Lab 1 – Introduction to VHDL and ASM Design

CEG 3155 – Digital Systems II

Fall 2023

School of Electrical Engineering and Computer Science

University of Ottawa

Harry Le 300229371

Siva Senthilkumaran 300230364

Submission Date: October 3, 2023

Problems and Objectives:

The primary objective of this lab was to successfully develop a functional circuit using the ASM design methodology and the VHDL programming language. The idea is to demonstrate how ASM and VHDL can be used to design robust and versatile circuits given a simple purpose and condition, saving time and money as well as reducing potential errors in design. Additionally, our design and solution to the problem given to us incorporates Register Transfer Logic (RTL), a high-level design adopted by many designs, using only registers and devices that modify data from registers.

The design in question is for a light display controller for a horizontal array of eight lights. When the left switch is triggered, a light will move from right to left, and vice versa for the right switch. If both switches are triggered, lights will move in both left and right directions, and if neither are triggered, neither light will display until a switch is triggered. There is an additional greset switch used to initialize the display, and to ensure all registers can be globally reset.

ASM Designs:

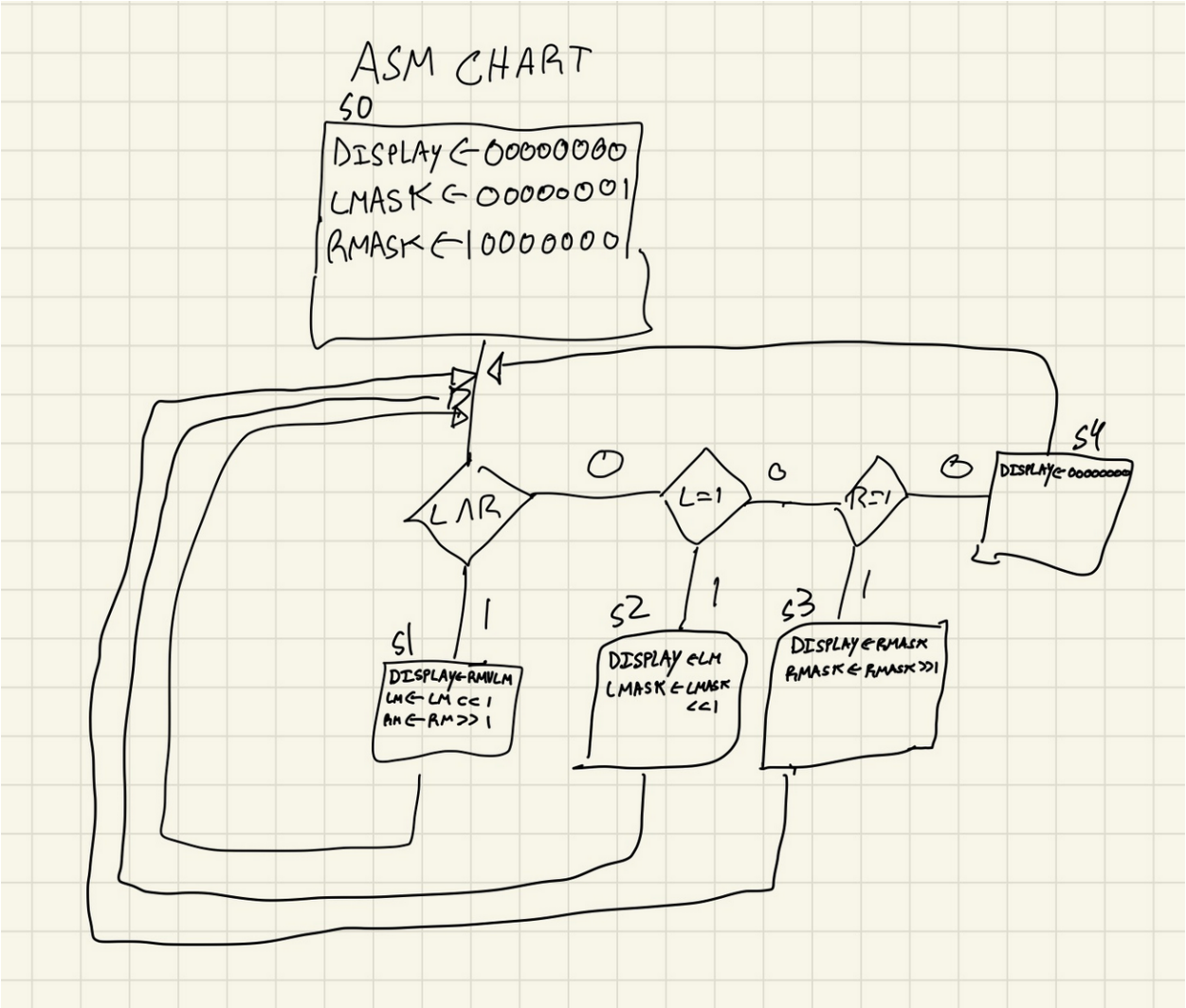

Figure 1: ASM Diagram depicting the function of the light controller based off the pseudocode

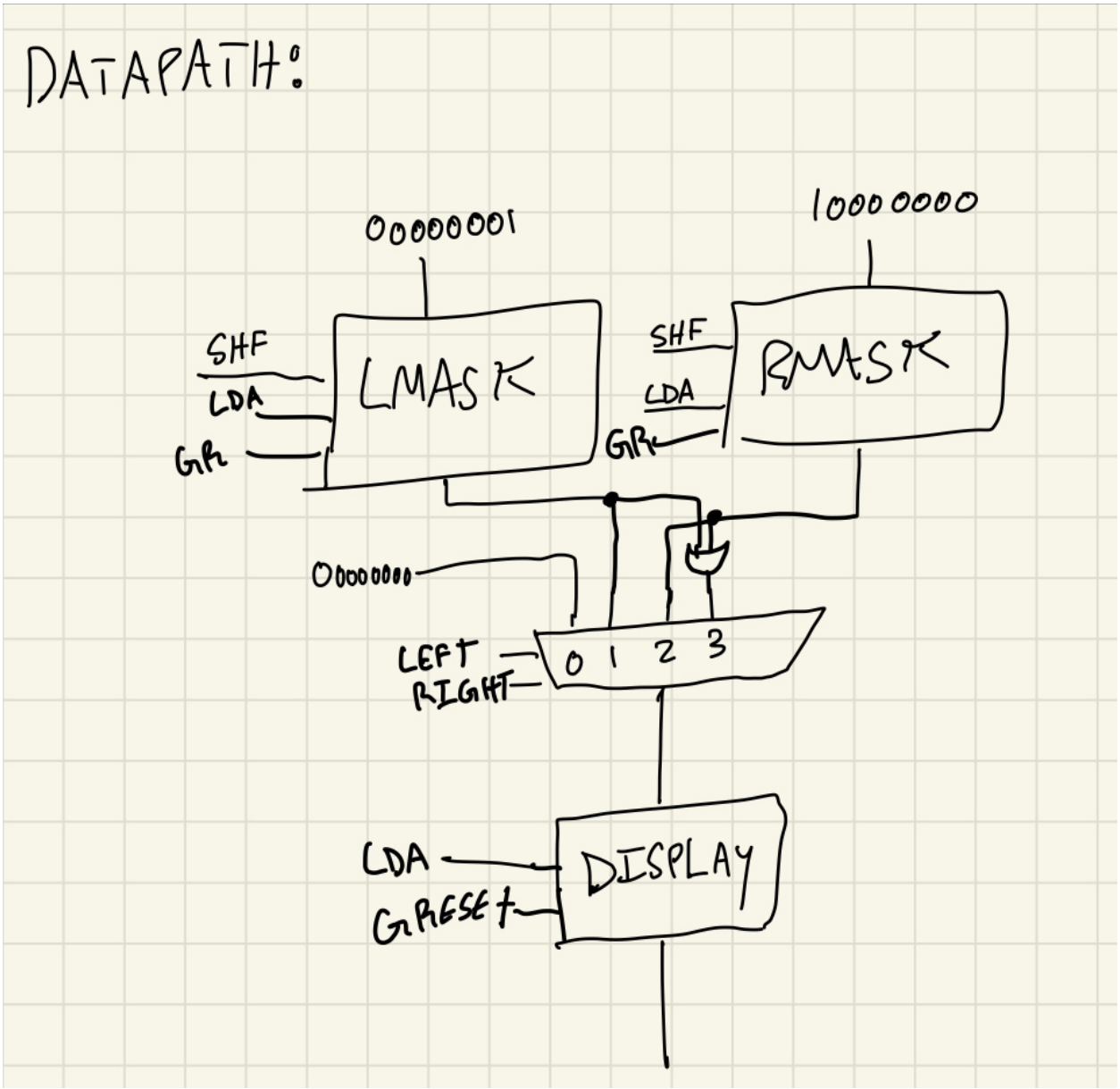

Figure 2: Datapath based off of the ASM Design, using Registers and a Multiplexer

