that I could interact with the RAM. Again, the Quartus IP catalog was a very powerful tool so I didn't have to do much. Testing required most of the efforts. Three testbenches were developed to ensure the

RAM read and wrote to the correct addresses. The on-board elements and their corresponding memory functions are listed below:

- SW[3:0] - write data for the RAM
- SW[8:4] - write address
- SW[9] - write enable
- KEY[0] - reset read address
- KEY[1] - clock reset
- HEX[5:4] - write address value
- HEX[3:2] - read address value
- HEX[1] - write data
- HEX[0] - data at the address

It was shown that dual-port RAM could be modeled using an FPGA.

Results

Task 1

The simulation for the top-level driver is depicted in Figure 3. The simulation verifies that the data is correctly written to the input address but only at the positive clock edge.

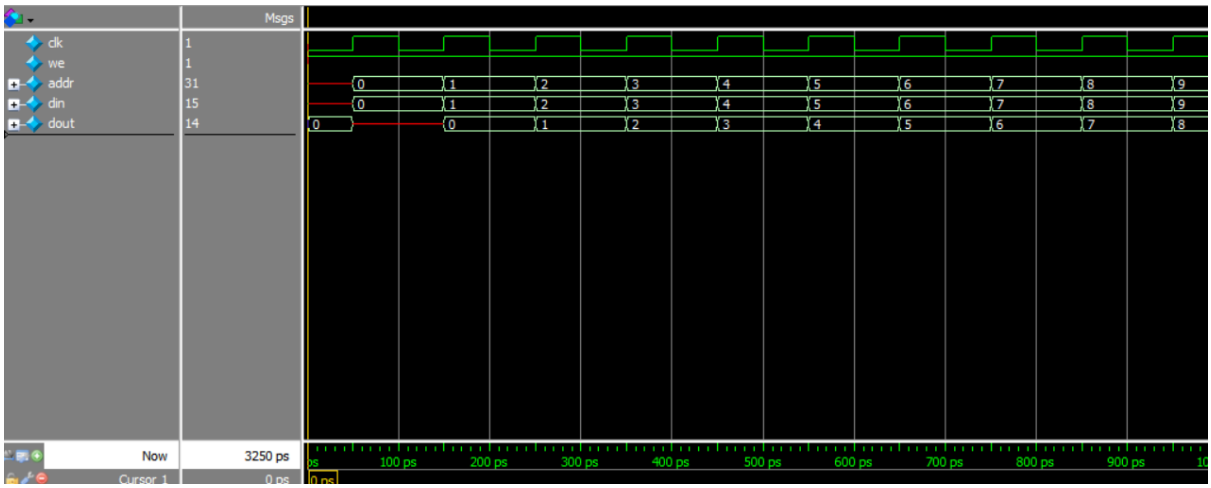

Figure 3: Top Simulation Waveform in ModelSim

Task 2

The simulation for the RAM is depicted in Figure 4. The simulation verifies that the data is correctly written to the input address but only at the positive clock edge, which corresponding to the negative edge of KEY[0] since it's actively low.

