Khoa Tran

EE 371

April 23, 2021

Lab 2 Report

**Procedure**

**Task #1**

Approaching this problem, I first drew up the block diagram to figure out the necessary inputs of the 32x4 ram and how the clock progresses each input into the ram, allowing it to output the data from the address given by SW8-4. The ram only has one address input, both for writing and reading. As the write signal is not true, the ram will output the data at the given address, which will initially be 0. However, as write signal is true, given by SW9, then the ram will write the given data, given by SW3-0, to the address location given and output that same value as a read. Block diagram shows that input address is shown on HEX5, HEX4, data on HEX2, and dataOut on HEX0.

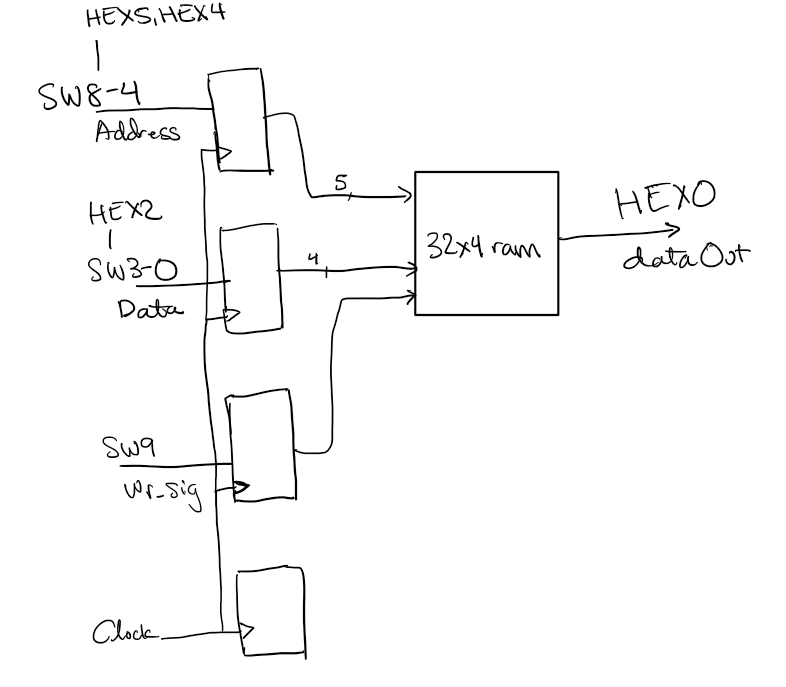
****

Figure 1: Block diagram for task 1

**Task #2**

Approaching this problem, I first drew up block diagram for the 32x4 ram as the ram now has an input for read address and write address. The read address is taken from a counter that is implemented by incrementing the value of the address by 1, for every 1 second, running in 1 Hz clock. This can be seen in figure 3. The ram also takes the input of SW8-4 for the write address, and SW3-0 as the data input and KEY[3] as the write signal. The ram inputs the data from the given address, running on CLOCK\_50. The output data of the 32x4 ram is the data at the read address of the counter. Using the IP Catalog, I was able to set the memory initialization of the ram so the output data would be different than all zeros to start with, this output can be seen on HEX0. Write address can be seen on HEX5 and HEX4, while read address on HEX3 and HEX2 and input data on HEX1.

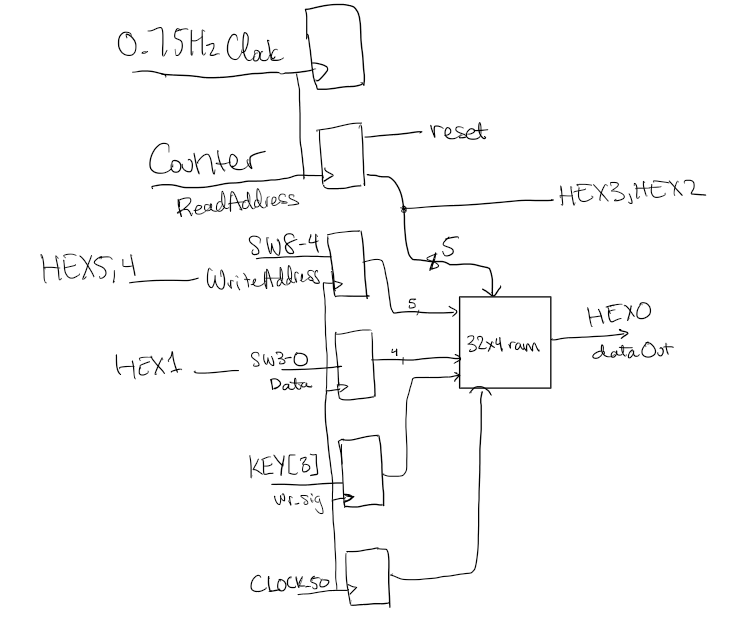


Figure 2: Block diagram for task 2

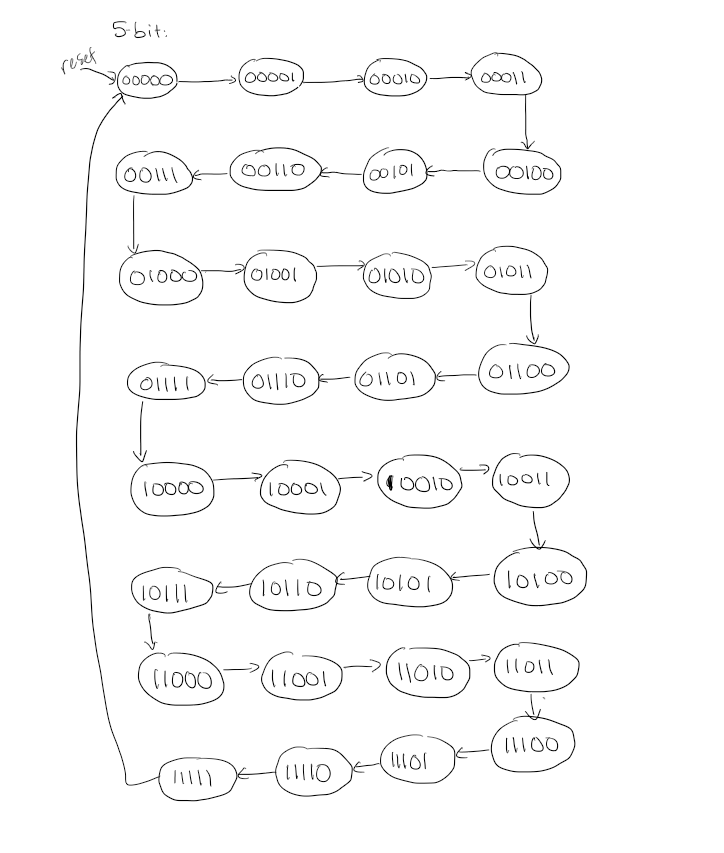
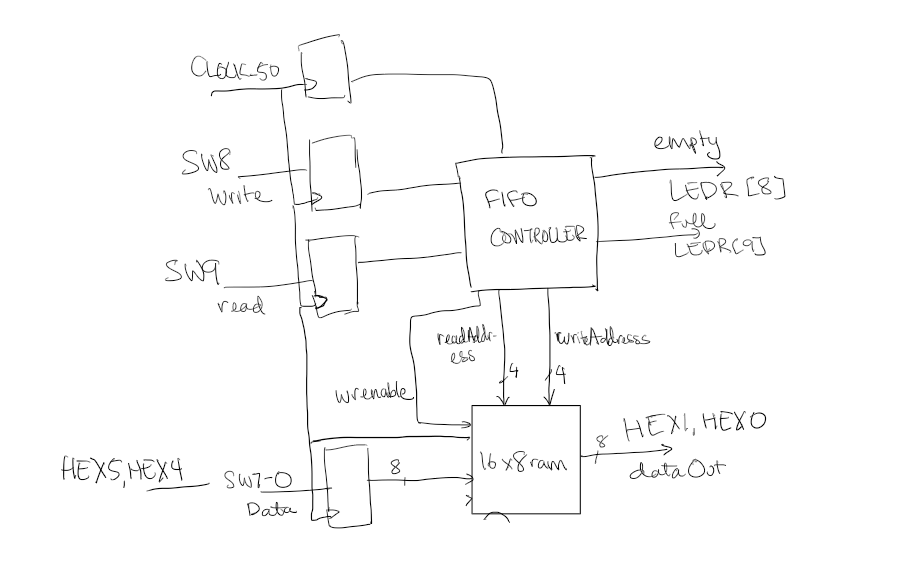
****

Figure 3: State Diagram for Counter

**Task #3**

Approaching this problem, I first drew the block diagram to design the interaction between the FIFO controller and the 16x8 ram as the ram requires read and write address as well as write enable from the FIFO as it determines the location to write and read based on a queue, implementing the idea of first in first out. Since the ram has to read the least recent data that is written to the ram, the FIFO controller takes in inputs of write and read in order to determine if the user wants to write or read and where the respective addresses of where the ram should store the data to continue the queue and where the read address has to be to maintain least recent read. To figure out the FIFO controller, I drew a state diagram showing the different states of the ram and how the different inputs of write and read from SW8 and SW9 affects the output of read and write pointer as the output address for the ram’s input and as well as full and empty. This can be seen in figure 5.



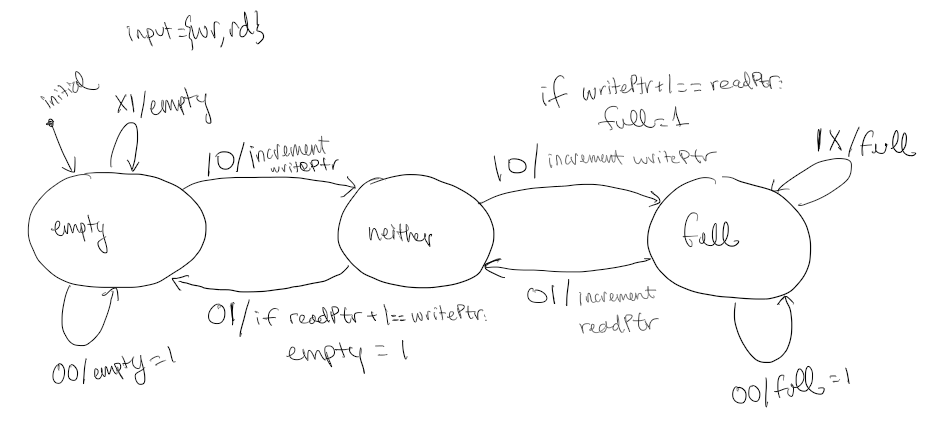
Figure 4: Block diagram for task 3

Figure 5: FSM for FIFO Controller

**Results**

**Task 1:**

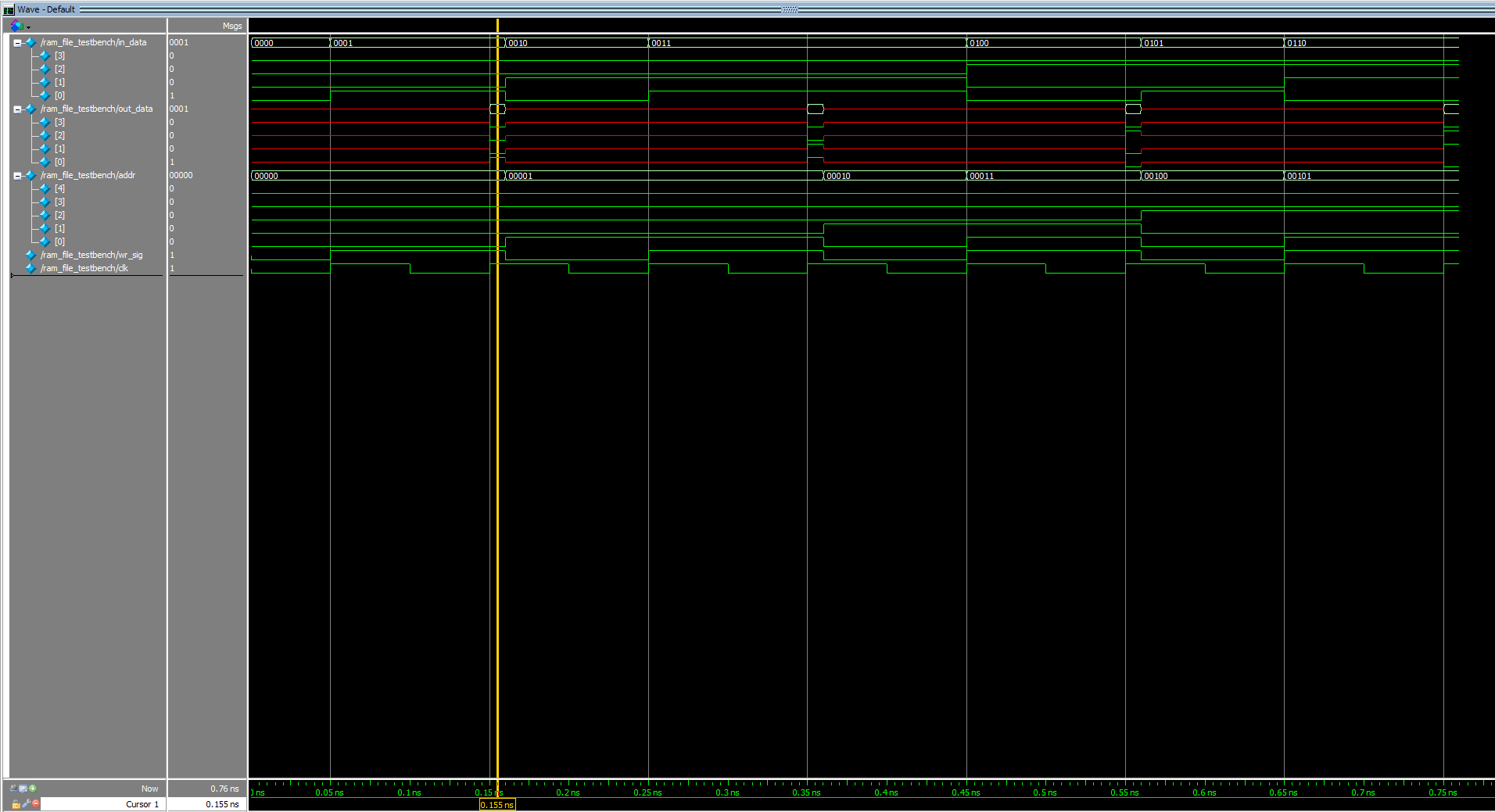
For the first part, I tested if the ram file module as the it goes through a series of inputs on in\_data, addr, and wr\_sig, seeing if the out\_data is correct after each write into the ram. The inputs are given to the ram at the positive edge of the clock, while out\_data will always output the data at the given address. 

