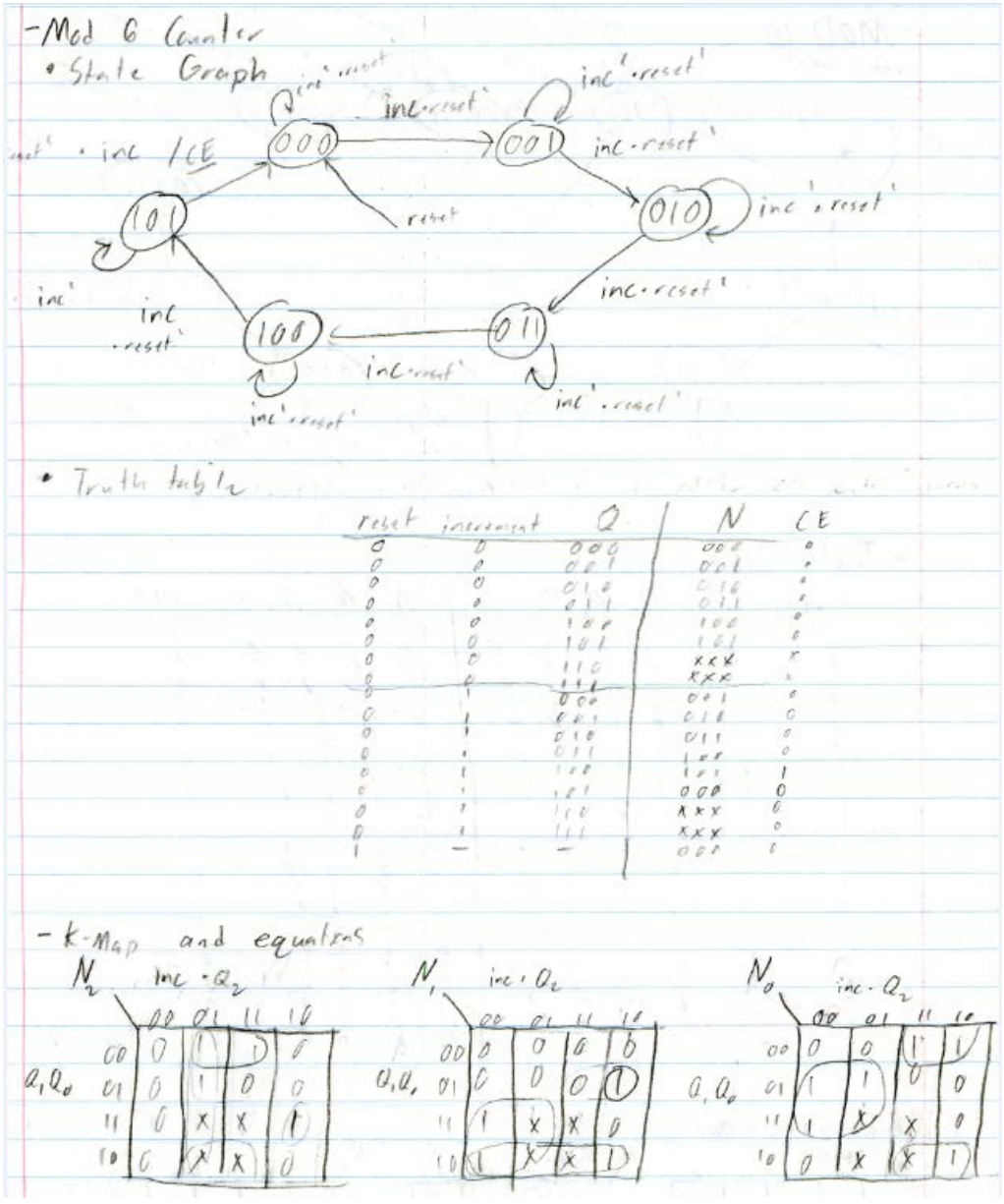
Colt Thomas

March 22, 2014

Lab 9 – Counters State Machine Drive©

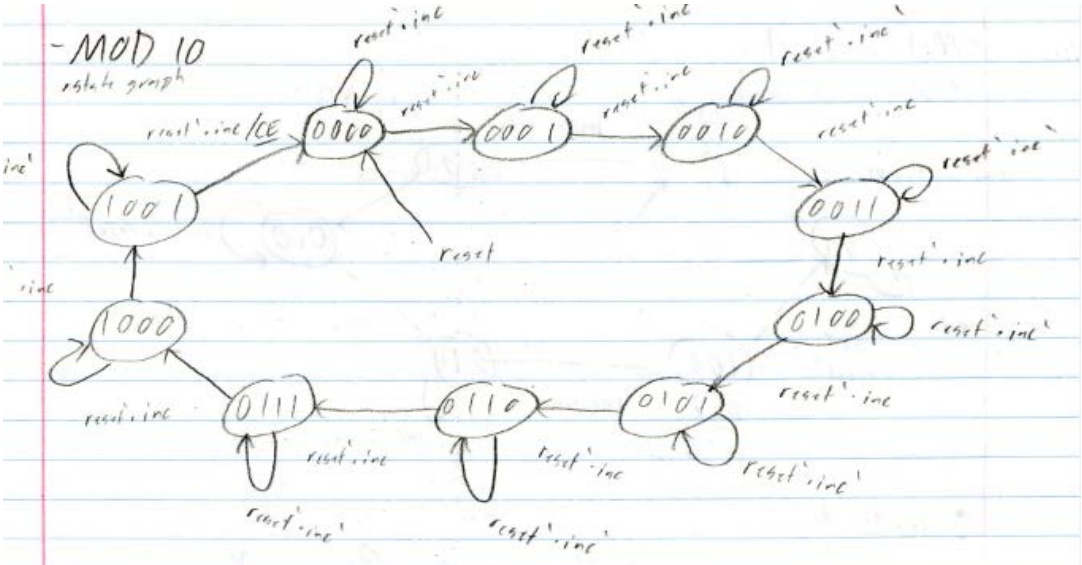
Lab Prep:

-below is the work that I did to get the logic down for both the mod6 and the mod10. Truth tables, kmaps and state graphs are included.



