End Semester Project Report on

Design of Arithmetic Logic Unit using FPGA

Submitted for the partial fulfillment of the subject

Logic Design Workshop (EET1200)

Submitted by

Sushil Mahendra Roul (2241018199)

S Gourav (2241007045)

Aryan Kumar (2241011244)

Shreyansha Mahatwo (2241013386)

B. Tech. (EE) 2nd Semester (Section – 2241048)

DEPT. OF ELECTRONICS & COMMUNICATION ENGINEERING
Institute of Technical Education and Research

SIKSHA 'O' ANUSANDHAN
DEEMED TO BE UNIVERSITY

Bhubaneswar, Odisha, India.

(June, 2023)

Abstract

This project is based on the design and implementation of a **Arithmetic Logic Unit (ALU) using FPGA**. The project aims to develop a compact and efficient 4-bit ALU capable of performing various arithmetic and logical operations on 4-bit binary numbers.

The purpose of this project is to address the research problem of designing an ALU with support for arithmetic and logical operations on 4-bit binary numbers. The project employs the Verilog hardware description language to model and implement the ALU's logic circuits and interconnections.

To achieve this, the project begins with a detailed study of the ALU's functional requirements, including the supported operations, input/output signals, and control signals. The design follows a modular approach, allowing for easier expansion and modification. The project analyzes relevant documents and references to identify the best practices and techniques for ALU design.

Based on the analysis, the project implements an ALU design that incorporates four main components: the arithmetic unit, logic unit, control unit, and multiplexer. The arithmetic unit handles addition, subtraction, multiplication, division, and other arithmetic operations. The logic unit performs logical AND, OR, XOR, NAND, NOR, and XNOR operations. The control unit generates appropriate control signals based on input select lines to determine the desired operation. The multiplexer selects the output based on the control signals.

Preliminary results of the investigation demonstrate that the implemented 4-bit ALU successfully performs the desired arithmetic and logical operations. The ALU exhibits efficiency, speed, and accuracy in executing operations on 4-bit binary numbers.

The significance of this research project lies in its contribution to the field of digital systems design and computer architecture. The developed ALU provides a practical solution for performing arithmetic and logical operations on 4-bit binary numbers, catering to the needs of various digital systems. The project enhances our understanding of FPGA-based ALU designs and their applications, paving the way for further advancements in the field.

In conclusion, this project successfully addresses the research problem of designing a 4-bit ALU and provides insights into the implementation and performance of such ALUs. The results of this inquiry contribute to the existing body of knowledge and offer a valuable resource for future researchers and designers in the field of digital systems.

Contents

Chapter 01: Introduction

1.1. Introduction

The design and implementation of Arithmetic Logic Units (ALUs) play a vital role in digital systems, enabling them to perform various arithmetic and logical operations. This project focuses on the development of a 4-bit ALU using Verilog language. The objective is to create a compact and efficient ALU capable of executing arithmetic and logical operations on 4-bit binary numbers. By exploring the functional requirements, employing a modular design approach, and utilizing Verilog, this project aims to provide a comprehensive solution for implementing a versatile 4-bit ALU. The project also emphasizes the significance of the ALU in digital systems and its potential applications in various domains.

1.2. Background

The ALU typically operates on binary data, which consists of 1s and 0s, representing the on and off states of electronic switches or transistors within a computer. The ALU's primary function is to perform arithmetic operations, such as addition, subtraction, multiplication, and division, as well as logical operations, including NOT, AND, OR, NAND, NOR, XOR and XNOR operations.

1.3. Project Objectives

The objective is to design and implement a 4-bit Arithmetic Logic Unit (ALU) capable of performing arithmetic and logical operations on 4-bit binary numbers.

The primary objectives of this project are as follows:

1. Design and implement a 4-bit Arithmetic Logic Unit (ALU): The main goal is to develop a functional ALU capable of performing arithmetic and logical operations on 4-bit binary numbers. The ALU should support essential operations such as addition, subtraction, multiplication, division,NOT, AND, OR, NAND, NOR, XOR, XNOR, and comparison.

2. Modular and efficient design: The ALU design should follow a modular approach, allowing for easy expansion and modification in the future. It should be optimized for efficient utilization of hardware resources, ensuring optimal performance and minimal resource consumption.

3. Verilog implementation: Utilize Verilog hardware description language to model and implement the logic circuits and interconnections of the ALU. The Verilog code should accurately represent the desired functionality of the ALU, facilitating synthesis, placement, and routing onto the FPGA chip.

4. Testing and verification: Develop comprehensive test cases to verify the correctness and accuracy of the implemented ALU. Thoroughly test the ALU for different input combinations, boundary cases, and corner cases to ensure reliable operation.

5. Performance analysis: Evaluate the performance of the ALU in terms of speed, power consumption, and resource utilization. Compare the performance metrics against design objectives to assess the efficiency of the implemented ALU.

6. Documentation and reporting: Prepare detailed documentation and project reports that provide an overview of the project, design considerations, implementation details, test results, and analysis. The documentation should be well-structured and organized, allowing for easy understanding and future reference.

1.4. Scope

In summary, the 4-bit ALU is a critical component in digital systems, responsible for performing arithmetic and logical operations on 4-bit binary numbers. Its design incorporates logic gates and arithmetic circuits to execute various operations efficiently. The ALU finds extensive applications in microprocessors and other digital systems, enabling them to process data and perform calculations accurately and swiftly, making it a key component in the foundation of digital systems and a stepping stone toward more complex computing technologies.

1.5. Project Management

According to the PMBOK Guide (Project Management Body of Knowledge), a project management life cycle consists of 5 distinct phases including initiation, planning, execution, review, and closure that combine to turn a project idea into a working product.

