Adrian Korwel
Fernando Zepeda
ECE 218 - L04
Nader Alnatsheh

Experiment 8 and 9
8) Intro to FPGAs and VHDL Programming
9) Design of Code Converter, Adders, and Traffic Light Controller using FPGA

October 31 2023, Intro to FPGAs and VHDL Programming
November 14th 2023, Design of Code Converter, Adders, and Traffic Light Controller using FPGA

Due date of laboratory report:
November 28th, 2023

Abstract:

My contribution to experiment 8 was setting up the Vivado software for the FPGA's, and also constructing the breadboard setup with my partner Fernando. For experiment 9, I contributed by setting up the Vivado software for the Traffic light controllers and then running the simulation to simulate light controllers.

Introduction:

The motivation behind the experiments I've done was to always make sure I was able to take my design and put it on the breadboard. I've noticed early on that I wasn't able to take the breadboard design and apply it on the breadboard because I didn't understand simple things like grounding, powering a chip, and even how to take a wire and put it into the input of an AND gate and take the output of that. I was able to overcome these challenges by looking up tutorials on youtube of people drawing their breadboard design and applying it onto their breadboard and it started to make a lot more sense after watching those videos.

Background / History:

A Field-Programmable Gate Array, is a programmable circuit designed to be configured by a user. It consists of an array of programmable logic blocks. The FPGA was first introduced by Ross Freeman in the 1980s. The first FPGA, known as the XC2064, was manufactured by Xilinx in 1985. Application-Specific Integrated Circuit, is a chip designed for a specific application. The difference between FPGA and ASIC is in their programmability. FPGA can be reprogrammed for different tasks and ASIC is designed for a specific purpose. The main languages used to program an FPGA are, (HDLs), such as Verilog and VHDL to program

FPGAs. Verilog is known for its concise syntax. VHDL is more verbose but offers strong type-checking and a better structure.

Vivado Design Suite is a software package developed by Xilinx for FPGA design. It uses all stages of the FPGA design flow, from designing and simulating the hardware in high-level languages to generating the final bit-stream file for configuration. FPGA design flow begins with creating design source files using HDLs, then by synthesis, implementation, and generating a bit-stream file. Behavioral simulation in the FPGA flow is for verifying the functionality of the design before synthesis. Vivado supports behavioral simulation by letting the users simulate the process of their designs. This makes sure that there will be a more efficient FPGA design process.

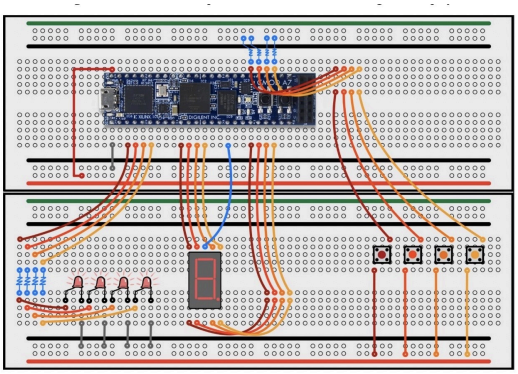
Experiment 1:

Theory:
The theoretical basis of this experiment is in the Field-Programmable Gate Arrays (FPGAs). FPGAs are integrated circuits which are designed by featuring an array of programmable logic blocks. Unlike methods using TTLs, FPGAs give an advantage of easy configuration by modifying the code instead of the circuit itself.

○ Procedure:

- First open up the Schematic Setup for FPGA labs on the blackboard
- Look at the figure 8.8 for experiment 8
- Setup your breadboard just like in the figure
- Press down your FPGA on the right side of your breadboard
- Next press and put your buttons on the left bottom side of your breadboard
- Next press and put your timer right above the buttons on your breadboard
- Grab a wire and power all of your buttons and your timer
- Next grab a lot of wires and follow the schematic setup given to you on figure 8.8 for experiment 8

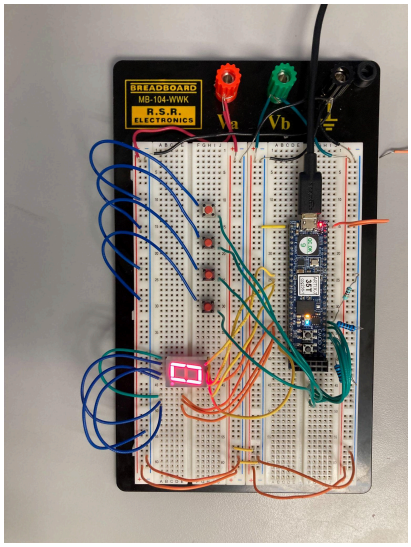
○ Breadboard Layout/Schematic:

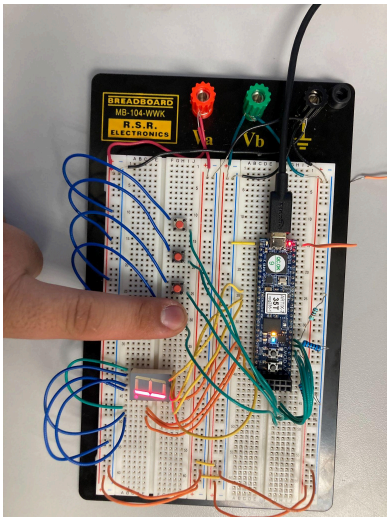


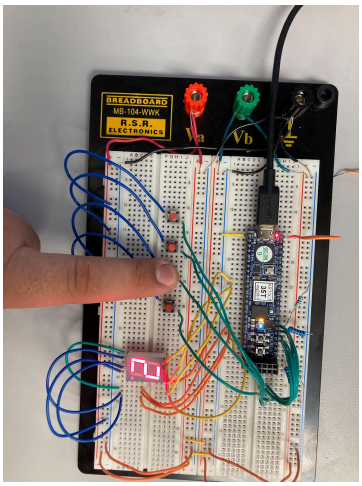
○ All apparatus:

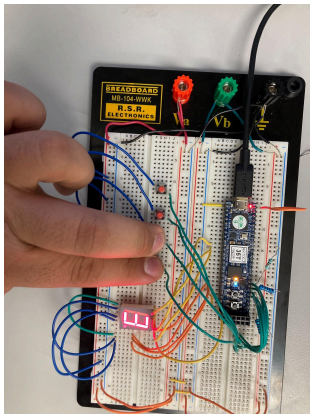
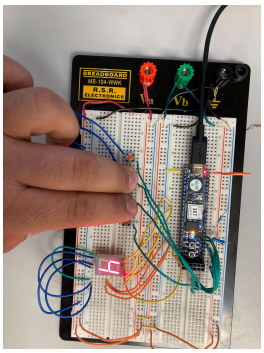
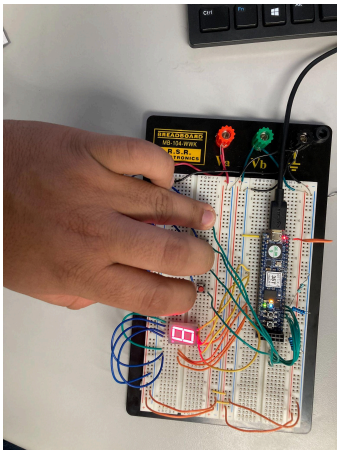
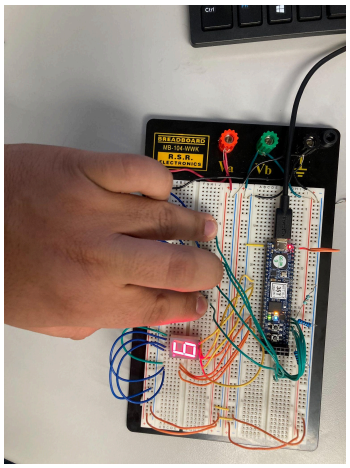
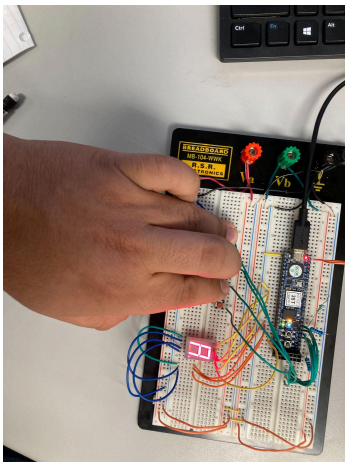
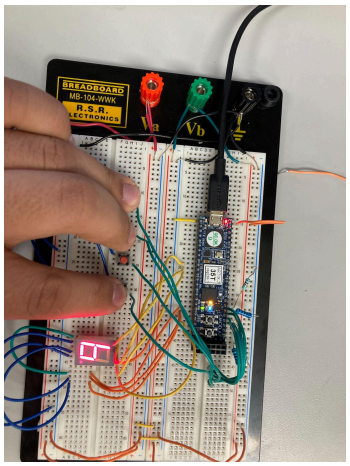
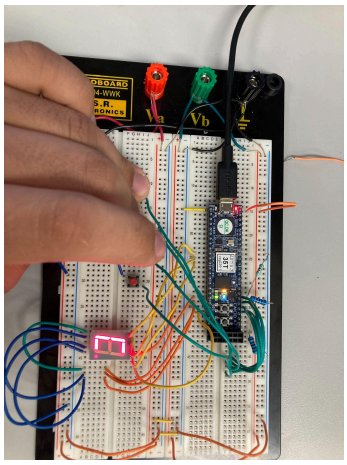
BreadBoard R.S.R Electronics: MB-104-WWK
CMOD A7 35T
Xilinx Artrix-7 FPGA
Seven-Segment Display
Pushbuttons

○ Results: //I have completed from 0-9 and A-F but you said to only put a couple of pictures for the results.







