Justin Liang

ECE 154A

Wang

Due: October 16, 2013

**Homework #1: Introduction to Verilog**

**Part 0**

For this homework, I have chosen ModelSim PE Student Edition 10.2c. Putting together a design using this software was fairly straight forward.

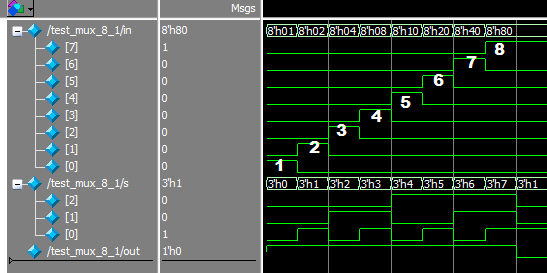
First, I created a new project by going to file -> new -> project. Once created, I am able to add new Verilog source files into the project by going to the project tab on the right, right click, and add to project a new file or existing file. There, I was able to edit my Verilog code in ModelSim. After I was done with my code, I went to compile -> compile all. The compiler automatically compiles the code for you and displays any errors in your code. Once compiled, I simulated the Verilog source file by using a separate Verilog test bench file. To do this, I went to simulate -> start simulation -> choose the test bench file under work -> press ok. To add a waveform, I went to add -> to waveform -> all objects in region. This allowed me to analyze the waveforms for each input and output of the Verilog source file using the pre-set values that I wrote in the test bench files. From here, I was able to confirm if my code was working properly by analyzing each specific set of inputs and determining if it was the correct set of outputs.

**Part 1 & 2**

A:

In this part of the homework, I designed and simulated an 8 to 1 mux. An 8 to 1 mux has 8 inputs, 3 select values, and 1 output. The combination of select values determines which input is passed to the output. For example, a combination of ~s2, s1, and ~s0 (010) allows for whatever value is in input2 to be passed onto the output. A combination of s2, s1, ~s0 (110) allows for whatever value is in input6 to be passed onto the output. Below is the waveform and corresponding truth table for verification.

|  |  |  |  |
| --- | --- | --- | --- |
| S2 | S1 | S0 | OUT |
| 0 | 0 | 0 | Value of in[0] |
| 0 | 0 | 1 | Value of in[1] |
| 0 | 1 | 0 | Value of in[2] |
| 0 | 1 | 1 | Value of in[3] |
| 1 | 0 | 0 | Value of in[4] |
| 1 | 0 | 1 | Value of in[5] |
| 1 | 1 | 0 | Value of in[6] |
| 1 | 1 | 1 | Value of in[0] |

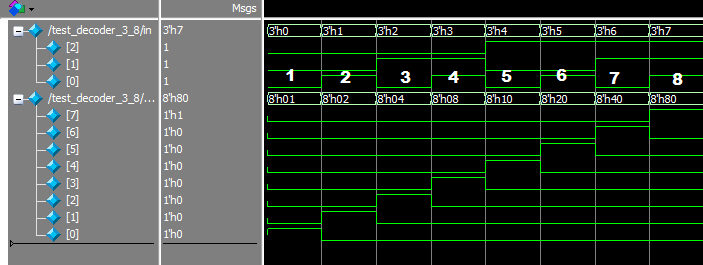


In this waveform, we can clearly see that it produces the right result. In my test bench, I applied the same technique that you did in your example test bench. I individually set an input to 1, while setting all the other inputs to 0. After that, I set the s value to select each input once. An output of 1 for each selection means that the Verilog code is working properly. For example, in the section marked #4, I set in[3] to be equal to 1, while all the other inputs are equal to 0. Having select values of s2 = 0, s1 = 1, s0 = 1, it should correctly have an output value of 1 if it selects in[3], which it does. The output is a constant 1, which means that the code is working properly.

