EECE 144
Fall 2011

Lab Report #12
Section 4
11/30/2011

Submitted by: Jeremiah Mahler

Signature	Printed Name	Date
	Jeremiah Mahler	**Nov 30, 2011**
	Marvanee Johnson	**Nov 30, 2011**

1 Description/Objectives

The objective of this lab is to implement the three bit non-sequential counter from Lab 11[1] in Verilog along with an additional synchronous reset input. The counter will count in ascending order from top to bottom when $X = 0$ and in reverse order when $X = 1$.

0	0	0
0	1	0
1	1	0
0	1	1
1	0	1
0	0	1
1	0	0
1	1	1

2 Procedure

Since the counter was already designed in Lab 11[1] this will not be reproduced here. Instead the details of the Verilog implementation will be discussed.

The first step in the Verilog implementation is to construct the flip-flops since they are not available pre-built.

2.1 D flip-flop in Verilog

The implementation for the D flip-flop is given in Listing 1. The always block is triggered only on the positive edge as shown on line 8. The operation of the reset is shown on line 9.

When the reset bit is set it clears the output (`q`). When flip-flops are implemented it is recommended to use the non-block assignment (`<=`) as was done here shown on lines 10 and 13. In order to achieve the inverted output (`q_n`) it was necessary to use the continuous assignment operation `assign` as is shown on line 17. This in effect makes the value `q_n` always equal to of `~q`. It was also necessary to make `q_n` a wire and not a reg so that the continuous assignment would work as shown on line 7.

```
1  /*
2   * D flip-flop with synchronous reset
3   *
4   * The reset is active high and sets 'q' to zero.
5   *
6   */
7  module dff (input wire d, clk, reset, output reg q, output wire q_n);
8      always @(posedge clk) begin
9          if (reset == 1'b1) begin
10             q <= 1'b0;
11         end
12         else begin
13             q <= d;
14         end
15     end
16
17     assign q_n = ~q;
18 endmodule
```

Listing 1: Verilog source for D flip-flop with synchronous reset.

2.2 T flip-flop in Verilog

The implementation for the T flip-flop is given in Listing 2. It is similar to the D flip-flop from Section 2.1 in that it uses: `q_n` as a wire, non blocking assignment (`<=`) and positive edge triggering. Code where the effects are moot have been commented out to increase efficiency as shown on line 13 and 15.

```
1  /*
2   * T flip-flop with synchronous reset
3   *
4   * The reset is active high and sets 'q' to zero.
5   *
6   */
7  module tff (input wire t, clk, reset, output reg q, output wire q_n);
8      always @(posedge clk) begin
9          if (reset == 1'b1) begin
10             q <= 1'b0;
11         end
12         else if (t == 1'b0) begin
13             //q <= q;
14         end
```

```
15          //else if (t == 1'b1) begin
16          else begin
17              q <= ~q;
18          end
19      end
20
21      assign q_n = ~q;
22  endmodule
```

Listing 2: Verilog source for T flip-flop with synchronous reset.

2.3 JK flip-flop in Verilog

The implementation for the JK flip-flop is given in Listing 2. In general it is similar in design to the D flip-flop from Section 2.1 and the T flip-flop from Section 2.2 except that it has more cases to test.

```
1   /*
2    * JK flip-flop with synchronous reset
3    *
4    * The reset is active high and sets 'q' to zero.
5    *
6    */
7   module jkff (input j, k, clk, reset, output reg q, output wire q_n);
8       always @(posedge clk) begin
9           if (reset == 1'b1) begin
10              q <= 1'b0;
11          end
12          else if (j == 1'b0 && k == 1'b0) begin
13              //q <= q;
14          end
15          else if (j == 1'b0 && k == 1'b1) begin
16              q <= 0;
17          end
18          else if (j == 1'b1 && k == 1'b0) begin
19              q <= 1;
20          end
21          //else if (j == 1'b1 && k == 1'b1) begin
22          else begin
23              q <= ~q;
24          end
25      end
26
27      assign q_n = ~q;
28  endmodule
```

Listing 3: Verilog source for JK flip-flop with synchronous reset.

2.4 3 bit counter

Once all the flip flops have been built they can be connected together to construct the 3 bit counter.