- MOD 10

state graph

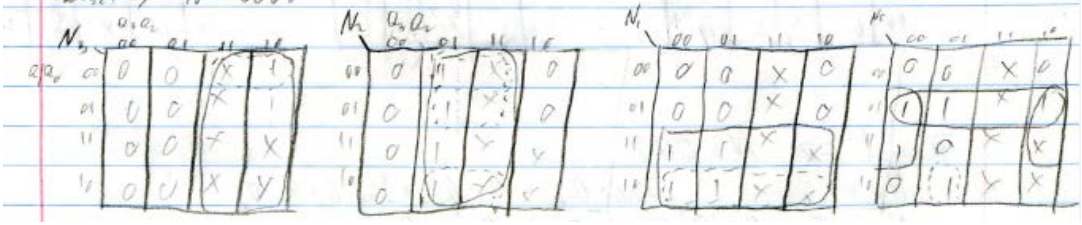


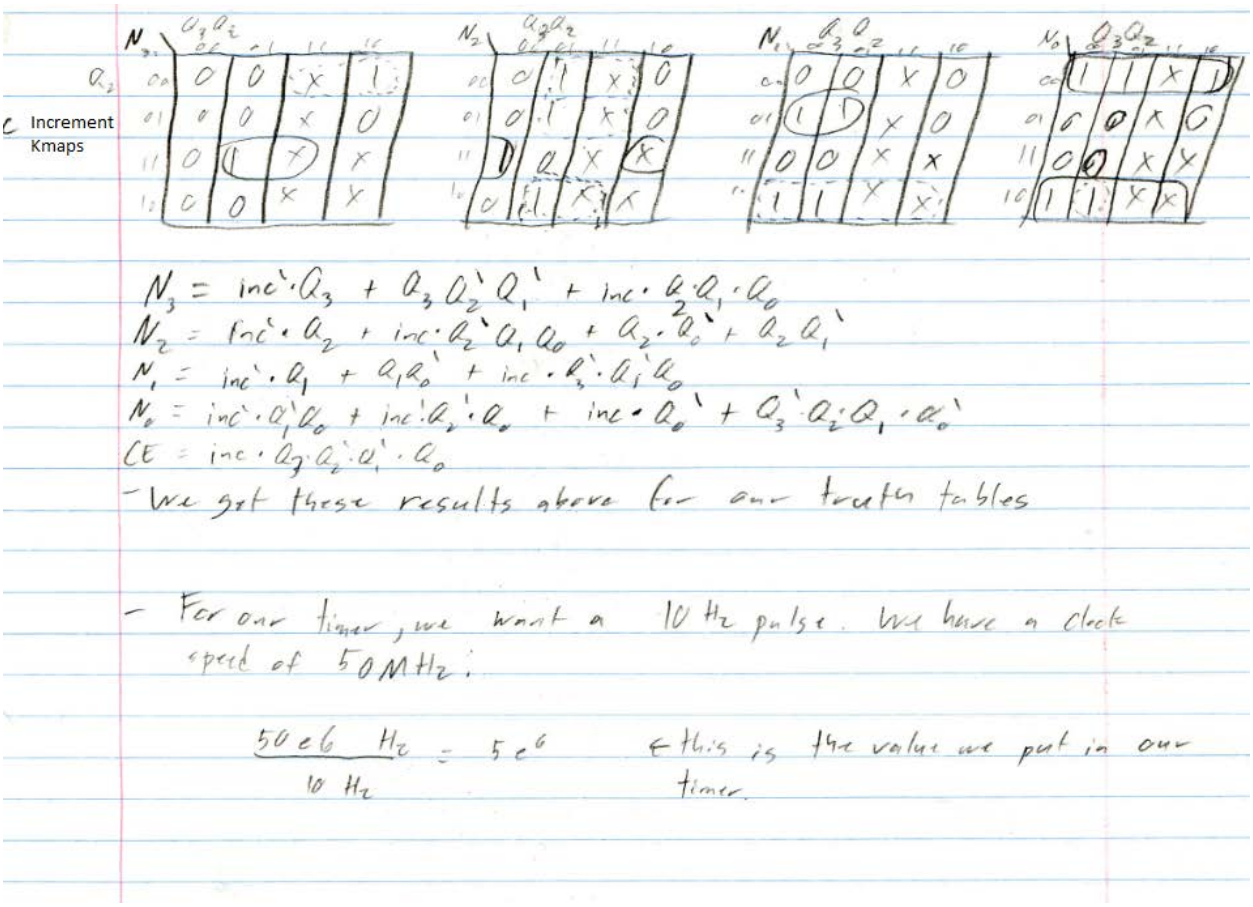
- Truth table

reset	inc	Q_3	Q_2	Q_1	Q_0	N_3	N_2	N_1	N_0	CE
0	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0	1	0	0
0	0	0	0	1	0	0	0	1	1	0
0	0	0	0	1	1	0	1	0	0	0
0	0	0	1	0	0	0	1	0	1	0
0	0	0	1	0	1	0	1	1	0	0
0	0	0	1	1	0	0	1	1	1	0
0	0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0	0	0	1
0	1	0	0	0	0	0	0	0	1	0
0	1	0	0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	0	1	1	0
0	1	0	0	1	1	0	1	0	0	0
0	1	0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	0	1	1	0	0
0	1	0	1	1	0	0	1	1	1	0
0	1	0	1	1	1	1	0	0	0	0
0	1	1	0	0	0	1	0	0	1	0
0	1	1	0	0	1	0	0	0	0	1
1	-	-	-	-	-	0	0	0	0	0

Reset => N = 0000

Inc' kmaps





-as you can see, the calculation we came out for our programmable timer is 5,000,000. If we need to adjust this to get exactly 10Hz, we will do so.

SR latch Verilog:

```
module SR_latch(
  input S,
  input R,
  output Q,
          output Q_not
  );

          assign Q = ~(R | Q_not);
          assign Q_not = ~( S | Q);

endmodule
```

SR latch tickleness:

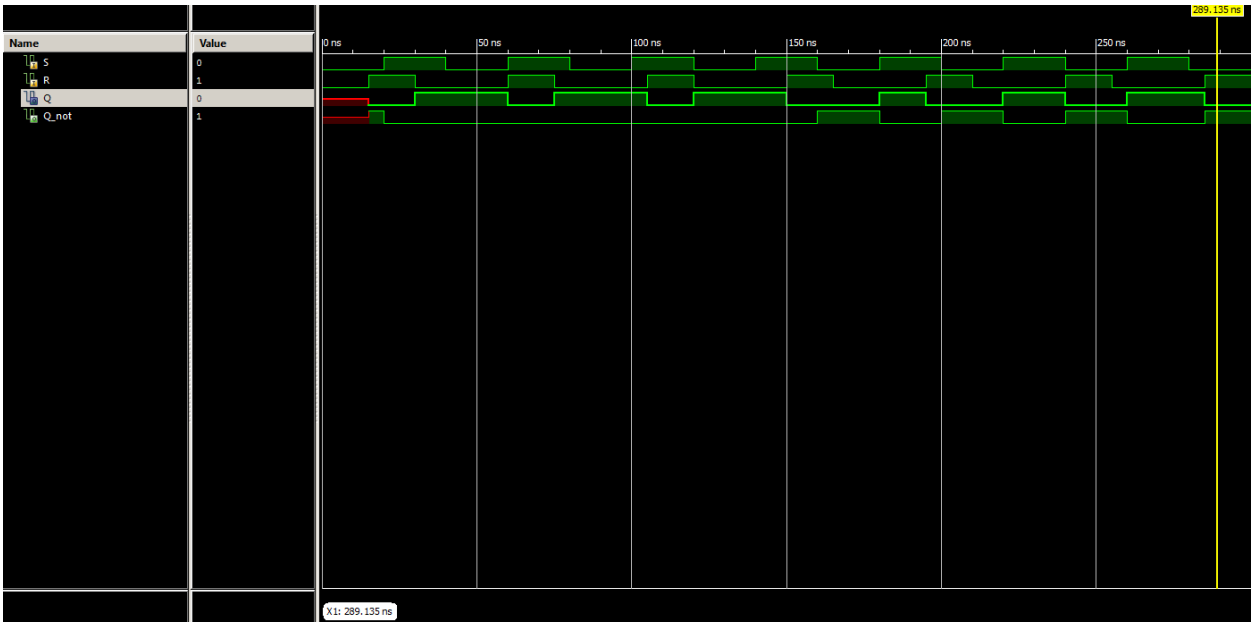
-Below is the .tcl file for the simulations

```
wave add / -radix hex

isim force add S 0 -time 0 -value 1 -time 20ns  -repeat 40ns
isim force add R 0 -time 0 -value 1 -time 15ns  -value 0 -time 30ns -repeat 45ns