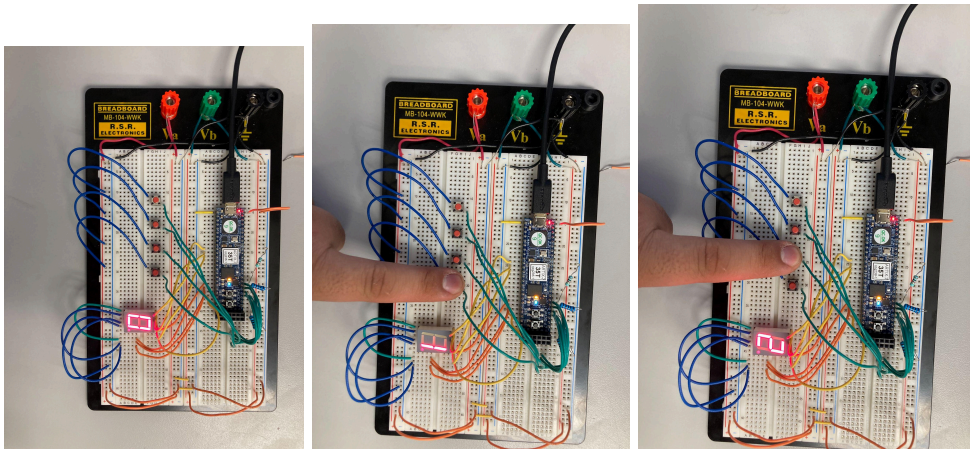








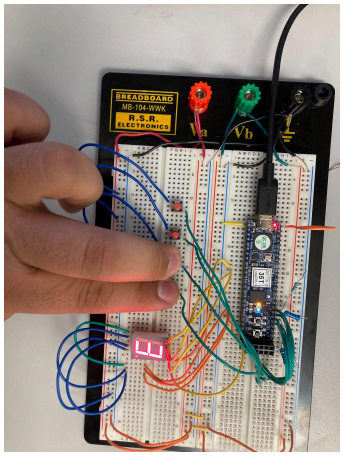
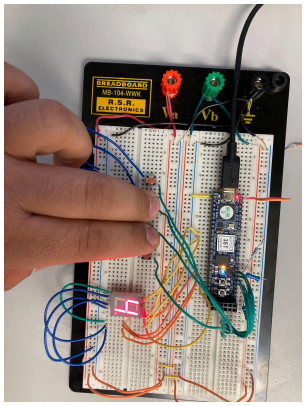
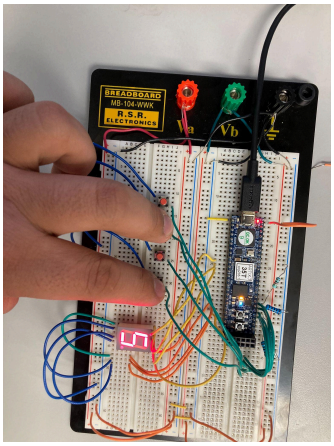
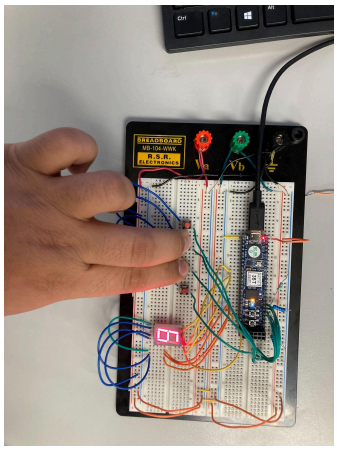
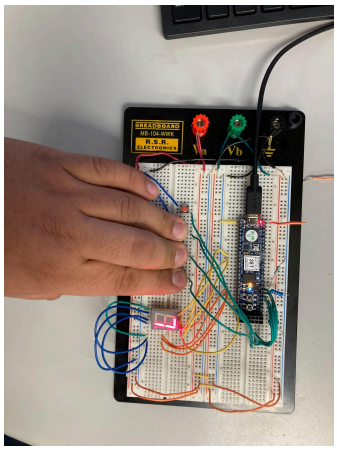
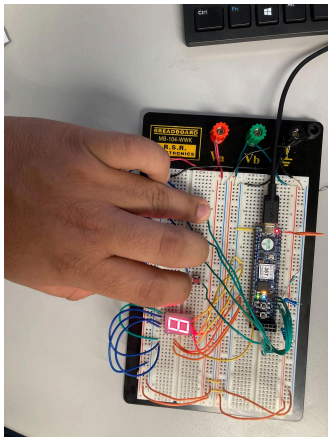
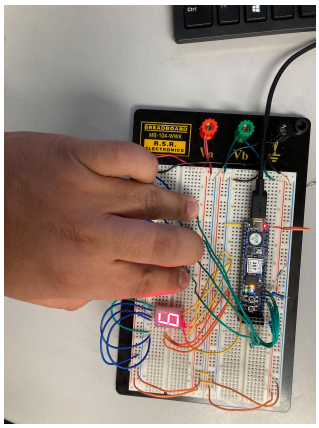
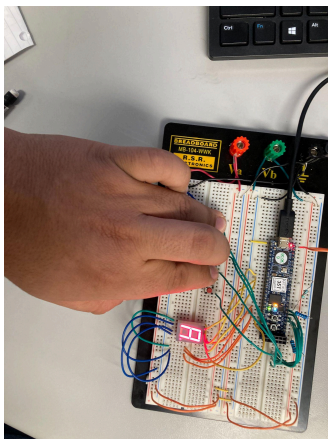
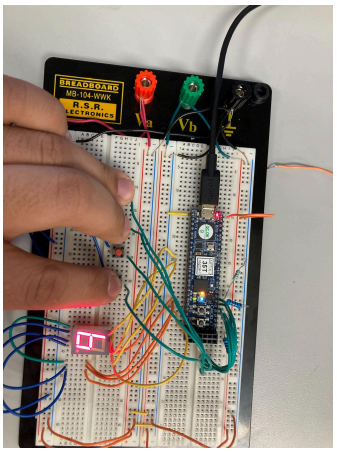
○ Interpretation:
My result was the same as the expected results which was to display 0-9 and A-F so I couldn't have any possible source of error.

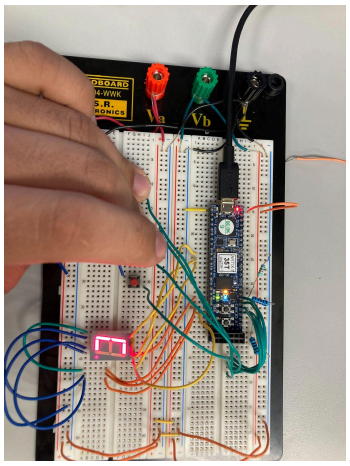
Conclusion and Future Work:

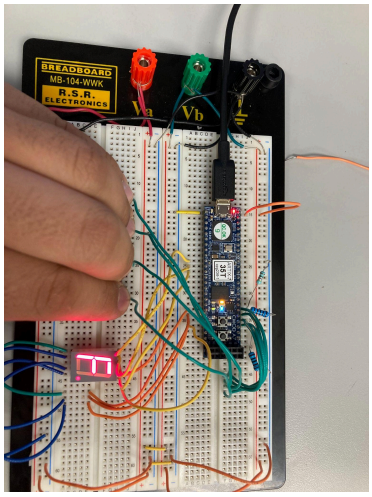
Mastering concepts learned in this lab is crucial for promising FPGA design. Understanding digital logic, VHDL, and the FPGA design flow is the foundation for implementation on the actual breadboard itself. Understanding the idea of digital design concepts is necessary for better FPGA functionality. These digital design concepts could be synchronous/asynchronous logic, flip-flops, and combinational circuits. This would directly impact the design decisions in FPGA development. Some future work in FPGA development could be to explore features and applications. Implementing complex circuits or using a HDL workflow advisor to integrate custom IP cores. You could also consider power optimization or even security applications. Continuous learning in FPGA projects is essential for students so they can adapt in this modern field.

Appendix:
VHDL code/Simulation screenshots.





//Didn’t take a picture of D


References:
https://www.parallel-systems.co.uk/what-is-fpga/

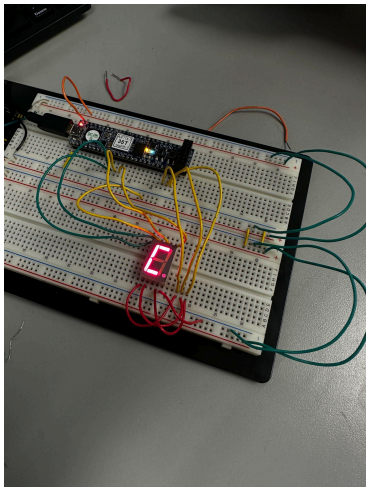
Experiment 2: Four-Bit Ripple-Carry Adder/Subtractor

Theory:
The theoretical basis of the Four-Bit Ripple-Carry Adder/Subtractor experiment is based on concepts like logic design. The goal of the experiment is to design and implement a four-bit ripple-carry adder/subtractor circuit using VHDL and program it onto an FPGA. The fundamental concepts in digital logic design are binary arithmetic, design theory, and the use of FPGA to implement it into hardware.

