**ECE211-L LAB 4**

**Lab 4 Report: Code Conversion – POSTNET to Binary**

*Efe CIVISOKEN*

*Fahendrena RAMIANDRASOA*

*03/05/2025*

*Statement of Collaboration:* Each team member contributed to the design, implementation, and testing of the code conversion module. Efe and Fahendrena both worked on SystemVerilog implementation, creation of the Boolean logic statements, and their appropriate Karnaugh map representation. While Efe was more involved in the creation of these, Fahendrena was more involved in making sure there aren’t any minor careless mistakes involved in each step. Efe wrote the whole report.

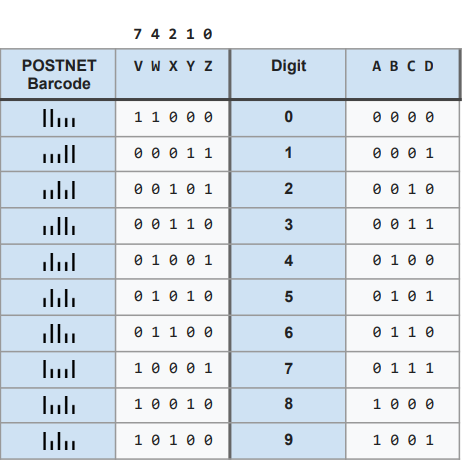
**Time Spent:** 1.5 hours.

**INTRODUCTION**

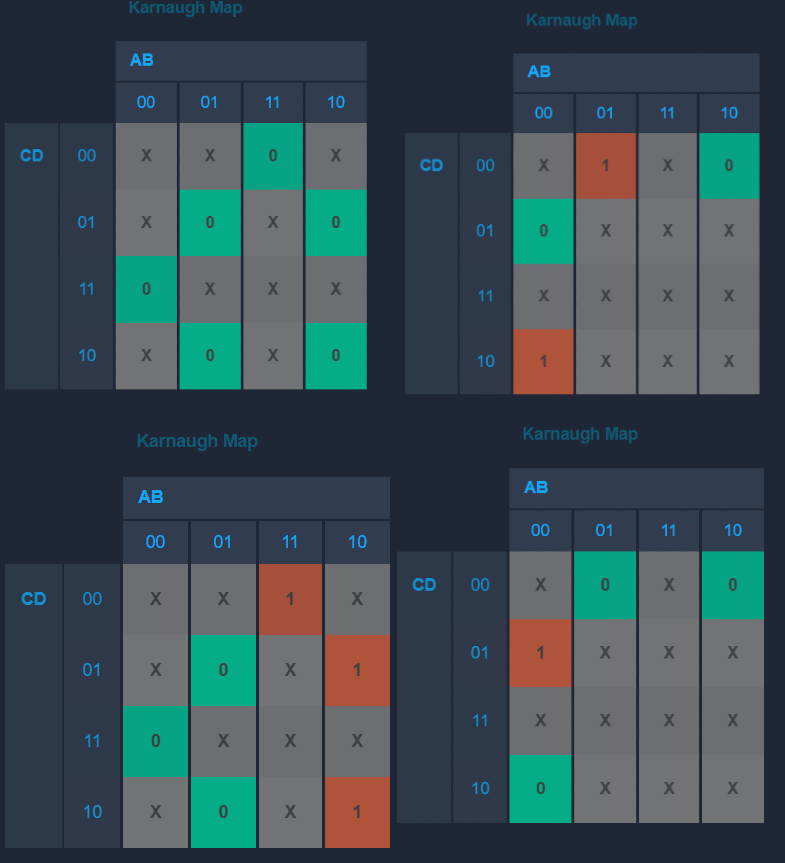
This lab focuses on designing a digital circuit that converts the POSTNET 2-out-of-5, allowing exactly ten different combinations, code into binary values using Karnaugh maps for simplification of the Boolean functions for each value in the display. The final implementation was tested on the Nexys A7 FPGA board using a seven-segment display. Through this lab, we applied Boolean algebra, Karnaugh maps, gate-level modeling, and FPGA implementation techniques.

