**ELEC 4200 Lab 8 Report**

**Xiaodong Song**

**4/5/2020**

**Goal for the Lab**

For lab 8, students would learn how to define a parameterized model at the first part, then model counters using behavior and IP Catalog, and compare and contrast the counters developed using the behavioral modeling and the IP Catalog at the in second part. For the last part, students would try to design timer circuits using the cores and using additional circuits modeled in HDL and create a real-time clock. After doing this lab, students must be more familiar with the IP Catalog’s wizard clock and binary counter application.

**Design for the Lab**

For lab 8-1-1, students would design a carry-look-ahead adder similar to that students designed in previous lab 2 but using gate-level modeling. After changing the design to be gate-level modeling, students should try to add delay in parameter style as the requirement.

For lab 8-1-2, students would continue modify the carry-look-ahead adder of 1-1 using the defparam statements, then changing the values of the delays from the testbench.

For lab 8-2-1, students would try to design an 8-Bit up/down counter using behavioral modeling. And use the clocking wizard to generate required clock. The task should define the synthesis attribute to not to use the DSP48 slices.

For lab 8-2-2, students would use the 8-2-1’s 8-bit Up/down counter design, but define the synthesis attribute the not to use the DSP48 slices.

For lab 8-2-3, students would design an 8-bit up/down counter using the 8-bit core generated using the IP Catalog.

For lab 8-2-4, students would use the 8-bit up/down counter design from 8-2-3 but with the counter regenerated to use the DSP48 slices.

For lab 8-3-1, students would design a stop-watch timer using the IP Catalog to generate an appropriately sized counter core with the desired input control signals to measure a time up to 2 minutes at a resolution of one-tenth of a second.

For lab 8-3-2, students would design a countdown timer using the behavioral modeling to model a parameterized count down counter with the desired input control signals to show the count down time from a desired initial value.

For lab 8-3-3, students would design a real-time clock using the IP Catalog to generate an appropriately sized counter core with the desired input control signals.

**Detailed Design**

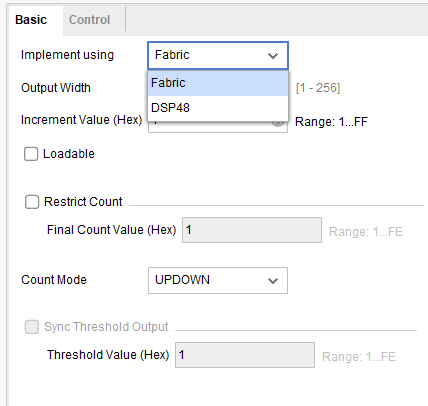
For lab 8-1-1, the first thing was to modify the carry-look-ahead adder design to be gate-level modeling. After doing this, students would define 2 units delay for each kind of gate that students use in the full-adder circuit using the parameter statements. Besides, at the process of creating hierarchical models, students have to use 1-unit delay for inverters, 3 units delay for “AND” and “OR” gates, and 4-units delay for “xor” gates. Then develop a testbench for verifying the result.

For lab 8-1-2, the 8-1-1’s design can be used again, but students need to use the defparam statements to change the values of the delays from the testbench and observe the delays propagated.

For lab 8-2-1, students would design an 8-bit up/down first, then the model should define COUNT\_SIZE as a parameter for using it. The main purpose of the design was to use the 100MHz source to generate a 5MHz clock and the appropriate clock divider circuit to generate further clock of 10kHz. Before doing this, students have to define 100Mhz clock in the testbench and use it for making the input clock. Besides, the synthesis attribute should not use the DSP48 slices which should be done as “(\* use\_dsp48 = “no” \*)”. After doing all of them, verify the functionality using simulation and try to figure out the number of BUFG/BUFGCTRL, Slice LUTs used, FF used, DSP48E1 slices used, and IOs used by reviewing the Project Summary tab.

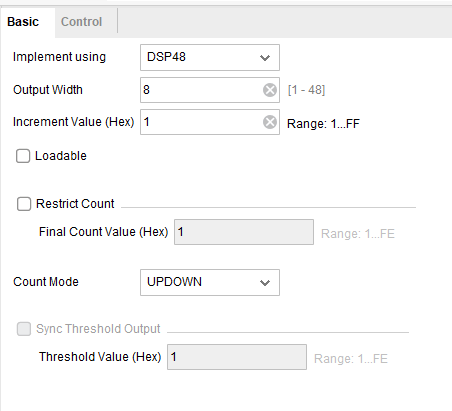
For lab 8-2-2, students would use the 8-2-1’s design, but need to set the synthesis attribute to use the DSP48 slices. Similarly, after doing all of them, students still need to verify the functionality using simulation and try to figure out the number of BUFG/BUFGCTRL, Slice LUTs used, FF used, DSP48E1 slices used, and IOs used by reviewing the Project Summary tab.

For lab 8-2-3, students would design an 8-bit up/down counter using the 8-Bit core generated using the IP Catalog first. Then define 100MHz clock in the testbench to use it for making an input clock. And use the 100MHz clock source to generate a 5MHz clock and the appropriate clock divider circuit to generate further clock of 10KHz. For this part, students need to set a clocking wizard and a binary counter. The DSP48 slices would not be used in this step. One significant thing was that the Binary counter basic setting’s implement using should not use the DSP48 and the count mode should use UPDOWN which is shown below (**Figure** **1**). After doing all of them, verify the functionality using simulation and try to figure out the number of BUFG/BUFGCTRL, Slice LUTs used, FF used, DSP48E1 slices used, and IOs used by reviewing the Project Summary tab.



**Figure** **1**: 8-2-3’s Binary Counter Set

For lab 8-2-4, students would use the 8-2-3’s design but the DSP48 slices should be used which means that the binary counter basic setting’s implement using should be DSP48 which is shown below(Figure 2). After doing all of them, verify the functionality using simulation and try to figure out the number of BUFG/BUFGCTRL, Slice LUTs used, FF used, DSP48E1 slices used, and IOs used by reviewing the Project Summary tab.

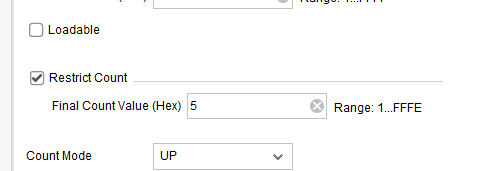


**Figure** **2**: 8-2-3’s Binary Counter Set

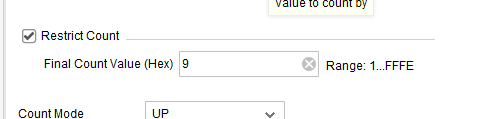
For lab 8-3-1, students would design as stop-watch timer using the IP Catalog. It should instantiate the core a number of required times and add the required additional circuitry to display the time in M.SS.f format on the four decimal number. The M.SS.f should be minutes, seconds, microseconds, and frequency, respectively. The design input should be a 100MHz clock source, a reset signal, and an enable. When the enable signal is asserted the clock counts, the it is de-asserted the clock pauses. At any time if the RESET signal is ON the clock reset to the 0.00.0 value. After doing all of them, students need to develop testbench for verifying. Hence, for this step, students need to use 1 clock divider and 4 binary counters which count minutes, seconds, microseconds, and frequency.

