An

Industrial Oriented Mini Project Report

On

An efficient ALU architecture design for low power IOT application

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Under the esteemed guidance of

Dr. T. Gayatri

B. Tech, M. Tech, Ph. D

Associate Professor

Submitted by

K. SAI SHIVANI (23J25A0415)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

JOGINPALLY B.R ENGINEERING COLLEGE

(Affiliated to JNTUH, Hyderabad and Approved by AICTE New Delhi)

(Accredited by NAAC **A+GRADE**, Recognized under section 2(f) of UGC Act, 1956 & ISO 9001:2015 Certified Institution)

Bhaskar Nagar, Yenkapally (V), Moinabad Mandal, R.R.Dist, Hyderabad, Telangana-75

2024-2025

JOGINPALLY B.R. ENGINEERING COLLEGE

Accredited by NAAC with A+ grade, Reconginsed by UGC2(f)Act.1956
(Approved by AICTE & Affiliated to JNTUH, Hyderabad) Yenkapally (V),
Moinabad (M), Ranga Reddy Dist. - 500 075.
Tel: 08413-235684, 235051, Fax: 08413-235125
Website: www.jbrec.edu.in, E-Mail: principal@jbrec.edu.in, principal_jbr@yahoo.com

CERTIFICATE

DATE: 09/06/2025

This is certified that this report entitled **"An efficient ALU architecture design for low power IOT application"** is the report of Industry oriented Mini Project submitted by **K. SAI SHIVANI (23J25A0415),** during **2024-2025** in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Electronics & Communication Engineering of Jawaharlal Nehru Technological University Hyderabad.

Supervisor
Dr. T. Gayatri
B. Tech, M. Tech, Ph. D
Associate Professor

Head Of The Department
Dr. B. Abdul Raheem
B E, M. Tech, Ph. D
Professor & Head
Department of ECE

Project Coordinator
Sri. A. Rajaiah
B E, M. Tech., (Ph. D)
Associate Professor

External Examiner

iGNITE

Embedded Systems

— Spark Towards Your Future —

ID No: IES25INT1217 **02-Jun-2025**

INTERNSHIP COMPLETION CERTIFICATE

To whom it may concern:

This certifies that **Mr./Ms. Kothapalli Sai Shivani,** Roll No. 23J25A0415, a student of 3rd Year B.Tech (ECE) from **Joginpally BR Engineering College, Hyderabad,** has successfully completed the internship program at "IGNITE EMBEDDED SYSTEMS PVT.LTD," Hyderabad-500036. As part of the internship, she was working on **"VLSI"** from **03rd April 2025 to 31st May 2025.**

She has been involved in the development of industrial projects, writing various software codes, and testing hardware components. During the period of her internship program with us, she was found to be punctual, hardworking, and inquisitive.

We wish her every success in her future endeavors.

Project Title : AN EFFICIENT ALU ARCHITECTURE DESIGN FOR LOW POWER IOT APPLICATION

www.igniteembeddedsystems.com

For

1IGNITE EMBEDDED SYSTEMS PVT.LTD

1IGNITE EMBEDDED SYSTEMS PVT.LTD
Flat no. 401, Sree Swathi Manors, Near Aditya Trade Center, Ameerpet, Hyderabad 500016
igniteembeddedsystems@gmail.com www.igniteembeddedsystems.com

ACKNOWLEDGEMENTS

I take this opportunity to remember and acknowledge the cooperation, good will and support both moral and technical extended by several individuals out of which my project has evolved. I shall always cherish my association with them.

I am greatly thankful to **Dr. B.V. RAMANA REDDY** Principal of our college, for extending his valuable help. I shall forever cherish our association with his for his constant encouragement, perennial.

I express my profound gratitude to **Dr. B. ABDUL RAHEEM**, professor, Head of Department of Electronics and Communication Engineering, for his constant support and encouragement in completing our project.

I would also like to thank **Dr. T. Gayatri**, my supervisor, without whose suggestions and encouragement, this project would not have been possible

I express my sincere thanks and gratitude to my project coordinators **Sri. A. RAJAIAH,** Associate professor, **Sri. M. GOVINDU.,** Associate professor, Department of ECE, Joginpally B.R. Engineering College for their valuable help and encouragement throughout the project work.

I am also greatly thankful to all the faculty members of the Department who provided their feedback and valuable suggestions at different stages of the project and helped in the success of the project. With immense gratitude and pleasure I take this opportunity to thank my parents and friends who have been a catalyst in the realization of our project.

PROJECT ASSOCIATE

K. SAI SHIVANI (23J25A0415)

An efficient ALU architecture design for low power IOT application

Abstract

This research proposed an efficient logic design for an Internet of Things (IoT) centric processor architecture. As it is evident that Arithmetic Logic Unit (ALU) is the prominent block in almost all processes and the control panel is used in the IoT board. The proposed ALU architecture uses a combination of clock gates and gray coding technique called CGGC, this minimizes the switching and specific selection of different operations using the architecture level optimization method. The current design was conceived using Verilog code in the Vivado 16 simulation platform. The modified architecture exhibits improved performance in terms of reduced LUTs of 18% leading to low power consumption and optimized resource utilization which is solicited in IoT application.

Keywords: Clock Gating, Gray Coding, ALU, Verilog, Vivado

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

This research proposed an efficient logic design for an Internet of Things (IoT) centric processor architecture. The proposed ALU architecture uses a combination of clock gates and gray coding technique called CGGC, this minimizes the switching and specific selection of different operations using the architecture level optimization method.

With the integration of millions of IoT devices in applications such as smart homes, healthcare, agriculture, and industrial automation, the need for compact, low-power processing units has become more significant than ever. Traditional ALU designs used in general-purpose processors are not optimized for such constraints. Therefore, there is a critical need to develop specialized ALU architectures that prioritize low power consumption without significantly compromising computational speed or accuracy.

This project addresses this need by proposing a novel ALU architecture optimized for low power usage, tailored specifically for IoT applications.

1.1.1 AIM

To design and implement an efficient Arithmetic Logic Unit (ALU) architecture optimized for low power consumption in Internet of Things (IoT) applications, focusing on minimizing energy usage while maintaining computational performance, compact area, and operational reliability suitable for resource-constrained environments.

1.1.2 EXISTING SYSTEM

The Power-Hungry Traditional ALU: A Challenge for IoT Applications

In the realm of digital circuits, the Arithmetic Logic Unit (ALU) reigns supreme as the computational powerhouse, performing the essential arithmetic and logical operations that underpin all digital processing. While traditional ALUs have proven highly effective in general-purpose computing, their design poses a significant challenge when applied to resource-constrained environments like those found in the Internet of Things (IoT).

The challenges posed by power consumption in traditional ALUs highlight the need for innovative solutions. The development of power-efficient ALU architectures, tailored for the specific requirements of resource-constrained IoT applications, becomes crucial for pushing the boundaries of IoT technology.

1.1.3 PROPOSED SYSTEM

The proposed system focuses on designing a custom ALU architecture optimized specifically for low power IoT applications. Key features of the proposed system include:

1.
2. Clock Gating: To reduce unnecessary clock signal propagation and switching activity.
3. Simplified Logic Blocks: Optimized logic circuits to minimize gate count and transitions.
4. Selective Functional Unit Activation: Activate only the required parts of the ALU during operation.
5. Low Power Modes: Design that supports sleep or standby modes for idle operation.
6. Compact Area: Layout optimized for integration into small embedded processors.

The proposed ALU will be implemented using Verilog and simulated/synthesized using industry-standard EDA tools. The architecture will be compared with traditional ALUs in terms of power consumption, area, and delay to demonstrate its suitability for IoT environments.

1.2 OBJECTIVES

1.To design an ALU that consumes minimal power, making it ideal for IoT devices.

2.To implement energy-efficient techniques such as clock gating or operand isolation.

3.To ensure the ALU supports the basic arithmetic and logic operations required for lightweight IoT tasks.

4.To evaluate the performance in terms of power, area, and delay using simulation tools.

5.To propose an optimized architecture that balances power consumption and computational efficiency.

1.3 ORGANIZATION OF THE CHAPTERS

- **Chapter 1:** Presents the introduction of the project and describes about its aim.
- **Chapter 2:** This literature survey explores the evolving landscape of low-power ALU design, highlighting key research efforts and highlighting the significant advancements in reducing power consumption while maintaining computational process.
- **Chapter 3:** Describes the project domain as it is an VLSI project it explains about the software used, here we are using Xilinx VIVADO 2023.1 software
- **Chapter 4:** Explains about the Methodology involved in the project and it consists of code.
- **Chapter 5:** Presents the outcome of the project
- **Chapter 6:** Presents the advantages, disadvantages and applications of proposal
- **Chapter 7:** Has the conclusion and future scope of the work

CHAPTER 2

LITERATURE SURVEY

[1] Ganeswar Sahu et al., 2019 – "Low Power ALU Design and Implementation Using Clock Gating and Carry Select Adder"

Approach**:** Implements clock gating to reduce dynamic power and integrates carry select adder (CSLA) for efficient arithmetic operations.

Conclusion**:** Achieves lower power consumption compared to conventional ALU designs.

Limitation**:** Requires careful optimization to balance power savings and computational speed.

[2] Mohana Durga et al., 2019 – "Design and Implementation of Optimized ALU"

Approach**:** Uses Clock Gating techniques to selectively disable idle functional units in ALU operations.

Conclusion**:** Significant power savings achieved while maintaining processing speed.

Limitation**:** Requires careful timing analysis to prevent performance degradation.

[3] Shinde et al., 2011 – "Clock Gating as a Power Optimization Method for VLSI Circuits"

Approach**:** Selectively disabling the clock to inactive blocks of circuits.

Conclusion**:** Efficiently minimizes power dissipation by reducing unwanted switching activity.

Limitation**:** Can introduce timing complexities in circuit design.

[4] Khadar Bhasha et al., 2016 – "VLSI Design of a Power Efficient ALU Using Clock Gating"

Approach: Applies latch-based clock gating to min

imize clock power and dynamic power consumption ALU.

Conclusion**:** Reduces power consumption significantly while maintaining processing efficiency.

