Colt Thomas

March 26, 2014

Lab 10 – Response Game Components

Preparation:

-Design the circuit for a 13- bit adder

Simple. Here is the code:

```
module adder_13bit(
    input [12:0] Ain,
    input [12:0] Bin,
    output [12:0] Sum
    );

                assign Sum = (Ain + Bin);
endmodule
```

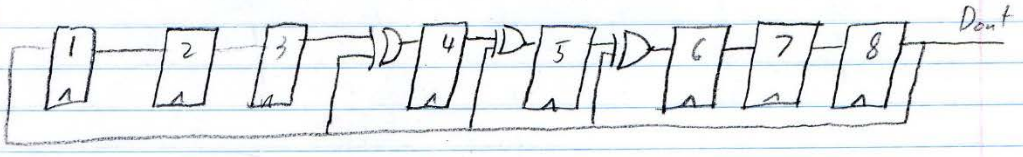
-Design the 8-bit LFSR

-Design the 13-bit down-count schematic

I did the work on paper on the following page…

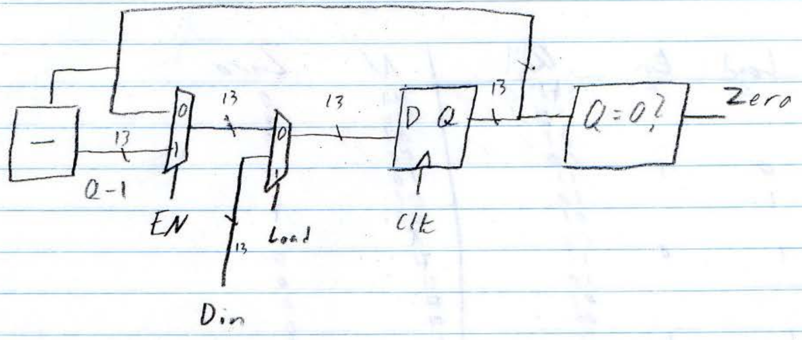
8 Bit LFSR schematic:

- Below is the way I implemented my LFSR:



13 Bit Counter Schematic

- I plan on implementing my logic like so.



- I have a "−" module and a "Q=0?" module that will decreme
and tell the user when Q=0 respectively. I plan on using beha
verilog to do these operations;

• Behavioral method

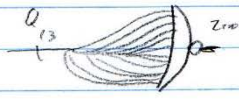
```
// Decrement module
    Q <= Q-1;
```

• Backup: Dataflow

• We can also use the "−"
dataflow

```
// Zero module
  if (Q == 13'b0)
      Zero <= 1
```

⇒



• Nand 13 can be use

- If these don't work I will use dataflow. The zero module can also
be a NAND gate.

Procedure:

• 8-bit LFSR Verilog module and TCL file (3)

```
        module LFSR_8bit(q, clk , reset);
input clk,reset;
```

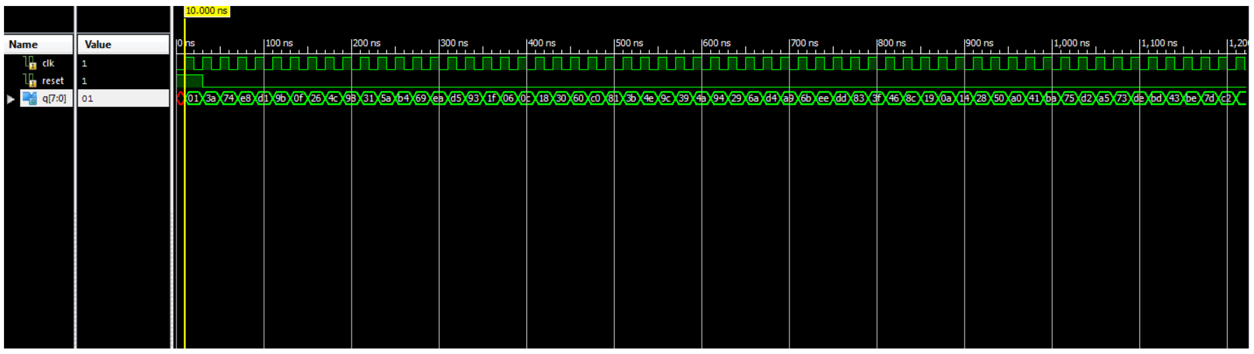
```
 output reg[7:0] q;

always @ (posedge clk)

if (reset)
		q[7:0] <= 8'b00000001;
else
		begin
		q[0] <= q[7];
		q[1] <= q[0];
		q[2] <= q[1];
		q[3] <= q[2] ^ q[0];
		q[4] <= q[3] ^ q[0];
		q[5] <= q[4] ^ q[0];
		q[6] <= q[5];
		q[7] <= q[6];

		end
endmodule
```

```
wave add / -radix hex

isim force add reset 1 -time 0 -value 0 -time 30ns
isim force add clk 0 -time 0 -value 1 -time 10ns -repeat 20 ns

run 99900ns
```

