CRN: 22840

CET3136 – Logic Devices Programming

Spring 2025

Experiment 3

Lights, Switches, and Multiplexers – Part 2

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

01/30/2025

Introduction

The intent of this lab is to create a three part project, wherein it is learned to connect simple input and output devices to an FPGA chip and implement circuits using those devices. The FPGA DE1 development board will be used to conduct this experiment, using switches 9-0, 5-0, and hex displays 3-0. The process used to demonstrate this hardware will be a rotating display mechanism which alternates between three states (d, 1, E). The three parts the experiment will be conducted in are as an implementation of a 7-segment decoder, a 3-to-1 multiplexer with 7-segment integration from the previous part, and finally an expanded form of the second part using a fourth 7-segment display and rotating a pattern through such displays.

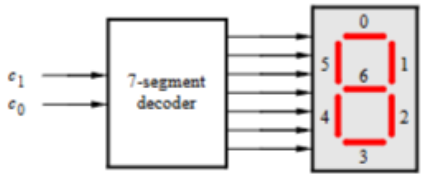
LIST OF EQUIPMENT/PARTS/COMPONENTS/SOFTWARE

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

PROCEDURE / DISCUSSION

Part A – 7-Segment Decoder

It is beneficial to the successful completion of the lab to analyze any truth tables or schematics involved with the design of the program, as it is integral to maintaining an efficient and coordinated design effort in both code and hardware integration.



c_1c_0	Character
00	d
01	E
10	1
11	

Figure 1 – 7-segment Decoder with Character Codes

Creating a truth table for the hex display will aid in writing the logic responsible for controlling said display. It should be noted that these 7-segment hex displays operate on a logic LOW basis.

C1	C0	Hex0[0]	Hex0[1]	Hex0[2]	Hex0[3]	Hex0[4]	Hex0[5]	Hex0[6]
0	**0**	1	0	0	0	0	1	0
0	**1**	0	1	1	0	0	0	0
1	**0**	1	0	0	1	1	1	1
1	**1**	1	1	1	1	1	1	1

The resulting logic statements are as follows, as pulled directly from the VHDL code written in part A of the experiment:

```
hex0(0) <= NOT (NOT c1 AND c0);
hex0(1) <= NOT (NOT c0);
hex0(2) <= NOT (NOT c0);
hex0(3) <= NOT (NOT c1);
hex0(4) <= NOT (NOT c1);
hex0(5) <= NOT (NOT c1 AND c0);
hex0(6) <= NOT (NOT c1);
```

Figure 2 - VHDL Logic for Part A

The code is written in a simple dataflow coding style, which will be the only component written in this style for the remainder of the experiment. Below, the block/schematic diagram for Part A can be found:

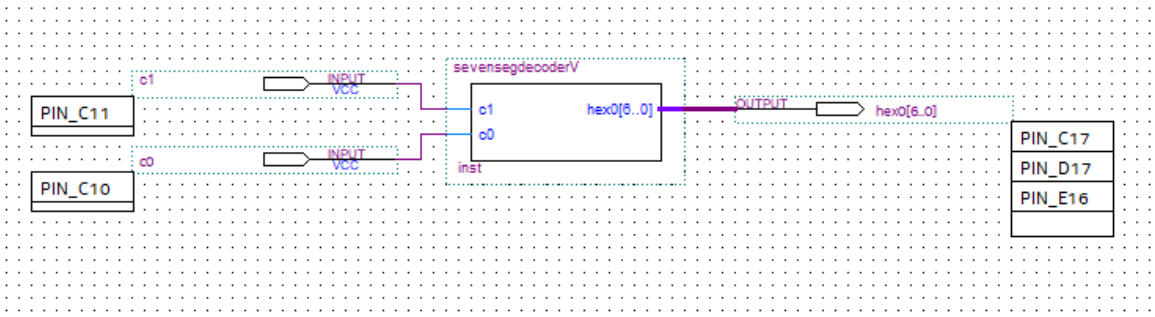


Figure 3 - Lab 3 Part A Schematic File

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

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
c0	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL		8mA (default)			
c1	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL		8mA (default)			
hex0[6]	Output	PIN_C17	7	B7_N0	PIN_C17	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[4]	Output	PIN_E16	7	B7_N0	PIN_E16	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[2]	Output	PIN_C15	7	B7_N0	PIN_C15	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[0]	Output	PIN_C14	7	B7_N0	PIN_C14	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:

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

entity sevensegdecoderv is
port (
        c1, c0: in STD_LOGIC;
        hex0: out STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
end sevensegdecoderv;

architecture dataflow of sevensegdecoderv is
begin
    hex0(0) <= NOT (NOT c1 AND c0);
    hex0(1) <= NOT (NOT c0);
    hex0(2) <= NOT (NOT c0);
    hex0(3) <= NOT (NOT c1);
    hex0(4) <= NOT (NOT c1);
    hex0(5) <= NOT (NOT c1 AND c0);
    hex0(6) <= NOT (NOT c1);

end dataflow;
```

Figure 5 - Image of Completed Code of Part A

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

entity sevensegdecoderV  is
port (
                    c1, c0: in STD_LOGIC;
                    hex0: out STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
end sevensegdecoderV;

architecture  dataflow of sevensegdecoderV  is
begin
        hex0(0) <= NOT (NOT c1 AND c0);
        hex0(1) <= NOT (NOT c0);
        hex0(2) <= NOT (NOT c0);
        hex0(3) <= NOT (NOT c1);
        hex0(4) <= NOT (NOT c1);
        hex0(5) <= NOT (NOT c1 AND c0);
        hex0(6) <= NOT (NOT c1);

end dataflow;
```

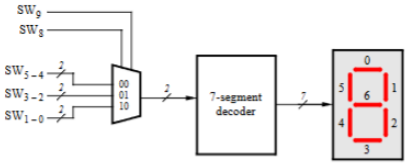
The entity portion of the code, where the inputs and outputs are created, is described above in **Part 1**.

The architecture section, which denotes a *dataflow* coding style, uses the aforementioned VHDL logic representation of the circuit, creates one statement for each segment of the display

Note the simplicity of the *dataflow* coding style is unique due to its exclusive use of concurrent signal assignment statements. However, programming larger projects using only this style would become convoluted and difficult.

Part B – 3-to-1 Multiplexer and 7-Segment Display Integration

In part B, a 2-bit e-to-1 multiplexer is implemented, and the structural coding style is used for the first time. The structural coding style is unique due to its high degree of modularity, and allows for the creation of "components", which can be used to efficiently and easily produce large logic systems through VHDL.



SW_9 SW_8	Character pattern		
00	d	E	1
01	E	1	d
10	1	d	E

Figure 6 - Symbol Diagram of Project Layout and Character Pattern Table

The top-level entity (*rotatingword*) is provided by the lab manual, and receives a 10-bit input in the form of selector switches, and character codes. Three of the aforementioned *components* are used, taking shape as 2-bit 3-to-1 multiplexers called *mux_2bit_3to1*. These components are *instantiated* and select one of three character codes (u, v or w) based on the current state of the selector switches 9 and 8. This output is subsequently distributed to the 7-segment decoder, which correctly displays segments on the hex displays. The multiplexer component implementation of the program utilizes a behavioral coding style, as visualizing the decoder using conditional based logic is satisfactory, while the 7-segment follows a dataflow approach, as conducted in part A.

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**.