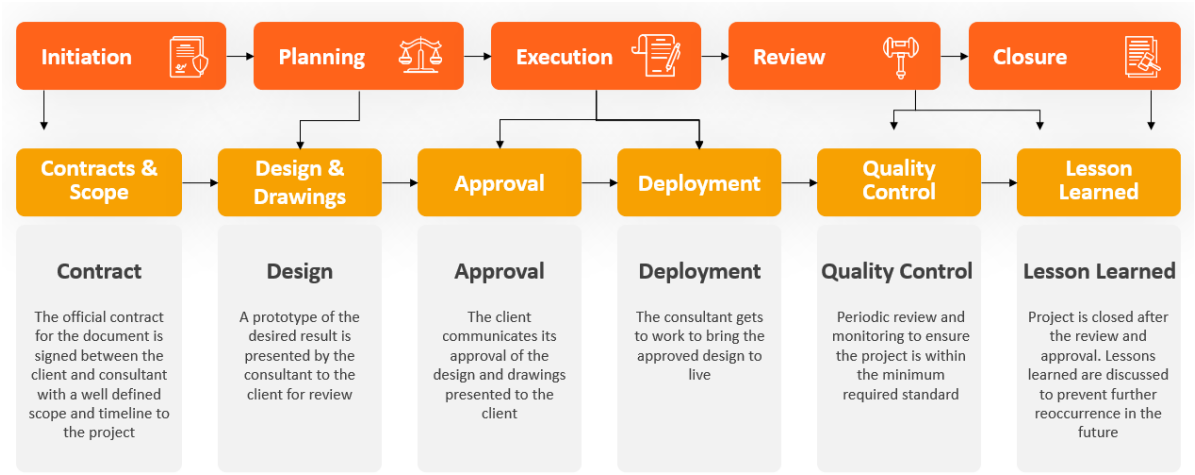


Figure 1. Model of phases in project management.

The project initiation phase is the first stage of turning an abstract idea into a meaningful goal. In this stage, we need to develop a business case and define the project on a broad level.

The project planning stage requires complete diligence as it lays out the project's roadmap.

The project execution stage is where the project team does the actual work. The job of a project manager is to establish efficient workflows and carefully monitor the progress of the team.

In the project management process, the third and fourth phases are not sequential in nature. The project monitoring and controlling phase run simultaneously with project execution.

The project closure stage indicates the end of the project after the final delivery.

1.6. Overview and Benefits

A 4-Bit ALU has various benefits ranging from arithmetic operations, having a small width size compared to larger bits ALU, has a small consumption of power as has few numbers of transistors. It can also be used in education setting to introduce students to digital circuits. It can also be integrated to more extensive systems or processors as they are building blocks of larger ALUs or CPUs with higher bit widths.

1.7. Organization of the Report

The report is organised into the following chapters. Each chapter is unique on its own and is described with necessary theory to comprehend it.

Chapter 2 deals with background survey and review, Chapter 3 has the description of the theoretical aspects that has been acquired to commence the project work.

Chapter 02: Theoretical Aspects

2.1. Background Theory and Modeling

Arithmetic Logic Units (ALUs) are essential components in digital systems that perform arithmetic and logical operations on binary numbers. They are widely used in various applications, including microprocessors and calculators, to execute calculations and decision-making tasks.

In this project, the focus is on designing and implementing a 4-bit ALU. The 4-bit ALU operates on binary numbers with four bits, allowing for computations on numbers ranging from 0 to 15. The design involves creating functional units for arithmetic operations (such as addition, subtraction, multiplication, and division) and logic operations (including NOT, AND, OR, XOR, XNOR, etc and comparison).