Limitation**:** Increases overall circuit complexity.

CHAPTER-3

DOMAIN SPECIFICATIONS

3.1 INTRODUCTION TO VLSI TECHNOLOGY

An entire range of innovative gadgets and frameworks have revolutionized the way we live thanks to the coordination of circuit (IC) innovation. Without the included circuit, neither semiconductors nor PCs would be as important as they are today. Jack Kill by and Robert Nonce received the 2000 Nobel Prize for Physics for their invention of the coordinated circuit. In comparison to the discrete pieces used to build electronic frameworks before to the 1960s, VLSI systems are substantially smaller and consume less power. Resolution enables us to build systems with many more transistors, allowing us to link far more computing power to solving a problem. Coordination circuits, on the other hand, are easier to plan and construct, and more reliable than discrete frameworks; this makes it possible to create distinct reason frameworks that are more efficient than PCs for the primary task.

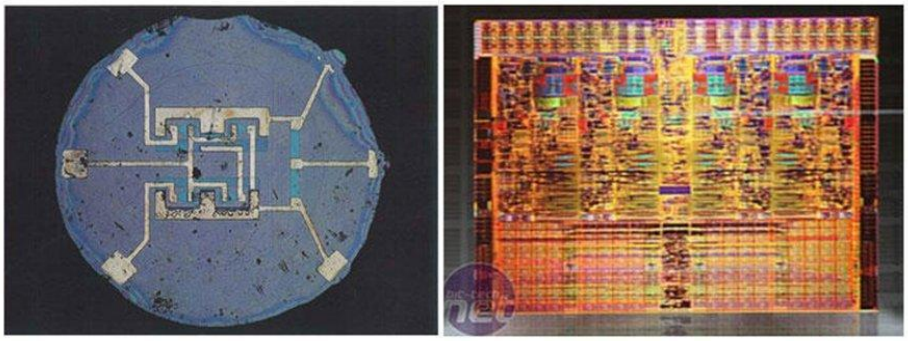

Fig:3.1 A comparison: first planar IC(1961) and Intel nehalem quad core die

3.2 VLSI

Very-large-scale integration (VLSI) is the process of creating an integrated circuit (IC) by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. Before the introduction of VLSI technology most ICs had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets IC designers add all of these into one chip.

3.3 HISTORY

In the early days of simple integrated circuits, the technology's large scale limited each chip to only a few transistors, and the low degree of integration meant the design process was relatively simple. Manufacturing yields were also quite low by today's standards. As the technology progressed, millions, then billions of transistors could be placed on one chip, and good designs required thorough planning, giving rise to new design methods.

3.3.1 SSI, MSI, LSI & VLSI

The first integrated circuits contained only a few transistors. Called "small-scale integration" (SSI), digital circuits containing transistors numbering in the tens provided a few logic gates for example, while early linear ICs such as the Plessey SL201 or the Philips TAA320 had as few as two transistors. The term Large Scale Integration was first used by IBM scientist Rolf Landauer when describing the theoretical concept from there came the terms for SSI, MSI, VLSI, and ULSI.

The next step in the development of integrated circuits, taken in the late 1960s, introduced devices which contained hundreds of transistors on each chip, called "medium-scale integration" (MSI).

Further development, driven by the same economic factors, led to "large-scale integration" (LSI) in the mid-1970s, with tens of thousands of transistors per chip. Integrated circuits such as 1K-bit RAMs, calculator chips, and the first microprocessors, that began to be manufactured in moderate quantities in the early 1970s, had under 4000 transistors. True LSI circuits, approaching 10,000 transistors, began to be produced around 1974, for computer main memories and second-generation microprocessors.

The final step in the development process, starting in the 1980s and continuing through the present, was "very large-scale integration" (VLSI). The development started with hundreds of thousands of transistors in the early 1980s, and continues beyond several billion transistors as of 2009.

The more energy efficient CMOS replaced NMOS and PMOS, avoiding a prohibitive increase in power consumption. In 1986 the first one-megabit RAM chips were introduced, containing more than one million transistors. Microprocessor chips passed the million-

transistor mark in 1989 and the billion-transistor mark in 2005. The trend continues largely unabated, with chips introduced in 2007 containing tens of billions of memory transistors.

3.3.2 ULSI, WSI, SOC, 3D-IC

To reflect further growth of the complexity, the term ULSI that stands for "ultra-large-scale integration" was proposed for chips of more than 1 million transistors.

Wafer-scale integration (WSI) is a means of building very large integrated circuits that uses an entire silicon wafer to produce a single "super-chip". Through a combination of large size and reduced packaging, WSI could lead to dramatically reduced costs for some systems, notably massively parallel supercomputers. The name is taken from the term Very-Large-Scale Integration, the current state of the art when WSI was being developed.

A system-on-a-chip (SoC or SOC) is an integrated circuit in which all the components needed for a computer or other system are included on a single chip. The design of such a device can be complex and costly, and building disparate components on a single piece of silicon may compromise the efficiency of some elements. However, these drawbacks are offset by lower manufacturing and assembly costs and by a greatly reduced power budget: because signals among the components are kept on-die, much less power is required (see Packaging).

A three-dimensional integrated circuit (3D-IC) has two or more layers of active electronic components that are integrated both vertically and horizontally into a single circuit. Communication between layers uses on-die signaling, so power consumption is much lower than in equivalent separate circuits. Judicious use of short vertical wires can substantially reduce overall wire length for faster operation.

3.4 Hardware Description Languages, or HDL

Hardware Description Languages, or HDLs, are languages used to design hardware with. As the name implies, an HDL can also be used to describe the functionality of hardware as well as its implementation.

The Verilog Hardware Description Language, usually just called Verilog, was designed and first implemented by Phil Moorby at Gateway Design Automation in 1984 and 1985. It was first used beginning in 1985 and was extended substantially through 1987. The implementation was the Verilog-XL simulator sold by Gateway.

3.4.1 Modelling Structures

Verilog models are made up of modules. Modules, in turn, are made of different types of components. These include

- Parameters
- Nets
- Registers
- Primitives and Instances
- Continuous Assignments
- Procedural Blocks
- Task/Function definitions

A module may contain any number, including zero, of these components. There is no required order among the different module components. However, net and register declarations must be appear before the net or register is used. However, other components can appear anywhere in the module.

CHAPTER-4

METHODOLOGY IMPLEMENTATION

4.1 Hardware Description

The Efficient ALU Architecture for Low Power IoT Applications is designed using VLSI techniques to optimize power efficiency, area utilization, and computational speed. The hardware implementation is structured into multiple modules, ensuring minimal energy consumption while maintaining high performance. The ALU integrates clock gating, operand isolation, and voltage scaling to reduce power dissipation. It consists of arithmetic, logic, and control units, all efficiently implemented using VHDL and synthesized on an FPGA.

4.1.1 Block Diagram of the CGGC ALU

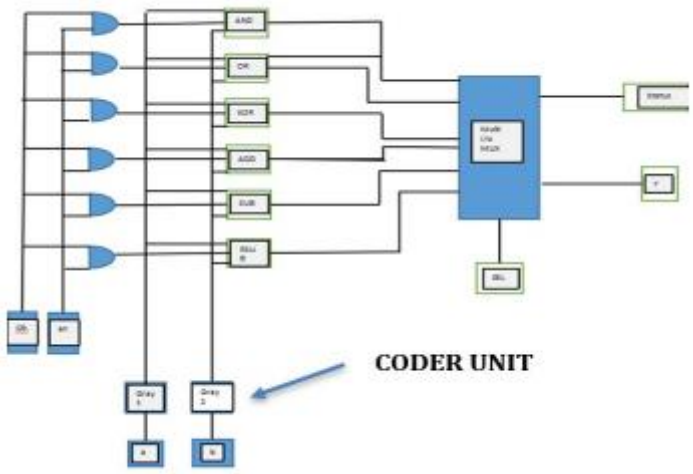

Figure 4.1 Block Diagram of the Modified ALU (CGGC ALU)

4.1.2 Overview of Project

This project focuses on designing an Arithmetic Logic Unit (ALU) optimized for low-power consumption in IoT applications. Since ALUs are fundamental components in processors, reducing their power usage significantly impacts the overall efficiency of IoT devices. The architecture employs Clock Gating and Gray Coding (CGGC) to minimize

switching activity, reducing dynamic power consumption. The design reduces Look-Up Tables (LUTs) by 18%, leading to lower power usage and improved efficiency. The ALU is developed using Verilog HDL and simulated in Vivado 16, ensuring compatibility with FPGA-based IoT systems. Techniques such as supply voltage scaling, switched capacitance minimization, and leakage power reduction are integrated to enhance energy efficiency.

4.1.3 Individual blocks

4.1.3.1 Arithmetic Unit

The Arithmetic module designed in this work, makes use of 4 components that are, Adder, Subtractor, and Multiplier. As a subtraction, and multiplication on n- bit data. The arithmetic unit uses conventional adder and subtractor, while the multiplier unit are made using Vedic Mathematics Algorithm. The control signals which guide the Arithmetic unit to perform a particular operation, i.e. Addition, subtraction, multiplication operation are s0 and s1, which are provided by the control circuit. The status of control lines s0 and s1 and the corresponding arithmetic operation being performed.

