System Design Document

for the ALU

University at Buffalo, The State University at New York

EE478 Fall 2019

The information enclosed in this document and the source code generated for this laboratory assignment were generated by the named students on this cover page. Detection of plagiarism in a submitted lab or project design document will result in an automatic failing grade in the course by all parties involved.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Written By** | Daniel Giovino |  | **Revision History** | | | |
| **Engineer** | Daniel Giovino |  | **Rev** | **Date** | **Session** | **Approved/Grade** |
| **Engineer** | Peter M. VanNostrand |  | Orig | 09/23/19 | Mon. 9AM |  |
| **Teaching Assistant** |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  | **EE478F19**  **LAB 2** |

# Table of Contents

Table of Contents 2

List of Figures 2

List of Tables 2

1. Introduction 3

1.1 Overview 3

1.2 Document Scope 3

1.3 Intended Audience 3

2. System Design Overview 4

2.1 System Block Diagram and Description 4

2.2 ALU Module 5

2.3 Adder Module 6

2.4 ASR Module 7

2.5 Multiplier Module 8

2.6 Comparator Module 9

3. Testing and Verification 10

3.1 Testbench and Simulation 10

3.2 Objective Verification 10

Glossary 11

List of Abbreviations 11

Hardware References 11

# List of Figures

Figure 1: ALU System Diagram 4

Figure 2: ALU Simulation Waveform 9

# List of Tables

Table 1: ALU Truth Table 5

Table 2: Adder Truth Table 5

Table 3: ASR Truth Table 6

Table 4: Multiplier Truth Table 7

Table 5: Comparator Truth Table 8

# Introduction

## Overview

An ALU is a device which will compute different operations on its inputs based on an opcode. This ALU supports five different opcodes for adding, multiplying, arithmetic shift right, comparing, and bitwise exclusive or. The inputs into these different operations are two 2-bit inputs from slider switches. The opcode is provided by four buttons. This circuit was designed on paper and then implemented and tested on a Xilinx FPGA using the Xilinx Vivado CAD tool. The output is displayed on the board using the four LEDs located on the board.

## Document Scope

This document has been written to provide information regarding the ALU circuit developed in Lab 2 of EE478. The inputs, outputs, and operation of this device are described below as well as details regarding implementation on an FPGA and testing using a simulated test bench.

## Intended Audience

This document is intended for use by the EE478 TAs and professors for use in grading as well as by the members of group 1L in the Monday 9am lab session.

# System Design Overview

## System Block Diagram and Description

The ALU takes two, 2-bit inputs A and B, which are labeled as A0, A1, B0, and B1 in Figure 1. These inputs are connected to four slider switches on the Xilinx board. It also takes an opcode from BTN0, BTN1, BTN2, and BTN3, which are labeled as such in the following diagram.

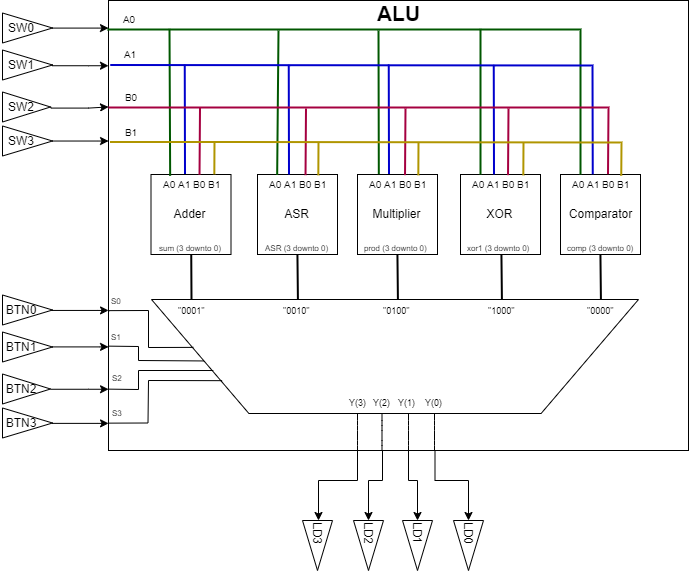


Figure 1: ALU System Diagram

The input bits from vectors A and B (each containing 2 bits) are fed to each of the following modules: Adder, Arithmetic Shift Right (ASR), Multiplier, XOR, Comparator (See following sections for details on each). Each of these modules output a 4-bit signal which is sent to a MUX which uses the opcode to determine which operation it should output to the final output of the system, which is displayed on the LEDs labeled LD0, LD1, LD2, and LD3.