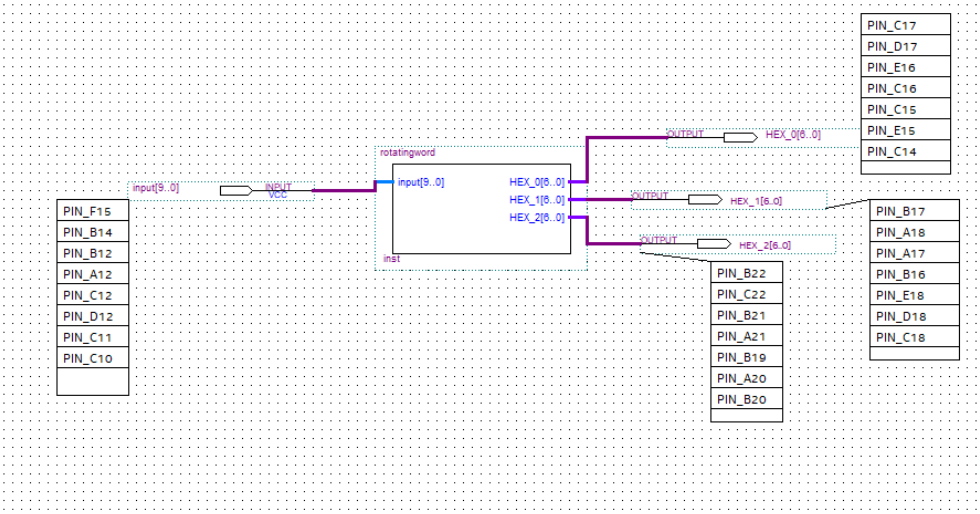


Figure 7 -Lab 3 Part B Schematic File

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].

HEX_0[6]	Output	PIN_C17	7	B7_N0	PIN_C17	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_0[5]	Output	PIN_D17	7	B7_N0	PIN_D17	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_0[4]	Output	PIN_E16	7	B7_N0	PIN_E16	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_0[3]	Output	PIN_C16	7	B7_N0	PIN_C16	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_0[2]	Output	PIN_C15	7	B7_N0	PIN_C15	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_0[1]	Output	PIN_E15	7	B7_N0	PIN_E15	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_0[0]	Output	PIN_C14	7	B7_N0	PIN_C14	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_1[6]	Output	PIN_B17	7	B7_N0	PIN_B17	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_1[5]	Output	PIN_A18	7	B7_N0	PIN_A18	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_1[4]	Output	PIN_A17	7	B7_N0	PIN_A17	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_1[3]	Output	PIN_B16	7	B7_N0	PIN_B16	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_1[2]	Output	PIN_E18	6	B6_N0	PIN_E18	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_1[1]	Output	PIN_D18	6	B6_N0	PIN_D18	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_1[0]	Output	PIN_C18	7	B7_N0	PIN_C18	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_2[6]	Output	PIN_B22	6	B6_N0	PIN_B22	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_2[5]	Output	PIN_C22	6	B6_N0	PIN_C22	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_2[4]	Output	PIN_B21	6	B6_N0	PIN_B21	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_2[3]	Output	PIN_A21	6	B6_N0	PIN_A21	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_2[2]	Output	PIN_B19	7	B7_N0	PIN_B19	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_2[1]	Output	PIN_A20	7	B7_N0	PIN_A20	3.3-V LVTTL	8mA (default)	2 (default)	
HEX_2[0]	Output	PIN_B20	6	B6_N0	PIN_B20	3.3-V LVTTL	8mA (default)	2 (default)	
input[9]	Input	PIN_F15	7	B7_N0	PIN_F15	3.3-V LVTTL	8mA (default)		
input[8]	Input	PIN_B14	7	B7_N0	PIN_B14	3.3-V LVTTL	8mA (default)		
input[7]	Input				PIN_M21	3.3-V LVTTL	8mA (default)		
input[6]	Input				PIN_R12	3.3-V LVTTL	8mA (default)		
input[5]	Input	PIN_B12	7	B7_N0	PIN_B12	3.3-V LVTTL	8mA (default)		
input[4]	Input	PIN_A12	7	B7_N0	PIN_A12	3.3-V LVTTL	8mA (default)		
input[3]	Input	PIN_C12	7	B7_N0	PIN_C12	3.3-V LVTTL	8mA (default)		
input[2]	Input	PIN_D12	7	B7_N0	PIN_D12	3.3-V LVTTL	8mA (default)		
input[1]	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL	8mA (default)		
input[0]	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL	8mA (default)		
<<new node>>									

Figure 8 - Pinout Assignments for Lab 3 Part B

Note that there are two unused input pins, switches 7 and 8. These are not necessary for part B of this experiment, and will be address in the final, part C of the lab.

