ECE 3544 – Digital Design 1
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.

**Purpose**
The purpose of this project is to use Verilog as a means for modeling the timing of a module. You will model two TTL components (the SN74185 binary-to-BCD converter and the SN74184 BCD-to-binary converter) and use a delay model to analyze the timing of a system that utilizes the components.

**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.

2. *Use the 4-bit counter module as the basis for developing a 6-bit counter. Use the 4-bit counter's test bench module as the basis for developing a test bench for your 6-bit counter.*

Copy and modify the 4-bit counter module supplied with this project description. Name the new counter module `counter6bit_YOURPID`. Your 6-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 6-bit counter from the 4-bit counter that I have given you. Remember that in a synchronous sequential circuit, 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 6-bit counter. Name the test bench `tb_counter6bit_YOURPID`. In your report, include waveforms that demonstrate the correct behavior of your counter.

3. *Develop and verify untimed Verilog models for the SN74185 binary-to-BCD converter and the SN74184 BCD-to-binary converter described in the 74184/74185 Datasheet included with this specification. Develop test benches for your modules.*

Implement Verilog models having the same behavior as the function tables on Pages 3-732 (for the 74184) and 3-733 (for the 74185) of the datasheet. When reading the function tables and making your models, take note of the following details:

- For the 74184, use the table on the left (BCD-to-binary converter). When reading the table, note that the LSB of the BCD input is the LSB of the binary output. Thus, each row of the table corresponds to two input cases.
- We will be operating the 74185 in a system where the valid binary inputs range from decimal 0 to decimal 39, inclusive. In your implementation, you may therefore ignore outputs Y8, Y7, and Y6. As before, the LSB of the binary input is the LSB of the BCD output. Each row of the table corresponds to two input cases.

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.*

Use the following module declarations for your circuits:

```
module sn184_YOURPID(g_n, bcd_in, bin_out);
   input        g_n;
   input  [5:0] bcd_in;
   output [5:0] bin_out;

module sn185_YOURPID(g_n, bin_in, bcd_out);
   input        g_n;
   input  [5:0] bin_in;
   output [5:0] bcd_out;
```

*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 6-bit counter to drive the inputs of your converter circuits. Name your test bench `tb_converters_YOURPID`. Your test bench should instantiate the clock generator, counter, and converters in a way that correctly connects their inputs and outputs together. Use the block diagram below as a model for structuring your test bench.

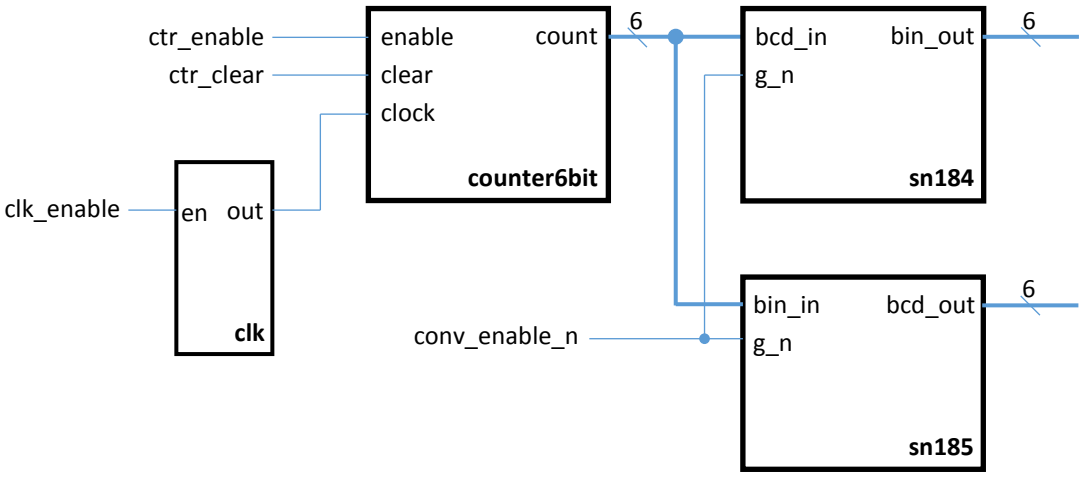


Figure 1: Converter test-bench structure

To simulate your converters with the counter module, the converter and counter modules 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.

