ECE 3544: Digital Design I

Project 1 (Part A): Introduction to the ModelSim Environment

Student Name: Sefunmi Ashiru______________________________

Honor Code Pledge: I have neither given nor received unauthorized assistance on this assignment.

Sefunmi Ashiru______________________________

Grading: The design project will be graded on a 100 point basis, as shown below:

Manner of Presentation (25 points)

__________ Completed cover sheet included with report (5 points)

__________ Organization: Clear, concise presentation of content; Use of appropriate, well-organized sections (10 points)

__________ Mechanics: Spelling and grammar (10 points)

Technical Merit (75 points)

__________ General discussion: *Did you describe the objectives in your own words? Did you address the questions posed in the project specification?* (10 points)

__________ Design discussion: *What was your approach to deriving the circuit you had to design?* (10 points)

__________ Testing discussion: *What was your approach to formulating your test benches, and how did you verify the correctness of your design and implementation?* (10 points)

__________ Conclusions: *Did you discuss the lessons you learned from the assignment? Did you discuss the applicability of your design process to larger and more general designs. Did you assess your implementation?* (10 points)

__________ Supporting figures (15 points total)

- *Correct waveforms showing simulation of both decoder modules.* (5 points)
- *Waveforms showing correct operation of your design.* (10 points)

__________ Verification: *Do the submitted files produce the correct response?* (20 points)

__________ **Project Grade**

Project Summary

In this project, we will learn how to use Modalism to run simulations. We must first simulate two pre-designed circuits. A 74138 3-to-8 decoder, which was written using a *structural* model using *primitive* gates, and a 4-to-16 decoder that uses two of the 3-to-8 decoders as *structural* blocks, along with an inverter as an additional *primitive* gate. We must use a test bench to simulate each decoder and produce output waveforms. We must then design a simple digital logic circuit that can be modelled and simulated using the Modalism tools. Through doing this we will be able to become familiar with the Modalism tools, experience with timing models and test benches for simulating digital circuits in Verilog.

Part 1 Test Bench Analysis

Both test benches apply different values to their input registers. The 74138 3-to-8 decoder testbench directly assigns its dec_en (This selects a specific enable condition for the model) and dec_in (This selects a specific input condition for the model). The input conditions run through all possible conditions for a 3-bit input (2^3 = 8) as does the input conditions for the 4-bit input (2^4 =16), dec_in, in the 4-to-16 decoder testbench. However, the a-to-16 decoder testbench assigns the value dynamically by using a for loop in Verilog.

Part 2 Design Decisions

When designing the comparator structural module for part 2 my first approach subtract the two 3 bit values and read the result and then assess whether it was greater than, less than, or equal to the value 'valA' is being compared to. While this would be a simple solution in software, it is very expensive when implemented in hardware as it requires: a 2 complement module, a 3 bit full adder and a set of logic gates to compare whether the output is negative (- = Most significant bit is 1), zero (all bits are 0), positive (+ = Most significant bit is 0), or a combination of these outputs.

My second approach was to then use logical analysis to see how these bits relate numerically. In the standard decimal system, we check the 100ths then 10th then 0th position in numbers to determine its magnitude and relation to another. So, by comparing the 4th, 2nd, 0 positions we can determine the same information in binary. By prioritizing certain events (such as comparing the most significant bits of the two values), we can access all cases and filter in order of the bits 3 properties: 4th,2th, 0th position. By using this logic, you can derive the values for A == B, A < B, A>B, however you only need 2 complete circuits for the comparison's needed to implement the 3rd. By negating the first 2 comparison statements you only require an extra inverter gate. Following this idea, the rest of the comparisons are combinations negations of the original two comparisons.

Position Priority based statements

A == B --- (A0'B0' + A0B0)(A1'B1' + A1B1)(A2'B2' + A2B2)

A < B --- A2'B2 + [(A2'B2' + A2B2) * A1'B1] + [(A2'B2' + A2B2) *[(A1'B' + A1B1) * A0'B0]

Combinations & Negations based statements

A > B --- (A>B + A==B)'