B:

In this part of the homework, I designed and simulated a 3 to 8 decoder. A 3 to 8 decoder has 3 inputs and 8 outputs. The combination of inputs corresponds to which output has a value of 1, while the other outputs have a value of 0. For example an input of 011 means that the 4th LSB of the output is 1, while all the other outputs are 0. Below is the waveform and corresponding truth table for verification.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| In[2] | In[1] | In[0] | Out[7] | Out[6] | Out[5] | Out[4] | Out[3] | Out[2] | Out[1] | Out[0] |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

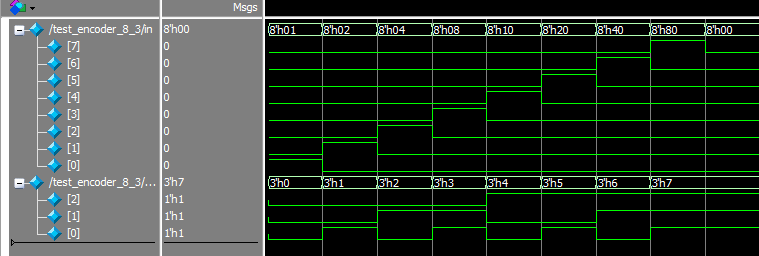


In this waveform, you can see that it produces the right result for a 3 to 8 decoder. In my test bench, I set each input to every combination allowed by 3 bits. Then, I observe the output to verify if my Verilog code is working properly, which it does. For example, in area 5 of the waveform, we can observe that the waveform has an input of 100. By verifying with the truth table, an input of 100 has 5th LSB to be 1 which is out[4].

C:

In this part of the homework, I designed and simulated an 8 to 3 encoder. An 8 to 3 encoder has the reverse characteristics of a 3 to 8 decoder, meaning that it has 8 inputs and 3 outputs. The input has a specific bit that is set to 1, while the other bits are set to 0. By doing this, the encoder outputs the (nth LSB) - 1 in binary. Below is the waveform and corresponding truth table for verification.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| In[7] | In [6] | In [5] | In [4] | In [3] | In [2] | In [1] | In [0] | Out[2] | Out[1] | Out[0] |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

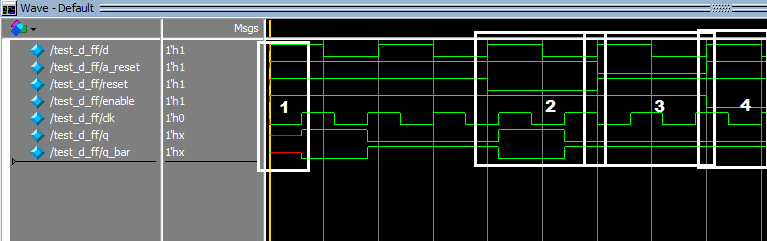


We can observe quickly that the truth table and waveform for a 3 to 8 encoder is the exact opposite of an 8 to 3 encoder. Because of this, we can quickly verify that the Verilog code for encoder is indeed working properly. The outputs of the encoder are the inputs of the decoder, and vice versa.

D:

In this part of the homework, I designed and simulated a D flip-flop with a synchronous reset and asynchronous reset and also an enable input. In this particular D flip-flop, there are 4 inputs and 2 outputs. 1 of the input is d, which is the data value. The way a D flip-flop works is that with each positive edge of the clock (assuming it’s a positive edge design), the Q output becomes whatever the value is at the D input. At any other time, the output Q does not change. The synchronous reset input will reset the value of Q to 0 at the next falling clock edge. The asynchronous reset input will reset the value of Q no matter what value the clock is. The enable input will enable the D flip-flop. The output Q\_Bar is the opposite value of Q. This is an active low design, meaning that if the signals are low, then the flip-flop will reset to 0. Below is the waveform and corresponding truth table for verification.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Inputs | | | | | Outputs | |
| Clk | D | Reset | A\_Reset | Enable | Q next | Q\_bar next |
| Rising Edge | X | 1 | 1 | 1 | D | ~D |
| Non-Rising Edge | X | 1 | 1 | 1 | Q | Q\_Bar |
| X | X | 0 | 1 | 1 | 0 | 1 |
| X | X | 1 | 0 | 1 | 0 (On next falling edge) | 1 (On next falling edge) |
| X | X | X | X | 0 | 0 | 0 |