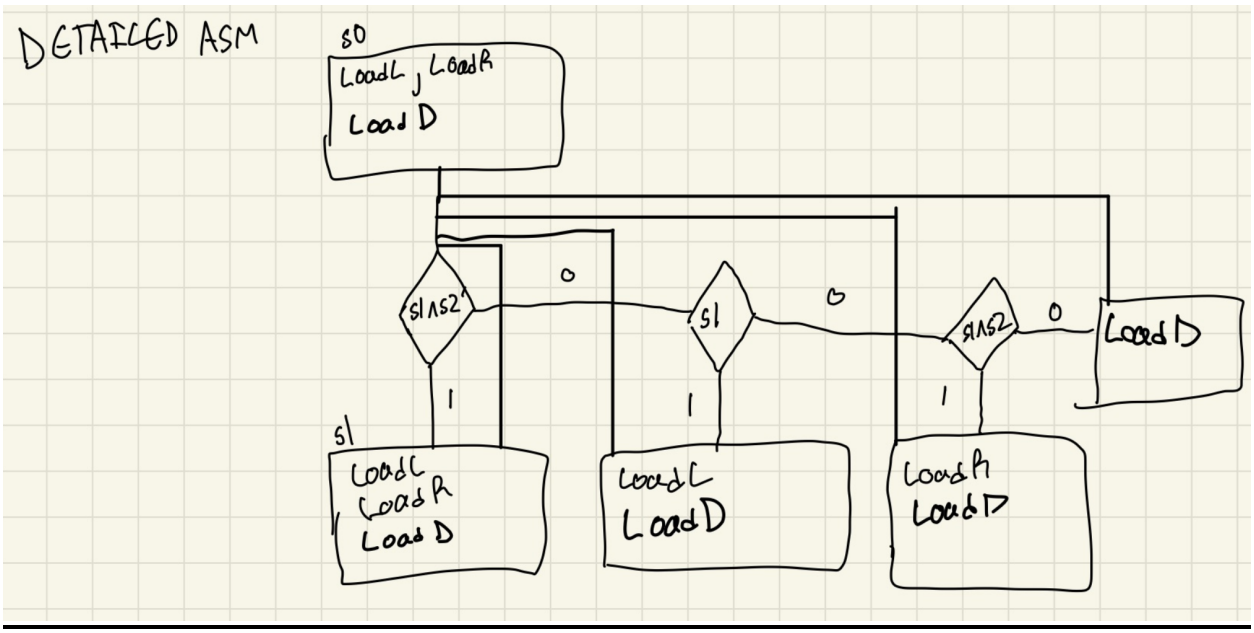

Figure 3: Detailed ASM Diagram for the design of the Control Path

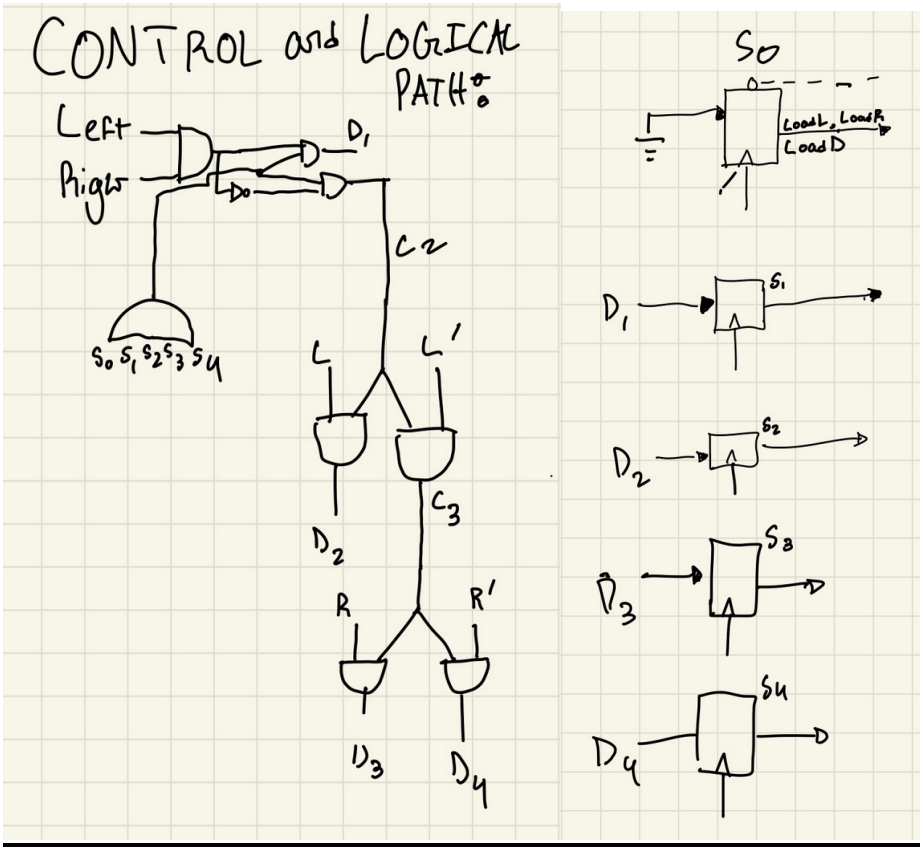

Figure 4: Control Logic for the Light Display Controller, using the Detailed ASM Diagram

VHDL Code:

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

entity leftshift8bitregister is
	port(
		i_reset,i_leftshift: in std_logic;
		i_clock: in std_logic;
		i_value: in std_logic_vector(7 downto 0);
		o_value	: out std_logic_vector(7 downto 0));
end leftshift8bitregister;

architecture rtl of leftshift8bitregister is
	signal int_value: std_logic_vector (7 downto 0);


	component enARdFF_2
	port(
		i_clock : in std_logic;
		i_d : in std_logic;
		o_q,o_qBar: out std_logic);
	end component;

	begin
		--component instantiation
		  b7: enARdFF_2
		      port map(
		      i_d => i_value(7) or (int_value(6) and i_leftshift),

		      i_clock => i_clock,
		      o_q => int_value(7));

		      b6: enARdFF_2
		      port map(
		      i_d => (i_value(6)) or (int_value(5) and i_leftshift),


		      i_clock => i_clock,
		      o_q => int_value(6));

		      b5: enARdFF_2
		      port map(
		      i_d => (i_value(5)) or (int_value(4) and i_leftshift),


		      i_clock => i_clock,
		      o_q => int_value(5));

		      b4: enARdFF_2
		      port map(
		      i_d => (i_value(4)) or (int_value(3) and i_leftshift),


		      i_clock => i_clock,
		      o_q => int_value(4));
```

```
                    b3: enARdFF_2
                    port map(
                    i_d => (i_value(3)) or (int_value(2) and i_leftshift),


                    i_clock => i_clock,
                    o_q => int_value(3));

                    b2: enARdFF_2
                    port map(
                    i_d => (i_value(2)) or (int_value(1) and i_leftshift),


                    i_clock => i_clock,
                    o_q => int_value(2));

                    b1: enARdFF_2
                    port map(
                    i_d => (i_value(1)) or (int_value(0) and i_leftshift),


                    i_clock => i_clock,
                    o_q => int_value(1));

                    b0: enARdFF_2
                    port map(
                    i_d => (i_value(0)) and not(i_leftshift),


                    i_clock => i_clock,
                    o_q => int_value(0));
                    --concurrent signals


