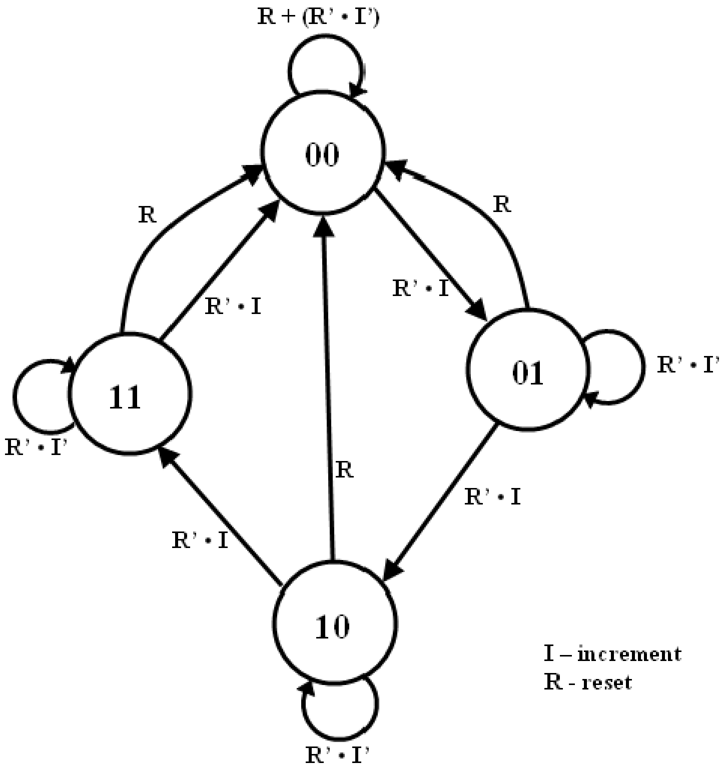
Colt Thomas

March 17, 2014

Lab 8 – 4x7 Segment Display Controller

Preparation:

Below is the work that I did for the design of the Mod4 state machine. There is also the included diagrams of the FSM



Here is the scanned work that I did:

- Truth Table

R	I	Q_1	Q_0	N_1	N_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	0	0
1	-	-	-	0	0

- K-maps

N_1

Q_1Q_0 \ RI	00	01	11	10
00	0	0	0	0
01	1	1	0	0
11	1	0	0	0
10	1	1	0	0

N_0

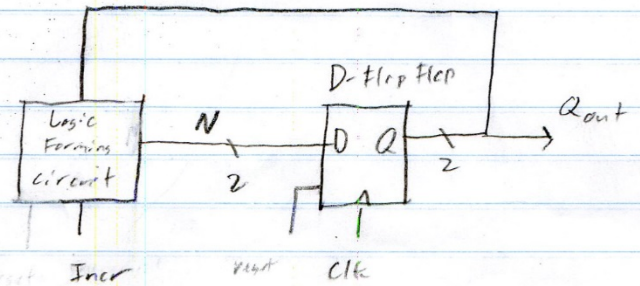
Q_1Q_0 \ RI	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	1	0	0	0
10	0	1	0	0