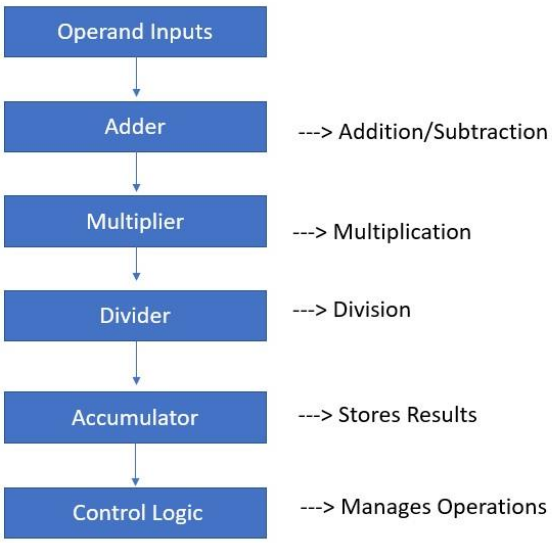

Figure 4.2: Arithmetic Unit

S1	S0	Operations
0	0	addition
0	1	subtraction
1	0	multiplicatio n
1	1	others

Figure 4.3: Truth table of Arithmetic Unit

4.1.3.2 Logic Unit

The basic building blocks of a simple logic unit are NOT gate, AND gate, OR gate, XOR gate and a multiplexer. The simple logic unit contain four basic logic gates to provide four basic logic operations like NOT (complement), AND, OR and XOR. To select one of four logic operations a selector is required. In this unit a simple 4 to 1 multiplexer is preferred for designing of logic unit. To select one of 4 inputs, there are two select inputs S1 and S0. The function table assumes that both the data inputs A, B are present. The values of select lines decide the nature of operation to be performed on the operands.

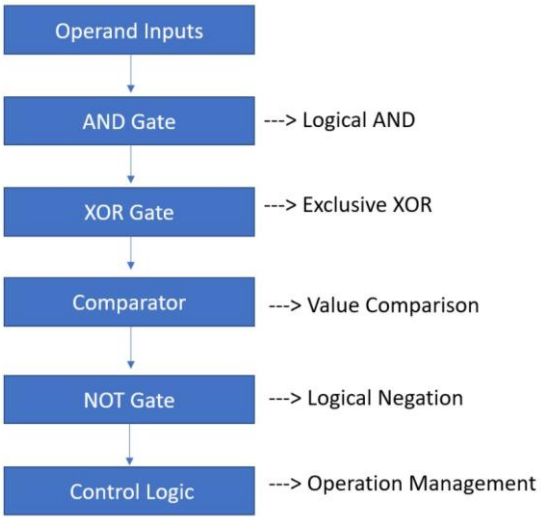

Figure 4.4: Logic Unit

S1	S0	Output	Operation
0	0	Y=A'	Complement
0	1	Y=A . B	AND
1	0	Y=A + B	OR
1	1	Y= A⊕ B	XOR

Figure 4.5: Truth table of Logic Unit

4.1.3.3 Control Unit

Uses a 3-bit ALU selection signal to select the desired arithmetic/logical operation. Manages power-saving clock gating and operand isolation dynamically. Controls data flow between registers, arithmetic, and logic units.

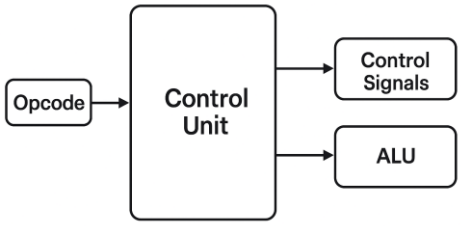

Figure 4.6: Control Unit

4.1.3.4 Clock Gating Control Block

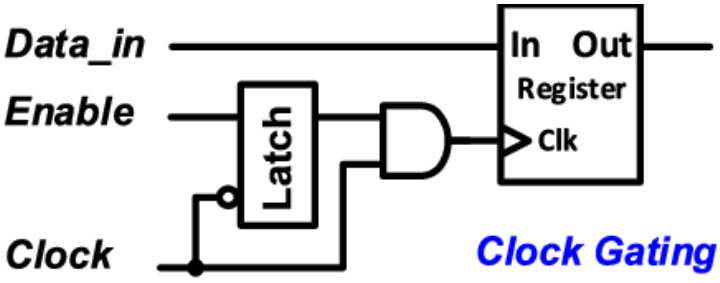

Figure 4.7: Clock Gating Control Block

Implements clock gating to disable inactive sections of the ALU dynamically. Reduces switching power by turning off unused modules. Ensures energy-efficient operations, making it ideal for IoT devices. This technique selectively disables unused functional units by controlling the clock signal, effectively turning off power to inactive components. Imagine turning off lights in a room you're not using to save energy.

4.1.3.5 Gray Code Encoder:

This coding scheme minimizes bit transitions between consecutive values, reducing the switching activity that contributes to power consumption. Think of it as using a more efficient route to avoid unnecessary turns and detours.

4.2 Software Description

4.2.1 Introduction to Xilinx VIVADO 2023.1 software

ISE® Virtual Machine (VM) for Windows 10 extends the ISE Design Suite 14.7 products for the latest Microsoft Windows offering. This solution is enabled for all Xilinx ISE 14.7 devices. Xilinx devices are cost-optimized FPGAs, offering industry leading connectivity features such as high logic-to-pin ratios, small form-factor packaging, and a diverse number of supported I/O protocols. Built on 45 nano meter technology, the devices are ideally suited for a range of advanced bridging applications found in automotive infotainment, consumer, and industrial automation.

ISE VM for Windows 10 executes on a virtualized environment. The ISE tools execute on an Oracle Linux Virtual Machine.

4.2.2 Supported Flows

The majority of the standard flows supported with ISE® 14.7 tools are supported with this solution as well. Following are the list of exceptions.

• Design entry is supported via Project Navigator only. The Plan Ahead™ tool as a design entry environment is not supported in this release.

• Symplify Model Sim and Mentor Graphics Questa Advanced Simulator integration are not supported.

• Smart Xplorer and System Generator are not supported.

• All ISE 14.7 devices are supported.

4.2.3 Operating Systems

This solution supports Windows 10 Professional and Windows 10 Enterprise.

1. System Requirements

Because this solution is enabled by virtualization, the processor of the Windows 10 machine must support virtualization technology.

This technology is called VT-x for Intel processors and AMD-V for AMD CPUs.

In addition to the CPU requirement, this virtualization technology also needs to be enabled in the BIOS setup. Note that in some cases this technology is not enabled by default and must be enabled by updating the BIOS settings.

Refer to your Windows 10 machine vendor documentation for instructions to access BIOS.

The ISE® Virtual Machine is deployed using Oracle VirtualBox hypervisor. Because only one hypervisor can be enabled on a system, you must disable/uninstall any other hypervisor, such as Microsoft Hyper-V, prior installation.

In order to run properly this solution has the following minimum hardware requirements:

- CPU: minimum of 2 cores
- RAM: 8 GB
- Disk Space: 85 GB

Xilinx recommends a minimum screen resolution of 1280 x 1024. Some GUI elements do not display properly at lower resolutions.

2. Architectures

The solution supports all Xilinx devices.

3. Compatible Third-Party Tools

All parts (ISE 14.7 VM for Win 10) do not provide support for any integrated third-party tools.

Xilinx supports importing of EDIF files generated using any supported version of SynplifyPro. This can be from Windows 10 or any RHEL Linux environment. In ISE, the project setting is Design Property >Top-Level Source type = EDIF.

4.2.4 Installation

Installing the ISE Design Tools for Designs

This explains the installation process for ISE® 14.7 tools for devices on Windows 10.

4.2.4.1 Installation Preparation

Before starting installation the follow steps must be completed.

I. Make sure your system meets the requirements described in, Architecture Support and Requirements.

II. Disable anti-virus software to reduce installation time

III. To install the ISE Virtual Machine First you will need to install the Oracle VM.

4.2.4.2 Installation Steps

Decompress the downloaded installation zip file, and run **xsetup.exe** to launch the installation.

Note: Oracle Virtual Box needs to be installed prior to the ISE VM installation.

Step 1: Oracle Virtual Box Installation

The version of Oracle Virtual Box officially supported by Xilinx is 5.2.34. See the Oracle Documentation to install Virtual Box and for more information.

Step 2: License Agreements

Carefully read the license agreements before continuing with the installation. If you do not agree to the term and conditions, cancel the installation and contact Xilinx.

Step 3: Installation Options

Unlike in a typical ISE 14.7 tools installation process, you are not provided with any options to select from. This is mostly the case because options have already customized for this solution.

• ISE Design Suite System Edition is installed. All devices are supported.

• ISE tools for all devices are already licensed.

• Cable drivers are already pre-installed.

Step 4: Shortcuts

You can customize the creation of desktop and program group shortcuts. Xilinx highly recommends installing shortcuts; these shortcuts greatly facilitate the access to ISE tools from the Windows 10 machine. By default, ISE tools shortcuts are provided on the Virtual Machine.

Step 5: Shared Folder

In order to facilitate the sharing of files between the Windows 10 host machine and the Virtual Machine running ISE tools, you are provided with an installation option to specify a shared folder. See the Oracle Virtual Box documentation. 4.3 Shared folders [Ref 3].

IMPORTANT: Keep all user data files, such as project source files, in the shared folder. That allows them to be accessible from both operating systems and will not be lost if ISE is uninstalled.

The directory that you specify will be available on the virtual machine under /home/ise/.

For instance, if you specify the Windows 10 host directory c:\xilinx_projects\test_design or all_ise_designs at installation time, its content will be accessible on the ISE Virtual Machine under /home/ise/test_design or all_ise_designs.

The installation process only allows you to specify one shared folder, but it is possible to configure additional shared folders post installation. See Chapter 6, ISE Virtual Machine Configuration for more details.

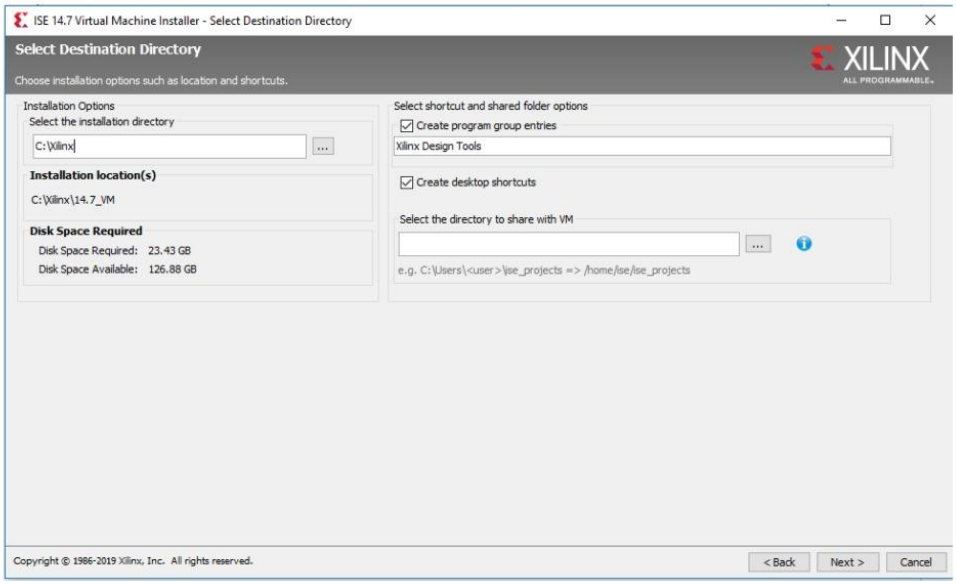

Figure 4.8: Installer Select Destination Directory Window

Step 6: Xilinx End-User license Agreement (EULA)

See the Xilinx End-User License Agreement (EULA) document on the ISE VM for Windows 10 lounge web page.

