Lab 1: Gate-Level Designs

Submission Due Dates: 2025/9/9
OJ Submission: 16:30
Demo: 17:30

Instructions

1. There are **4 design problems** in this lab with a PDF specification of eight pages in total.
2. Download the ***.cpp** files from OJ. Change them to ***.zip** and decompress them. (Skip the *.h files. The OJ enforces that there must be a *.h file.)
3. Submit each Verilog file to its corresponding OJ problem.
   a. It is your responsibility to ensure the submission is successful.
   b. The module names should be **lab1_1**, **lab1_1_t**, **lab1_2**, and **lab1_2_t**.
4. You must **submit your source code** to OJ before **16:30** in order to receive a score for this lab.
5. To demo your design, **write down your student ID on the whiteboard** and wait for a TA to come to your seat. You may only demo **twice**, and your highest score will be considered for grading.
6. Before leaving, remember to **claim your FPGA and sign the attendance sheet.**

Objective

Getting familiar with basic Verilog gate-level modeling for combinational/sequential circuits.

Action Items

1 `lab1_1.v` (40%)

Design a function selector using gate-**level** modeling. It should include **a 4-to-1 multiplexer constructed from three 2-to-1 multiplexers.** A 2-bit select signal **c** = {**c1**, **c0**} determines which operation is performed on inputs **a** and **b**.

Select signal (c)	Operation
00	AND
01	OR
10	XOR
11	NAND

For example, when (a, b) = (1, 0):
Output = 0 if c = 00
Output = 1 if c = 01, 10, 11

a. I/O list:

Signal Name	I/O	Definition
a	Input	The first operand
b	Input	The second operand

c	Input	The 2-bit select signal
out	Output	The final output of the multiplexer

b. Note that only **gate-level** descriptions (e.g., **AND**, **OR**, **NAND**, **NOR**, **NOT, …**) are permitted. **Behavior-level** constructions such as **if-else** statements are not allowed in practice_1.v.

c. You may use the **mux** module in the template file to implement 2-to-1 multiplexers.

d. You must use the following I/O definition for your design. Remember to remove the blue-colored comments:

```
`timescale 1ns/100ps
module lab1_1 (
  input wire a,
  input wire b,
  input wire [1:0] c,
  output wire out
);
  // Write your code here
endmodule
```

e. Grading

Design Approach	**Score (Each Test Case)**
Use **only basic gates** (AND, OR, XOR, NAND, NOR, NOT) and **implement with three 2-to-1 MUXs**	5%
Use gate-level description only, but **not using three 2-to-1 MUXs**	5% * 0.6 = 3%
Use a **non-gate-level description**	0 %

2 lab1_1_t.v (15%)

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 error messages like this:

```
Error: a=X, b=X, c=X, out=X
Correct answer should be: out=X
```

where X is the corresponding data bit.

3 lab1_2.v (30%)

Write a Verilog module that models a **rising edge-triggered T Flip-Flop** using the gate-level description only.

a. I/O List:

Signal Name	I/O	Definition
clk	Input	Clock input
T	Input	Toggle input
Q	Output	Output
Qn	Output	Complementary output

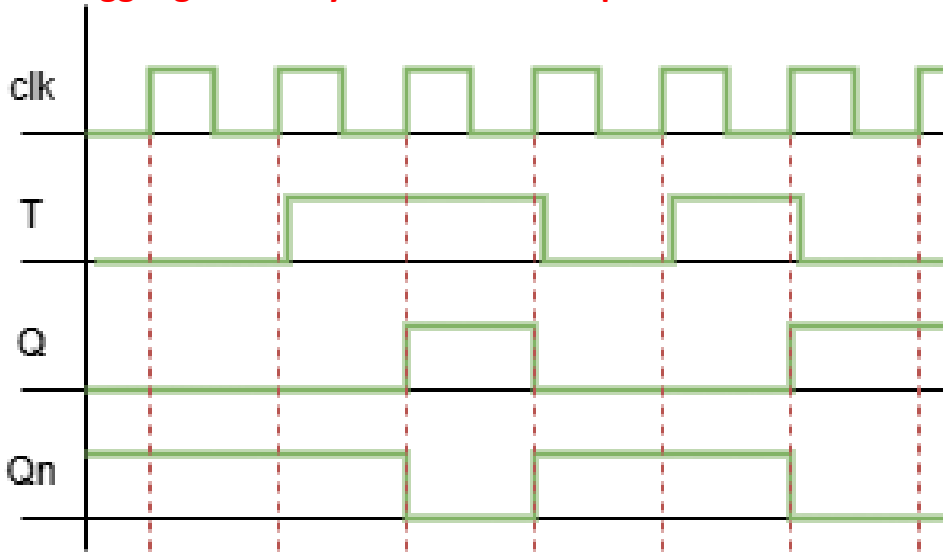
b. Implementation Detail:

Recall from your Logic Design course that a rising edge-triggered T Flip-Flop (TFF) is a sequential circuit that toggles its output state based on the value of the T input at each rising edge of the clock signal **clk**. Specifically, when **clk** transitions on the rising edge (from 0 to 1), the TFF checks the value of **T**:

- If **T = 1**, the output **Q** toggles (i.e., switches from 0 to 1 or from 1 to 0).
- If **T = 0**, the output **Q** remains unchanged.

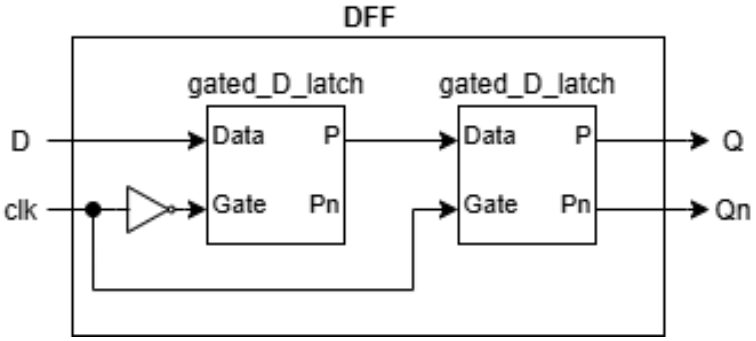
The output **Q** retains its value between clock edges, and the TFF also provides **Qn**, the complement of **Q**.

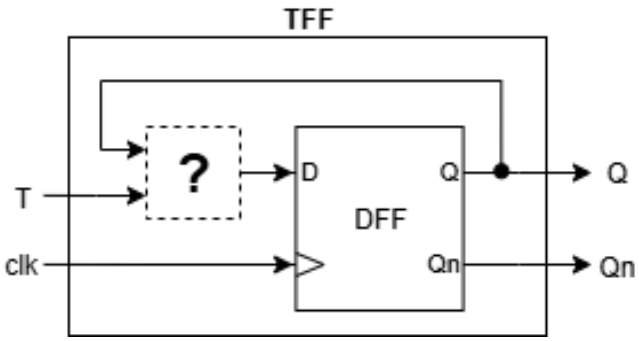
For example, in the following timing diagram, we can observe that when **clk** transitions from 0 to 1 (rising edge), the output **Q** toggles if the **T** input is 1 just before the rising edge. If **T** is 0, the output **Q** holds its previous value. The output **Qn** always reflects the complement of **Q**. This behavior repeats on every rising edge of the clock. **In this part, please initialize the output Q to 0 and Qn to 1 before the clock starts toggling. You may refer to the template code as needed.**



You may come up with your own design, or you may follow the block diagram

below to implement a TFF. The module **gated_D_latch** in the block diagram below is given to you in the template file already.





Hint: The logic block marked with "?" should implement a basic gate (AND, OR, XOR, …) such that the overall circuit functions as a T Flip-Flop.

c. You must use the following I/O definition for your design. Remember to remove the blue-colored comments:

```
`timescale 1ns/100ps
module lab1_2 (
  input wire clk,
  input wire T,
  output wire Q,
  output wire Qn
);
  // Write your code here
endmodule
```

d. Grading: Total of 30%

Design (Total of 20 cycles)	**Score (Each Cycle)**
Correct **Q** output at each cycle	0.75%
Correct **Qn** output at each cycle	0.75%
Use a **non-gate-level description**	0%

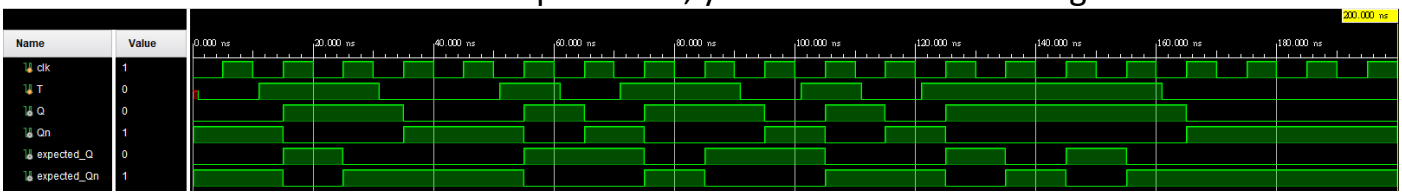
4 lab1_2_t.v (15%)

