Lab 1: 32-bit ALU & Register File

Computer Organization 2024

1. Goal

ALU (Arithmetic Logic Unit) is the basic computing component of CPU. It takes in two 32-bit source and output one 32-bit result. The arithmetic operation includes bitwise AND/OR/NOR and addition/subtraction/set-less-than of signed integer. Register File is a pack of registers that acts like a tiny piece of memory which is inside CPU, providing fast access to data during computation.

The main goal of this lab is to implement a 32-bit ALU and a MIPS register file. This lab will help you understand:

1. How ALU works and how to implement it.
2. The basic concepts of writing combinational logic in Verilog.
3. The basic concepts of simple testbench in SystemVerilog.
4. How to implement register file in Verilog (details about how addressing will not be covered).
5. The basic concepts of writing sequential logic together with combinational logic in Verilog.

Note: Future labs will be highly related to previous ones, so please try your best to complete every lab correctly.

2. Description

We recommend you to complete this lab by:

1. Read textbook *"Appendix C.4 Using a Hardware Description Language"* to get familiar with Verilog syntax.
2. Reference textbook *"Appendix C.5 Constructing a Basic Arithmetic Logic Unit"* when sequentially complete the code provided:
    1. `bit_alu.v` 1-bit ALU
    2. `msb_bit_alu.v` 1-bit ALU for the most significant bit
    3. `alu.v` 32-bit ALU

    There's a lot of instruction and guidance in the code
3. Test your design using `tb_alu.sv` and `tb_alu.0.txt` . We strongly recommend you to read the testbench code since you might need to debug by your own in other labs.
4. Implement the register file in `reg_file.v` . Be aware how to handle clock edges since it is critical for later labs.
5. Test your design using `tb_reg_file.sv` . We strongly recommend you to read the testbench code since sequential logic is harder to test properly.

(50%) ALU Operations

Your ALU must be **combinational** logic and output `result` according to these operations:

Operation	ALU Control	Function
AND	`0000`	bitwise AND
OR	`0001`	bitwise OR
ADD	`0010`	addition of signed integers
SUB	`0110`	subtraction of signed integers
NOR	`1100`	bitwise NOR
SLT	`0111`	set `result=1` if the first signed integer is less than the second one. Otherwise, `result=0` . (Set-Less-Than)

Hint: ALU Control = {A_invert, B_invert, operation}

(30%) Zero & Overflow Detection

When result and operation meets certain conditions, ALU should set some flags.

1. `zero` must be set when the result equals to zero.
2. `overflow` must be set if and only if overflow occurs during ADD/SUB.

Verilog module must be named `alu` within file `alu.v` , and the I/O ports must be defined as follows:

```
module alu (
    input  [31:0] a,        // 32 bits, source 1 (A)
    input  [31:0] b,        // 32 bits, source 2 (B)
    input  [ 3:0] ALU_ctl,  // 4 bits, ALU control input
    output [31:0] result,   // 32 bits, result
    output        zero,     // 1 bit, set to 1 when the output is 0
    output        overflow  // 1 bit, overflow
);
```

Although we provide sample code, you can still implement by your own style as long as it meets the I/O port definition and can pass the testbench.

(10%) Report

1. **Architecture Diagrams**

   Show your 1-bit & 32-bit ALU design by "Schematic" tool in Vivado.

2. **Answer the following Questions**

   1. How `overflow` is calculated?
   2. Explain why ALU control signal of SUB is `0110` and NOR is `1100` ?
   3. (2 cont'd) If you assign different signal to these operation, what problems you may encountered?

    4. True or false: Because the register file is both read and written on the same clock cycle, any MIPS datapath using edge-triggered writes must have more than one copy of the register file. Explain your answer.
3. **Experimental Result** (ALU Only)
    1. Show the waveform screen shot of the testbench `tb_alu.0.txt` result.
    2. What other cases you've tested? Why you choose them?
4. **Problems Encountered & Solution** (optional)

    List some important problem you've met during this lab and there solution.
5. **Feedback** (optional)

    Any thing you want to say to TA team about this lab. How can we improve the lab?

(10%) Register File

The register file has the following I/O.

```
module reg_file (
    input         clk,          // Clock
    input         rstn,         // Negative Reset
    input  [ 4:0] read_reg_1,   // Read Register 1 (address)
    input  [ 4:0] read_reg_2,   // Read Register 2 (address)
    input         reg_write,    // RegWrite: write data when posedge clk
    input  [ 4:0] write_reg,    // Write Register (address)
    input  [31:0] write_data,   // Write Data
    output [31:0] read_data_1,  // Read Data 1
    output [31:0] read_data_2   // Read Data 2
);
```

Check the description in the textbook by your own (FIGURE 4.7).
Since lab 1 consumed less time to complete, we recommend you to read textbook 4.1 & 4.2 before you implement the register file. These sections contains basics concepts which are useful in the later labs.

