Counters Driven by a Clock

Vincent Martin
TUID: 913012274
ECE 2613
Lab #: 6 (10/6/2012)

Introduction:
The objective of this lab is to create two counters, one that will count from 0 to 5 and a second one that will count from 0 to 9. Unlike our previous counter from lab 5 which relied on input switches to get the number to be incremented, these will use the clock to drive logic which will allow the module to increment itself.

<u>The Theory of our 4-bit binary coded decimal counters</u>

The main goal of our two counters is to increment a 4 bit number from 4'b0000 to 4'b1001 in the case of our 0-9 counter and from 4'b0000 to 4'b0101 in the case of our 0-5 counter.

Both of these counters start at 4'b0000 and are incremented by 1 on each occasion that the clock reaches it's positive edge. In addition, once the number being incremented reaches it's max, either 5 or 9, it will then loop back to start at 0 again. The module will also keep track of any appropriate carry out bits and set the carry_out output.

Also, the modules will have two other inputs, clr and ena. If clr = 1 and ena = 1 the output will be set to the counters maximum value. If clr = 1 and ena = 0 the output will be set to 0 and finally, if clr = 0 and ena = 1 then the counter will begin.

In our previous lab our counter/incrementer included only one always block. This always block implemented was of the combinational type and was driven by a series of switches.

The current modules work in a similar manner in their combinational blocks, however, this time they will be joined by a sequential always block. This sequential block will allow the combinational block to take its calculated incremented value Next_q and then use it as the input during each positive clock edge.

This explanation will be much more clear once the truth table and code is seen below.

<u>Applying the Theory to Verilog:</u>
In order to transfer our understanding of theory to verilog code we will have to understand the following two modules seen in figures 1 and 2.

Module Description for counter_0to9:

- Input
  - ena: 1 bit to enable counter
  - clr: 1 bit for clear.
  - clk: 1 bit input for clock signal.
- Output
  - q: 4-bit current number after incrementation.
  - carry_out: 1 bit carry bit.
- Internal Signal
  - Next_q: Incremented q.

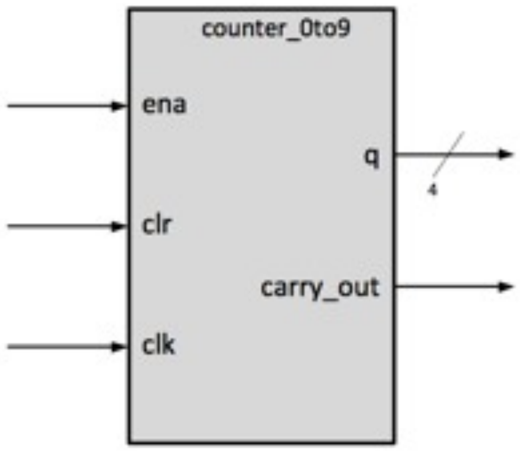


Figure 1: counter_0to9 module

Module Description for counter_0to5:

- Input
  - ena: 1 bit to enable counter
  - clr: 1 bit for clear.
  - clk: 1 bit input for clock signal.
- Output
  - q: 4-bit current number after incrementation.
  - carry_out: 1 bit carry bit.
- Internal Signal
  - Next_q: Incremented q.

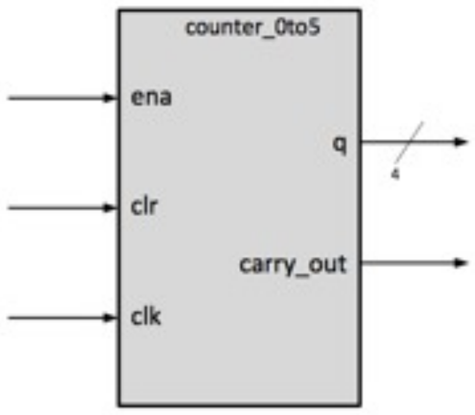
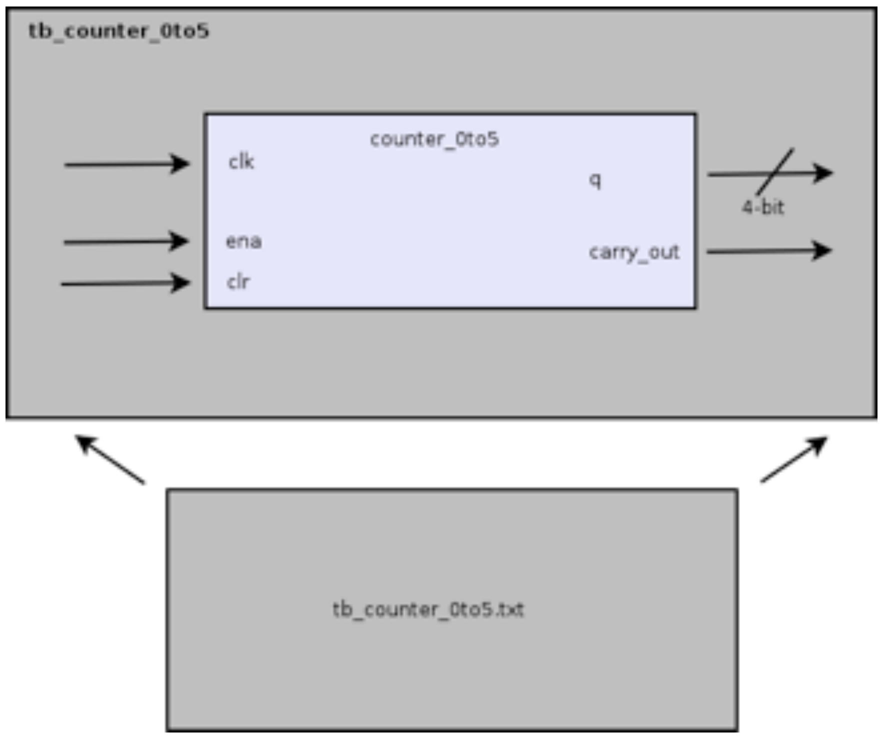