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

run 300ns

SR latch simulation:

-Notice that in cases where R = S = ‘1’ we have a conflict between Q and Q_not. This is normal, but must be avoided in implementation.



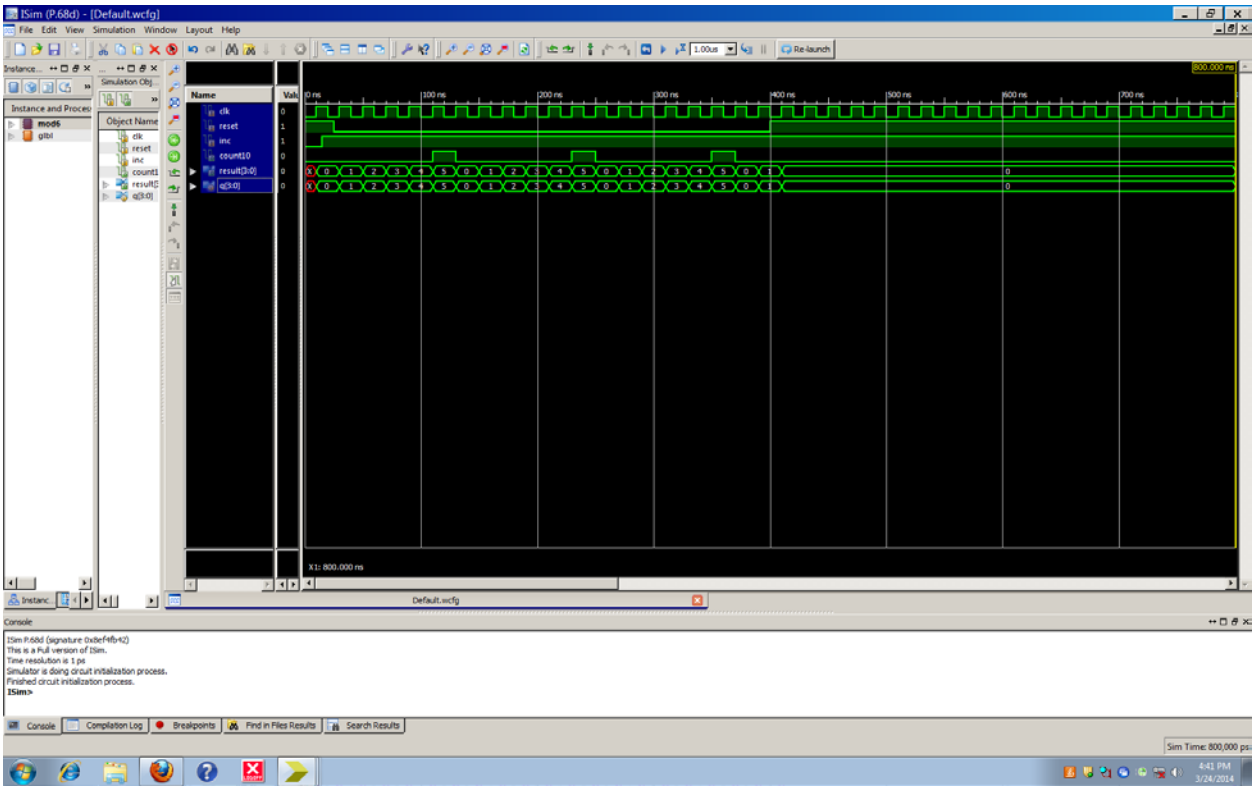
Verilog for Mod6:

```
module mod6 ( input clk, reset, inc,
 output count10);
 reg [3:0] q;

 always @(posedge clk)
 if (reset || (inc && q==5)) q <= 0;
 else if (inc) q <= q+1;
 assign count10 = inc & (q == 5);
endmodule
```

.tcl file for the mod6

```
wave add / -radix hex

isim force add clk 0 -time 0 -value 1 -time 10ns -repeat 20 ns
isim force add inc 0 -time 0 -value 1 -time 15ns
isim force add reset 1 - time 0  -value 0 -time 25ns -value 1 -time 400ns

#isim force add Q 00 -time 0 -value 01 -time 10ns -value 10 -time 20ns -value 11 -time 30ns
run 800ns
```

Mod6 simulation:



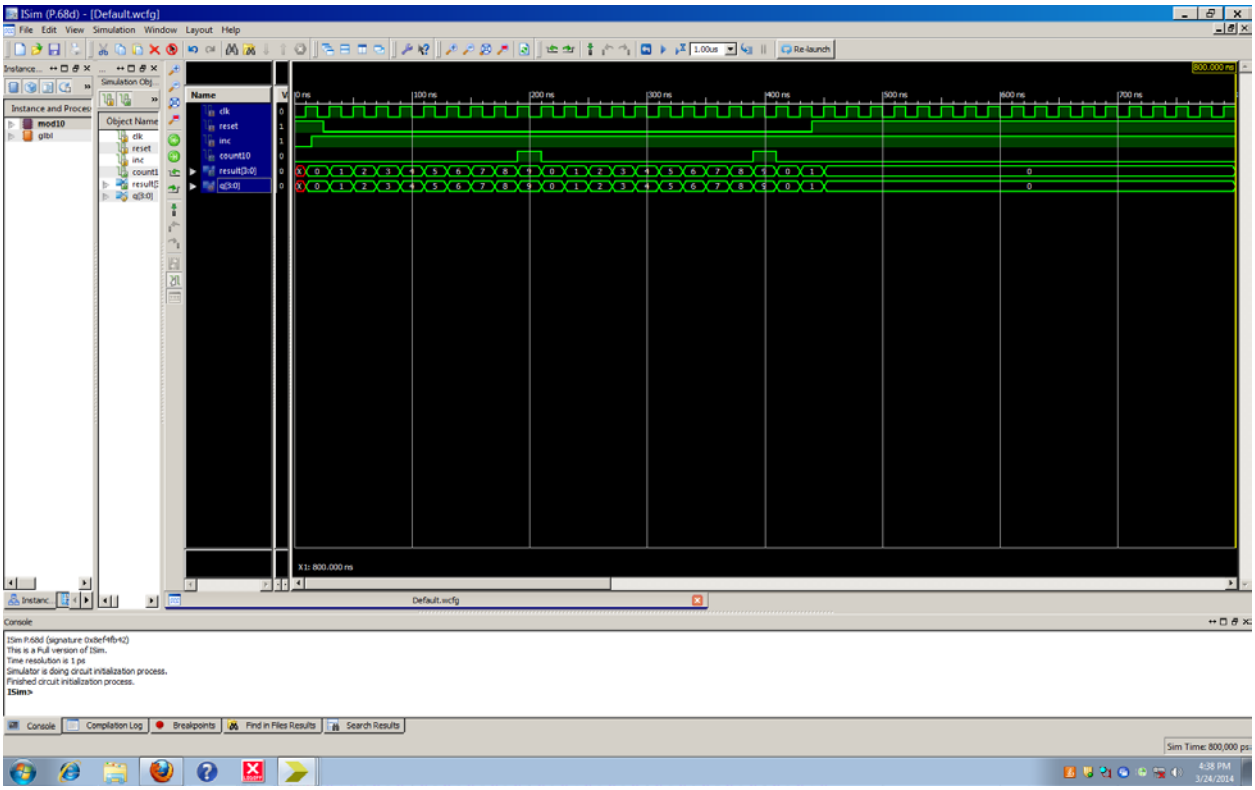
Mod10 Verilog:

```
module mod10 ( input clk, reset, inc,
 output count10);

 reg [3:0] q;

 always @(posedge clk)
 if (reset || (inc && q==9)) q <= 0;
 else if (inc) q <= q+1;

 assign count10 = inc & (q == 9);

endmodule
```

Mod10 tickle file:

```
wave add / -radix hex

isim force add clk 0 -time 0 -value 1 -time 10ns -repeat 20 ns
isim force add inc 0 -time 0 -value 1 -time 15ns
isim force add reset 1 - time 0  -value 0 -time 25ns -value 1 -time 440ns

#isim force add Q 00 -time 0 -value 01 -time 10ns -value 10 -time 20ns -value 11 -time 30ns
run 800ns
```

Mod10 simulation:



Counter Block Verilog:

```
module CounterBlock(
   input inc,
   input Reset,
   input SysClk,
   output [3:0] sec_tenths,
   output [3:0] sec_ones,
   output [2:0] sec_tens,
   output [3:0] min_ones
   );

wire rolloverTo_ones , rolloverTo_tens, rolloverTo_min, rolloverTo_ground;

mod10 tenths(SysClk , Reset, inc, rolloverTo_ones , sec_tenths);
mod10 ones(SysClk , Reset, rolloverTo_ones, rolloverTo_tens , sec_ones);
mod6 tens(SysClk , Reset, rolloverTo_tens, rolloverTo_min , sec_tens);
mod10 min(SysClk , Reset, rolloverTo_min, rolloverTo_ground, min_ones);

endmodule
```

Counter Block .tcl file:

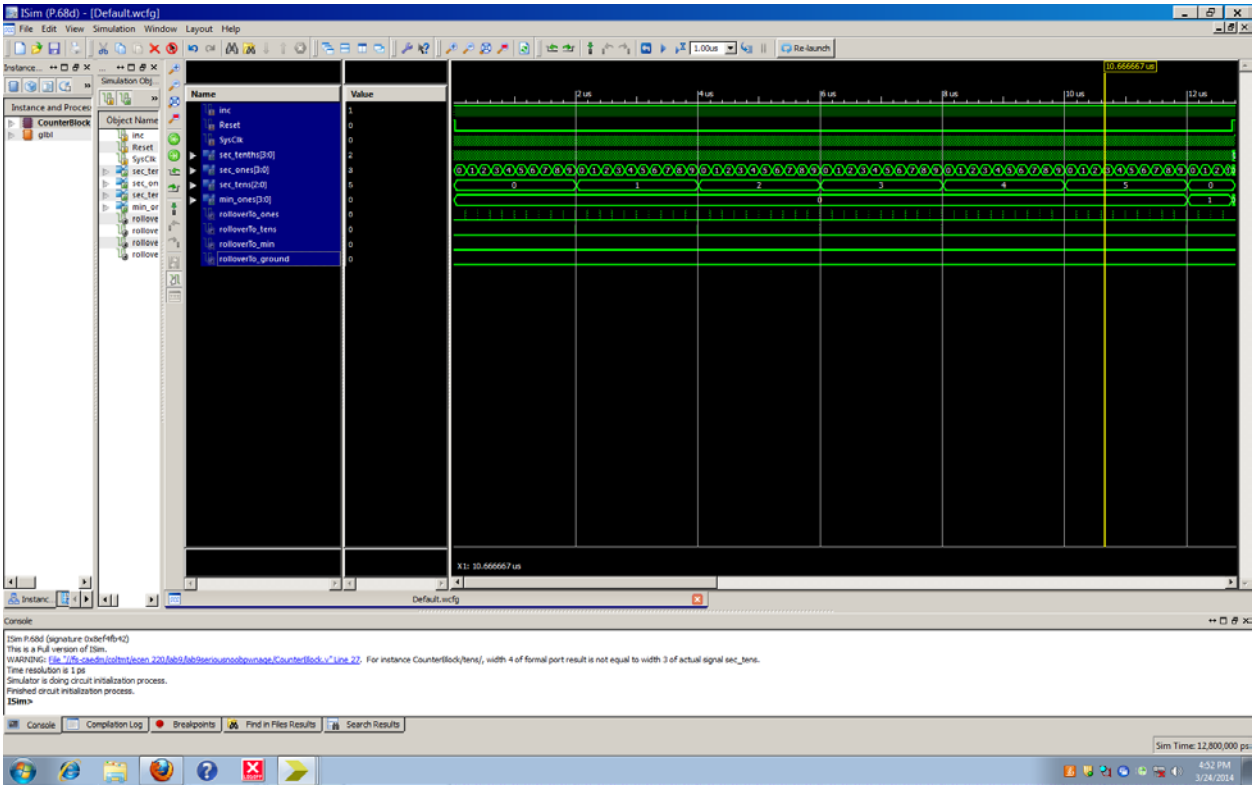
-this .tcl file tests enough to make sure that the mod6 works. It doesn't go as far to test the minutes since the seconds has working mod10 modules.

