**CS M152A / EE M116L**

**Lab 4: FSM and Stopwatch**

**Scott Shi**

**Minh Le**

**FSM Pattern Recognizer**

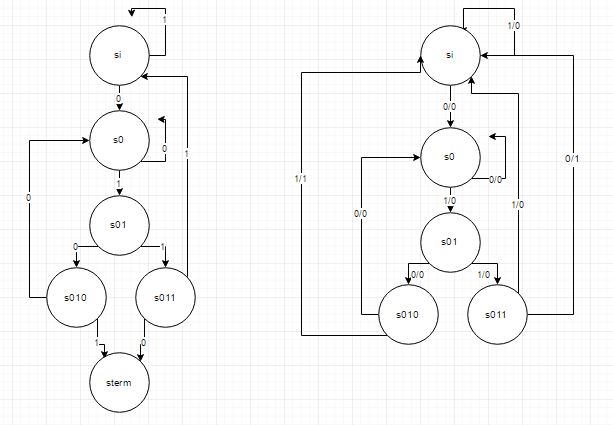
**I. INTRODUCTION**

**Summary:**

The first part of the lab, we implemented a pattern detector using 3 different encodings (1hot, grey, and binary) in both a Mealy and Moore sequential system for a total of 6 different implementations. Once the pattern reached either ‘0110’ or ‘0101’, we would set ‘valid’ to high.

**Schematic and Modules Description of FSM Pattern Recognizer**

**Moore Mealy**



**High Level Description of FSM Pattern Recognizer**

We implemented 6 different pattern recognizers. As shown above, we wanted to recognize either 0110 or 0101 and set valid to high after either of the patterns are detected. Above is the Mealy and Moore schematic of the state transition table. The implementations with different encodings (1-hot, Grey, Binary) can be seen in the code section below.

**II. DESIGN DESCRIPTION**

**TESTBENCH and UCF for FSM Pattern Recognizer**

======================================

\*TB.v -- TESTBENCH USED FOR ALL \*.v FILES

======================================

`timescale 1ns / 1ps

module FSM\_TB;

// Inputs

reg in;

reg clk;

reg rst;

// Outputs

wire valid;

wire [4:0] curr\_st;

// Instantiate the Unit Under Test (UUT)

FSM uut ( // change this according to which implementation you want to test

.in(in),

.clk(clk),

.rst(rst),

.valid(valid),

.curr\_st(curr\_st)

);

initial begin

// Initialize Inputs

in = 0;

clk = 0;

rst = 1;

#1 rst = 0;

//simulating inputs

//simulate success -- 0101

#20 in = 1;//back to si

#20 in = 0;//to s0

#20 in = 0;//back to s0

#20 in = 1;//to s01

#20 in = 0;//to s010

#20 in = 1;//success

#100

//simulate success -- 0110

#20 in = 0;//to s0

#20 in = 1;//to s01

#20 in = 1;//to s011

#20 in = 1;//to si

#20 in = 0;//to s0

#20 in = 1;//to s01

#20 in = 1;//to s011

#20 in = 0;//success

// Wait 100 ns for global reset to finish

#10000 $finish;

// Add stimulus here

end

always @(\*)

#1 clk <= ~clk;

endmodule

====

\*.ucf

====

## Clock signal

NET "clk" LOC = "V10" | IOSTANDARD = "LVCMOS33"; #Bank = 2, pin name = IO\_L30N\_GCLK0\_USERCCLK, Sch name = GCLK

Net "clk" TNM\_NET = sys\_clk\_pin;

TIMESPEC TS\_sys\_clk\_pin = PERIOD sys\_clk\_pin 100000 kHz;

## Leds

NET "curr\_st<0>" LOC = "U16" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L2P\_CMPCLK, Sch name = LD0

NET "curr\_st<1>" LOC = "V16" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L2N\_CMPMOSI, Sch name = LD1

NET "curr\_st<2>" LOC = "U15" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L5P, Sch name = LD2

#NET "Led<3>" LOC = "V15" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L5N, Sch name = LD3

#NET "Led<4>" LOC = "M11" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L15P, Sch name = LD4

#NET "Led<5>" LOC = "N11" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L15N, Sch name = LD5

#NET "Led<6>" LOC = "R11" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L16P, Sch name = LD6

NET "valid" LOC = "T11" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L16N\_VREF, Sch name = LD7

## Switches

NET "in" LOC = "T10" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L29N\_GCLK2, Sch name = SW0

## Buttons

NET "rst" LOC = "B8" | IOSTANDARD = "LVCMOS33"; #Bank = 0, Pin name = IO\_L33P, Sch name = BTNS

**MEALY BINARY**

================

FSMMealyBinary.v

================

`timescale 1ns / 1ps

`define N\_BIT 29

`define DIVIDE 399999999

`define ST\_BIT\_CNT 3

//mealy implementation

module FSM(

in, //intput line -- switch

clk,

valid, //output line -- led

curr\_st, //led

rst

);

//clock divider

input clk;

reg trig;

reg [`N\_BIT-1:0] in\_cnt;

//input signals

input in;

input rst;

//ouput bit and state

output valid;

output reg[`ST\_BIT\_CNT-1:0] curr\_st;

//binary encoding

parameter si = 3'd0;

parameter s0 = 3'd1;

parameter s01 = 3'd2;

parameter s010 = 3'd3;

parameter s011 = 3'd4;

//clock divider logic

always @ (posedge rst, posedge clk) begin

if (rst | trig)

begin

in\_cnt <= `N\_BIT'b0;

trig <= 1'b0;

end

else

begin

in\_cnt <= in\_cnt +1'b1;

trig <= (in\_cnt == `N\_BIT'd`DIVIDE);

end

end

//next state computation

always @ (posedge trig, posedge rst)begin

if (rst)

curr\_st <= si;

else begin

case (curr\_st)

si: curr\_st <= in ? si : s0;

s0: curr\_st <= in ? s01 : s0;

s01: curr\_st <= in ? s011 : s010;

s010: curr\_st <= in ? si : si;

s011: curr\_st <= in ? si : si;

endcase

end

end

//output

assign valid = ((curr\_st == s010)&in) | ((curr\_st == s011)&(!in));

endmodule

**Synthesis Report**

========================================

\* Synthesis Report \*

========================================

Macro Statistics

# Adders/Subtractors : 1

29-bit adder : 1

# Registers : 2

1-bit register : 1

29-bit register : 1

# Multiplexers : 2

1-bit 2-to-1 multiplexer : 1

29-bit 2-to-1 multiplexer : 1

# FSMs : 1

# Registers : 35

Flip-Flops : 35

========================================

\* Design Summary \*

========================================

Primitive and Black Box Usage:

------------------------------

# BELS : 98

# GND : 1

# INV : 1

# LUT3 : 31

# LUT4 : 2

# LUT5 : 1

# LUT6 : 5

# MUXCY : 28

# XORCY : 29

# FlipFlops/Latches : 35

# FDC : 35

# Clock Buffers : 1

# BUFGP : 1

# IO Buffers : 6

# IBUF : 2

# OBUF : 4

Device utilization summary:

---------------------------

Selected Device : 6slx16csg324-3

Slice Logic Utilization:

Number of Slice Registers: 35 out of 18224 0%

Number of Slice LUTs: 40 out of 9112 0%

Number used as Logic: 40 out of 9112 0%

Slice Logic Distribution:

Number of LUT Flip Flop pairs used: 42

Number with an unused Flip Flop: 7 out of 42 16%

Number with an unused LUT: 2 out of 42 4%

Number of fully used LUT-FF pairs: 33 out of 42 78%

Number of unique control sets: 2

IO Utilization:

Number of IOs: 7

Number of bonded IOBs: 7 out of 232 3%

Specific Feature Utilization:

Number of BUFG/BUFGCTRLs: 1 out of 16 6%

**Device Utilization Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Slice Logic Utilization | Used | Available | Utilization |
| # Slice Registers | 35 | 18,224 | 1% |
| # used as Flip Flops | 35 |  |  |
| # Slice LUTs | 38 | 9,112 | 1% |
| # used as logic | 38 | 9,112 | 1% |
| # using O6 output only | 8 |  |  |
| # using O5 and O6 | 30 |  |  |
| # of occupied Slices | 15 | 2,278 | 1% |
| # of MUXCYs used | 32 | 4,556 | 1% |
| # of LUT Flip Flop pairs used | 39 |  |  |
| # with and unused Flip Flop | 4 | 39 | 12% |
| # with an unused LUT | 1 | 39 | 2% |
| # fully used LUT-FF pairs | 33 | 39 | 84% |
| # unique control sets | 2 |  |  |
| # of slice register sites lost to control set restrictions | 5 | 18,224 | 1% |
| # of bonded IOBs | 7 | 232 | 3% |
| # of LOCed IOBs | 7 | 7 | 100% |
| # of BUFG/BUFGMUXs | 1 | 16 | 6% |
| # used as BUFGs | 1 |  |  |
| Average Fanout of Non-Clock Nets | 2.50 |  |  |

**Maximum Clock Frequency: 325.098 MHz**

**MEALY ONEHOT**

================

FSMMealyOnehot.v

================

`timescale 1ns / 1ps

`define N\_BIT 29

`define DIVIDE 399999999

`define ST\_BIT\_CNT 5

//mealy implementation

module FSM(

in, //intput line -- switch

clk,

valid, //output line -- led

curr\_st, //led

rst

);

//clock divider

input clk;

reg trig;

reg [`N\_BIT-1:0] in\_cnt;

//input signals

input in;

input rst;

//ouput bit and state

output valid;

output reg[`ST\_BIT\_CNT-1:0] curr\_st;

//one hot bit encoding

parameter si = 5'b00001;

parameter s0 = 5'b00010;

parameter s01 = 5'b00100;

parameter s010 = 5'b01000;

parameter s011 = 5'b10000;

//clock divider logic

always @ (posedge rst, posedge clk) begin

if (rst | trig)

begin

in\_cnt <= `N\_BIT'b0;

trig <= 1'b0;

end

else

begin

in\_cnt <= in\_cnt +1'b1;

trig <= (in\_cnt == `N\_BIT'd`DIVIDE);

end

end

//next state computation

always @ (posedge trig, posedge rst)begin

if (rst)

curr\_st <= si;

else begin

case (curr\_st)//0110 or 0101

si: curr\_st <= in ? si : s0;

s0: curr\_st <= in ? s01 : s0;

s01: curr\_st <= in ? s011 : s010;

s010: curr\_st <= in ? si : si;

s011: curr\_st <= in ? si : si;

endcase

end

end

//output

assign valid = ((curr\_st == s010)&in) | ((curr\_st == s011)&(!in));

endmodule

**Synthesis Report**

========================================

\* Synthesis Report \*

========================================

Macro Statistics

# Adders/Subtractors : 1

29-bit adder : 1

# Registers : 2

1-bit register : 1

29-bit register : 1

# Multiplexers : 2

1-bit 2-to-1 multiplexer : 1

29-bit 2-to-1 multiplexer : 1

# FSMs : 1

# Registers : 36

Flip-Flops : 36

========================================

\* Design Summary \*

========================================

Primitive and Black Box Usage:

------------------------------

# BELS : 100

# GND : 1

# INV : 1

# LUT2 : 3

# LUT3 : 31

# LUT4 : 1

# LUT5 : 1

# LUT6 : 5

# MUXCY : 28

# XORCY : 29

# FlipFlops/Latches : 36

# FDC : 35

# FDP : 1

# Clock Buffers : 1

# BUFGP : 1

# IO Buffers : 8

# IBUF : 2

# OBUF : 6

Device utilization summary:

---------------------------

Selected Device : 6slx16csg324-3

Slice Logic Utilization:

Number of Slice Registers: 36 out of 18224 0%

Number of Slice LUTs: 42 out of 9112 0%

Number used as Logic: 42 out of 9112 0%

Slice Logic Distribution:

Number of LUT Flip Flop pairs used: 43

Number with an unused Flip Flop: 7 out of 43 16%

Number with an unused LUT: 1 out of 43 2%

Number of fully used LUT-FF pairs: 35 out of 43 81%

Number of unique control sets: 2

IO Utilization:

Number of IOs: 9

Number of bonded IOBs: 9 out of 232 3%

Specific Feature Utilization:

Number of BUFG/BUFGCTRLs: 1 out of 16 6%

**Device Utilization Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Slice Logic Utilization | Used | Available | Utilization |
| # Slice Registers | 36 | 18,224 | 1% |
| # used as Flip Flops | 36 |  |  |
| # Slice LUTs | 39 | 9,112 | 1% |
| # used as logic | 39 | 9,112 | 1% |
| # using O6 output only | 8 |  |  |
| # using O5 and O6 | 31 |  |  |
| # of occupied Slices | 17 | 2,278 | 1% |
| # of MUXCYs used | 32 | 4,556 | 1% |
| # of LUT Flip Flop pairs used | 40 |  |  |
| # with and unused Flip Flop | 6 | 39 | 12% |
| # with an unused LUT | 1 | 39 | 2% |
| # fully used LUT-FF pairs | 33 | 39 | 84% |
| # unique control sets | 2 |  |  |
| # of slice register sites lost to control set restrictions | 4 | 18,224 | 1% |
| # of bonded IOBs | 9 | 232 | 3% |
| # of LOCed IOBs | 7 | 7 | 100% |
| # of BUFG/BUFGMUXs | 1 | 16 | 6% |
| # used as BUFGs | 1 |  |  |
| Average Fanout of Non-Clock Nets | 2.53 |  |  |

**Maximum Clock Frequency: 330.688 MHz**

**MEALY GREY**

================

FSMMealyGrey.v

================

`timescale 1ns / 1ps

`define N\_BIT 29

`define DIVIDE 399999999

`define ST\_BIT\_CNT 3

//mealy implementation

module FSM(

in, //intput line -- switch

clk,

valid, //output line -- led

curr\_st, //led

rst

);

//clock divider

input clk;

reg trig;

reg [`N\_BIT-1:0] in\_cnt;

//input signals

input in;

input rst;

//ouput bit and state

output valid;

output reg[`ST\_BIT\_CNT-1:0] curr\_st;

// grey encoding

parameter si = 3'b000;

parameter s0 = 3'b001;

parameter s01 = 3'b011;

parameter s010 = 3'b010;

parameter s011 = 3'b110;

//clock divider logic

always @ (posedge rst, posedge clk) begin

if (rst | trig)

begin

in\_cnt <= `N\_BIT'b0;

trig <= 1'b0;

end

else

begin

in\_cnt <= in\_cnt +1'b1;

trig <= (in\_cnt == `N\_BIT'd`DIVIDE);

end

end

//next state computation

always @ (posedge trig, posedge rst)begin

if (rst)

curr\_st <= si;

else begin

case (curr\_st)

si: curr\_st <= in ? si : s0;

s0: curr\_st <= in ? s01 : s0;

s01: curr\_st <= in ? s011 : s010;

s010: curr\_st <= in ? si : si;

s011: curr\_st <= in ? si : si;

endcase

end

end

//output

assign valid = ((curr\_st == s010)&in) | ((curr\_st == s011)&(!in));

endmodule

**Synthesis Report**

========================================

\* Synthesis Report \*

========================================

Macro Statistics

# Adders/Subtractors : 1

29-bit adder : 1

# Registers : 2

1-bit register : 1

29-bit register : 1

# Multiplexers : 2

1-bit 2-to-1 multiplexer : 1

29-bit 2-to-1 multiplexer : 1

# FSMs : 1

# Registers : 35

Flip-Flops : 35

========================================

\* Design Summary \*

========================================

Primitive and Black Box Usage:

------------------------------

# BELS : 98

# GND : 1

# INV : 1

# LUT3 : 32

# LUT4 : 1

# LUT5 : 1

# LUT6 : 5

# MUXCY : 28

# XORCY : 29

# FlipFlops/Latches : 35

# FDC : 35

# Clock Buffers : 1

# BUFGP : 1

# IO Buffers : 6

# IBUF : 2

# OBUF : 4

Device utilization summary:

---------------------------

Selected Device : 6slx16csg324-3

Slice Logic Utilization:

Number of Slice Registers: 35 out of 18224 0%

Number of Slice LUTs: 40 out of 9112 0%

Number used as Logic: 40 out of 9112 0%

Slice Logic Distribution:

Number of LUT Flip Flop pairs used: 42

Number with an unused Flip Flop: 7 out of 42 16%

Number with an unused LUT: 2 out of 42 4%

Number of fully used LUT-FF pairs: 33 out of 42 78%

Number of unique control sets: 2

IO Utilization:

Number of IOs: 7

Number of bonded IOBs: 7 out of 232 3%

Specific Feature Utilization:

Number of BUFG/BUFGCTRLs: 1 out of 16 6%

**Device Utilization Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Slice Logic Utilization | Used | Available | Utilization |
| # Slice Registers | 35 | 18,224 | 1% |
| # used as Flip Flops | 35 |  |  |
| # Slice LUTs | 38 | 9,112 | 1% |
| # used as logic | 38 | 9,112 | 1% |
| # using O6 output only | 8 |  |  |
| # using O5 and O6 | 30 |  |  |
| # of occupied Slices | 14 | 2,278 | 1% |
| # of MUXCYs used | 32 | 4,556 | 1% |
| # of LUT Flip Flop pairs used | 39 |  |  |
| # with and unused Flip Flop | 5 | 39 | 12% |
| # with an unused LUT | 1 | 39 | 2% |
| # fully used LUT-FF pairs | 33 | 39 | 84% |
| # unique control sets | 2 |  |  |
| # of slice register sites lost to control set restrictions | 5 | 18,224 | 1% |
| # of bonded IOBs | 7 | 232 | 3% |
| # of LOCed IOBs | 7 | 7 | 100% |
| # of BUFG/BUFGMUXs | 1 | 16 | 6% |
| # used as BUFGs | 1 |  |  |
| Average Fanout of Non-Clock Nets | 2.50 |  |  |

**Maximum Clock Frequency: 307.031 MHz**

**MOORE BINARY**

================

FSMMooreBinary.v

================

`timescale 1ns / 1ps

`define N\_BIT 29

`define DIVIDE 399999999

`define ST\_BIT\_CNT 3

//mealy implementation

module FSM(

in, //intput line -- switch

clk,

valid, //output line -- led

curr\_st, //led

rst

);

//clock divider

input clk;

reg trig;

reg [`N\_BIT-1:0] in\_cnt;

//input signals

input in;

input rst;

//ouput bit and state

output valid;

output reg[`ST\_BIT\_CNT-1:0] curr\_st;

parameter si = 3'd0;

parameter s0 = 3'd1;

parameter s01 = 3'd2;

parameter s010 = 3'd3;

parameter s011 = 3'd4;

parameter sterm = 3'd5;

//clock divider logic

always @ (posedge rst, posedge clk) begin

if (rst | trig)

begin

in\_cnt <= `N\_BIT'b0;

trig <= 1'b0;

end

else

begin

in\_cnt <= in\_cnt +1'b1;

trig <= (in\_cnt == `N\_BIT'd`DIVIDE);

end

end

//next state computation

always @ (posedge trig, posedge rst)begin

if (rst)

curr\_st <= si;

else begin

case (curr\_st)

si: curr\_st <= in ? si : s0;

s0: curr\_st <= in ? s01 : s0;

s01: curr\_st <= in ? s011 : s010;

s010: curr\_st <= in ? sterm : s0;

s011: curr\_st <= in ? si : sterm;

sterm: curr\_st <= in ? si : si;

endcase

end

end

//output

assign valid = (curr\_st == sterm);

endmodule

**Synthesis Report**

========================================

\* Synthesis Report \*

========================================

Macro Statistics

# Adders/Subtractors : 1

29-bit adder : 1

# Registers : 2

1-bit register : 1

29-bit register : 1

# Multiplexers : 2

1-bit 2-to-1 multiplexer : 1

29-bit 2-to-1 multiplexer : 1

# FSMs : 1

# Registers : 35

Flip-Flops : 35

========================================

\* Design Summary \*

========================================

Primitive and Black Box Usage:

------------------------------

# BELS : 98

# GND : 1

# INV : 1

# LUT2 : 1

# LUT3 : 29

# LUT4 : 3

# LUT5 : 1

# LUT6 : 5

# MUXCY : 28

# XORCY : 29

# FlipFlops/Latches : 35

# FDC : 35

# Clock Buffers : 1

# BUFGP : 1

# IO Buffers : 6

# IBUF : 2

# OBUF : 4

Device utilization summary:

---------------------------

Selected Device : 6slx16csg324-3

Slice Logic Utilization:

Number of Slice Registers: 35 out of 18224 0%

Number of Slice LUTs: 40 out of 9112 0%

Number used as Logic: 40 out of 9112 0%

Slice Logic Distribution:

Number of LUT Flip Flop pairs used: 42

Number with an unused Flip Flop: 7 out of 42 16%

Number with an unused LUT: 2 out of 42 4%

Number of fully used LUT-FF pairs: 33 out of 42 78%

Number of unique control sets: 2

IO Utilization:

Number of IOs: 7

Number of bonded IOBs: 7 out of 232 3%

Specific Feature Utilization:

Number of BUFG/BUFGCTRLs: 1 out of 16 6%

**Device Utilization Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Slice Logic Utilization | Used | Available | Utilization |
| # Slice Registers | 35 | 18,224 | 1% |
| # used as Flip Flops | 35 |  |  |
| # Slice LUTs | 38 | 9,112 | 1% |
| # used as logic | 38 | 9,112 | 1% |
| # using O6 output only | 8 |  |  |
| # using O5 and O6 | 30 |  |  |
| # of occupied Slices | 16 | 2,278 | 1% |
| # of MUXCYs used | 32 | 4,556 | 1% |
| # of LUT Flip Flop pairs used | 39 |  |  |
| # with and unused Flip Flop | 5 | 39 | 12% |
| # with an unused LUT | 1 | 39 | 2% |
| # fully used LUT-FF pairs | 33 | 39 | 84% |
| # unique control sets | 2 |  |  |
| # of slice register sites lost to control set restrictions | 5 | 18,224 | 1% |
| # of bonded IOBs | 7 | 232 | 3% |
| # of LOCed IOBs | 7 | 7 | 100% |
| # of BUFG/BUFGMUXs | 1 | 16 | 6% |
| # used as BUFGs | 1 |  |  |
| Average Fanout of Non-Clock Nets | 2.52 |  |  |

**Maximum Clock Frequency: 325.098 MHz**

**MOORE ONEHOT**

================

FSMMooreOnehot.v

================

`timescale 1ns / 1ps

`define N\_BIT 29

`define DIVIDE 399999999

`define ST\_BIT\_CNT 6

//mealy implementation

module FSM(

in, //intput line -- switch

clk,

valid, //output line -- led

curr\_st, //led

rst

);

//clock divider

input clk;

reg trig;

reg [`N\_BIT-1:0] in\_cnt;

//input signals

input in;

input rst;

//ouput bit and state

output valid;

output reg[`ST\_BIT\_CNT-1:0] curr\_st;

//one hot bit encoding

parameter si = 6'b000001;

parameter s0 = 6'b000010;

parameter s01 = 6'b000100;

parameter s010 = 6'b001000;

parameter s011 = 6'b010000;

parameter sterm = 6'b100000;

