Digital Systems Design

Engr 378

Lab 5

Scoreboard Controller

Date:
4/19/21

Grade:

By:
Aaron Luong and Richard Couto

Problem Analysis

This lab requested that we design a score counter circuit that could be used in a game setting. This design needed to be responsive to three unique inputs. The first being a reset that could always return the count to zero. Then two different incrementors, an increase of one and an increase of ten. We need to be able to go all the way up to ninety-nine and then hold as we wait for the reset to be triggered/ This value will also be displayed on two, seven-segment displays

Hardware Design

We took a few methods to approach this issue. Our initial plan was to program output for every single digit from zero to ninety-nine, which would have taken hours to write, with hundreds of lines of code and plenty of mistakes to make. Then that idea was condensed where both the full value and digit values were stored and passed through a mux for just zero through nine.

The design we finally implemented saved the true count value and used the divide and modulo operators to grab both the tens and ones place values. We sorted these as the most significant out (msbout) and the least significant out (lsbout) respectively. This way we had fewer states to account for. We still need an encoder for translating both msbout and lsbout into the appropriate seven-segment display signal, which we used one lab prior.

Then we added a module to account for the situation where multiple buttons would be activated simultaneously so the system would know in what order the processes should be carried out. We also see the reset button to be the first step in the coding, so once it is satisfied that reset=0 then it will go through the steps of updating the value.

Verilog Modeling

For the lab, we are making a scoreboard that can count up to ninety-nine. There are two buttons that either increment the scoreboard number by one or ten depending on which button you press. Once the scoreboard reaches ninety-nine, it will stay at ninety-nine until the reset button is pressed.

Condition:

```
module condition(one, ten, CLK, cond1, cond2);
	input one, ten, CLK;
	output cond1, cond2;
	wire oneone, twoone, tenone, tentwo;

	DFlip Oneone(one, CLK, oneone);
	DFlip Onetwo(oneone, CLK, twoone);
	assign cond1 = twoone & ~oneone;
```

```
        DFlip Tenone(ten, CLK, tenone);
        DFlip Tentwo(tenone, CLK, tentwo);
        assign cond2 = tenone & ~tentwo;
endmodule
```

This module makes sure that the button pressed on the FPGA will register as one-click even if the button press takes longer than one clock cycle. The results of this will go to either cond1 or cond2 depending on whether or not the corresponding button has been pressed. This module has used another module called DFlip that acts like how a D-Flip-Flop would act.

D-Flip:

```
module DFlip (in, clock, out);
        input in, clock;
        output reg out;

        always @(posedge clock)begin
                out = in;
        end
endmodule
```

This module is a simple D-Flip-Flop similar to the ones that we have used in previous labs. When the clock pulse becomes positive, it will make the output the input, otherwise, the output won't change.

Increment:

```
module Increment(current, CLK, out, cond1, cond2);
        input CLK, cond1, cond2;
        input [6:0] current;
        output reg[6:0] out;

        always @(posedge CLK)begin
                if(cond1) begin
                        out = current + 1;
                end
                else if(cond2) begin
                        out = current + 10;
                end
                else begin
                        out = current;
                end
                if(out >= 7'b1100011)begin
```

```
                out = 7'b1100011;
            end
        end
endmodule
```

The purpose of this module is to add either one or ten to the current number that is being displayed on the seven-segment display. If cond1 or cond2 is true then the output will become the product of the current number and either one or ten. There is also another state in case an error occurs and neither cond1 nor cond2 is true or if they are both true. If this is the case, then the output will become the current number. The module also makes sure that the maximum number that the output can be is ninety-nine and if the output is more than ninety-nine then it becomes ninety-nine.

Base Number:

```
module basenumber(in, msbout, lsbout);
        input [6:0] in;
        output reg[6:0] msbout, lsbout;

        always @(*)begin
            msbout = in/10;
            lsbout = in%10;
        end
endmodule
```

The purpose of this module is to split the two-digit number into two one-digit numbers so that the numeric number can be displayed on the two seven-segment displays. This is possible by dividing the number by ten to get the left-most number and modulating by ten to get the remainder when dividing by ten and displaying the result as the right-most number.

Encoder:

```
module encoder(in, out);
        input [6:0] in;
        output reg [6:0] out;

        always@(*)begin
            if(in==7'b0000000)begin
                out=7'b1000000;
            end
            else if(in==7'b0000001)begin
                out=7'b1111001;
            end
            else if(in==7'b0000010)begin
```

```
                out=7'b0100100;
            end
            else if(in==7'b0000011)begin
                out=7'b0110000;
            end
            else if(in==7'b0000100)begin
                out=7'b0011001;
            end
            else if(in==7'b0000101)begin
                out=7'b0010010;
            end
            else if(in==7'b0000110)begin
                out=7'b0000010;
            end
            else if(in==7'b0000111)begin
                out=7'b1111000;
            end
            else if(in==7'b0001000)begin
                out=7'b0000000;
            end
            else if(in==7'b0001001)begin
                out=7'b0011000;
            end
            else begin
                out=7'b0000110; //E
            end
        end
endmodule
```

The purpose of this module is to convert the seven-bit binary number to a seven-bit binary number that can be interrupted by the seven-segment display. If the input number is not a binary number between zero and nine, then the output will be a seven-bit number that will display the letter E, meaning error.

