

GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101



Project Report On

‘8-bit ALU Design Using Open – Source EDA Tools’

Submitted to

Institute Code: 017

Institute Name: (Vishwakarma Government Engineering College)

Under the Guidance of

Prof. Jagruti K. Naik

(Assistant Professor)

In partial Fulfilment of the Requirement of the award of the degree of

Bachelor of Engineering

Offered By

Gujarat Technological University

Ahmedabad

Prepared by:

Kapadiya Harshkumar Girishkumar

210170111128

ELECTRONICS & COMMUNICATION Semester - VIII)

Month & Year:

April/May 2022



GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101



Vishwakarma Government Engineering College

Near Visat Three Roads, Sabarmati – Koba Highway, Chandkheda

Ahmedabad, Gujarat 382424, India

CERTIFICATE

This is to certify that the internship report submitted along with the project entitled “**8-bit ALU Design Using Open – Source EDA Tools**” has been carried out by **Kapadiya Harshkumar Girishkumar** under my guidance in partial fulfilment for the degree of Bachelor of Engineering in Department of Electronics & Communication Engineering, 8th Semester of Gujarat Technological University, Ahmedabad during the Academic year 2024-2025.

Prof. Jagruti K. Naik

Internal Guide

Dr. Arun B. Nandurbarkar

Head of the Department

GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101



Vishwakarma Government Engineering College

Near Visat Three Roads, Sabarmati – Koba Highway, Chandkheda

Ahmedabad, Gujarat 382424, India

DECLARATION

I hereby declare that the project titled “**8-bit ALU Design Using Open-Source EDA Tools**” is a genuine and original piece of work carried out by me as part of my academic curriculum. This project has been completed in partial fulfillment of the requirements for the award of my undergraduate degree, and has not been submitted to any other university or institution for the award of any degree, diploma, or certificate.

The design, implementation, and documentation presented in this report are the results of my own efforts and research, under the guidance and supervision of my project mentor. All sources of information, tools, and references — including open-source EDA tools such as **Yosys, Qflow, Magic, GrayWolf, OpenSTA**, and the **SkyWater 130nm PDK** — have been properly cited and acknowledged throughout the report.

The objective of this project was to gain hands-on experience with the RTL-to-GDSII VLSI design flow using open-source platforms, and to explore how theoretical digital design concepts can be transformed into physical silicon-ready layouts using Verilog and open-source workflows. I take full responsibility for the work presented here and affirm its authenticity, accuracy, and academic integrity.

Prof. Jagruti Naik

Internal Guide

Dr. Arun B. Nandurbarkar

Head of the Department

GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project **Team ID:** 741180

Subject Code: 3181101

Abstract

This project presents the design and implementation of an 8-bit Arithmetic Logic Unit (ALU) using open-source Electronic Design Automation (EDA) tools, following a complete **RTL to GDSII** design flow. The ALU is a fundamental digital block capable of performing arithmetic and logical operations such as addition, subtraction, bitwise AND, OR, XOR, and shift operations. The design was described in Verilog HDL and synthesized using **Yosys**, an open-source logic synthesis tool. The backend flow was implemented using **Qflow**, which integrates tools like **GrayWolf** for placement, **Magic VLSI** for layout generation and DRC/LVS verification, and **OpenSTA** for static timing analysis. This project demonstrates the effectiveness of open-source EDA tools in executing the full digital IC design cycle—from register-transfer level (RTL) design to final GDSII layout—making it a cost-effective and educational approach for academic and prototyping purposes.

Acknowledgement

I would like to express my sincere gratitude to my faculty members, whose guidance and support have been instrumental in the successful completion of this project. I extend my deepest appreciation to Prof. Jagruti K. Naik for their valuable insights, encouragement, and mentorship throughout the research and development process.

I would also like to acknowledge the invaluable contributions of the open-source community and the developers of EDA tools such as **Yosys**, **Magic**, **GrayWolf**, **OpenSTA**, **Qflow**, and other essential software used in this project. Their continuous development and support have been integral to the success of this design. Additionally, the comprehensive documentation and resources provided by platforms like **VSDFlow**, **SkyWater PDK**, **efabless**, and numerous research papers have been instrumental in facilitating the understanding and implementation of the RTL-to-GDSII design flow. These resources enabled the seamless integration of tools and ensured that each stage of the ALU design process was executed efficiently.

I would also like to thank Mr. Kunal Ghosh and the VSD community for providing access to open-source design flows and educational content, which played a crucial role in my understanding of the complete RTL-to-GDSII VLSI design process.

Finally, I am grateful to my peers, friends, and family for their continuous support and motivation. Their encouragement has been essential in overcoming challenges and completing this project successfully.

GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

Table of Contents

Chapter 1: Introduction	9
Chapter 2: Introduction to Arithmetic and Logic Unit	10
2.1: Role in Digital Systems	10
2.2: Key Functions	10
2.3: Structure of ALU	10
2.4: ALU in Digital Systems	11
2.5: Block Diagram	11
2.6: How ALU Works	12
2.6.1: Arithmetic Operations	12
2.6.2: Logical Operations	12
2.6.3: Shift Operations	12
2.6.4: Comparison	12
2.6.5: Status Flag	13
2.7: Defining the ALU using Verilog HDL Language	13
2.7.1: Define the Requirements	13
2.7.2: Declare the Module and Ports	13
2.7.3: Use a case statement to select the operation	13
2.7.4: Implement the Status Flag	14
2.7.5: Test the ALU	14
2.7.6: Synthesis the Design	14
2.8: Code	15
2.9: Code Explanation	15
2.9.1: Module Declaration	15
2.9.2: Always Block: Core Operation Logic	16
2.9.3: Explanation of Each Opcode	16
2.9.4: Overflow Flag Logic	17
2.9.5: Zero Flag Logic	17
Chapter 3: Introduction to Open-Source EDA Tools	18
3.1: What are EDA Tools?	18
3.1.1: Main Roles of EDA Tools	18
3.1.2: Why are EDA Tools Important?	18
3.2: What are Open-Source EDA Tools?	19
3.2.1: Why Open Source?	19
3.2.2: Examples of Popular Open-Source EDA Tools	19
3.2.3: Benefits of Open-Source EDA Tools	19
3.2.4: Limitations	19
3.3: What are the components of EDA Tools?	20
3.3.1: RTL Design (Register Transfer Level)	20
3.3.2: Logic Synthesis	20
3.3.3: Floorplanning and Placement	20
3.3.4: Routing	20
3.3.5: Layout Editing	20
3.3.6: Static Timing Analysis	20
3.3.7: DRC/LVS Verification	20
3.3.8: PDK (Process Design Kit)	20
3.4: Which Platform Do We Use for Using Tools.....	21
3.4.1: Why Linux?	21
3.4.2: Platform Details for this project	21
3.4.3: Optional for Windows Users	21
3.5: How to install the Tools?	22
3.6: How to use these Tools?	25
3.7: How the Process Work?	27

GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project **Team ID:** 741180

Subject Code: 3181101

3.8: How can we check the progress?	29
3.9: How can we assure that our design is correct?	31
3.10: Final Output and Further Processes	32
Chapter 4: Final Layout.....	34
Chapter 5: Conclusion	35
Chapter 5: References	36

GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project **Team ID:** 741180

Subject Code: 3181101

List of Figures

Figure 1: CPU Block Diagram	11
Figure 2: ALU Block Diagram	11
Figure 3: Installation Step-1	22
Figure 4: Installation Step-2	23
Figure 5: Installation Step-3	23
Figure 6: Installation Step-4	24
Figure 7: Installation Step-5	24
Figure 8: Installation Step-6	24
Figure 9: Installation Step-7	25
Figure 10: ALU Verilog Code	25
Figure 11: Qflow Manager	26
Figure 12: File Selection	27
Figure 13: Log File Structure	30
Figure 14: Layout	34

❖ 1. Introduction

The design and implementation of an **8-bit Arithmetic Logic Unit (ALU)** forms the core of this project, with a focus on utilizing open-source Electronic Design Automation (EDA) tools. An ALU is a key building block in digital systems, particularly in microprocessors, where it performs arithmetic and logical operations. In this project, the ALU is designed to carry out fundamental operations such as **addition, subtraction, bitwise AND, OR, XOR, and shift operations**. These operations are crucial for a wide range of computational tasks in digital systems, making the ALU a critical component in the design of any computational hardware.

The objective of this project is to implement the complete design flow of an 8-bit ALU, from high-level **Register Transfer Level (RTL)** coding in **Verilog HDL** to the final **GDSII layout** using open-source EDA tools. The design flow followed in this project is based on the **RTL-to-GDSII** methodology, which encompasses several stages including RTL design, synthesis, placement, routing, layout editing, and verification. The tools used in this flow include:

- **Yosys:** For logic synthesis, which converts the Verilog RTL code into a gate-level netlist.
- **Qflow:** For managing the complete backend design flow, coordinating the various tools and processes.
- **GrayWolf:** For placement of logic cells in the design, optimizing for area and performance.
- **Magic VLSI:** For layout generation and verification, including **Design Rule Checking (DRC)** and **Layout Versus Schematic (LVS)** checks.
- **OpenSTA:** For performing **static timing analysis** to ensure that the design meets timing constraints and operates at the desired speed.

