**ECE 6213 – Design of VLSI Circuits**

**Fall 2022 – The George Washington University – Dr. Jerry Wu**

**Osama Yousuf – HW 1**

**Problem 1 – 40 points – Design a testbench**

* **Ch-5 – Problem 13: Verify the functionality of *Universal\_Shift\_Reg* in Example 5.45.**

Verilog code: *universal\_shift\_reg.v*

Testbench: *universal\_shift\_reg\_tb.v*

Output waveform from my testbench:

A screenshot of a computer

Description automatically generated with medium confidence

Corresponding waveform from the book – the input and output waveforms align one-to-one.

A sheet of music

Description automatically generated with low confidence

The waveform vcd file is provided: *universal\_shift\_reg\_vcd.vcd*

* **Ch5 - Problem 14: Write a Verilog description of the circuit shown in Figure P5-15 and verify that the circuit's output, , is asserted if successive samples of have an odd number of Is.**

Verilog code: *d\_flipflop.v, p5-15.v*

Testbench: *p5-15\_tb.v*

My testbench tests for the following cases:

* When successful samples of have an even number of 1s, the output remains 0
* When successful samples of have an odd number of 1s, the output remains 1
* When is maintained at 1, alternates between 1 and 0 as the count of 1s in alternates between odd and even, respectively.
* The reset pin resets the count of the module correctly, and the clk pin of the flip flop operates correctly.

Waveform:

Graphical user interface, application

Description automatically generated

The waveform vcd file is provided: *p5\_15\_vcd.vcd*

**Problem 2 – 40 points – 8 bit counter**

* **Ch-5 – Problem 15: Develop and verify a Verilog model of a 4-bit binary synchronous counter with the following specifications: negative edge-triggered synchronization, synchronous load and reset, parallel load of data, active-low enabled counting.**

Verilog code: *counter\_4bit.v*

Testbench: *counter\_4bit\_tb.v*

My testbench tests for the following cases:

* Reset pin works correctly, output is unknown when the counter has not been reset
* Counting is done synchronously with the clock pin, on negative edge
* The counter up-counts only when the enable pin is 0, i.e. that the counter is active-low enabled
* When the load pin is 1, the counter synchronously loads in parallel whatever 4-bit input is present on its data in port (D in my results)

Text format output:

*A picture containing table

Description automatically generated*

Waveform level verification:

Graphical user interface

Description automatically generated

The full output text file is provided: *counter\_4bit\_output.txt*

The waveform vcd file is provided: *counter\_4bit\_vcd.vcd*

* **Ch-5 – Problem 16: Modify the counter of the previous problem to have an additional output (ripple carry output [RCO]) that asserts while the counter is at 1111)2. Cascade two such counters and demonstrate tbat the unit now works as an 8-bit counter.**

Verilog code:

* + *counter\_4bit\_rco.v* – This is a modified version of my implementation with *counter\_4bit.v*. The load and reset are asynchronous for simplicity.
  + *counter\_8bit.v –* This is the 8-bit counter. It instantiates two 4 bit rco counters, and the cascading is done such that the rco output of the first counter acts as the clock of the second counter.

Testbench:

* + *counter\_8bit\_tb.v*

My testbench tests for the following cases:

* Reset pin works correctly, output is unknown when the counter has not been reset
* 8-bit counting is done synchronously with the clock pin, on negative edge
* The counter up-counts only when the enable pin is 0, i.e. that the counter is active-low enabled
* When the load pin is 1, the counter asynchronously loads in parallel whatever 8-bit input is present on its data in port (D in my results)

Text format output:

Table

Description automatically generated with medium confidence

The complete text format output is not shown, but the following highlights the scenario when the rco of the first 4-bit counter is 1, and triggers the negative edge of the second counter (indicated by the black arrow):

A picture containing table

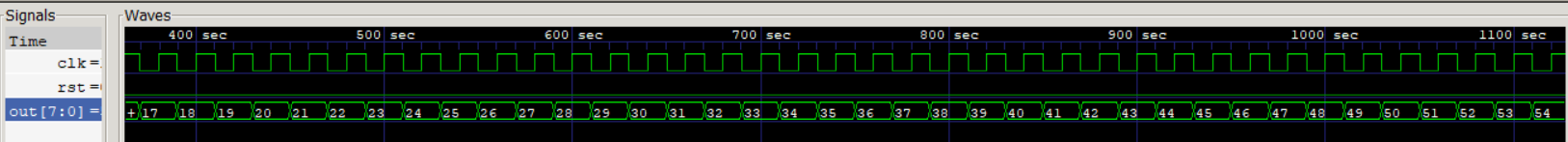
Description automatically generated

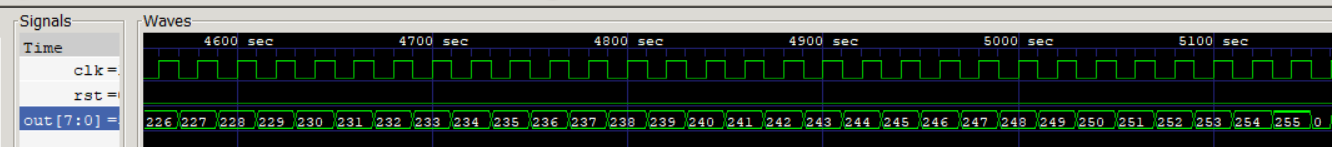
Waveform output, focusing on that all 8-bits are working:

The total digit that the counter can represent is 28 – 1 = 255.

