➢ FLOW DIAGRAM

The block diagram shown in Figure illustrates the complete workflow followed in the design, simulation, and verification of the Arithmetic Logic Unit (ALU) using Verilog and Python-based testbenches. This structured flow ensures a smooth integration of hardware description and verification using modern simulation tools.

Description of the Flow

1. **Environment Setup:**

The process begins with setting up a Linux-based development environment. This includes configuring the system for simulation and installing essential tools such as Python, Icarus Verilog, GTKWave, and Cocotb (a coroutine-based cosimulation library for writing Python testbenches).

2. **Project Initialization:**

A new folder is created to organize all project files, including the Verilog source code, Python testbench, Makefile, and simulation outputs.

3. **Verilog Code Development:**

The core ALU logic is written in Verilog (e.g., alu.v). This module includes the implementation of basic arithmetic and logic operations such as addition, subtraction, AND, OR, and XOR, controlled through select inputs.

4. **Testbench Creation:**

A Python testbench (alu_test.py) is developed using the Cocotb framework. This testbench generates input stimuli, drives them to the ALU, and verifies the correctness of the output by comparing it with expected values.

5. **Makefile Configuration:**

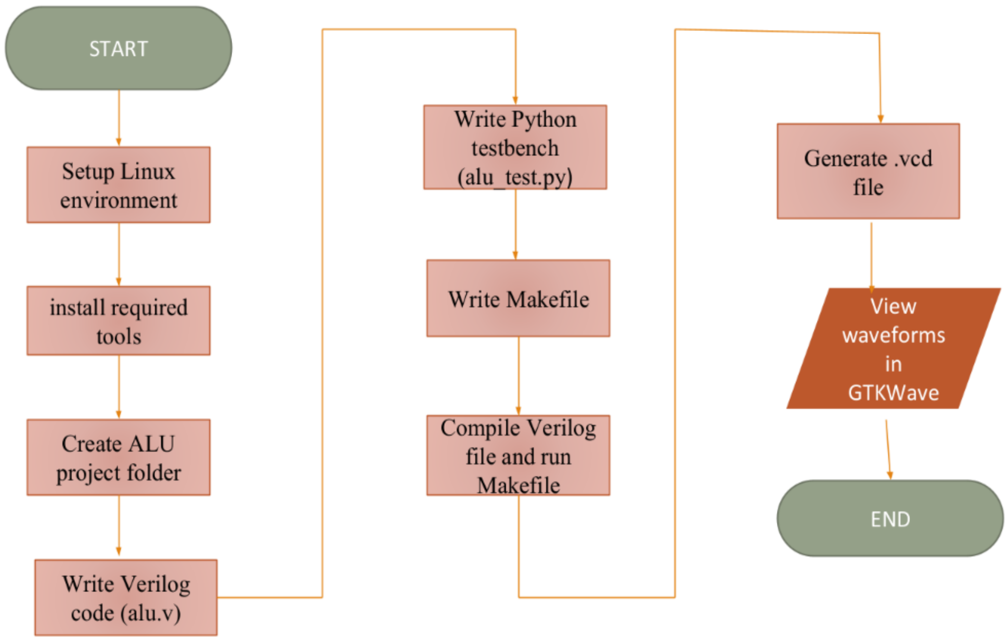
A Makefile is written to automate the compilation and simulation process. It includes instructions for running the Verilog compiler, launching Cocotb, and generating the output waveform file.

6. **Simulation Execution:**

The Verilog code is compiled and simulated using the Makefile. This step integrates the Python testbench with the Verilog design to perform cosimulation.

7. **Waveform Generation and Analysis:**

The simulation generates a .vcd (Value Change Dump) file that captures signal transitions. This file is loaded into GTKWave, a waveform viewer used to analyze the timing and behavior of the ALU under various test scenarios.

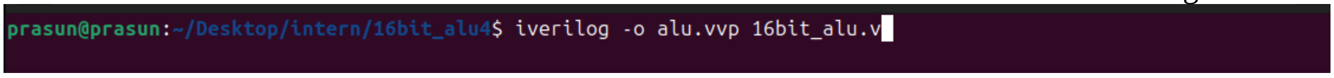

START
Setup Linux environment
install required tools
Create ALU project folder
Write Verilog code (alu.v)
Write Python testbench (alu_test.py)
Write Makefile
Compile Verilog file and run Makefile
Generate .vcd file
View waveforms in GTKWave
END

➢ Linux Terminal Commands Use:

This section provides the list of Linux terminal commands used throughout the project for compiling, simulating, and analyzing the ALU design. The project utilizes **Icarus Verilog** for simulation, **Cocotb** for testbench automation, and **GTKWave** for waveform analysis.