Figure 6: Waveform simulation for ram\_file

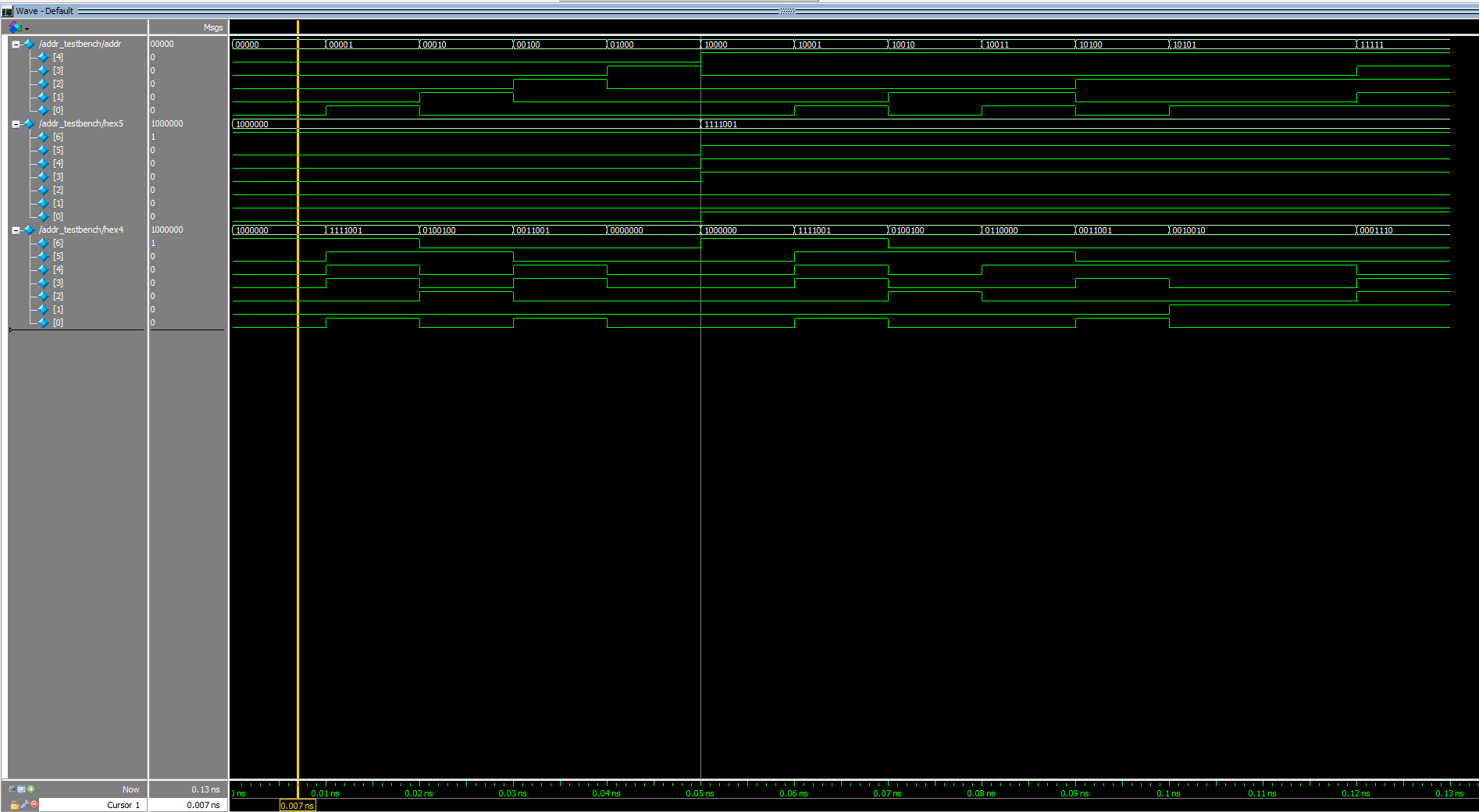
I also created two files that follows combinational logic, outputing to hex display the hexadecimal value of the address and data given. The waveform simulation is below as the output for the hex display matches the value of the data or address given, seen in figure 7 and 8.

Figure 7: Waveform for display address

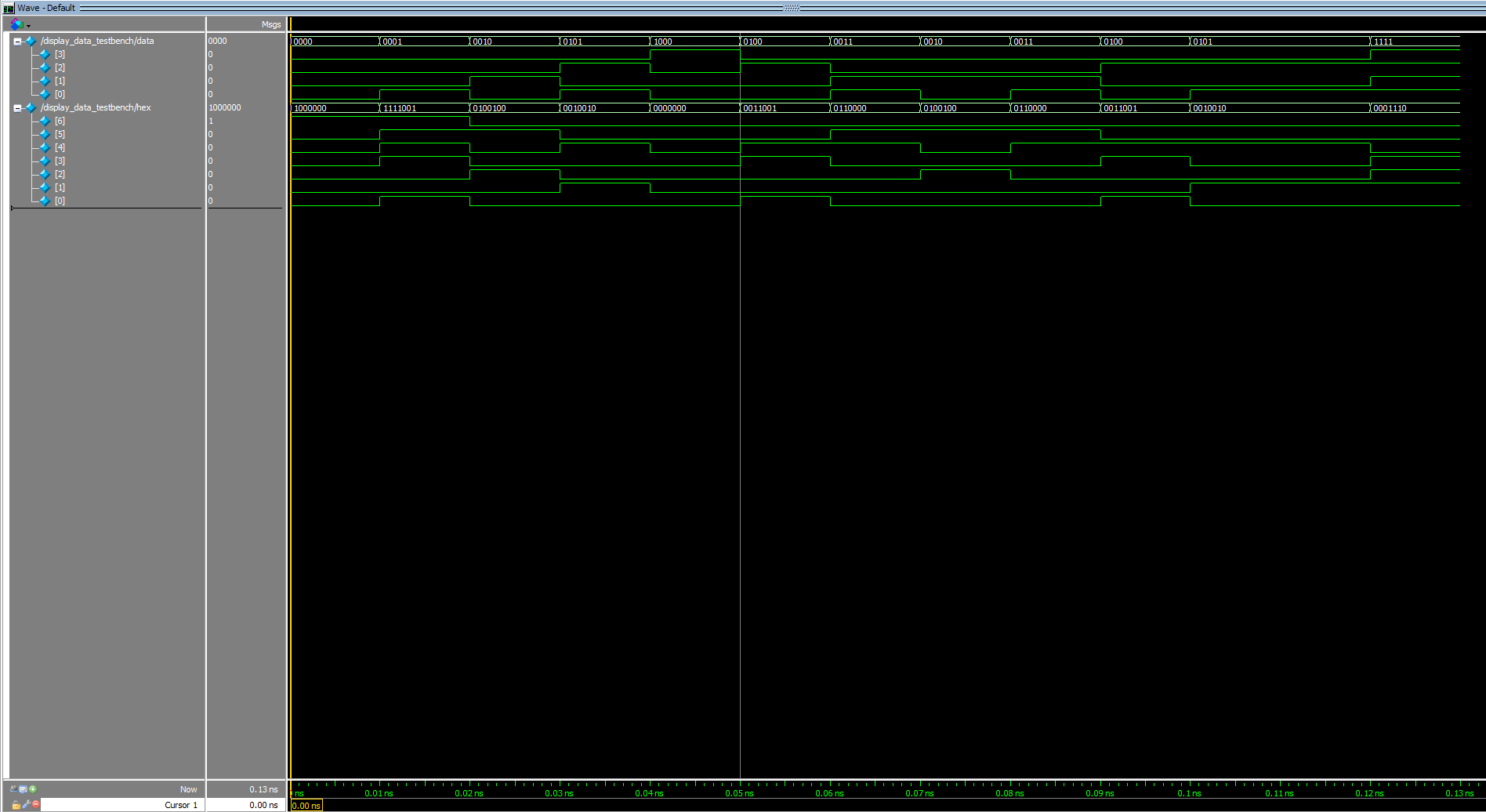


Figure 8: Waveform simulation for display data

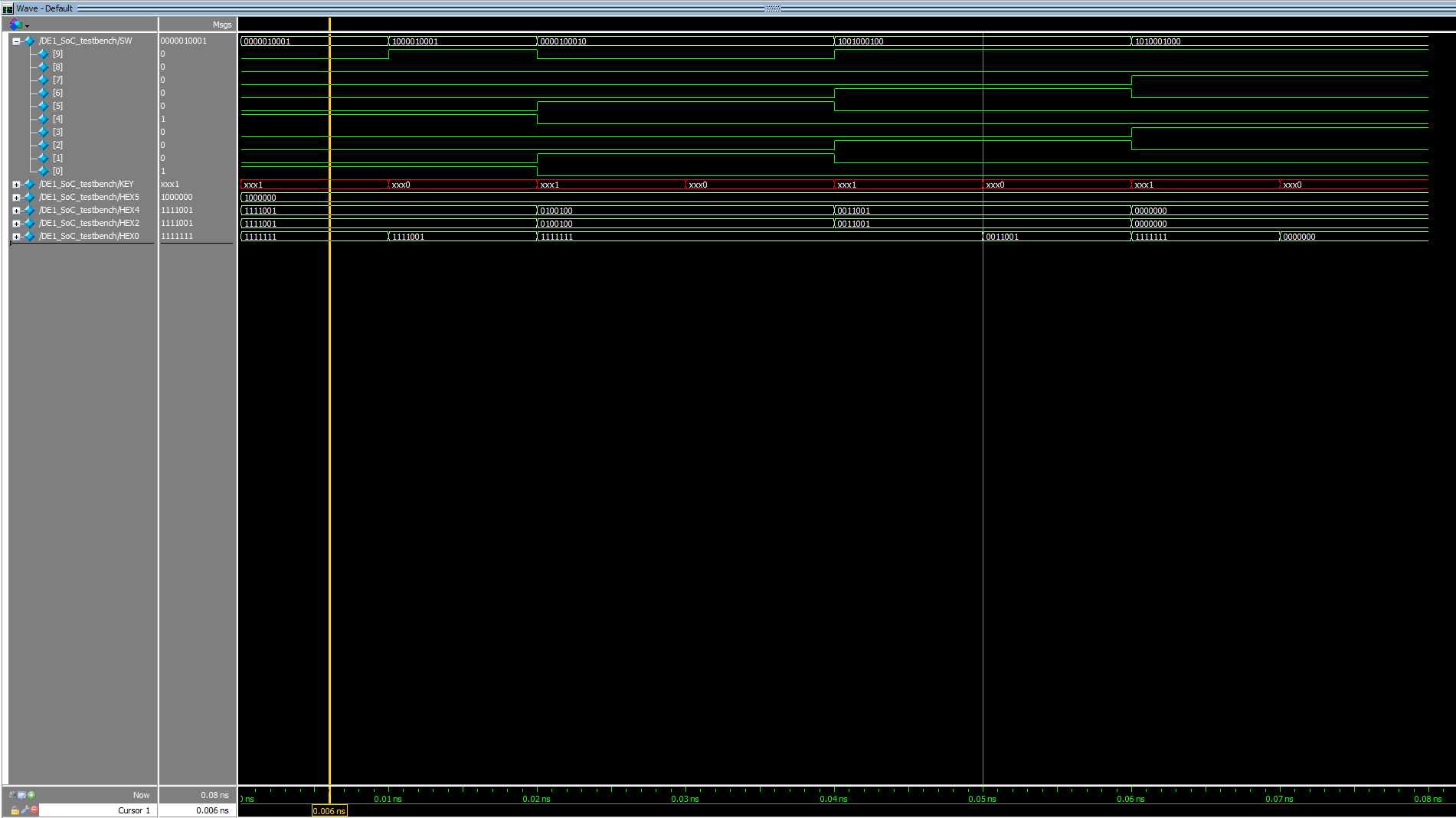
For DE1\_SoC, The address is taken from SW8-4, as the data is taken from SW3-0. The signal to write to the ram is given from SW9. As the clk that is set to KEY[0] goes to posedge, the ram writes the given data into the given address. However, the output on HEX0 will be the data at the address given by SW8-4. As seen in the simulation in figure 9, HEX5 and HEX4 displays the address value using hexadecimal and HEX2 for input data.

Figure 9: The waveform simulation generated by the DE1\_SoC for task1

**Task 2:**

This task replicated the exact same ram as the previous task with the size of 32x4, however, there are two address inputs, one for the write address and one for the read address. The write address and the data is given from the user using the switches. While the read address is taken from a counter, incrementing by 1 every second. As the output data from the ram is from the read address location. As seen below in figure 10, the counter is incrementing by 1 in order to output the read address for the ram to receive and output the data. 

Figure 10: The waveform simulation generated by counter module

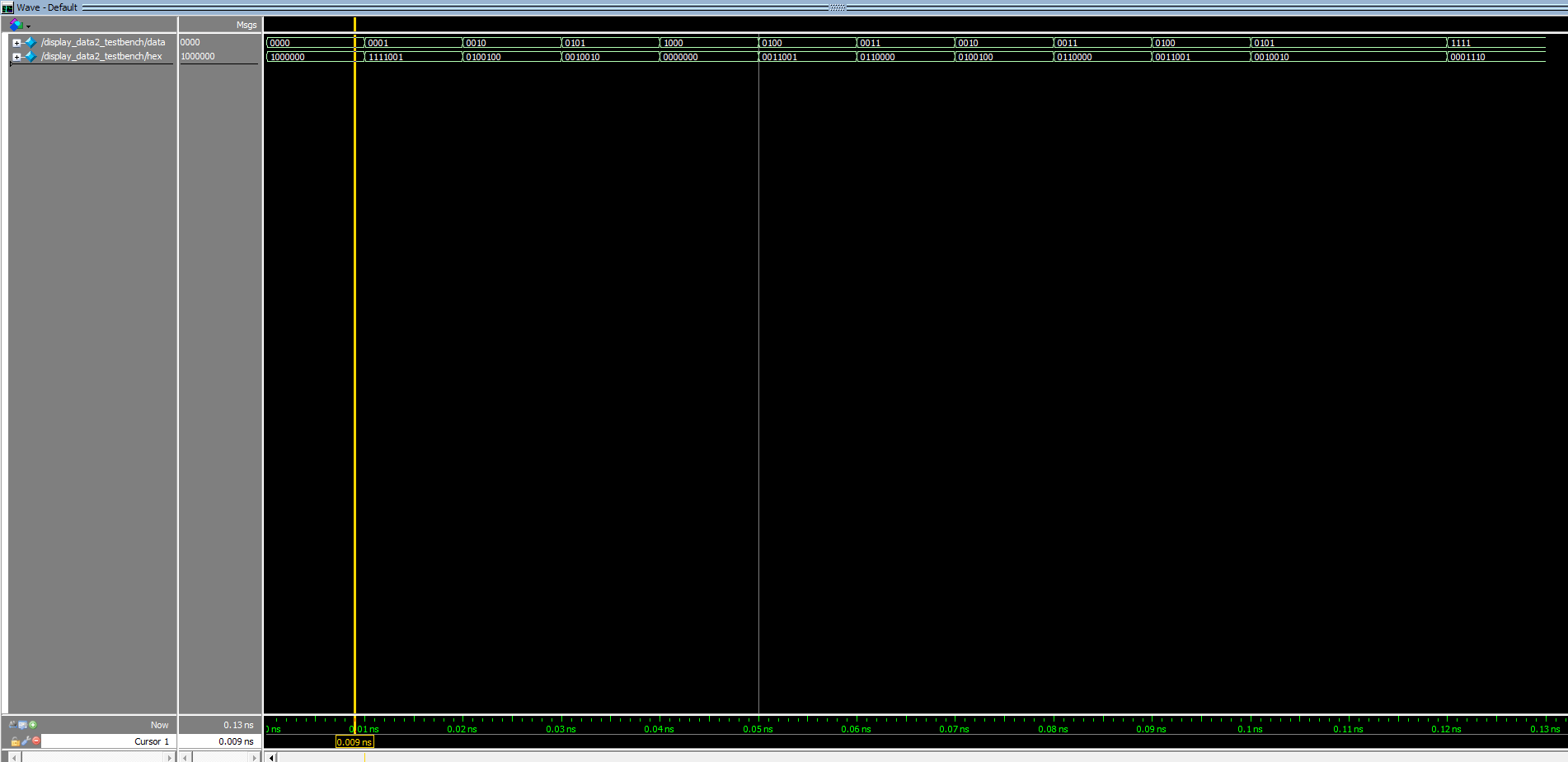
I also created two files that follows combinational logic, outputing to hex display the hexadecimal value of the address and data given. The waveform simulation is below as the output for the hex display matches the value of the data or address given, seen in figure 11 and 12, similar to modules in task 1.