For lab 8-3-2, students would design a countdown timer. The desired input control signals to show the count down time from a desired initial value set by the two-bit LOAD input at a second resolution. The display format should be MM.SS format on he three decimal numbers. Input should be a 100MHz clock source, a re-load signal, an enable signal, and 2-bit LOAD as the starting value in umber if whole minutes. The clock would count when enable signal is ON, the clock would pause when enable is OFF. And the re-load signal is on the timer loads to MM.00, where the value of MM is determined by the 2-bit LOAD setting. After doing all of them, students still need to develop a testbench for verifying this. And for this design, it just needs one clocking wizard.

For lab 8-3-3, students would design a real-time clock using the IP Catalog. This should instantiate clock two times and add the required circuit to display the time in MM.SS format on the four decimal numbers. The design input should be 100MHz clock source and a reset signal. When the reset is ON, the clock should be reset to 00.00. After designing, students need to develop testbench for verification. And for the design, the IP Catalog need 1 clocking wizard and 4 binary counters which count two minutes bit and two seconds bit. So, they should need restrict count. For the last one seconds, the restrict count should be 9 then when it arrives to A, the left seconds would become 1, then the right one seconds go back to 0 again. And for the left seconds, the restrict count should be 5 which means that when it arrives the largest 4-bit number, the minutes will start working. And for minutes, the restrict count should be same with the seconds restrict count. And the followings are the set for restrict count.



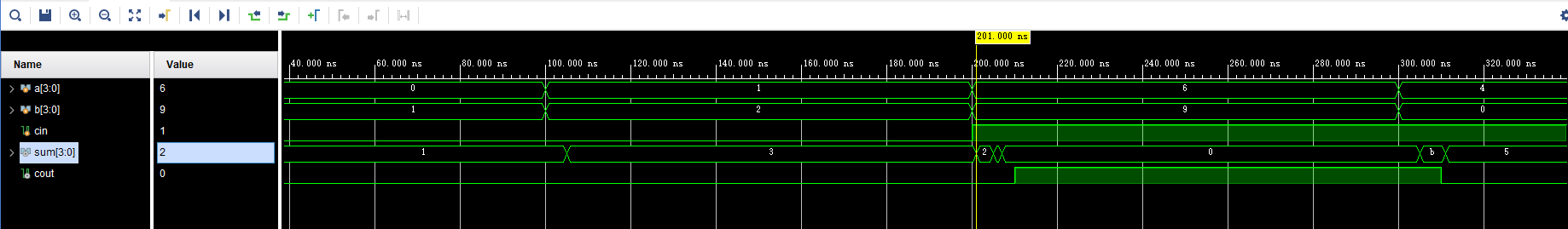
**Figure 3**: 8-3-3’s Restricts Count(left)



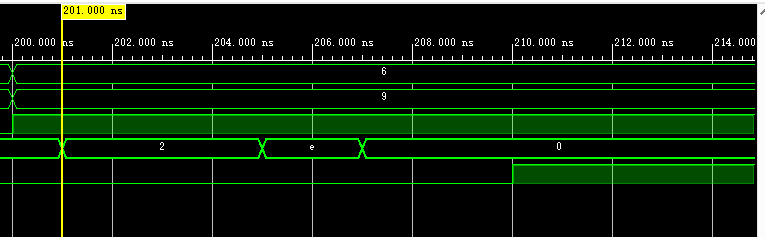
**Figure 4**: 8-3-3’s Restricts Count(right)

**Design Verification**

For lab 8-1-1, the following is the simulation result, there is totally four couple numbers were added up. And as the figure shown, there is a 4-units delay for xor gate which is on the 201.000ns, for the next one value, there is 2-units delay which should be each kind of gate’s delay.

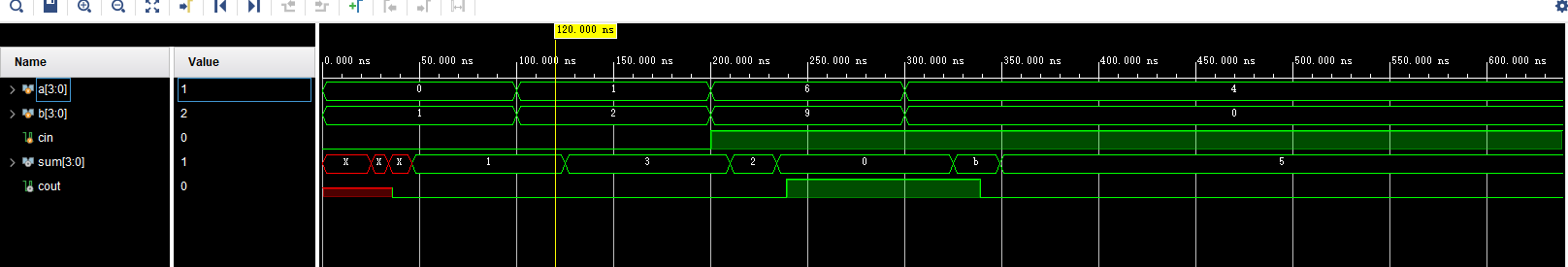


**Figure 5**: 8-1-1’s whole simulation

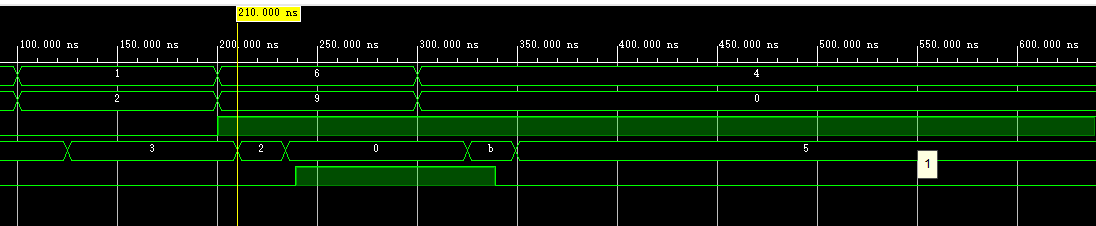


**Figure 5:** 8-1-1’s simulation at 201.000ns

For lab 8-1-2, at the testbench file, the ­delay has been set as following: inverter\_delay was set as 10, andor\_delay was set as 12, and xor\_delay was set as 15. As the second figure shows the 210.000ns, if there is no delay here. The sum should change at 200.00ns, however, it changes at 210.000ns because the inverter’s delay is 10.



