**Lab 6 – Memory FIFO**

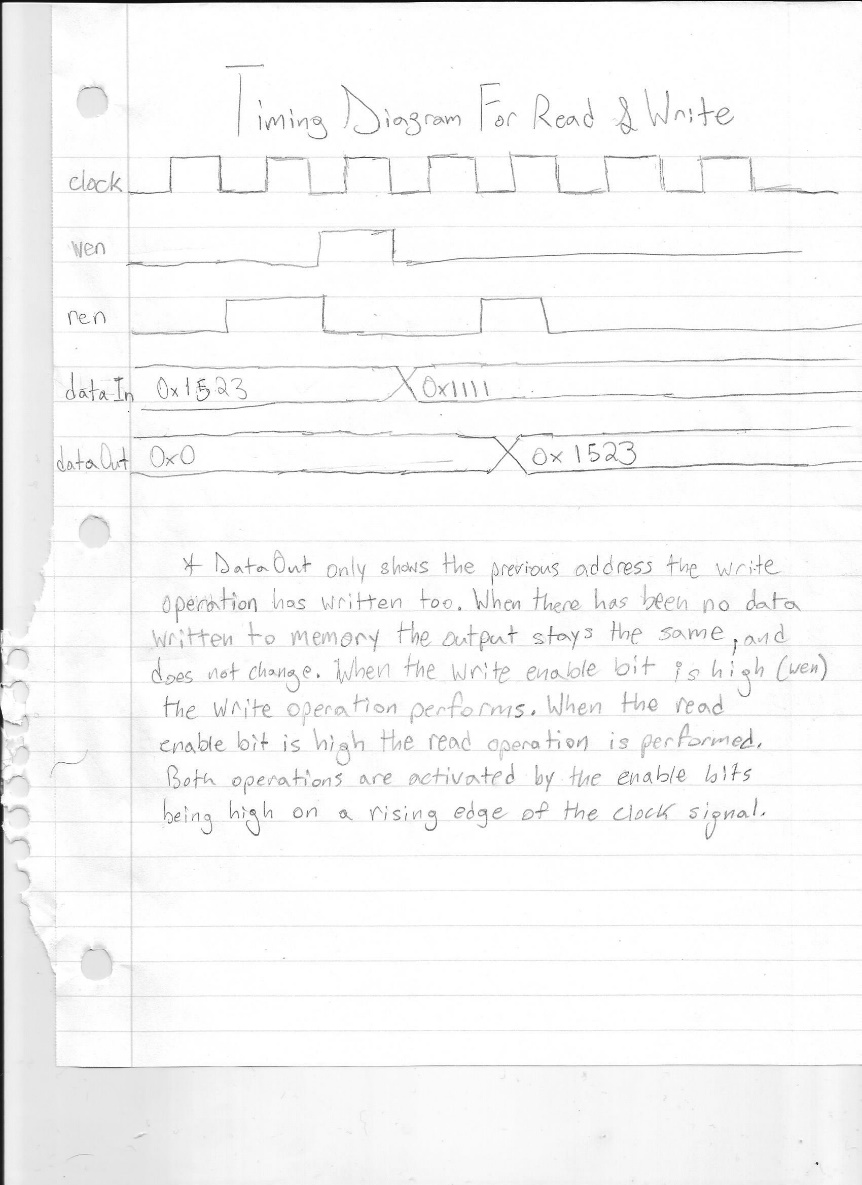
**Problem:**

The goal of this lab is to design a FIFO memory system to connect with M9K memory blocks on the Cyclone IV E board. The system is required to save 15-bit numbers, and store a minimum of 8 off them. The write signal in the system has to be active-high, and always reads the read address location. Memory needs to be initialized so that each entry’s value equals its address. The memory system needs to use a toggle switch to switch between the two write and read modes. When the system is in write mode, the push of a button causes it to read the values of 15 toggle switches and stores them into the next available location in memory. After the value is stored from the write mode, the memory write address needs to be incremented so that the next memory location is used on another write to memory. When the system is in read mode, the push of a button needs to read values out of the memory starting with the first unread location. Each push of a button in read mode needs to cause the next location in memory to be read, and at the end of memory, the system needs to wrap to the start of memory.

The system needs to generate two signals, full and empty. If the FIFO is full, the system should not write or increment the write address. If the FIFO is empty, the system should not read or increment the read address. Every input and output signals is required to be viewed through the 7-segment LEDs, or regular LEDs.

**System Design:**

In order to implement the design, an understanding of the overall system was first established. The FIFO needed to be able to read and write to the memory of the Cyclone IV E board, and it needs to be able to generate write and read enable bits from the user’s interactions. The behavior of the memory unit for the read and write operations are below in figure 1.