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:

Figure 9 – Image of Completed Code for Part B

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

--top level entity--
entity rotatingword  IS --10-bit  input to 2bit-3to1 muxs
   PORT (
      input : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
      HEX_0, HEX_1, HEX_2: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
   );
end rotatingword;

--3 muxs and 3 hexs
architecture  structure  of rotatingword  is

component mux_2bit_3to1  is --mux component info
   port (
      S, U, V, W: IN STD_LOGIC_VECTOR(1  DOWNTO 0);
      M : OUT STD_LOGIC_VECTOR(1  DOWNTO 0)
   );
end component;

component sevensegdecoder  is --decoder  component info
   port (
```

```
      C : in std_logic_vector(1 DOWNTO 0);
      hex : out STD_LOGIC_VECTOR(6 DOWNTO 0)
   );
end component;

signal mux0, mux1, mux2 : STD_LOGIC_VECTOR(1 DOWNTO 0);
begin

   M0: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(5 DOWNTO 4), V => input(3
DOWNTO 2), W => input(1 DOWNTO 0), M => mux0); --mux instantiation
   M1: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(3 DOWNTO 2), V => input(1
DOWNTO 0), W => input(5 DOWNTO 4), M => mux1);
   M2: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(1 DOWNTO 0), V => input(5
DOWNTO 4), W => input(3 DOWNTO 2), M => mux2);

   display0 : sevensegdecoder port map (C => mux0, hex => HEX_0);--instantiation decoder
   display1 : sevensegdecoder port map (C => mux1, hex => HEX_1);
   display2 : sevensegdecoder port map (C => mux2, hex => HEX_2);
end structure;

-- Implements a 2-bit wide 3-to-1 multiplexer
library ieee;
use ieee.std_logic_1164.all;

entity mux_2bit_3to1 IS
   port (
      S, U, V, W: IN STD_LOGIC_VECTOR(1 DOWNTO 0); --2bit data inputs
      M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
   );
end mux_2bit_3to1;

architecture Behavior OF mux_2bit_3to1 IS
begin
   process (U, V, W, S) --determine which data is passed
   begin
      if (S(1) = '0') and (S(0) = '0') then
         M <= U;
      elsif (S(1) = '1') and (S(0) = '0') then
         M <= V;
      elsif (S(1) = '0') and (S(0) = '1') then
         M <= W;
      else
         M <= "11";
      end if;
   end process;
end Behavior;

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

```
entity sevensegdecoder is --decoder (use part A dataflow logic)
  port (
    C: in std_logic_vector(1 DOWNTO 0);
    hex: out std_logic_vector(6 DOWNTO 0)
  );
end sevensegdecoder;

architecture dataflow of sevensegdecoder is
begin
  hex(0) <= not C(0) or C(1);
  hex(1) <= C(0);
  hex(2) <= C(0);
  hex(3) <= C(1);
  hex(4) <= C(1);
  hex(5) <= not C(0) or C(1);
  hex(6) <= C(1);
end dataflow;
```

Please see above for a discussion of Part B's code.

Part C – 4-Character Rotating Display

Part C is simply an extension of part B, and adds certain components to the program. Namely, an additional multiplexer, input, and hex display are added, bringing the total of each to 4. The final character pattern table is offered below:

SW_9 SW_8	Character pattern			
00	d	E	1	
01	E	1		d
10	1		d	E
11		d	E	1

Figure 10 – Character Pattern Table for Part C

Switches 7 and 6 now play a role as inputs in this part of the experiment, and follow the same routes as the other three inputs did in the previous part of the lab. In fact, this portion of the experiment is nearly identical to the previous part were it not for these few additions. The program still rotates through these characters.

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**.

