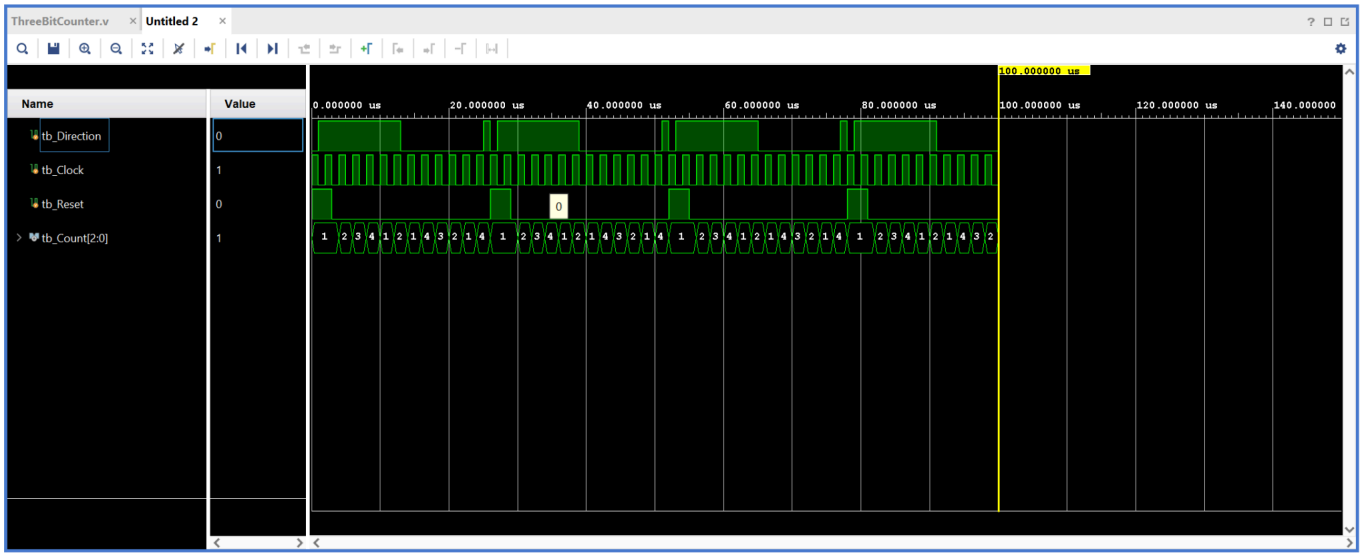
Ayden Dauenhauer, Seana Corners
Prof. Yang
ELEN 122L Tuesday 2:15 p.m.
21 January 2024