**Figure 6:** 8-1-2’s whole simulation



**Figure 7:** 8-1-2’s simulation at 210.000ns

For lab 8-2-1, the simulation result was shown below, for the counter size, it has been set as 8-bit decimal number 255, then if the enable and updown are ON(they are all equal to 1), the output out would be keep counting up which was shown as below first figure. And for the second figure, that is the enable and updown are all 0, then the counter would not work. And if the BTNU is ON, the counter would not keep counting up and go back to 0 as the third figure show. And for the Summary tab, **Figure 11** has shown the required data:

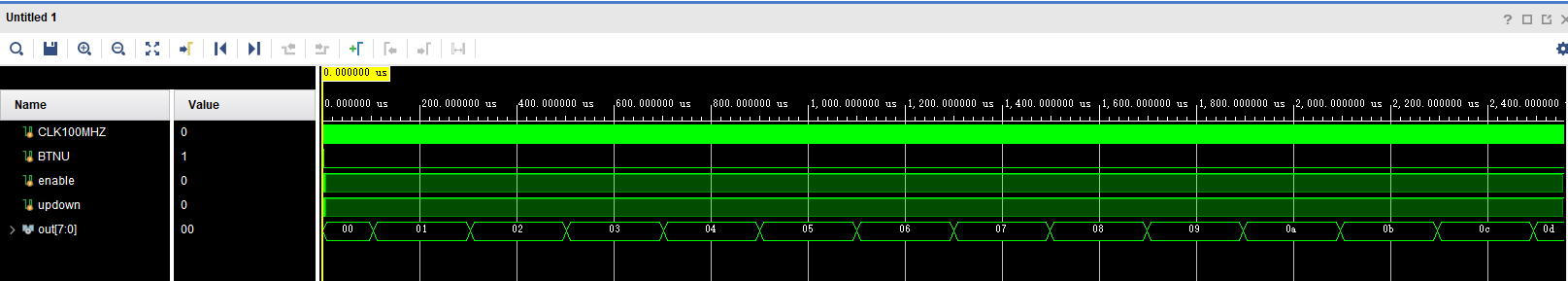
Number of BUFG/BUFGCTRL: 34

Number of Slice LUTs used: 41

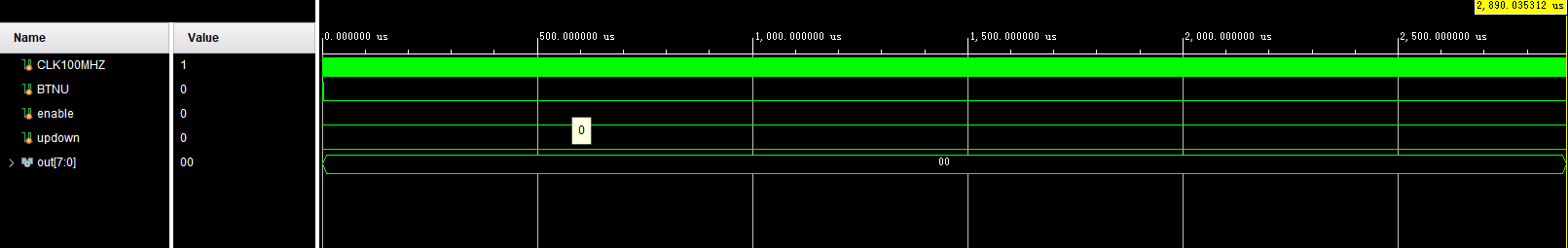
Number of FF used: 12

Number of DSP48E1 slices used: 2

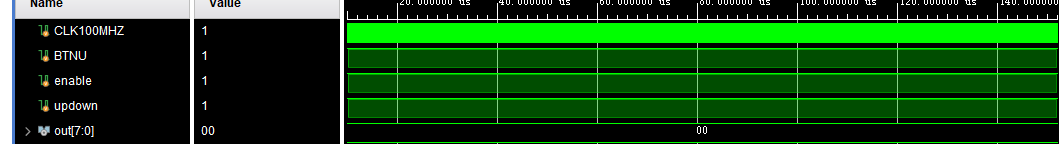
Number of IOs used: 1



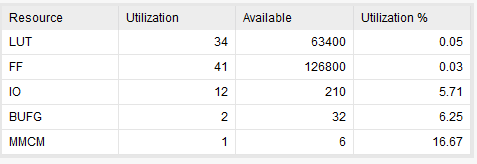
**Figure 8:** Enable and Updown are 1



**Figure 9:** Enable, Updown and BTNU are 0



**Figure 10:** Enable, Updown and BTNU are 1



**Figure 11**: Summary Tab of 8-2-1

For lab 8-2-2, the design was used the exact same design, the only one difference was the DSP48 was used in 8-2-2 but not used in 8-2-1, so the simulation result are exactly same with 8-2-1. And following are the summary tab for showing the required data:

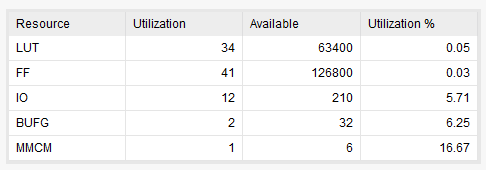
Number of BUFG/BUFGCTRL: 34

Number of Slice LUTs used: 41

Number of FF used: 12

Number of DSP48E1 slices used: 2

Number of IOs used: 1



**Figure 12:** Summary Tab of 8-2-2

For lab 8-2-3, the simulation result was shown below, and the summary tab was shown below, the design’s functionality is exact same with 8-2-1.

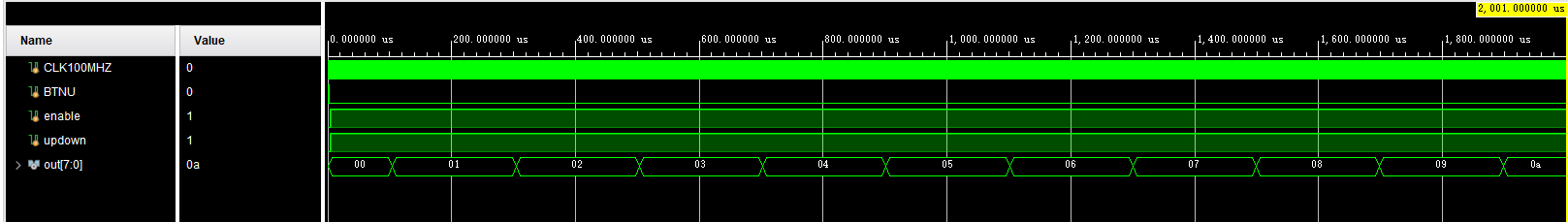
Number of BUFG/BUFGCTRL: 153

Number of Slice LUTs used: 74

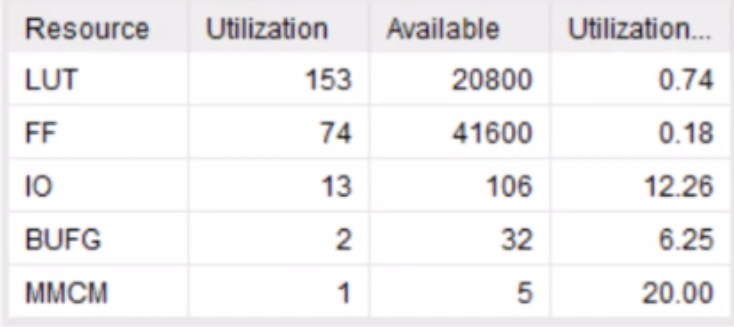
Number of FF used: 13

Number of DSP48E1 slices used: 2

Number of IOs used: 1



**Figure 13:** 8-2-3’s simulation



**Figure 14:** 8-2-3’s Summary Tab

For lab 8-2-4, the simulation was shown below, and the summary tab was shown below. The design’s functionality is same with 8-2-3.

