CRN: 22840

CET3136 – Logic Devices Programming

Spring 2025

Experiment 4

Numbers and Displays, Part 1

Performed By:

Anthony Paul Sevarino

Submitted to:

Prof. Ashley Evans

Department of Electrical & Computer Engineering Technology (ECET)

School of Engineering, Technology, and Advanced Manufacturing (ETAM)

Valencia College

Date Submitted

02/06/2025

Introduction

The intent of this lab is to create a two part project, wherein prior skills learned are used to create two combinatorial circuits within the Quartus environment. The first of which is a binary-to-decimal number conversion system, and the other is a binary-coded-decimal addition system. The FPGA DE1 development board will be used to conduct this experiment, using switches 7-0 and hex displays 0 and 1 for the first system, and switches 3-0 and hex displays 0 and 1 for the second system. While both systems will utilize a dataflow architecture coding style, it should be noted that there are fairly different approaches to the logic flow in each. The use of truth tables and Karnaugh maps will play a significant role in this experiment, particularly so in part B.

LIST OF EQUIPMENT/PARTS/COMPONENTS/SOFTWARE

- DE10-Lite MAX-10 Dev Board
- USB 2.0 Type B Cable
- Windows Desktop
- Quartus FPGA Design Software

PROCEDURE / DISCUSSION

Part A – Binary to Decimal Number Conversion

Assessing the hex display segment states aids heavily in writing the VHDL logic for the system, as well as understanding that these segments are powered on a *logic low* basis.

Figure 1 – 7-Segment Diagram

In a dataflow structure, a *select case* statement can set a value to a signal based on the current value of a specific input signal. For example The following program concatenates two inputs (a 3-bit vector called *sel* and a 1 bit input called *enable*) and designs a rotating 8-bit binary counter to be assigned to the development board's lights:

```
tmp <= (sel & enable);
--SECTION A: What does this do?  and how
--this rotates a 1 bit value through an
   with tmp select
      output <= "00000001" when "0001",
                "00000010" when "0011",
                "00000100" when "0101",
                "00001000" when "0111",
                "00010000" when "1001",
                "00100000" when "1011",
                "01000000" when "1101",
                "10000000" when "1111",
                "00000001" when others;
```

Figure 2 – Select Case Example

This style of dataflow coding style will be used to write this program. Below, the block/schematic diagram for Part A can be found:

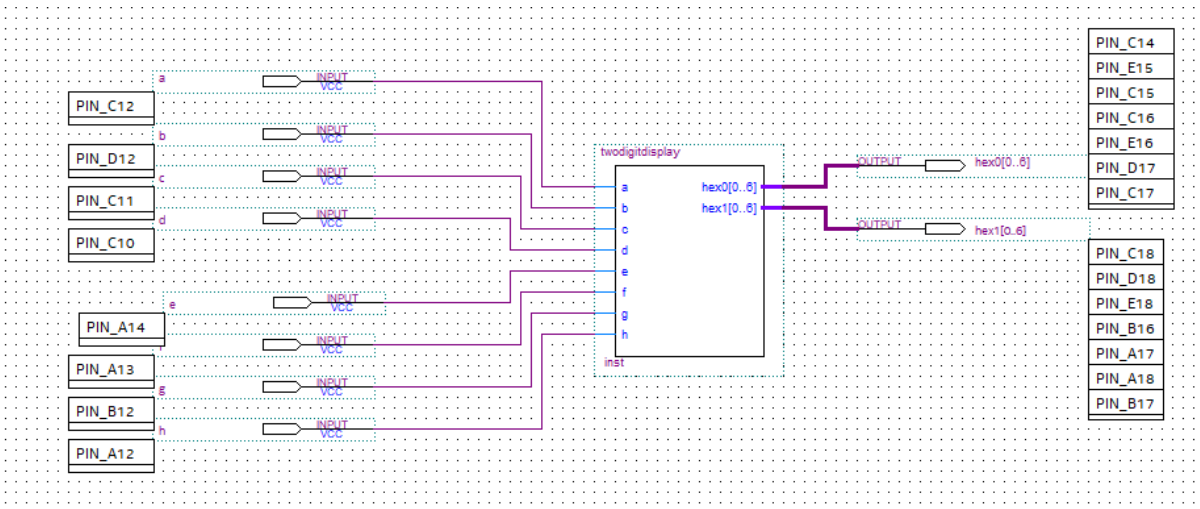

Figure 3 - Lab 4 Part A Block Diagram with Pinouts

Note that the name of each respective input pin and output pin matches the syntax within the symbol file. (i.e. hex0[0..6] denotes the vector *hex0* with values 0 -> 6)

Next, pin assignments are formed in accordance with specifications made by the lab manual[1].

