Lab 1: ALU with Arbiter

Submission Due Dates:

Demo: 2022/09/20 17:20
Source Code: 2022/09/20 18:30
Report: 2022/09/25 23:59

Objective

Getting familiar with basic Verilog behavior modeling.

Action Items

1 lab1_1.v (20%)

Write a Verilog module that models a **4-bit arbiter** and test your module using the testbench file **lab1_1_t.v**, Figure 1 is the block diagram of **lab1_1**.

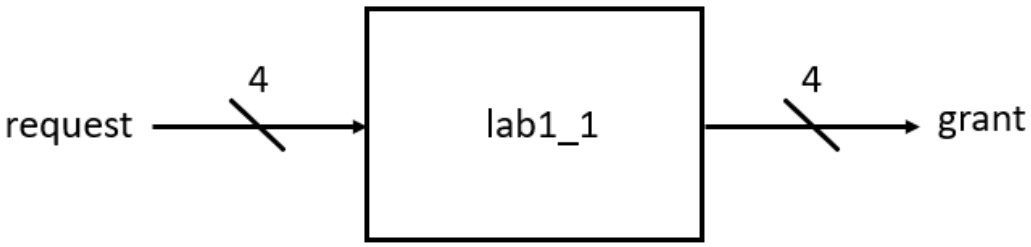


Figure 1: lab1_1 block diagram

a. IO list:

- ✓ Input: **`request[3:0]`**

  A 4-bit input vector: the i-th bit indicates the request from the i-th port, where i=0..3.

  Example:

request	Description
4'b0101	Port 0 and port 2 have requests
4'b1000	Only port 3 has a request
4'b0000	No one has a request

- ✓ Output: **`grant[3:0]`**

  A 4-bit one-hot vector outputs the grant based on following rule.

  - If one port has a request, it will be granted (e.g., if the second port has a request, the **request** = 4'b0100. The **grant** should be 4'b0100 to grant this only request).
  - If there are multiple requests, you should grant the port with the largest index. For example, if ports 0, 1, and 3 have requests at the same time, the **request** = 4'b**1**011. Then port 3 will be granted with the **grant** = 4'b**1**000.
  - If there is no request, such that the **request** = 4'b0000, the **grant** should be

4'b0000.

Note: Please use Verilog *if-else* statement to implement this module, *"case" or "casex"* is not allowed

Example:

Input: **request**[3:0]	Output: **grant**[3:0]
4'b0101	4'b0100
4'b1000	4'b1000
4'b0000	4'b0000
4'b0011	4'b0010

b. You must use the following template for your design:

```
`timescale 1ns/100ps
module lab1_1 (
  input wire [3:0] request,
  output reg [3:0] grant
);
  /* Note that grant can be either reg or wire.
   * e.g.,    output reg [3:0] grant
   * or       output wire [3:0] grant
   * It depends on how you design your module. */
  // add your design here

endmodule
```

2 lab1_1_t.v (20%)

Complete the testbench **lab1_1_t.v** to verify your design. Check the TODO hints in the template code carefully. For incorrect results, you may see an error message like this:

Error: request = XXXX, grant = XX, correct grant should be XX

where X is the corresponding data bit.

3 lab1_2.v (30%)

Write a Verilog module that models a **4-bit Arithmetic Logic Unit (ALU)** and test your module using the testbench **lab1_2_t.v**. The **4-input arbiter**, **lab1_1**, must be reused to do the arbitration**.** Figure 2 is the block diagram of **lab1_2**.