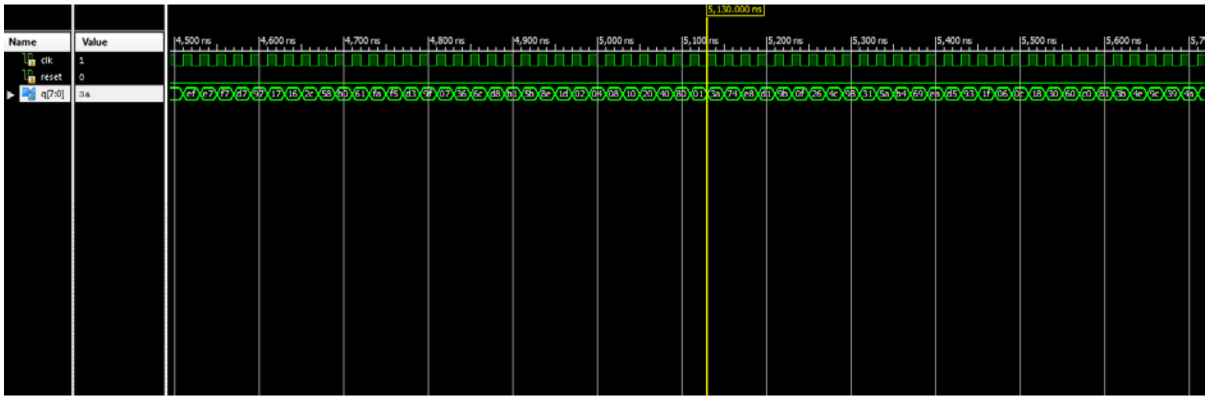
• 8-bit LFSR simulation waveform (4)

We have the .tcl file simulate from 0 – 9900ns. The clock should repeat itself after 20ns * 255 = 5100ns

-start point at 10ns



```
Name     Value
clk      1
reset    1
q[7:0]   01

q[7:0]: 01 3a 74 e8 d1 9b 0f 26 4c 98 31 5a b4 69 ea d5 93 1f 06 0c 18 30 60 c0 81 3b 4e 9c 39 4a 94 29 6a d4 a9 6b ee dd 83 3f 46 8c 19 0a 14 28 50 a0 41 ba 75 d2 a5 73 de bd 43 be 7d c2
```



Check it out. The church is true.

• 13-bit down-counter Verilog module and TCL file (3)

Below is the Verilog file using behavioral Verilog:

```
module count_down_13bit(
   input [12:0] Din,
   input Load,
   input EN,
                input Clk,
   output reg Zero
   );

reg[12:0] current;
always @ (posedge Clk)
if (Load)
               begin
                              current <= Din;
                              Zero <= 0;
               end
else if (current == 13'b0000000000000)
               begin
                              Zero <= 1;
               end
else if (EN && !Load)
               begin
                              current <= current - 1;
               end

endmodule
```

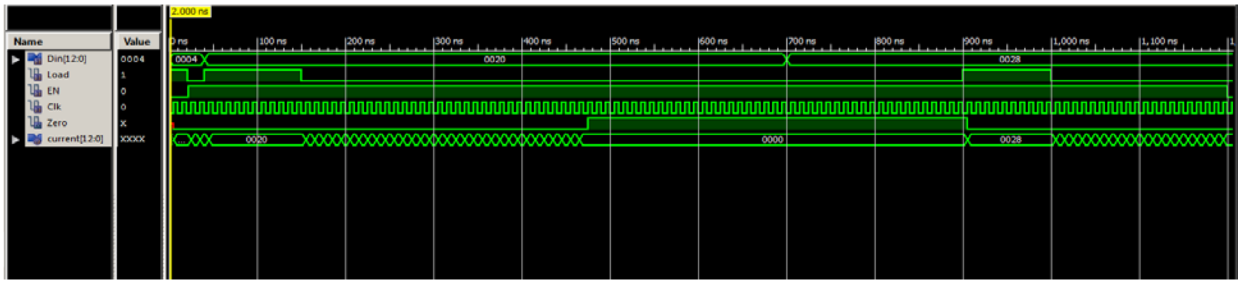
Below is the .tcl file:

