Ashika Palacharla
Nithya Subramanian
EE371
January 29, 2025
Lab 2 Report

Section 1: Procedure

This lab asked to modify and implement various types of RAM modules on a FPGA. This system used M10K memory blocks and both single and dual port connections for the read and write functions of the memory. We divided the assignment into the following tasks, based on the major components:

1. Single Port RAM: Implement a single port 32x4 RAM with 5-bit address and a 4 bit input
2. Double Port RAM: Implement a double port 32x4 RAM with different read and write controls
3. FIFO Design: Build a FIFO design

Task #1: Single Port Ram

Task one we implemented a Single Port 32x4 RAM which has read and write capabilities. To do this we created three modules that had their functionalities (2 modules in block diagram, as they were all initialized following this format on the 3rd DE1_SoC module).

1. The first module is called “ram342x4” which takes in a 1-bit signal called wr_en(write enable), a 4-bit signal called d_in(the input data), and a 5-bit signal called addr(address of the write/read signal). This module allows us to write/store input data synchronously, into any given address as well as read this data from any given address. To do this we used a always_ff block that only writes to the address if the 1-bit write signal is high and if it is not high it reads and outputs the data already at that given address.
2. We also used a “seg7_hex” module to show the user what data they are inputting, and outputting, and the address at which this is happening. HEX display HEX0 showed the readout data, HEX2 showed the input to the memory, and HEX4-HEX5 showed the address value all in hexadecimal form. This module handles converting the values of the inputs to hexadecimal values, shown on the HEX displays.
3. The final module we used was our “DE1_SoC” which was our top-level module. This module assigned the different input and output signals to their respective Key and SW values.
    a. Signal d_in was connected to SW3 - SW0
    b. Signal addr was connected to SW8 - SW4
    c. Signal write was connected to SW[9]
    d. KEY[0] was connected to the clk

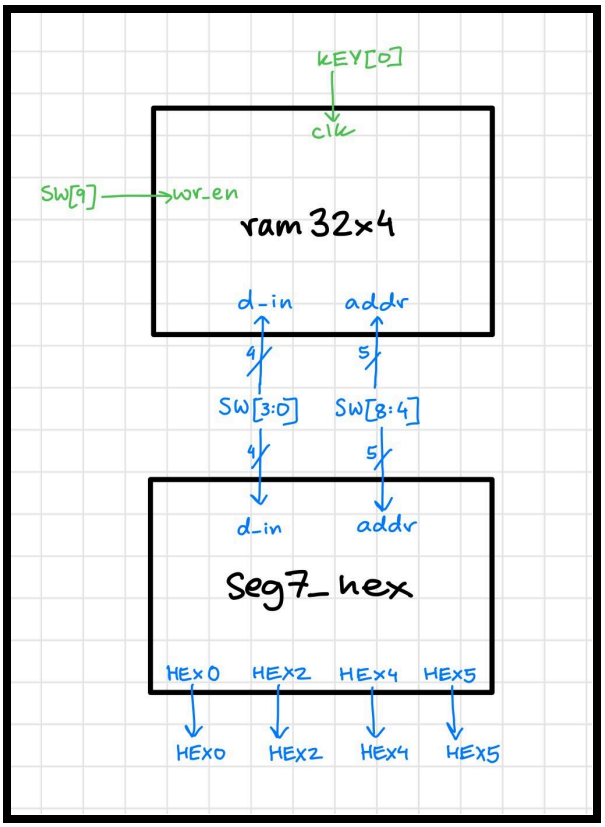

Figure 1: Block diagram of Task 1 - Single Port RAM

Task #2: Dual Port RAM

The second task was to create dual-port RAM by implementing an existing 32x4 RAM. We used 4 modules (3 modules in block diagram, as they were all initialized following this format on the 4th DE1_SoC module)

1. One module was called “counter” where the read address is incremented once every second. We did this by using a ff_block to only increase the addr_read (read address) when the count reaches a certain number.
2. We then created a “seg7_hex” module that is similar to task 1 in which it helps display the corresponding(listed in the Lab manual) values in the HEX display. This module handles converting the values of the inputs to hexadecimal values, shown on the HEX displays.
    a. The write address (wr_addr) was displayed on HEX5-4 displays
    b. The read address (r_addr) is shown on HEX3-2 displays
    c. The d_in is displayed on HEX1 (write data)
    d. The d_out is displayed on HEX0 (read data)
3. The third module we implemented was the “ram32x4”, which was done using the IP Catalog, following the instructions given to us by the lab manual.
4. We finally used our “DE1_SoC”, which is our top module, to connect the signals to their corresponding SW or KEY values.
    a. wraddress (and wr_addr) is connected to SW[8:4]
    b. data(and d_in) is connected to SW[3:0]
    c. The wren (wite enable) signal is connected to KEY[3]
    d. The reset signal is connected to KEY[0]
    e. The clock is connected to CLOCK_50

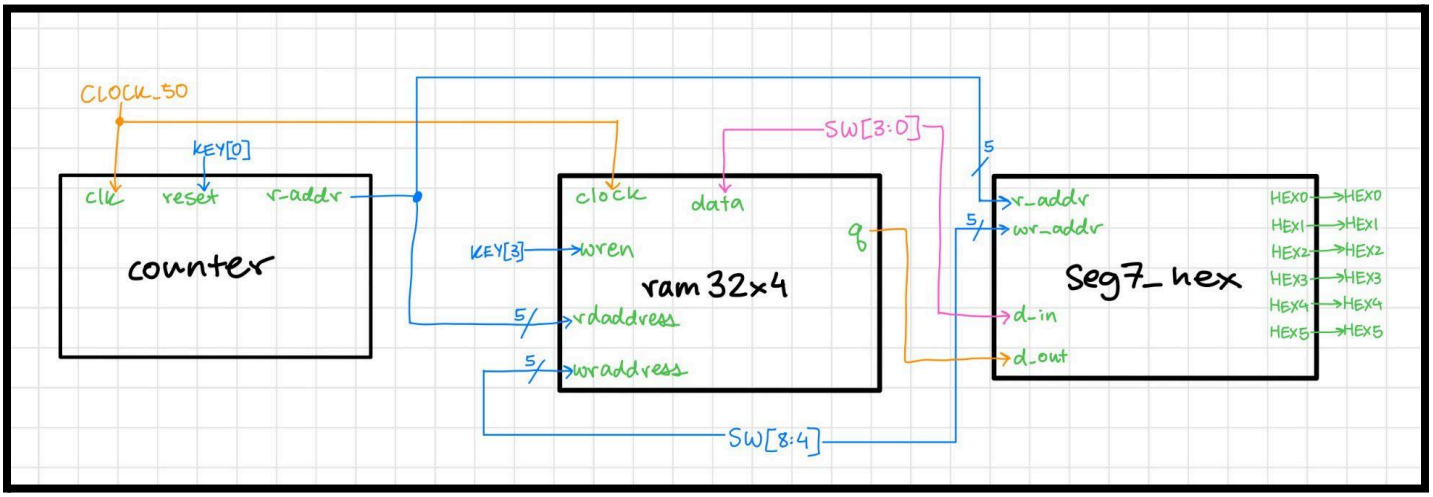

Figure 2: Block Diagram of Task 2 - Dual Port RAM

Task #3: FIFO Design

The third task was to design a FIFO system that uses a dual-port 16x8 RAM where we used helper modules to ensure the read/write functions of the FIFO were processed correctly.

1. The "FIFO" module would use the underlying 16x8 RAM as its memory and the FIFO_Control to detail the proper behavior of the FIFO (upholding the first-in, first-out nature based on different combinations of write and read actions)
    a. Used the "ram16x8" module, created using the IP catalog.
        i. This module would use the readAddr and writeAddr produced from the FIFO_Control module
        ii. This module had a 16x8 memory array, meaning it would create a FIFO with a depth of 16 and width of 8, as the FIFO has capacity for 16 entries, each the capacity to store an 8-bit value.
    b. Used the "FIFO_Control" module, to track the behavior of the FIFO
        i. Took in read and write inputs, based on the actions requested to the FIFO
        ii. Produced empty, full for the state of the FIFO. Also produced readAddr and writeAddr to determine where the FIFO would read/eject data from (read address) and write/insert data into (write address).
    c. Produces the empty and full signals to track the state of the FIFO (to LEDR[8] and LEDR[9], respectively)
    d. Produces the outputBus to track the content being ejected (read) from the FIFO
2. The "seg7_hex" module would correctly display the necessary information needed on the HEX displays, as it takes a 4-bit val input and displays a 7-bit output to the specified HEX. We established that this module would need to be instantiated 4 times, as there were 4 HEX displays that needed to be shown. This module handles converting the values of the inputs to hexadecimal values, shown on the HEX displays.
    a. Bottom 4-bits of the data inserted/written from the FIFO (inputBus[3:0]) on HEX0 display
    b. Top 4-bits of the data inserted/written from the FIFO (inputBus[7:4]) to HEX1 display
    c. Bottom 4-bits of the data ejected/read from the FIFO (outputBus[3:0]) on HEX4 display
    d. Top 4-bits of the data ejected/read from the FIFO (outputBus[7:4]) to HEX5 display
3. The "buttonPress" module ensures that any KEY press is held for one clock cycle to ensure that the input is not lost/not read if it starts and ends before the rising edge of a clock. We established this module would be instantiated 3 times, as the read, write, and reset signals were all controlled using KEYs. This module used an FSM to ensure that the KEY press would only be held for one clock cycle, as the rest of the modules depended on this specific behavior producing a valid signal from any KEY input. As such, the FSM has two states (no_press and hold_press).
    a. The KEY[3] maps to the read signal
    b. The KEY[2] maps to the write signal
    c. The KEY[0] maps to the reset signal

The following is important specifications of the FIFO behavior, detailed in the FIFO Control module (based on inputs to the module for read, write, empty, full):

1. When "read" and "write" are requested simultaneously:
    a. Set the read address to the least recent element address

b. Enable the write signal
c. Increment the least recent element address by 1
d. Set the write address to the next available spot (least recent element address plus the number of elements)

2. When “read” is requested and the FIFO is not “empty”
   a. If there is only one element in the FIFO, ensure the empty signal is turned on since this read action occurs. Ensure the full signal is off.
   b. Turn off the full signal.
   c. Turn off the write signal.
   d. Update the read address to the least recent element address.
   e. Decrease the number of elements in the FIFO.
   f. Increment the address of the least recent element.
3. When “write” is requested and the FIFO is not “full”
   a. If the FIFO is one element away from being full, ensure the full signal is turned on since this write action occurs. Ensure the empty signal is off.
   b. Turn on the write signal.
   c. Turn off the empty signal.
   d. Update the write address to the next available spot in the FIFO (least recent element address plus the number of elements)
   e. Increase the number of elements in the FIFO.