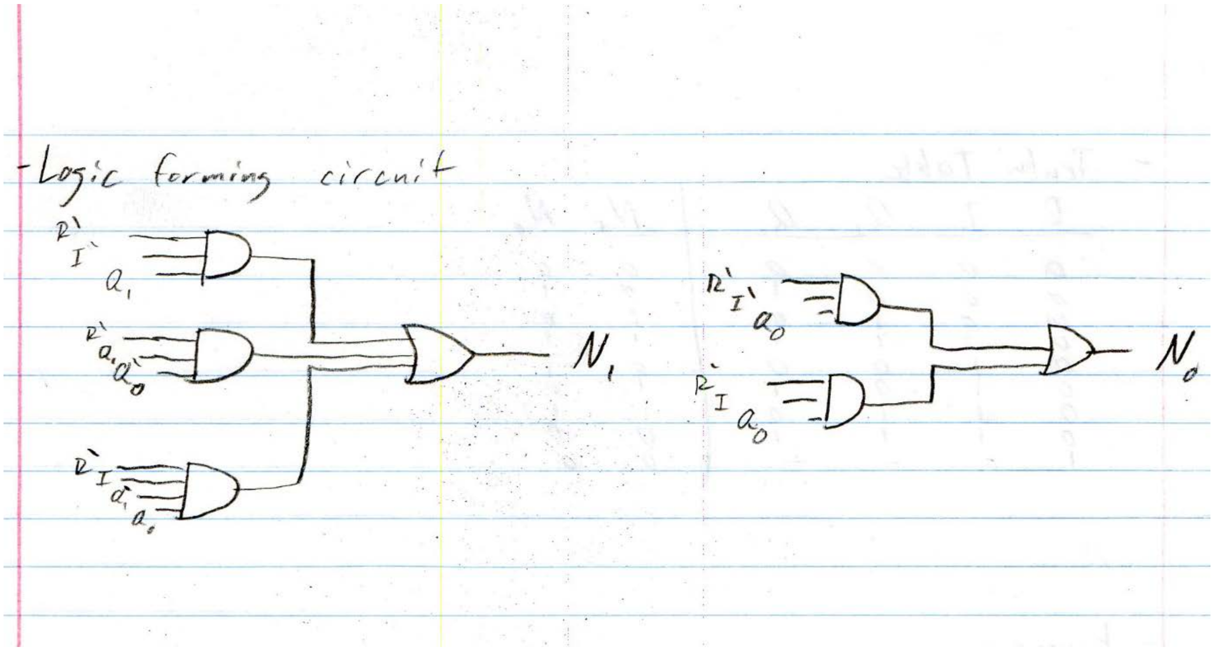
- Logic equations

$$N_1 = R'I'Q_1 + R'Q_1Q_0' + R'IQ_1'Q_0$$

$$N_0 = R'I'Q_0 + R'IQ_0'$$

- Below is an implementation of the Mod 4 State machine.





Procedure:

1.) Build the mod4 state machine using Verilog code

Below we have the modules (in Verilog) for the mod4 FSM:

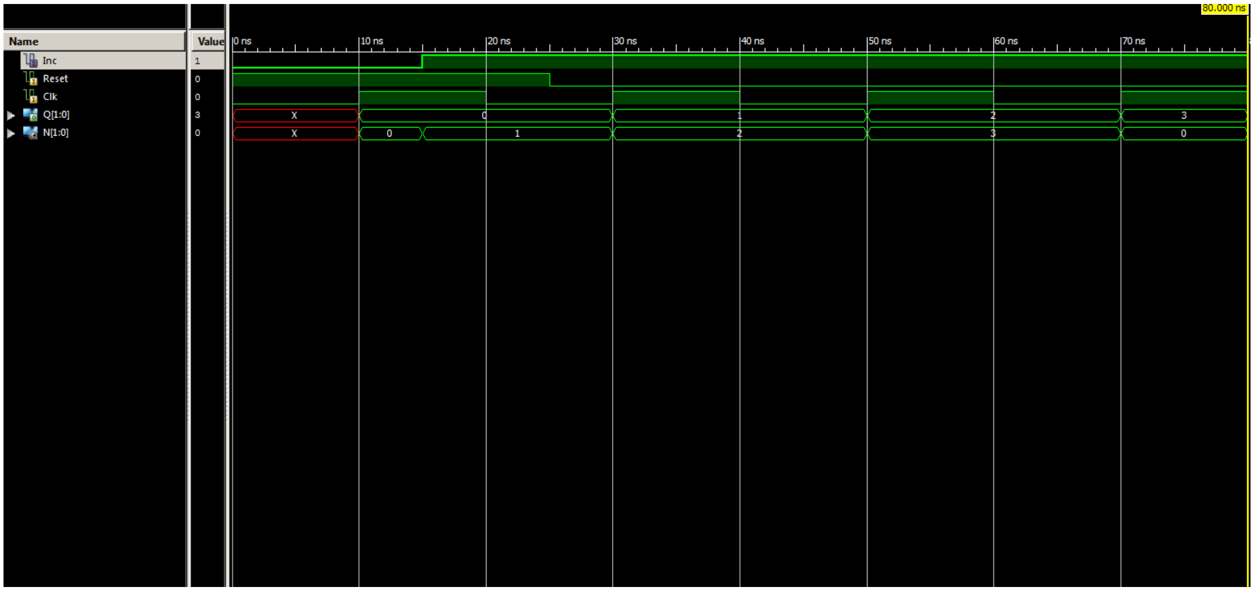
```
module mod4(
   input Inc,
   input Reset,
   input Clk,
   output[1:0] Q
   );
                              wire [1:0] N;


                              LFC Logic( Inc , Q , N);
                              FF_DC D_1(Q[1] , Clk , Reset , N[1]);
                              FF_DC D_0(Q[0] , Clk , Reset , N[0]);



endmodule
```

```
module LFC(

  input Inc,
  input [1:0] Q,
  output [1:0] N
  );
                    wire a_1 , b_1 , c_1 , a_0 , b_0;
                    wire final_1 , final_0;


                    //potential errors: the ~ doesn't negate
                    and(a_1, !Inc , Q[1]);
                    and(b_1, Q[1] , !Q[0]);
                    and(c_1, Inc , !Q[1], Q[0]);

                    and(a_0 , !Inc, Q[0]);
                    and(b_0 , Inc  , !Q[0]);

                    or (final_1 , a_1 , b_1 , c_1);
                    or (final_0 , a_0 , b_0);

                    assign N[0] = final_0;
                    assign N[1] = final_1;

endmodule

module FF_DC(
  input q,
  input clk,
  input clr,
  output d
  );
          input clk, clr, d;
          output reg q;

          always @(posedge clk)
                    if (clr) q <= 0;
                    else q <= d;

endmodule
```