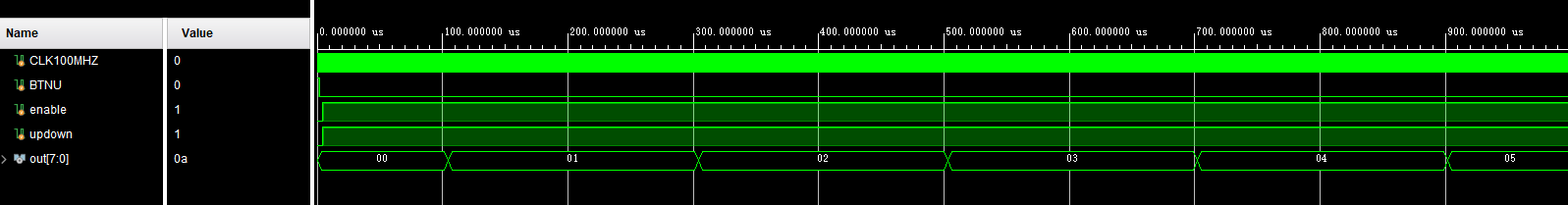
Number of BUFG/BUFGCTRL: 34

Number of Slice LUTs used: 41

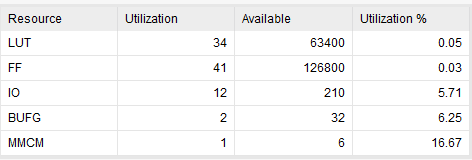
Number of FF used: 12

Number of DSP48E1 slices used: 2

Number of IOs used: 1

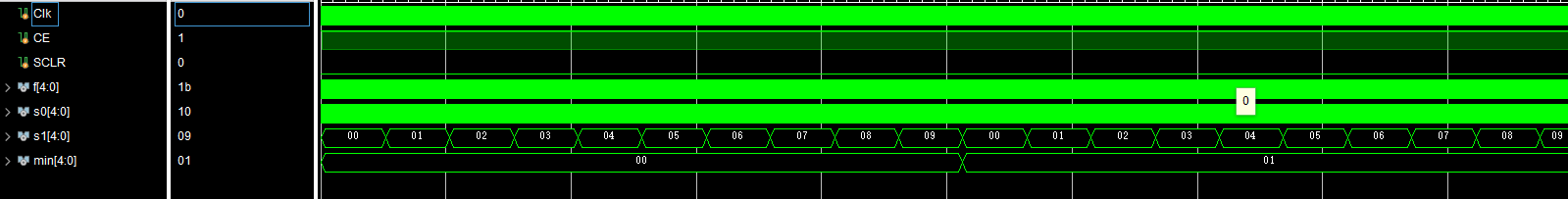


**Figure 15:** 8-2-4’s simulation

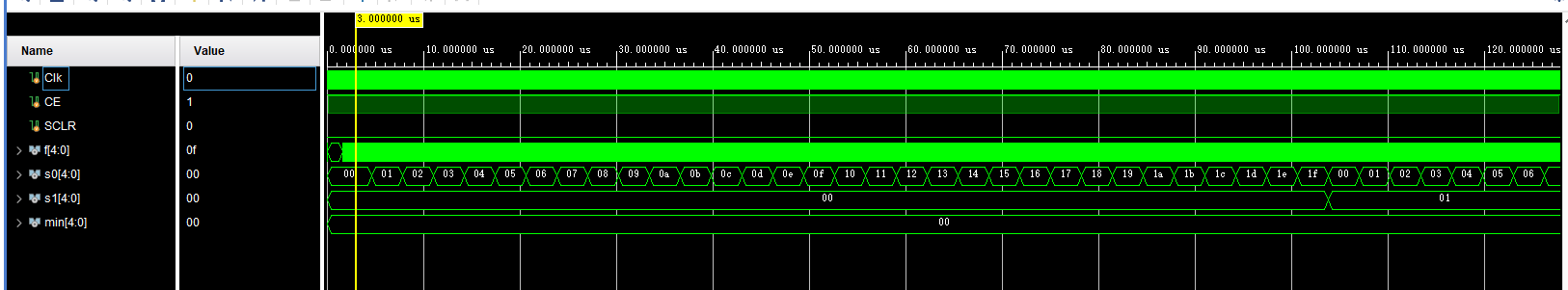


**Figure 16:** 8-2-4’ Summary Tab

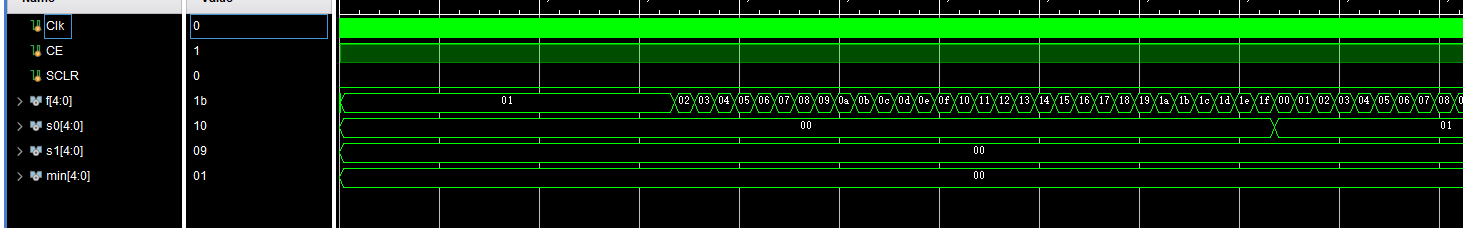
For lab 8-3-1, the real-time clock design simulation result was shown below, as the figure 17 shown that, when s1 arrives at 10, the min would change to be 1. “s1” represents the seconds and “s2” represents the microseconds. “min” represents minutes and “f” represents frequency.