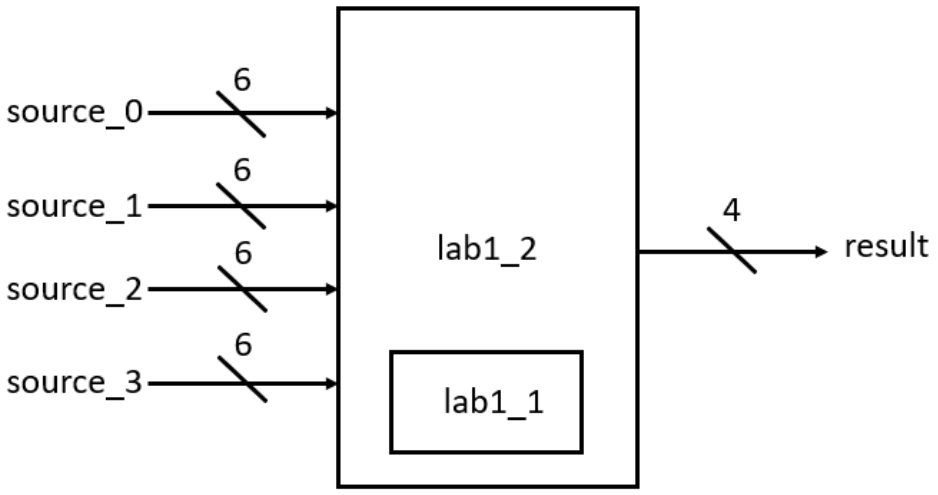


Figure 2: lab1_2 block diagram

a. IO list:

✓ Inputs: **`source_0[5:0]`**, **`source_1[5:0]`**, **`source_2[5:0]`**, **`source_3[5:0]`**

Here are four 6-bit source ports **source_X**. The format of each port is listed below:

Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Control		**Data**			

- The **Control** of each source port indicates whether the port sends a request, and also the operation on its **Data**. The **Control** signal's function is listed below:

Control signal (Bit 5:4)	Function
2'b00	No request
2'b01	Request; masking with 4'b1010 result = **Data** **AND** 4'b1010
2'b10	Request; increment by 3 (ignoring the overflow) result = **Data** + 4'd3
2'b11	Request; local-left-shifting by two bits result = **Data** << 2

✓ Output: **`result[3:0]`**

A 4-bit vector to output the result, based on the following rules.

- There may be multiple source ports having requests at the same time, **but only the port with the greatest index will be granted**. To determine which port gets the grant, you must reuse the **lab1_1** module to do the arbitration and generate the **result**. E.g., if multiple source ports send their requests, only the granted port produces the **result** based on its **Control** and **Data**.

- If there is no request from **source_X**, set the **result** to 0.

Note: Please use Verilog "*case*" to implement this module, *if-else* statement is prohibited.

b. You must use the following template for your design:

```
`timescale 1ns/100ps
module lab1_2 (
  input wire [5:0] source_0,
  input wire [5:0] source_1,
  input wire [5:0] source_2,
  input wire [5:0] source_3,
  output reg [3:0] result
);
  /* Note that result can be either reg or wire.
  * It depends on how you design your module. */
  // add your design here
endmodule
```

c. Follow the **Appendix** to extend the simulation time and add the test patterns.

4 lab1_2_t.v (30%)

Complete the testbench **lab1_2_t.v** to verify your design. Check the TODO hints in the template code carefully. For incorrect results, you may see an error message like this:

Error: source_0 = XX, source_1 = XX source_2 = XX, source_3 = XX your result = XX, correct result = XX.

where X are the corresponding data bits.

5 Questions and Discussion

Please answer the following questions in your report.

A. In the testbench **lab1_1_t.v**, please explain why we place **#DELAY** between input assignment and output verification. Hint: Gate delay.
B. In the testbench **lab1_1_t.v,** does the Verilog keyword **`function`** synthesizable? If yes, what kind of circuit will it be?
C. Please explain the difference between **if-else** and **case** statements after the synthesis at the gate/cell-level. Hint: priority encoder.

6 Guidelines for the report

Your report should include but not limit to the following items.

Grading policy (subject to change): Part (A): **35%**; Part (B): **50%**; Part (C): **10%**; (D): **5%**

A. **Lab Implementation**
   You may elaborate the followings.
   1. Block diagram of the design with explanation
   2. Partial code screenshot with the explanation: you don't need to paste the entire code into the report. Just explain the kernel part.
   3. Event-based finite state machine (FSM) with the explanation:
      Note1: Use events to describe the transition of FSM but not signals logic statement.
      Note2: You don't need to do this part if there is no FSM in this lab.

B. **Questions and Discussions**
   Provide your answer to the Questions and Discussions in the lab assignment.

C. **Problem Encountered**
   Describe the problems you encountered, solutions you developed, and the discussion. Explaining them with code segments or diagrams is recommended.

