Diane Shan

862148900

EE120A Section 23

Jack Huang

Lab 4 – Sequential Logic Design

**Overview**

There were three parts to this lab. In the first part, we did the implementation of a sequential circuit of a flight attendant call system. Then in the second part, we did the design and implementation of a state machine. In the third part, we designed time multiplexing circuits for four-LED display. We were able to successfully implement the call system and reinforced the importance of FSM’s and state output/transition table. We also experienced working with multiplexing circuits for the four-LED display.

**New Concepts**

Excitation Equation – Boolean expressions that represent the inputs to the flip-flops of the sequential circuit in the next clock cycle

Digilent Basys Board – complete, ready-to-use digital circuit development platform and can be used with FPGA

Rising Edge Detector – detects the rising edge of the input signal and produce a pulse with a user defined width as an output signal

**Analysis**

*Procedure*:

Part 1:

1. Read flight attendant system functions and obtain state output/transition table
2. Derive excitation equation leading to a schematic
3. Conduct the behavioral simulation
4. Implement the flight attendant call system using Verilog

Excitation Equation:

c = call; l = cancel; q = current state; d = rising clock edge

D = c’l’q + cl’q’ + cl’q’ + cl’q + clq’ + clq

= l’q + cl’ + cl

Part 2:

1. Rising edge detector description
2. Implement proposed FSM developed in previous step in Verilog
3. Edge detector module implemented using two blocks
   1. One to compute FSM next state
   2. One to store FSM state
4. To produce slower clock, make signal clk come from the board and the clk\_out signal drive the FSM
5. Derive state diagram from spec’s description
6. Show output/transition table
7. Derive excitation equations
8. Design a sequential logic circuit that implements the excitation equation

Part 3:

1. Study the Diligent Basys Board
2. Copy and paste hexto7segment verilog code
3. Implement in verilog the behavioral structure of the board circuit where refresh rate of enable signal is fast enough that it looks like all four displays are on at the same time
4. Use code from other figure and then use source files to synthesize code

*FSM for part 2*:



Truth Table for part 2:

|  |  |  |  |
| --- | --- | --- | --- |
| Input | PS | NS | Output |
| 0 | 00 | 00 | 0 |
| 1 | 00 | 01 | 0 |
| 0 | 01 | 00 | 1 |
| 1 | 01 | 11 | 1 |
| 0 | 11 | 00 | 0 |
| 1 | 11 | 11 | 0 |

Boolean equation for part 2:

NS1 = ((Input)(PS1’)(PS0)) + ((Input)(PS1’)(PS0)) = Input(PS0)

NS0 = ((Input)(PS1’)(PS0’)) + ((Input)(PS1’)(PS0)) + ((Input)(PS1)(PS0)) = Input

Out = ((Input’)(PS1’)(PS0)) + ((Input)(PS1’)(PS0)) = PS1’(PS0)

**Records**

*Schematics*:

Diagram

Description automatically generated

Diagram

Description automatically generated

Schematic for part 2:

Diagram, schematic

Description automatically generated

Part 1

design.sv

timescale 1ns / 1ps

module fasystem\_bh(

input wire clk,

input wire call\_button ,

input wire cancel\_button ,

output reg light\_state

);

// Internal wire

reg c\_state ;

// Combinatorial block

always @(\*) begin

case ({call\_button,cancel\_button})

2'b00: c\_state = light\_state ? 'd1 : 'd0 ;

2'b01: c\_state = 'd0 ;

2'b10: c\_state = 'd1 ;

2'b11: c\_state = 'd1 ;

default : c\_state = 'd0 ;

endcase

end

// Sequential block

always @( posedge clk ) begin

light\_state <= c\_state ;

end

Part 2

module edgedetector\_bh(

input wire clk,

input wire signal,

output reg outedge );

wire slow\_clk ;

reg [1:0] c\_state ;

reg [1:0] r\_state ;

// Define your FSM states

localparam ZERO = 'd0;

localparam CHANGE = 'd1;

localparam ONE = 'd2;

