**ECE 211: Digital Circuits I — Spring 2025**

**Lab 9: Sequential LED Tail Light**

By: *Efe CIVISOKEN*

23/04/2025

**Statement of Collaboration:** All the work is done by Efe, including both the lab demo and report.

**Time Spent**: 1 hour.

**INTRODUCTION**

This lab centered on implementing a finite state machine (FSM) to control a sequential LED tail light system on the Nexys A7 FPGA. The pattern was inspired by the iconic tail light animations of the Ford Mustang, where LEDs activate sequentially from the inner to the outer light to indicate a turn. Our task was to implement this design using SystemVerilog, simulate and validate the circuit behavior with testbenches, synthesize it, and deploy it on the FPGA. Additionally, an optional custom light pattern was implemented and integrated into the top-level design to explore creative FSM design extensions. All behavior was controlled interactively using on-board switches.

**DESIGN**

The design of the submodules required constructing two finite state machines. *(Appendix 1)* The first was the standard taillight module, which cycled through four distinct states: 3'b000 → 3'b100 → 3'b110 → 3'b111. Each of these 3-bit outputs controls three LEDs representing the inner, middle, and outer segments. This pattern simulates a sweeping light animation outward from the vehicle’s center. The state machine used a synchronous always\_ff block to update the state on the rising edge of the clock and a purely combinational always\_comb block to define the next state. If the reset signal was high, the system returned to the initial 000 state.

The second FSM, defined in the customtaillight module, alternates between the states 3'b101 and 3'b111, after starting from the reset state 000. This pattern lights up the outer and inner LEDs simultaneously in one cycle and then lights all three in the next cycle, creating a more symmetric flashing appearance. This FSM is useful as a test of system flexibility and demonstrates how state transitions can be configured to support non-traditional behavior. Both modules share identical internal structure in terms of signal flow and FSM logic style but differ in their combinational transition logic.

More details on the design of the submodules can be found in the following code blocks Code 1 and Code 2 as well as the detail schematic view of the submodules Schema 1.

module taillight( input logic rst, clk,

output logic [2:0] IMO\_current);

logic [2:0] IMO\_next;

// Update state on rising clock edge

always\_ff @(posedge clk)

begin

if(rst)

IMO\_current <= 3'b000;

else

IMO\_current <= IMO\_next;

end

// Compute next\_state with combinational logic

always\_comb

begin

if(IMO\_current == 3'b000)

IMO\_next = 3'b100;

else if(IMO\_current == 3'b100)

IMO\_next = 3'b110;

else if(IMO\_current == 3'b110)

IMO\_next = 3'b111;

else if(IMO\_current == 3'b111)

IMO\_next = 3'b000;

else

// like a default case, should not happen but if it does, it is like a safety.

IMO\_next = 3'b000;

end

endmodule

**Code 1:** taillight.sv module.

module customtaillight( input logic rst, clk,

output logic [2:0] IMO\_current);

logic [2:0] IMO\_next;

// Update state on rising clock edge

always\_ff @(posedge clk)

begin

if(rst)

IMO\_current <= 3'b000;

else

IMO\_current <= IMO\_next;

end

// Compute next\_state with combinational logic

always\_comb

begin

if(IMO\_current == 3'b000)

IMO\_next = 3'b101;

else if(IMO\_current == 3'b101)

IMO\_next = 3'b111;

else if(IMO\_current == 3'b111)

