ECE 3544: Digital Design I

Project 2: Modeling the Timing of a Device

**Honor Code Requirements**

You must complete this assignment individually. You may not discuss or share any element or detail of a design or solution with any other student. Treat all such details as proprietary. In particular, *you may not discuss or share any source code with any other student*, and *you may not obtain any source code from any source other than your instructor, your course notes, or the textbook*. Some modules may be submitted to the MOSS service (theory.stanford.edu/~aiken/moss/) to check for plagiarism. Any copying flagged by MOSS will be treated as violations of the Virginia Tech Undergraduate Honor Code, and prosecuted as such.

**Objectives**

The purpose of this project is to use Verilog as a means for modelling the timing of a module. You will model an available component (the 74HC/HCT280 9-bit odd/even parity generator/checker) and use a delay model to analyze the timing of a system that utilizes the component.

**Instructions**

1. Study the clk and counter4bit modules. Simulate clk and counter4bit using the tb\_clk and tb\_counter modules.

I have provided Verilog models for a clock generator and a 4-bit synchronous counter. Study these two models to gain a good understanding of the behavior that they are meant to simulate. In particular, observe the manner in which each one uses parameters to establish default delays for their behavior in simulation. Please note that these modules have not been written with synthesizability in mind. In the particular case of the clock, the module cannot be used to generate a real clock signal, and we will not use it for that purpose. It is only meant to simulate the behavior of a clock in test benches.

I have also provided test benches for the clock generator and the counter. Study these test benches. Remember that test benches do not have their own ports. Instead, the test bench module generates stimulus values for the module you want to test. Use the test benches to simulate the clock and counter modules. Study the simulation results to further your understanding of how the clock generator and counter modules operate.

Among other things, the test benches will demonstrate the operation of the modules and the manner in which they should be connected within a system. The test benches also demonstrate the manner in which the designer can override a parameter when the module is instantiated. This technique is very useful, as it also works in situations where a synthesizable module is being instantiated in a higher-level module that we also intend to synthesize.

1. Use the 4-bit counter module as the basis for developing a 9-bit counter. Use the 9-bit counter’s test bench module as the basis for developing a test bench for your 9-bit counter.

Copy and modify the 4-bit counter module supplied with this project description. Name the new counter module counter9bit\_YOURPID. Your 9-bit counter should have the same delay parameters as the 4-bit counter. Even though we have not yet formally studied synchronous counters, you should be able to infer the proper structure for your 9-bit counter from the 4-bit counter that I have given you. Remember that in a synchronous sequential, your procedural model should use a non-blocking assignment to target the counter’s state.

Use the test bench for the 4-bit counter as a model for making a test bench for the 9-bit counter. Name the test bench tb\_counter9bit\_YOURPID. In your report, include waveforms that demonstrate the correct behavior of your counter. You do not have to include the entire sequence of 11-bit counter states, but you should provide representative sections.

1. Develop and verify an untimed Verilog model for the 74HC/HCT280 9-bit odd/even parity generator/checker. Develop a test bench for your circuit.

Page 4 of the datasheet presents a function table and a logic circuit that represent the behavior of the 74HC280. You are strongly encouraged to use a behavioral model. You may use Verilog’s built-in gate primitives to create a structural model, a behavioral model using continuous assignment, or a behavioral model using procedural assignment. (I strongly encourage you to use a behavioral model.) If there are discrepancies between the logic diagram and the function table, consider the function table to be correct.

Use the following module declaration for your circuit:

module hc280\_YOURPID(data\_in, even, odd);

input [8:0] data\_in;

output even, odd;

*In each case where the specification provides the module declaration for a component, you must use that module declaration exactly as it appears. Failure to use the module declarations as provided will prevent the assignment graders from verifying the operation of your modules. If this happens, you will receive no credit for the affected portions of the project.*

Write a test bench that uses the outputs of your 9-bit counter to drive the inputs of your parity circuit. Name your test bench tb\_hc280\_YOURPID. Your test bench should instantiate the clock generator, counter, and parity circuit models in a way that correctly connects their inputs and outputs together. Use the block diagram below as a model for structuring your test bench.

count[8:0]

enable

clear

clock

ctr\_enable

ctr\_clear

clk\_enable

**clk**

**counter9bit**

data\_in[8:0]

even

odd

**hc280**

9

en

out

Figure 1: HC280 test-bench structure

To simulate your hc280 module with the counter module, your hc280 module must have a `timescale directive before the module declaration. Use the same directive as the 4-bit counter example: `timescale 1 ns/100 ps. The timescale directive tells the simulator that the value of one unit of time (#1) is 1 ns, and the precision (the smallest valid fraction of a time unit) is 100 ps.