Reset Module:

```
module reset(in,CLK,out);
    input in, CLK;
    output reg out;
    reg [2:0] count;

    always @(posedge CLK)begin
        if(in)begin
            if(count==3'b000)begin
                count = 3'b001;
```

```
                end
            end
            else begin
                count = 3'b000;
            end
            if(count>3'b000)begin
                count = count + 1;
                if(count>3'b101)begin
                    count = 3'b000;
                    out = 1;
                end
            end
            else begin
                out = 0;
            end
        end
endmodule
```

The purpose of this class is to make sure that it takes five clock cycles for a reset to take place. If the reset signal lasts for less than five clock cycles, then a reset won't occur. If a reset signal lasts for five or more clock cycles then the module will output a one to the "out" output and will be a zero at any other case.

"Parent" Module:

```
module Lab5(addone, addten, reset, CLK, lsb, msb);
    input addone, addten, reset, CLK;
    output [6:0] lsb, msb;
    reg [6:0] current;
    wire [6:0] LSB, MSB, updated;
    wire cond1, cond2,state;

    initial
    begin
        current = 7'b0000000;
    end

    condition con(addone, addten, CLK, cond1, cond2);
    Increment inc(current, CLK, updated, cond1, cond2);
    reset re(reset, CLK, state);

    always@(*)begin
        current = updated;
```

```
            if(state)begin
                current = 7'b0000000;
            end
        end

        basenumber base(updated, MSB, LSB);
        encoder disp1(LSB, lsb);
        encoder disp2(MSB, msb);

endmodule
```

The purpose of this module is to bring all the modules explained previously together. This module also makes sure that the initial number that the current number is zero.

Results/ Verification

ModelSim:

```
module Lab5Test();
  reg addone, addten, reset, CLK;
  wire [6:0] lsb, msb;
  parameter time_out = 100;

  initial $monitor (addone, addten, reset, CLK, msb, lsb);
  always
            begin
                if (CLK ==1)
                    #1 CLK=0;
                else
                    #1 CLK=1;
            end
        always
            begin
                #0 reset=1;
                #5 reset=0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 addone=0; addten=1;
                #5 addone = 0; addten = 0;
                #5 addone=0; addten=1;
                #5 addone = 0; addten = 0;
                #5 addone=0; addten=1;
```

```
        #5 addone = 0; addten = 0;
        #5 addone=0; addten=1;
        #5 addone = 0; addten = 0;
        #5 addone=0; addten=1;
        #5 addone = 0; addten = 0;
        #5 addone=0; addten=1;
        #5 addone = 0; addten = 0;
        #5 addone=0; addten=1;
        #5 addone = 0; addten = 0;
        #5 addone=0; addten=1;
        #5 addone = 0; addten = 0;
        #5 addone=0; addten=1;
        #5 addone = 0; addten = 0;
        #5 addone=1; addten=0;
        #5 addone = 0; addten = 0;
        #5 addone=1; addten=0;
        #5 addone = 0; addten = 0;
        #5 addone=1; addten=0;
        #5 addone = 0; addten = 0;
        #5 addone=1; addten=0;
        #5 addone = 0; addten = 0;
        #5 addone=1; addten=0;
        #5 addone = 0; addten = 0;
        #5 addone=1; addten=0;
        #5 addone = 0; addten = 0;
        #5 addone=1; addten=0;
        #5 addone = 0; addten = 0;
        #5 addone=1; addten=0;
        #5 addone = 0; addten = 0;
        #5 reset = 1;
        #5 reset = 0;
        #5 reset = 0;
    end
    Lab5 test (addone, addten, reset, CLK, lsb, msb);
endmodule
```

The testbench used in this lab is similar to previous testbenches. The only changes are to the parameters that are being monitored as well as the frequencies that these parameters are changed.

