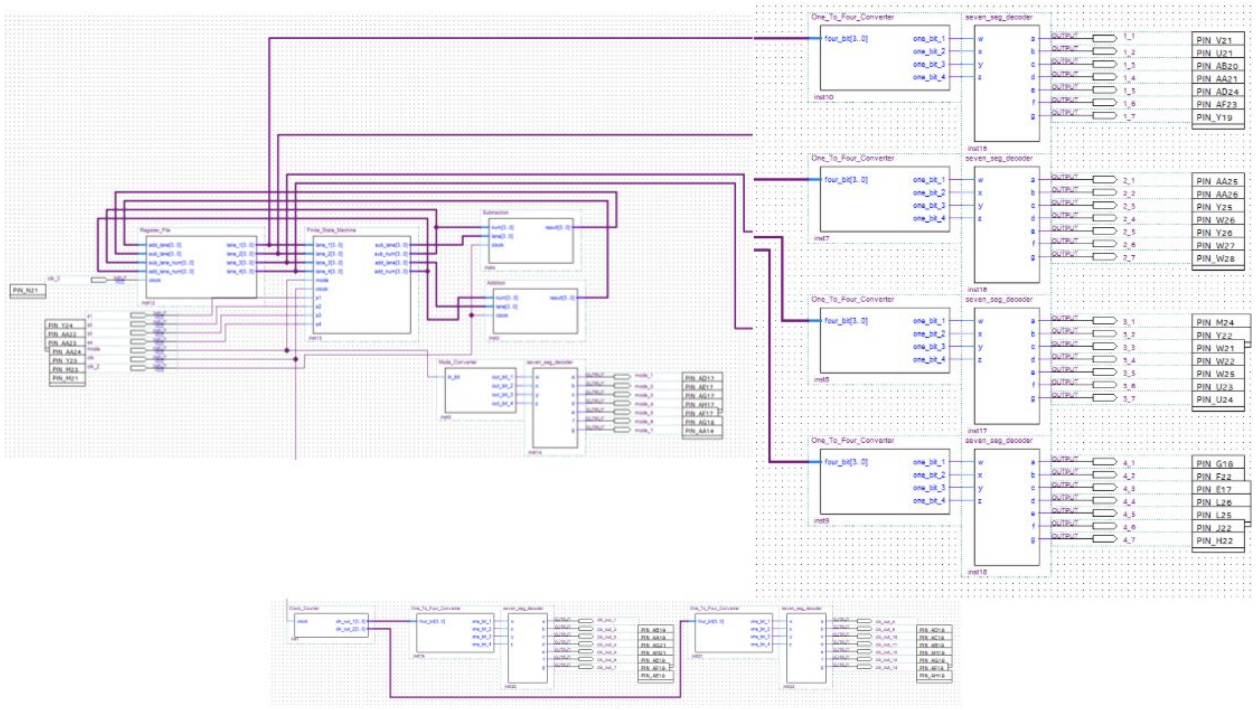
Josh Loftus
Section: U
SID: 055980026

Stop Light Project

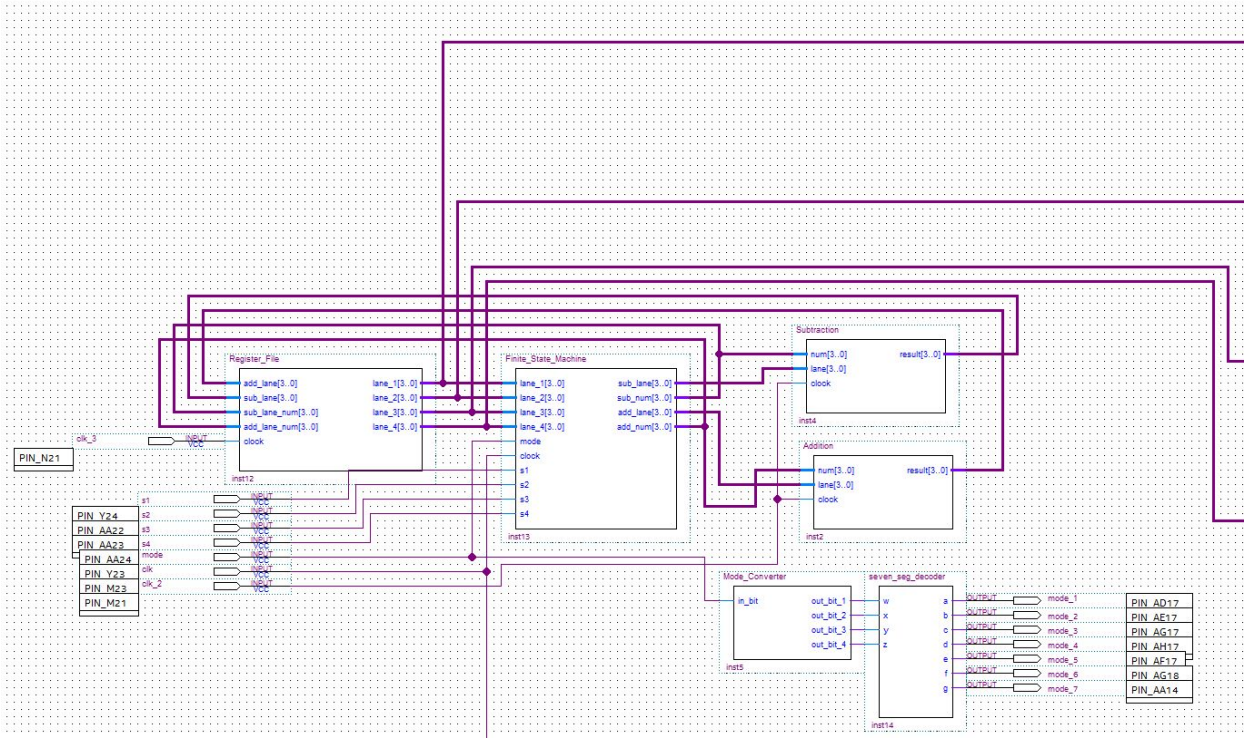
All verilog files found at the bottom and enlarged circuit pictures found in email.