○ Procedure:

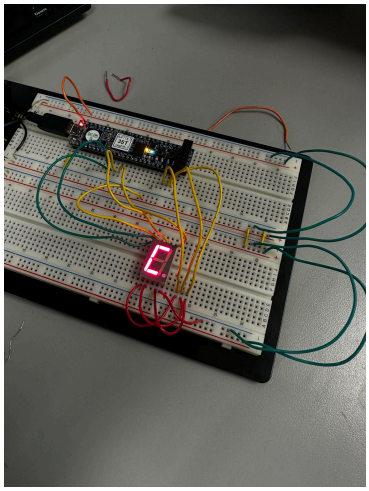
- Look at how you set up your experiment 1 for this lab
- Take out all the connections for the buttons and take out the buttons themselves
- Now take out all the connections for the timer but leave the timer where it is on the breadboard
- Connect a wire from the first port on the bottom of the timer to somewhere on the right of it
- Do this with the next port
- Skip the port in the middle and grab a wire and do the same thing on the port next to the on in the middle
- Grab a wire on the last slot of the bottom of the timer and connect it to ground
- Next look at how you setup the connections from the top of the timer to the fpga in experiment 1 and replicate it

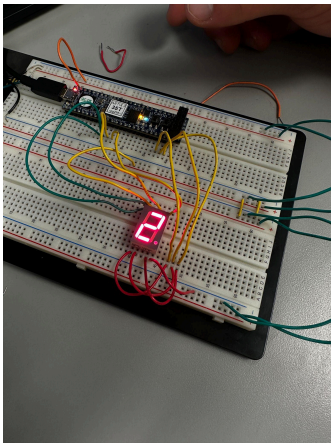
○ Breadboard Layout/Schematic:

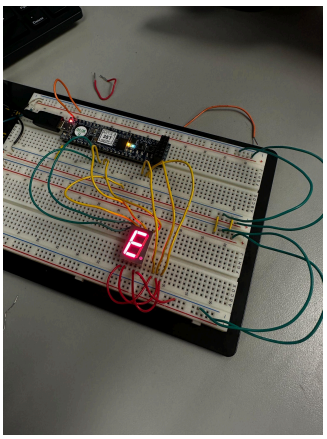


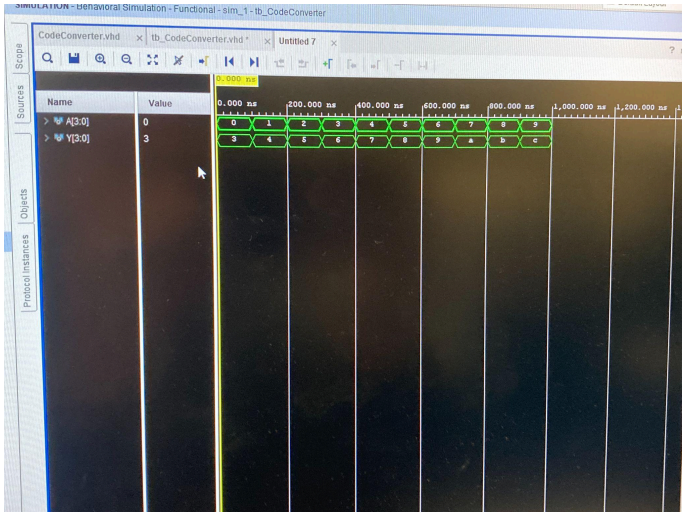
○ All apparatus:
BreadBoard R.S.R Electronics: MB-104-WWK
Xilinx Artrix-7 FPGA
Seven-Segment Display