**Figure 17**: Simulation for 8-3-1 (Seconds to Minutes)

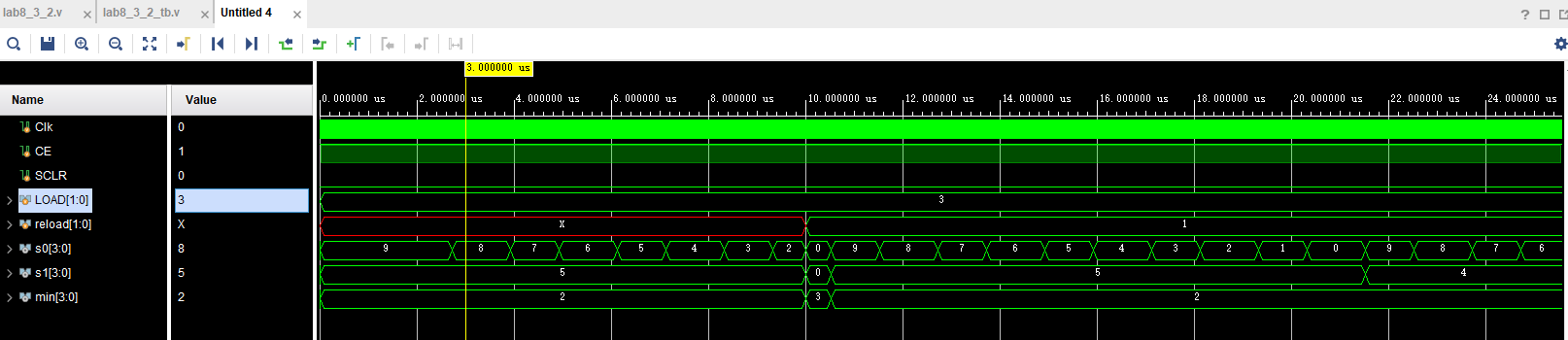


**Figure 18:** Simulation for 8-3-1 (Microseconds to Seconds)

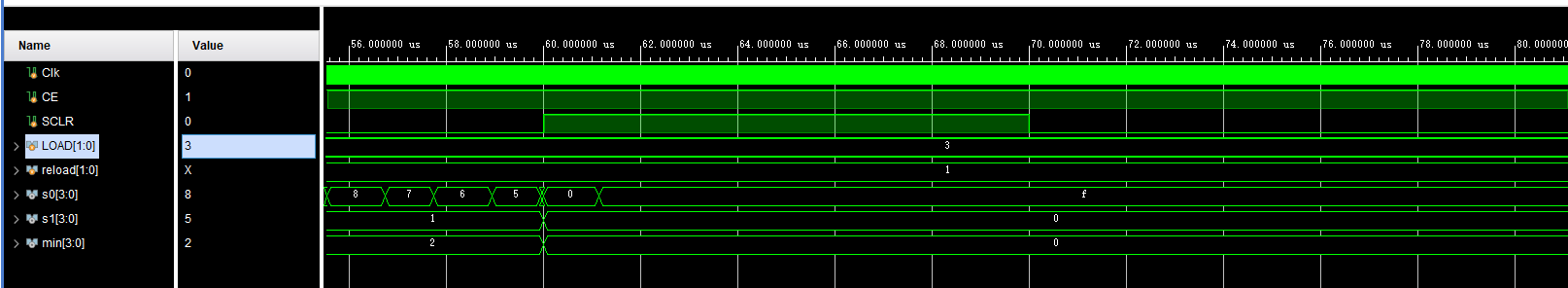


**Figure 18:** Simulation for 8-3-1 (Frequency to Microseconds)

For lab 8-3-2, the following shown was the simulation result, as the **Figure 19** showing, reload works for letting timer be equal to input load when reload is ON (at 10.000us). “s0” represents microseconds, “s1” represents seconds, and “min” represents minutes. As the **Figure 20** showing, when the SCLR (clear) is ON, the timer would go back to 0. And for the relationships between s0, s1, and min. 1min = 60s1 = 600s0.

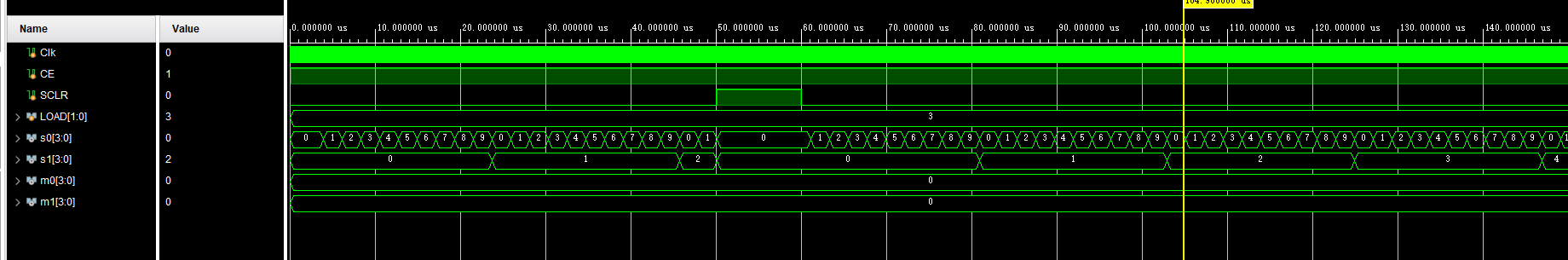


**Figure 19:** Simulation for 8-3-2

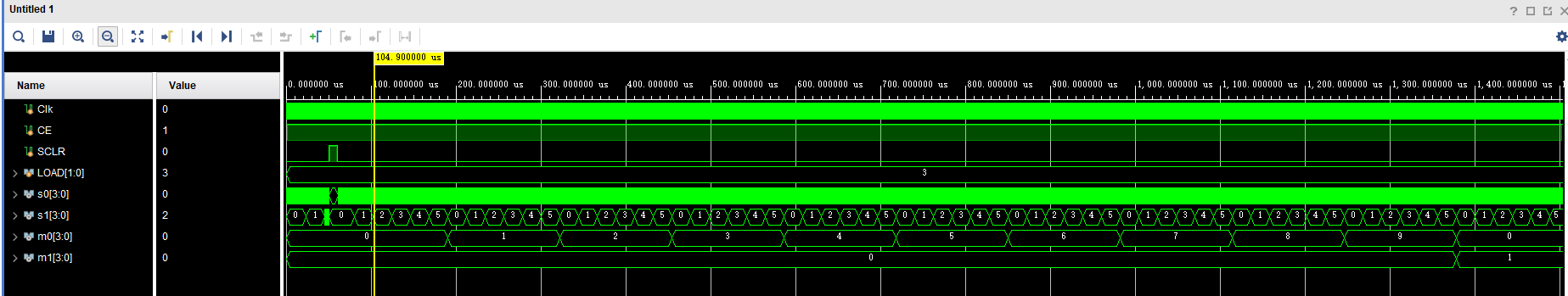


**Figure 20:** Simulation for 8-3-2 (Clear function)

For lab 8-3-3, the following shown was the simulation results, first figure shows the transfer between s0 and s1. And the function of clear and load. Second figure show the transfer between s1 and m0, and between m0 and m1. The function has been illustrated in the detailed design part.



**Figure 20:** Simulation for 8-3-3



**Figure 21:** Simulation for 8-3-3

**Conclusion**

After finishing the lab, students would be more familiar with the IP Catalog’s clocking wizard and binary counter. But the lab has been becoming harder and taking more time. Any of the parts are not difficult. But after combining them together, when some mistakes happened, it is difficult to figure out where is the problem. Then most time of designing program should be in the problem figuring out. Obviously, being familiar with the details is so significant.