References

- ISE Product Page: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav / vivado-design-tools/archive-ise.html

Step 7: PDIR is For Project Folders

Take a moment to consider the storage media where you will work your projects. ISE/Webpack makes intensive use of storage media and the files are somewhat large so don't consider using a floppy disk.

• If you are using your own computer then the C: drive or other attached hard drive is a good choice.

• For a computer lab ask if there is a networked drive available. The advantages of a networked drive are that it can be accessed from any lab computer and it's likely going to be a RAID system which means it's more reliable than any one disk drive.

• You can perform the tutorials with a Flash memory stick but we discourage you from using one to work Significant projects as the write time is slow. A better choice is an external USB hard drive, such devices are available for less than $50 from a number of vendors. You can use a Flash memory stick to archive and transport a project.

Step 8: Design Description

The tutorial design is a demonstration of the Dynamic Reconfiguration feature of the Virtex®-5 Digital Clock Manager (DCM). Using the Virtex-5 DCM, the design generates an output clock using the following relationship: Output Clock = Input Clock * (Multiplier / Divider) Using the Dynamic Reconfiguration Ports (DRP) in the DCM, the design lets you re-define the Multiplier and Divider parameters to generate different output frequencies. See the “Clock Resources” section of the Virtex-5 FPGA Users Guide, (UG190). Appendix A, Additional Resources, cites a link to the document.

Step 9: Functional Blocks

The following functional blocks are in the tutorial design.

1.drp_dcm (drp_dcm.vhd)

The drp_dcm.vhd is a Virtex-5 DCM macro with internal feedback, frequency controlled output, duty-cycle correction, and dynamic reconfiguration ability.

The CLKFX_OUT output provides a clock that is defined by the following relationship:

CLKFX_OUT = CLKIN_IN * (Multiplier/Divider)

For example, using a 100 MHz input clock, setting the multiplier factor to 6, and the divider factor to 5, produces a 120 MHz CLKFX_OUT output clock.

Using the DRP ports of the DCM, the Multiplier (M) and Divider (D) parameters can be dynamically redefined to produce different CLKFX_OUT frequencies. This tutorial shows how the Multiply and Divide parameters are provided to the DCM in hexidecimal format using the 16-bit wide DI_IN port:

DI_IN[15:8] = M – 1

DI_IN[7:0] = D – 1

For example, for an M/D factor of 6/5, DI_IN = 0504h.

2.drp_stmach (drp_stmach.vhd)

The drp_stmach.vhd module describes a Dynamic Reconfiguration Port (DRP) controller. The DRP controller asserts and monitors the DCM DRP signals to perform a dynamic reconfiguration cycle.

A dynamic reconfiguration cycle is started by asserting the drp_start signal. Following this step, the DRP controller asserts the appropriate DCM DRP pins to complete a full cycle.

The drp_done signal indicates successful cycle completion.

3.drp_demo (drp_demo.vhd)

This is the top module of the tutorial design that connects the DCM macro and the DRP controller modules to the external I/O ports.

4.drp_demo_tb (drp_demo_tb.vhd)

The drp_demo_tb.vhd is a self-checking HDL test bench, which is described in the following subsection.

Step 10: Design Self-Checking Test bench

To test the functionality of this design, the zip file provides a self-checking test bench. (Refer to source file drp_demo_tb.vhd in the /sources folder.) The self-checking test bench contains a validation routine that compares sampled values from the simulation against expected results. The self-checking test bench provided for this design performs the following functions.

• Generates a 100 MHz input clock for the design system clock (clk_in).

• Performs four different tests to dynamically change the output frequency of the design. In each test, a DRP cycle is started (using the drp_start signal) to set the output clock to a different frequency. Below table shows the expected output frequency and Multiplier/Divider parameters used for each test.

Test	Freq. (MHz)	Period (ps)	Multiplier (M)	Divider (D)
1	75	13,332	3	4
2	120	8,332	6	5
3	250	4,000	5	2
4	400	2,500	4	1

Figure 4.9: Expected Output Frequency and Multiplier/Divider Parameter

• In each test, the test bench compares the expected clock period and the clock period that is measured during simulation. Based upon the comparison results, the testbench writes messages to the simulator indicating success or failure. Two of the source files used by the test bench deliberately contain errors so you can exercise the debugging features in the ISim tool later in the tutorial.

• Upon completion of the simulation, a summary report provides a list of tests, both passed or failed.

Step 11: Compile the Design

The ISim integrated flow lets you perform behavioral and timing simulations of your design in either the ISE Project Navigator tool.

In this tutorial flow, you create an ISE project for the tutorial design first. You then set behavioral simulation properties, and launch the ISim simulator to perform a behavioral simulation of the design.

4.2.5 Create a Project in ISE Project Navigator

Use the New Project Wizard in ISE Project Navigator to create an ISE project for the tutorial design.

Note: Be sure that you have read the Installing Tutorial Design Files, page 7 to obtain the files required for this design.

Step 1: Launch Project Navigator and Use New Project Wizard

To launch the Project Navigator tool and create an ISE project.

1. Launch ISE Project Navigator.

2. Select File > New Project to launch the New Project Wizard.

3. In the Create New Project dialog box, provide a name (for example, Isim_Tutorial) and an appropriate location for the project.

4. Click Next.

5. In the Project Settings dialog box, select the device and project properties. Change the settings to match the settings shown in Figure 4-1, , and click Next to continue.

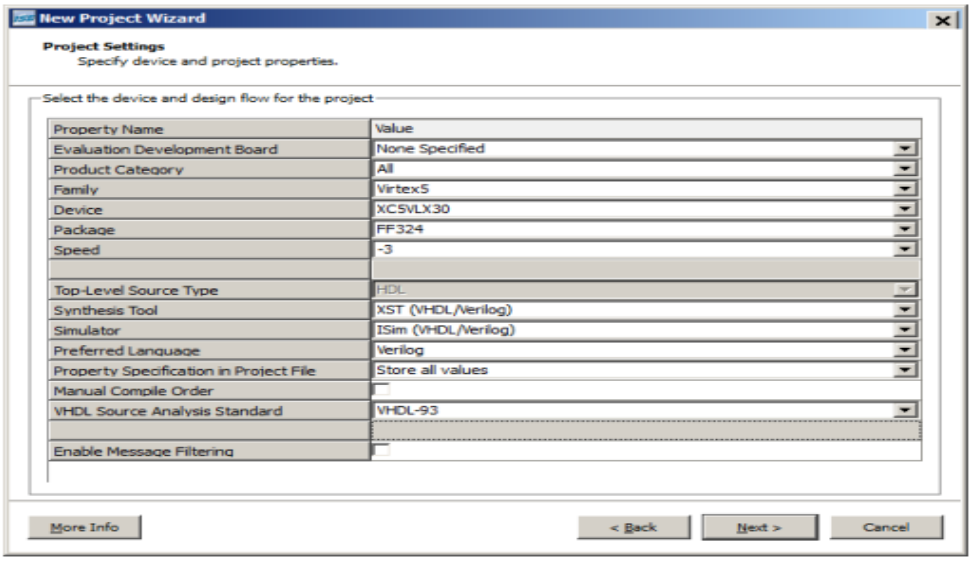

Figure 4.10: New Project Wizard: Project Settings

6. Review the Project Summary page and click Finish.

Step 2: Add Tutorial Source Files to the Project

1. Select Project > Add Source to add a source file.

2. Navigate to the location where you saved your source files in Installing Tutorial Design Files,

3. In the /sources subfolder, select and open all the files.

4. In the Adding Source Files dialog box, ensure that the association and libraries are properly specified for the tutorial sources. Compare your settings with those in Figure 5.3.

