University of Miami

EEN 414 Computer Organization and Design

EXPERIMENT #4. Divider

Submitted by: Brandon Rubio, Isabela Bandrich, Rainier Young, Nikeem Dunkelly-Allen

On: 10/28/22

to : Dr. Onur Tigli

Objective

In this experiment, you are required to apply datapath and control unit design techniques in Verilog HDL to design and optimize a division algorithm. Coding, verification, simulation, synthesis and timing analysis will be carried out using Xilinx ISE tools. Verified designs will be implemented on Digilent's NEXYS-4 FPGA boards for demonstration of correct operation.

Verilog HDL and Testbench Code

MAIN CODE:

Counter:

```
always@(posedge clk)
	begin
		if(clr)
			count <= 0;
		else
		begin
			if(increment)
				count <= count + 1'b1;
			else
				count <= count;
		end
	end
endmodule
```

Controller:

```
33      begin
34        if(reset) state<=S0;
35        else begin
36           case(state)
37              S0: state<=S1;
38              S1: begin
39                    if(sub_count) state<=S3;
40                    else state<=S2;
41                  end
42              S2: begin
43                    if(counter==nBit-1) state<=S5;
44                    else state<=S4;
45                  end
46              S3: begin
47                    if(counter==nBit-1) state<=S5;
48                    else state<=S4;
49                  end
50              S4: state<=S1;
51              S5: state<=S5;
52              default: state<=S0;
53           endcase
54        end
55      end
```

```
70          end
71          else if(state==S2) begin
72             R_Load<=1'b0;
73             R_ShiftLeft<=1'b0;
74             Q_ShiftLeft<=1'b1;
75          end
76          else if(state==S3) begin
77             R_Load<=1'b1;
78             R_ShiftLeft<=1'b0;
79             Q_ShiftLeft<=1'b1;
80          end
81          else if(state==S4) begin
82             A_ShiftLeft<=1'b1;
83             R_Load<=1'b0;
84             R_ShiftLeft<=1'b0;
85             Q_ShiftLeft<=1'b0;
86          end
87          else if(state==S5) begin
88             R_Load<=1'b0;
89             Q_ShiftLeft<=1'b0;
90          end
91       end
92    endmodule
```

Datapath:

```
30      output reg [nBit-1:0] Q
31      );
32      parameter nBit = 16;
33      wire [3:0] count;
34      wire [nBit-1:0] A_output, B_output, R_output, Q_output;
35      wire [nBit-1:0] Subtract;
36      wire [nBit-1:0] is_empty;
37      assign is_empty = 0;
38
39      Counter bit_count(A_ShiftLeft, clk, reset, count);
40      Shift_Registers #(nBit) ra(A, clk, AB_Load, 1'b0, A_ShiftLeft, 1'b0, 1'b0, A_output);
41      Shift_Registers #(nBit) rb(B, clk, AB_Load, 1'b0, 1'b0, 1'b0, 1'b0, B_output);
42      Shift_Registers #(nBit) rr(Subtract, clk, R_Load, AB_Load, R_ShiftLeft, 1'b0, A_output[nBit-1], R_output);
43      N_Subtractor #(nBit) sub(R_output, B_output, Subtract, sub_count);
44      Shift_Registers #(nBit) rq(is_empty, clk, 1'b0, AB_Load, Q_ShiftLeft, 1'b0, R_Load, Q_output);
45
46      assign shift_count = count;
```

```
52          Q<=Q_output;
53        end
54        else begin
55          R<=1'b0;
56          Q<=1'b0;
57        end
58      end
59
60  endmodule
```

Divider:

```
27      wire [nBit-1:0] R_output;
28      wire [nBit-1:0] Q_output;
29      wire AB_Load, A_ShiftLeft;
30      wire R_Load, R_ShiftLeft;
31      wire Q_ShiftLeft;
32      wire sub_count;
33      wire [3:0] counter;
34
35      Controller #(nBit) cont(sub_count, clk, reset, counter, AB_Load, A_ShiftLeft, R_Load, R_ShiftLeft, Q_ShiftLeft);
36      Datapath #(nBit) data(A, B, clk, reset, AB_Load, A_ShiftLeft, R_Load, R_ShiftLeft, Q_ShiftLeft, counter, sub_count, R_output, Q_output);
37      assign R = R_output;
38      assign Q = Q_output;
39
40  endmodule
```

