Lab 12 Submission

**Four 4-Bit Number Sort**

CPE 133 - 03

Michael Hegglin

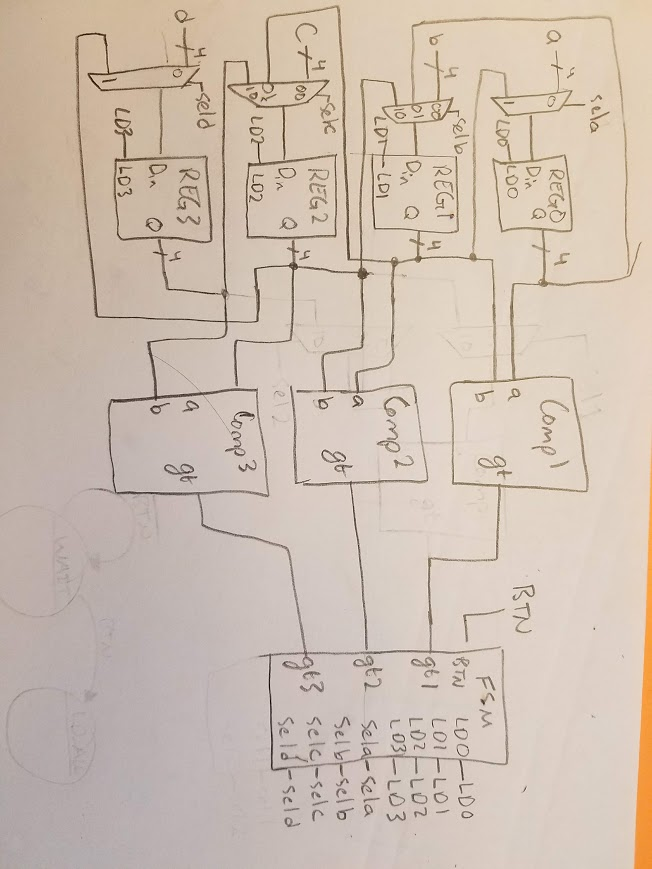
Jonathan Skelly

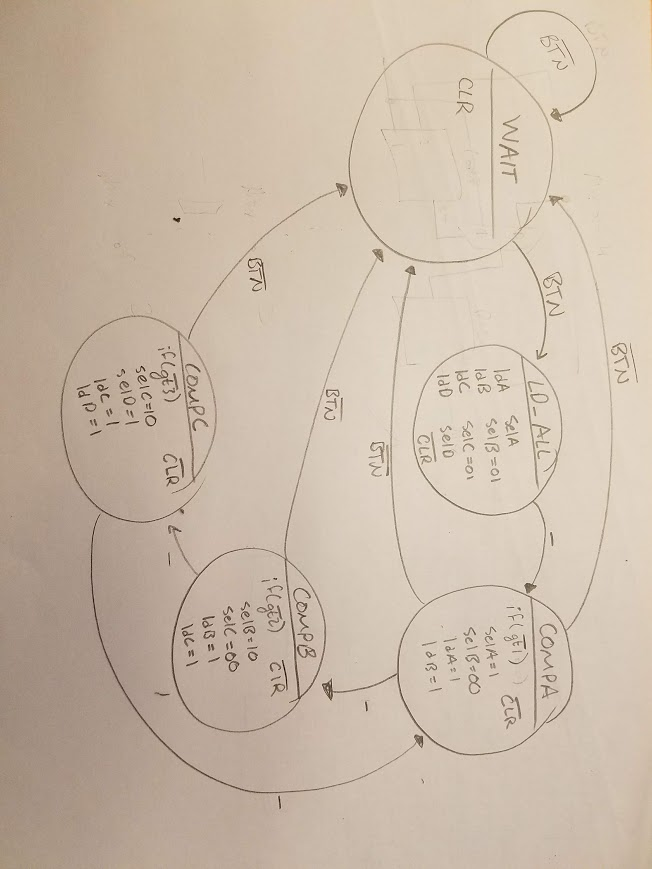
**Executive Summary:**

We designed an 4-bit number sort with an unsigned binary number. The sort used an FSM, a seven segment decoder, registers, a counter, muxes, and external inputs like buttons and switches. The logic decided which numbers would be switched out and stored in each register to sort.