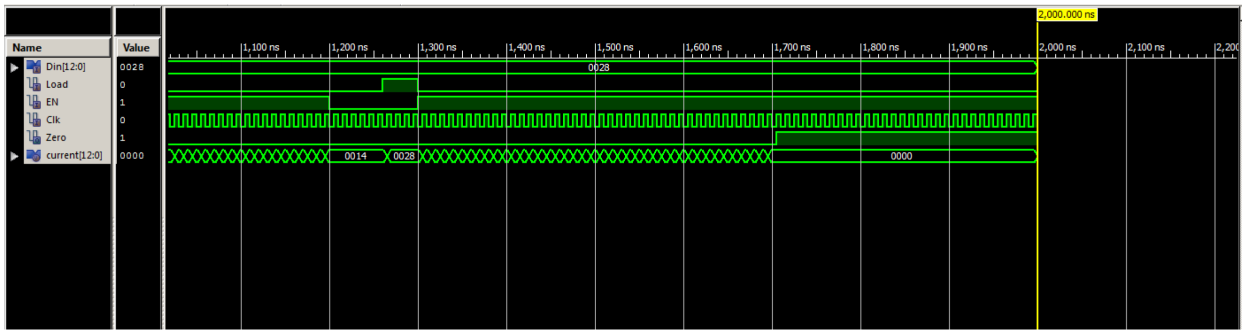
```
wave add / -radix hex

isim force add Din 0000000000100 -time 0 -value 0000000100000 -time 40ns -value 0000000101000 -time 700ns
isim force add Load 1 -time 0 -value 0 -time 21ns -value 1 -time 40ns -value 0 -time 150ns -value 1 -time 900ns -value 0 -time 1000ns -value 1 -
time 1260ns -value 0 -time 1300ns
isim force add EN 0 -time 0 -value 1 -time 22ns -value 0 -time 1200ns -value 1 -time 1300ns
isim force add Clk 0 -time 0 -value 1 -time 5ns -repeat 10 ns
```

run 2000ns

• 13-bit down-counter simulation waveform (4)

Below we have the simulation of our 13 bit counter. We enabled the clock to go off after 10ns. The zero output was asserted when the current value wire was 0000. We also ensured that when the Load was enabled the Din became the current value, and it didn't decrement until it was deasserted (whether EN was asserted or not).





• Delay timer Verilog module and TCL file (3)

Here is the Verilog code for the Delay Timer:

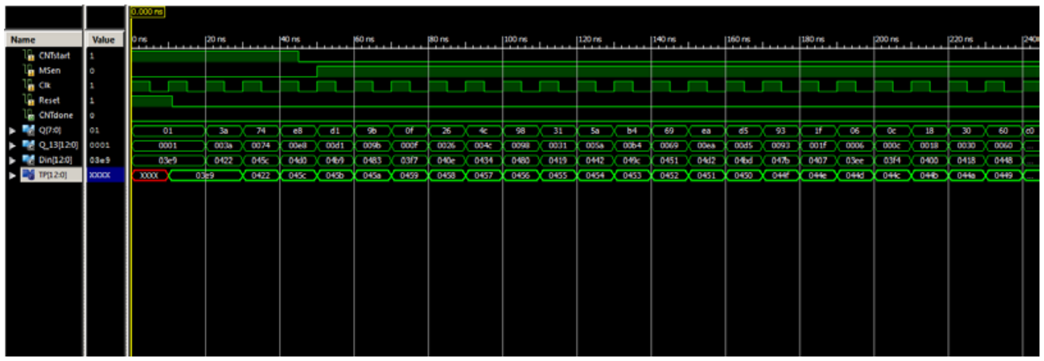
```
module Delay_Timer(
   input CNTstart,
   input MSen,
               input Clk,
               input Reset,
   output CNTdone
   );

               wire [7:0] Q;
               wire [12:0] Q_13, Din;
               wire [12:0] TP;          //this is a test point to see where the counter is at
LFSR_8bit LFSR(Q , Clk , Reset);
shift4 shift(Q , Q_13);
adder_13bit adder(Q_13 , 13'd1000 , Din);
count_down_13bit counter(Din , CNTstart , MSen , Clk , CNTdone, TP);
endmodule
```

Here is out .tcl file:

