**CMPE-260 Laboratory Exercise 06**

**Design and Verification of an SRAM with Memory Controller**

By submitting this report, I attest that its contents are wholly my individual writing about this exercise and that they reflect the submitted code. I further acknowledge that permitted collaboration for this exercise consists only of discussions of concepts with course staff and fellow students; however, other than code provided by the instructor for this exercise, all code was developed by me.

John Judge

John Judge

Performed 4/28/16

Submitted 5/12/16

Lab Section 04

Instructor: Dr. Lou Beato

TAs: Nick Barlow

Sergio Martin

Lecture Section 01

Professor: Dr. Reza Azarderakhsh

**Table of Contents**

1. Abstract …………………………………………………………………………...1
2. Design Methodology………………………………………………………………1
3. Results……………………………………………………………………………..3
4. Conclusion………………………………………………………………………...4
5. Appendix…………………………………………………………………………..4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Abstract:**

In this exercise, and SRAM unit with a memory controller were designed in VHDL using the Xilinx IDE. These units were then instantiated in a top-level wrapper called the IO\_BUS and the entire design was then simulated using ModelSim. Once the design was verified to be correct it was then routed to a NEXYS 3 Spartan 6 FPGA and through the use of the Seven Segmented Decoder and Display units developed in previous exercises the entity was once again tested to verify the unit worked as desired. Using several key test cases the unit was determined to be operating correctly and the exercise was determined to be successful.

**Design Methodology:**

*The SRAM Unit*

In this exercise the first unit created was the SRAM unit. An SRAM is essentially a memory array unit with read and write capabilities. The SRAM developed in this exercise was capable of storing up to eight 4-bit vectors at a time. The unit had a 4-bit data-in, a 3-bit address, a 1-bit write enable, and a 1-bit output enable input signals. There was also two 1-bit *Addr1* and *Addr2* address signals used to offset the address when more than one address was desired to be read from sequentially in what was called a “burst read.” The output of the unit was a single 4-bit data-out signal representing data stored at the desired address when the read functionality was used.

The SRAM was described using a behavioral architecture for ease since an architectural description would require the instantiation of many sequential components. The memory storage of the unit was created with a 2-D array that was eight addresses long and 4-bits wide. On instantiation, the array held undetermined values until data was written to an address. There also was no reset functionality, meaning that data could not be cleared from memory. Instead memory could be overwritten and if desired, null could be stored at an address essentially clearing that spot in memory. Using two case statements, the SRAM was designed to allow the reading of data on an active-high read enable signal and the writing of data on an active-high read enable signal. Both of course happening at the address desired by the address signal and its offsets, *Addr1* and *Addr2*. This unit also did not function with regard to a clock, instead a separate Memory Controller unit needed to be designed.

*The Memory Control Unit*

The Memory Control unit was used to handle desired read and write operations and drive the inputs of the SRAM unit. This unit was essentially a Moore State Machine used to control the SRAM unit. Using an 8-bit *Bus\_ID* signal the memory controller could select a desired SRAM unit, however, in this exercise only one SRAM unit was instantiated with its ID being “00000000.” The Memory Controller also had a 1-bit Read/Write signal, *RW,* where ‘0’ represented reading and a ‘1’ represented writing. To handle burst reading, the reading of four sequential addresses in a circular operation, a 1-bit signal called *burst* was used where ‘0’ represented a desired read from one address and ‘1’ enabled the burst read*.* A signal called *ready* was used as the determinate to move between states and a *reset* signal was used to return to the Idle state. Since the unit was a Moore State Machine, all changes in states happened on the rising edge of a *clk* signal. Using this inputs the Memory Controller could drive the output enable (*oe*), write enable (*we*), and address offsets *addr1* and *addr2* the SRAM used to determine which memory operation should take place.

As stated, the Memory Control Unit was a Moore State Machine and as a result was described using a behavioral description. A Moore machine was determined to be preferential over a Mealy design since the output of the controller would change dependent on the state the machine was in and not on clock edges. This was preferential since outputs dependent on the state have less of a delay associated with them than if they changed on a clock edge. This created a more efficient design than if a Mealy State Machine had been used.

In the behavioral description of the memory controller a total of fifteen states were declared. A graph representing the states and the required conditions to change between states can be seen in Figure (1.0). As seen in the figure, all state changes required a change of the *ready* signal. On reset and after all operations the machine returned to the Idle sate. In order to leave Idle a valid *bus\_ID* signal and a high *ready* signal were necessary. From here the machine entered the Decision state where the state of the *RW* signal, as well as a change in the *ready* signal, determined if the machine would begin reading or writing memory. If a write operation was desired than the machine would transition to the Writing state where the *oe* signal was driven to ‘1’ while remaining in the writing state thus indicating to the SRAM unit to store it’s data-in at the address signified. On the change of the *ready* signal back to ‘0,’ the machine would return to the Idle state.