IMO\_next = 3'b101;

else

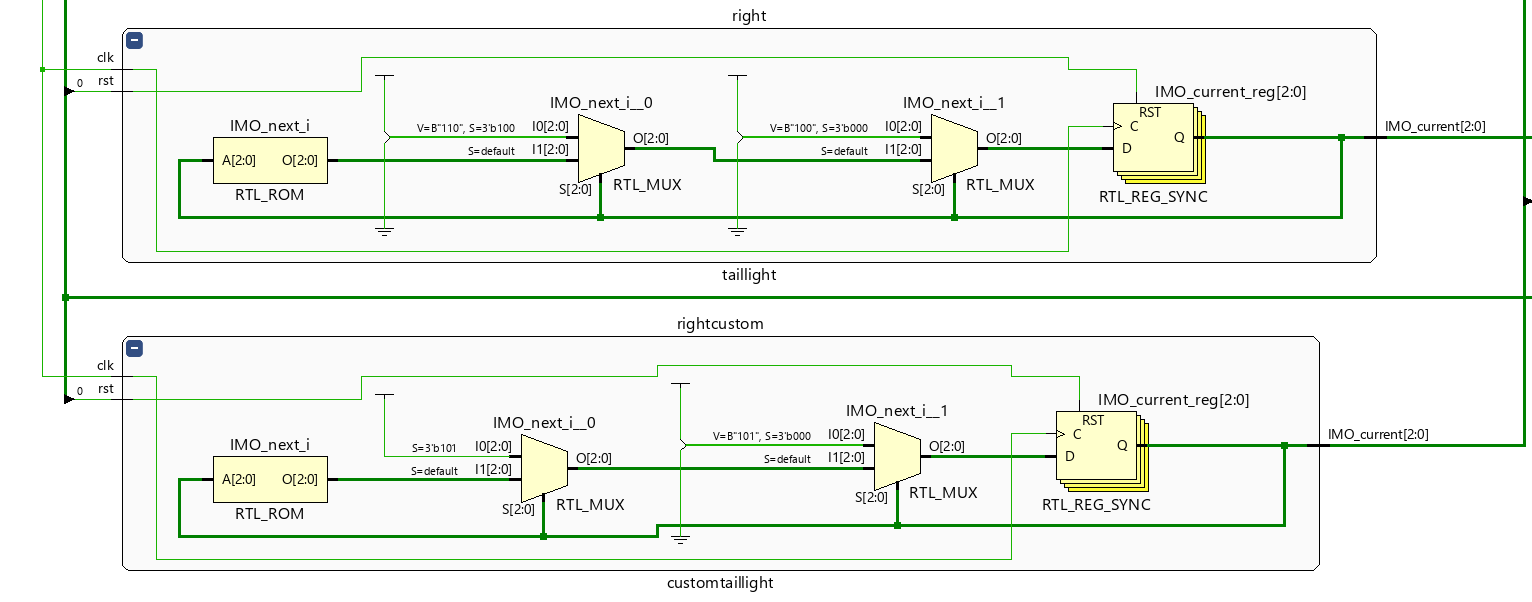
// like a default case, should not happen but if it does, it is like a safety.

IMO\_next = 3'b000;

end

endmodule

**Code 2**: customtaillight.sv module.



**Schema 1** : Detail schematic view of the submodules.

**IMPLEMENTATION**

In the top-level module lab09\_top, both standard and custom FSMs were instantiated. The module takes a 100 MHz input clock (clk100MHz) and reduces its frequency using a clkdiv module to make the LED transitions observable. The clock divider was configured with a frequency division factor such that the final clock was approximately 2 Hz, suitable for human-visible LED flashing.

Four FSM instances were created: taillight and customtaillight for the left and right sides. Each of these modules receives the downscaled clock and a separate reset signal via switches SW[0] and SW[1], controlling the right and left indicators respectively. Switch SW[2] functions as a selector to choose between standard and custom modes.

The outputs of the FSMs were multiplexed based on the state of SW[2]. This means that when SW[2] is off, the LED outputs reflect the standard FSMs. When SW[2] is on, the custom FSM outputs are routed to the same LEDs instead. The mappings to the LED array were as follows: for the right-side signals, FSM outputs were connected to LED[2:0], and for the left-side, to LED[15:13], matching the board’s physical layout.