By using these open-source tools, the project not only demonstrates the feasibility of implementing complex digital designs without the need for costly commercial EDA tools but also emphasizes the potential of open-source solutions in the field of **VLSI design education and research**.

The overall process involves writing Verilog code to define the ALU operations, followed by synthesis to convert the high-level code into a gate-level representation. Placement and routing are performed to map the logic cells into physical locations on the chip, and the final layout is generated and verified using **Magic**. The design's functionality is ensured through verification steps, including timing analysis with **OpenSTA** and DRC/LVS checks using **Magic**.

This project not only showcases the practical application of open-source EDA tools but also highlights the importance of each stage in the design flow. The use of these tools makes the design process accessible, cost-effective, and efficient, allowing for the development of complex VLSI systems without relying on expensive commercial software. Furthermore, it provides an excellent learning experience for anyone interested in the field of digital design and VLSI.

❖ 2. Introduction to Arithmetic and Logic Unit

- The **Arithmetic and Logic Unit (ALU)** is a fundamental component of digital computers and processors, responsible for carrying out a wide range of arithmetic and logical operations. These operations include basic arithmetic functions such as addition, subtraction, and multiplication, as well as logical operations like AND, OR, XOR, NOT, and bit shifts. The ALU is crucial for the execution of instructions in a processor, directly affecting the processor's performance and computational ability.

➤ 2.1 Role in Digital Systems:

An ALU's primary role is to perform the necessary computations for a wide variety of applications, from simple calculations to complex data processing tasks. ALUs are typically found in the central processing unit (CPU) of a computer, where they work in conjunction with registers, buses, and memory to process data. They also serve as the core component for handling arithmetic and decision-making tasks in embedded systems and microcontrollers.

➤ 2.2 Key Functions:

The main operations performed by an ALU can be categorized into:

- **Arithmetic Operations:** These include basic operations such as addition, subtraction, and multiplication. The ALU can also handle more complex operations like division or modular arithmetic in advanced designs.
- **Logical Operations:** These include bitwise logical operations such as AND, OR, XOR, and NOT. These operations are essential for comparison, bit manipulation, and control flow in processors.
- **Shift Operations:** ALUs are capable of performing both logical and arithmetic shifts, which are commonly used for tasks like data encoding, encryption, or digital signal processing.

➤ 2.3 Structure of ALU:

The structure of an ALU generally consists of the following components:

- **Input registers:** To store the operands for the operation.
- **Control unit:** Determines the operation to be performed based on the ALU control signal.
- **Arithmetic and Logic Circuit:** This is the core of the ALU, where arithmetic and logical operations are performed.
- **Output register:** To store the result of the operation.

➤ 2.4 ALU in Digital Systems:

Research papers often highlight that the ALU's design influences the efficiency of the processor in terms of processing speed, area, and power consumption. A high-performance ALU is essential for fast computation, and optimizing its design is crucial for improving the overall system performance, especially in applications like high-speed computing, data processing, and real-time systems.

In modern processors, ALUs are typically designed as part of larger **central processing units (CPUs)** or **system on chips (SoCs)**, where their operations are integrated with other units such as control units, memory management units, and input/output interfaces. The ALU's efficiency is directly tied to the performance of these systems, and advancements in ALU design continue to focus on optimizing speed, power consumption, and chip area.

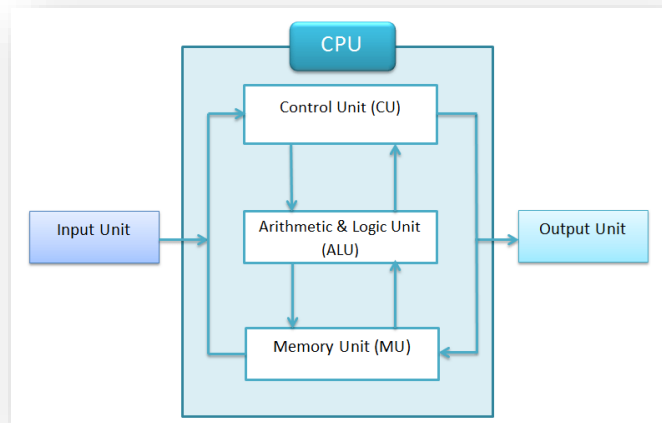


Figure 1: CPU Block Diagram

➤ 2.5 Block Diagram:

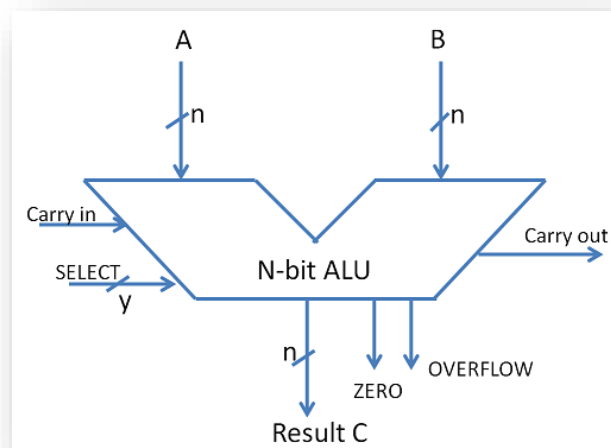


Figure 2: ALU Block Diagram

➤ 2.6 How ALU Works:

When the ALU receives inputs (A, B) and an opcode, the internal logic routes these values through different functional blocks depending on the operation selected.

Here's a breakdown of each functionality:

2.6.1 Arithmetic Operations:

- **Addition(Opcode:0000):**
The ALU adds inputs A and B. This operation is useful in counters, arithmetic calculations, and increment operations. The result is stored in an 8-bit register. If the result exceeds the 8-bit range, the **overflow flag** is triggered.
- **Subtraction(Opcode:0001):**
The ALU subtracts B from A. This is essential in comparison, loops, and calculations involving differences. Like addition, subtraction may also trigger an overflow in signed computations.

2.6.2 Logical Operations:

These operations are performed **bit by bit** on the inputs A and B.

- **AND(Opcode:0010)**
Outputs 1 only where both bits of A and B are 1. Commonly used in masking operations.
- **OR(Opcode:0011)**
Outputs 1 wherever either A or B has a 1. Useful in flag setting and enabling specific bits.
- **XOR(Opcode:0100)**
Outputs 1 where A and B differ. Often used in toggling bits or parity operations.
- **NOT(Opcode:0101)**
Inverts each bit of A. B is ignored in this case. Used for one's complement and logical inversion.

2.6.3 Shift Operations:

- **Logical Shift Left (Opcode: 0110)**
Shifts A one bit to the left, inserting 0 at the LSB. This is equivalent to multiplying A by 2.
- **Logical Shift Right (Opcode: 0111)**
Shifts A one bit to the right, inserting 0 at the MSB. This is equivalent to integer division by 2. Both shift operations are crucial in arithmetic operations, bit manipulation, and fast multipliers/dividers.

2.6.4 Comparison:

- **Less-Than Comparison (Opcode: 1000)**
Compares if A is less than B. Returns 1 if true, else 0. This is helpful in conditional decision-making and branching logic in processors.

2.6.5 Status Flags:

Our ALU design includes three output flags to provide feedback about the result:

- **Zero Flag:** Set if the result of any operation is 0. Indicates equality or a neutral result.
- **Negative Flag:** Set if the most significant bit (MSB) of the result is 1. This indicates a negative number in 2's complement representation.
- **Overflow Flag:** Set during addition or subtraction if the signed result goes beyond the range representable by 8 bits (i.e., -128 to +127 in 2's complement).

These flags are essential for control flow in processors (e.g., conditional branching or loop exits).

2.7 Defining the ALU using Verilog HDL Language:

- Step-by-Step Approach:
- Designing an ALU using Verilog involves the following steps:

2.7.1 Define the Requirements:

- Before writing the code, decide:
 - How many bits the inputs and output will be (e.g., 8-bit).
 - What operations the ALU should support (add, subtract, AND, OR, NOT, shift, etc.).
 - What **flags** you want to generate (e.g., zero, negative, overflow).
 - How you will **select operations** — usually via an opcode.

2.7.2 Declare the Module and Ports:

- Start with a Verilog module where you define:
 - Inputs: operands A, B, and control signal opcode.
 - Outputs: result, and optional flags like zero, negative, overflow.
- Example:
- ```
module ALU (
 input [7:0] A, B, // Operands
 input [3:0] opcode, // Operation selector
 output reg [7:0] result, // Output
 output wire overflow, zero, negative // Flags
);
```

### 2.7.3 Use a case Statement to Select the Operation:

- Use the opcode to switch between different ALU operations using a case block inside an always block:

# GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

```
always @(*) begin
 case (opcode)
 4'b0000: result = A + B;
 4'b0001: result = A - B;
 4'b0010: result = A & B;
 4'b0011: result = A | B;
 4'b0100: result = A ^ B;
 4'b0101: result = ~A;
 4'b0110: result = A << 1;
 4'b0111: result = A >> 1;
 4'b1000: result = (A < B) ? 8'b1 : 8'b0;
 default: result = 8'b0;
 endcase
end
```

Each case corresponds to a unique operation your ALU can perform.