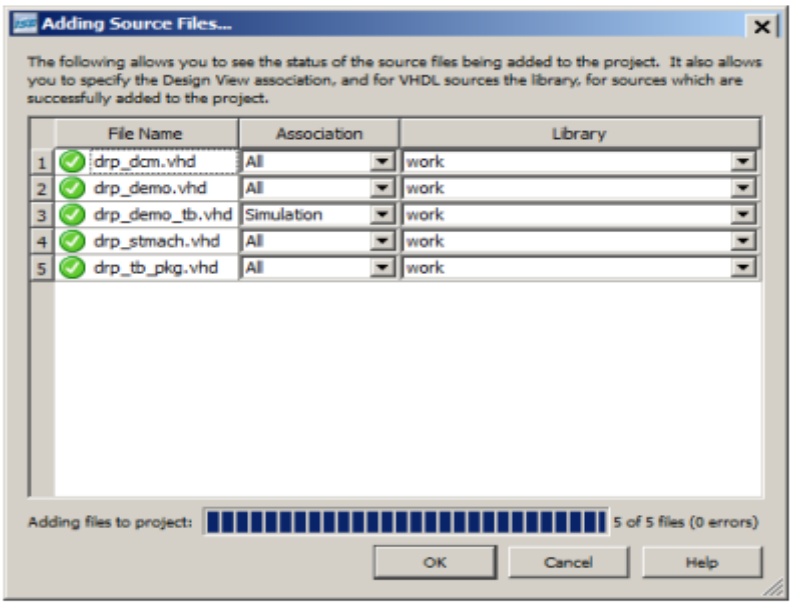

Figure 4.11: Adding Sources Files Dialog Box: Status of Source Files and Associations

5. Click OK. Project Navigator adds the source files to the project. Your Project Navigator main window should now look like Figure 5.4

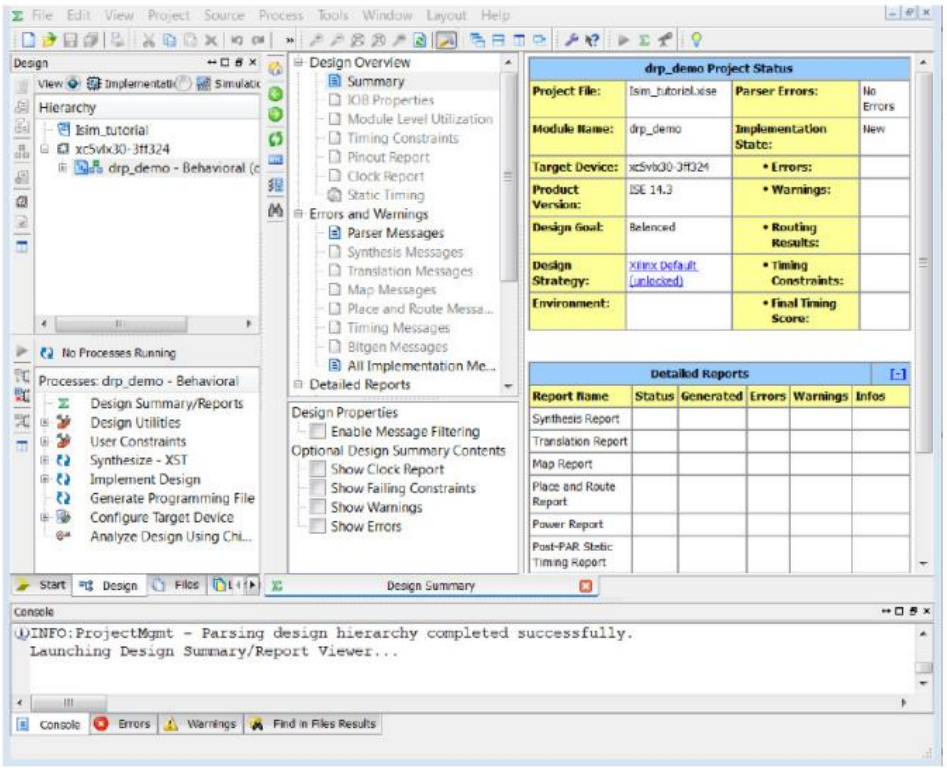

Figure 4.12: ISE Project Navigator Design Summary

Step 3: Create a VHDL Library

Next, you create a user VHDL library for a VHDL package (drp_tb_pkg.vhd) that is used by the design test bench. The VHDL package contains VHDL functions used by the test bench to perform verification routines. After you create the VHDL library, you move the VHDL package file from the /work library to the newly-created VHDL library.

To create a VHDL library.

1. In Project Navigator, select Project > New Source. The New Source Wizard opens.

2. Select VHDL Library as the source type.

3. Type drp_tb_lib for the VHDL library name. (Refer to Figure 4-4). Note: Leave the Add to project check box selected.

4. Click Next.

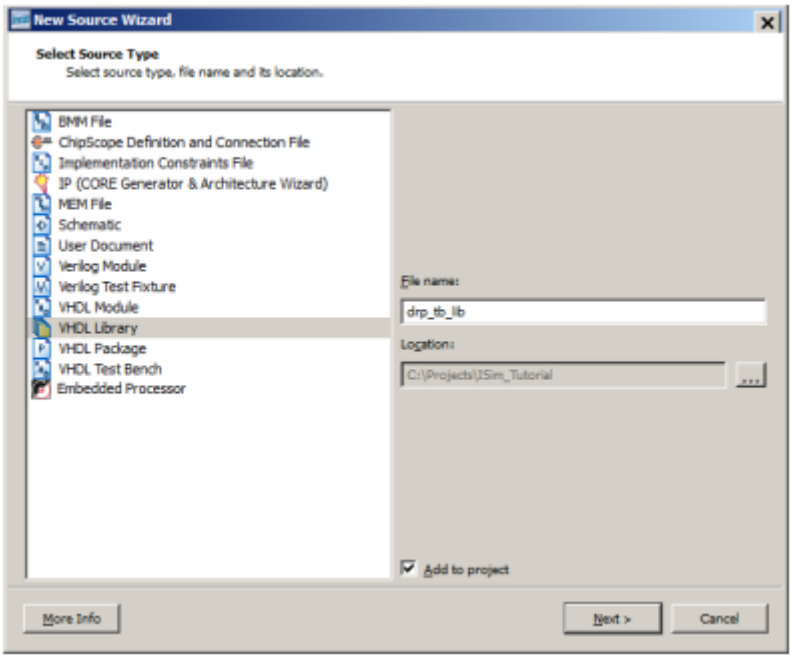

Figure 4.13: Select Source Type

5. In the Summary dialog box, click Finish.

Step 4: Move VHDL Files to a Library

Move the VHDL package file to the drp_tb_lib library:

1. Click the Libraries tab to switch to the libraries.

2. Expand the /work library (see Figure 5.6.)

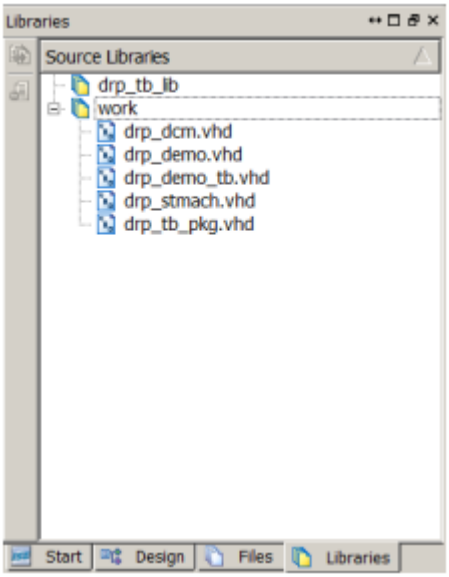

Figure 4.14: Libraries Pane

3. Right-click the drp_tb_pkg.vhd file, and select Move to Library.

4. In the Move to Library dialog box, select drp_tb_lib as the library into which to move the drp_tb_pkg.vhd file. Alternatively, you can drag and drop the files into the library. Recall that the fuse command can automatically invoke vlog comp and vhpcomp for each VHDL or Verilog source code in a project file (.prj), so you can compile sources “on-the-fly”.

5. Click OK. Observe that the drp_tb_lib library contains the VHDL drp_tb_pkg.vhd package file. (See Figure 5.7.)

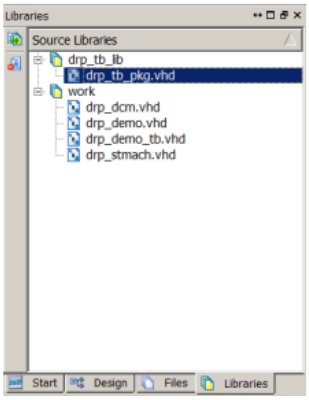

Figure 4.15: Source Libraries

Step 5: Setup and Launch Behavioral Simulation

Now that you have created an ISE project for the tutorial design, you can set up and launch an ISim behavioral simulation.

Set Behavioral Simulation Properties

To set behavioral simulation properties in ISE:

1. In the View pane of the Design pane, select the Simulation radio button at the top. The Simulation type drop-down list opens.

2. Select Behavioral from the drop-down list.

3. Select the drp_demo_tb test bench file. The simulation processes available for the design display in the Processes pane. (See Figure 5.8).

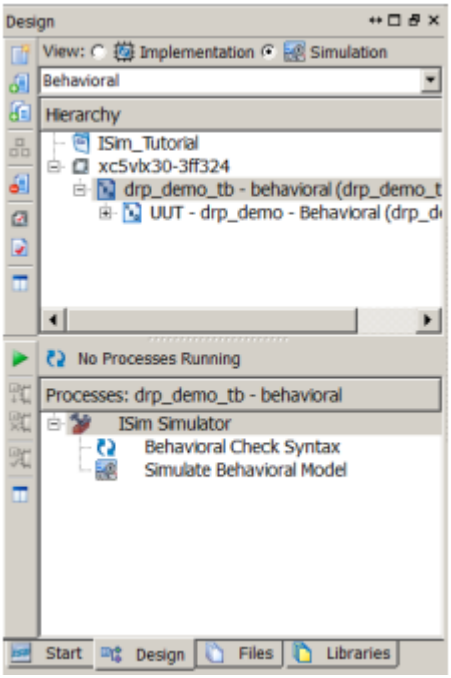

Figure 4.16: Behavioral Simulation Processes

4. In the Processes pane, expand the ISim Simulator tree and right-click Simulate Behavioral Model under the ISim Simulator process and select Process Properties. The ISim Properties dialog box opens (see Figure 5.9).

In this dialog box you can set different simulation properties with which to launch the simulation, such as:

• Simulation runtime

• Waveform database file location

• User-defined simulation command file

5. Uncheck the Run for Specified Time property, and click OK.

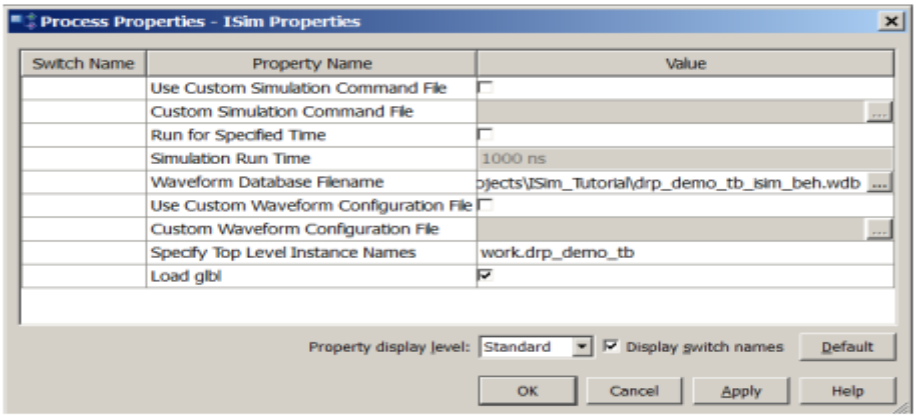

Figure 4.17: Process Properties: ISim Properties Dialog Box

Launch Behavioral Simulation

You can now launch ISim to perform a behavioral simulation of the tutorial design. In the Processes Panel, double-click Simulate Behavioral Model.