Overview of Whole Circuit (Picture 1-3):

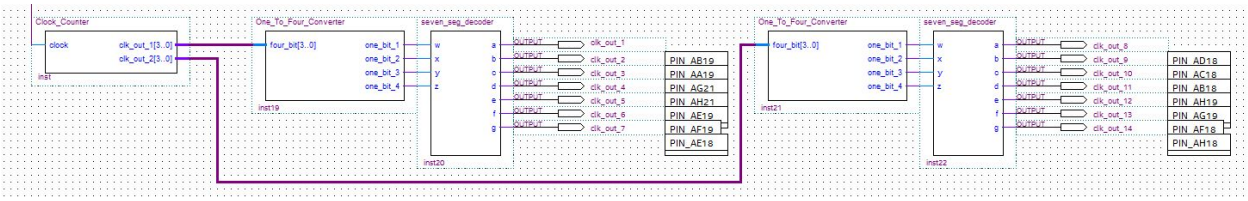


Note in perfect alignment

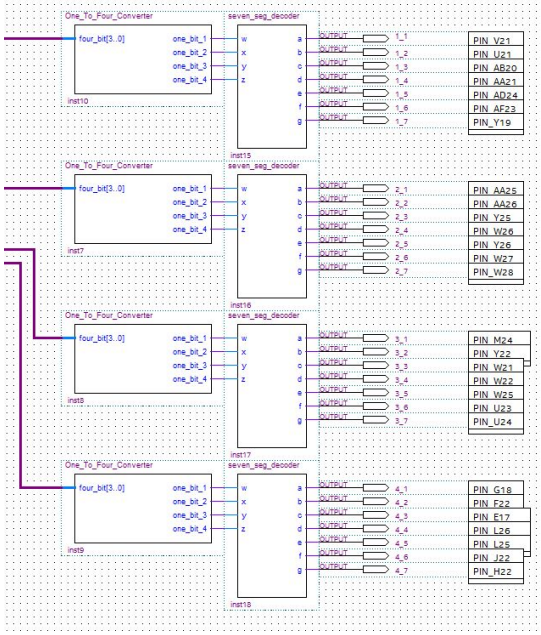
Overview of Usefully Part of the Circuit (Top-Left, Picture 1):