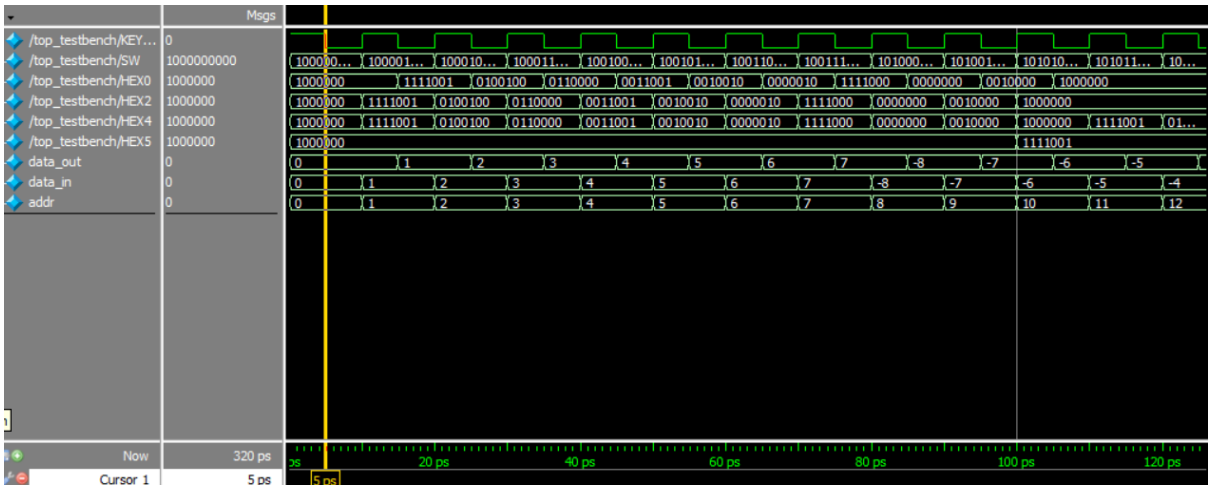

Figure 4: Top Simulation Waveform in ModelSim

The compilation report is shown in Figure 5.

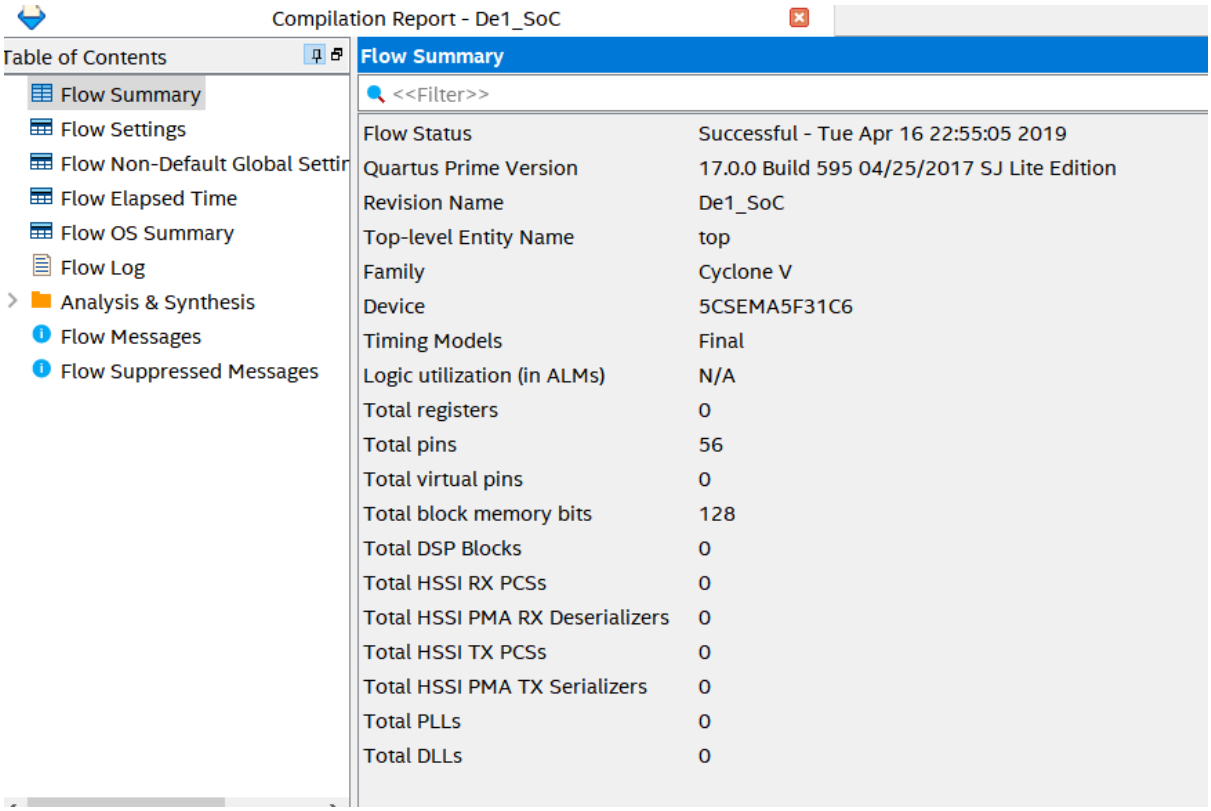

Figure 5: task2 top compilation flow summary

Task 3

The simulation for the RAM is depicted in Figure 6. The simulation verifies that the data is correctly written to the input address but only at the positive clock edge. The red interruptions between the read output are caused by the memory being uninitialized before the writes.

