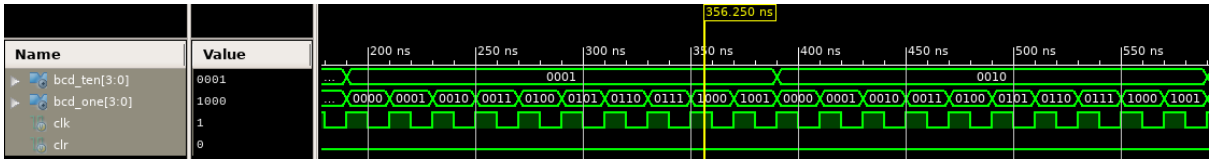
Lab07 Report

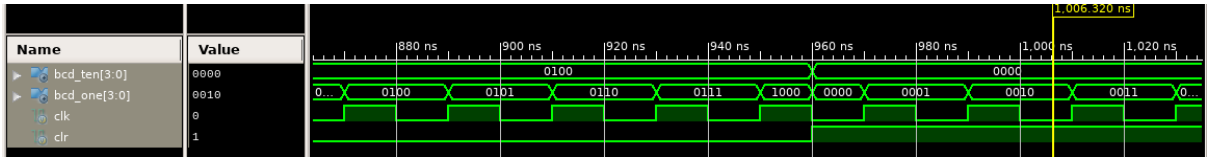
Park Chaehyun

1. Counter module

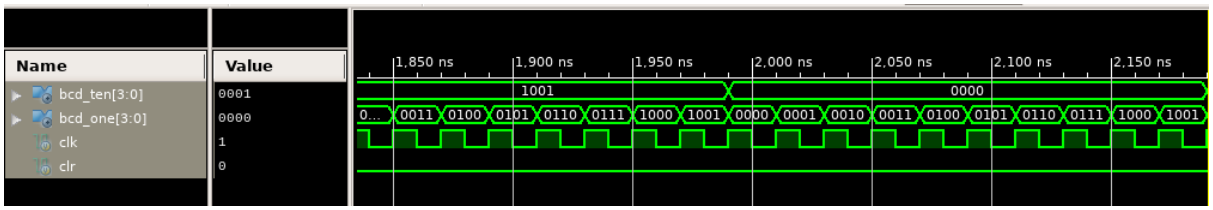
Simulation result:



Here, we check that bcd_one is incremented by 1 at every positive edge of the clock. And when bcd_one is 1001 and another positive edge of the clock comes, bcd_one becomes 0000 and bcd_ten is incremented by 1.



When reset=1, both bcd_ten and bcd_one are set to 0000.



When bcd_ten=1001 and bcd_one=1001, meaning the counter has counted to 99, and another positive edge of the clock comes, both bcd_ten and bcd_one are set to 0000.

2. Two digit counter

(1) Verilog source code

(i) TDC main:

```
module TDC(
        input clk,
        input reset,
        output [6:0] bcd7_ten,
        output [6:0] bcd7_one
        );

        wire new_clk;
        wire [3:0] bcd_ten;
        wire [3:0] bcd_one;

        freq_divider T1(.clr(reset), .clk(clk), .clkout(new_clk));
        counter T2(.clk(new_clk), .reset(reset), .bcd_ten(bcd_ten), .bcd_one(bcd_one));
        bcd_to_7 T3(.bcd(bcd_ten), .seg(bcd7_ten));
        bcd_to_7 T4(.bcd(bcd_one), .seg(bcd7_one));

endmodule
```

It receives clk and reset as input, and gives output for 7-segment display of the ten's digit and one's digit of the counter. It makes an instance of frequency divider to obtain a new clock whose frequency is 1Hz, and gives the new clock as the clock for the counter and the two bcd-to-7-segment decoders. The outputs bcd_ten and bcd_one of the counter is given to the decoders as inputs so the BCD value can be converted to 7-segment.

(ii) counter:

```
module counter(
```

```
	input clk,
	input reset,
	output reg [3:0] bcd_ten,
	output reg [3:0] bcd_one
	);

	always@ (posedge clk or posedge reset) begin
		if(reset) begin
			bcd_ten = 0;
			bcd_one = 0;
		end
		else if(bcd_one == 4'b1001) begin
			if(bcd_ten == 4'b1001) begin
				bcd_ten = 4'b0000;
				bcd_one = 4'b0000;
			end
			else begin
				bcd_ten = bcd_ten + 1;
				bcd_one = 4'b0000;
			end
		end
		else begin
			bcd_one = bcd_one + 1;
		end
	end

endmodule
```

The always statement is triggered at the positive edge of the clock or reset. If reset=1, both outputs are set to 0. If both bcd_one and bcd_ten are 9, then both are set to 0. If bcd_one is 9 and bcd_ten is not 9,bcd_one is set to 0 and bcd_ten is incremented by 1. Otherwise, only bcd_one is incremented by 1.