Observe that in this test, the output of the counter represents a different value to each of the converters. For example, the counter output 6'b011001 represents the BCD input 19, which should cause the sn184 to output 6'b010011. The same counter output of 6'b011001 represents the decimal number 25, which should cause the sn185 to output 6'b100101. If you run the counter through all of its states, you will have applied all possible cases to both circuits. You can use the waveform to check the valid cases. However, not every input case is valid for either converter; your models should treat invalid inputs as don't cares as appropriate, and you may ignore these cases in your waveform output.

In your report, include *waveforms* of inputs to and outputs from the untimed converter modules that confirm their consistency with the function tables in the datasheet. The stimulus procedure in your test bench should operate the control inputs of the converters, the clock, and the counter in an appropriate order to demonstrate correct operation.

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

4. *Use a `specify` block to add propagation delays to your hc85 model.*

For your `specify` block, use the typical propagation delays from Page 3-734 of the datasheet. You do not have to worry about output transition times, only the propagation delays. Simulate the timed model using your test bench. Choose a convenient clock period that is longer than the converter delays. Review the clock test bench for an approach to parameterizing the clock period at instantiation.

Your report should include waveforms showing the correct operation of the model with delays. Your waveforms should have sufficient resolution to show that the delays are correct.

5. *Create a behavioral model of a **12-bit register**.*

Your register must have the following module declaration:

```
module register12bit_YOURPID(clk, d_in, q_out);
    input         clk;
    input  [11:0] d_in;
    output [11:0] 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 12-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 12-bit register from information I have provided in lecture. Remember that in a synchronous sequential circuit, 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 12-bit register. Name the test bench `tb_register12bit_YOURPID`. Your report should include a waveform showing its correct operation. You need not show all possible input cases, but your test bench and waveform should reasonably demonstrate that your register works correctly.

6. *Use your models for the clock generator, the counter, the converters, and the register to create the model of a digital system described below.*

The system consists of two blocks and a clock generator. You should instantiate these elements in a test bench module test bench module named `tb_systemX_YOURPID`. Replace X with a number that denotes the version number of the test bench, as described further below.

- In the `transmit` module, the counter provides its outputs as the inputs to a `sn185`. This represents a binary-to-BCD conversion. The 6-bit counter output and the 6-bit converter output are both registered.

  ```
  module transmit_YOURPID (clk, clear, ctr_en, conv_en_n, reg_out);
     input         clk, clear, ctr_en, conv_en_n;
     output [11:0] reg_out
  ```

- In the `receive` module, an input value representing the 6-bit counter value and the 6-bit BCD value are registered. The register output applies the 6-bit BCD value to a `sn184` to produce a 6-bit binary value. The 6-bit binary output from the register and the 6-bit binary output from the SN184 should be applied to a comparator. The comparator output `valid` should equal 1 if both 6-bit binary values are the same and 0 otherwise. The comparator should have no delay.