Appendix

Lab 8\_1\_1:

module lab8\_1\_1 :

#(parameter inverter\_delay = 1, andor\_delay = 3, xor\_delay = 4)

(

input [3:0]a,

input [3:0]b,

input cin,

output [3:0]sum,

output cout

);

wire cout1,cout2,cout3;

wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4,c0;

wire pg0,pg1,pg2,pg3;

wire pp1,pp2,pp3;

wire ppp1,ppp2,ppp3;

xor #xor\_delay (p0, a[0], b[0]), (p1, a[1], b[1]), (p2, a[2], b[2]), (p3, a[3], b[3]);

and #andor\_delay (g0, a[0], b[0]), (g1, a[1], b[1]), (g2, a[2], b[2]), (g3, a[3], b[3]);

and #andor\_delay (pg0, p0, cin), (pg1, p1, g0), (pg2, p2, g1), (pg3, p3, g2),

(pp1, p1, p0, cin), (pp2, p2, p1, g0), (pp3, p3, p2, g1),

(ppp1, p1, p1, p1, cin), (ppp2, p3, p2, p1, g0), (ppp3, p3, p2, p1, p0, cin);

or #andor\_delay (c1, g0, pg0), (c2, g2, pg1, pp1), (c3, g2, pg2, pp2, ppp1), (cout, g3, pg3, pp3, ppp2, ppp3);

xor #inverter\_delay (sum[0], p0, cin), (sum[1], p1, c1), (sum[2], p2, c2), (sum[3], p3, c3);

endmodule

Lab 8\_1\_1\_tb:

module lab8\_1\_1\_tb;

reg [3:0] a;

reg [3:0] b;

reg cin;

wire [3:0] sum;

wire cout;

lab8\_1\_1 DUT (.a(a), .b(b), .cin(cin),.sum(sum), .cout(cout));

initial begin

a = 4'b0000;b = 4'b0001;cin = 0;

#100 a = 4'b0001; b = 4'b0010; cin = 0;

#100 a = 4'b0110; b = 4'b1001; cin = 1;

#100 a = 4'b0100; b = 4'b0000; cin = 1;

#100 $finish;

end

endmodule

Lab 8\_1\_2:

module lab8\_1\_2

#(parameter inverter\_delay = 1, andor\_delay = 3, xor\_delay = 4)(

input [3:0]a,

input [3:0]b,

input cin,

output [3:0]sum,

output cout

);

wire cout1,cout2,cout3;

wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4,c0;

wire pg0,pg1,pg2,pg3;

wire pp1,pp2,pp3;

wire ppp1,ppp2,ppp3;

xor #xor\_delay (p0, a[0], b[0]), (p1, a[1], b[1]), (p2, a[2], b[2]), (p3, a[3], b[3]);

and #andor\_delay (g0, a[0], b[0]), (g1, a[1], b[1]), (g2, a[2], b[2]), (g3, a[3], b[3]);

and #andor\_delay (pg0, p0, cin), (pg1, p1, g0), (pg2, p2, g1), (pg3, p3, g2),

(pp1, p1, p0, cin), (pp2, p2, p1, g0), (pp3, p3, p2, g1),

(ppp1, p1, p1, p1, cin), (ppp2, p3, p2, p1, g0), (ppp3, p3, p2, p1, p0, cin);

or #andor\_delay (c1, g0, pg0), (c2, g2, pg1, pp1), (c3, g2, pg2, pp2, ppp1), (cout, g3, pg3, pp3, ppp2, ppp3);

xor #inverter\_delay (sum[0], p0, cin), (sum[1], p1, c1), (sum[2], p2, c2), (sum[3], p3, c3);

endmodule

Lab 8\_1\_2\_tb:

module lab8\_1\_2\_tb;

reg [3:0] a;

reg [3:0] b;

reg cin;

wire [3:0] sum;

wire cout;

defparam DUT.inverter\_delay = 10, DUT.andor\_delay = 12, DUT.xor\_delay = 15;

lab8\_1\_2 DUT (.a(a), .b(b), .cin(cin),.sum(sum), .cout(cout));

initial begin

a = 4'b0000;b = 4'b0001;cin = 0;

#100 a = 4'b0001; b = 4'b0010; cin = 0;

#100 a = 4'b0110; b = 4'b1001; cin = 1;

#100 a = 4'b0100; b = 4'b0000; cin = 1;

#100 $finish;

end

endmodule

Lab 8\_2\_1:

(\* use\_dsp48 = "no" \*)

module lab8\_2\_1

#(parameter COUNT\_SIZE =8'd25)

(

input wire CLK100MHZ,

input wire BTNU ,

input wire enable ,

input wire updown ,

output wire [7:0] out

);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

wire CLK5MHZ;

wire CLK\_1HZ;

clk\_wiz\_0 clk\_wiz\_div\_inst0

(

// Clock out ports

.clk\_out1(CLK5MHZ), // output clk\_out1

// Status and control signals

.reset(BTNU), // input reset

.locked(), // output locked

// Clock in ports

.clk\_in1(CLK100MHZ));

clock\_div clock\_div\_inst1(

.CLK5MHZ(CLK5MHZ),

.reset(BTNU),

.CLK\_1HZ(CLK\_1HZ)

);

// Instantiate the module

Counter\_Module

#(

.COUNT\_SIZE(COUNT\_SIZE)

)

Counter\_Module\_inst2

(

.CLK\_1HZ(CLK\_1HZ),

.BTNU(BTNU),

.enable(enable),

.updown(updown),

.out(out)

);

Endmodule

module clock\_div(

input CLK5MHZ,

input reset,

output reg CLK\_1HZ

);

integer n;

initial begin

CLK\_1HZ = 0;

n = 0;

end

always @(posedge CLK5MHZ) begin

if (reset == 1'b1) begin

n <= 0;

CLK\_1HZ <= 0;

end

else begin

n = n + 1;

if (n >= 249) begin////

CLK\_1HZ = ~CLK\_1HZ;

n = 0;

end

end

end

endmodule

module Counter\_Module

#(parameter COUNT\_SIZE = 8'd255)

(

input wire CLK\_1HZ,

input wire BTNU ,

input wire enable ,

input wire updown ,

output wire [7:0] out

);

reg [7:0] Counter = 8'd0;

always @ (posedge CLK\_1HZ) begin

if (BTNU == 1'b1)begin

Counter <= 8'b0;

end

else if (enable == 1'b1) begin

if (updown == 1'b1) begin

if (Counter == COUNT\_SIZE)

Counter <= 8'd0;

else

Counter <= Counter + 1'b1;

end

else begin

if (Counter == 8'd0)

Counter <= COUNT\_SIZE;

else

Counter <= Counter - 1'b1;

end

end

else

Counter <= Counter;

end

assign out = Counter;

endmodule

module tb\_Counter\_Design;

// Inputs

reg CLK100MHZ;

reg BTNU;

reg enable;

reg updown;

// Outputs

wire [7:0] out;

// Instantiate the Unit Under Test (UUT)

lab8\_2\_1 uut (

.CLK100MHZ(CLK100MHZ),

.BTNU(BTNU),

.enable(enable),

.updown(updown),

.out(out)

);

initial begin

// Initialize Inputs

CLK100MHZ = 0;

