CRN: 22840

<u>CET3136 – Logic Devices Programming</u>

Spring 2025

Experiment 2

Lights, Switches, and Multiplexers – 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

01/22/2025

Introduction

The intent of this lab is to create a two part project, wherein a 4-bit 2-to-1 multiplexer, and a 1-bit 3-to-1 multiplexer respectively. Two different programming styles will be practiced and analyzed through the course of the study, those being behavioral and dataflow, the difference being their type of concurrent statements used. The first part, or part 'A' of the experiment focuses on the 2-to-1 multiplexer programmed using a 'dataflow' style of coding. Part 'B' of the study will follow the creation of the 3-to-1 multiplexer, instead using a 'behavioral' method of coding. Recall that a multiplexer is a circuit which takes an input from a selection of inputs, and passes it through the circuit toward the output, this is vital to understand for the successful commencement of the experiment.

LIST OF EQUIPMENT/PARTS/COMPONENTS/SOFTWARE

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

PROCEDURE / DISCUSSION

Part A – 2-to-1 Multiplexer in Dataflow Coding Style

It is beneficial to the successful completion of the lab to analyze the provided logic schematic for a 4-bit 2-to-1 multiplexer, as it lays out precisely how the circuit should function and lays a literal blueprint for designing it's code.

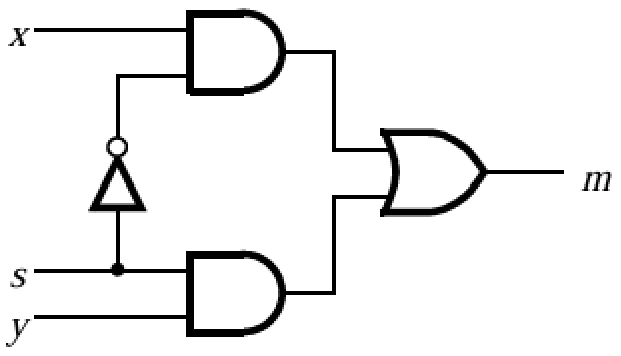


Figure 1 - 2-to-1 Multiplexer Schematic

An input signal 'x' moves through an *AND* gate, along with the inverted signal of an input 's'. The raw signal input 's' moves through a different *AND* gate, along with a signal input 'y'. The result of these two *AND* gates finally moves through an *OR* gate to produce an output ‘m’. Defining this logic schematic into a VHDL statement is critical if the design is to be programmed, and is written as such:

$$m <= (NOT\ s\ AND\ x) OR\ (s\ AND\ y);$$

The implementation of this statement is expounded upon in the **Code Explanation Part A** section of the report.

Within the *entity* body of the VHDL file, three inputs and one output must be instantiated. The first of which being ‘x’, which acts as a vector of magnitude 4. The second, input ‘y’ also with a size of 4. These two inputs ‘x’ and ‘y’ will be created as vectors to accomplish this 4-bit size. The input ‘s’ is a simple scalar value, and the output ‘m’ also serves as a vector, however of magnitude 2.

The *Architecture* segment of the contains the aforementioned VHDL logic interpretation of the schematic, however with syntax to accommodate the vector nature of ‘x’, ‘y’, and ‘m’.

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 2**.

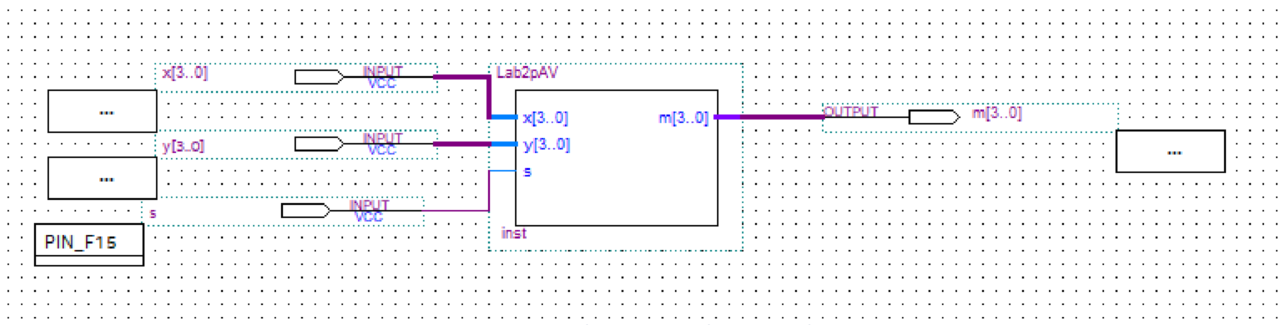
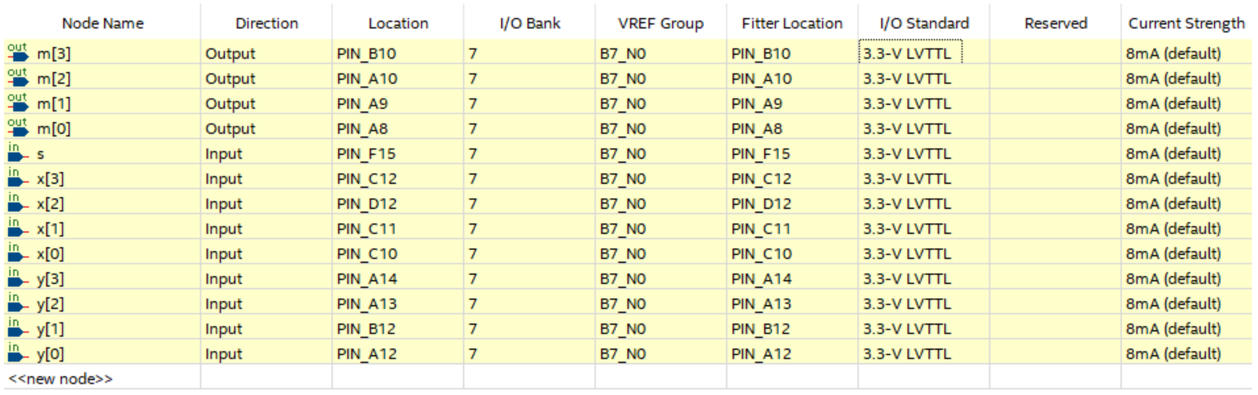