Step 6: Result

The ISim GUI (Figure 5.10) opens after successfully parsing and compiling the design.

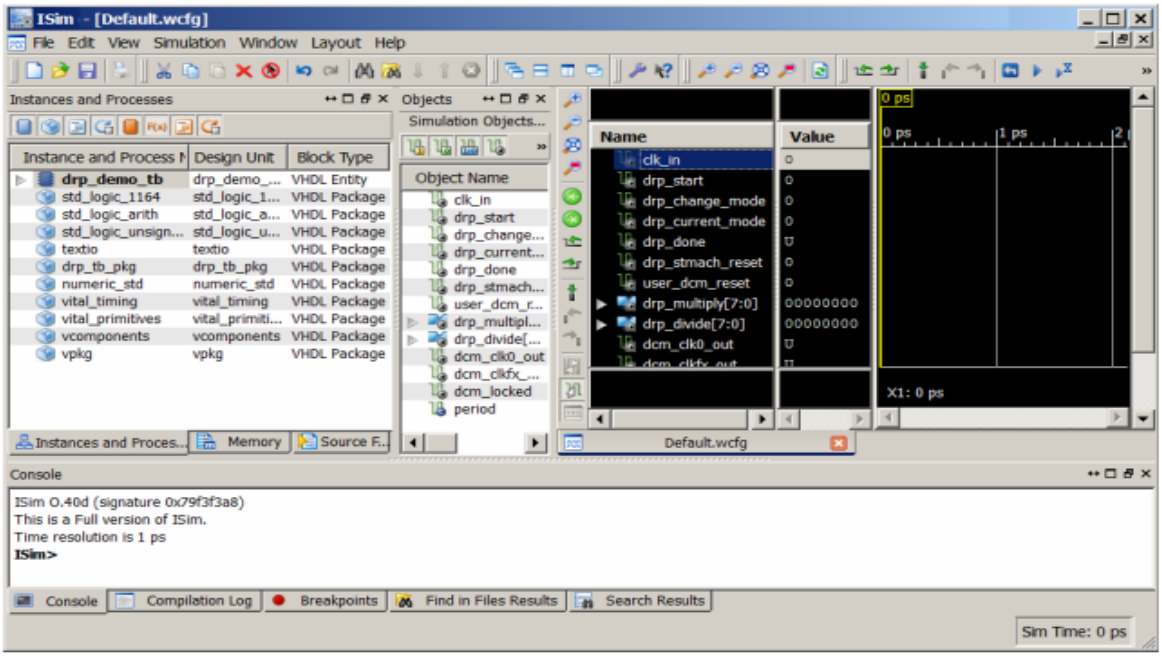

Figure 4.18: ISim GUI

4.2.6 Flow chart

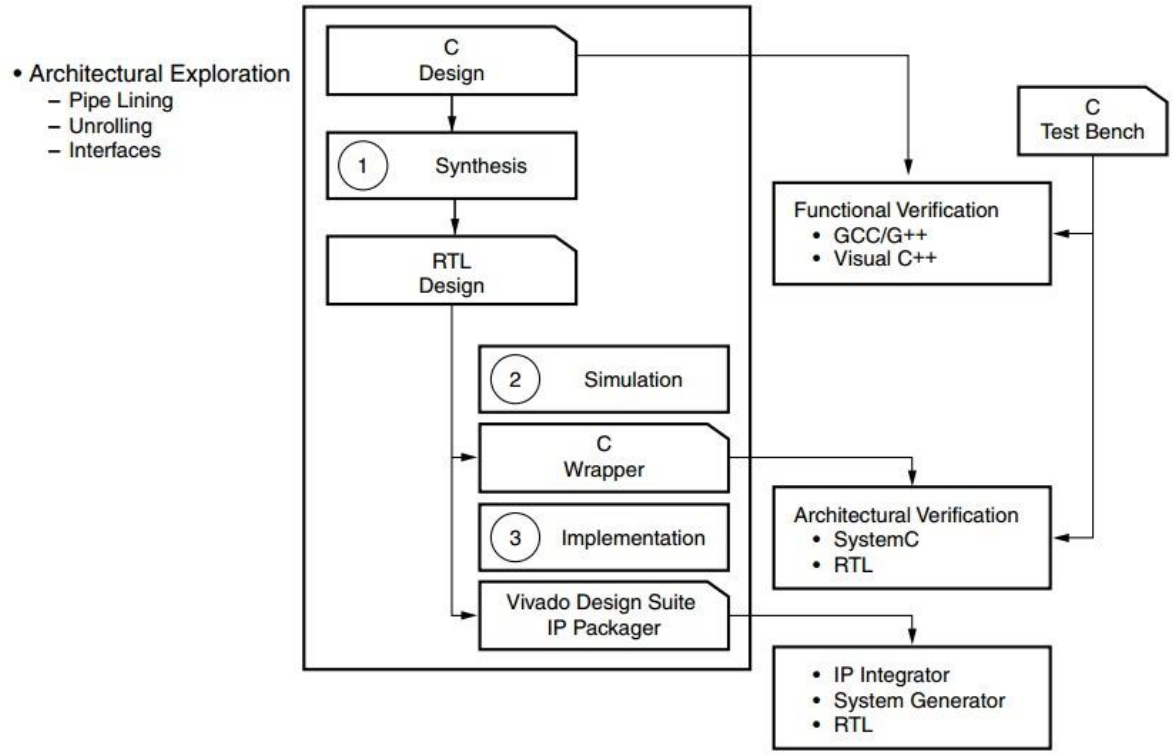

Figure 4.19 Flow Chart

CHAPTER 5

RESULT ANALYSIS

5.1 Behavioral Simulation

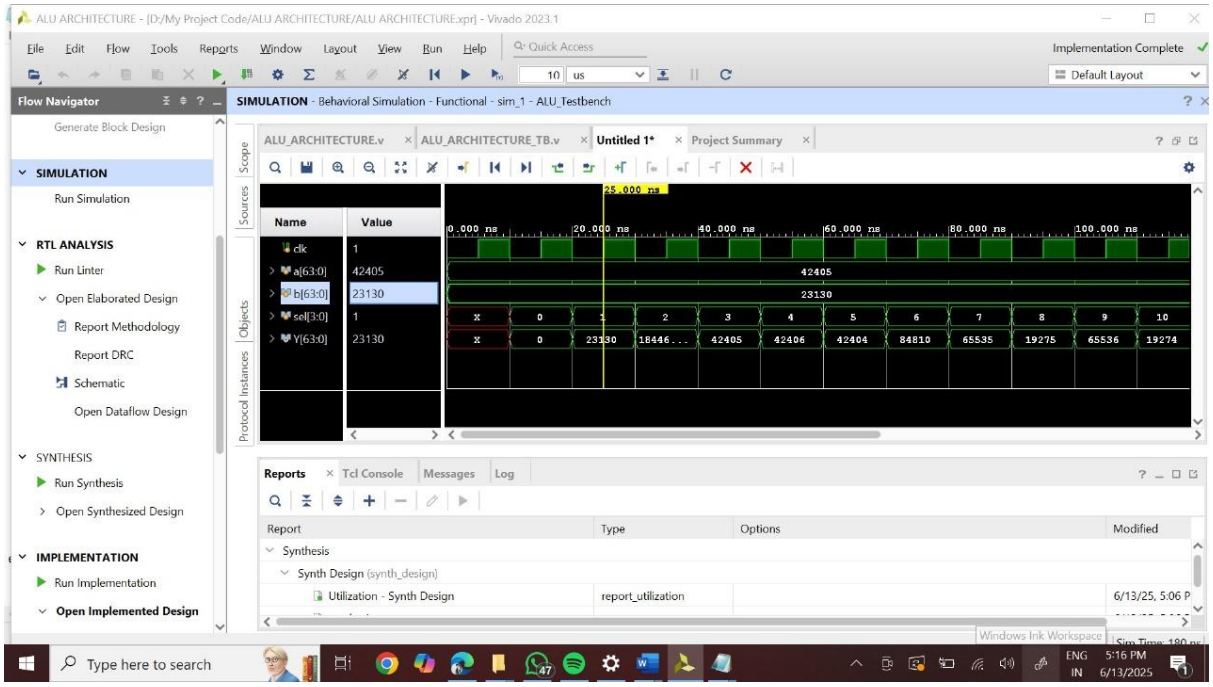

Fig 5.1 Behavioral Simulation

Inputs:

- clk: Clock signal
- a[63:0]: Operand A
- b[63:0]: Operand B
- sel[3:0]: Operation selector

Output:

Y[63:0]: Computed result based on selected ALU operation

Observations:

- Correct operation switching based on sel values.
- No visible glitches; smooth waveform transitions.
- Latency analysis required for optimal timing constraints.

