COMPUTER SYSTEMS DESIGN

LAB 2

John Gangemi
U 6871-4612
CDA4203L.001S15

Introduction

In Verilog, design a 7-bit programmable counter that is able to stop and hold its current state given a maximum count value as input. The programmable counter must not exceed a decimal range of 0-99, although the maximum count value can be greater than 99 decimal.

When a single-bit input 'run' is set to zero the counter will accept changes to its maximum count value, however when 'run' is one the counter will ignore any changes to the maximum count value.

A binary-to-binary coded decimal converter must take as input the output of a 7-bit counter and transform that binary value into two 4-bit outputs, digit_1 and digit_2, where digit_1 represents the ones digit and digit_2 represents the tens digit.

This lab is meant to serve as an introduction to sequential and combinational hardware design using Verilog, a hardware description language.

Design

Viewing the "Top-Level block diagram" for the circuit showed two inputs and two outputs, their names and bit-widths were given by the "Port Mapping" table.

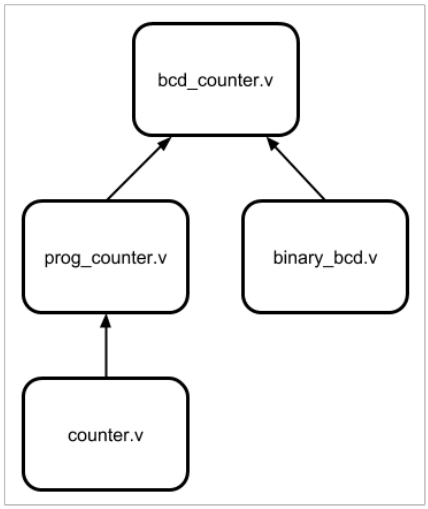
Port: max_count
- 7-bit input
- set maximum counter value

Port: run
- 1-bit input
- 0 = set, 1 = run

Port: digit_2
- 4-bit output
- tens digit BCD

Port: digit_1
- 4-bit output
- ones digit BCD

Following the design constraints specified above and in the introduction section, implementation of the overall hardware design in Verilog was accomplished through the use of provided templates (optional). The Verilog template files were separated according to a hierarchical order, this structure simplified the code needed to define the behavioral aspect of the entire design.



It should be noted that all arrows indicate those modules from which they stem are ‘instances’ in the module being referenced.

Binary-to-BCD module

The behavior of this module was implemented using the “Double Dabble” algorithm, something that was discovered last semester when working on the CMOS-VLSI final lab. Essentially this algorithm takes an n-bit number and performs a number of left-shifts, carrying the shifted-out msb to a “scratch” register, all while adding three when either the ones or tens column of the “scratch” register holds a value greater than 4.

For a 7-bit input the “Double Dabble” algorithm requires eight left shifts and necessary addition by 3 to fill a temporary large register with the correct two digit BCD representation of the 7-bit input.

To accomplish this task a 15-bit register was created to hold the 7-bit input, 4-bit ones digit, and 4-bit tens digit (order is lsb to msb). Using the repeat function in Verilog simplified the looping mechanism of the algorithm. Once the algorithm had completed the outputs were assigned to specific bit-width intervals of the 15-bit register in a non-blocking manner.

Counter module

Main functionality of the synchronous 7-bit counter is a synchronous reset and enable input. Behaviorally the reset takes precedence over the enable signal and therefore it is the first ‘test case’ statement for the ‘if-else’ conditional structure. Assuming a positive edge of the system clock then the function of the counter in terms of its inputs can be described as shown in the table below.

Enable	Reset	Function
0	0	hold current state
0	1	set counter to zero
1	0	increment counter by one
1	1	set counter to zero

Programmable Counter module
Unlike the other modules that only change its outputs on the 'positive edge' of the clock this module updates its state on the 'negative edge' of clock. The justification for this implementation is to give the outputs of the counter time to settle before evaluating the counter's current state. Also, the conditional statement for the logic of the programmability is easier to read.

A single-bit register is updated every 'negative edge' of the clock cycle and sent as the 'enable' input to the 7-bit counter instance. The register equals 1 only if the current state of the counter is less than the maximum count value **and** if the current state of the counter is less than 99 decimal.

BCD and Programmable Counter module
In this module the single-bit input 'run' determines whether or not to update a 7-bit register 'set' with the contents of the 7-bit input 'max_count'. Sending the current value of the 7-bit register 'set' as the 'max_count' input to the programmable counter instance ensures that when the value of 'run' is equal to one the maximum count value for the sub-module 'programmable counter' will remain constant.

Testing
Verilog test fixtures were created for all modules in the design to ensure correct functionality as this helps eliminate errors when debugging a large design. Each module included test vectors chosen to expose fundamental errors quickly, thus the reduced the amount of time needed to validate a module's implementation.

Binary-to-BCD module
A set of input values ranging from 0 - 100 were tested to check the conversion process for errors, either expected or unexpected. Test vectors 0, 99, and 100 were particularly important as vectors 0 and 99 are the end cases for the minimum, maximum allowable value. Also, test vector 100 is important as the resulting output should be an erroneous value as expected.