Figure 2 - Lab 2 Part A Schematic File

Note that the name of each respective input pin and output pin matches the syntax within the symbol file. (i.e. x[3..0] denotes the vector *x* with values 3 -> 0)

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



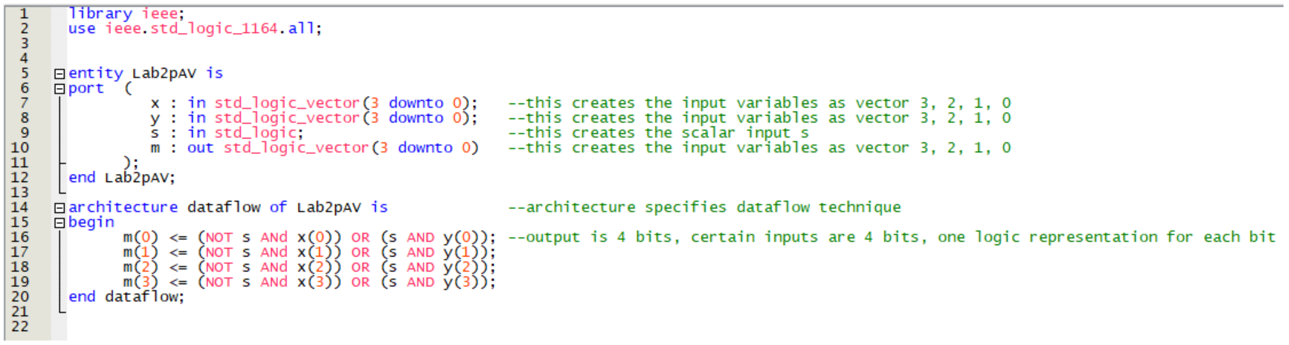
Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength
m[3]	Output	PIN_B10	7	B7_N0	PIN_B10	3.3-V LVTTL		8mA (default)
m[2]	Output	PIN_A10	7	B7_N0	PIN_A10	3.3-V LVTTL		8mA (default)
m[1]	Output	PIN_A9	7	B7_N0	PIN_A9	3.3-V LVTTL		8mA (default)
m[0]	Output	PIN_A8	7	B7_N0	PIN_A8	3.3-V LVTTL		8mA (default)
s	Input	PIN_F15	7	B7_N0	PIN_F15	3.3-V LVTTL		8mA (default)
x[3]	Input	PIN_C12	7	B7_N0	PIN_C12	3.3-V LVTTL		8mA (default)
x[2]	Input	PIN_D12	7	B7_N0	PIN_D12	3.3-V LVTTL		8mA (default)
x[1]	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL		8mA (default)
x[0]	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL		8mA (default)
y[3]	Input	PIN_A14	7	B7_N0	PIN_A14	3.3-V LVTTL		8mA (default)
y[2]	Input	PIN_A13	7	B7_N0	PIN_A13	3.3-V LVTTL		8mA (default)
y[1]	Input	PIN_B12	7	B7_N0	PIN_B12	3.3-V LVTTL		8mA (default)
y[0]	Input	PIN_A12	7	B7_N0	PIN_A12	3.3-V LVTTL		8mA (default)
<<new node>>								