Had a writing operation been desired when in the Decision state the machine would have entered the Reading state. In the reading state the value of the *burst* signal, as well as a change to the *ready* signal, would determine whether the machine would transition to the Read Single state or Read Burst0 state accordingly. When in the Read Single or its subsequent Read Single Wait state the *oe* was driven to ‘1,’ indicating to the SRAM to read the data in memory at the address specified by it’s input. The use of the wait state, which similar states were also used when in the burst read states, was necessary because all state changes needed happen dependent to *ready* but *ready* could not be added to the sensitivity list because the process also needed to only be dependent on the clock, since it was determined a Moore State Machine was the best. Had *burst* signal indicated a burst read was desired the machine would have iterated through the burst read states and their subsequent hold states. In these states the output signals of *addr1* and *addr2* were incremented to count between 0 and 3 in binary if *addr2* was the most significant bit and *addr1* was the least significant bit. These signals could then be used by the SRAM to increment past the beginning address to the sequential three addresses to be read from when burst reading. After finishing the burst read the machine would return to Idle. These states, and the use of *ready* to change states provided the control for the SRAM unit.

*The IO Bus*

In order to simulate and test the SRAM and Control Units a top-level wrapper was needed. This wrapper was called the IO Bus and a diagram of its internal sub components can be seen in Figure (1.1). As seen in the figure, the IO Bus mapped the output of the control unit to the inputs of the SRAM with the SRAM’s address and data-in ports still connected to an outside source. All input ports of the Memory Controller were connected to outside sources. In addition, the use of the Seven-Segmented Decoder and Display units were included in the IO Bus to convert the data-out of the SRAM to binary-coded decimal so that when the design was later implemented on the FPGA the data-out of the SRAM could be seen and analyzed. The output of the IO Bus was therefore the outputs of the Seven Segmented Display, which have been described in previous exercises. The four cathode outputs enabled or disabled the individual seven-segmented LEDs and the seven anode signals each enabled or disabled the LEDs within the display. Using this wrapper as a unit under test a test bench could be created to verify the design operated as desired.

**Results:**

*The Design of the Test Bench*

In order to test the code developed during this exercise a test bench was created with the IO Bus as the unit under test. In this test bench the inputs of the IO Bus were driven to attempt to read from invalid Bus ID’s and then to preform memory operations using a valid Bus ID. Data was then written to addresses in the SRAM by adjusting the *RW, Ready, Address,* and *Data-in* signals accordingly. Then several single read functions were used to read data in the SRAM at single addresses. This was followed by two subsequent burst reads, the first of which read data from addresses 1 to 4 and the second which test the wrapping functionality of the SRAM by reading from addresses 6 to 1. The result of this test bench was analyzed for both behavioral and post-route simulations.

*The Results of the Behavioral Simulation*

The IO Bus was then simulated using the test bench in a behavioral simulation. The results of the simulation were then analyzed to determine if the unit was correctly handling the memory operations desired. Upon inspection the unit correctly handled the writing functionality. A capture of the waveform generated by ModelSim indicating that the writing function was working can be seen in Figure (1.2). In this figure it can be seen that one clock cycle after the desired address and data inputs changed, the memory array changed.

To determine if the reading functionality was working properly the simulation results were further analyzed. Figure (1.3) shows the complete simulation results. In this figure, the writing functionality, which can also be seen in Figure (1.2), is shown again. Following the writing of data at every point in the array, the memory is then accessed with several single read functions. Then two burst read operations, one from address 1 to 4 and one from address 6 to 1, occurred. A zoomed-in waveform showing the burst read functionality can be seen in Figure (1.4). In this figure the burst read operations could be seen as well as the states the state machine was in during the operations. This waveform verifies that the unit correctly handled writing, reading, and burst read functions.

*Results of the Post-Route Simulation*

Following the Behavioral testing a Post-Route simulation took place in order to simulate the delays to be expected in the hardware implantation. During this simulation the same test bench used for the behavioral simulation was used. A portion of the resulting waveform can be seen in Figure (1.5). In this figure the delays can be seen in the encircled portions of the figure. The delays can further be seen in Figure (1.6) where the image has been zoomed in and the delays appear more substantial. In this figure it can also be seen hat the resulting delays are less than once 100 ns clock period in width. The design therefore was verified to be operating correctly since none of the operations resulted in an output different from the behavioral testing and no timing issues were discovered. It was therefore determined that the design could be implemented on an FPGA.