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
a	Input	PIN_C12	7	B7_N0	PIN_C12	3.3-V LVTTL		8mA (default)			
b	Input	PIN_D12	7	B7_N0	PIN_D12	3.3-V LVTTL		8mA (default)			
c	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL		8mA (default)			
d	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL		8mA (default)			
e	Input	PIN_A14	7	B7_N0	PIN_A14	3.3-V LVTTL		8mA (default)			
f	Input	PIN_A13	7	B7_N0	PIN_A13	3.3-V LVTTL		8mA (default)			
g	Input	PIN_B12	7	B7_N0	PIN_B12	3.3-V LVTTL		8mA (default)			
h	Input	PIN_A12	7	B7_N0	PIN_A12	3.3-V LVTTL		8mA (default)			
hex0[0]	Output	PIN_C14	7	B7_N0	PIN_C14	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[1]	Output	PIN_E15	7	B7_N0	PIN_E15	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[2]	Output	PIN_C15	7	B7_N0	PIN_C15	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[3]	Output	PIN_C16	7	B7_N0	PIN_C16	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[4]	Output	PIN_E16	7	B7_N0	PIN_E16	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[5]	Output	PIN_D17	7	B7_N0	PIN_D17	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[6]	Output	PIN_C17	7	B7_N0	PIN_C17	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[0]	Output	PIN_C18	7	B7_N0	PIN_C18	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[1]	Output	PIN_D18	6	B6_N0	PIN_D18	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[2]	Output	PIN_E18	6	B6_N0	PIN_E18	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[3]	Output	PIN_B16	7	B7_N0	PIN_B16	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[4]	Output	PIN_A17	7	B7_N0	PIN_A17	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[5]	Output	PIN_A18	7	B7_N0	PIN_A18	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[6]	Output	PIN_B17	7	B7_N0	PIN_B17	3.3-V LVTTL		8mA (default)	2 (default)		

Figure 4 - Pinout Assignments for Part A

Finally, the project can be compiled, and ensuring that the USB-Blaster hardware is added to the FPGA environment, and programmed to the development board. The results are discussed in the **Validation of Data** section of the report.

Code Explanation Part A

Included below are both an image, and text inclusion of the complete code segment for part A of the experiment:

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity twodigitdisplay is
5   port (
6            a, b, c, d, e, f, g, h: in std_logic; --input switches
7            hex0, hex1 : out std_logic_vector (0 to 6) --output hex displays
8         );
9   end twodigitdisplay;
10
11  architecture dataflow of twodigitdisplay is
12
13  signal x : std_logic_vector(0 to 3); --signal variables referencing concantenated input
14  signal y : std_logic_vector(0 to 3);
15
16  begin
17
18     x <= (a & b & c & d); --assigning input switches to concantenated variables
19     y <= (e & f & g & h);
20
21     with y select --hex1 select states for each switch combination
22        hex1 <=  "0000001" when "0000",
23                 "1001111" when "0001",
24                 "0010010" when "0010",
25                 "0000110" when "0011",
26                 "1001100" when "0100",
27                 "0100100" when "0101",
28                 "0100000" when "0110",A
29                 "0001111" when "0111",
30                 "0000000" when "1000",
31                 "0000100" when "1001",
32                 "1111111" when others;
33
34     with x select --hex0 select states for each switch combination
35        hex0 <=  "0000001" when "0000",
36                 "1001111" when "0001",
37                 "0010010" when "0010",
38                 "0000110" when "0011",
39                 "1001100" when "0100",
40                 "0100100" when "0101",
41                 "0100000" when "0110",
42                 "0001111" when "0111",
43                 "0000000" when "1000",
44                 "0000100" when "1001",
45                 "1111111" when others;
46
47  end dataflow;
```

Figure 5 - Image of Completed Code of Part A