Figure 3 - 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 Lab2pAV is
port  (
          x : in std_logic_vector(3 downto 0);   --this creates the input variables as vector 3, 2, 1, 0
          y : in std_logic_vector(3 downto 0);   --this creates the input variables as vector 3, 2, 1, 0
          s : in std_logic;                      --this creates the scalar input s
          m : out std_logic_vector(3 downto 0)   --this creates the input variables as vector 3, 2, 1, 0
      );
end Lab2pAV;

architecture dataflow of Lab2pAV is               --architecture specifies dataflow technique
begin
      m(0) <= (NOT s AND x(0)) OR (s AND y(0));   --output is 4 bits, certain inputs are 4 bits, one logic representation for each bit
      m(1) <= (NOT s AND x(1)) OR (s AND y(1));
      m(2) <= (NOT s AND x(2)) OR (s AND y(2));
      m(3) <= (NOT s AND x(3)) OR (s AND y(3));
end dataflow;
```

Figure 4 - Image of Completed Code of Part A

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


entity Lab2pAV is
port	(
				x : in std_logic_vector(3 downto 0); --this creates the input variables as vector 3, 2, 1, 0
				y : in std_logic_vector(3 downto 0);--this creates the input variables as vector 3, 2, 1, 0
				s : in std_logic;			--this creates the scalar input s
				m : out std_logic_vector(3 downto 0) --this creates the input variables as vector 3, 2, 1,
0
			);
end Lab2pAV;

architecture dataflow of Lab2pAV is				--architecture specifies dataflow technique
begin
			m(0) <= (NOT s ANd x(0)) OR (s AND y(0));
						--output is 4 bits, certain inputs are 4 bits, one logic
					--representation for each bit
			m(1) <= (NOT s ANd x(1)) OR (s AND y(1));
			m(2) <= (NOT s ANd x(2)) OR (s AND y(2));
			m(3) <= (NOT s ANd x(3)) OR (s AND y(3));
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 bit of the output signal ‘m’. Note that in each, the respective value of each input ‘x’ and ‘y’ are called using parentheses.

The *dataflow* coding style is unique due to its exclusive use of concurrent signal assignment statements. The difference between this and other coding styles will be more apparent in **Part 2** when the behavioral coding style is analyzed.

Part 2 – 2-to-1 Multiplexer in Dataflow Coding Style

Similarly to **Part 1**, the symbol diagram for a 2-bit 3-to-1 multiplexer will be analyzed to help in designing the programming for the circuit:

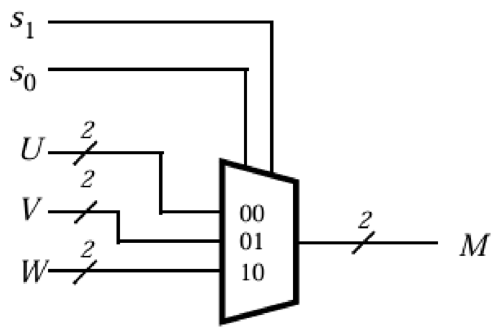


Figure 5 - Symbol Diagram for 2-bit 3-to-1 Multiplexer

Here, each input is two-bits wide (u, v, w), and is denoted by the 2 / along the input lines. The two-bit output, ‘m’, is selected based on whichever signal input, ‘s1’ or ‘s0’, is logic HIGH. In other words, the two active bits in any of the respective two-bit inputs will only be active in ‘m’ based on the logical combination of the ‘s1’ and ‘s0’ input. Below

Now, instead of writing a VHDL logic representation equation based on the above schematic, the behavioral coding style will be used, which is more reminiscent of programming languages such as Java, Python, or C++. The implementation of this coding style is expounded upon in the **Code Explanation Part A** section of the report, specifically regarding the *architecture* segment of the program.