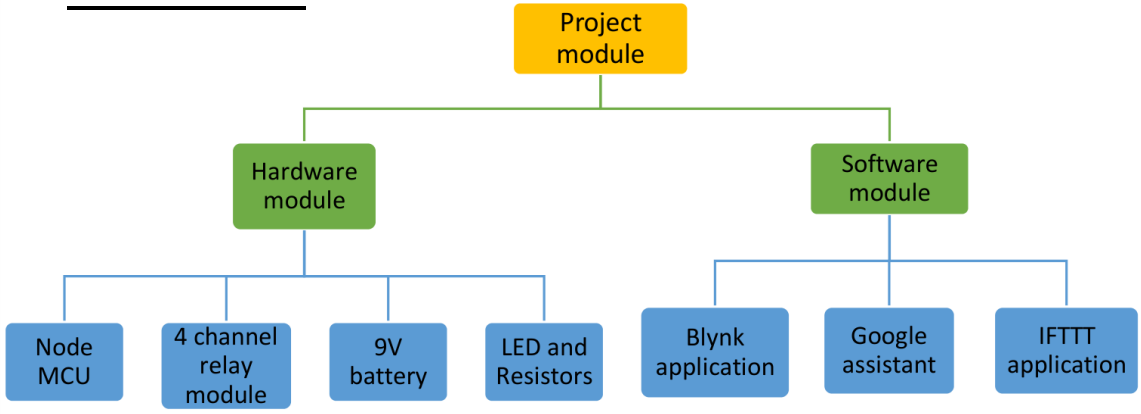


Figure 2. Layout of project module

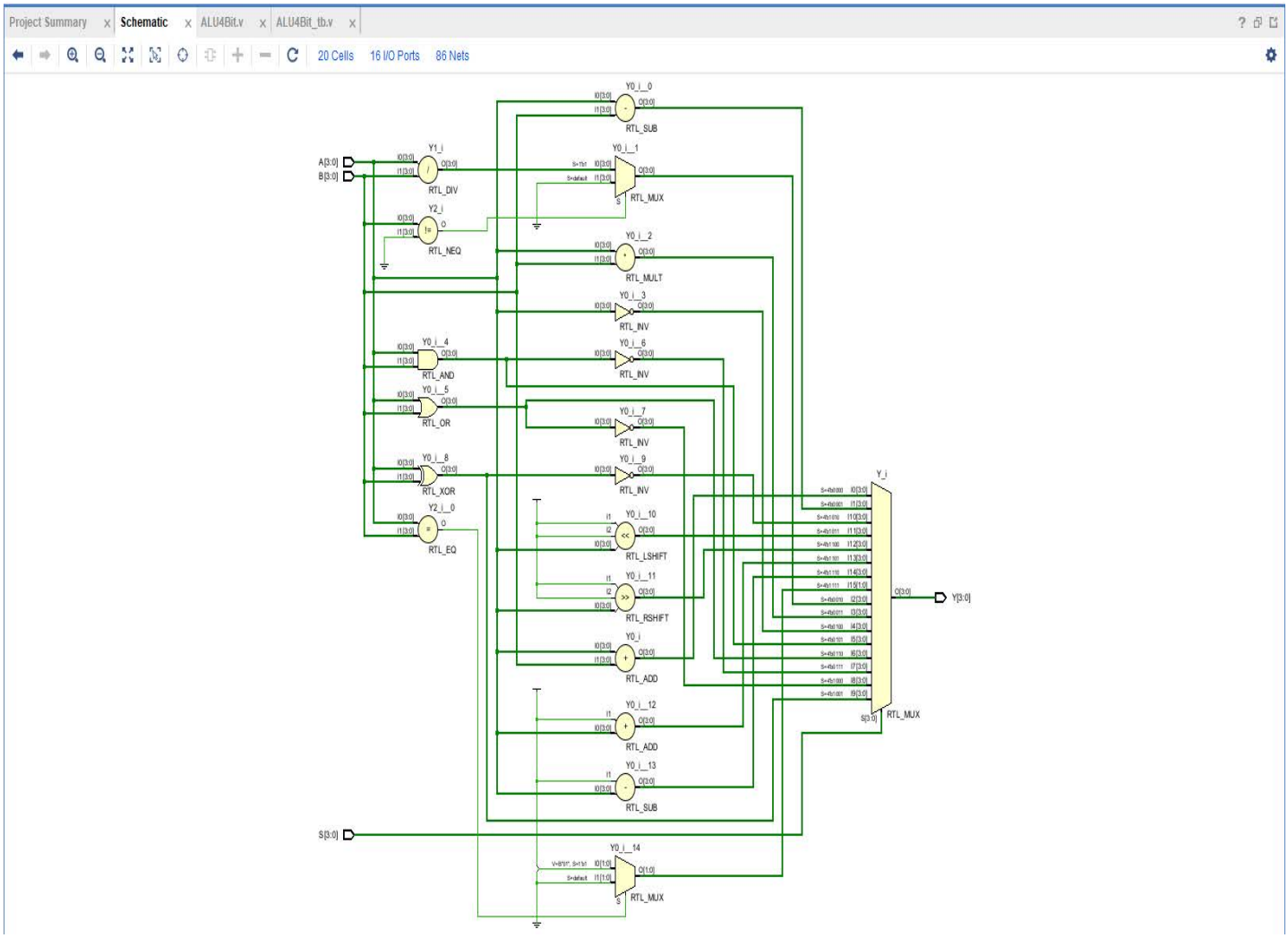
2.2.1. Brief Description

The 4-bit ALU is commonly used in various digital systems, including microprocessors, microcontrollers, and calculators. It forms a vital part of the central processing unit (CPU) in a computer, where it carries out arithmetic and logical operations on binary data. The ALU's compact size and efficiency make it suitable for integration into larger circuits, enabling the development of complex computational systems.

The design follows a modular approach, dividing the ALU into distinct units for arithmetic operations, logic operations, control, and multiplexing. Through the use of Verilog, the logic circuits and interconnections of the ALU are described and implemented, ensuring correct functionality and performance. The project offers valuable insights into the design and implementation of ALUs, contributing to the field of digital systems design.

Advancements in technology have led to the development of ALUs with higher bit widths, such as 8-bit, 16-bit, and 32-bit ALUs, to handle more extensive calculations and support larger data processing. However, the 4-bit ALU serves as a fundamental building block for understanding the principles and functionality of ALUs.

2.3. Block diagram of the Proposed System



2.3.1. Working of the system

In the block diagram of the 4-bit ALU, there are two 4-bit inputs, A and B, a 4-bit select line, S, and a 4-bit output, Y. The select line allows for 16 different inputs, each corresponding to a unique operation.

The operations performed by the ALU based on the select line are as follows:

S0 - Addition of A and B

S1 - Subtraction of B from A

S2 - Division of A by B

S3 - Multiplication of A and B

S4 - Logical NOT operation on A

S5 - Logical AND operation between A and B

S6 - Logical OR operation between A and B

S7 - Logical NAND operation between A and B

S8 - Logical NOR operation between A and B

S9 - Logical XOR operation between A and B

S10 - Logical XNOR operation between A and B

S11 - Logical shift left of A by 1

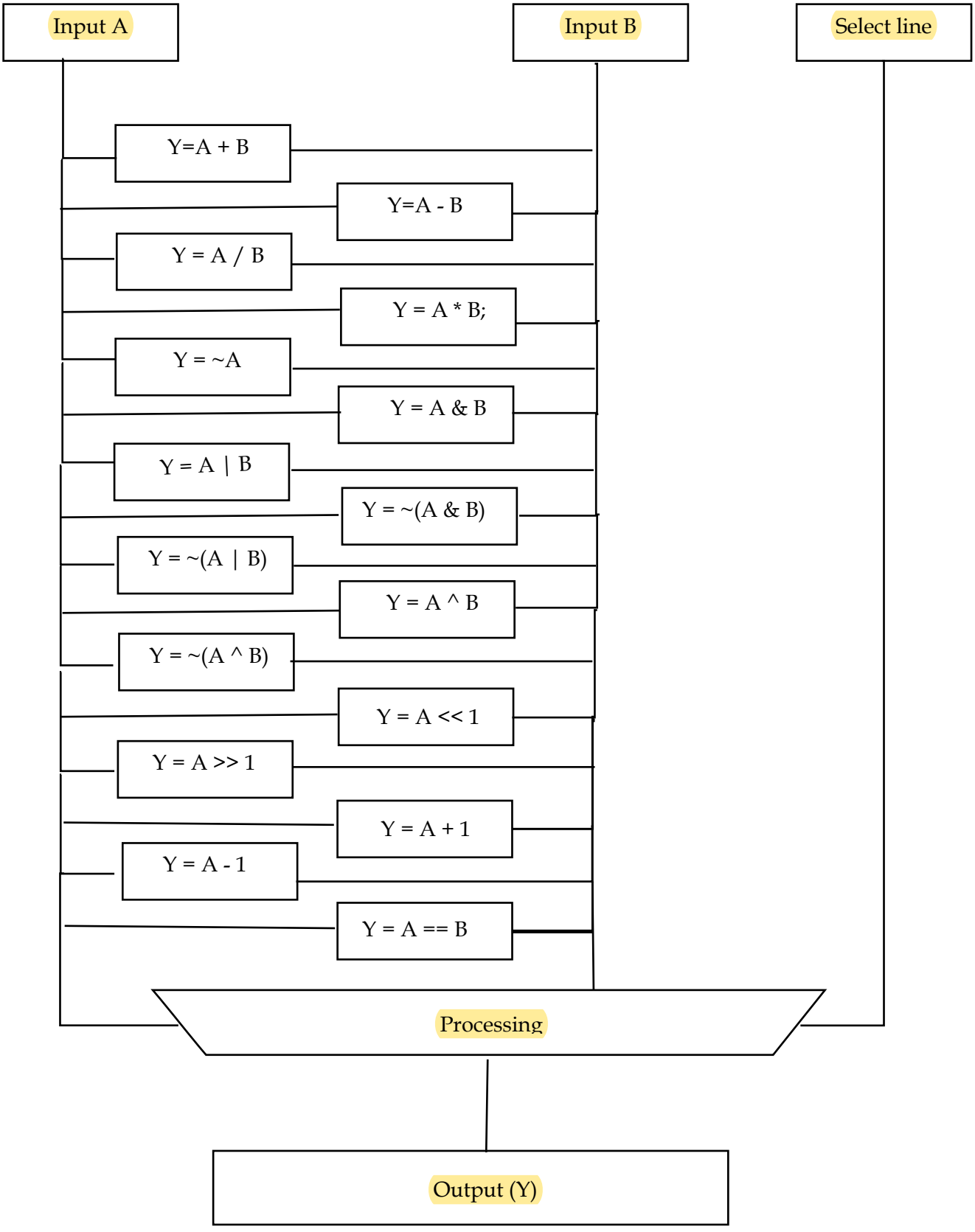
S12 - Logical shift right of A by 1

S13 - Increment of A by 1

S14 - Decrement of A by 1

S15 - Equal comparison between A and B

2.3.2. Flow Chart / Pseudocode



Chapter 03: Hardware and Software Requirements

3.1. Xilinx Vivado

Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of hardware description language (HDL) designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis. Vivado represents a ground-up rewrite and re-thinking of the entire design flow (compared to ISE). Like the later versions of ISE, Vivado includes the in-built logic simulator. Vivado also introduces high-level synthesis, with a toolchain that converts C code into programmable logic.

Vivado was introduced in April 2012, and is an integrated design environment (IDE) with system-to-IC level tools built on a shared scalable data model and a common debug environment. Vivado includes electronic system level (ESL) design tools for synthesizing and verifying C-based algorithmic IP; standards based packaging of both algorithmic and RTL IP for reuse; standards based IP stitching and systems integration of all types of system building blocks; and the verification of blocks and systems.

3.1.1. Why VIVADO ?

Vivado design suite accelerates design productivity due to the following reasons;

1. Vivado Design Suite allows you to push the device-density envelope farther
2. Vivado Design Suite delivers robust performance and low power with predictable results
3. Vivado Design Suite delivers unparalleled run time and memory utilization
4. Vivado HLS lets you quickly generate IP using descriptions written in C, C++, or SystemC
5. Vivado Design Suite supports model-based DSP design integration using Simulink and MATLAB from the MathWorks.
6. Vivado IP Integrator shatters the RTL design-productivity barrier
7. Vivado Integrated Design Environment provides a unified IDE for Design and Simulation
8. Vivado Design Suite provides comprehensive hardware debug
9. Vivado HLS accelerates verification by >100X using C, C++ or SystemC

3.1.2. Extending the Vivado IP Repository

Vivado has an extensible IP catalog that can include Xilinx and third-party IPs. Vivado enables engineers to quickly turn a part of their design or algorithm into a reusable IP added to the Vivado IP catalog.

As illustrated in Figure 1, with the Vivado IP Packager, all the associated files of the design, such as constraints, test benches and documentation, can be added to the created IP.