The purpose of `rstn` is to clear the content of register file because we want to make sure every register should be zero when the computer boots up.

The testbench will test both combinational and sequential logic of your design's output during different phases of each clock cycle.

3. Submission

Your submission must be a zip file named `Lab1_ID.zip` where `ID` is your student ID, and structured as below:

```
Lab1_123456789.zip      # There should be NO sub-directory e.g. Lab1_123456789/
├── Lab1_123456789.pdf  # Report (Must be PDF)
└── src                 # contains your source code
    ├── alu.v           # Must included
    ├── bit_alu.v
    ├── msb_bit_alu.v
    └── reg_file.v      # Must included
```

There should be only one module per `.v` file, and the module name should be the same as file name.
The report should be named `Lab1_ID.pdf` and we ONLY accept PDF, any other format will not be scored.
If you want to use System Verilog, the filename must end with `.sv` .

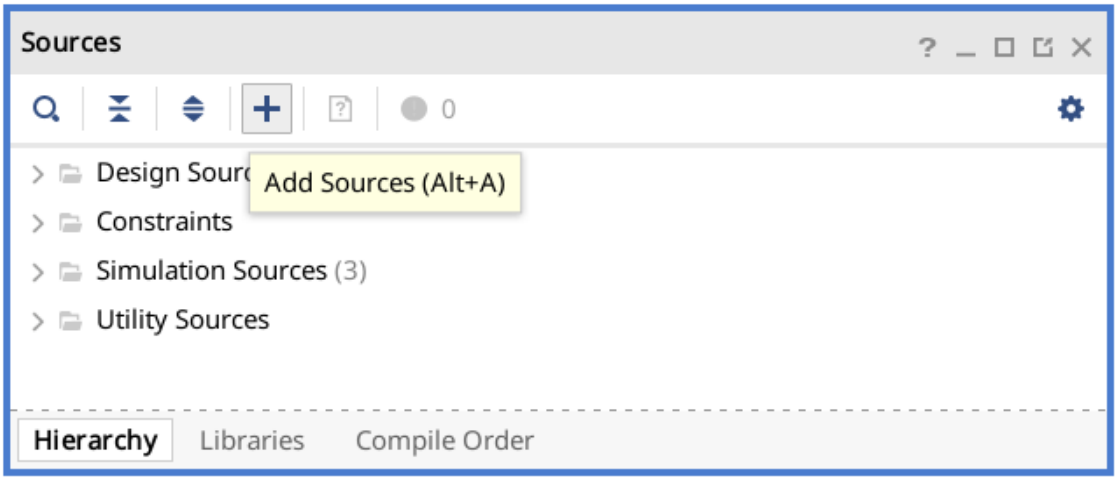
Before you submit, make sure to pass the testbenches we provided.

!! Any Plagiarism is NOT allowed !!
Any late submission gets only 80% of original score.
Any submission after E3 window is closed will not be accepted!

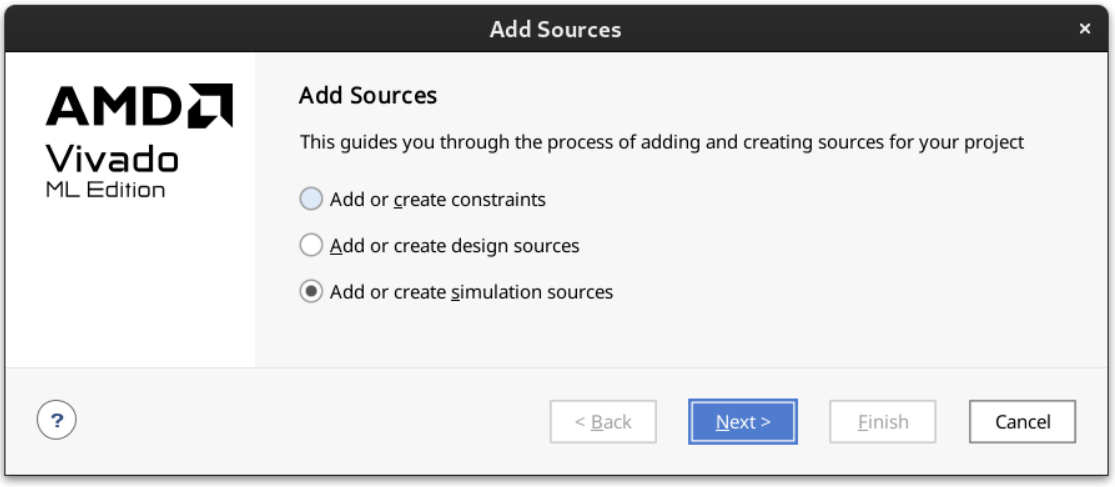
Appendix: Some Vivado Tips

Add Test Files (.txt)