The following snippets show the counting operation works as intended for all 8 bits. The output cycles back to 0 after the maximum possible digit – 255 – has been counted.

Note: Waveforms are in decimal format.





The following snippet shows that the counter can asynchronously load input data directly, and resume counting from the loaded data. It also shows that the reset pin can reset the counter from its initial state i.e. 0.

A screenshot of a computer

Description automatically generated

The full output text file is provided: *counter\_8bit\_output.txt*

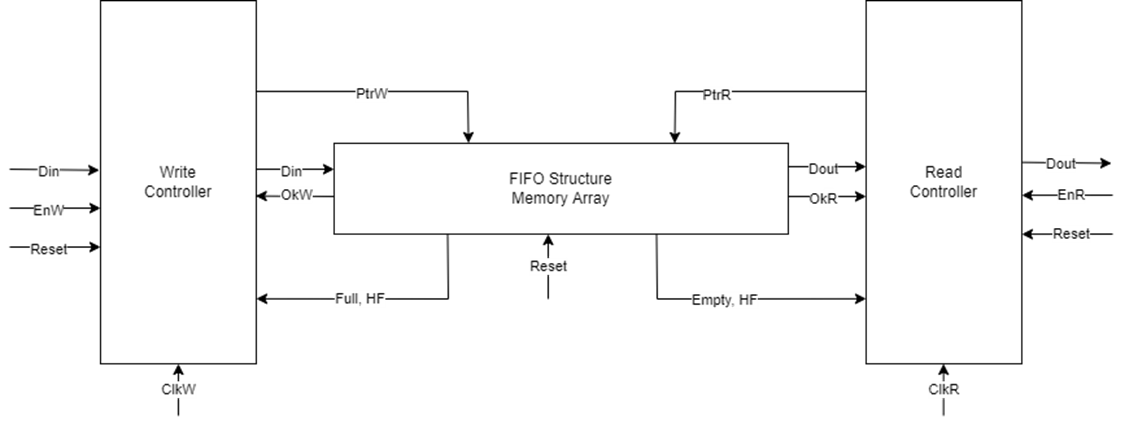
The waveform vcd file is provided: *counter\_8bit\_vcd.vcd*

**Problem 3 – 20 points – Block Diagram and testing plan for FIFO structure**

The following I/O pins will be present in my FIFO structure:

|  |  |
| --- | --- |
| **Pin** | **Description** |
| Din | This will be the input data from the writer to be pushed into the FIFO structure |
| Dout | This will be the port for outputting data to the reader, to be popped from the FIFO structure |
| ClkR, ClkW | These will be the individual clocks for the reader and writer respectively |
| EnR, EnW | This will be the enable pins for the reader and writer respectively |
| Full | This will indicate whether the FIFO structure is full or not |
| Empty | This will indicate whether the FIFO structure is empty or not |
| Reset | This will be an asynchronous reset which will reset the state of the FIFO structure, reader, and writer |
| OkR, OkW | These will be pins in the FIFO structure indicating whether or not there’s room to read and write respectively |
| PtrR, PtrW | These will be pins in the FIFO structure pointing to the memory address within the FIFO block from where reading or writing would be done |
| HF | This will indicate whether or not the FIFO structure is at least half full |

The following block diagram captures the overall module:



|  |  |
| --- | --- |
| **Block** | **Function** |
| Write Controller | The write controller will contain the logical blocks for writing data inside the FIFO structure. |
| Read Controller | The read controller will contain the logical blocks for reading data from the FIFO structure. |
| FIFO Array | This will be the main FIFO array that will implement the queue. It will communicate with the write and read controllers. The write controller would write data into the FIFO array, and the read controller would read data from the FIFO array. |

Testing plans:

|  |  |  |
| --- | --- | --- |
| **Test Case #** | **Name** | **Description** |
| 1 | Write until full | The full output of FIFO array should be asserted only when the array has been fully filled as a result of successive writes by the writer |
| 2 | Half Full | The Half Full output of FIFO array should be asserted only when the array has been at least half filled |
| 3 | Read until empty | The empty output of the FIFO array should be asserted when the array is empty as a result of either no data being filled, or the data being completely consumed by the reader |
| 4 | Empty Read | When read controller attempts to read an empty FIFO array, no data should be read |
| 5 | Full Write | When write controller attempts to write to a full FIFO array, the array should remain unchanged, and no data should be written |
| 6 | Write, Read Sequence Check | Successive samples of data should be written to the FIFO array by the write controller, followed by a sequence of reads until the array is empty. The test would check that the reader reads all the data in the correct sequence (first in, first out) |
| 7 | Write Controller Clock, Enable | The writing should only update the FIFO array in synchronization with its clock – ClkW, when it’s enabled – EnW, and when there’s data to be written – OkW. |
| 8 | Read Controller Clock, Enable | The Read Controller should only update the FIFO array in synchronization with its clock – ClkR, when it’s enabled – EnR, and when there’s data to be read – OkR. |