Ryan Sullivan

U73687595

21 November 2019

EC311

Lab 2

**Objective:**

Create a counter with both manual and automatic counting functions with a universal reset and a seven-segment display.

**Methodology:**

Create multiple submodules that could be put together to create the desired product:

Counter:

The first iteration of the counter module that could only store up to one byte (8-bits), it would add a single bit to the total stored value for every time an increment is registered at a clock positive edge and would reset on a positive reset.

Counter16:

A modified version of the first counter module the only difference being that it could now store up to two bytes (16-bits) of data.

Debouncer:

A module used to account for the inaccuracy of a button push, it would register a single button push only after adding up enough of its signals and would not continually send a positive signal to the counter, thus negating the problem with having such a fast clock with such slow button pushes.

Clock Divider:

There were two separate clock divider modules, one for a fast clock (1 kHz) which would be used to refresh the seven-segment display, and one slow clock (1 Hz) used for the input for the automatic counting function. Both the clock dividers would only change their signal (from 1 to 0 or vice versa) once a certain number of clock pulses from the onboard clock had been registered, this number varies from clock to clock.

Display Control:

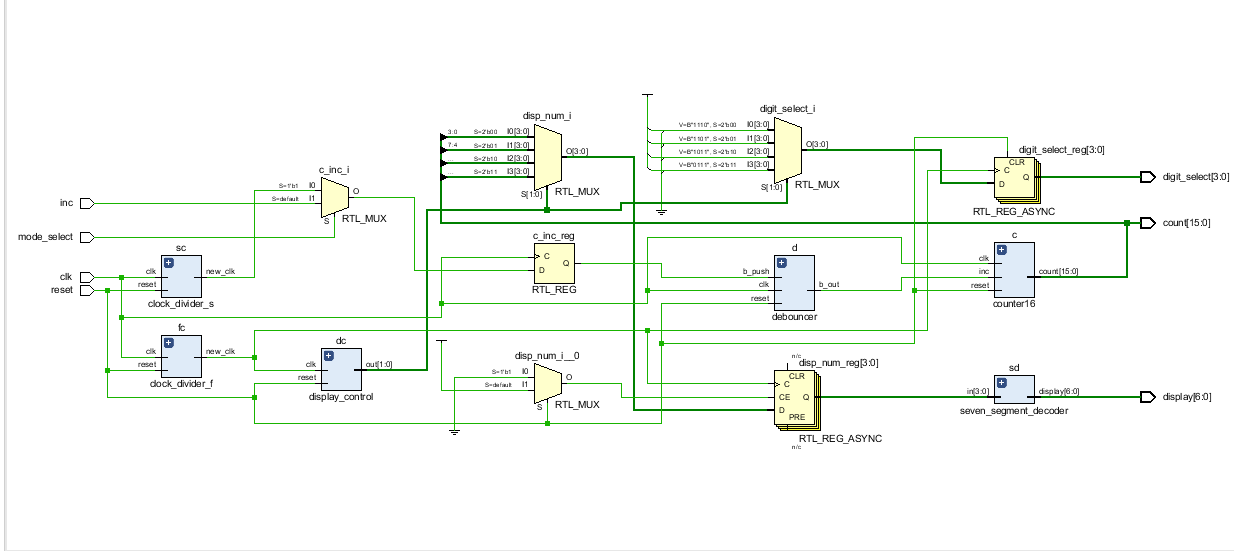
This was very simply a 2-bit counter that fed into 2:4 decoder, wherein the counter would register every clock pulse from the fast clock (1kHz) and increment by 1 bit, then the 2:4 decoder was fed into the seven-segment display to select a digit anode, refreshing each digit in sequence at a refresh rate of 1 kHz.

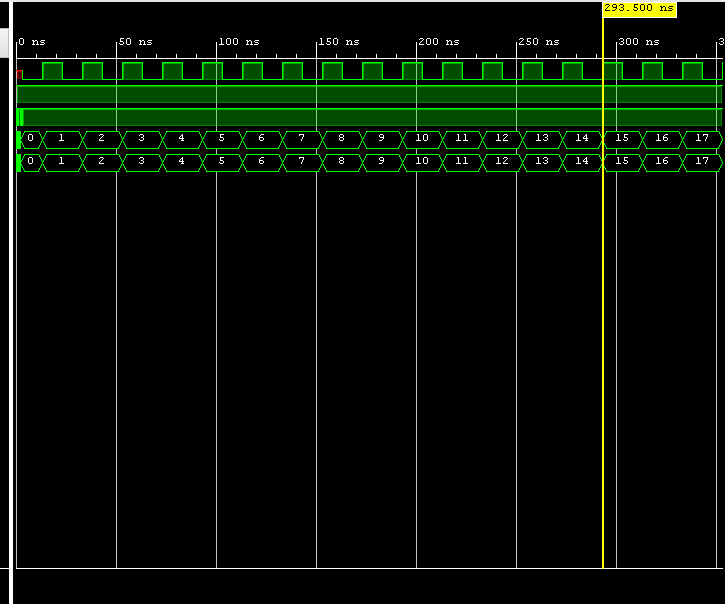
Seven-Segment Decoder:

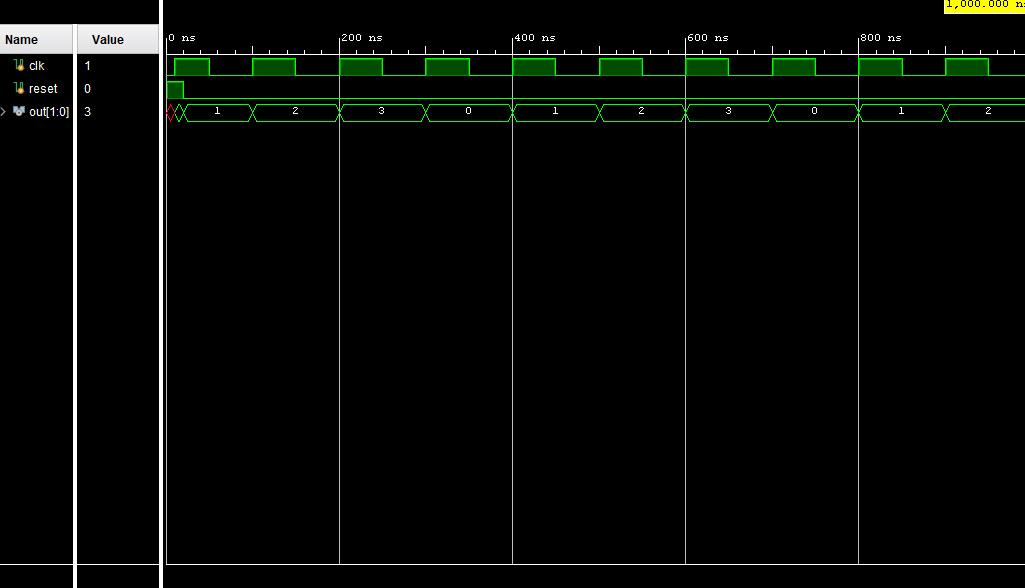
This was a 4:7 decoder that would take a 4-bit input and output a 7-bit bitstream that would be fed directly into the seven-segment display digits that would light up each segment in accordance with the 4-bit input (taken from the counter16 module in 4-bit chunks each corresponding to their own digit), which would translate the binary value into a more palatable hexadecimal value. The output is updated with every change to the input.