## 2.7.4 Implement the Status Flags:

- You can assign flag values outside the always block based on the result:  
assign zero = (result == 8'b0); // All bits are 0  
assign negative = result[7]; // MSB = 1 means negative in 2's complement  
assign overflow = (opcode == 4'b0000) ? (A[7] == B[7]) && (result[7] != A[7]) :  
(opcode == 4'b0001) ? (A[7] != B[7]) && (result[7] != A[7]) : 1'b0;

These flags give additional information about the result, just like in real processors.

## 2.7.5 Test the ALU:

- After writing the code, **test** it using a Verilog **testbench**. Provide different inputs, opcodes, and observe if the result and flags behave as expected. We will test the code Using Xilinx Vivado Tool.

## 2.7.6 Synthesize the Design:

- Once your simulation works:  
You can **synthesize** the code using tools like **Yosys**.  
Perform **Place & Route** using **Qflow** or **GrayWolf**.  
Visualize the layout using **Magic**.  
Perform static timing analysis with **OpenSTA**.  
This flow takes your Verilog (RTL) all the way to GDSII.

# GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

## 2.8 Code:

```
module ALU (
 input [7:0] A, B, // 8-bit Inputs
 input [3:0] opcode, // 4-bit Operation Code
 output reg [7:0] result, // 8-bit Result
 output wire overflow, // Overflow Flag
 output wire negative, // Negative Flag
 output wire zero // Zero Flag
);

always @(*) begin
 case (opcode)
 4'b0000: result = A + B; // Addition
 4'b0001: result = A - B; // Subtraction
 4'b0010: result = A & B; // Bitwise AND
 4'b0011: result = A | B; // Bitwise OR
 4'b0100: result = A ^ B; // Bitwise XOR
 4'b0101: result = ~A; // Bitwise NOT (A)
 4'b0110: result = A << 1; // Logical Shift Left
 4'b0111: result = A >> 1; // Logical Shift Right
 4'b1000: result = (A < B) ? 8'b1 : 8'b0; // Compare A < B
 default: result = 8'b0;
 endcase
end

// Overflow occurs for signed addition and subtraction
assign overflow = (opcode == 4'b0000) ? (A[7] == B[7]) && (result[7] != A[7]) :
 (opcode == 4'b0001) ? (A[7] != B[7]) && (result[7] != A[7]) : 1'b0;

// Negative flag: MSB of result
assign negative = result[7];

// Zero flag: result is all zeros
assign zero = (result == 8'b0);

endmodule
```

## 2.9 Code explanation:

Below is the explanation of each section of the ALU code and how it contributes to the working of the arithmetic and logic unit:

### 2.9.1 Module Declaration:

```
module ALU (
 input [7:0] A, B, // 8-bit Inputs
 input [3:0] opcode, // 4-bit Operation Code
 output reg [7:0] result, // 8-bit Result
 output wire overflow, // Overflow Flag
 output wire negative, // Negative Flag
 output wire zero // Zero Flag
);
```

# GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

- This defines a module named ALU.
- It takes two **8-bit inputs** A and B (operands).
- A **4-bit opcode** decides which operation to perform.
- The result is an **8-bit output**.
- There are **three status flags** to describe the nature of the output: overflow, negative, and zero.

## 2.9.2 Always Block: Core Operation Logic:

```
always @(*) begin
 case (opcode)
 4'b0000: result = A + B;
 4'b0001: result = A - B;
 4'b0010: result = A & B;
 4'b0011: result = A | B;
 4'b0100: result = A ^ B;
 4'b0101: result = ~A;
 4'b0110: result = A << 1;
 4'b0111: result = A >> 1;
 4'b1000: result = (A < B) ? 8'b1 : 8'b0;
 default: result = 8'b0;
 endcase
end
```

This always block reacts to any change in inputs and executes logic based on the opcode.

## 2.9.3 Explanation of Each Opcode:

| Opcode  | Operation | Description                           |
|---------|-----------|---------------------------------------|
| 0000    | A + B     | Adds inputs A and B.                  |
| 0001    | A - B     | Subtracts B from A.                   |
| 0010    | A & B     | Bitwise AND of A and B.               |
| 0011    | A   B     | Bitwise OR of A and B.                |
| 0100    | A ^ B     | Bitwise XOR of A and B.               |
| 0101    | ~A        | Bitwise NOT (inverts all bits of A).  |
| 0110    | A << 1    | Logical left shift (multiply A by 2). |
| 0111    | A >> 1    | Logical right shift (divide A by 2).  |
| 1000    | A < B     | Comparison: 1 if A < B else 0.        |
| default | 8'b0      | Default to 0 if opcode is invalid.    |



# GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

## 2.9.4 Overflow Flag Logic:

```
assign overflow = (opcode == 4'b0000) ? (A[7] == B[7]) && (result[7] != A[7]) :
 (opcode == 4'b0001) ? (A[7] != B[7]) && (result[7] != A[7]) : 1'b0;
```

- For **addition**, overflow occurs when both inputs have the same sign but the result has a different sign.
- For **subtraction**, overflow occurs when inputs have different signs and the result's sign doesn't match A's sign.
- For other operations, overflow is not relevant and defaults to 0.

## 2.9.4 Zero Flag Logic:

```
assign zero = (result == 8'b0);
```

- This flag is set when **all bits of the result are 0**, indicating a **zero output**.

### **❖ 3. Introduction to Open-Source EDA Tools:**

#### **3.1 What are EDA Tools?**

**EDA** stands for **Electronic Design Automation**. EDA tools are specialized **software programs used to design, analyze, simulate, verify, and layout electronic systems** such as digital integrated circuits (ICs), analog circuits, and printed circuit boards (PCBs).

These tools automate complex design processes that were traditionally done manually, enabling engineers to handle the **design of large-scale, high-performance chips** like microprocessors, memory, and digital signal processors (DSPs).

##### **3.1.1 Main Roles of EDA Tools:**

- **Design Entry:** Writing hardware description language (HDL) code such as Verilog or VHDL.
- **Simulation:** Testing and verifying the logic functionality before implementation.
- **Synthesis:** Converting RTL (Register Transfer Level) code into a gate-level netlist.
- **Place and Route (P&R):** Physically placing logic cells and routing wires on silicon.
- **Static Timing Analysis:** Ensuring the design meets timing constraints.
- **Layout Generation:** Producing a GDSII file for fabrication.
- **Verification & Signoff:** Checking for correctness, performance, and manufacturability.

##### **3.1.2 Why are EDA Tools Important?**

- **Accuracy:** Enables precise modeling of complex behavior in hardware.
- **Automation:** Reduces manual labor, errors, and design time.
- **Scalability:** Handles billions of transistors in modern SoCs (System-on-Chip).
- **Fabrication Ready:** Generates manufacturing-ready files.

### 3.2 What are Open-Source EDA Tools?

**Open-source EDA tools** are **free-to-use software** packages for designing and verifying electronic circuits, made available with their **source code open to the public**. These tools allow individuals, students, startups, and even companies to **develop, test, and implement hardware designs** without the need for expensive commercial licenses.

They provide functionality similar to commercial tools (like Cadence, Synopsys, Mentor Graphics) but are community-driven and transparent.

#### 3.2.1 Why Open Source?

- Open-source EDA tools are essential for **academic learning, research, and prototyping**, especially when access to industry-grade tools is limited.
- With increasing demand for **open silicon development** and **custom chip design**, open-source tools have become a strong alternative.

#### 3.2.2 Examples of Popular Open-Source EDA Tools:

| Tool                | Purpose                                                |
|---------------------|--------------------------------------------------------|
| <b>Yosys</b>        | RTL synthesis – converts Verilog to gate-level netlist |
| <b>Qflow</b>        | Complete RTL-to-GDSII digital flow using open tools    |
| <b>Magic</b>        | Layout editing and DRC/LVS checks                      |
| <b>GrayWolf</b>     | Placement tool used in Qflow                           |
| <b>OpenSTA</b>      | Static Timing Analysis                                 |
| <b>KLayout</b>      | Layout viewer and verification                         |
| <b>SkyWater PDK</b> | Open-source process design kit (for 130nm fabrication) |

#### 3.2.3 Benefits of Open-Source EDA Tools:

- **Free & Accessible** – No license fees.
- **Educational Value** – Transparent process for learning how chip design works.
- **Community Support** – Active contributions and updates from global users.
- **Customizability** – Source code can be modified for advanced users.

#### 3.2.4 Limitations:

- May lack the performance or maturity of commercial tools.
- Support may depend on community forums rather than dedicated teams.
- Integration might require more manual setup.

### **3.3 What are the Components of EDA Tools?**

The **EDA toolchain** is made up of several components, each responsible for a specific phase in the design and development of a digital circuit or chip. From writing the logic to generating a physical layout, each tool has its role in the **RTL-to-GDSII** flow.