```
library ieee;
use ieee.std_logic_1164.all;

entity twodigitdisplay is
port (
                    a, b, c, d, e, f, g, h: in std_logic; --input switches
                    hex0, hex1 : out std_logic_vector (0 to 6) --output hex displays
          );
end twodigitdisplay;

architecture dataflow of twodigitdisplay is
```

```
signal x : std_logic_vector(0 to 3); --signal variables referencing concantenated input
signal y : std_logic_vector(0 to 3);

begin

	x <= (a & b & c & d); --assigning input switches to concantenated variables
	y <= (e & f & g & h);

	with y select --hex1 select states for each switch combination
		hex1 <=		"0000001" when "0000",
				"1001111" when "0001",
				"0010010" when "0010",
				"0000110" when "0011",
				"1001100" when "0100",
				"0100100" when "0101",
				"0100000" when "0110",A
				"0001111" when "0111",
				"0000000" when "1000",
				"0000100" when "1001",
				"1111111" when others;

	with x select --hex0 select states for each switch combination
		hex0 <=		"0000001" when "0000",
				"1001111" when "0001",
				"0010010" when "0010",
				"0000110" when "0011",
				"1001100" when "0100",
				"0100100" when "0101",
				"0100000" when "0110",
				"0001111" when "0111",
				"0000000" when "1000",
				"0000100" when "1001",
				"1111111" when others;