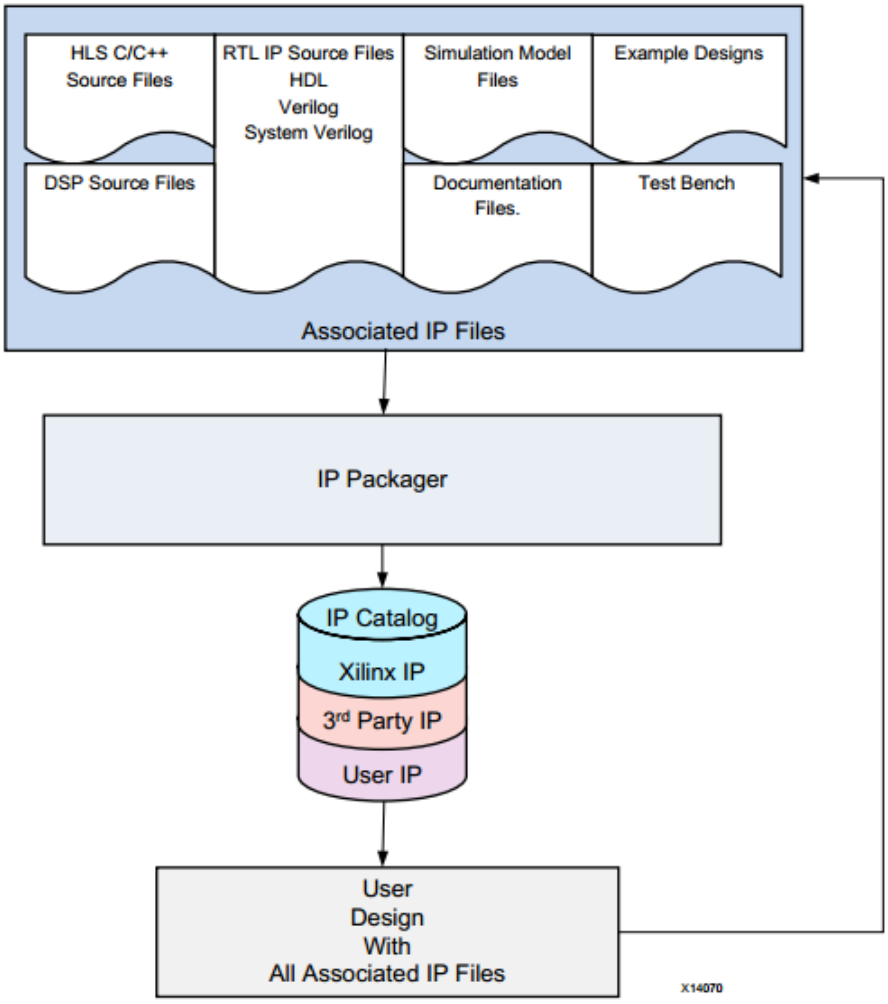


Figure 1. Vivado IP creation flow. Image courtesy of Xilinx.

One of the important features of the Vivado IP flow is the ability to create an IP at any level of a design, no matter if it's a Register-Transfer Level (RTL) design, a netlist, a placed netlist, or even a placed-and-routed netlist.

Also note that, as shown in Figure 1, the source files of the IP Packager can include MATLAB/Simulink algorithms from Xilinx System Generator or C/C++/SystemC algorithms from the Vivado High-Level Synthesis (HLS). In these cases, the proprietary IP generation is further accelerated because a higher level of description is utilized to develop the target algorithm. The HLS will be briefly discussed in the next section.

3.1.3. Applications of VIVADO

Aerospace and Defense	Engaging the industry, developing deep understanding, and collaborating on development; AMD delivers the broadest product portfolio for enabling the next generation of advancements in ground-based, airborne, and space systems.

Emulation and Prototyping	Powering Next-Generation in Hardware Assisted Verification
Automotive Solutions	Powering Next-Generation Automotive Systems
AMD in Professional AV & Broadcast	AMD platforms can quickly adapt to emerging audio and video technologies, enable access to AV-over-IP networks with lossy and lossless codecs where needed, and integrate multimedia pipelines with the latest AI/ML techniques in cost-optimized devices
Emerging Consumer Trends	AMD offers the most comprehensive portfolio of SoCs and FPGAs combined with industry leading tools for software and hardware developers. Our solutions support a wide range of consumer applications, whether its 8K video networking, fast and reliable printing, complex AI analytics, or just integrating multiple devices into a single chip, we can help you meet your commercial and technical needs.
Acceleration for the Modern Data Center	Advances in artificial intelligence, increasingly complex workloads, and an explosion of unstructured data are forcing rapid evolution of the data center. The AMD platform is powering this revolution through adaptable acceleration of compute, storage, and networking.
High Performance Computing	Teams of modern-day explorers leverage high performance computing (HPC) as their tool for discovery and path to solving some of the world's most impactful and complex problems. Accelerating time to insight and deploying HPC at scale requires incredible amounts of raw compute capability, energy efficiency, and adaptability that are uniquely found in the AMD platform.
Industrial Solutions across Edge and Cloud	AMD is the leader in scalable Industrial IoT platforms by offering heterogeneous embedded processing, I/O flexibility, hardware-based deterministic control, and comprehensive solutions for the lowest total cost of ownership. AMD powers intelligent and adaptive assets in harsh environments over industrial lifecycles.
Smart Solutions for Healthcare: Imaging, Diagnostics, and Clinical Equipment	Address the growing needs of scalable healthcare platforms with heterogenous multi-processing, I/O flexibility, hardware-based deterministic controls and comprehensive solutions in cybersecurity, safety and machine learning.
Test & Measurement	AMD Test & Measurement Solutions offer I/O Performance and Flexibility, Signal Processing Bandwidth and Partial Re-configurability to Enable Next Generation T&M Platforms
Industry Solutions	AMD has been a trusted provider of programmable devices in wired and wireless networking equipment, spanning all corners of the network. From 3G, 4G, and 5G towers, gigabit to multi-terabit transport gear, all the way through the edge and into the cloud, chances are this very website passed through more than one AMD device on its way to your screen.

3.2. Basys 3 Artix-7 FPGA Board

The Basys 3 board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx. With its high-capacity FPGA (Xilinx part number XC7A35T-1CPG236C), low overall cost, and collection of USB, VGA, and other ports, the Basys 3 can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs and other I/O devices to allow a large number designs to be completed without the need for any additional hardware,

and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits.