*FPGA Implementation*

Following the post route simulation the design was routed to the NEXYS 3 FPGA and was tested using the board’s switches to represent all bits of the *data-in* and *address* inputs. The final switch was used for the *RW* signal. The *burst, ready, reset,* and least significant bit of the *bus\_ID* were mapped to buttons. Three of the four possible seven-segmented displays were used to show the data being read from any address in memory. For this implementation the use of any debouncing code was unnecessary due to the hold states used in the Memory Controller. Therefore the design operated correctly despite using the buttons as inputs. The design was then tested using similar test cases to those used in the test bench. In all of these tests the board responded appropriately. For each single read or burst read the seven segment display updated appositely and displayed zero’s when not reading from memory. As a result the design was verified to be implemented correctly and the design summary was consulted to determine the efficiency of the design in terms of components and area.

*Design Summary*

The Design Summary tab of the project was then consulted to check the occupied number of slices, flip flops, and LUTs. A capture of the design summary can be seen in Figure (1.6). Based on the Design summary the project used a total of 106 LUTs as well as 76 registers for a total of 42 slices. The design therefore used 1% of the total amount of slices available in the FPGA.

**Conclusion:**

There were some lessons to take from doing the exercise. The most major of which was the use of SRAMs in data storage. The design and use of Moore State Machines was also further explored. Although some minor bugs occurred during the design of the Memory Controller and the SRAM they were eventually rectified through routine debugging procedures. Once the design was properly synthesized and tested during simulation it was implemented on the FPGA. Following some more testing the self-testing function, as well as the rest of the design, was verified to be correct. Overall, the exercise was conducted successfully, with the units working together to successfully create a memory storage device. As a result this exercise was considered to be successful and the lessons learned during this exercise will make it invaluable in the future exercises when memory units are needed to be designed and used.