// EECS150 - Digital Design Lecture 17 - Finite State Machines Revisited

// See slide 19

// Code for clkdiv module is given below. Create a new Verilog module in the

//same project with the given code.

clkdiv c1(clk, slow\_clk );

// Comb. logic.

always @(\*) begin

case (r\_state)

ZERO : begin

c\_state = signal ? CHANGE : ZERO ;

outedge = 'd0 ;

end

CHANGE : begin

c\_state = signal ? ONE : ZERO;

outedge = 'd1;

//Your code ;

end

ONE : begin

c\_state = signal ? ONE : ZERO;

outedge = 'd0;

//Your code ;

end

default : begin

c\_state = ZERO ;

outedge = 'd0 ;

end

Part 3

`timescale 1ns / 1ps

module bcdto7led\_bh(

input wire sw0 ,

input wire sw1 ,

input wire sw2 ,

input wire sw3 ,

output reg a ,

output reg b ,

output reg c ,

output reg d ,

output reg e ,

output reg f ,

output reg g ,

output reg dp

);

// Internal wire

wire [3:0] bundle ;

assign bundle = {sw3,sw2,sw1,sw0 } ;

always @(\*) begin

a = 1'b1 ;

b = 1'b1 ;

c = 1'b1 ;

d = 1'b1 ;

e = 1'b1 ;

f = 1'b1 ;

g = 1'b1 ;

dp = 1'b1;

case ( bundle )

4'b0000 : begin // 0

a = 1'b0 ;

b = 1'b0 ;

c = 1'b0 ;

d = 1'b0 ;

e = 1'b0 ;

f = 1'b0 ;

end

4'b0001 : begin // 1

b = 1'b0 ;

c = 1'b0 ;

end

4'b0010 : begin // 2

a = 1'b0 ;

b = 1'b0 ;

d = 1'b0 ;

e = 1'b0 ;

g = 1'b0 ;

end

4'b0011 : begin // 3

a = 1'b0 ;

b = 1'b0 ;

c = 1'b0 ;

d = 1'b0 ;

g = 1'b0 ;

end

4'b0100 : begin // 4

b = 1'b0 ;

c = 1'b0 ;

f = 1'b0 ;

g = 1'b0 ;

end

4'b0101 : begin // 5

a = 1'b0 ;

c = 1'b0 ;

d = 1'b0 ;

f = 1'b0 ;

g = 1'b0 ;

end

4'b0110 : begin // 6

a = 1'b0 ;

c = 1'b0 ;

d = 1'b0 ;

e = 1'b0 ;

f = 1'b0 ;

g = 1'b0 ;

end

4'b0111 : begin // 7

a = 1'b0 ;

b = 1'b0 ;

c = 1'b0 ;

end

4'b1000 : begin // 8

a = 1'b0 ;

b = 1'b0 ;

c = 1'b0 ;

d = 1'b0 ;

e = 1'b0 ;

f = 1'b0 ;

g = 1'b0 ;

end

4'b1001 : begin // 9

a = 1'b0 ;

b = 1'b0 ;

c = 1'b0 ;

d = 1'b0 ;

f = 1'b0 ;

g = 1'b0 ;

end

4'b1010 : begin // A

a = 1'b0 ;

b = 1'b0 ;

c = 1'b0 ;

e = 1'b0 ;

f = 1'b0 ;

g = 1'b0 ;

end

4'b1011 : begin // B

c = 1'b0 ;

d = 1'b0 ;

e = 1'b0 ;

f = 1'b0 ;

g = 1'b0 ;

end

4'b1100 : begin // C

a = 1'b0 ;

d = 1'b0 ;

e = 1'b0 ;

f = 1'b0 ;

end

4'b1101 : begin // D

b = 1'b0 ;

c = 1'b0 ;

d = 1'b0 ;

e = 1'b0 ;

g = 1'b0 ;

end

4'b1110 : begin // E

a = 1'b0 ;