○ Results:
//Full Adder Results

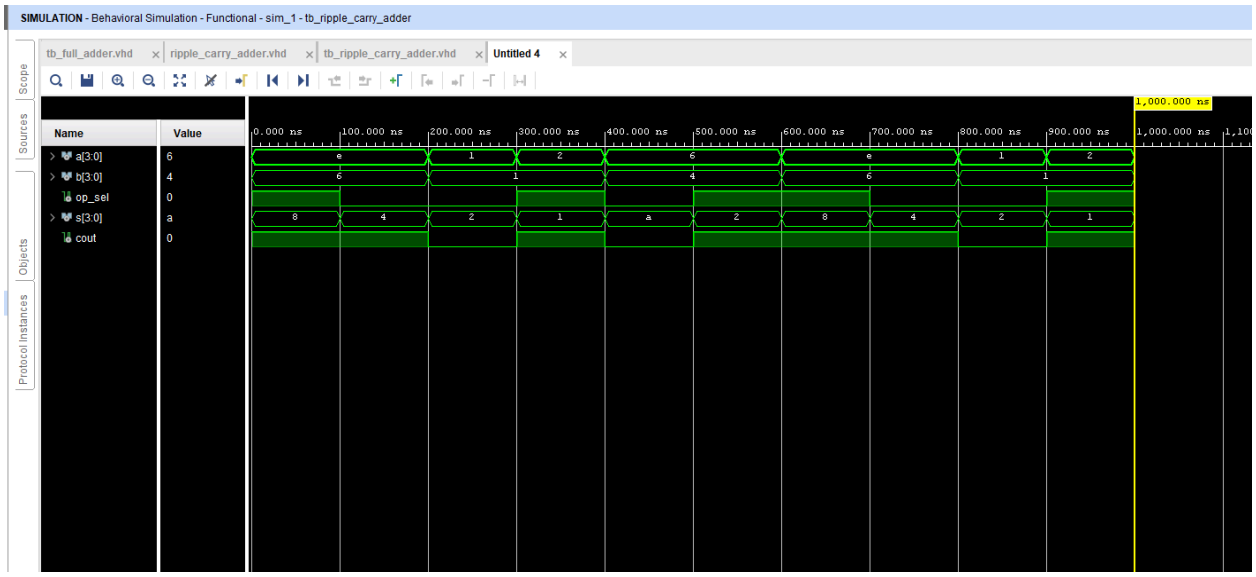




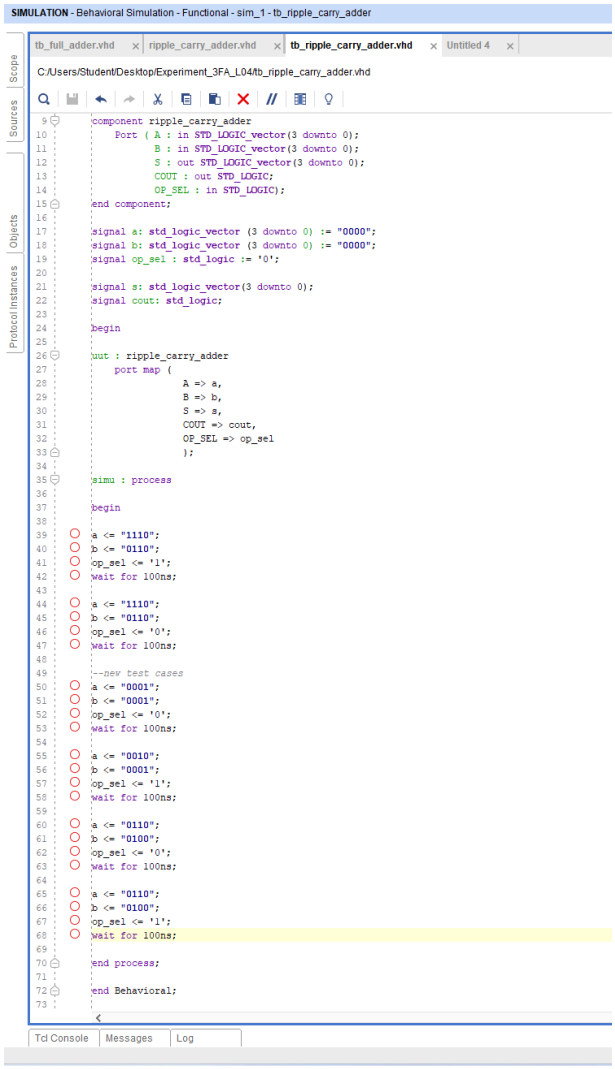




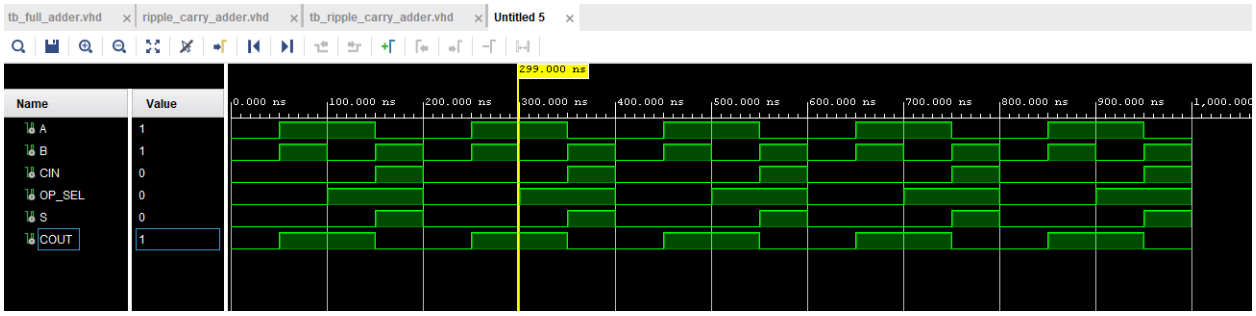
// Ripple Carry Results



//Ripple Carry Code



//Full Adder Results



//Full Adder Code

tb_full_adder.vhd | ripple_carry_adder.vhd | tb_ripple_carry_adder.vhd | Untitle

C:/Users/Student/Desktop/Experiment_3FA_L04/tb_full_adder.vhd

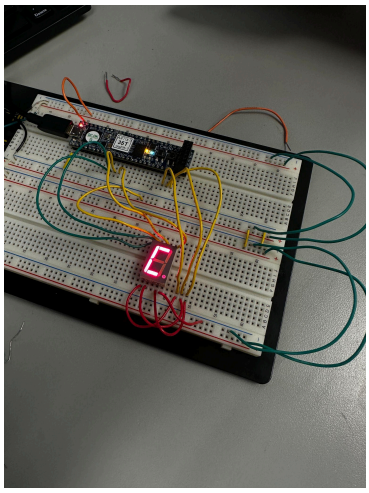
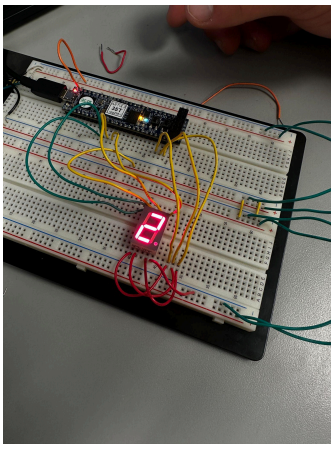
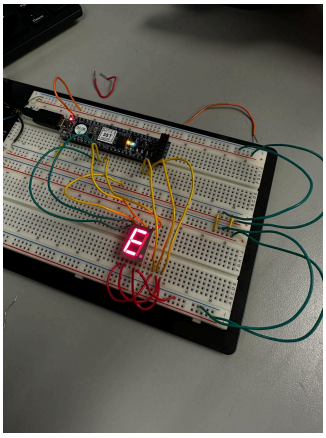
```
18      end component;
19
20      signal A : std_logic := '0';
21      signal B : std_logic := '0';
22      signal CIN  : std_logic := '0';
23      signal OP_SEL : std_logic := '0';
24
25      signal S : std_logic;
26      signal COUT : std_logic;
27
28      begin
29
30      uut : full_adder
31      port map (
32                a => A,
33                b => B,
34                cin => CIN,
35                op_sel => OP_SEL,
36                cout => COUT,
37                s => S
38                );
39
40      sim : process
41
42      begin
43
44      A <= '0';
45      B <= '0';
46      CIN <= '0';
47      op_sel <= '0';
48      wait for 50ns;
49
50      A <= '1';
51      B <= '1';
52      CIN <= '0';
53      op_sel <= '0';
54      wait for 50ns;
55
56      A <= '1';
57      B <= '0';
58      CIN <= '0';
59      op_sel <= '1';
60      wait for 50ns;
61
62      A <= '0';
63      B <= '1';
64      CIN <= '1';
65      op_sel <= '1';
66      wait for 50ns;
67
68      end process;
69
70      end Behavioral;
71
```