Shift Registers:

```
28
29      always @(negedge clk)
30      begin
31        if(clr) Output<=1'b0;
32        else begin
33           if(load) Output<=Input;
34           else if(Shift_Left) Output<={Output[nBit-2:0], Shift_In};
35           else if(Shift_Right) Output<={Shift_In, Output[nBit-1:1]};
36           else Output<=Output;
37        end
38      end
39  endmodule
```

Half Adder:

```
23      output S, Cout
24      );
25      xor(S, A, B);
26      and(Cout, A, B);
27
28  endmodule
```

Full Adder:

```
24      );
25      wire D, E, F;
26      Half_Adder ha1(A, B, D, E);
27      Half_Adder ha2(Cin, D, S, F);
28      or(Cout, E, F);
29
30  endmodule
```

N_Subtractor:

```
27      wire [nBit-1:0] Bin, B;
28      assign Bin = ~C + 1'b1;
29
30      genvar i;
31      generate
32        for(i=0;i<=nBit-1;i=i+1)
33           begin: generate_subtractor
34              if(i==0) Half_Adder ha(A[0], Bin[0], D[0], B[0]);
35              else Full_Adder fa(A[i], Bin[i], B[i-1], D[i], B[i]);
36           end
37           assign Bout = B[nBit-1];
38     endgenerate
39
40  endmodule
```

Seven Segment Decoder:

```
31              4'h2 : tmp = 8'b10010001;
32              4'h3 : tmp = 8'b01100001;
33              4'h4 : tmp = 8'b00110011;
34              4'h5 : tmp = 8'b00100101;
35              4'h6 : tmp = 8'b00000101;
36              4'h7 : tmp = 8'b11110001;
37              4'h8 : tmp = 8'b00000001;
38              4'h9 : tmp = 8'b00100001;
39              4'hA : tmp = 8'b00010001;
40              4'hB : tmp = 8'b00000111;
41              4'hC : tmp = 8'b10001101;
42              4'hD : tmp = 8'b01000011;
43              4'hE : tmp = 8'b00001101;
44              4'hF : tmp = 8'b00011101;
45              default : tmp = 8'b01111111;
46          endcase
47      end
48      assign sev_seg_leds[7:0] = tmp;
49  endmodule
```

Seven Segment Display with Clock:

```
30      assign led_disable = 4'b1111;
31
32      always @(posedge clk or posedge reset)
33      begin
34        if(reset) begin
35           sel<=3'b000;
36           led_enable<=4'b1111;
37        end
38        else begin
39           sel[2]<=sel[1]&sel[0];
40           sel[1]<=sel[1]^sel[0];
41           sel[0]<=~sel[0];
42           if(sel==3'b001) begin
43              led_enable<=4'b0111;
44              bus<=num_1;
45           end
```

```
52              bus<=num_3;
53           end
54           else if(sel==3'b100) begin
55              led_enable<=4'b1110;
56              bus<=num_4;
57           end
58        end
59     end
60
61     Sev_Seg_Decoder dec1(bus, sev_seg_leds);
62
63  endmodule
```

Seven Segment Display with Clock (Top Module):

```
    reg [14:0] clk_div;

    Divider #(8) d(num_1, num_2, clk_slw, reset, num_3, num_4);
    Sev_Seg_with_Clk dd(num_3[7:4], num_3[3:0], num_4[7:4], num_4[3:0], clk_div[14], reset, sev_seg_leds, led_disable, led_enable);
    ip_clk_div clk_5M(clk_main, clk_slw);

    always @(posedge clk_slw)
    begin
      if(reset) clk_div <= 15'd0;
      else clk_div <= clk_div + 15'd1;
    end

endmodule
```