This hierarchical and modular structure allows for a clean and flexible implementation, where each tail light controller is separated into its own FSM and the top module handles only integration and control.

module lab09\_top(

input logic clk100MHz,

input logic [15:0] SW, // SW1 left SW0 right

output logic [15:0] LED

);

logic clk;

logic [2:0] IMO\_current\_right;

logic [2:0] IMO\_current\_left;

logic [2:0] IMO\_current\_right\_custom;

logic [2:0] IMO\_current\_left\_custom;

// Implement your lab here

clkdiv #(.DIVFREQ(2)) D0(.clk(clk100MHz), .reset(1'b0), .sclk(clk));

taillight right(.rst(SW[0]), .clk(clk), .IMO\_current(IMO\_current\_right));

taillight left(.rst(SW[1]), .clk(clk), .IMO\_current(IMO\_current\_left));

customtaillight rightcustom(.rst(SW[0]), .clk(clk), .IMO\_current(IMO\_current\_right\_custom));

customtaillight leftcustom(.rst(SW[1]), .clk(clk), .IMO\_current(IMO\_current\_left\_custom));

assign LED[15] = SW[2] ? IMO\_current\_left\_custom[2] : IMO\_current\_left[2];

assign LED[14] = SW[2] ? IMO\_current\_left\_custom[1] : IMO\_current\_left[1];

assign LED[13] = SW[2] ? IMO\_current\_left\_custom[0] : IMO\_current\_left[0];

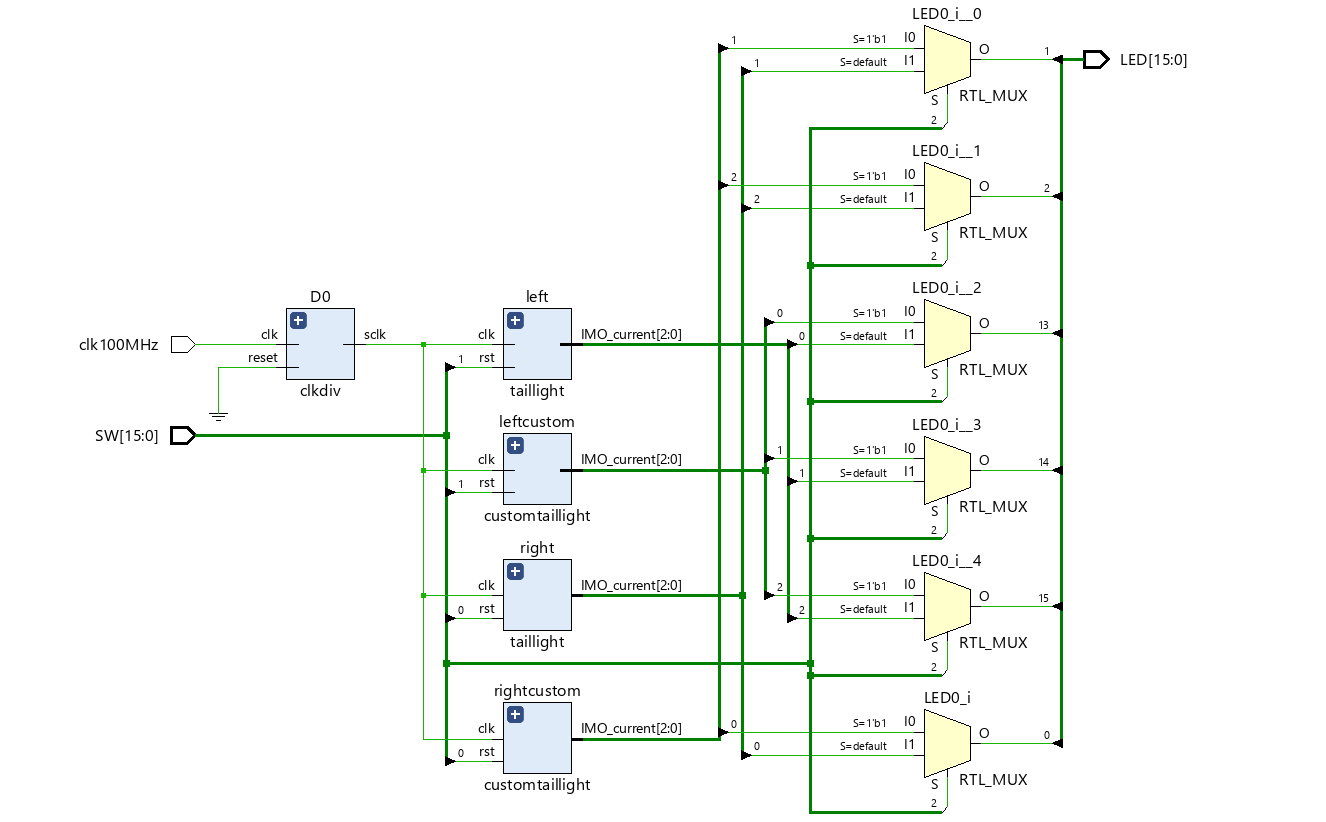
assign LED[2] = SW[2] ? IMO\_current\_right\_custom[2] :IMO\_current\_right[2];