In your report, include *waveforms* of inputs to and outputs from the untimed hc280 module that confirms its consistency with the function table on Page 4 of the datasheet. You do not have to include all combinations of the inputs on your waveforms, but you must show that your module behaves correctly for each line of the function table.

*Do not proceed to Step 4 until you have verified the correct operation of the untimed model.*

1. Use a specify block to add propagation delays to your hc280 model.

Use the *typical propagation delays at 25 degrees C and 4.5V from page 5* of the 74HC280 datasheet. You do not have to worry about the output transition time, only the propagation delays.

Simulate the timed model using your test bench. Your report should include waveforms showing the correct operation of the model with delays. You should also include waveforms with sufficient resolution to show that the delays are correct.

1. Create a behavioral model of a 10-bit register.

Your register must have the following module declaration:

module register10bit\_YOURPID(clk, d\_in, q\_out);

input clk;

input [9:0] d\_in;

output [9:0] q\_out;

This register should parallel load the values of d\_in as the register state q\_out on each rising edge of clk. The functionality of this module is essentially that of the 74FCT821 component described in the 74821 data sheet, but without the ability to tri-state the outputs. Your model of the 10-bit register should not include any delays.

Even though we have not yet formally studied synchronous registers, you should be able to infer the proper structure for your 10-bit register from information I have provided in lecture. Remember that in a synchronous sequential, your procedural model should use a non-blocking assignment to target the register’s state.

Create a test bench to verify the correct operation of your 10-bit register. Name the test bench tb\_register10bit\_YOURPID. Your report should include a waveform showing its correct operation.

1. Use your models for the counter, the 74HC270, and the 10-bit register (74FCT821) to create the model of a digital system described below.



The channel is for 9 parallel data bits where the transmitter generates and sends an odd parity bit, and the receiver computes the odd parity of the data bits and compares the computed parity bit to the transmitted parity bit.

The schematic above describes two separate Verilog models.

transmit\_YOURPID(clk, enable, clear, data\_out);

input clk, enable, clear;

output [9:0] data\_out;

The transmit module should contain an instance of your counter\_9bit, hc280, and register\_10bit modules. These are labeled as U1, U2, and U3 respectively. The counter\_9bit should use the default propagation delay values that are in counter\_4bit.

receive\_YOURPID(clk, data\_in, data\_valid, data\_out);

input clk;

input [9:0] data\_in;

output data\_valid;

output [9:0] data\_out;

The receive module should contain an instance of your register\_10bit, hc20, and a parity bit comparator. These are labeled as U4, U5, and U6 respectively.

Finally, the overall schematic should be created in a tb\_channelX() module, which should instantiate and connect the transmit and receive modules. You will replace X with a number that denotes the version number of the test bench, as described below. The test bench should also contain an instance of the clk module supplied with this project. Use a single clock to source all of the clocked elements. The clock and clock connections are not shown in the schematic for clarity.

You should create two versions of tb\_channelX():

* The first version, tb\_channel1(), should define the PERIOD parameter for clk that is sufficiently large to respect the propagation delays of the system. This should cause the data\_valid signal shows the correct behavior at all times.
* The second version, tb\_channel2(), should define the PERIOD parameter for clk that is small enough for the data\_valid signal *not* to be asserted for some values of the receive module’s data\_out.

For both versions of the test-bench, the delays in your 9-bit counter should be the default delays in the 4-bit counter.

**Project Submission**

Your project submission should include the following items:

1. A project report containing the following elements:

* A restatement of the assignment’s objectives.
* A discussion of the approach you took to modeling your modules, and a description of any design decisions you made as a part of implementing your modules.
* Waveforms that demonstrate the correct operation of each of the modules you create. Discuss how you formulated the tests contained in your test benches and how you verified the correctness of your implementation.
* An analysis of how you determined the PERIOD parameter of the clk module to demonstrate valid and invalid behavior of the system implemented in tb\_channelX(). In particular, you should provide an analysis of the shortest clock period that will always allow correct (valid) behavior of the system.
* Waveforms that demonstrate valid and invalid behavior of the system.
* A discussion of your conclusions and the lessons you learned from the assignment.

Your report should be in PDF format. Include your PID in your report filename, *e.g.*, project2report\_jthweatt.pdf.

1. Source files for:
2. hc280 with delay implemented, and test bench
3. counter9bit, and test bench
4. register10bit, and test bench
5. transmit
6. receive
7. channel\_tb1
8. channel\_tb2

Remember to include your Virginia Tech PID in the names of your files. Submit your report and source as a single zip file on the project assignment page. Include your PID in the name of your archive file, *e.g.*, project2files\_jthweatt.zip.

Grading for your submission will be as described on the cover sheet included with this description. You must include the provided cover sheet as the first page of your report. I have provided it as a .doc so that you can make it the first page of your report prior to converting it into a PDF file.