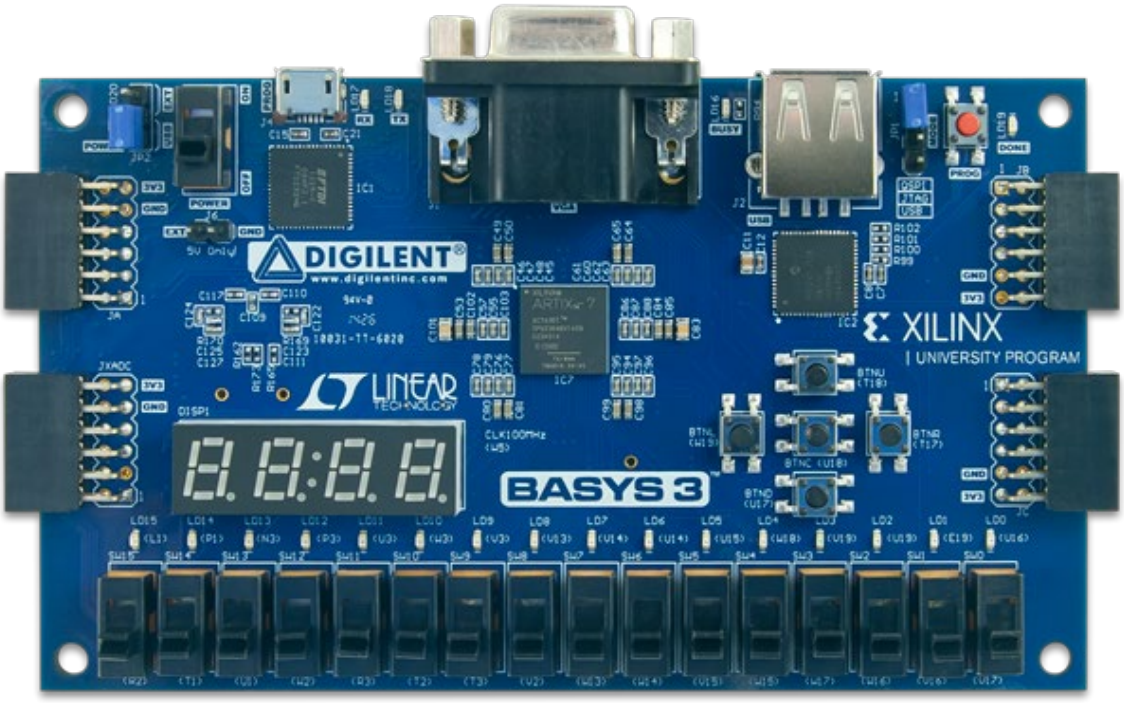


Fig.3.1. The Basys 3 FPGA Board

3.2.1 Features of Basys 3

The Artix-7 FPGA is optimized for high performance logic, and offers more capacity, higher performance, and more resources than earlier designs. Artix-7 35T features include:

- 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops);
- 1,800 Kbits of fast block RAM;
- Five clock management tiles, each with a phase-locked loop (PLL);
- 90 DSP slices;
- Internal clock speeds exceeding 450MHz;
- On-chip analog-to-digital converter (XADC).

The Basys 3 also offers an improved collection of ports and peripherals, including:

- 16 user switches
- 16 user LEDs
- 5 user pushbuttons
- 4-digit 7-segment display
- Three Pmod ports
- Pmod for XADC signals
- 12-bit VGA output
- USB-UART Bridge
- Serial Flash
- Digilent USB-JTAG port for FPGA programming and communication
- USB HID Host for mice, keyboards and memory sticks

3.2.2 Overview of the Board and is description

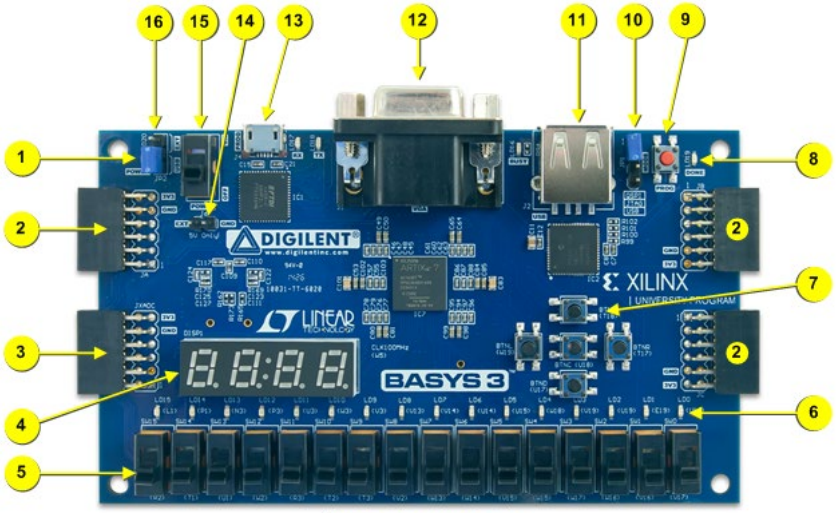


Figure 1. Basys3 board features

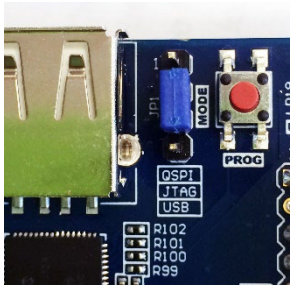
Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

The Basys 3 works with Xilinx's new high-performance Vivado ® Design Suite. Vivado includes many new tools and design flows that facilitate and enhance the latest design methods. It runs faster, allows better use of FPGA resources, and allows designers to focus their time evaluating design alternatives. The System Edition includes an on-chip logic analyzer, high-level synthesis tool, and other cutting-edge tools, and the free "WebPACK" version allows Basys 3 designs to be created at no additional cost.

3.2.3 Programming Modes

JTAG Programming

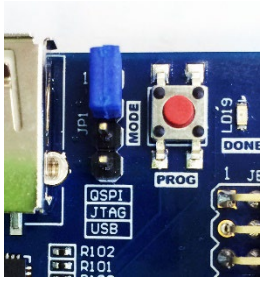
The Xilinx tools typically communicate with FPGAs using the Test Access Port and Boundary-Scan Architecture, commonly referred to as JTAG. During JTAG programming, a .bit file is transferred from the PC to the FPGA using the onboard Digilent USB-JTAG circuitry (port J4) or an external JTAG programmer, such as the Digilent JTAG-HS2, attached to port J5 (located below port JA on the underside of the board). You can perform JTAG programming any time after the Basys 3 has been powered on, regardless of what the mode jumper (JP1) is set to. If the FPGA is already configured, then the existing configuration is overwritten with the bitstream being transmitted over JTAG. Setting the mode jumper to the JTAG setting (seen in Fig 3) is useful to prevent the FPGA from being configured from any other bitstream source until a JTAG programming occurs. Programming the Basys 3 with an uncompressed bitstream using the on-board USB_JTAG circuitry usually takes around five seconds. JTAG programming can be done using the hardware server in Vivado. The



demonstration project available at digilentinc.com provides an in-depth tutorial on how to program your board.