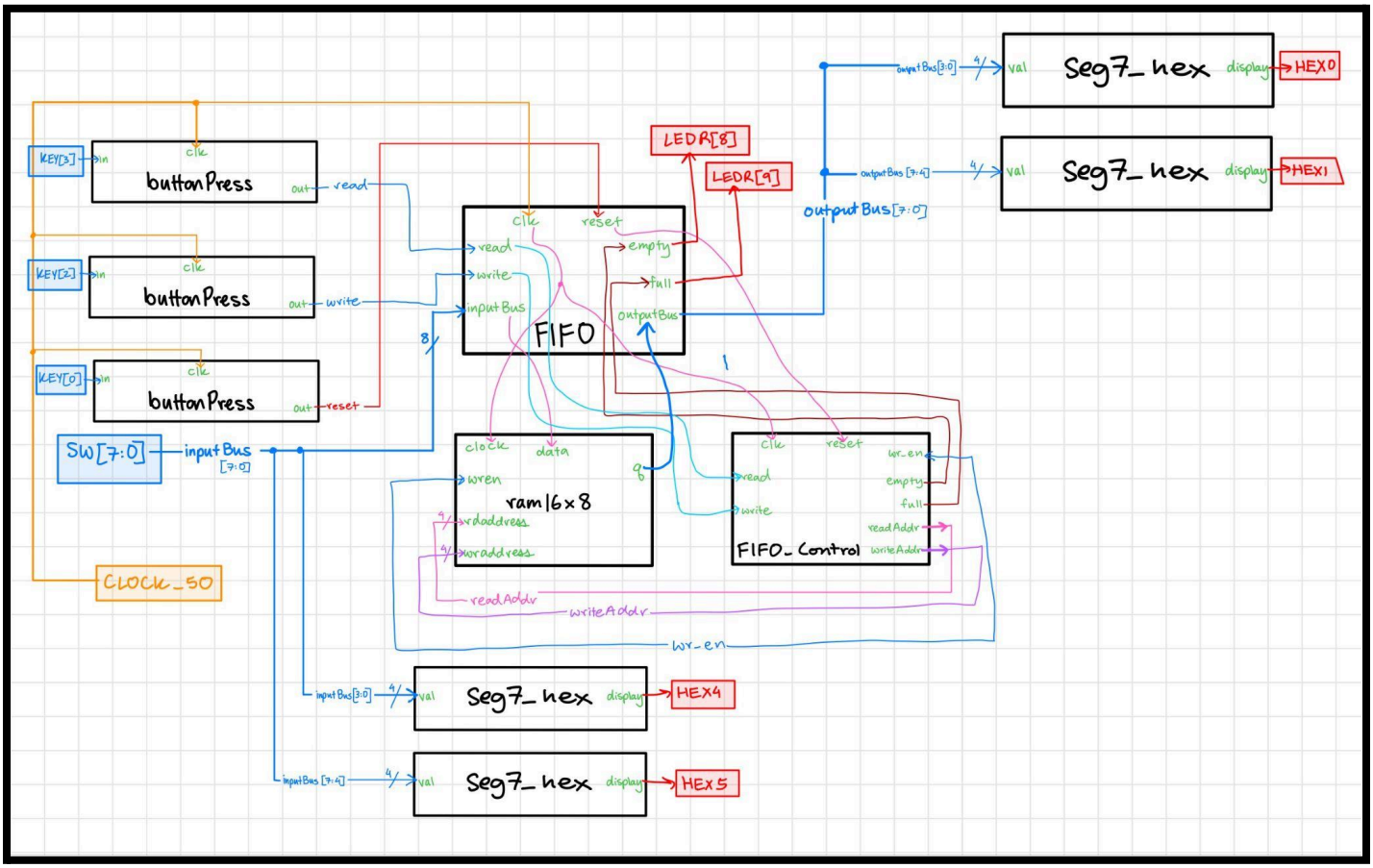

Figure 4: Block diagram of Task 3 - FIFO

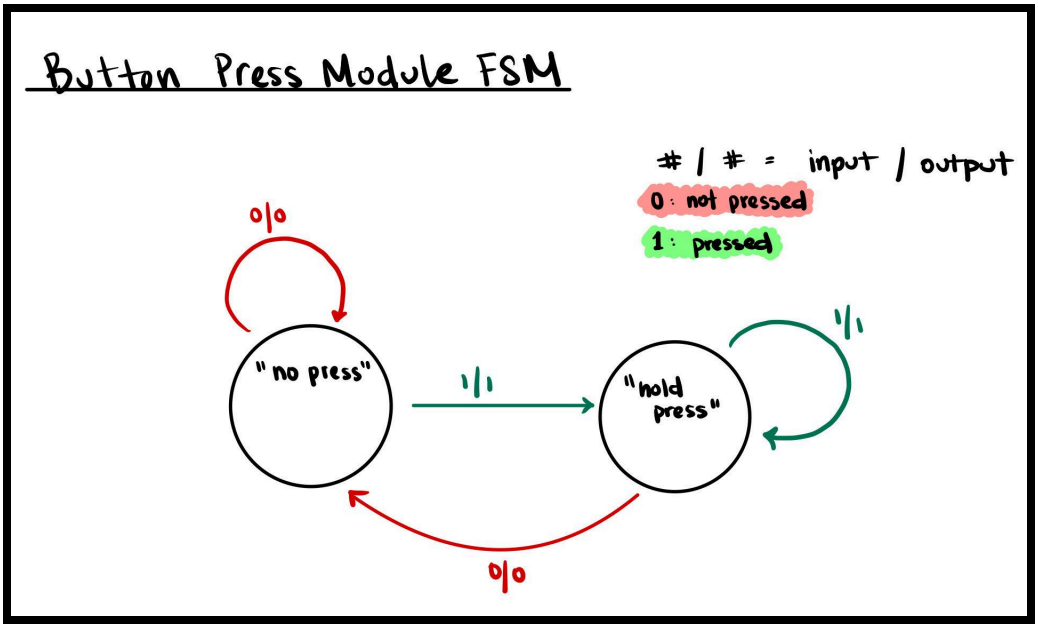

Figure 3: FSM diagram for buttonPress.sv, showing the states, transitions, and format of the inputs

Section 2: Results

Task #1: Single Port RAM

Each of the testbench simulations tested the following situations:

1. **seg7_hex_tb:** Tests the 7-segment HEX display, based on different values of the data input, data output, and address which all trace to different HEX displays.
    a. Test different combinations of addr, d_in, d_out and observe the related HEX.
    b. HEX5-4 is intended to display addr. HEX2 is intended to display d_in. HEX0 is intended to display d_out. Assigned based on lab spec.
2. **ram32x4_tb:** Tests behavior of the RAM as the data for an address is requested to be read, and if data is written to an address.
    a. Write signal is enabled. Write actions will be executed 32 times to write in data for each of the memory addresses (data written in is the value of the address).
        i. There is no d_out since there is no data output being read.
    b. Write signal is off. Read actions will be executed 32 times to read out data for each of the memory addresses.
        i. There is now a d_out since there is data output being read by the repeated read actions.
3. **DE1_SoC_tb**: Similar to ram32x4_tb. Tests behavior of the ram32x4 and seg7_hex as the data for an address is requested to be read, and if data is written to an address.
    a. Write signal is enabled. Write actions will be executed 32 times to write in data for each of the memory addresses (data written in is the value of the address).
        i. HEX0 will not have a display since there is no data output being read.
        ii. HEX5-4 displays the address.
        iii. HEX2 displays the data input.
    b. Write signal is off. Read actions will be executed 32 times to write in data for each of the memory addresses.
        i. HEX1 and HEX3 is off.
        ii. HEX5-4 displays the address.
        iii. HEX2 displays the data input.
        iv. HEX0 displays the data output, and now has a display.

Descriptions of the simulation waveforms are provided below the figures. Or labeled with the situation that it corresponds to, listed above.

Simulation waveforms of seg7_hex_tb:

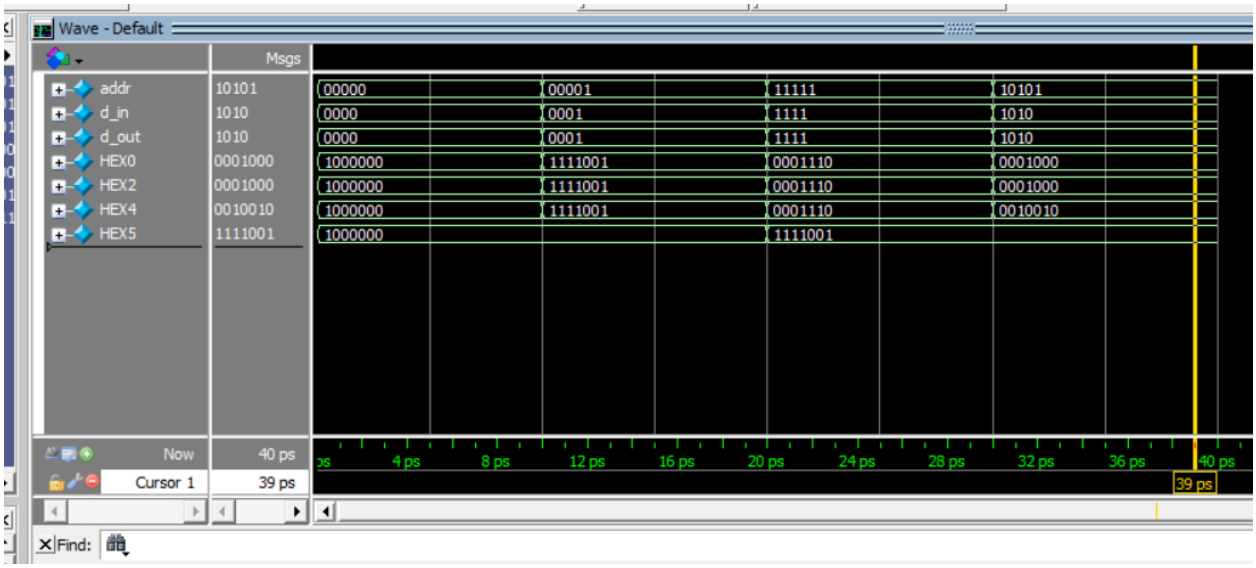

Figure 5: ModelSim waveform of the seg7_hex_tb

Shows Situation a-b for the seg7_hex_tb. Tests 4 different combinations of the addr, d_in, d_out; shows how the corresponding HEX display values are shown on HEX5-4, HEX2, HEX0 respectively.

Simulation waveforms of ram32x4_tb:

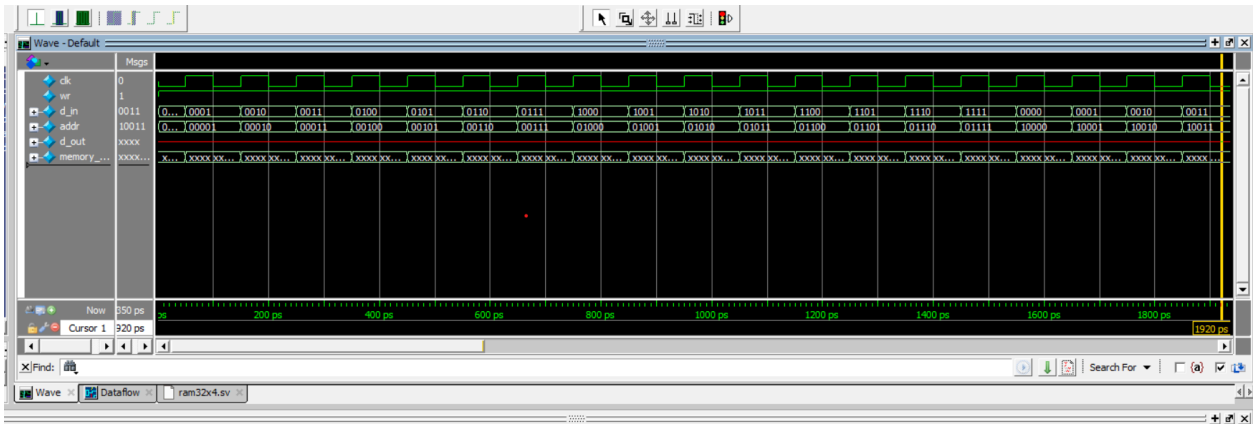

Figure 6.1: ModelSim waveform of the ram32x4_tb, part1

Shows Situation a, as the write signal is enabled and the write actions are being executed to write in data for each of the memory addresses. Nothing is read out on d_out, since the write signal is enabled so no read action is occurring.

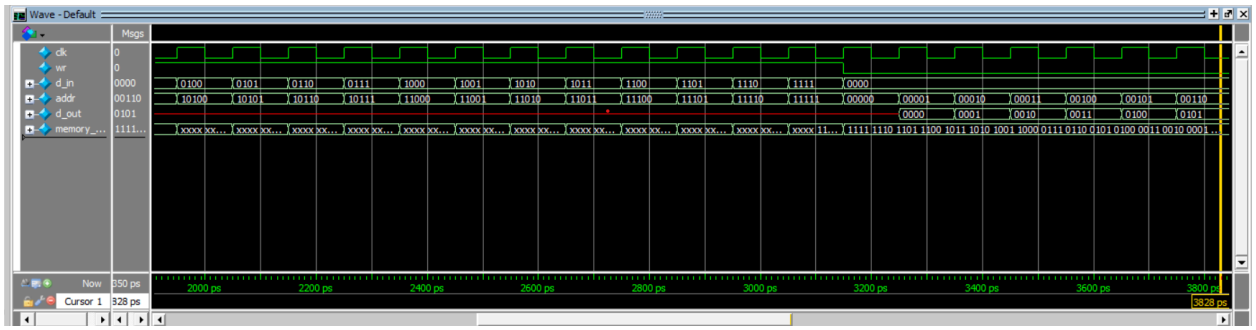

Figure 6.2: ModelSim waveform of the ram32x4_tb, part2

Continues to show Situation a, as the write signal is enabled and the write actions are being executed to write in data for each of the memory addresses. Shows Situation b, where write signal is turned off and the read actions begin to be executed. Corresponding with the write signal being turned off, d_out begins to be filled as there is data output being read.