                    --Output Driver
                    o_value <= int_value;
end rtl;
```

Figure 5: VHDL Code for 8-bit left shift register

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

entity rightshift8bitregister is
      port(
            i_reset,i_rightshift: in std_logic;
            i_clock: in std_logic;
            i_value: in std_logic_vector(7 downto 0);
            o_value     : out std_logic_vector(7 downto 0));
end rightshift8bitregister;

architecture rtl of rightshift8bitregister is
      signal int_value: std_logic_vector (7 downto 0);
```

```
component enARdFF_2
port(
      i_clock : in std_logic;
      i_d : in std_logic;
      o_q,o_qBar: out std_logic);
end component;

begin
      --component instantiation
        b7: enARdFF_2
            port map(
            i_d => i_value(7) and not(i_rightshift),
            i_clock => i_clock,
            o_q => int_value(7));

            b6: enARdFF_2
            port map(
            i_d => (i_value(6)) or (int_value(7) and i_rightshift),


            i_clock => i_clock,
            o_q => int_value(6));

            b5: enARdFF_2
            port map(
            i_d => (i_value(5)) or (int_value(6) and i_rightshift),


            i_clock => i_clock,
            o_q => int_value(5));

            b4: enARdFF_2
            port map(
            i_d => (i_value(4)) or (int_value(5) and i_rightshift),


            i_clock => i_clock,
            o_q => int_value(4));

            b3: enARdFF_2
            port map(
            i_d => (i_value(3)) or (int_value(4) and i_rightshift),


            i_clock => i_clock,
            o_q => int_value(3));

            b2: enARdFF_2
            port map(
            i_d => (i_value(2)) or (int_value(3) and i_rightshift),


            i_clock => i_clock,
            o_q => int_value(2));

            b1: enARdFF_2
```

```
                port map(
                i_d => (i_value(1)) or (int_value(2) and i_rightshift),


                i_clock => i_clock,
                o_q => int_value(1));

                b0: enARdFF_2
                port map(
                i_d => (i_value(0)) or (int_value(1) and i_rightshift),


                i_clock => i_clock,
                o_q => int_value(0));



                --Output Driver
                o_value <= int_value;
end rtl;
```

Figure 6: VHDL Code for 8-bit right shift register

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

entity enARdFF_2 is
      port(
            i_clock : in std_logic;
            i_d : in std_logic;
            o_q,o_qBar: out std_logic);
end enARdFF_2;

architecture rtl of enARdFF_2 is
      SIGNAL int_q, int_qBar : STD_LOGIC;
 SIGNAL int_d, int_dBar : STD_LOGIC;
 SIGNAL int_notD, int_notClock : STD_LOGIC;


      component enabledSRLatch
      port(
             i_set, i_reset : IN STD_LOGIC;
            i_enable : IN STD_LOGIC;
            o_q, o_qBar : OUT STD_LOGIC);
      end component;

      begin
            --concurrent signal
            masterLatch: enabledSRLatch
             PORT MAP ( i_set => i_d,
             i_reset => int_notD,
             i_enable => int_notClock,
             o_q => int_q,
             o_qBar => int_qBar);
             slaveLatch: enabledSRLatch
```

```
            PORT MAP ( i_set => int_q,
            i_reset => int_qBar,
            i_enable => i_clock,
            o_q => o_q,
            o_qBar => o_qBar);

                --Output Driver
                int_notD <= not(i_d);
                int_notClock <= not(i_clock);