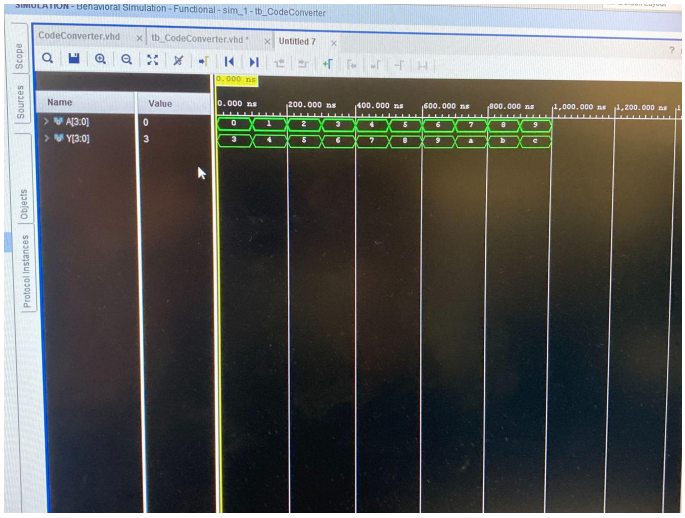
○ Interpretation:

My results for the full adder were just like the expected results, which was that the counter would start at 0 and continue to count to 9 and then switch to A(which is 10) then count until it reached F. My ripple carry results were just like the expected results, which was you gave the variable, a, value of 6. Then you gave the variable, b, value of 4. Then in the ripple carry code you would set the op_sel to either a value of 0, which would add the two values, or 1, which would subtract the two values. In this case it was 0 and, “s”, outputted a value of “a.” Me and my partner did the same values, 6 and 4, but instead we changed the op_sel to a value of 1. This in result subtracted the two values and it outputted a value of 2.

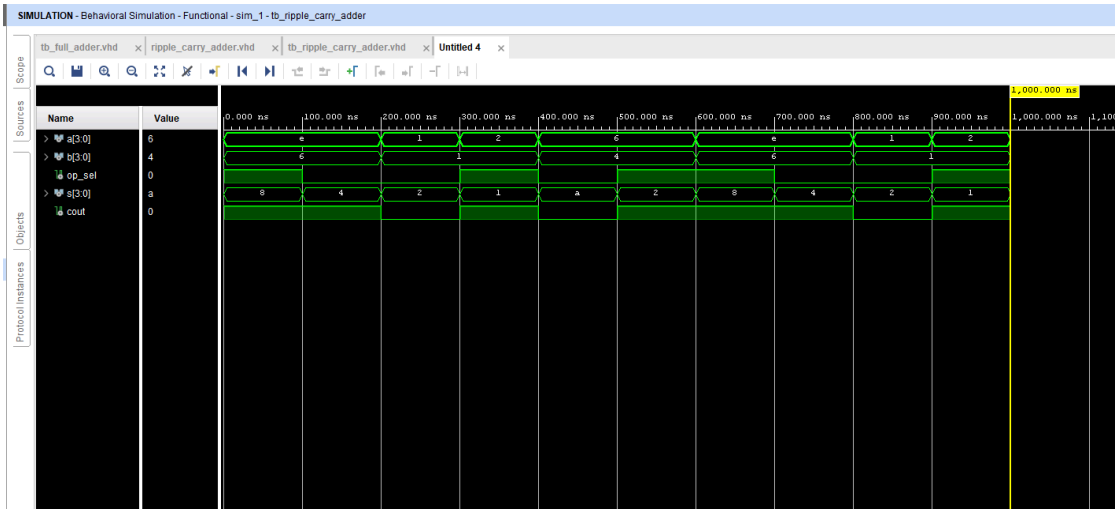
Conclusion and Future Work:
Mastering concepts learned in the lab is necessary for designing FPGA systems. Understanding digital logic concepts, VHDL programming, and design theory makes reliable FPGA circuits. Designing circuits for code conversion, adders/subtractors, and even traffic light controllers shows real-life application. In FPGA design, the proficiency in VHDL coding gives us better application to complex circuits and functions. Future work for FPGAs could involve advanced FPGA features, parallel processing to enhance circuit performance. Further work could dive down to real-time applications and even a combination of other technologies.