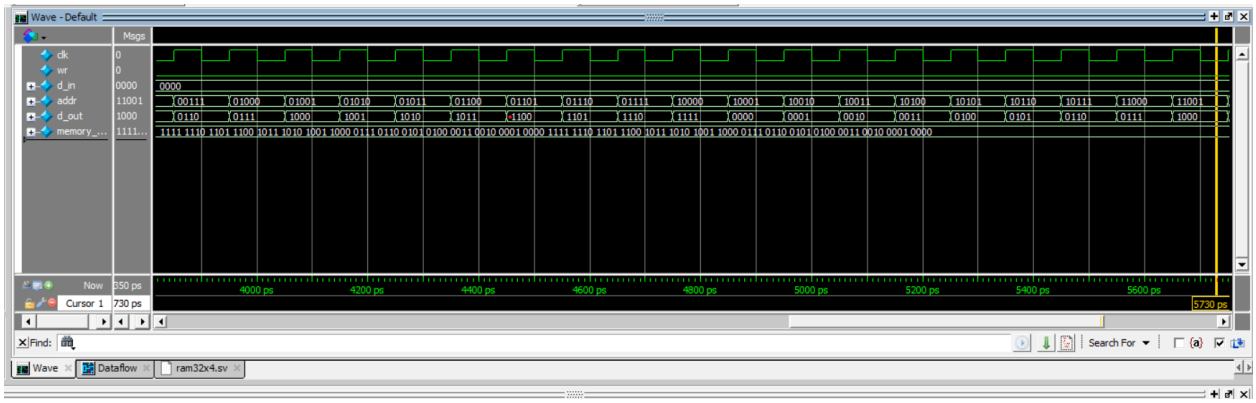

Figure 6.3: ModelSim waveform of the ram32x4_tb, part3

Continues to show Situation b, where the write signal is off, so read actions are being executed to read out data for each of the 32 memory addresses.

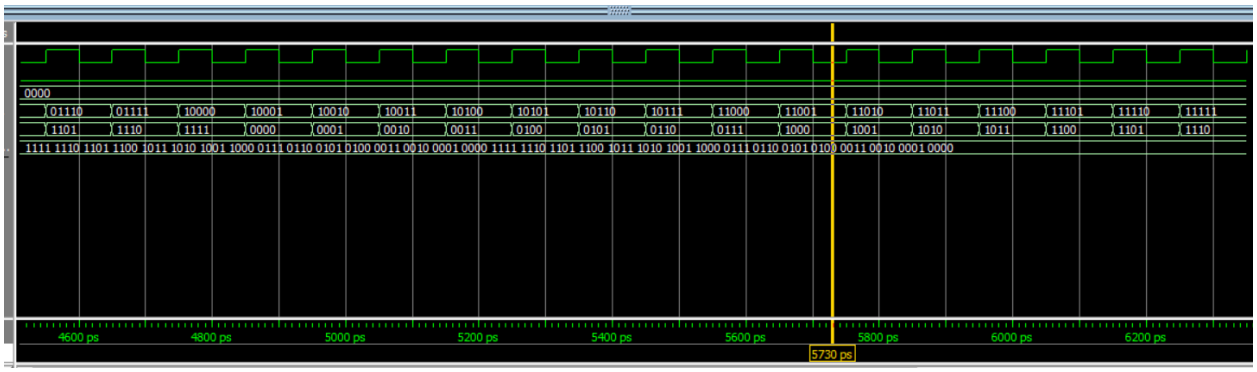

Figure 6.4: ModelSim waveform of the ram32x4_tb, part4

Continues to show Situation b, where the write signal is off, so read actions are being executed to read out data for each of the 32 memory addresses.

Simulation waveforms of DE1_SoC_tb:

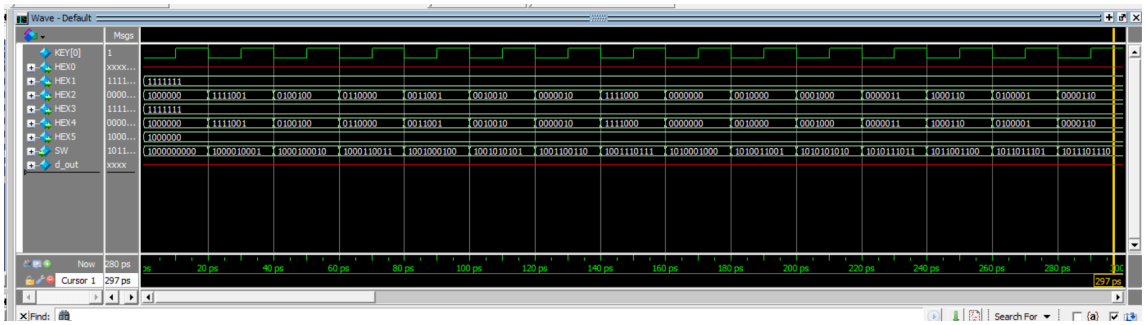

Figure 7.1: ModelSim waveform of the DE1_SoC_tb, part1

Shows Situation a, as the write signal is enabled (mapped to the SW[9]) and the write actions are being executed to write in data for each of the memory addresses. Nothing is read out on d_out, since the write signal is enabled so no read action is occurring. Nothing is displayed on HEX0 since there is no d_out. HEX5-4 and HEX2 have displays.

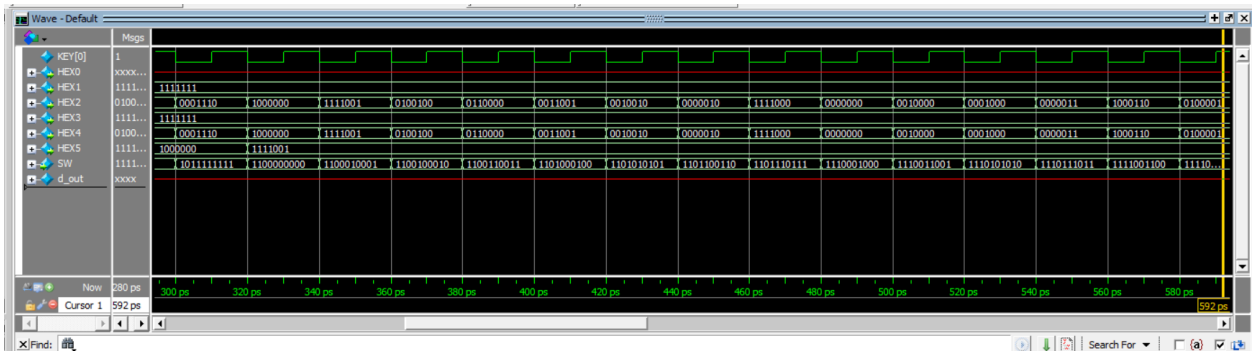

Figure 7.2: ModelSim waveform of the DE1_SoC_tb, part2

Shows Situation a, as the write signal is enabled (mapped to the SW[9]) and the write actions are being executed to write in data for each of the memory addresses. Nothing is read out on d_out, since the write signal is enabled so no read action is occurring. Nothing is displayed on HEX0 since there is no d_out. HEX5-4 and HEX2 have displays.

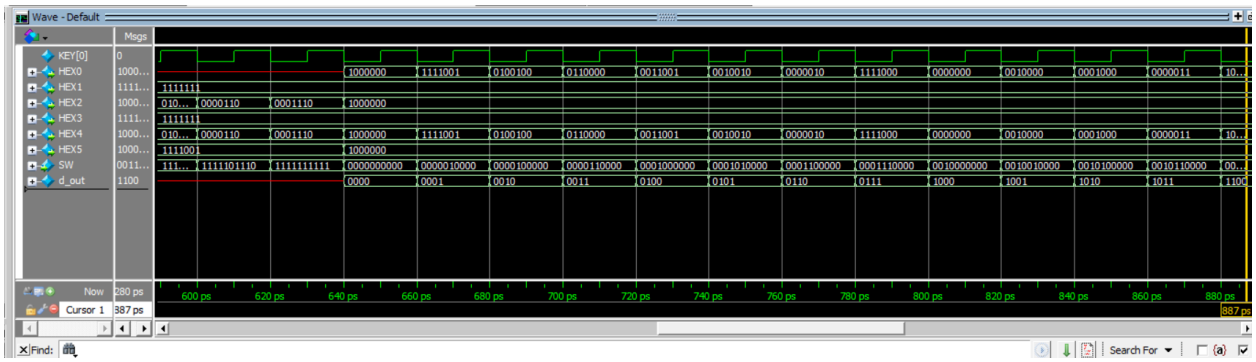

Figure 7.3: ModelSim waveform of the DE1_SoC_tb, part3

Shows the end of Situation a, as the write signal is enabled (mapped to the SW[9]) and the write actions are being executed to write in data for each of the memory addresses. Nothing is read out on d_out, since the write signal is enabled so no read action is occurring. Nothing is displayed on HEX0 since there is no d_out. Shows the start of Situation b, as the write signal is turned off (mapped to the SW[9]), and the read actions begin to be executed. Corresponding with the write signal being turned off, d_out begins to be filled as there is data output being read. HEX0 will also now have a display, since it shows the d_out. HEX5-4 and HEX2 have displays.

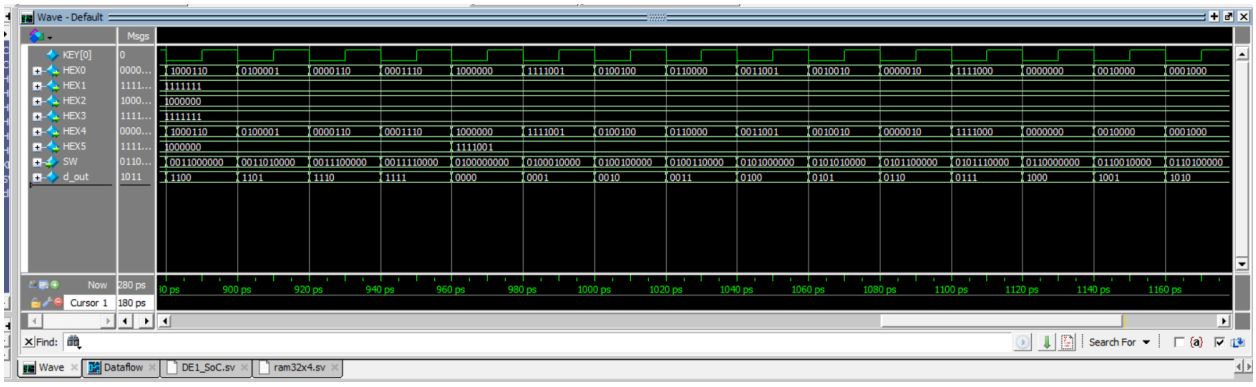

Figure 7.4: ModelSim waveform of the DE1_SoC_tb, part4

Shows Situation b, still executing the read action as write signal is turned off.

Figure 7.5: ModelSim waveform of the DE1_SoC_tb, part5

Shows Situation b, still executing the read action as write signal is turned off.