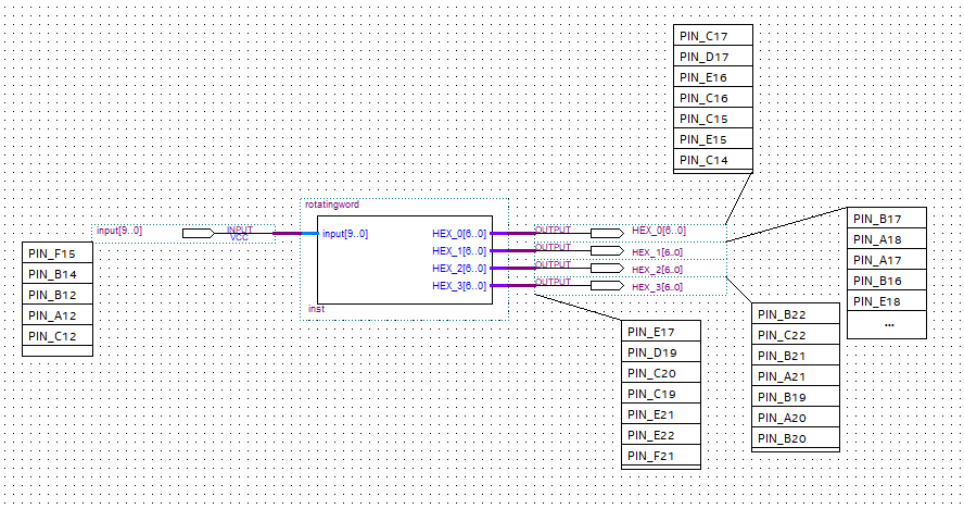


Figure 1 -Lab 3 Part C Schematic File

Now the fourth previously mentioned additions can be visualized on the block diagram schematic, reaffirming its simple implementation into the program, thanks to the nature of the structural coding style.

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

HEX_0[3]	Output	PIN_C16	7	B7_N0	PIN_C16	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_0[2]	Output	PIN_C15	7	B7_N0	PIN_C15	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_0[1]	Output	PIN_E15	7	B7_N0	PIN_E15	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_0[0]	Output	PIN_C14	7	B7_N0	PIN_C14	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_1[6]	Output	PIN_B17	7	B7_N0	PIN_B17	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_1[5]	Output	PIN_A18	7	B7_N0	PIN_A18	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_1[4]	Output	PIN_A17	7	B7_N0	PIN_A17	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_1[3]	Output	PIN_B16	7	B7_N0	PIN_B16	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_1[2]	Output	PIN_E18	6	B6_N0	PIN_E18	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_1[1]	Output	PIN_D18	6	B6_N0	PIN_D18	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_1[0]	Output	PIN_C18	7	B7_N0	PIN_C18	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_2[6]	Output	PIN_B22	6	B6_N0	PIN_B22	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_2[5]	Output	PIN_C22	6	B6_N0	PIN_C22	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_2[4]	Output	PIN_B21	6	B6_N0	PIN_B21	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_2[3]	Output	PIN_A21	6	B6_N0	PIN_A21	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_2[2]	Output	PIN_B19	7	B7_N0	PIN_B19	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_2[1]	Output	PIN_A20	7	B7_N0	PIN_A20	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_2[0]	Output	PIN_B20	6	B6_N0	PIN_B20	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_3[6]	Output	PIN_E17	6	B6_N0	PIN_E17	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_3[5]	Output	PIN_D19	6	B6_N0	PIN_D19	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_3[4]	Output	PIN_C20	6	B6_N0	PIN_C20	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_3[3]	Output	PIN_C19	7	B7_N0	PIN_C19	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_3[2]	Output	PIN_E21	6	B6_N0	PIN_E21	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_3[1]	Output	PIN_E22	6	B6_N0	PIN_E22	3.3-V LVTTL		8mA (default)	2 (default)		
HEX_3[0]	Output	PIN_F21	6	B6_N0	PIN_F21	3.3-V LVTTL		8mA (default)	2 (default)		
input[9]	Input	PIN_F15	7	B7_N0	PIN_F15	3.3-V LVTTL		8mA (default)			
input[8]	Input	PIN_B14	7	B7_N0	PIN_B14	3.3-V LVTTL		8mA (default)			
input[7]	Input	PIN_A14	7	B7_N0	PIN_A14	3.3-V LVTTL		8mA (default)			
input[6]	Input	PIN_A13	7	B7_N0	PIN_A13	3.3-V LVTTL		8mA (default)			
input[5]	Input	PIN_B12	7	B7_N0	PIN_B12	3.3-V LVTTL		8mA (default)			
input[4]	Input	PIN_A12	7	B7_N0	PIN_A12	3.3-V LVTTL		8mA (default)			
input[3]	Input	PIN_C12	7	B7_N0	PIN_C12	3.3-V LVTTL		8mA (default)			
input[2]	Input	PIN_D12	7	B7_N0	PIN_D12	3.3-V LVTTL		8mA (default)			
input[1]	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL		8mA (default)			
input[0]	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL		8mA (default)			