**DESIGN**

To design the POSTNET to binary converter, we first analyzed the structure of the POSTNET encoding system. POSTNET encodes decimal values (0-9) using a 2-out-of-5 format, meaning each five-bit sequence contains exactly two ones. Since there are five input variables (V, W, X, Y, Z), a five-variable Karnaugh map (through eight 4-variable Karnaugh maps) was used to simplify the logic expressions.



**Table 1**. POSTNET Code Conversion (POSTNET to Binary)



WX

V: 1

V: 0

V: 1

V: 0

YZ

YZ

YZ

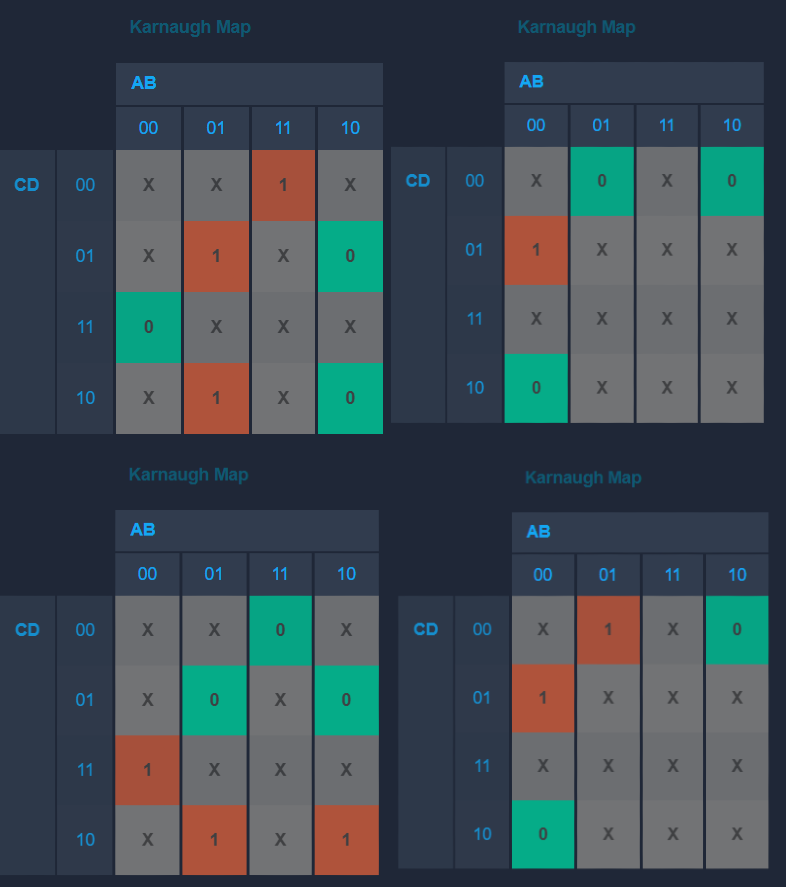
YZ

WX

WX

WX

**K-Map 1 & 2**: The Karnaugh Map Representations for the Boolean Outputs A and B in Table 1, respectively.



V: 1

V: 0

V: 0

V: 1

YZ

YZ

YZ

YZ

WX

WX

WX

WX

**K-Map 3 & 4**: The Karnaugh Map Representations for the Boolean Outputs A and B in Table 1, respectively.

Furthermore, in the creation of a five-variable Karnaugh map, we assigned 'don't care' values for invalid POSTNET combinations, and then minimized the Boolean equations for outputs A, B, C, and D using the most efficient implicants for the most optimal simplification. The final Boolean expressions derived from the K-map simplifications were implemented using basic logic gates.