Let's break down the **main components** involved in a typical digital IC design flow using open-source EDA tools:

#### **3.3.1 RTL Design (Register Transfer Level):**

**Purpose:** Describes the logic behavior of the system.

**Language Used:** Verilog (used in your ALU).

**Tool Example:** Any text editor + simulator.

#### **3.3.2 Logic Synthesis:**

**Purpose:** Converts RTL code into a gate-level netlist (collection of logic gates).

**Tool Example:** Yosys

**Output:** A netlist in a format like .blif or .v.

#### **3.3.3 Floorplanning and Placement:**

**Purpose:** Defines the layout of standard cells and macro blocks on silicon.

**Tool Example:** GrayWolf

**Role:** Decides where each logic cell goes within the chip layout.

#### **3.3.4 Routing:**

**Purpose:** Connects all the placed cells with metal wires based on the netlist.

**Tool Example:** Integrated in Qflow

**Goal:** Establish signal paths ensuring no short circuits or open nets.

#### **3.3.5 Layout Editing:**

**Purpose:** Allows viewing and editing of the chip's physical layout.

**Tool Example:** Magic, KLayout

**Use:** Check DRC, visualize final layout, export GDSII.

#### **3.3.6 Static Timing Analysis (STA):**

**Purpose:** Verifies if the timing constraints are met (i.e., the circuit is fast enough).

**Tool Example:** OpenSTA

**Output:** Timing reports (setup/hold violations, clock paths).

#### **3.3.7 DRC/LVS Verification:**

**DRC:** Design Rule Check – ensures the layout follows foundry rules.

**LVS:** Layout vs Schematic – ensures layout matches the design logic.

**Tool Example:** Magic, Netgen

#### **3.3.8 PDK (Process Design Kit):**

**Purpose:** Contains technology-specific data (e.g., for SkyWater 130nm).

**Example:** Sky130 PDK

**Role:** Provides standard cell libraries, rules for DRC/LVS, layers, etc.

### 3.4 Which Platform Do We Use for Using Open-Source Tools?

Open-source EDA tools are **flexible and platform-independent**, but for optimal performance, most of them are **developed and tested primarily on Linux-based systems**. For this project, we used the **Ubuntu (Linux) platform**, which is widely recommended by the open-source hardware design community.

#### 3.4.1 Why Linux?

- Most EDA tools are designed for **Linux terminal-based environments**.
- It supports scripting, automation, and package management (e.g., `apt`, `make`, `bash`) which are essential for running flows like **Qflow**, **Yosys**, and **Magic**.
- Open-source PDKs like **SkyWater130** are also packaged with scripts compatible with Linux distributions.

#### 3.4.2 Platform Details for This Project:

| Component                 | Details                                      |
|---------------------------|----------------------------------------------|
| <b>Operating System</b>   | Ubuntu 20.04 LTS (or later recommended)      |
| <b>Processor</b>          | Intel/AMD 64-bit                             |
| <b>RAM</b>                | Minimum 4 GB (8 GB or more recommended)      |
| <b>Storage</b>            | At least 10 GB free space                    |
| <b>Software Interface</b> | Terminal + Text Editor (like VS Code or Vim) |
| <b>Flow Setup</b>         | VSDFlow by Kunal Ghosh (VSDOpen platform)    |

#### 3.4.3 Optional for Windows Users:

Windows users can run the tools via:

- **WSL (Windows Subsystem for Linux)**
- **Virtual Machines (VMs)** using VirtualBox or VMware with an Ubuntu image

However, using native Ubuntu or other Linux distributions is **strongly preferred** for stability and compatibility.

### **3.5 How to Install Open-Source EDA Tools Using VSDFlow (by Kunal Ghosh):**

Installing open-source EDA tools manually can be complicated, but thanks to **VSDFlow**, the entire toolchain needed for digital VLSI design can be set up with a few commands. VSDFlow automates the installation of synthesis, placement, routing, STA, and layout tools — all ready to use for RTL to GDSII flow.

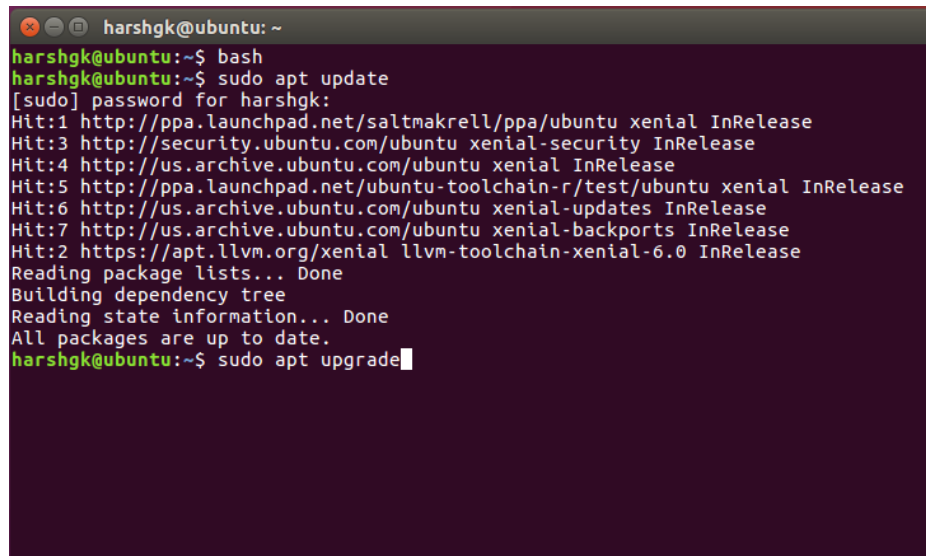
Let's break it down from scratch:

#### **Step-by-Step Installation Guide:**

##### **✓ Step-1:**

- Open the terminal and run below listed commands:

- 1 Bash**
- 2 sudo apt update**
- 3 sudo apt upgrade**



```
harshgk@ubuntu: ~
harshgk@ubuntu:~$ bash
harshgk@ubuntu:~$ sudo apt update
[sudo] password for harshgk:
Hit:1 http://ppa.launchpad.net/saltmakrell/ppa/ubuntu xenial InRelease
Hit:3 http://security.ubuntu.com/ubuntu xenial-security InRelease
Hit:4 http://us.archive.ubuntu.com/ubuntu xenial InRelease
Hit:5 http://ppa.launchpad.net/ubuntu-toolchain-r/test/ubuntu xenial InRelease
Hit:6 http://us.archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:7 http://us.archive.ubuntu.com/ubuntu xenial-backports InRelease
Hit:2 https://apt.llvm.org/xenial llvm-toolchain-xenial-6.0 InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
harshgk@ubuntu:~$ sudo apt upgrade
```

*Figure 3 Installation Step-1*

##### **✓ Step-2:**

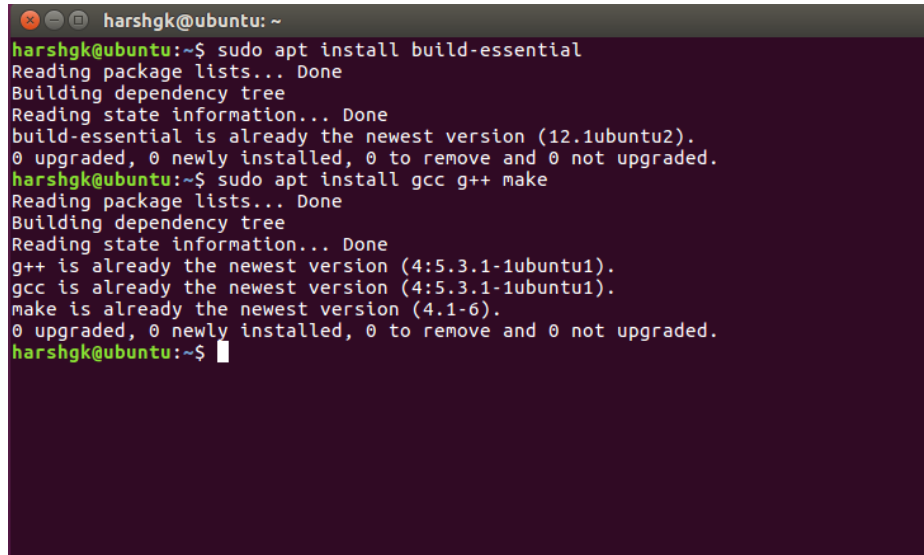
- Installing Essential Tools, Run below listed commands:

- 1. bash**
- 2. sudo apt install build-essential**
- 3. sudo apt install gcc g++ make**

# GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101



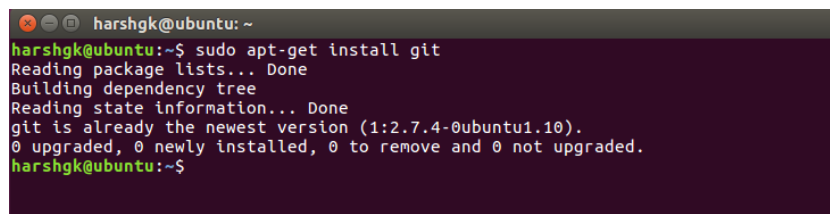
```
harshgk@ubuntu: ~
harshgk@ubuntu:~$ sudo apt install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.1ubuntu2).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
harshgk@ubuntu:~$ sudo apt install gcc g++ make
Reading package lists... Done
Building dependency tree
Reading state information... Done
g++ is already the newest version (4:5.3.1-1ubuntu1).
gcc is already the newest version (4:5.3.1-1ubuntu1).
make is already the newest version (4.1-6).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
harshgk@ubuntu:~$
```

