ECE 124- Lab2 Report

VHDL CODE

Code for Logic processor

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity LogicProcessor is port (

INPUT\_A : in std\_logic\_vector(3 downto 0);

INPUT\_B : in std\_logic\_vector(7 downto 4);

OPTION: in std\_logic\_vector(3 downto 0);

OUTPUT: OUT std\_logic\_vector(7 downto 0)

);

end LogicProcessor;

architecture Behavioral of LogicProcessor is

begin

with OPTION(3 downto 0) select

-- based on the button clicked, determine the logic used for each digit

OUTPUT <= "0000" & INPUT\_A AND INPUT\_B when "1110",

"0000" & INPUT\_A OR INPUT\_B when "1101",

"0000" & INPUT\_A XOR INPUT\_B when "1011",

"11111111" when others;

-- we handle unexpected input in the MUX

end architecture Behavioral;

Code for Mux-led signals

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity MUX\_LED is port (

INPUT\_A : in std\_logic\_vector(7 downto 0);

INPUT\_B : in std\_logic\_vector(7 downto 0);

CONTROL: in std\_logic\_vector(3 downto 0);

OUTPUT: OUT std\_logic\_vector(7 downto 0)

);

end MUX\_LED;

architecture Behavioral of MUX\_LED is

begin

--if nothing is pressed, then output 00000000

--if pb3 is pressed, output addition result on LED

--if anything but pb3 is pressed, output logic on LED

--if received unexpected result, light up all LEDs

with CONTROL (3 downto 0) select

OUTPUT <= "00000000" when "1111",

INPUT\_A when "0111",

INPUT\_B when "1011",

INPUT\_B when "1101",

INPUT\_B when "1110",

"11111111" when others;

end architecture Behavioral;

Code for the Logic Adder

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity LogicAdder is port (

INPUT\_A : in std\_logic\_vector(3 downto 0);

INPUT\_B : in std\_logic\_vector(7 downto 4);

OUTPUT: OUT std\_logic\_vector(7 downto 0)

);

end LogicAdder;

architecture Behavioral of LogicAdder is

begin

-- First convert logic to integer so we can perform addition on INPUT\_A and INPUT\_B,

-- then convert it back into the logic, so we can feed it to both led and 7seg

OUTPUT(7 downto 0) <= std\_logic\_vector(unsigned("0000" & INPUT\_B) + unsigned("0000" & INPUT\_A));

end architecture Behavioral;

Code for the edited main VHDL file

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity LogicalStep\_Lab2\_top is port (

clkin\_50 : in std\_logic;

pb : in std\_logic\_vector(3 downto 0);

sw : in std\_logic\_vector(7 downto 0); -- The switch inputs

leds : out std\_logic\_vector(7 downto 0); -- for displaying the switch content

seg7\_data : out std\_logic\_vector(6 downto 0); -- 7-bit outputs to a 7-segment

seg7\_char1 : out std\_logic; -- seg7 digit1 selector

seg7\_char2 : out std\_logic -- seg7 digit2 selector

);

end LogicalStep\_Lab2\_top;

architecture SimpleCircuit of LogicalStep\_Lab2\_top is

--

-- Components Used ---

-------------------------------------------------------------------

component SevenSegment port (

hex : in std\_logic\_vector(3 downto 0); -- The 4 bit data to be displayed

sevenseg : out std\_logic\_vector(6 downto 0) -- 7-bit outputs to a 7-segment

);

end component;

component segment7\_mux port (

clk : in std\_logic :='0';

DIN2 : in std\_logic\_vector(6 downto 0);

DIN1 : in std\_logic\_vector(6 downto 0);

DOUT : out std\_logic\_vector(6 downto 0);

DIG2 : out std\_logic;

DIG1 : out std\_logic

);

end component;

component LogicProcessor port (

INPUT\_A : in std\_logic\_vector(3 downto 0);

INPUT\_B : in std\_logic\_vector(7 downto 4);

OPTION : in std\_logic\_vector(3 downto 0);

OUTPUT : OUT std\_logic\_vector(7 downto 0)

);

end component;

component LogicAdder port (

INPUT\_A : in std\_logic\_vector(3 downto 0);

INPUT\_B : in std\_logic\_vector(7 downto 4);

OUTPUT : OUT std\_logic\_vector(7 downto 0)

);

end component;

component MUX port (

INPUT\_A : in std\_logic\_vector(7 downto 0);

INPUT\_B : in std\_logic\_vector(7 downto 0);

CONTROL : in std\_logic\_vector(3 downto 0);

OUTPUT : OUT std\_logic\_vector(7 downto 0)

);

end component;

component MUX\_LED port (

INPUT\_A : in std\_logic\_vector(7 downto 0);

INPUT\_B : in std\_logic\_vector(7 downto 0);

CONTROL : in std\_logic\_vector(3 downto 0);

OUTPUT : OUT std\_logic\_vector(7 downto 0)

);

end component;

-- Create any signals, or temporary variables to be used

--

-- std\_logic\_vector is a signal which can be used for logic operations such as OR, AND, NOT, XOR

--

signal seg7\_A : std\_logic\_vector(6 downto 0);

signal seg7\_B : std\_logic\_vector(6 downto 0);

signal hex\_A : std\_logic\_vector(3 downto 0);

signal hex\_B : std\_logic\_vector(7 downto 4);

--stores result for adder

signal adder : std\_logic\_vector(7 downto 0);

--stores result for logic

signal logic : std\_logic\_vector(7 downto 0);

--stores result for the chosen seg7

signal seg7 : std\_logic\_vector(7 downto 0);

--NOTE, we don't store result for LED since we can directly implement it on the output

-- Here the circuit begins

begin

--seperate sw into two 4 bit logic for convinient usage in the future

hex\_A <= sw(3 downto 0);

hex\_B <= sw(7 downto 4);

--compute logic output based on sw and pb

LOGIC\_OUTPUT: LogicProcessor port map(hex\_A, hex\_B, pb, logic);

--compute adder result based on sw and pb

ADDER\_OUTPUT: LogicAdder port map(hex\_A, hex\_B, adder);

--choose the right input for LED

LED\_INPUT: MUX\_LED port map(adder, logic, pb, leds);

--choose the right input for 7Seg

SEG7\_INPUT: MUX port map(sw, adder, pb, seg7);

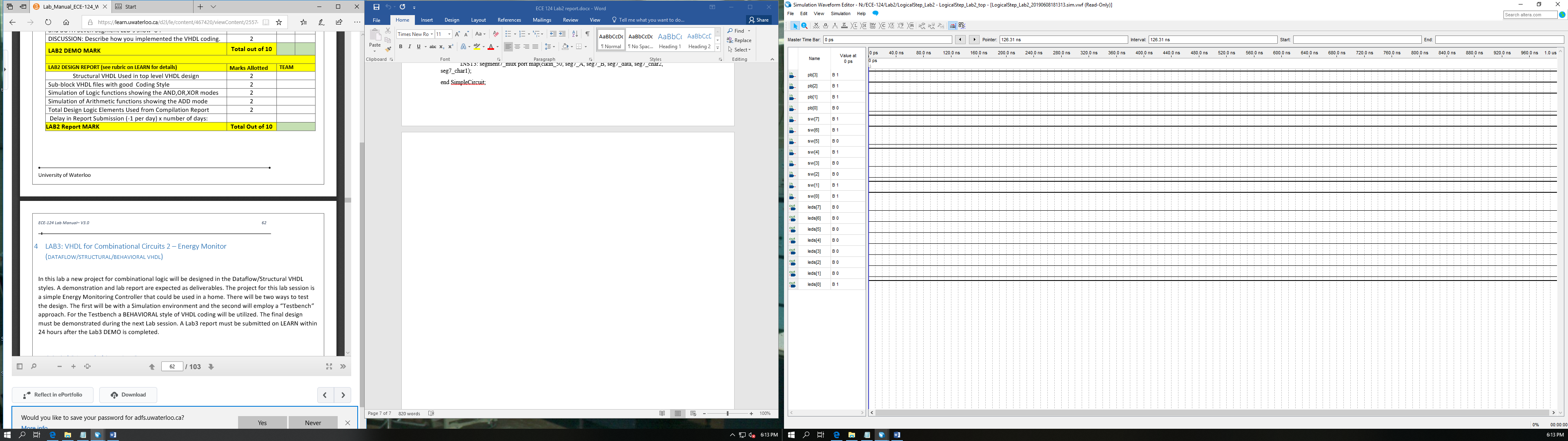
--processing 7Seg and output it to seg7\_data, seg7\_char2, seg7\_char1

INST1: SevenSegment port map(Seg7(3 downto 0), seg7\_A);

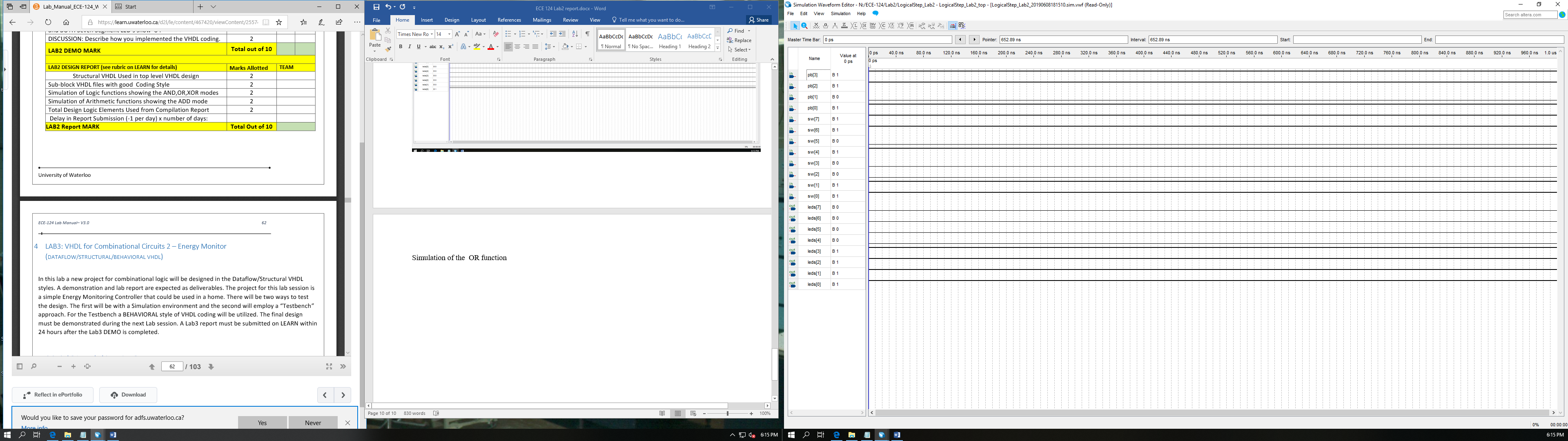
INST2: SevenSegment port map(Seg7(7 downto 4), seg7\_B);

INST3: segment7\_mux port map(clkin\_50, seg7\_A, seg7\_B, seg7\_data, seg7\_char2, seg7\_char1);

end SimpleCircuit;

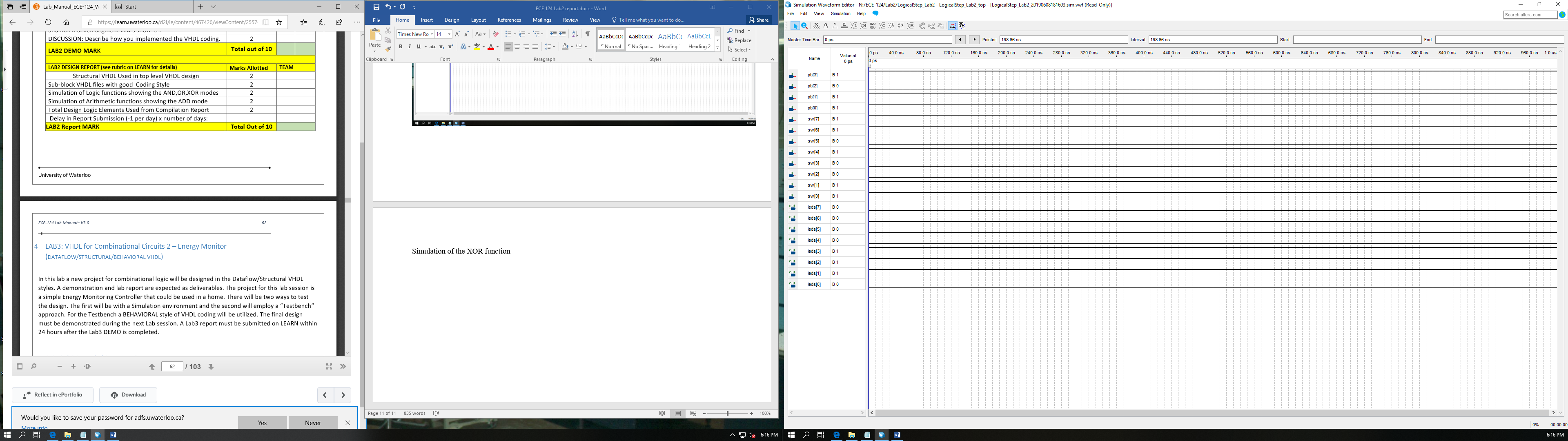
Simulation of the AND function

The And operation is on effect here. Pb[0] is operation which entails the and operation. The switches 0,1,4,6 and 7 are switched on. The ANDed result of the number can be seen in the leds as led[0] is on, which is the correct result.

Simulation of the OR function

The OR operation is on effect here. Pb[1] is operation which entails the and operation. The switches 0,1,4,6 and 7 are switched on. The ORed result of the number can be seen in the leds as leds[0,1,2,3] are on, which is the correct result.

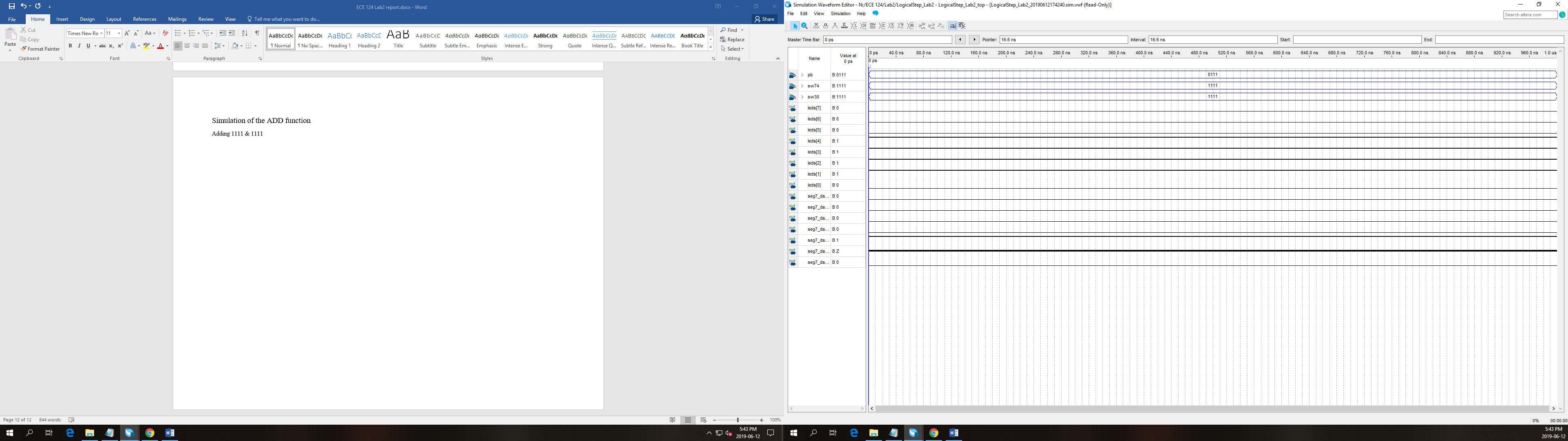
Simulation of the XOR function



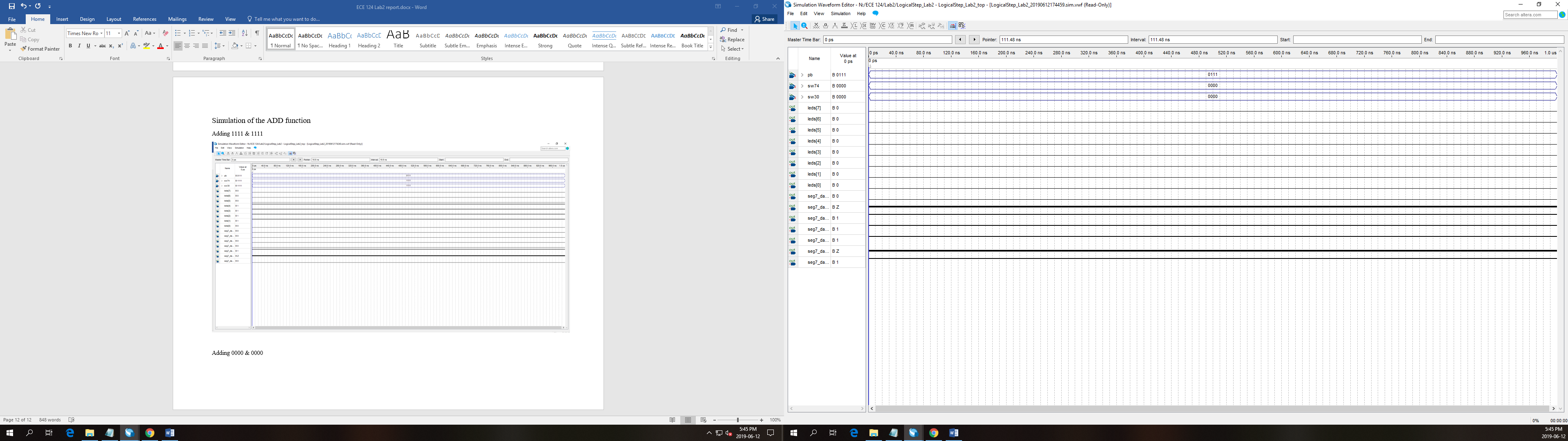
The XOR operation is on effect here. Pb[2] is operation which entails the and operation. The switches 0,1,4,6 and 7 are switched on. The ORed result of the number can be seen in the leds as leds[1,2,3] are on, which is the correct result.

Simulation of the ADD function

Adding 1111 & 1111

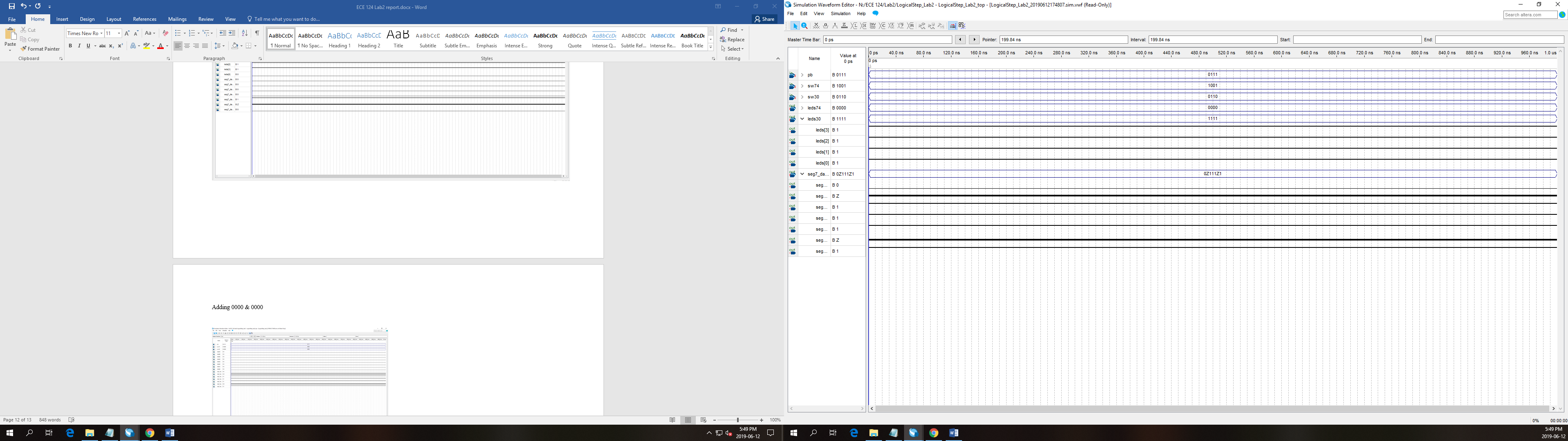


The arithmetic ADD function is displayed in the simulation above. Pb[3] is active which is meant for the ADD function. In this simulation, all the switch were set to 1 and the add operation was performed. The inputs were 15+15 which should evaluate to 30. The leds[4,3,2,1] were turned on, which is the correct output. The leds display the output “00011110” which is 30. The seven segments show the “1E” which is 30 in hex.

Adding 0000 & 0000

The arithmetic ADD function is displayed in the simulation above. Pb[3] is active which is meant for the ADD function. In this simulation, all the switch were set to 0 and the add operation was performed. The inputs were 0+0 which should evaluate to 0. None of the leds were turned on, which shows the correct output of 0. The seven segment adder shows the “0” which is the correct value.

Adding 1001 & 0110

In this case, the numbers “1001” & “0110” were added together, which evaluates to 15. The leds[0,1,2,3] were turned on, which is the correct result. The seven segment shows the output “F” which is 15 in hexadecimal.

Total Design Logic elements Used from the Compilation report.