**Upper Level BBD**

**Lower Level BBD**

****

**FSM State Diagram**

**Questions**:

1. In your own words, provide a complete written description of how the circuit you designed in this experiment operates. Be sure to reference the block diagram for your circuit. This description should be more than a detailed description of the state diagram.

* The circuit’s main elements for operation are the registers, comparators, and the finite state machine. The registers hold each of the four numbers inputted, which connect to comparators that compare each pair of adjacent register values. The state machine uses these comparisons to determine whether or not to swap each register’s values (if the “higher” reg has a less value, the values will swapped). This cycle repeats until completion. Using this method, the values in the registers are changed into ascending order.

2. How many clock cycles did your circuit require to complete the sort? Your answer to this question should address best and worst case scenarios (if your approach had both scenarios).

* Worst case scenario the numbers will be sorted within 9 clock cycles, when the highest value is initially in the lowest register. Best case scenario, the numbers are sorted within 3 clock cycles, when the inputted values are already in ascending order.

3. Relating to the previous problem, if the system clock frequency was 100MHz, how much time was required to complete the complete sort operation for the best and least case scenarios. Show your calculations for this problem. Assume your circuit is not using a clock divider. Don’t even think of not writing your answer using engineering notation.

* Worst case : 9\*10^-6 s
* Best case : 3\*10^-6 s

4. There are many sort algorithms out there. State two reasons why you would choose one over another. For this problem, consider your task is to implement the problems in hardware.

* I choose to take a certain sort method over another for speed of the sorting, and efficiency. I want something sorted as fast as possible in the most efficient way.

5. Sorting of course can also be done in software. If would be hard to state whether the algorithm you implemented in hardware in this experiment would run faster than the same algorithm implemented in software. Just for kicks, name a few parameters that you would need to consider if you were to make such a comparison.

* You would need a time variable that keeps track of when each process finishes to see which is fastest. I would also look at the amount of memory each program is eating up.

6. Briefly describe how you would need to change your hardware if your circuit used a switch to decide between sorting the switches in ascending and descending order.

* I would put the switch in an if-statement in the finite state machine. This way the it tells the register what value to take based off the external input.

7. HDL allows you to write complex if statements (if statements containing many conditions) as well as nested if statements. Make a statement about the complexity of an “if” statement and the size of the hardware generated by such statements.

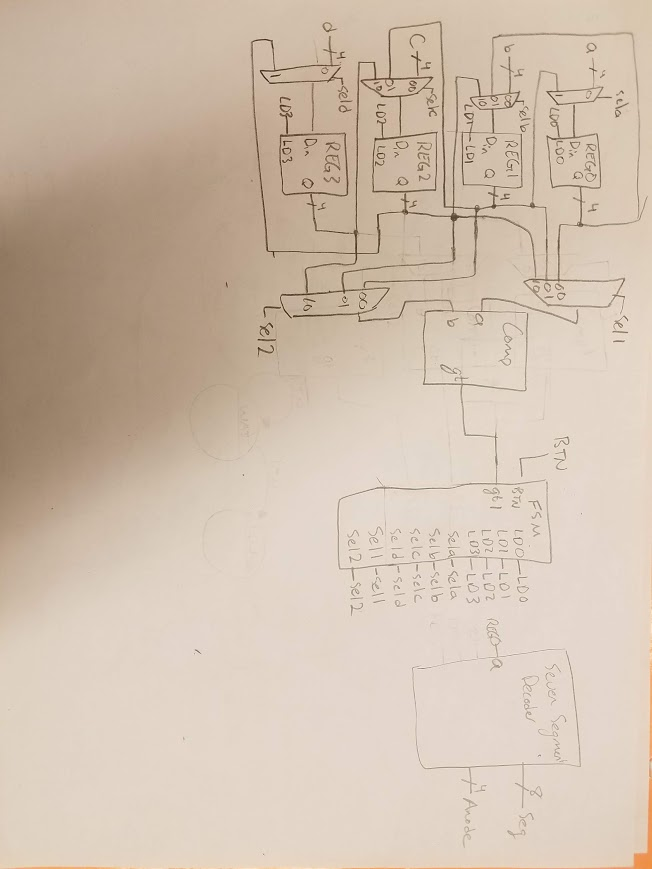
* “If” statements are high-level logical operations. Thus, they do not exactly translate into simple circuits, and compose somewhat of a black-body element. These elements, if containing large lists and many nested “if” statements, would form large black-body hardware sections.