*Figure 1 - Timing Diagram of Memory*

When the read enable bit (ren) is high, the system reads the 15-bit data saved to the memory block only when they are not empty. When the write enable bit (wen) is high, the system writes the 15-bit data inputted to the memory blocks for the read operations to output. The write enable bit cannot go high when the 8 memory blocks allocated to the system are full and unread. This avoids writing over data that needs to be outputted. To ensure that the write enable bit and the read enable bit are activated appropriately, a Moore Finite State Machine was used for the control logic. The Moore FSM style provided simplistic mapping of the operations, as shown below in figure 2.



*Figure 2 - FSM Diagram of the Control Unit*

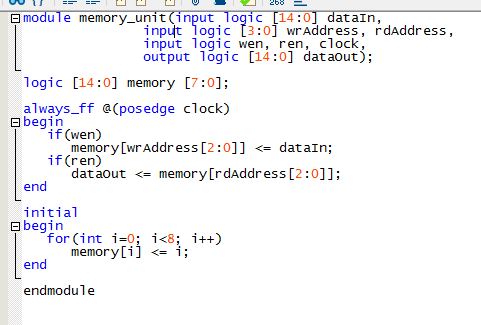
The FSM allows the wen and ren signals to only be active when the FIFO has the appropriate amount of memory available, and the user passes in the correct inputs. The Control Unit uses the asynchronous button signal with a synchronized module to determine when to perform an operation, and the wren signal is used to determine what operation to perform. When the wren signal is high the system attempts to perform a write operation, and when the wren signal is low the system attempts to perform a read operation. The full and empty signals stop the read and write operations from inappropriately using the memory blocks of the board. When the full signal is high, the write operation cannot be performed, and when the empty signal is high, the read operation cannot be performed. As a result, the wen signal is activated when the full signal is low, the button is high, and the wren signal is high. In addition, the ren signal is activated when the empty signal is low, the button is high, and the wren signal is low. The block diagram of the system is in figure 3 below.



*Figure 3 – block diagram*

The full and empty signals of the system are generated internally to the full\_empty module of the FIFO. In order to generate the signals, the full\_empty module uses the pointers rdAddress and wrAddress. RdAddress is the 4-bit pointer that passes into the memory module to find data to output, and the wrAddress is the 4-bit pointer that passes into the memory module to access blocks to write data too. Since there are 8 blocks of memory, only a 3-bit number is needed to access the memory array of data. The pointers use an extra wrap around bit in order to enable the empty and full signals for the system. The wrap around bit is the most significant bit of the pointers. When the first three bits of the rdAddress and the wrAddress are the same but the wrap around bits are different, the module pulls the full signal high. When the rdAddress and the wrAddress hold the exact same values, then the module pulls the empty signal high.

The memory module of the FIFO was implemented as a simple dual port, in order to read and write data simultaneously. When the wen signal is high, the memory module uses the first 3 bits of the wrAddress to access its memory array and writes the 15-bit dataIn value. When the ren signal is high, the memory module uses the first 3 bits of the rdAddress to access its memory address array passes the value into the dataOut for the display. The flip flop logic for the memory module is below in figure 4.



*Figure 4 – System Verilog Code Snippet for memory\_unit*

The memory\_unit of the module needs to initialize the blocks of the system with a certain value. The initial block allows each block in the memory array to be initialize with the index of the memory block. For example, the first memory block is initialized with the value of 0 in 15-bit binary, and the second memory block is initialized with the value of 1 in 15-bit binary. After the memory module finds the value to output with the rdAddress pointer, it passes the dataOut signal into the display module.

The display module of the FIFO converts the 15-bit binary dataOut signal, and converts it into hexadecimal values. After it creates four hexadecimal values, it uses the 7-segment module to create a 7-bit value that the LEDs can output.