TESTBENCH:

tb_Controller:

```
35    wire A_ShiftLeft;
36    wire R_Load;
37    wire R_ShiftLeft;
38    wire Q_ShiftLeft;
39
40    // Instantiate the Unit Under Test (UUT)
41    Controller uut (
42        .sub_count(sub_count),
43        .clk(clk),
44        .reset(reset),
45        .counter(counter),
46        .AB_Load(AB_Load),
47        .A_ShiftLeft(A_ShiftLeft),
48        .R_Load(R_Load),
49        .R_ShiftLeft(R_ShiftLeft),
50        .Q_ShiftLeft(Q_ShiftLeft)
51    );
```

```
55    end
56
57    initial begin //reset
58        reset = 1'b1; //S0 and S1
59        #10;
60        reset = 1'b0;
61        #10;
62    end
```

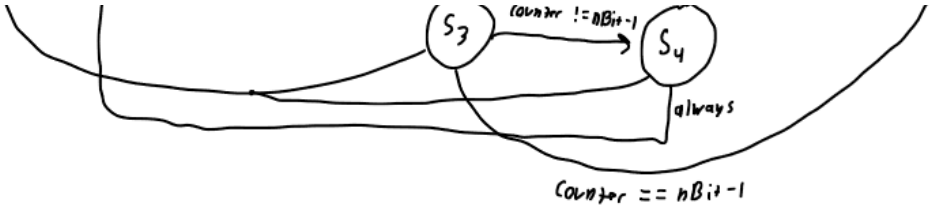
```
73
74          reset = 1'b1; //S0 and S1
75          #10;
76          reset = 1'b0;
77          #10;
78
79          #10; //S3
80          sub_count = 1;
81          counter = 0;
82          #20;
83
84          sub_count = 1; //S4
85          counter = 0;
86          #20;
87      end
88
89  endmodule
```

tb_Divider:

```
34      wire [15:0] R;
35      wire [15:0] Q;
36
37      // Instantiate the Unit Under Test (UUT)
38      Divider uut (
39          .A(A),
40          .B(B),
41          .clk(clk),
42          .reset(reset),
43          .R(R),
44          .Q(Q)
45      );
46      initial begin //clock
47          clk = 1'b0;
48          forever #1 clk = ~clk;
49      end
```

```
59          B = 20;
60          #200;
61
62          reset = 1'b1;
63          #10;
64          reset = 1'b0;
65
66          A = 55;
67          B = 7;
68          #200;
69      end
70
71  endmodule
```

ASM chart:



Control Signals

S_0: AB_load_QR_clr = 1
A_sl = 0
R_load_Q_shiftIn = 0
R_sl = 0
Q_sl = 0

S_1: AB_load_QR_clr = 0
A_sl = 0
R_sl = 1

S_2: R_load_Q_shiftIn = 0
A_sl = 0
Q_sl = 1

S_3: R_load_Q_shiftIn = 1
R_sl = 0
Q_sl = 1

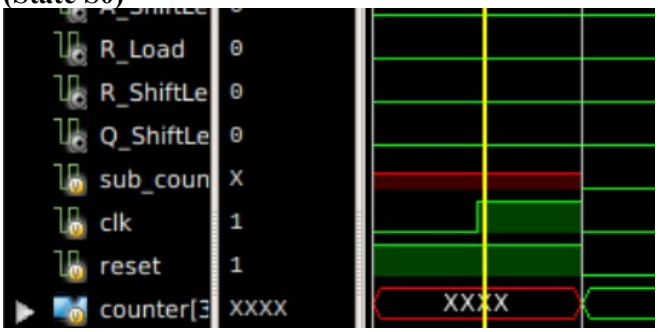
S_4: A_sl = 1
R_load_Q_shiftIn = 0
R_sl = 0
Q_sl = 0

S_5: R_load_Q_shiftIn = 0
Q_sl = 0

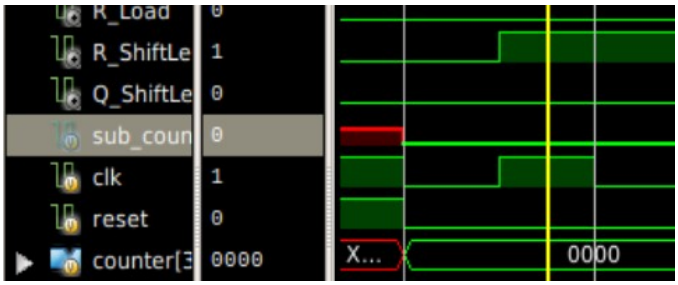
Simulation Results

Controller:

(State S0)

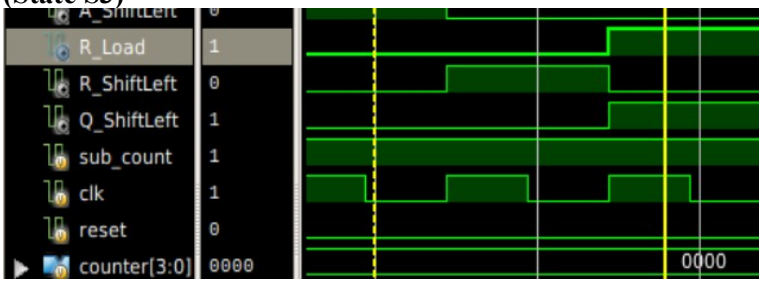


(State S1)

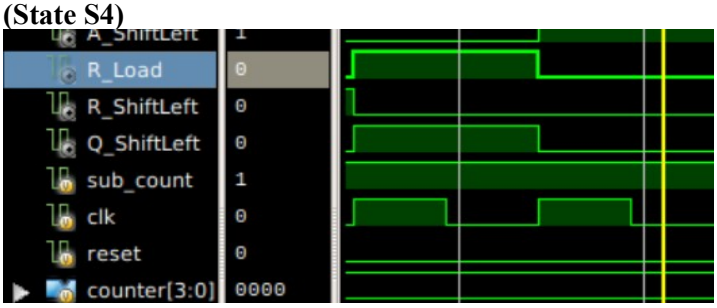


(State S2)

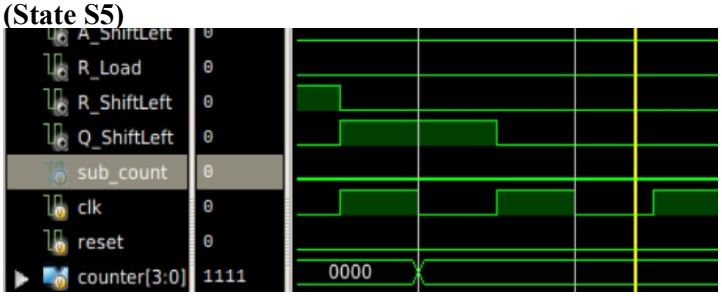
(State S3)



(State S4)

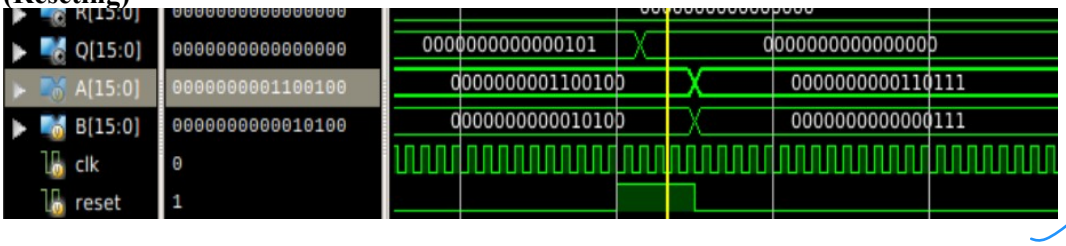


(State S5)