end rtl;
```

Figure 7: VHDL Code for Enabled AR D-Flip Flop

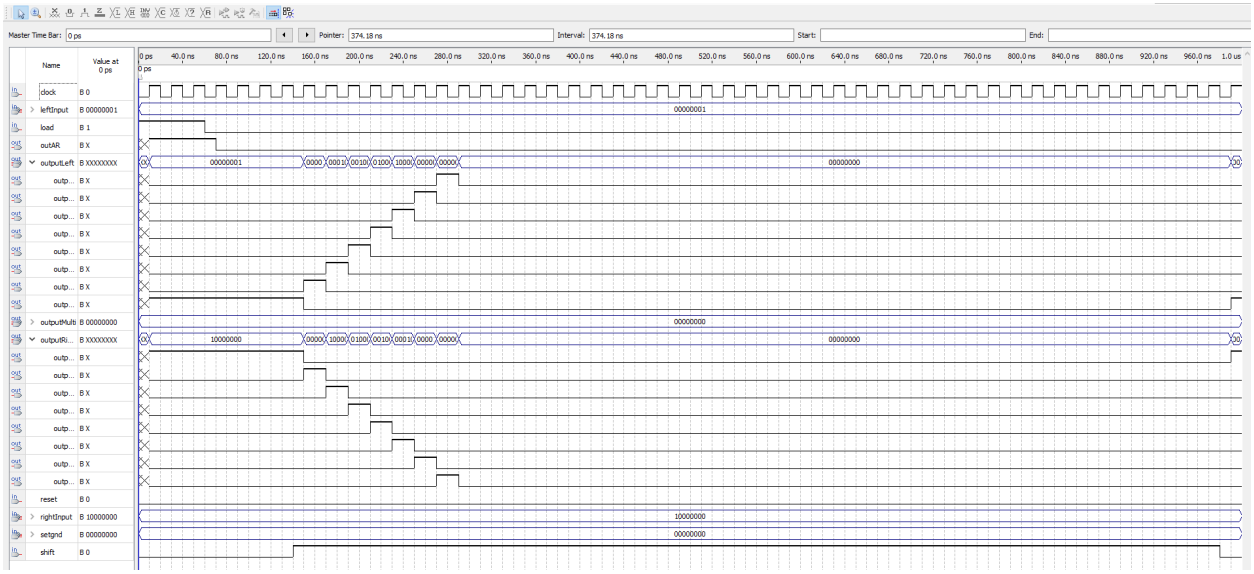

Figure 8: Timing Diagrams for the DFF, L/R Shift Registers (from top to bottom).

All FFs and Registers load and clear on rising clock edges.

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

entity bit1comparator1on2 is
	port(
		i_value1, i_value2: in std_logic;
		i_clock: in std_logic;
		o_gt,o_lt,o_eq	: out std_logic);
end bit1comparator1on2;

architecture rtl of bit1comparator1on2 is
	signal int_gt,int_lt,int_eq: std_logic;

	begin
		--concurrent signal
		 int_gt <= i_value1 and not(i_value2);
		 int_lt <= not(i_value1) and i_value2;
		 int_eq <= not(i_value1 xor i_value2);
		--Output Driver
		o_gt <= int_gt;
		o_lt <= int_lt;
		o_eq <= int_eq;


end rtl;
```

Figure 9: VHDL Code for comparator

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

entity bit8register is
	port(
		i_reset, i_load: in std_logic;
		i_clock: in std_logic;
		i_value: in std_logic_vector(7 downto 0);
		o_value	: out std_logic_vector(7 downto 0));
end bit8register;

architecture rtl of bit8register is
	signal int_value: std_logic_vector (7 downto 0);


	component enARdFF_2
	port(
		i_reset : in std_logic;
		i_d,i_enable,i_clock : in std_logic;
		o_q : out std_logic);
	end component;

	begin
		--component instantiation
		   b7: enARdFF_2
		      port map(
		      i_d => i_value(7) and i_load,
```

```
    i_reset => i_reset,
    i_enable => i_load,
    i_clock => i_clock,
    o_q => int_value(7));

b6: enARdFF_2
    port map(
    i_d => i_value(6) and i_load,
    i_reset => i_reset,
    i_enable => i_load,
    i_clock => i_clock,
    o_q => int_value(6));

b5: enARdFF_2
    port map(
    i_d => i_value(5) and i_load,
    i_reset => i_reset,
    i_enable => i_load,
    i_clock => i_clock,
    o_q => int_value(5));

b4: enARdFF_2
    port map(
    i_d => i_value(4) and i_load,
    i_reset => i_reset,
    i_enable => i_load,
    i_clock => i_clock,
    o_q => int_value(4));

    b3: enARdFF_2
    port map(
    i_d => i_value(3) and i_load,
    i_reset => i_reset,
    i_enable => i_load,
    i_clock => i_clock,
    o_q => int_value(3));

    b2: enARdFF_2
    port map(
    i_d => i_value(2) and i_load,
    i_reset => i_reset,
    i_enable => i_load,
    i_clock => i_clock,
    o_q => int_value(2));

    b1: enARdFF_2
    port map(
    i_d => i_value(1) and i_load,
    i_reset => i_reset,
    i_enable => i_load,
    i_clock => i_clock,
    o_q => int_value(1));

    b0: enARdFF_2
    port map(
    i_d => i_value(0) and i_load,
    i_reset => i_reset,
```

```
                    i_enable => i_load,
                    i_clock => i_clock,
                    o_q => int_value(0));