D. **Suggestions**
   Any suggestions for this course are more welcome.
   (If not, you may also post a joke. A funny one, please. It is not mandatory and is nothing to do with the grading. But it would undoubtedly amuse us. ☺)

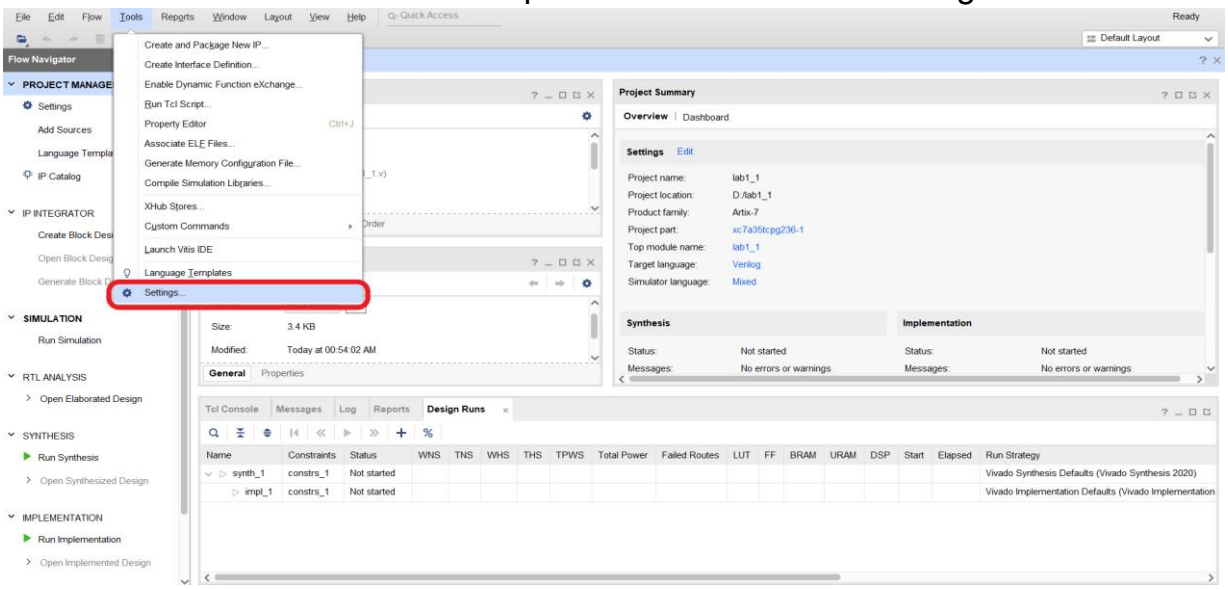
Attention

- ✓ DO NOT copy-and-paste code segments from the PDF materials. It may also paste invisible non-ASCII characters, leading to hard-to-debug syntax errors.
- ✓ You should hand in four source files, including **lab1_1.v, lab1_1_t.v, lab1_2.v, lab1_2_t.v**. Upload each source file individually. DO NOT hand in any compressed ZIP files, which will be considered an incorrect format.
- ✓ **lab1_2.v** must include the module of **lab1_1**. That is, you should copy lab1_1 into **lab1_2.v**. Otherwise, you will get a penalty of 20 points.
- ✓ We will use **hidden test patterns** to test your design in this lab.
- ✓ You should also hand in your report as **lab1_report_StudentID.pdf** (i.e., lab1_report_108456789.pdf).
- ✓ You should be able to answer questions of this lab from TA during the demo.
- ✓ You may also add a **$monitor** in the testbench to show all the inputs and outputs during the simulation.

