

**Department of Electrical Engineering**

**Lab Report 4: 2-Bit Adder**

Name: Socheath Sok, ID: 014470701

Partner: Ashley Tran

Group: 11

Professor: Dr. Ray Wang

Class: EE 301

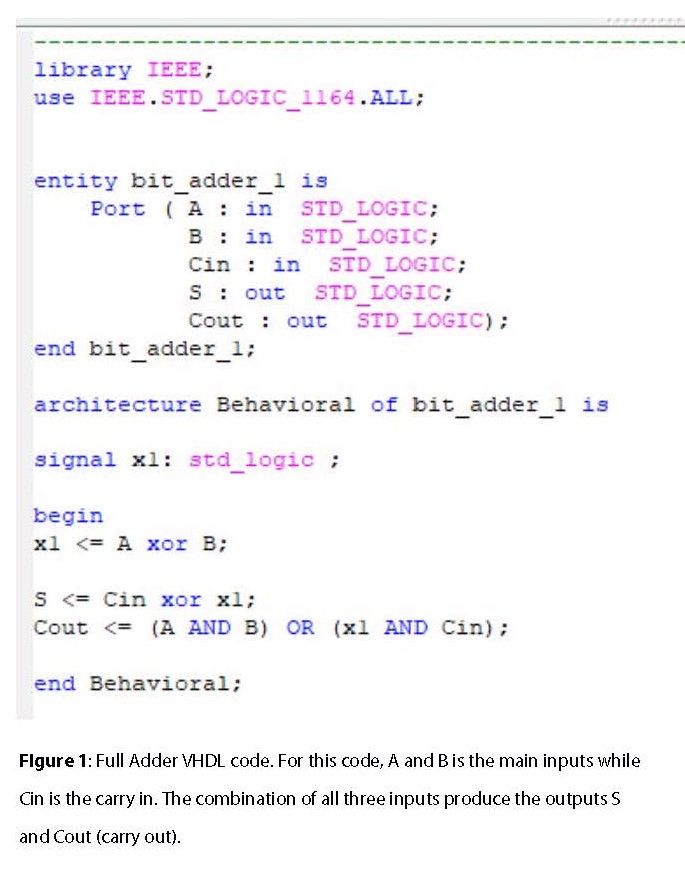
Date Performed: October 9, 2017

**Abstract**

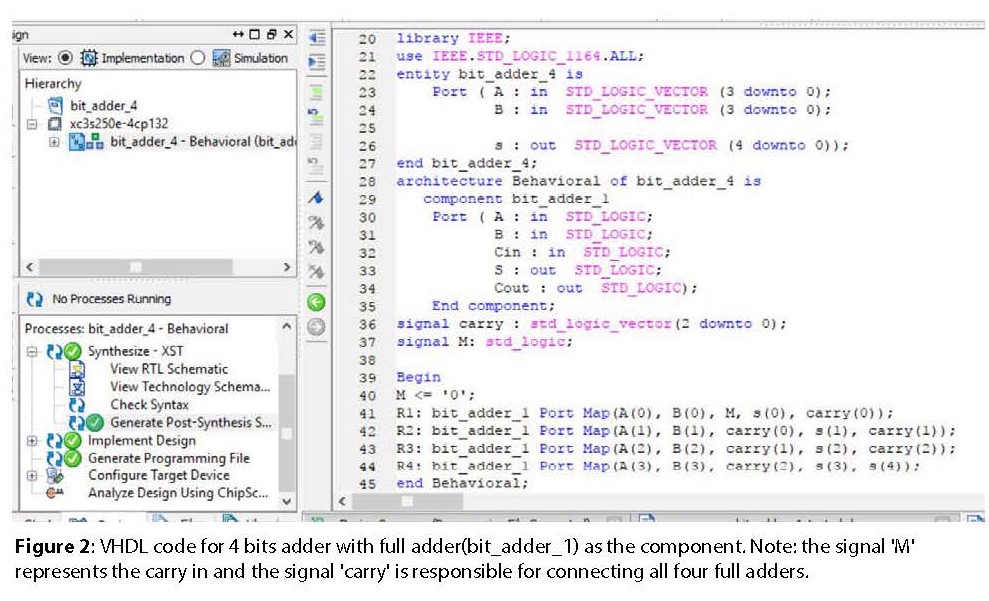
The goal of this lab is to design a 2-bit adder using methods learned in the previous labs, and creating a timing diagram to test the code as well as implementing the design on the board. An optional lab of making a 2-bit subtractor by manipulating the 2-bit adder VHDL code and adding some additional gates was also given.. For our group, we decided to make a 4-bit adder, a 4-bit subtractor, and a 4-bit selectable. This report will focus mainly on how we made the 4-bit adder and extra information about the subtractor and the selectable will be given in the the ‘Extra Credit’ section at the end.

**Introduction**

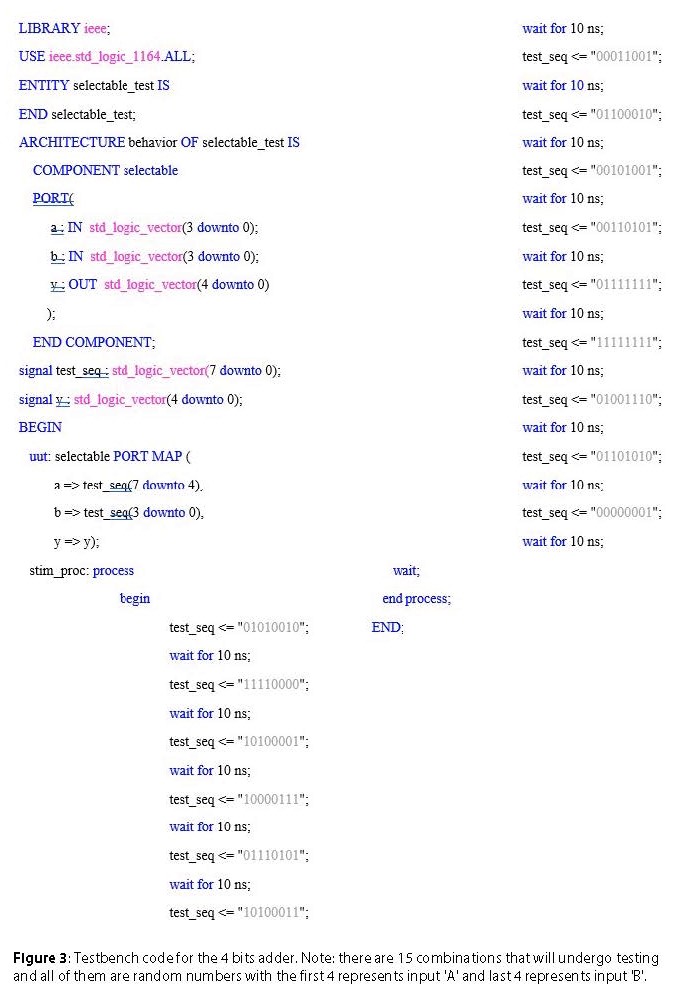
For this lab, we decided to make a full adder first and used it as a component for the 4-bit adder. The full adder that we made has two inputs, one carry in, and one output and one carry out. The VHDL code that explained how those ports are connected is shown in **Figure 1**.

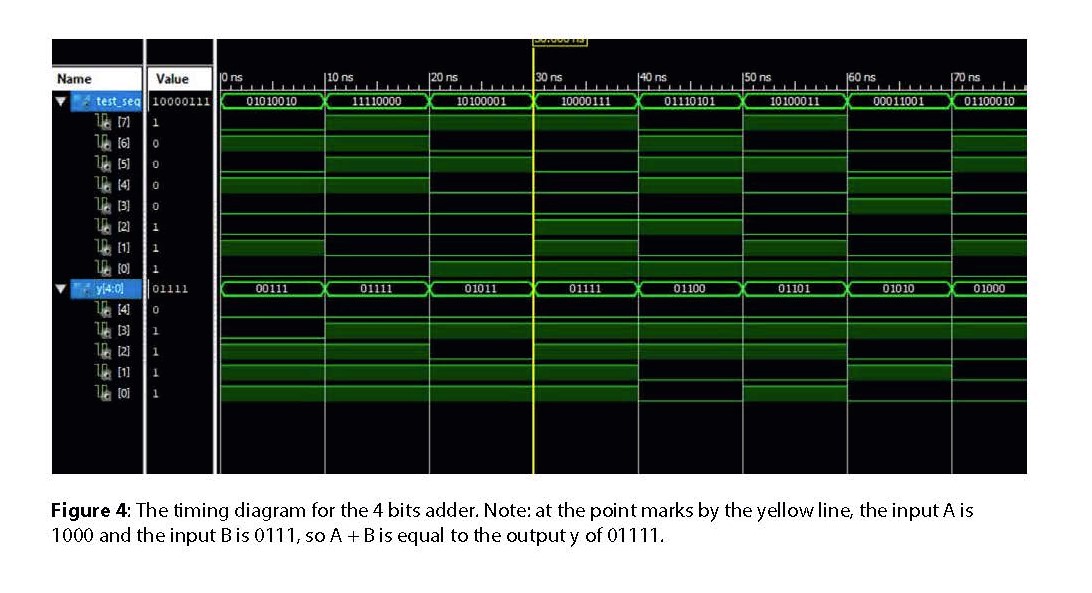


With the full adder code saved, we created a new VHDL module for our 4-bit adder and called it ‘bit\_4\_adder’. For this code, there are 8 inputs (‘A and ’B’) and 4 outputs (‘s’). Input ‘A’ is responsible for the 4 most significant bits and B is responsible for 4 least significant bits. Since we know that the carry in is always ‘0’, we chose to put it as a signal so that we do not have to defined it in the constraint file later. With everything defined, we used the component/port map method to connect the 4-bit adder to the full adder code created in **Figure 1** and the final result is shown in **Figure 2**.

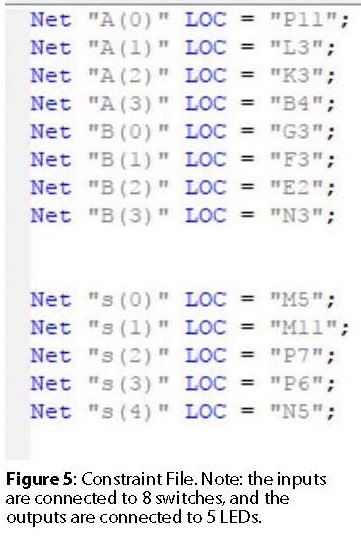


The next step is to create a testbench code to test our design. After the skeleton file was created, we added the ‘test\_seq’ signal and assigned it to input ‘A’ and ‘B’. By doing so, signal ‘A’ and ‘B’ created automatically by the program can be removed from the code and they will not show up in the timing diagram. Under the ‘process’ section, we put in different random combinations with ‘10 ns’ period for each one. The testbench code with all the changes is shown in **Figure 3** below and the timing diagram created as the result is shown in **Figure 4**.





After checking the addition and finding no mistakes, we moved on to create the constraint file for implementing our design. Since we had 8 inputs, we ended up using all the switches, but we only need 5 LEDs for the outputs, so we decided to use the first 5, thus making ‘led4’ on the Basys2 our most significant bit (carry out/sign bit) and ‘led0’ our least significant bit. The connection between the inputs and outputs to the pin on the board is shown in **Figure 5**.

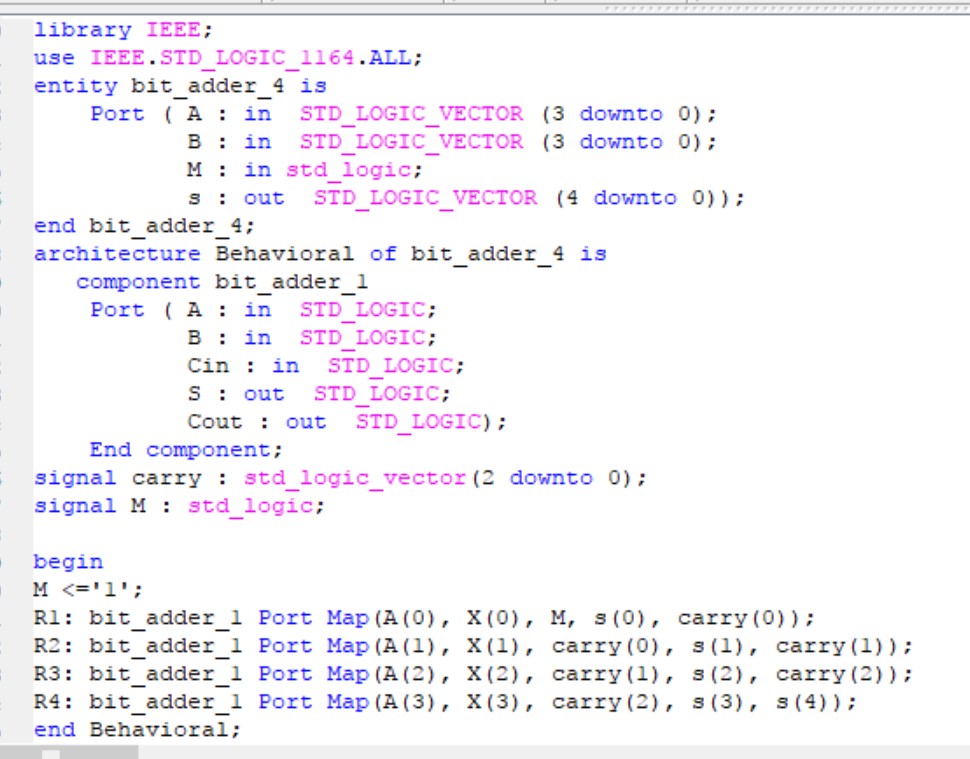


**Extra Credit**

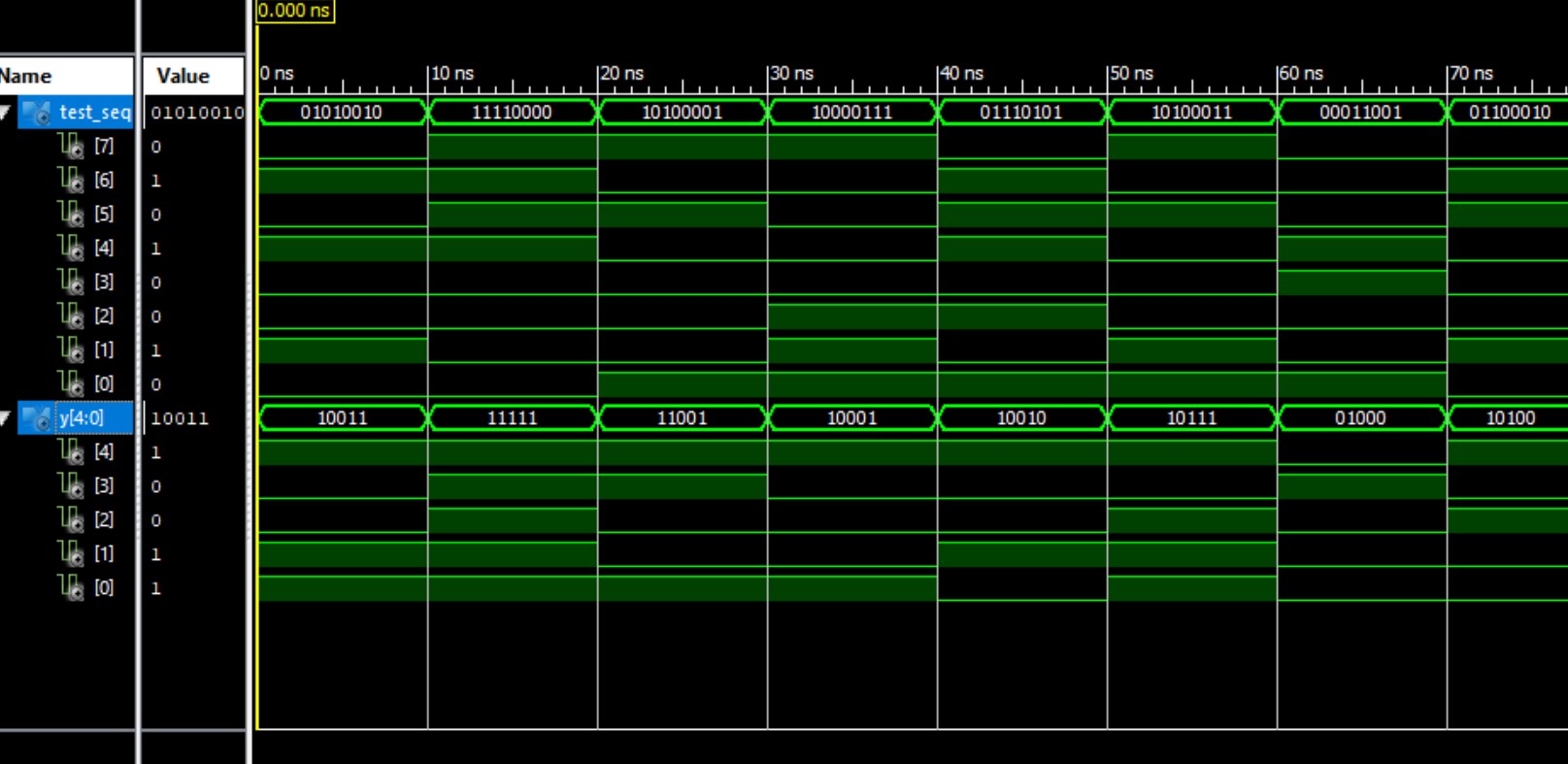
***Subtraction***

In order to get the 4-bit subtractor, we made a few changes to our 4-bit adder code. To change from addition to subtraction, we put an inverter on all of the ‘B’ inputs and change the carry in signal to ‘1’. Then, we created a truth table for the combinations used for the adder, but change them to subtraction. Using the same testbench code, we see that the output matches the number of the table. The VHDL code for the subtraction is shown below along with the timing diagram and the truth table. The constraint file remained the same as the adder.

**VHDL CODE for Subtractor**



**Waveform**



In the waveform, at 60 ns, the first four bits (0001) are ‘A’ and the last four bits (1001) are ‘B’. Since ‘B’ is the negative binary according to the code, we applied first and second complement to it and get a positive ‘B’ of 0111. When we added ‘A’ and positive ‘B’ together, the result was 01000, which is the same as the truth table below. For subtractor, the MSB (carry out) is the sign bit and can often be ignored, however, if we were to do the operation in terms of decimal, one minus nine is equal to negative eight. To get this, we can apply first and second compliments to the output (01000) and the answer would 11000 or (-8).

**Truth table for the subtractor**

A B S

0101 0010 10011

1111 0000 11111

1010 0001 11001

1000 0111 10001

0111 0101 10010

1010 0011 10111

0001 1001 01000 @ 60 ns

0110 0010 10100

0010 1001 01001

0011 0101 01110

0111 1111 01000

1111 1111 10000

0100 1110 00110

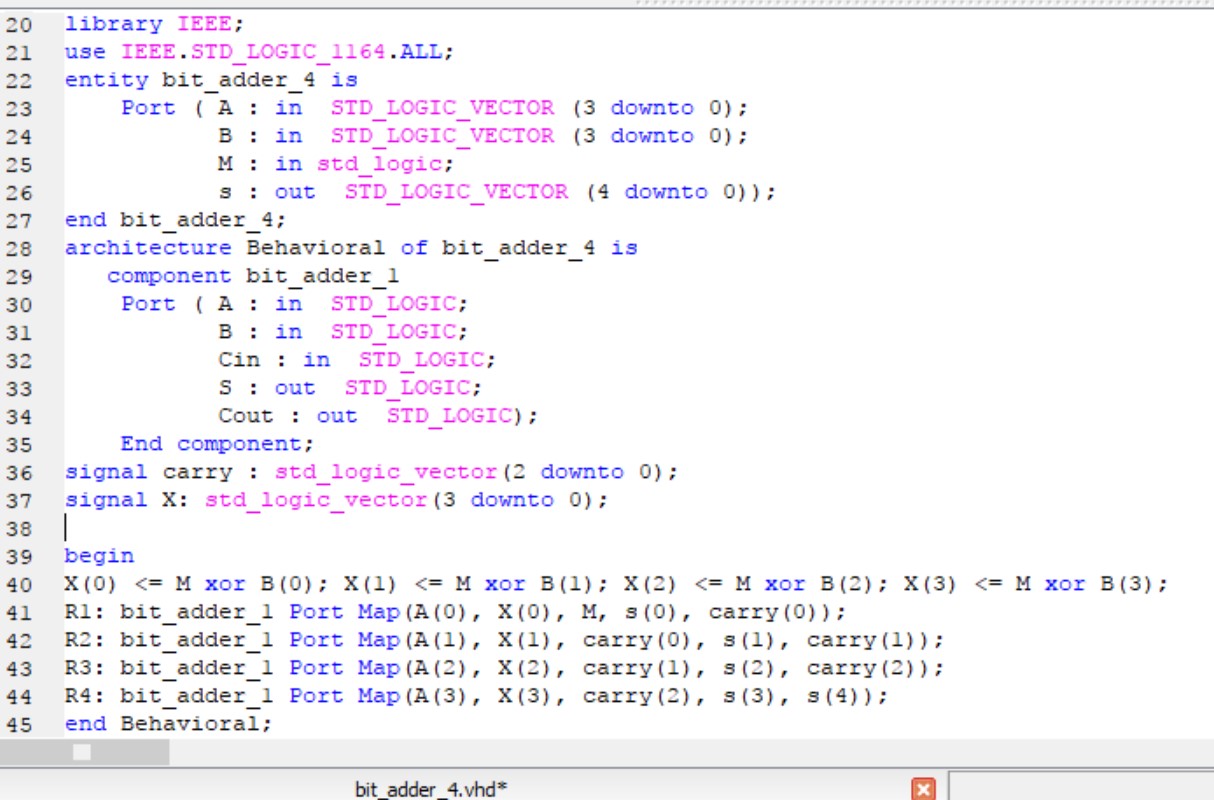
0110 1010 01100

0000 0001 0111

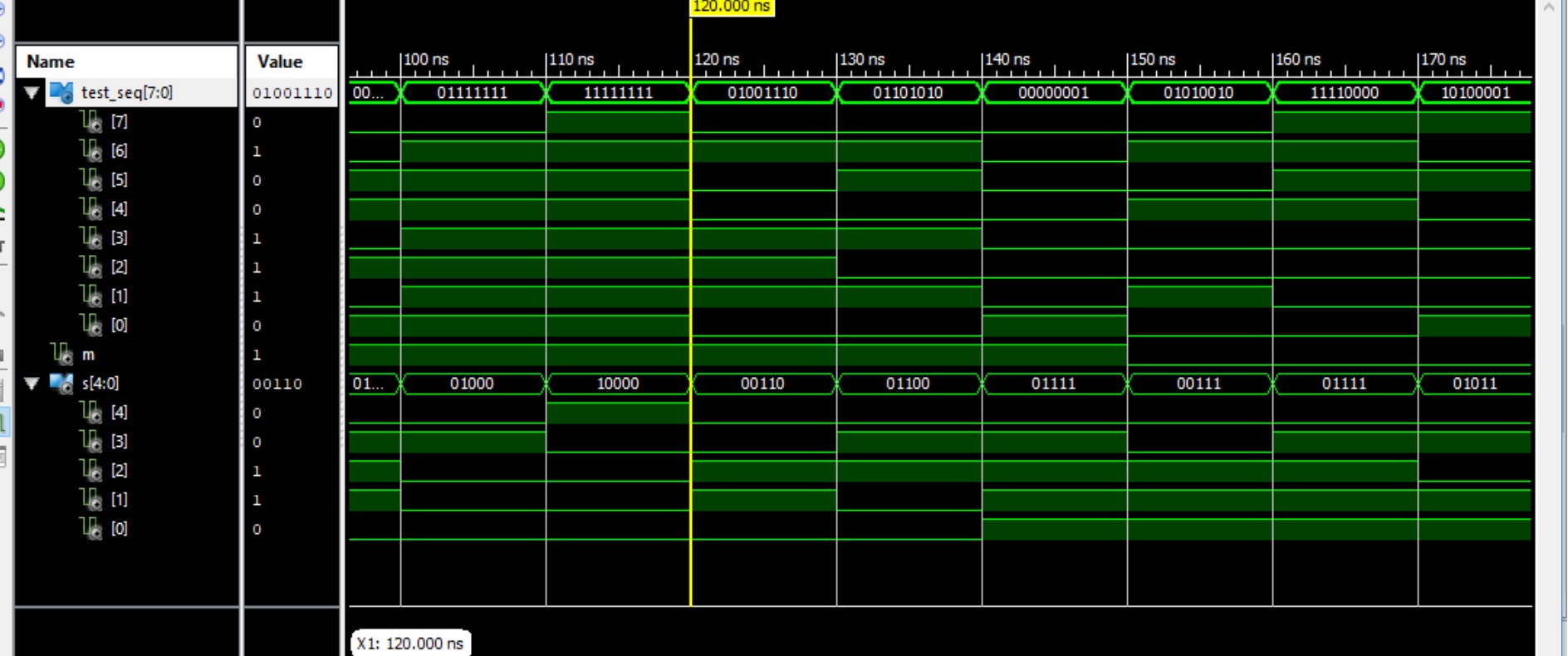
***Selectable***

For the selectable, we changed ‘M’ from a signal to an input. Then, we added 4 XOR gates with binded ‘M’ and ‘B’ together as the inputs. The waveform for the selectable is the combination of the timing diagrams for the addition and the subtraction. Since we had no more switch left, we assigned input ‘M’ to a button on the board. When ‘M’ is one, the output is a subtraction and when ‘M’ is zero, the output is an addition. The VHDL code for the selectable, the constraint file, the waveform and the testbench code are shown below.

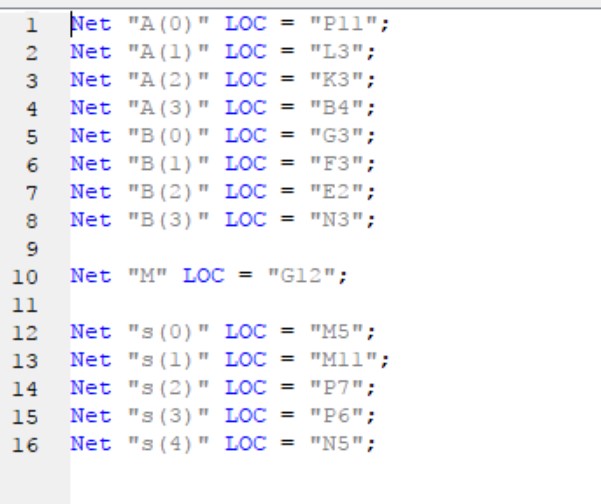
**VHDL code for 4-bit Selectable**

****

**Waveform**



**Constraint File**



**Testbench code for 4-bit selectable**

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

Use ieee.std\_logic\_unsigned.all;

ENTITY bit\_adder\_4\_test IS

END bit\_adder\_4\_test;

ARCHITECTURE behavior OF bit\_adder\_4\_test IS

COMPONENT bit\_adder\_4

PORT(

A : IN std\_logic\_vector(3 downto 0);

B : IN std\_logic\_vector(3 downto 0);

M : in std\_logic;

s : OUT std\_logic\_vector(4 downto 0) );

END COMPONENT;

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

signal M : std\_logic := '0';

signal s : std\_logic\_vector(4 downto 0);

BEGIN

uut: bit\_adder\_4 PORT MAP (

A => test\_seq(7 downto 4),

B => test\_seq(3 downto 0),

M => M,

s => s);

process

begin

M <= '1' ;

test\_seq <= "01111111";

wait for 10 ns;

test\_seq <= "11111111";

wait for 10 ns;

test\_seq <= "01001110";

wait for 10 ns;

test\_seq <= "01101010";

wait for 10 ns;

test\_seq <= "00000001";

wait for 10 ns;

M <='0';

test\_seq <= "01010010";

wait for 10 ns;

test\_seq <= "11110000";

wait for 10 ns;

test\_seq <= "10100001";

wait for 10 ns;

test\_seq <= "10000111";

wait for 10 ns;

test\_seq <= "01110101";

wait;

end process;

END;

**Discussion**

From this lab, we know that a full adder is the main component in making an adder and a subtractor. The things that make a subtractor different are that we need to perform the first and the second compliments by putting inverters on the ‘B’ inputs and change the ‘carry in’ to high in order to make ‘B’ negative. To make the selectable, we created a new input ‘M’ and combined it with input ‘B’ into 4 XOR gates. By manipulating the input ‘M’, we can get different operations. Lastly, by knowing how the connections work in a 4-bit adder, we can put in more full adders and increase total bits input to produce a bigger desired outputs.