assign LED[1] = SW[2] ? IMO\_current\_right\_custom[1] :IMO\_current\_right[1];

assign LED[0] = SW[2] ? IMO\_current\_right\_custom[0] :IMO\_current\_right[0];

endmodule

**Code 3:** The top module, lab09\_top.sv module.



**Schema 2:** General implementation-wise schematic of the lab.

**TESTING (Submodular and Behavioral)**

The initial testing phase focused on verifying the behavior of the standard taillight FSM module using simulation. A SystemVerilog testbench was constructed to instantiate the taillight module, and a toggling clock was generated using a forever loop with 1-time unit delays between transitions. The rst signal was pulsed at controlled times to observe whether the FSM would return to the 3'b000 state properly. The waveform from the simulation confirmed that the FSM followed the expected sequence: starting from 000, moving to 100, then to 110, then to 111, and looping back to 000. Each transition occurred exactly one clock cycle after the previous, validating the synchronous state progression. The testbench also included multiple reset sequences at varying states to confirm the system’s robustness. The simulation environment helped ensure that the FSM never entered an undefined state and that the next-state logic performed exactly as designed.

After confirming the correctness of the standard taillight module and implementing in the top module, the top-level design was synthesized and loaded onto the Nexys A7 FPGA. Behavioral testing was then conducted by toggling the switches to activate left and right tail lights and observing the physical LED outputs. The LEDs transitioned from inner to outer segments with the desired delay, replicating the expected behavior seen in simulation. Toggling SW[0] and SW[1] allowed us to activate either the right or left side independently, confirming that the logic in lab09\_top was correctly routing inputs to each FSM.

Once the standard functionality was fully verified in both simulation and on the board, we extended the lab by designing and implementing a custom FSM in the customtaillight module. This module followed a different state sequence—alternating between 3'b101 and 3'b111—to produce a visually symmetric pattern. The design and logic followed the same structural format as the original FSM but used alternate state transitions in its always\_comb block.

After writing and verifying the new FSM logic, we added customtaillight instances for both left and right signals in the top-level module. A selector switch (SW[2]) was introduced to toggle between standard and custom patterns using conditional assignments. We then repeated the simulation process with the customtaillight FSM, ensuring that it correctly cycled between the two visual states. Upon successful simulation, the updated design was synthesized and deployed to the board again.

In the final hardware test, we confirmed that toggling SW[2] activated the custom sequence. LEDs correctly alternated between lighting the inner and outer segments (101) and all three segments (111). All state transitions were clean, synchronized to the clock, and reset properly. The integration of this optional component allowed for dynamic switching between designs and demonstrated that both FSMs could coexist and operate correctly within the same system architecture.