Here is the TCL file for the MOD4:

```
wave add / -radix hex



isim force add Inc 0 -time 0 -value 1 -time 15ns

isim force add Reset 1 - time 0  -value 0 -time 25ns

isim force add Clk 0 -time 0 -value 1 -time 10ns -value 0 -time 20ns -value 1 -time 30ns -value 0 -time 40ns -value 1 -time 50ns -value 0 -time 60ns -value 1 -time 70ns -value 0 -time 80ns

#isim force add Q 00 -time 0 -value 01 -time 10ns -value 10 -time 20ns -value 11 -time 30ns

run 80ns
```

And this is the simulation that I used:



2.) Download the Programmable timer. Add it to the design
-Self Explanatory. We just added it to the project.

3.) See the Programming the Timer tutorial
-In the tutorial I was able to learn how to set the timing for the timer

4.) Build a Test Bench for the programmable timer

5.) Program the timer to give a 200 Hz signal
Below is the verilog code and the UCF file:

```
module timer_testbench(
   input clk,
   input reset,
   input clken,
   input [23:0] load_number,
   output [23:0] counter,
   output zero,
   output tp
   );

               prog_timer timer(clk, reset, clken , 24'd250000, counter, zero, tp);


endmodule
```

.UCF file

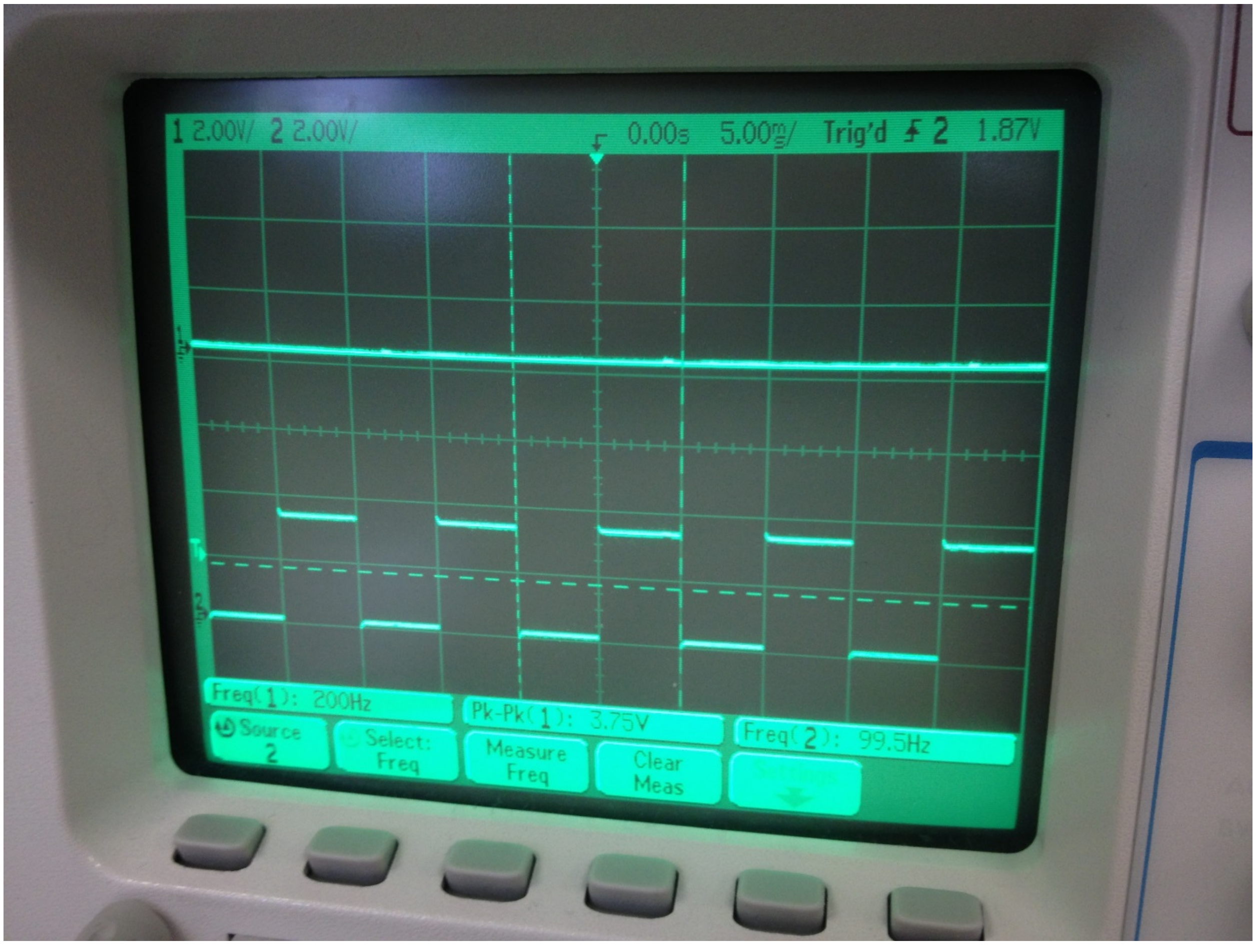
```
NET clk   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type = GCLK, Sch name = GCLK0
NET reset LOC = "H13"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN3
NET zero LOC = "B4";  # Bank = 0, Pin name = IO_L24N_0, Type = I/O, Sch name = R-IO1
NET tp  LOC = "A4";  # Bank = 0, Pin name = IO_L24P_0, Type = I/O, Sch name = R-IO2
```

6.) Determine the value to load into the programmable timer to achieve this result
