Lab #3

Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Name: Chen, Kevin

Date: 04/01/2019

Course: CSC 34200-G & CSC 34300-DE

Table of Contents

Objective:

The goal of this lab is to develop a circuit that performs bitwise operations, bit shifts and bit rotations. We will be creating a total of eight functions, four bitwise operations, and, or, xor, not; two bit shifts, shift left and shift right; two bit rotations, rotation left, and rotation right. These functions are performed by a selector that determines which operation to perform at a given time. The selector would be part of the operation file, which is what would control the operation of the circuit. That file would also have an implementation of a less than function, which checks the inputs as signed numbers and compares them to see if one is less than another. The inputs that we would be comparing are 6-bit inputs and would result in a 6-bit output. The selector would be a 3-bit input, which each input would perform a different function. All theses inputs and outputs would be connected to the operation file and would be triggered by the start input.

Functionality and Specification:

Bitwise AND:

A Bitwise AND function compares two inputs and lets out a result of one output. It compares the inputs bit by bit and would output ‘1’ when the two bits are ‘1’, otherwise, it’s ‘0’.

The VHDL code written for Bitwise AND is shown on figure 25.

The truth table and example are shown below:

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Table 1: Truth Table for Bitwise AND

Input	101001
Function	AND
Input	110110
Output	100000

Table 2: Example for Bitwise AND

Bitwise OR:

A Bitwise OR function compares two inputs and lets out a result of one output. It compares the inputs bit by bit and would output ‘1’ when any of the two bits are ‘1’, otherwise, it’s ‘0’.

The VHDL code written for Bitwise OR is shown on figure 27.

The truth table and example are shown below:

A	**B**	**Output**
0	0	0
0	1	1
1	0	1
1	1	1

Table 3: Truth Table for Bitwise OR

Input	101001
Function	OR
Input	110110
Output	111111

Table 4: Example for Bitwise OR

Bitwise XOR:

A Bitwise XOR function compares two inputs and lets out a result of one output. It compares the inputs bit by bit and would output ‘1’ when the two bits are both ‘1’ and ‘0’, otherwise, it’s ‘0’.

The VHDL code written for Bitwise XOR is shown on figure 29.

The truth table and example are shown below:

A	**B**	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

Table 5: Truth Table for Bitwise XOR

Input	101001
Function	XOR
Input	110110
Output	011111

Table 6: Example for Bitwise XOR

Bitwise NOT:

A Bitwise NOT function takes in one input and lets out a result of one output. The output is ‘1’ when input is 0 and ‘0’ when input is 1.

The VHDL code written for Bitwise NOT is shown on figure 31.

The truth table and example are shown below:

A	**Output**
0	1
1	0

Table 7: Truth Table for Bitwise NOT

Input	101001
Function	NOT
Output	010110

Table 8: Example for Bitwise XOR

Shift Left:

A Shift Left function takes in one input and lets out a result of one output. The output is the input shifted to the most significant bit. The input’s most significant bit goes pushed out and 0 gets pushed into the least significant bit.

The VHDL code written for Shift Left is shown on figure 33.

The diagram and example are shown below:

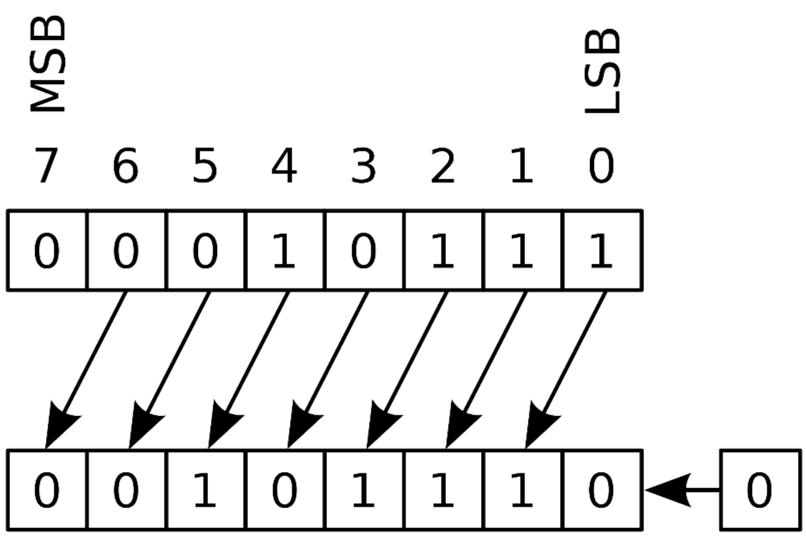

Figure 1: Diagram for Shift Left

Input	101001
Function	Shift Left
Output	010010

Table 9: Example for Shift Left

<u>Shift Right:</u>

A Shift Right function takes in one input and lets out a result of one output. The output is the input shifted to the least significant bit. The input's least significant bit goes pushed out and 0 gets pushed into the most significant bit.

The VHDL code written for Shift Right is shown on figure 35.

The diagram and example are shown below:

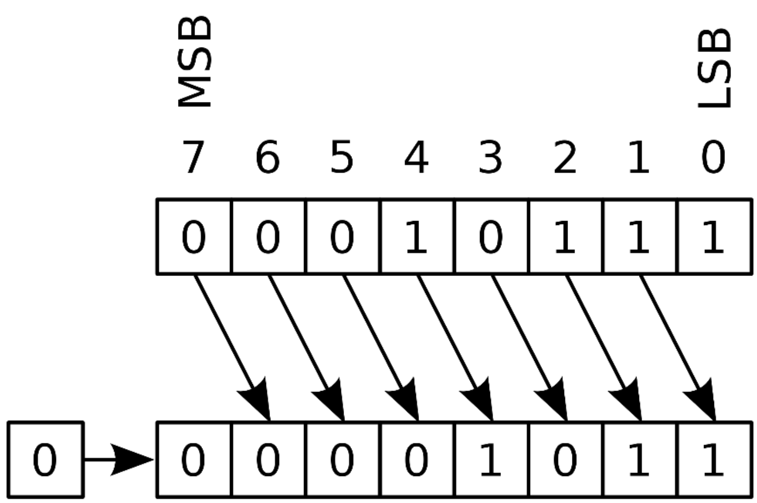

Figure 2: Diagram for Shift Right

Input	101001
Function	Shift Right
Output	010100

Table 10: Example for Shift Right

<u>Rotation Left:</u>

A Rotation Left function takes in one input and lets out a result of one output. The output is the input rotated to the most significant bit. The input's most significant bit goes to the least significant bit.

The VHDL code written for Rotation Left is shown on figure 37.

The diagram and example are shown below:

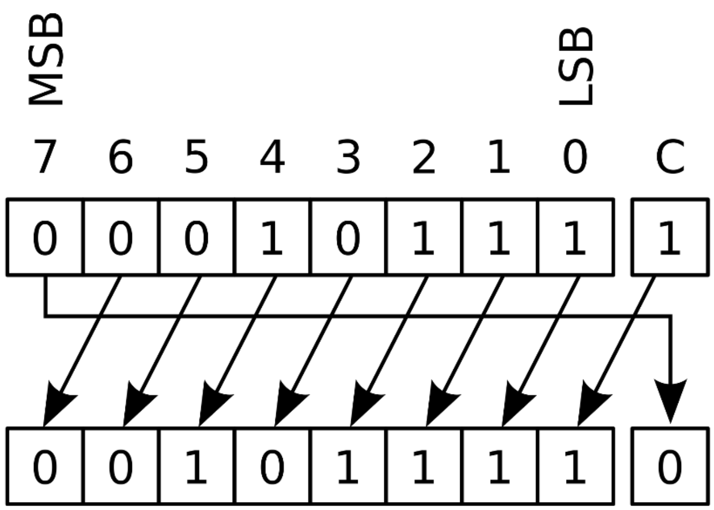

Figure 3: Diagram for Rotation Left

Input	101001
Function	Rotation Left
Output	010011

Table 11: Example for Rotation Left

Rotation Right:

A Rotation Right function takes in one input and lets out a result of one output. The output is the input rotated to the least significant bit. The input's least significant bit goes to the most significant bit.

The VHDL code written for Rotation Right is shown on figure 39.

The diagram and example are shown below:

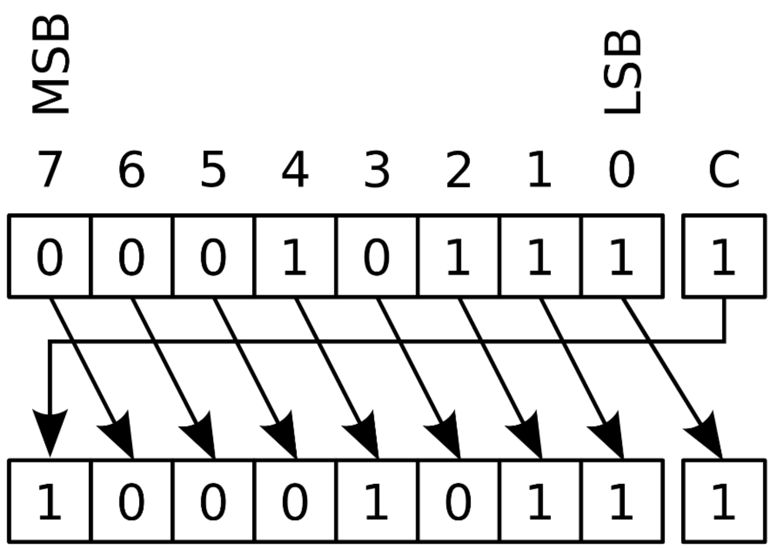

Figure 4: Diagram for Rotation Right

Input	101001
Function	Rotation Right
Output	110100

Table 12: Example for Rotation Right

Operations:

The Operations function takes in sixteen inputs and lets out a result of seven outputs. Six of the outputs is determined based on which function it uses, the start key, and the given inputs. Depending on the "op" input, it's able to select which function to use, bitwise and, bitwise or, bitwise xor, bitwise not, shift left, shift right, rotation left, or rotation right. When "Start" is '1', it changes the output based on the function it's using, provided with the given inputs, otherwise the output remains the same. We need a "Start" input because we need a trigger for the circuit to switch the output. The last output is determined based on the two 6-bit inputs, it checks to see if the first 6-bit input is less than the second 6-bit input, in signed. When the first 6-bit input is less than the second 6-bit input then the output would be'1', otherwise it's '0'.

The VHDL code written for Operations is shown is figure 41.

The block diagram and example are shown below:

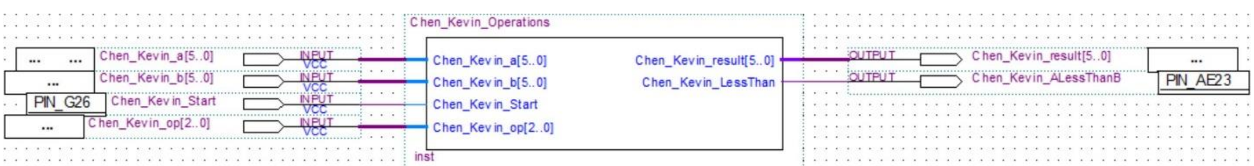

Figure 5: Block Diagram/Schematic File of Operations (Chen_Kevin_Operations.BDF)

A	B	Start	OP	Function	Result	A < B
101001	110110	1	000	AND	100000	1
101001	110110	1	101	Shift Left	010010	1

Table 13: Example for Operations

Simulations:

Bitwise AND:

The functionality/behavior of Bitwise AND is to follow the truth table (Table 1), which describes exactly what the output is supposed to be given a certain input.

A ModelSim waveform was created using the testbench written in VHDL for Bitwise AND (Figure 6). The testbench written provided the inputs to test all possible combinations. The wave generated matches those from table 1, therefore, confirming the correctness of the VHDL code.

Figure 6.a: ModelSim of VHDL Code for Bitwise AND

Figure 6.b: ModelSim of VHDL Code for Bitwise AND

<u>Bitwise OR:</u>

The functionality/behavior of Bitwise OR is to follow the truth table (Table 3), which describes exactly what the output is supposed to be given a certain input.

A ModelSim waveform was created using the testbench written in VHDL for Bitwise OR (Figure 7). The testbench written provided the inputs to test all possible combinations. The wave generated matches those from table 3, therefore, confirming the correctness of the VHDL code.

Figure 7.a: ModelSim of VHDL Code for Bitwise OR

Figure 7.b: ModelSim of VHDL Code for Bitwise OR

<u>Bitwise XOR:</u>

The functionality/behavior of Bitwise XOR is to follow the truth table (Table 5), which describes exactly what the output is supposed to be given a certain input.

A ModelSim waveform was created using the testbench written in VHDL for Bitwise XOR (Figure 8). The testbench written provided the inputs to test all possible combinations. The wave generated matches those from table 5, therefore, confirming the correctness of the VHDL code.

Figure 8.a: ModelSim of VHDL Code for Bitwise XOR

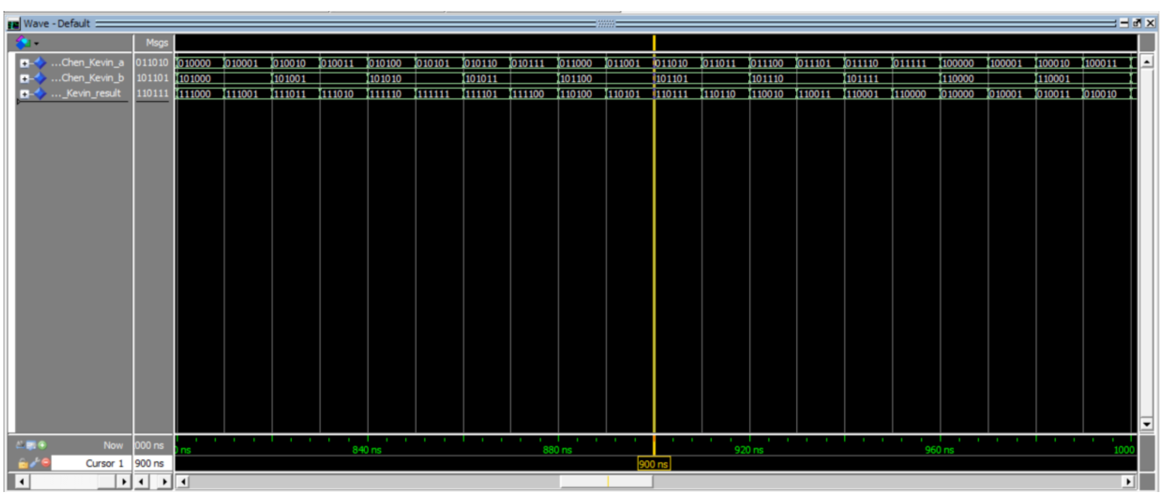

Figure 8.b: ModelSim of VHDL Code for Bitwise XOR

<u>Bitwise NOT:</u>

The functionality/behavior of Bitwise NOT is to follow the truth table (Table 7), which describes exactly what the output is supposed to be given a certain input.

A ModelSim waveform was created using the testbench written in VHDL for Bitwise NOT (Figure 9). The testbench written provided the inputs to test all possible combinations. The wave generated matches those from table 7, therefore, confirming the correctness of the VHDL code.

Figure 9.a: ModelSim of VHDL Code for Bitwise NOT

Figure 9.b: ModelSim of VHDL Code for Bitwise NOT

Shift Left:

The functionality/behavior of Shift Left is to follow the diagram (Figure 1), which describes exactly what the output is supposed to be given a certain input.

A ModelSim waveform was created using the testbench written in VHDL for Shift Left (Figure 10). The testbench written provided the inputs to test all possible combinations. The wave generated matches those from figure 1, therefore, confirming the correctness of the VHDL code.

Figure 10.a: ModelSim of VHDL Code for Shift Left

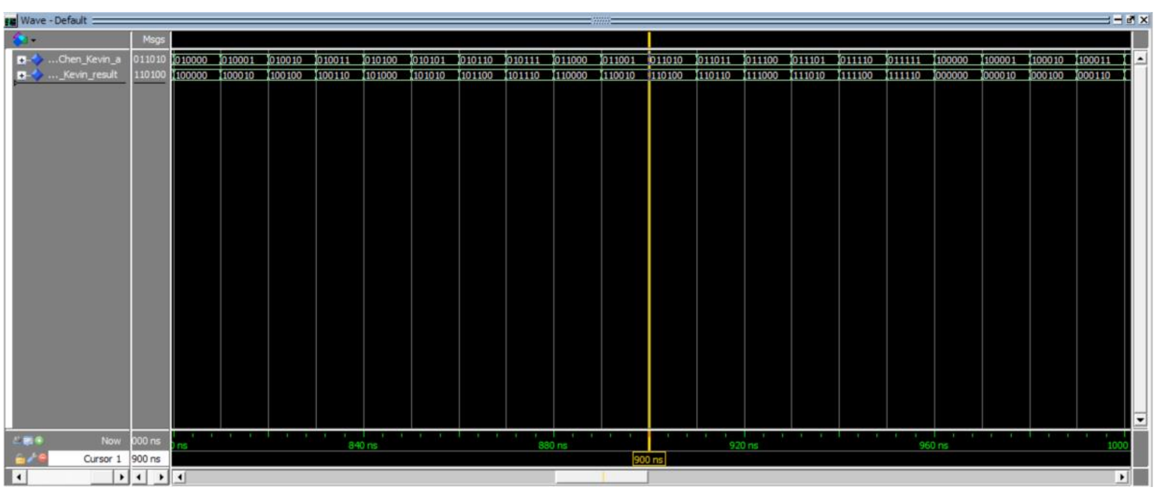

Figure 10.b: ModelSim of VHDL Code for Shift Left

Shift Right:

The functionality/behavior of Shift Right is to follow the diagram (Figure 2), which describes exactly what the output is supposed to be given a certain input.

A ModelSim waveform was created using the testbench written in VHDL for Shift Right (Figure 11). The testbench written provided the inputs to test all possible combinations. The wave generated matches those from figure 2, therefore, confirming the correctness of the VHDL code.

Figure 11.a: ModelSim of VHDL Code for Shift Right

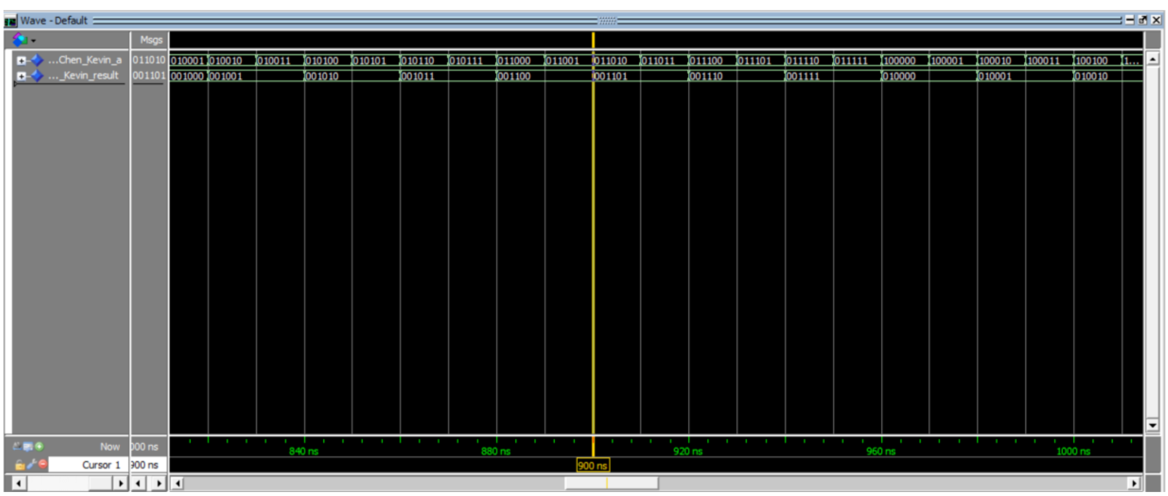

Figure 11.b: ModelSim of VHDL Code for Shift Right

<u>Rotation Left:</u>

The functionality/behavior of Rotation Left is to follow the diagram (Figure 3), which describes exactly what the output is supposed to be given a certain input.

A ModelSim waveform was created using the testbench written in VHDL for Rotation Left (Figure 12). The testbench written provided the inputs to test all possible combinations. The wave generated matches those from figure 3, therefore, confirming the correctness of the VHDL code.

Figure 12.a: ModelSim of VHDL Code for Rotation Left

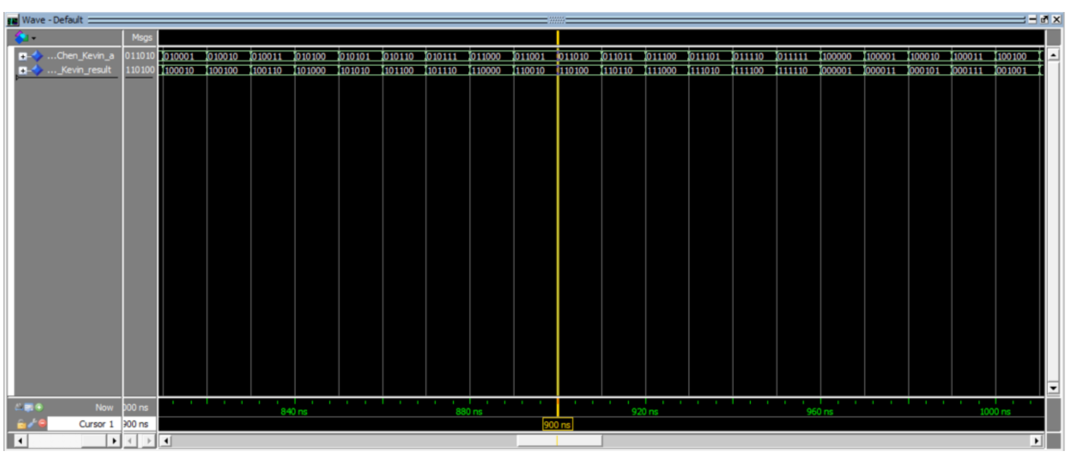

Figure 12.b: ModelSim of VHDL Code for Rotation Left

Rotation Right:

The functionality/behavior of Rotation Right is to follow the diagram (Figure 4), which describes exactly what the output is supposed to be given a certain input.

A ModelSim waveform was created using the testbench written in VHDL for Rotation Right (Figure 13). The testbench written provided the inputs to test all possible combinations. The wave generated matches those from figure 4, therefore, confirming the correctness of the VHDL code.

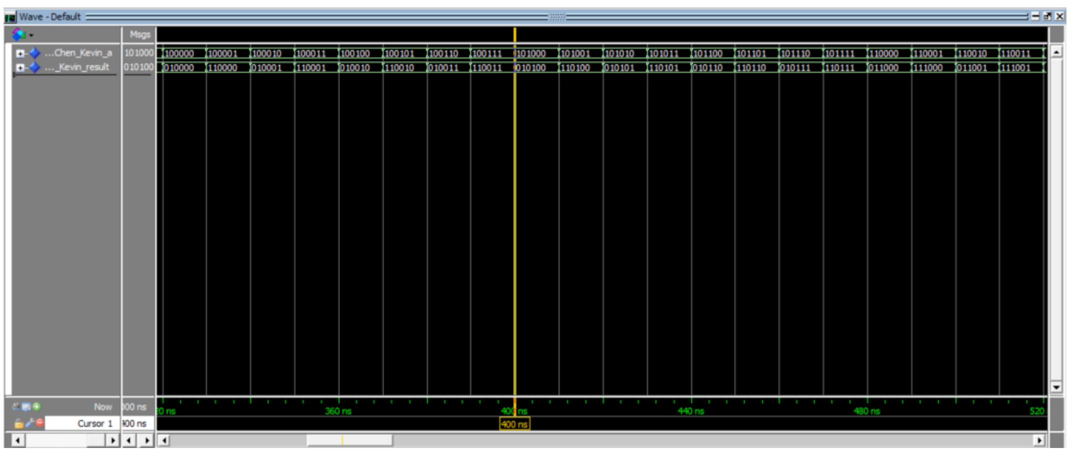

Figure 13.a: ModelSim of VHDL Code for Rotation Right

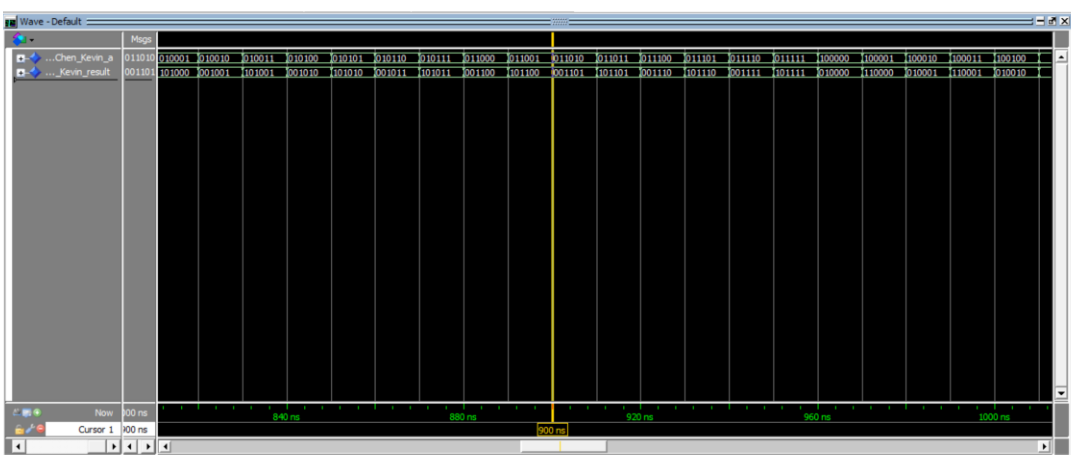

Figure 13.b: ModelSim of VHDL Code for Rotation Right

<u>Operations:</u>

The functionality/behavior of Operations is to operate which function to use and produce an output based on the given inputs. It would also check if one of the two 6-bit inputs are less than the other and if it is, it would output ‘1’.

A ModelSim waveform was created using the testbench written in VHDL for Operations (Figure 14). The testbench written provided the inputs to test some combinations. The wave generated matches those from figure 15, provided by the professor, therefore, confirming the correctness of the VHDL code.

Figure 14: ModelSim of VHDL Code for Operations

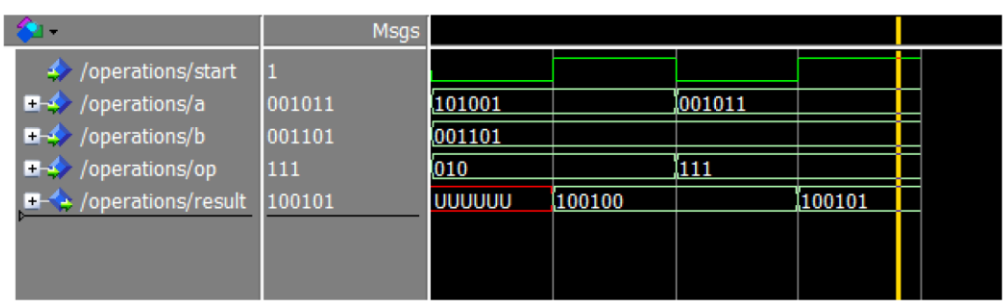

Figure 15: ModelSim of Operations Provided by the Professor

Demonstration:

Operations:

In figures 16 to 24, demonstrations using all the operations are shown in addition to showing the less than for comparing two signed numbers.

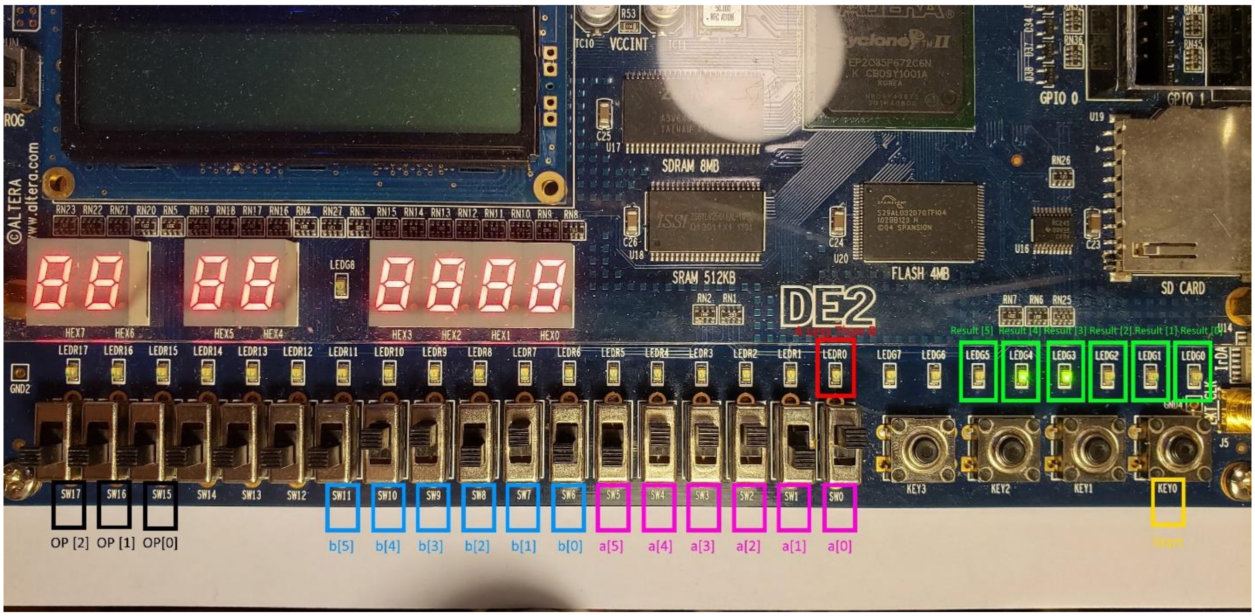

Figure 16: AND Operation

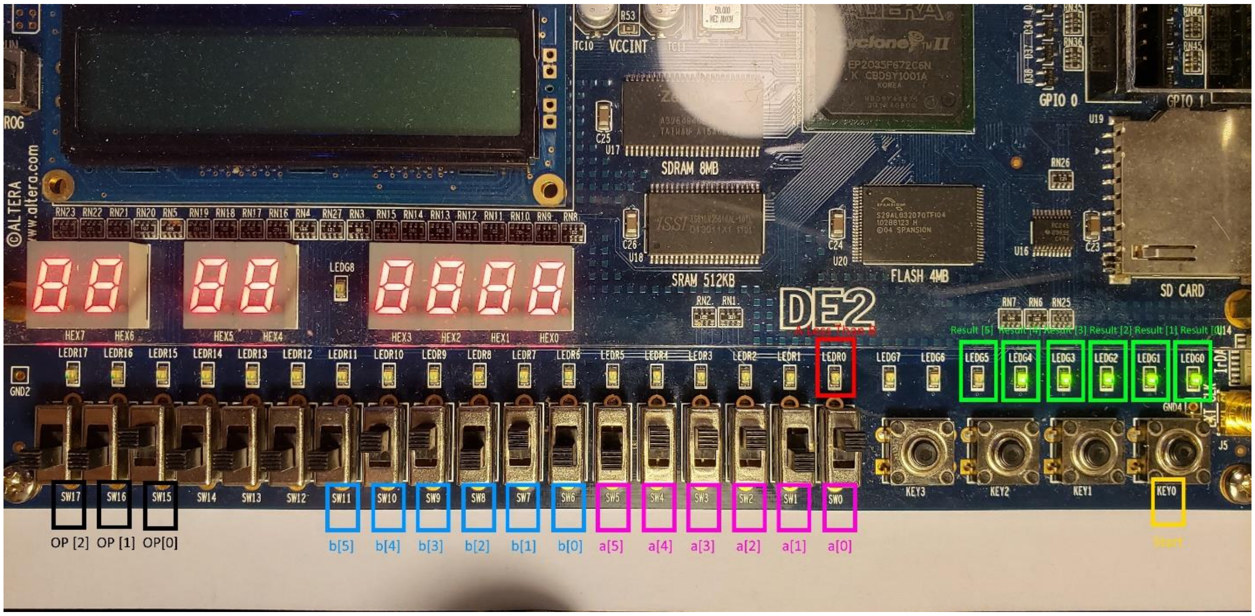

Figure 17: OR Operation

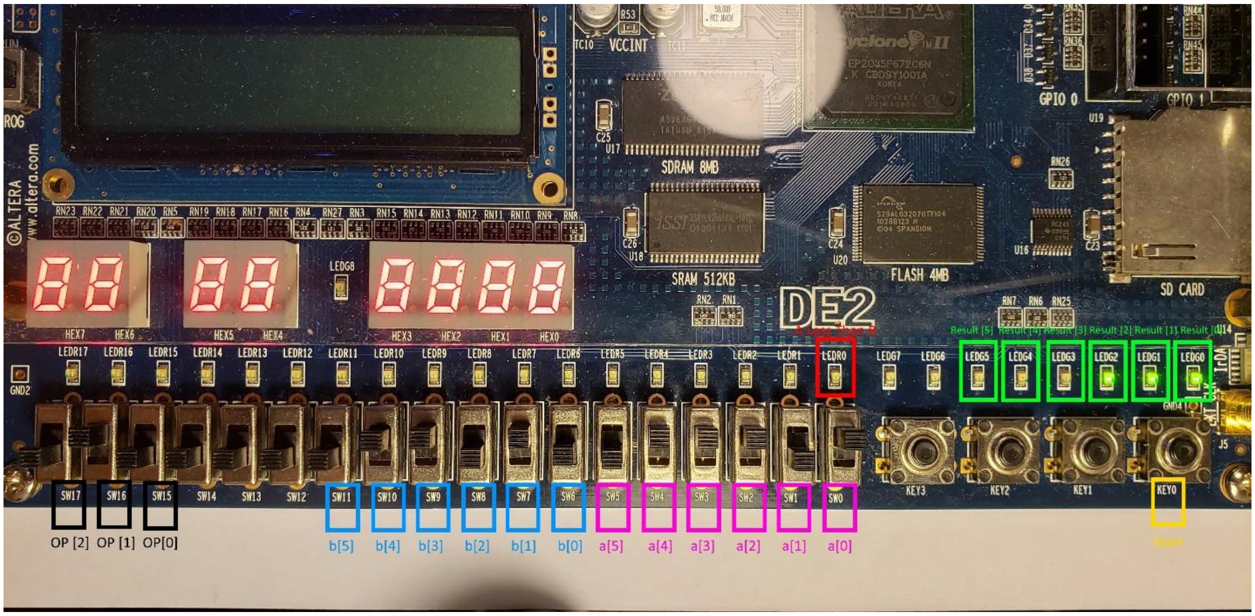

Figure 18: XOR Operation

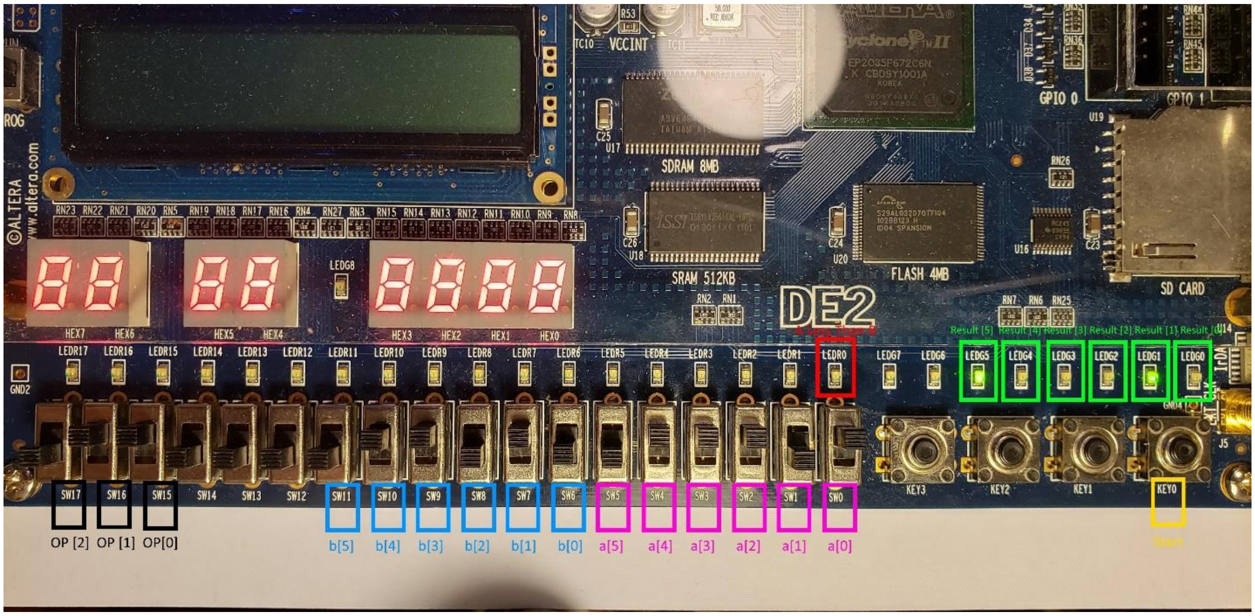

Figure 19: NOT Operation

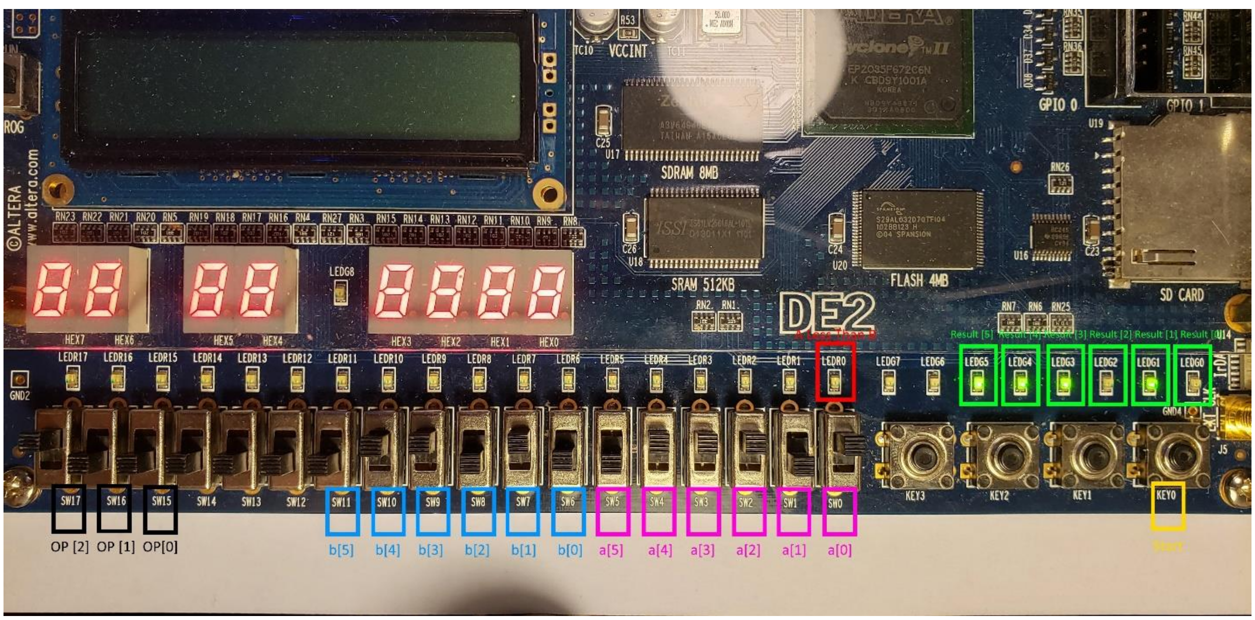

Figure 20: Shift Left Operation

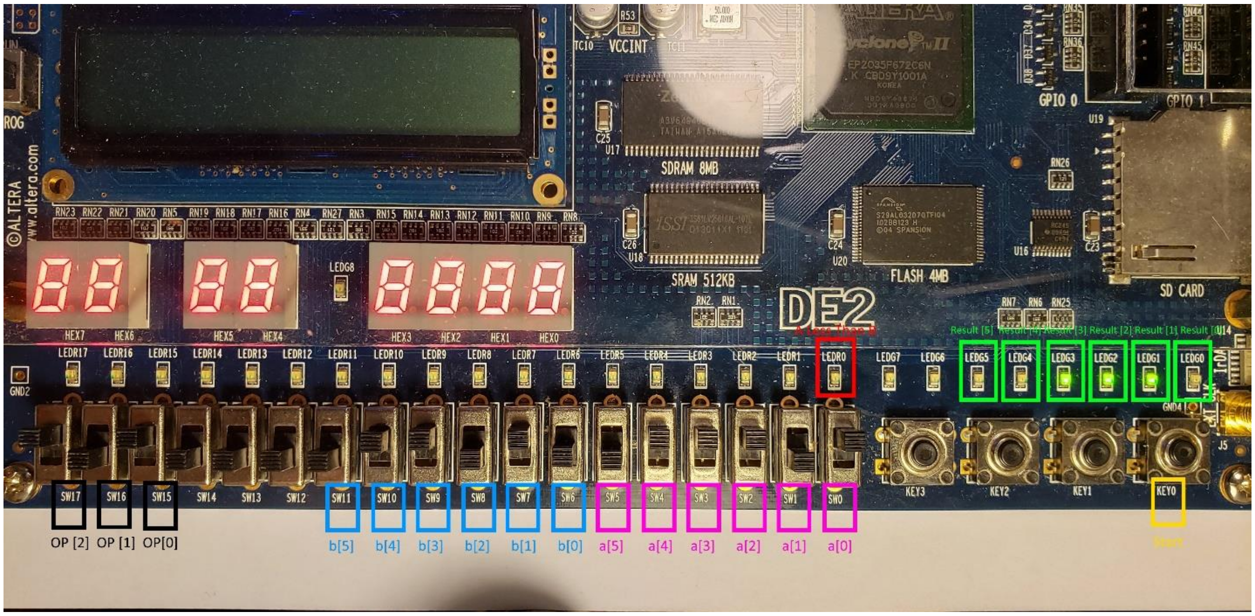

Figure 21: Shift Right Operation

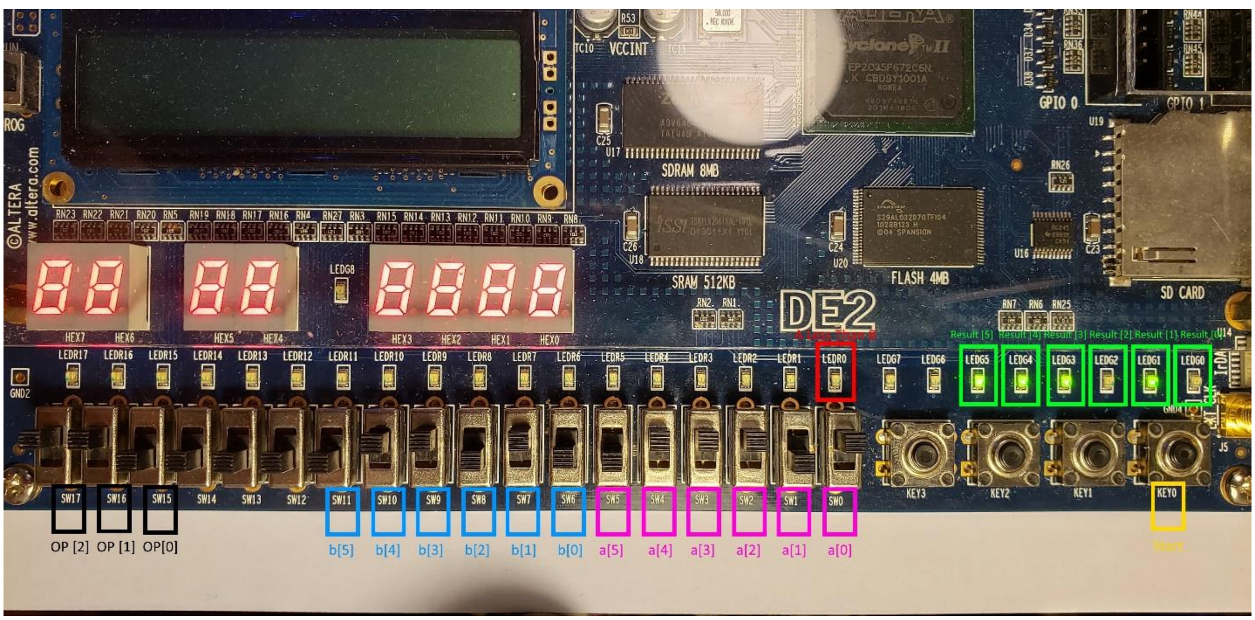

Figure 22: Rotation Left Operation

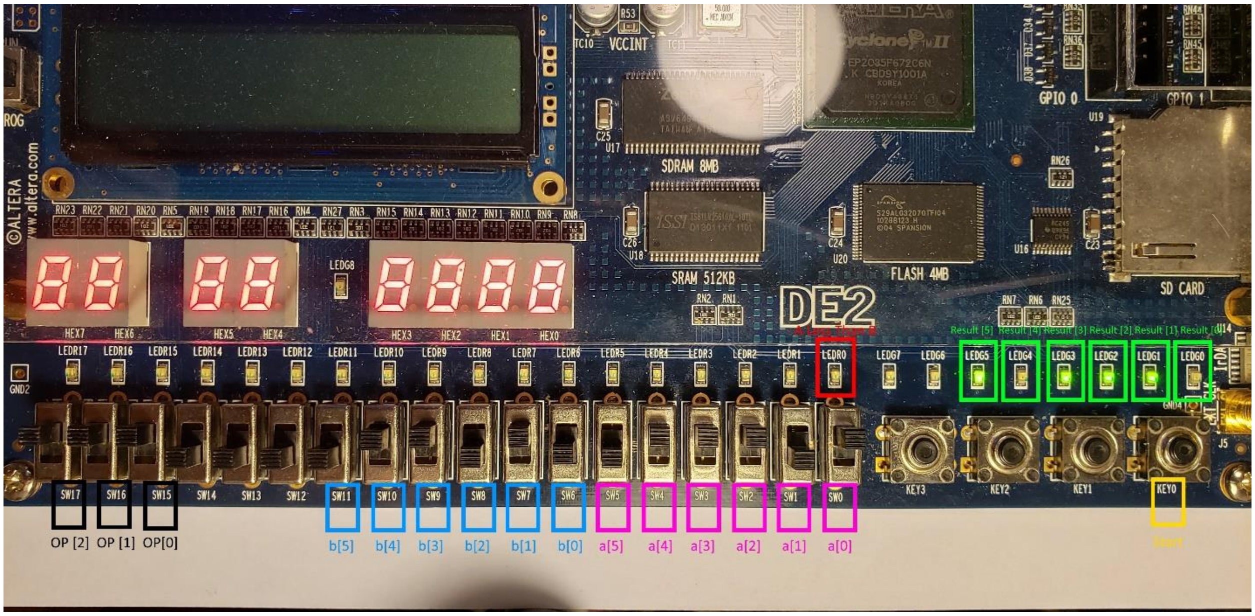

Figure 23: Rotation Right Operation

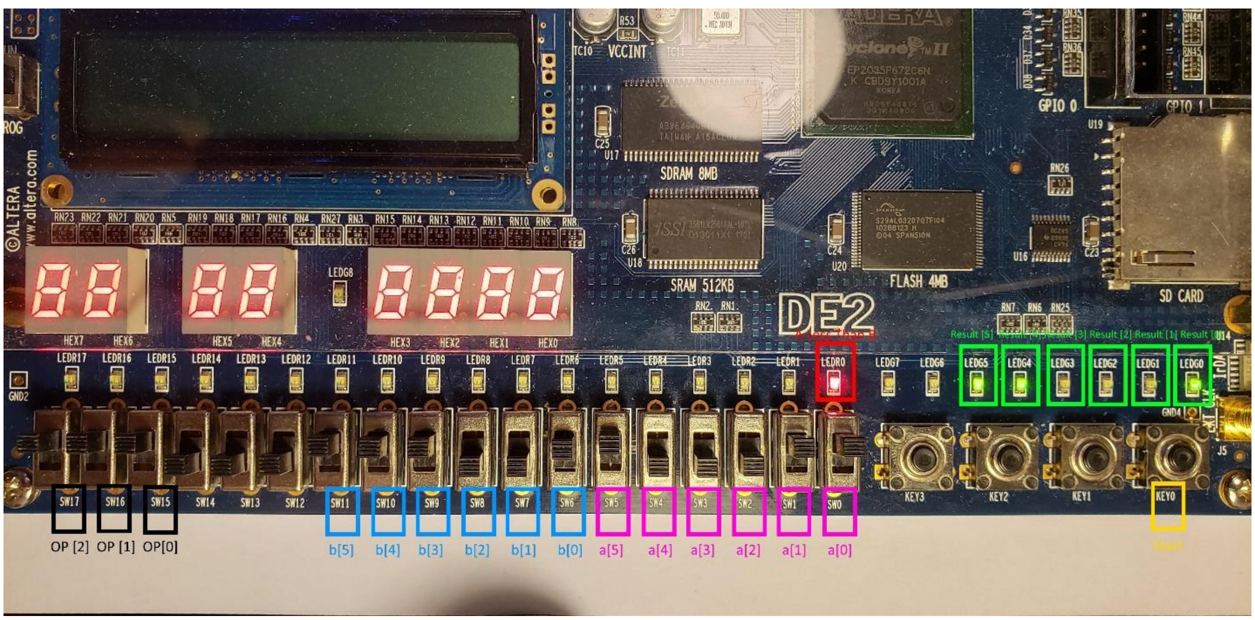

Figure 24: Less Than Check

Conclusion:

The lab taught me how to create VHDL code for bitwise functions, shifting, and rotating. It also taught me how to use them as components in another VHDL file that would operate them; in addition to learning how to make testbenches for everything and making sure the outcome is what we wanted. I was able to control the inputs and outputs by using the VHDL file that had all the functions as components; by doing so, I was able to make sure the operations VHDL file works the way I wanted to.

1. For the operations you have implemented, list corresponding operations in High level language, such as C++, Java, etc.

Implementation	VHDL	C++	Java
AND	A AND B	A & B	A & B
OR	A OR B	A | B	A | B
XOR	A XOR B	A ^ B	A ^ B
NOT	NOT A	~A	~A
Shift Left	A sll	A <<	A <<
Shift Right	A srl	A >>	A >>
Rotation Left	A rol	N/A	N/A
Rotation Right	A ror	N/A	N/A

Table 14: Functions Represented in Different Languages

2. Explain how and which of the operations you have designed in this lab can be used as multiplications or division by 2. Please give examples.

   The shift left operation that I have designed in this lab can be used as multiplication by 2 and the shift right operation can be used as division by 2.

   Example:

Binary	Decimal	Operation	Binary	Decimal
000101	5	Shift Left	001010	10
011111	31	Shift Left	111110	62
010110	22	Shift Right	001011	11
101010	42	Shift Right	010101	21

Table 15: Example for Multiplications or Divisions by 2

3. How about multiplication or division by multiples of 2, can this be implemented? How would your code need to be changed? Again, please give examples.

   By changing the amount we shift left or right, we're able to implement multiplication or division by multiples of 2. The code would change by letting the user enter in the amount it'll be shifted by.

   Example:

Binary	Decimal	Operation	Binary	Decimal
000101	5	Shift Left by 2	010100	20
000111	7	Shift Left by 3	111000	56
010110	22	Shift Right by 2	000101	5
101010	42	Shift Right by 3	001010	10

Table 16: Example for Multiplications or Divisions by Multiples of 2

4. Is there any difference in performance between doing an explicit multiplication/division by multiples of 2 in your code versus operating at the bit level of an input?

   There is a performance difference between doing an explicit multiplication/division by multiples of 2 in your code versus operating at the bit level of an input, the difference is the runtime and how fast it'll take to perform the operation.

Appendix:

Bitwise AND:

```
Library ieee;
use ieee.std_logic_1164.all;

entity Chen_Kevin_Bitwise_AND is
port(
            Chen_Kevin_a,Chen_Kevin_b: in std_logic_vector(5 downto 0);
            Chen_Kevin_result: out std_logic_vector(5 downto 0)
            );
end Chen_Kevin_Bitwise_AND;

architecture arch of Chen_Kevin_Bitwise_AND is

begin
            Chen_Kevin_result <= Chen_Kevin_a and Chen_Kevin_b;
end arch;
```

Figure 25: VHDL Code for Bitwise AND (Chen_Kevin_Bitwise_AND.VHD)

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

entity Chen_Kevin_Bitwise_AND_tb is
end Chen_Kevin_Bitwise_AND_tb;

architecture Chen_Kevin_tb of Chen_Kevin_Bitwise_AND_tb is
    signal Chen_Kevin_a, Chen_Kevin_b : std_logic_vector(5 downto 0);  -- inputs
    signal Chen_Kevin_result : std_logic_vector(5 downto 0);  -- outputs
     begin

    UUT : entity work.Chen_Kevin_Bitwise_AND port map (Chen_Kevin_a => Chen_Kevin_a,
                                                       Chen_Kevin_b => Chen_Kevin_b,
                                                       Chen_Kevin_result => Chen_Kevin_result);

        process
        begin

        Chen_Kevin_a <= "000000";
        Chen_Kevin_b <= "000000";

        for i in 0 to 63
        loop
        wait for 10 ns;
        Chen_Kevin_a <= std_logic_vector(unsigned(Chen_Kevin_a) + 1);
      wait for 10 ns;
        Chen_Kevin_a <= std_logic_vector(unsigned(Chen_Kevin_a) + 1);
      Chen_Kevin_b <= std_logic_vector(unsigned(Chen_Kevin_b) + 1);
        end loop;

        end process;

end Chen_Kevin_tb;
```

Figure 26: VHDL Testbench Code for Bitwise AND (Chen_Kevin_Bitwise_AND_tb.VHD)

Bitwise OR:

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

entity Chen_Kevin_Bitwise_OR is
port(
            Chen_Kevin_a,Chen_Kevin_b: in std_logic_vector(5 downto 0);
            Chen_Kevin_result: out std_logic_vector(5 downto 0)
            );
end Chen_Kevin_Bitwise_OR;

architecture arch of Chen_Kevin_Bitwise_OR is

begin
            Chen_Kevin_result <= Chen_Kevin_a OR Chen_Kevin_b;
end arch;

```

Figure 27: VHDL Code for Bitwise OR (Chen_Kevin_Bitwise_OR.VHD)

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

entity Chen_Kevin_Bitwise_OR_tb is
end Chen_Kevin_Bitwise_OR_tb;

architecture Chen_Kevin_tb of Chen_Kevin_Bitwise_OR_tb is
    signal Chen_Kevin_a, Chen_Kevin_b : std_logic_vector(5 downto 0);  -- inputs
    signal Chen_Kevin_result : std_logic_vector(5 downto 0);  -- outputs
begin

    UUT : entity work.Chen_Kevin_Bitwise_OR port map (Chen_Kevin_a => Chen_Kevin_a,
                                                       Chen_Kevin_b => Chen_Kevin_b,
                                                       Chen_Kevin_result => Chen_Kevin_result);
        process
        begin

        Chen_Kevin_a <= "000000";
        Chen_Kevin_b <= "000000";

        for i in 0 to 63
        loop
        wait for 10 ns;
        Chen_Kevin_a <= std_logic_vector(unsigned(Chen_Kevin_a) + 1);
      wait for 10 ns;
        Chen_Kevin_a <= std_logic_vector(unsigned(Chen_Kevin_a) + 1);
      Chen_Kevin_b <= std_logic_vector(unsigned(Chen_Kevin_b) + 1);
        end loop;

        end process;
end Chen_Kevin_tb;
```

Figure 28: VHDL Testbench Code for Bitwise OR (Chen_Kevin_Bitwise_OR_tb.VHD)

Bitwise XOR:

```
Library ieee;
use ieee.std_logic_1164.all;

entity Chen_Kevin_Bitwise_XOR is
port(
			Chen_Kevin_a,Chen_Kevin_b: in std_logic_vector(5 downto 0);
			Chen_Kevin_result: out std_logic_vector(5 downto 0)
			);
end Chen_Kevin_Bitwise_XOR;

architecture arch of Chen_Kevin_Bitwise_XOR is

begin
			Chen_Kevin_result <= Chen_Kevin_a XOR Chen_Kevin_b;
end arch;
```

Figure 29: VHDL Code for Bitwise XOR (Chen_Kevin_Bitwise_XOR.VHD)

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

entity Chen_Kevin_Bitwise_XOR_tb is
end Chen_Kevin_Bitwise_XOR_tb;

architecture Chen_Kevin_tb of Chen_Kevin_Bitwise_XOR_tb is
    signal Chen_Kevin_a, Chen_Kevin_b : std_logic_vector(5 downto 0);  -- inputs
    signal Chen_Kevin_result : std_logic_vector(5 downto 0);  -- outputs
begin

    UUT : entity work.Chen_Kevin_Bitwise_XOR port map (Chen_Kevin_a => Chen_Kevin_a,
                                                        Chen_Kevin_b => Chen_Kevin_b,
                                                        Chen_Kevin_result => Chen_Kevin_result);
        process
        begin

        Chen_Kevin_a <= "000000";
        Chen_Kevin_b <= "000000";

        for i in 0 to 63
        loop
        wait for 10 ns;
        Chen_Kevin_a <= std_logic_vector(unsigned(Chen_Kevin_a) + 1);
      wait for 10 ns;
        Chen_Kevin_a <= std_logic_vector(unsigned(Chen_Kevin_a) + 1);
      Chen_Kevin_b <= std_logic_vector(unsigned(Chen_Kevin_b) + 1);
        end loop;

        end process;
end Chen_Kevin_tb;
```

Figure 30: VHDL Testbench Code for Bitwise XOR (Chen_Kevin_Bitwise_XOR_tb.VHD)

Bitwise NOT:

```
Library ieee;
use ieee.std_logic_1164.all;

entity Chen_Kevin_Bitwise_NOT is
port(
			Chen_Kevin_a: in std_logic_vector(5 downto 0);
			Chen_Kevin_result: out std_logic_vector(5 downto 0)
			);
end Chen_Kevin_Bitwise_NOT;

architecture arch of Chen_Kevin_Bitwise_NOT is

begin
			Chen_Kevin_result <= NOT Chen_Kevin_a;
end arch;
```

Figure 31: VHDL Code for Bitwise NOT (Chen_Kevin_Bitwise_NOT.VHD)

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

entity Chen_Kevin_Bitwise_NOT_tb is
end Chen_Kevin_Bitwise_NOT_tb;

architecture Chen_Kevin_tb of Chen_Kevin_Bitwise_NOT_tb is
    signal Chen_Kevin_a : std_logic_vector(5 downto 0);  -- inputs
    signal Chen_Kevin_result : std_logic_vector(5 downto 0);  -- outputs
begin

    UUT : entity work.Chen_Kevin_Bitwise_NOT port map (Chen_Kevin_a => Chen_Kevin_a,
                                                       Chen_Kevin_result => Chen_Kevin_result);
        process
        begin

        Chen_Kevin_a <= "000000";

        for i in 0 to 63
        loop
        wait for 10 ns;
        Chen_Kevin_a <= std_logic_vector(unsigned(Chen_Kevin_a) + 1);
        end loop;

        end process;
end Chen_Kevin_tb;
```

Figure 32: VHDL Testbench Code for Bitwise NOT (Chen_Kevin_Bitwise_NOT_tb.VHD)

Shift Left:

```
Library ieee;
use ieee.std_logic_1164.all;

entity Chen_Kevin_Shift_Left is
port(
            Chen_Kevin_a: in std_logic_vector(5 downto 0);
            Chen_Kevin_result: out std_logic_vector(5 downto 0)
            );
end Chen_Kevin_Shift_Left;

architecture arch of Chen_Kevin_Shift_Left is

begin
            Chen_Kevin_result <= to_stdlogicvector(to_bitvector(Chen_Kevin_a) sll 1);
end arch;
```

Figure 33: VHDL Code for Shift Left (Chen_Kevin_Shift_Left.VHD)

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

entity Chen_Kevin_Shift_Left_tb is
end Chen_Kevin_Shift_Left_tb;

architecture Chen_Kevin_tb of Chen_Kevin_Shift_Left_tb is
    signal Chen_Kevin_a: std_logic_vector(5 downto 0);  -- inputs
    signal Chen_Kevin_result : std_logic_vector(5 downto 0);  -- outputs
begin

    UUT : entity work.Chen_Kevin_Shift_Left port map (Chen_Kevin_a => Chen_Kevin_a,
                                                       Chen_Kevin_result => Chen_Kevin_result);
        process
        begin

        Chen_Kevin_a <= "000000";

        for i in 0 to 63
        loop
        wait for 10 ns;
        Chen_Kevin_a <= std_logic_vector(unsigned(Chen_Kevin_a) + 1);
        end loop;

        end process;
end Chen_Kevin_tb;
```

Figure 34: VHDL Testbench Code for Shift Left (Chen_Kevin_Shift_Left_tb.VHD)

<u>Shift Right:</u>

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

entity Chen_Kevin_Shift_Right is
port(
		Chen_Kevin_a: in std_logic_vector(5 downto 0);
		Chen_Kevin_result: out std_logic_vector(5 downto 0)
		);
end Chen_Kevin_Shift_Right;

architecture arch of Chen_Kevin_Shift_Right is

begin
		Chen_Kevin_result <= to_stdlogicvector(to_bitvector(Chen_Kevin_a) srl 1);
end arch;
```

Figure 35: VHDL Code for Shift Right (Chen_Kevin_Shift_Right.VHD)

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

entity Chen_Kevin_Operations_tb is
end Chen_Kevin_Operations_tb;

architecture Chen_Kevin_tb of Chen_Kevin_Operations_tb is
    signal Chen_Kevin_a, Chen_Kevin_b : std_logic_vector(5 downto 0);  -- inputs
     signal Chen_Kevin_Start: std_logic;
     signal Chen_Kevin_op: std_logic_vector(2 downto 0);
    signal Chen_Kevin_result : std_logic_vector(5 downto 0);  -- outputs
    signal Chen_Kevin_LessThan : std_logic;
     begin

    UUT : entity work.Chen_Kevin_Operations port map (Chen_Kevin_a => Chen_Kevin_a,
                                                       Chen_Kevin_b => Chen_Kevin_b,
                                                       Chen_Kevin_Start => Chen_Kevin_Start,
                                                       Chen_Kevin_op => Chen_Kevin_op,
                                                       Chen_Kevin_result => Chen_Kevin_result,
                                                       Chen_Kevin_LessThan => Chen_Kevin_LessThan);

        Chen_Kevin_a <= "000000", "101001" after 10 ns, "101001" after 20 ns, "001011" after 30 ns,
        "001011" after 40 ns;
        Chen_Kevin_b <= "000000", "001101" after 10 ns, "001101" after 20 ns, "001101" after 30 ns,
        "001101" after 40 ns;
        Chen_Kevin_Start <= '0', '0' after 10 ns, '1' after 20 ns, '0' after 30 ns, '1' after 40 ns;
        Chen_Kevin_op <= "000","010" after 10 ns, "010" after 20 ns, "111" after 30 ns,
        "111" after 40 ns;


end Chen_Kevin_tb;
```

Figure 36: VHDL Testbench Code for Shift Right (Chen_Kevin_Shift_Right_tb.VHD)

<u>Rotation Left:</u>

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

entity Chen_Kevin_Rotation_Left is
port(
		Chen_Kevin_a: in std_logic_vector(5 downto 0);
		Chen_Kevin_result: out std_logic_vector(5 downto 0)
		);
end Chen_Kevin_Rotation_Left;

architecture arch of Chen_Kevin_Rotation_Left is

begin
		Chen_Kevin_result <= to_stdlogicvector(to_bitvector(Chen_Kevin_a) rol 1);
end arch;
```

Figure 37: VHDL Code for Rotation Left (Chen_Kevin_Rotation_Left.VHD)

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

entity Chen_Kevin_Rotation_Left_tb is
end Chen_Kevin_Rotation_Left_tb;

architecture Chen_Kevin_tb of Chen_Kevin_Rotation_Left_tb is
    signal Chen_Kevin_a: std_logic_vector(5 downto 0);  -- inputs
    signal Chen_Kevin_result : std_logic_vector(5 downto 0);  -- outputs
begin

    UUT : entity work.Chen_Kevin_Rotation_Left port map (Chen_Kevin_a => Chen_Kevin_a,
                                                       Chen_Kevin_result => Chen_Kevin_result);
        process
        begin

        Chen_Kevin_a <= "000000";

        for i in 0 to 63
        loop
        wait for 10 ns;
        Chen_Kevin_a <= std_logic_vector(unsigned(Chen_Kevin_a) + 1);
        end loop;

        end process;
end Chen_Kevin_tb;
```

Figure 38: VHDL Testbench Code for Rotation Left (Chen_Kevin_Rotation_Left_tb.VHD)

Rotation Right:

```
Library ieee;
use ieee.std_logic_1164.all;

entity Chen_Kevin_Rotation_Right is
port(
            Chen_Kevin_a: in std_logic_vector(5 downto 0);
            Chen_Kevin_result: out std_logic_vector(5 downto 0)
            );
end Chen_Kevin_Rotation_Right;

architecture arch of Chen_Kevin_Rotation_Right is

begin
            Chen_Kevin_result <= to_stdlogicvector(to_bitvector(Chen_Kevin_a) ror 1);
end arch;
```

Figure 39: VHDL Code for Rotation Right (Chen_Kevin_Rotation_Right.VHD)

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

entity Chen_Kevin_Operations_tb is
end Chen_Kevin_Operations_tb;

architecture Chen_Kevin_tb of Chen_Kevin_Operations_tb is
    signal Chen_Kevin_a, Chen_Kevin_b : std_logic_vector(5 downto 0);  -- inputs
     signal Chen_Kevin_Start: std_logic;
     signal Chen_Kevin_op: std_logic_vector(2 downto 0);
    signal Chen_Kevin_result : std_logic_vector(5 downto 0);  -- outputs
    signal Chen_Kevin_LessThan : std_logic;
     begin

    UUT : entity work.Chen_Kevin_Operations port map (Chen_Kevin_a => Chen_Kevin_a,
                                                          Chen_Kevin_b => Chen_Kevin_b,
                                                          Chen_Kevin_Start => Chen_Kevin_Start,
                                                          Chen_Kevin_op => Chen_Kevin_op,
                                                          Chen_Kevin_result => Chen_Kevin_result
                                                          Chen_Kevin_LessThan => Chen_Kevin_LessThan);

        Chen_Kevin_a <= "000000", "101001" after 10 ns, "101001" after 20 ns, "001011" after 30 ns,
        "001011" after 40 ns;
        Chen_Kevin_b <= "000000", "001101" after 10 ns, "001101" after 20 ns, "001101" after 30 ns,
        "001101" after 40 ns;
        Chen_Kevin_Start <= '0', '0' after 10 ns, '1' after 20 ns, '0' after 30 ns, '1' after 40 ns;
        Chen_Kevin_op <= "000","010" after 10 ns, "010" after 20 ns, "111" after 30 ns,
        "111" after 40 ns;


end Chen_Kevin_tb;
```

Figure 40: VHDL Testbench Code for Rotation Right (Chen_Kevin_Rotation_Right_tb.VHD)

<u>Operations:</u>

```vhdl
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Chen_Kevin_Operations is
port(
            Chen_Kevin_a, Chen_Kevin_b: in std_logic_vector (5 downto 0);
            Chen_Kevin_result: out std_logic_vector (5 downto 0);
            Chen_Kevin_LessThan: out std_logic;
            Chen_Kevin_Start: in std_logic;
            Chen_Kevin_op: in std_logic_vector(2 downto 0)
            );
end Chen_Kevin_Operations;

architecture arch of Chen_Kevin_Operations is
    signal Chen_Kevin_in1, Chen_Kevin_in2: std_logic_vector(5 downto 0);
    signal Chen_Kevin_AND, Chen_Kevin_OR, Chen_Kevin_XOR, Chen_Kevin_NOT,
    Chen_Kevin_ShiftLeft, Chen_Kevin_ShiftRight, Chen_Kevin_RotateLeft,
    Chen_Kevin_RotateRight: std_logic_vector(5 downto 0);

component Chen_Kevin_Bitwise_AND
  port (Chen_Kevin_a, Chen_Kevin_b : in std_logic_vector(5 downto 0);
        Chen_Kevin_result : out std_logic_vector(5 downto 0)
    );
end component;

component Chen_Kevin_Bitwise_OR
  port (Chen_Kevin_a, Chen_Kevin_b : in std_logic_vector(5 downto 0);
        Chen_Kevin_result : out std_logic_vector(5 downto 0)
    );
end component;

component Chen_Kevin_Bitwise_XOR
  port (Chen_Kevin_a, Chen_Kevin_b : in std_logic_vector(5 downto 0);
        Chen_Kevin_result : out std_logic_vector(5 downto 0)
    );
end component;

component Chen_Kevin_Bitwise_NOT
  port (Chen_Kevin_a: in std_logic_vector(5 downto 0);
        Chen_Kevin_result : out std_logic_vector(5 downto 0)
    );
end component;

component Chen_Kevin_Shift_Left
  port (Chen_Kevin_a: in std_logic_vector(5 downto 0);
        Chen_Kevin_result : out std_logic_vector(5 downto 0)
    );
end component;

component Chen_Kevin_Shift_Right
  port (Chen_Kevin_a: in std_logic_vector(5 downto 0);
        Chen_Kevin_result : out std_logic_vector(5 downto 0)
    );
end component;

component Chen_Kevin_Rotation_Left
  port (Chen_Kevin_a: in std_logic_vector(5 downto 0);
        Chen_Kevin_result : out std_logic_vector(5 downto 0)
    );
end component;

component Chen_Kevin_Rotation_Right
  port (Chen_Kevin_a: in std_logic_vector(5 downto 0);
        Chen_Kevin_result : out std_logic_vector(5 downto 0)
    );
end component;
```

```vhdl


begin
Chen_Kevin_in1 <= Chen_Kevin_a;
Chen_Kevin_in2 <= Chen_Kevin_b;

Chen_Kevin_ANDFunction: Chen_Kevin_Bitwise_AND
    port map( Chen_Kevin_a => Chen_Kevin_in1,
              Chen_Kevin_b => Chen_Kevin_in2,
              Chen_Kevin_result => Chen_Kevin_AND);

Chen_Kevin_ORFunction: Chen_Kevin_Bitwise_OR
    port map( Chen_Kevin_a => Chen_Kevin_in1,
              Chen_Kevin_b => Chen_Kevin_in2,
              Chen_Kevin_result => Chen_Kevin_OR);

Chen_Kevin_XORFunction: Chen_Kevin_Bitwise_XOR
    port map( Chen_Kevin_a => Chen_Kevin_in1,
              Chen_Kevin_b => Chen_Kevin_in2,
              Chen_Kevin_result => Chen_Kevin_XOR);

Chen_Kevin_NOTFunction: Chen_Kevin_Bitwise_NOT
    port map( Chen_Kevin_a => Chen_Kevin_in1,
              Chen_Kevin_result => Chen_Kevin_NOT);

Chen_Kevin_ShiftLeftFunction: Chen_Kevin_Shift_Left
    port map( Chen_Kevin_a => Chen_Kevin_in1,
              Chen_Kevin_result => Chen_Kevin_ShiftLeft);

Chen_Kevin_ShiftRightFunction: Chen_Kevin_Shift_Right
    port map( Chen_Kevin_a => Chen_Kevin_in1,
              Chen_Kevin_result => Chen_Kevin_ShiftRight);

Chen_Kevin_RotateLeftFunction: Chen_Kevin_Rotation_Left
    port map( Chen_Kevin_a => Chen_Kevin_in1,
              Chen_Kevin_result => Chen_Kevin_RotateLeft);

Chen_Kevin_RotateRightFunction: Chen_Kevin_Rotation_Right
    port map( Chen_Kevin_a => Chen_Kevin_in1,
              Chen_Kevin_result => Chen_Kevin_RotateRight);

process (Chen_Kevin_Start,Chen_Kevin_a,Chen_Kevin_b,Chen_Kevin_op)
begin
    if (Chen_Kevin_Start='1') then
        case Chen_Kevin_op is
            when "000" =>
            Chen_Kevin_result <= Chen_Kevin_AND;
            when "001" =>
            Chen_Kevin_result <= Chen_Kevin_OR;
            when "010" =>
            Chen_Kevin_result <= Chen_Kevin_XOR;
            when "011" =>
            Chen_Kevin_result <= Chen_Kevin_NOT;
            when "100" =>
            Chen_Kevin_result <= Chen_Kevin_ShiftLeft;
            when "101" =>
            Chen_Kevin_result <= Chen_Kevin_ShiftRight;
            when "110" =>
            Chen_Kevin_result <= Chen_Kevin_RotateLeft;
            when "111" =>
            Chen_Kevin_result <= Chen_Kevin_RotateRight;
            when others =>
                NULL;
        end case;
        if (signed(Chen_Kevin_b) > (signed(Chen_Kevin_a))) then
        Chen_Kevin_LessThan <= '1';
        else
        Chen_Kevin_LessThan <= '0';
        end if;
    end if;
end process;

end arch;
```

Figure 41: VHDL Code for Operations (Chen_Kevin_Operations.VHD)

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

entity Chen_Kevin_Operations_tb is
end Chen_Kevin_Operations_tb;

architecture Chen_Kevin_tb of Chen_Kevin_Operations_tb is
    signal Chen_Kevin_a, Chen_Kevin_b : std_logic_vector(5 downto 0);  -- inputs
     signal Chen_Kevin_Start: std_logic;
     signal Chen_Kevin_op: std_logic_vector(2 downto 0);
    signal Chen_Kevin_result : std_logic_vector(5 downto 0);  -- outputs
    signal Chen_Kevin_LessThan : std_logic;
     begin

    UUT : entity work.Chen_Kevin_Operations port map (Chen_Kevin_a => Chen_Kevin_a,
                                                        Chen_Kevin_b => Chen_Kevin_b,
                                                        Chen_Kevin_Start => Chen_Kevin_Start,
                                                        Chen_Kevin_op => Chen_Kevin_op,
                                                        Chen_Kevin_result => Chen_Kevin_result,
                                                        Chen_Kevin_LessThan => Chen_Kevin_LessThan);

        Chen_Kevin_a <= "000000", "101001" after 10 ns, "101001" after 20 ns, "001011" after 30 ns,
        "001011" after 40 ns;
        Chen_Kevin_b <= "000000", "001101" after 10 ns, "001101" after 20 ns, "001101" after 30 ns,
        "001101" after 40 ns;
        Chen_Kevin_Start <= '0', '0' after 10 ns, '1' after 20 ns, '0' after 30 ns, '1' after 40 ns;
        Chen_Kevin_op <= "000","010" after 10 ns, "010" after 20 ns, "111" after 30 ns,
        "111" after 40 ns;


end Chen_Kevin_tb;
```

Figure 42: VHDL Testbench Code for Operations (Chen_Kevin_Operations_tb.VHD)

Variable Name	Variable Type	Signal Name	FPGA Pin No.
Chen_Kevin_Start	Input	KEY[0]	PIN_G26
Chen_Kevin_a[0]	Input	SW[0]	PIN_N25
Chen_Kevin_a[1]	Input	SW[1]	PIN_N26
Chen_Kevin_a[2]	Input	SW[2]	PIN_P25
Chen_Kevin_a[3]	Input	SW[3]	PIN_AE14
Chen_Kevin_a[4]	Input	SW[4]	PIN_AF14
Chen_Kevin_a[5]	Input	SW[5]	PIN_AD13
Chen_Kevin_b[0]	Input	SW[6]	PIN_AC13
Chen_Kevin_b[1]	Input	SW[7]	PIN_C13
Chen_Kevin_b[2]	Input	SW[8]	PIN_B13
Chen_Kevin_b[3]	Input	SW[9]	PIN_A13
Chen_Kevin_b[4]	Input	SW[10]	PIN_N1
Chen_Kevin_b[5]	Input	SW[11]	PIN_P1
Chen_Kevin_op[0]	Input	SW[15]	PIN_U4
Chen_Kevin_op[1]	Input	SW[16]	PIN_V1
Chen_Kevin_op[2]	Input	SW[17]	PIN_V2
Chen_Kevin_result[0]	Output	LEDG[0]	PIN_AE22
Chen_Kevin_result[1]	Output	LEDG[1]	PIN_AF22
Chen_Kevin_result[2]	Output	LEDG[2]	PIN_W19
Chen_Kevin_result[3]	Output	LEDG[3]	PIN_V18
Chen_Kevin_result[4]	Output	LEDG[4]	PIN_U18
Chen_Kevin_result[5]	Output	LEDG[5]	PIN_U17
Chen_Kevin_ALessThanB	Output	LEDR[0]	PIn_AE23

Table 17: Pin Assignments for Operations

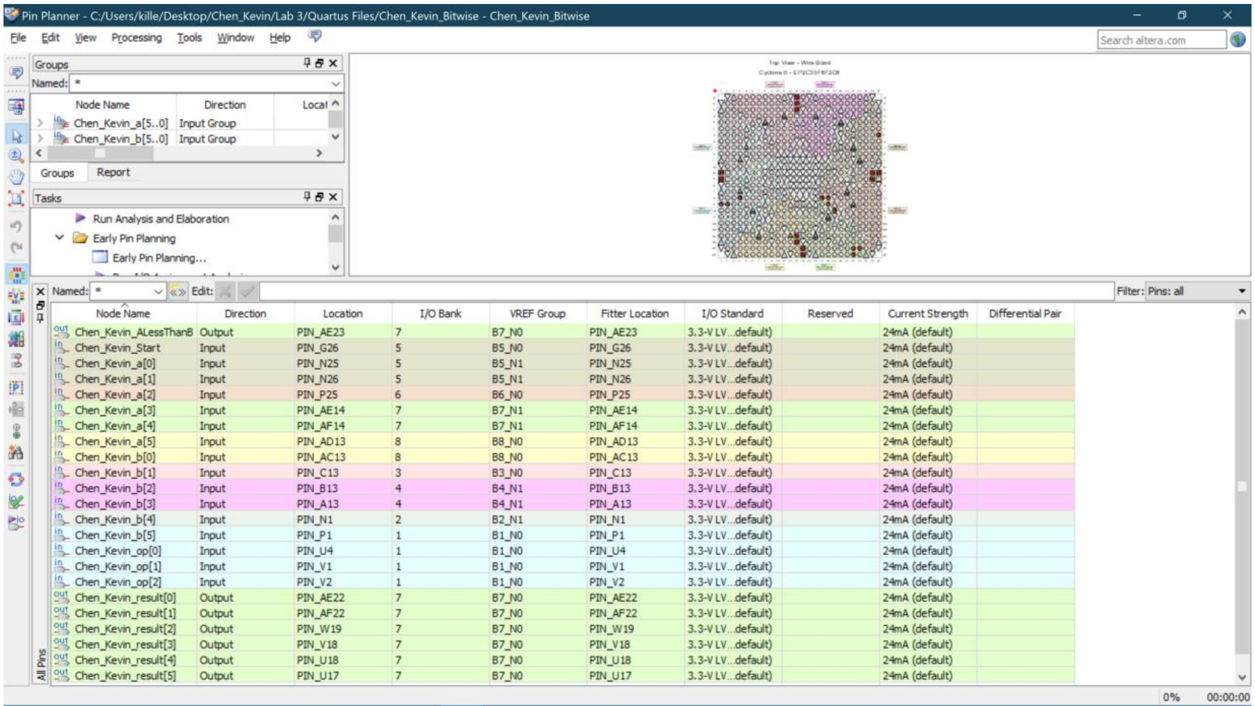

Figure 43: Pin Planner of Block Diagram/Schematic File of Operations