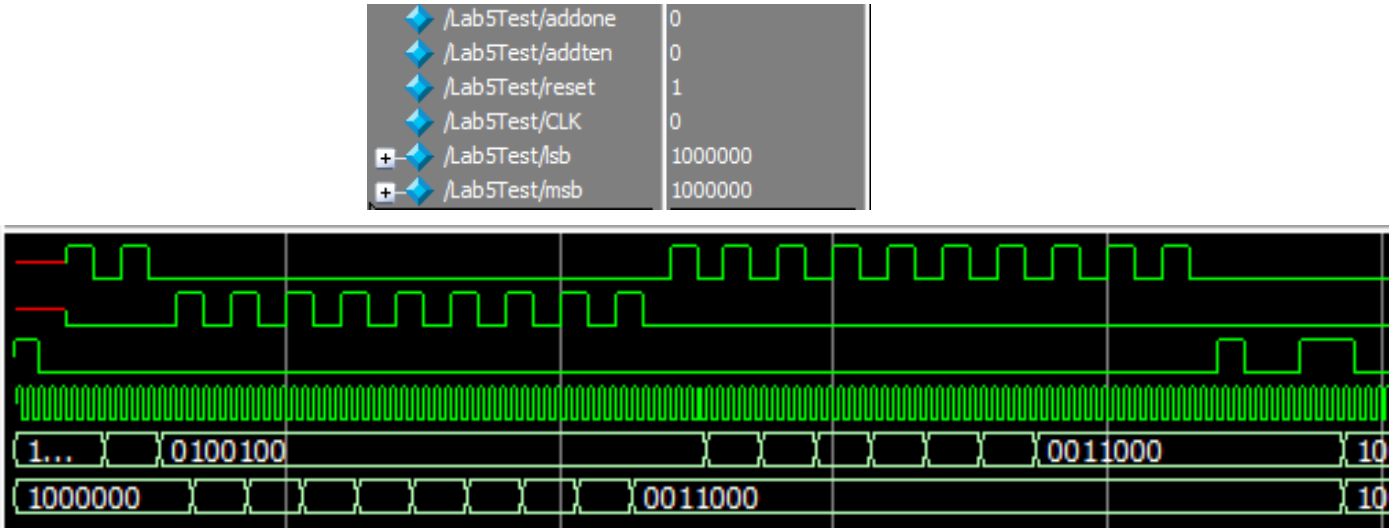

Fig1: ModelSim results

From the results, we can see that the scoreboard works as intended. When the increment-by-one button is pressed, the scoreboard increments by one, and when the increment-by-ten button is pressed, the scoreboard increments by ten. This will continue till the scoreboard reaches ninety-nine and anymore increment button that is pressed are not registered. When the reset button is pressed for less than five clock cycle, nothing will happen and when it last for five or more clock cycle, the value displayed will change to zero.

FPGA:

```
// Include the top-level pin ip file
`include "../hdl/pin_ip.v"

// GUI is DE10-Lite, but actual FPGA is DE0-CV

module top( input CYCLONEV_CLK_50 );

        // Peripheral interconnect wires

                // Use these wires to feed outputs to LED widgets
                wire [9:0]      LEDR;
                // Use these wires as 8-bit active-low 7-segment
                // outputs, where the 8th bit is the decimal point
                wire [7:0]      HEX0, HEX1, HEX2,
                                HEX3, HEX4, HEX5;
                // Use these wires as slide switch inputs
                wire [9:0]      SW;
                // Use these wires as active low, push-button inputs
                wire [1:0]      KEY;
                // These function as programmable register inputs to a
```

```
		// design, useful for adjusting inputs using a keyboard
		wire [31:0] param1, param2, param3;

	// User instantiates design below
		Lab5 test(KEY[0],KEY[1],SW[0],CYCLONEV_CLK_50,HEX0,HEX1);

		/*
		**
		**	Instantiated design connects to pin_ip connected wires
		** these wires function as memory-mapped i/o which is
		** translated to widgets on the GUI
		**
		*/

	// IP to allow simple user design interfacing with developent kit



		pin_ip			platform_designer_pin_ip(
.CYCLONEV_CLK_50(CYCLONEV_CLK_50),

			.LEDR(LEDR),

			.HEX0(HEX0),

			.HEX1(HEX1),

			.HEX2(HEX2),

			.HEX3(HEX3),

			.HEX4(HEX4),

			.HEX5(HEX5),

			.SW(SW),

			.KEY(KEY),

			.param1(param1),

			.param2(param2),

			.param3(param3));
```

endmodule

This FPGA implementation code was mainly provided, by Intel and the only thing that has been changed is that the user-defined design was added. This is the single line of code that calls to use Lab5.v