Figure 2: counter_0to5 module

Additionally we will want to implement two testing modules (Figure 3) based on Figure 2. This scheme will utilize two .txt files, based on our truth tables, which will allow us to test all of our expected outcomes. These text files will be included in the lab report.



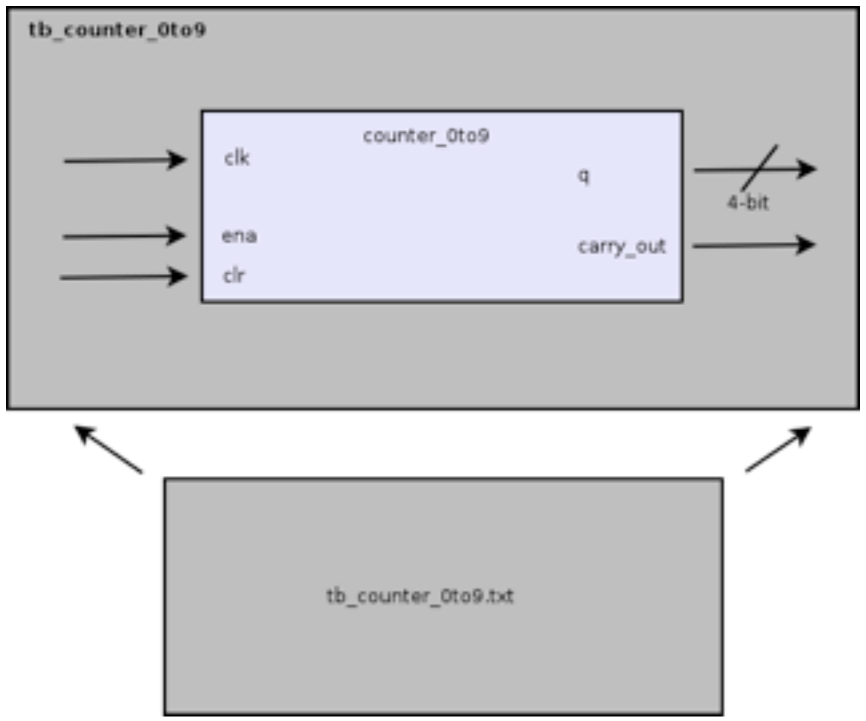


Figure 3: tb_counter_0to5 and tb_counter_0to9 test modules

Procedures:

Create the Truth Tables for Both Modules

1. Create a truth table for our module (figure 4).
2. Use the truth tables to devise the logic needed in our verilog modules.

Implement the Design in Software for part

1. Make a secure connection to electro9.eng.temple.edu using the no machine client.
2. Once terminal opens on the local workstation type the ‘remote_xilinx.sh’ shell command to launch the ISE development environment.
3. Open the lab6 project in ~/Xilinx/lab6/ directory.
4. Create the counter_0to5.v module with the following input and output settings

    a. input clk
    b. input ena
    c. input clr
    d. output reg [3:0] q
    e. output reg carry_out
    f. reg [3:0] Next_q
5. Modify the source to include an always block with the appropriate logic to match the results dictated by the truth table.
6. Modify the source to include an always block for the positive clock.
7. Repeat steps 4 through 5 for a second module called counter_0to9.v.
8. Save all files.

<u>Prepare for Testing the Design</u>

1. Verify that the tb_count_0to5.txt and tb_count_0to9.txt files suit the needs of our truth table by navigating to
    - View: Implementation
2. Verify that the tb_count_0to5.txt and tb_count_0to9.txt files contain all possible input bit combinations needed by our modules.
3. Save all files.