**Appendix:**

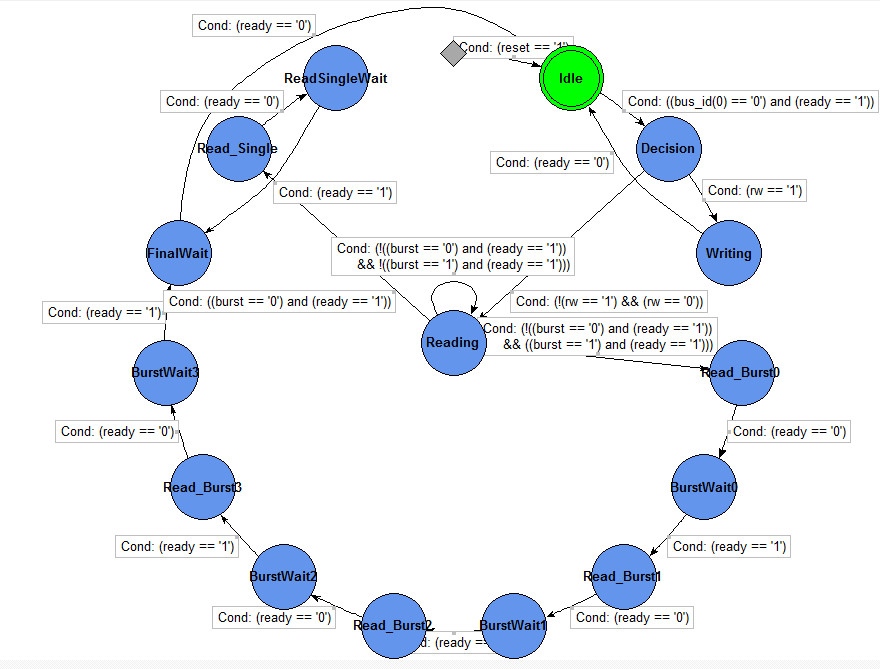


Figure (1.0): The Memory Controller

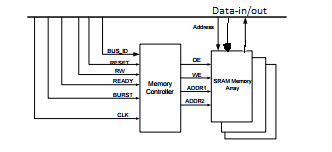


Figure (1.1): The IO Bus

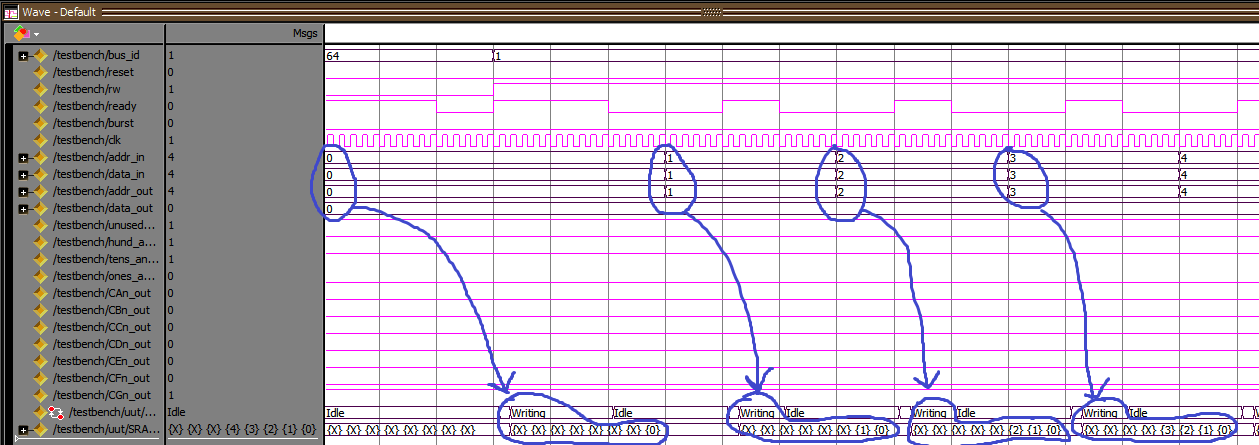


Figure (1.2): The Behavioral Simulation Writing to Memory

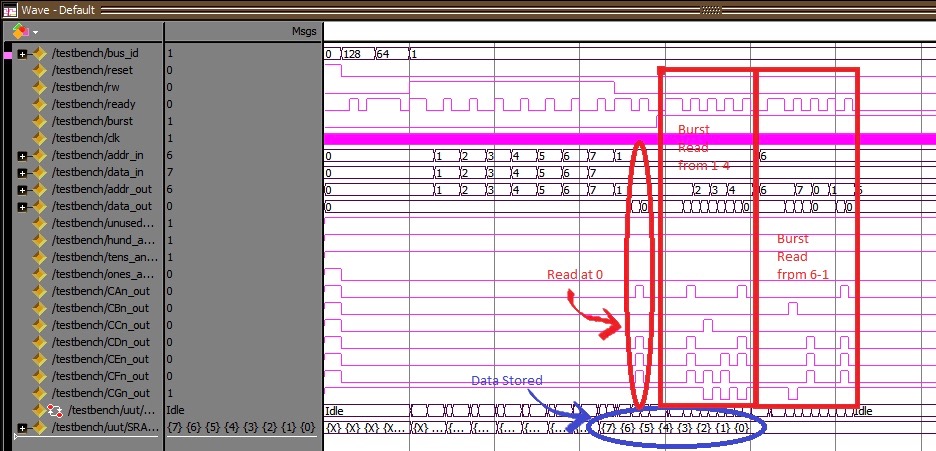


Figure (1.3): The Behavioral Simulation Results