                    --Output Driver
                          o_value <= int_value;

end rtl;
```

Figure 10: VHDL Code for 8-bit register holding light display

Light Display Controller Design:

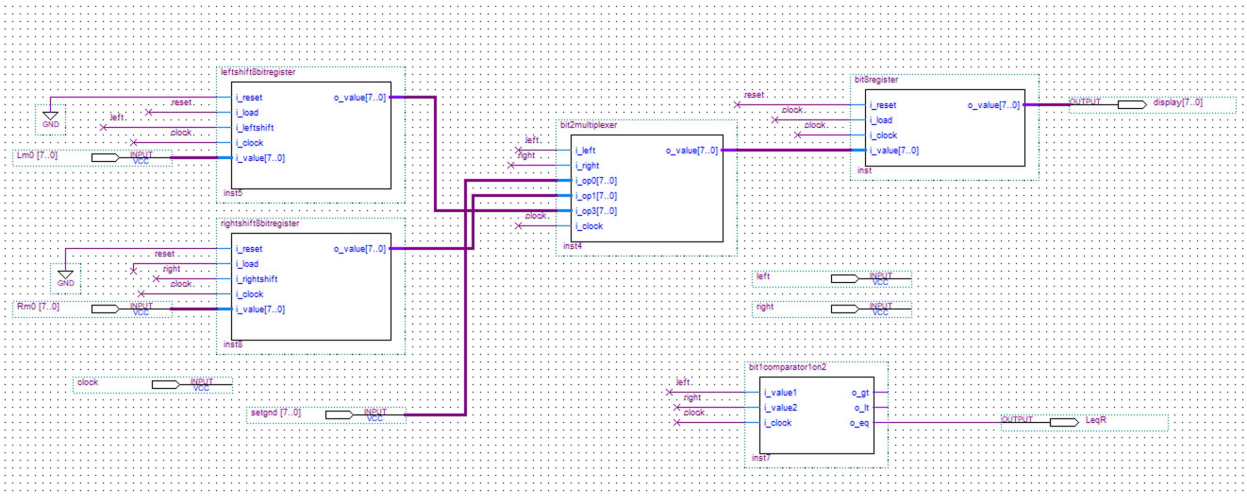

Figure 11: Placeholder for Light Display Controller Schematic

Solution:

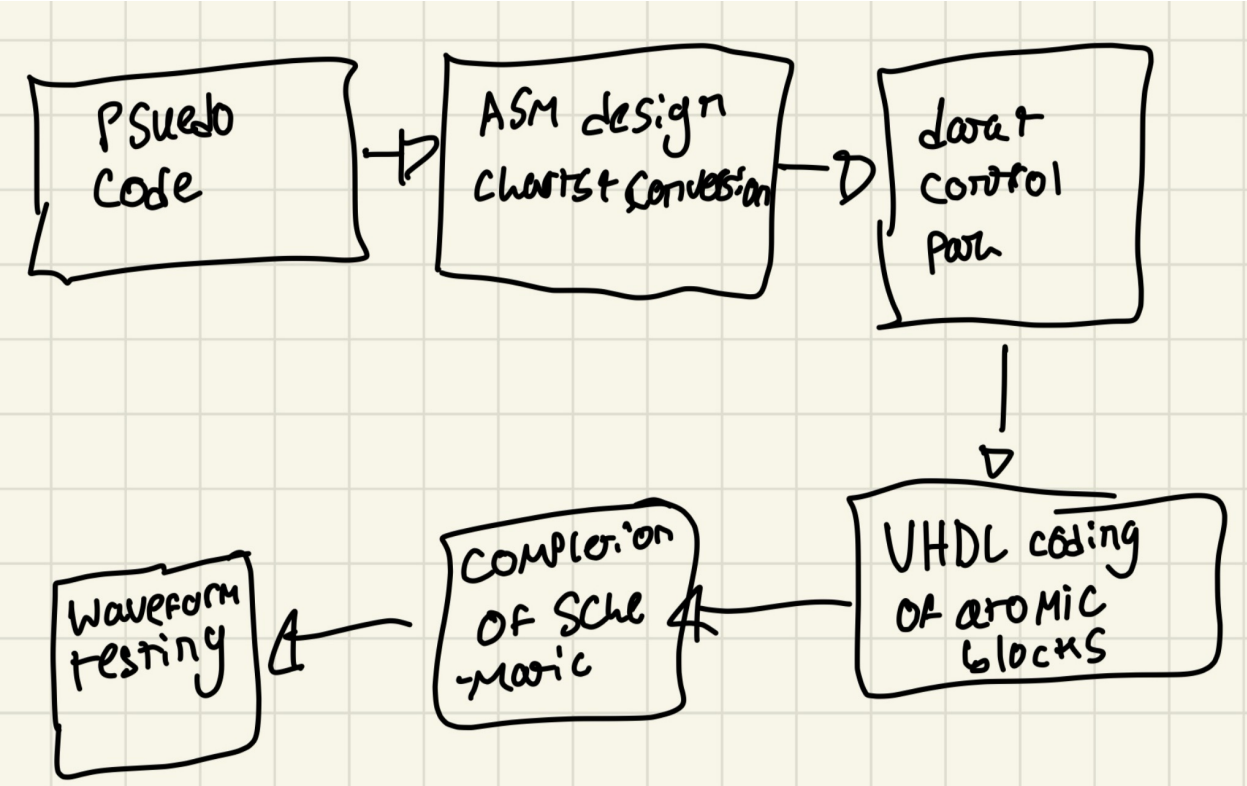

Figure 12: Flowchart for Solution

Using the provided pseudocode, we designed the datapath and control logic with ASM and implemented it in VHDL. We programmed an Asynchronous D-Flip Flop with Enable, a 1-bit comparator, two registers with right/left shift capabilities, and an 8 bit register to store the light display and positioning, all in structural VHDL code. We then created a schematic for the data and control path using these components, designing the circuit's top level graphically, while adding atomic units and blocks to the circuit that were designed in VHDL.

Design Challenges:

Our primary design challenge in this laboratory was testing. We had several issues with Quartus II and running our compiled project and VHDL code. Namely, Quartus would not run on our personal devices properly, and on the lab computers we spent a lot of time changing settings in our VHDL files and waveforms to even get the project to compile and display properly. An example of this would be the ModelSim simulation option in the waveform generator. It took us time to figure out that we needed to switch the simulator to Quartus II for our waveforms to even finish compiling. We additionally encountered several errors while attempting to build and compile the project on different personal computers, which made the issues even harder to rectify. As a result, we were unable to properly verify our code until the last laboratory session, and most of that session was spent trying to fix issues in the code.

Additionally, one related issue was getting familiar with VHDL/ASM and rectifying mistakes in such a limited timeframe. Developing code for a programming language we had little experience with was quite challenging. While the syntax is not too different from a typical OOP programming language, the main challenge is getting accustomed to the “hardware mindset”, as many principles in OOP and software design simply do not apply, such