// Wait 100 ns for global reset to finish

forever #10 CLK100MHZ=!CLK100MHZ;

// Add stimulus here

end

initial begin

BTNU = 0;

enable = 1;

updown = 1;

#2000;

BTNU = 0;

#2000;

#500 enable = 1;

#500 updown = 1;

#500 BTNU = 0;

#200 enable = 1;

#300 updown = 1;

#3000 BTNU = 1;

end

endmodule

Lab 8\_2\_2:

(\* use\_dsp48 = "yes" \*)

module lab8\_2\_2

#(parameter COUNT\_SIZE =8'd25)

(

input wire CLK100MHZ,

input wire BTNU ,

input wire enable ,

input wire updown ,

output wire [7:0] out

);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

wire CLK5MHZ;

wire CLK\_1HZ;

clk\_wiz\_0 clk\_wiz\_div\_inst0

(

// Clock out ports

.clk\_out1(CLK5MHZ), // output clk\_out1

// Status and control signals

.reset(BTNU), // input reset

.locked(), // output locked

// Clock in ports

.clk\_in1(CLK100MHZ));

clock\_div clock\_div\_inst1(

.CLK5MHZ(CLK5MHZ),

.reset(BTNU),

.CLK\_1HZ(CLK\_1HZ)

);

// Instantiate the module

Counter\_Module

#(

.COUNT\_SIZE(COUNT\_SIZE)

)

Counter\_Module\_inst2

(

.CLK\_1HZ(CLK\_1HZ),

.BTNU(BTNU),

.enable(enable),

.updown(updown),

.out(out)

);

Endmodule

module clock\_div(

input CLK5MHZ,

input reset,

output reg CLK\_1HZ

);

integer n;

initial begin

CLK\_1HZ = 0;

n = 0;

end

always @(posedge CLK5MHZ) begin

if (reset == 1'b1) begin

n <= 0;

CLK\_1HZ <= 0;

end

else begin

n = n + 1;

if (n >= 249) begin////

CLK\_1HZ = ~CLK\_1HZ;

n = 0;

end

end

end

endmodule

module Counter\_Module

#(parameter COUNT\_SIZE = 8'd255)

(

input wire CLK\_1HZ,

input wire BTNU ,

input wire enable ,

input wire updown ,

output wire [7:0] out

);

reg [7:0] Counter = 8'd0;

always @ (posedge CLK\_1HZ) begin

if (BTNU == 1'b1)begin

Counter <= 8'b0;

end

else if (enable == 1'b1) begin

if (updown == 1'b1) begin

if (Counter == COUNT\_SIZE)

Counter <= 8'd0;

else

Counter <= Counter + 1'b1;

end

else begin

if (Counter == 8'd0)

Counter <= COUNT\_SIZE;

else

Counter <= Counter - 1'b1;

end

end

else

Counter <= Counter;

end

assign out = Counter;

endmodule

module tb\_Counter\_Design;

// Inputs

reg CLK100MHZ;

reg BTNU;

reg enable;

reg updown;

// Outputs

wire [7:0] out;

// Instantiate the Unit Under Test (UUT)

lab8\_2\_2 uut (

.CLK100MHZ(CLK100MHZ),

.BTNU(BTNU),

.enable(enable),

.updown(updown),

.out(out)

);

initial begin

// Initialize Inputs

CLK100MHZ = 0;

// Wait 100 ns for global reset to finish

forever #10 CLK100MHZ=!CLK100MHZ;

// Add stimulus here

end

initial begin

BTNU = 1;

enable = 0;

updown = 0;

#2000;

BTNU = 0;

#2000;

enable = 1;

updown = 1;

end

endmodule

Lab 8\_2\_3:

(\* use\_dsp48 = "no" \*)

module lab8\_2\_3

#(parameter COUNT\_SIZE =8'd25)

(

input wire CLK100MHZ,

input wire BTNU ,

input wire enable ,

input wire updown ,

output wire [7:0] out

);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

wire CLK5MHZ;

wire CLK\_1HZ;

clk\_wiz\_0 clk\_wiz\_div\_inst0

(

// Clock out ports

.clk\_out1(CLK5MHZ), // output clk\_out1

// Status and control signals

.reset(BTNU), // input reset

.locked(), // output locked

// Clock in ports

.clk\_in1(CLK100MHZ));

clock\_div clock\_div\_inst1(

.CLK5MHZ(CLK5MHZ),

.reset(BTNU),

.CLK\_1HZ(CLK\_1HZ)

);

// Instantiate the module

Counter\_Module

#(

.COUNT\_SIZE(COUNT\_SIZE)

)

Counter\_Module\_inst2

(

.CLK\_1HZ(CLK\_1HZ),

.BTNU(BTNU),

.enable(enable),

.updown(updown),

.out(out)

);

c\_counter\_binary\_0 counter

(

CLK, Q

);

// always @ (negedge out) begin

// counterclock = 1; #100 counterclock = 0;

// end

Endmodule

module clock\_div(

input CLK5MHZ,

input reset,

output reg CLK\_1HZ

);

integer n;

initial begin

CLK\_1HZ = 0;

n = 0;

end

always @(posedge CLK5MHZ) begin

if (reset == 1'b1) begin

n <= 0;

CLK\_1HZ <= 0;

end

else begin

n = n + 1;

if (n >= 249) begin////

CLK\_1HZ = ~CLK\_1HZ;

n = 0;

end

end

end

endmodule

module Counter\_Module

#(parameter COUNT\_SIZE = 8'd255)

(

input wire CLK\_1HZ,

input wire BTNU ,

input wire enable ,

input wire updown ,

output wire [7:0] out

);

reg [7:0] Counter = 8'd0;

always @ (posedge CLK\_1HZ) begin

if (BTNU == 1'b1)begin

Counter <= 8'b0;

end

else if (enable == 1'b1) begin

if (updown == 1'b1) begin

if (Counter == COUNT\_SIZE)

Counter <= 8'd0;

else

Counter <= Counter + 1'b1;

end

else begin

if (Counter == 8'd0)

Counter <= COUNT\_SIZE;

else

Counter <= Counter - 1'b1;

end

end

else

Counter <= Counter;

end

assign out = Counter;

endmodule

module tb\_Counter\_Design;

// Inputs

reg CLK100MHZ;

reg BTNU;

reg enable;

reg updown;

// Outputs

wire [7:0] out;

// Instantiate the Unit Under Test (UUT)

lab8\_2\_3 uut (

.CLK100MHZ(CLK100MHZ),

.BTNU(BTNU),

.enable(enable),

.updown(updown),

.out(out)

);

initial begin

// Initialize Inputs

CLK100MHZ = 0;

// Wait 100 ns for global reset to finish

forever #10 CLK100MHZ=!CLK100MHZ;

// Add stimulus here

end

initial begin

BTNU = 1;

enable = 0;

updown = 0;

#2000;

BTNU = 0;

#2000;

enable = 1;

updown = 1;

end

endmodule

Lab 8\_2\_4:

(\* use\_dsp48 = "yes" \*)

module lab8\_2\_4

#(parameter COUNT\_SIZE =8'd25)

