System Synthesis and Modeling (CSCE 3953)

Project 5

Zack Fravel

10/6/16

zpfravel@uark.edu

Parent Objective

The parent objective of project 5 was to be introduced to and get familiar with the synthesis tool Design Vision. Once we finished the tutorial and were familiar with the flow of how to synthesize a behavioral design, we were assigned to synthesize our previous bitCounter and bcdCounter modules and show a simulation with the gate delay added from the synthesis process (post synthesis timing simulation).

Synthesis process / Constraints added

The synthesis process for both of my designs, once I had a handle on the tool and set up the necessary SAED 90 nm library, went fairly smoothly in general. After creating the necessary libraries on my turing account I used them to house all of my source files. I uploaded my behavioral code for bitCounter and bcdCounter to the server using an ssh file transfer client. Once my designs were on the server, I was able to read them into Design Vision and compile the design. I then saved these to a separate folder with the name "synthesis-" added to the front of each in order to differentiate them from the original behavioral designs. Both the behavioral code and the synthesized code is included at the end of the report.

Once the designs were compiled, it was time to set the timing constraints. For the bitCounter, since it is a combinational circuit there is no clock input; all that was needed was to set the input and output delay using the commands set_input_delay 2.0 -max [all_inputs] and set_output_delay 0.5 -max [all_outputs]. For the bcdCounter, our sequential circuit, before those two commands I use the commands create_clock -name "CLK_0" -period 30 -waveform {0.000 15.000} {clk} and set_clock_uncertainty 0.14 CLK_0 to instantiate the clock and set the skew.

After that, the last thing that needs to be done before we can save our changes and simulate is set the design constraints and run a timing report. For both designs, we input the following four commands to set our design constraints.

- set_max_area 1000.000000
- set_max_dynamic_power 0.2 uW
- set_max_leakage_power 0.005 uW
- set_max_total_power 0.5 uW

After that is saved, we run a timing report to make sure our constraints meet the design requirements. For both my designs, no changes needed to be made to meet the timing requirement. Below are the timing constraint reports for bitCounter and bcdCounter respectively.

```
Operating Conditions: TYPICAL   Library: saed90nm_typ
Wire Load Model Mode: enclosed

  Startpoint: in[4] (input port)
  Endpoint: out[2] (output port)
  Path Group: (none)
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  bitCounter         8000                  saed90nm_typ

  Point                                    Incr       Path
  ----------------------------------------------------------
  input external delay                     0.00       0.00
  in[4] (in)                               0.00       0.00
  U60/Q (XOR2X2)                           0.13       0.13
  U52/Q (AO22X2)                           0.15       0.29
  U51/Q (XOR2X2)                           0.15       0.44
  U49/QN (NAND2X0)                         0.09       0.53
  U40/Q (ISOLANDX1)                        0.10       0.63
  U44/QN (NAND3X0)                         0.08       0.71
  U43/Q (XNOR2X2)                          0.16       0.87
  out[2] (out)                             0.00       0.87
  data arrival time                                   0.87
  ----------------------------------------------------------
  (Path is unconstrained)


***** End Of Report *****
```