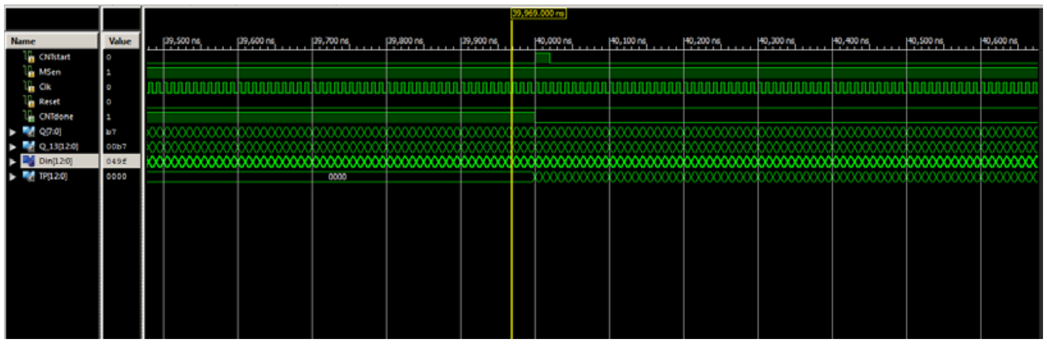
```
wave add / -radix hex

isim force add Clk 1 -time 0 -value 0 -time 5ns -repeat 10ns
isim force add CNTstart 1 -time 0 -value 0 -time 45ns -value 1 -time 40000ns -value 0 -time 40020ns
isim force add MSen 0 -time 0 -value 1 -time 50ns -value 0 -time 100ns -value 1 -time 200ns
isim force add Reset 1 -time 0 -value 0 -time 11ns
run 200000ns
```

• Delay timer simulation waveform (5)

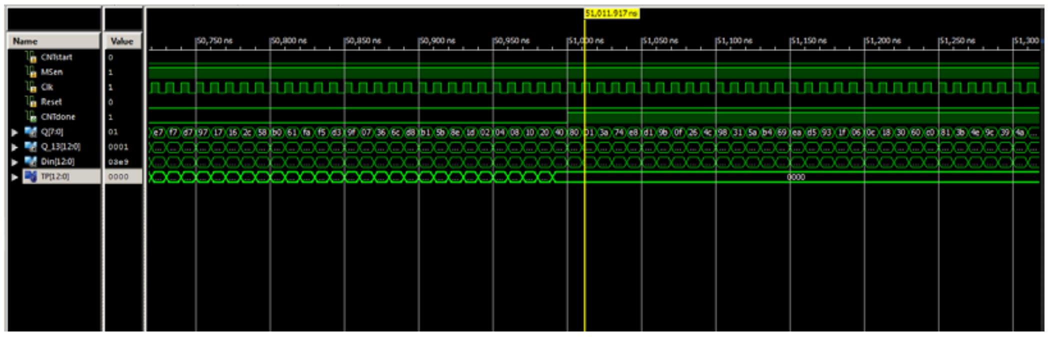
For our simulation we wanted to make sure that the CNTstart would load whatever random value was asserted for Din, but wouldn't decrement. We also tested to see if MSen would begin the decrement. We were able to attain the function as seen below.



Since the random number was rather large, we had to wait a long time to see the Zero output asserted, which it eventually did. We tested to see if we could load another value….



…and see if it reached zero again. We were successful.



• Scoring unit Verilog module and TCL file

Here is my Verilog code that I used for the scoring unit:

```
module Scoring_Unit(
   output [15:0] SCout,
   input SCupdate,
   input [15:0] SCin,
   input BESTSCORE,
                input Clk,
                input reset
   );

               wire [15:0] Q_Last , Q_Best;

register_16bit Last_time_register(Q_Last,SCin,SCupdate,Clk,reset);

assign Q_Best = (reset==1'b1)?16'h9999:
                (Q_Last < Q_Best)?Q_Last:
                Q_Best;

assign SCout = (BESTSCORE==1'b0)?Q_Last:
                         Q_Best;
endmodule
```

This is the register_16bit that was used to implement the 16 bit register; I used the FF_DCE that was given to implement each bit:

```
module register_16bit(
   output [15:0] Q,
   input [15:0] D,
   input EN,
   input Clk,
                input reset
   );

               wire [15:0] current_state;

               FF_DCE registerbit_0(current_state[0] ,Clk , D[0] ,1'b0 , EN);
               FF_DCE registerbit_1(current_state[1] ,Clk , D[1] ,1'b0 , EN);
               FF_DCE registerbit_2(current_state[2] ,Clk , D[2] ,1'b0 , EN);
               FF_DCE registerbit_3(current_state[3] ,Clk , D[3] ,1'b0 , EN);
               FF_DCE registerbit_4(current_state[4] ,Clk , D[4] ,1'b0 , EN);
               FF_DCE registerbit_5(current_state[5] ,Clk , D[5] ,1'b0 , EN);
               FF_DCE registerbit_6(current_state[6] ,Clk , D[6] ,1'b0 , EN);
               FF_DCE registerbit_7(current_state[7] ,Clk , D[7] ,1'b0 , EN);
               FF_DCE registerbit_8(current_state[8] ,Clk , D[8] ,1'b0 , EN);
               FF_DCE registerbit_9(current_state[9] ,Clk , D[9] ,1'b0 , EN);
               FF_DCE registerbit_10(current_state[10] ,Clk , D[10] ,1'b0 , EN);
```