Figure 11: Waveform simulation generated by display data 2 module



Figure 12: Waveform simulation generated by addr2 module

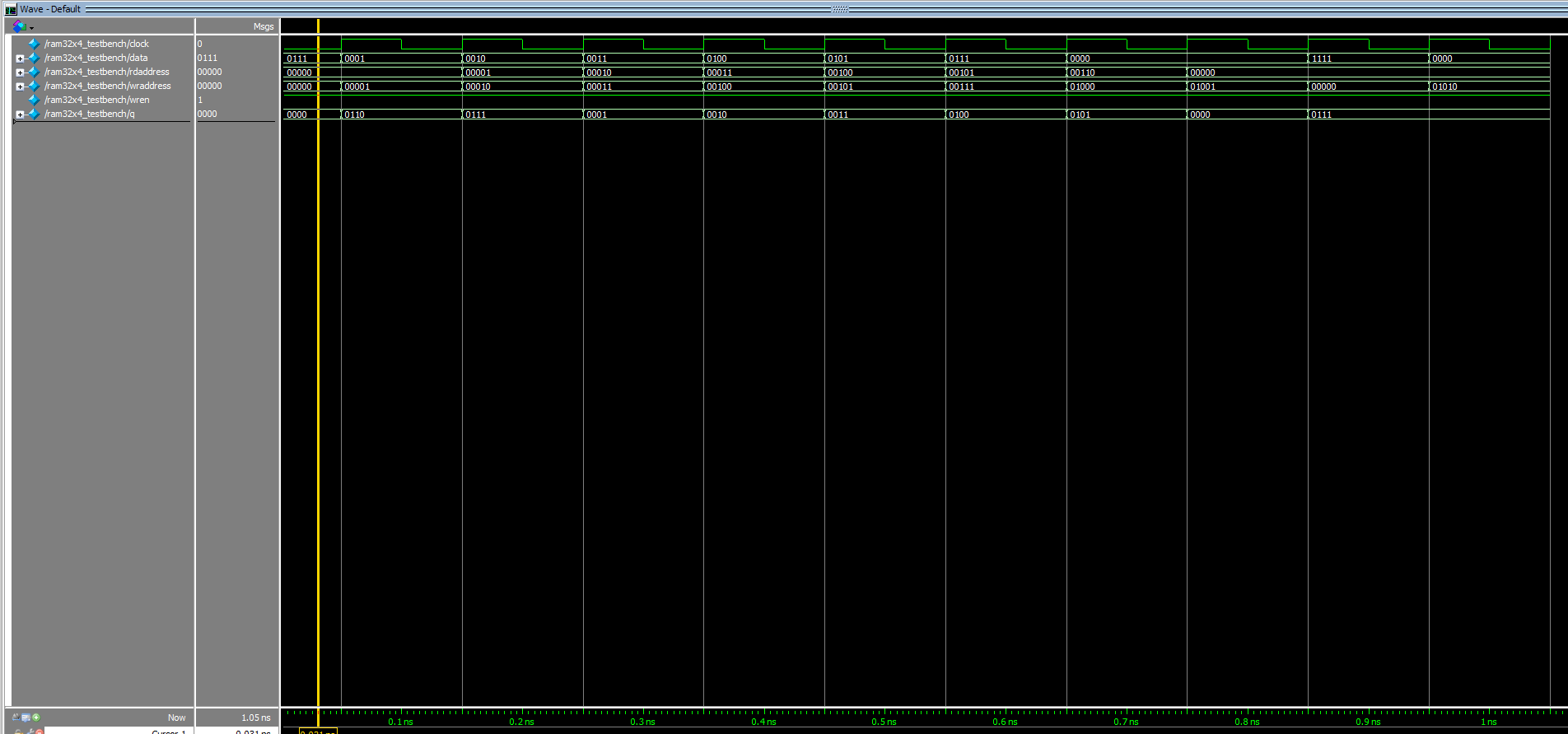
Below in figure 13 is the waveform simulation for ram32x4 testing for the output of q, as the inputs of rdaddress, wraddress, data, and wren testing if the q output is correct based on the given read address as the data that outputs is the value from the memory initialization file, for the initial memory of the ram, seen in figure 14.

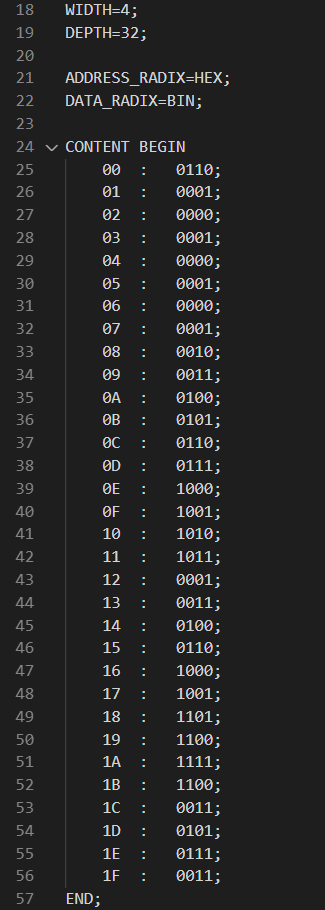
Figure 13: Waveform simulation generated by ram32x4 module

Figure 14: ram32x4 memory initialization file

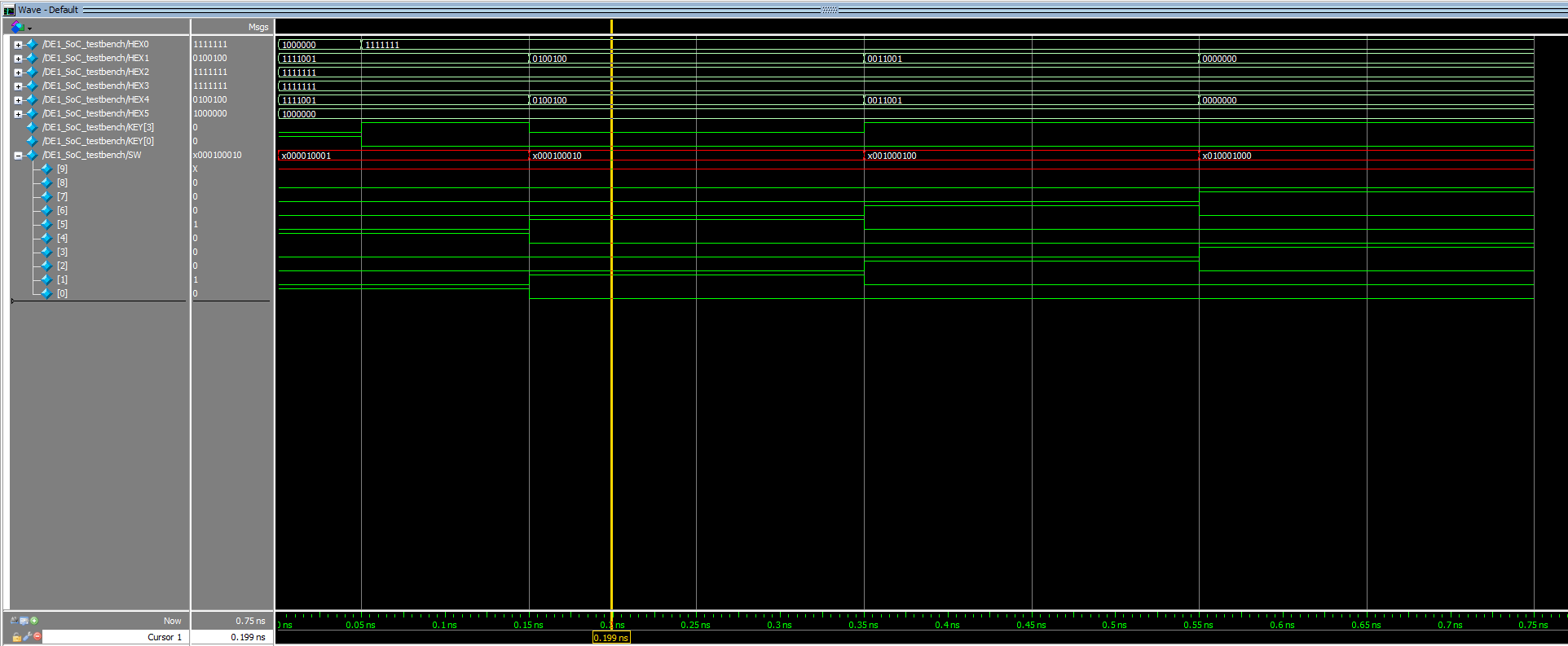
Below is the simulation for DE1\_SoC for task2, as SW8-4 is used for the write address and SW3-0 is used for the data displayed on HEX5, HEX4, and HEX1. The KEY0 is used as reset input and KEY 3 is used for wr\_en. As seen on HEX0, the ram outputs the data and is displayed on the 7-segment in hexadecimal and the address on HEX3 and HEX2.

Figure 15: Waveform simulation for DE1\_SoC for task 2

**Task 3:**

This task implements a concept of a queue for the ram, having data being first in, first out. As in the ram will only read the least recent data when the user desires to. As well as only writes when the ram isn’t full and the user wants to write. In the simulation below, the FIFO control is being tested in a series of inputs on read, write, and seeing if wr\_en, full, empty, write address, and read address is correct along each positive edge of the clock. As seen in the simulation in figure 16, the outputs follow a finite state machine that I designed in the procedure, as the state of empty is initially true and progressing through a series of writes in order for full to output true.

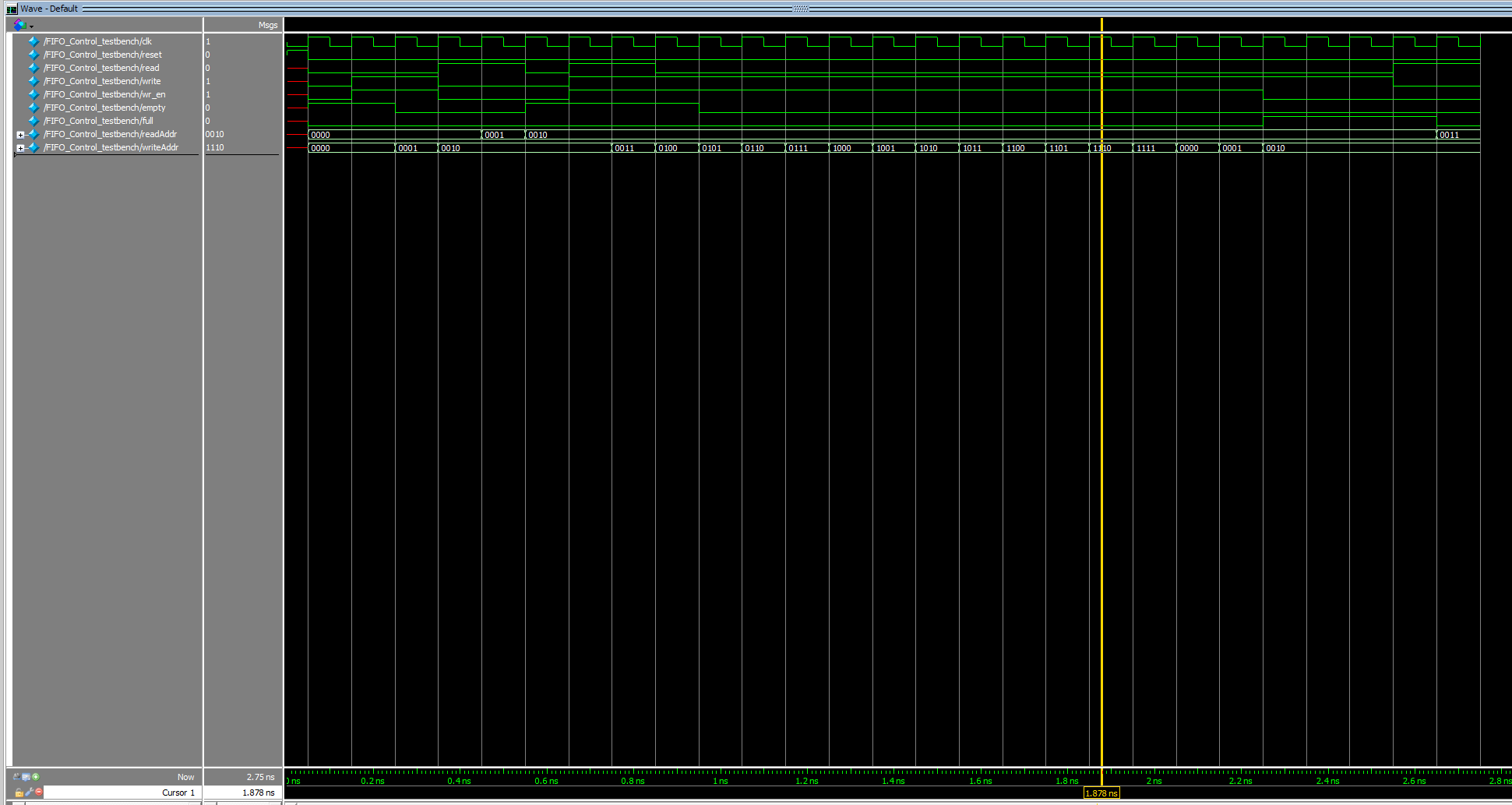
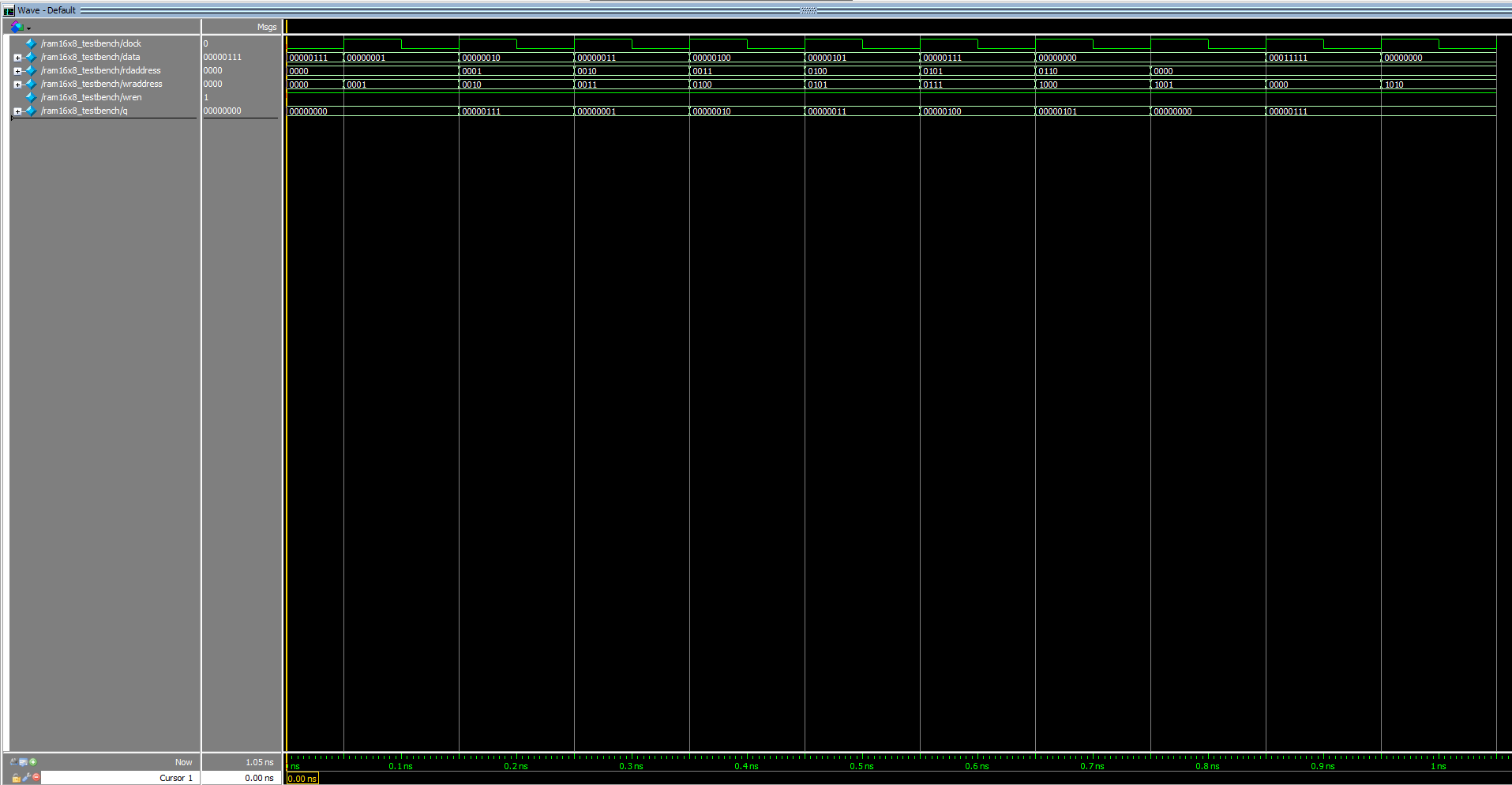
****