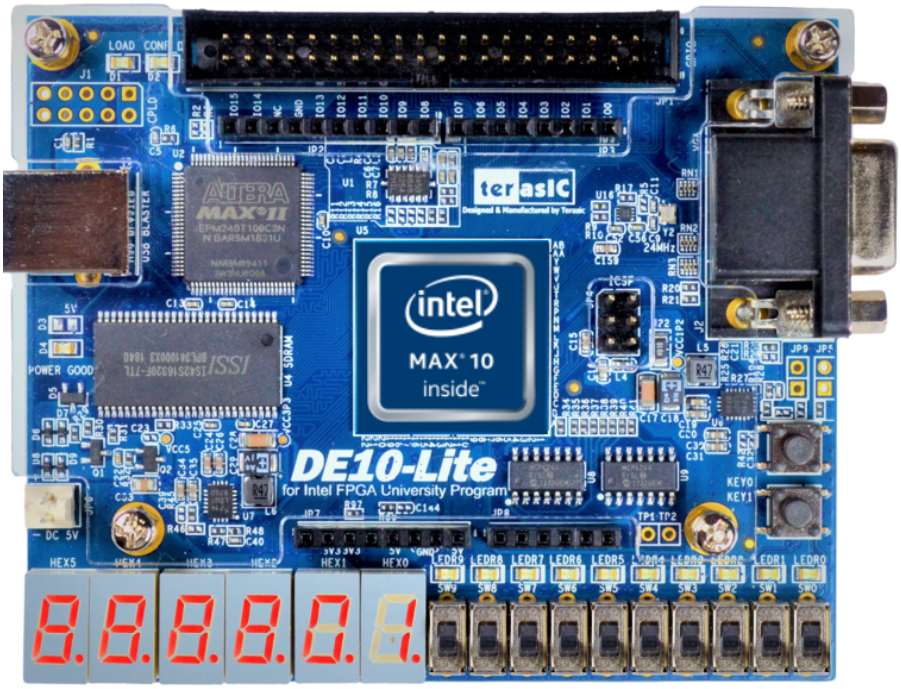

Fig 2: FPGA (increment by one)

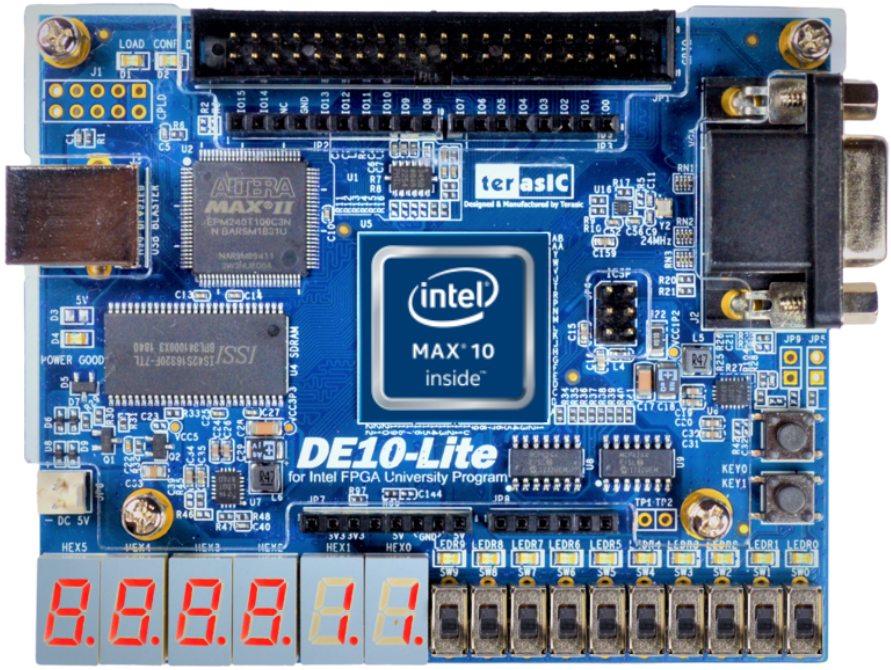

Fig 3: FGPA (increment by ten)

