**Lab 6 Report**

**Name: Xander Bonner**

**Date: December 2nd, 2023**

**Checklist:**

**Part A –**

1. HLSM describing the system.

**Part B –**

1. Processor architecture with the datapath and controller FSM

**A computer diagram of a computer network

Description automatically generated with medium confidence**

**Part C –**

1. Verilog, Constraint, and Bitstream Files
2. Verilog code

module hexto7segment(

input [3:0] x,

output reg [6:0] r

);

always @(\*)

case(x)

4'b0000 : r = 7'b0000001;//0

4'b0001 : r = 7'b1001111;//1

4'b0010 : r = 7'b0010010;//2

4'b0011 : r = 7'b0000110;//3

4'b0100 : r = 7'b1001100;//4

4'b0101 : r = 7'b0100100;//5

4'b0110 : r = 7'b0100000;//6

4'b0111 : r = 7'b0001111;//7

4'b1000 : r = 7'b0000000;//8

4'b1001 : r = 7'b0000100;//9

endcase

endmodule

module time\_multiplexing\_main(

input clk,

input reset,

input Start\_Stop,

input [7:0] sw,

output [1:0] mode,

output [6:0] sseg,

output [3:0] an

);

wire [6:0] in0, in1, in2, in3;

wire rf\_clk;

wire time\_clk;

wire[3:0] display\_0; //least sig fig or right most

wire[3:0] display\_1;

wire[3:0] display\_2;

wire[3:0] display\_3; //most sig or left most

//Module Instantiation of hexto7segment module

hexto7segment c1 (.x(display\_0), .r(in0));

hexto7segment c2 (.x(display\_1), .r(in1));

hexto7segment c3 (.x(display\_2), .r(in2));

hexto7segment c4 (.x(display\_3), .r(in3));

//Module instantiation of the clock divider

clkdiv c5 (.clk(clk), .reset(reset), .rf\_clk(rf\_clk), .time\_clk(time\_clk));

//Module instatiation of the Stopwatch\_Timer

Stopwatch\_Timer c7(

.clk(time\_clk),

.Start\_Stop(Start\_Stop),

.reset(reset),

.mode(mode[1:0]),

.sw(sw[7:0]),

.display\_0(display\_0),

.display\_1(display\_1),

.display\_2(display\_2),

.display\_3(display\_3)

);

//Module instantiation of the multiplexer

time\_mux\_state\_machine c6(

.clk(rf\_clk),

.reset(reset),

.in0(in0),

.in1(in1),

.in2(in2),

.in3(in3),

.an(an),

.sseg(sseg));

Endmodule

module time\_mux\_state\_machine(

input clk,

input reset,

input [6:0] in0,

input [6:0] in1,

input [6:0] in2,

input [6:0] in3,

output reg [3:0] an,

output reg [6:0] sseg

);

reg [1:0] state;

reg [1:0] next\_state;

always @(\*) begin

case(state) //state tranisiton

2'b00: next\_state = 2'b01;

2'b01: next\_state = 2'b10;

2'b10: next\_state = 2'b11;

2'b11: next\_state = 2'b00;

endcase

end

always @(\*) begin

case(state) //multiplexer

2'b00: sseg = in0;

2'b01: sseg = in1; //\*\*\*\*\*\*\*\*\*\*check

2'b10: sseg = in2; //\*\*\*\*\*\*\*\*\*\*check

2'b11: sseg = in3; //\*\*\*\*\*\*\*\*\*\*check

endcase

case(state) //decoder

2'b00: an = 4'b1110;

2'b01: an = 4'b1101; //\*\*\*\*\*\*\*\*\*\*check

2'b10: an = 4'b1011; //\*\*\*\*\*\*\*\*\*\*check

2'b11: an = 4'b0111; //\*\*\*\*\*\*\*\*\*\*check

endcase

end

always @(posedge clk or posedge reset) begin

if(reset)

state <= 2'b00;

else

state <= next\_state;

end

endmodule

module clkdiv(

input clk,

input reset,

output rf\_clk,

output time\_clk

);

reg refresh;

reg time\_clock;

reg[16:0] rf;

reg[19:0] COUNT;

assign time\_clk = time\_clock;

assign rf\_clk = refresh;

always @(posedge clk)

begin

if(rf <10000)

begin

rf<= rf + 1;

end

else begin

refresh <= ~refresh;

rf <= 0;

end

end

always @(posedge clk)

begin

if(COUNT<10000)

begin

COUNT <= COUNT + 1;

end

else begin

time\_clock <= ~time\_clock;