Figure 16: Waveform simulation for FIFO Control module

The simulation below is for the 16x8 ram as test for the output of q, as the inputs of rdaddress, wraddress, data, and wren testing if the q output is correct based on the given read address as the data that outputs is the value from the memory initialization file, for the initial memory of the ram, as it is empty to start with.

Figure 17: Waveform simulation generated by ram16x8 module

As well as modules for displaying the data of the input and the output data, the addition with the output data module is that it implements SW9, only outputting the value if SW9 is also true as the user intends to read.

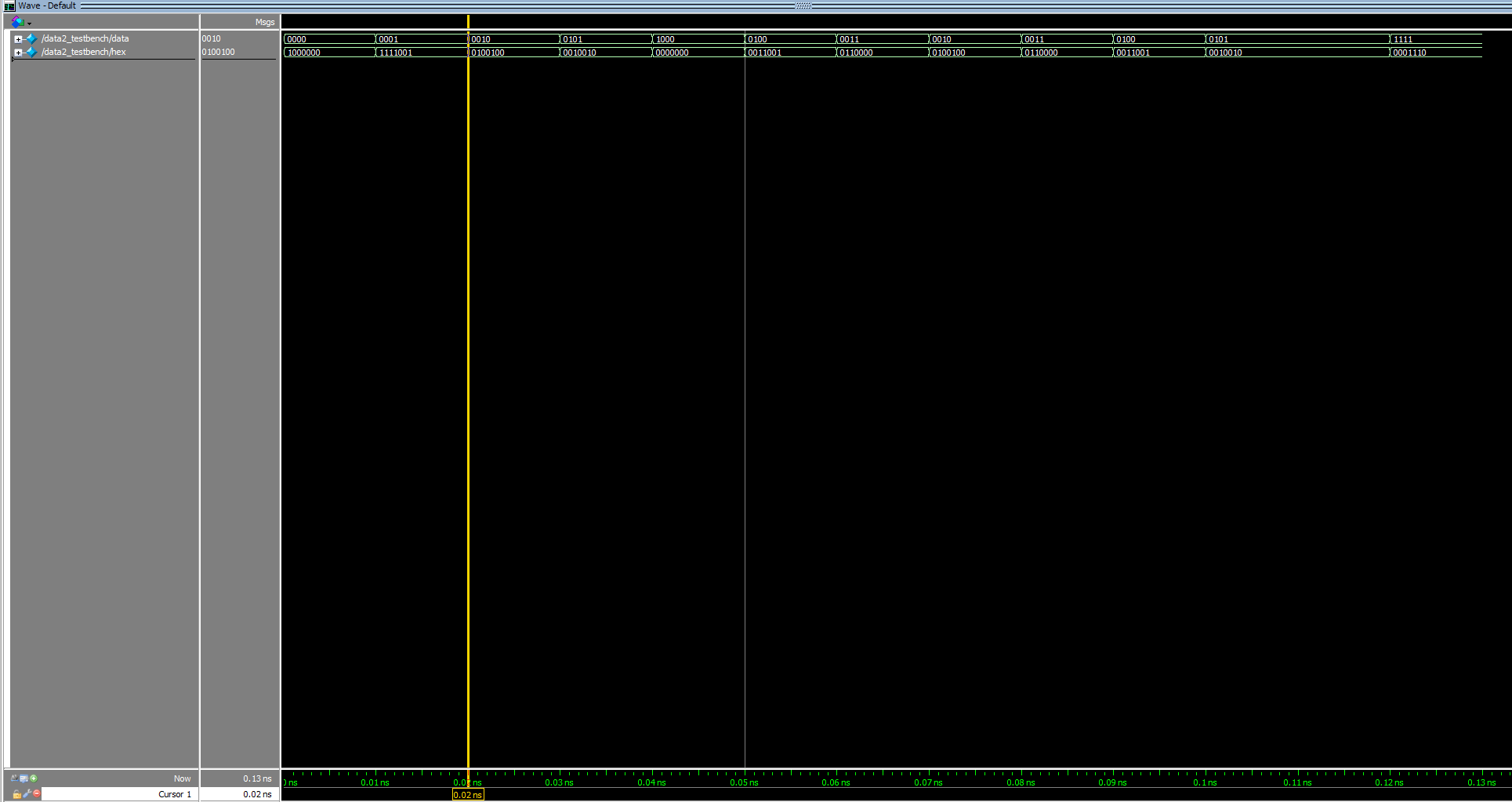


Figure 18: Waveform simulation generated by data2 module

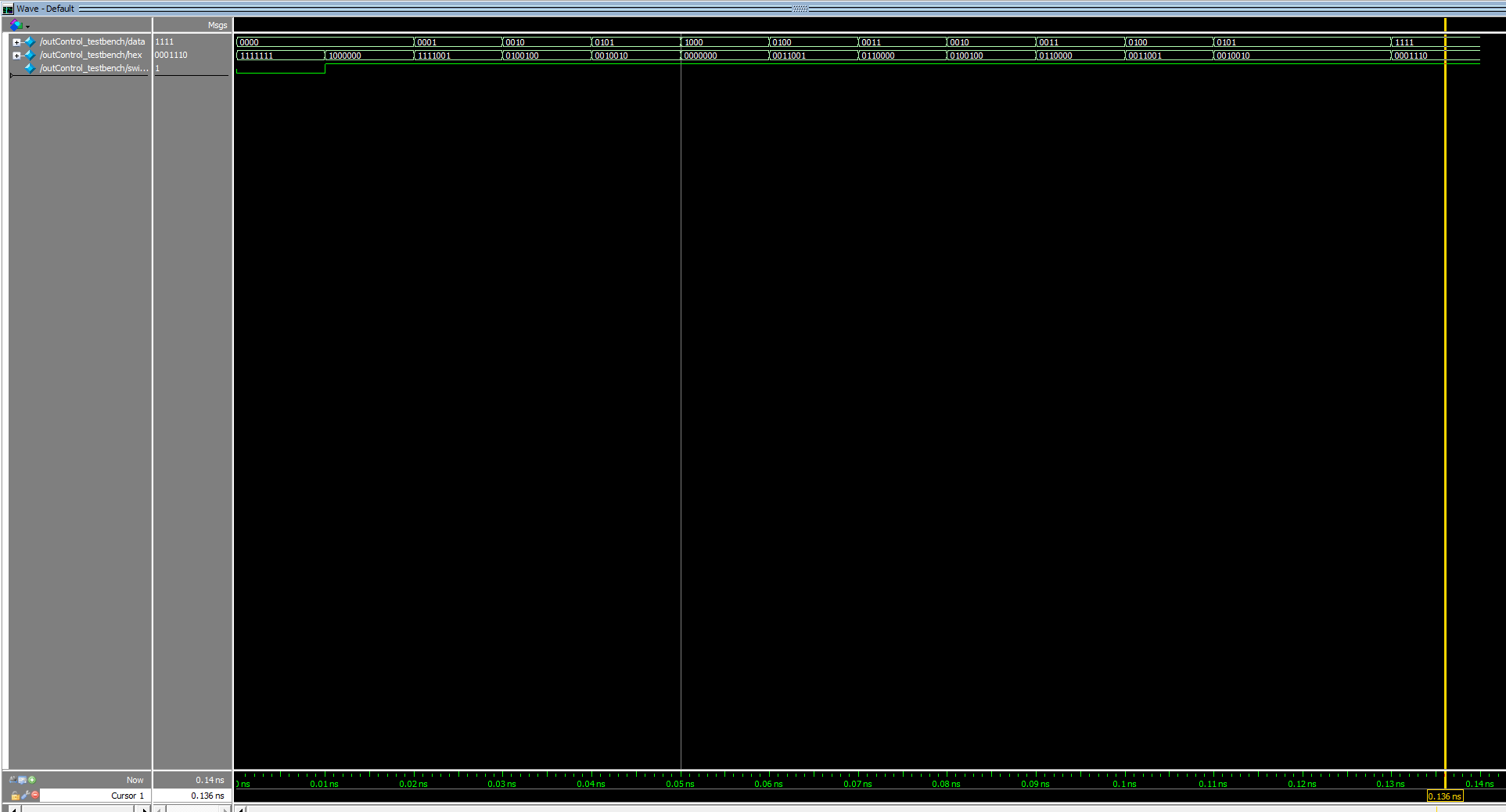


Figure 19: Waveform simulation generated by outControl module

For the simulation of DE1\_SoC, the SW7-0 is the data input as SW8 and SW9 is the write and read inputs for the FIFO, as seen in figure 20, the current data output of the FIFO is on HEX1-0, and LEDR9 shows full, and LEDR8 shows empty. As well as HEX5-4 displaying the value of input data in hexadecimal. As the inputs of SW8 and 9, the ram will in written in the order of a queue, implementing first in first out for read data.