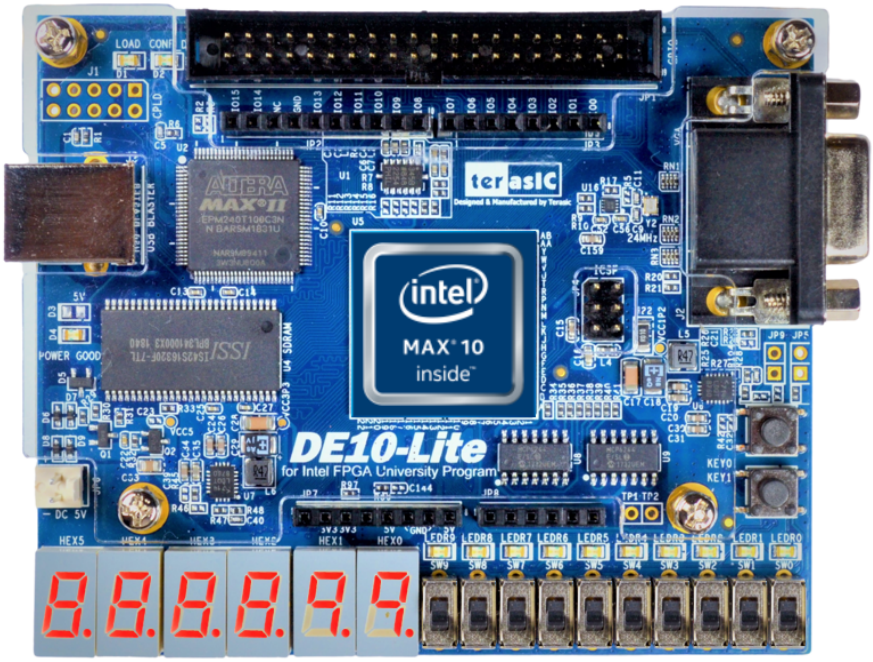

Fig 4: FPGA (max output number)

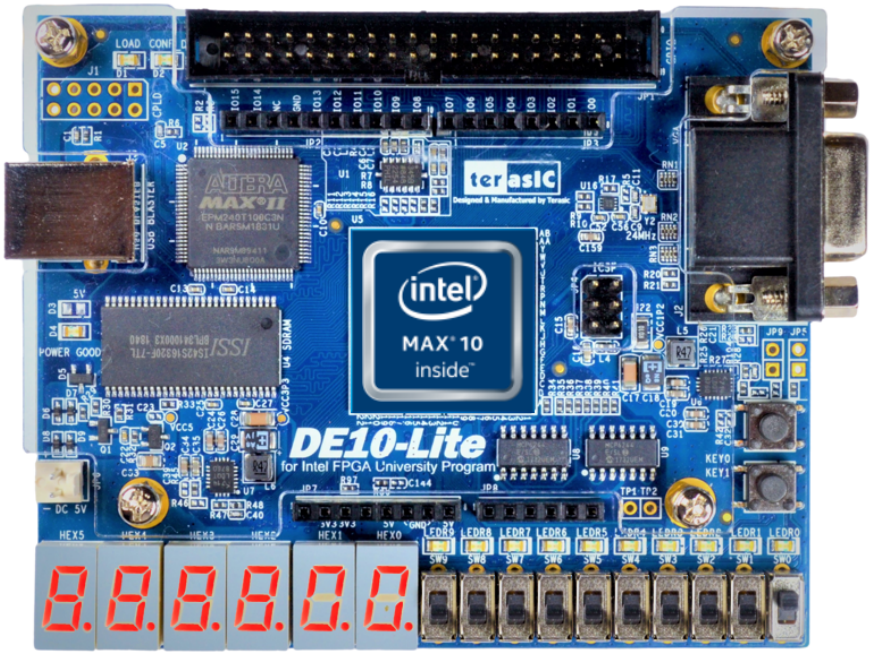

Fig 5: FPGA (reset)

From loading the code onto the FPGA, we can see that the ModelSim results that we gathered earlier are similar to these results. When the increment-by-one button is pressed the seven-segment display updates to show that the current number has been increased by one even when the button is pressed for one whole second with the clock of the FPGA being less

than one second. This also is similar to when the increment-by-ten button is pressed, however, the resulting number is increased by ten instead of one. This continues until the display shows ninety-nine and even when the increment buttons are pressed. Finally when the reset switch is flipped, the seven-segment display changes to zero.

Conclusion and Discussion

Our main issue with this lab was finding an effective alternative to our first method. After we had devised a way to make it work, it became harder to be more creative with the issue. The main problem that we ran into was how to display a number bigger than ten. We initially were thinking about brute-forcing it, but it seemed inefficient and would take up too much time to write. We ended up figuring out that we could use the modulo operator to get the remainder when we divide the current number. With this, we can divide by ten to get the ten's spot number and modulo to get the one's spot number.

Work Breakdown

The encoder and the D-Flip-Flop modules were written by Ricky. The condition, increment, base number, and the main "parent" modules were done by Aaron. The ModelSim code was written by Ricky and the FPGA code was modified by Aaron.

Prelabs

Ricky Prelab:

This was designed while we were under the impression that the value needed to increment from 99 to zero, then to 1, on a loop. For this plan, we are also missing how a tens increment would send us back to the first row from any value on the bottom, however, the state table was beginning to get messy and hard to read.