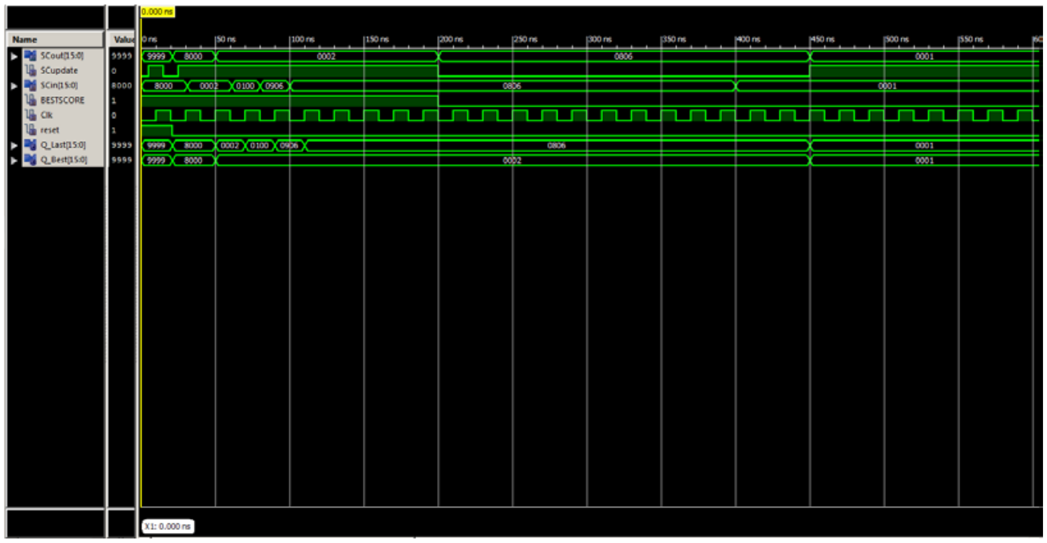
```
        FF_DCE registerbit_11(current_state[11] ,Clk , D[11] ,1'b0 , EN);
        FF_DCE registerbit_12(current_state[12] ,Clk , D[12] ,1'b0 , EN);
        FF_DCE registerbit_13(current_state[13] ,Clk , D[13] ,1'b0 , EN);
        FF_DCE registerbit_14(current_state[14] ,Clk , D[14] ,1'b0 , EN);
        FF_DCE registerbit_15(current_state[15] ,Clk , D[15] ,1'b0 , EN);


        assign Q = (reset == 1'b1)?16'h9999:
                current_state;
endmodule
```

This is my .tcl script:

```
wave add / -radix hex

isim force add BESTSCORE 1 -time 0 -value 0 -time 200ns

isim force add reset 1 -time 0 -value 0 -time 21ns

isim force add SCin 1000000000000000  -time 0 -value 0000000000000010 -time 31ns -value 0000000100000000 -time 61ns -value 0000100100000110 -time 80ns -value 0000100000000110 -time 100ns -value 0000000000000001 -time 400ns

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

isim force add SCupdate 0 -time 0 -value 1 -time 5ns -value 0 -time 15ns -value 1 -time 25ns -value 0 -time 200ns -value 1 -time 450ns


run 800ns
```

• Scoring unit simulation

We implemented a test to see if the best score would be saved and if the SCupdate input updated the best score and the SCout. We had positive results below:



Anomalies:

A majority of my anomalies had to do with the way I was implementing my .tcl files. There were no bugs with the .tcl files, yet I would adjust them to try to get the results that I wanted at times, and other times I would try to test only specific outputs within a module rather than the whole thing altogether. This caused me to think there were problems with my Verilog that weren't actually there. I eventually fixed the .tcl files to test what I needed to see and then I could track down the real bugs. I also implemented my 13 bit counter using behavioral Verilog. I had some syntax errors that were hard to catch, but after so long it was better for me to just finish it in behavioral to save time. I got it to work and now I have a better feel for it. As far as suggestions go, I don't have any for this lab.