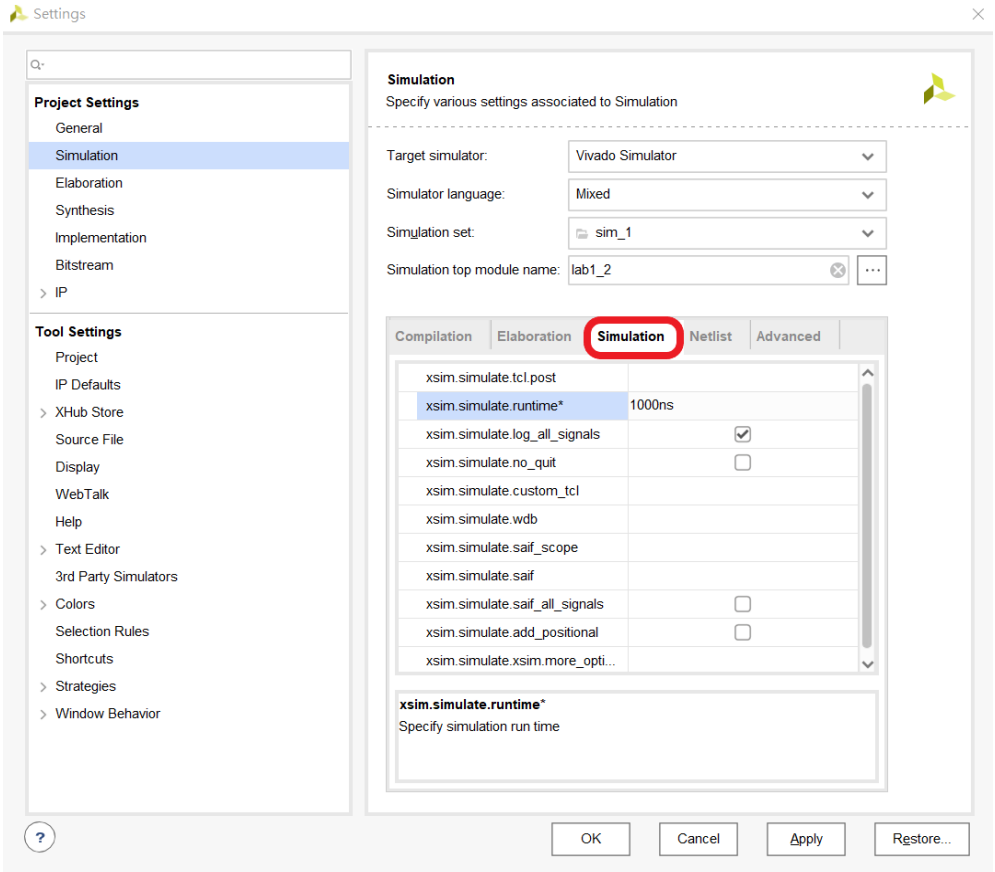
Appendix

- ✓ Before the simulation, you need to change the runtime to 10000ns (or a large enough period) in "Simulation Settings". See the guide below.

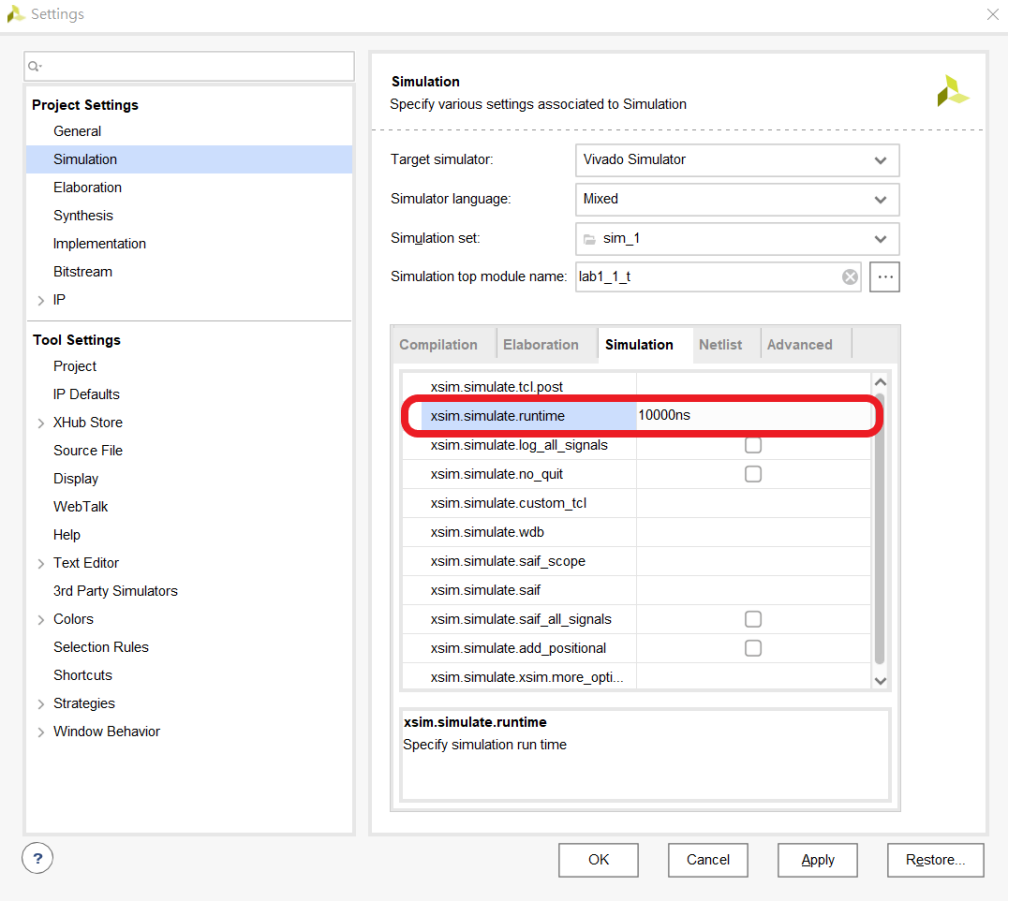
1. Click Tools at the top of Vivado and select settings.