Task #2: Double Port RAM

Each of the testbench simulations tested the following situations:

1. **seg7_hex_tb:** Tests the 7-segment HEX display, based on different values of the data input, data output, and address which all trace to different HEX displays.
    a. Test different combinations of r_addr, wr_addr, d_in, d_out and observe the related HEX.
    b. HEX5-4 is intended to display wr_addr. HEX3-2 is intended to display r_addr. HEX1 is intended to display d_in. HEX0 is intended to display d_out. Assigned based on lab spec.
2. **counter_tb:** Tests the counter, based on letting the clock run to see how the read address changes over the course of the clock edges.
    a. Turns reset on and off.
    b. Runs the clock 50 times, intending for the read address to change each time, as the FREQ_CONSTANT is 0 so the clock should increment every clock cycle.
3. **ram32x4_tb:** Tests behavior of the RAM as the data for an address is requested to be read, and if data is written to an address.
    a. Write signal is enabled. Write actions will be executed 32 times to write in data for each of the memory addresses (data written in is the value of the address).
        i. Nothing is read out on the d_out because the value would only be updated on the next clock cycle, reading for the same address. Since the testbench ensures that the address is updated each clock cycle (to replicate the counter), we are never reading the same address two clocks in a row -- so we have to wait for Situation b to read the data written to each of the addresses.
    b. Write signal is off. Read actions will be executed 32 times to read out data for each of the memory addresses.
        i. We can now observe the d_out, since we are reading the addresses that had been previously written to in the first 32 clock cycles.
4. **DE1_SoC_tb**: Tests behavior of the ram32x4, counter, and seg7_hex as the data for an address is requested to be read, and if data is written to an address.
    a. Write signal is enabled. Write to an address. Data input is given.
        i. SW[3:0] and SW[8:4] will have values. KEY[3] is off(active-low).
    b. Write signal is enabled. Write to the same address as previously. Data input is given.
        i. SW[3:0] and SW[8:4] will have values. KEY[3] is off(active-low).
    c. Write to new addresses, write signal is enabled. Data input is given.
        i. SW[3:0] and SW[8:4] will have values. KEY[3] is off(active-low).
    d. Write signal is off. Check read on the addresses. Running for multiple clock edges, so the counter would automatically increment to ensure that different addresses are being read from each clock cycle (each 1 second).
        i. KEY[3] is on(active-high). No write address and data input is given (so SW[8:4] for write address and SW[3:0] for data input will hold old values).
    e. HEX displays for each of the situations above:
        i. HEX5-4 displays the write address.
        ii. HEX3-2 displays the read address.
        iii. HEX1 displays the write data.
        iv. HEX0 shows the data output.
    f. On-board inputs to the DE1_SoC for each of the situations above:

i. KEY[3] is the write signal
ii. KEY[0] is the reset signal
iii. SW[3:0] is the data input
iv. SW[8:4] is the write address

Descriptions of the simulation waveforms are provided below the figures. Or labeled with the situation that it corresponds to, tracing back to the detailed descriptions listed above.

Simulation waveforms of seg7_hex_tb:

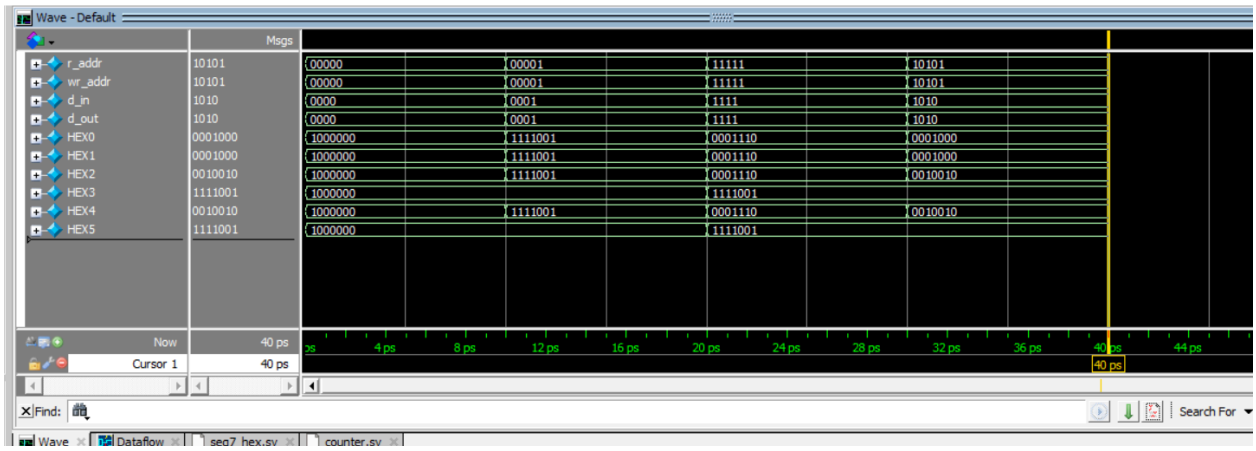

Figure 8: ModelSim waveform of the seg7_hex_tb

Shows situation a and b. 4 sets of random inputs are given to r_addr, wr_addr, d_in, and d_out to verify that the proper values are displayed on HEX0-5.

Simulation waveforms of counter_tb:

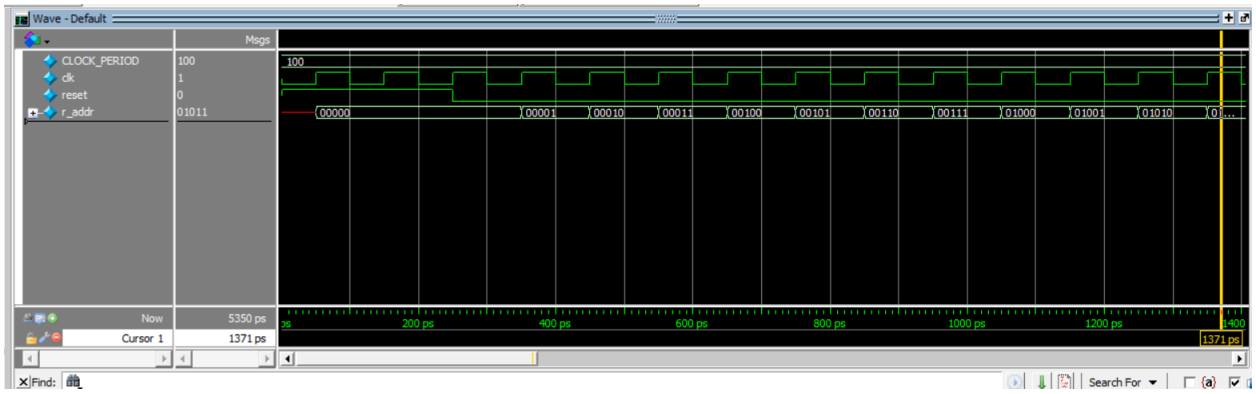

Figure 9.1: ModelSim waveform of the counter_tb, part1

Shows Situation a (where reset is turned on and off). Shows Situation b, with each clock cycle, the r_addr is incremented by one bit.

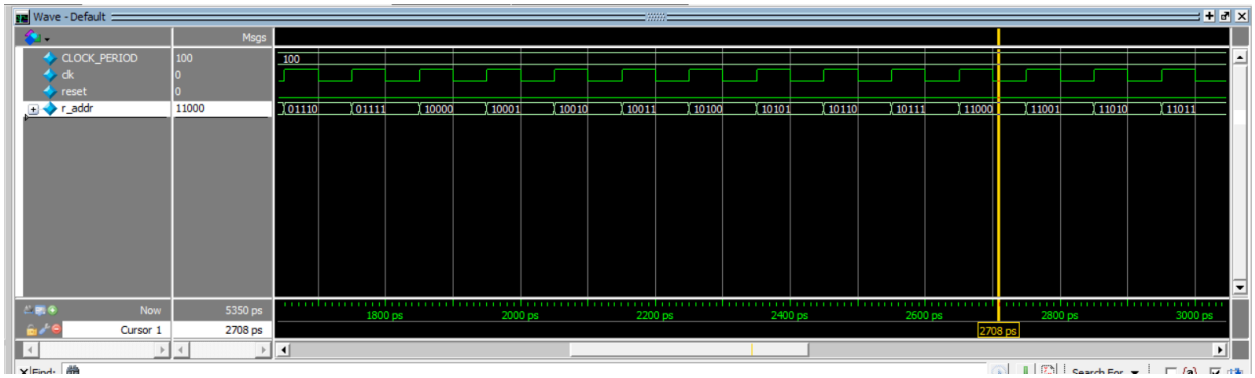

Figure 9.2: ModelSim waveform of the counter_tb, part2

Continues to show situation b, with each clock cycle, the r_addr is incremented by one bit.

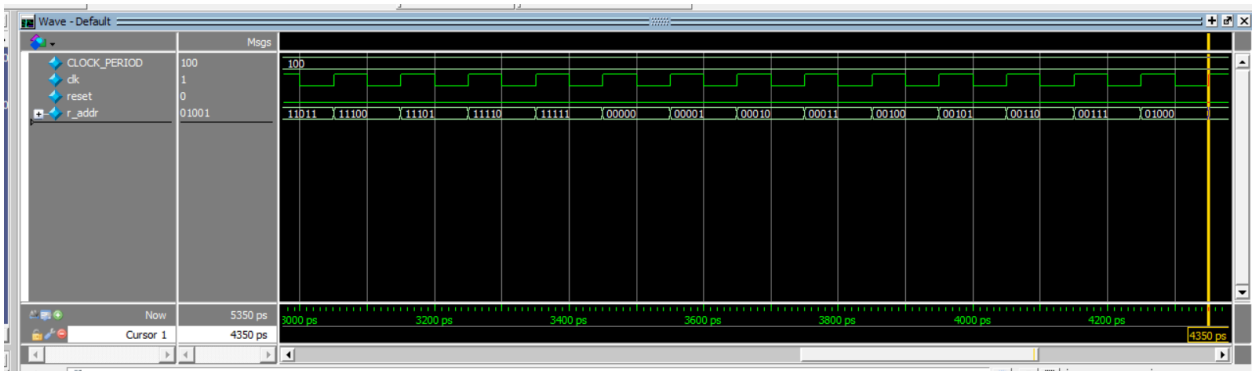

Figure 9.3: ModelSim waveform of the counter_tb, part3

Continues to show situation b, with each clock cycle, the r_addr is incremented by one bit.

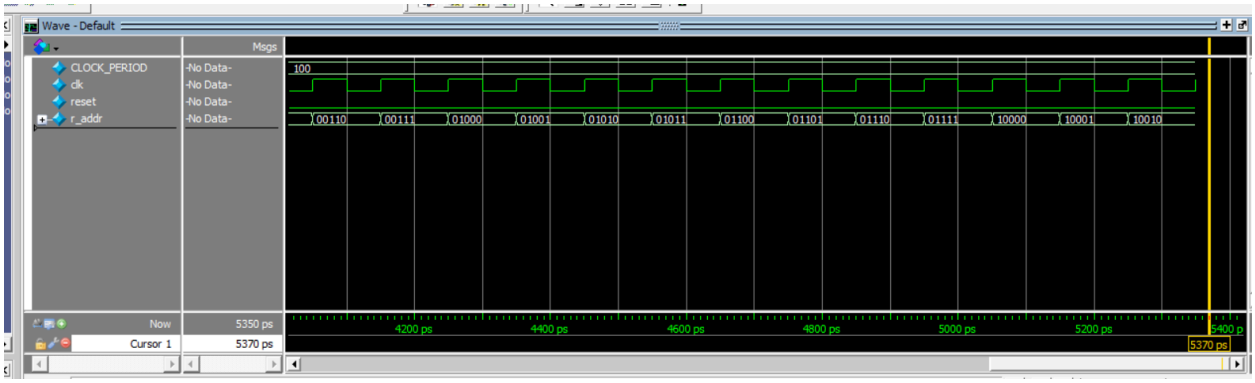

Figure 9.4: ModelSim waveform of the counter_tb, part4

Continues to show situation b, with each clock cycle, the r_addr is incremented by one bit.