COUNT <= 0;

end

end

endmodule

module Stopwatch\_Timer(

input reset,

input clk,

input Start\_Stop,

input[7:0] sw,

input[1:0] mode,

output reg[3:0] display\_3, //left digit

output reg[3:0] display\_2, //mid left

output reg[3:0] display\_1, //mid right

output reg[3:0] display\_0 //right digit

);

reg Start\_Stop\_grab; //captures vaule of Start\_Stop

reg strstp = 1; //register that controls Start\_Stop

reg done; //flag used to indicate stopwatch has stop counting

reg idle = 1; //flag used for idle state

always @(posedge clk) begin //captures Start\_Stop signal on falling edge

Start\_Stop\_grab <= Start\_Stop;

if(Start\_Stop\_grab && !Start\_Stop)

strstp<=~strstp;

end

always @ (posedge clk) begin

//Mode 1//

if(mode == 2'b00) begin

if(strstp == 1 && reset == 1) //when both strstp and reset is high than we need to reset counters

begin

display\_0<=0; //counter 0

display\_1<=0; //counter 0

display\_2<=0; //counter 0

display\_3=0; //counter 0

done = 0;

end

//if strstp is high and reset is not zero than we need to store the current count

else if(strstp == 1 && reset!= 0)

begin

display\_0<=0; //display 0 stored

display\_1<=0; //display 1 stored

display\_2<=0; //display 2 stored

display\_3<=0; //display 3 stored

end

//if srtstp is not high and done is not high than we need to increment the counters, which involves carry-over for each digit

else if(strstp != 1 && done != 1) //checks to see if stopwatch has stop, if not than the count is not complete

begin

if(display\_0 == 9) begin //start from least signicant digit to most, so display\_0 needs to be counted from 0-9

display\_0 <= 0; //if we hit 0 than we need to move onto display 1 since we've carried over.

if(display\_1 <= 9) begin //move to next digit, or second sig, count until 9

display\_1 <= 0; //if we hit 0 than we have carried over, trigger display\_2

if(display\_2 == 9) begin //move to next digit, or third sig

display\_2 <= 0; //if we hit 0, than we have carried over, trigger display\_3

if(display\_3 == 9) begin //count until we have hit 9, once we hit 9 than we have hit the limit allowed by our counter so all digits carry over to 0.

display\_2 <= 9;

display\_1 <= 9;

display\_0 <= 9;

done = 1; //flag saying we've hit roll over

end else

display\_3 <= display\_3 + 1; //increment

end else

display\_2 <= display\_2 + 1; //increment

end else

display\_1 <= display\_1 + 1; //increment

end else

display\_0 <= display\_0 + 1; //increment

end

end

//Mode 2//

if(mode == 2'b01) begin

if(strstp == 1 && reset == 0 && idle == 1) //if strstp and idle are high and reset is low than reset display 0 & 1 to zero and then display 2 & 3 to swithces

begin

display\_0<=0; //counter is 0

display\_1<=0; //counter is 0

display\_2<=sw[3:0]; //counter is 0

display\_3<=sw[7:4]; //counter is 0

done = 0;

end

//if strstp is high and reset is high than store the current count into the displays

if(strstp == 1 && reset == 1)

begin

//now we need to store the current count

display\_0<=0; //display 0 counter is now stored

display\_1<=0; //1 counter is now stored

display\_2<=sw[3:0]; //2 counter is now stored

display\_3<=sw[7:4]; //3 counter is now stored

done = 0; //not finished

idle = 1; //stopwatch is in idle state and not currenly counting

end

//if strstp is high and reset is not zero than we are idling but not activley counting so we need to store the current state into displays

else if(strstp == 1 && reset != 0)

begin

display\_0<= display\_0; //store current state

display\_1<= display\_1; //store current state

display\_2<= display\_2; //store current state

display\_3<= display\_3; //store current state

idle = 1;

end

//if the strstp is not high and the reset is also not high than we need to start the stopwatch and increment counters

else if(strstp != 1 && reset != 1)

begin

idle = 0; //set idle to zero because we are currently counting

if(display\_0 == 9) begin //start incrementing counter until we get to 9

display\_0 <= 0; //if we get to 0 then we have rolled over and we need to set display to zero and initiate next sig fig

if(display\_1 <= 9) begin //start incrementing counter until we get to 9

display\_1 <= 0; //we got to 0 so we have rolled over, set to 0 and move to next sig fig