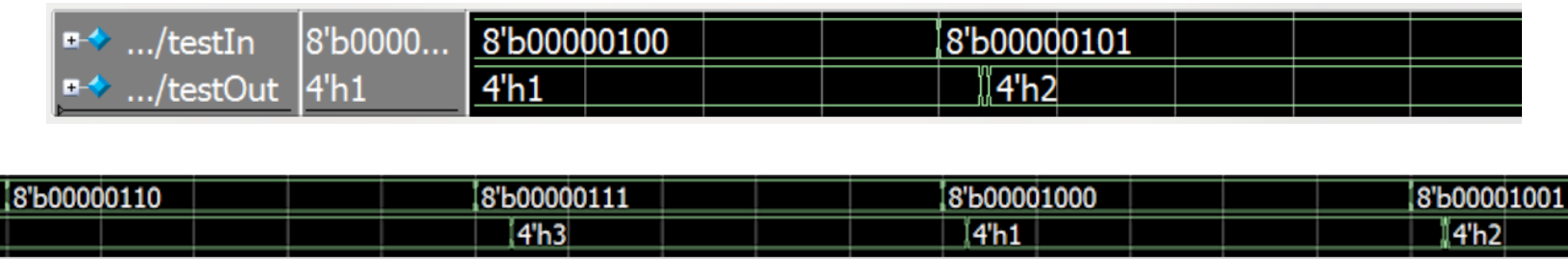
```
Startpoint: rst (input port clocked by CLK_0)
Endpoint: out_reg[1] (rising edge-triggered flip-flop clocked by
Path Group: CLK_0
Path Type: max

Des/Clust/Port     Wire Load Model       Library
------------------------------------------------
bcdCounter         ForQA                 saed90nm_typ

Point                                    Incr       Path
----------------------------------------------------------
clock CLK_0 (rise edge)                  0.00       0.00
clock network delay (ideal)              0.00       0.00
input external delay                     2.00       2.00 f
rst (in)                                 0.00       2.00 f
U31/Q (AO21X2)                           0.53       2.53 f
U24/QN (INVX8)                           0.34       2.87 r
U39/Q (AND2X2)                           0.32       3.19 r
U37/QN (INVX2)                           0.43       3.62 f
U38/QN (INVX2)                           0.49       4.11 r
U35/Q (AND2X1)                           0.48       4.59 r
U32/Q (ISOLORX2)                         0.32       4.91 r
out_reg[1]/D (DFFX2)                     0.00       4.92 r
data arrival time                                   4.92

clock CLK_0 (rise edge)                 30.00      30.00
clock network delay (ideal)              0.00      30.00
clock uncertainty                       -0.14      29.86
out_reg[1]/CLK (DFFX2)                   0.00      29.86 r
library setup time                      -0.14      29.72
data required time                                 29.72
----------------------------------------------------------
data required time                                 29.72
data arrival time                                  -4.92
----------------------------------------------------------
slack (MET)                                        24.81


***** End Of Report *****
```

Simulation Method

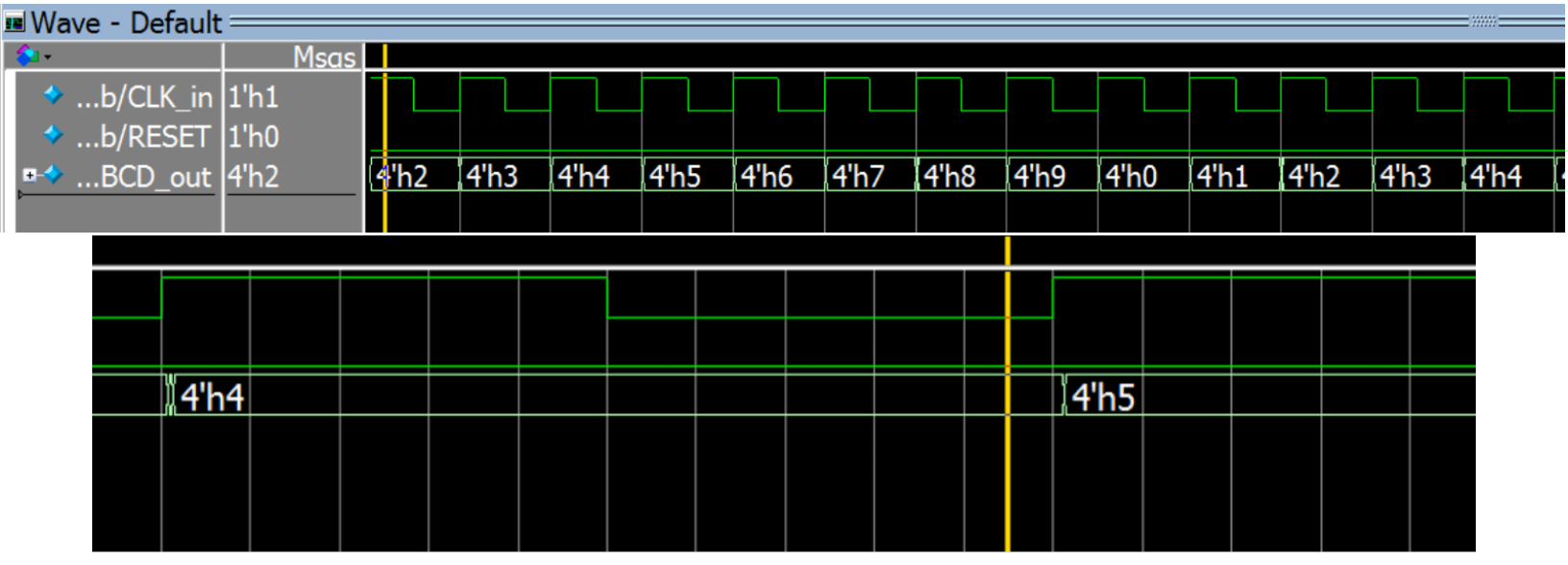
Finally, after checking the timing constraints I was able to save all the changes are start work on the simulation. The first step in simulation with the synthesis code was to obtain the 90 nm library we were to use from the turing server as well as the synthesized design files that we saved from Design Vision and compile all the individual gate design files into our new Modelsim library "work_syn." With the 90nm library compiled in Modelsim, I compiled the synthesized designs and the old test benches into the work_syn library. From here I was able to run the simulator on the test benches.