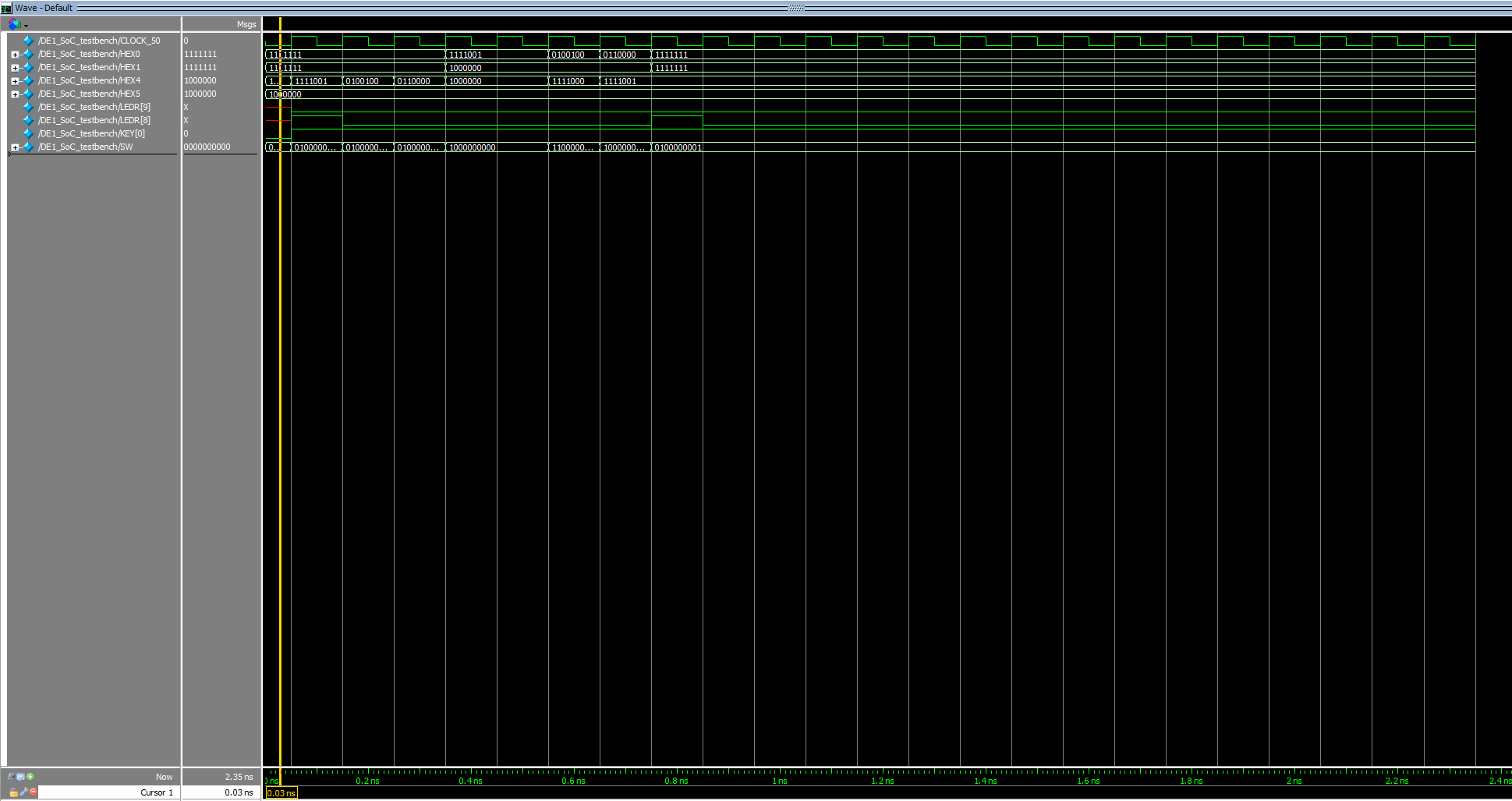


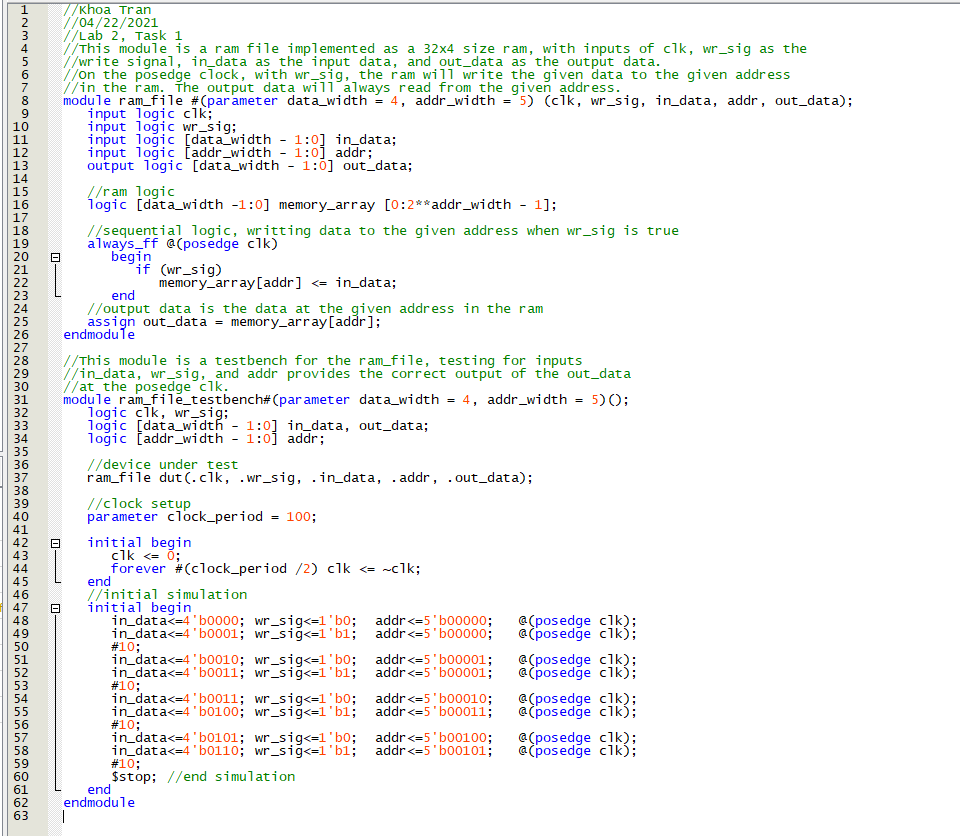
Figure 20: Waveform simulation generated by DE1\_SoC for task 3

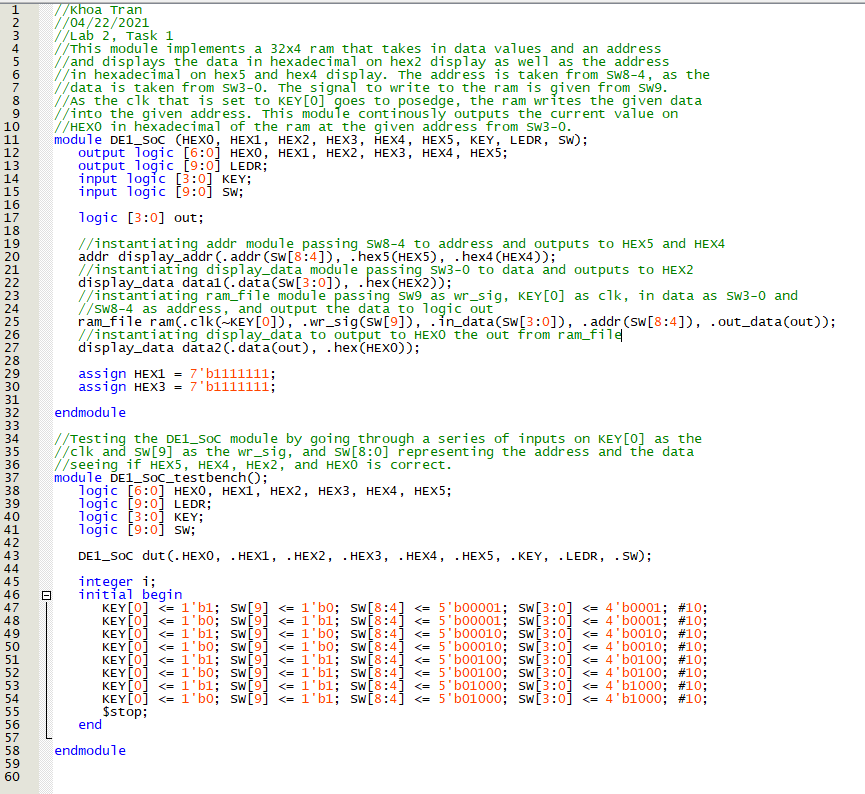
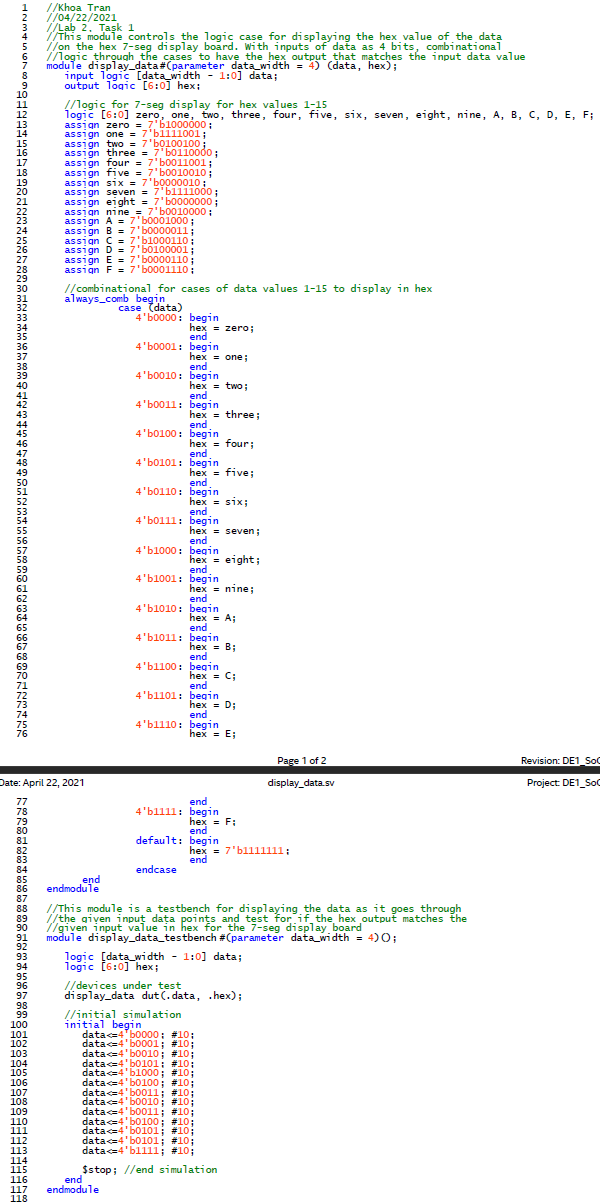
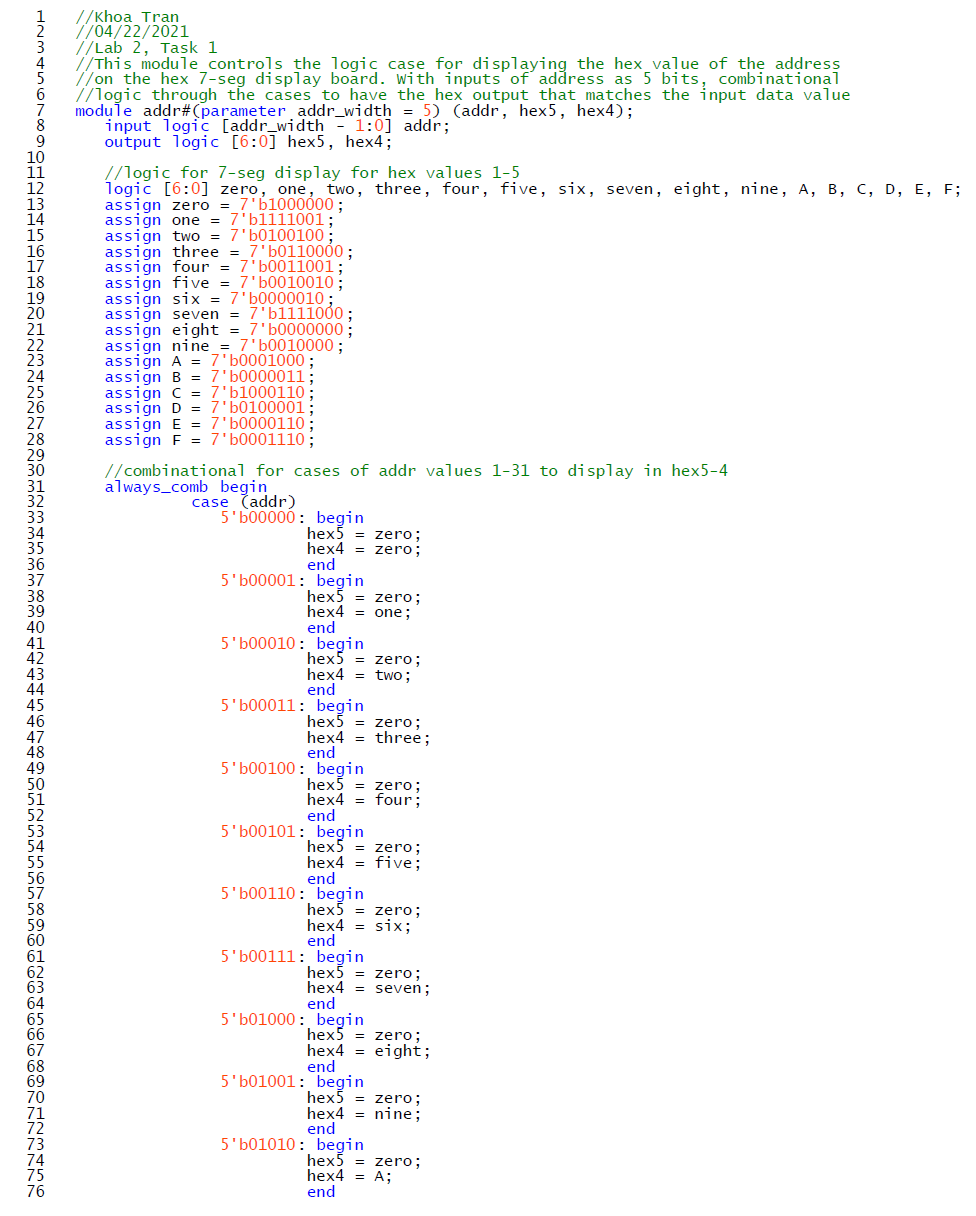
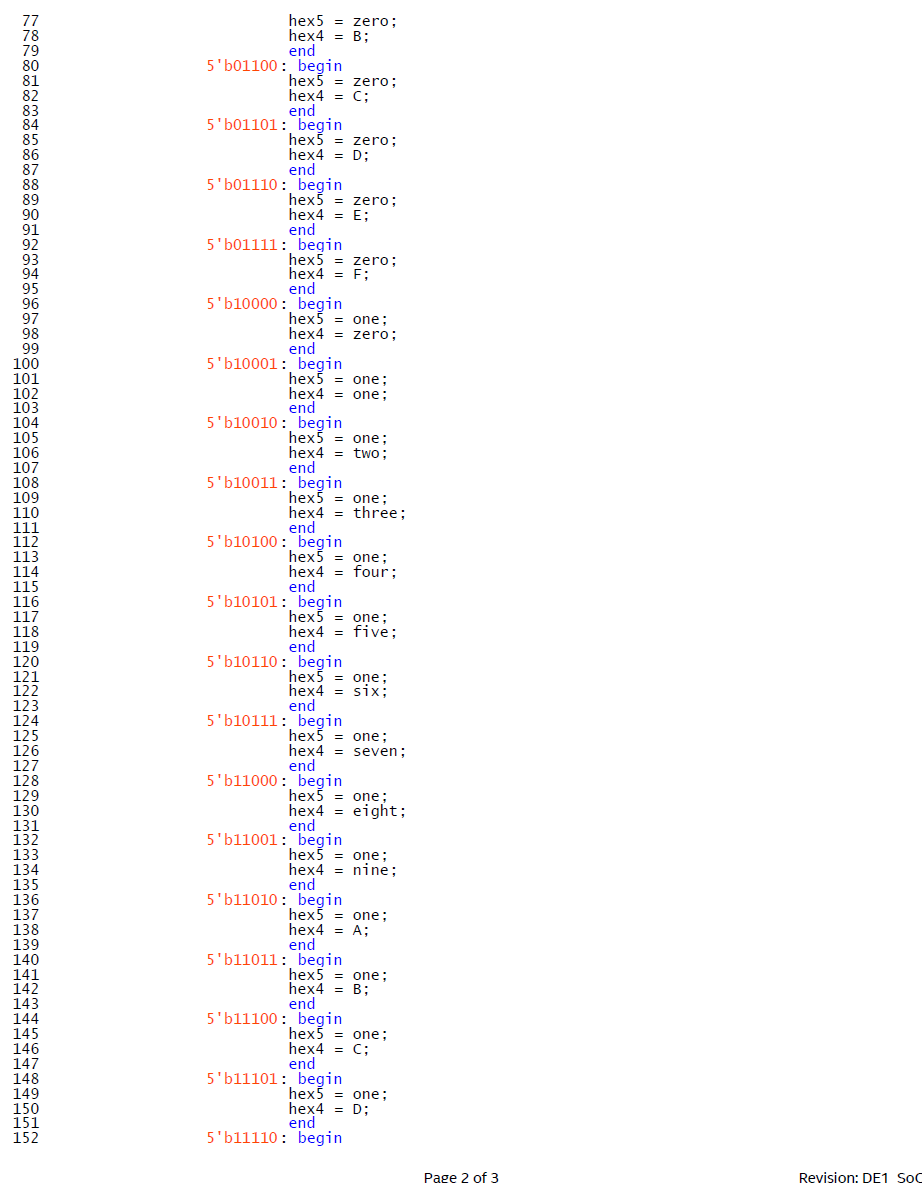
**Final Product**

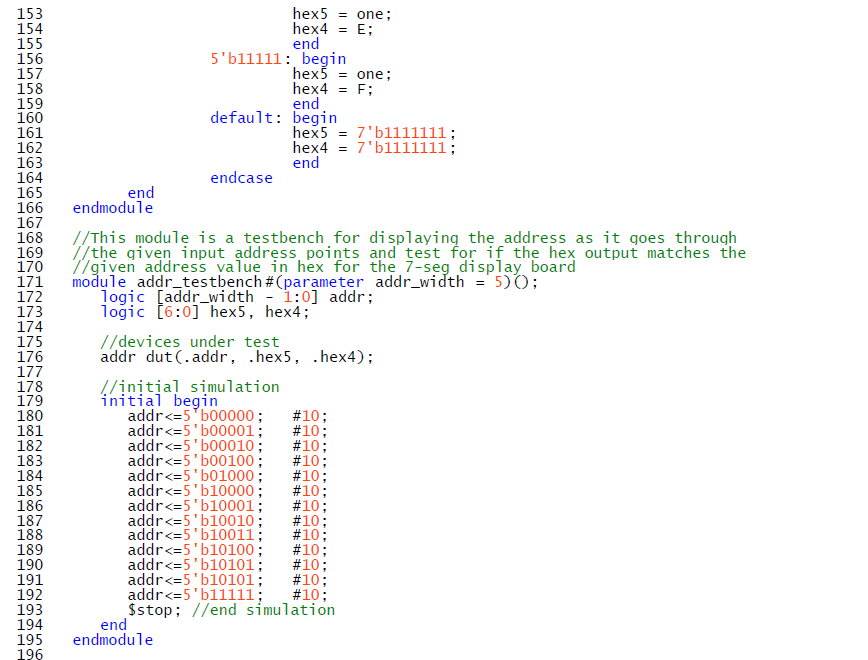
The overarching goal of this project was to design a ram that was able to read and write based on given data and write and read addresses. The project goes further in depth as task 3 implements a system of storing and outputting data that was in a first in first out basis like a queue. The goal was to create a FIFO controller that is a finite state machine, with inputs read and write determining the output and the current state of the state. Overall, I was able to complete the project and done what was asked. There wasn’t any issues and everything that was asked in each task was completed to the expectation and simulation and demonstration of the lab allowed me to understand the transitions between modules and systems allowing the finish product to be similar to what is asked.