//clock divider logic

always @ (posedge rst, posedge clk) begin

if (rst | trig)

begin

in\_cnt <= `N\_BIT'b0;

trig <= 1'b0;

end

else

begin

in\_cnt <= in\_cnt +1'b1;

trig <= (in\_cnt == `N\_BIT'd`DIVIDE);

end

end

//next state computation

always @ (posedge trig, posedge rst)begin

if (rst)

curr\_st <= si;

else begin

case (curr\_st)

si: curr\_st <= in ? si : s0;

s0: curr\_st <= in ? s01 : s0;

s01: curr\_st <= in ? s011 : s010;

s010: curr\_st <= in ? sterm : s0;

s011: curr\_st <= in ? si : sterm;

sterm: curr\_st <= in ? si : si;

endcase

end

end

//output

assign valid = (curr\_st == sterm);

endmodule

**Synthesis Report**

========================================

\* Synthesis Report \*

========================================

Macro Statistics

# Adders/Subtractors : 1

29-bit adder : 1

# Registers : 2

1-bit register : 1

29-bit register : 1

# Multiplexers : 2

1-bit 2-to-1 multiplexer : 1

29-bit 2-to-1 multiplexer : 1

# FSMs : 1

# Registers : 37

Flip-Flops : 37

========================================

\* Design Summary \*

========================================

Primitive and Black Box Usage:

------------------------------

# BELS : 100

# GND : 1

# INV : 1

# LUT2 : 3

# LUT3 : 30

# LUT4 : 2

# LUT5 : 1

# LUT6 : 5

# MUXCY : 28

# XORCY : 29

# FlipFlops/Latches : 37

# FDC : 36

# FDP : 1

# Clock Buffers : 1

# BUFGP : 1

# IO Buffers : 9

# IBUF : 2

# OBUF : 7

Device utilization summary:

---------------------------

Selected Device : 6slx16csg324-3

Slice Logic Utilization:

Number of Slice Registers: 37 out of 18224 0%

Number of Slice LUTs: 42 out of 9112 0%

Number used as Logic: 42 out of 9112 0%

Slice Logic Distribution:

Number of LUT Flip Flop pairs used: 43

Number with an unused Flip Flop: 6 out of 43 13%

Number with an unused LUT: 1 out of 43 2%

Number of fully used LUT-FF pairs: 36 out of 43 83%

Number of unique control sets: 2

IO Utilization:

Number of IOs: 10

Number of bonded IOBs: 10 out of 232 4%

Specific Feature Utilization:

Number of BUFG/BUFGCTRLs: 1 out of 16 6%

**Device Utilization Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Slice Logic Utilization | Used | Available | Utilization |
| # Slice Registers | 37 | 18,224 | 1% |
| # used as Flip Flops | 37 |  |  |
| # Slice LUTs | 40 | 9,112 | 1% |
| # used as logic | 40 | 9,112 | 1% |
| # using O6 output only | 10 |  |  |
| # using O5 and O6 | 30 |  |  |
| # of occupied Slices | 14 | 2,278 | 1% |
| # of MUXCYs used | 32 | 4,556 | 1% |
| # of LUT Flip Flop pairs used | 41 |  |  |
| # with and unused Flip Flop | 6 | 39 | 12% |
| # with an unused LUT | 1 | 39 | 2% |
| # fully used LUT-FF pairs | 34 | 39 | 84% |
| # unique control sets | 2 |  |  |
| # of slice register sites lost to control set restrictions | 3 | 18,224 | 1% |
| # of bonded IOBs | 10 | 232 | 3% |
| # of LOCed IOBs | 7 | 7 | 100% |
| # of BUFG/BUFGMUXs | 1 | 16 | 6% |
| # used as BUFGs | 1 |  |  |
| Average Fanout of Non-Clock Nets | 2.64 |  |  |

**Maximum Clock Frequency: 339.789 MHz**

**MOORE GREY**

==============

FSMMooreGrey.v

==============

`timescale 1ns / 1ps

`define N\_BIT 29

`define DIVIDE 399999999

`define ST\_BIT\_CNT 3

//mealy implementation

module FSM(

in, //input line -- switch

clk,

valid, //output line -- led

curr\_st, //led

rst

);

//clock divider

input clk;

reg trig;

reg [`N\_BIT-1:0] in\_cnt;

//input signals

input in;

input rst;

//ouput bit and state

output valid;

output reg[`ST\_BIT\_CNT-1:0] curr\_st;

parameter si = 3'b000;

parameter s0 = 3'b001;

parameter s01 = 3'b011;

parameter s010 = 3'b010;

parameter s011 = 3'b110;

parameter sterm = 3'b111;

//clock divider logic

always @ (posedge rst, posedge clk) begin

if (rst | trig)

begin

in\_cnt <= `N\_BIT'b0;

trig <= 1'b0;

end

else

begin

in\_cnt <= in\_cnt +1'b1;

trig <= (in\_cnt == `N\_BIT'd`DIVIDE);

end

end

//next state computation

always @ (posedge trig, posedge rst)begin

if (rst)

curr\_st <= si;

else begin

case (curr\_st)

si: curr\_st <= in ? si : s0;

s0: curr\_st <= in ? s01 : s0;

s01: curr\_st <= in ? s011 : s010;

s010: curr\_st <= in ? sterm : s0;

s011: curr\_st <= in ? si : sterm;

sterm: curr\_st <= in ? si : si;

endcase

end

end

//output

assign valid = (curr\_st == sterm);

endmodule

**Synthesis Report**

========================================

\* Synthesis Report \*

========================================

Macro Statistics

# Adders/Subtractors : 1

29-bit adder : 1

# Registers : 2

1-bit register : 1

29-bit register : 1

# Multiplexers : 2

1-bit 2-to-1 multiplexer : 1

29-bit 2-to-1 multiplexer : 1

# FSMs : 1

# Registers : 35

Flip-Flops : 35

========================================

\* Design Summary \*

========================================

Primitive and Black Box Usage:

------------------------------

# BELS : 98

# GND : 1

# INV : 1

# LUT2 : 1

# LUT3 : 29

# LUT4 : 3

# LUT5 : 1

# LUT6 : 5

# MUXCY : 28

# XORCY : 29

# FlipFlops/Latches : 35

# FDC : 35

# Clock Buffers : 1

# BUFGP : 1

# IO Buffers : 6

# IBUF : 2

# OBUF : 4

Device utilization summary:

---------------------------

Selected Device : 6slx16csg324-3

Slice Logic Utilization:

Number of Slice Registers: 35 out of 18224 0%

Number of Slice LUTs: 40 out of 9112 0%

Number used as Logic: 40 out of 9112 0%

Slice Logic Distribution:

Number of LUT Flip Flop pairs used: 42

Number with an unused Flip Flop: 7 out of 42 16%

Number with an unused LUT: 2 out of 42 4%

Number of fully used LUT-FF pairs: 33 out of 42 78%

Number of unique control sets: 2

IO Utilization:

Number of IOs: 7

Number of bonded IOBs: 7 out of 232 3%

Specific Feature Utilization:

Number of BUFG/BUFGCTRLs: 1 out of 16 6%

**Device Utilization Summary**

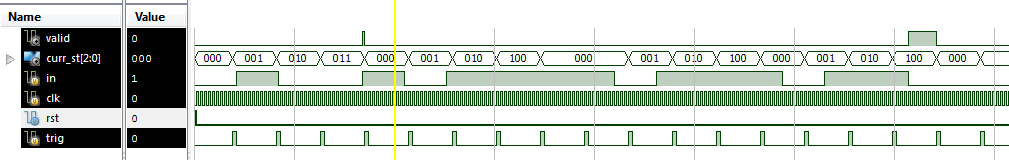
|  |  |  |  |
| --- | --- | --- | --- |
| Slice Logic Utilization | Used | Available | Utilization |
| # Slice Registers | 35 | 18,224 | 1% |
| # used as Flip Flops | 35 |  |  |
| # Slice LUTs | 38 | 9,112 | 1% |
| # used as logic | 38 | 9,112 | 1% |
| # using O6 output only | 8 |  |  |
| # using O5 and O6 | 30 |  |  |
| # of occupied Slices | 16 | 2,278 | 1% |
| # of MUXCYs used | 32 | 4,556 | 1% |
| # of LUT Flip Flop pairs used | 39 |  |  |
| # with an unused Flip Flop | 5 | 39 | 12% |
| # with an unused LUT | 1 | 39 | 2% |
| # fully used LUT-FF pairs | 33 | 39 | 84% |
| # unique control sets | 2 |  |  |
| # of slice register sites lost to control set restrictions | 5 | 18,224 | 1% |
| # of bonded IOBs | 7 | 232 | 3% |
| # of LOCed IOBs | 7 | 7 | 100% |
| # of BUFG/BUFGMUXs | 1 | 16 | 6% |
| # used as BUFGs | 1 |  |  |
| Average Fanout of Non-Clock Nets | 2.52 |  |  |