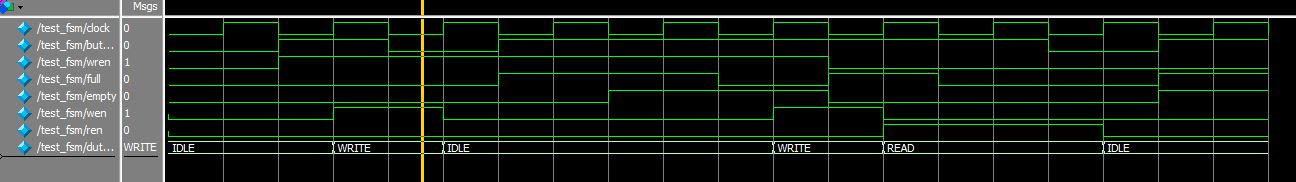
**Testing Approach:**

The system was tested using the ModelSim simulation software with test benches that verify the separate modules of the FIFO. Modules such as the display, 7-segment, and synchronizer have been tested in previous designs and did not require a test bench. The control\_fsm was verified with a test bench that tested all of the possible combinations for inputs and the related states that output. The full\_empty module, and control\_fsm module were then tested together to confirm the correct generation of the full and empty signals. The test bench verified that the read and write pointers were incrementing, and tested the empty and full signal with different scenarios. The full signal was tested by performing over 9 write operations, and the empty signal was tested by performing a read operation when there was no data left to read. The memory module was tested with cases that verified the spaces in memory were initialized by forcing the module to read without writing any data to memory. The memory module was then tested with cases that wrote data to memory, and then read the data for the output. The FIFO module’s test bench connects everything together, and ensures everything flowed properly throughout the system. The FIFO test bench used the same test cases used for the full\_empty module to verify the full and empty signals. The full and empty test cases also cover the testing for the read and write operations, because they show how the memory handles saving and outputting data. The FIFO module’s test bench also properly tests that the memory addresses increment properly by outputting the pointers on the waveform.

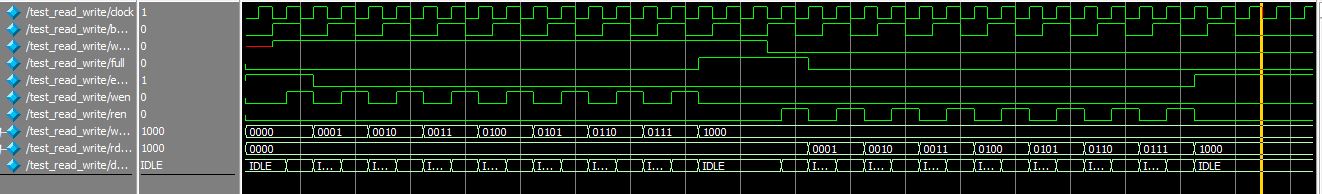
The Cyclone IV E board is used to further confirm the cases used in the test benches with the LEDs on the board. The full and empty signals are verified with the standard LEDs, and the 7-segment display verifies the output of the Read operation in hexadecimal.

**Results:**

The design of the memory FIFO combined all of the modules directly to each other to provide the desired results. The FIFO achieved the first in first out characteristics for the output of data in memory. The simulations generated by the test benches are in the figures 5-10 below.



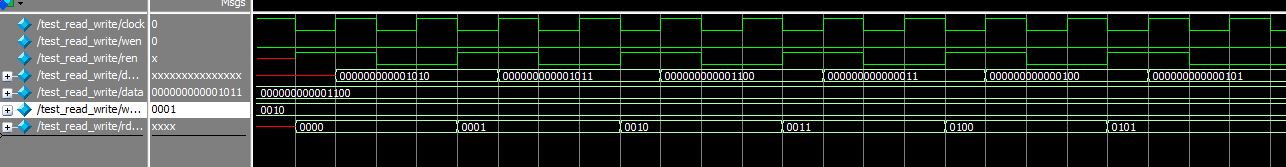
*Figure 5 – Control\_FSM simulation*

**

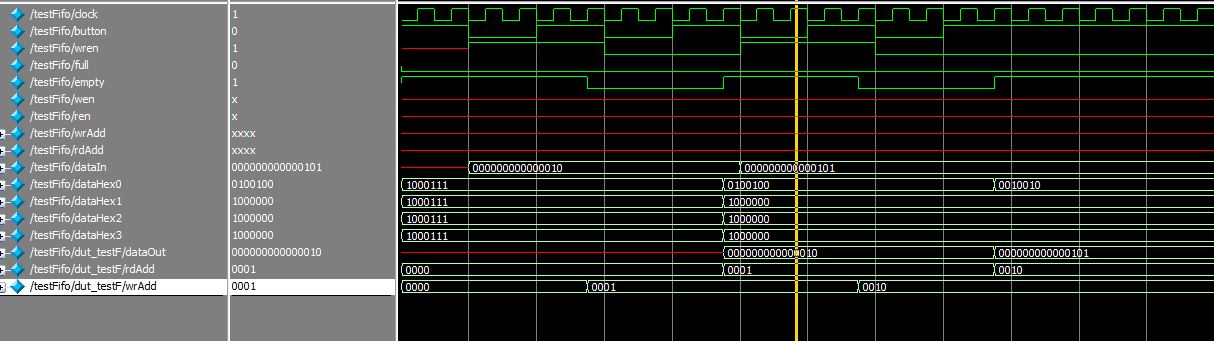
*Figure 6 – Full\_Empty Module with Control\_FSM read and write increments*