## ALU Module

This module implements the ALU. The ALU takes two 2-bit inputs A and B from the switches and a 4-bit input S from the buttons. It is assumed that no more than one button will be pressed at any given time. The truth table for the ALU is given by:

Table : ALU Truth Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S(3) | S(2) | S(1) | S(0) | Y(3 downto 0) |
| 0 | 0 | 0 | 0 | comp(3 downto 0) |
| 0 | 0 | 0 | 1 | sum(3 downto 0) |
| 0 | 0 | 1 | 0 | ASR(3 downto 0) |
| 0 | 1 | 0 | 0 | prod(3 downto 0) |
| 1 | 0 | 0 | 0 | xor1(3 downto 0) |

The above truth table depicts the MUX at the output of the ALU. To include the inputs A and B in the truth table, the table would have needed to be 256 lines long, which seems impractical. The vectors comp, sum, ASR, prod, and xor1 are all outputs of submodules described in the following sections with how they are derived from A and B.

## Adder Module

This module implements an adder circuit. The adder receives two two-bit signed inputs A and B computes their four-bit signed sum. The sum is stored to the vector “sum” which is 4 bits in length. The truth table of the adder is given by:

Table : Adder Truth Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A(1) | A(0) | B(1) | B(0) | sum(3) | sum(2) | sum(1) | sum(0) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

The adder was implemented using behavioral VHDL. The signals A and B are casted to signed vectors of length 4 by first casting them to integers and then back to signed with a length of 4. Then, they are simply added using behavioral VHDL and connected to the “sum” vector.

## ASR Module

This module implements an arithmetic shift right (ASR) circuit. The module receives two two-bit signed inputs A and B computes the ASR of A by B bits. Then, two ‘0’ bits are concatenated to the beginning of the output so all outputs will have the same lengths for the MUX. The truth table of the ASR module is:

Table : ASR Truth Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A(1) | A(0) | B(1) | B(0) | ASR(3) | ASR(2) | ASR(1) | ASR(0) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

The ASR was implemented in VHDL using two cases, when B is “00” and when B is not “00”. This creates a MUX. When B is “00”, the output is A with “00” concatenated in front. When B is not “00”, the output is “00” concatenated in front of “A(1) A(1)”.

## Multiplier Module

This module implements a multiplier circuit. The adder receives two two-bit unsigned inputs A and B computes their four-bit unsigned product. The product is stored to the vector “prod” which is 4 bits in length. The truth table of the multiplier is given by:

Table : Multiplier Truth Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A(1) | A(0) | B(1) | B(0) | prod(3) | prod(2) | prod(1) | prod(0) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

The multiplier was implemented using behavioral VHDL. The signals A and B are first casted to unsigned vectors. Then, they are simply multiplied using behavioral VHDL and connected to the “prod” vector. This module needs a 4-bit output since the maximum possible input “11” times “11” is “1001”.

## Comparator Module

This module implements a comparator circuit. The comparator receives two two-bit unsigned inputs A and B, then determines if A is greater than B or not. The output is a 4-bit vector named comp.

Table : Comparator Truth Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A(1) | A(0) | B(1) | B(0) | comp(3) | comp(2) | comp(1) | comp(0) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

The comparator was implemented in VHDL by first casting A and B to unsigned vectors. If A is greater than B, the least significant bit of the output is set to ‘1’. If it is not, the least significant bit of the output is set to ‘0’. In either case, the first 3 bits of the output are set to ‘0’ for consistency with the MUX that follows this module.

# Testing and Verification

To ensure that our circuit functioned as intended we performed two forms of testing. First we simulated the designed using Xilinx Vivado and examined the simulated response to different inputs. After debugging and iterating we synthesized our design and loaded in onto an FPGA where we performed testing of the physical circuit.

## Testbench and Simulation

To test our design in simulation we developed a test bench which tests the full range of functionality of the circuit. Our simulation first creates an ALU module, then drives the inputs to important test values. We then visualized these circuits using the ISim tool in Xilinx Vivado and compared the output to the expected values, which are shown just below the output value (Y) in the screenshot shown below.