Within the *entity* body of the VHDL file, three 2-bit inputs, two 1-bit inputs, and a 2-bit output must be instantiated. The first of which being ‘u’, which acts as a vector of magnitude 2. The second, input ‘v’ also with a size of 2, followed by the third 2-bit vector, ‘w’. The inputs ‘s1’ and ‘s0’ are simple scalar single bit inputs, and the output ‘leds’ also serves as a vector, of magnitude 2.

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 6**.

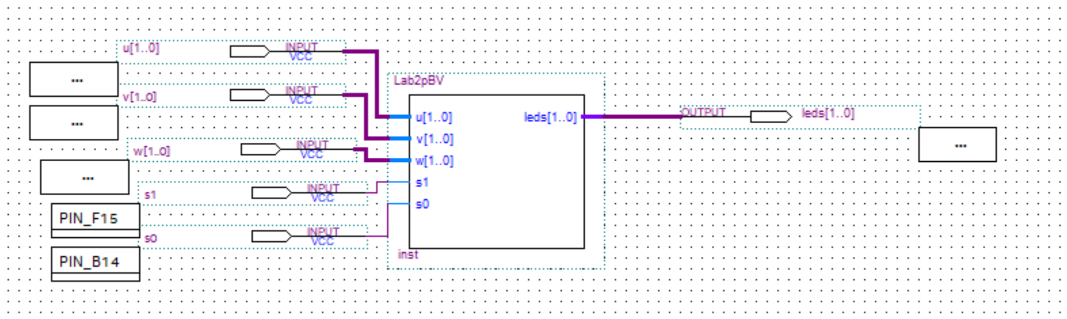
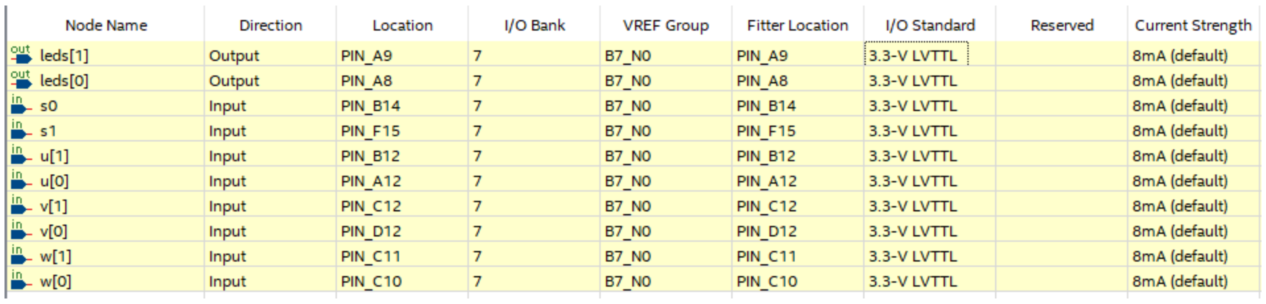


Figure 6 -Lab 2 Part B Schematic File

Note that the name of each respective input pin and output pin again matches the syntax within the symbol file. (i.e. u[1..0] denotes the vector *u* with values 1 -> 0). A successful connection to a vector value within the schematic diagram is indicated by a thicker line than found in single-bit pin connections.

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



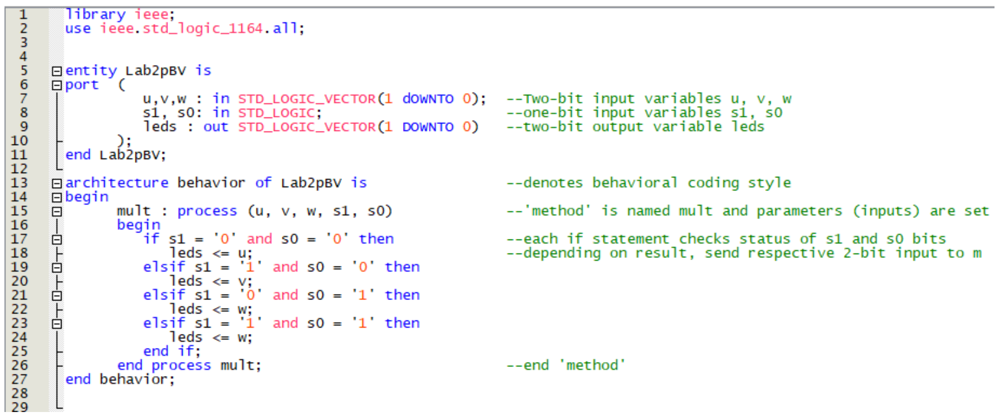
Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength
leds[1]	Output	PIN_A9	7	B7_N0	PIN_A9	3.3-V LVTTL		8mA (default)
leds[0]	Output	PIN_A8	7	B7_N0	PIN_A8	3.3-V LVTTL		8mA (default)
s0	Input	PIN_B14	7	B7_N0	PIN_B14	3.3-V LVTTL		8mA (default)
s1	Input	PIN_F15	7	B7_N0	PIN_F15	3.3-V LVTTL		8mA (default)
u[1]	Input	PIN_B12	7	B7_N0	PIN_B12	3.3-V LVTTL		8mA (default)
u[0]	Input	PIN_A12	7	B7_N0	PIN_A12	3.3-V LVTTL		8mA (default)
v[1]	Input	PIN_C12	7	B7_N0	PIN_C12	3.3-V LVTTL		8mA (default)
v[0]	Input	PIN_D12	7	B7_N0	PIN_D12	3.3-V LVTTL		8mA (default)
w[1]	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL		8mA (default)
w[0]	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL		8mA (default)

Figure 7 - Pinout Assignments for Lab 2 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, along with a logic waveform for each part of the experiment, rather than contributing additional information on the programming involved.

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 Lab2pBV is
port   (
         u,v,w : in STD_LOGIC_VECTOR(1 dOWNTO 0);   --Two-bit input variables u, v, w
         s1, s0: in STD_LOGIC;                       --one-bit input variables s1, s0
         leds : out STD_LOGIC_VECTOR(1 DOWNTO 0)    --two-bit output variable leds
      );
end Lab2pBV;

architecture behavior of Lab2pBV is                 --denotes behavioral coding style
begin
      mult : process (u, v, w, s1, s0)              --'method' is named mult and parameters (inputs) are set
      begin
         if s1 = '0' and s0 = '0' then              --each if statement checks status of s1 and s0 bits
            leds <= u;                               --depending on result, send respective 2-bit input to m
         elsif s1 = '1' and s0 = '0' then
            leds <= v;
         elsif s1 = '0' and s0 = '1' then
            leds <= w;
         elsif s1 = '1' and s0 = '1' then
            leds <= w;
         end if;
      end process mult;                              --end 'method'
end behavior;
```