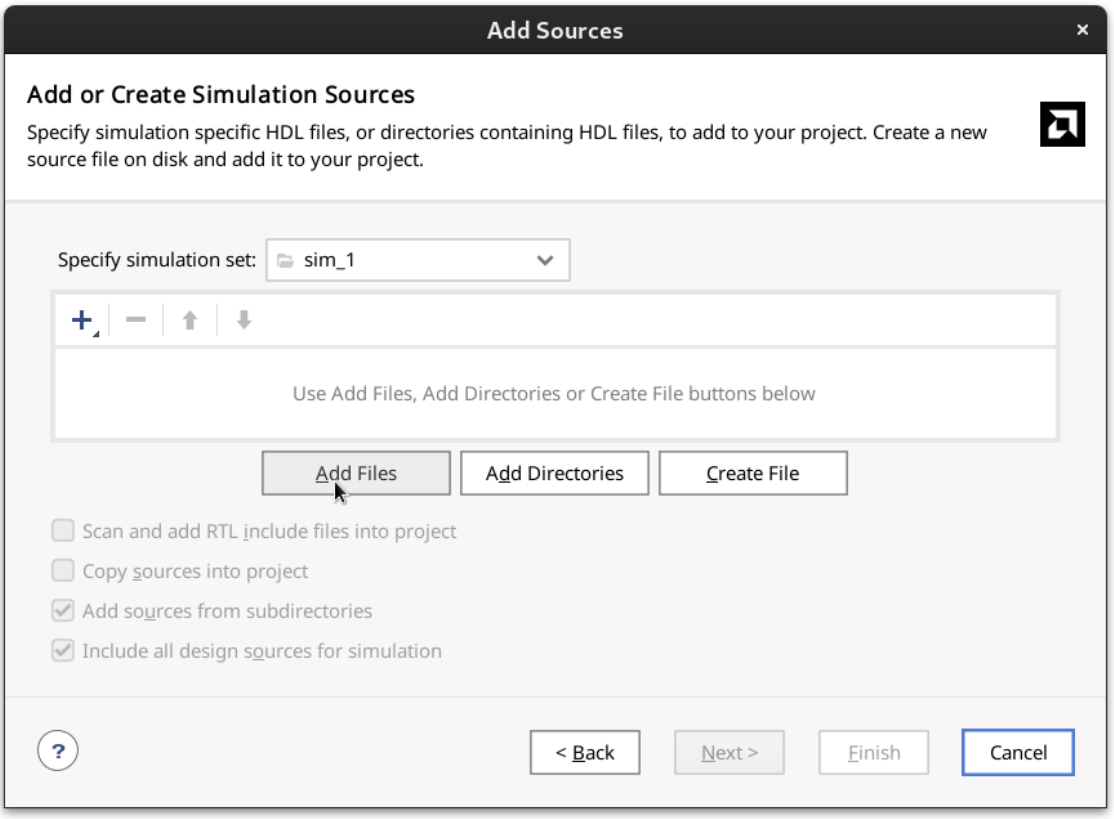
1. in "PROJECT MANAGER", press "Add Sources".