Figure 4 Installation Step-2

## ✓ Step-3:

- Installation Of git, Run below listed commands:

### 1. **sudo apt-get install git**



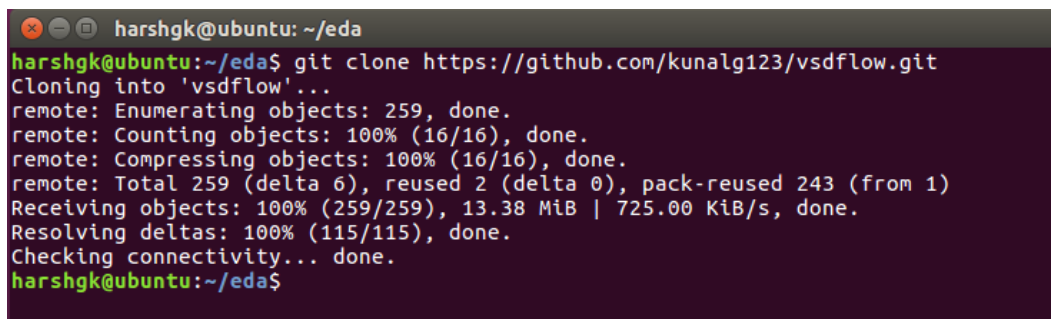
```
harshgk@ubuntu: ~
harshgk@ubuntu:~$ sudo apt-get install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.7.4-0ubuntu1.10).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
harshgk@ubuntu:~$
```

Figure 5 Installation Step-3

## ✓ Step-4:

- Cloning Repository, Run below listed commands:

### 1. **git clone <https://github.com/kunalg123/vsdfLOW.git>**



```
harshgk@ubuntu: ~/eda
harshgk@ubuntu:~/eda$ git clone https://github.com/kunalg123/vsdfLOW.git
Cloning into 'vsdfLOW'...
remote: Enumerating objects: 259, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 259 (delta 6), reused 2 (delta 0), pack-reused 243 (from 1)
Receiving objects: 100% (259/259), 13.38 MiB | 725.00 KiB/s, done.
Resolving deltas: 100% (115/115), done.
Checking connectivity... done.
harshgk@ubuntu:~/eda$
```

Figure 6 Installation Step-4

# GUJARAT TECHNOLOGICAL UNIVERSITY

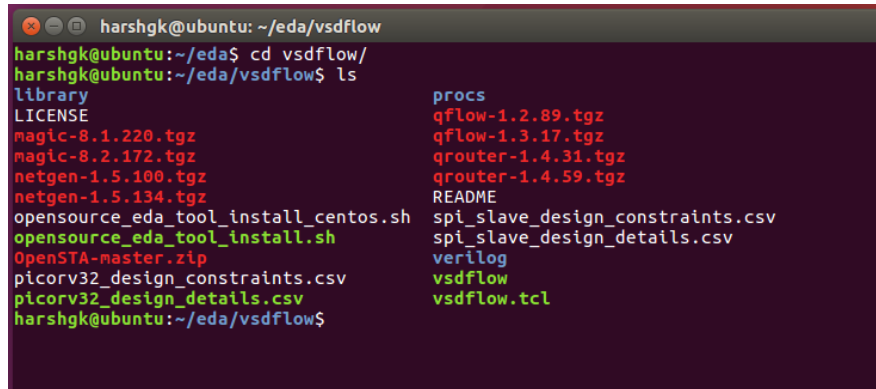
Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

## ✓ Step-5:

- Getting in to vsdflow directory, Run below listed commands:

1. `cd vsdflow`
2. `ls` (to list the items of vsdflow directory)



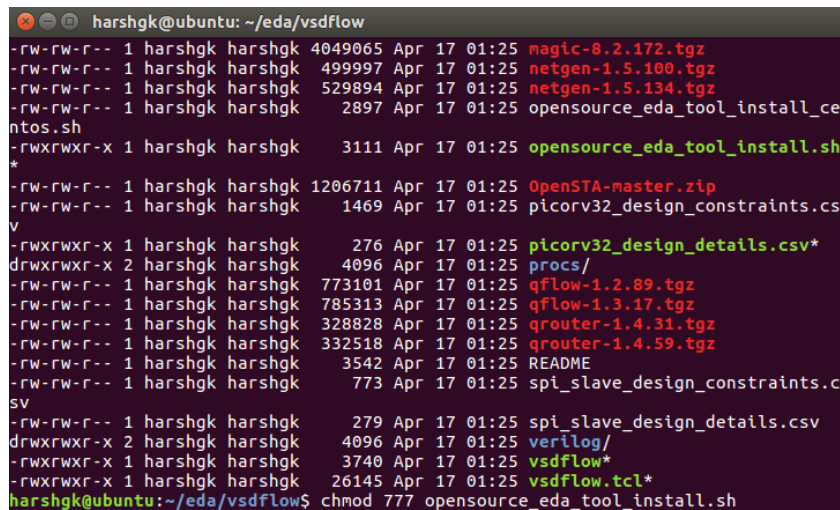
```
harshgk@ubuntu: ~/eda/vsdflow
harshgk@ubuntu:~/eda$ cd vsdflow/
harshgk@ubuntu:~/eda/vsdflow$ ls
library procs
LICENSE qflow-1.2.89.tgz
magic-8.1.220.tgz qflow-1.3.17.tgz
magic-8.2.172.tgz qrouter-1.4.31.tgz
netgen-1.5.100.tgz qrouter-1.4.59.tgz
netgen-1.5.134.tgz README
opensource_eda_tool_install_centos.sh spi_slave_design_constraints.csv
opensource_eda_tool_install.sh spi_slave_design_details.csv
OpenSTA-master.zip verilog
picorv32_design_constraints.csv vsdflow
picorv32_design_details.csv vsdflow.tcl
harshgk@ubuntu:~/eda/vsdflow$
```

Figure 7 Installation Step-5

## ✓ Step-7:

- Changing permission to read, write and execute, Run below listed commands:

1. `chmod 777 opensource_eda_tool_install.sh`



```
harshgk@ubuntu: ~/eda/vsdflow
-rw-rw-r-- 1 harshgk harshgk 4049065 Apr 17 01:25 magic-8.2.172.tgz
-rw-rw-r-- 1 harshgk harshgk 499997 Apr 17 01:25 netgen-1.5.100.tgz
-rw-rw-r-- 1 harshgk harshgk 529894 Apr 17 01:25 netgen-1.5.134.tgz
-rw-rw-r-- 1 harshgk harshgk 2897 Apr 17 01:25 opensource_eda_tool_install_centos.sh
-rwxrwxr-x 1 harshgk harshgk 3111 Apr 17 01:25 opensource_eda_tool_install.sh
*
-rw-rw-r-- 1 harshgk harshgk 1206711 Apr 17 01:25 OpenSTA-master.zip
-rw-rw-r-- 1 harshgk harshgk 1469 Apr 17 01:25 picorv32_design_constraints.csv
-rwxrwxr-x 1 harshgk harshgk 276 Apr 17 01:25 picorv32_design_details.csv*
drwxrwxr-x 2 harshgk harshgk 4096 Apr 17 01:25 procs/
-rw-rw-r-- 1 harshgk harshgk 773101 Apr 17 01:25 qflow-1.2.89.tgz
-rw-rw-r-- 1 harshgk harshgk 785313 Apr 17 01:25 qflow-1.3.17.tgz
-rw-rw-r-- 1 harshgk harshgk 328828 Apr 17 01:25 qrouter-1.4.31.tgz
-rw-rw-r-- 1 harshgk harshgk 332518 Apr 17 01:25 qrouter-1.4.59.tgz
-rw-rw-r-- 1 harshgk harshgk 3542 Apr 17 01:25 README
-rw-rw-r-- 1 harshgk harshgk 773 Apr 17 01:25 spi_slave_design_constraints.csv
-rw-rw-r-- 1 harshgk harshgk 279 Apr 17 01:25 spi_slave_design_details.csv
drwxrwxr-x 2 harshgk harshgk 4096 Apr 17 01:25 verilog/
-rwxrwxr-x 1 harshgk harshgk 3740 Apr 17 01:25 vsdflow*
-rwxrwxr-x 1 harshgk harshgk 26145 Apr 17 01:25 vsdflow.tcl*
harshgk@ubuntu:~/eda/vsdflow$ chmod 777 opensource_eda_tool_install.sh
```

Figure 8 Installation Step-6

## ✓ Step-8:

- Executing script “opensource\_eda\_tool\_install.sh”, Run below listed commands:

1. `./opensource_eda_tool_install.sh`



```
harshgk@ubuntu: ~/eda/vsdfow
harshgk@ubuntu:~/eda/vsdfow$./opensource_eda_tool_install.sh
Reading package lists... Done
Building dependency tree
Reading state information... Done
bison is already the newest version (2:3.0.4.dfsg-1).
build-essential is already the newest version (12.1ubuntu2).
flex is already the newest version (2.6.0-11).
gawk is already the newest version (1:4.1.3+dfsg-0.1).
libffi-dev is already the newest version (3.2.1-4).
libreadline-dev is already the newest version (6.3-8ubuntu2).
pkg-config is already the newest version (0.29.1-0ubuntu1).
python3 is already the newest version (3.5.1-3).
tcl-dev is already the newest version (8.6.0+9).
tk-dev is already the newest version (8.6.0+9).
xdot is already the newest version (0.6-3).
git is already the newest version (1:2.7.4-0ubuntu1.10).
graphviz is already the newest version (2.38.0-12ubuntu2.1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Reading package lists... Done
Building dependency tree
Reading state information... Done
libglu1-mesa-dev is already the newest version (9.0.0-2.1).
freeglut3-dev is already the newest version (2.8.1-2).
```

Figure 9 Installation Step-7

## 3.6 How to use these tools:

### ✓ Step-1: Write the Verilog RTL File

- Create your digital design using Verilog HDL.

```
harshgk@ubuntu: ~/vsdfow/alu_design_file
module ALU (
 input [7:0] A, B, // 8-bit Inputs
 input [3:0] opcode, // 4-bit Operation Code
 output reg [7:0] result, // 8-bit Result
 output wire overflow, // Overflow Flag
 output wire negative, // Negative Flag
 output wire zero // Zero Flag
);

always @(*) begin
 case (opcode)
 4'b0000: result = A + B; // Addition
 4'b0001: result = A - B; // Subtraction
 4'b0010: result = A & B; // Bitwise AND
 4'b0011: result = A | B; // Bitwise OR
 4'b0100: result = A ^ B; // Bitwise XOR
 4'b0101: result = ~A; // Bitwise NOT (A)
 4'b0110: result = A << 1; // Logical Shift Left
 4'b0111: result = A >> 1; // Logical Shift Right
 4'b1000: result = (A < B) ? 8'b1 : 8'b0; // Compare A < B
 default: result = 8'b0;
 endcase
end

// Overflow occurs for signed addition and subtraction
assign overflow = (opcode == 4'b0000) ? (A[7] == B[7]) && (result[7] != A[7]) :
 (opcode == 4'b0001) ? (A[7] != B[7]) && (result[7] != A[7]) :
 1'b0;

// Negative flag: MSB of result
assign negative = result[7];

// Zero flag: result is all zeros
assign zero = (result == 8'b0);

endmodule

~
1,1 All
```

Figure 10 ALU Verilog Code

# GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

- For this project, the Verilog file contains the logic of the **8-bit Arithmetic and Logic Unit (ALU)**, with various operations like addition, subtraction, bitwise logic, shift operations, and comparisons.
- Save the file as alu.v.

This file describes the behavior of the ALU, and it is the **only code input** you provide to Qflow.

## ✓ Step-2: Run Qflow and Load the Verilog File:

Once the Verilog file is ready, Qflow takes care of the entire backend process:

1. Open a terminal and navigate to your project folder.
2. Run the complete flow with a single command: **qflow gui**

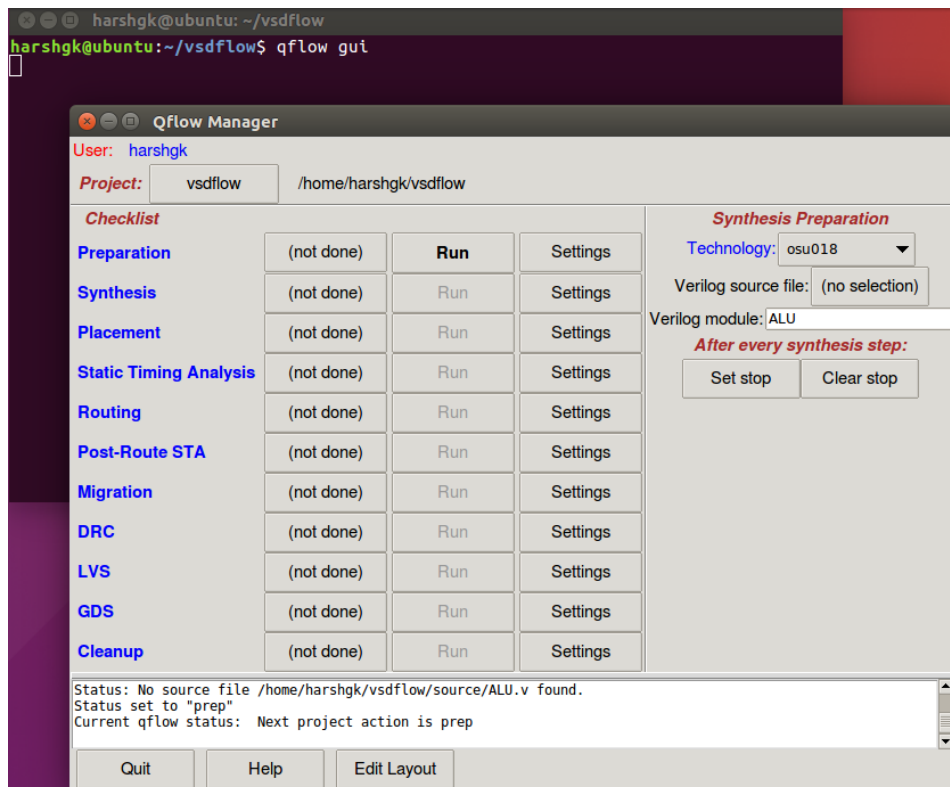


Figure 11 Qflow Manager

3. In the GUI:
  1. Select your design (alu.v)

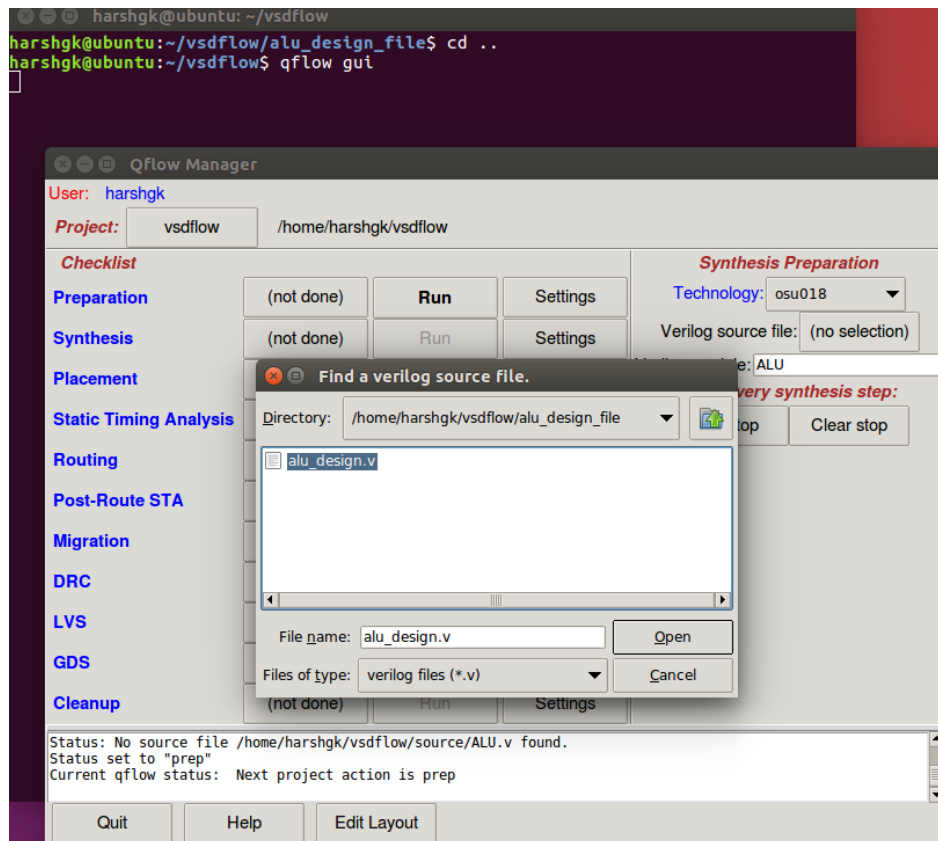


Figure 12 File Selection

2. Click on each step: **Synthesis** → **Placement** → **Routing** → **STA** → **GDSII**

Qflow will automatically use:

- **Yosys** for synthesis
- **GrayWolf** for placement
- **QRouter** for routing
- **OpenSTA** for timing analysis
- **Magic** for layout and GDSII generation

## ✓ Final Output:

By the end of this process, Qflow generates:

- The physical layout (alu.gds)
- Timing reports
- DRC and LVS checks
- A complete backend implementation of your ALU design

## 3.7 Topic 4 – Subtopic 9: How the Process Works?

In digital VLSI design, especially when using open-source tools like **Qflow**, the process of converting a hardware description (written in Verilog) into a physical layout (GDSII) follows a structured and automated flow known as the **RTL-to-GDSII flow**.

