Assignment 3 (Programming Part) - 50%

Updated on 2024/10/27: Instructions for installing GTKWave on macOS have been revised.

Updated on 2024/10/31: Added module name requirements and clarified the behavior of addition and subtraction when results exceed the range of a 32-bit integer.

Updated on 2024/11/21: Revised the descriptions of **ALU Design** and **RISC-V Register File Design** for improved clarity.

In this assignment, you will set up a Verilog development environment, learn the basics of Verilog programming, and implement an Arithmetic Logic Unit (ALU) and a RISC-V register file. We will use Icarus Verilog as our simulator and GTKWave for waveform viewing.

Please be familiar with Verilog in this assignment, as you'll be building a processor that supports a subset of RISC-V instructions in the next one.

Setting Up the Verilog Environment

We will use **Icarus Verilog** for simulation and **GTKWave** for waveform viewing. Follow the setup instructions for your operating system below:

Linux

Most Linux distributions include **Icarus Verilog** and **GTKWave** in their package repositories.

- **Debian-based distributions:** Install using:

  ```
  sudo apt-get install iverilog gtkwave
  ```

- **Fedora-based distributions:** Install using:

  ```
  sudo dnf install iverilog gtkwave
  ```

For other distributions, refer to the documentation or consider building from source.

Windows

Download prebuilt **Icarus Verilog** and **GTKWave** binaries here.

Alternatively:

- Install **MSYS2** and use its package manager to install **Icarus Verilog** and **GTKWave**.
- Use **Windows Subsystem for Linux (WSL2)** and install **Icarus Verilog** and **GTKWave** within WSL2.

macOS

Icarus Verilog and **GTKWave** are available through Homebrew. If you don't have Homebrew, install it here.

To install, use these commands:

```
brew install icarus-verilog
brew install yanjiew1/gtkwave/gtkwave
```

Verilog Tutorial and Resources

If you're not familiar with Verilog, you may find the following resources helpful:

- Supplementary resources by previous TA
- HDLBits Verilog Exercises
- ASIC World's Verilog Tutorial
- Icarus Verilog & GTKWave Tutorial

Implementing an ALU and a RISC-V Register File

Follow the guidelines for each module carefully, using the exact port names specified below. Port names are case-sensitive, so ensure accuracy. We will grade your implementation using our testbenches, and any name mismatches will cause the tests to fail.

ALU Design

The Arithmetic Logic Unit (ALU) performs arithmetic and bitwise operations on two 32-bit inputs based on a 3-bit control signal (`ALUCtrl_i`). The design includes functionality for addition, subtraction, bitwise operations, and various shifts.

Port Definitions

Port Name	Bits	I/O	Description
`data1_i`	32	Input	First input data
`data2_i`	32	Input	Second input data
`ALUCtrl_i`	3	Input	Control signal specifying the operation

`data_o`	32	Output	Result of the ALU operation
`zero_o`	1	Output	Flag indicating if the result is zero

Design Description

1. **Operations:**
   The ALU supports the following operations based on the value of `ALUCtrl_i`:

`ALUCtrl_i`	Operation	Description
`000`	Addition	`data_o = data1_i + data2_i`
`001`	Subtraction	`data_o = data1_i - data2_i`
`010`	Bitwise AND	`data_o = data1_i & data2_i`
`011`	Bitwise OR	`data_o = data1_i | data2_i`
`100`	Bitwise XOR	`data_o = data1_i ^ data2_i`
`101`	Left Shift	`data_o = data1_i << (data2_i[4:0])`
`110`	Arithmetic Right Shift	`data_o = data1_i >>> (data2_i[4:0])`
`111`	Logical Right Shift	`data_o = data1_i >> (data2_i[4:0])`

2. **Zero Flag (`zero_o`):**
   - `zero_o` is set to `1` if the result (`data_o`) is zero.
   - Otherwise, `zero_o` is set to `0`.

3. **Overflow Behavior:**
   - For addition and subtraction, results exceeding the range of a 32-bit signed integer wrap around, following the behavior of unsigned integers in C. For example, `0xFFFFFFFF + 1` should result in `0x00000000`.

4. **Shift Operations:**
   - The shift amount is determined by the lower 5 bits of `data2_i` (`data2_i[4:0]`).
   - **Arithmetic Right Shift:** The leftmost bits should be sign-extended to preserve the sign of the original value. For example, shifting `0xF0000000` right by 1 arithmetically produces `0xF8000000`.

Module Name and Implementation File

- **Module name:** `ALU`
- **File name:** `ALU.v`

The implementation must strictly adhere to the provided port definitions and operation specifications to ensure compatibility with testbenches.

RISC-V Register File Design

The RISC-V register file consists of 32 registers, each 32 bits wide. **Register 0 is hardwired to zero**, meaning its value is always zero and cannot be modified by any write operation. The register file is designed to support simultaneous reading from two registers and writing to one register.

Port Definitions

Port Name	Bits	I/O	Description
`clk_i`	1	Input	Clock signal
`RS1addr_i`	5	Input	Address of the first source register
`RS2addr_i`	5	Input	Address of the second source register
`RDaddr_i`	5	Input	Address of the destination register
`RDdata_i`	32	Input	Data to be written
`RegWrite_i`	1	Input	Write enable signal
`RS1data_o`	32	Output	Data from the first source register
`RS2data_o`	32	Output	Data from the second source register

Design Description

1. **Read Operations:**
    - Reads from `RS1addr_i` and `RS2addr_i` occur asynchronously and are not dependent on the clock.
    - `RS1data_o` outputs the content of the register specified by `RS1addr_i`, while `RS2data_o` outputs the content of the register specified by `RS2addr_i`.
    - Both `RS1data_o` and `RS2data_o` must always reflect the current values of the registers specified by `RS1addr_i` and `RS2addr_i`.

2. **Write Operation:**

- When `RegWrite_i` is high, the value of `RDdata_i` is written to the register specified by `RDaddr_i` on the rising edge of `clk_i`.

3. **Special Case for Register 0:**
    - Register 0 is always hardwired to zero.
    - Any attempt to write to register 0 (i.e., when `RDaddr_i` is 0) should be ignored, and its value must remain zero at all times.

Module Name and Implementation File

- **Module name:** `Registers`
- **File name:** `Registers.v`

The implementation should strictly follow the provided port definitions and functionality to ensure compatibility with the provided testbenches.

Report Guidelines

Submit a report in **Markdown** format, viewable in **Visual Studio Code**. The report may be written in either Chinese or English and should include:

1. A description of each module's design.
2. A description of your testing process, including:
    - Commands used to run tests.
    - Descriptions of your testbenches or any other testing scripts.
    - Test results, including screenshots or waveform images where applicable.

Submission Guidelines

Submit a **zip** file named `<Student_ID>_HW3.zip` via NTU COOL, following the directory structure below:

```
<Student_ID>_HW3
├── ALU.v
├── Registers.v
├── report.md
├── <testbenches or scripts used for testing>
└── <any additional files referenced in report.md>
```

Please make sure the directory `<Student_ID>_HW3` is included in the zip file. Replace `<Student_ID>` with your student ID, using uppercase letters.

Policy

- **Plagiarism is strictly prohibited.** Copying or sharing work will result in zero points for this assignment.
- **AI-generated content is not allowed.** Submitting AI-generated code will also result in zero points.