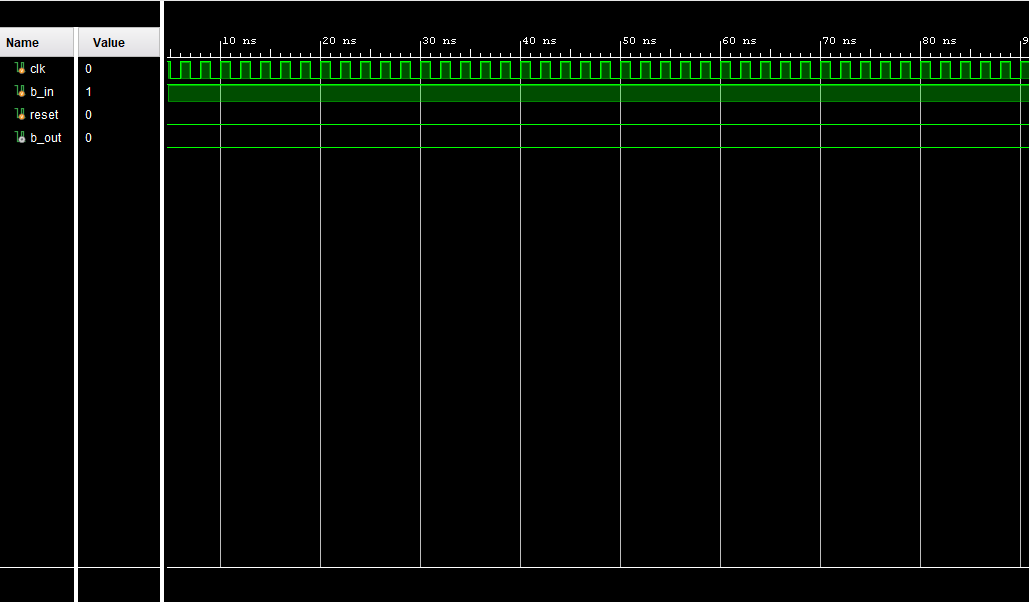
**Observation:**

Elaborated Design:

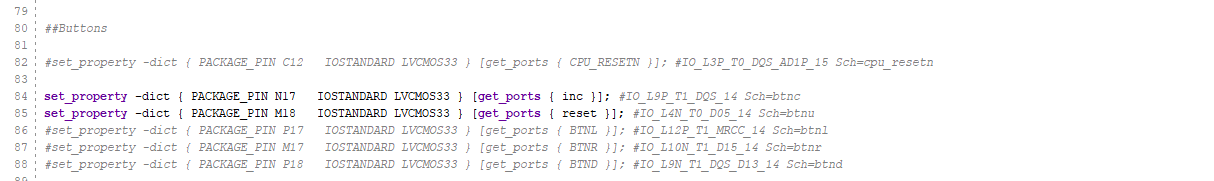
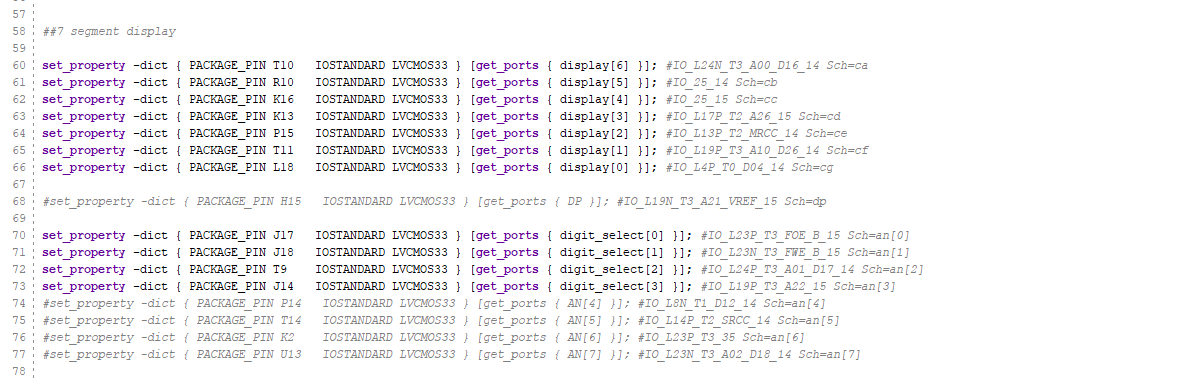
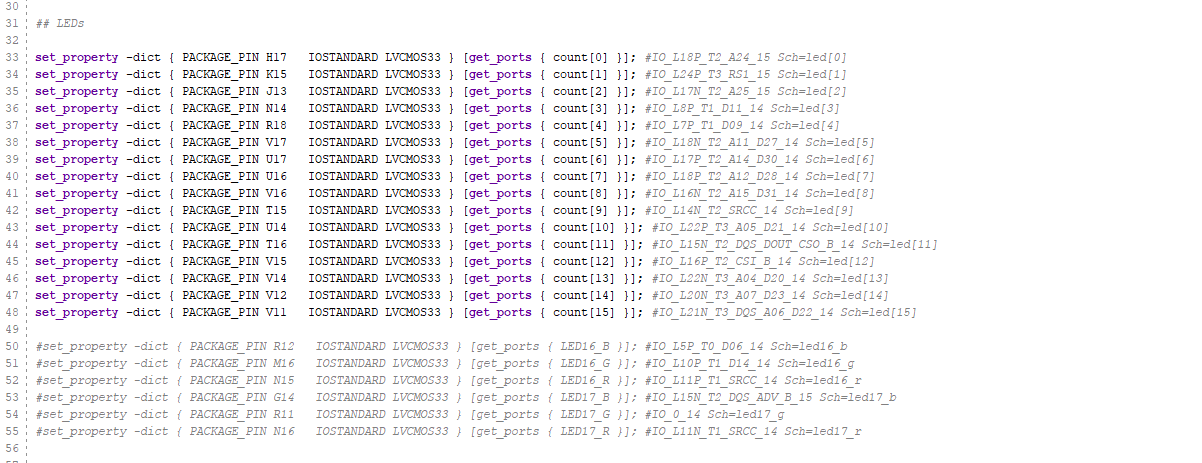
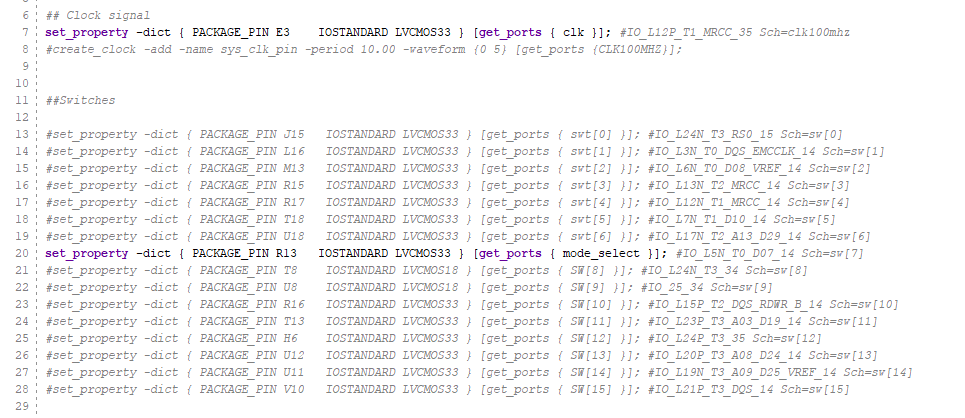