**

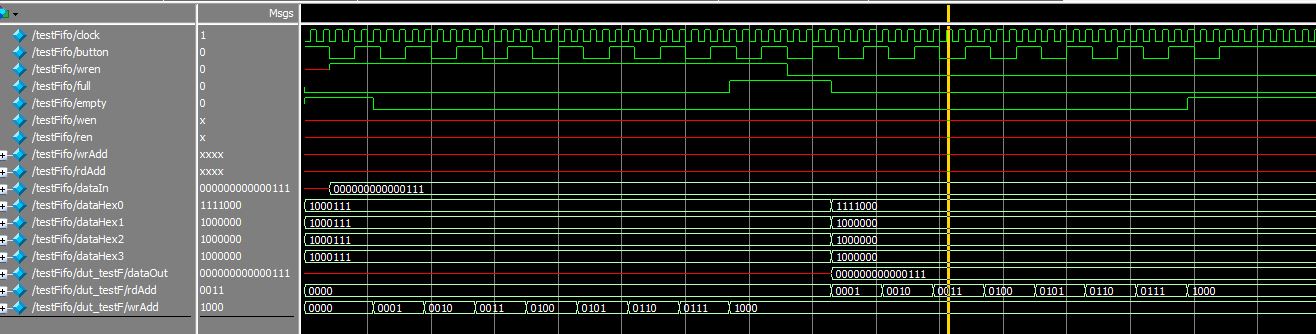
*Figure 7 – Memory\_Unit section part 1*

**

*Figure 8 – Memory\_Unit section part 2*

**

*Figure 9 –FIFO top level entity simulation part 1*

**

*Figure 10 – FIFO top level entity simulation part 2*

The simulations of the design were not successful for the first few test cases, due to the lack of understanding how test benches relate to the button signal’s input. The button signal in the simulations were not initially held high long enough for the modules. It was initially high for one negative clock edge to another negative clock edge in the system. This caused the output of the simulation to be inaccurate from how the system actually performs. Keeping in mind that the button input signal will be pressed with a high frequency and use a synchronizer module, the simulation test cases were recreated by holding the signal high for two clock cycles. This solved all the problems that were occurring with the test bench simulations in ModelSim.

All of the simulations were successful, and proved that the design would work appropriately on the Cyclone board. The control\_fsm was able to provide the proper wen and ren signals, and it was able to use the full and empty signals form the full\_empty module. The memory\_unit was capable of initializing the memory blocks in simulation, and was able to save and fetch values. The top-level FIFO module confirmed all the previous simulations, and was able to have to connect everything together.

The Cyclone board further confirmed the full functionality of the design. A green LED on the board was used to show when the memory blocks are full, and red LED was used to show when the memory blocks were empty. The 7-segments LEDs were used to display the hexadecimal value of data that was inputted into the system. By using a switch for the wren signal and a synchronized button, read and write operations to the board could be confirmed. The Cyclone board proved that the simulations accurately portrayed the behavior of the FIFO, and that the design was a success.

**Analysis:**

Overall, the design for the FIFO efficiently solved the problems presented for writing to and reading from memory. Using a simple dual port for the memory\_unit module allowed proper access to the memory blocks. The Control\_FSM module combined with the full\_empty module carried out most of the functionality of the design. When it comes to FIFOs knowing when the memory is full or empty is main issue. Having the read and write pointers add an extra wrap around bit, solved the potential flaw implementing a FIFO can bring. The system does not need improvement, because it solves this particular problem case. However, there may be some modifications that could be done to allow two devices with different clock speeds to communicate. The design that is currently being used assumes that the read and write pointers are coming from devices with the same clock signal (or in our example the same cyclone board). With some flip-flops and gray binary coding, the design could edited to synchronize the input of the pointers to allow communication between two devices with different clock frequencies.

A range of different memory elements were inputted to the memory\_unit module in order to understand the circuitry used. The Quartus software compilations were able to provide values of pins for the different ranges of elements. The table of results is in figure 11 below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Memory size |  | logic registers | | pins |  | memory bits |
| 4 |  | 75 |  | 41 |  | 0 |
| 8 |  | 135 |  | 41 |  | 0 |
| 16 |  | 0 |  | 41 |  | 120 |
| 24 |  | 0 |  | 41 |  | 120 |

*Figure 11 – testing memory unit table*

When using 8 elements or below, the System Verilog is compiled using only logic registers. When 16 elements or more are used for the memory array, the System Verilog is compiled using 120 memory bits and no logic registers. This is most likely because the system felt that it did not require the memory bits for lower amounts of elements. The memory bits seemed only to be utilized when too many logic registers were required to work on registers alone. The memory blocks’ connection to memory bits seem to fail only when the elements are around 8 or lower.

The main assumption that had to be made to complete the design was that the design was made for communication between two devices with the same clock signal. If this assumption could not be made, additional circuitry would have been added to synchronize the pointers. It was not difficult to connect the memory to the control unit, because testing was done prior to their connection. In future designs, testing will be implemented on the control logic and memory modules before connecting them together.