CS342 Lab 3 Report

Professor Gertner

FAHEEM KAMAL

2019

**Table of Contents:**

**Objective 2**

**Functionality 2 - 5**

**Simulation 5 - 18**

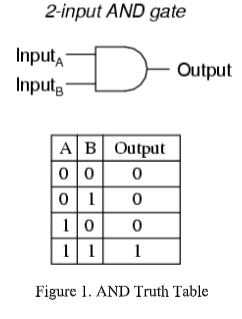
**Analysis 18**

**Conclusion 19**

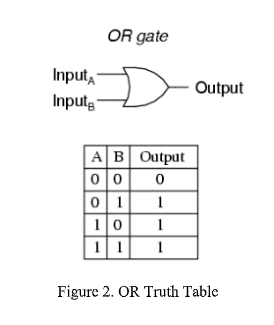
**Objective:**

In this lab, we had to create bitwise operations such as AND, OR, XOR, NOT, Shift and Right Rotations using VHDL files and testbench on Quartus and ModelSim. We will then test the designs on a DE2-115 board to see our results.

**Functionality:**

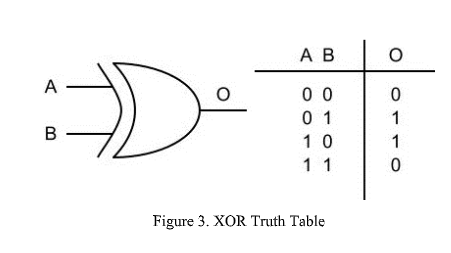
**AND operation**

The output of AND operation results in 1 for inputs that are 1. Bitwise AND performs an operation for each bit separately. Ex. Input 010110 and 010111 will go through the AND gate and result in 0101.10

**OR operation**

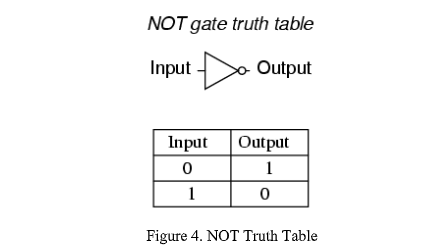
The output of OR will result in 0 for inputs that are both 0. Bitwise OR performs an operation for each bit separately. Ex: Input 010110 and 010111 will go through the OR gate and will result in 010111.

**XOR operation**



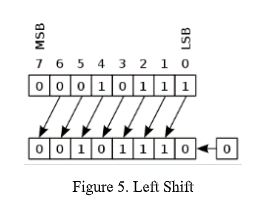
The XOR operations will result in 0 if the inputs are either 0 or 1. Other than that, it will be 1. Bitwise XOR performs operations for each bit separately. Ex: Two Inputs 010110 and 010111 will go through the bitwise XOR gate, then it will result in 101001.

**NOT operation**



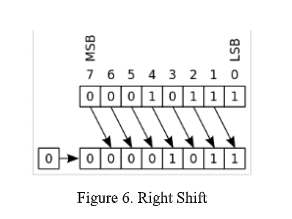
The NOT operations will result in a binary number where if the output is toggled from the input provided. When input is 1, the output will reflect as 0 and vice versa. Bitwise NOT performs operations for each bit separately. Ex: Input 010110 will go through the bitwise NOT gate, then it will result in 101001.

**Left Shift operation**



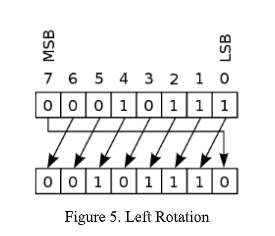
The Left Shift operation will result in a binary number that is shifted by 1 bit to the left from the input provided by the user. The least significant bit is filled up with a 0. Ex: For the input 010110 is applied, the output is then 001011.

**Right Shift operation**



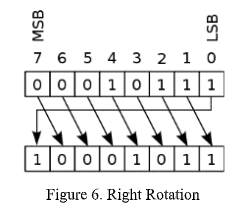
The Right Shift operation will result in a binary number that is shifted by 1 bit to the right from the input provided by the user. The most significant bit is filled up with a 0. Ex: For the input 010110 is applied, the output is then 101100.

**Left Rotation operation**



For the output of the Left Rotation operation, it results in a binary number where the output is shifted by 1 bit to the left from its input. The least significant bit is filled up with the input’s most significant value. Ex: Input is 010101, then the output is 101010.

**Right Rotation operation**



For the output of the Right Rotation operation, it results in a binary number where the output is shifted by 1 bit to the right from its input. The most significant bit is filled up with the input’s most significant value. Ex: Input is 010101, then the output is 101010.

**Simulation:**

The operations were created using a VHDL code. We have 4 input signals (Kamal\_input1, Kamal\_input2, Kamal\_opcode) and a start button, which is also an input signal (Kamal\_start). There is also a vector output, Kamal\_result from the operations. They were then tested using ModelSim and VHDL testbench code.

Kamal\_input1 and Kamal\_input2 serve as signal operands that take 6-bit inputs and Kamal\_start is the button that enables the operations. Kamal\_opcode identifies the operation that needs to be executed (ex: AND, OR, etc). We use the switches on the DE2-115 board to choose the operations as well as the specific inputs. We have 8 operations in total, therefore we use 3 switches. Kamal\_result shows the results from the inputs and operations. The following are used in the simulation:

**VHDL Code**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity Kamal\_Faheem\_Lab3\_Circuit is

Port(

Kamal\_input1: in std\_logic\_vector(5 downto 0);

Kamal\_input2: in std\_logic\_vector(5 downto 0);

Kamal\_start: in std\_logic;

Kamal\_opcode: in std\_logic\_vector(2 downto 0);

Kamal\_result: out std\_logic\_vector(5 downto 0)

);

end Kamal\_Faheem\_Lab3\_Circuit;

architecture Kamal\_arch of Kamal\_Faheem\_Lab3\_Circuit is

begin

process(Kamal\_start)

begin

if(Kamal\_start = '1') then

case Kamal\_opcode is

when "000" =>

Kamal\_result <= Kamal\_input1 and Kamal\_input2; -- AND Gate

when "001" =>

Kamal\_result <= Kamal\_input1 or Kamal\_input2; -- OR Gate

when "010" =>

Kamal\_result <= not Kamal\_input1; -- NOT Input1

when "011" =>

Kamal\_result <= Kamal\_input1 xor Kamal\_input2; -- XOR Gate

when "100" =>

Kamal\_result <= to\_stdlogicvector(to\_bitvector(Kamal\_input1) sll 1); -- Shift Left Logical by 1

when "101" =>

Kamal\_result <= to\_stdlogicvector(to\_bitvector(Kamal\_input1) srl 1); -- Shift Right Logical by 1

when "110" =>

Kamal\_result <= to\_stdlogicvector(to\_bitvector(Kamal\_input1) rol 1); -- Rotate Left by 1

when "111" =>

Kamal\_result <= to\_stdlogicvector(to\_bitvector(Kamal\_input1) ror 1); -- Rotate Right by 1

when others =>

NULL;

end case;

end if;

end process;

end Kamal\_arch;

**VHDL Testbench**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity Kamal\_Faheem\_Lab3\_CircuitTestBench is

end Kamal\_Faheem\_Lab3\_CircuitTestBench;

architecture arch of Kamal\_Faheem\_Lab3\_CircuitTestBench is

component Kamal\_Faheem\_Lab3\_Circuit

port(

Kamal\_input1: in std\_logic\_vector(5 downto 0);

Kamal\_input2: in std\_logic\_vector(5 downto 0);

Kamal\_start: in std\_logic;

Kamal\_opcode: in std\_logic\_vector(2 downto 0);

Kamal\_result: out std\_logic\_vector(5 downto 0)

);

end component;

--Inputs

signal Kamal\_input1: std\_logic\_vector(5 downto 0) := (others => '0');

signal Kamal\_input2: std\_logic\_vector(5 downto 0) := (others => '0');

signal Kamal\_opcode: std\_logic\_vector(2 downto 0) := (others => '0');

signal Kamal\_start: std\_logic;

--Outputs

signal Kamal\_result: std\_logic\_vector(5 downto 0);

begin

uut: Kamal\_Faheem\_Lab3\_Circuit port map(

Kamal\_input1 => Kamal\_input1,

Kamal\_input2 => Kamal\_input2,

Kamal\_opcode => Kamal\_opcode,

Kamal\_start => Kamal\_start,

Kamal\_result => Kamal\_result

);

stim\_proc: process

begin

wait for 100 ns;

Kamal\_input1 <= "010110";

Kamal\_input2 <= "010111";

Kamal\_start <= '1';

Kamal\_opcode <= "000";

wait for 100 ns;

Kamal\_opcode <= "001";

wait for 100 ns;

Kamal\_opcode <= "010";

wait for 100 ns;

Kamal\_opcode <= "011";

wait for 100 ns;

Kamal\_opcode <= "100";

wait for 100 ns;

Kamal\_opcode <= "101";

wait for 100 ns;

Kamal\_opcode <= "110";

wait for 100 ns;

Kamal\_opcode <= "111";

end process;

end;

**Pin Assignments:**

To, Location

Kamal\_input1[0], PIN\_AB28

Kamal\_input1[1], PIN\_AC28

Kamal\_input1[2], PIN\_AC27

Kamal\_input1[3], PIN\_AD27

Kamal\_input1[4], PIN\_AB27

Kamal\_input1[5], PIN\_AC26

Kamal\_input2[0], PIN\_AD26

Kamal\_input2[1], PIN\_AB26

Kamal\_input2[2], PIN\_AC25

Kamal\_input2[3], PIN\_AB25

Kamal\_input2[4], PIN\_AC24

Kamal\_input2[5], PIN\_AB24

Kamal\_start, PIN\_M23

Kamal\_opcode[2], PIN\_Y23

Kamal\_opcode[1], PIN\_Y24

Kamal\_opcode[0], PIN\_AA22

Kamal\_result[0], PIN\_E21

Kamal\_result[1], PIN\_E22

Kamal\_result[2], PIN\_E25

Kamal\_result[3], PIN\_E24

Kamal\_result[4], PIN\_H21

Kamal\_result[5], PIN\_G20

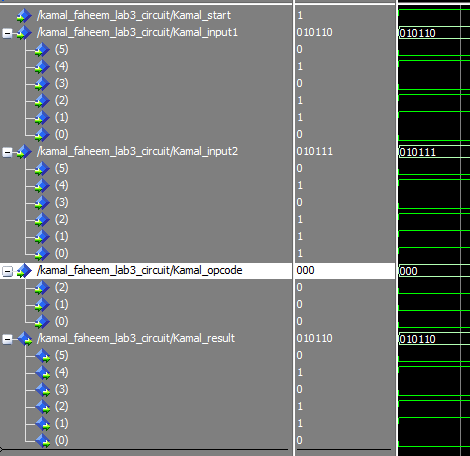
**Inputs:** Kamal\_input1, Kamal\_input2, Kamal\_opcode and Kamal\_start

**Outputs:** Kamal\_result

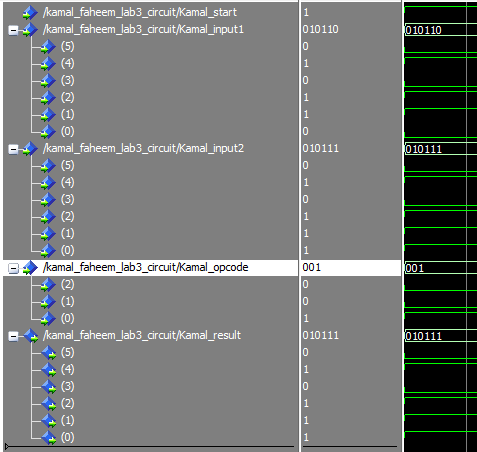
The figure above shows the opcode used in the lab and each code was set to a specific operation.

|  |  |
| --- | --- |
| **Opcode** | **Operation** |
| 000 | Result = A and B |
| 001 | Result = A or B |
| 010 | Result = not A |
| 011 | Result = A xor B |
| 100 | Result = A << shamt (Left Shift) |
| 101 | Result = A >> shamt (Right Shift) |
| 110 | Result = A rol (Rotation Left) |
| 111 | Result = A ror (Rotation Right) |

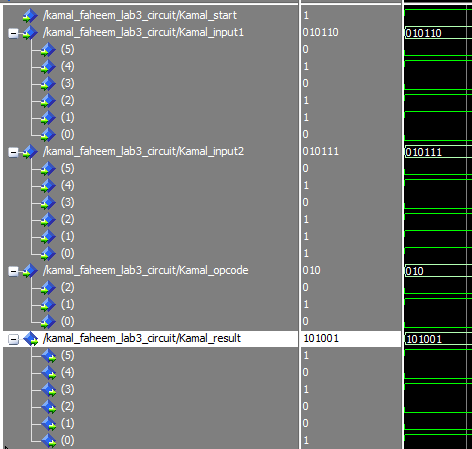
**Opcode 000 (AND)**



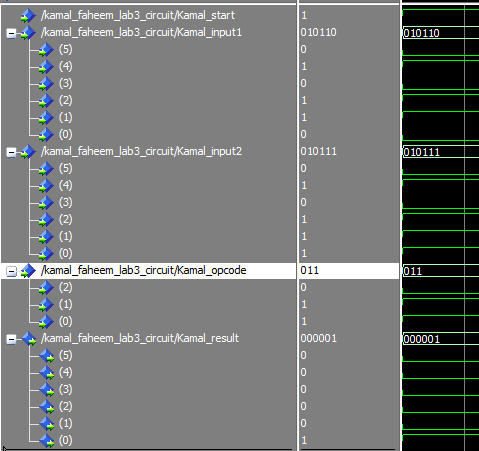
**Opcode 001 (OR)**



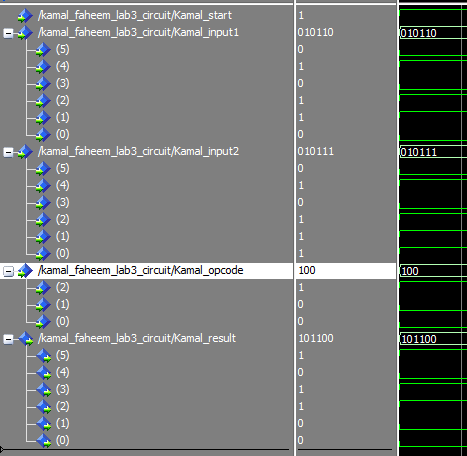
**Opcode 010 (NOT)**



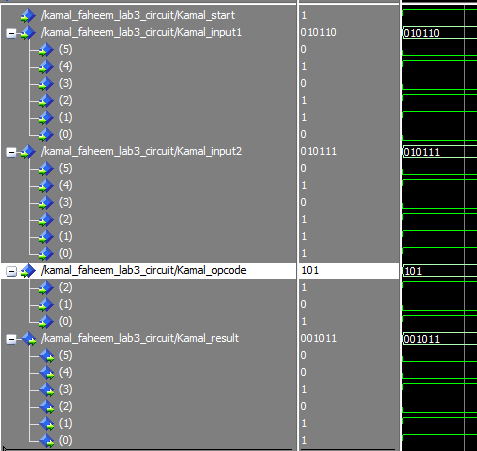
**Opcode 011 (XOR)**



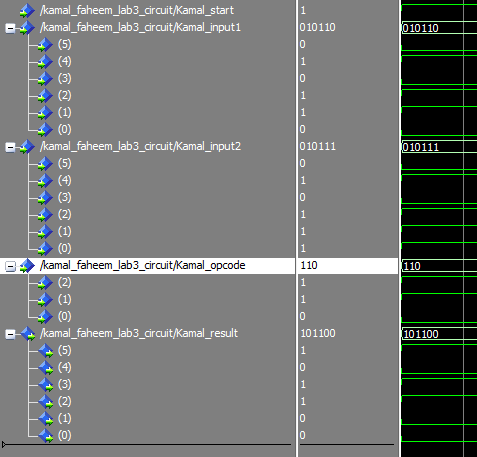
**Opcode 100 (Left Shift)**



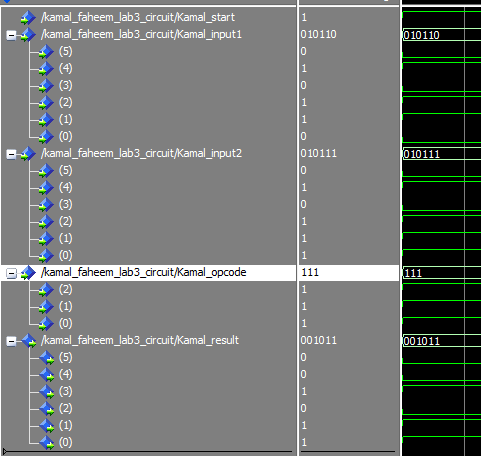
**Opcode 101 (Right Shift)**



**Opcode 110 (Left Rotation)**

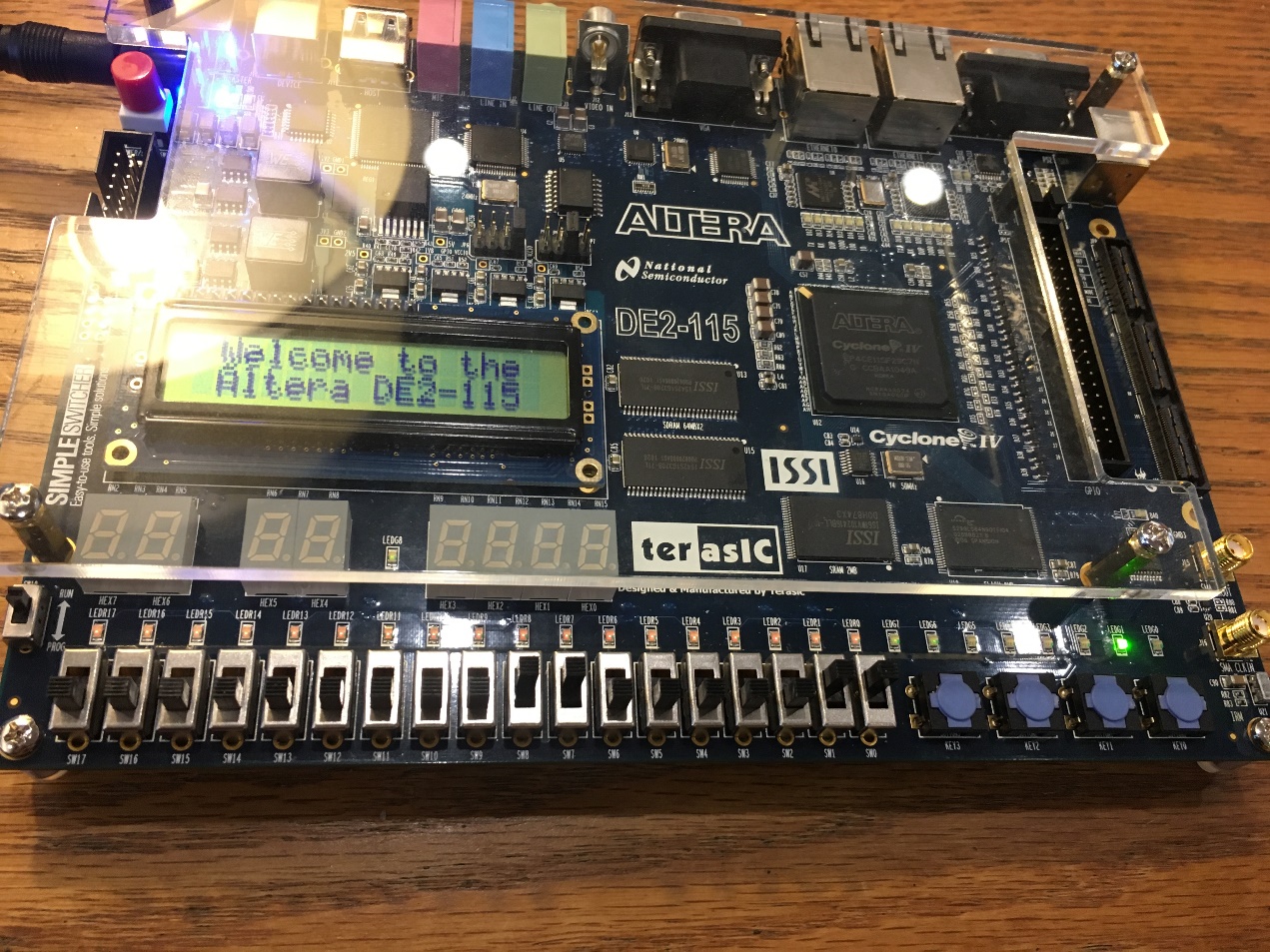


**Opcode 111 (Right Rotation)**

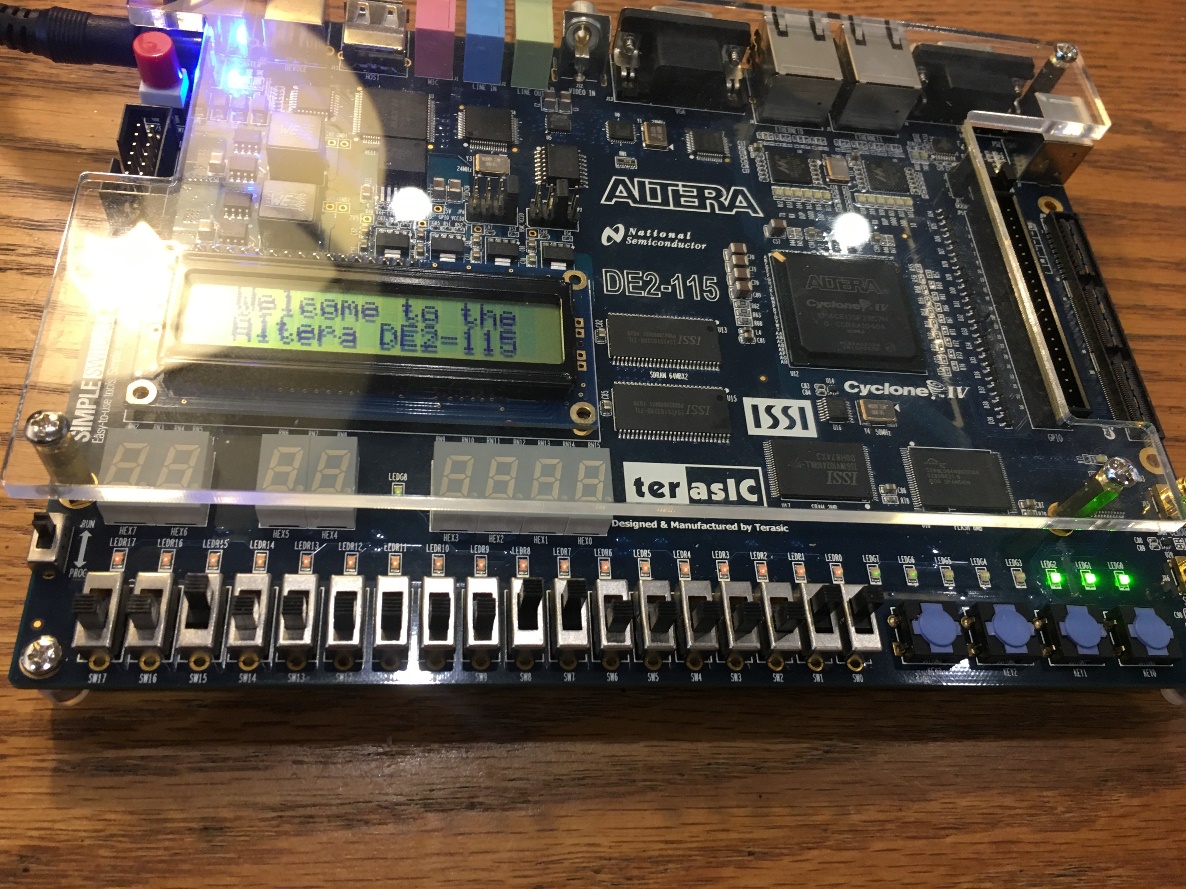


The figures below show the pin assignments used for the DE2-115 Board. Kamal\_input1 and Kamal\_input2 were assigned to switches 0 to 11, Kamal\_opcode was assigned to switches 15 to 17. Kamal\_start was assigned to key 0 and the output Kamal\_result was assigned to green LEDs 0 to 5 on the board.

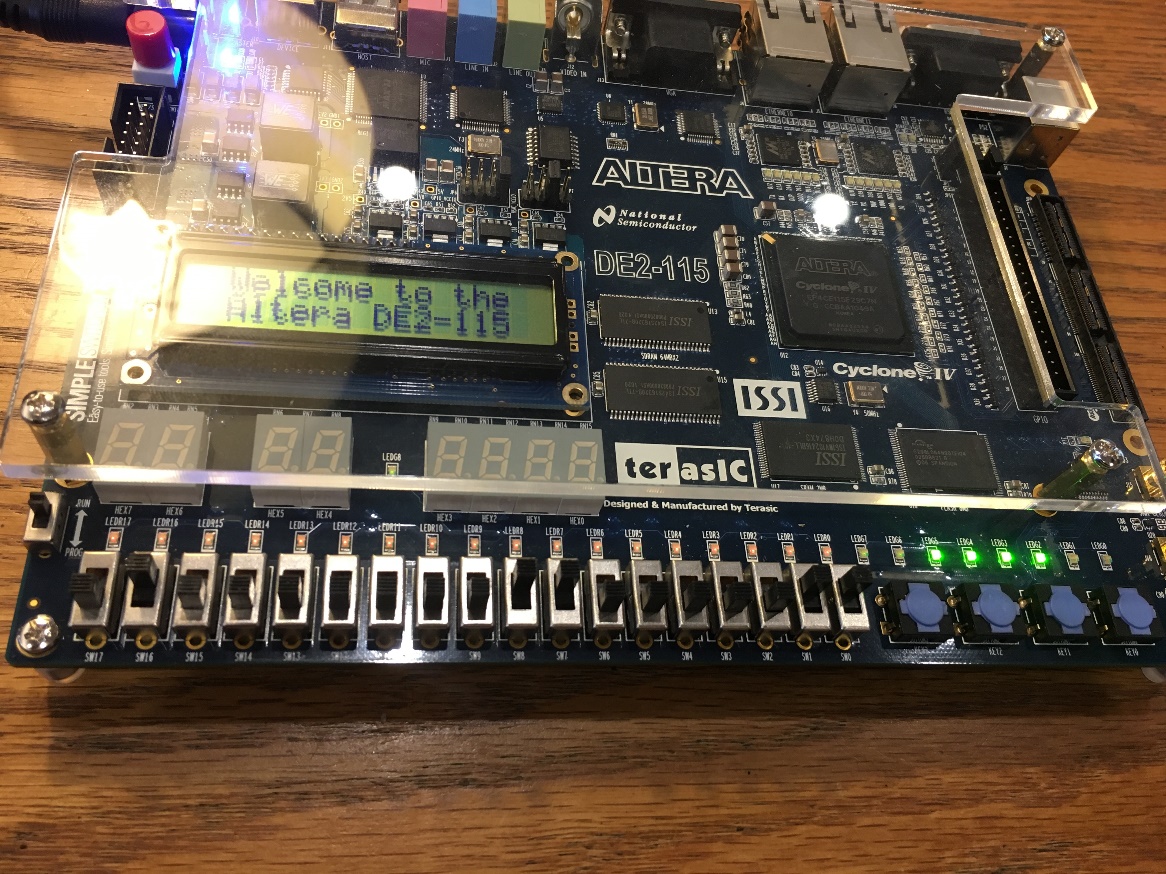
**DE2-115 AND Opcode**



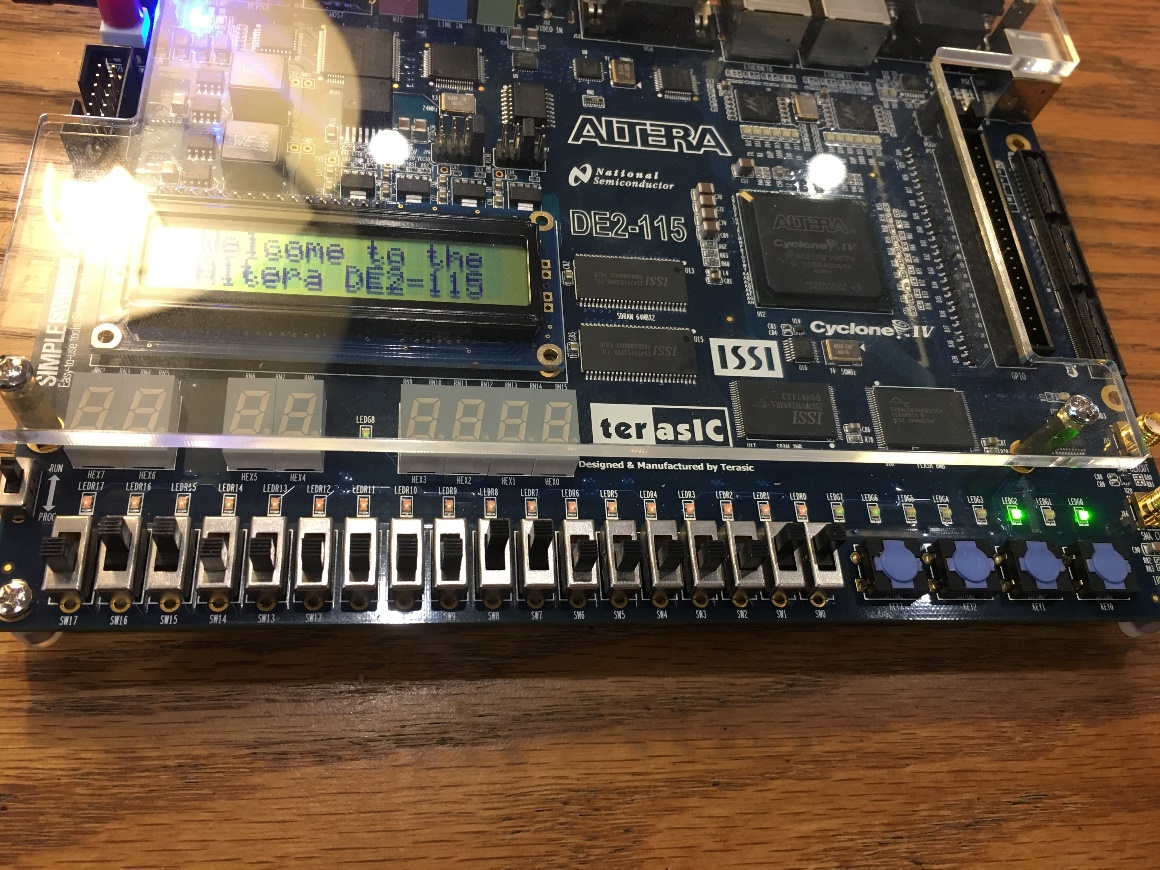
**DE2-115 OR Opcode**



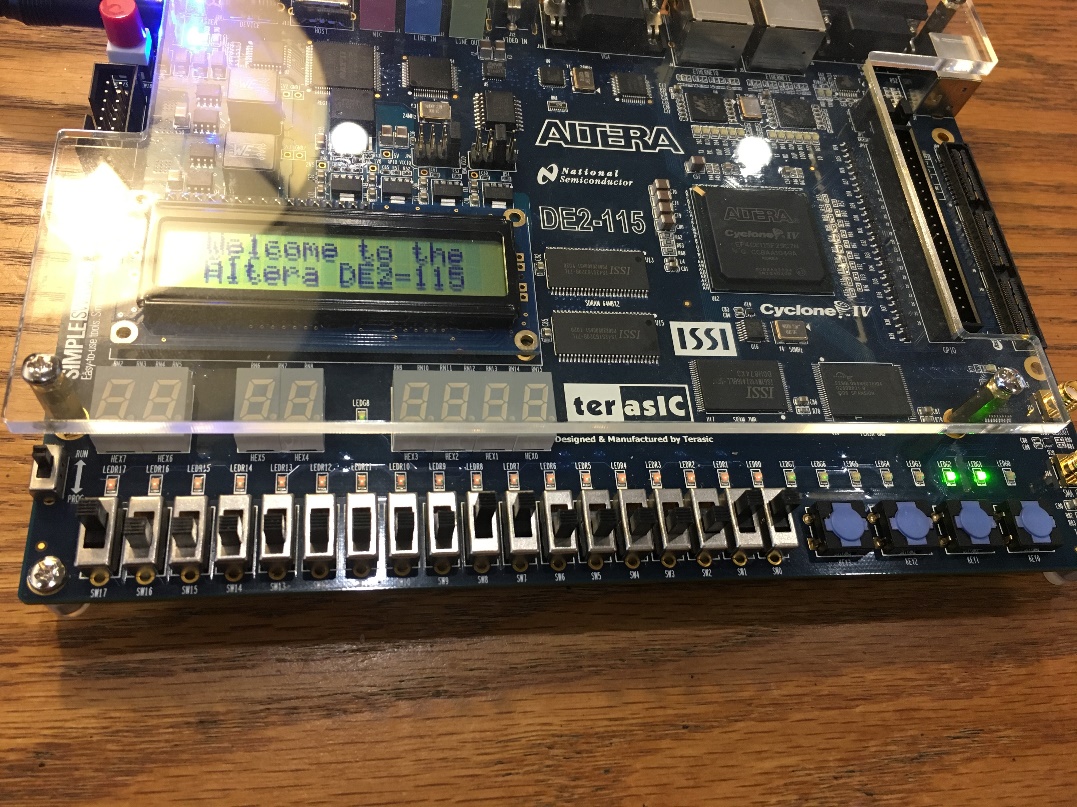
**DE2-115 XOR Opcode**

****

**DE2-115 NOT Opcode**

****

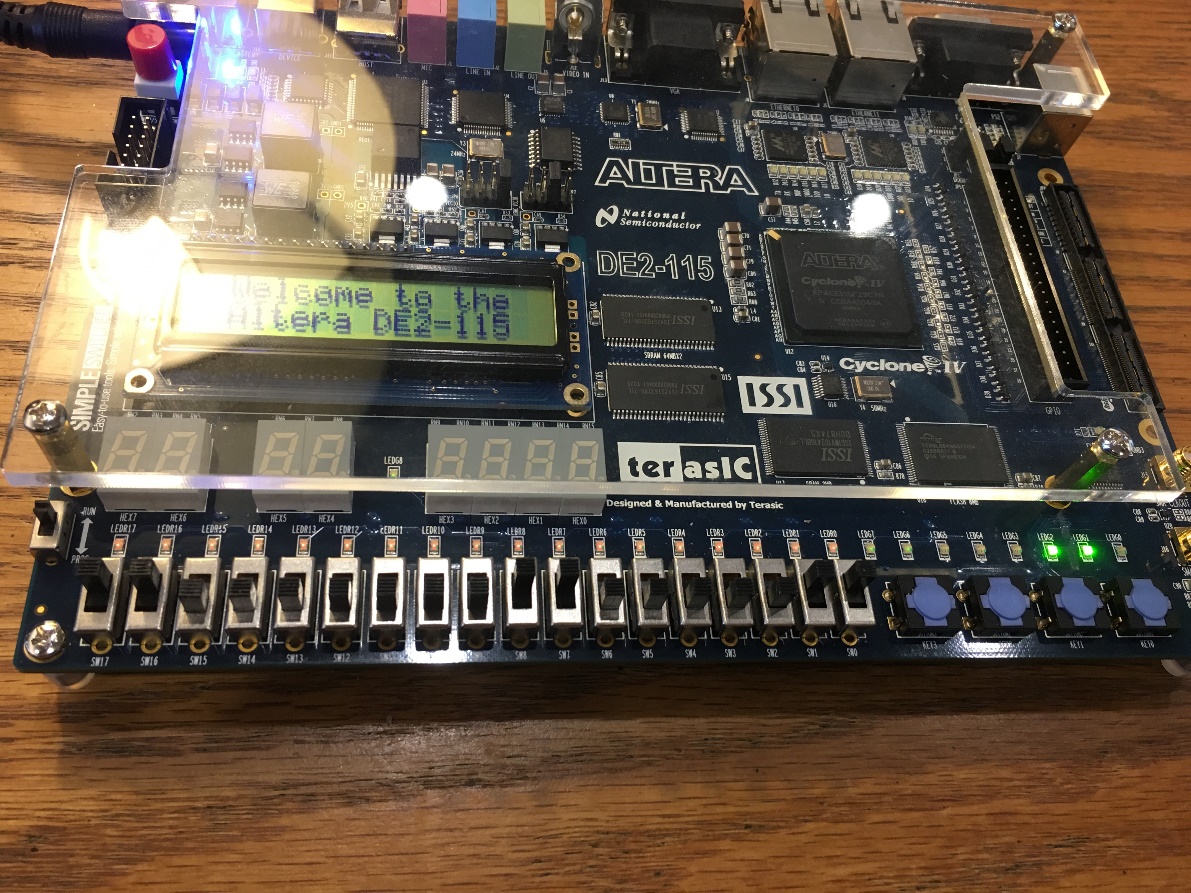
**DE2-115 Left Shift Opcode**

****

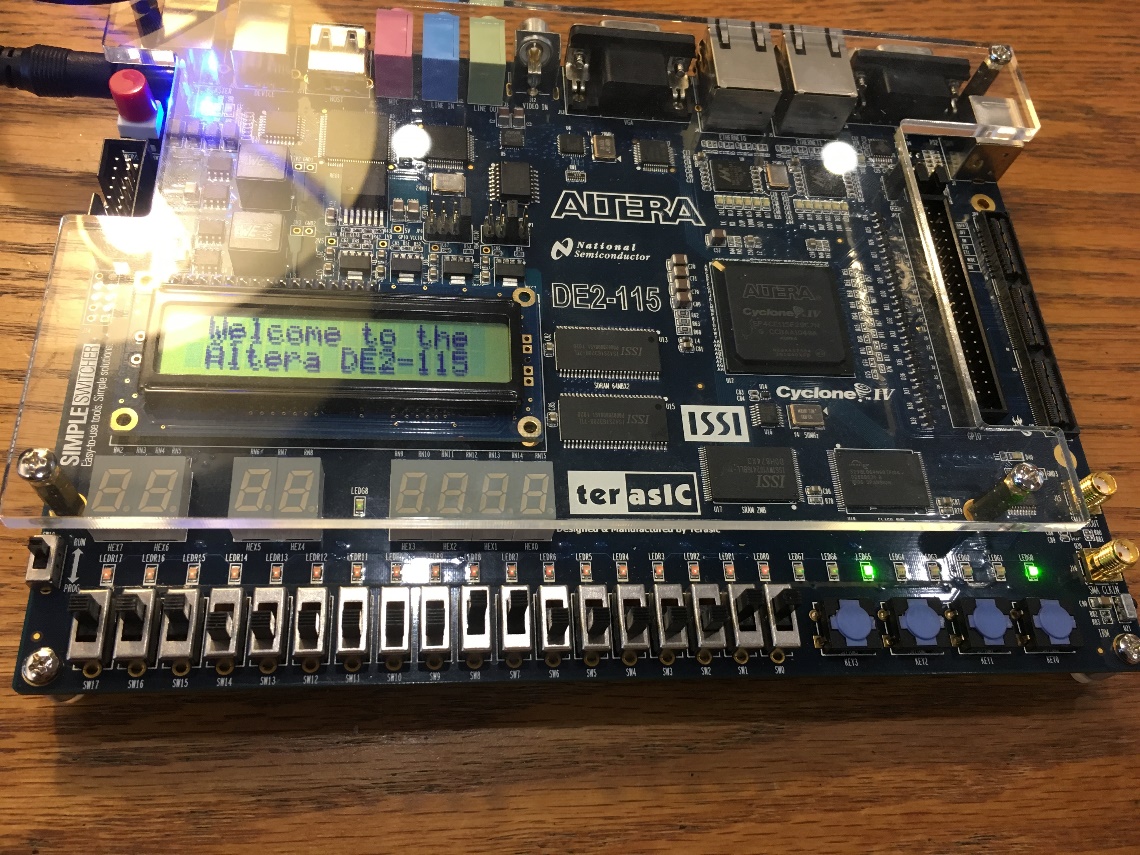
**DE2-115 Right Shift Opcode**

****

**DE2-115 Left Rotation Opcode**

****

**DE2-115 Right Rotation Opcode**

****

**Analysis:**

For the operations implemented in this lab using VHDL, these can be implemented in high-level languages such as C++, C or Java. We have such operators (&, |, *<<*, >>, ~, ^) can be used as bitwise operators. Using loops for the left and right rotations are useful in a way where you use temporary variables to store values and move their locations in either direction.

In order to multiply by 2, we can use the Left Shift operation since the operations results with two of the same inputs being added together. This only applies with binary numbers since with binary numbers, moving to the left or right requires only multiplication or division by 2. For other systems, such as octal, we need to use 8 to move to the right or left bit. Another such example, with decimal, we need to use 10 to move to the right or left.

For multiplication or division by multiplies of 2, we can use left and right shift operations. If we want to divide or multiply by 2, we can make it by shifting to the left or right by 1 bit. If we want to multiply by 2x2 or 2^2, we can shift to the left or right by 2 bits. So, we can perform this by shifting bits by n/2.

**Conclusion:**

This lab is essential in helping me understand how bitwise operations work. I learned that the start button acts as a trigger that allows the operation that I select to execute. Furthermore, I learned to use similar operations successfully on paper but with this lab, each operation gave me the results I had expected.