ECE3300L Lab 4

Group B

By Faris Khan (ID #: 012621102)

AND

Nicholas Williams (ID#:016556982)

11 July 2025

Introduction

The purpose of this lab was to implement a switch-to-7-segment display using structural verilog on the Nexys FPGA board. The system reads input from 16 switches, converts the binary values into hexadecimal digits, and displays them across all 8 digits of the 7-segment display using multiplexing. The design was broken down into a 4-bit-to-7-segment decoder, an anode controller for digit selection, and a multiplexer for selecting input values. Each module was instantiated in a top-level driver module (seg7_driver) that handled clock-based multiplexing. This approach not only allowed full utilization of the display despite limited input bits but also demonstrated modular design and decoding logic.

Code

seg7_driver.v

```
module seg7_driver(
    input clk,
    input rst,
    input [15:0] SW,
    output [6:0] Cnode,
    output dp,
    output [7:0] AN,
    output [15:0] LED
);

    assign dp = 1'b1;
    assign LED = SW;

    reg [19:0] tmp;
    wire [3:0] digit_wire;

    always @(posedge clk or posedge rst)
        if (rst) tmp <= 0;
        else tmp <= tmp + 1;

    wire [2:0] s = tmp[19:17];


    mux8 mux_inst (
        .SW(SW),
        .s(s),
        .digit(digit_wire)
    );


    seg7_decoder seg_inst (
        .digit(digit_wire),
        .Cnode(Cnode)
    );


    anode_decoder an_inst (
        .s(s),
        .AN_tmp(AN)
    );

endmodule
```

The seg7_driver module controls the 8-digit 7-segment display on the Nexys A7 using structural Verilog. It reads input from 16 switches (SW[15:0]) and displays the

resulting hexadecimal values by multiplexing through all 8 digits. A 20-bit counter (tmp) cycles through each digit using a 3-bit selector (s), which is passed to a multiplexer (mux8) to select the appropriate 4-bit input, a segment decoder (seg7_decoder) to generate the 7-segment pattern, and an anode decoder (anode_decoder) to activate the correct digit. Although only 16 switches are available, the design is repeated in order to accommodate all 8 digits on the display. The LEDs mirror the switch states, and the decimal point is disabled.

seg7_decoder.v

```
module seg7_decoder (
    input [3:0] digit,
    output reg [6:0] Cnode
);
    always @(digit)
        case (digit)
            4'd0: Cnode=7'b0000001; 4'd1: Cnode=7'b1001111; 4'd2: Cnode=7'b0010010;
            4'd3: Cnode=7'b0000110; 4'd4: Cnode=7'b1001100; 4'd5: Cnode=7'b0100100;
            4'd6: Cnode=7'b0100000; 4'd7: Cnode=7'b0001111; 4'd8: Cnode=7'b0000000;
            4'd9: Cnode=7'b0001100; 4'd10:Cnode=7'b0001000;4'd11:Cnode=7'b1100000;
            4'd12:Cnode=7'b0110001;4'd13:Cnode=7'b1000010;4'd14:Cnode=7'b0110000;
            4'd15:Cnode=7'b0111000;default: Cnode=7'b1111111;
        endcase
endmodule
```

The seg7_decoder module converts a 4-bit hexadecimal digit into the corresponding 7-segment display pattern. It takes a 4-bit input (digit) and outputs a 7-bit value (Cnode) that controls which segments (a–g) of the display are turned on. The encoding is written in active-low format, meaning a 1 turns a segment off. Each case in the always block maps a hex digit (4'd0 to 4'd15) to the correct segment combination to represent values 0–F. The default case sets all segments off (7'b1111111) for error handling.

anode_decoder.v

```
module anode_decoder(
    input [2:0] s,
    output reg [7:0] AN_tmp
    );

    always @(s)
        case(s)
            3'd0:AN_tmp=8'b11111110;3'd1:AN_tmp=8'b11111101;
            3'd2:AN_tmp=8'b11111011;3'd3:AN_tmp=8'b11110111;
            3'd4:AN_tmp=8'b11101111;3'd5:AN_tmp=8'b11011111;
            3'd6:AN_tmp=8'b10111111;3'd7:AN_tmp=8'b01111111;
            default:AN_tmp=8'b11111111;
        endcase
endmodule
```