Clock 7 Segment Decoder Section (Bottom, Picture 2):



Lane 7 Segment Decoder Section (Top-Right, Picture 3):



List of all block elements:

1. One_to_four converter
2. Seven_seg_decoder
3. Clock_counter
4. Register_file
5. Finite_state_machine
6. Addition
7. Subtraction

Summary of each block:

1. One_to_four_converter: A single four bit input to four single bit outputs.
2. Seven_seg_decoder: Same as the one in previous labs.
3. Clock_counter: Keeps track of the number of cycles the program has completed
4. Register_File: Takes the add lane changed and the add_lane number then replaces the appropriate line with the changed line. Same for the subtraction lane changed and sub_lane number.
5. Finite_State_Machine: Checks which lane will be added to and which lane will be subtracted from. If the mode is not b the subtraction lane number will always be zero and thus ignored in the subtraction block.
6. Addition: Takes given lane car count and increases it by one.
7. Subtraction: Takes given lane car count and decreases it by one.
8. *Note that this circuit is driven by 3 seperate clocks attached to each of their own keys in order to prevent a loop that set all of my lanes to "F" or capacity right after programming the board and the number of cycles still being zero.*

Test Case 1:

1. Have lane 1 and lane 4 both in the on state and have the mode be set to A.
2. Trigger 16 cycles, lane 4 will be increased because the priority is set to 4, 3, 2, 1. After 16 cycles lane 4 will be at capacity and after that lane 1 will be increased. This proves the capacity stops cars from being added and also that lane priority works.
3. After a few cycles at lane 1 switch lane 1 off to show that even if no other lane is on, a lane at capacity will not be filled.

Test Case 2:

1. Have lane 1 and 3 increased to 3 each in mode A then switch to mode B, since subtraction priority is reverse of addition priority (i.e. 1, 2, 3, 4) lane 1 will begin to decrease and lane 3 will continue to increase for the 3 cycles until lane 1 reaches zero.
2. Once lane 1 reaches zero the subtractor will move to the next available lane with is 3 that now has a car count of 6, however since this lane is also being added to there is no change.
3. This test proves subtraction lane priority works, addition and subtraction of the same lane results in no change, and that once the prioritized lane reaches zero the subtractor will move to a lane of less priority. Also note that subtraction is not dependant on a lane's switch being active like the addition function is, if the mode is switched to B there will always be a lane being subtracted from.

Work done outside the project file

1. A diagram included in my email that breaks down each part of this project, remade due to bad handwriting.
2. Besides the diagram mentioned above, all of my work was done in entire separate projects that ended up being abandoned. I definitely learned from my mistakes for this project as I completed a total of 3 unique projected before getting this last one to work. Previous projects were more dependant on block diagrams

rather than verilog and had different symbol names so I didn't want to include them as this is the only project I got to work.