Quad SPI Programming

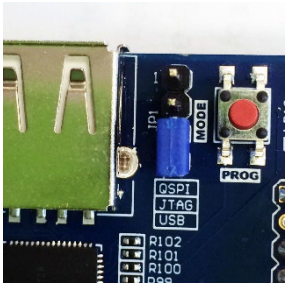
When programming a nonvolatile flash device, a bitstream file is transferred to the flash in a two-step process. First, the FPGA is programmed with a circuit that can program flash devices, and then data is transferred to the flash device via the FPGA circuit (this complexity is hidden from the user by the Xilinx tools). After the flash device has been programmed, it can automatically configure the FPGA at a subsequent power-on or reset event as determined by the mode jumper setting (see Fig 3). Programming files stored in the flash device will remain until they are overwritten, regardless of power-cycle events. Programming the flash can take as long as one or two minutes, which is mostly due to the lengthy erase process inherent to the memory technology. Once written however, FPGA configuration can be very fast– less than a second. Bitstream compression, SPI bus width, and configuration rate are factors controlled by the Xilinx tools that can affect configuration speed. Quad-SPI programming can be performed using Vivado.



USB Host Programming

1. Format the storage device (Pen drive) with a FAT32 file system.
2. Place a single .bit configuration file in the root directory of the storage device.
3. Attach the storage device to the Basys 3.
4. Set the JP1 Programming Mode jumper on the Basys 3 to “USB”.
5. Push the PROG button or power-cycle the Basys 3.

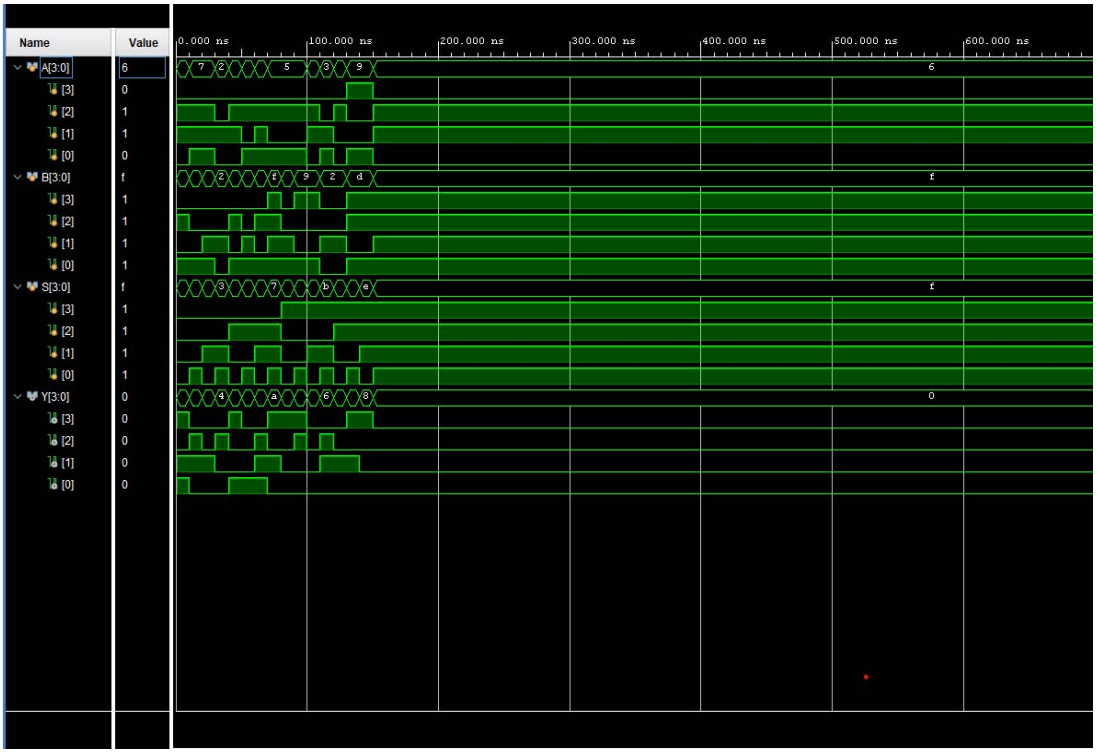
The FPGA will automatically be configured with the .bit file on the selected storage device. Any .bit files that are not built for the proper Artix-7 device will be rejected by the FPGA. The Auxiliary Function Status, or “BUSY” LED (LD16), gives visual feedback on the state of the configuration process when the FPGA is not yet programmed:



When steadily lit the auxiliary microcontroller is either booting up or currently reading the configuration medium (pen drive) and downloading a bitstream to the FPGA. A slow pulse means the microcontroller is waiting for a configuration medium to be plugged in.In case of an error during configuration the LED will blink rapidly.

Chapter 04: Project Development & Testing Aspects

4.1. Test Results



4.2. Interpretation of Results

During the evaluation of the implemented 4-bit ALU, we conducted a series of simulations by providing different input values for each select line ranging from S-0 to S-15. The simulation results revealed that the ALU successfully generated the expected mathematical outputs for the corresponding input combinations.

The comprehensive testing of the ALU using various input values demonstrated its reliability and adherence to the specified functionality. The simulation outcomes validated the design's capability to deliver accurate mathematical computations and logical operations, ensuring the correctness of the ALU's implementation.

Chapter 05: Conclusion & Future Scope

5.1. Conclusion

In conclusion, the project presented a comprehensive study of the design and implementation of a 4-bit Arithmetic Logic Unit (ALU). By providing inputs to A and B for each select line and obtaining the corresponding outputs, it was observed that the ALU effectively performed a wide range of mathematical operations. The successful operation of the ALU demonstrated its capability to handle different 4-bit input values and produce the desired mathematical outputs. The report provided a detailed analysis of the ALU's components, operation principles, and control mechanisms. The implemented ALU proved to be an efficient and reliable component, contributing to the functionality of computer systems.

5.2. Limitations

The 4-bit ALU in this project has a few inherent limitations due to its restricted word size. One major limitation is its limited capacity to process only 4-bit binary numbers, which restricts its applicability to smaller numerical ranges. This limitation poses challenges when handling complex calculations or operations involving larger numbers. Additionally, the 4-bit output size of the ALU may lead to issues like overflow or truncation errors when dealing with computations that require a larger result size. Furthermore, the limited word size affects the precision of the ALU, making it less accurate for operations demanding higher levels of precision. It is important to note that the functionality of the 4-bit ALU may not meet the requirements of more advanced computational tasks or modern computing needs, where larger word sizes and enhanced capabilities are often necessary.