Counter module
Testing all possible cases of reset and enable is needed to verify that the synchronous reset does indeed take precedence over the enable input. While reset is low, toggling the enable from high to low test the counter's ability to hold its current state, likewise toggling from low to high test the ability to continue counting.

Programmable Counter module

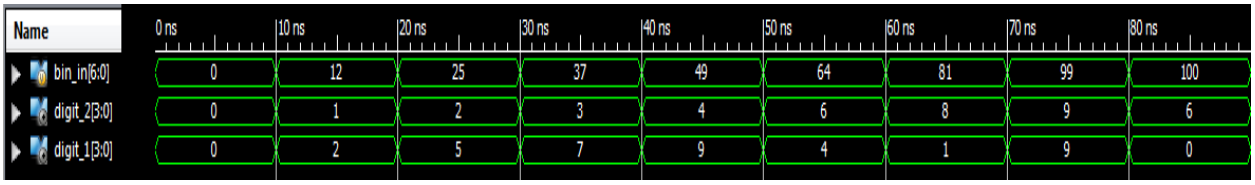
Validating the programmability is implemented correctly means checking the two conditions for which the counter is enabled. Therefore, testing involves using max_count input to count to a specific value and waiting a couple clock cycles longer than needed to see if the counter holds its state. More So, inputting a max_count value larger than 99 decimal should show the counter does not exceed 99 decimal.

BCD and Programmable Counter module

Overall design validation occurs after testing this module. Testing includes checking the module's output when 'run' equals 0 or 1. When 'run' is 0 the last value of max_count before 'run' is set to 1 should be the maximum value for which the counter is incremented. Changing max_count while 'run' equals 1 should not affect the counter's progression.

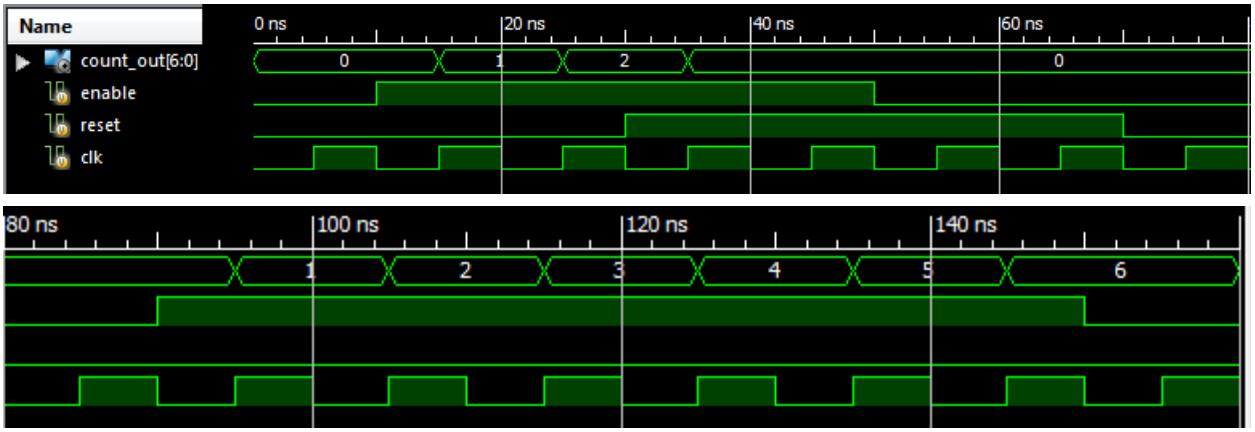
Results

Binary-to-BCD module



From 0ns to 80ns the binary input is converted correctly for digit_1 and digit_2. After 80ns the binary value 100 is not converted correctly as expected. Functionality of this module is complete and correct.

Counter module



When enable is 1 and reset is 0 the counter will increment appropriately, however when reset = 1 and enable is either 1 or 0 the counter will reset upon a positive edge of the clock. Also when both inputs are zero the counter will hold its current state, as seen most notably from 150ns to 160ns. Alas, the functionality of the module is complete and correct.

Programmable Counter module

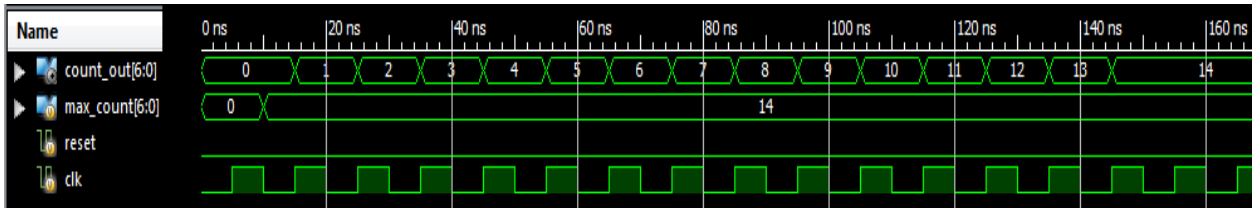


Image shows the counter reaching max_count and holding for 2 extra clock cycles as coded in the test bench with the command “#160”.

