**CS211 Lab-7**

Faculty: Dr. Sharad Sinha

TA: Prachi Kashikr and Pavitra Bhade

In this lab you need to find the area and timing of arithmetic circuits like adders and multipliers and circuits for basic logic operations for different bit-widths. Besides, you will experiment with the functionality of ALU as well as the area and timing of ALU of different bit-widths. You need to find out which of the arithmetic or logic circuits affect the area and time complexity of ALU significantly, and the impact of word-length on such complexity. You are provided with parametrizable Verilog code where the bit-width can be changed. You should also understand the Verilog code completely and should develop competence to write the code.

# PART I: ARITHMETIC CIRCUITS

In this part of the Lab, we consider a simple adder and a multiplier. You will experiment with the functionality of each arithmetic circuits and find their area and time complexity for different bit-widths.

## ARITHMETIC CIRCUITS DESCRIPTION

1. **Adder:** The function of an adder is to add two input operands (A and B) to produce output (out). You can change the bit width of inputs to find the area and delay complexity of the adders of different bit-width.

A

+

B

o

u

t

Figure-1: Block diagram of adder.

1. **Multiplier :** The function of the multiplier is to multiply two input operands (A and B) to produce the output (out). Assuming both A and B to be W-bit words in 2’s complement representation, we can find that ‘out’ is a (2W)-bit word in 2’s complement representation.

**X**

out

A

B

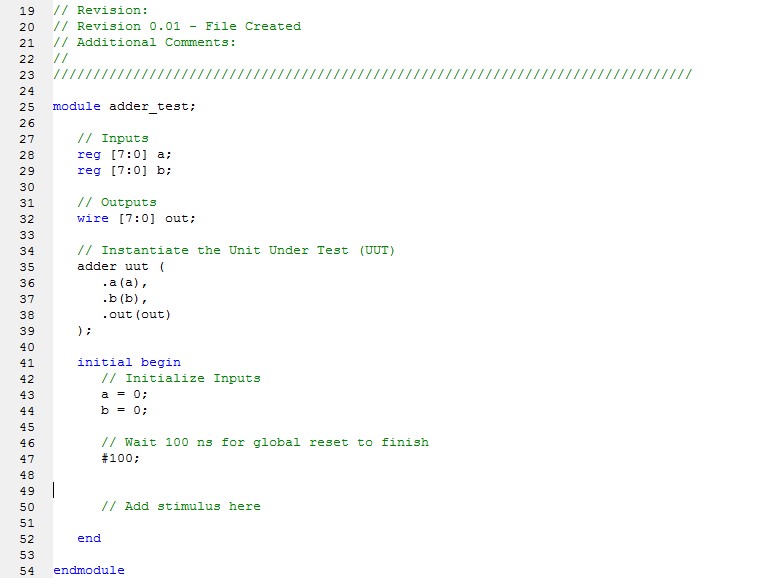
Figure-2: Block diagram of multiplier.

## ARITHMETIC CIRCUITS SYNTHESIS, TESTING AND ANALYSIS

1. You will be given the Verilog codes of an adder and a multiplier. You have to generate the test bench and test whether the Verilog modules give correct results.
2. You need to take the input bit-width 8 and 16 find out the number of slices used and the maximum combinational path delay of adders and multipliers of different bit-widths. You need to plot area (vs) bit-width as well as delay (vs) bit-width for the adder and the multiplier modules.

To find out whether the design is functioning correctly you need to synthesize code and to see if that gives the correct results. Besides you need to analyse the increase in complexity (in terms of number of slices and computational delay) of adder and multiplier along with the increase in bitwidth.

**For simulation:**



1. Add the following inputs below the ‘//Add stimulus here’ section in the program.

# 200 a=8’h01; //after 200ns make a=1;

#200 b=8’h02; // after 200ns make b=2;

1. Click ‘Simulate Behavioural model’ below ‘XSIM simulator’ for the processes ‘adder\_test’ to generate the output waveform. Verify whether the functionality of adder is correct.

A: The **adder** code is parametrizable. That means you can change the bit-width of the adder by changing the value for parameter **DSIZE**. If parameter **DSIZE** is set to 8, then after synthesis an eight bit adder circuit will be created in the FPGA. If the parameter **DSIZE** =64, then the bitwidth of the operands of the adder would be 64 bit.

1) Set the parameter **DSIZE** 8, 16, 32 and 64 one by one and note the number of slices used and delay (in ns) in different cases. You need to synthesize each time (steps 6-13) after you change the parameter value to find out the number of slices and delay in ns. Plot the graph, area (vs) bit-width as well as delay (vs) bit-width for the adder module.

Table 1: Slices and delay for adder

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter: **DSIZE** | BIT-WIDTH | No of LUT slices | Delay in ns |
| 8 | 8 |  |  |
| 16 | 16 |  |  |

To synthesize and to see its circuit diagram you need to make it as the topmodule and **repeat the steps.** You may have to provide input values of proper bit-width to the testbench program for adder in-order to verify the same. Note that the bit-width of the inputs is same as the bit-width parameter that you have set in the code.