Result

For my bitCounter results, I had to zoom in fairly far to be able to observe the noticeable gate delay in the new design. Below is the waveform screenshot.



The bcdCounter also synthesized without issue. Below are the simulation screenshots with the gate delay for the bcdCounter, the delay is very small.



Analysis and Conclusion

Overall, after the initial battle with turing and the Design Vision tool to get it ready for synthesis, the actual process of synthesizing a behavioral design and running a new gate-delay simulation is a very straight forward process. Now we are able to observe something closer to real life behavior with a design that I made myself! All of the source code and test benches mentioned in the report are attached.

Source Code

bitCounter (behavioral)

```
Ln#
 1  // Zack Fravel
 2  // System Synthesis and Modeling
 3  // Project 3
 4
 5  module bitCounter(
 6          input wire [7:0] in,
 7          output wire [3:0] out
 8  );
 9
10  assign out = in[7] + in[6] + in[5] + in[4] + in[3] + in[2] + in[1] + in[0];
11
12  endmodule
```

bcdCounter (behavioral)

```
 1  // Zack Fravel
 2  // System Synthesis and Modeling
 3  // Project 4
 4
 5  `timescale 1ns/1ns
 6  module bcdCounter(
 7          input wire clk,
 8          input wire rst,
 9          output reg [3:0] out
10     );
11
12
13  initial   out = 5;              // Set inital output value to 5
14
15     always @ (posedge (clk))
16     begin
17
18          if(rst)
19             out <= 0;
20          else
21             if (out < 9)
22                  out <= out + 1;
23             else
24                  out <= 0;
25     end
26
27
28
29  endmodule
```

bitCounter (synthesized)

```
module bitCounter ( in, out );
  input [7:0] in;
  output [3:0] out;
  wire   n47, n48, n49, n50, n51, n52, n53, n54, n55, n56, n57, n58, n59, n60,
         n61, n62, n63;

  ISOLANDX1 U40 ( .D(n53), .ISO(n47), .Q(n52) );
  NOR2X1 U41 ( .IN1(n58), .IN2(n59), .QN(n47) );
  ISOLANDX1 U42 ( .D(n48), .ISO(n49), .Q(out[3]) );
  XNOR2X2 U43 ( .IN1(n48), .IN2(n49), .Q(out[2]) );
  NAND3X0 U44 ( .IN1(n50), .IN2(n51), .IN3(n52), .QN(n49) );
  NAND2X0 U45 ( .IN1(n53), .IN2(n54), .QN(n48) );
  NAND2X0 U46 ( .IN1(n55), .IN2(n56), .QN(n54) );
  XNOR2X2 U47 ( .IN1(n52), .IN2(n57), .Q(out[1]) );
  NAND2X0 U48 ( .IN1(n50), .IN2(n51), .QN(n57) );
  NAND2X0 U49 ( .IN1(n59), .IN2(n58), .QN(n53) );
  AO22X2 U50 ( .IN1(in[6]), .IN2(in[7]), .IN3(n60), .IN4(n61), .Q(n58) );
  XOR2X2 U51 ( .IN1(n55), .IN2(n56), .Q(n59) );
  AO22X2 U52 ( .IN1(in[3]), .IN2(in[4]), .IN3(in[5]), .IN4(n62), .Q(n56) );
  AO22X2 U53 ( .IN1(in[0]), .IN2(in[1]), .IN3(in[2]), .IN4(n63), .Q(n55) );
  XOR2X2 U54 ( .IN1(n51), .IN2(n50), .Q(out[0]) );
  XOR2X2 U55 ( .IN1(n61), .IN2(n60), .Q(n50) );
  XOR2X2 U56 ( .IN1(in[6]), .IN2(in[7]), .Q(n60) );
  XOR2X2 U57 ( .IN1(in[2]), .IN2(n63), .Q(n61) );
  XOR2X2 U58 ( .IN1(in[0]), .IN2(in[1]), .Q(n63) );
  XOR2X2 U59 ( .IN1(in[5]), .IN2(n62), .Q(n51) );
  XOR2X2 U60 ( .IN1(in[3]), .IN2(in[4]), .Q(n62) );
endmodule
```