This staged approach to testing—starting from a minimal implementation and progressively adding complexity—allowed us to confidently verify the correctness of each component before advancing, leading to a fully functional and flexible LED control system.

module lab09\_tb();

logic rst, clk;

logic [2:0] IMO\_current;

// Instantiate the devices under test (DUTs)

taillight DUT(.rst(rst), .clk(clk), .IMO\_current(IMO\_current));

always begin

assign clk = 0; # 1;

assign clk = 1; # 1;

end

initial begin

$display("[ECE 211] Simulation starting...");

// Add your test cases here

assign rst = 1; #5;

assign rst = 0; #8;

assign rst = 1; #2;

assign rst = 0; #14;

assign rst = 1; #2;

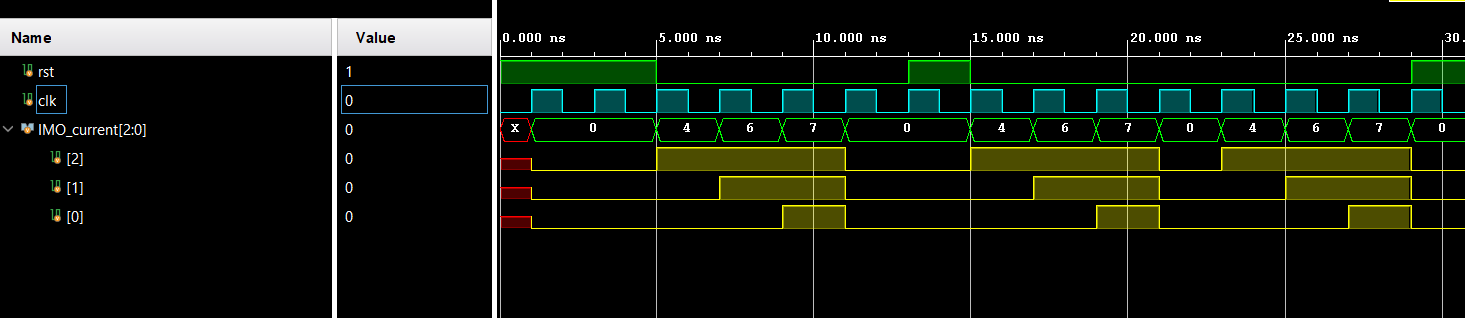
$display("[ECE 211] Simulation complete...");

$stop;