-To do this we need to do some math:

50MHz / 200 Hz = 250000

This value we put into the timer. We were able to attain the desired frequency that we needed.

7.) Download the circuit to the Xilinx board and view the **tp** and **zero** clock on the scope



8.) The zero signal's frequency should be 200Hz with a pulse width of one clock cycle. The tp signal will be half of the zero frequency.
-As you can see above, we have 200 Hz for zero (line 1) and about 100 Hz for tp.

9.) Set the scope to measure frequency for you
-done

10.) Build the 4x7 Segment Controller using the parts listed in the preparation
-The parts that were listed included:

11.) Create a test bench for the Seven Segment Controller part
I built a testbench for the 4x7 segment controller. Below is the code for both the testbench and the 4x7 Seg Controller:

Testbench:

```
module testbench_SegCtrl4x7(

   input SysClk,
   input Reset,
   input Dp0,
   input Dp1,
   input Dp2,
   input Dp3,
   output AN0,
   output AN1,
   output AN2,
   output AN3,
   output DP,
   output Ca,
   output Cb,
   output Cc,
   output Cd,
   output Ce,
   output Cf,
   output Cg,
   output Q1,
   output Q0,
   output tp,
   output zero
   );
               wire [3:0] Digit1;
   wire [3:0] Digit2;
   wire [3:0] Digit3;
   wire [3:0] Digit4;
assign Digit1 = 4'b0001;
assign Digit2 = 4'b1010;
assign Digit3 = 4'b1011;
assign Digit4 = 4'b1000;

SegmentController4x7 thing(Digit1 , Digit2 , Digit3 , Digit4 , SysClk , Reset , Dp0, Dp1, Dp2, Dp3, Ca , Cb , Cc , Cd , Ce ,Cf , Cg ,AN0,
AN1, AN2, AN3, DP , Q1 , Q0 , tp , zero);

endmodule
```