Appendix:
//Full Adder Results

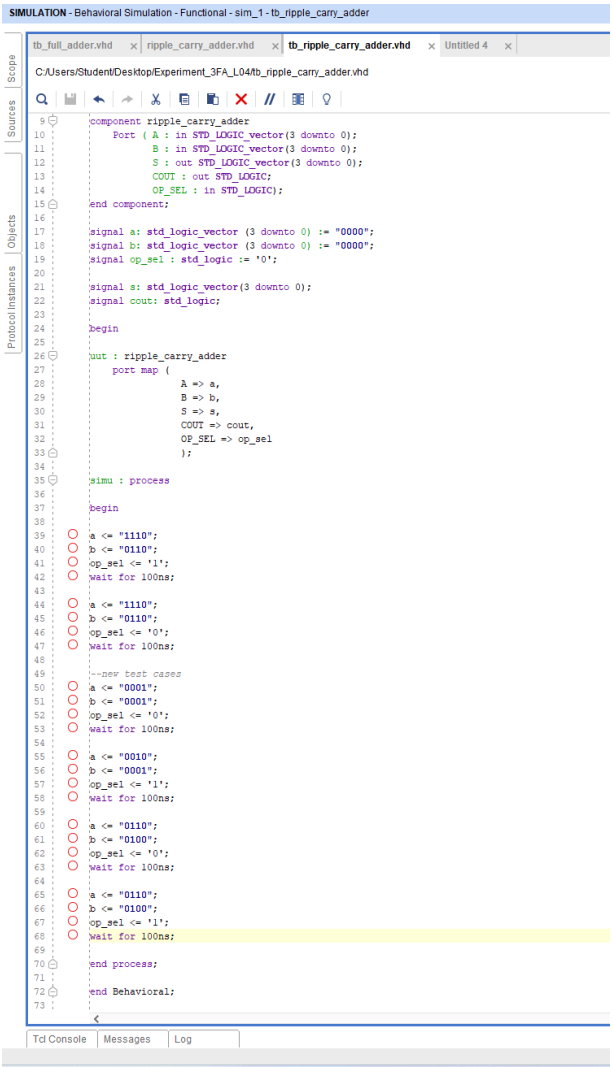






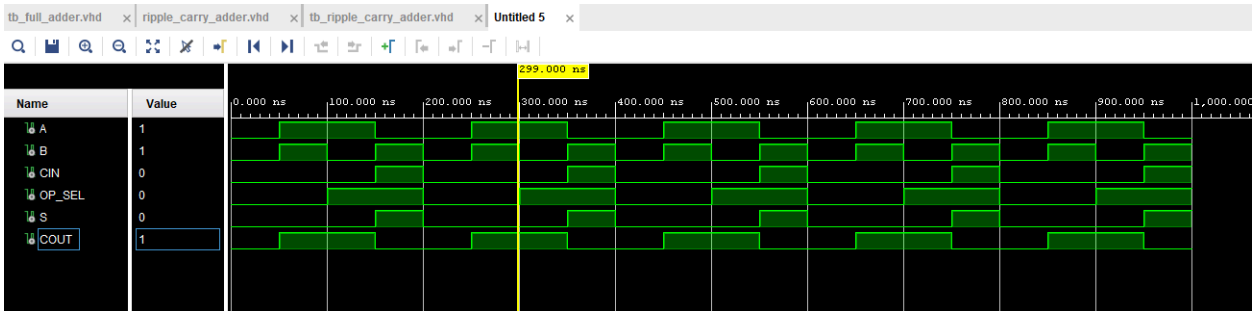
// Ripple Carry Results



//Ripple Carry Code



//Full Adder Results



//Full Adder Code

tb_full_adder.vhd | ripple_carry_adder.vhd | tb_ripple_carry_adder.vhd | Untitle

C:/Users/Student/Desktop/Experiment_3FA_L04/tb_full_adder.vhd

```
18      end component;
19
20      signal A : std_logic := '0';
21      signal B : std_logic := '0';
22      signal CIN  : std_logic := '0';
23      signal OP_SEL : std_logic := '0';
24
25      signal S : std_logic;
26      signal COUT : std_logic;
27
28      begin
29
30      uut : full_adder
31      port map (
32                  a => A,
33                  b => B,
34                  cin => CIN,
35                  op_sel => OP_SEL,
36                  cout => COUT,
37                  s => S
38                  );
39
40      sim : process
41
42      begin
43
44      A <= '0';
45      B <= '0';
46      CIN <= '0';
47      op_sel <= '0';
48      wait for 50ns;
49
50      A <= '1';
51      B <= '1';
52      CIN <= '0';
53      op_sel <= '0';
54      wait for 50ns;
55
56      A <= '1';
57      B <= '0';
58      CIN <= '0';
59      op_sel <= '1';
60      wait for 50ns;
61
62      A <= '0';
63      B <= '1';
64      CIN <= '1';
65      op_sel <= '1';
66      wait for 50ns;
67
68      end process;
69
70      end Behavioral;
71
```

References:

Experiment 3: Traffic Light Controller

Theory:

The theoretical basis of the traffic light controller experiment lies in the design and implementation of a traffic light controller using sequential logic circuits. This experiment shows how a simple traffic light system can be controlled without the use of microcontrollers but instead by relying on sequential logic circuits. Overall, this experiment involves the concepts of digital logic design, finite state machines, sequential circuits, and implementation of a traffic light controller using VHDL code.

○ Procedure:

- Didn't have to do anything for this experiment as long as you downloaded and uploaded the correct VHDL file into Vivado you were able to do this experiment.

○ Breadboard Layout/Schematic:

- The breadboard layout was just an fpga connected on top of the breadboard.

○ All apparatus:
BreadBoard R.S.R Electronics: MB-104-WWK
Xilinx Artrix-7 FPGA

○ Results:
// Sensor Traffic Light Code Results

trafficlightFSM_tb.vhd | trafficlight_sensored_tb.vhd | Untitled 6

C:/Users/Student/Desktop/Experiment_3FA_L04/trafficlightFSM_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity trafficlightFSM_tb is
end trafficlightFSM_tb;

architecture Behavioral of trafficlightFSM_tb is

component trafficControlFSM
    port (  clk : in std_logic;
            ns_r, ns_y, ns_g,
            ew_r, ew_y, ew_g : out std_logic);
end component;

signal ns_r, ns_y, ns_g, ew_r, ew_y, ew_g : std_logic;
signal clk : std_logic := '1';

begin

    uut : trafficControlFSM
    port map (  ns_r => ns_r,
                ns_g => ns_g,
                ns_y => ns_y,
                ew_r => ew_r,
                ew_g => ew_g,
                ew_y => ew_y,
                clk => clk);

    clock : process is
    begin
    clk <= '0'; wait for 5ns;
    clk <= '1'; wait for 5ns;
    -- clock period is 10ns
    end process clock;

end Behavioral;
```

//Regular Traffic Light Results

//Regular Traffic Light Code

trafficlightFSM_tb.vhd | trafficlight_sensored_tb.vhd | Untitled 6

C:/Users/Student/Desktop/Experiment_3FA_L04/trafficlightFSM_tb.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity trafficlightFSM_tb is
5   end trafficlightFSM_tb;
6
7   architecture Behavioral of trafficlightFSM_tb is
8
9   component trafficControlFSM
10      port (  clk : in std_logic;
11              ns_r, ns_y, ns_g,
12              ew_r, ew_y, ew_g : out std_logic);
13  end component;
14
15  signal ns_r, ns_y, ns_g, ew_r, ew_y, ew_g : std_logic;
16  signal clk : std_logic := '1';
17
18  begin
19
20      uut : trafficControlFSM
21      port map (  ns_r => ns_r,
22                  ns_g => ns_g,
23                  ns_y => ns_y,
24                  ew_r => ew_r,
25                  ew_g => ew_g,
26                  ew_y => ew_y,
27                  clk => clk);
28
29      clock : process is
30      begin
31      clk <= '0'; wait for 5ns;
32      clk <= '1'; wait for 5ns;
33      -- clock period is 10ns
34      end process clock;
35
36  end Behavioral;
37
```

○ Interpretation:

My results were just like the expected results which was S0 had N.S = G, E.W = R / S1 had N.S = Y, E.W = R / S2 had N.S = R, E.W = R / S3 had N.S = R, E.W = G / S4 had N.S = R, E.W = Y / and S5 had N.S = R, E.W = R. This means that if we look at the regular traffic light results it would always have N.S_g = 1 and E.W_R = 1 and the rest be 0, then it would switch to N.S_Y = 1, then N.S_R = 1, then E.W_R = 1 and E.W_G = 1, then it would repeat itself in a constant loop. This would simulate a traffic light controller.