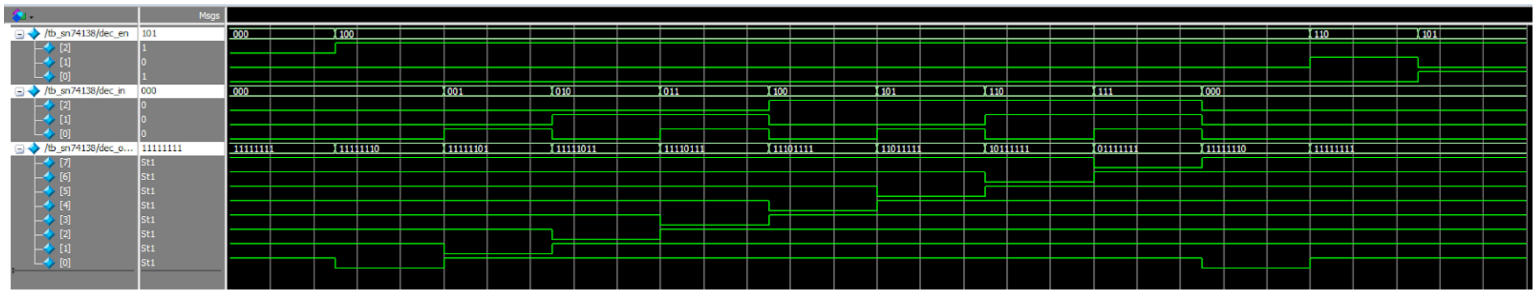
A != B --- (A == B)'