Simulation waveforms of ram32x4_tb:

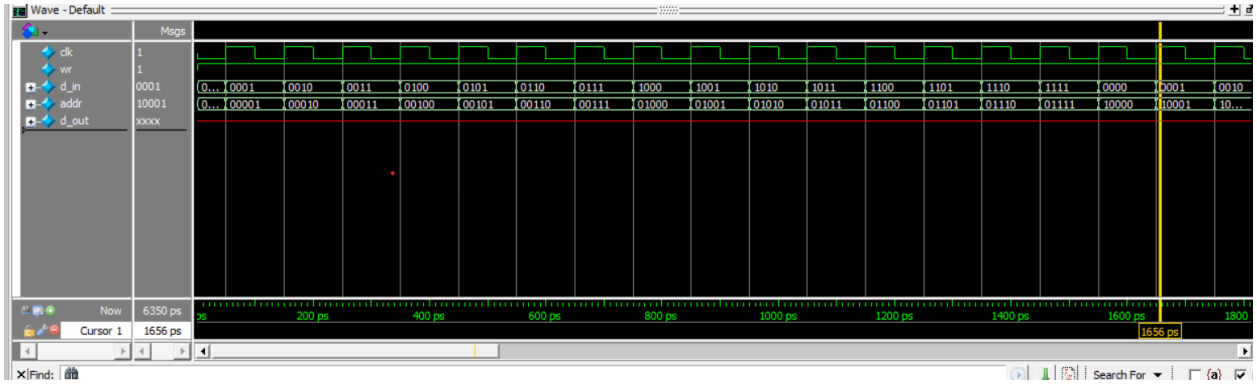

Figure 10.1: ModelSim waveform of the ram32x4_tb, part1

Shows Situation a, as the write signal is enabled and the write actions are being executed to write in data for each of the memory addresses.

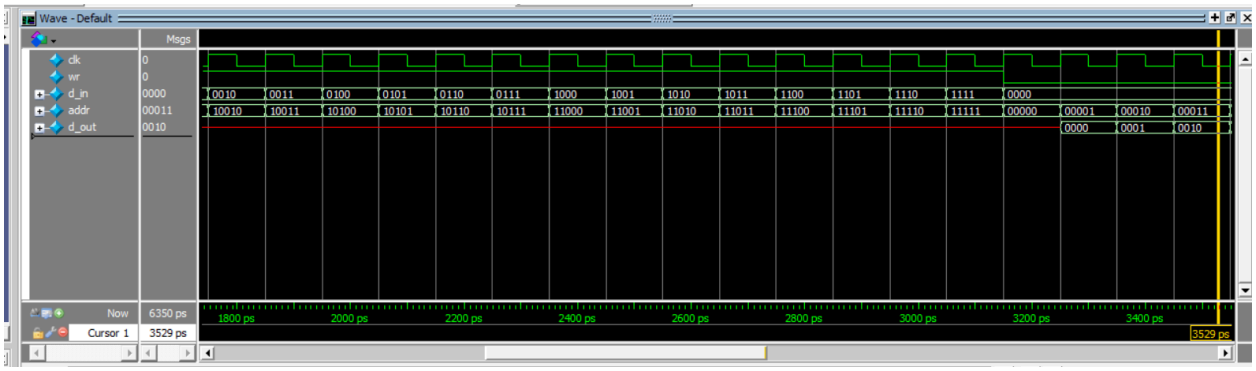

Figure 10.2: ModelSim waveform of the ram32x4_tb, part2

Shows Situation a, as the write signal is enabled and the write actions are being executed to write in data for each of the memory addresses. Shows beginning of Situation b, where we begin to read out the data at each of the addresses, which are incremented each clock cycle to replicate the counter in the test bench.

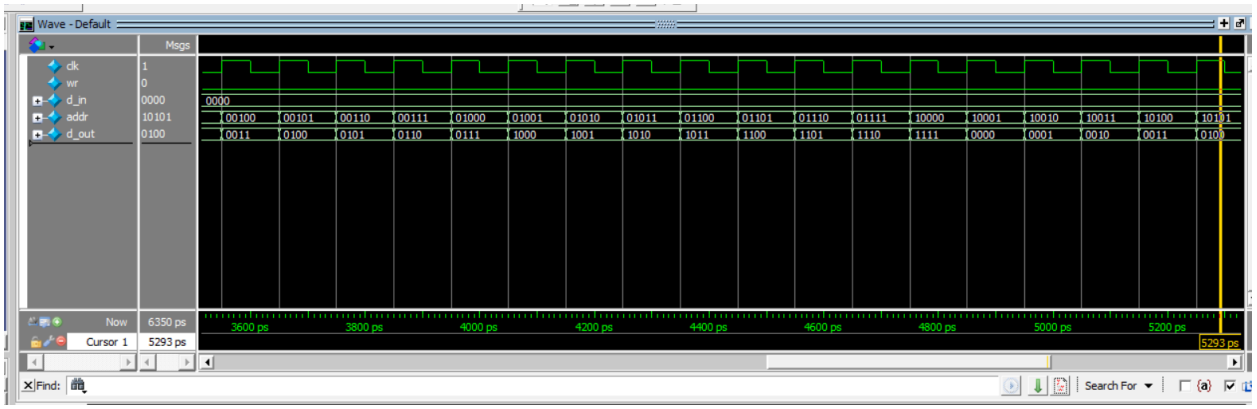

Figure 10.3: ModelSim waveform of the ram32x4_tb, part3

Shows Situation b, where read out the data at each of the addresses, which are incremented each clock cycle to replicate the counter in the test bench

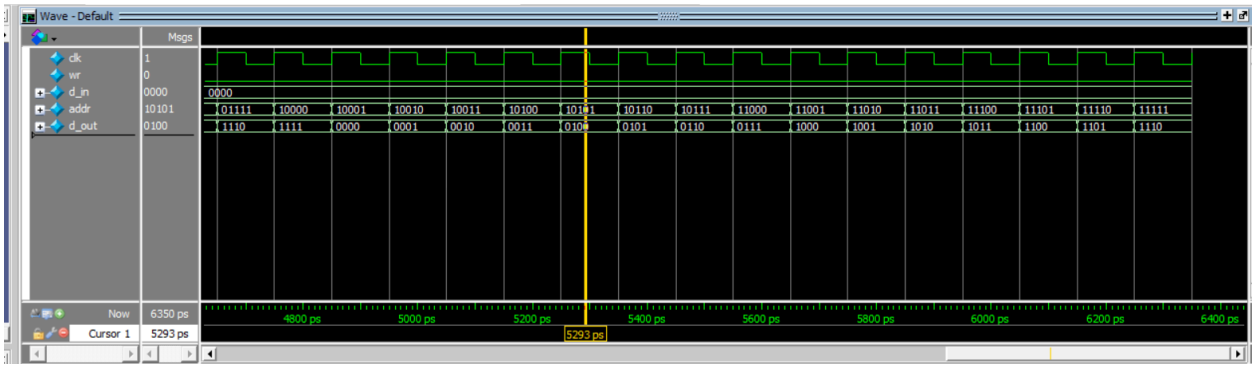

Figure 10.4: ModelSim waveform of the ram32x4_tb, part4

Shows Situation b, where read out the data at each of the addresses, which are incremented each clock cycle to replicate the counter in the test bench

Simulation waveforms of DE1_SoC_tb:

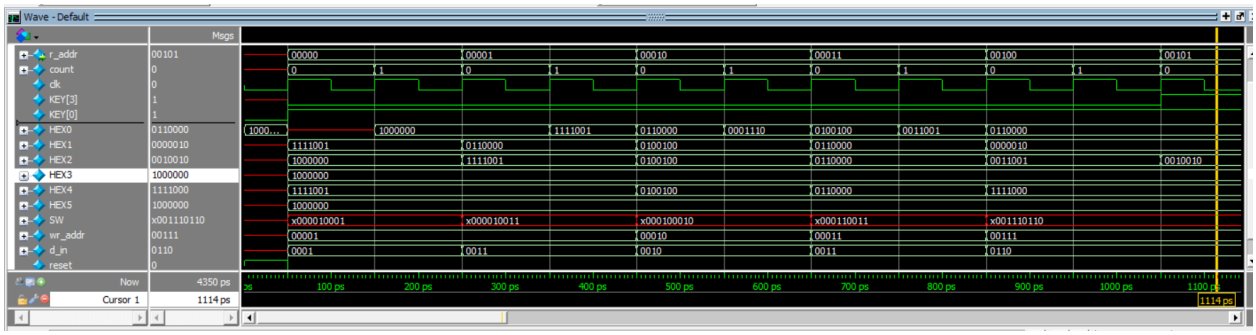

Figure 11.1: ModelSim waveform of the DE1_SoC_tb, part1

Shows situation a. KEY[3] is active-low/off, so write signal is enabled and write actions are occurring. Shows situation a-b, write to the same address. Situation c, writing to new addresses. HEX displays correspond. Shows beginning of situation d, where KEY[3] is active-low/on, so write-signal is turned off and read actions are occurring.

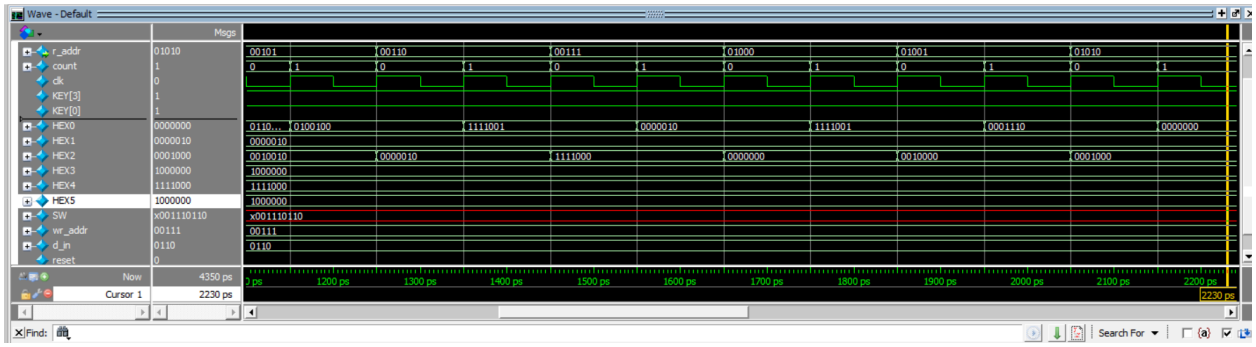

Figure 11.2: ModelSim waveform of the DE1_SoC_tb, part2

Shows situation d, where KEY[3] is active-low/on, so write-signal is turned off and read actions are occurring. Write address and data in are held the same as previously, since they are not updating and write actions are not occurring. The read address is changing every cycle, as intended, to read off the data output at each of the addresses (shown on HEX display).

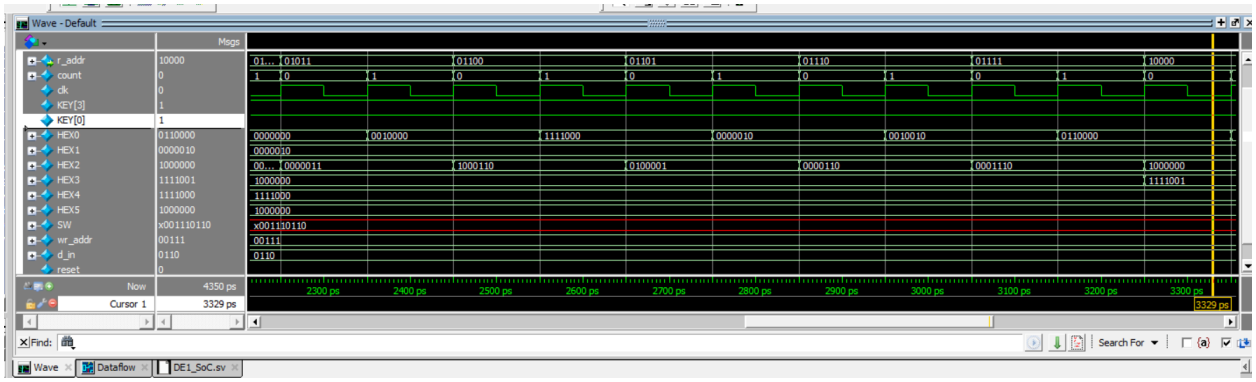

Figure 11.3: ModelSim waveform of the DE1_SoC_tb, part3

Continues to show situation d, where KEY[3] is active-low/on, so write-signal is turned off and read actions are occurring. Write address and data in are held the same as previously, since they are not updating and write actions are not occurring. The read address is changing every cycle, as intended, to read off the data output at each of the addresses (shown on HEX display).