**Maximum Clock Frequency: 325.098 MHz**

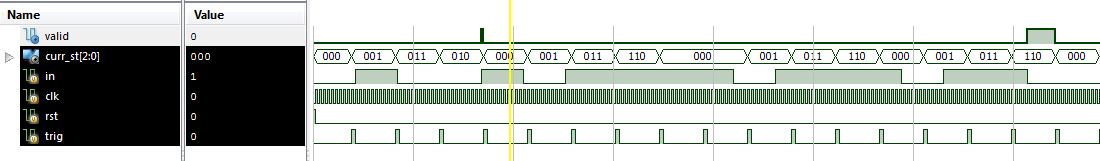
**III. SIMULATION DOCUMENTATION (testbench above)**

**WAVEFORMS**

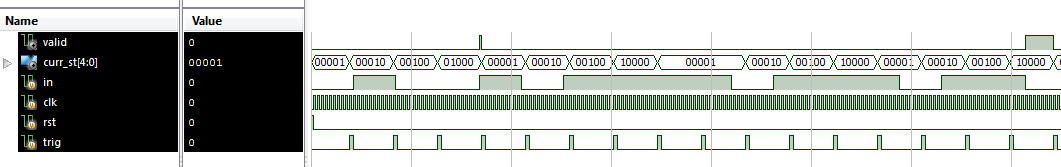
\*\*For simplicity’s sake, we commented out the clock divider for each of the \*.v files

when trying to simulate their respective waveforms. I simulated both 0110 and 0101 **MEALY BINARY** 

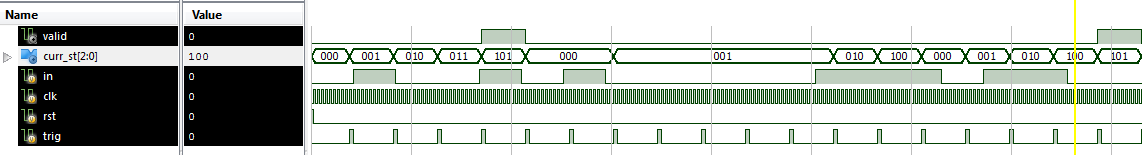
**MEALY GREY**



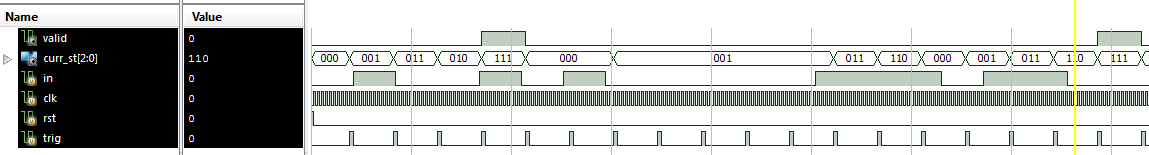
**MEALY ONEHOT**



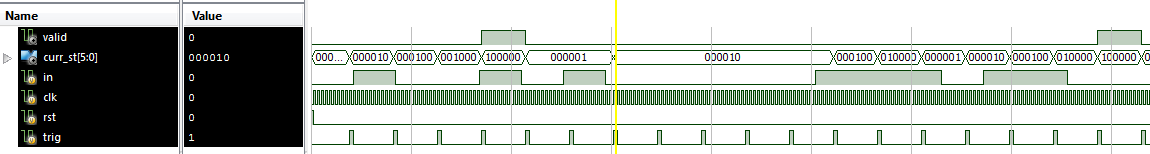
**MOORE BINARY**



**MOORE GREY**



**MOORE ONEHOT**



**Test Cases:**

For all our test cases, we put in the inputs that would test both successes -- 0110 and 0101. We would also try other inputs to see if each state went to the correct next state given the input. We did test cases for both Mealy and Moore machines, although all that was different was that for the Moore machine we had to add a couple more inputs to account for the fact that we had a terminal state (output depended only on state).

**IV. CONCLUSION**

In this lab, we got to play with a finite state machine to implement a pattern recognizer. We had to first design by hand the state transition tables for both Mealy and Moore machines, translate that to Verilog code, and include implementations for different state encodings (1-hot, binary, and grey). A mistake that we encountered was mixing up Mealy and Moore machines. Additionally, we made the mistake of having the trigger as a wire instead of a register that activated on the posedge of the clock. This fixed an issue that we were having in which it would check the input too quickly (i.e. jump from s0 to s011 if the input was ‘1’ in one clock cycle)**.**

**STOPWATCH**

**I. INTRODUCTION**

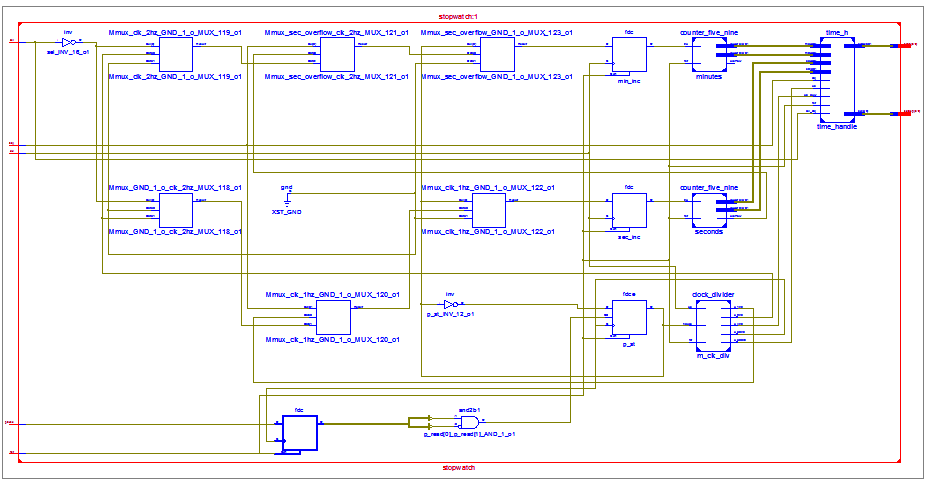
**Summary:**

In the second part of this lab we designed and built a stopwatch using Verilog. The functionality includes: being able to count minutes:seconds from 00:00 to 59:59 before resetting, a *pause* button that allows us to toggle pause the clock, a *reset* button that resets it back to 00:00, and an *adj* switch that allows us to adjust either minutes or seconds by speeding up the clock cycles of whichever one’s selected using the *sel* switch.

**Schematic and Module Description of Stopwatch (modules explanation)**

**Flow Diagram**

**RTL Schematic**



**Module Description**

1. Clock Divider

This module is responsible for generating slower clock signals used for the other parts of the stopwatch. This is done by keeping a count and triggering when a certain count is met. A 500 hz, 50 hz, 4 hz, 2 hz, and 1 hz cycle is generated, each used by the other modules.

2. Five Nine Counter

This module is responsible for keeping the number states of the seconds and minutes. It involves a modulo 10 counter and a modulo 6 counter. The input to the module is an increment signal(can be a 2 hz, 1 hz) which will increment the count at a pos edge. The module will output the count of each place and also an overflow signal.

3. Seven Seg Handler

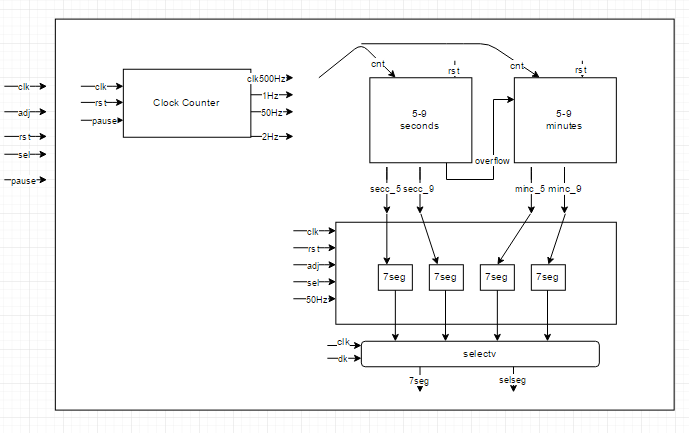
This combinational module translate a binary number into a seven bit seven segment encoding of the number.

4. Time Handler

Time handler module takes in 4 numbers, the seconds and minute digits from 2 five nine counter and uses the seven seg handlers to translate them. Based on a 50hz clk and some other control signals, it alternates the number on the physical seven segment to set and the control bits to update it.

5. Stopwatch

Is the high level module that wraps all the above. See the diagram below for the flow.



**II. DESIGN DESCRIPTION**

**Code for Stopwatch**

================

stopwatch.v

================

`timescale 1ns / 1ps

module stopwatch(clk, adj,rst,pause,sel, selSeg,seg7);

input clk;

input adj;

input rst;

input pause;

input sel;

output [3:0]selSeg;

output [6:0]seg7;

wire clk\_500hz;

wire clk\_1hz;

wire clk\_2hz;

wire clk\_4hz;

wire clk\_50hz;

//pause state

reg p\_st;

//clock divider signals

clock\_divider m\_clk\_div(.clk(clk),.rst(rst),.o\_500hz(clk\_500hz),.o\_4hz(clk\_4hz),.o\_1hz(clk\_1hz),.o\_2hz(clk\_2hz),.o\_50hz(clk\_50hz),.pause (p\_st));

reg [1:0] p\_read;

always @(posedge clk\_50hz, posedge rst) begin

if (rst) begin

p\_st <= 1'b0;

p\_read <= 2'b0;

end

else begin

p\_read[1] <= p\_read[0];

p\_read[0] <= pause;

if (p\_read[0] & ~p\_read[1])

p\_st <= ~p\_st;

end;

end

////5-9 counters/////

reg sec\_inc;

wire sec\_trig;

wire [3:0] sec\_count\_5;

wire [3:0] sec\_count\_9;

wire sec\_overflow;

// Instantiate the Unit Under Test (UUT)

counter\_five\_nine seconds (

.inc(sec\_inc),

.rst(rst),

.count\_5(sec\_count\_5),

.count\_9(sec\_count\_9),

.overflow(sec\_overflow)

);

always @ (posedge clk, posedge rst) begin

if (rst) begin

sec\_inc <= 1'b0;

min\_inc <= 1'b0;

end

else begin

if (p\_st == 1) begin

sec\_inc <= 1'b0;

min\_inc <= 1'b0;

end

else if (adj) begin

if (~sel) begin

sec\_inc <= clk\_2hz;

min\_inc <= 1'b0;

end

else begin

min\_inc <= clk\_2hz;

sec\_inc <= 1'b0;

end

end

else begin

sec\_inc <= clk\_1hz;

min\_inc <= sec\_overflow;

end

end

end

reg min\_inc;

wire min\_trig;

wire min\_overflow;

wire [3:0] min\_count\_5;

wire [3:0] min\_count\_9;

counter\_five\_nine minutes (

.inc(min\_inc),

.rst(rst),

.count\_5(min\_count\_5),

.count\_9(min\_count\_9),

.overflow(min\_overflow)

);

///seven segments////

time\_h time\_handle(

.clk(clk\_500hz),

.rst(rst),

.s5(sec\_count\_5),

.s9(sec\_count\_9),

.m5(min\_count\_5),

.m9(min\_count\_9),

.sel(selSeg),

.seg7(seg7),

.sel\_adj(sel),

.adj(adj),

.clk\_slow(clk\_4hz));

endmodule

================

clock\_divider.v

================

`timescale 1ns / 1ps

`define CNT\_500HZ 100000

`define CNT\_4HZ 12500000

`define CNT\_50HZ 1000000

`define CNT\_1HZ 50000000

`define CNT\_2HZ 25000000

`define NBITS\_500HZ 17

`define NBITS\_50HZ 20

`define NBITS\_4HZ 24

`define NBITS\_1HZ 26

`define NBITS\_2HZ 25

module clock\_divider(

clk,

rst,

pause,

o\_500hz,

o\_1hz,

o\_2hz,

o\_4hz,

o\_50hz

);

//input outputs

input clk;

input rst;

input pause;

output reg o\_500hz;

output reg o\_1hz;

output reg o\_2hz;

output reg o\_4hz;

output reg o\_50hz;

//register counters

reg[`NBITS\_500HZ-1:0] cnt\_500hz;

reg[`NBITS\_1HZ-1:0] cnt\_1hz;

reg[`NBITS\_2HZ-1:0] cnt\_2hz;

reg[`NBITS\_4HZ-1:0] cnt\_4hz;

reg[`NBITS\_50HZ-1:0] cnt\_50hz;

//counter trig;

reg trig\_500hz;

reg trig\_1hz;

reg trig\_2hz;

reg trig\_4hz;

reg trig\_50hz;

always @(posedge clk, posedge rst) begin

if (rst) begin

trig\_500hz <= 1'b0;

trig\_1hz <= 1'b0;

trig\_2hz <= 1'b0;

trig\_4hz <= 1'b0;

trig\_50hz <= 1'b0;

end

else begin

trig\_500hz <= (cnt\_500hz == `NBITS\_500HZ'd`CNT\_500HZ);

trig\_1hz <= (cnt\_1hz == `NBITS\_1HZ'd`CNT\_1HZ);

trig\_2hz <= (cnt\_2hz == `NBITS\_2HZ'd`CNT\_2HZ);

trig\_4hz <= (cnt\_4hz == `NBITS\_4HZ'd`CNT\_4HZ);

trig\_50hz <= (cnt\_50hz == `NBITS\_50HZ'd`CNT\_50HZ);

end

end

//500 hz loop DOES NOT STOP FOR PAUSE BECAUSE IT'S USED FOR DEBOUNCING

always @(posedge clk, posedge rst) begin

if (rst | trig\_500hz)

cnt\_500hz <= `NBITS\_500HZ'd1;

else

cnt\_500hz <= cnt\_500hz + 1'b1;

end

//50 hz loop DOES NOT STOP FOR PAUSE BECAUSE IT'S USED FOR DEBOUNCING

always @(posedge clk, posedge rst) begin

if (rst | trig\_50hz)

cnt\_50hz <= `NBITS\_50HZ'd1;

else

cnt\_50hz <= cnt\_50hz + 1'b1;

end

//50 hz loop DOES NOT STOP FOR PAUSE BECAUSE IT'S USED FOR DEBOUNCING

always @(posedge clk, posedge rst) begin

if (rst | trig\_4hz)

cnt\_4hz <= `NBITS\_4HZ'd1;

else

cnt\_4hz <= cnt\_4hz + 1'b1;

end

//2 hz loop

always @(posedge clk, posedge rst) begin

if (rst | trig\_2hz)

cnt\_2hz <= `NBITS\_2HZ'd1;

else if (pause)

cnt\_2hz <= cnt\_2hz;

else

cnt\_2hz <= cnt\_2hz + 1'b1;

end

//1 hz loop

always @(posedge clk, posedge rst) begin

if (rst | trig\_1hz)

cnt\_1hz <= `NBITS\_1HZ'd1;

else if (pause)

cnt\_1hz <= cnt\_1hz;

else

cnt\_1hz <= cnt\_1hz + 1'b1;

end

always @(posedge rst, posedge trig\_1hz) begin

if (rst)

o\_1hz <= 1'b0;

else

o\_1hz <= ~o\_1hz;

end

always @(posedge rst, posedge trig\_500hz) begin

if (rst)

o\_500hz <= 1'b0;

else

o\_500hz <= ~o\_500hz;

end

always @(posedge rst, posedge trig\_50hz) begin

if (rst)

o\_50hz <= 1'b0;

else

o\_50hz <= ~o\_50hz;

end

always @(posedge rst, posedge trig\_2hz) begin

if (rst)

o\_2hz <= 1'b0;

else

o\_2hz <= ~o\_2hz;

end

always @(posedge rst, posedge trig\_4hz) begin

if (rst)

o\_4hz <= 1'b0;

else

o\_4hz <= ~o\_4hz;

end

endmodule

================

counter\_five\_nine.v

================

`timescale 1ns / 1ps

module counter\_five\_nine(

inc,

rst,

count\_5,

count\_9,

overflow

);

input inc;

input rst;

wire overflow\_p;

output reg overflow;

output reg[3:0] count\_5;

output reg[3:0] count\_9;

wire trig\_9 = (count\_9 == 4'd9);

assign overflow\_p = (count\_9 == 4'd9) & (count\_5 == 5'd5);

//overflow handle

always @(posedge inc, posedge rst)begin

if (rst)

overflow <= 1'b0;

else if (overflow\_p)

overflow <= 1'b1;

else

overflow <= 1'b0;

end

//9 handle

always @(posedge inc, posedge rst) begin

if (rst)

count\_9 <= 4'd0;

else

count\_9 <= (count\_9 == 4'd9) ? 4'd0 : count\_9 + 1'd1;

end

//5 trig

always @(posedge inc, posedge rst) begin

if (rst)

count\_5 <= 4'd0;

else if (trig\_9)

count\_5 <= (count\_5 == 4'd5) ? 4'd0 : count\_5 + 1'd1;

else

count\_5 <= count\_5;

end

endmodule

================

time\_h.v

================

`timescale 1ns / 1ps

module time\_h(

clk,

rst,

s5,

s9,

m5,

m9,

sel,

seg7,

adj,

sel\_adj,

clk\_slow

);

input clk;

input clk\_slow;

input rst;

input adj;

input sel\_adj;

input [3:0] s5;

input [3:0] m5;

input [3:0] s9;

input [3:0] m9;

output reg [3:0] sel;

output reg [6:0] seg7;

wire [6:0] s5\_enc\_reg;

wire [6:0] s9\_enc\_reg;

wire [6:0] m5\_enc\_reg;

wire [6:0] m9\_enc\_reg;

//module in translations

seven\_seg s5\_trans(.in\_num(s5), .out\_num(s5\_enc\_reg));

seven\_seg s9\_trans(.in\_num(s9), .out\_num(s9\_enc\_reg));

seven\_seg m5\_trans(.in\_num(m5), .out\_num(m5\_enc\_reg));

seven\_seg m9\_trans(.in\_num(m9), .out\_num(m9\_enc\_reg));

reg blink;

always @(posedge clk\_slow, posedge rst) begin

if (rst)

blink <= 1'b0;

else

blink <= ~blink;

end

//switching numbers

always @ (posedge clk, posedge rst) begin

//clear out all numbers

if (rst) begin

sel <= 4'b0000;

seg7 <= 7'b0000001;//zero out all

end

else if (sel == 4'b1110)begin //s9

sel <= 4'b1101;

seg7 <= (adj & ~sel\_adj) ?

blink ? s5\_enc\_reg :

7'b1111111 :

s5\_enc\_reg;

end

else if (sel == 4'b1101)begin//s5

sel <= 4'b1011;

seg7 <= (adj & sel\_adj) ?

blink ? m9\_enc\_reg :

7'b1111111 :

m9\_enc\_reg;

end

else if (sel == 4'b1011)begin//m9

sel <= 4'b0111;

seg7 <= (adj & sel\_adj) ?

blink ? m5\_enc\_reg :

7'b1111111 :

m5\_enc\_reg;

end

else if (sel == 4'b0111)begin//m5

sel <= 4'b1110;

seg7 <= (adj & ~sel\_adj) ?

blink ? s9\_enc\_reg :

7'b1111111 :

s9\_enc\_reg;

end

else begin

sel <= 4'b1110;

end

end

endmodule

================

seven\_seg.v

================

`timescale 1ns / 1ps

module seven\_seg(

in\_num,

out\_num

);

input [3:0] in\_num;

output [6:0] out\_num;

/\*

0 - middle

1 - top left

2 - bottom left

3 - bottom

4 - bottom right

5 - top right

6 - top

\*/

assign out\_num =

(in\_num == 4'd9) ? 7'b0000100 :

(in\_num == 4'd8) ? 7'b0000000 :

(in\_num == 4'd7) ? 7'b0001111 :

(in\_num == 4'd6) ? 7'b0100000 :

(in\_num == 4'd5) ? 7'b0100100 :

(in\_num == 4'd4) ? 7'b1001100 :

(in\_num == 4'd3) ? 7'b0000110 :

(in\_num == 4'd2) ? 7'b0010010 :

(in\_num == 4'd1) ? 7'b1001111 :

(in\_num == 4'd0) ? 7'b0000001 :

7'b1111111;//DOESN'T OCCUR

endmodule

================

stopwatch.ucf

================

## Clock signal

NET "clk" LOC = "V10" | IOSTANDARD = "LVCMOS33"; #Bank = 2, pin name = IO\_L30N\_GCLK0\_USERCCLK, Sch name = GCLK

Net "clk" TNM\_NET = sys\_clk\_pin;

TIMESPEC TS\_sys\_clk\_pin = PERIOD sys\_clk\_pin 100000 kHz;

## 7 segment display

NET "seg7<6>" LOC = "T17" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L51P\_M1DQ12, Sch name = CA

NET "seg7<5>" LOC = "T18" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L51N\_M1DQ13, Sch name = CB

NET "seg7<4>" LOC = "U17" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L52P\_M1DQ14, Sch name = CC

NET "seg7<3>" LOC = "U18" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L52N\_M1DQ15, Sch name = CD

NET "seg7<2>" LOC = "M14" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L53P, Sch name = CE

NET "seg7<1>" LOC = "N14" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L53N\_VREF, Sch name = CF

NET "seg7<0>" LOC = "L14" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L61P, Sch name = CG

#NET "seg7<7>" LOC = "M13" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L61N, Sch name = DP

NET "selSeg<0>" LOC = "N16" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L50N\_M1UDQSN, Sch name = AN0

NET "selSeg<1>" LOC = "N15" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L50P\_M1UDQS, Sch name = AN1

NET "selSeg<2>" LOC = "P18" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L49N\_M1DQ11, Sch name = AN2

NET "selSeg<3>" LOC = "P17" | IOSTANDARD = "LVCMOS33"; #Bank = 1, Pin name = IO\_L49P\_M1DQ10, Sch name = AN3

## Leds

#NET "adj\_led" LOC = "U16" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L2P\_CMPCLK, Sch name = LD0

## Switches

NET "adj" LOC = "T10" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L29N\_GCLK2, Sch name = SW0

NET "sel" LOC = "T9" | IOSTANDARD = "LVCMOS33"; #Bank = 2, Pin name = IO\_L32P\_GCLK29, Sch name = SW1

## Buttons

NET "rst" LOC = "B8" | IOSTANDARD = "LVCMOS33"; #Bank = 0, Pin name = IO\_L33P, Sch name = BTNS

NET "pause" LOC = "D9" | IOSTANDARD = "LVCMOS33"; #Bank = 0, Pin name = IO\_L34P\_GCLK19, Sch name = BTNR

================

stopwatch\_TB.v

================

`timescale 1ns / 1ps

module stopwatch\_TB;

// Inputs

reg clk;

reg adj;

reg rst;

reg pause;

reg sel;

// Outputs

wire [3:0] selSeg;

wire [6:0] seg7;

// Instantiate the Unit Under Test (UUT)

stopwatch uut (

.clk(clk),

.adj(adj),

.rst(rst),

.pause(pause),

.sel(sel),

.selSeg(selSeg),

.seg7(seg7)

);

initial begin

// Initialize Inputs

clk = 0;

adj = 0;

rst = 0;

pause = 0;

sel = 0;

#1 rst = 1;

#0.5 rst = 0;

//testing out pause

#500 pause = 1;

#0.005 pause = 0;

#200 pause = 1;

#0.005 pause = 0;

//testing out adj

#50 adj = 1;

//pause while in adj mode

#20 pause = 1;

#0.005 pause = 0;

#50 pause = 1;

#0.005 pause = 0;

#20 sel = 1;

#50 adj = 0;

// Wait 100 ns for global reset to finish

#10000 $finish;

// Add stimulus here

end

always @ (\*)

#0.001 clk <= ~clk;

endmodule

**Synthesis Report**

========================================

\* Synthesis Report \*

========================================

Macro Statistics

# RAMs : 4

16x7-bit single-port Read Only RAM : 4

# Adders/Subtractors : 9

17-bit adder : 1

20-bit adder : 1

24-bit adder : 1

25-bit adder : 1

26-bit adder : 1

4-bit adder : 4

# Registers : 27

1-bit register : 16

17-bit register : 1

2-bit register : 1

20-bit register : 1

24-bit register : 1

25-bit register : 1

26-bit register : 1

4-bit register : 4

7-bit register : 1

# Multiplexers : 33

1-bit 2-to-1 multiplexer : 6

17-bit 2-to-1 multiplexer : 1

20-bit 2-to-1 multiplexer : 1

24-bit 2-to-1 multiplexer : 1

25-bit 2-to-1 multiplexer : 1

26-bit 2-to-1 multiplexer : 1

4-bit 2-to-1 multiplexer : 4

7-bit 2-to-1 multiplexer : 18

# FSMs : 1

========================================

\* Design Summary \*

========================================

Primitive and Black Box Usage:

------------------------------

# BELS : 469

# GND : 1

# INV : 15

# LUT2 : 4

# LUT3 : 141

# LUT4 : 29

# LUT5 : 14

# LUT6 : 43

# MUXCY : 107

# MUXF7 : 2

# VCC : 1

# XORCY : 112

# FlipFlops/Latches : 167

# FDC : 104

# FDCE : 57

# FDP : 4

# FDPE : 2

# Clock Buffers : 2

# BUFG : 1

# BUFGP : 1

# IO Buffers : 15

# IBUF : 4

# OBUF : 11

Device utilization summary:

---------------------------

Selected Device : 6slx16csg324-3

Slice Logic Utilization:

Number of Slice Registers: 167 out of 18224 0%

Number of Slice LUTs: 246 out of 9112 2%

Number used as Logic: 246 out of 9112 2%

Slice Logic Distribution:

Number of LUT Flip Flop pairs used: 262

Number with an unused Flip Flop: 95 out of 262 36%

Number with an unused LUT: 16 out of 262 6%

Number of fully used LUT-FF pairs: 151 out of 262 57%

Number of unique control sets: 15

IO Utilization:

Number of IOs: 16

Number of bonded IOBs: 16 out of 232 6%

Specific Feature Utilization:

Number of BUFG/BUFGCTRLs: 2 out of 16 12%

**Device Utilization Summary**

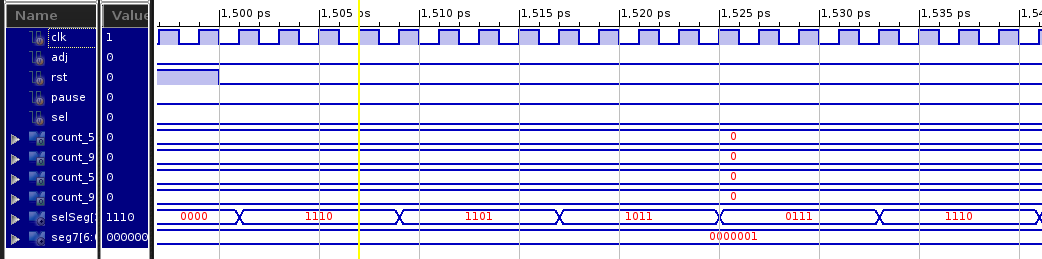
|  |  |  |  |
| --- | --- | --- | --- |
| Slice Logic Utilization | Used | Available | Utilization |
| # Slice Registers | 167 | 18,224 | 1% |
| # used as Flip Flops | 167 |  |  |
| # Slice LUTs | 220 | 9,112 | 2% |
| # used as logic | 219 | 9,112 | 2% |
| # using O6 output only | 86 |  |  |
| # using O5 and O6 | 133 |  |  |
| # used exclusively as route-thrus | 1 |  |  |
| # with same-slice register load | 1 |  |  |
| # of occupied Slices | 81 | 2,278 | 3% |
| # of MUXCYs used | 120 | 4,556 | 2% |
| # of LUT Flip Flop pairs used | 228 |  |  |
| # with and unused Flip Flop | 66 | 228 | 28% |
| # with an unused LUT | 8 | 228 | 3% |
| # fully used LUT-FF pairs | 154 | 228 | 67% |
| # unique control sets | 15 |  |  |
| # of slice register sites lost to control set restrictions | 81 | 18,224 | 1% |
| # of bonded IOBs | 16 | 232 | 6% |
| # of LOCed IOBs | 16 | 16 | 100% |
| # of BUFG/BUFGMUXs | 2 | 16 | 12% |
| # used as BUFGs | 2 |  |  |
| Average Fanout of Non-Clock Nets | 3.06 |  |  |

**Maximum Clock Frequency: 304.971 MHz**

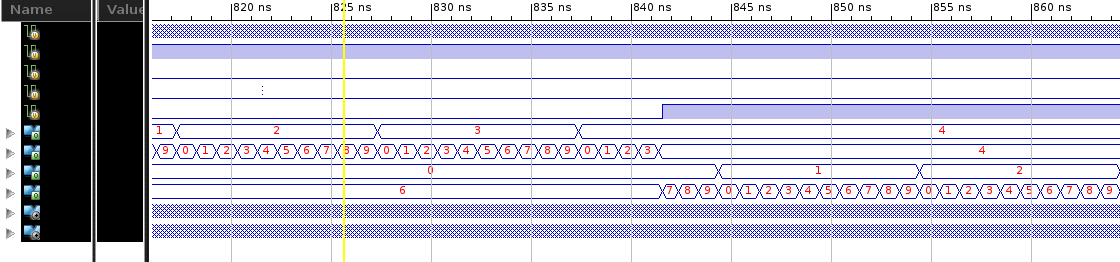
**III. SIMULATION DOCUMENTATION (testbench above)**

**Waveform**

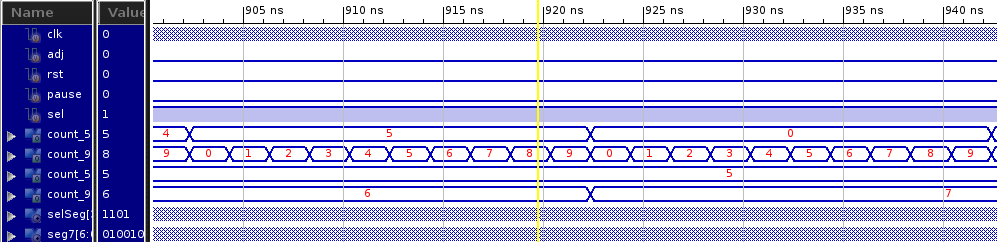
**Reset**



**Adjust**



**Normal Counting**



**Test Cases:**

The test case tests all the possible control and outputs. It tests both selects, making sure the output blinks and counts faster. It tests the reset randomly, making sure every module resets. It also tests pause, making sure that the clock timers don’t count up during the pause state. The test also just watches the time goes up and overflows.

**IV. CONCLUSION**

In this lab, we got to design a relatively large project with many components. We had to work with clocks, synchronization in different modules, some I/O from button, switches, seven segment timers, and needed some special techniques like debouncing. Some mistakes we made were just careless specs reading and also trying to do posedge of a signal without doing debouncing.We fixed this by doing multiple samples over a slower clock frequency. Other mistakes we made were mostly the design was overly complicated at times. In the future, for a large project like this, we’ll flush out details instead of just creating a flow diagram.