Here's how the process works, step by step:

# GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

## ➤ Step-by-Step Flow in Qflow:

### 1. RTL Design (Verilog):

The design begins by writing the **Register Transfer Level (RTL)** code in Verilog.

For this project, the RTL describes an 8-bit ALU that performs arithmetic and logical operations.

### 2. Synthesis (Using Yosys):

Qflow uses **Yosys** to convert the Verilog code into a **gate-level netlist**.

It maps logical expressions into standard logic gates (AND, OR, NOT, etc.) using standard cells from the technology library (e.g., Sky130).

### 3. Placement (Using GrayWolf):

The **GrayWolf** tool places the logic gates (cells) onto a chip layout in a way that minimizes wire length and congestion.

This step determines the physical location of each standard cell.

### 4. Routing (Using QRouter):

**QRouter** connects the placed standard cells by drawing metal wires according to the netlist.

It ensures proper electrical connectivity while respecting design rules.

### 5. Static Timing Analysis (Using OpenSTA):

**OpenSTA** checks whether the design meets required timing constraints.

It verifies if the data signals arrive at the correct time considering delays due to logic and interconnects.

### 6. Design Rule Check (Using Magic):

**Magic** checks the layout for **DRC violations** (spacing, width, overlap issues, etc.).

Ensures the layout complies with the fabrication process rules.

### 7. LVS (Using Netgen – optional):

**LVS (Layout vs. Schematic)** is used to ensure that the layout matches the original gate-level design.

This step verifies correctness of the physical implementation.

### 8. GDSII Generation (Using Magic):

Magic finally exports the chip layout to a **GDSII file**, which is the industry-standard format used for fabrication.

### Outcome:

At the end of this process, you have:

A fully verified and timed **chip layout**

A GDSII file ready for tape-out or further simulations

A deeper understanding of how digital design moves from **code to chip**.

### **3.8 How Can We Check the Progress of Each Step?**

While Qflow automates the full RTL-to-GDSII flow, it still provides detailed logs and intermediate files at every step — allowing you to monitor the **progress**, **success**, or **issues** of your design at each stage. This is especially useful in educational and project-based settings to build understanding and confidence in digital design flow.

#### **Monitoring Progress in Qflow:**

Each time you run a step in Qflow (synthesize, place, route, sta, gdsii), Qflow does the following:

- Creates a **log file** specific to that step
- Stores **intermediate output files**
- Displays messages in the **terminal** or **GUI window**

Here's how to monitor progress at each stage:

#### **1. Synthesis – qflow synthesize <design>**

**Log File:** synthesis.log

**Output File:** design.blif

**Check for:**

Warnings/errors about missing modules

Whether synthesis completed successfully

The generated gate-level netlist

#### **2. Placement – qflow placement <design>**

**Log File:** placement.log

**Output Files:** design.placement, .cel, .rcfile

**Check for:**

Cell overlap errors

Cell density and arrangement quality

Warnings from GrayWolf

#### **3. Routing – qflow route <design>**

**Log File:** route.log

**Output Files:** design.route, .def, .mag

**Check for:**

Routed net completion

Any unconnected wires (opens/shorts)

Wire congestion

#### **4. Static Timing Analysis – qflow sta <design>**

**Log File:** sta.log

**Check for:**

Setup/Hold violations

Slack values of critical paths

Paths that might limit clock speed

# GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

## 5. GDSII Export – qflow gdsii <design>

Log File: gdsii.log

Output File: design.gds

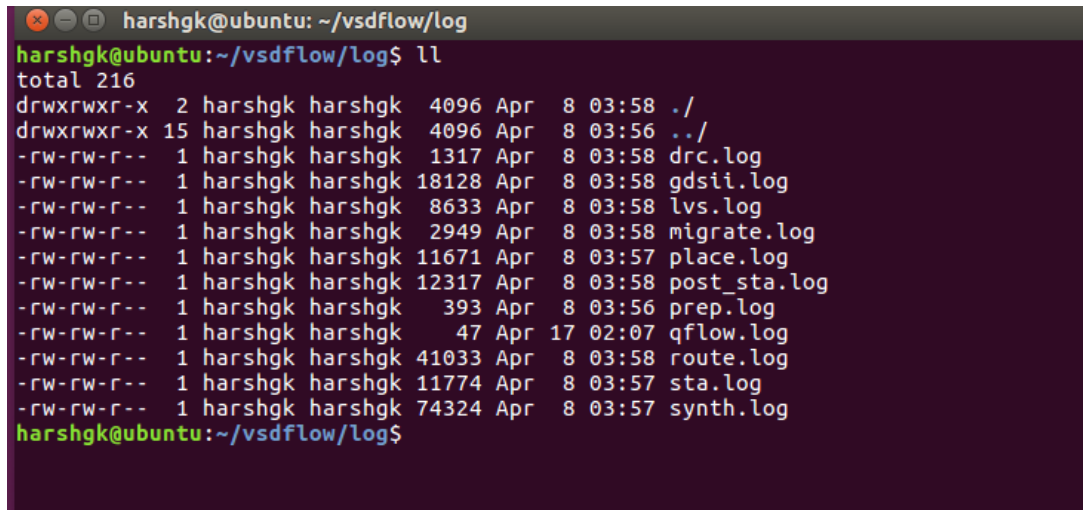
Check for:

Confirmation of GDSII file generation

Layout completion

## File Structure Overview:

Once the full flow is run, your project folder contains:

A terminal window screenshot showing the file structure of the directory ~/vsdfwflow/log. The terminal title is 'harshgk@ubuntu: ~/vsdfwflow/log'. The user has entered the command 'll' (ls -l). The output shows a list of files and directories with their permissions, owner, group, size, date, and time. The files are: ./, ../, drc.log, gdsii.log, lvs.log, migrate.log, place.log, post\_sta.log, prep.log, qflow.log, route.log, sta.log, and synth.log. The total size of the directory is 216.

```
harshgk@ubuntu: ~/vsdfwflow/log
harshgk@ubuntu:~/vsdfwflow/log$ ll
total 216
drwxrwxr-x 2 harshgk harshgk 4096 Apr 8 03:58 ./
drwxrwxr-x 15 harshgk harshgk 4096 Apr 8 03:56 ../
-rw-rw-r-- 1 harshgk harshgk 1317 Apr 8 03:58 drc.log
-rw-rw-r-- 1 harshgk harshgk 18128 Apr 8 03:58 gdsii.log
-rw-rw-r-- 1 harshgk harshgk 8633 Apr 8 03:58 lvs.log
-rw-rw-r-- 1 harshgk harshgk 2949 Apr 8 03:58 migrate.log
-rw-rw-r-- 1 harshgk harshgk 11671 Apr 8 03:57 place.log
-rw-rw-r-- 1 harshgk harshgk 12317 Apr 8 03:58 post_sta.log
-rw-rw-r-- 1 harshgk harshgk 393 Apr 8 03:56 prep.log
-rw-rw-r-- 1 harshgk harshgk 47 Apr 17 02:07 qflow.log
-rw-rw-r-- 1 harshgk harshgk 41033 Apr 8 03:58 route.log
-rw-rw-r-- 1 harshgk harshgk 11774 Apr 8 03:57 sta.log
-rw-rw-r-- 1 harshgk harshgk 74324 Apr 8 03:57 synth.log
harshgk@ubuntu:~/vsdfwflow/log$
```

Figure 13 Log File Structure

### 3.9 How Can We Assure That Our Design is Correct?

Ensuring the correctness of a digital VLSI design — from RTL to layout — is critical before moving to fabrication or final usage. In the open-source design flow using **Qflow**, multiple verification and analysis steps are provided to help ensure your **8-bit ALU** design is **functionally accurate**, **timing-correct**, and **physically valid**.

#### 1. RTL Simulation (Before Qflow):

Before entering the Qflow flow, you can verify the Verilog code using **simulation tools** like:

**Icarus Verilog** (for compiling Verilog)

**GTKWave** (for waveform viewing)

**Why?**

To test the **logical behavior** of your ALU for all opcodes. You can write a testbench to verify addition, subtraction, bitwise ops, shifts, and comparisons work correctly.

- `iverilog -o alu_tb alu.v alu_testbench.v`
- `vvp alu_tb`
- `gtkwave dump.vcd`

#### 2. Synthesis Validation (Yosys Output):

After Yosys synthesizes the Verilog, you can:

Inspect the **gate-level netlist** (.blif)

Use **Yosys reports** to ensure correct module instantiation and logic mapping

This assures that the RTL is properly transformed into a gate-level structure.

#### 3. Static Timing Analysis (Using OpenSTA):

Qflow runs **OpenSTA** to ensure that the design meets:

**Setup and Hold constraints**

**Critical path timing**

No **timing violations**

If the timing is correct, your design will work at the expected clock frequency without glitches.