Figure 8 – Image of Completed Code for Part B

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


entity Lab2pBV is
port      (
                        u,v,w : in STD_LOGIC_VECTOR(1 dOWNTO 0);      --Two-bit input variables u, v, w
                        s1, s0: in STD_LOGIC;                         --one-bit input variables s1, s0
                        leds : out STD_LOGIC_VECTOR(1 DOWNTO 0)       --two-bit output variable leds
                );
end Lab2pBV;

architecture behavior of Lab2pBV is                                   --denotes behavioral coding style
begin
                mult : process (u, v, w, s1, s0)      --'method' is named mult and parameters (inputs) are set
                begin
                        if s1 = '0' and s0 = '0' then         --each if statement checks status of s1 and s0 bits
                                leds <= u;
                --depending on result, send respective 2-bit input to m
                        elsif s1 = '1' and s0 = '0' then
                                leds <= v;
                        elsif s1 = '0' and s0 = '1' then
                                leds <= w;
                        elsif s1 = '1' and s0 = '1' then
                                leds <= w;
                        end if;
                end process mult;
        --end 'method'
end behavior;
```

The entity portion of the code, where the inputs and outputs are created, is described above in **Part 2**.

The architecture section, which denotes a *behavioral* coding style, uses an approach more reminiscent of traditional programming languages, rather than a VHDL interpretation of logic. Thus, a fundamental understanding as to the function of the circuit is critical in programming in *behavioral* style. It is known that the ‘s1’ and ‘s0’ signals must essentially act as “activators” for the ‘u’, ‘v’, and ‘w’ input signals. Thus, a series of if statements works well enough for this scenario. Note the syntax used in this coding style, where a ‘method’ is created which takes all the inputs as parameters to be used. Each if statement checks the status of ‘s1’ and ‘s0’ bits until all potential scenarios have been checked. Within each statement, the respective 2-bit input is sent to the ‘m’ output signal.

The *behavioral* coding style is unique due to its exclusive use of process statements.

VALIDATION OF DATA

Below are the waveforms for both Part A and Part B of the experiment, along with images indicating successful code logic workflow, and logic board programming.