A <= B --- A < B + A == B

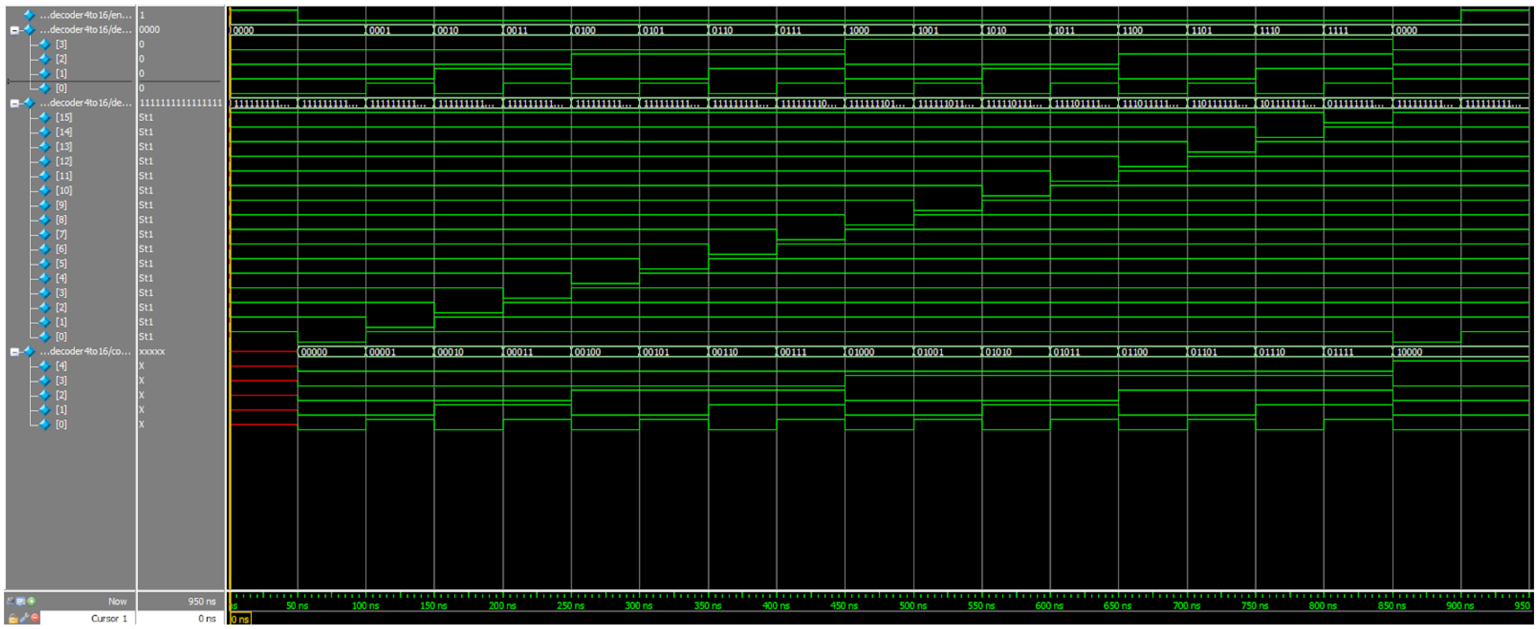
A >= B --- A > B + A == B

Test results

Graph results from 74138 decoder testbench module



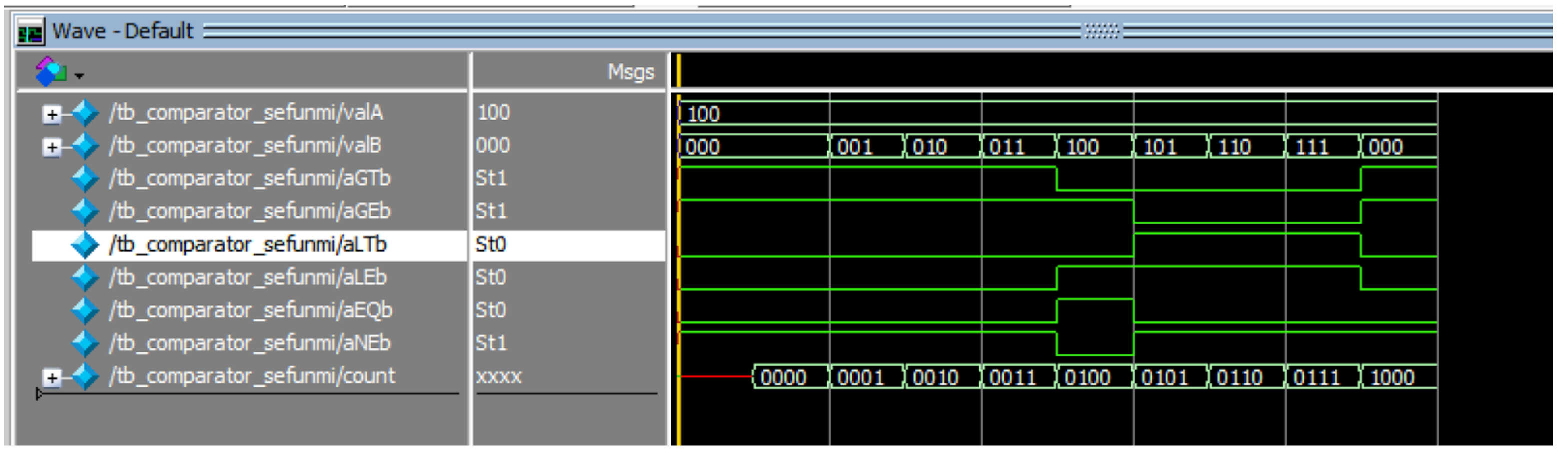
Graph results from 4-to-16 decoder testbench module



Graph results from comparator structural testbench module

Comparator structural module key:

- aGTb: 1 if A is greater than B, 0 otherwise.
- aGEb: 1 if A is greater than or equal to B, 0 otherwise.
- aLTb: 1 if A is less than B, 0 otherwise.
- aLEb: 1 if A is less than or equal to B, 0 otherwise.
- aEQb: 1 if A is equal to B, 0 otherwise.
- aNEb: 1 if A is not equal to B, 0 otherwise.


Wave - Default
Msgs
/tb_comparator_sefunmi/valA 100
/tb_comparator_sefunmi/valB 000
/tb_comparator_sefunmi/aGTb St1
/tb_comparator_sefunmi/aGEb St1
/tb_comparator_sefunmi/aLTb St0
/tb_comparator_sefunmi/aLEb St0
/tb_comparator_sefunmi/aEQb St0
/tb_comparator_sefunmi/aNEb St1
/tb_comparator_sefunmi/count xxxx
100
000 001 010 011 100 101 110 111 000
0000 0001 0010 0011 0100 0101 0110 0111 1000

Discuss how you formulated the tests contained in the test bench that you had to generate, and how you verified the correctness of your implementation.

When designing the testbench for the Comparator structural module I assigned the input registered for valA statically and set it to central value (4 =3’b100) to ensure that the results should be out but all cases have all base cases (>, <, and ==) output. ValB was assigned dynamically using a for loop to test all outputs for a 3-bit value. It should be noted that the counter had to have 4 bits so that it could reach a value greater than a 3-bit number and trigger the for loop to stop.

Conclusion

While I believe my design was effective for designing using logic gates in Verilog and should scale well. I believe that if wanted to optimize this design I would take advantage of the and-or-inv & or-and-inv gate combinations to design a custom transistor gate design combination in Verilog that requires 2*(input) for the transistors.