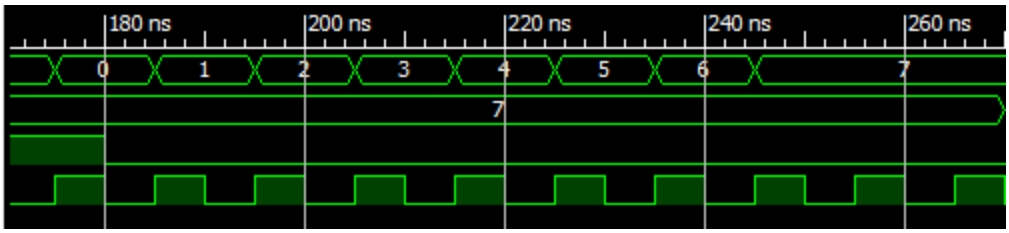
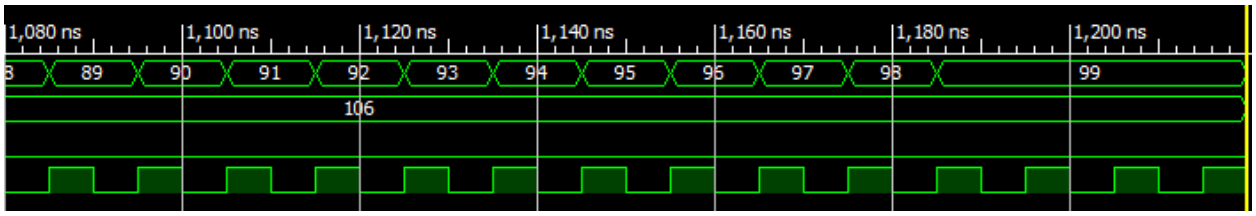


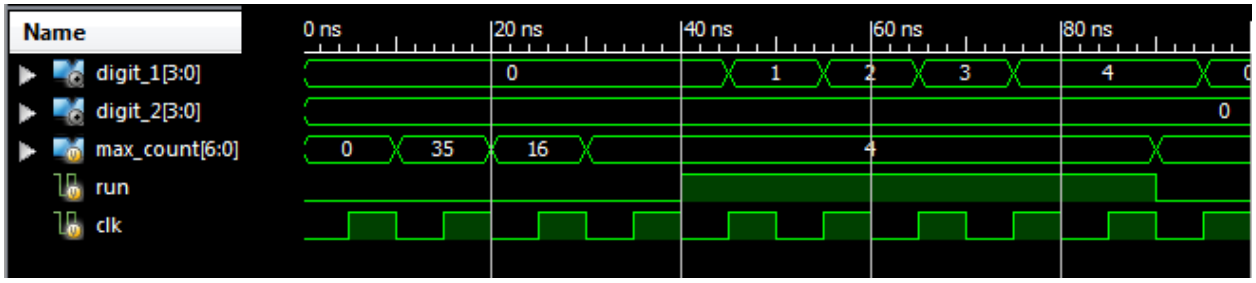
Image shows the counter resetting initially and counting till the specified max_count of 7, again the counter holds as the time directive is set to allow extra clock cycles so that the hold is distinguishable.



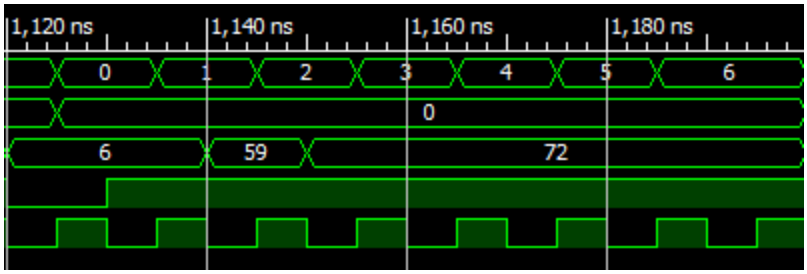
Given a max_count of 106 the counter only reaches a value of 99 decimal and holds that state until the simulation is over.

The timing diagrams confirm this module was implemented correctly.

BCD and Programmable Counter module



While 'run' is zero changes to max_count are updating a temporary register such that the last change to max_count before 'run' switches to one is the maximum value the counter will reach as shown above. Since the value of 4 for max_count was the last stored value for the counter's input the counter only ever reaches the value of 4, this is confirmed by adding extra clock cycles to see '4' holding.



The counter accepts the value of max_count equal to 6 when 'run' is zero and justly increments to the value 6 ignoring changes to max_count when 'run' is one.

Having reviewed all timing diagrams generated by test bench files associated with each module it can be confirmed that the overall hardware design of the circuit in Verilog is fully operational.

Conclusion

With the knowledge gained from the first lab this lab was rather simple but still very informative and beneficial for sequential designs in Verilog. Provided templates helped a lot with realizing how a number of modules can be arranged in a hierarchical order to create a "top" module that implements the behavior of all sub-modules and therefore the entire hardware design efficiently and intuitively.