Waveforms

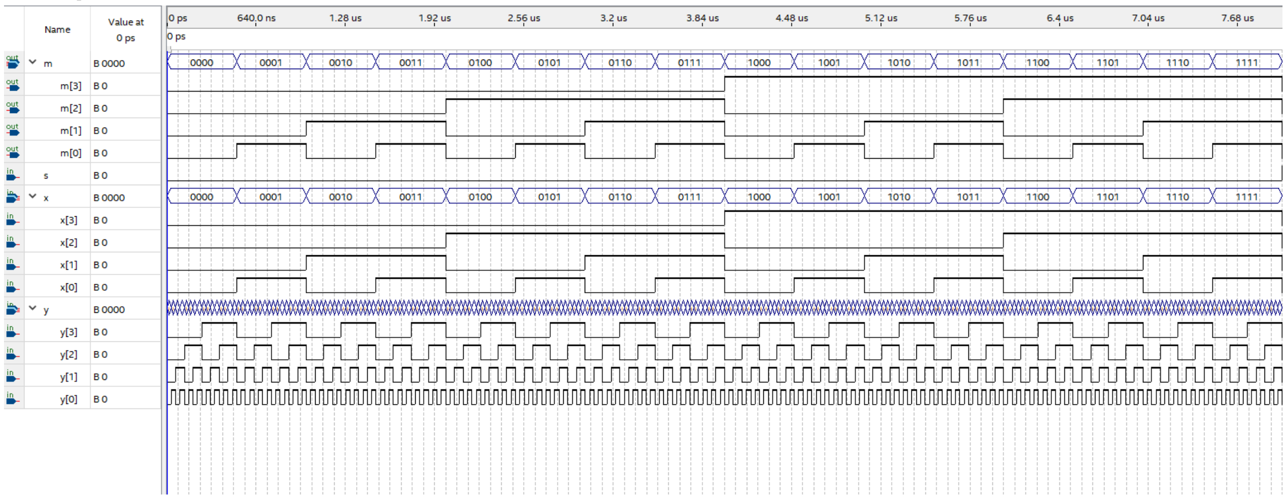


Figure 9 - Logic Waveform for 4-bit 2-to-1 Multiplexer

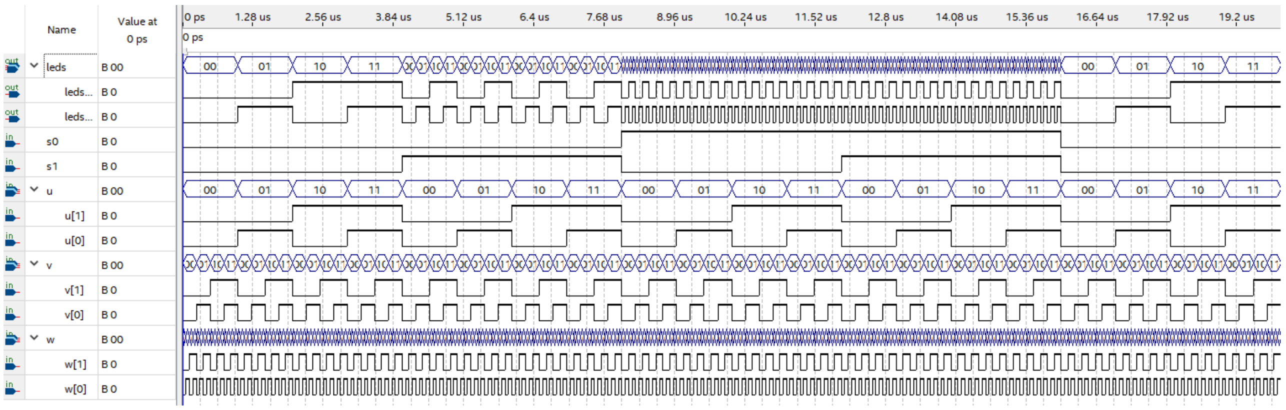


Figure 10 - Logic Waveform for 2-bit 3-to-1 Multiplexer

Bench Analysis Part A

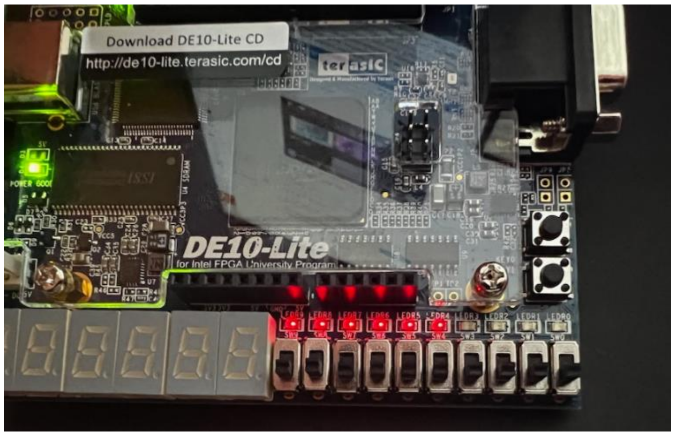

Download DE10-Lite CD
http://de10-lite.terasic.com/cd
terasic
DE10-Lite
for Intel FPGA University Program

Figure 11 – s = 0 | X[3…0] = 0 | Y[3..0] = 0

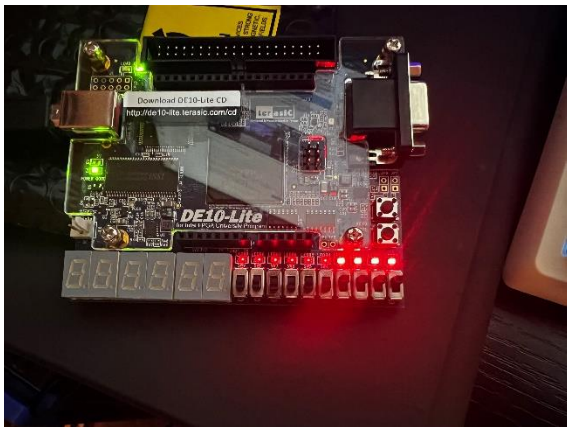


Figure 12 - s = 0 | X[3…0] = 1 | Y[3..0] = 0

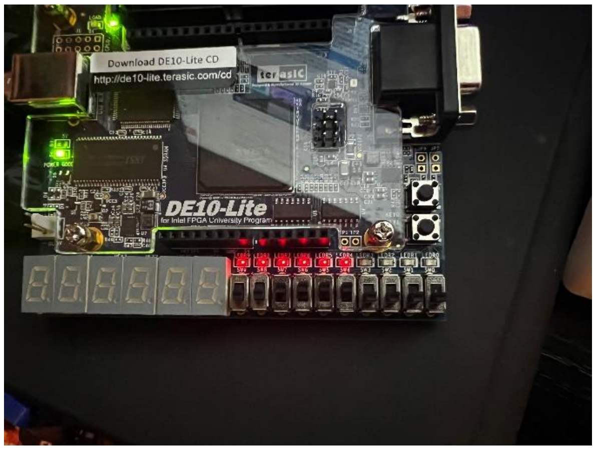


Figure 13 - s = 1 | X[3…0] = 0 | Y[3..0] = 0

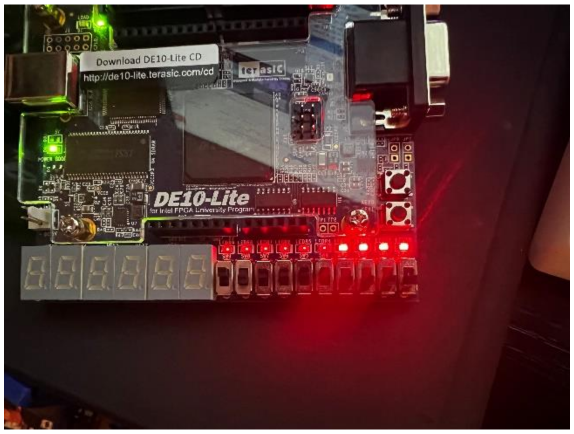


Figure 14 - s = 1 | X[3…0] = 0 | Y[3..0] = 1

Bench Analysis Part B

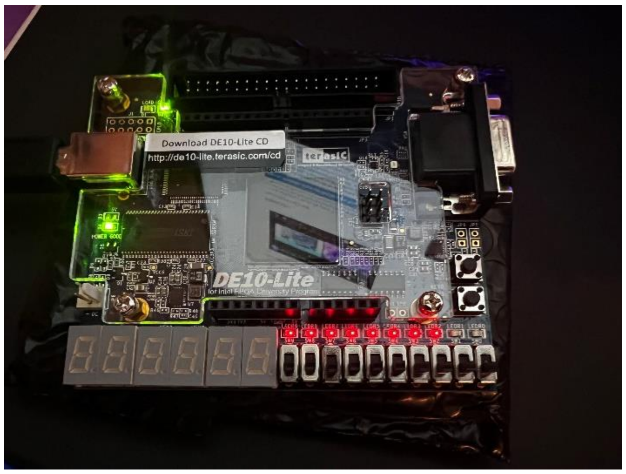


Figure 15 – s1,s0 = 00 | u= 00 | v = 00 | w = 00

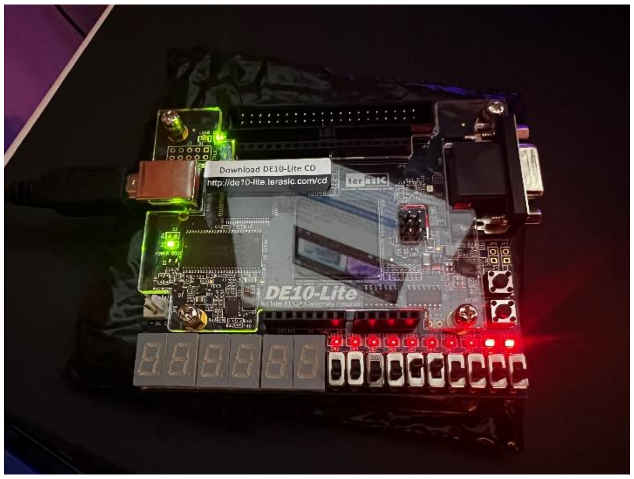


Figure 16 – s1,s0 = 00 | u= 11 | v = 00 | w = 00

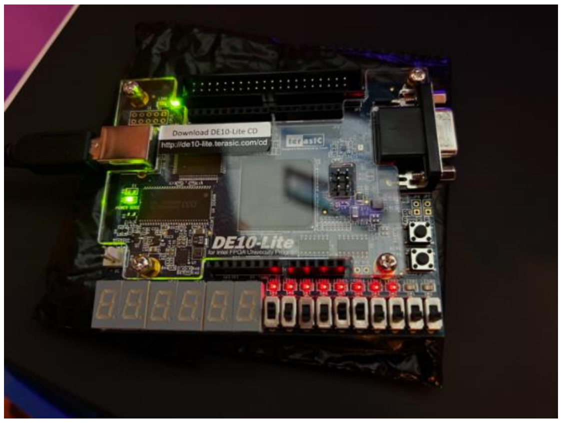


Figure 17 – s1,s0 = 10 | u= 00 | v = 00 | w = 00

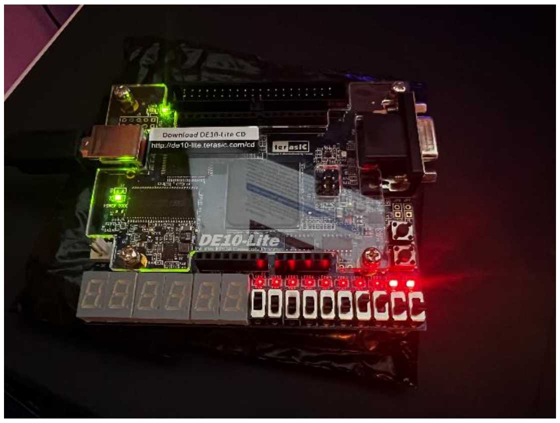


Figure 18 – s1,s0 = 10 | u= 00 | v = 11 | w = 00

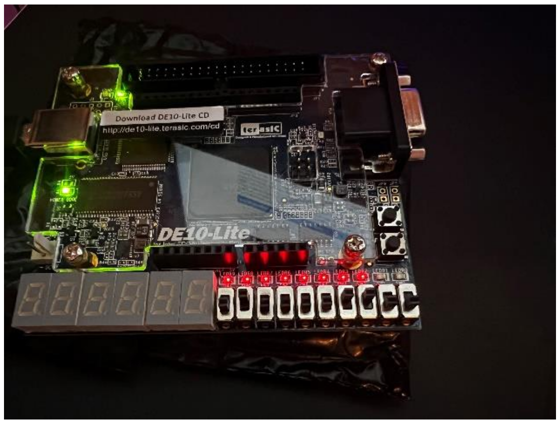


Figure 19 – s1,s0 = 01 | u= 00 | v = 00 | w = 00

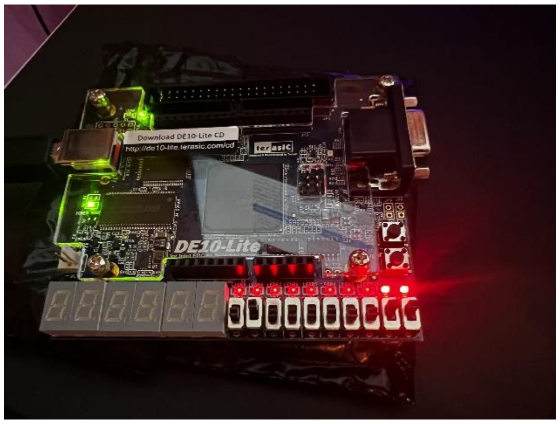