B: The given **Multiplier** code is also parametrizable. That means you can change the bit-width of the operands of multiplier by changing the value for parameter **DSIZE**; If parameter **DSIZE**=8, then the bit-width of the operands in multiplier is 8 bit. If parameter **DSIZE**=64, then the bitwidth of the operands of the multiplier would be 64 bit.

* 1. Set the parameter **DSIZE** 8 and 16 (one by one) and note the number of slices occupied and delay in terms of (ns) for all cases. You need to synthesise each time after you change the parameter value to find out the number of slices and delay in ns. Plot the graph, area (vs) bit-width as well as delay (vs) bit-width for the multiplier module.

* 1. To synthesize and to see its circuit diagram set it as the topmodule and repeat the steps You have to add respective input values to the testbench program for multiplier in-order to verify the same. Note that the bit-width of the inputs is the same as the bit-width parameter that you set in the code.

Table 2: Slices and delay for Multiplier

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter: **DSIZE** | BIT-WIDTH | No of LUT slices | Delay in ns |
| 8 | 8 |  |  |
| 16 | 16 |  |  |

## EVALUATION -1

1. Plot the graph, area (vs) bit-width as well as delay (vs) bit-width for the adder module for DSIZE =8 and 16.
2. Plot the graph, area (vs) bit-width as well as delay (vs) bit-width for the multiplier module for DSIZE=8 and 16.
3. ARITHMETIC LOGIC UNIT (ALU)

ARITHMETIC LOGIC UNIT (ALU) SPECIFICATION

In this part of the Lab, we consider a simple ALU that performs the computation for eight arithmetic and logical operation. The seven operations are: ADD, SUB, AND, XOR, COM, MUL and ADDI as described in Table 1.

## **Table 1 - Description of ALU Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Equation** | **Operation** | **Description** |
| ADD | A+B | Addition | Addition of A and B, where both A and B are in 2’s complement representation |
| SUB | A-B | Subtraction | Subtraction of A and B, where both A and B are in 2’s complement representation |
| AND | A&B | Logical AND | Bit-wise AND of A, B |
| XOR | A^B | Logical XOR | Bit-wise XOR of A, B |
| COM | A << = B | Comparison | If A<=B then it results “true” (output 1) else “false (output=0) A and B are in 2’s complement representation |
| MUL | A\*B | Multiplication | Multiplication of A and B. A and B are in 2’s complement representation |
| ADDI | Note that the ALU can do another operation ADDI which will be explained and used in Lab-8. | | |

A and B are the data input ports of the ALU to feed maximum of two operands to the ALU. The ALU has 3-bit control input to perform one out of the 7 possible instructions which the ALU can perform. The information is also listed in Table 2. The encoding of the 7 instructions is listed in Table 3.

**Table 2 - Port List Specification. The bit-width to be varied from 8 bit to 64 bit.**

|  |  |  |
| --- | --- | --- |
| **Port Name** | **Port Direction** | **Description** |
| A | Input | First operand |
| B | Input | Second operand |
| op | Input | Indicates the ALU about the operation to be performed |
| Out | Output | Output of the operation |

## **Table 3 - ALU operation encoding**

|  |  |
| --- | --- |
| **Operation** | **‘op’ value** |
| ADD | 000 |
| SUB | 001 |
| AND | 010 |
| XOR | 011 |
| COM | 100 |
| MUL | 101 |
| ADDI | 110 |

## ALU IMPLEMENTATION, TESTING AND ANALYSIS

For this assignment,

1. You are given the Verilog code as well as the test bench for ALU. You have to test whether the ALU gives correct results.
2. You need to set the input bit-width to 8 and 16 (one by one) and find out the number of slices used and the maximum combinational path delay in each case. You need to plot area (vs) bit-width as well as delay (vs) bit-width for the ALU module. In FPGA you cannot find area directly so instead of area you can take number of slices, which would be considered proportional to the area.

## DETAILS

1. The given ALU code is parametrizable but initialized to 8 bits. To find out whether the design is functioning correctly you need to synthesise code and to see if that gives the correct results.

1. The ALU code given in parametrizable. That means we can change the bit-width of the ALU by changing the value for parameter DSIZE;

If parameter DSIZE=8, then the bit-width of the operands of ALU is 8 bit.

If parameter DSIZE=16, then the bit-width of the operands of ALU is 64 bit.

* + - * 1. Change the parameter DSIZE to 8 and 16 (one by one) and note the number of slices occupied and delay in terms of (ns). Plot the graph, area (vs) bitwidth as well as delay (vs) bit-width for the ALU module.

Table 3: Slices and delay for ALU module

|  |  |  |  |
| --- | --- | --- | --- |
| DSIZE | BIT-WIDTH | No of slices | Delay in ns |
| 8 | 8 |  |  |
| 16 | 16 |  |  |

## EVALUATION -2

1. Plot the graph, area (vs) bit-width as well as delay (vs) bit-width for the ALU module for DSIZE =8 and 16.