Once the logic equations were finalized, a schematic diagram was created to represent the gate-level implementation. This schematic was used as a reference during the SystemVerilog implementation. The information concerning the final logic expressions in SystemVerilog and their final schematic representation can be seen in the following figures and pieces of code for your reference.

module postnet\_to\_binary(

input logic v,w,x,y,z,

output logic a,b,c,d);

// The Boolean expressions for the output functions A, B, C, and D, respectively. // using inputs V, W, X, Y, Z.

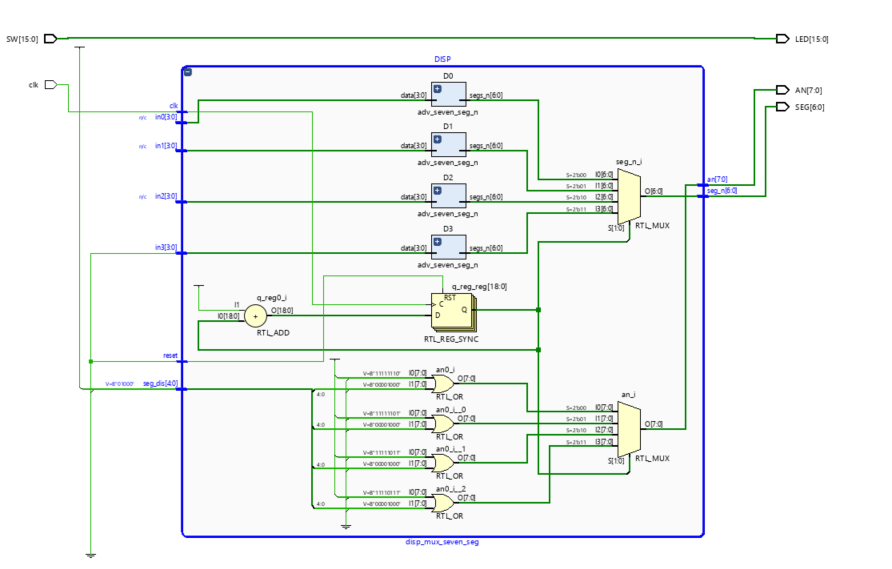
or(a, v&~w&~x&y&~z, v&~y&~z&~w&x);

or(b, ~v&w&x&~y&~z, ~v&w&~x&~y&z, ~v&w&~x&y&~z, v&~w&~x&~y&z);

or(c, ~v&x,v&~w&~y&z);

or(d, ~v&y,v&~w&~y );

endmodule



**Schema 1**. Schematic Representation of the Final Implementation

**IMPLEMENTATION:**

The logic circuit was implemented as a SystemVerilog module named postnet\_to\_binary as can be seen in the DESIGN part. The module takes a five-bit POSTNET input and produces a four-bit binary output. The equations derived from the K-map were directly implemented using combinational logic as mentioned beforehand.

The design was integrated with the top-level ‘lab04\_top.sv’ SystemVerilog module to interface with the Nexys A7 FPGA board. Three instances of the conversion module were created to display multiple POSTNET values on the seven-segment display. The FPGA switches SW[4]-SW[0], SW[9]-SW[5], and SW[14]-SW[10] were used to input three different two-out-of-five POSTNET codes, with the corresponding binary outputs displayed on the seven-segment display thanks to previously given modules called disp\_mux\_seven\_seg and adv\_seven\_seg\_n (APPENDIX), which basically took vector representation of our vector outputs as inputs and displayed them on the seven-segment display. The further details on the top-module code can be seen below:

module lab04\_top(

input logic clk,

input logic [15:0] SW,

output logic [15:0] LED,

output logic [6:0] SEG,

output logic [7:0] AN );

// Configure connections between switches and LEDs

assign LED[15:0] = SW[15:0];