4x7 Seg. Controller:

```
module SegmentController4x7(Digit1 , Digit2, Digit3, Digit4 , SysClk, Reset, Dp0 , Dp1, Dp2, Dp3,Ca , Cb , Cc , Cd , Ce , Cf , Cg , AN0 ,
AN1 ,AN2 , AN3, DP ,Q1 , Q0 , tp , zero );

input [3:0] Digit1 , Digit2 , Digit3 , Digit4;
input SysClk, Reset, Dp0, Dp1, Dp2, Dp3;
output  AN0, AN1, AN2, AN3;
output DP, Ca , Cb , Cc , Cd  ,Ce , Cf ,Cg;
//test signals for the clock
output Q1 , Q0;
output tp , zero;
wire [1:0] adrr;
wire[23:0] counter;
wire [3:0] Digit_Out;
wire [3:0] AN;

//since inc = '1', we just put 1'b1 for that input.  Reset will start it over
prog_timer Timer(SysClk , Reset , 1'b1 , 24'd250000 , counter,zero , tp);
mod4 Mod_4(zero, Reset, SysClk , adrr);
```

```
assign Q1 = adrr[1];
assign Q0 = adrr[0];
assign AN3 = AN[3];
assign AN2 = AN[2];
assign AN1 = AN[1];
assign AN0 = AN[0];
mux16to4 MUX16_4(Digit_Out , adrr , Digit1 , Digit2 , Digit3 , Digit4);
TestBench Seven_Seg_Decoder(Digit_Out , Ca , Cc ,Cf , Cg, Cb , Ce , Cd ) ;

Decoder24 Decoder(adrr , AN );
mux41 MUX_41( DP, adrr , Dp3 , Dp2 , Dp1 , Dp0);

endmodule
```

UCF file for the 4x7 Segment Controller

```
NET SysClk   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type = GCLK, Sch name = GCLK0
NET Dp0 LOC = "G18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW0
NET Dp1 LOC = "H18"; # Bank = 1, Pin name = IP/VREF_1, Type = VREF, Sch name = SW1
NET Dp2 LOC = "K18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW2
NET Dp3 LOC = "K17"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW3

NET Reset LOC = "B18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN0

NET Ca LOC = "L18"; # Bank = 1, Pin name = IO_L10P_1, Type = I/O, Sch name = CA
NET Cb LOC = "F18"; # Bank = 1, Pin name = IO_L19P_1, Type = I/O, Sch name = CB
NET Cc LOC = "D17"; # Bank = 1, Pin name = IO_L23P_1/HDC, Type = DUAL, Sch name = CC
NET Cd LOC = "D16"; # Bank = 1, Pin name = IO_L23N_1/LDC0, Type = DUAL, Sch name = CD
NET Ce LOC = "G14"; # Bank = 1, Pin name = IO_L20P_1, Type = I/O, Sch name = CE
NET Cf LOC = "J17"; # Bank = 1, Pin name = IO_L13P_1/A6/RHCLK4/IRDY1, Type = RHCLK/DUAL, Sch name = CF
NET Cg LOC = "H14"; # Bank = 1, Pin name = IO_L17P_1, Type = I/O, Sch name = CG
NET DP     LOC = "C17"; # Bank = 1, Pin name = IO_L24N_1/LDC2, Type = DUAL, Sch name = DP

NET AN0 LOC = "F17"; # Bank = 1, Pin name = IO_L19N_1, Type = I/O, Sch name = AN0
NET AN1 LOC = "H17"; # Bank = 1, Pin name = IO_L16N_1/A0, Type = DUAL, Sch name = AN1
NET AN2 LOC = "C18"; # Bank = 1, Pin name = IO_L24P_1/LDC1, Type = DUAL, Sch name = AN2
NET AN3 LOC = "F15"; # Bank = 1, Pin name = IO_L21P_1, Type = I/O, Sch name = AN3

NET zero  LOC = "B4";  # Bank = 0, Pin name = IO_L24N_0, Type = I/O, Sch name = R-IO1
NET tp  LOC = "A4";  # Bank = 0, Pin name = IO_L24P_0, Type = I/O, Sch name = R-IO2
```

12.) Test the result of our design
-After the code above was implemented as above, it worked according to specifications

Anomalies

There were a lot of anomalies due to misunderstanding of directions. I tried to implement all of my old projects without testing anything. I didn't think to use .tcl files to test the functionality of my design. I finally found all of the little bugs that were in my design. They were mainly from me making the output to the MOD4 as an ouput reg. I also forgot to negate some things in my 7 segment decoder. Most of this could have been avoided had I tested a module before implementing it.