if(display\_2 == 9) begin //start incrementing counter

display\_2 <= 0; //carried over, set to 0 and move to next

if(display\_3 == 9) begin //if display\_3 is set to 9 than we have reached the end and we can set all to 9999 to show we've reached the end

display\_2 <= 9; //set to 9

display\_1 <= 9; //set to 9

display\_0 <= 9; //set to 9

done = 1; //if we get to 9999 than we have reached the end and the done flag can be set 1

end else

display\_3 <= display\_3 + 1; //increment most sig fig

end else

display\_2 <= display\_2 + 1; //increment display 2

end else

display\_1 <= display\_1 + 1; //increment display 1

end else

display\_0 <= display\_0 + 1; //increment display 0 or least sig fig

end

end

//Mode 3//

if(mode == 2'b10) begin

if(strstp == 1 && reset == 1) //if both strstp and rest are high than set all displays to 9

begin

display\_0<=9; //display 0 set to 9

display\_1<=9; //display 1 set to 9

display\_2<=9; //display 2 set to 9

display\_3<=9; //display 3 set to 9

end

else if(strstp == 1 && reset!= 0) //if strstp is set to high and reset is not zero than store current count

begin

//store current count

display\_0 <= display\_0; //display 0 is stored to current vaule

display\_1 <= display\_1; //display 1 is stored to current vaule

display\_2 <= display\_2; //display 2 is stored to current vaule

display\_3 <= display\_3; //display 3 is stored to current vaule

end

else if(strstp != 1 && done == 0) //if strstp is not set to high and done is 0 than we need to decrement counters and start counting down

begin

if(display\_0 == 0) begin //start decrementing display counter until we hit 0

display\_0 <= 9; //if we have hit 9 than we have carried over and we can move onto next sig fig

if(display\_1 == 0) begin //decrement counter until 0

display\_1 <= 9; //if we hit 9 than we have carried over and move on to next fig

if(display\_2 == 0) begin // start decrementing until 0

display\_2 <= 9; //carry over ocurred now move onto next sig fig

if(display\_3 == 0)begin //start decrementing final sig fig, if we hit 0 than we have hit the end of 0000

display\_0 <= 0; //set to 0, end

display\_1 <= 0; //set to 0, end

display\_2 <= 0; //set to 0, end

display\_3 <= 0; //set to 0, end

done = 1; //set done flag to 0

end else

display\_3 <= display\_3 - 1; //decrement counter

end else

display\_2 <= display\_2 - 1; //decrement counter

end else

display\_1 <= display\_1 -1; //decrement counter

end else

display\_0 <= display\_0 - 1; //decrement counter

end

end

//Mode 4//

if(mode == 2'b11) begin

if(strstp == 1 && reset == 0 && idle == 1) begin //if strstp is high and idle is high and reset is 0 than reset counters to begin

display\_0 <= 0; //set display 0 to 0

display\_1 <= 0; //set display 1 to 0

display\_2 <= sw[3:0]; //set display 2 to switches 3:0

display\_3 <= sw[7:4]; //set display 3 to swtiches 3:0

end

else if(strstp == 1 && reset != 0) begin //if strstp is high and reset is not 0 than store current count because we are idling

//store previous count

display\_0 <= display\_0; //store display 0

display\_1 <= display\_1; //store display 1

display\_2 <= display\_2; //store display 2

display\_3 <= display\_3; //store display 3

idle = 1; //set idle to high because we still counting but not currently counting

end

else if(strstp != 1 && done == 0) //if strstp is not set to high and done is 0 than we need to decrement counters and start counting down

begin

if(display\_0 == 0) begin //start decrementing display counter until we hit 0

display\_0 <= 9; //if we have hit 9 than we have carried over and we can move onto next sig fig

if(display\_1 == 0) begin //decrement counter until 0

display\_1 <= 9; //if we hit 9 than we have carried over and move on to next fig

if(display\_2 == 0) begin // start decrementing until 0

display\_2 <= 9; //carry over ocurred now move onto next sig fig

if(display\_3 == 0)begin //start decrementing final sig fig, if we hit 0 than we have hit the end of 0000

