16-bit ALU calculator

Andrea Grassi

26/02/2025

1 Introduction

This repository contains a testbench for a simple 16-bit calculator (simplified ALU) implemented in VHDL. The calculator supports basic arithmetic and logical operations, including addition, subtraction, AND, OR, NOT, XOR, multiplication, and shifting. This testbench simulates various operations to verify the correct functionality of the calculator and includes tests for overflow detection.

2 Features

- **Arithmetic Operations**: Addition ('+'), Subtraction ('-'), Multiplication ('*')
- **Logical Operations**: AND, OR, XOR
- **Unary Operations**: NOT for both inputs ('in1' and 'in2')
- **Overflow Detection**: Overflow detection for arithmetic operations like addition, subtraction, and multiplication
- **Shifting**: Left and right shift operations (logical shifts)
- **Zero Detection**: Output signal to indicate if the result is zero
- **Various Test Cases**: Tests for different combinations of inputs, including edge cases like overflow and zero results

3 Entity and Architecture Overview

3.1 Entity `tb_alu`

The `tb_alu` entity is a testbench for the ALU component (`calculator`) and does not have any ports.

3.2 Architecture `testbench`

The `testbench` architecture includes signals to stimulate the ALU and capture its outputs. It uses a `process` to apply various test vectors to the ALU and verify the results. The architecture connects to the `calculator` component and performs a series of tests to verify the correct behavior of all operations.

4 Signals

- **in1_tb**: 16-bit input vector for the first operand
- **in2_tb**: 16-bit input vector for the second operand
- **sig_tb**: A 4-bit control signal that determines which operation to perform:
  - `0`: Addition ('+')
  - `1`: Subtraction ('-')
  - `2`: AND
  - `3`: OR
  - `4`: NOT for `in1`
  - `5`: NOT for `in2`
  - `6`: XOR
  - `7`: Right shift
  - `8`: Left shift
  - `9`: Multiplication
- **result_tb**: 16-bit output vector to store the result of the operation
- **overflow_tb**: A signal indicating if there was an overflow during an arithmetic operation

5 Key Test Cases

- **Addition**: Tests basic addition, including cases with negative numbers and overflow detection.
- **Subtraction**: Tests subtraction, including negative and positive results, as well as overflow conditions.
- **AND Operation**: Tests bitwise AND operation between the two input operands.
- **OR Operation**: Tests bitwise OR operation between the two input operands.

- **NOT Operation**: Tests bitwise NOT operation applied to the first or second input operand.
- **XOR Operation**: Tests bitwise XOR operation between the two input operands.
- **Shifting**: Tests logical right and left shifts, checking if the bits are correctly shifted and handling edge cases.
- **Multiplication**: Tests multiplication with various combinations of positive and negative numbers, and checks for overflow.

6 How to Run the Testbench

1. Ensure you have a VHDL simulator installed (e.g., ModelSim, Xilinx Vivado, or GHDL).
2. Compile the ALU entity (`calculator`) and the testbench (`tb_alu.vhdl`).
3. Run the simulation to see the results of the test cases.

7 Simple Script Execution

To facilitate the compilation, simulation, and cleanup of temporary files, two scripts are included: `run_alu.sh` and `clear_files.sh`.

7.1 Compilation and Simulation: `run_alu.sh`

This script automates the process of compiling and simulating the ALU project, and it also handles the visualization of the results via GTKWave.

7.2 How to Execute

1. Make the scripts executable giving them permissions:

```
chmod +x run_alu.sh
chmod +x clear_files.sh

./clear_files.sh
./run_alu.sh
```

8 Author

This project was developed by Andrea Grassi, for educational purposes for Embedded Systems exam.