Figure 12 - Pinout Assignments for Lab 3 Part C

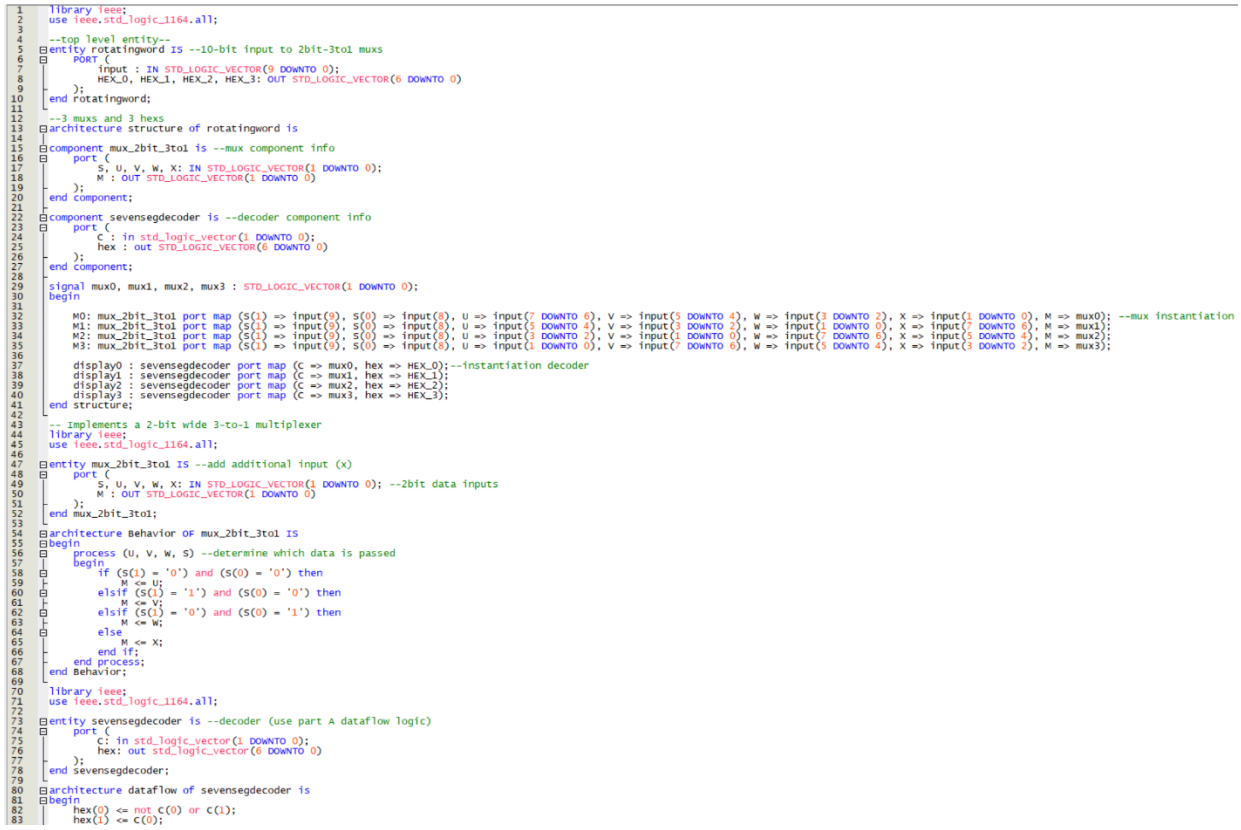
Note that the two previously unused input pins, switches 7 and 8, are now assigned.