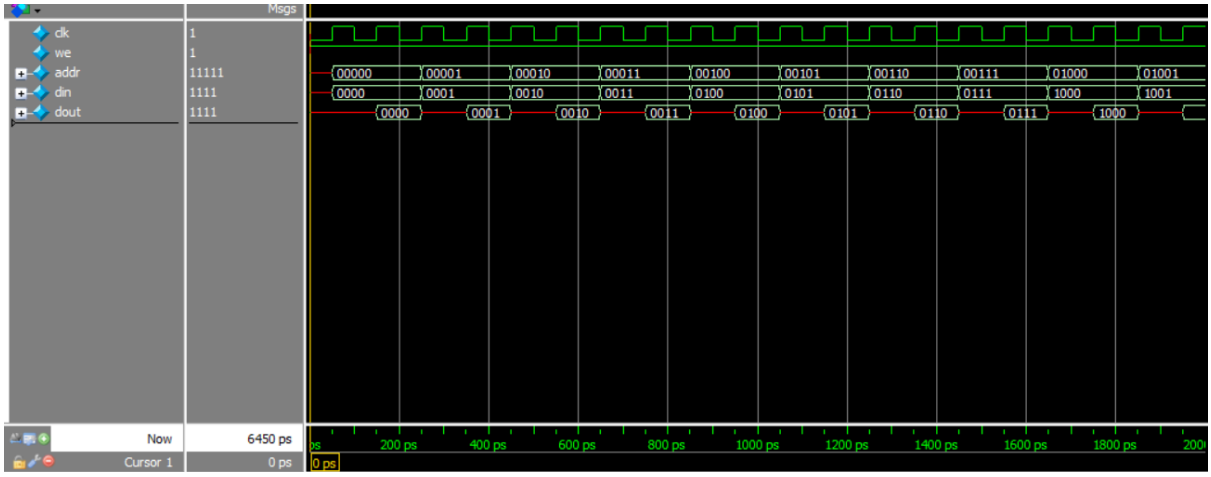

Figure 6: R32 x 4 RAM Simulation Waveform in ModelSim

The simulation for the top-level driver is depicted in Figure 7. The simulation verifies that the on-board elements behave correctly. I set the radix to decimal instead of unsigned integer and caused the signals to display negative values while they are at higher addresses (i.e. larger than 5'b10000).

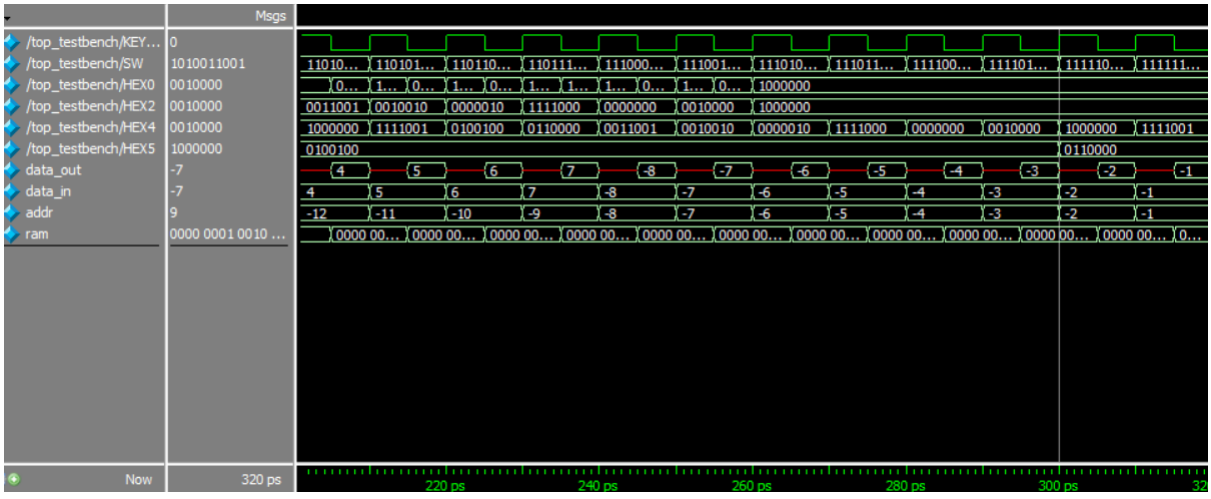

Figure 7: Top Simulation Waveform in ModelSim

The compilation report is shown in Figure 8.