Counter Testbenches:

Display Control Testbench:

Debouncer Testbench:

Seven-Segment Decoder Testbench:

Constraints File:

**Conclusion:**

This lab has helped immensely in improving my understanding of Verilog and the workings of an FPGA board. The lab also taught me that a modular design is extremely efficient and the best in terms of testing and fixing any bugs that arise in project creation. Finally working with a physical board has improved my overall coding skillset and has better prepared me for the professional landscape in the future.

**Code:**

module top(

input clk,

input reset,

input inc,

output [7:0] count

);

wire b\_out;

debouncer d(

.clk(clk),

.reset(reset),

.b\_push(inc),

.b\_out(b\_out)

);

counter c(

.clk(clk),

.reset(reset),

.inc(b\_out),

.count(count)

);

endmodule

module top2(

input clk,

input reset,

input mode\_select,

input inc,

output [15:0] count,

output reg [3:0] digit\_select,

output [6:0] display

);

wire b\_out, f\_clk, s\_clk;

wire [1:0] dig\_sel;

reg [3:0] disp\_num;

reg c\_inc;

clock\_divider\_f fc(

.clk(clk),

.reset(reset),

.new\_clk(f\_clk)

);

clock\_divider\_s sc(

.clk(clk),

.reset(reset),

.new\_clk(s\_clk)

);

seven\_segment\_decoder sd(

.in(disp\_num),

.display(display)

);

display\_control dc(

.clk(f\_clk),

.reset(reset),

.out(dig\_sel)

);

debouncer d(

.clk(clk),

.reset(reset),

.b\_push(c\_inc),

.b\_out(b\_out)

);

counter16 c(

.clk(clk),

.reset(reset),

.inc(b\_out),

.count(count)

);

always@(posedge clk) begin

if(mode\_select) begin

c\_inc = s\_clk;

end else begin

c\_inc = inc;

end

end

always@(posedge f\_clk, negedge reset) begin

if(reset) begin

digit\_select <= 4'b0000;

end else if(f\_clk) begin

case(dig\_sel)

2'b00: begin

digit\_select <= 4'b1110;

disp\_num <= count[3:0];

end

2'b01: begin

digit\_select <= 4'b1101;

disp\_num <= count[7:4];

end

2'b10: begin

digit\_select <= 4'b1011;

disp\_num <= count[11:8];

end

2'b11: begin

digit\_select <= 4'b0111;

disp\_num <= count[15:12];

end

endcase

end

end

endmodule

module counter(

input clk,

input reset,

input inc,

output reg [7:0] count

);

always @(posedge clk, negedge reset) begin

if(reset) begin

count <= 8'b00000000;

end else if(clk) begin

if(inc) begin

count <= count + 8'b00000001;