The anode_decoder module is used to control which of the 8 digits on the 7-segment display is active. It takes a 3-bit selector signal (s) and outputs an 8-bit value (AN_tmp) driven by active-low encoding.

mux8.v

```
module mux8 (
    input [15:0] SW,
    input [2:0] s,
    output reg [3:0] digit
);
    always @(*) begin
        case(s)
            3'd0: digit = SW[3:0];
            3'd1: digit = SW[7:4];
            3'd2: digit = SW[11:8];
            3'd3: digit = SW[15:12];
            3'd4: digit = SW[3:0];
            3'd5: digit = SW[7:4];
            3'd6: digit = SW[11:8];
            3'd7: digit = SW[15:12];
        endcase
    end
endmodule
```

The mux8 module is a 3-bit controlled multiplexer that selects a 4-bit segment of the 16 switch input based on the selector (s), and outputs the corresponding digit of the display. Since the Nexys A7-100T only has 16 switches which is enough for 4 hex

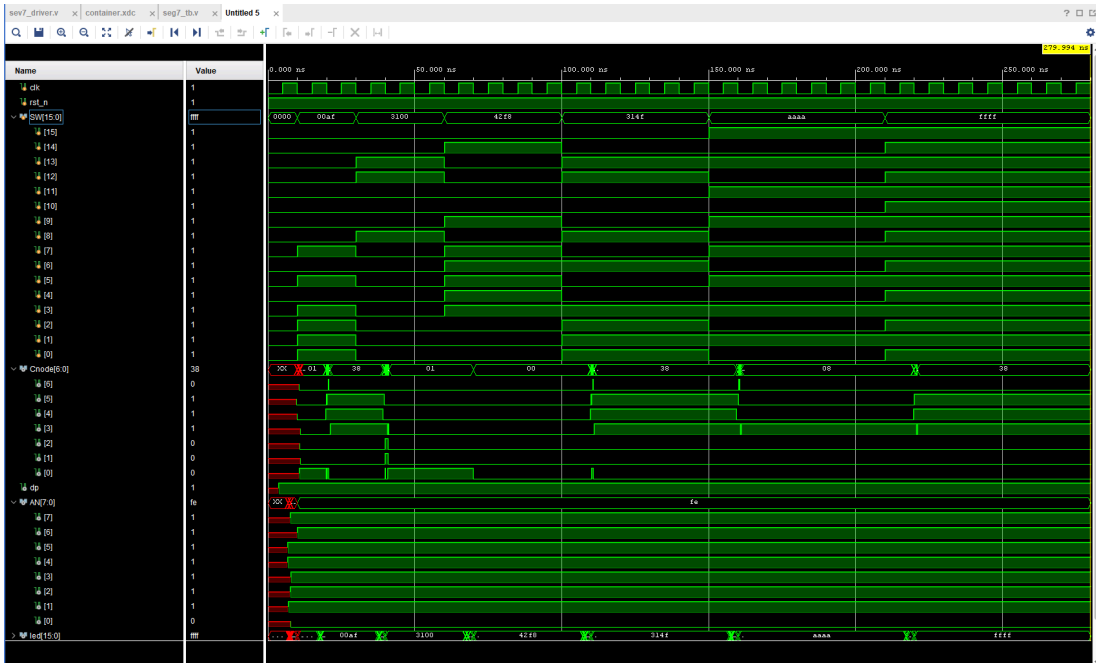
values, the switches are reused for the remaining digits. This allows us to utilize the entire display.