Verilog of all blocks

```
module Register_File (lane_1, lane_2, lane_3, lane_4, add_lane, sub_lane, sub_lane_num, add_lane_num, clock);

	input [3:0] add_lane, sub_lane, sub_lane_num, add_lane_num;
	input clock;
	output lane_1, lane_2, lane_3, lane_4;

	reg [3:0] lane_1, lane_2, lane_3, lane_4;
	reg [3:0] write_1, write_2, write_3, write_4;

	always @(posedge clock) begin
		if (add_lane_num == 4'b0001) begin
			write_1 = add_lane;
		end
		if (add_lane_num == 4'b0010) begin
			write_2 = add_lane;
		end
		if (add_lane_num == 4'b0011) begin
			write_3 = add_lane;
		end
		if (add_lane_num == 4'b0100) begin
			write_4 = add_lane;
		end
		if (sub_lane_num == 4'b0001) begin
			write_1 = sub_lane;
		end
		if (sub_lane_num == 4'b0010) begin
			write_2 = sub_lane;
		end
		if (sub_lane_num == 4'b0011) begin
			write_3 = sub_lane;
		end
		if (sub_lane_num == 4'b0100) begin
			write_4 = sub_lane;
		end
	end

	assign lane_1 = write_1;
	assign lane_2 = write_2;
	assign lane_3 = write_3;
	assign lane_4 = write_4;

endmodule
```

```
module Finite_State_Machine (lane_1, lane_2, lane_3, lane_4, mode, clock, s1, s2, s3, s4, sub_lane, sub_num,
add_lane, add_num);

	input [3:0] lane_1, lane_2, lane_3, lane_4;
	input clock, s1, s2, s3, s4, mode;
	output [3:0] sub_lane, sub_num, add_lane, add_num;

	reg [3:0] sub_lane, sub_num, add_lane, add_num, last_sub_num;
	integer  index, len1, len2, count;

	always @(posedge clock) begin
		if (lane_4 != 4'b1111 & s4 == 1) begin
			add_num = 4'b0100;
			add_lane = lane_4;
		end
		else if (lane_3 != 4'b1111 & s3 == 1) begin
			add_num = 4'b0011;
			add_lane = lane_3;
		end
		else if (lane_2 != 4'b1111 & s2 == 1) begin
			add_num = 4'b0010;
			add_lane = lane_2;
		end
		else if (lane_1 != 4'b1111 & s1 == 1) begin
			add_num = 4'b0001;
			add_lane = lane_1;
		end
		else begin
			add_num = 4'b0000;
		end




		if (mode == 1) begin
			if (lane_1 != 4'b0000 & (count != 5 | last_sub_num !== 4'b0001)) begin
				sub_num = 4'b0001;
				sub_lane = lane_1;
			end
			else if (lane_2 != 4'b0000 & (count != 5 | last_sub_num != 4'b0010)) begin
				sub_num = 4'b0010;
				sub_lane = lane_2;
			end
			else if (lane_3 != 4'b0000 & (count != 5 | last_sub_num != 4'b0011)) begin
				sub_num = 4'b0011;
				sub_lane = lane_3;
```

```
                end
                else if (lane_4 != 4'b0000 & (count != 5 | last_sub_num != 4'b0100)) begin
                        sub_num = 4'b0100;
                        sub_lane = lane_4;
                end
                else begin
                        sub_num = 4'b0000;
                end
        end
        else begin
                sub_num = 4'b0000;
        end



        if (sub_num == last_sub_num & count < 5) begin
                count = count + 1;
        end
        else if (sub_num == last_sub_num & count == 5) begin
        end
        else begin
                count = 1;
        end



        last_sub_num = sub_num;
        if (mode == 1 && sub_num == add_num) begin
                add_num = 4'b0000;
                sub_num = 4'b0000;
        end
    end

endmodule

module Clock_Counter (clk_out_1, clk_out_2, clock);

    input clock;
    output clk_out_1, clk_out_2;

    reg [3:0] clk_out_1, clk_out_2;

    always @(posedge clock) begin
        if (clk_out_1 == 4'b0000) begin
                clk_out_1 = 4'b0001;
        end
        else if (clk_out_1 == 4'b0001) begin
                clk_out_1 = 4'b0010;
```

```
end
else if (clk_out_1 == 4'b0010) begin
        clk_out_1 = 4'b0011;
end
else if (clk_out_1 == 4'b0011) begin
        clk_out_1 = 4'b0100;
end
else if (clk_out_1 == 4'b0100) begin
        clk_out_1 = 4'b0101;
end
else if (clk_out_1 == 4'b0101) begin
        clk_out_1 = 4'b0110;
end