8. Verilog always blocks contain “sensitivity lists”. If you modeled two MUXes in Verilog that were identical except for the fact that one MUX included the select signals in the sensitivity list, and the other not, would the hardware generated by these two MUXes be identical? Briefly explain.

* Yes, they would be identical. In a sensitivity list, different outputs are given based on the number of 1’s in an input to the always block (thus, a list of outputs based on a list of sensitivities). A MUX gives different outputs based on which input is selected via different bit values (also a list of sensitivities). Thus, both instances would work the same if given the same inputs.

**Design Problems**:

1. Design a circuit that has four 8-bit data inputs and one 8-bit output (each are unsigned binary numbers). After a button is pressed, the circuit processes then outputs the largest of the circuit’s four inputs; the value remains on the circuit outputs. The output can only change if the button is pressed again. The button must be released before the circuit finds a new maximum value. The circuit ignores button presses after the initial button press until it finds the maximum value. Assume the circuit inputs do not change after the initial button press. Minimize your use of hardware in this design. Use no more than one comparator in your design. State how the circuit is controlled.



**Source Code:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company: DogsWithJobs

// Engineer: Hegglin Skelly

//

// Create Date: 12/01/2018 08:05:03 AM

// Design Name: Sorter

// Module Name: Main

// Project Name: Four 4-Bit Number Sort

// Target Devices:

// Tool Versions:

// Description:Circuit takes 4 4-bit values and sorts them in

// ascending order

//

// Dependencies:

//

// Revision:

// Revision 1.00 - File Created (07-07-2018)

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module Main(a, b, c, d, startSort\_btn, clk, LED, an, seg);

input [3:0] a, b, c, d;

input startSort\_btn, clk;

wire [5:0] cnt;

reg up;

output reg LED;

output [3:0] an;

output [7:0] seg;

wire [3:0] muxA\_out, muxB\_out, muxC\_out, muxD\_out, comp1, comp2, num;

wire [3:0] reg\_outA, reg\_outB, reg\_outC, reg\_outD;

wire [1:0] selA, selB, selC, selD;

wire eq, lt, gt1, gt2, gt3, ldA, ldB, ldC, ldD, nclk, clr;

clk\_divder\_nbit #(24) MY\_DIV (

.clockin (clk),

.clockout (nclk)

);

//initial muxes choose max number to load into reg

mux\_2t1\_nb #(.n(4)) muxA (

.SEL (selA),

.D0 (a),

.D1 (reg\_outB),

.D\_OUT (muxA\_out) );

mux\_4t1\_nb #(.n(4)) muxB (

.SEL (selB),

.D0 (reg\_outA),

.D1 (b),

.D2 (reg\_outC),

.D3 (0),

.D\_OUT (muxB\_out) );

mux\_4t1\_nb #(.n(4)) muxC (

.SEL (selC),

.D0 (reg\_outB),

.D1 (c),

.D2 (reg\_outD),

.D3 (0),

.D\_OUT (muxC\_out) );

mux\_2t1\_nb #(.n(4)) muxD (

.SEL (selD),

.D0 (d),

.D1 (reg\_outC),

.D\_OUT (muxD\_out) );

//regs store values and output them to the sseg decoder

reg\_nb #(16) MY\_REG1 (

.data\_in (muxA\_out),

.ld (ldA),

.clk (nclk),

.clr (0),

.data\_out (reg\_outA)

);

reg\_nb #(16) MY\_REG2 (

.data\_in (muxB\_out),

.ld (ldB),

.clk (nclk),

.clr (0),

.data\_out (reg\_outB)