Seg7_tb();

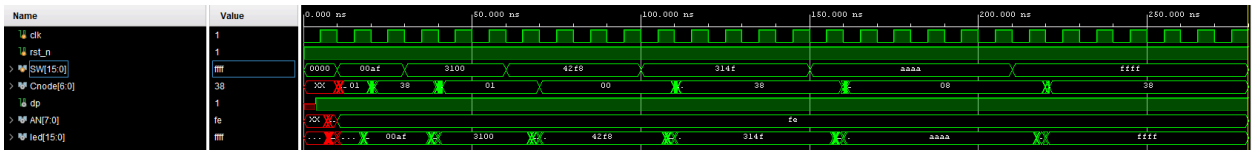
```
module seg7_tb();

    reg clk = 0;
    reg rst_n = 0;
    reg [15:0] SW;
    wire [6:0] Cnode;
    wire dp;
    wire [7:0] AN;
    wire [15:0] led;
    seg7_driver TB_testing (
        .clk(clk),
        .SW(SW),
        .rst_n(rst_n),
        .Cnode(Cnode),
        .dp(dp),
        .AN(AN),
        .led(led)
    );
    always #5 clk = ~clk;
    initial begin
        clk = 0;
        rst_n = 1;
        SW = 16'h0000;
        #10 rst_n = 1;
        SW = 16'h00AF;
        #20;
        SW = 16'h3100;
        #30;
        SW = 16'h42F8;
        #40;
        SW = 16'h314F;
        #50;
        SW = 16'hAAAA;
        #60;
        SW = 16'hFFFF;
        #70;
        $stop;
    end

endmodule
```

Data analysis

When running a simulation to see what was to happen we gathered these results below.