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 C

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;

--top level entity--
entity rotatingword IS --10-bit input to 2bit-3to1 muxs
    PORT (
        input : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
        HEX_0, HEX_1, HEX_2, HEX_3: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
end rotatingword;

--3 muxs and 3 hexs
architecture structure of rotatingword is

component mux_2bit_3to1 is --mux component info
    port (
        S, U, V, W, X: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
    );
end component;

component sevensegdecoder is --decoder component info
    port (
        C : in std_logic_vector(1 DOWNTO 0);
        hex : out STD_LOGIC_VECTOR(6 DOWNTO 0)
    );
end component;

signal mux0, mux1, mux2, mux3 : STD_LOGIC_VECTOR(1 DOWNTO 0);
begin

    M0: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(7 DOWNTO 6), V => input(5 DOWNTO 4), W => input(3 DOWNTO 2), X => input(1 DOWNTO 0), M => mux0); --mux instantiation
    M1: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(5 DOWNTO 4), V => input(3 DOWNTO 2), W => input(1 DOWNTO 0), X => input(7 DOWNTO 6), M => mux1);
    M2: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(3 DOWNTO 2), V => input(1 DOWNTO 0), W => input(7 DOWNTO 6), X => input(5 DOWNTO 4), M => mux2);
    M3: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(1 DOWNTO 0), V => input(7 DOWNTO 6), W => input(5 DOWNTO 4), X => input(3 DOWNTO 2), M => mux3);

    display0 : sevensegdecoder port map (C => mux0, hex => HEX_0);--instantiation decoder
    display1 : sevensegdecoder port map (C => mux1, hex => HEX_1);
    display2 : sevensegdecoder port map (C => mux2, hex => HEX_2);
    display3 : sevensegdecoder port map (C => mux3, hex => HEX_3);
end structure;

-- Implements a 2-bit wide 3-to-1 multiplexer
library ieee;
use ieee.std_logic_1164.all;

entity mux_2bit_3to1 IS --add additional input (x)
    port (
        S, U, V, W, X: IN STD_LOGIC_VECTOR(1 DOWNTO 0); --2bit data inputs
        M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
    );
end mux_2bit_3to1;

architecture Behavior OF mux_2bit_3to1 IS
begin
    process (U, V, W, S) --determine which data is passed
    begin
        if (S(1) = '0') and (S(0) = '0') then
            M <= U;
        elsif (S(1) = '1') and (S(0) = '0') then
            M <= V;
        elsif (S(1) = '0') and (S(0) = '1') then
            M <= W;
        else
            M <= X;
        end if;
    end process;
end Behavior;

library ieee;
use ieee.std_logic_1164.all;

entity sevensegdecoder is --decoder (use part A dataflow logic)
    port (
        C: in std_logic_vector(1 DOWNTO 0);
        hex: out std_logic_vector(6 DOWNTO 0)
    );
end sevensegdecoder;

architecture dataflow of sevensegdecoder is
begin
    hex(0) <= not C(0) or C(1);
    hex(1) <= C(0);
```

Figure 13 – Image of Completed Code for Part C