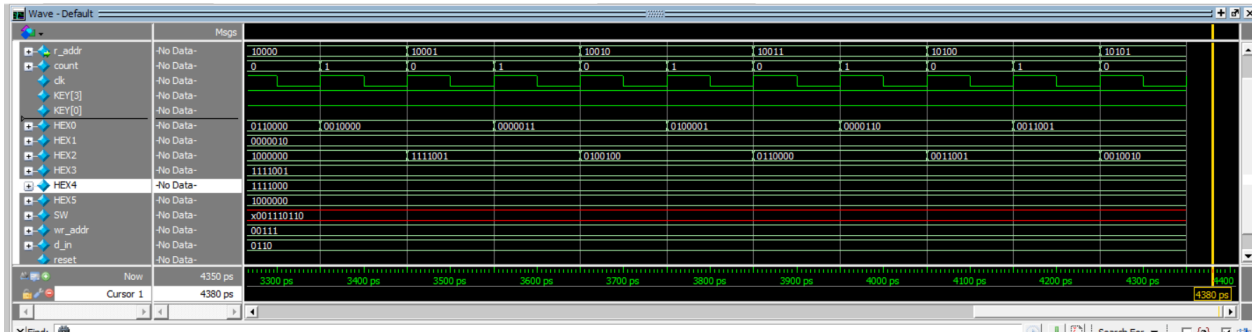

Figure 11.4: ModelSim waveform of the DE1_SoC_tb, part4

Continues to show situation d, where KEY[3] is active-low/on, so write-signal is turned off and read actions are occurring. Write address and data in are held the same as previously, since they are not updating and write actions are not occurring. The read address is changing every cycle, as intended, to read off the data output at each of the addresses (shown on HEX display).

Task #3: FIFO Design

Each of the testbench simulations tested the following situations:

1. **seg7_hex_tb:** Tests the 7-segment HEX display, based on different values of the val(4-bit), which outputs to a 7-segment(7-bit) display.
    a. Test different combinations of 'val' and observe the related 'display'.
2. **buttonPress_tb:** Tests the buttonPress, ensuring it converts to one-pulse even if it is held down for multiple clock cycles.
    a. 4 scenarios of pressing the button for different durations ('in' signal is true for different number of clock cycles in the scenarios)
3. **ram16x8_tb:** Tests behavior of the RAM as the data for an address is requested to be read, and if data is written to an address.
    a. Write signal is enabled. Write actions will be executed 16 times to write in data for each of the memory addresses (data written in is the value of the address).
    b. Write signal is off. Read actions will be executed 16 times to read out data for each of the memory addresses.
        i. d_in is set to 0 for these read actions in the testbench, but it will not update the value at the write address since the write signal is off.
4. **FIFO_Control_tb:** Tests the behavior of the FIFO Control, at a simpler level. Will be tested further in FIFO_tb and DE1_SoC_tb.
    a. Executes write actions (write signal on and read signal off) 25 times, over maximum memory capacity of 16 for FIFO. Write address is incremented over this time, to find next available location to add the new entries to the FIFO.
    b. Executes read actions (write signal off and read signal on) 25 times, over maximum memory capacity of 16 for FIFO. Read address is incremented over this time, to find next location to eject data from FIFO.
    c. Executes write actions (write signal on and read signal off) 10 times, within FIFO capacity. Write address is incremented over this time, to find next available location to add the new entries to the FIFO.
    d. Executes read actions (write signal off and read signal on) 10 times, within FIFO capacity. Read address is incremented over this time, to find next location to eject data from FIFO.
5. **FIFO_tb:** Tests the behavior of the ram16x8 and FIFO_Control as the inputBus is provided, and the status of the FIFO (through empty and full outputs) and data output is outputted from the FIFO module. The following situations occur in the testbench:
    a. Execute read action on empty FIFO - will display empty signal as true
    b. Fill FIFO to full capacity - will turn on full signal
    c. Attempt to add to already full FIFO - full signal will stay on
    d. Read from the full FIFO - full signal will turn off, outputBus will change
    e. Read and write simultaneously - full signal stays off, outputBus will change as reads are done, since new output data is being read out
6. **DE1_SoC_tb**: Tests behavior of the buttonPress, seg7_hex, and FIFO modules to ensure that the reset, write, and read signals are processed properly, then the FIFO behavior is correct based on these signals/actions, and then the HEX displays correspond with the data being inserted into and ejected from the FIFO. The LEDR displays should also correspond with the status of the FIFO,

with LED's representing the full and empty status of the FIFO. This testbench is modeled similarly to the FIFO_tb, but it instead toggles the on-board SW and KEY inputs for the corresponding signals (listed below).

a. Execute read action on empty FIFO - will display empty signal as true
b. Fill FIFO to full capacity - will turn on full signal. HEX5-4 displays will change as there is data being inputted.
c. Attempt to add to already full FIFO - full signal will stay on. HEX5-4 displays will change as there is data being inputted.
d. Read from the full FIFO - full signal will turn off, HEX1-0 displays will change. Write is off, and HEX5-4 displays are not changing since data is not being inputted.
e. Read and write simultaneously - full signal stays off, HEX1-0 displays will change as reads are done, since new output data is being read out. HEX5-4 displays will change as there is data being inputted.
f. HEX displays for each of the situations above:
    i. HEX1-0 shows current data output
    ii. HEX5-4 shows data input
g. On-board inputs to the DE1_SoC for each of the situations above:
    i. KEY[3] is the read action
    ii. KEY[2] is the write action
    iii. KEY[0] is the reset signal
    iv. SW[7:0] provides the data input to the FIFO
h. On-board outputs to the DE1_SoC for each of the situations above:
    i. LEDR[9] shows if the FIFO is full
    ii. LEDR[8] shows if the FIFO is empty

Descriptions of the simulation waveforms are provided below the figures. Or labeled with the situation that it corresponds to, tracing back to the detailed descriptions listed above.

Simulation waveforms of seg7_hex_tb:

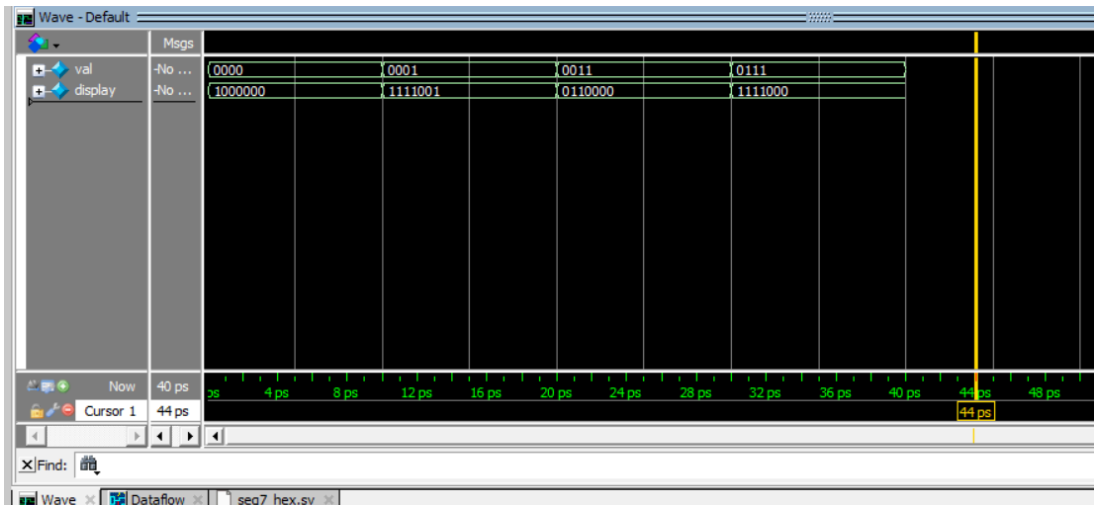

Figure 12: ModelSim waveform of the seg7_hex_tb

Shows situation a. Drives in 4 different random values of val, which corresponds to the proper 7-segment display that would show the hexadecimal value of 'val' on a HEX display.

Simulation waveforms of buttonPress_tb:

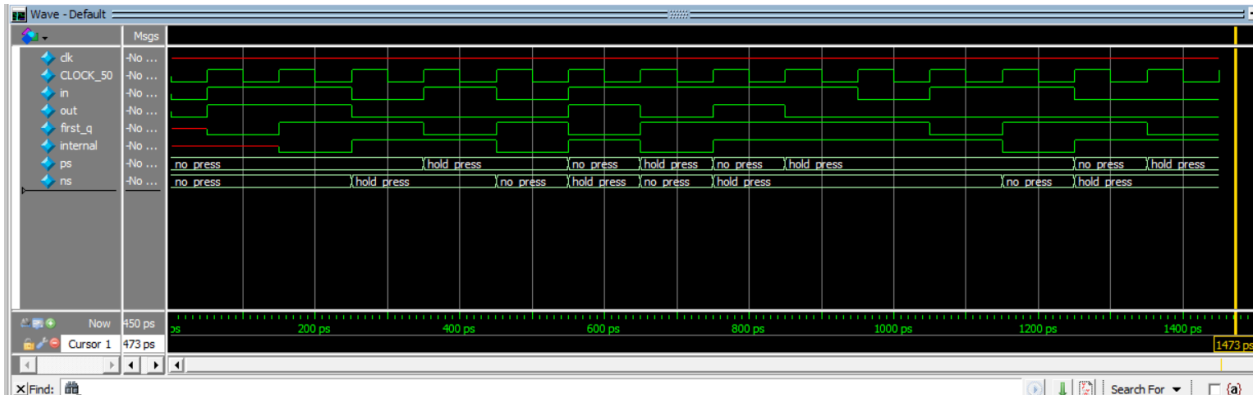

Figure 13: ModelSim waveform of the buttonPress_tb

Shows situation a. Ensuring that the press of the button converts to one-pulse.

Simulation waveforms of ram16x8_tb:

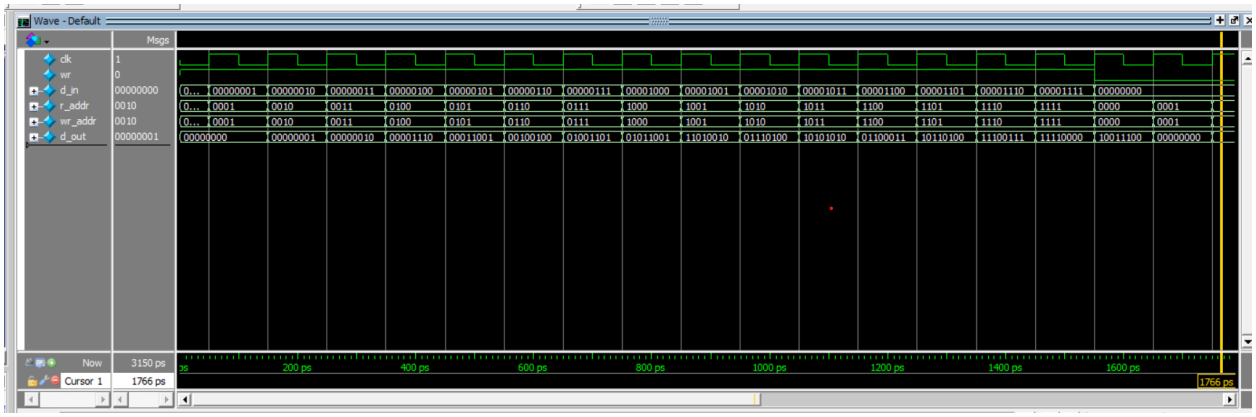

Figure 14.1: ModelSim waveform of the ram16x8_tb, part 1.

Shows situation a, write signal is on.

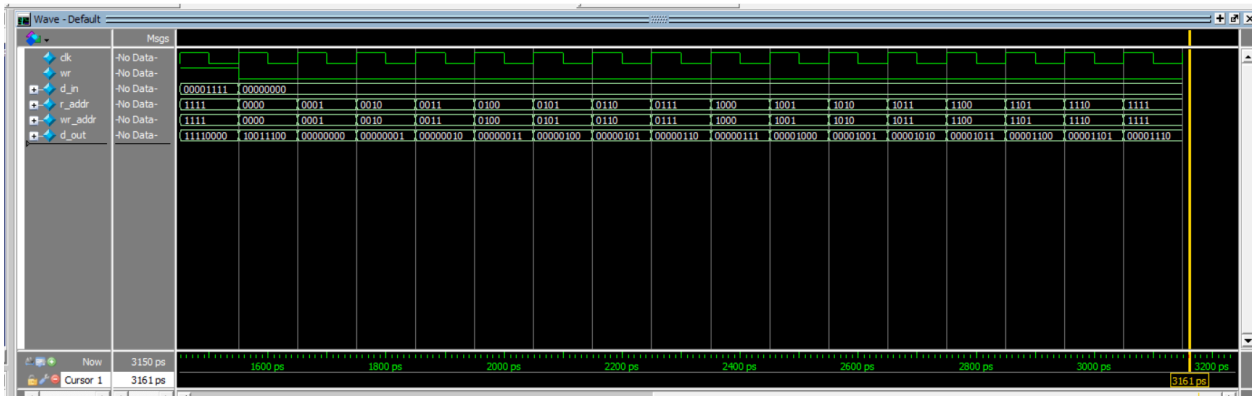

Figure 14.2: ModelSim waveform of the ram16x8_tb, part 2.