#### 4. Zero Warnings/Errors in Log Files:

Each Qflow step (synthesis, placement, routing, STA, GDSII) produces a log file:

You must **check for errors or warnings**

Look for messages like "Error", "Violation", "Unconnected nets"

Smooth logs = high confidence in design correctness.

#### 5. Visual Layout Inspection (Using Magic):

Using **Magic**, you can load the .mag file and:

See the physical layout of the ALU

Zoom into gates and connections

Ensure the layout is clean, compact, and without overlaps

## **6. Design Rule Check (DRC):**

Magic performs **DRC** automatically:

It ensures the layout complies with foundry rules (like wire width, spacing, etc.)

No DRC errors = Layout is ready for fabrication

## **7. Layout vs Schematic (Optional LVS):**

Using **Netgen** (can be integrated manually), you can compare:

The **schematic netlist** (from synthesis)

The **extracted netlist** (from layout)

This ensures that layout functionally matches your Verilog logic.

## **3.10 What is the Final Output of the Design? And What Are the Processes Further?**

After going through all the stages of the **RTL-to-GDSII flow** using open-source EDA tools integrated via **Qflow**, the project produces several critical outputs that represent the completed digital IC design. These outputs are used either for simulation, layout verification, or even fabrication (in advanced cases).

### **➤ Final Outputs of the Design:**

- **GDSII File (.gds)**  
This is the **final physical layout** file of your 8-bit ALU.  
It is a standard format used by fabrication foundries to manufacture the IC.  
You can view it using tools like **KLayout** or **Magic**.
- **Layout File (.mag)**  
A Magic-specific layout file that can be visually inspected for DRC and debugging.  
Used during intermediate steps in Qflow.
- **Synthesized Netlist (.blif or .v)**  
Contains the gate-level logic of the ALU.  
Used in the synthesis and verification stages.
- **DEF File (.def)**  
Describes the placement and routing details of cells and wires.  
Can be loaded into place-and-route tools for inspection.
- **Timing Report (sta.log)**  
Generated by OpenSTA, it shows delay paths and confirms if your design meets timing requirements.
- **DRC Report**  
Ensures the layout complies with the fabrication process rules.  
Generated by Magic.
- **Log Files (for every stage)**  
Used to trace the progress and identify errors during synthesis, placement, routing, and analysis.



# GUJARAT TECHNOLOGICAL UNIVERSITY

**Subject Name:** Internship/ Project **Team ID:** 741180

**Subject Code:** 3181101

## ➤ **Further Processes After GDSII:**

Once the design has reached this stage, the following paths are possible:

### **1. Further Verification (Optional):**

You can perform more **LVS (Layout vs. Schematic)** using **Netgen**.

You can simulate using **Post-layout simulation** (e.g., using iverilog + VCD).

### **2. Fabrication (Advanced):**

The .gds file can be sent to a foundry (like SkyWater) for real IC fabrication via services like:

**Efabless Open MPW Shuttle**

**Google/Sky130 Program**

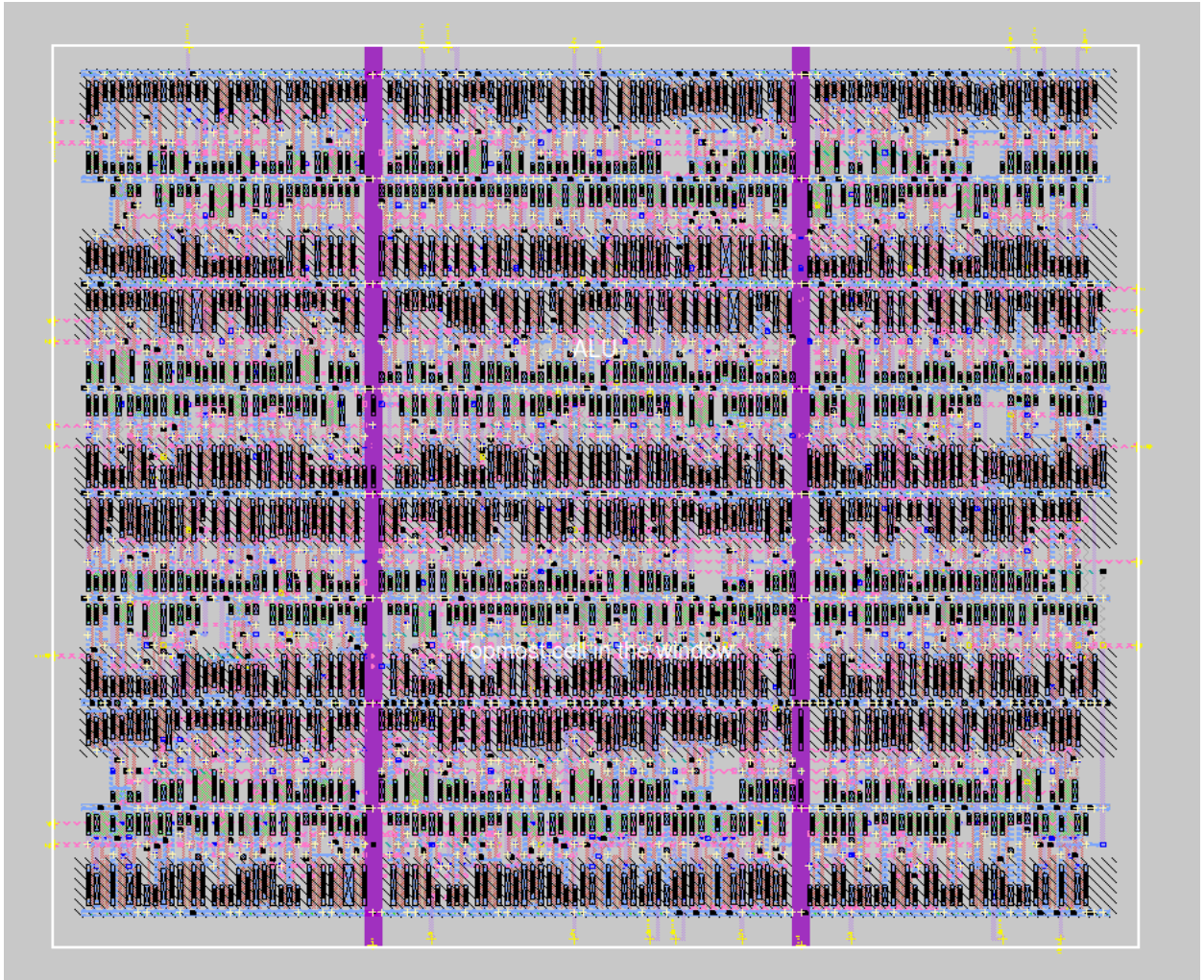
### **3. Integration into Larger Systems:**

The ALU GDSII or netlist can be integrated into a CPU or controller core for system-on-chip (SoC) design.

### **4. Reuse in Other Projects:**

The verified ALU design can be reused as a standalone IP core in future digital projects.

❖ **4. Final Layout:**



*Figure 14 Layout*

## ❖ 5. Conclusion:

The design and implementation of an **8-bit Arithmetic and Logic Unit (ALU)** using open-source EDA tools successfully demonstrate the complete digital VLSI design flow — from **Register Transfer Level (RTL)** design in Verilog to the **final GDSII layout**. This project highlights not only the functional aspects of digital design but also the practical workflow of physical design automation using tools such as **Yosys, Qflow, GrayWolf, Magic, and OpenSTA**.

By simulating the ALU at the RTL level and verifying it through synthesis, placement, routing, and timing analysis, we ensured that the design was both **functionally correct** and **physically realizable**. Open-source toolchains and the **Sky130 PDK** played a critical role in making this process transparent, cost-effective, and accessible for academic exploration.

This project not only deepened our understanding of **ALU architecture** and **digital design principles**, but also provided hands-on experience with **EDA workflows** and **layout tools** — essential skills for VLSI engineers. The final GDSII output of the design is ready for further fabrication steps or integration into more complex systems like processors or SoCs.

In conclusion, this project demonstrates how open-source design tools can empower students and researchers to contribute to silicon-level innovation with minimal resources, while gaining end-to-end experience in the VLSI design process.

# GUJARAT TECHNOLOGICAL UNIVERSITY

Subject Name: Internship/ Project Team ID: 741180

Subject Code: 3181101

## ❖ 6. References:

[1] Vsdflow Installation Support: {<https://www.vlsisystemdesign.com/vsd-a-complete-guide-to-install-open-source-eda-tools/>}

[2] GitHub Kunal Ghosh Sir: {<https://github.com/kunalg123/vsdflow>}

[3] Tool Installation Guide by ADCB Innovations:  
{<https://www.youtube.com/watch?app=desktop&v=tn64OU1GA7I>}

[4] Qflow Tutorial by ADCB Innovations:  
{[https://www.youtube.com/watch?app=desktop&v=oE\\_YnBhw\\_SY](https://www.youtube.com/watch?app=desktop&v=oE_YnBhw_SY)}

[5] Verilog Tutorial By NPTEL: {<https://www.youtube.com/watch?v=w3jNkZ-5s-U&list=PLwdnzlV3ogoVGq4TIpX4NH6QEFYiAnyvA>}