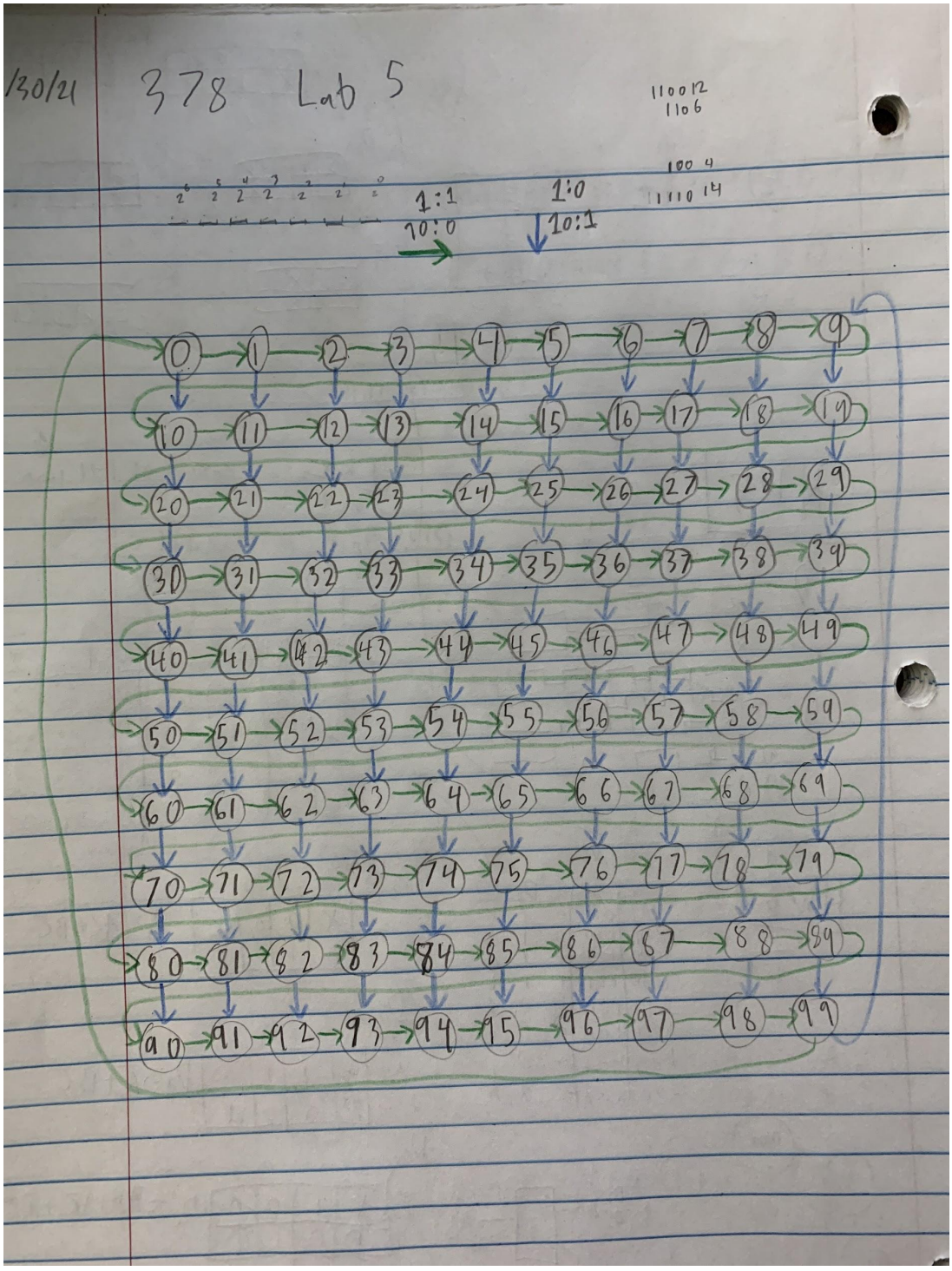

Fig 6: Ricky's Prelab

Aaron Luong:

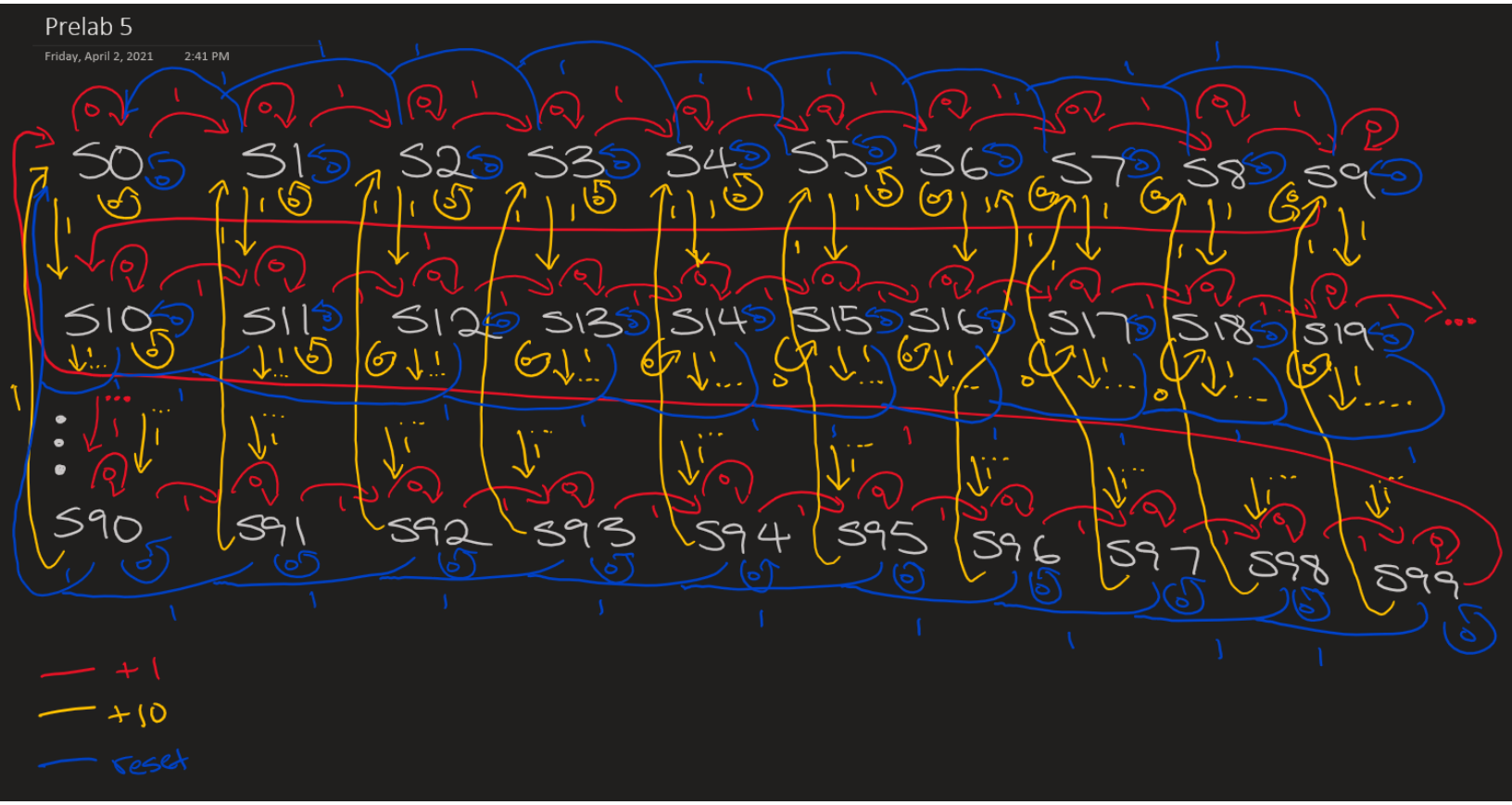

Fig 7: Aaron's Prelab

HDL Source Code

```
module condition(one, ten, CLK, cond1, cond2);
        input one, ten, CLK;
        output cond1, cond2;
        wire oneone, twoone, tenone, tentwo;

        DFlip Oneone(one, CLK, oneone);
        DFlip Onetwo(oneone, CLK, twoone);
        assign cond1 = twoone & ~oneone;

        DFlip Tenone(ten, CLK, tenone);
        DFlip Tentwo(tenone, CLK, tentwo);
        assign cond2 = tenone & ~tentwo;
endmodule

module DFlip (in, clock, out);
```

```
	input in, clock;
	output reg out;

	always @(posedge clock)begin
		out = in;
	end
endmodule

module Increment(current, CLK, out, cond1, cond2);
	input CLK, cond1, cond2;
	input [6:0] current;
	output reg[6:0] out;

	always @(posedge CLK)begin
		if(cond1) begin
			out = current + 1;
		end
		else if(cond2) begin
			out = current + 10;
		end
		else begin
			out = current;
		end
		if(out >= 7'b1100011)begin
			out = 7'b1100011;
		end
	end
endmodule

module basenumber(in, msbout, lsbout);
	input [6:0] in;
	output reg[6:0] msbout, lsbout;

	always @(*)begin
		msbout = in/10;
		lsbout = in%10;
	end
endmodule

module encoder(in, out);
	input [6:0] in;
	output reg [6:0] out;

	always@(*)begin
```

```
                if(in==7'b0000000)begin
                        out=7'b1000000;
                end
                else if(in==7'b0000001)begin
                        out=7'b1111001;
                end
                else if(in==7'b0000010)begin
                        out=7'b0100100;
                end
                else if(in==7'b0000011)begin
                        out=7'b0110000;
                end
                else if(in==7'b0000100)begin
                        out=7'b0011001;
                end
                else if(in==7'b0000101)begin
                        out=7'b0010010;
                end
                else if(in==7'b0000110)begin
                        out=7'b0000010;
                end
                else if(in==7'b0000111)begin
                        out=7'b1111000;
                end
                else if(in==7'b0001000)begin
                        out=7'b0000000;
                end
                else if(in==7'b0001001)begin
                        out=7'b0011000;
                end
                else begin
                        out=7'b0000110; //E
                end
        end
endmodule

module reset(in,CLK,out);
        input in, CLK;
        output reg out;
        reg [2:0] count;

        always @(posedge CLK)begin
                if(in)begin
                        if(count==3'b000)begin
```

```
                        count = 3'b001;
                    end
                end
                else begin
                    count = 3'b000;
                end
                if(count>3'b000)begin
                    count = count + 1;
                    if(count>3'b101)begin
                        count = 3'b000;
                        out = 1;
                    end
                end
                else begin
                    out = 0;
                end
        end
endmodule

module Lab5(addone, addten, reset, CLK, lsb, msb);
        input addone, addten, reset, CLK;
        output [6:0] lsb, msb;
        reg [6:0] current;
        wire [6:0] LSB, MSB, updated;
        wire cond1, cond2,state;

        initial
        begin
            current = 7'b0000000;
        end

        condition con(addone, addten, CLK, cond1, cond2);
        Increment inc(current, CLK, updated, cond1, cond2);
        reset re(reset, CLK, state);

        always@(*)begin