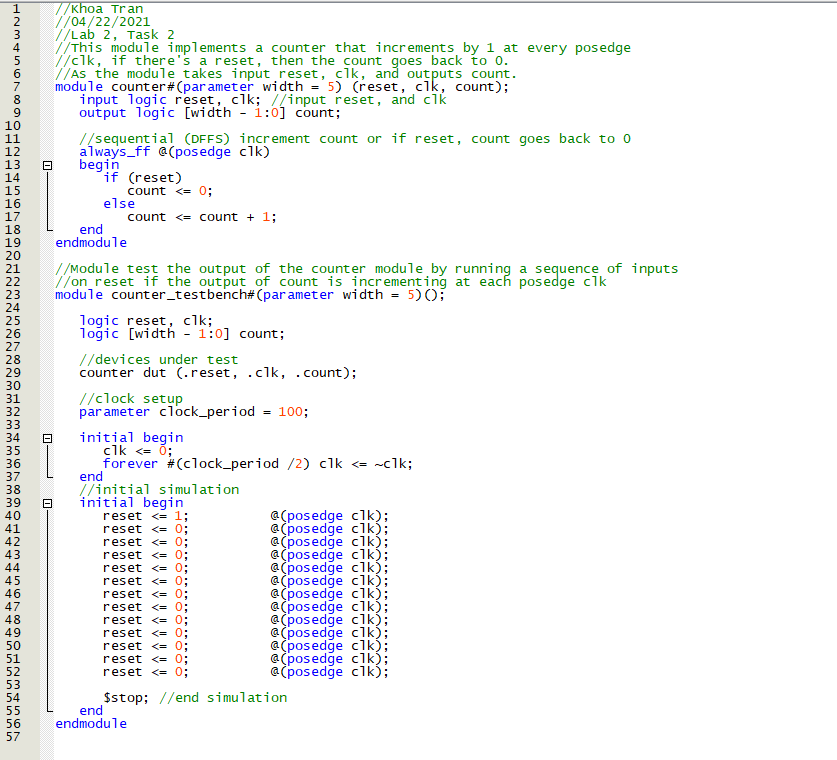
**Appendix: SystemVerilog Code**

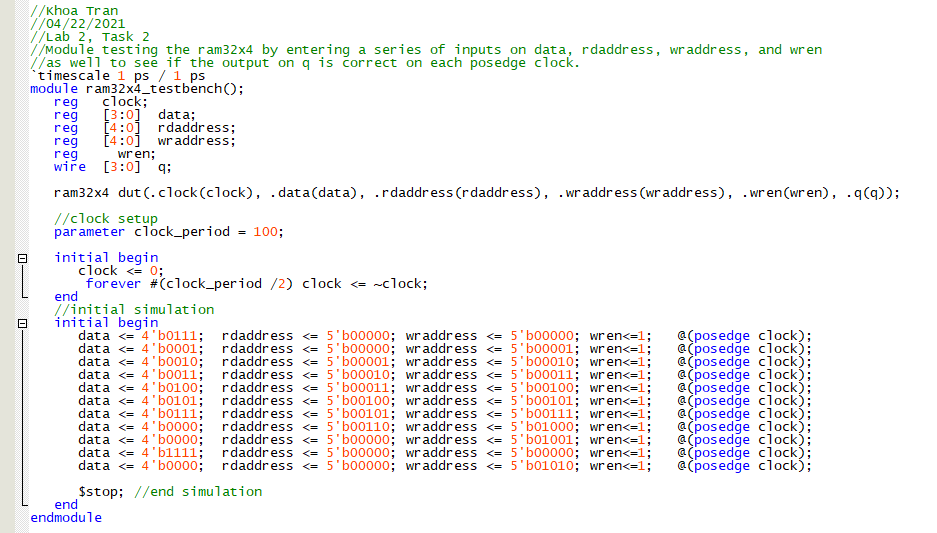
1. **ram\_file.sv (task 1)**



1. **DE1\_SoC.sv (task 1)** 
2. **display\_data.sv (task 1) **
3. **addr.sv******