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 error messages like this:

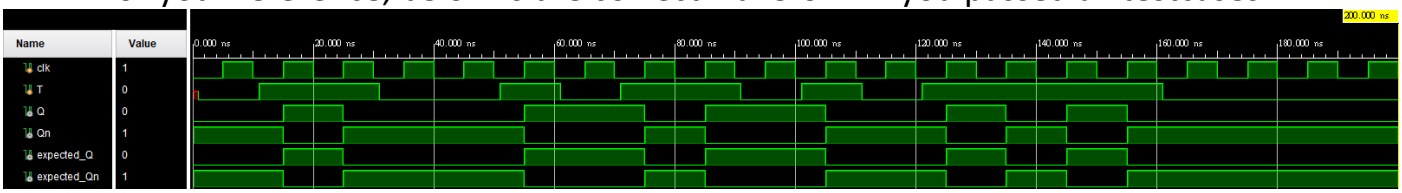
```
[ERROR]: There were          14 errors.
     Data input T: 0 -> 1 -> 1 -> 0 -> 0 -> 1 -> 0 -> 1 -> 1 -> 0 -> 1 -> 0 -> 1 -> 1 -> 1 -> 1 -> 0 -> 0 -> 0 -> 0
    Your output Q: 0 -> 0 -> 1 -> 1 -> 0 -> 0 -> 1 -> 0 -> 1 -> 1 -> 0 -> 1 -> 0 -> 1 -> 1 -> 1 -> 1 -> 0 -> 0 -> 0
   Your output Qn: 1 -> 1 -> 0 -> 0 -> 1 -> 1 -> 0 -> 1 -> 0 -> 0 -> 1 -> 0 -> 1 -> 0 -> 0 -> 0 -> 0 -> 1 -> 1 -> 1
 Correct output Q: 0 -> 0 -> 1 -> 0 -> 0 -> 0 -> 1 -> 1 -> 0 -> 1 -> 1 -> 0 -> 0 -> 1 -> 0 -> 1 -> 0 -> 0 -> 0 -> 0
Correct output Qn: 1 -> 1 -> 0 -> 1 -> 1 -> 1 -> 0 -> 0 -> 1 -> 0 -> 0 -> 1 -> 1 -> 0 -> 1 -> 0 -> 1 -> 1 -> 1 -> 1
```

In this error message, for example, one of the errors can be spotted on the 4th cycle. When T is 1 and Q is 1 before the rising edge, the correct output Q should be 0 (toggle the value right before the rising edge), instead of 1.

If you think this error message is too hard to read, you may also use the waveform viewer in Vivado. For the example above, you will see the following waveform:



For your reference, below is the correct waveform if you passed all testcases:

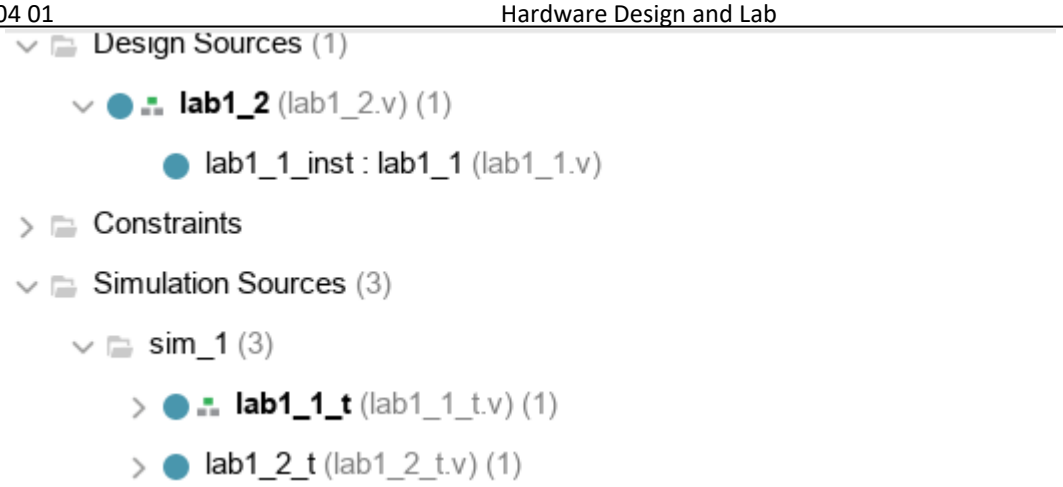


Attention

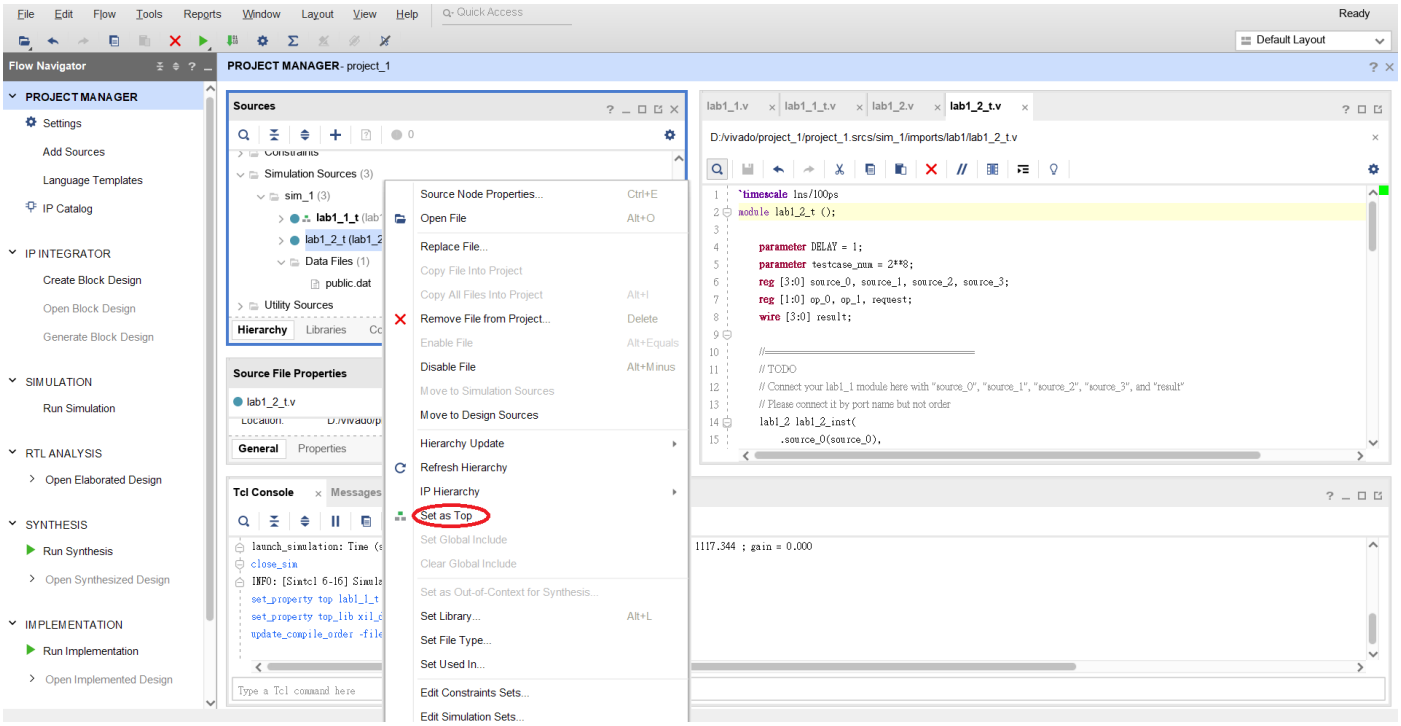
- ✓ You are only allowed to use the gate-level modeling. The dataflow modeling and behavioral modeling are NOT permitted in this lab.
- ✓ You may only write code in the TODO sections in the testbench. Do not modify other parts of the testbench. TA's will use the same testbench to grade your design during demo.
- ✓ DO NOT copy-and-paste code segments from the PDF materials to compose your design. 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 code individually to its corresponding OJ problem. DO NOT hand in any compressed ZIP files, which will be considered an incorrect format.

Appendix

- ✓ We can add two or more testbenches in one single project, e.g., lab1_1_t and lab1_2_t, in the following example. Different testbench can provide various test patterns, or even integrate different design modules to test. In this example, lab1_1_t.v is used to test lab1_1.v; lab1_2_t.v is used to test lab_1_2.v + lab1_1.v.