// Instantiate your postnet\_to\_binary modules here

logic w1,w2,w3,w4;

logic w5, w6, w7, w8;

logic w9, w10,w11,w12;

postnet\_to\_binary ptb( .z(SW[0]), .y(SW[1]),.x(SW[2]), .w(SW[3]),.v(SW[4]), .a(w1) ,.b(w2) , .c(w3), .d(w4) );

postnet\_to\_binary ptb1( .z(SW[5]), .y(SW[6]),.x(SW[7]), .w(SW[8]),.v(SW[9]), .a(w5) ,.b(w6) , .c(w7), .d(w8) );

postnet\_to\_binary ptb2( .z(SW[10]), .y(SW[11]),.x(SW[12]), .w(SW[13]),.v(SW[14]), .a(w9) ,.b(w10) , .c(w11), .d(w12) );

//ptb ptb\_ins( .z(SW[0]), .y(SW[1]),.x(SW[2]), .w(SW[3]),.v(SW[4]), .a(output\_wire\_a), .b(output\_wire\_b) .c(output\_wire\_c) .d(output\_wire\_d));

// Vectorization

logic [3:0] B0 = {w1,w2,w3,w4};

logic [3:0] B1 = {w5,w6,w7,w8};

logic [3:0] B2 = {w9,w10,w11,w12};

// Display binary output on seven-segment display

disp\_mux\_seven\_seg DISP(.clk(clk), .reset(1'b0),

.in0(B0),

.in1(B1),

.in2(B2),

.in3(4'b0000), .seg\_dis(4'b1000), .seg\_n(SEG), .an(AN));

endmodule

**TESTING & RESULTS:**

To ensure the correctness of our Boolean expressions, we first verified them using Karnaugh maps. By systematically mapping and simplifying the expressions, we confirmed that they accurately represented the intended POSTNET to binary conversion.

Once the Boolean expressions in postnet\_to\_binary module were validated, we proceeded with the SystemVerilog implementation. The correctness of the final code in top and postnet\_to\_binary modules was further verified and then was programmed onto the Nexys A7 FPGA board, where we tested all possible 2-out-of-5 POSTNET code combinations. Each input configuration was carefully checked against its corresponding output on the seven-segment display using the information given in Table 1 (DESIGN part).

Finally, we cross-verified our results with expected outputs, ensuring that the implementation functioned correctly for all cases. Our testing was further confirmed by Professor Biernacki, who validated that the circuit produced accurate results for all POSTNET representations.

**DISCUSSION & CONCLUSION:**

This lab successfully demonstrated the process of designing and implementing a POSTNET to binary code converter. Through the use of Karnaugh maps, we were able to minimize Boolean expressions and implement an optimized logic circuit in SystemVerilog. The integration with the FPGA allowed real-world testing and verification.

Challenges encountered included correctly identifying don’t-care values in the K-map and debugging syntax errors in SystemVerilog. Despite these minor problems, the project was completed successfully, and the design performed perfectly as expected. Future improvements could include incorporating error detection mechanisms to identify invalid, not conforming to 2-out-of-5 logic, POSTNET codes dynamically, which was also communicated to dear Professor Biernacki during the lab.

Overall, this lab provided valuable experience in digital circuit design, logic minimization, and FPGA implementation, as well as introducing us to new concepts such as logic vectorization and its practical usage in display systems.

APPENDIX:

* Lab04 Demo File on Moodle Page.
* OTHER SYSTEMVERILOG MODULES:

//////////////////////////////////////////////////////////////////////////////////

// Company: Lafayette College

// Create Date: 01/09/2024 10:11:26 AM

// Design Name:

// Module Name: disp\_mux\_seven\_seg

// Project Name: ECE 211 Digital Circuits 1

//-----------------------------------------------------------------------------

// Author : Lauren Biernacki <biernacl@lafayette.edu>

// Created : Jan 2024

// Acknowledgment: Faraz Khan <https://simplefpga.blogspot.com/2012/07/seven-segment-led-multiplexing-circuit.html>, July 2012