(iii) frequency divider:

```
module freq_divider(
	input clk,
	input clr,
	output reg clkout
	);

	reg[31:0] cnt;

	always@ (posedge clk) begin
		if(clr) begin
			cnt <=32'd0;
			clkout <= 1'b0;
		end
		else if (cnt == 32'd25000000) begin
			cnt <= 32'd0;
			clkout <= ~clkout;
		end
		else begin
			cnt <= cnt + 1;
		end
	end
endmodule
```

The frequency divider receives clock and clear as input, and gives another clock as output. It has an internal variable count. The count and the output clock is set to 0 when clear is 1. Then count is incremented by 1 at every positive edge of the input clock. When count reaches 25000000(=25M), the output clock is toggled. So if 50MHz clock is given as input, the output clock is flipped every 0.5s, meaning the period is 1s, meaning the frequency is 1Hz.

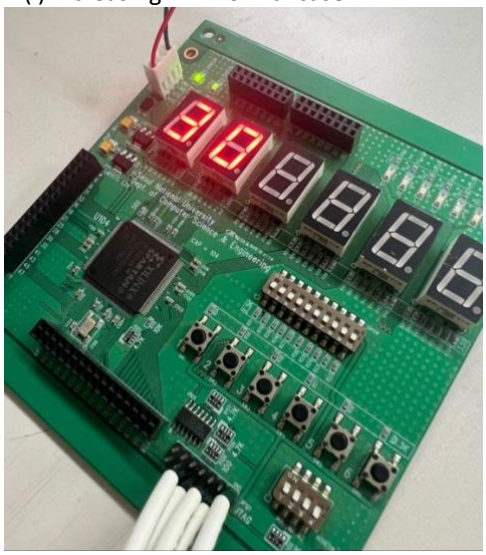
(iv) BCD to 7-segment decoder:

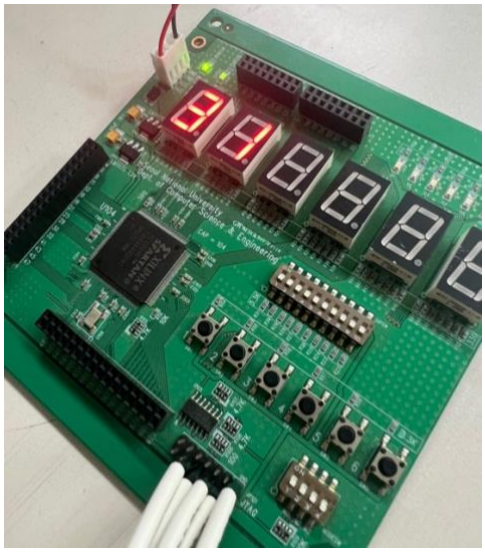
```
module bcd_to_7(
        input [3:0] bcd,
        output reg [6:0] seg
        );

        always@(bcd) begin
                case(bcd)
                        4'd0: seg <= 7'b0111111;
                        4'd1: seg <= 7'b0000110;
                        4'd2: seg <= 7'b1011011;
                        4'd3: seg <= 7'b1001111;
                        4'd4: seg <= 7'b1100110;
                        4'd5: seg <= 7'b1101101;
                        4'd6: seg <= 7'b1111101;
                        4'd7: seg <= 7'b0000111;
                        4'd8: seg <= 7'b1111111;
                        4'd9: seg <= 7'b1101111;
                endcase
        end
endmodule
```

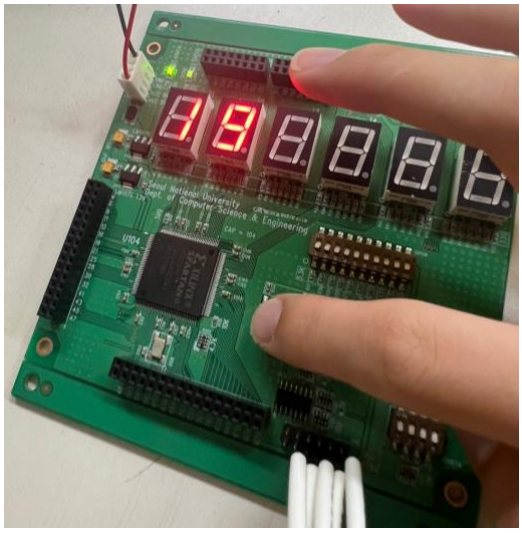
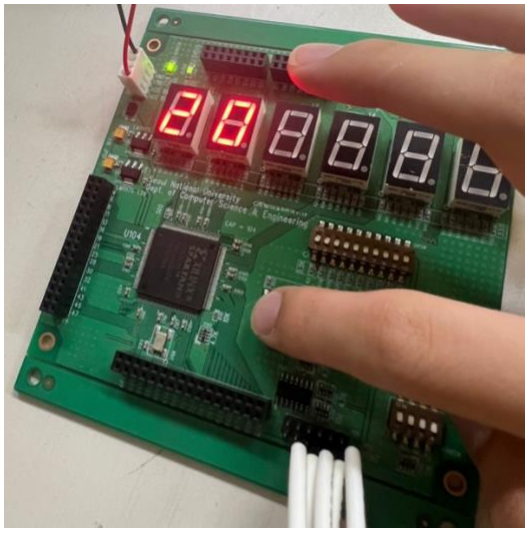
When the input bcd is changed the always statement is triggered, where the output seg is given the 7-segment values of the corresponding BCD value.