end dataflow;
```

The entity portion of the code creates 8 scalar input switch variables, from a to h, as well as two 7-bit output vectors for both hex displays (1 and 0).

The architecture section, which denotes a *dataflow* coding style, uses the aforementioned *switch case* statement to program the logic for this system. Note the inclusion of two *signal* statements, which are used to create a concatenated storage variable for the combination of the input switches (a-d for the hex0 and e through h for the hex1). Each select case sets the each of the 7-bits in a hex display for each possible combination of inputs. Since this first system only increments 0 to 9 on each hex display, there are only ten specific assignments, with a general *when others* statement for the remainder of cases (treated as don't cares). For example, *"0010010" when "0010"* sets the hex

display to the decimal value 2 (segments 2 and 5 disabled) when its respective switches are set to the binary value 0010, or decimal value 2.

Note the level of modularity present in this dataflow coding style. While it may not be as modular as a structural coding style, it still offers a heightened level of responsiveness to the program that would have otherwise been left absent.

Part B – Binary Coded Decimal Addition

In part B, a Binary Coded Decimal Adder is created, and the dataflow coding style is once again used. The lab instructions state, however, that this system must be created by exclusively using basic Boolean expressions, and thus the *switch case* solution used in part A will not be allowed. Thus, the creation of truth tables and Karnaugh maps will be vital in creating this logic workflow. Below is the first step, the truth table for each Hex display.

Table 1 – Hex1 7-Segment Display Truth Table

Hex1											
v3	v2	v1	v0	Hex1[0]	Hex1[1]	Hex1[2]	Hex1[3]	Hex1[4]	Hex1[5]	Hex1[6]	
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	1	1
0	0	1	0	0	0	0	0	0	0	1	2
0	0	1	1	0	0	0	0	0	0	1	3
0	1	0	0	0	0	0	0	0	0	1	4
0	1	0	1	0	0	0	0	0	0	1	5
0	1	1	0	0	0	0	0	0	0	1	6
0	1	1	1	0	0	0	0	0	0	1	7
1	0	0	0	0	0	0	0	0	0	1	8
1	0	0	1	0	0	0	0	0	0	1	9
1	0	1	0	1	0	0	1	1	1	1	10
1	0	1	1	1	0	0	1	1	1	1	11
1	1	0	0	1	0	0	1	1	1	1	12
1	1	0	1	1	0	0	1	1	1	1	13
1	1	1	0	1	0	0	1	1	1	1	14
1	1	1	1	1	0	0	1	1	1	1	15

Denoted in yellow on the right hand side of the table is the decimal value for each input *v* combination. The inputs (green), *v3* through *v0* represent the input switches a,b,c, and d respectively, which are attached to SW3-0 on the development board. Finally, each hex1 segment (orange) state is listed per combinatorial input state. The Karnaugh maps for each are as follows:

Hex1[0]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	0	0	0
v3v2'	0	0	0	0
v3v2	1	1	1	1
v3'v2	0	0	1	1

Hex1[1]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	0	0	0
v3v2'	0	0	0	0
v3v2	0	0	0	0
v3'v2	0	0	0	0

Hex1[2]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	0	0	0
v3v2'	0	0	0	0
v3v2	0	0	0	0
v3'v2	0	0	0	0

Hex1[3]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	0	0	0
v3v2'	0	0	0	0
v3v2	1	1	1	1
v3'v2	0	0	1	1

Hex1[4]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	0	0	0
v3v2'	0	0	0	0
v3v2	1	1	1	1
v3'v2	0	0	1	1

Hex1[5]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	0	0	0
v3v2'	0	0	0	0
v3v2	1	1	1	1
v3'v2	0	0	1	1

Hex1[6]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	1	1	1	1
v3v2'	1	1	1	1
v3v2	1	1	1	1
v3'v2	1	1	1	1

Next, in an identical process, the truth table for Hex0 is created. Note that for Hex1, there are only two potential states for the display to read, either 0 or 1. This is due to the fact that the value will only ever reach a maximum of 15, and a minimum of 0, so the left-most (most significant bit) can only ever be either of those two states. In Hex0, any value from 0 to 9 may be shown, thus the truth table is far more complex.

Table 2 – Hex0 7-Segment Display Truth Table