<u>Test the Design with iSim</u>

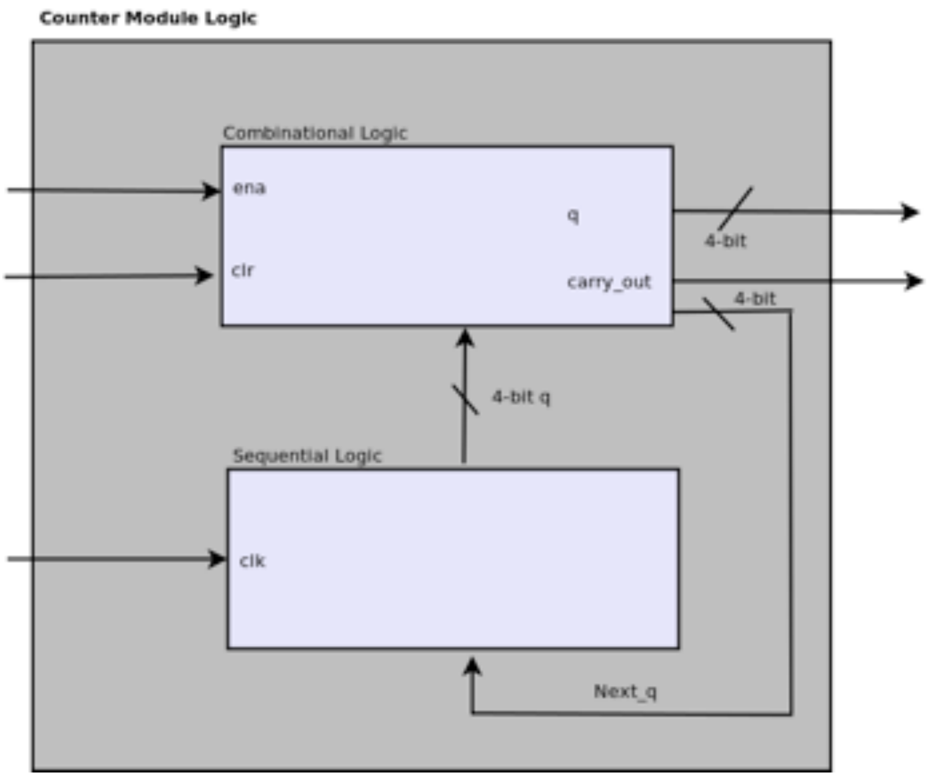
1. Switch to Simulation mode by clicking on
    a. View:Simulation
        i. Xc3s500e-4fg320.
        ii. tb_count_0to5
2. Run iSim simulator by clicking on
    a. iSim Simulator
    b. Right click Simulate Behavioral Model and then run.
3. Once iSim runs, verify that the Mismatch—index messages match what you are expecting in your test bench text file.
4. If the results are not what you expect either edit your module code or your test bench code and then attempt to test again.
5. If the results are what you expected repeat the steps below instead for the tb_count_0to9 module.
6. If the results are expected and you receive no mismatch errors your verilog implementation was successful.

Results:
Below you will find the truth tables for both of our counter modules.

Figure 4: Truth tables for counter_0to5 and counter_0to9 modules

0to5									0 to 9							
clr	ena	_	carry_over	q3	q2	q1	q0		clr	ena	_	carry_over	q3	q2	q1	q0
1	1		1	0	1	0	1		1	1		1	1	0	0	1
1	0		0	0	0	0			1	0		0	0	0	0	0
0	1		0	0	0	0	1		0	1		0	0	0	0	1
0	1		0	0	0	1			0	1		0	0	0	1	0
0	1		0	0	0	1	1		0	1		0	0	0	1	1
0	1		0	0	1	0			0	1		0	0	1	0	0
0	1		1	0	1	0	1		0	1		0	0	1	0	1
0	0		0	0	1	0	1		0	1		0	0	1	1	0
0	1		0	0	0	0	0		0	1		0	0	1	1	1
0	1		0	0	0	0	1		0	1		0	1	0	0	0
0	1		0	0	0	1	0		0	1		1	1	0	0	1
0	1		0	0	0	1	1		0	0		0	1	0	0	1
0	1		0	0	1	0	0		0	1		0	0	0	0	0
0	0		0	0	1	0	0		0	1		0	0	0	0	1
0	1		1	0	1	0	1		0	1		0	0	0	1	0
0	1		0	0	0	0	0		0	1		0	0	0	1	1
0	1		0	0	0	0	1		0	1		0	0	1	0	0
1	1		1	0	1	0	1		0	0		0	0	1	0	1
									0	1		0	0	1	1	0
									0	1		0	0	1	1	1
									0	1		0	1	0	0	0
									0	1		1	1	0	0	1
									0	1		0	1	0	0	1
									0	1		0	0	0	0	0
									0	1		0	0	0	0	1
									1	1		1	1	0	0	1

Figure 5: Block Diagram For Counters



Discussion:
This module was a good introduction to sequential logic and clocks. Before this I was not quite sure how they worked, but now I think I have a reasonable grasp on the subject.