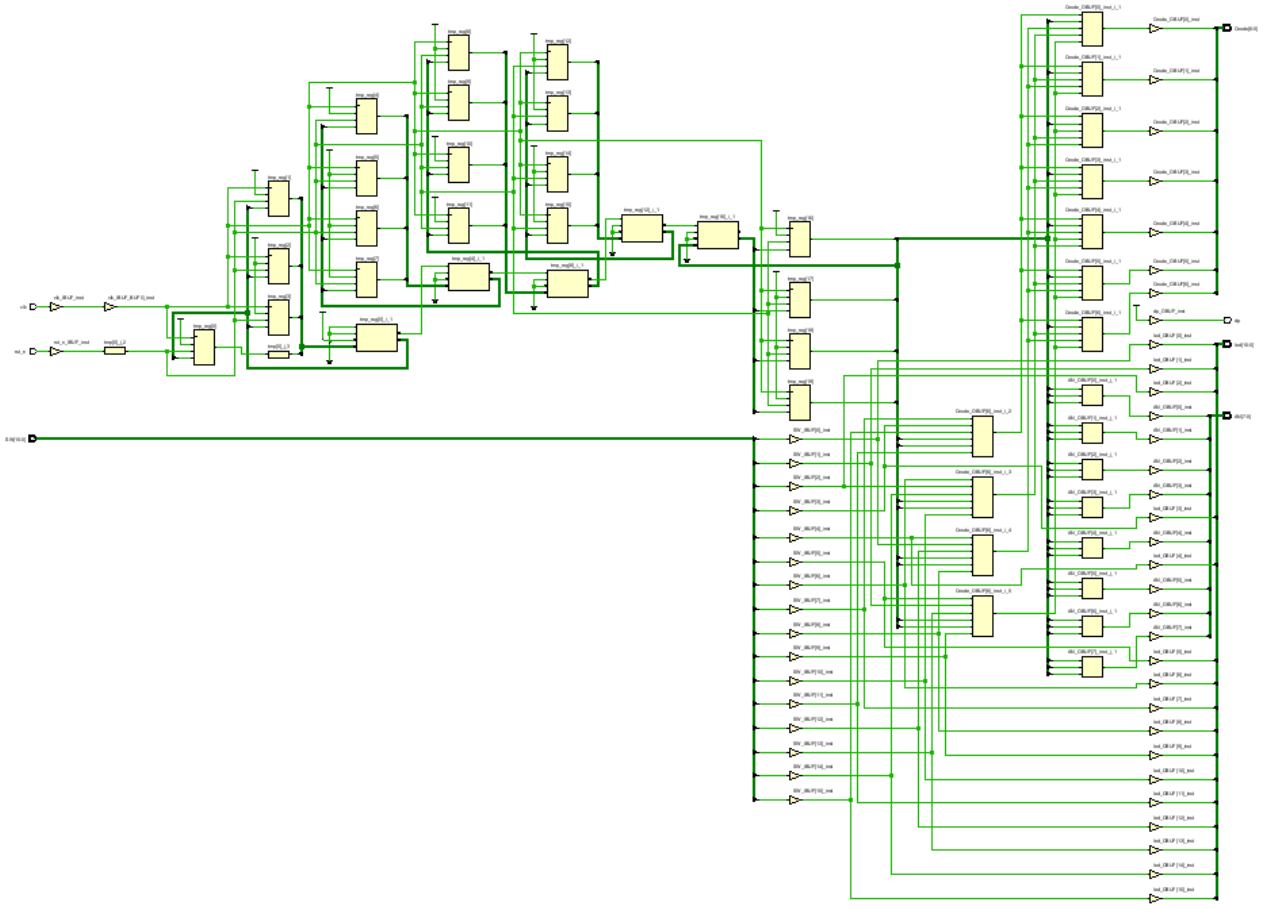


This is inline with what we would want to see based on the testbench we made that is posted in the code section. In the simulation the resulting output given by the seven segment display is inline with what we want to receive based on the switch configuration that was made.



Screenshots

Schematics



LUT & FF

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF
synth_1	constrs_1	synth_design Complete!												13	20
impl_1	constrs_1	route_design Complete!	7.437	0.000	0.324	0.000		0.000	0.125	0	16 Warn			13	20

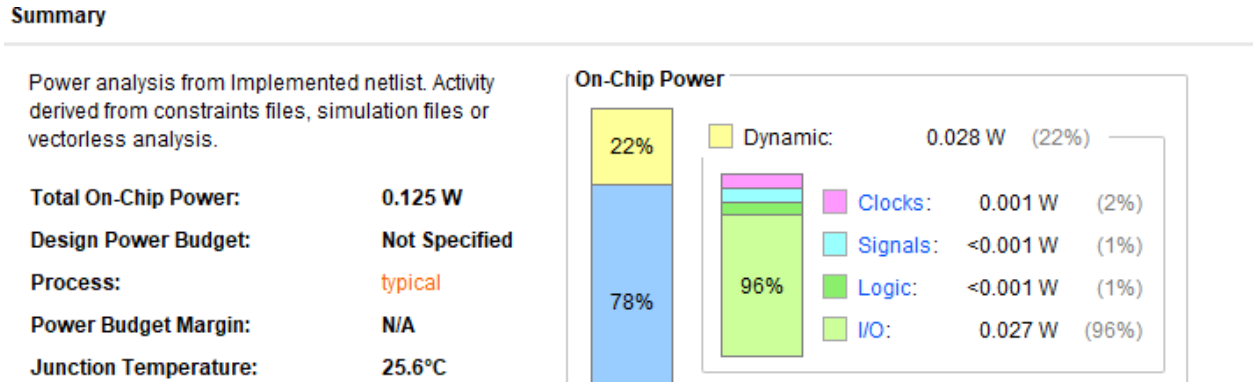
Timing Summary

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.437 ns	Worst Hold Slack (WHS):	0.324 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	20	Total Number of Endpoints:	20	Total Number of Endpoints:	21

All user specified timing constraints are met.

Power Summary



Conclusion

In this lab we were able to synthesis, implement, and simulate with Vivado and our nexus FPGA board a system that read input from 16 switches that then converted the values into hexadecimal digits of 0 to F and then displays them on all 8 digits of the 7-segment display. In this lab we were able to utilize Verilog within Vivado to drive a testbench that gave us values that we expected to based on the inputting switches

Contributions

Faris (50%) - Demo, introduction of lab report, and screenshots of code with explanation and summary screenshots

Nicholas (50%) - Testbench, Helped with Screenshots. Data analysis of the lab report with screenshots and summarizing the testbench.