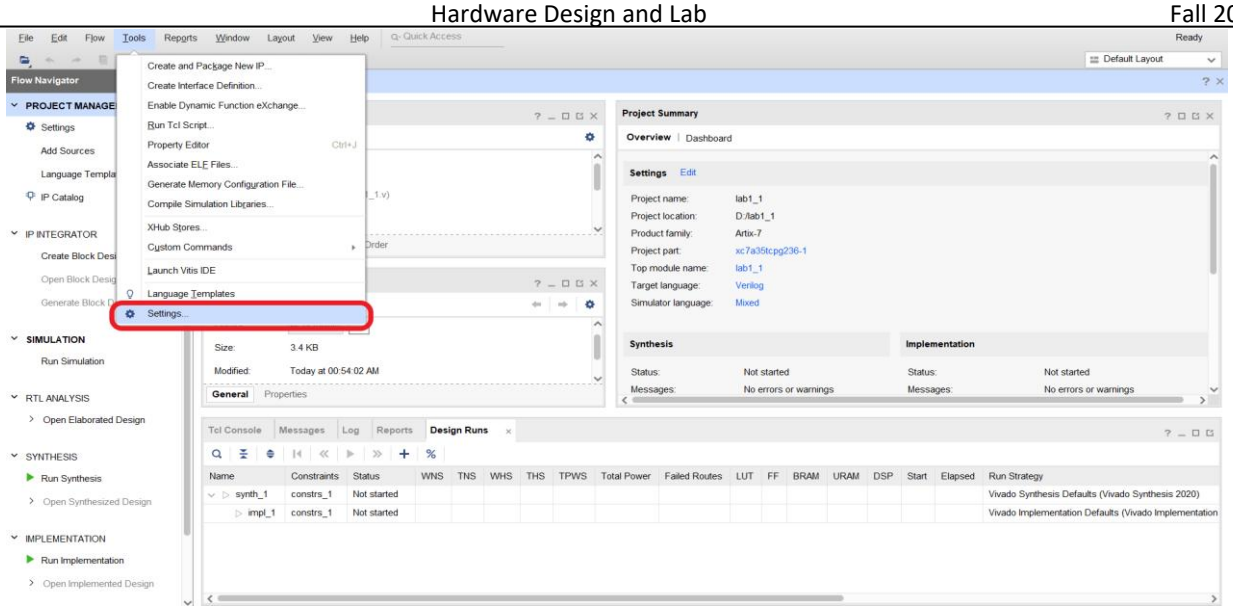


Right click with the mouse on the testbench and select "Set as Top" to activate it. In the following example, we activate the lab1_2_t.v and run the corresponding simulation. (Note: if the two testbenches use the identical top module name, you may need to “Disable File” first in order to activate the other one).