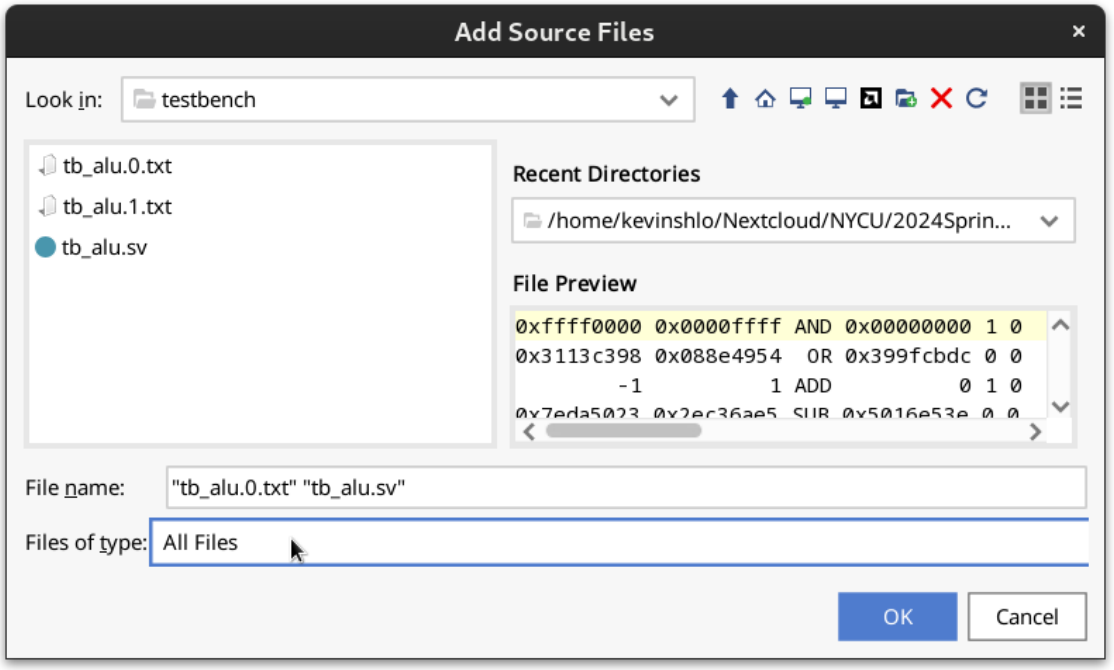
2. Select "Add or create simulation sources" in the pop-up window.



3. Press "Add Files"

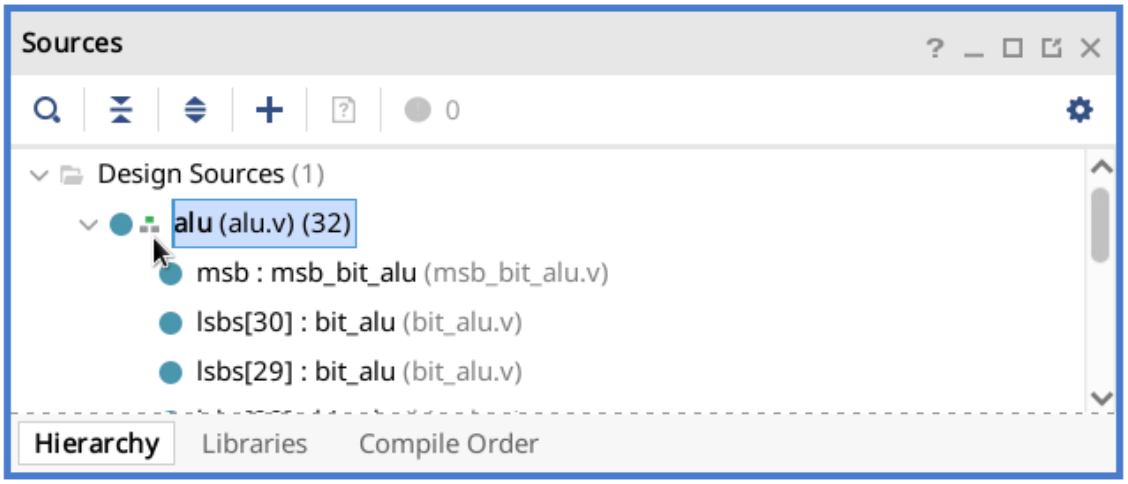


4. Navigate to the source file directory, select "Files of type" to "All Files" & select files...

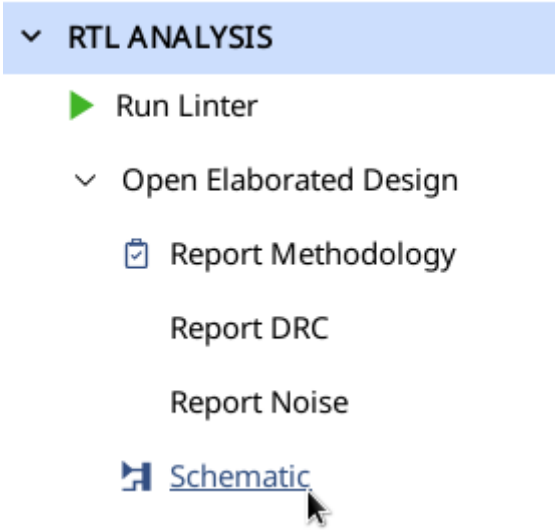


Schematic Tool

1. Remember to set the desired module to "Top module" (green box icon)



2. Select "Schematic" in the sidebar



3. Schematic will then show in tabs