5.3. Further Enhancement and Future Scope

To further enhance the capabilities of the 4-bit ALU, there are several potential areas for improvement and future scope. Firstly, expanding the word size beyond 4 bits, such as to 8 or 16 bits, would increase its range and allow for computations involving larger numbers. This would enable the ALU to handle more complex operations and accommodate a wider range of applications. Additionally, incorporating additional arithmetic and logical operations, such as multiplication, division, and bit shifting, would enhance its functionality and make it more versatile. Optimizing the ALU's performance by implementing advanced adder architectures, such as carry-lookahead or carry-select adders, would improve speed and reduce carry propagation delays. Moreover, integrating mechanisms for detecting and handling overflow conditions would ensure accurate results when dealing with large calculations. Furthermore, enhancing the input selection mechanism using multiplexers and incorporating a more advanced control unit would enable the ALU to support a broader range of operations and increase its versatility in different computing scenarios. These enhancements and future developments would contribute to the continued improvement and advancement of the 4-bit ALU design.

References

Mano, M. Morris. *Digital Design With an Introduction to the Verilog HDL, VHDL, and SystemVerilog* (Sixth Edition). Pearson.

Appendix 01

A01.1. Code Listing

Verilog Module

```
module ALU(
   input [3:0] A,       // 4-bit input A
   input [3:0] B,       // 4-bit input B
   input [3:0] S,       // 4-bit input S for operation selection
   output reg [3:0] Y   // 4-bit output Y
);

   always @(A, B, S) begin
      case (S)
         4'b0000: Y = A + B;                        // Addition operation
         4'b0001: Y = A - B;                        // Subtraction operation
         4'b0010: Y = (B != 4'b0000) ? (A / B) : 4'b0000;  // Division operation
         4'b0011: Y = A * B;                        // Multiplication operation
         4'b0100: Y = ~A;                           // Logical NOT operation
         4'b0101: Y = A & B;                        // Logical AND operation
         4'b0110: Y = A | B;                        // Logical OR operation
         4'b0111: Y = ~(A & B);                     // Logical NAND operation
         4'b1000: Y = ~(A | B);                     // Logical NOR operation
         4'b1001: Y = A ^ B;                        // Logical XOR operation
         4'b1010: Y = ~(A ^ B);                     // Logical XNOR operation
         4'b1011: Y = A << 1;                       // Logical shift left operation
         4'b1100: Y = A >> 1;                       // Logical shift right operation
         4'b1101: Y = A + 1;                        // Increment operation
         4'b1110: Y = A - 1;                        // Decrement operation
         4'b1111: // Equal comparison
            Y = (A == B) ? 4'b0001 : 4'b0000;
      endcase
   end
```

Verilog Module:

```
module ALU_TB;

reg [3:0] A;
reg [3:0] B;
reg [3:0] S;
wire [3:0] Y;

ALU myALU (A, B, S, Y);

initial begin
   A = 4'b0110; B = 4'b0101; S = 4'b0000; #10;
   A = 4'b0111; B = 4'b0001; S = 4'b0001; #10;
   A = 4'b0111; B = 4'b0011; S = 4'b0010; #10;
   A = 4'b0010; B = 4'b0010; S = 4'b0011; #10;
   A = 4'b0110; B = 4'b0101; S = 4'b0100; #10;
   A = 4'b0101; B = 4'b0011; S = 4'b0101; #10;
   A = 4'b0111; B = 4'b0101; S = 4'b0110; #10;
   A = 4'b0101; B = 4'b1111; S = 4'b0111; #10;
   A = 4'b0101; B = 4'b0011; S = 4'b1000; #10;
   A = 4'b0101; B = 4'b1001; S = 4'b1001; #10;
   A = 4'b0110; B = 4'b1001; S = 4'b1010; #10;
   A = 4'b0011; B = 4'b0010; S = 4'b1011; #10;
   A = 4'b0100; B = 4'b0010; S = 4'b1100; #10;
   A = 4'b1001; B = 4'b1101; S = 4'b1101; #10;
   A = 4'b1001; B = 4'b1101; S = 4'b1110; #10;
   A = 4'b0110; B = 4'b1111; S = 4'b1111; #10;
end

endmodule
```

A01.2. Main Code:

```
module ALU(
    input [3:0] A,       // 4-bit input A
    input [3:0] B,       // 4-bit input B
    input [3:0] S,       // 4-bit input S for operation selection
    output reg [3:0] Y   // 4-bit output Y
);

    always @(A, B, S) begin
        case (S)
            4'b0000: Y = A + B;                      // Addition operation
            4'b0001: Y = A - B;                      // Subtraction operation
            4'b0010: Y = (B != 4'b0000) ? (A / B) : 4'b0000;  // Division operation
            4'b0011: Y = A * B;                      // Multiplication operation
            4'b0100: Y = ~A;                         // Logical NOT operation
            4'b0101: Y = A & B;                      // Logical AND operation
            4'b0110: Y = A | B;                      // Logical OR operation
            4'b0111: Y = ~(A & B);                   // Logical NAND operation
            4'b1000: Y = ~(A | B);                   // Logical NOR operation
            4'b1001: Y = A ^ B;                      // Logical XOR operation
            4'b1010: Y = ~(A ^ B);                   // Logical XNOR operation
            4'b1011: Y = A << 1;                     // Logical shift left operation
            4'b1100: Y = A >> 1;                     // Logical shift right operation
            4'b1101: Y = A + 1;                      // Increment operation
            4'b1110: Y = A - 1;                      // Decrement operation
            4'b1111: // Equal comparison
                Y = (A == B) ? 4'b0001 : 4'b0000;
        endcase
    end
```

A01.3. Descriptions of Libraries / Inbuilt function Used :

No Libraries or Inbuilt functions were used in this code