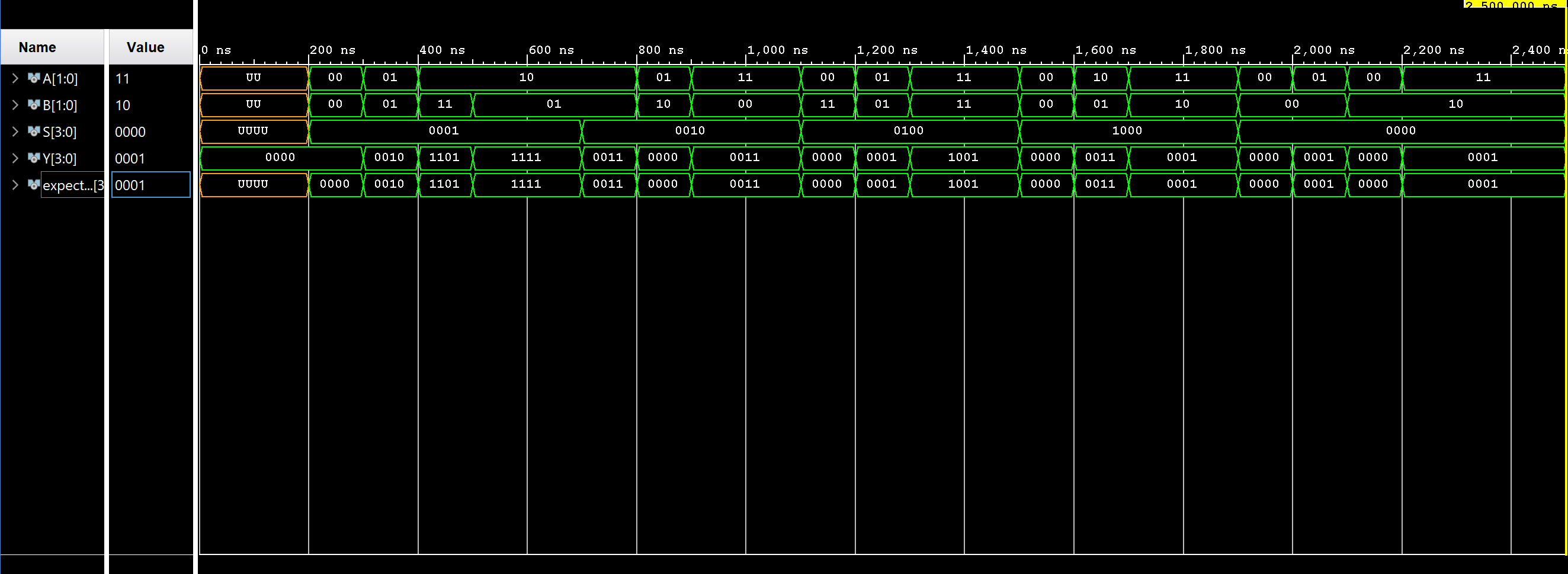


Figure : ALU Simulation Waveform

The time period from 200ns – 700ns tests the adder functionality, 700ns – 1,110ns tests the ASR functionality, 1,100ns – 1,500ns tests the multiplier functionality, 1,500ns – 1,900ns tests the XOR functionality, and finally from 1,900ns to the end of the chart tests the comparator functionality. Since the output values match the expected values throughout the each of these tests, the circuit has been verified and we were able to proceed to synthesize the circuit onto the FPGA board.

## Objective Verification

Once we synthesized our design and loaded it onto the FPGA we tested the circuit by manipulating the four slider switches built into the Xilinx board to drive the inputs A and B. We observed the expected output signals on the LEDs. We repeated this process while in each of the different select modes by pressing the proper buttons and verified the entire circuit.

# Glossary

This section includes helpful supplemental information to aid in understanding the content of this report.

## List of Abbreviations

Below are selected abbreviations used in this document

* VHDL: VHSIC Hardware Description Language
* VHISC: Very High Speed Integrated Circuit
* FPGA: Field Programmable Gate Array
* LED: Light Emitting Diode
* CAD: Computer Aided Design
* MUX: Multiplexor
* XOR: Exclusive Or
* ALU: Arithmetic Logic Unit
* ASR: Arithmetic Shift Right

## Hardware References

* Zybo Z7 Manual ([DigitalInc](https://reference.digilentinc.com/_media/reference/programmable-logic/zybo-z7/zybo-z7_rm.pdf))
* Xilinx Vivado CAD tool ([Xilinx](https://www.xilinx.com/products/design-tools/vivado.html))