(

input wire CLK100MHZ,

input wire BTNU ,

input wire enable ,

input wire updown ,

output wire [7:0] out

);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

wire CLK5MHZ;

wire CLK\_1HZ;

clk\_wiz\_0 clk\_wiz\_div\_inst0

(

// Clock out ports

.clk\_out1(CLK5MHZ), // output clk\_out1

// Status and control signals

.reset(BTNU), // input reset

.locked(), // output locked

// Clock in ports

.clk\_in1(CLK100MHZ));

clock\_div clock\_div\_inst1(

.CLK5MHZ(CLK5MHZ),

.reset(BTNU),

.CLK\_1HZ(CLK\_1HZ)

);

// Instantiate the module

Counter\_Module

#(

.COUNT\_SIZE(COUNT\_SIZE)

)

Counter\_Module\_inst2

(

.CLK\_1HZ(CLK\_1HZ),

.BTNU(BTNU),

.enable(enable),

.updown(updown),

.out(out)

);

c\_counter\_binary\_0 counter

(

CLK, Q

);

Endmodule

module clock\_div(

input CLK5MHZ,

input reset,

output reg CLK\_1HZ

);

integer n;

initial begin

CLK\_1HZ = 0;

n = 0;

end

always @(posedge CLK5MHZ) begin

if (reset == 1'b1) begin

n <= 0;

CLK\_1HZ <= 0;

end

else begin

n = n + 1;

if (n >= 249) begin////

CLK\_1HZ = ~CLK\_1HZ;

n = 0;

end

end

end

endmodule

module tb\_Counter\_Design;

// Inputs

reg CLK100MHZ;

reg BTNU;

reg enable;

reg updown;

// Outputs

wire [7:0] out;

// Instantiate the Unit Under Test (UUT)

lab8\_2\_4 uut (

.CLK100MHZ(CLK100MHZ),

.BTNU(BTNU),

.enable(enable),

.updown(updown),

.out(out)

);

initial begin

// Initialize Inputs

CLK100MHZ = 0;

// Wait 100 ns for global reset to finish

forever #10 CLK100MHZ=!CLK100MHZ;

// Add stimulus here

end

Lab 8\_3\_1:

module lab8\_3\_1(

input Clk, CE, SCLR,

output [4:0] f, s0, s1, min

);

wire clk5;

clk\_wiz\_0 clk\_wiz\_div\_inst0

(

.clk\_out1(clk5),

.locked(locked),

.clk\_in1(Clk)

);

c\_counter\_binary\_f ff (

.CLK(clk5),

.CE(CE),

.SCLR(SCLR),

.Q(f)

);

c\_counter\_binary\_s0 S0(

.CLK(f == 4'b0000),

.CE(CE),

.SCLR(SCLR),

.Q(s0)

);

c\_counter\_binary\_s1 S1(

.CLK(s0 == 4'b0000),

.CE(CE),

.SCLR(SCLR),

.Q(s1)

);

c\_counter\_binary\_m M(

.CLK(s1 == 4'b0000),

.CE(CE),

.SCLR(SCLR),

.Q(min)

);

endmodule

module lab8\_3\_1\_tb;

reg Clk, CE, SCLR;

wire [4:0] f, s0, s1, min;

lab8\_3\_1 DUT (Clk, CE, SCLR, f, s0, s1, min);

initial begin

Clk = 0;

forever

#5 Clk = ~Clk;

end

initial begin

CE = 1; SCLR = 0;

end

endmodule

Lab 8\_3\_2:

module lab8\_3\_2(

input Clk, CE, SCLR,

input [1:0] LOAD,

input [1:0] reload,

output reg [3:0] s0, s1, min

);

integer count = 0, negcounter = 0;

wire clk10;

wire lo;

reg Q;

clk\_wiz\_0 clock\_inst

(

.clk\_out1(clk10),

.locked(locked),

.clk\_in1(Clk)

);

always @ (posedge clk10)

begin

if(CE) begin

count <= count + 1;

if (count >= 5)

Q <= 1'b0;

else Q <= 1'b1;

if (count >= 10)

count <= 0;

end

end

always @ (LOAD or reload)

begin

if (CE) begin

//min = LOAD;

//s1 = 4'b0000;

//s0 = 4'b0000;

negcounter = (LOAD \* 60);

end

if (reload) begin

s0 = (LOAD % 10);

s1 = ((LOAD % 60) - s0)/10;

min = (LOAD - (LOAD % 60)) / 60;

end

end

always @ (Q)

begin

if (CE && (negcounter >= 0))

begin

negcounter = negcounter - 1;

end

end

always @ (posedge SCLR)

begin

negcounter = 0;

end

always @ (negcounter)

begin

if (CE) begin

s0 = (negcounter % 10);

s1 = ((negcounter % 60) - s0)/10;

min = (negcounter - (negcounter % 60)) / 60;

end

end

endmodule

module lab8\_3\_2\_tb;

reg Clk, CE, SCLR;

reg [1:0] LOAD, reload;

wire [3:0] s0, s1, min;

lab8\_3\_2 DUT (Clk, CE, SCLR, LOAD, reload, s0, s1, min);

initial begin

Clk = 0;

forever #5 Clk = ~Clk;

end

initial begin

LOAD = 2'b11; CE = 1; SCLR = 0;

#10000 reload = 1;

#50000 SCLR = 1;

#10000 SCLR = 0;

end

endmodule

Lab 8\_3\_3

module lab8\_3\_3(

input Clk, CE, SCLR,

input [1:0] LOAD,

output [3:0] s0, s1, m0, m1

);

integer count = 0, negcounter = 0;

wire clk10;

reg Q;

clk\_wiz\_0 clock\_inst

(

.clk\_out1(clk10),

.locked(locked),

.clk\_in1(Clk)

);

always @ (posedge clk10)

begin

if (CE)

begin

count <= count + 1;

if (count >= 5)

Q <= 1'b0;

else Q <= 1'b1;

if (count >= 10)

count <= 0;

end

end

c\_counter10 sec0 (.CLK(Q), .CE(CE), .SCLR(SCLR), .Q(s0));

c\_counter6 sec1 (.CLK(s0 == 4'b0000), .CE(CE), .SCLR(SCLR), .Q(s1));

c\_counter\_10 min0 (.CLK(s1 == 4'b0000), .CE(CE), .SCLR(SCLR), .Q(m0));

c\_counter\_6 min1 (.CLK(m0 == 4'b0000), .CE(CE), .SCLR(SCLR), .Q(m1));

endmodule

module lab8\_3\_3\_tb();

reg Clk, CE, SCLR;

reg [1:0] LOAD;

wire [3:0] s0, s1, m0, m1;

lab8\_3\_3 DUT (Clk, CE, SCLR, LOAD, s0, s1, m0, m1);

initial begin

Clk = 0;

forever #5 Clk = ~Clk;

end

initial begin

LOAD = 2'b11; CE = 1; SCLR = 0;

#50000 SCLR = 1;

#10000 SCLR = 0;

end

endmodule