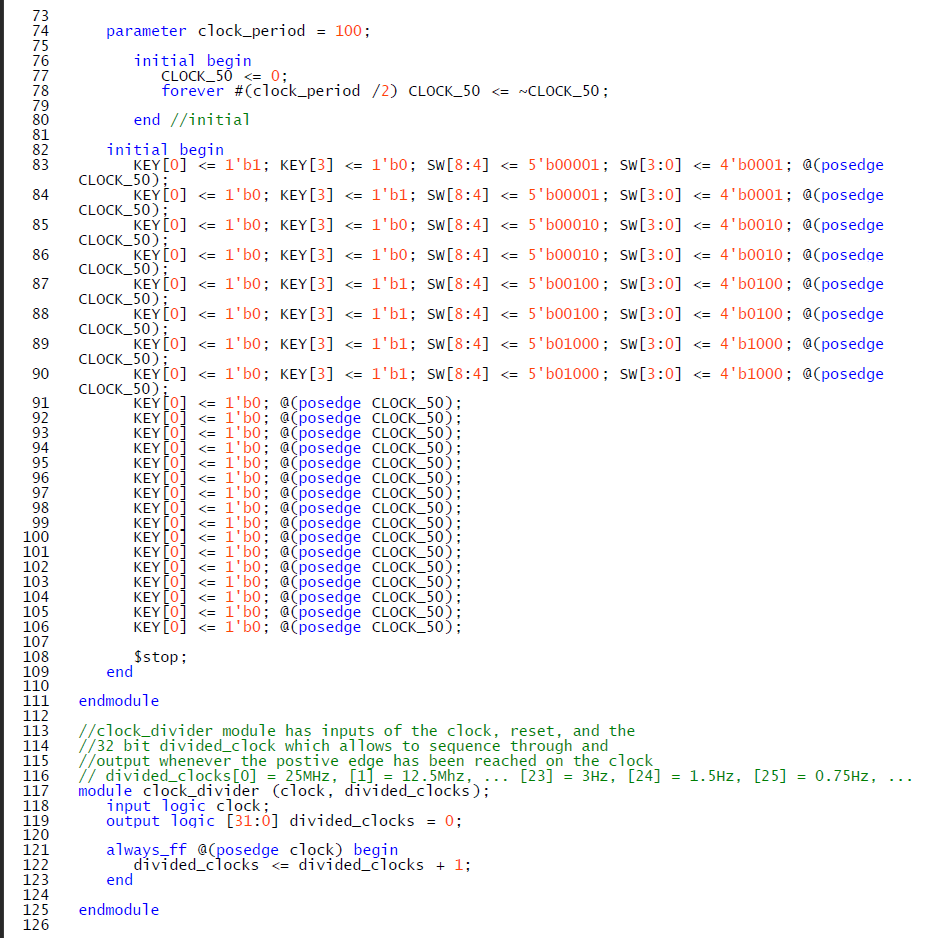


1. **counter.sv (task 2)**
2. **ram32x4\_testbench.sv (task 2)**

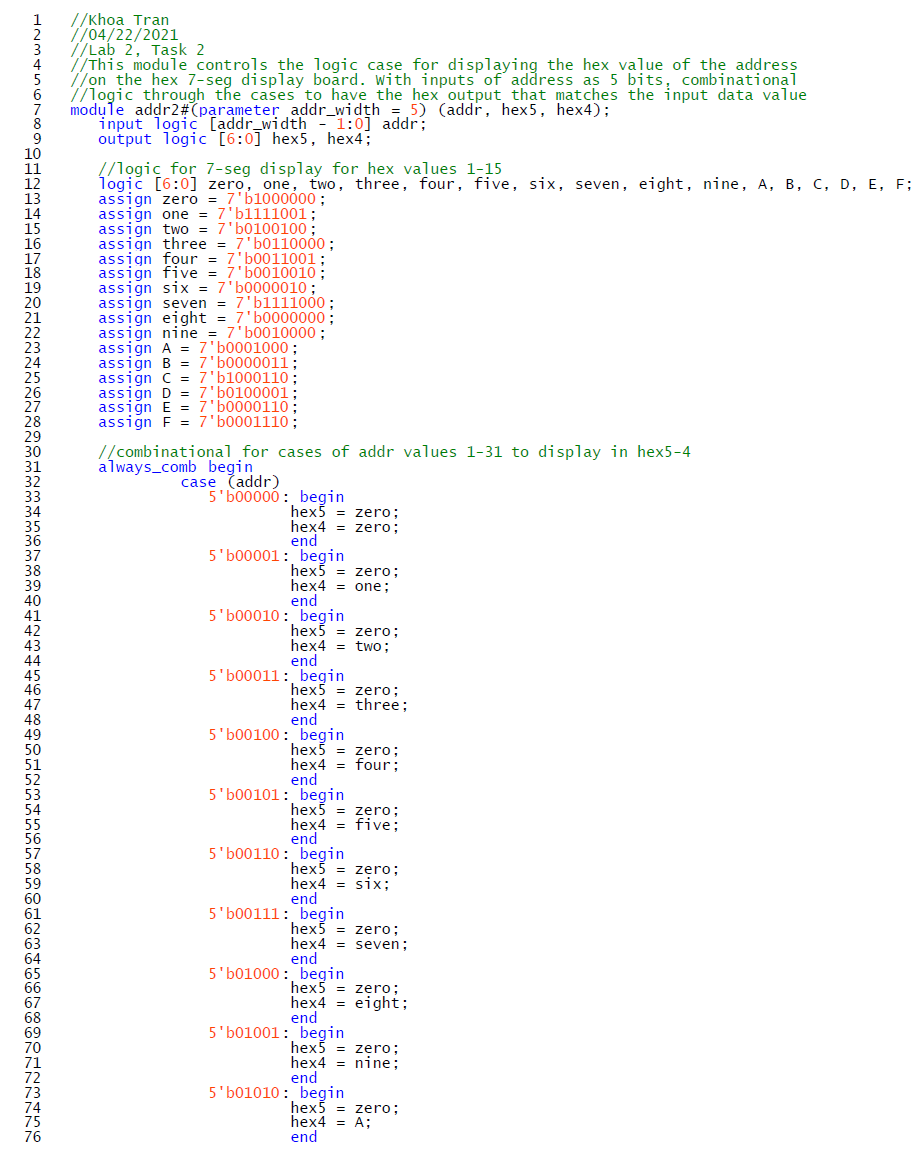
****

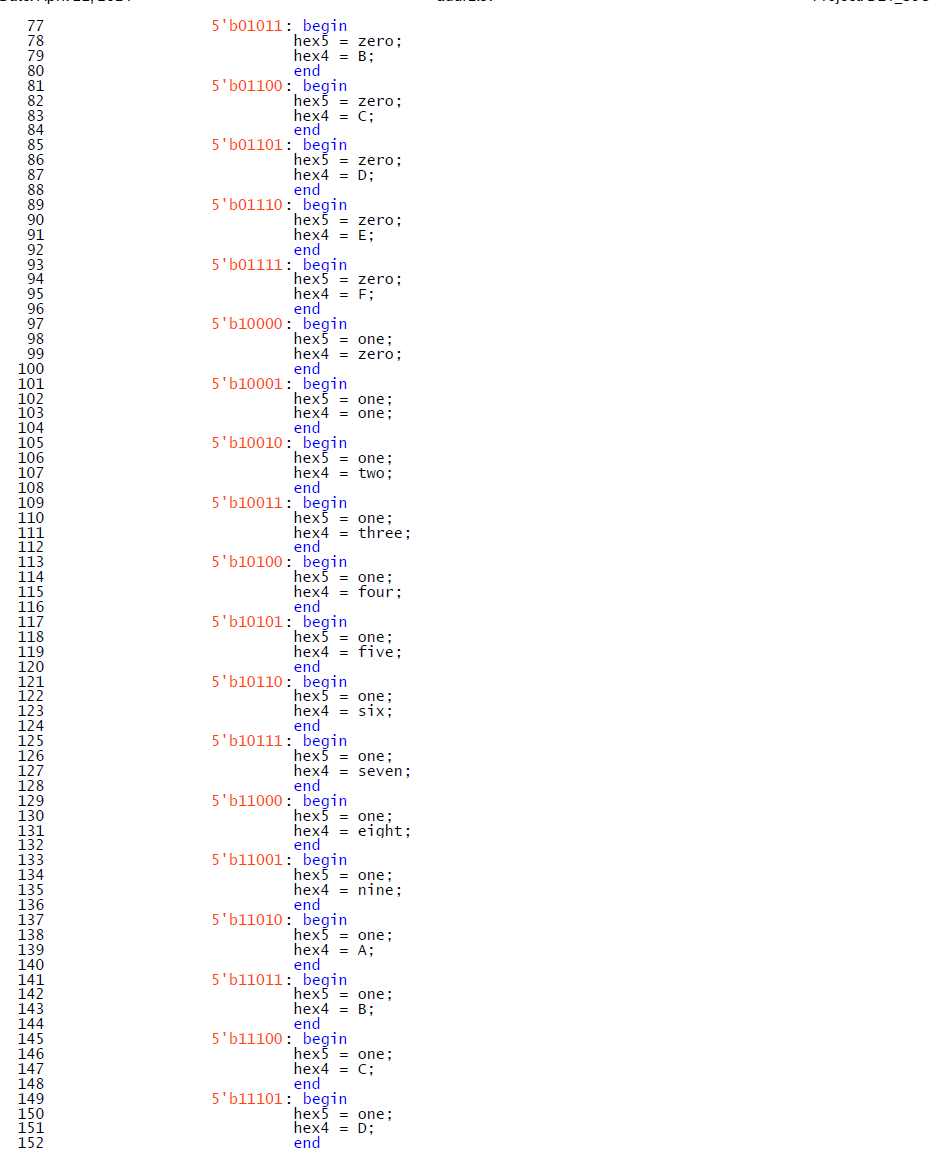
1. **DE1\_SoC (task 2)**

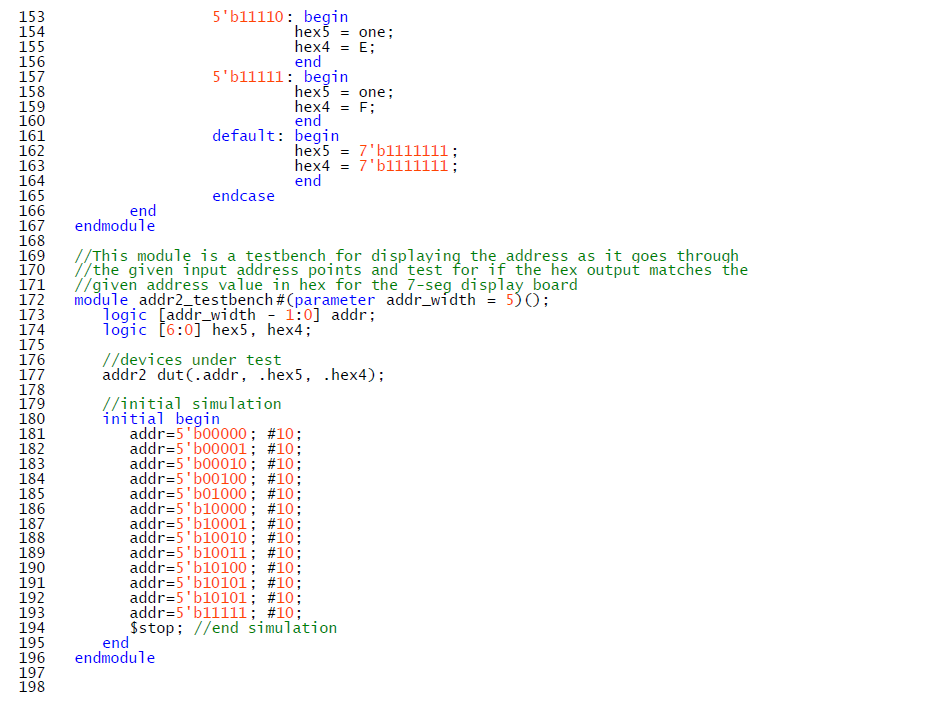
****

****

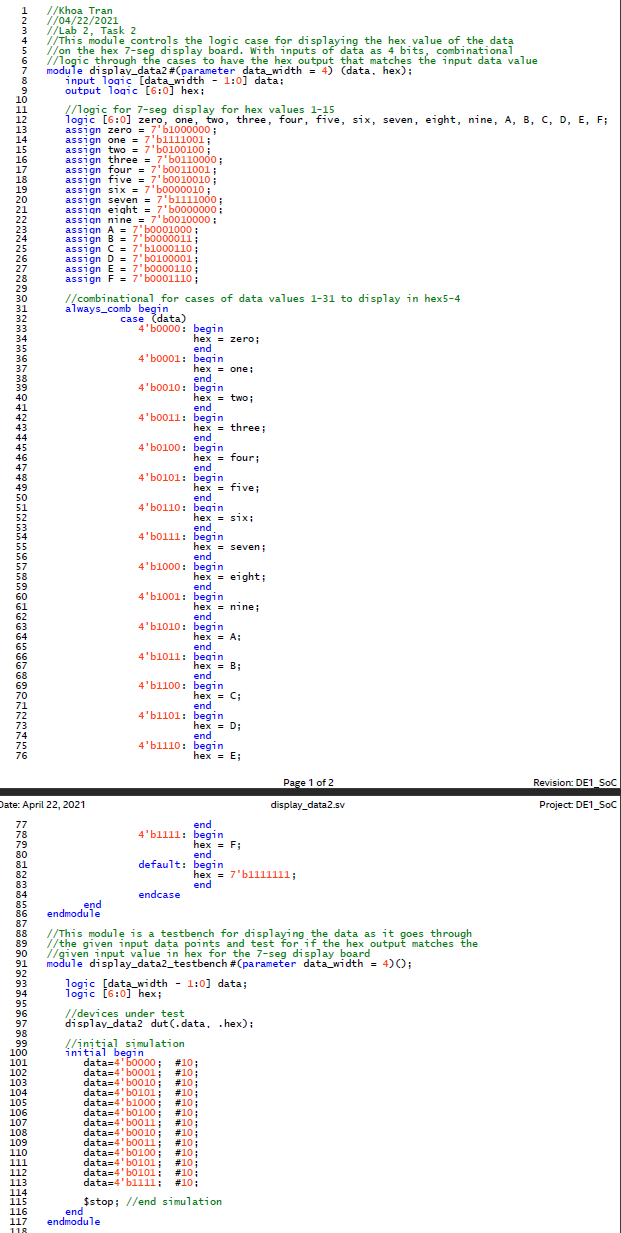
1. **addr2.sv (task 2)**

****

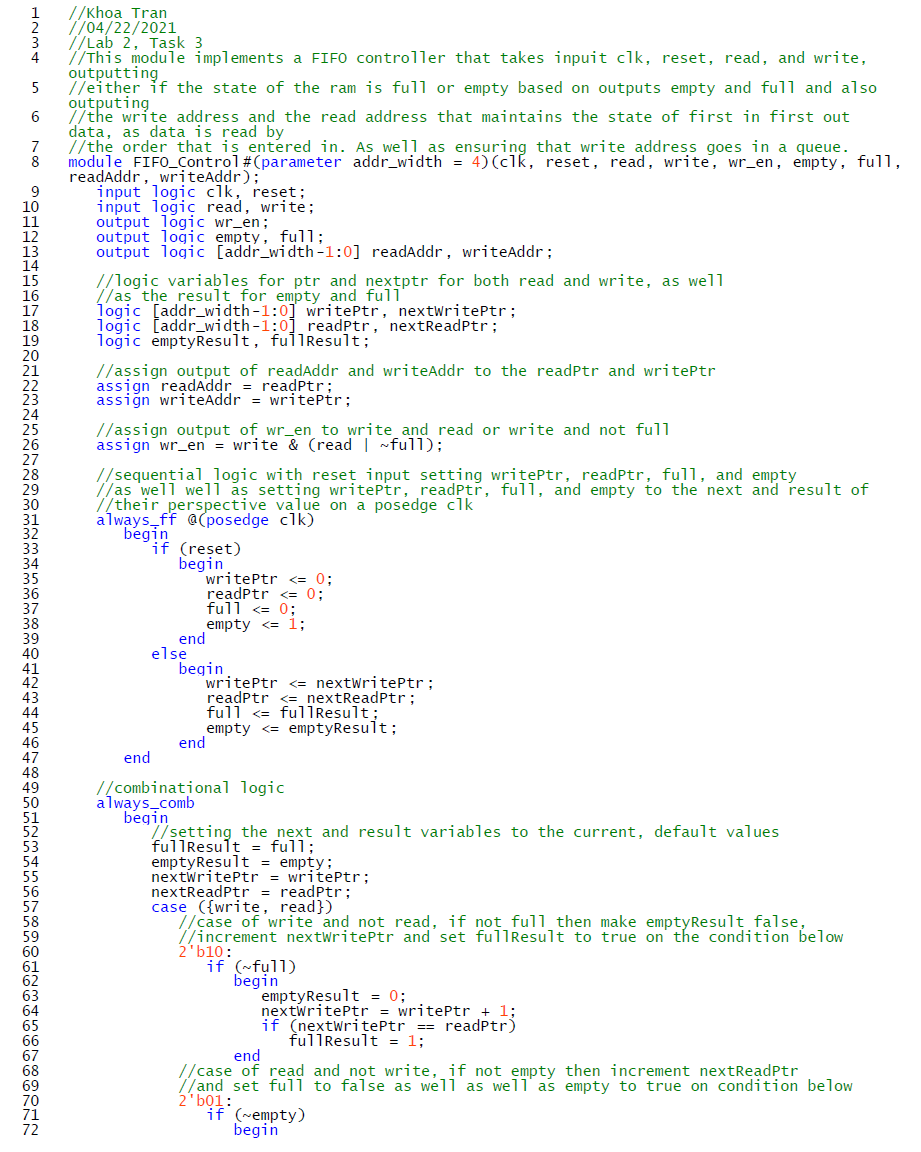
****

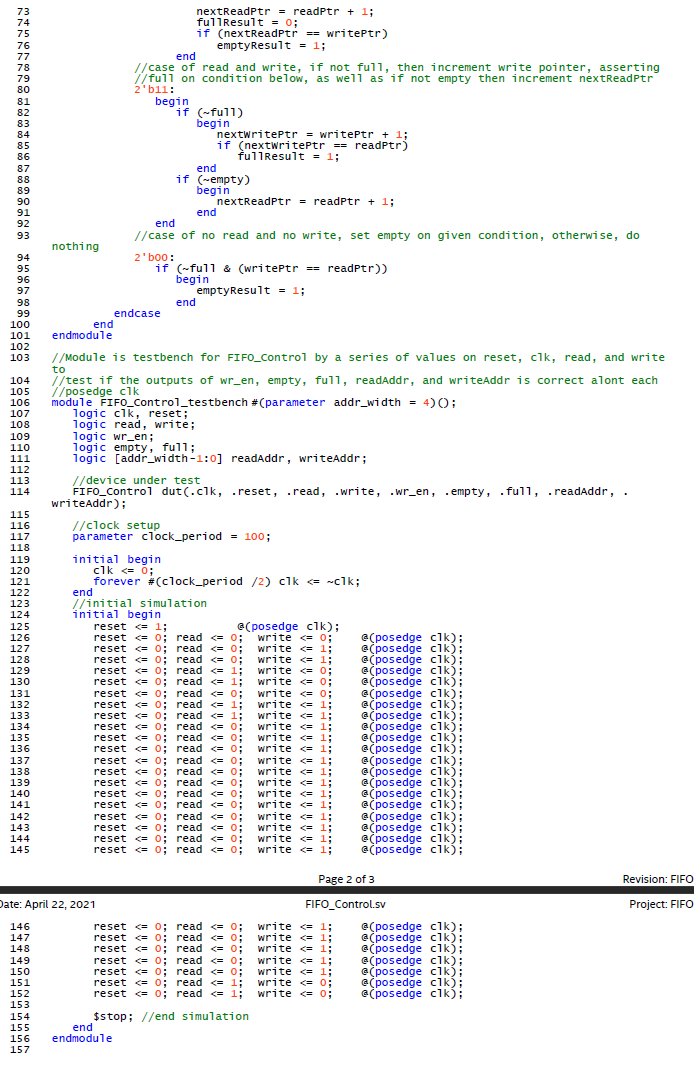
****

1. **display\_data2.sv (task 2)**

