**Lab 5 Report Hardik J Patel**

**EEC 180B 999121498**

**Sequential Design**

**Objective**

The purpose of this lab was to implement different types of controllers on a FPGA using Verilog. The project included two main parts, a code decoder and a morse code encoder, which were both implemented as finite state machines.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Design and Test Procedure**

Part 1: In this part of the lab a finite state machine was created to decode a sequence of numbers. Whenever the sequence was 0000 or 1111 for four consecutive clock cycles, the output was set to 1. To implement this, a state diagram was created in the prelab, and was implement in verilog using one hot and binary encoding. The design was verified using functional simulation and then downloaded to the DE2 board to verify the correct operation of the circuit. While comparing between the two encodings, one hot was faster and contained the shorter critical path while having one flip flop for each state, and hence increasing the power consumption. In contrast, the binary encoding had a longer critical path, but used less number of flip flops and consumed less power.

*The code for this part is attached at the end.*

Part 2: For this part in the lab, we implement the same code decoder from part 1 using shift registers instead of finite state machine. This becomes a fully combinational circuit and works faster than the state machine.

*The code for this part is attached at the end.*

Pat 3: This part required us to implement a Morse Code Encoder in Verilog to show the Morse code using a LED for letters A-H. The morse code was given in the Lab Manual as dots (0.5 s) and dashes (1.5 s) for each of the letters. This implementation was also done using a state diagram. Each letter was broken into a length and code which were then used to produce the Morse code. Since the DE2 board provides a 50 MHz clock and we need a conversion into seconds a counter was implemented for that purpose. Hence the code includes, a code breaker, a state diagram, a counter and an output assignment.

*The code for this part is attached at the end.*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Conclusion**

This lab taught us the implementation of some real life circuits using behavioral verilog.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Part 1: Finite State Machines in Verilog with Different State Assignments (using one hot encoding)**

module moore(w, z, rst, clk);

input w, clk, rst;

output z;

reg z;

reg [8:0] state;

reg [8:0] nextstate;

localparam state0 = 9'b000000001;

localparam state1 = 9'b000000010;

localparam state2 = 9'b000000100;

localparam state3 = 9'b000001000;

localparam state4 = 9'b000010000;

localparam state5 = 9'b000100000;

localparam state6 = 9'b001000000;

localparam state7 = 9'b010000000;

localparam state8 = 9'b100000000;

always @(posedge clk or posedge rst) begin

if (rst)

state <= state0;

else

state <= nextstate;

end

always @ (\*) begin

case (state)

state0: begin

z = 0;

if(w)

nextstate = state1;

else

nextstate = state5;

end

state1: begin

z = 0;

if(w)

nextstate = state2;

else

nextstate = state5;

end

state2: begin

z = 0;

if(w)

nextstate = state3;

else

nextstate = state5;

end

state3: begin

z = 0;

if(w)

nextstate = state4;

else

nextstate = state5;

end

state4: begin

z = 1;

if(w)

nextstate = state4;

else

nextstate = state5;

end

state5: begin

z = 0;

if(~w)

nextstate = state6;

else

nextstate = state1;

end

state6: begin

z = 0;

if(~w)

nextstate = state7;

else

nextstate = state1;

end

state7: begin

z = 0;

if(~w)

nextstate = state8;

else

nextstate = state1;

end

state8: begin

z = 1;

if(~w)

nextstate = state8;

else

nextstate = state1;

end

default: nextstate = state0;

endcase

end

endmodule

**Part 1: (using binary encoding)**

module moore(w, z, rst, clk);

input w, clk, rst;

output z;

reg z;

reg [8:0] state;

reg [8:0] nextstate;

localparam state0 = 4’b0000;

localparam state1 = 4’b0001;

localparam state2 = 4’b0010;

localparam state3 = 4’b0011;

localparam state4 = 4’b0100;

localparam state5 = 4’b0101;

localparam state6 = 4’b0110;

localparam state7 = 4’b0111;

localparam state8 = 4’b1000;

always @(posedge clk or posedge rst) begin

if (rst)

state <= state0;

else

state <= nextstate;

end

always @ (\*) begin

case (state)

state0: begin

z = 0;

if(w)

nextstate = state1;

else

nextstate = state5;

end

state1: begin

z = 0;

if(w)

nextstate = state2;

else

nextstate = state5;

end

state2: begin

z = 0;

if(w)

nextstate = state3;

else

nextstate = state5;

end

state3: begin

z = 0;

if(w)

nextstate = state4;

else

nextstate = state5;

end

state4: begin

z = 1;

if(w)

nextstate = state4;

else

nextstate = state5;

end

state5: begin

z = 0;

if(~w)

nextstate = state6;

else

nextstate = state1;

end

state6: begin

z = 0;

if(~w)

nextstate = state7;

else

nextstate = state1;

end

state7: begin

z = 0;

if(~w)

nextstate = state8;

else

nextstate = state1;

end

state8: begin

z = 1;

if(~w)

nextstate = state8;

else

nextstate = state1;

end

default: nextstate = state0;

endcase

end

endmodule

**Part 2: Implementing the Sequence Detector with Shift Register**

module shift(enable, reset, clock, w, z);

input enable, reset, clock, w;

output z;

reg z;

reg [3:0] myreg;

initial myreg = 4'b0000;

always @ (posedge clock ) begin

if(reset)

myreg = 4'b0000;

else begin

if(enable) begin

myreg[3] <= myreg[2];

myreg[3] <= myreg[1];

myreg[1] <= myreg[0];

myreg[0] <= w;

end

else

myreg = myreg + 4'b0000;

end

end

always @ \* begin

if(myreg == 4'b1111 | myreg == 4'b0000)

z=1;

else

z=0;

end

endmodule

**Part 3: Designing and Implementation of a Morse Code Encoder**

module Lab53(SW, KEY, CLOCK\_50, LEDR);

input [2:0] SW;

input [1:0] KEY; // key 0 = reset and key 1 = start

output [0:0] LEDR;

input CLOCK\_50;

reg [25:0] count;

reg [3:0] Code;

reg [2:0] length;

reg [3:0] Q;

reg z;

reg [2:0] counter;

reg[2:0] state, nextstate;

parameter AA = 3'b000, BB = 3'b001, CC = 3'b010, DD = 3'b011, EE = 3'b100, FF = 3'b101, GG = 3'b110, HH = 3'b111;

parameter A = 3'b000, B = 3'b001, C = 3'b010, D = 3'b011, E = 3'b100;

assign LEDR = z;

always @(SW)

begin: letter\_selection

case(SW[2:0])

AA: begin //A

length = 2'b10;

Code = 4'b0100;

end

BB: begin //B

length = 2'b11;

Code = 4'b1000;

end

CC: begin //C

length = 2'b11;

Code = 4'b1010;

end

DD: begin //D

length = 3;

Code = 4'b1000;

end

EE: begin //E

length = 1;

Code = 4'b0000;

end

FF: begin //F