bcdCounter (synthesized)

```
module bcdCounter ( clk, rst, out );
  output [3:0] out;
  input clk, rst;
  wire   n34, n35, n62, n36, N11, N12, n28, n37, n38, n39, n40, n41, n42, n43,
         n44, n45, n49, n50, n51, n52, n53, n54, n56, n58, n59, n60;
  assign out[3] = n34;

  DFFX2 \out_reg[0]  ( .D(n37), .CLK(clk), .Q(n36), .QN(n28) );
  DFFX2 \out_reg[1]  ( .D(n50), .CLK(clk), .Q(n62), .QN(n60) );
  DFFX2 \out_reg[2]  ( .D(N11), .CLK(clk), .Q(n35), .QN(n56) );
  DFFX2 \out_reg[3]  ( .D(N12), .CLK(clk), .Q(n34), .QN(n58) );
  NAND3X4 U16 ( .IN1(n36), .IN2(n39), .IN3(out[1]), .QN(n43) );
  NAND3X4 U21 ( .IN1(n40), .IN2(n41), .IN3(n42), .QN(n49) );
  INVX8 U24 ( .IN(n45), .QN(n39) );
  AO22X2 U28 ( .IN1(n39), .IN2(n34), .IN3(out[2]), .IN4(n38), .Q(N12) );
  OAI22X2 U29 ( .IN1(n35), .IN2(n43), .IN3(n44), .IN4(n42), .QN(N11) );
  OA21X1 U30 ( .IN1(out[1]), .IN2(n45), .IN3(n54), .Q(n44) );
  AO21X2 U31 ( .IN1(n59), .IN2(n49), .IN3(rst), .Q(n45) );
  ISOLORX2 U32 ( .D(n52), .ISO(n51), .Q(n50) );
  AND3X2 U33 ( .IN1(n39), .IN2(n41), .IN3(out[0]), .Q(n51) );
  INVX2 U34 ( .IN(n35), .QN(n42) );
  AND2X1 U35 ( .IN1(n37), .IN2(n62), .Q(n52) );
  INVX2 U36 ( .IN(n62), .QN(n41) );
  INVX2 U37 ( .IN(n53), .QN(n54) );
  INVX2 U38 ( .IN(n54), .QN(n37) );
  AND2X2 U39 ( .IN1(n39), .IN2(n40), .Q(n53) );
  INVX2 U40 ( .IN(n40), .QN(out[0]) );
  INVX4 U41 ( .IN(n36), .QN(n40) );
  INVX2 U42 ( .IN(n56), .QN(out[2]) );
  AOINVX2 U43 ( .IN(n58), .QN(n59) );
  AOINVX4 U44 ( .IN(n60), .QN(out[1]) );
  AOINVX2 U45 ( .IN(n43), .QN(n38) );
endmodule
```

bitCounter testbench

```
Ln#
  1 // Zack Fravel
  2 // System Synthesis and Modeling
  3 // Project 3
  4
  5 `timescale 1ns/1ns
  6 module bitCounter_tb;
  7
  8 // Inputs
  9         reg[7:0] testIn;
 10 // Output
 11         wire[3:0] testOut;
 12 // Instantiate Module
 13         bitCounter test (testIn, testOut);
 14
 15 // Testbench
 16 initial
 17
 18    begin
 19
 20         #20;  // Wait 20ns and assign input to all 0's
 21         testIn = 8'b00000000;
 22
 23         while(testIn < 256)
 24         begin
 25            $display("input: ", testIn, "   output:", testOut);
 26            testIn = testIn + 1;
 27            #5;
 28         end
 29
 30    end
 31
 32 endmodule
```

bcdCounter testbench

```
1  // Zack Fravel
2  // System Synthesis and Modeling
3  // Project 4
4
5  `timescale 1ns/1ns
6  module bcdCounter_tb;
7
8  // Inputs
9          reg CLK_in;
10         reg RESET;
11 // Output
12         wire[3:0] BCD_out;
13 // Instantiate Module
14         bcdCounter test (CLK_in, RESET, BCD_out);
15 // Testbench
16 initial
17
18    begin
19         #20;  // Wait 20ns
20         CLK_in = 0;
21         RESET = 0;
22         #20;  // Wait 20ns
23    end
24
25 initial
26
27    begin
28         #180;
29         RESET = 1;
30         #10;
31         RESET = 0;
32    end
33
34 always #10 CLK_in = ~CLK_in;
35
36 endmodule
37
```