```
wave add / -radix hex

isim force add SysClk 0 -time 0 -value 1 -time 10ns -repeat 20 ns
```

isim force add inc 0 -time 0 -value 1 -time 15ns
isim force add Reset 1 - time 0 -value 0 -time 25ns -value 1 -time 12750ns

#isim force add Q 00 -time 0 -value 01 -time 10ns -value 10 -time 20ns -value 11 -time 30ns
run 12800ns

Counter Block Simulation:



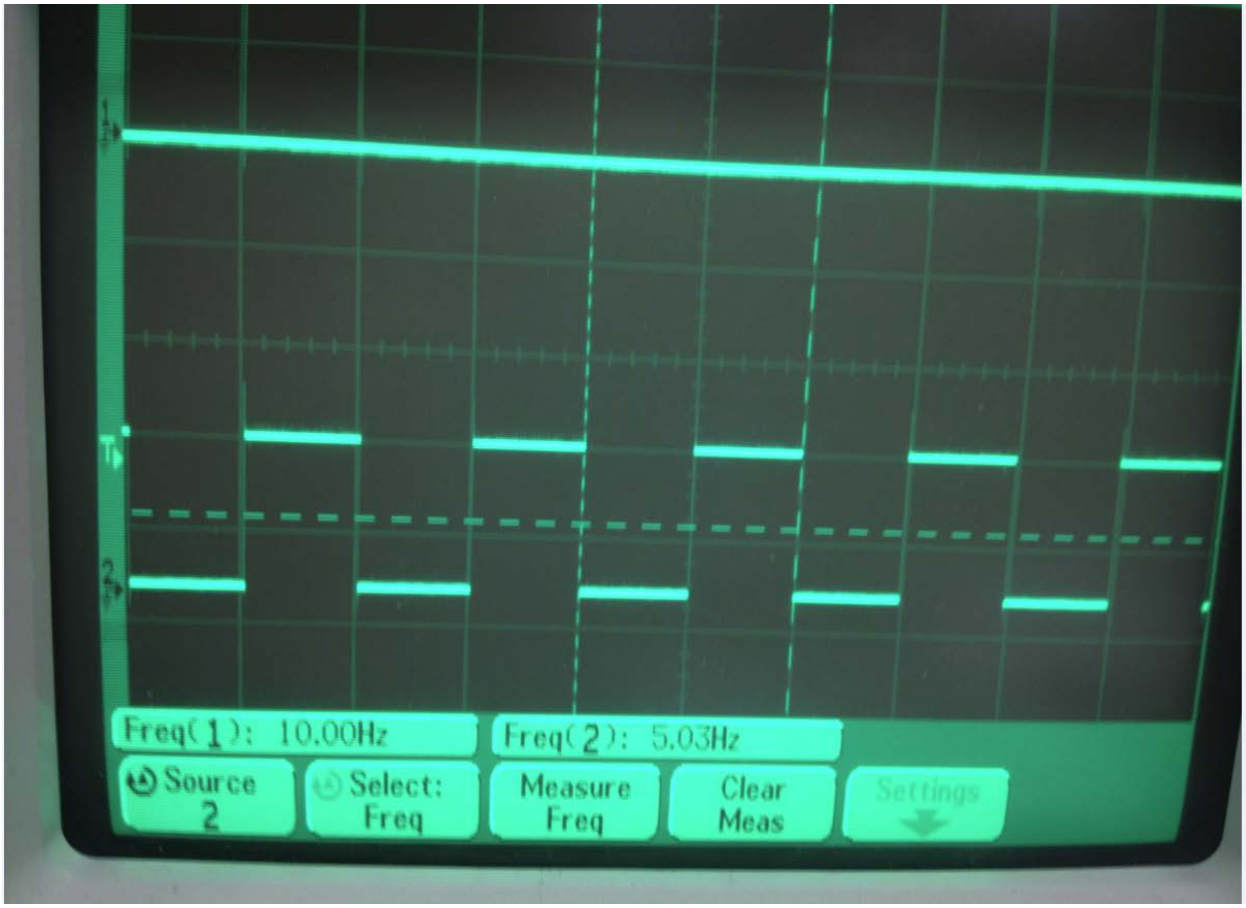
Programmable Timer verilog:

Below is the verilog code and the UCF file. Notice how we have an input value of 4,950,000 instead of 5,000,000. This is so we get closer to 10Hz on the zero output.