Shows situation b, write signal off.

Simulation waveforms of FIFO_Control_tb:

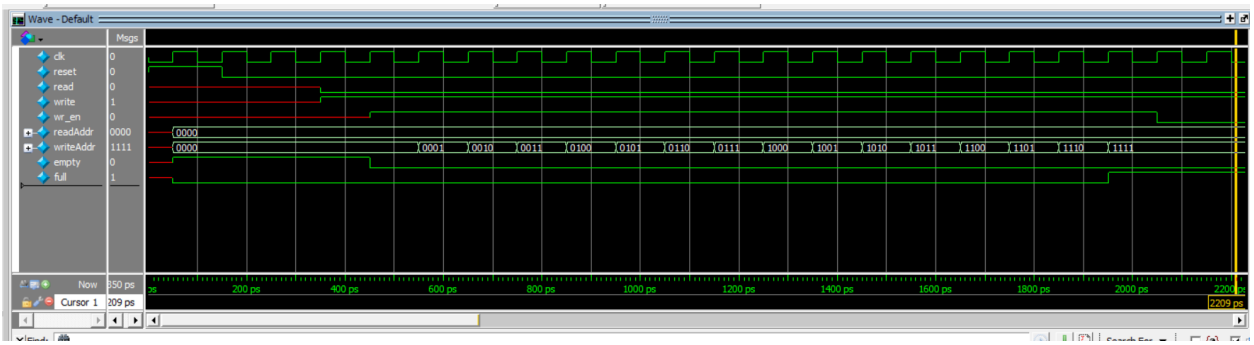

Figure 15.1: ModelSim waveform of the FIFO_Control_tb, part 1.

Shows situation a, where full signal will turn on because write actions have occured to reach capacity of FIFO. writeAddr is incremented to find next available location to add new entries to the FIFO.

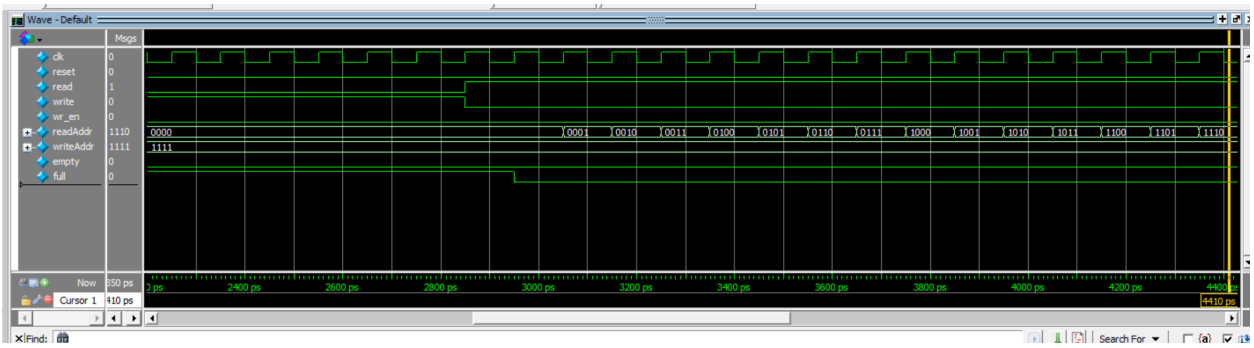

Figure 15.2: ModelSim waveform of the FIFO_Control_tb, part 2.

Shows situation a, where full signal will turn on because write actions have occured to reach capacity of FIFO. Shows situation b, where readAddr is incremented to find next location to eject data from FIFO.

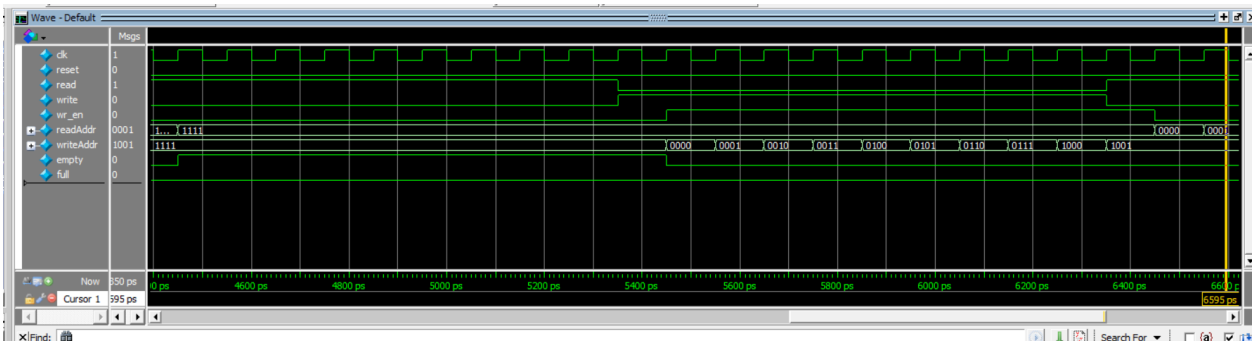

Figure 15.3: ModelSim waveform of the FIFO_Control_tb, part 3.

Shows situation b, where empty signal will turn on because read actions have occured to empty the capacity of the FIFO.

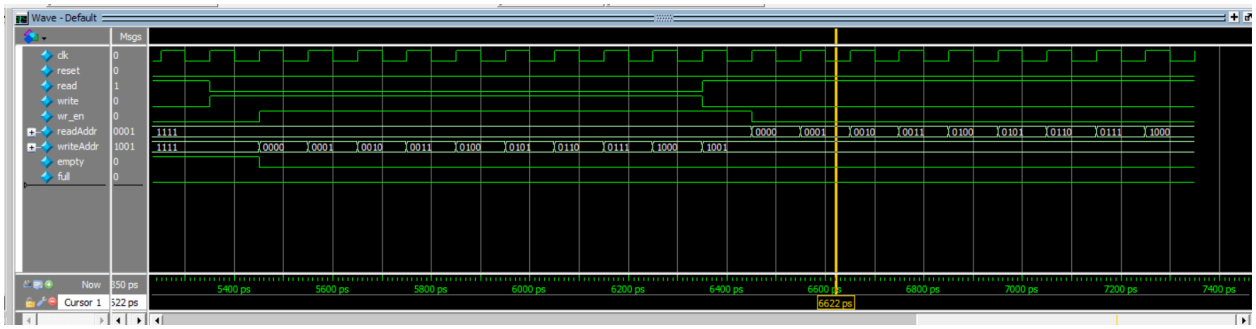

Figure 15.4: ModelSim waveform of the FIFO_Control_tb, part 4.

Shows situation c-d. Where FIFO is written to 10 times, then read from 10 times. So write signal is held on for a period of time, and read signal is held on for a period of time.

Simulation waveforms of FIFO_tb:

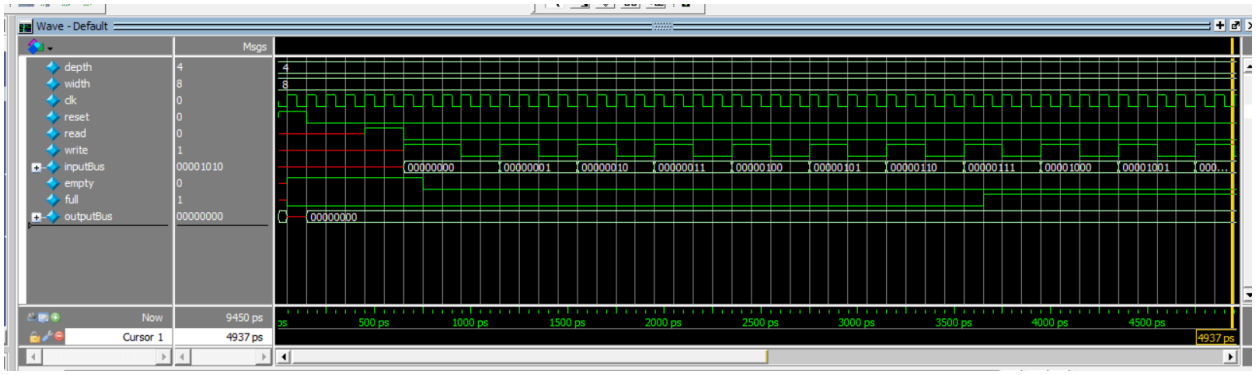

Figure 16.1: ModelSim waveform of the FIFO_tb, part 1.

Shows situation a (displays empty), read action on empty FIFO. Shows situation b, fill FIFO to full capacity (full signal is turned on near end of screenshot). Shows situation c, attempting to add to already full FIFO, as write signal is still turned on, but full continues to stay on.

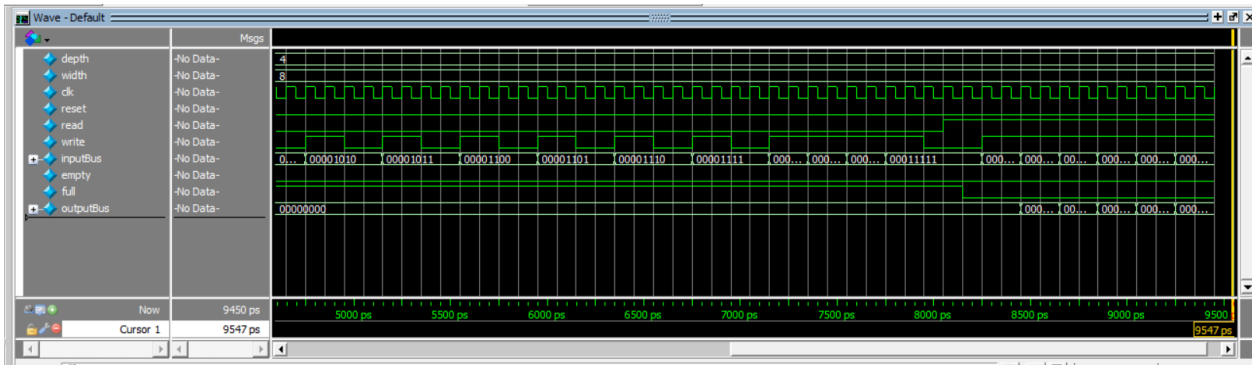

Figure 16.2: ModelSim waveform of the FIFO_tb, part 2.

Continues to show situation c, attempting to add to already full FIFO, full signal continues to stay on. Shows situation d, reading from the full FIFO (read is on and write is off, so outputBus changes, and full is turned off). Shows situation e, reading and writing simultaneously, so outputBus changes, and full continues to stay off.

Simulation waveforms of DE1_SoC_tb:

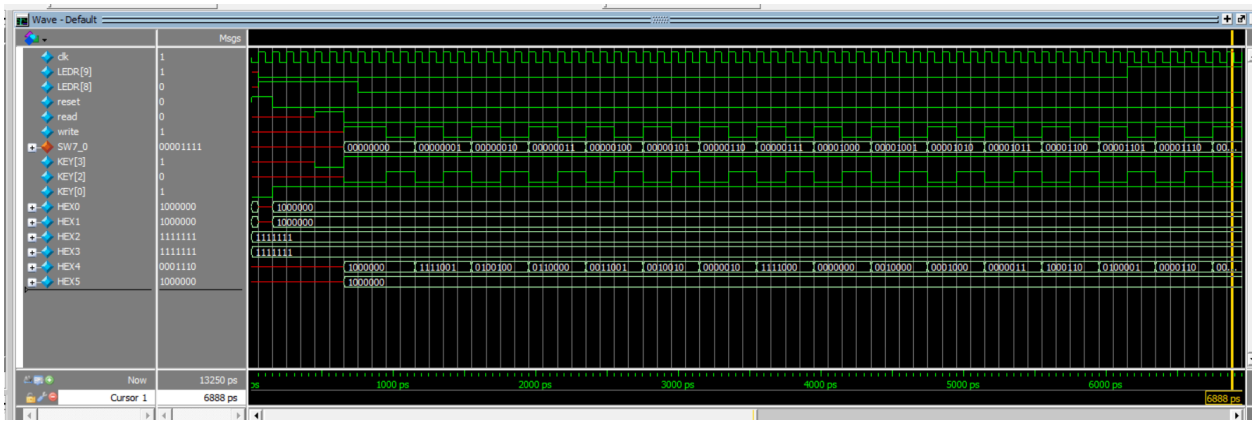

Figure 17.1: ModelSim waveform of the DE1_SoC_tb, part 1.