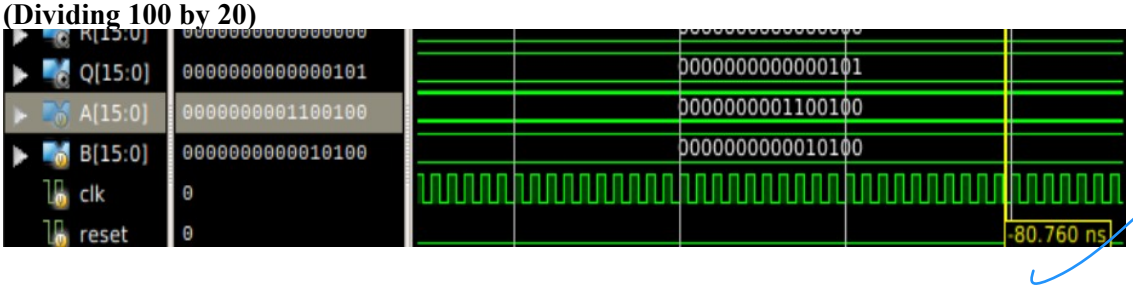


Divider:

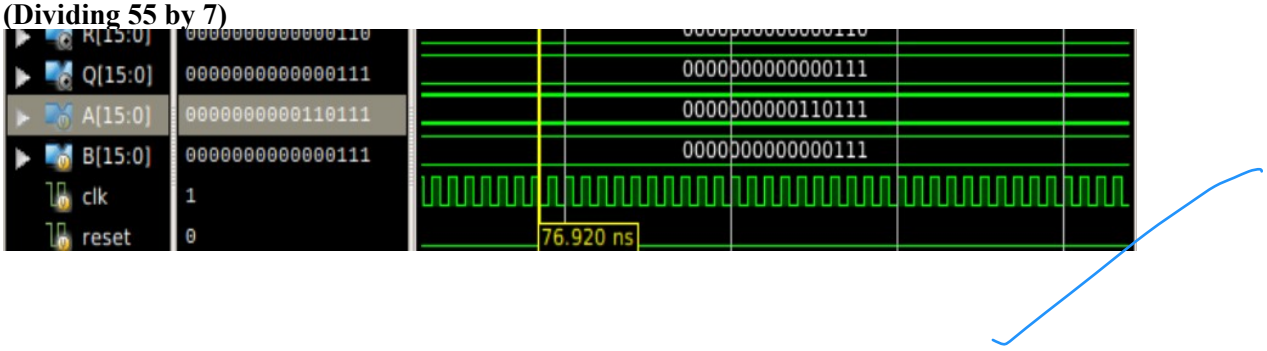
(Reseting)



(Dividing 100 by 20)



(Dividing 55 by 7)



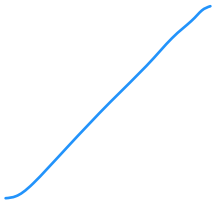
<u>Pad2pad timing constraints</u>

				Slack	Achievable	Errors	Score
1	Yes	Autotimespec constraint for clock net clk_BUFGP	SETUP HOLD	0.178ns	8.678ns	0	0 0

Device Utilization Summary

Divider:

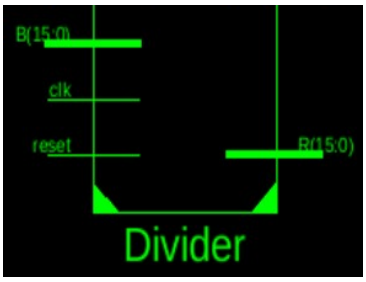
Registers			
Number of Slice LUTs	133	63400	0%
Number of fully used LUT-FF pairs	79	165	47%
Number of bonded IOBs	66	170	38%
Number of BUFG/BUFGCTRLs	1	32	3%