```
component mux_2bit_3to1 is --mux component info
  port (
    S, U, V, W, X: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
    M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
  );
end component;

component sevensegdecoder is --decoder component info
  port (
    C : in std_logic_vector(1 DOWNTO 0);
    hex : out STD_LOGIC_VECTOR(6 DOWNTO 0)
  );
end component;

signal mux0, mux1, mux2, mux3 : STD_LOGIC_VECTOR(1 DOWNTO 0);
begin

  M0: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(7 DOWNTO 6), V => input(5
DOWNTO 4), W => input(3 DOWNTO 2), X => input(1 DOWNTO 0), M => mux0); --mux instantiation
  M1: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(5 DOWNTO 4), V => input(3
DOWNTO 2), W => input(1 DOWNTO 0), X => input(7 DOWNTO 6), M => mux1);
  M2: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(3 DOWNTO 2), V => input(1
DOWNTO 0), W => input(7 DOWNTO 6), X => input(5 DOWNTO 4), M => mux2);
  M3: mux_2bit_3to1 port map (S(1) => input(9), S(0) => input(8), U => input(1 DOWNTO 0), V => input(7
DOWNTO 6), W => input(5 DOWNTO 4), X => input(3 DOWNTO 2), M => mux3);

  display0 : sevensegdecoder port map (C => mux0, hex => HEX_0);--instantiation decoder
  display1 : sevensegdecoder port map (C => mux1, hex => HEX_1);
  display2 : sevensegdecoder port map (C => mux2, hex => HEX_2);
  display3 : sevensegdecoder port map (C => mux3, hex => HEX_3);
end structure;

-- Implements a 2-bit wide 3-to-1 multiplexer
library ieee;
use ieee.std_logic_1164.all;

entity mux_2bit_3to1 IS --add additional input (x)
  port (
    S, U, V, W, X: IN STD_LOGIC_VECTOR(1 DOWNTO 0); --2bit data inputs
    M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
  );
end mux_2bit_3to1;

architecture Behavior OF mux_2bit_3to1 IS
begin
  process (U, V, W, S) --determine which data is passed
  begin
    if (S(1) = '0') and (S(0) = '0') then
```

```
        M <= U;
    elsif (S(1) = '1') and (S(0) = '0') then
        M <= V;
    elsif (S(1) = '0') and (S(0) = '1') then
        M <= W;
    else
        M <= X;
    end if;
  end process;
end Behavior;

library ieee;
use ieee.std_logic_1164.all;

entity sevensegdecoder is --decoder (use part A dataflow logic)
  port (
    C: in std_logic_vector(1 DOWNTO 0);
    hex: out std_logic_vector(6 DOWNTO 0)
  );
end sevensegdecoder;

architecture dataflow of sevensegdecoder is
begin
  hex(0) <= not C(0) or C(1);
  hex(1) <= C(0);
  hex(2) <= C(0);
  hex(3) <= C(1);
  hex(4) <= C(1);
  hex(5) <= not C(0) or C(1);
  hex(6) <= C(1);
end dataflow;
```

Please see above for a discussion of Part C's code.

VALIDATION OF DATA

Bench Analysis Part A

https://youtu.be/zidg_LfAFys

Bench Analysis Part B

https://youtu.be/6LhWhr7mA4o

Bench Analysis Part C

https://youtu.be/gQRFyIATId8

CONCLUSION

The experiment was conducted successfully and efficiently demonstrated the use of the structural VHDL coding style in performing the implementation of a 7-segment decoder and multiplexer within the Quartus software and DE10-Lite hardware integration environment. Taking the experiment sequentially allowed for the clear visualization of both the significance of the structural coding style, but also its efficiency alongside its shortcomings. This lab represents a clear jump in difficulty when compared to the previous labs, as many may find following the structural coding style in a new environment difficult and confusing. Maintaining fundamental understandings of any components of a project using this methodology is vital to nurturing positive programming and logic design habits, as well as completing experiments such as this successfully. In the future, breaking this lab into two separate segments may assist in helping students new to the FPGA environment along, however this is a niche recommendation. For a more difficult or challenging addition to the lab, additional character sets or different animations rather than simply a rotation could be implemented.

REFERENCES

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