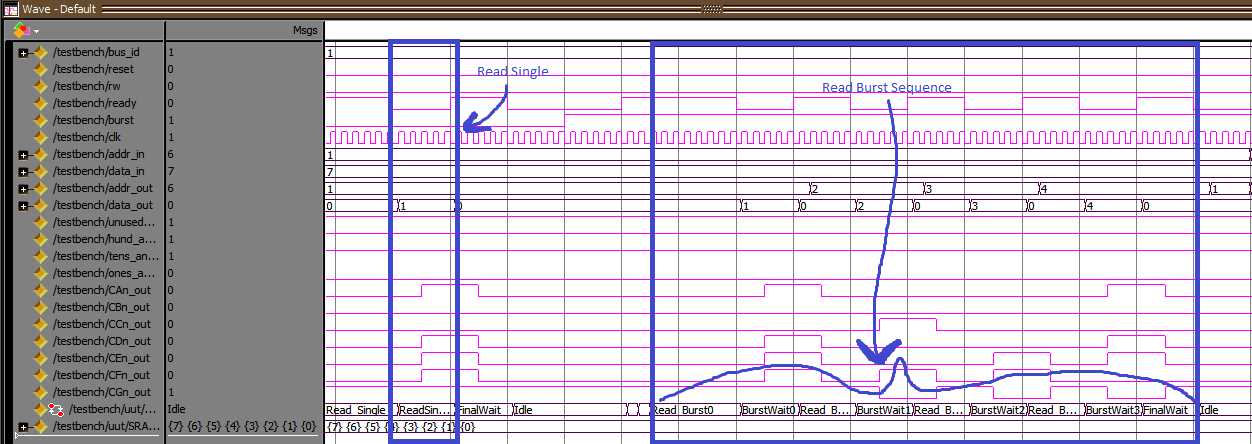


Figure (1.4): The Behavioral Simulation Reading From Memory

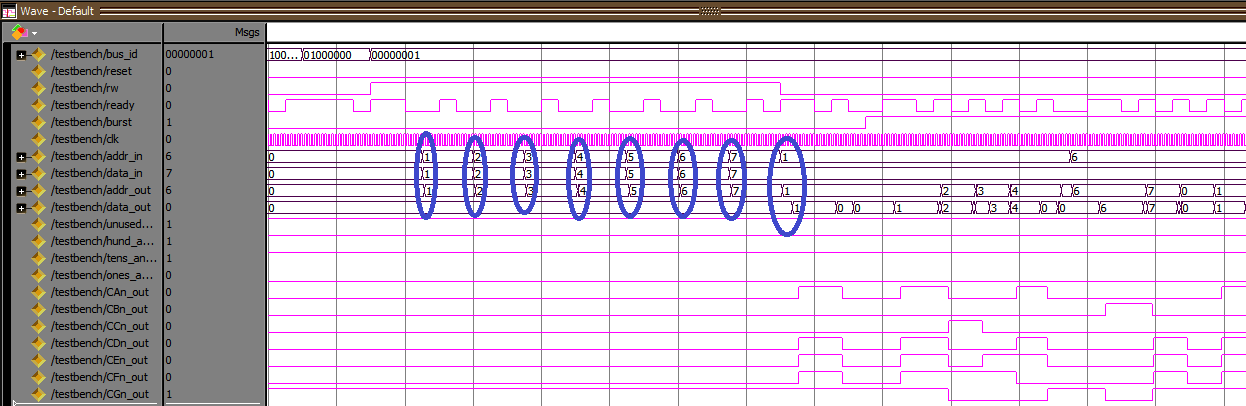


Figure (1.5): The Post-Route Simulation Results

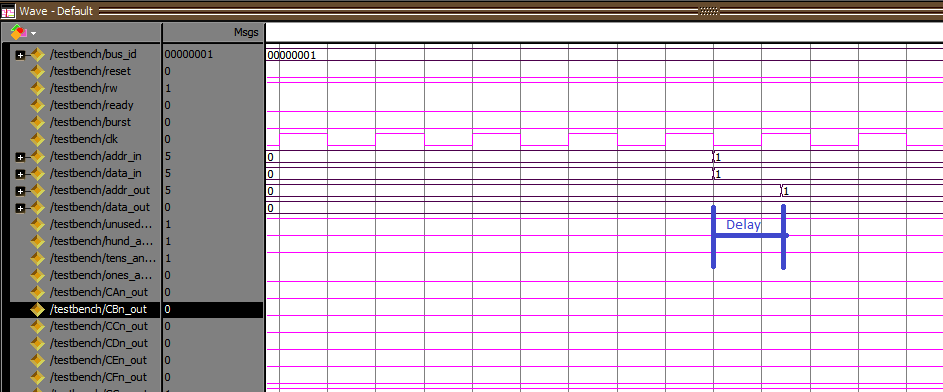


Figure (1.5): Post-Route Delay

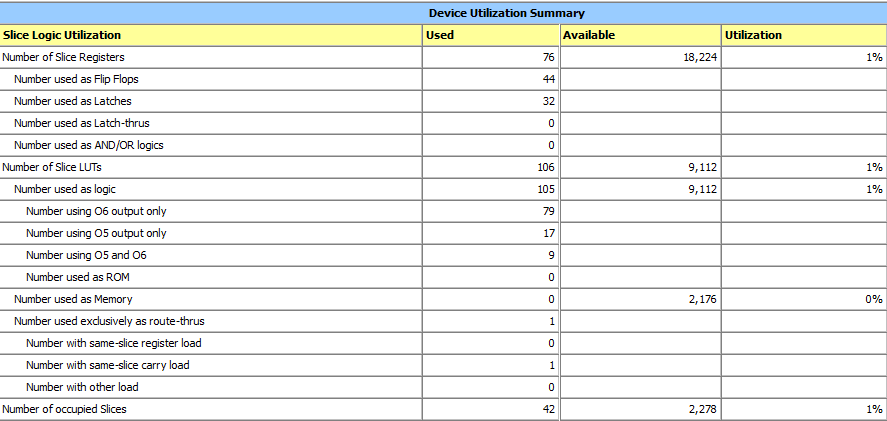


Figure (1.6): Design Summary