Figure 20 – s1,s0 = 01 | u= 00 | v = 00 | w = 11

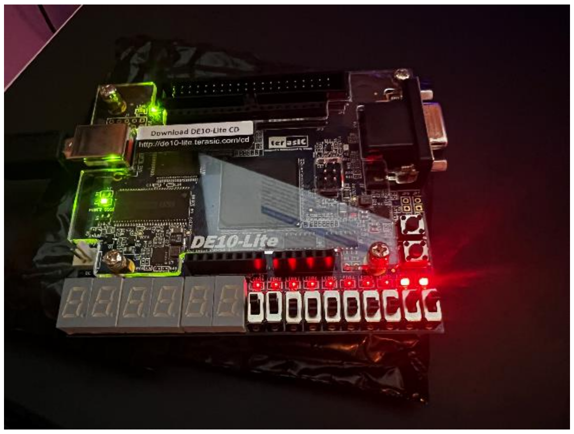


Figure 21 – s1,s0 = 11 | u= 00 | v = 00 | w = 11

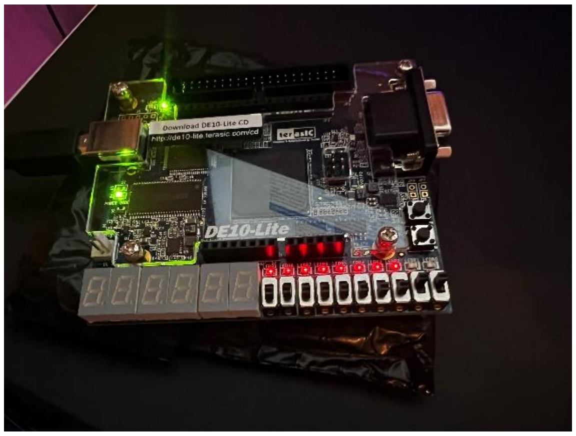


Figure 22 – s1,s0 = 11 | u= 00 | v = 00 | w = 00

Located in each figure caption is the logical output result for each bit pair and potential combination. The final two LEDs on the development board represent the output.

CONCLUSION

The experiment was conducted successfully and efficiently demonstrated programming various multiplexers in both a behavioral and dataflow coding style. If the lab were to be conducted again, more intricate, detailed circuits could be analyzed to be further exposed to these new coding styles in a VHDL programming environment. Also, it would be beneficial to observe the third common programming style, that being structural coding, to obtain knowledge on all three of these techniques. It should be noted, however, that this study is a predecessor to an experiment where that very topic is introduced, and it is known to exhibit more difficult behavior and is therefore more difficult to grasp, therefore there may be good reason as to the absence of it within this study.

REFERENCES

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