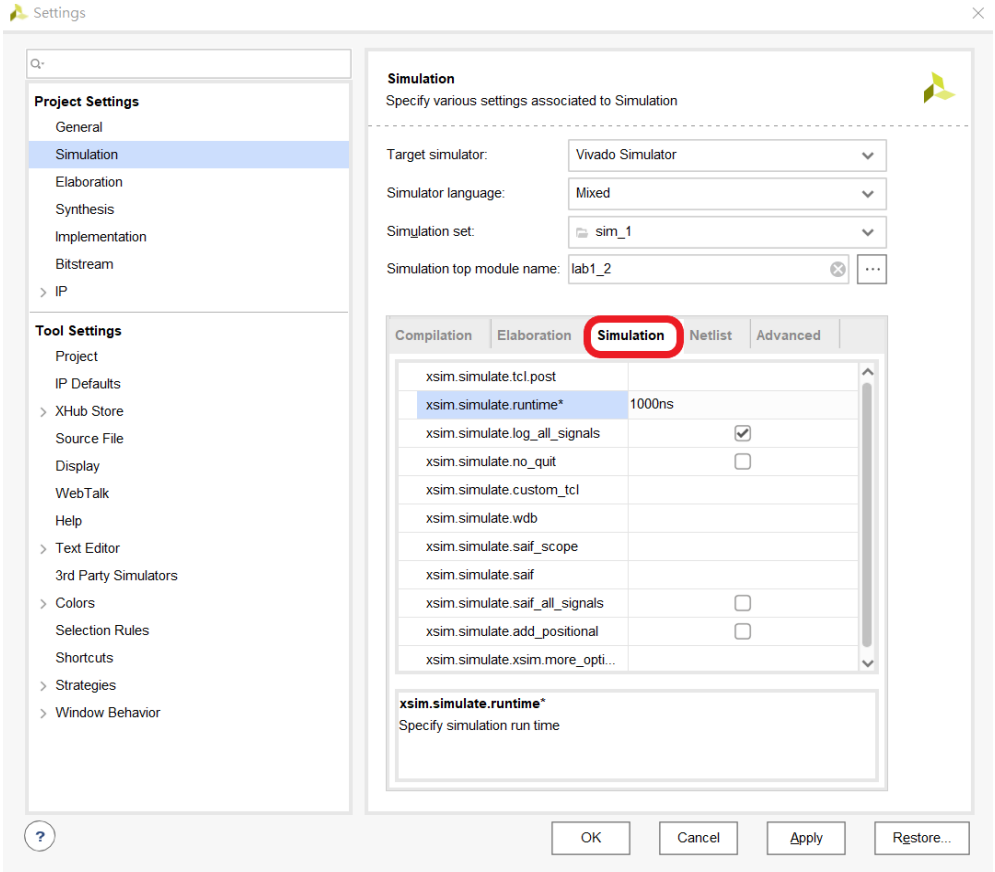


✓ Follow these steps to increase your simulation runtime if you need to.

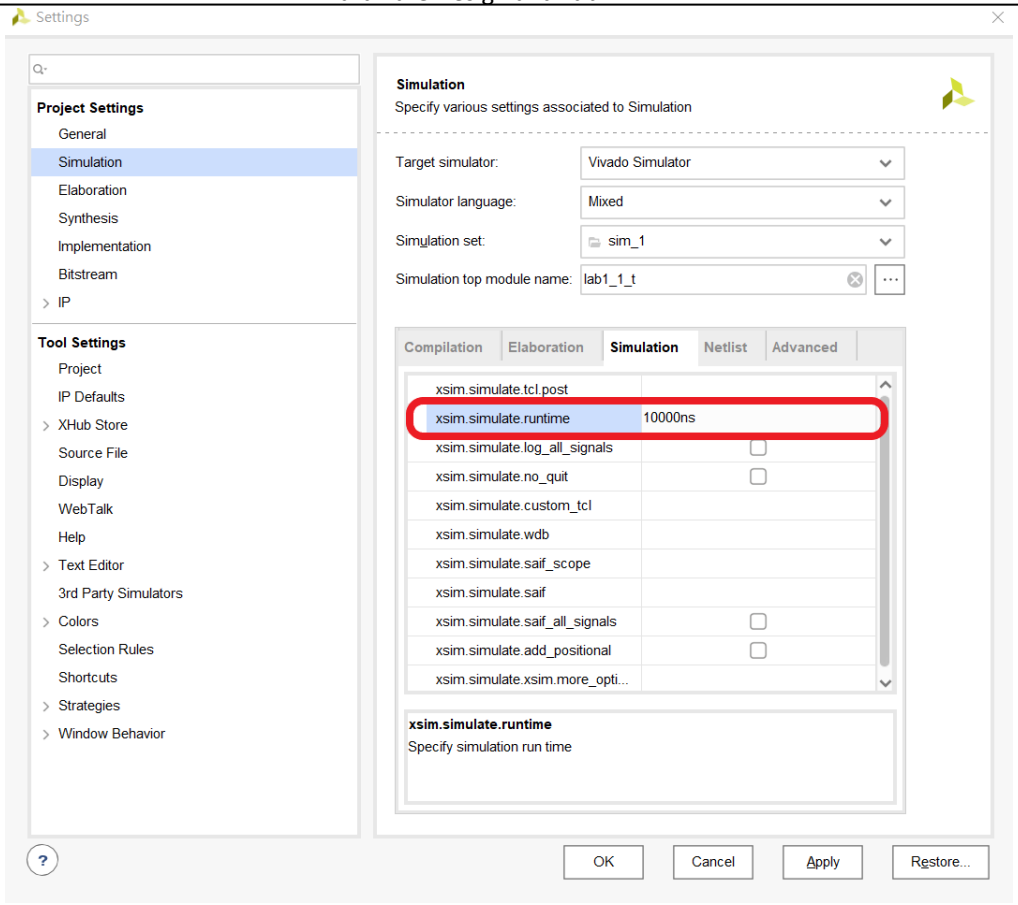
1. Click Tools at the top of Vivado and select settings.



2. Select Simulation.



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



4. Remember to click Apply.