❖ Compile Verilog Design:

The following command was used to compile the Verilog source file (`16bit_alu.v`) using Icarus Verilog:

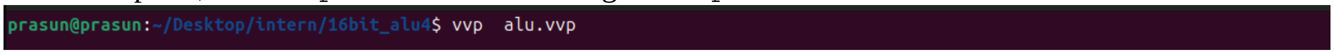


```
prasun@prasun:~/Desktop/intern/16bit_alu4$ iverilog -o alu.vvp 16bit_alu.v
```

- Iverilog -o is the Icarus Verilog compiler.
- alu.vvp specifies the output simulation file.
- 16bit_alu.v is the Verilog source file for the ALU

❖ Run the Simulation :

Once compiled, the `.vvp` file is executed using the `vvp` runtime command:

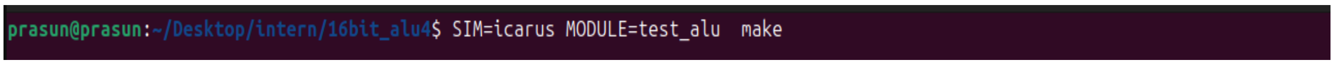


```
prasun@prasun:~/Desktop/intern/16bit_alu4$ vvp  alu.vvp
```

This step runs the compiled simulation and generates the `.vcd` (waveform) file if configured properly in the testbench.

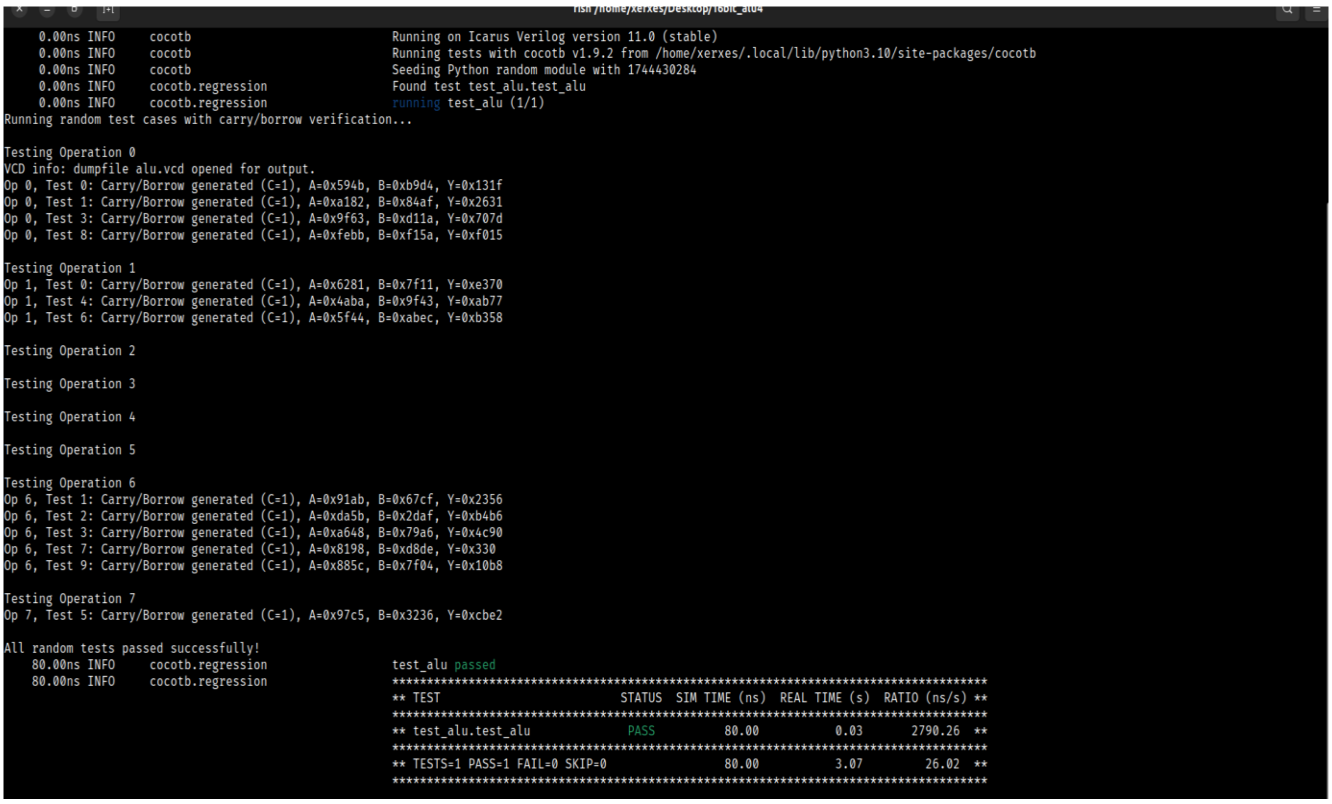
❖ Run Simulation with Cocotb (Makefile-driven) :