else if (clk_out_1 == 4'b0110) begin
        clk_out_1 = 4'b0111;
end
else if (clk_out_1 == 4'b0111) begin
        clk_out_1 = 4'b1000;
end
else if (clk_out_1 == 4'b1000) begin
        clk_out_1 = 4'b1001;
end
else if (clk_out_1 == 4'b1001 & clk_out_2 == 4'b1001) begin
        clk_out_1 = 4'b0000;
        clk_out_2 = 4'b0000;
end
else if (clk_out_1 == 4'b1001) begin
        clk_out_1 = 4'b0000;
        if (clk_out_2 == 4'b0000) begin
        clk_out_2 = 4'b0001;
        end
        else if (clk_out_2 == 4'b0001) begin
                clk_out_2 = 4'b0010;
        end
        else if (clk_out_2 == 4'b0010) begin
                clk_out_2 = 4'b0011;
        end
        else if (clk_out_2 == 4'b0011) begin
                clk_out_2 = 4'b0100;
        end
        else if (clk_out_2 == 4'b0100) begin
                clk_out_2 = 4'b0101;
        end
        else if (clk_out_2 == 4'b0101) begin
                clk_out_2 = 4'b0110;
        end
        else if (clk_out_2 == 4'b0110) begin
                clk_out_2 = 4'b0111;
```

```
                        end
                        else if (clk_out_2 == 4'b0111) begin
                                clk_out_2 = 4'b1000;
                        end
                        else if (clk_out_2 == 4'b1000) begin
                                clk_out_2 = 4'b1001;
                        end
                end
        end

endmodule
```

```
module One_To_Four_Converter (four_bit, one_bit_1,  one_bit_2,  one_bit_3,  one_bit_4);

        input [3:0] four_bit;
        output one_bit_1, one_bit_2, one_bit_3, one_bit_4;

        reg [0:0] one_bit_1, one_bit_2, one_bit_3, one_bit_4;

        // bit 1 = LSB

        always @(four_bit, one_bit_1, one_bit_2, one_bit_3, one_bit_4) begin
                if (four_bit == 4'b0000) begin
                        one_bit_4 = 1'b0;
                        one_bit_3 = 1'b0;
                        one_bit_2 = 1'b0;
                        one_bit_1 = 1'b0;
                end
                else if (four_bit == 4'b0001) begin
                        one_bit_4 = 1'b1;
                        one_bit_3 = 1'b0;
                        one_bit_2 = 1'b0;
                        one_bit_1 = 1'b0;
                end
                else if (four_bit == 4'b0010) begin
                        one_bit_4 = 1'b0;
                        one_bit_3 = 1'b1;
                        one_bit_2 = 1'b0;
                        one_bit_1 = 1'b0;
                end
                else if (four_bit == 4'b0011) begin
                        one_bit_4 = 1'b1;
                        one_bit_3 = 1'b1;
                        one_bit_2 = 1'b0;
                        one_bit_1 = 1'b0;
                end
                else if (four_bit == 4'b0100) begin
```

```
            one_bit_4 = 1'b0;
            one_bit_3 = 1'b0;
            one_bit_2 = 1'b1;
            one_bit_1 = 1'b0;
end
else if (four_bit == 4'b0101) begin
            one_bit_4 = 1'b1;
            one_bit_3 = 1'b0;
            one_bit_2 = 1'b1;
            one_bit_1 = 1'b0;
end
else if (four_bit == 4'b0110) begin
            one_bit_4 = 1'b0;
            one_bit_3 = 1'b1;
            one_bit_2 = 1'b1;
            one_bit_1 = 1'b0;
end
else if (four_bit == 4'b0111) begin
            one_bit_4 = 1'b1;
            one_bit_3 = 1'b1;
            one_bit_2 = 1'b1;
            one_bit_1 = 1'b0;
end
else if (four_bit == 4'b1000) begin
            one_bit_4 = 1'b0;
            one_bit_3 = 1'b0;
            one_bit_2 = 1'b0;
            one_bit_1 = 1'b1;
end
else if (four_bit == 4'b1001) begin
            one_bit_4 = 1'b1;
            one_bit_3 = 1'b0;
            one_bit_2 = 1'b0;
            one_bit_1 = 1'b1;
end
else if (four_bit == 4'b1010) begin