5.2 Schematic Diagram:

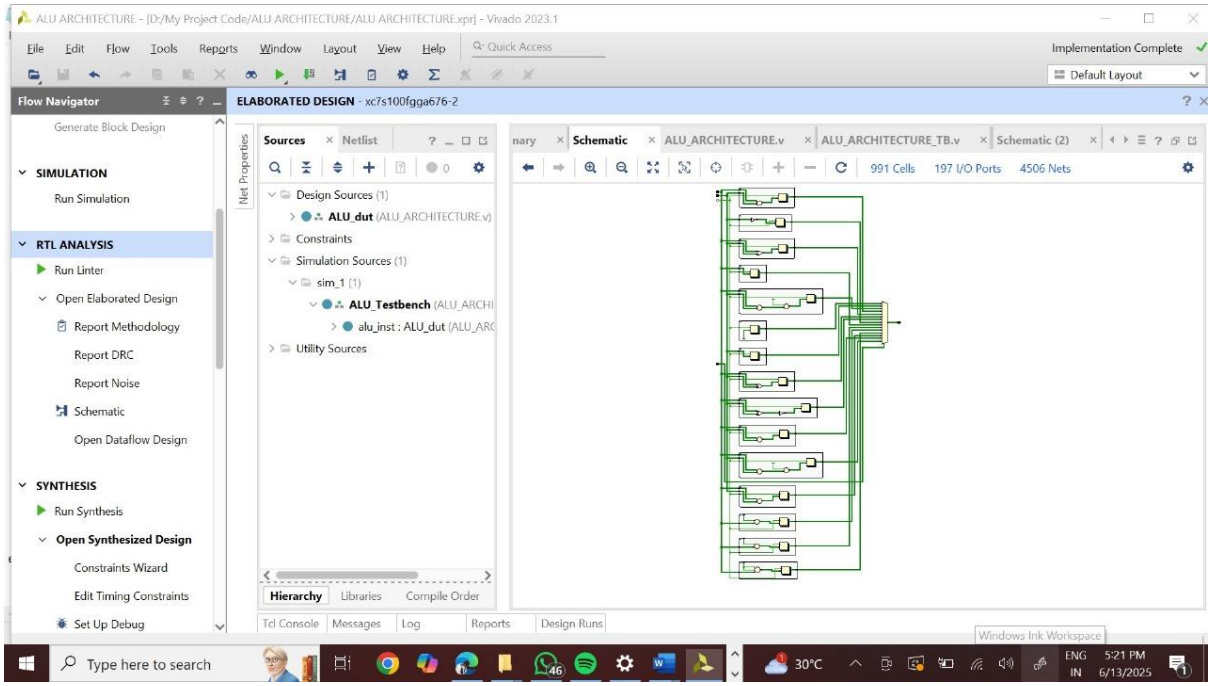

Fig 5.2 ALU Architecture RTL Diagram

1. **Flow Navigator (Left Panel)**

Contains options for simulation, synthesis, RTL analysis, and design constraints. Steps such as Run Simulation, Open Elaborated Design, Run Synthesis are listed for FPGA-based ALU implementation.

- **Sources Panel (Middle Section)**

Lists the ALU architecture source files, including design sources, constraints, and simulation sources. The ALU test bench is visible, indicating that the design is undergoing functional verification through simulation.

- **Schematic Panel (Right Side)**

Displays a detailed schematic diagram of the ALU, showing interconnections among logic gates and components. Key stats include 991 Cells, 197 I/O Ports, and 4506 Nets, indicating a complex logic circuit design.

5.3 Synthesis Design:

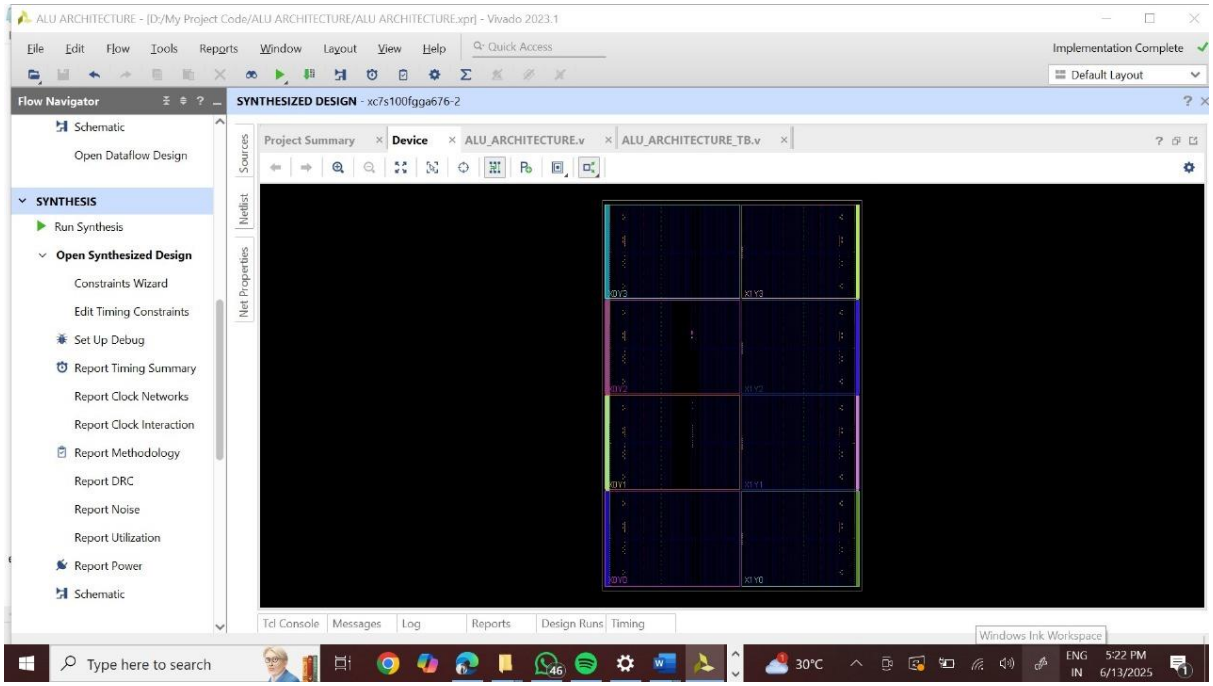

Fig:5.3 Synthesis Design

- Synthesized ALU Design mapped across FPGA logic cells.
- Grid Mapping (X0Y0, X0Y1, etc.) showing logic placements.

Utilization Summary:

- Well-distributed placement with reduced congestion.
- Timing constraints need optimization for smoother execution.

5.4 Area and Power Utilization:

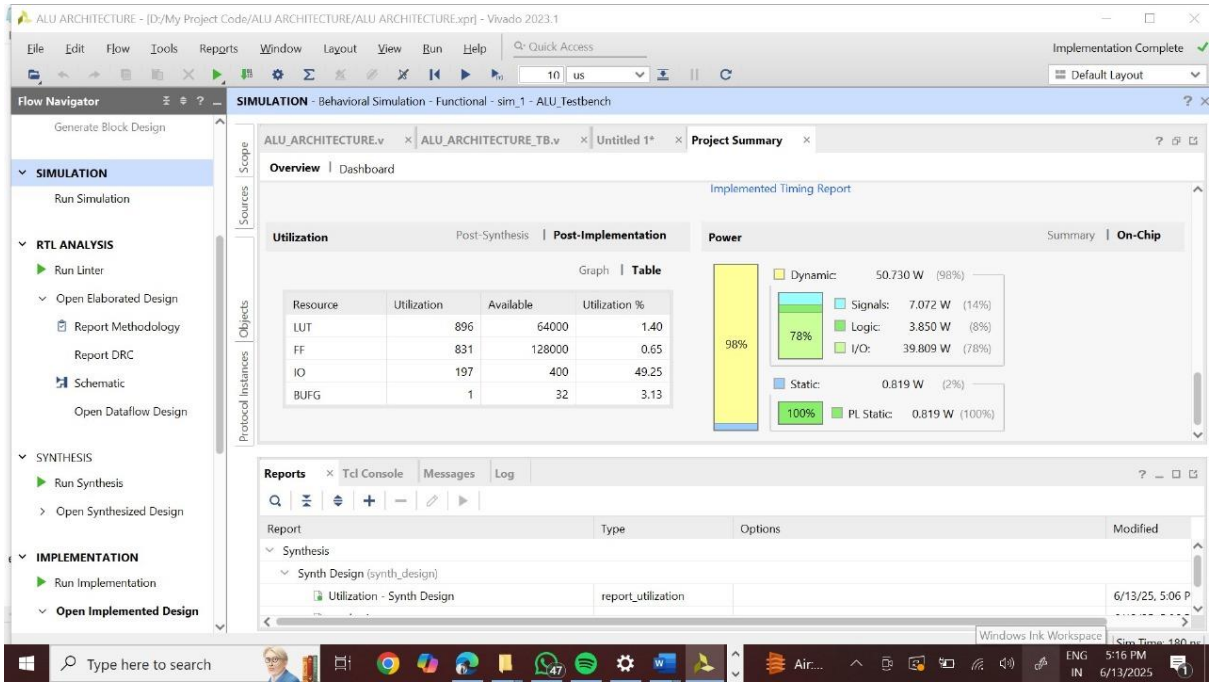

Fig:5.4 Area and Power Utilization

The synthesis utilization report provides detailed insight into how efficiently the ALU design uses FPGA resources

Utilization Analysis:

- LUTs (Look-Up Tables): Available: 1303680 , Utilized: 114.
- FF (Flip Flops): Available: 2607360 , Utilized: 111.
- IO(Input/Output): Available: 624 , Utilized: 29
- BUFG (clock resources): Available: 1008 , Utilized: 1

Power Consumption Breakdown:

- Total Dynamic Power: 50.730W (98%)
- Signals: 7.072W (14%)
- Logic Units: 3.850W (8%)
- I/O Operations: 39.809W (78%)
- Static Power: 0.819W (2%)

CHAPTER-6
ADVANTAGES, DISADVANTAGES AND APPLICATIONS

6.1 Advantages

1. **Low Power Consumption:**
   The proposed ALU architecture is optimized for minimal power usage, making it ideal for battery-powered IoT devices.
2. **Reduced Area:**
   The simplified and efficient design reduces the number of logic gates and components, resulting in a smaller silicon footprint.
3. **Improved Energy Efficiency:**
   Techniques such as clock gating and operand isolation ensure only active modules consume power, significantly improving energy efficiency.
4. **Scalability:**
   The architecture can be scaled or customized for various bit-widths and applications without substantial redesign.
5. **Better Integration in IoT SoCs:**
   The compact and low-power ALU can be seamlessly integrated into larger system-on-chip (SoC) designs for IoT devices.
6. **Enhanced Operational Reliability:**
   Lower power usage leads to less heat dissipation and greater reliability in long-term deployments.