end

endmodule

**Code 4**: lab09\_tb.sv module, the top testbench module code resulting in *Waveform 1.*



**Waveform 1:** Waveform simulation of taillight.sv module, confirming the intended behavior.

**RESULTS, CONCLUSION, AND DISCUSSION**

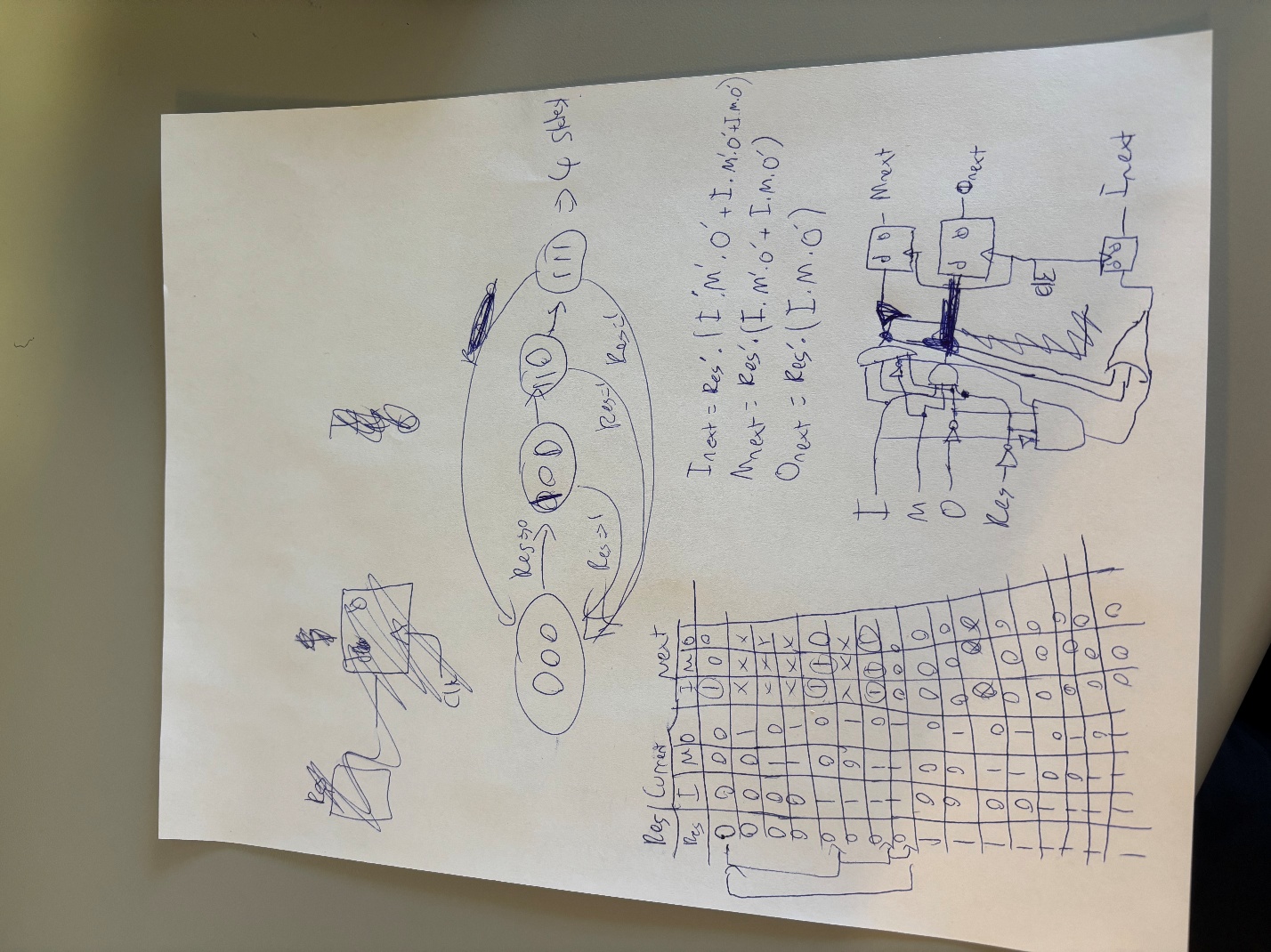
The project was successfully completed with full functional and visual verification through both simulation and hardware testing. The FSMs behaved as intended, with all transitions occurring properly/timely, and reset behavior working cleanly. The modular design made it easy to manage complexity and isolate behavior for both standard and custom light patterns.

One small issue occurred during the optional top module implementation part of the customtaillight because of a very minor careless mistake which involved naming the instance intended to be customtaillight as taillight, which was then swiftly solved after double-checking the code right before the synthesis part of the top module.

This lab, overall, helped us understand how FSMs are applied in real-world scenarios like automotive lighting. It also demonstrated the importance of synchronizing all operations with the clock and the value of modular SystemVerilog design. The custom FSM served as a creative exercise and gave us confidence in modifying FSM logic to achieve alternate behaviors. The testing process, especially simulation before hardware deployment, was critical in catching and resolving logic issues early.

Overall, this lab reinforced best practices in digital circuit design, FSM construction, clock domain control, and hardware integration using the Vivado design suite and FPGA hardware. The ability to visually debug and compare simulation to physical hardware behavior greatly enhanced our understanding of synchronous systems.

**APPENDIX:**



**Appendix 1:** The state transition diagram, truth table, and sketch circuit design of our intended sequential LED taillight design. It was especially important to see the big picture of our design before starting to code.