            one_bit_4 = 1'b0;
            one_bit_3 = 1'b1;
            one_bit_2 = 1'b0;
            one_bit_1 = 1'b1;
end
else if (four_bit == 4'b1011) begin
            one_bit_4 = 1'b1;
            one_bit_3 = 1'b0;
            one_bit_2 = 1'b1;
            one_bit_1 = 1'b1;
end
else if (four_bit == 4'b1100) begin
```

```
                one_bit_4 = 1'b0;
                one_bit_3 = 1'b0;
                one_bit_2 = 1'b1;
                one_bit_1 = 1'b1;
        end
        else if (four_bit == 4'b1101) begin
                one_bit_4 = 1'b1;
                one_bit_3 = 1'b0;
                one_bit_2 = 1'b1;
                one_bit_1 = 1'b1;
        end
        else if (four_bit == 4'b1110) begin
                one_bit_4 = 1'b0;
                one_bit_3 = 1'b1;
                one_bit_2 = 1'b1;
                one_bit_1 = 1'b1;
        end
        else if (four_bit == 4'b1111) begin
                one_bit_4 = 1'b1;
                one_bit_3 = 1'b1;
                one_bit_2 = 1'b1;
                one_bit_1 = 1'b1;
        end
    end

endmodule
```

```
module seven_seg_decoder (w, x, y, z, a, b, c, d, e, f, g);
input w, x, y, z;
output a, b, c, d, e, f, g;
reg a, b, c, d, e, f, g;

always @(w or x or y or z)

    begin

        case({w, x, y, z})

        4'b0000 : {a, b, c, d, e, f, g} = 7'b0000001;
        4'b0001 : {a, b, c, d, e, f, g} = 7'b1001111;
        4'b0010 : {a, b, c, d, e, f, g} = 7'b0010010;
        4'b0011 : {a, b, c, d, e, f, g} = 7'b0000110;
        4'b0100 : {a, b, c, d, e, f, g} = 7'b1001100;
        4'b0101 : {a, b, c, d, e, f, g} = 7'b0100100;
        4'b0110 : {a, b, c, d, e, f, g} = 7'b0100000;
        4'b0111 : {a, b, c, d, e, f, g} = 7'b0001111;
        4'b1000 : {a, b, c, d, e, f, g} = 7'b0000000;
```

```
                4'b1001 : {a, b, c, d, e, f, g} = 7'b0000100;
                4'b1010 : {a, b, c, d, e, f, g} = 7'b0001000;
                4'b1011 : {a, b, c, d, e, f, g} = 7'b1100000;
                4'b1100 : {a, b, c, d, e, f, g} = 7'b0110001;
                4'b1101 : {a, b, c, d, e, f, g} = 7'b1000010;
                4'b1110 : {a, b, c, d, e, f, g} = 7'b0110000;
                4'b1111 : {a, b, c, d, e, f, g} = 7'b0111000;

                endcase
        end
endmodule
```

```
module Mode_Converter (in_bit, out_bit_1, out_bit_2, out_bit_3, out_bit_4);

        input in_bit;
        output out_bit_1, out_bit_2, out_bit_3, out_bit_4;
        reg out_bit_1, out_bit_2, out_bit_3, out_bit_4;

        // MSB = 1
        always @(in_bit, out_bit_1, out_bit_2, out_bit_3, out_bit_4) begin
                if (in_bit == 0) begin
                        out_bit_4 = 1'b0;
                        out_bit_3 = 1'b1;
                        out_bit_2 = 1'b0;
                        out_bit_1 = 1'b1;
                end
                else begin
                        out_bit_4 = 1'b1;
                        out_bit_3 = 1'b1;
                        out_bit_2 = 1'b0;
                        out_bit_1 = 1'b1;
                end
        end

endmodule
```

```
module Subtraction (num, lane, result, clock);

        input clock;
        input [3:0] num, lane;
        output [3:0] result;

        reg [3:0] result;

        always @(posedge clock) begin
```

```
                if (num != 4'b0000) begin
                        result = lane - 1'b1;
                end
        end
endmodule
```

```
module Addition (result, num, lane, clock);

        input clock;
        input [3:0] num, lane;
        output [3:0] result;

        reg [3:0] result;

        always @(posedge clock) begin
                if (num != 4'b0000 & clock == 1) begin
                        result = lane + 1'b1;
                end
        end
endmodule
```