display\_0 <= 0; //set to 0, end

display\_1 <= 0; //set to 0, end

display\_2 <= 0; //set to 0, end

display\_3 <= 0; //set to 0, end

done = 1; //set done flag to 0

end else

display\_3 <= display\_3 - 1; //decrement counter

end else

display\_2 <= display\_2 - 1; //decrement counter

end else

display\_1 <= display\_1 -1; //decrement counter

end else

display\_0 <= display\_0 - 1; //decrement counter

end

end

end

endmodule

module timer(

input clk,

input reset,

output reg[3:0] display\_0,

output reg[3:0] display\_1,

output reg[3:0] display\_2,

output reg[3:0] display\_3

);

reg[8:0] counter = 1'd9999; //this puts the counter to 9999 so when the timer is called we will have the inital vaule

reg Start\_Stop;

always @(\*) begin

if(Start\_Stop == 1 && reset == 1) //if both the Start\_Stop and Reset are high than we can store 9's since we have not started yet

begin

display\_0 <= 1'd9;

display\_1 <= 1'd9;

display\_2 <= 1'd9;

display\_3 <= 1'd9;

end

else if(Start\_Stop == 1) //if start\_Stop is high than stop the count and store the current values

begin

display\_0 <= display\_0;

display\_1 <= display\_1;

display\_2 <= display\_2;

display\_3 <= display\_3;

end

else if(Start\_Stop != 1) //if start\_stop is not high than we can start/continue the countdown timer and begin counting

begin

if(display\_0 == 9) //display 0 is set to 9, start decrementing

begin

display\_0 <= 0; //if we hit 0 than a carry over has occurred and we can start the next sig fig

if(display\_1 == 9) //display\_ 1 is set to 9, start decrementing

begin

display\_1 <= 0; //display\_1 has hit 0 so a carry over has occured, again start decrementing next sig fig

if(display\_2 == 9) //display\_2 has been initiated so start decrementing

begin

display\_2 <= 0; //display\_2 has hit zero, therefore start decrementing next sig fig

if(display\_3 == 9) //start decremeting counter

display\_3 <= 0; //zero has hit therefore we have hit the end

else

display\_3 <= display\_3 -1; //decrement

end else

display\_2 <= display\_2 -1; //decrement

end else

display\_1 <= display\_1 -1; //decrement

end else

display\_0 <= display\_0 -1; //decrement

end

end

endmodule

##Constraint

# Clock signal - Uncomment if needed (will be used in future labs)

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk]

## Switches

set\_property PACKAGE\_PIN V17 [get\_ports {mode[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {mode[0]}]

set\_property PACKAGE\_PIN V16 [get\_ports {mode[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {mode[1]}]

set\_property PACKAGE\_PIN V2 [get\_ports {sw[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[0]}]

set\_property PACKAGE\_PIN T3 [get\_ports {sw[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[1]}]

set\_property PACKAGE\_PIN T2 [get\_ports {sw[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[2]}]

set\_property PACKAGE\_PIN R3 [get\_ports {sw[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[3]}]

set\_property PACKAGE\_PIN W2 [get\_ports {sw[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[4]}]

set\_property PACKAGE\_PIN U1 [get\_ports {sw[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[5]}]

set\_property PACKAGE\_PIN T1 [get\_ports {sw[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[6]}]

set\_property PACKAGE\_PIN R2 [get\_ports {sw[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[7]}]

##7 segment display

set\_property PACKAGE\_PIN W7 [get\_ports {sseg[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sseg[6]}]

set\_property PACKAGE\_PIN W6 [get\_ports {sseg[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sseg[5]}]

set\_property PACKAGE\_PIN U8 [get\_ports {sseg[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sseg[4]}]

set\_property PACKAGE\_PIN V8 [get\_ports {sseg[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sseg[3]}]

set\_property PACKAGE\_PIN U5 [get\_ports {sseg[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sseg[2]}]

set\_property PACKAGE\_PIN V5 [get\_ports {sseg[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sseg[1]}]

set\_property PACKAGE\_PIN U7 [get\_ports {sseg[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sseg[0]}]

set\_property PACKAGE\_PIN U2 [get\_ports {an[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[0]}]

set\_property PACKAGE\_PIN U4 [get\_ports {an[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[1]}]

set\_property PACKAGE\_PIN V4 [get\_ports {an[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[2]}]

set\_property PACKAGE\_PIN W4 [get\_ports {an[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {an[3]}]

##Buttons

set\_property PACKAGE\_PIN U18 [get\_ports Start\_Stop]

set\_property IOSTANDARD LVCMOS33 [get\_ports Start\_Stop]

set\_property PACKAGE\_PIN T18 [get\_ports reset]

set\_property IOSTANDARD LVCMOS33 [get\_ports reset]