When using Cocotb for Python-based testbenches, the simulation was launched using:



```
prasun@prasun:~/Desktop/intern/16bit_alu4$ SIM=icarus MODULE=test_alu  make
```

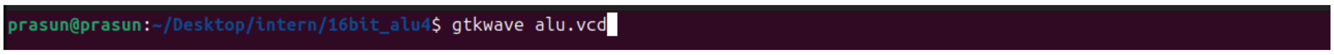
- SIM=icarus tells Cocotb to use Icarus Verilog.
- MODULE=test_alu specifies the Python testbench module name (test_alu.py).
- make invokes the Makefile that manages the simulation process.



```
     0.00ns INFO     cocotb                             Running on Icarus Verilog version 11.0 (stable)
     0.00ns INFO     cocotb                             Running tests with cocotb v1.9.2 from /home/xerxes/.local/lib/python3.10/site-packages/cocotb
     0.00ns INFO     cocotb                             Seeding Python random module with 1744430284
     0.00ns INFO     cocotb.regression                  Found test test_alu.test_alu
     0.00ns INFO     cocotb.regression                  running test_alu (1/1)
Running random test cases with carry/borrow verification...

Testing Operation 0
VCD info: dumpfile alu.vcd opened for output.
Op 0, Test 0: Carry/Borrow generated (C=1), A=0x594b, B=0xb9d4, Y=0x131f
Op 0, Test 1: Carry/Borrow generated (C=1), A=0xa182, B=0x84af, Y=0x2631
Op 0, Test 3: Carry/Borrow generated (C=1), A=0x9f63, B=0xd11a, Y=0x707d
Op 0, Test 8: Carry/Borrow generated (C=1), A=0xfebb, B=0xf15a, Y=0xf015

Testing Operation 1
Op 1, Test 0: Carry/Borrow generated (C=1), A=0x6281, B=0x7f11, Y=0xe370
Op 1, Test 4: Carry/Borrow generated (C=1), A=0x4aba, B=0x9f43, Y=0xab77
Op 1, Test 6: Carry/Borrow generated (C=1), A=0x5f44, B=0xabec, Y=0xb358

Testing Operation 2

Testing Operation 3

Testing Operation 4

Testing Operation 5

Testing Operation 6
Op 6, Test 1: Carry/Borrow generated (C=1), A=0x91ab, B=0x67cf, Y=0x2356
Op 6, Test 2: Carry/Borrow generated (C=1), A=0xda5b, B=0x2daf, Y=0xb4b6
Op 6, Test 3: Carry/Borrow generated (C=1), A=0xa648, B=0x79a6, Y=0x4c90
Op 6, Test 7: Carry/Borrow generated (C=1), A=0x8198, B=0xd8de, Y=0x330
Op 6, Test 9: Carry/Borrow generated (C=1), A=0x885c, B=0x7f04, Y=0x10b8

Testing Operation 7
Op 7, Test 5: Carry/Borrow generated (C=1), A=0x97c5, B=0x3236, Y=0xcbe2

All random tests passed successfully!
    80.00ns INFO     cocotb.regression                  test_alu passed
    80.00ns INFO     cocotb.regression                  ******************************************************************************
                                                        ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                                        ******************************************************************************
                                                        ** test_alu.test_alu              PASS          80.00           0.03       2790.26  **
                                                        ******************************************************************************
                                                        ** TESTS=1 PASS=1 FAIL=0 SKIP=0                 80.00           3.07         26.02  **
                                                        ******************************************************************************
```

- **View Simulation Results in GTKWave:**

To view the waveform output (`alu.vcd`), GTKWave was launched using:



```
prasun@prasun:~/Desktop/intern/16bit_alu4$ gtkwave alu.vcd
```

This command opens the waveform in GTKWave GUI, allowing users to analyze signal transitions and verify the correctness of the ALU operations.