);

reg\_nb #(16) MY\_REG3 (

.data\_in (muxC\_out),

.ld (ldC),

.clk (nclk),

.clr (0),

.data\_out (reg\_outC)

);

reg\_nb #(16) MY\_REG4 (

.data\_in (muxD\_out),

.ld (ldD),

.clk (nclk),

.clr (0),

.data\_out (reg\_outD)

);

//comparators compare a reg's stored value with that of the reg below it, outputting which is greater

comp\_nb #(.n(4)) MY\_COMPAB (

.a (reg\_outA),

.b (reg\_outB),

.eq (0),

.gt (gt1),

.lt (0)

);

comp\_nb #(.n(4)) MY\_COMPBC (

.a (reg\_outB),

.b (reg\_outC),

.eq (0),

.gt (gt2),

.lt (0)

);

comp\_nb #(.n(4)) MY\_COMPCD (

.a (reg\_outC),

.b (reg\_outD),

.eq (0),

.gt (gt3),

.lt (lt)

);

//fsm controls loading of regs

fsm\_template myFSM(

.clk (nclk),

.startSort\_btn (startSort\_btn),

.ldA (ldA),

.ldB (ldB),

.ldC (ldC),

.ldD (ldD),

.gt1 (gt1),

.gt2 (gt2),

.gt3 (gt3),

.selA (selA),

.selB (selB),

.selC (selC),

.selD (selD),

.clr (clr)

);

//counter and always block to turn off LED when 18 clock cycles (maximum time to sort) has passed

cntr\_up\_clr\_nb #(.n(5)) MY\_CNTR (

.clk (nclk),

.clr (clr),

.up (up),

.ld (0),

.D (0),

.count (cnt),

.rco (0) );

always @ (cnt)

begin

if (cnt == 9)

begin

LED = 0;

up = 0;

end

else

begin

LED = 1;

up = 1;

end

end

//sseg decoder outputs vaues to LED display

univ\_sseg mySSEG(

.reg\_outA (reg\_outA),

.reg\_outB (reg\_outB),

.reg\_outC (reg\_outC),

.reg\_outD (reg\_outD),

.clk (clk),

.seg (seg),

.an (an)

);

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company: Ratner Surf Designs

// Engineer: James Ratner

//

// Create Date: 07/07/2018 08:05:03 AM

// Design Name:

// Module Name: fsm\_template

// Project Name:

// Target Devices:

// Tool Versions:

// Description: Generic FSM model with both Mealy & Moore outputs.

// Note: data widths of state variables are not specified

//

// Dependencies:

//

// Revision:

// Revision 1.00 - File Created (07-07-2018)

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module fsm\_template(clk, startSort\_btn, ldA, ldB, ldC, ldD, gt1, gt2, gt3, selA, selB, selC, selD, clr);

input clk, startSort\_btn, gt1, gt2, gt3;

output reg [1:0] selA, selB, selC, selD;

output reg ldA, ldB, ldC, ldD, clr;

//- next state & present state variables

reg [2:0] NS, PS;

//- bit-level state representations

parameter [2:0] st\_WAIT=3'b000, st\_LDALL=3'b001, st\_COMPA=3'b010, st\_COMPB=3'b011, st\_COMPC=3'b100; //st\_COMPD=3'b101;

//- model the state registers

always @ (negedge startSort\_btn, posedge clk)

if (startSort\_btn == 0)

PS <= st\_WAIT;

else

PS <= NS;

//- model the next-state and output decoders

always @ (startSort\_btn, gt1, gt2, gt3, PS)

begin

ldA = 0; ldB = 0; ldC = 0; ldD = 0; // assign all outputs

selA = 0; selC = 00; selB = 00; selD = 0;

case(PS)

st\_WAIT:

begin

clr = 1;

if (startSort\_btn == 1)

begin

NS = st\_LDALL;

end

else

begin

NS = st\_WAIT;

end

end

st\_LDALL:

begin

ldA = 1;

ldB = 1;

ldC = 1;

ldD = 1;

selA = 0;

selB = 01;

selC = 01;

selD = 0;

clr = 0;