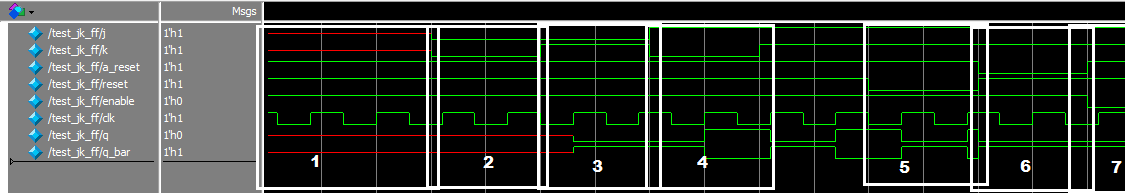


In this waveform, there are many things happening. First of all, the clock is alternated every 30 time units while the data is alternated every 50 time units. This is because I was verifying to see if the asynchronous reset was working properly in that it only resets In area 1, we have arbitrary values of d, a\_reset, reset, and enable but we see that the value of q and q\_bar is red. This is correct because the positive edge of the clock has not happened yet, so q­next and q\_barnext cannot determine the previous vales of q and q\_bar since it does not exist. Between areas of 1 and 2, we can see that the d flip-flop is functioning properly. The first output of q = d is only when the clock is at a positive edge. When the clock is at 0, the output q holds the previous value as seen in the second time frame. In area 2, we have values of a\_reset = 1, reset = 0, and enable = 1. We can observe that the output q does not reset to 0 until the next rising edge of the clock, which means that reset is working properly. In area 3, a\_reset = 0, reset = 1, and enable = 1. We can observe that the output q remains at 0 which means it is working properly. Lastly, in area 4, a\_reset = 1, reset = 1, and enable = 0. We can observe that the output q remains at 0, which means enable is working properly.

E:

In the last part of the homework, I designed and simulated a JK flip flop. The characteristics of a JK flip flop resembles the D flip flop. However, instead of a single D input, the JK flip flop has 2 inputs: J and K. When J = 0 and K = 0, the JK flip flop will hold the current state. When J = 0 and K = 1, this resets the flip flop so that Q is 0. When J = 1 and K = 0, the flip flop will set the value of Q to 1. When J = 1 and K = 1, the JK flip flop toggles the flip flop. This means that Q becomes Q\_bar and Q\_bar becomes Q. Because of these similarities, I constructed a JK flip-flop using a D flip-flop by using the JK flip-flop’s Boolean equation: Qnext = J\*Q\_bar + ~K\*Q to replace D. Below is the following truth table for a JK flip flop, omitting the reset, a\_reset, and enable inputs because it is the same as a D flip-flop.

|  |  |  |  |
| --- | --- | --- | --- |
| J | K | Q next | Q\_Bar next |
| 0 | 0 | Q | ~Q |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | ~Q | Q |



In area 1 of this waveform, J and K are set to 1 which means that Q is undefined since there is nothing to toggle from the previous state. In area 2 of this waveform, J and K are set to 0, which means that Q is undefined as well since there was nothing to hold from the previous state. In area 3 of the waveform, J is set to 0 and K is set to 1. This resets Q to 0, as shown in the waveform. However, it doesn’t take into effect until the next rising edge of the clock. In area 4, J is set to 1 and K is set to 0, which sets Q to 1. As before, Q is not set to 1 until the next rising edge of the clock. Between the areas of 4 and 5, J is set to 1 and K is set to 1, which toggles Q to become Q\_bar and toggles Q\_bar to become Q. This does not take place until the next rising edge of the clock, as shown on the diagram. In areas 5, 6, and 7, the same outcomes happen as areas 2, 3, and 4 respectively from the D flip-flop’s waveform. Having an active low signal on reset, a\_reset, and enable all make Q set to 0, with the exception of reset, which resets only on the next positive edge clock.