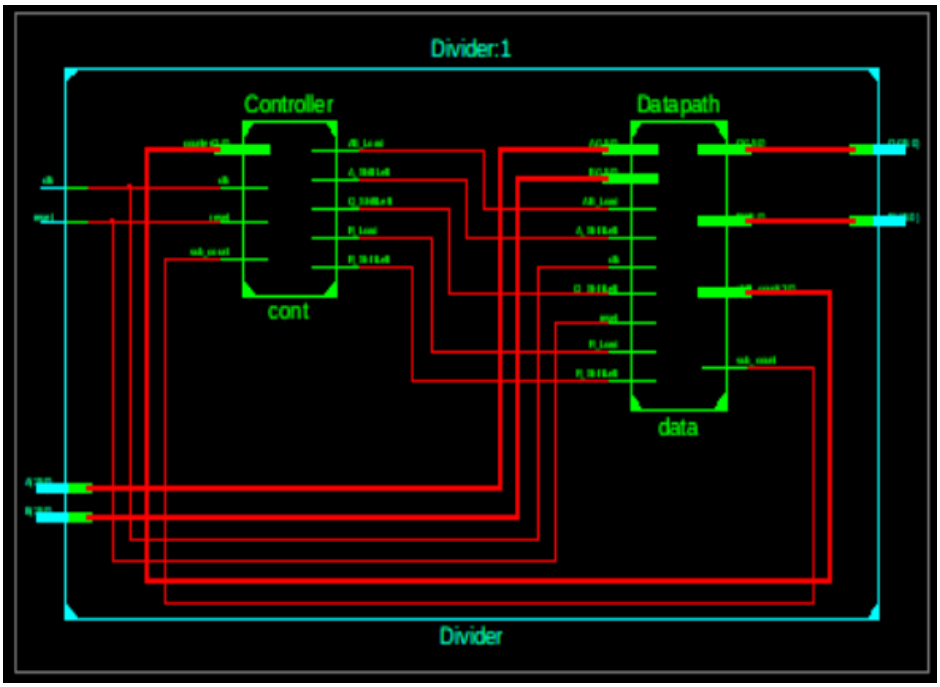


RTL Schematic and Technology Schematic

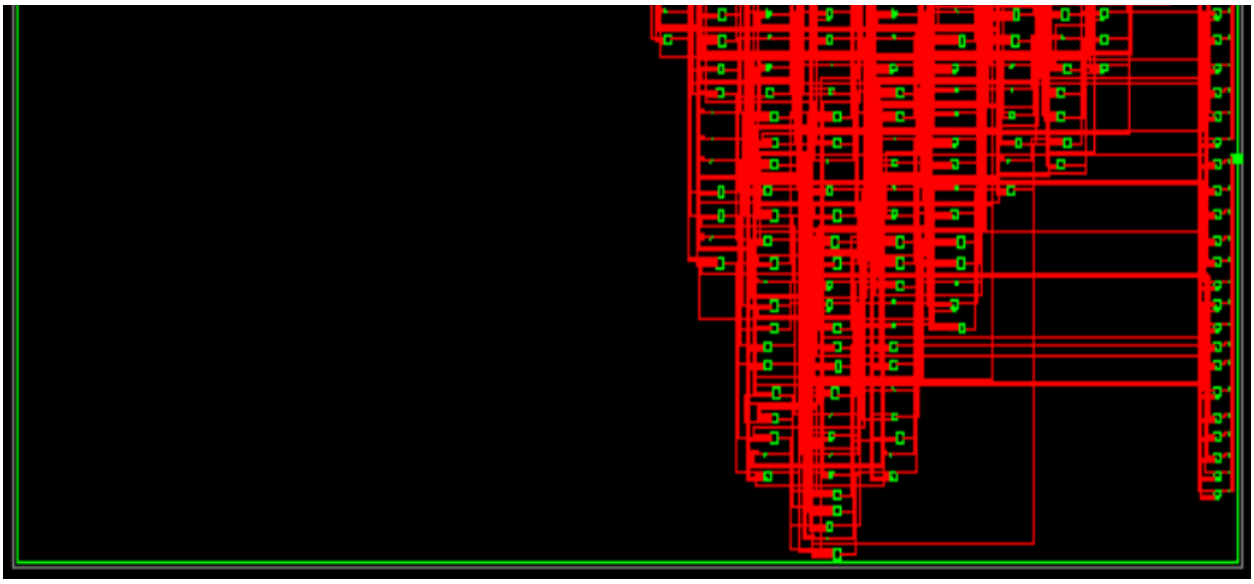
Divider:

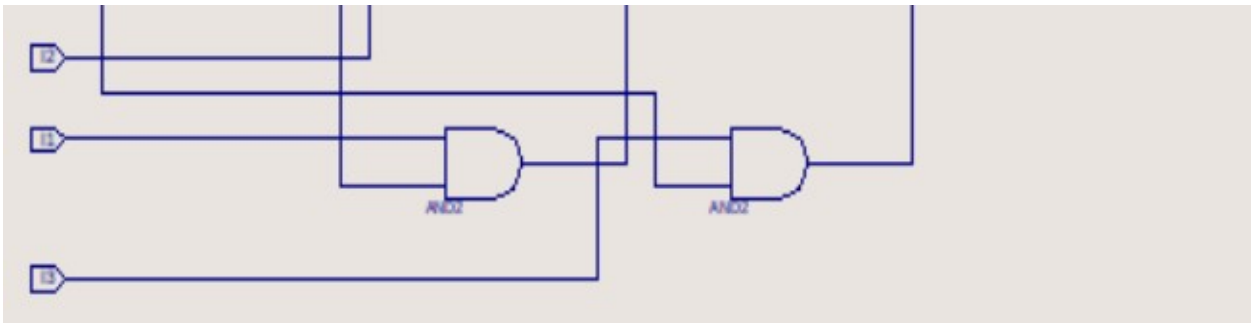
(RTL Schematic)





(Technology Schematic)





Post Place and Route Static Timing

```
INFO:Timing:2752 - To get complete path coverage, use the unconstrained paths
   option. All paths that are not constrained will be reported in the
   unconstrained paths section(s) of the report.
INFO:Timing:3339 - The clock-to-out numbers in this timing report are based on
   a 50 Ohm transmission line loading model.  For the details of this model,
   and for more information on accounting for different loading conditions,
   please see the device datasheet.
```

A<3>	-0.468(F)	FAST	3.254(F)	SLOW	clk_BUFGP	0.000
A<4>	-0.574(F)	FAST	3.427(F)	SLOW	clk_BUFGP	0.000
A<5>	-0.607(F)	FAST	3.535(F)	SLOW	clk_BUFGP	0.000
A<6>	-0.447(F)	FAST	3.238(F)	SLOW	clk_BUFGP	0.000
A<7>	-0.450(F)	FAST	3.253(F)	SLOW	clk_BUFGP	0.000
A<8>	-0.484(F)	FAST	3.241(F)	SLOW	clk_BUFGP	0.000
A<9>	-0.424(F)	FAST	3.144(F)	SLOW	clk_BUFGP	0.000
A<10>	-0.590(F)	FAST	3.455(F)	SLOW	clk_BUFGP	0.000
A<11>	-0.471(F)	FAST	3.274(F)	SLOW	clk_BUFGP	0.000
A<12>	-0.031(F)	FAST	2.513(F)	SLOW	clk_BUFGP	0.000
A<13>	-0.043(F)	FAST	2.531(F)	SLOW	clk_BUFGP	0.000
A<14>	-0.311(F)	FAST	3.050(F)	SLOW	clk_BUFGP	0.000
A<15>	-0.242(F)	FAST	2.927(F)	SLOW	clk_BUFGP	0.000
B<0>	-0.216(F)	FAST	2.789(F)	SLOW	clk_BUFGP	0.000
B<1>	-0.102(F)	FAST	2.560(F)	SLOW	clk_BUFGP	0.000
B<2>	-0.166(F)	FAST	2.672(F)	SLOW	clk_BUFGP	0.000
B<3>	-0.074(F)	FAST	2.654(F)	SLOW	clk_BUFGP	0.000
B<4>	-0.283(F)	FAST	2.962(F)	SLOW	clk_BUFGP	0.000
B<5>	0.033(F)	FAST	2.471(F)	SLOW	clk_BUFGP	0.000
B<6>	-0.389(F)	FAST	3.139(F)	SLOW	clk_BUFGP	0.000
B<7>	-0.367(F)	FAST	3.008(F)	SLOW	clk_BUFGP	0.000

B<11>	-0.509(F)	FAST	3.337(F)	SLOW	clk_BUFGP	0.000
B<12>	-0.378(F)	FAST	3.067(F)	SLOW	clk_BUFGP	0.000
B<13>	-0.516(F)	FAST	3.318(F)	SLOW	clk_BUFGP	0.000
B<14>	-0.426(F)	FAST	3.171(F)	SLOW	clk_BUFGP	0.000
B<15>	-0.448(F)	FAST	3.231(F)	SLOW	clk_BUFGP	0.000
reset	-0.025(R)	FAST	2.648(R)	SLOW	clk_BUFGP	0.000