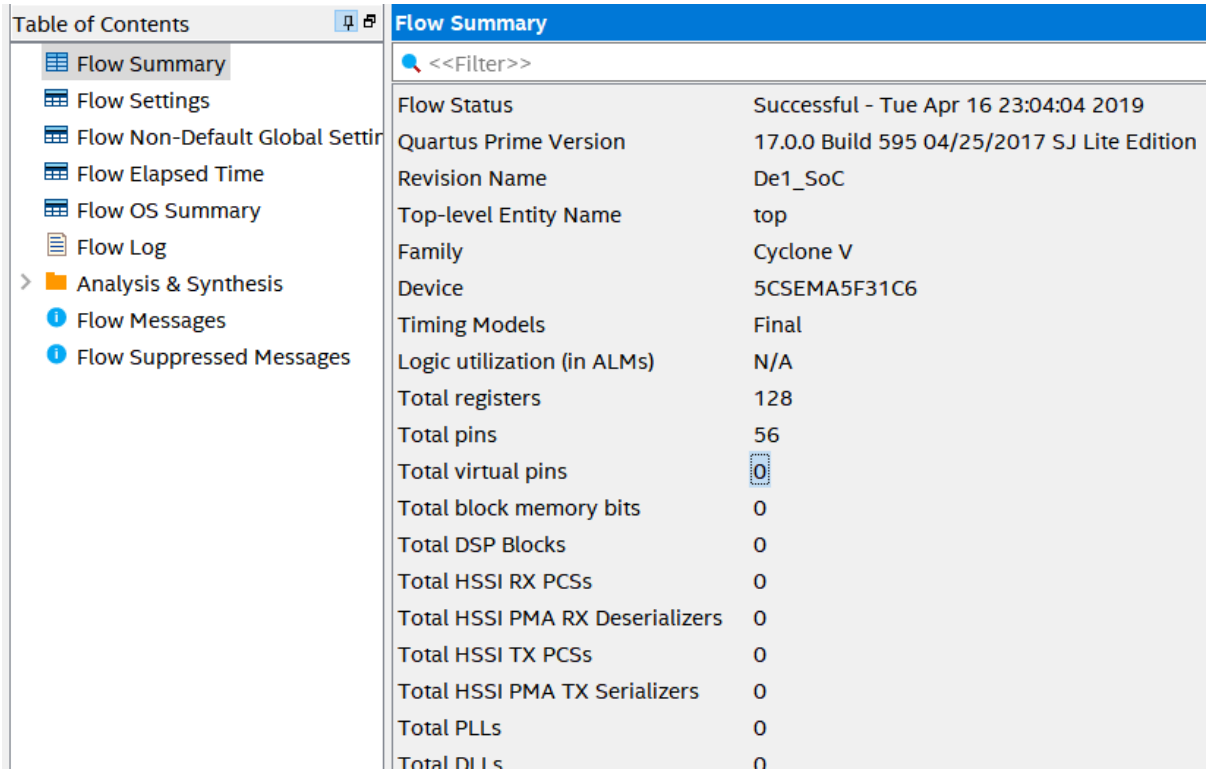

Figure 8: task3 top compilation flow summary

Task 4

The simulation for the top-level driver is depicted in Figure 9-11. The simulation verifies that the dual-port RAM is read and written correctly. It also verifies that the on-board elements control and display the RAM correctly.