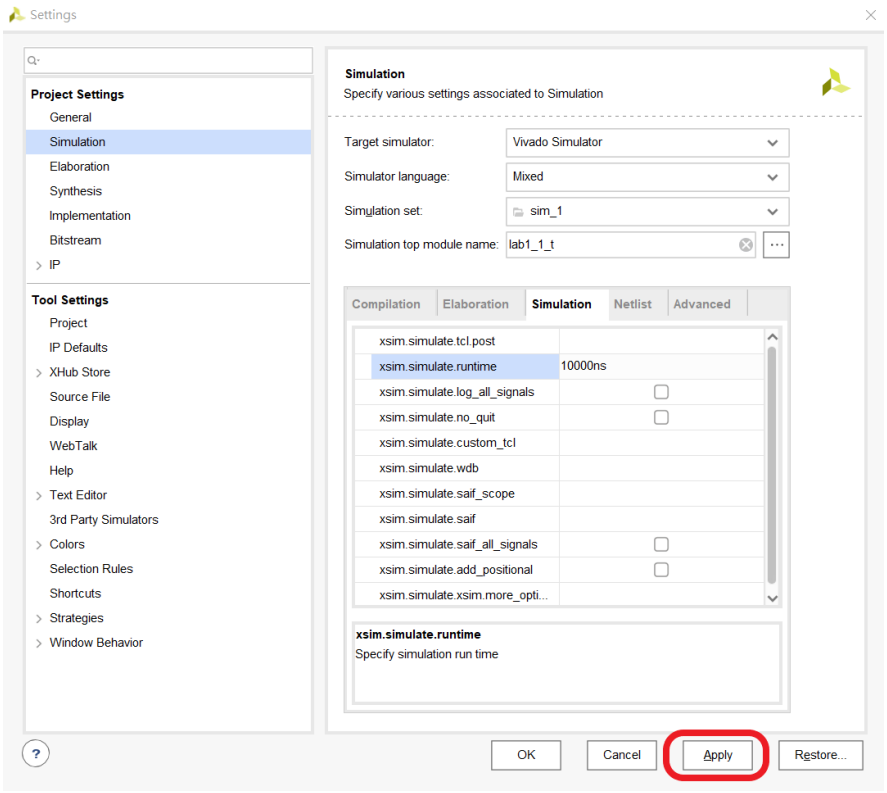
2. Select Simulation.



3. Change runtime to 10000ns or a larger value.



4. Remember to click Apply.



✓ In lab1_2, testing exhausted (all possible) patterns will be time-consuming. So, we provide a public test set of 2^8 patterns with the file of public.dat. Note that we will use

additional hidden test patterns to verify your design.