            current = updated;
            if(state)begin
                current = 7'b0000000;
            end
        end

        basenumber base(updated, MSB, LSB);
        encoder disp1(LSB, lsb);
```

```
        encoder disp2(MSB, msb);

endmodule

module Lab5Test();
  reg addone, addten, reset, CLK;
  wire [6:0] lsb, msb;
  parameter time_out = 100;

  initial $monitor (addone, addten, reset, CLK, msb, lsb);
  always
            begin
                  if (CLK ==1)
                        #1 CLK=0;
                  else
                        #1 CLK=1;
            end
      always
            begin
                  #0 reset=1;
                  #5 reset=0;
                  #5 addone=1; addten=0;
                  #5 addone = 0; addten = 0;
                  #5 addone=1; addten=0;
                  #5 addone = 0; addten = 0;
                  #5 addone=0; addten=1;
                  #5 addone = 0; addten = 0;
                  #5 addone=0; addten=1;
                  #5 addone = 0; addten = 0;
                  #5 addone=0; addten=1;
                  #5 addone = 0; addten = 0;
                  #5 addone=0; addten=1;
                  #5 addone = 0; addten = 0;
                  #5 addone=0; addten=1;
                  #5 addone = 0; addten = 0;
                  #5 addone=0; addten=1;
                  #5 addone = 0; addten = 0;
                  #5 addone=0; addten=1;
                  #5 addone = 0; addten = 0;
                  #5 addone=1; addten=0;
```

```
                #5 addone = 0; addten = 0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 addone=1; addten=0;
                #5 addone = 0; addten = 0;
                #5 reset = 1;
                #5 reset = 0;
                #5 reset = 0;
        end
        Lab5 test (addone, addten, reset, CLK, lsb, msb);
endmodule

// Include the top-level pin ip file
`include "../hdl/pin_ip.v"

// GUI is DE10-Lite, but actual FPGA is DE0-CV

module top( input CYCLONEV_CLK_50 );

        // Peripheral interconnect wires

                // Use these wires to feed outputs to LED widgets
                wire [9:0]      LEDR;
                // Use these wires as 8-bit active-low 7-segment
                // outputs, where the 8th bit is the decimal point
                wire [7:0]      HEX0, HEX1, HEX2,
                                HEX3, HEX4, HEX5;
                // Use these wires as slide switch inputs
                wire [9:0]      SW;
                // Use these wires as active low, push-button inputs
```

```
        wire [1:0]      KEY;
        // These function as programmable register inputs to a
        // design, useful for adjusting inputs using a keyboard
        wire [31:0] param1, param2, param3;

    // User instantiates design below
        Lab5 test(KEY[0],KEY[1],SW[0],CYCLONEV_CLK_50,HEX0,HEX1);

        /*
        **
        **      Instantiated design connects to pin_ip connected wires
        ** these wires function as memory-mapped i/o which is
        ** translated to widgets on the GUI
        **
        */

    // IP to allow simple user design interfacing with developent kit


        pin_ip                  platform_designer_pin_ip(
.CYCLONEV_CLK_50(CYCLONEV_CLK_50),

                .LEDR(LEDR),

                .HEX0(HEX0),

                .HEX1(HEX1),

                .HEX2(HEX2),

                .HEX3(HEX3),

                .HEX4(HEX4),

                .HEX5(HEX5),

                .SW(SW),

                .KEY(KEY),

                .param1(param1),

                .param2(param2),
```

```
                .param3(param3));

endmodule
```