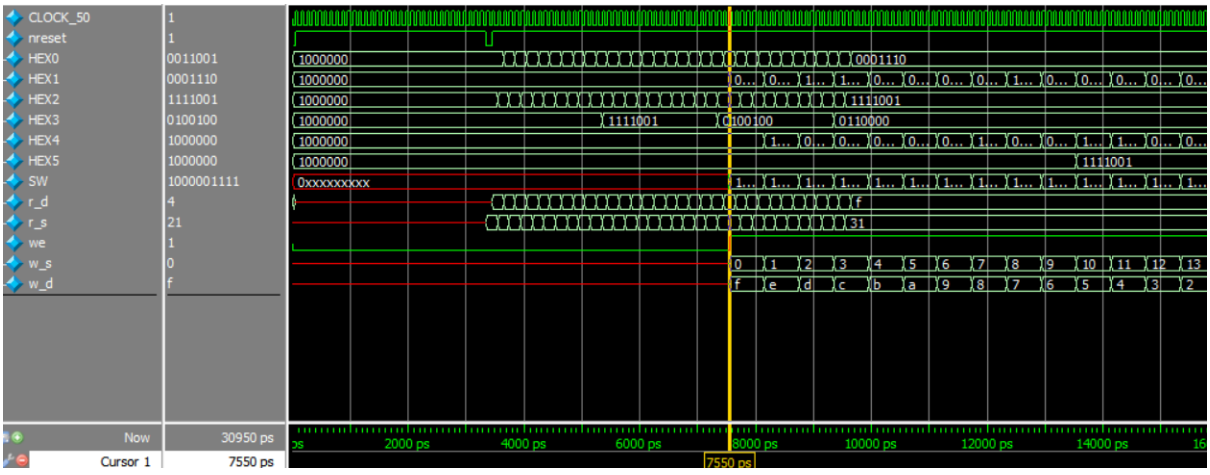

Figure 9: Top Simulation Waveform in ModelSim, part1: reading an initialized memory

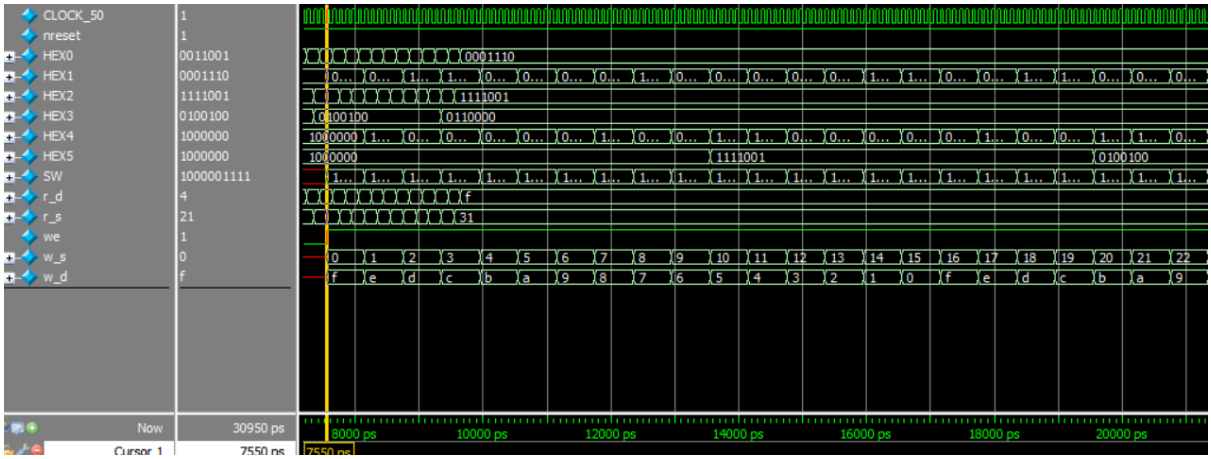

Figure 10: Top Simulation Waveform in ModelSim, part2: writing to every address of the memory

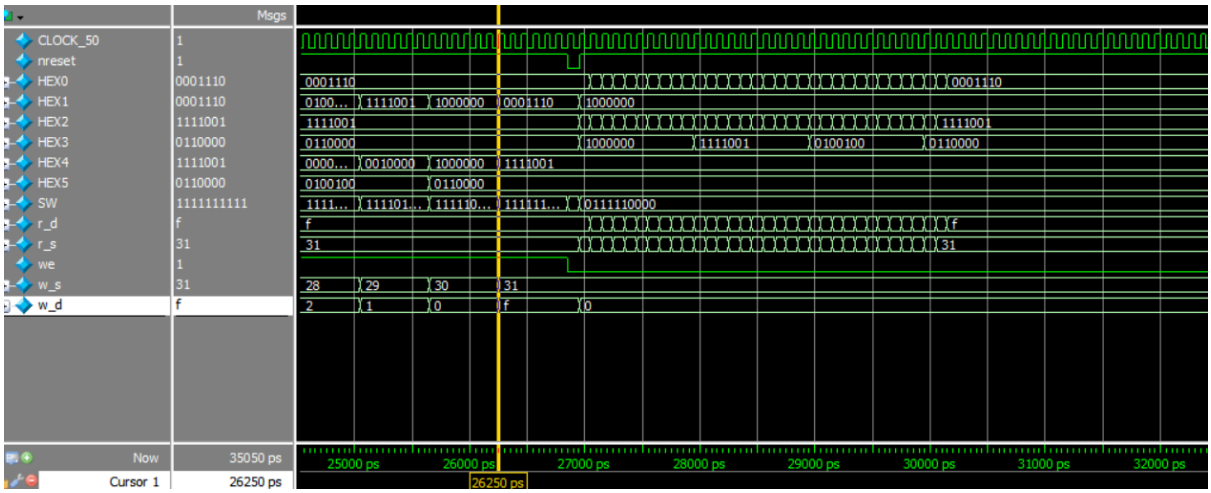

Figure 11: Top Simulation Waveform in ModelSim: part3: reading after the writes and writing while write enable is low

The compilation report is shown in Figure 12.