//-----------------------------------------------------------------------------

// Description : Time-multiplexing circuit to display multi-digit seven-segment

// display with active-low outputs

// Takes four data values (in0-in3) and displays each in hexadecimal

// on the corresponding seven-segment LED

// The refresh rate should be near 1000 Hz to achieve the desired

// visual effect. The refresh rate is determined by the clock period

// and the parameter N (i.e., clk\_period/(2^N-2))

//-----------------------------------------------------------------------------

//////////////////////////////////////////////////////////////////////////////////

module disp\_mux\_seven\_seg(

input logic clk,

input logic reset,

input logic [3:0] in0, //data for seg LED 0

input logic [3:0] in1, // data for seg LED 1

input logic [3:0] in2, // data for seg LED 2

input logic [3:0] in3, // data for seg LED 3

input logic [4:0] seg\_dis, //4-bit value to disable each seg LED(active-low)

output logic [6:0] seg\_n, //connect to board seg pin

output logic [7:0] an //connect to board en pin

);

localparam N = 19; // Refresh rate of ~800Hz for a 100MHz clock

// Update counter on clock tick (for time multiplexing)

logic [N-1:0] q\_reg;

always\_ff @(posedge clk)

begin

if (reset) q\_reg <=0;

else q\_reg <= q\_reg + 1;

end

// Instantiate seven-segment decoder for each 4-bit input

logic [6:0] seg0;

logic [6:0] seg1;

logic [6:0] seg2;

logic [6:0] seg3;

adv\_seven\_seg\_n D0(.data(in0), .segs\_n(seg0));

adv\_seven\_seg\_n D1(.data(in1), .segs\_n(seg1));

adv\_seven\_seg\_n D2(.data(in2), .segs\_n(seg2));

adv\_seven\_seg\_n D3(.data(in3), .segs\_n(seg3));

// Multiplex which seg LED is displayed

always\_comb

begin

case(q\_reg[N-1:N-2])

2'b00: begin

seg\_n = seg0;

an = 8'b11111110 | seg\_dis;

end

2'b01: begin

seg\_n = seg1;

an = 8'b11111101 | seg\_dis;

end

2'b10: begin

seg\_n = seg2;

an = 8'b11111011 | seg\_dis;

end

2'b11: begin

seg\_n = seg3;

an = 8'b11110111 | seg\_dis;

end

endcase

end

endmodule: disp\_mux\_seven\_seg

//////////////////////////////////////////////////////////////////////////////////

// Company: Lafayette College

// Create Date: 01/09/2024 10:11:26 AM

// Design Name:

// Module Name: adv\_seven\_seg\_n

// Project Name: ECE 211 Digital Circuits 1

//-----------------------------------------------------------------------------

// Author : John Nestor <nestorj@lafayette.edu>

// Created : Feb 2020

// Revised : Lauren Biernacki <biernacl@lafayette.edu> Jan 2024

//-----------------------------------------------------------------------------

// Description : Advanced seven-segment decoder displaying

// hexidecimal values 0-F with active low outputs

// Segments are ordered as follows: segs\_n[6]=g, segs\_n[0]=a

//-----------------------------------------------------------------------------

//////////////////////////////////////////////////////////////////////////////////

module adv\_seven\_seg\_n(

input logic [3:0] data,

output logic [6:0] segs\_n // ordered g(6) - a(0)

);

always\_comb

case (data) // Seg: gfedcba

4'd0: segs\_n = 7'b1000000;

4'd1: segs\_n = 7'b1111001;

4'd2: segs\_n = 7'b0100100;

4'd3: segs\_n = 7'b0110000;

4'd4: segs\_n = 7'b0011001;

4'd5: segs\_n = 7'b0010010;

4'd6: segs\_n = 7'b0000010;

4'd7: segs\_n = 7'b1111000;

4'd8: segs\_n = 7'b0000000;

4'd9: segs\_n = 7'b0010000;

default: segs\_n = 7'b0111111;

endcase

endmodule: adv\_seven\_seg\_n