Eugene Ngo: 1965514

1/20/2023

EE 371 – Lab 2 Report

**Procedure**

The lab comprises of three tasks. The first task was to design a 32x4 single port RAM that can takes an input 5-bit address and has a 4-bit data port. Then a write control input, its address, data to write and data to read will be displayed on the HEX display in hexadecimal form. The second task was to design a 32x4 2-port RAM using the IP catalog in Quartus that uses one port supplying a 5-bit address for read operations and the other port is used for a 5-bit address for write operations. The read address should be internally updating by a designed counter on an approximately 1 Hz frequency, and all addresses, the 4-bit data to read and data to write, should be display on the HEX display in hexadecimal format. The third and last task was to design a FIFO that uses a 16x8 2 port RAM to store data, and display the “queue’s” full/empty status on LEDR9/8, and display read data, write data on the HEX display in hexadecimal form.

*Task 1*

By using built-in memory blocks, I designed a 32x4 single port ram that can takes inputs of a 5-bit address (SW[8:4]), a 4-bit data to write (SW[3:0]), a write control (SW[9]), and a user-operated clock (KEY[0]) and output a 4-bit data to read. The read/write address was displayed on HEX5 and HEX4, data to write is displayed on HEX2, and data to read is displayed on HEX0. These are all displayed in hexadecimal form. The RAM always reads the data in the specified address, but only when write control is enabled, the input data can be written into the address on the next self-operated positive clock edge.

*Task 2*

Task 2 built upon task 1 but is slightly different since task 1 uses 1 port, and task 2 uses 2. Using the IP catalog in Quartus Design, I designed a 32x4 2-port RAM as directed by the lab spec that is able to use one port supplying 5-bit address for read operations where the address is specified by a internal counter and scrolls through the memory locations at a rate of approximately 1 Hz and the other port is a 5-bit address controlled by SW[8:4] for write operations. It also takes in 4-bit data to write and a write control using KEY[3]. The write address is displayed on HEX5 and HEX4, the read address is displayed on HEX3 and HEX2, the data to write on HEX1 and the data to read on HEX0, all in hexadecimal form. So the read and write operations are completely separate: counter and read works on a set frequency independently and the write operation is only working when wr\_en, which is KEY[3], is pressed.

*Task 3*

For task 3, the final task, I designed a FIFO, using the given skeleton, that uses a 16x8 2 port RAM to store data, and operates like a ‘Queue;.’ It displays the “Queue” status, full or empty, on LEDR9 and LEDR8, respectively. When read is enabled, it reads the data that was first input and displays it on HEX1 and HEX0; it follows the rule: “first in, first out.” When write is enabled, it writes the data specified using SW[7:0] to the newest address and displays this value on HEX5 and HEX4. All HEX displays are in hexadecimal form.

**Results**

*Task 1*

A screenshot of a computer

Description automatically generated

Displayed above is the DE1\_SoC testbench which shows the data input being displayed on HEX2 and the specified address on HEX4 and HEX5. HEX1 and HEX3 are off. Then when written, using SW 9, and then read, the read data is displayed on HEX0.

A picture containing graphical user interface

Description automatically generated

This RAM test bench shows the data, data\_in, being written in at a specified address addr\_in, and then shows the output in data\_out.Diagram

Description automatically generated

The seg7 testbench displays the HEX displays functioning properly to display data to the appropriate HEX displays.

*Task 2*

A picture containing graphical user interface

Description automatically generated

This testbench shows that the HEX displays are functioning properly as the input data is being specified using SW 3-0 and shown on HEX1 and the address is specified using SW 8-4 and shown on HEX5-4. The data is then written using KEY3 and displayed on HEX0.

Graphical user interface

Description automatically generated

The counter testbench shows that the 5-bit addresses are being scrolled though incrementally.Diagram

Description automatically generated

The seg7 testbench is testing random inputs of addr\_r, addr\_w, data\_in, and data\_out, and is displaying is on the appropriate HEX displays as seen above.

*Task 3*

A picture containing schematic

Description automatically generated

The DE1\_SoC testbench writes in a bunch of data and fills the queue and once the queue is full, LEDR9 turns on to indicate such. Then excessive inputs is tested which will not be added to the queue as shown above and then the queue is emptied by reading out the entire queue in the order that it was entered in as displayed in the HEX displays. Once the queue is emptied, LEDR8 lights up to signal that the queue is now empty. Excessive reading is tested but once the queue is empty, no more data is read. Simultaneous reading and writing is tested at the end.Graphical user interface

Description automatically generated

FIFO tests to ensure that data is stored in order and once the input bus is full, no more data is stored. Then it reads the entire queue and attempts to read excessively. The testbench also attempts a simultaneous read and write operation.Graphical user interface, timeline

Description automatically generated

The FIFO\_Control testbench ensures that this module is sending the correct signals out to the FIFO module such a when the queue is full or empty.

A picture containing shape

Description automatically generated

The input process testbench shows that the input process module is controlling user inputs correctly and processing that input uniformly 2 clock cycles after the initial button press.Diagram

Description automatically generated

Seg7 demonstrates random inputs being displayed properly.

*Overview*

Overall, the lab taught me how to design RAM, reading and writing specified data to specified addresses, on multiple ports. The spec was easy to follow, and while long, was thorough in teaching us how to use the IP catalog to make the RAM.

Overall, the systems work as expected.

**Appendix**

\*See following code\*