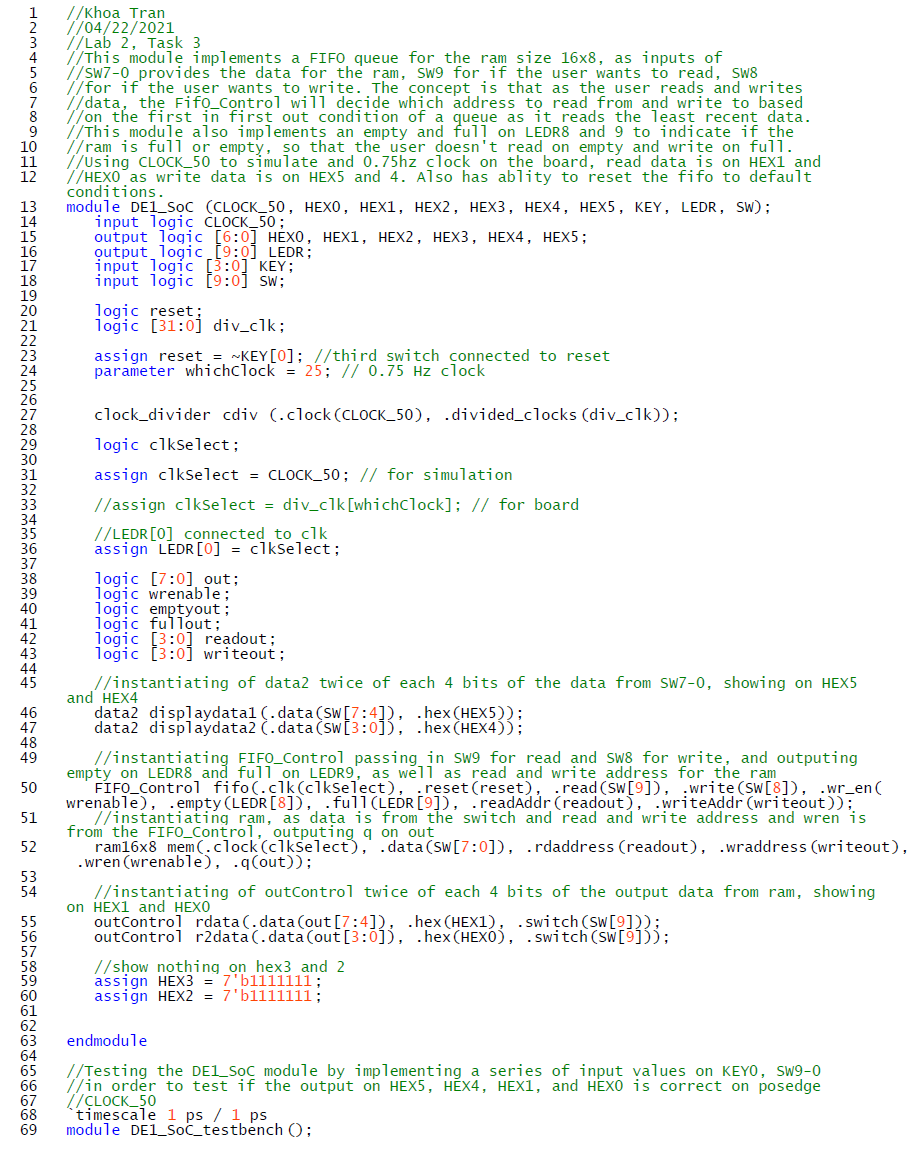
****

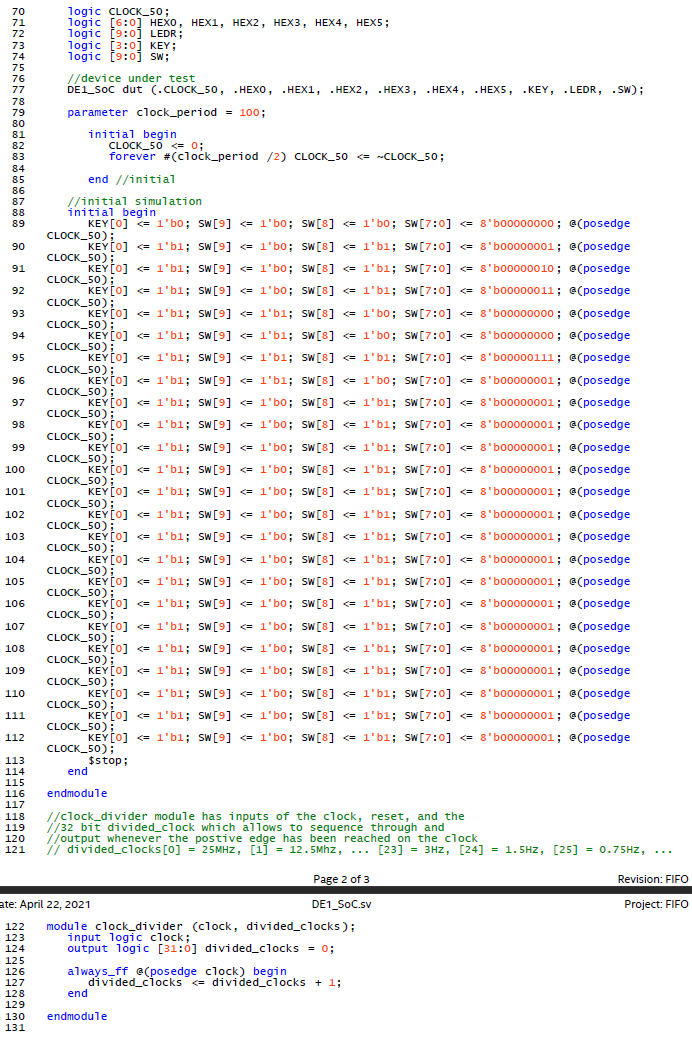
1. **FIFO\_Control.sv (task 3)**

****

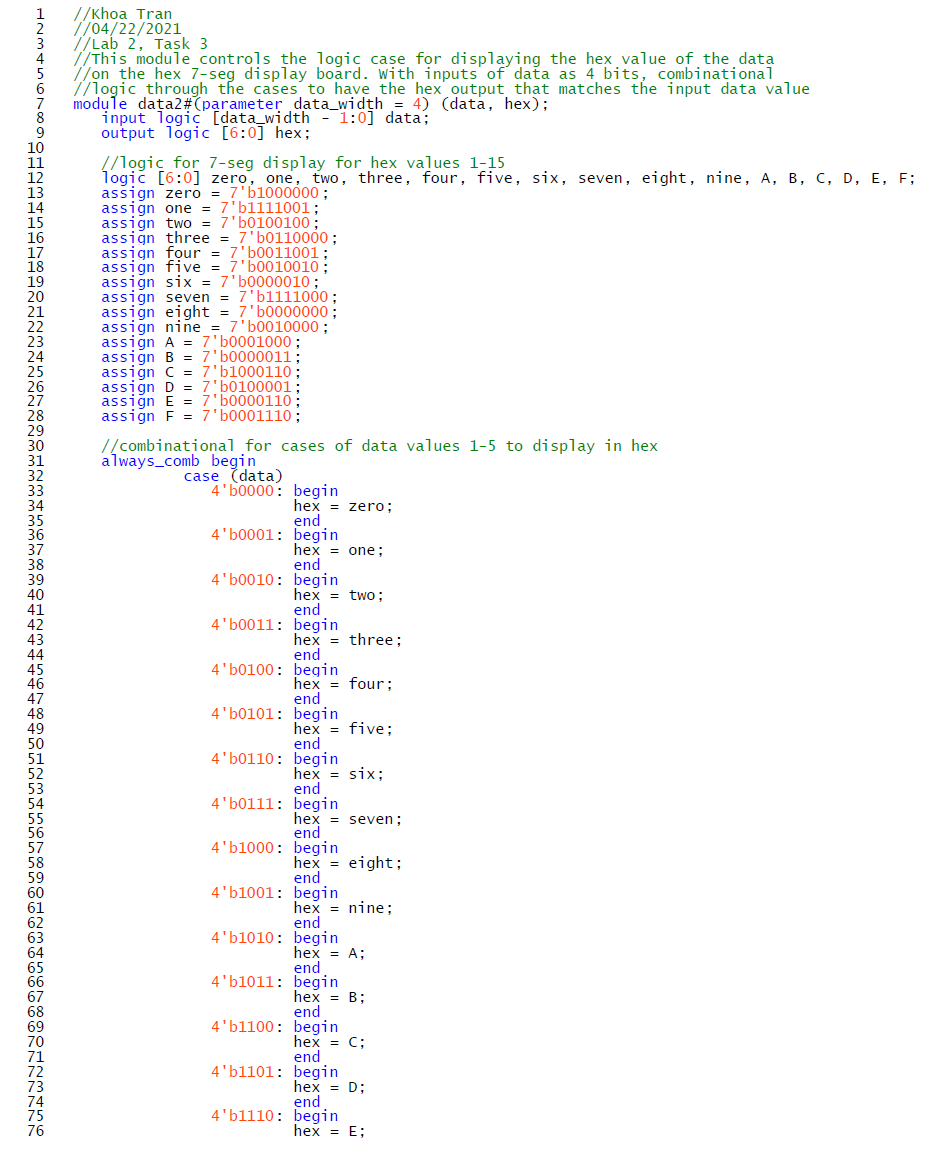
****

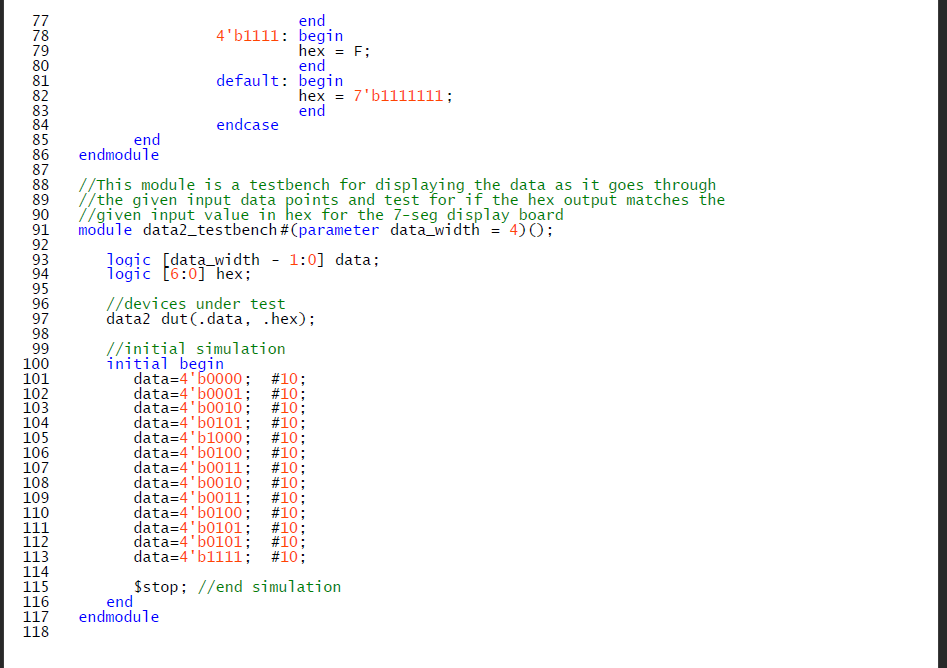
1. **DE1\_SoC.sv (task 3)**

****

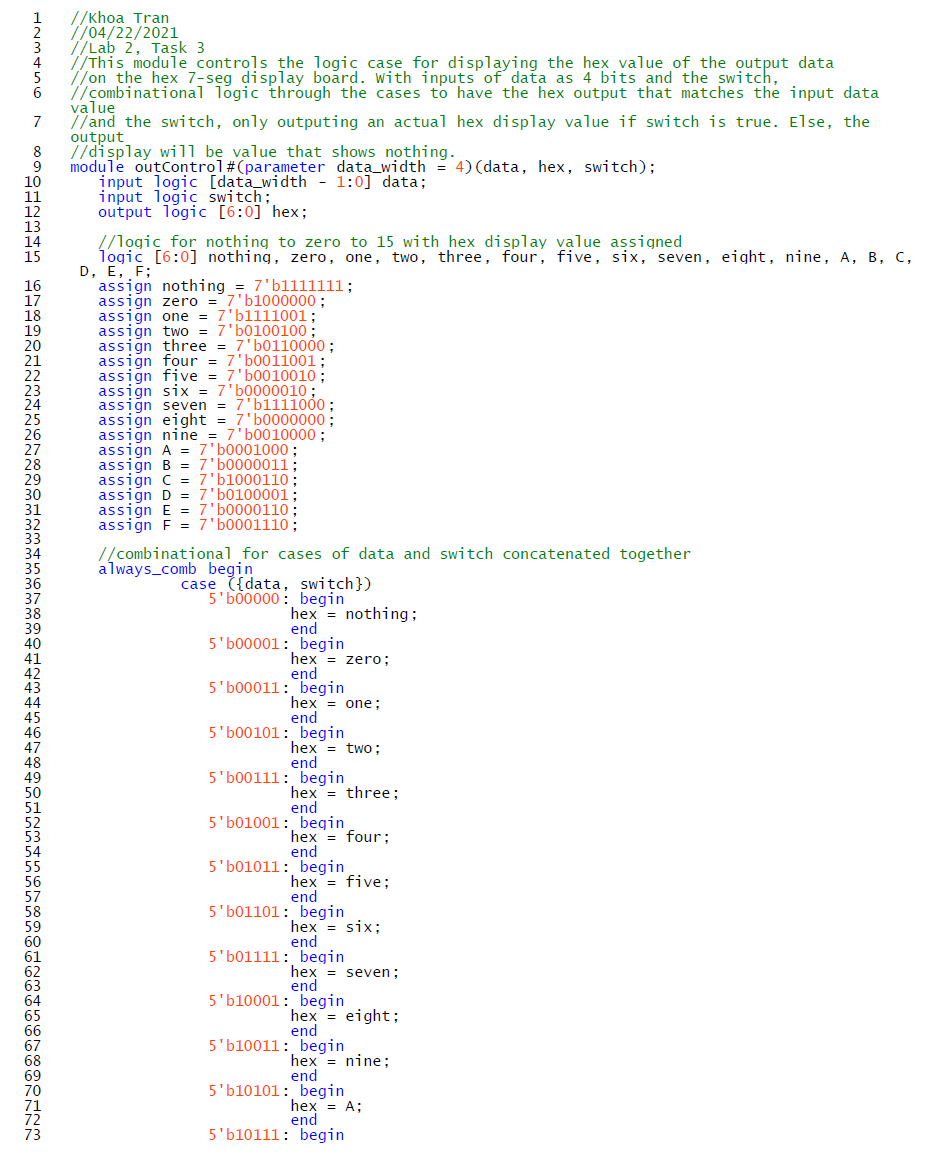
****

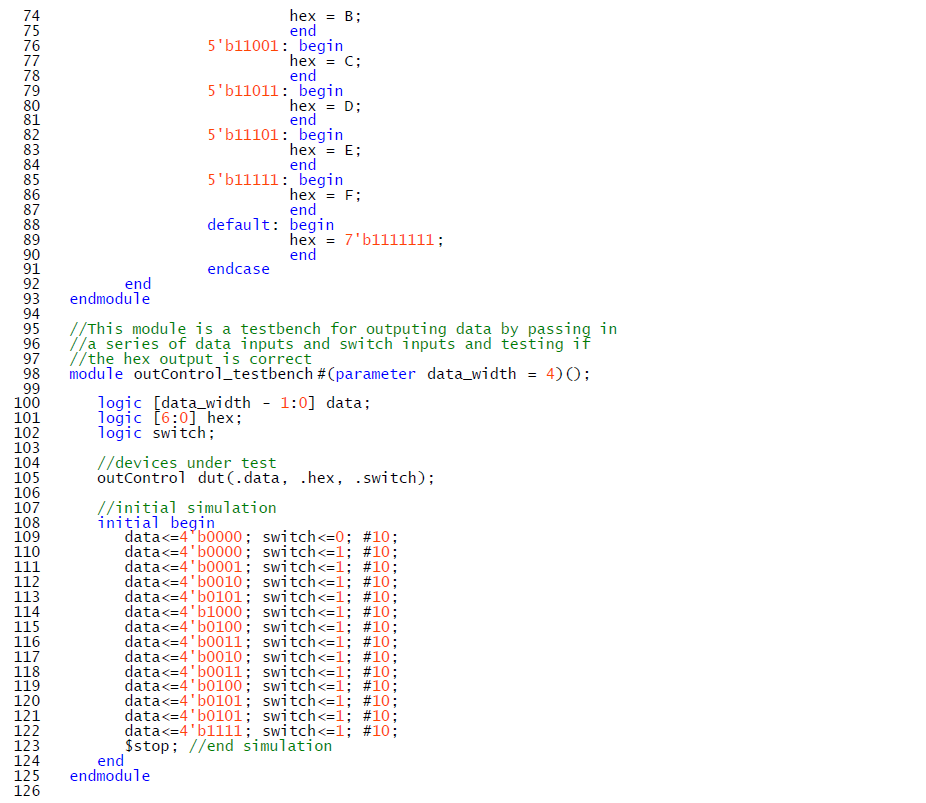
1. **data2.sv (task 3)**

****

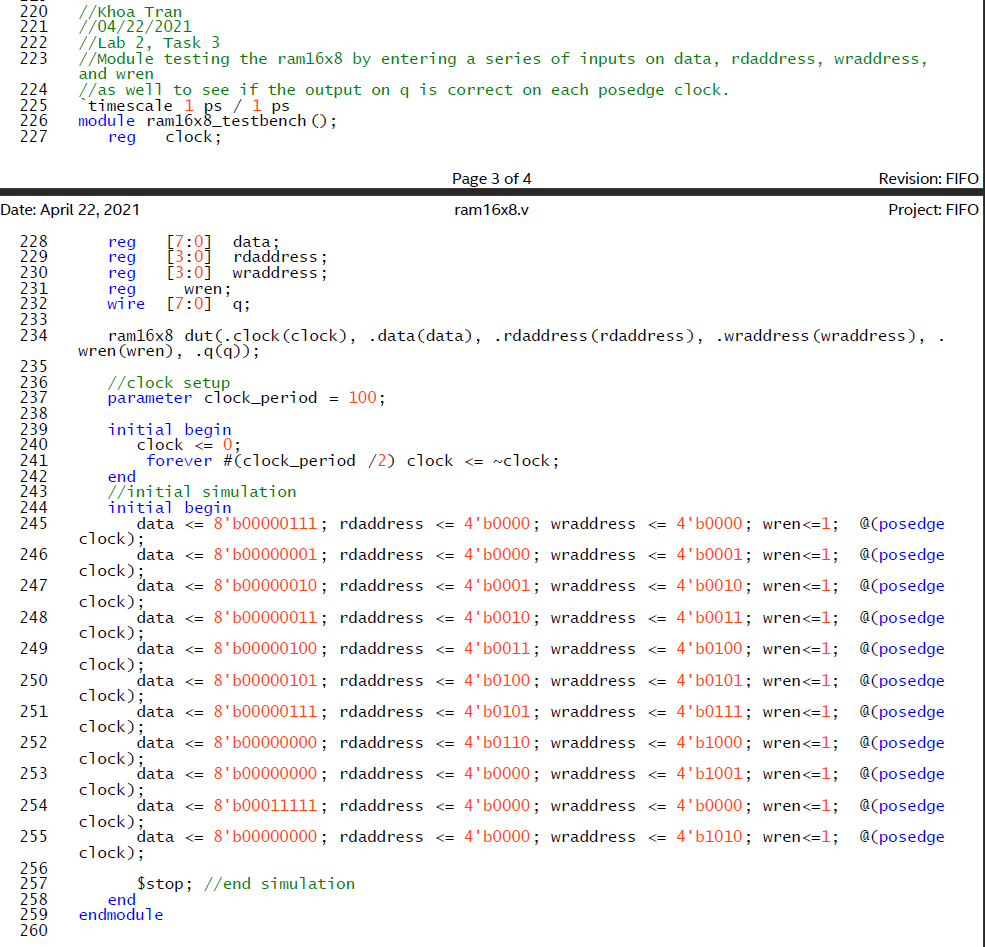
****

1. **outControl.sv (task 3)**

****

****

1. **ram16x8\_testbench.sv (task 3)**

****