Q<9>	10.156(R)	SLOW	3.323(R)	FAST	clk_BUFGP	0.000
Q<10>	10.146(R)	SLOW	3.324(R)	FAST	clk_BUFGP	0.000
Q<11>	10.372(R)	SLOW	3.405(R)	FAST	clk_BUFGP	0.000
Q<12>	10.251(R)	SLOW	3.386(R)	FAST	clk_BUFGP	0.000
Q<13>	10.242(R)	SLOW	3.369(R)	FAST	clk_BUFGP	0.000
Q<14>	10.261(R)	SLOW	3.387(R)	FAST	clk_BUFGP	0.000
Q<15>	10.490(R)	SLOW	3.418(R)	FAST	clk_BUFGP	0.000
R<0>	9.928(R)	SLOW	3.186(R)	FAST	clk_BUFGP	0.000
R<1>	9.766(R)	SLOW	3.126(R)	FAST	clk_BUFGP	0.000
R<2>	9.762(R)	SLOW	3.119(R)	FAST	clk_BUFGP	0.000
R<3>	9.743(R)	SLOW	3.119(R)	FAST	clk_BUFGP	0.000
R<4>	9.596(R)	SLOW	3.057(R)	FAST	clk_BUFGP	0.000
R<5>	9.626(R)	SLOW	3.070(R)	FAST	clk_BUFGP	0.000
R<6>	9.773(R)	SLOW	3.113(R)	FAST	clk_BUFGP	0.000
R<7>	9.883(R)	SLOW	3.150(R)	FAST	clk_BUFGP	0.000
R<8>	9.649(R)	SLOW	3.070(R)	FAST	clk_BUFGP	0.000
R<9>	9.678(R)	SLOW	3.082(R)	FAST	clk_BUFGP	0.000
R<10>	9.817(R)	SLOW	3.118(R)	FAST	clk_BUFGP	0.000
R<11>	9.876(R)	SLOW	3.167(R)	FAST	clk_BUFGP	0.000
R<12>	9.664(R)	SLOW	3.083(R)	FAST	clk_BUFGP	0.000
R<13>	9.689(R)	SLOW	3.082(R)	FAST	clk_BUFGP	0.000
R<14>	9.813(R)	SLOW	3.112(R)	FAST	clk_BUFGP	0.000
R<15>	9.880(R)	SLOW	3.170(R)	FAST	clk_BUFGP	0.000

clk	4.052	5.010		7.304

```
Analysis completed Mon Oct 31 13:54:30 2022
--------------------------------------------------------------------------------

Trace Settings:
-------------------------
Trace Settings

Peak Memory Usage: 752 MB
```

Conclusions

In conclusion, we successfully implemented a Divider on Digilent's NEXYS-4 FPGA board using an Algorithmic State Machine. We did this by utilizing datapath and control unit design

techniques to encode the algorithmic divider with the XIlinx ISE tools. This allowed us to solidify our understanding of ASMs and their underlying mechanics and logic.

References

[1]: Nexys4™ FPGA Board Reference Manual, Nexys-4 rev. B; Revised November 19, 2013 by Diligent Beyond Theory

[2]: Introduction to FPGA ECE414 - Fall 2020 pdf by Randil Gajasinghe and Prof. Onur Tigli

[3]: ECE 414 Computer Organization and Design - Experiment 2. Adder/Subtractor with 7-Segment Display

[4]: Brown, S., Vranesic, Z., “Fundamentals of Digital Logic with VHDL Design,” McGraw Hill, 2000.

(-1) for turning in late report.

Rainier Young	sev-seg Schematics	
Brandon Rubio	Divider controller	Brandon Rubio
Nikeem Dunkley-Allen	data path report	
Isabela Bandrich	Divider Code	IsabelaBandrich

Lab Demonstration

Teaching Assistant Signature	Date
	28-10-22
Comments: Demonstration shown on time. Divider working.	