CS 343, Spring 2019

Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Instructor: Professor Izidor Gertner

April 1, 2019 by 11:59 AM

Objective:

In this lab you will extend your knowledge of VHDL, ModelSim and Quartus by developing a circuit that performs bitwise operations, bit shifts and bit rotations.
These operations are chosen by an OPCODE - a selector that determines which operation to perform at a given time.

Please implement all 32 bit bitwise operation shown on "MIPS GREEN PAGES", demonstrate on DE 2 board. E.g 32 bitwise "AND" operation.

In addition, Implement conditional operation: ***Set Less Than -- SLT (signed comparison)***

Set result to 1 if a condition is true Otherwise, set to 0

```
slt register_d, register_s, register_t
if (rs < rt) rd = 1; else rd = 0;

slti rt, rs, constant
if (rs < constant) rt = 1; else rt = 0;
```

Write a test-bench program to test all bitwise operation.

Write a complete report including VHDL code, screenshots, and pics of demo on the DE2 board.

Create a 2 min video of your bitwise operation demo.

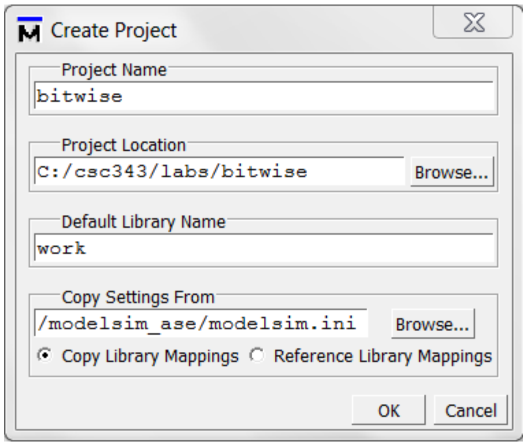
CS 343, Spring 2019

Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Instructor: Professor Izidor Gertner
April 1, 2019 by 11:59 AM

1. TUTORIAL: BITWISE AND

1. Open ModelSim and create a new project. Call it **bitwise** and save it in a location of your preference. For the Default Library Name call it **work**



Create Project

Project Name: bitwise

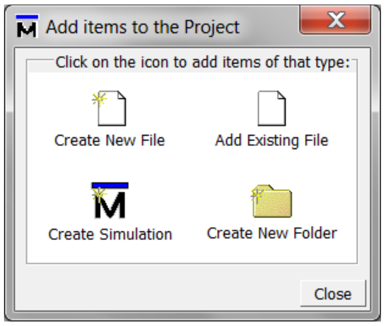
Project Location: C:/csc343/labs/bitwise

Default Library Name: work

Copy Settings From: /modelsim_ase/modelsim.ini

[✓ Copy Library Mappings] Reference Library Mappings

2. When the **Add Items to the Project** dialog appears, select **Create New File**



Add items to the Project

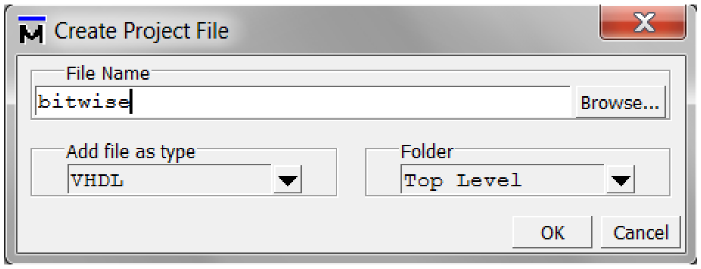
Click on the icon to add items of that type:

3. Call your file bitwise, then click OK

CS 343, Spring 2019

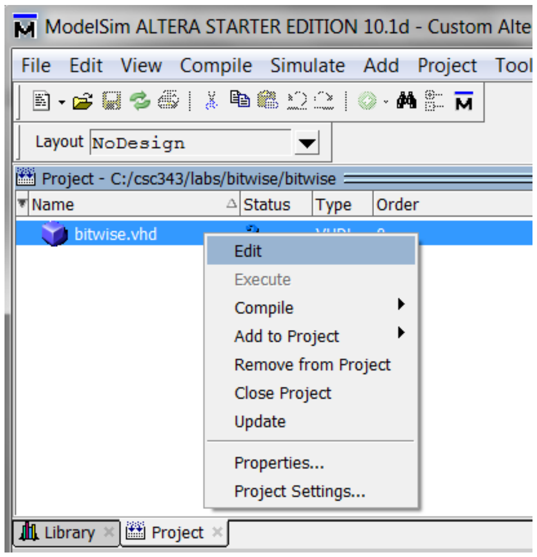
Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Instructor: Professor Izidor Gertner
April 1, 2019 by 11:59 AM



File Name: bitwise

4. In the Project panel, right click the bitwise file you just created and select **Edit**.



5. Copy the following code and save your file (**File > Save**):

bitwise.vhd

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

entity bitwise is
port(
        a,b: in std_logic_vector(2 downto 0);
        result: out std_logic_vector(2 downto 0)
  );
end bitwise;

architecture arch of bitwise is
begin
        result <= a and b;
end arch;
```

Read the code and try to understand its logic.
In the entity part we are declaring our inputs and outputs: two vector inputs (a and b) and one vector output (result)
In the architecture part we specify the behavior these inputs and outputs will have: result takes the value of applying the BITWISE AND operation between a and b.
Notice that in order for result to receive the corresponding bits of the operation between a and b, it has to be of the same width as the inputs, otherwise you will experience compilation errors.
Remember the truth table for AND:

A	B	out
0	0	0
0	1	0
1	0	0
1	1	1

Therefore, let's say we have a=100 and b=101, we should obtain result=100 if we compare the bits at each position between a and b:

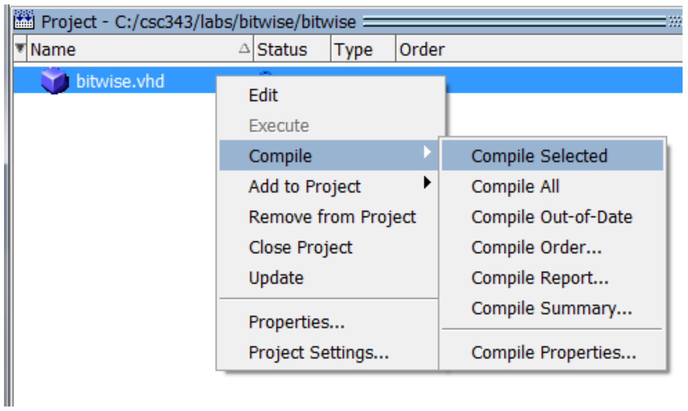
```
        a  = 100
AND     b  = 101
-----------------
result     = 100
```

6. Compile your file by right-clicking on it in the project panel. Choose **Compile > Compile Selected.** Make sure your file compiles with no errors. Check for typos, missing semicolons, etc. in your code if you get compilation errors.

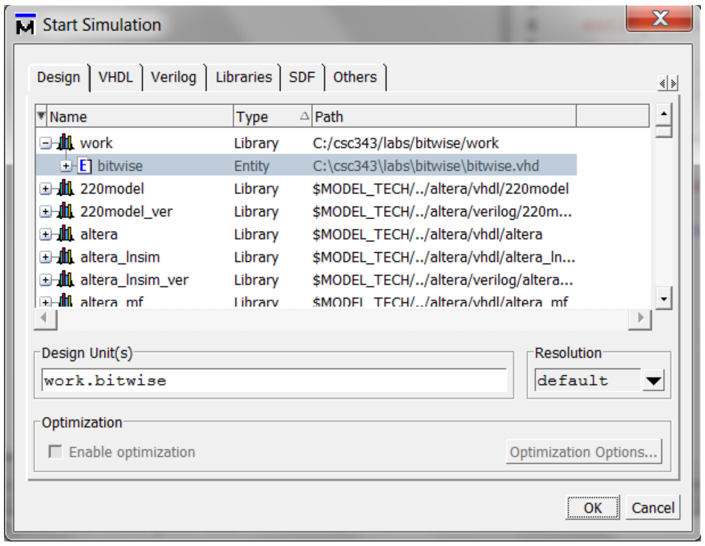
CS 343, Spring 2019

Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Instructor: Professor Izidor Gertner
April 1, 2019 by 11:59 AM



7. Let's now run a waveform simulation. Go to Simulate > Start Simulation. Under the work Library, choose the bitwise entity and click OK.



Name	Type	Path
work	Library	C:/csc343/labs/bitwise/work
bitwise	Entity	C:\csc343\labs\bitwise\bitwise.vhd
220model	Library	$MODEL_TECH/../altera/vhdl/220model
220model_ver	Library	$MODEL_TECH/../altera/verilog/220m...
altera	Library	$MODEL_TECH/../altera/vhdl/altera
altera_lnsim	Library	$MODEL_TECH/../altera/vhdl/altera_ln...
altera_lnsim_ver	Library	$MODEL_TECH/../altera/verilog/altera...
altera_mf	Library	$MODEL_TECH/../altera/vhdl/altera_mf

Design Unit(s): work.bitwise

Resolution: default

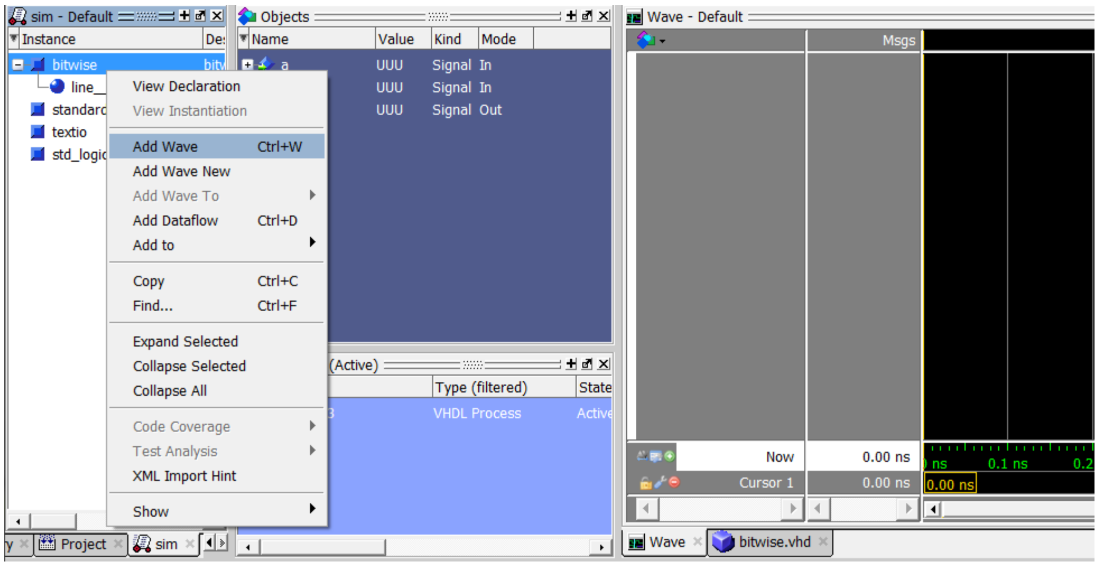
Optimization: Enable optimization

8. Right-click the bitwise entity in the leftmost panel (sim panel) as shown in the figure below and click **Add Wave** to add the inputs and outputs to the wave panel.

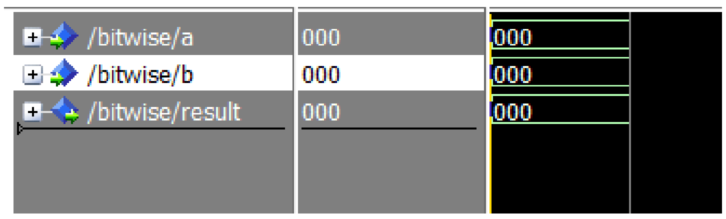
CS 343, Spring 2019

Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

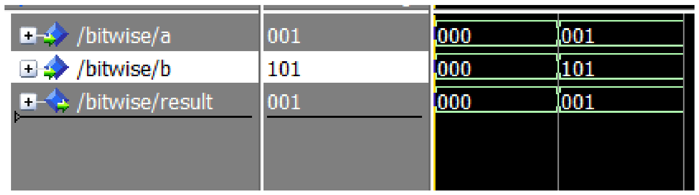
Instructor: Professor Izidor Gertner
April 1, 2019 by 11:59 AM



9. Now force some values to input a and input b. Right click /bitwise/a and force a value of 000. Do the same for /bitwise/b. Then **hit F9** to run simulation for 100 ps. Alternatively, you can go to **Simulate > Run > Run 100**. You should obtain a waveform as in the following figure:



10. Let's force some different values to test our design again. Test the following pair of values:
    (a=001, b=101), (a=010, b=111), (a=011, b=010).
    Feel free to try additional combinations.



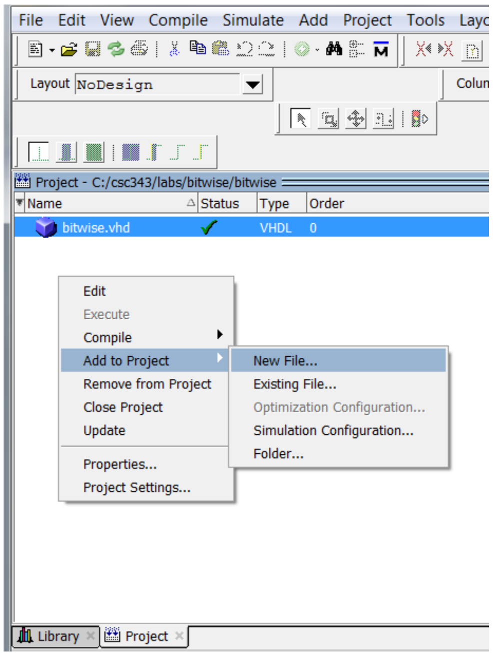
11. Once we see that our code behaves as expected, let's end the simulation. Go to **Simulate > End Simulation**.

2. TUTORIAL: LEFT SHIFT

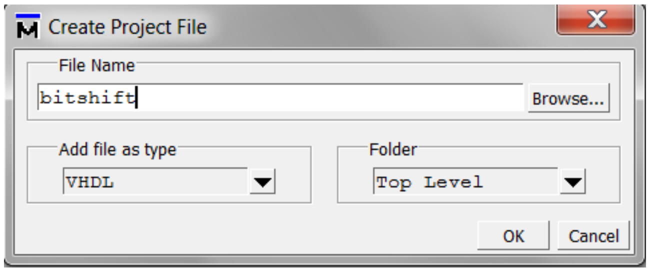
1. In the same project that you have opened in ModelSim for bitwise, let's create a new VHDL file to include in our current project. Right click on the project panel and select **Add to Project > New File…**

Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Instructor: Professor Izidor Gertner
April 1, 2019 by 11:59 AM



2. Call this new file **bitshift**.



File Name: bitshift
Add file as type: VHDL
Folder: Top Level

3. The bitshift file has now been added to your project panel. Right-click on it and select **Edit**. In the code editor that appears, copy the following code:

bitshift.vhd

```
library ieee;
```

```
 2 use ieee.std_logic_1164.all;
 3
 4 entity bitshift is
 5 port(
 6         a: in std_logic_vector(4 downto 0);
 7         result: out std_logic_vector(4 downto 0)
 8         );
 9 end bitshift;
10
11 architecture arch of bitshift is
12
13 begin
14         result <= to_stdlogicvector(to_bitvector(a) sll 1);
15 end arch;
```

This time we are not performing operations or comparisons between two inputs. Instead, we have only a 5-bit wide input and we aim to shift its bits to the left by one position. As the bits are shifted to the left by one position, the empty bit that is created on the rightmost position (LSB) is filled up with a zero.

There is a command in VHDL that allows us to achieve the shift to the left effect, this command is sll, its syntax is:

```
bit_vector_name sll number_of_positions_to_shift
```

In our code, line 14 achieves this effect:

```
result <= to_stdlogicvector(to_bitvector(a) sll 1);
```

Notice that our input **a** is declared as a logic-vector. However, the sll operation is only available to inputs declared as bit-vectors; therefore, we need to cast input a. The function to_bitvector(*vector-name*) achieves this.
The output ***result*** was also declared as a logic-vector in the entity part, so once we cast input ***a*** as bit-vector and apply the sll operation, we cast it back to logic-vector for result to accept the outcome in the format that we declared in the entity part of our code.

Alternatively, we could have initialized our input and output as bit-vectors from the very beginning and avoid casting, as shown in the code below. However, we advise you to work with logic-vectors whenever possible. Bit-vectors allow only two values ‘0’ or ‘1’. On the other hand, logic-vectors allow 9 values ('U', 'X', '0', '1', 'Z', 'W', 'L' ,'H' and '-'.) and offer a more real behavior of circuit logic. For more on the difference between logic-vectors and bit-vectors we suggest you do some research online.

If we had used the bit-vector approach, our code would have looked like this:

bitshift_bitvector.vhd (no need to implement your code this way, just for demonstration)

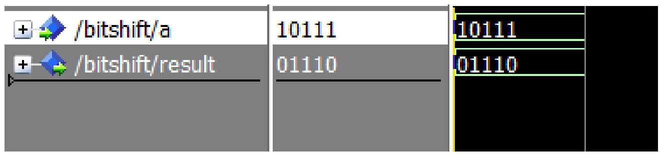
```
library ieee;
use ieee.std_logic_1164.all;

entity bitshift_bitvector is
port(
        a: in bit_vector(5 downto 0);
        result: out bit_vector(5 downto 0)
        );
end bitshift_bitvector;

architecture arch of bitshift_bitvector is

begin
        result <= a sll 1;
end arch;
```

4. Run a waveform simulation for bitshift.vhd and analyze your results. For example, we ran a simulation with input a=10111. As we shift the bits to the left by one position, we should obtain result=01110. The LSB in result is filled with a zero. See the figure below.



```
/bitshift/a        10111   10111
/bitshift/result   01110   01110
```

3. YOUR TASK

CS 343, Spring 2019

Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Instructor: Professor Izidor Gertner
April 1, 2019 by 11:59 AM

1. Circuit behavior:

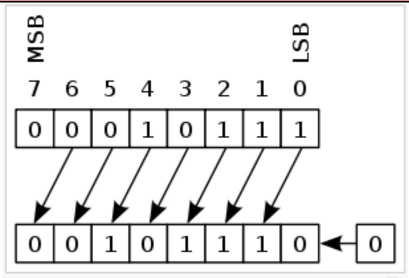
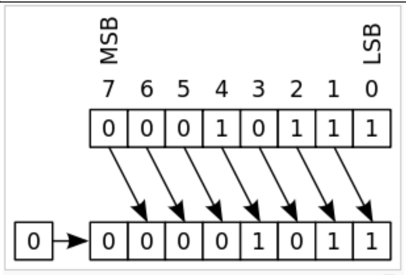
For this lab you have to create a circuit that accepts 8 operations:

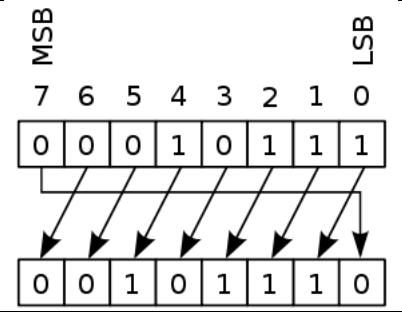
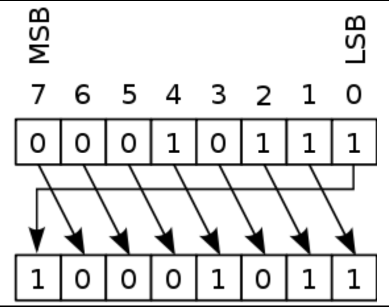
- 4 Bitwise operations: AND, OR, XOR, NOT
- 2 Shifts: left shift and right shift
- 2 Rotations: left rotation and right rotation

Only one operation is performed at a time. The selection is based on the OpCode – "operations code". Since we need to represent 8 operations, you need $\log_2 8 = 3$ switches.

For example, OpCode "010" may specify Bitwise XOR operation, OpCode "111" may specify Rotation-To-Right operation. Choose the word size for inputs/outputs in all these operations as 6 bits.

The following table shows the shift and rotation operations you have to perform for this lab, the corresponding VHDL command as well as a reference diagram:

Operation	VHDL command	Representation
Left shift (shift left logical)	sll	 MSB LSB 7 6 5 4 3 2 1 0 0 0 0 1 0 1 1 1 0 0 1 0 1 1 1 0 0 Logical left shift one bit
Right shift (shift right logical)	srl	 MSB LSB 7 6 5 4 3 2 1 0 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 Logical right shift one bit

Rotation left	rol	 MSB LSB 7 6 5 4 3 2 1 0 0 0 0 1 0 1 1 1 0 0 1 0 1 1 1 0
Rotation right	ror	 MSB LSB 7 6 5 4 3 2 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 0 1 1

To demonstrate your design:

- You have to select positions of 6 switches corresponding to the binary value of the FIRST operand input.
- You have to select positions of 6 switches corresponding to the binary value of the SECOND operand input.
- You select an operation to perform with the 3-bit OpCode.
- You have to press Start KEY to tell the circuit to do the specified operation (you have to incorporate this Start behavior in your design).
- The LEDs should display correct result of corresponding BITWISE operation.
- Realize that for AND, XOR, OR you need to perform the operations using two operands. However for NOT, as well as the shifts and rotations, you only perform it using one operand. In this case choose the FIRST operand.
- In your report you have to provide pictures and/or video that proves you have done this.

2. <u>Circuit design</u>

CS 343,   Spring 2019

Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Instructor: Professor   Izidor Gertner
April 1, 2019 by 11:59 AM

We are providing you some skeleton code for the architecture part of your file that you might find helpful. Note that we are making use of an if statement to make the operations perform only when a certain requirement is met. Then there is a case statement to address the different values that ***op*** can have.

NOTE: You might find it more helpful to start coding in Quartus because as you write and compile your code, Quartus gives you more meaningful compilation errors than ModelSim.

```
architecture arch of operations is
--declare signals in1, in2, out1

begin
in1 <= --complete this;
in2 <= --complete this;
result <= --complete this;

process( --/*processing_signal*/ )
begin
       if( --/*processing_signal*/ ='1') then
              case op is
                     when "000" =>
                            --output <= input1 operation input2;
                     when "001" =>
                            --output <= input1 operation input2;

                     --complete all cases!

                     -- you might figure that as you complete
                     -- all your cases, this "when others => NULL;" case
                     -- is not necessary; however, we mention it here
                     -- for completeness.
                     when others =>
                            NULL;
              end case;
       end if;
end process;
end arch;
```

3. Simulation

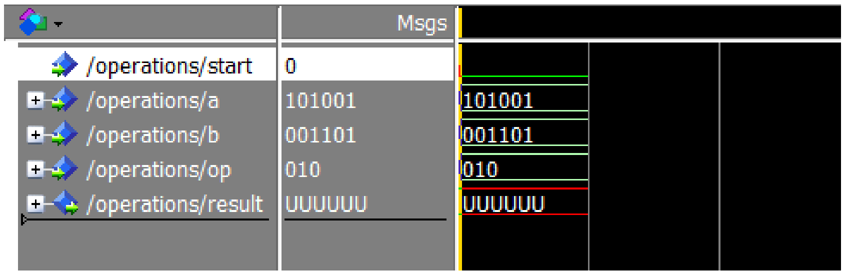
After you complete your circuit, you have to simulate every operation and check that it performs as expected. Here are some sample waveform simulations:

We decided to call our file **operations.vhd**. Feel free to name it any way you want.

Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

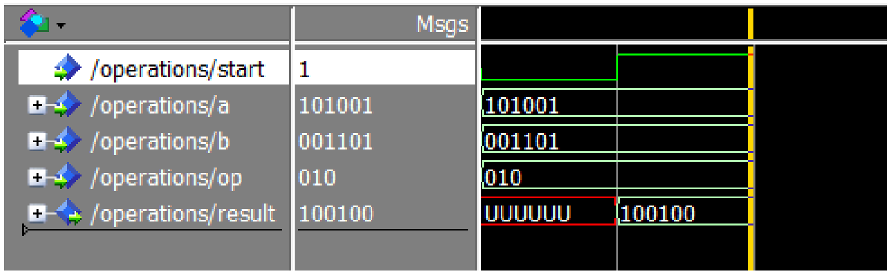
Instructor: Professor Izidor Gertner
April 1, 2019 by 11:59 AM

For our first run, we set start to 0 and set some arbitrary 6-bit values for our operands (in our case a and b). We set the value of op to 010, which we decided it will be the code for XOR. Result will be Undefined, since start=0, so operations are disabled.



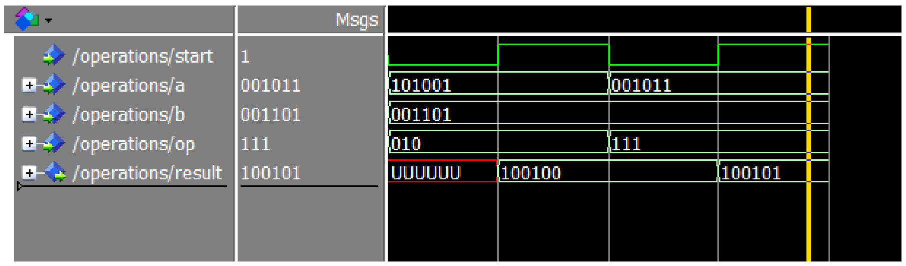
```
                       Msgs
/operations/start      0
/operations/a          101001    101001
/operations/b          001101    001101
/operations/op         010       010
/operations/result     UUUUUU    UUUUUU
```

For the second run we have start=1, so we allow the circuit to perform the specified operation, in this case XOR. Notice that now Result has a value which is the correct a XOR b value.



```
                       Msgs
/operations/start      1
/operations/a          101001    101001
/operations/b          001101    001101
/operations/op         010       010
/operations/result     100100    UUUUUU    100100
```

For our third run we disabled Start again (Start=0). We set our first operand and the operation we will perform is one-bit rotation to the right. At the fourth run we again enable Start and Result obtains the correct value of operand a rotated one bit to the right (001011 to 100101)



```
                       Msgs
/operations/start      1
/operations/a          001011    101001              001011
/operations/b          001101    001101
/operations/op         111       010                 111
/operations/result     100101    UUUUUU    100100              100101
```

4. Pin Assigments and behavior on DE2 board:

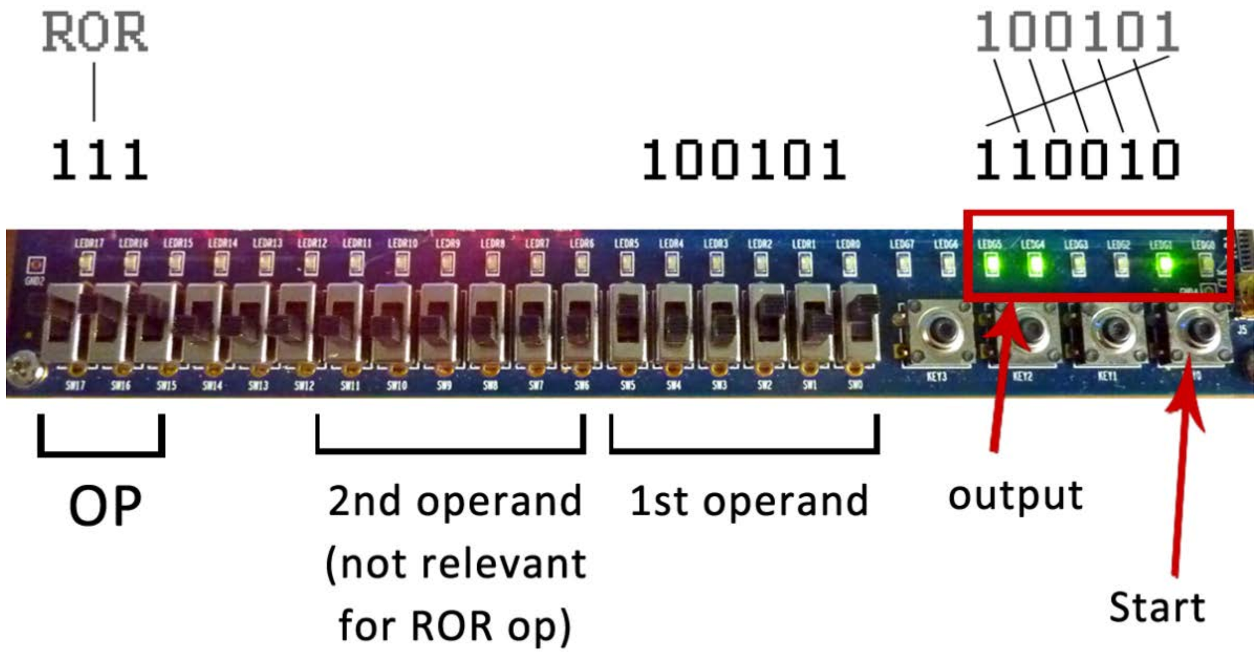
After testing your circuit's behavior using waveforms, it is now time to run your design in the DE2 board. Have your VHDL code in Quartus and compile. Then create a new text file to add your pin assignments:

For your inputs:
Start: Assign to Key 0
FIRST OPERAND[6 bits]: Assign to switches 0 to 5
SECOND OPERAND[6 bits]: Assign to switches 6 to 11
op[3 bits]: Assign to switches 15 to 17

For your output:
Result[6 bits]: Assign to green lights LEDG 0 to 5

Click here for pin assignment manual

Below is an example of the behavior in the DE2 board. Here we are performing rotation right, and in our case it is indicated by the code op=111 (your codes for op may differ). When we press the Start button, the operation is performed using whatever values we specify in the first operand using the switches 0 to 5. The switches for the second operand are not taken in consideration for this operation.



Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Instructor: Professor Izidor Gertner
April 1, 2019 by 11:59 AM

5. Report

Please write a lab report that addresses the following bullet points. Do not limit yourself to these; feel free to address more ideas/experiences with this lab in your summary.

Objective

- What is the goal of this assignment?

Design

- Include the corresponding VHDL code for this lab.
- Describe the VHDL code, inputs, outputs, width, etc.

Functionality

- Provide a walkthrough of the circuit behavior.
- List all the operations your circuit performs. Explain what each is supposed to do.
- Explain why you needed to implement a Start input.

Simulation

- Include screenshots of your waveform results from ModelSim for each operation.
- Analyze the waveforms obtained. Include an explanation of the waveform runs for each operation.
- Show what happens when Start=0 and Start=1 for different operations
- Take screenshots of the behavior in the DE2 board and include in your report as well. Provide a walkthrough of several operations as implemented in the DE2 board.

Analysis

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

Laboratory Project: Bitwise, Logical Shift, Arithmetic Shift and Rotation Operations

Instructor: Professor Izidor Gertner
April 1, 2019 by 11:59 AM

- Explain how and which of the operations you have designed in this lab can be used as multiplication or division by 2. Please give examples.
- How about multiplication or division by multiples of 2, can this also be implemented? How would your code need to be changed? Again, please give examples.
- 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?

Conclusion

- Give a brief explanation on what you learned about this assignment.