In addition to this it also made clear in my mind that my Next_q register was an internal signal and not an output as it was in Lab 5. Initially I made this mistake early on in this lab and still got OK testing results. However, if I were to keep it an output it would violate the requirements of the lab. Now it is just an internal register signal instead.

I am however left curious about how we will control how fast things count up. I think we would have to slow down the clock cycle, or perhaps, change our logic so that it will only operate on every N numbers of cycles so that it may go up every second or every time interval that we choose to count.

Finally, I am still slightly confused about the testing portion of this lab. For example I can see that while in iSim that my clk Value will fluctuate from 0 to 1 even though I am currently not running or stepping through the simulation. Perhaps this is a bug, or more likely, it is me not understanding the test software fully. I will have to ask about this.

Source Code:

counter_0to5.v:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:19:11 10/08/2012
// Design Name:
// Module Name:    counter_0to5
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module counter_0to5(
   input ena,
   input clr,
   input clk,
   output reg [3:0] q,
   output reg carry_out,

         //Declare reg for Next_q
         reg [3:0] Next_q
   );

always @(posedge clk)
                begin
                q <= Next_q;
                end

always @(q or ena or clr)
                begin

                        Next_q = 4'b0000;
                        carry_out = 4'b0;


                        // ** HOLD **
                        //Logic when ena and clr are set to 0 to hold current values
                        if( (ena == 1'b0) && (clr == 1'b0))
                                 begin
                                         carry_out = 1'b0;
                                         Next_q[0] = q[0];
                                         Next_q[1] = q[1];
                                         Next_q[2] = q[2];
                                         Next_q[3] = q[3];
                                 end

                        // ** MAX **
                        //Logic to handle when ena =1 and clr = 1 to get ouput
                        //of 5

                        if ( (ena ==1'b1) && (clr == 1'b1) )
                                begin
                                        carry_out = 1'b1;
                                        Next_q = 4'b0101;
                                end


                        // ** NEXT COUNT **
                        //Logic when ena = 1 to calculate our next bit

                        if ((ena == 1'b1) && (clr == 1'b0))
                                begin

                                        Next_q = q + 1;

                                        if ( q == 5) carry_out = 1;
                                        if ( q >= 5) Next_q = 0;

                                end // end of if handling the next bit


                        // ** CLEAR **
                        //Logic when clr = 1 and ena = 0 to clear
                        if ((clr == 1'b1) && (ena == 1'b0))
                                begin
                                        Next_q = 4'b0000; // set to 0000.
                                        carry_out = 1'b0;                  // set the
carry bit to 0.
                                end // end if to set Next_1 and carry_out to 0.


                end  // End my entire always block
endmodule
```

counter_0to9.v:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    13:28:37 10/08/2012
// Design Name:
// Module Name:    counter_0to9
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module counter_0to9(
    input ena,
    input clr,
    input clk,
    output reg[3:0] q,
    output reg carry_out,

            //Declare Next_q reg type
            reg[3:0] Next_q
    );

        //Let us do some logic on our input switches


        always @(posedge clk)
                begin
                q <= Next_q;
                end


        always @(q or ena or clr)
                begin

                        Next_q = 4'b0000;
                        carry_out = 4'b0;


                        // ** HOLD **
                        //Logic when ena and clr are set to 0 to hold current values
                        if( (ena == 1'b0) && (clr == 1'b0))
                        begin
                                carry_out = 1'b0;
                                Next_q[0] = q[0];
                                Next_q[1] = q[1];
                                Next_q[2] = q[2];
                                Next_q[3] = q[3];
                        end



                // ** MAX **
                //Logic to handle when ena =1 and clr = 1 to get ouput
                //of 9

                if ( (ena ==1'b1) && (clr == 1'b1) )
                        begin
                                carry_out = 1'b1;
                                Next_q = 4'b1001;
                        end

                // ** NEXT COUNT **
                //Logic when ena = 1 to calculate our next bit

                if ((ena == 1'b1) && (clr == 1'b0))
                        begin

                                Next_q = q + 1;

                                if ( q == 9) carry_out = 1;
                                if ( q >= 9) Next_q = 0;

                        end // end of if handling the next bit



                // ** CLEAR **
                //Logic when clr = 1 and ena = 0 to clear
                if ((clr == 1'b1) && (ena == 1'b0))
                        begin
                                Next_q = 4'b0000; // set to 0000.
                                carry_out = 1'b0;   // set the carry bit to 0.
                        end // end if to set Next_1 and carry_out to 0.


        end  // End my entire always block


endmodule
```