Shows situation a. Shows beginning of situation b, as the LEDR[9] for full turns on near the end of the screenshot. During this time, HEX5-4 was changing as there is data input being driven into the FIFO for the write actions.

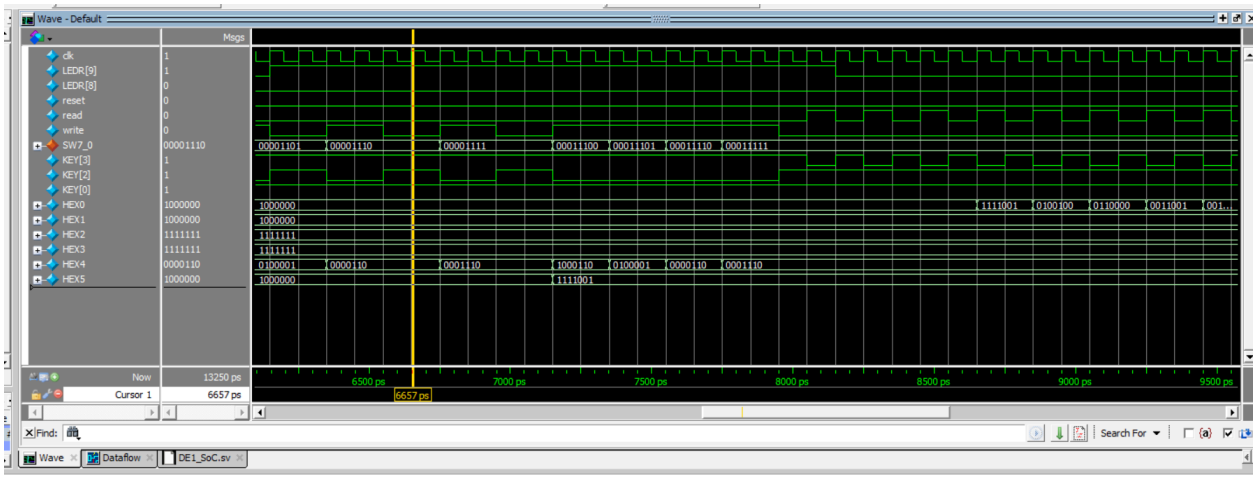

Figure 17.2: ModelSim waveform of the DE1_SoC_tb, part 2.

Shows situation b, filling the FIFO to full capacity. Full signal/LEDR[9] will stay on for a period of time. Continues to show situations c, attempting to add to already full FIFO, so full signal stays on. Shows situation d, which begins to read from the full FIFO, so the full signal will turn off. The HEX1-0 displays will also change, as the current data output changes due to the read actions being executed (since data is being ejected from the FIFO). During this time, HEX5-4 was changing as there is data input being driven into the FIFO for the write actions.

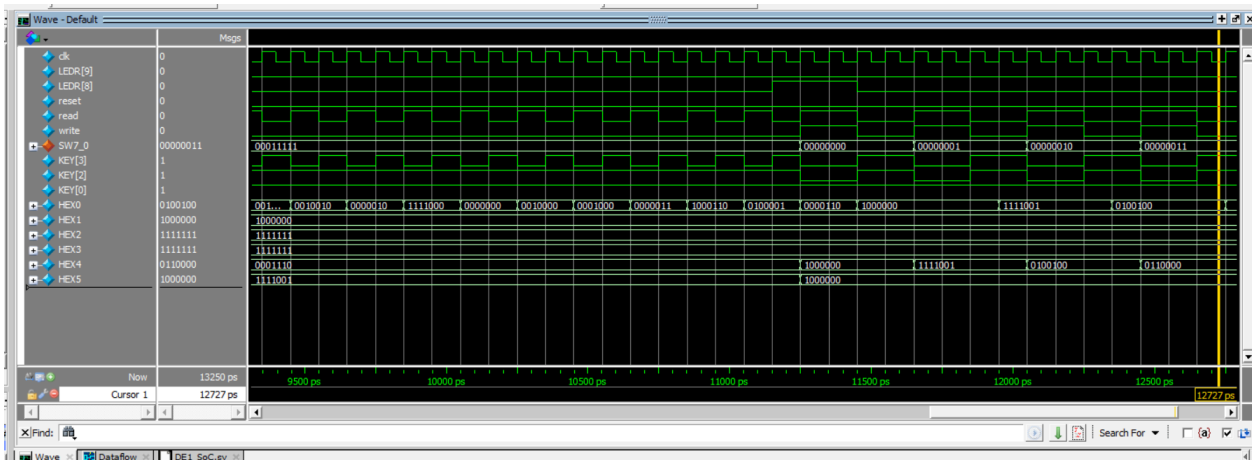

Figure 17.3: ModelSim waveform of the DE1_SoC_tb, part 3.

Shows situation d, which begins to read from the full FIFO, so the full signal will turn off. The HEX1-0 displays will also change, as the current data output changes due to the read actions being executed (since data is being ejected from the FIFO). Eventually, the empty signal turns on momentarily since many read actions have been executed and the FIFO is now empty. Shows situation e, where read and write are done simultaneously. HEX1-0 displays will change as reads are done, since new output data is being read out. During this time, HEX5-4 was changing as there is data input being driven into the FIFO for the write actions.

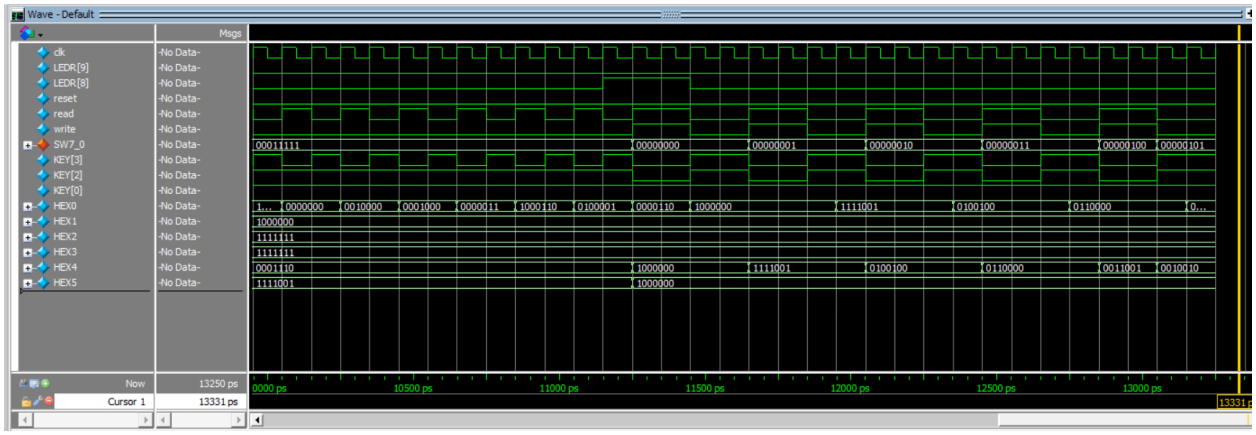

Figure 17.4: ModelSim waveform of the DE1_SoC_tb, part 4.

Shows continuation of situation d, which has read actions occurring from the FIFO, getting to an empty FIFO. Shows end of situation e, which had read and write actions occurring simultaneously. During this time, HEX5-4 was changing as there is data input being driven into the FIFO for the write actions.

Final Product

The overarching goal of this project was to implement RAM modules for three different tasks. Below, we have overviewed the finished system for each task, presenting the implemented components asked for in the lab spec.

Task 1: Single Port 32x4 RAM

1. We implemented a single-port 32x4 RAM. The system takes in an address and data input and stores it in the RAM.
    a. The system will "read"/display the data stored at an address (always carrying out a "read" action, as it is not specifically requested by the user due to the nature of the single-port RAM).
    b. The system will "write" the data input to an address when a "write" occurs.
2. The system takes in an address (5-bit) and data input (4-bit), which are driven in by switches accessible to the user (SW[8:4] for address and SW[3:0] for data input)
3. It also takes in a write-signal, which will then drive the data input to the address
    a. Write signal is controlled by a switch accessible to the user (SW[9])
4. The system also displays the address value, data input, and data output on HEX displays
    a. The address value is shown on the HEX5-4 displays
    b. The data input is shown on the HEX2 display
    c. The data read output is shown on the HEX0 display

Task 2: Dual port 32x4 RAM with separate read and write controls

1. We implemented a dual-port 32x4 RAM. The system takes in a read address and write address, as well as a write enable, to determine when a write action will occur.
    a. A "write" action occurs when a write address is provided, data input is provided, and write enable is high—this action will write the data input to the location (write address) provided.
    b. A "read" action occurs when a read address is provided, which is provided every 1 second by the counter in the system. The "read" action displays the data at the read address.
2. The system also includes a counter, which ensures the data of a different read address is displayed every 1 second.
3. The system displays the read address on HEX3-2 displays, and the read data output on HEX0.
4. The system displays the write address on HEX5-4 displays, and the write data on HEX1.
5. It also takes in a write enable signal, which will then drive the data input to the address
    a. Write signal is controlled by a switch accessible to the user (KEY[3])
6. The system also implements a reset signal, which is controlled by KEY[0].

Task 3: FIFO Design

1. We implemented a FIFO, which uses a memory module to store data when written to and ejects data when read from.
    a. The ram16x8 served as the memory module, it had 16 words and 8 bits to produce a FIFO with the space for 16 entries, each entry having 8-bits available for its value.
2. We used a control module to organize the process and expected behavior of the FIFO. The FIFO Control module included specifications for the following types of actions: (1) when "read" and "write" are requested simultaneously, (2) when read is requested and the FIFO is not empty, (3) when write is requested and the FIFO is not full. For each of the actions, the FIFO control

includes details on behavior to update the write address and read address accordingly, as well as the full and empty signals.

3. When a “write” action occurs, the data input is added as a new entry to the FIFO, first-in.
    a. The new value is stored at the next-available spot, first-in.
4. When a “read” action occurs, the data output is ejected from the FIFO, first-out.
    a. The “least recently stored” data is ejected, first-out.
5. The system displays the output data on the HEX1-0 displays.
    a. Resulting from a “read” action
6. The system displays the input data on the HEX5-4 displays.
    a. Resulting from a “write” action
7. The system’s read signal is controlled by the KEY[3]. The write signal is controlled by the KEY[2]. The reset signal is controlled by the KEY[0]. All of the KEY inputs are processed by a ‘buttonPress’ module to ensure they last for one clock-cycle, allowing the FIFO to properly process these actions and make the necessary updates to the system.
8. The system also includes an empty signal displayed on LEDR[8] and full signal displayed on LEDR[9], to show the status of the FIFO as it is being written to or read from.

Section 3: Appendix: SystemVerilog Code

1) Task 1, DE1_SoC.sv

2) Task 1, ram32x4.sv

3) Task 1. Seg7_hex.sv

4) Task 2, DE1_SoC.sv

5) Task 2, seg7_hex.sv

6) Task 2, counter.sv

7) Task 2, ram32x4.v

8) Task 2, ram32x4_tb.sv

9) Task 3, DE1_SoC.sv

10) Task 3, seg7_hex.sv

11)Task 3, buttonPress.sv

12) Task 3, ram16x8.v

13) Task 3, ram16x8_tb.sv

14) Task 3, FIFO_Control.sv

15) Task 3, FIFO.sv