Lab 1 Report

Part 1: Simulation

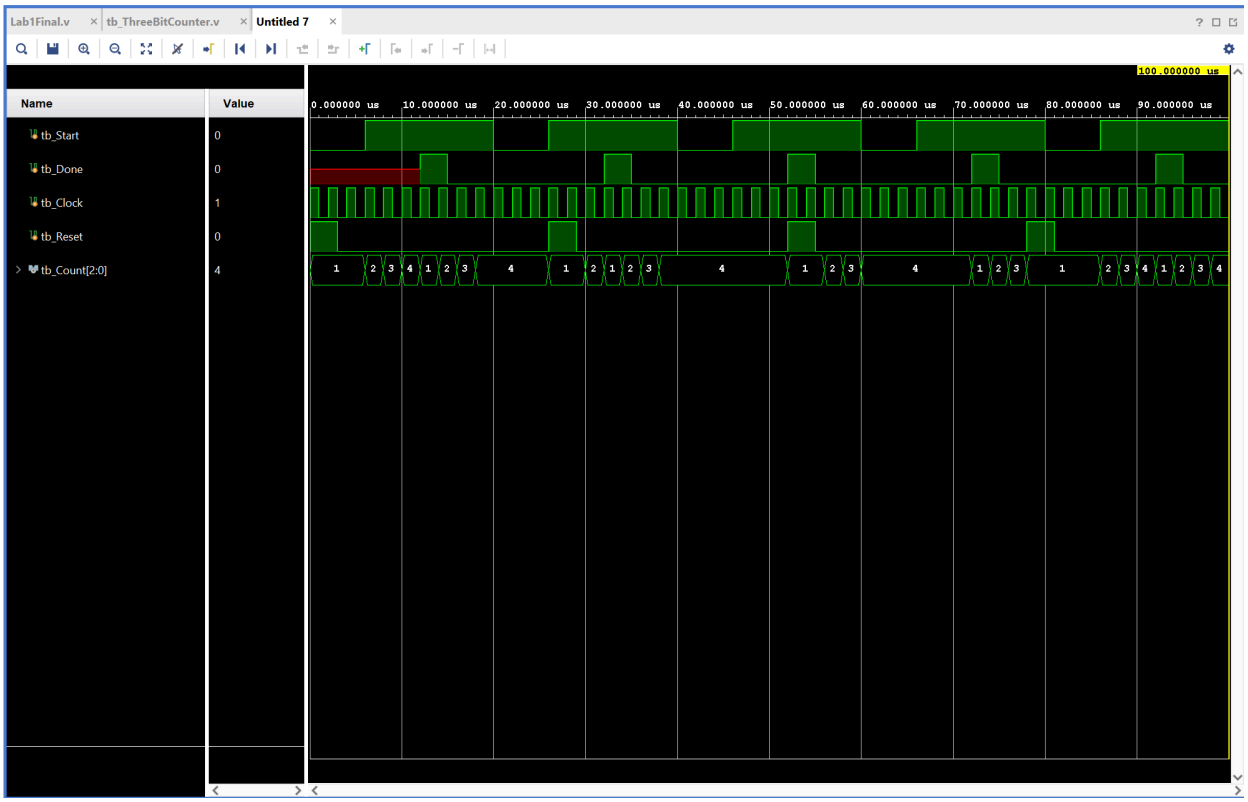


Part 1: Final Code

```
module ThreeBitCounter (Direction, Clock, Reset, Count);

        input Direction, Clock, Reset;
        output reg [2:0] Count;
        reg[1:0] currState, nextState;

        always @(posedge Clock)
                if(Reset)currState = 0;
                else currState = nextState;

  always @(*)
        case (currState)
                0: begin
                        Count = 3'b001;
                                if (Direction==1) nextState = 1;
                                else nextState = 3;
                        end
                1: begin
                        Count = 3'b010;
```

```
                    if (Direction==1) nextState = 2;
                    else nextState = 0;
                end
            2: begin
                Count = 3'b011;
                    if (Direction==1) nextState = 3;
                    else nextState = 1;
                end
            3: begin
                Count = 3'b100;
                    if (Direction==1) nextState = 0;
                    else nextState = 2;
                end
        endcase
endmodule
```

Part 2: Simulation



Part 2: Test Bench

```
`timescale 100 ns / 1 ps
module tb_ThreeBitCounter ();
```

```
    reg tb_Start, tb_Done, tb_Clock, tb_Reset;
    wire [2:0] tb_Count;

    ThreeBitCounter UUT (.Start(tb_Start), .Done(tb_Done), .Clock(tb_Clock), .Reset(tb_Reset),
.Count(tb_Count));

    /* free running clock */
    always begin
        tb_Clock = 1; #10;
        tb_Clock = 0; #10;
    end

    /* direction */
    always begin
        tb_Start = 0; #60;
        tb_Start = 1; #60;
        tb_Done = 1; #30;
        tb_Done = 0; #50;
    end

    /* reset */
    always begin
        tb_Reset = 1; #30;
        tb_Reset = 0; #230;
    end
endmodule
```

Part 2: Final Code

```
module ThreeBitCounter (Start, Done, Clock, Reset, Count);

        input Start, Done, Clock, Reset;
        output reg [2:0] Count;
        reg[1:0] currState, nextState;

        always @(posedge Clock)
                if(Reset)currState = 0;
                else currState = nextState;

    always @(*)
```

```
        case (currState)
                0: begin
                        Count = 3'b001;
                                if (Start==1)
                                        nextState = 1;
                           else
                              nextState = 0;
                        end
                1: begin
                        Count = 3'b010;
                                if (Done==1)
                                        nextState = 0;
                                else
                                   nextState = 2;
                        end
                2: begin
                        Count = 3'b011;
                                if (Done==1)
                                        nextState = 0;
                                else
                                        nextState = 3;
                        end
                3: begin
                        Count = 3'b100;
                                if (Done==1)
                                        nextState = 0;
                                else
                                   nextState = 3;
                        end
        endcase
endmodule
```

REPORT

For your lab report, include the completed source code for your working state machine and the screenshot of simulation results. In addition, include answers to the following questions.

• What problems did you encounter while implementing and testing your system?

We encountered a few problems while implementing and testing our system. We struggled a lot with using the software, for instance, adding files and debugging took the majority of the time. Getting familiar with a new software and simulating while managing syntax errors that we were

not familiar with proved to be the most challenging aspect of this lab. We also did not consider changing aspects of the test bench which needed to be done in order to create a clean, even, simulation. Overall, the concepts of this lab were fairly easy, however, the software was difficult to use initially.

• Did any problems arise when demonstrating for the TA? In other words, did the TA ask you to demonstrate something that you did not think of yourself? What was the scenario that you were asked to demonstrate? Provide some thoughts about why you didn't think of this yourself.
We did not face these kinds of problems. However, we misinterpreted the instructions and had to reevaluate our approach in order to meet every bullet point in our instructions.

• In the counter design that we've done for this lab, we frequently observe the counter values wrapping around from the maximum value to minimum (or vice versa), e.g., 4 →1 or 1→4. If you want to design a Verilog module that can detect these, how would you approach this? In your answer, be specific whether you would describe the module in a Moore machine or a Mealy machine.
We could create a variable to detect if a wrap around occurs. It's set to 1 (true) when a wrap-around is detected and reset to 0 (false) in all other cases. This would be a Mealy machine since the variable output is dependent on the path in the state diagram, rather than just the state.