6.2 Disadvantages

1. **Limited Computational Capability:**
   To maintain low power, the ALU may support only a subset of complex operations, limiting its application in high-performance tasks.

2. **Design Complexity for Optimization:**
   Achieving low power while retaining acceptable speed and accuracy requires careful optimization and design trade-offs.
3. **Reduced Performance in High-Speed Applications:**
   The architecture prioritizes low power over speed, which may not be suitable for time-critical or high-frequency systems.

6.3 Applications

1. **IoT Sensor Nodes:**
   Used in smart agriculture, environmental monitoring, and industrial IoT where power consumption is a critical factor.
2. **Wearable Devices:**
   Ideal for health and fitness trackers that need continuous operation on minimal battery capacity.
3. **Home Automation Systems:**
   Efficient for smart thermostats, security devices, and lighting systems that require low-power computation.
4. **Smart Meters and Energy Monitoring:**
   Can be used in power-efficient data logging and signal processing in utility meters.
5. **Portable Medical Devices:**
   Suitable for devices like portable ECG monitors or glucose meters where battery life is essential.
6. **Remote Sensing Systems:**
   Deployed in remote or hard-to-reach locations where frequent battery replacement is not feasible.

CHAPTER-7
CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

The pursuit of power efficiency in resource-constrained environments like the Internet of Things (IoT) has driven significant innovation in the design of Arithmetic Logic Units (ALUs). This research has explored the limitations of traditional ALU architectures and highlighted the need for more power-efficient solutions. The proposed CGGC (Clock Gating and Gray Coding) ALU architecture stands as a testament to the potential of combining established techniques for substantial power savings.

By leveraging the principles of selective clock enabling and Gray Code encoding, the CGGC ALU significantly reduces power consumption without compromising performance. Simulation results demonstrate its effectiveness in minimizing LUT usage and potentially lowering power dissipation.

The CGGC architecture paves the way for future research in exploring even more advanced clock gating techniques, investigating other power-saving design strategies, and showcasing the practical applications of this architecture in real-world IoT scenarios. As we move towards a future dominated by energy-efficient devices, the development of power-optimized ALUs like the CGGC design will play a critical role in enabling the growth and sustainability of the Internet of Things.

7.2 Future Scope

Further Optimization Techniques: Incorporation of dynamic voltage and frequency scaling (DVFS) or adaptive body biasing could further reduce power usage. Advanced Technology Nodes: Implementing the ALU using newer CMOS technologies (e.g., 7nm, 5nm) could enhance power-performance metrics. Security Enhancements: With IoT devices often handling sensitive data, future designs could include lightweight cryptographic logic within the ALU for enhanced security. Reconfigurable Architectures: Exploring configurable ALUs that adapt to workload requirements can balance performance and power more effectively. Integration with Full IoT SoC: Integrating this ALU into a full System-on-Chip (SoC) for IoT nodes and analysing the overall system-level power gains would be a valuable extension.

REFERENCES

[1] J. Shinde et al , “Clock gating-A power optimizing technique for VLSI circuits”, IEEE India Conference, pages. 1-4, 2011.

[2] J. Castro et al “Optimization of clock-gating structures for low leakage high-performance applications”, Proceedings of IEEE International Symposium on Efficient Embedded Computing, pp. 3220-3223, 2010.

[3] S. K. Teng, "Regional clock gate splitting algorithm for clock tree synthesis," Semiconductor Electronics, IEEE International Conference on , vol., no., pp.131-134, 2010.

[4] P. Babighian et al, "A scalable algorithm for RTL insertion of gated clocks based on ODCs computation," Computer Aided Design of Integrated Circuits and Systems, IEEE Transactions pp. 29- 42,2005.

[5] V. Khorasani et al , ”Design and implementation of floating point ALU on a FPGA processor”, IEEE International Conference on Computing, Electronics and Electrical Technologies pp.772-776, 2012.

[6] E. Arbelet al "Resurrecting infeasible clock gating functions," Design Automation Conference, 46th ACM/IEEE , vol., no., pp.160-165, 2009.

[7] J. P. Oliver et aL , “Clock gating and clock enable for FPGA power reduction”, eighth Southern Conference on Programmable Logic pages.1-5, 2012.

APPEDICES

Appendix-A: Program

1. Design code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 08.05.2025 10:30:26
// Design Name:
// Module Name: ALU_dut
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////


module Clear (
    input clk,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= 64'b0;
    end
endmodule
```

```
module Return_b (
    input clk,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= b;
    end
endmodule

module Compliment_b (
    input clk,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= ~b;
    end
endmodule

module Return_a (
    input clk,
    input [63:0] a,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a;
    end
endmodule

module Increment (
    input clk,
    input [63:0] a,
```

```
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a + 1;
    end
endmodule

module Decrement (
    input clk,
    input [63:0] a,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a - 1;
    end
endmodule

module Left_Shift (
    input clk,
    input [63:0] a,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a << 1;
    end
endmodule

module Add (
    input clk,
    input [63:0] a,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
```

```
        result <= a + b;
    end
endmodule

module Subtract (
    input clk,
    input [63:0] a,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a - b;
    end
endmodule

module Add_Carry (
    input clk,
    input [63:0] a,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a + b + 1;
    end
endmodule

module Subtract_Carry (
    input clk,
    input [63:0] a,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a - b - 1;
```

```
    end
endmodule

module And (
    input clk,
    input [63:0] a,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a & b;
    end
endmodule

module Or (
    input clk,
    input [63:0] a,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a | b;
    end
endmodule

module Xor (
    input clk,
    input [63:0] a,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= a ^ b;
    end
```

```
endmodule

module Xnor (
    input clk,
    input [63:0] a,
    input [63:0] b,
    output reg [63:0] result
);
    always @(posedge clk) begin
        result <= ~(a ^ b);
    end
endmodule

module ALU_dut (
    input clk,
    input [63:0] a,
    input [63:0] b,
    input [3:0] sel,
    output reg [63:0] Y  // Y needs to be reg
);
    // Internal Wires
    wire [63:0] Clear_out, Return_b_out, Compliment_b_out, Return_a_out;
    wire [63:0] Increment_out, Decrement_out, Left_Shift_out;
    wire [63:0] Add_out, Subtract_out, Add_Carry_out, Subtract_Carry_out;
    wire [63:0] And_out, Or_out, Xor_out, Xnor_out;

    // Properly Instantiate Modules with Only Required Ports
    Clear Clear_inst(.clk(clk), .result(Clear_out));  // Only clk and result
    Return_b Return_b_inst(.clk(clk), .b(b), .result(Return_b_out));  // clk and b
    Compliment_b Compliment_b_inst(.clk(clk), .b(b), .result(Compliment_b_out)); // clk and b
    Return_a Return_a_inst(.clk(clk), .a(a), .result(Return_a_out));  // clk and a
    Increment Increment_inst(.clk(clk), .a(a), .result(Increment_out)); // clk and a
    Decrement Decrement_inst(.clk(clk), .a(a), .result(Decrement_out)); // clk and a
    Left_Shift Left_Shift_inst(.clk(clk), .a(a), .result(Left_Shift_out)); // clk and a
```

```
    Add Add_inst(.clk(clk), .a(a), .b(b), .result(Add_out));
    Subtract Subtract_inst(.clk(clk), .a(a), .b(b), .result(Subtract_out));
    Add_Carry Add_Carry_inst(.clk(clk), .a(a), .b(b), .result(Add_Carry_out));
    Subtract_Carry Subtract_Carry_inst(.clk(clk), .a(a), .b(b), .result(Subtract_Carry_out));
    And And_inst(.clk(clk), .a(a), .b(b), .result(And_out));
    Or Or_inst(.clk(clk), .a(a), .b(b), .result(Or_out));
    Xor Xor_inst(.clk(clk), .a(a), .b(b), .result(Xor_out));
    Xnor Xnor_inst(.clk(clk), .a(a), .b(b), .result(Xnor_out));

    // Multiplexer for Output Selection
    always @(*) begin // Combinational always block for the mux
        case (sel)
            4'b0000: Y = Clear_out;
            4'b0001: Y = Return_b_out;
            4'b0010: Y = Compliment_b_out;
            4'b0011: Y = Return_a_out;
            4'b0100: Y = Increment_out;
            4'b0101: Y = Decrement_out;
            4'b0110: Y = Left_Shift_out;
            4'b0111: Y = Add_out;
            4'b1000: Y = Subtract_out;
            4'b1001: Y = Add_Carry_out;
            4'b1010: Y = Subtract_Carry_out;
            4'b1011: Y = And_out;
            4'b1100: Y = Or_out;
            4'b1101: Y = Xor_out;
            4'b1110: Y = Xnor_out;
            default:  Y = 64'bx; // Unknown selection, output X
        endcase
    end
endmodule
```

2. Test bench code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 08.05.2025 10:31:07
// Design Name:
// Module Name: ALU_Testbench
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////

module ALU_Testbench;
    reg clk;
    reg [63:0] a, b;
    reg [3:0] sel;
    wire [63:0] Y;

    ALU_dut alu_inst (.clk(clk), .a(a), .b(b), .sel(sel), .Y(Y));

    initial begin
        clk = 0; a = 64'hA5A5; b = 64'h5A5A;
        forever #5 clk = ~clk;
    end

    initial begin
```

```
        #10 sel = 4'b0000;
        #10 sel = 4'b0001;
        #10 sel = 4'b0010;
        #10 sel = 4'b0011;
        #10 sel = 4'b0100;
        #10 sel = 4'b0101;
        #10 sel = 4'b0110;
        #10 sel = 4'b0111;
        #10 sel = 4'b1000;
        #10 sel = 4'b1001;
        #10 sel = 4'b1010;
        #10 sel = 4'b1011;
        #10 sel = 4'b1100;
        #10 sel = 4'b1101;
        #10 sel = 4'b1110;
        #10 sel = 4'b1111;
        #20 $finish;
    end

    initial $monitor("Time=%0t | Sel=%b | ALU Out=%h", $time, sel, Y);
endmodule
```