Hex0											
v3	v2	v1	v0	Hex0[0]	Hex0[1]	Hex0[2]	Hex0[3]	Hex0[4]	Hex0[5]	Hex0[6]	
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0	2
0	0	1	1	0	0	0	0	1	1	0	3
0	1	0	0	1	0	0	1	1	0	0	4
0	1	0	1	0	1	0	0	1	0	0	5
0	1	1	0	0	1	0	0	0	0	0	6
0	1	1	1	0	0	0	1	1	1	1	7
1	0	0	0	0	0	0	0	0	0	0	8
1	0	0	1	0	0	0	0	1	0	0	9
1	0	1	0	0	0	0	0	0	0	1	10
1	0	1	1	1	0	0	1	1	1	1	11
1	1	0	0	0	0	1	0	0	1	0	12
1	1	0	1	0	0	0	0	1	1	0	13
1	1	1	0	1	0	0	1	1	0	0	14
1	1	1	1	0	1	0	0	1	0	0	15

The Karnaugh maps are found below, and follow the same color coded structure as both the truth tables, as well as the previous K-maps:

Hex0[0]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	1	0	0
v3v2'	1	0	0	0
v3v2	0	0	0	1
v3'v2	0	0	1	0

Hex0[1]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	0	0	0
v3v2'	0	1	0	1
v3v2	0	0	1	0
v3'v2	0	0	0	0

Hex0[2]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	0	0	1
v3v2'	0	0	0	0
v3v2	1	0	0	0
v3'v2	0	0	0	0

Hex0[3]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	1	0	0
v3v2'	1	0	1	0
v3v2	0	0	0	1
v3'v2	0	0	1	0

Hex0[4]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	1	1	0
v3v2'	1	1	1	0
v3v2	0	1	1	1
v3'v2	0	1	1	0

Hex0[5]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	0	1	1	1
v3v2'	0	0	1	0
v3v2	1	1	0	0
v3'v2	0	0	1	0

Hex0[6]	v1'v0'	v1v0'	v1v0	v1'v0
v3'v2'	1	1	0	0
v3v2'	0	0	1	0
v3v2	0	0	0	0
v3'v2	0	0	1	1

Now, the symbol file must be created from the VHDL file, and the symbol added to a block diagram/schematic file, as pictured in **figure 7**.

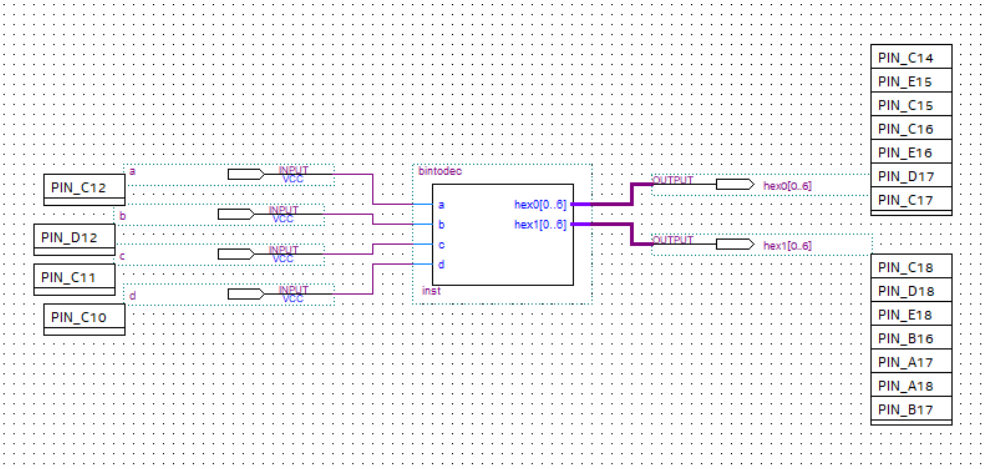

Figure 6 -Lab 4 Part B Schematic File

Note the individual HEX outputs for each respective 7-segment display, as they each have 7 individual segments that which require logic to flow into.

Next, pin assignments are formed in accordance with specifications made by the lab manual[1].

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
a	Input	PIN_C12	7	B7_N0	PIN_C12	3.3-V LVTTL		8mA (default)			
b	Input	PIN_D12	7	B7_N0	PIN_D12	3.3-V LVTTL		8mA (default)			
c	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL		8mA (default)			
d	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL		8mA (default)			
hex0[0]	Output	PIN_C14	7	B7_N0	PIN_C14	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[1]	Output	PIN_E15	7	B7_N0	PIN_E15	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[2]	Output	PIN_C15	7	B7_N0	PIN_C15	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[3]	Output	PIN_C16	7	B7_N0	PIN_C16	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[4]	Output	PIN_E16	7	B7_N0	PIN_E16	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[5]	Output	PIN_D17	7	B7_N0	PIN_D17	3.3-V LVTTL		8mA (default)	2 (default)		
hex0[6]	Output	PIN_C17	7	B7_N0	PIN_C17	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[0]	Output	PIN_C18	7	B7_N0	PIN_C18	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[1]	Output	PIN_D18	6	B6_N0	PIN_D18	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[2]	Output	PIN_E18	6	B6_N0	PIN_E18	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[3]	Output	PIN_B16	7	B7_N0	PIN_B16	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[4]	Output	PIN_A17	7	B7_N0	PIN_A17	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[5]	Output	PIN_A18	7	B7_N0	PIN_A18	3.3-V LVTTL		8mA (default)	2 (default)		
hex1[6]	Output	PIN_B17	7	B7_N0	PIN_B17	3.3-V LVTTL		8mA (default)	2 (default)		

Figure 7 - Pinout Assignments for Lab 4 Part B

Finally, the project can be compiled, and ensuring that the USB-Blaster hardware is added to the FPGA environment, and programmed to the development board. The results are discussed in the **Validation of Data** section of the report. This section of the report covers the resulting development board behavior, with supporting images and evidence.

Code Explanation Part B

Included below are both an image, and text inclusion of the complete code segment for part B of the experiment:

```
library ieee;
use ieee.std_logic_1164.all;

entity bintodec is
port (
        a, b, c, d : in std_logic;
        hex0, hex1 : out std_logic_vector (0 to 6)
    );
end bintodec;

architecture dataflow of bintodec is
begin

hex1(0) <= (a and c) or (a and b); --logic statements for each hex segment in hex 0
hex1(1) <= '0';
hex1(2) <= '0';
hex1(3) <= (a and c) or (a and b);
hex1(4) <= (a and c) or (a and b);
hex1(5) <= (a and c) or (a and b);
hex1(6) <= '1';

hex0(0) <= (not a and not b and not c and d) or (not a and b and not c and not d) or (a and not b and c and d) or (a and b and c and not d); --logic statements for each hex segment in hex 1
hex0(1) <= (not a and b and not c and d) or (not a and b and c and not d) or (a and b and c and d);
hex0(2) <= (not a and not b and c and not d) or (a and b and not c and not d);
hex0(3) <= (not a and not b and not c and d) or (not a and b and not c and not d) or (not a and b and c and d) or (a and not b and c and d) or (a and b and c and not d);
hex0(4) <= (d) or (not a and b and not c) or (a and b and c);
hex0(5) <= (not a and not b and d) or (not a and not b and c) or (not a and c and d) or (not b and c and d) or (a and b and not c);
hex0(6) <= (not a and not b and not c) or (not a and b and c and d) or (a and not b and c);

end dataflow;
```

Figure 8 – Image of Completed Code for Part B

The program opens with an entity statement which defines the four input switches a, b, c, and d (SW3-0), and the two 7-segment hex displays as 7-bit vectors. In the dataflow architecture, each line is its own Boolean logic statement for each segment of the two hex displays. Note that for hex1, there are only three distinct statements, which aligns with the truth table wherein there are only two possible states. Segments 0, 3, 4, and 5 are always logic high if inputs a and c are high, or if a and b are high. Segments 1 and 2 are always logic low regardless of circumstance, and Segment 6 is always logic high, as the middle bar of the hex display should never show (if the only two states are 0 and 1).

The logic for the first hex display is far more complex, and is dependent on each Karnaugh map derived from the Hex0 truth table. In fact, this section of the logic workflow of the program contains 7 distinct Boolean logic expressions for 7 possible scenarios.

```
library ieee;
use ieee.std_logic_1164.all;

entity bintodec is
port (
        a, b, c, d : in std_logic;
        hex0, hex1 : out std_logic_vector (0 to 6)
    );
end bintodec;

architecture dataflow of bintodec is
begin

hex1(0) <= (a and c) or (a and b); --logic statements for each hex segment in hex 0
hex1(1) <= '0';
hex1(2) <= '0';
```

```
hex1(3) <= (a and c) or (a and b);
hex1(4) <= (a and c) or (a and b);
hex1(5) <= (a and c) or (a and b);
hex1(6) <= '1';

hex0(0) <= (not a and not b and not c and d) or (not a and b and not c and not d) or (a and not b and c and d) or
(a and b and c and not d); --logic statements for each hex segment in hex 1
hex0(1) <= (not a and b and not c and d) or (not a and b and c and not d) or (a and b and c and d);
hex0(2) <= (not a and not b and c and not d) or (a and b and not c and not d);
hex0(3) <= (not a and not b and not c and d) or (not a and b and not c and not d) or (not a and b and c and d) or
(a and not b and c and d) or (a and b and c and not d);
hex0(4) <= (d) or (not a and b and not c) or (a and b and c);
hex0(5) <= (not a and not b and d) or (not a and not b and c) or (not a and c and d) or (not b and c and d) or (a
and b and not c);
hex0(6) <= (not a and not b and not c) or (not a and b and c and d) or (a and not b and c);

end dataflow;
```

VALIDATION OF DATA

Bench Analysis Part A

https://youtu.be/T6_OPWrLxM4

Part A shows an individual counter on each of the two hex displays, each which count up to the decimal value 9.

Bench Analysis Part B

https://youtu.be/Y0kIJGUSj1Y

Part B shows a joint counter which utilizes the two hex displays to count up to the decimal value of 15.

CONCLUSION

The experiment was conducted successfully and efficiently demonstrated the use of both the new *switch case* statement, as well as the use of Karnaugh maps in generating Boolean logic based VHDL code. The dataflow coding style is the perfect choice of the three main VHDL coding styles to use in this experiment as it allows clear and direct representation of how data flows through a system. Conducting the experiment in two distinct parts allowed for a progressive educational development of the content of the lab. In other words, part A allowed for obtaining an understanding of using switches as inputs and the hex displays as outputs in use of a binary conversion counter, allowing for simpler more user friendly coding behavior, while part B necessitated the direct

manipulation of the Boolean logic for each operation conducted in the experiment. In the future, adding an additional part in the lab wherein advanced operations, such as arithmetic on the displays, would prove beneficial in learning more about the interaction between such Boolean logic and VHDL, dataflow programming.

REFERENCES

[1] Professor Ashley Evans, *Logic Devices Programming Lab Manual*, 1st ed. Orlando, FL: Valencia College, 2025.