as the principle of concurrency (everything running simultaneously at once). Additionally, the ASM methodology is quite detailed and meticulous, and took some time to properly implement in this project. We spent most of our lab sessions writing and developing the control and data path through ASM, as we were still familiarizing ourselves with the content to begin with. This was one of the very first ASM data and control paths we ever designed, as we had only done a handful before during class and as part of our first assignment.

Finally, despite having a complete block schematic ready to test, our schematic did not work because we coded our shift registers incorrectly, not accounting for concurrency properly. We developed testbenches to check every possible issue with the code until we were able to get the right shift register working, as demonstrated by the waveform in figure 8. We were also able to fix this issue in the left shift register as demonstrated in the waveform in figure 6. However, we were ultimately unable to fix the issues with the final light display controller and could not demo our work.

Conclusion:

Despite being unable to demo a finished light display controller, our work still demonstrated an understanding of the ASM methodology and ability to code in VHDL. We were able to develop a viable data and control path using the design steps of ASM, and we were also able to code the atomic modules and provide working waveform simulations as proof of their validity, as well as create the graphical structure of the light display controller. Although we were ultimately unsuccessful in our programming implementations, we still produced results that demonstrated our project's potential capabilities.

We will take steps to prevent this in future labs and the final project. By taking advantage of the laboratory's scheduled hours to test our code on the FPGA in advance, we can catch potential errors that would cost time in the actual lab session. Additionally, this experience serves as an indicator that we may need to further study the contents of the course, as future labs will most likely depend on already developed experience with ASM and VHDL. Ultimately, we will use this as a learning experience, and a step towards success in our future labs.