(2) Result of implementation

(i) Increasing 1 in normal case

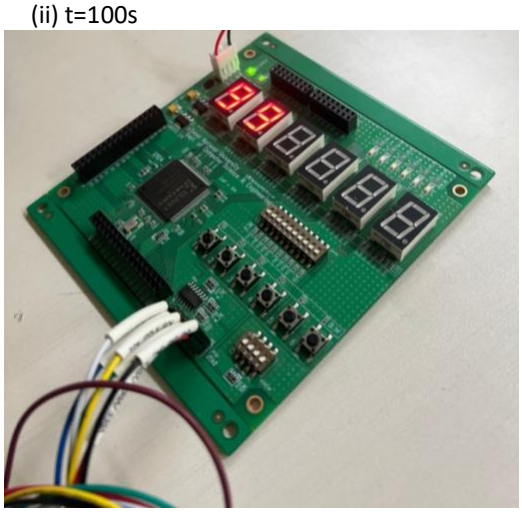
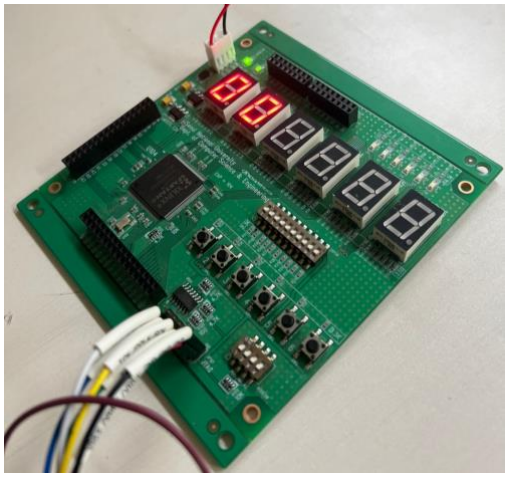




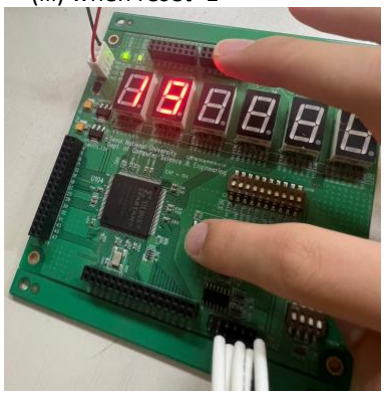
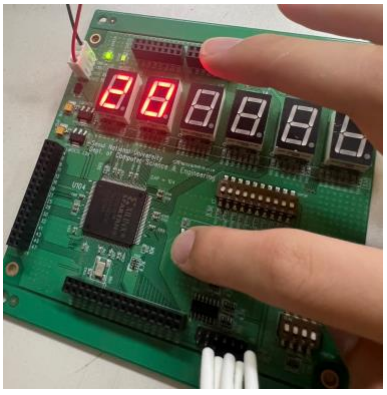
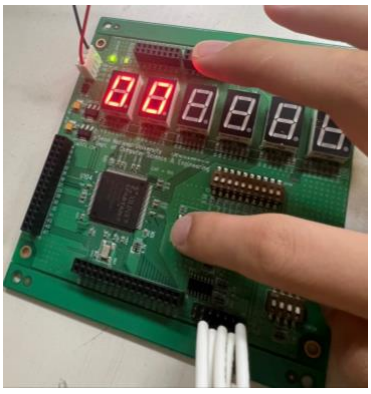
When the counter is not 99 and reset button is not pressed (reset = 0) it increases by 1 every second. We also check when bcd_one is 9 and 1 second passes, bcd_one becomes 0 and bcd_ten is incremented by 1.

(ii) t=100s

When the counter is 99 and 1 second passes, both bcd_ten and bcd_one are set to 0.

(iii) when reset=1

When the counter is counting, and the reset button is pressed to make reset=1, both outputs are im mediately set to 1.

Discussion

During the Verilog implementation, there was a problem where the clock was oscillating as intended but the counter remained undefined. We found that the problem was because the the always statem

ent in the counter module did not have the positive edge of the reset in its sensitivity list. Adding t hat to the sensitivity list, we were able to solve the problem. Using

always@ (posedge clk or posedge reset)

we were able to make a counter that is incremented at every positive edge of the clock and is rese t when reset becomes 1, independent of the clock timing.