  ```
  module receive_YOURPID (clk, conv_en_n, reg_in, valid);
     input         clk, conv_en_n;
     input  [11:0] reg_in;
     output        valid;
  ```

The `transmit` and `receive` modules both use the same clock, as generated by the `clk` module. Use the counter and converters containing delays, as described respectively in Steps 2 and 4 of this procedure.

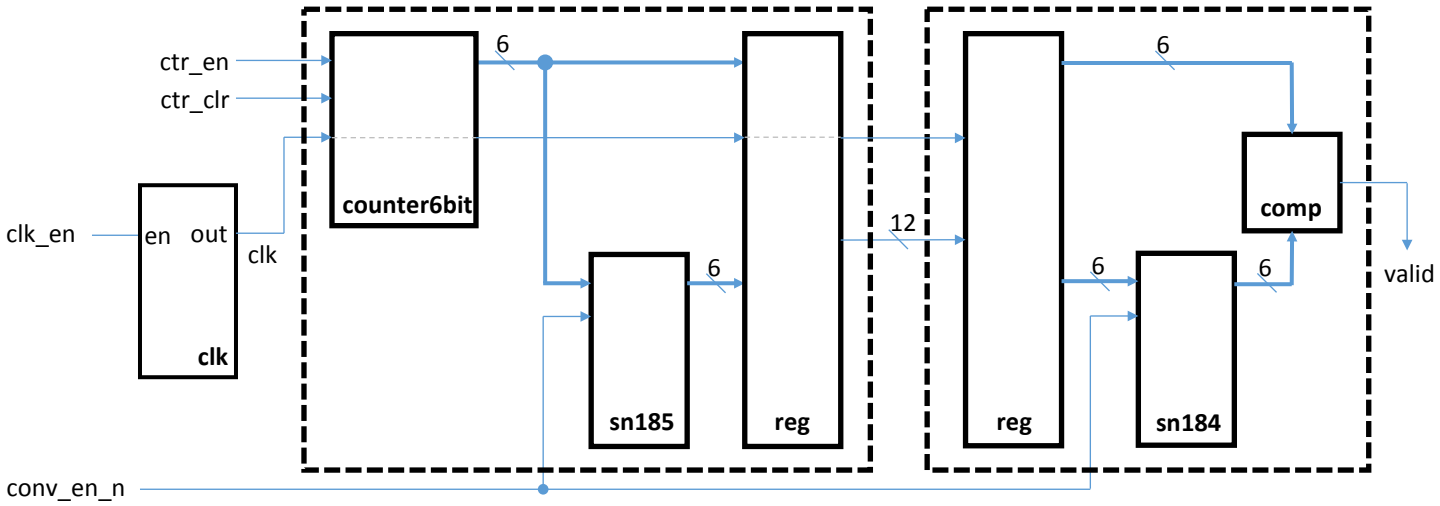


Figure 2: System test bench structure

The first version (`tb_system1_YOURPID`) should define a value for the `PERIOD` parameter of the `clk` module that is sufficiently large to respect the propagation delays of the modules in the system. This should allow you to demonstrate that the system shows the correct behavior at all times, *e.g.*, the correct values for the register output of the transmit module and a correct value for valid during each clock period.

The second version (`tb_system2_YOURPID`) should define a value for the `PERIOD` parameter of the `clk` module that is just small enough to cause the system to show incorrect behavior at some point for some aspect of the system – for example, the value of valid, or the values of the registered transmit values.

**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 the comparator and counter 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_systemX`. 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.

2. Source files and test benches, as follows:
   - `counter6bit_YOURPID`
   - `tb_counter6bit_YOURPID`
   - `converters_YOURPID` with delay implemented
   - `tb_converters_YOURPID`
   - `register12bit_YOURPID`
   - `tb_register12bit_YOURPID`
   - `transmit_YOURPID`
   - `receive_YOURPID`
   - `tb_system1_YOURPID`
   - `tb_system2_YOURPID`

Rplace instances of YOURPID in file names with your Virginia Tech PID. 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.

**Other Helpful Hints**

- It helps to improve the readability of your waveforms before taking screen shots of them. You will not receive credit for waveforms that are ambiguous in demonstrating the correct working of your models – this includes low-resolution waveforms and hard-to-read signal names.
  - Use the zoom buttons to adjust the time scale of the waveforms.
  - On the Waveform window, choose **Format** > **Toggle Leaf Names** to shorten the display names of your signals, then adjust the width of the portion of the window containing the signal names.
  - Change the radix of the value so that fewer digits appear in the waveform.
  - Re-arrange the order of your signals as needed, and consider adding dividers (Right-click in the window containing signal names, **Add** > **New Divider**; Double-click on a Divider to edit it after adding) to make your simulation easier to follow.
- Design units in this system have been purposefully written to run forever. You can run the simulation for a specified amount of time by entering an amount of time in the window to the left of the “Run” toolbar button and then pressing the “Run” button for as many times as you want to advance the simulation for that amount of time.