NS = st\_COMPA;

end

st\_COMPA:

begin

clr = 0;

ldA = 0; ldB = 0; ldC = 0; ldD = 0;

if (gt1 == 0)

begin

selA = 1;

selB = 00;

ldA = 1;

ldB = 1;

end

if (startSort\_btn == 0)

begin

NS = st\_WAIT;

end

else

begin

NS = st\_COMPB;

end

end

st\_COMPB:

begin

clr = 0;

ldA = 0; ldB = 0; ldC = 0; ldD = 0;

if (gt2 == 0)

begin

selB = 10;

selC = 00;

ldB = 1;

ldC = 1;

end

if (startSort\_btn == 0)

begin

NS = st\_WAIT;

end

else

begin

NS = st\_COMPC;

end

end

st\_COMPC:

begin

clr = 0;

ldA = 0; ldB = 0; ldC = 0; ldD = 0;

if (gt3 == 0)

begin

selC = 10;

selD = 1;

ldC = 1;

ldD = 1;

end

if (startSort\_btn == 0)

begin

NS = st\_WAIT;

end

else

begin

NS = st\_COMPA;

end

end

default: NS = st\_WAIT;

endcase

end

endmodule

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company: DogsWithJobs

// Engineer: Skelly/Hegglin

//

// Create Date: 11/02/2018 08:05:03 AM

// Design Name: Seven Seg Decoder

// Module Name: univ\_sseg

// Project Name: 4-Bit Up/Down Counter w/ Multiplexed 7-Segment

// Display

// Target Devices:

// Tool Versions:

// Description: Decoder for outputting binary values as decimal values // on the 7-segment LED display

//

// Dependencies:

//

// Revision:

// Revision 1.00 - File Created (07-07-2018)

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module univ\_sseg(reg\_outA, reg\_outB, reg\_outC, reg\_outD, clk, seg, an );

input [3:0] reg\_outA, reg\_outB, reg\_outC, reg\_outD;

input clk;

wire nclk;

wire [3:0] num;

output reg [7:0] seg;

output reg [3:0] an;

reg [2:0] m\_cnt;

reg [1:0] choose;

clk\_divder\_nbit #(10) MY\_DIV (

.clockin (clk),

.clockout (nclk)

);

mux\_4t1\_nb #(.n(4)) my\_4t1\_mux (

.SEL (choose),

.D0 (reg\_outA),

.D1 (reg\_outB),

.D2 (reg\_outC),

.D3 (reg\_outD),

.D\_OUT (num) );

always @(num, an, seg)

begin

//an = 4'b0111;

case (num)

0: seg = 8'b00000011;

1: seg = 8'b10011111;

2: seg = 8'b00100101;

3: seg = 8'b00001101;

4: seg = 8'b10011001;

5: seg = 8'b01001001;

6: seg = 8'b01000001;

7: seg = 8'b00011111;

8: seg = 8'b00000001;

9: seg = 8'b00011001;

10: seg = 8'b00010001;

11: seg = 8'b11000001;

12: seg = 8'b01100011;

13: seg = 8'b10000101;

14: seg = 8'b01100001;

15: seg = 8'b01110001;

default: seg = 8'b11111111;

endcase

end

always @ (m\_cnt)

begin

case (m\_cnt)

0: an = 4'b1110;

1: an = 4'b1111;

2: an = 4'b1101;

3: an = 4'b1111;

4: an = 4'b1011;

5: an = 4'b1111;

6: an = 4'b0111;

7: an = 4'b1111;

default an = 0;

endcase

end

always @ (m\_cnt)

begin

case (m\_cnt)

0: choose = 2'b00;

1: choose = 2'b00;

2: choose = 2'b01;

3: choose = 2'b01;

4: choose = 2'b10;

5: choose = 2'b10;

6: choose = 2'b11;

7: choose = 2'b11;

default choose = 0;

endcase

end

//- counter that drives the multiplexed display

always @ (posedge nclk)

begin

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

end

endmodule