```
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'd4950000, counter, zero, tp);


endmodule
```

.UCF file for programmable timer

```
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
```



Verilog for the timer testbench:

```
module testbench_counter(
                    input SysClk,
                    input Start,
                    input Stop,
                    input Reset,
                  output tp, zero_out,
                  output  AN0, AN1, AN2, AN3,
                  output DP, Ca , Cb , Cc , Cd  ,Ce , Cf ,Cg,
                  output Q1 , Q0
                     );
                  wire [3:0] Min_Ones;
                   wire[3:0] Sec_Tens;
                   wire [3:0] Sec_Ones;
                   wire [3:0] Sec_Tenths;
                  wire CEn , CE_not ,zero;
                  wire [23:0] counter;
                  wire tp_fake , zero_fake;
```

```
        wire Dp0, Dp1, Dp2, Dp3;
        assign Dp3 = 1'b1;
        assign Dp2 = 1'b0;
        assign Dp1 = 1'b1;
        assign Dp0 = 1'b0;

        SR_latch latch(Stop , Start, CEn , CE_not);
        prog_timer timer(SysClk, Reset, CEn , 24'd4950000, counter, zero, tp);


        CounterBlock counter_block(zero , Reset , SysClk , Sec_Tenths , Sec_Ones , Sec_Tens, Min_Ones);
        SegmentController4x7 Seg_Ctr(Min_Ones , Sec_Tens , Sec_Ones , Sec_Tenths, SysClk , 1'b0 ,Dp0 , Dp1, Dp2, Dp3, Ca , Cb , Cc , Cd ,
Ce , Cf , Cg , AN0 , AN1 ,AN2 , AN3, DP ,Q1 , Q0 , tp_fake , zero_fake );
        assign zero_out = zero;
endmodule
```

UCF file lines for the timer testbench:

```
NET SysClk   LOC = "B8"; # Bank = 0, Pin name = IP_L13P_0/GCLK8, Type = GCLK, Sch name = GCLK0

NET Reset LOC = "B18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN0
NET Start LOC = "E18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN2
NET Stop LOC = "H13"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN3

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
```

Anomalies

-During the lab I didn't have any anomalies until I started to put together all the modules. An example of some things that I had go wrong were my mod6 and mod10 modules; I forgot to make the output. I had a reg that I was using to see the current state that it was in. I also had some problems in my timer testbench. For some reason I was having issues with my 4x7. I eventually found that I had some mix-up with my variables for my inputs to the 4x7 segment controller. I fixed those and then my timer started to function properly. I had no problems with any logic, thanks to the simulations that I did.