end

end

end

endmodule

module debouncer(

input clk,

input reset,

input b\_push,

output reg b\_out

);

parameter MAX = 1600000;

reg [31:0] count;

always @(posedge clk) begin

if(reset)

count <= 0;

else if(b\_push)

count <= count +1;

else

count <= 0;

end

always @(posedge clk) begin

if(reset)

b\_out <= 1'b0;

else if(count == MAX)

b\_out <= 1'b1;

else

b\_out <= 1'b0;

end

endmodule

module counter16(

input clk,

input reset,

input inc,

output reg [15:0] count

);

always @(posedge clk, negedge reset) begin

if(reset) begin

count <= 16'b000000000000000;

end else if(clk) begin

if(inc) begin

count <= count + 16'b000000000000001;

end

end

end

endmodule

module clock\_divider\_f(

input clk,

input reset,

output reg new\_clk

);

reg [31:0] count;

parameter slow\_clk = 50000000;

parameter fast\_clk = 50000;

always @(posedge clk, negedge reset) begin

if(reset) begin

count <= 32'b0;

new\_clk <= 1'b0;

end

else if(count == fast\_clk -1)begin

count <= 32'b0;

new\_clk <= ~new\_clk;

end

else begin

count <= count + 32'b1;

new\_clk <= new\_clk;

end

end

endmodule

module clock\_divider\_s(

input clk,

input reset,

output reg new\_clk

);

reg [31:0] count;

parameter slow\_clk = 50000000;

parameter fast\_clk = 50000;

always @(posedge clk, negedge reset) begin

if(reset) begin

count <= 32'b0;

new\_clk <= 1'b0;

end

else if(count == slow\_clk -1)begin

count <= 32'b0;

new\_clk <= ~new\_clk;

end

else begin

count <= count + 32'b1;

new\_clk <= new\_clk;

end

end

endmodule

module seven\_segment\_decoder(

input [3:0] in,

output reg [6:0] display

);

always @(in)begin

case(in)

4'b0000: display <= 7'b0000001;

4'b0001: display <= 7'b1001111;

4'b0010: display <= 7'b0010010;

4'b0011: display <= 7'b0000110;

4'b0100: display <= 7'b1001100;

4'b0101: display <= 7'b0100100;

4'b0110: display <= 7'b0100000;

4'b0111: display <= 7'b0001111;

4'b1000: display <= 7'b0000000;

4'b1001: display <= 7'b0000100;

4'b1010: display <= 7'b0001000;

4'b1011: display <= 7'b1100000;

4'b1100: display <= 7'b0110001;

4'b1101: display <= 7'b1000010;

4'b1110: display <= 7'b0110000;

4'b1111: display <= 7'b0111000;

endcase

end

endmodule

module display\_control(

input clk,

input reset,

output reg [1:0] out

);

always @(posedge clk, negedge reset) begin

if(reset) begin

out <= 2'b00;

end else if(clk) begin

out <= out + 2'b01;

end

end

endmodule

**Testbenches:**

module seven\_seg\_test();

reg[3:0] in;

wire[6:0] display;

seven\_segment\_decoder s(

.in(in),

.display(display)

);

initial begin

in <= 4'b0000;

end

always begin

#55 in <= in + 4'b1;

end

endmodule

module deb\_test();

reg clk, b\_in, reset;

wire b\_out;

debouncer d(

.clk(clk),

.b\_push(b\_in),

.reset(reset),

.b\_out(b\_out)

);

initial begin

b\_in <= 1'b1;

reset <= 1'b1;

#1 reset = 1'b0;

clk <= 1'b0;

forever begin

#1 clk = ~clk;

end

#1;

end

endmodule

module count\_test();

reg clk, inc, reset;

wire [7:0] count;

wire [15:0] count2;

counter c(

.clk(clk),

.inc(inc),

.reset(reset),

.count(count)

);

counter16 c2(

.clk(clk),

.inc(inc),

.reset(reset),

.count(count2)

);

initial begin

inc <= 1'b1;

#1 reset <= 1'b1;

#1 reset <= 1'b0;

#1 reset <= 1'b1;

clk <= 1'b0;

forever begin

#10 clk = ~clk;

end

end

endmodule

module display\_control(

input clk,

input reset,

output reg [1:0] out

);

always @(posedge clk, negedge reset) begin

if(reset) begin

out <= 2'b00;

end else if(clk) begin

out <= out + 2'b01;

end

end

endmodule