The functions which define each bit were determined in Lab 11[1] are given in Equation 1, 2, and 3.

$$D_0 = Q_2Q_1' + X'Q_2Q_0' + XQ_1' + X'Q_2'Q_1Q_0 \tag{1}$$
$$T_1 = X'Q_1'Q_0' + X'Q_1Q_0 + XQ_2Q_0 + XQ_2'Q_0' \tag{2}$$
$$J_2/K_2 = X'Q_1 + X'Q_0 + Q_2Q_1Q_0' + Q_2'Q_0 + XQ_1' \tag{3}$$

The implementation for the 3 bit counter is given in Listing 4. Lines 1-3 show that it includes the flip flops that were defined previously. Lines 16-18 shown the use of the previously defined flip flops. Conceptually this is like soldering these chips to the circuit board. On lines 21-28 the continuous assignment operation is used to connect all the wires together and to define the output of each flip-flop. These define the equations that were given previously.

```
1  `include "jkff.v"
2  `include "dff.v"
3  `include "tff.v"
4
5  /*
6   * 3 bit up down counter with synchronous reset.
7   *
8   * The reset is active high and sets all outputs to zero.
9   *
10  */
11 module threebitudc (input wire x, clk, reset, output wire q2, q1, q0);
12     wire jk2, t1, d0;
13     wire q0_n, q1_n, q2_n;
14
15     // Include the flip-flops
16     dff q0ff(.d(d0), .clk(clk), .reset(reset), .q(q0), .q_n(q0_n));
17     tff q1ff(.t(t1), .clk(clk), .reset(reset), .q(q1), .q_n(q1_n));
18     jkff q2ff(.j(jk2), .k(jk2), .clk(clk), .reset(reset), .q(q2), .q_n(q2_n));
19
20     // connect all the wires to each other
21     assign jk2 = (~x && q1) || (~x && q0)  || (q2 && q1 && ~q0)
22                             || (~q2 && q0) || (x && ~q1);
23
24     assign t1 = (~x && ~q1 && ~q0)  || (~x && q1 && q0)
25                                     || (x && q2 && q0) || (x && ~q2 && ~q0);
26
27     assign d0 = (q2 && ~q1) || (~x && q2 && ~q0) || (x && ~q1)
28                             || (~x && ~q2 && q1 && q0);
```

```
29 endmodule
```

Listing 4: Verilog source for 3 bit up/down counter with synchronous reset.

2.5 Compiling Verilog source and running GTKWave

Once all the code has been defined it can be compiled and run. The test bench is given in AppendixA.

To compile the source code using Icarus Verilog[2] under Linux the following commands can be run.

```
iverilog test.v
```

This will produce a filed named 'a.out'. Under Linux this can be executed directly as shown below.

```
./a.out
```

Alternatively the `vvp` command can be used. This method works under Linux or Windows.

```
vvp a.out
```

Because the `$dumpfile` and `$dumpvars` have been added to the test bench it will produce output suitable for GTKWave[3]. The output file should have the extension '.vcd'. This file can then be run using GTKWave as shown below.

```
gtkwave output.vcd
```

And from within GTKWave the various variables can be selected to be displayed.

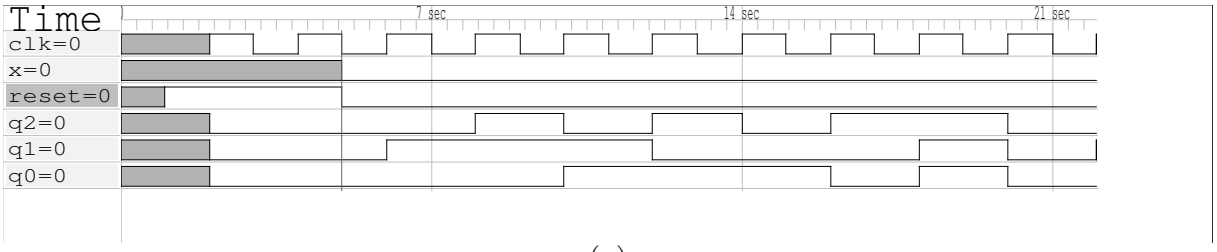
3 Observations

The output of GTKWave for a counting sequence going up/down is given in Figure 1. All sequences triggered on the positive edge and followed the correct order.

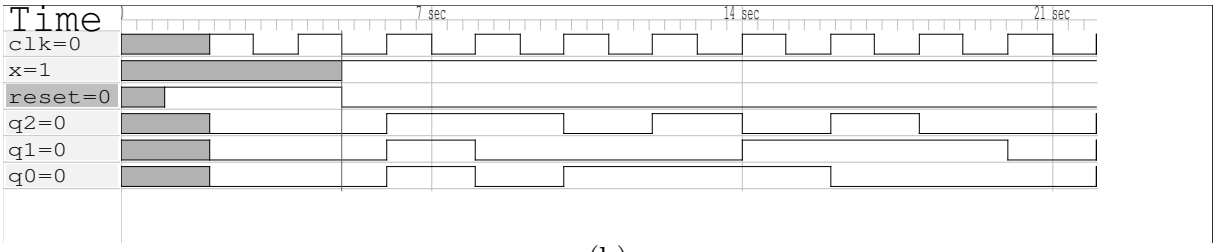
The output of GTKWave for a counting sequence going up with a reset in the middle is given in Figure 2. The outputs are reset to zero and it resumes counting up with the next value of 0 1 0 as it should.