d = 1'b0 ;

e = 1'b0 ;

f = 1'b0 ;

g = 1'b0 ;

end

4'b1111 : begin // F

a = 1'b0 ;

e = 1'b0 ;

f = 1'b0 ;

g = 1'b0 ;

end

endcase

end

endmodule

//PART B

module dispmux\_main\_bh(

input clk , // Clock signal

input sw0, // Switch input

input sw1, // Switch input

input sw2, // Switch input

input sw3, // Switch input

output [3:0] an , // LED selector

output [7:0] sseg // Segment signals

);

wire [7:0] in0;

wire [7:0] in1;

wire [7:0] in2;

wire [7:0] in3;

// ---------------------------------

// Module instantiation bcdto7led

// ---------------------------------

bcdto7led\_bh c1(sw0, sw1, sw2, sw3,

in0[0],in0[1],in0[2],in0[3], in0[4],in0[5],in0[6],in0[7] );

// Your code (3 more modules)

bcdto7led\_bh c2(sw0, sw1, sw2, sw3,

in1[0],in1[1],in1[2],in1[3], in1[4],in1[5],in1[6],in1[7] );

bcdto7led\_bh c3(sw0, sw1, sw2, sw3,

in2[0],in2[1],in2[2],in2[3], in2[4],in2[5],in2[6],in2[7] );

bcdto7led\_bh c4(sw0, sw1, sw2, sw3,

in3[0],in3[1],in3[2],in3[3], in3[4],in3[5],in3[6],in3[7] );

// ---------------------------------

// Module instantiation Mux

// ---------------------------------

disp\_mux\_bh c5(

.clk (clk) ,

.in0 (in0) ,

.in1 (in1) ,

.in2 (in2) ,

.in3 (in3) ,

.an (an) ,

.sseg (sseg ) ) ;

endmodule

module disp\_mux\_bh(

input clk ,

input wire [7:0] in0 ,

input wire [7:0] in1 ,

input wire [7:0] in2 ,

input wire [7:0] in3 ,

output reg [3:0] an ,

output reg [7:0] sseg

);

reg [16:0] r\_qreg ;

reg [16:0] c\_next ;

// Mux \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

always @(\*) begin

case (r\_qreg[1:0])

2'b00 : sseg = in0 ;

2'b01 : sseg = in1 ;

2'b10 : sseg = in2 ;

2'b11 : sseg = in3 ;

endcase

end

// Decoder \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

always @(\*) begin

case (r\_qreg[1:0])

2'b00 : an = ~(4'b0001) ;

2'b01 : an = ~(4'b0010) ;

2'b10 : an = ~(4'b0100) ;

2'b11 : an = ~(4'b1000) ;

endcase

end

// Counter \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

always @(\*) begin

c\_next = r\_qreg + 'd1;

end

// Register

always @(posedge clk) begin

r\_qreg <= c\_next ;

end

endmodule

**Discussion**

The system works according to the provided specifications for all three parts. There were no problems encountered that resulted in system redesign and the code ran well. We were able to make everything work and there are not really any improvements that can be made to any of the systems. We synthesized the code and used sequential blocks to make the code better already, so I don’t think there are really any new things we can add to make it better.

**Conclusion**

In this lab, the purpose was to get familiar with clock synchronous state machine design, synthesis, and implementation. We also got more familiar with constructing a finite state machine state/output diagram along with the state/output transition table. We also learned about how to produce a clock that goes slower. We were able to synthesize our code as well and implement things using sequential blocks. We were able to implement successfully a flight attendant call system, a rising-edge detector, and an LED display.

**Questions**

Part 1:

1. What will happen if the “clock” signal is of very low frequency (1 Hz)?

If the “clock” signal is of very low frequency, this means the time it will take for the clock to make a full cycle will be quite large since period is 1/frequency and the smaller the frequency, the longer the period. This means the clock will take longer to get back to the rising or falling edge dependent on situation. This results in a noticeable difference when output changes.