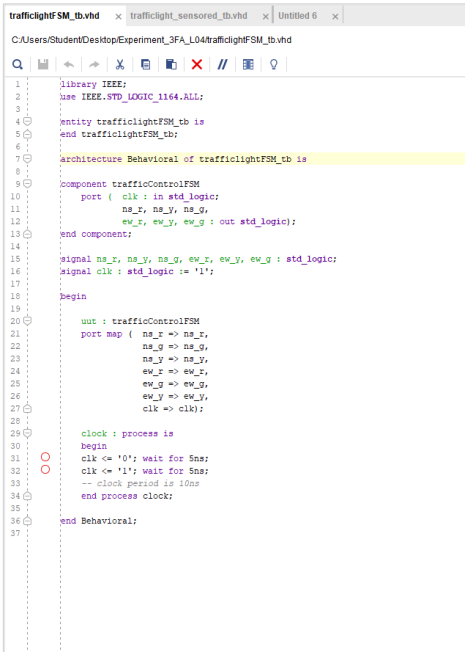
Conclusion and Future Work:

In the traffic light controller lab, learning the concepts of digital logic design and some VHDL programming gave me an understanding of FPGA systems. Learning these concepts is ideal if you want to design your own FPGAs because it's almost a real-life hardware configuration. Understanding sequential logic, VHDL coding, and finite state machines is

necessary in order to create reliable FPGA systems. The traffic light controller experiment on a FPGA board shows real-life application. Implementing a finite state machine and sequential logic circuits shows us students know how to control basic traffic lights. Future work should focus on how to enhance the traffic light controllers composure and security by maybe using data from a real-time traffic sensor. Using communication modules to coordinate multiple traffic intersections or even having the system respond to emergency vehicles faster could play a bigger role for the effectiveness of FPGA based traffic systems.

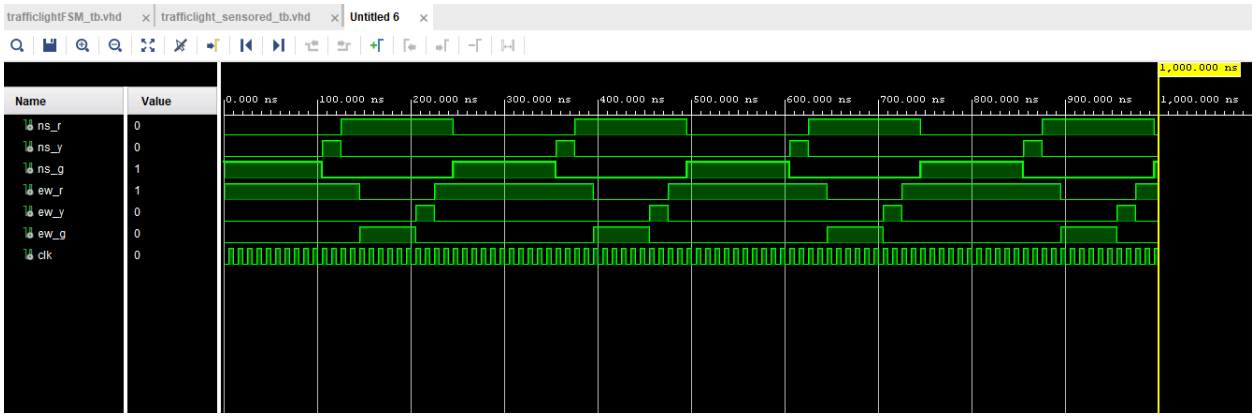
Appendix:

// Sensor Traffic Light Code Results



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity trafficlightFSM_tb is
end trafficlightFSM_tb;

architecture Behavioral of trafficlightFSM_tb is

component trafficControlFSM
    port (  clk : in std_logic;
            ns_r, ns_y, ns_g,
            ew_r, ew_y, ew_g : out std_logic);
end component;

signal ns_r, ns_y, ns_g, ew_r, ew_y, ew_g : std_logic;
signal clk : std_logic := '1';

begin

    uut : trafficControlFSM
    port map (  ns_r => ns_r,
                ns_g => ns_g,
                ns_y => ns_y,
                ew_r => ew_r,
                ew_g => ew_g,
                ew_y => ew_y,
                clk => clk);

    clock : process is
    begin
    clk <= '0'; wait for 5ns;
    clk <= '1'; wait for 5ns;
    -- clock period is 10ns
    end process clock;

end Behavioral;
```

//Regular Traffic Light Results



//Regular Traffic Light Code

trafficlightFSM_tb.vhd | trafficlight_sensored_tb.vhd | Untitled 6

C:/Users/Student/Desktop/Experiment_3FA_L04/trafficlightFSM_tb.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity trafficlightFSM_tb is
5   end trafficlightFSM_tb;
6
7   architecture Behavioral of trafficlightFSM_tb is
8
9   component trafficControlFSM
10      port (  clk : in std_logic;
11              ns_r, ns_y, ns_g,
12              ew_r, ew_y, ew_g : out std_logic);
13  end component;
14
15  signal ns_r, ns_y, ns_g, ew_r, ew_y, ew_g : std_logic;
16  signal clk : std_logic := '1';
17
18  begin
19
20      uut : trafficControlFSM
21      port map (  ns_r => ns_r,
22                  ns_g => ns_g,
23                  ns_y => ns_y,
24                  ew_r => ew_r,
25                  ew_g => ew_g,
26                  ew_y => ew_y,
27                  clk => clk);
28
29      clock : process is
30      begin
31      clk <= '0'; wait for 5ns;
32      clk <= '1'; wait for 5ns;
33      -- clock period is 10ns
34      end process clock;
35
36  end Behavioral;
37
```

References: