Lab 4: Parameterized N-bit structural multiplier

Goals

- The objective of this lab is to implement a 4-bit multiplier and then convert that into a parameterized N-bit structural multiplier
- The goal is that the student is familiarized with parameterization and the usage of busses and procedural description logic and generate statements mixing also module instantiation.

Introduction

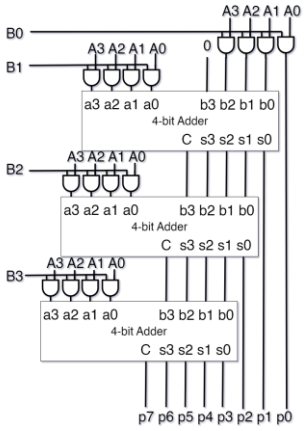
A multiplier circuit is a digital circuit designed to perform multiplication of binary numbers. It is a fundamental component in arithmetic logic units (ALUs), processors, and digital signal processors (DSPs). In this lab we will learn how to implement a 4-bit multiplier and then convert that into a N-bit multiplier. Then we will implement a multiplier using operands and operators to compare the output of both circuits.

Lab description

Design and implement a 4-bit multiplier and then scale this to a N-bit multiplier.

Step 1.

Create a 4-bit multiplier following the following diagram:



Step 2.

Once you have the 4-bit multiplier, you will create a function called “check_multiplication” where we will use the * operator multiplier. We will create a testbench and instantiate both our structural multiplier and use our function that uses the operator. In this testbench we will send stimuli to both (at least 1000) and then we will compare the output of both circuit. We will use the * multiplier function as our reference model and then print a “TEST PASSED” if all comparations are ok or “TEST FAILED” if one of the comparations are not.

Step 3.

After we have a working 4-bit multiplier, we will create a N-bit multiplier using this as a base. Create a draft of how the connections of the instances and wires would look like if they were named variables (for example: i, j, k) to figure out how you have to calculate the index for each of the iterations of the circuit for N-bits, use a generate block to generate the needed wires and instantiations.

We will add parameterization to the * operator multiplier as well.

In the testbench we will modify it to use parameterized inputs and outputs and then we will set the N of the multiplier to 32 and 64 and re-run the test to verify the functionality

Step 4.

Once it's complete we will use the circuit to implement the multiplier in the FPGA, we will use lab 1 once again for more readability in the output, however we will only use the decodifier (not the double dabble algorithm) because we want to see both input and outpuf as hexadecimal. For this the decodifier will have to be changed to support a,b,c,d,e,f:

The decodifier will use case statements and procedural blocks for easier implementation.

The multiplier will be changed to N=8.

Step 5.

In the FPGA implementation we will have the following:

- Use the following **switches** for input A: **SW[7] to SW[0]**
- Use the following **switches** for input B: **SW[15] to SW[8]**
- Use the following **7-segment** display for hexadecimal input A: **HEX[7] to HEX[6]**
- Use the following **7-segment** display for hexadecimal input B: **HEX[5] to HEX[4]**
- Use the following **7-segment** display for hexadecimal output product P: **HEX[3] to HEX[0]**

Submission

Implementation description

Generate a document providing a detailed description of the implementation. This should include an explanation of design decisions, challenges encountered, and how they were overcome. Include diagrams, schematics, tables, or equations used in the design development.

Video:

Upload a short video to the Teams group, recording the operation of the implemented circuit on the FPGA and providing a brief description of its functionality.