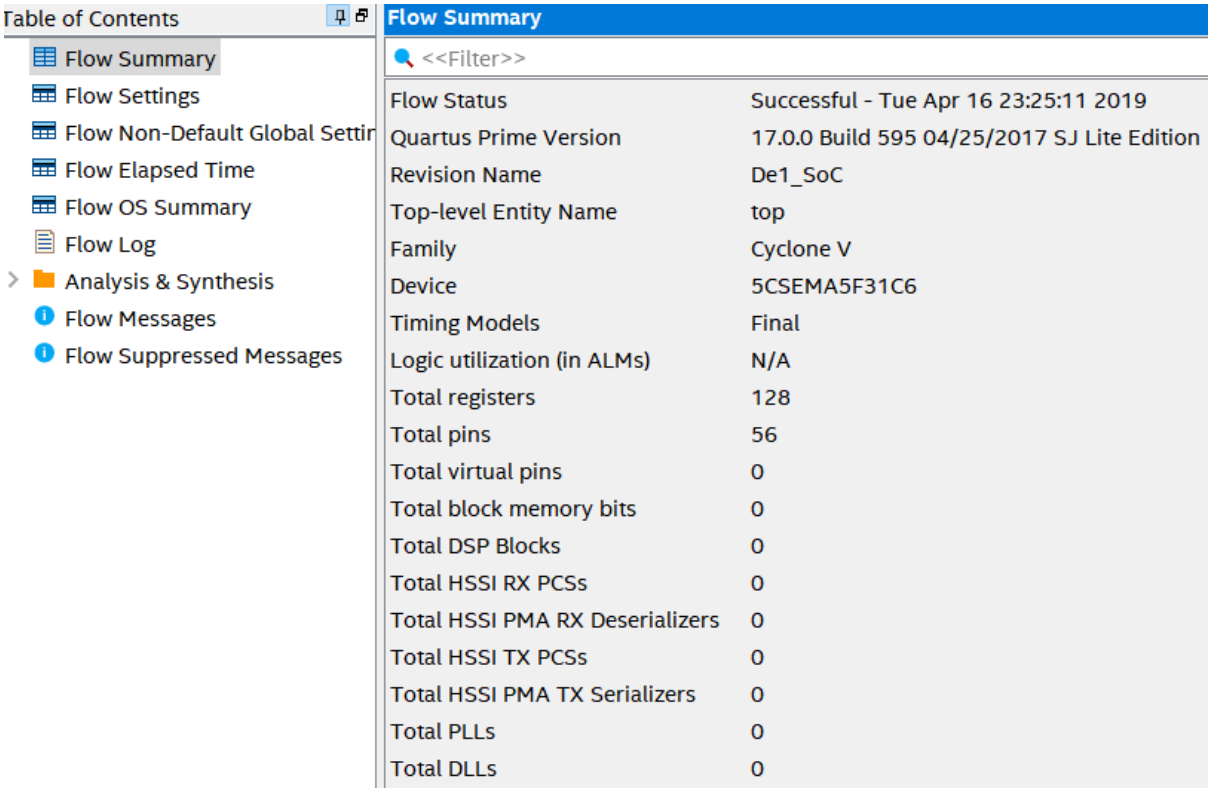

Figure 12: task4 top compilation flow summary

Problem Faced & Feedback

Overall, this lab is fairly straight-forward and helpful in terms of improving the understanding of memory implementation. Approximately 10 hours were spent on completing this lab. The only difficulty lies in a shared reset button that controls both the counter reset and clock reset, causing the counter never gets back to zero since the reset makes the clock low. I overcame it by thinking through all possible causes of the problem before getting into hard debugging. I suggest myself to continue working smart rather than working hard in the future. In addition, I suggest myself to continue simulating the modules before loading to the hardware.