4 Conclusion

This lab was a complete success in implementing a three bit non-sequential counter with a synchronous reset in Verilog. All design requirements were met and the counter behaved as expected.



(a)



(b)

Figure 1: Output of GTKWave when counting up (a) ($X = 0$) and down (b) ($X = 1$).

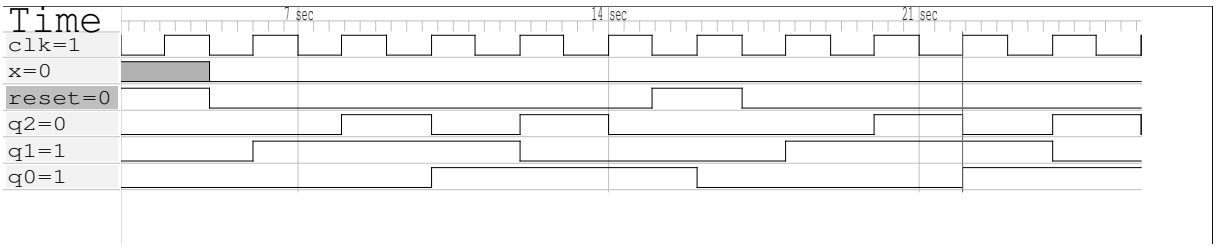


Figure 2: Output of GTKWave for the reset condition in the middle of a counting sequence.

5 References

[1] J. Mahler and M. Johnson, "Lab 11 (eece 144)." `https://github.com/jmahler/EECE-144-labs/tree/master/lab11`, 2011.

[2] S. Williams, "Icarus verilog." `http://iverilog.icarus.com/`, 2011.

[3] T. Bybell, "Gtkwave." `http://gtkwave.sourceforge.net/`, 2011.

A Verilog Test Bench

```
1  /*
2   * Test bench for 3 bit counter.
3   *
4   * To compile this file run:
5   *
6   *   iverilog this_file.v
7   *
8   * Run the executable:
9   *
10  *   ./a.out
11  * OR
12  *   vvp a.out
13  *
14  * Because $dumpfile and $dumpvars have been
15  * added it will generate data for gtkwave
16  * in 3bitcounter.vcd.
17  *
18  * This output file can then be shown with Gtkwave.
19  *
20  *   gtkwave 3bitcounter.vcd
21  *
22  * And from within Gtkwave you can pick and choose
23  * variables to see their waveforms over time.
24  *
25  *
26  * This project was completed as part of lab 12 in the
27  * class EECE-144 taught by Kurtis Kredo II at Chico State
28  * during the Fall of 2011.
29  *
30  * Author(s): Jeremiah Mahler <jmmahler@gmail.com>
31  *                    Marvanee Johnson
32  */
33
34 `include "3bitcounter.v"
35
36 module test;
37     reg clk, reset, x;
38     wire q2, q1, q0;
39         integer i;
40
41     threebitudc tbudc(.x(x), .clk(clk), .reset(reset), .q2(q2), .q1(q1), .q0(q0));
42
43     initial begin
44         $dumpfile("gtkwave-output.vcd");
45         $dumpvars(0,test);
46
47         //#1 $display("clk   x   q2   q1   q0");
48         //#1 $display("--------------------");
49
```

```
50          // reset and initialize the clock
51          #1 reset = 1;
52          #1 clk = 0;
53          #1 clk = 1;
54          #1 clk = 0;
55          #1 reset = 0;
56
57          x = 0; // 0 up, 1 down
58                  for (i = 0; i <= 8; i = i + 1) begin
59                          #1;
60                  end
61
62                  // reset in the middle
63                  #1 reset = 1;
64          #1 clk = 0;
65          #1 clk = 1;
66          reset = 0;
67
68          //x = 0; // 0 up, 1 down
69                  for (i = 0; i <= 8; i = i + 1) begin
70                          #1;
71                  end
72
73          //$monitor("%b    %b  %b  %b  %b", clk, x, q2, q1, q0);
74
75          //#16 $finish;
76          /* Each time unit switches the clock up/down (see below).
77           * We need twice as many here since the flip-flop is triggered
78           * only on the positive edge.
79          */
80
81                  $finish;
82      end
83
84      always begin
85          #1 clk = ~clk;
86      end
87  endmodule
```