計算機組織 Lab2

學號：111613025

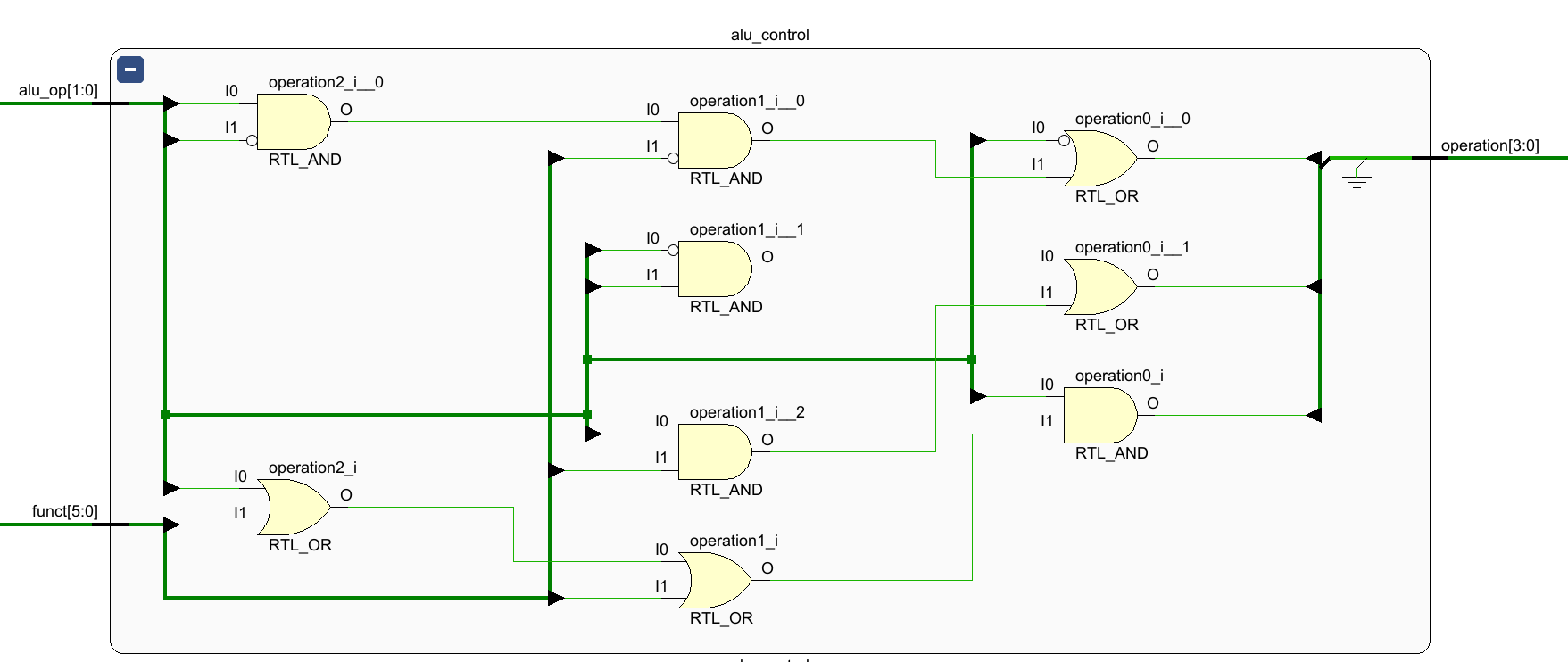
系級：材料15

姓名：黃綵誼

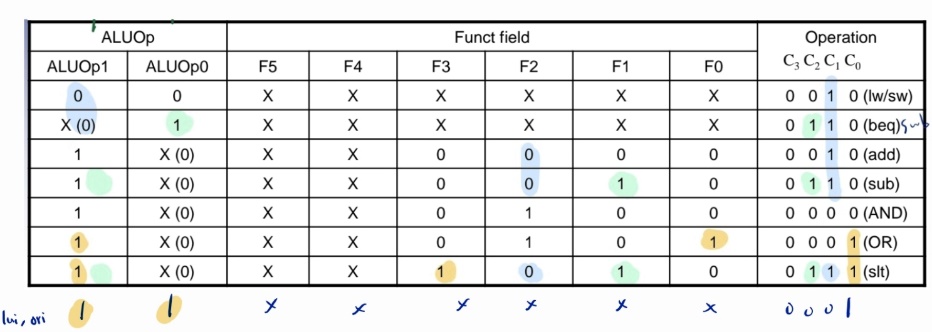
# Architecture Diagrams

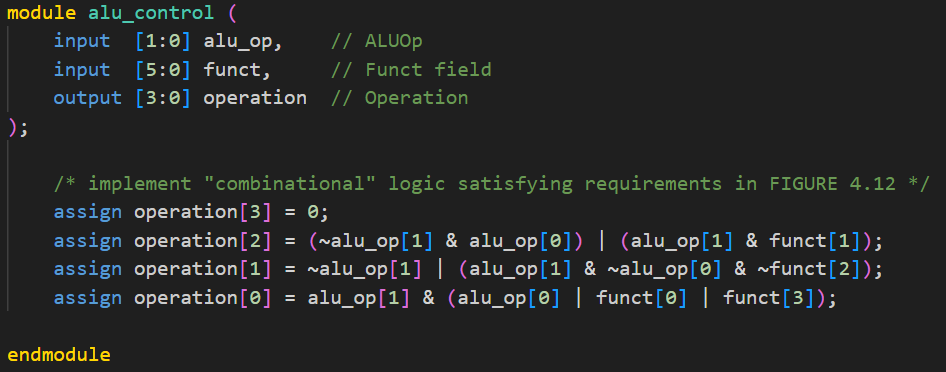
## *Show your ALU control (if you have one), main control and single-cycle processor design by "Schematic" tool in Vivado, or draw them by yourself. And briefly explain them.*

## ALU control

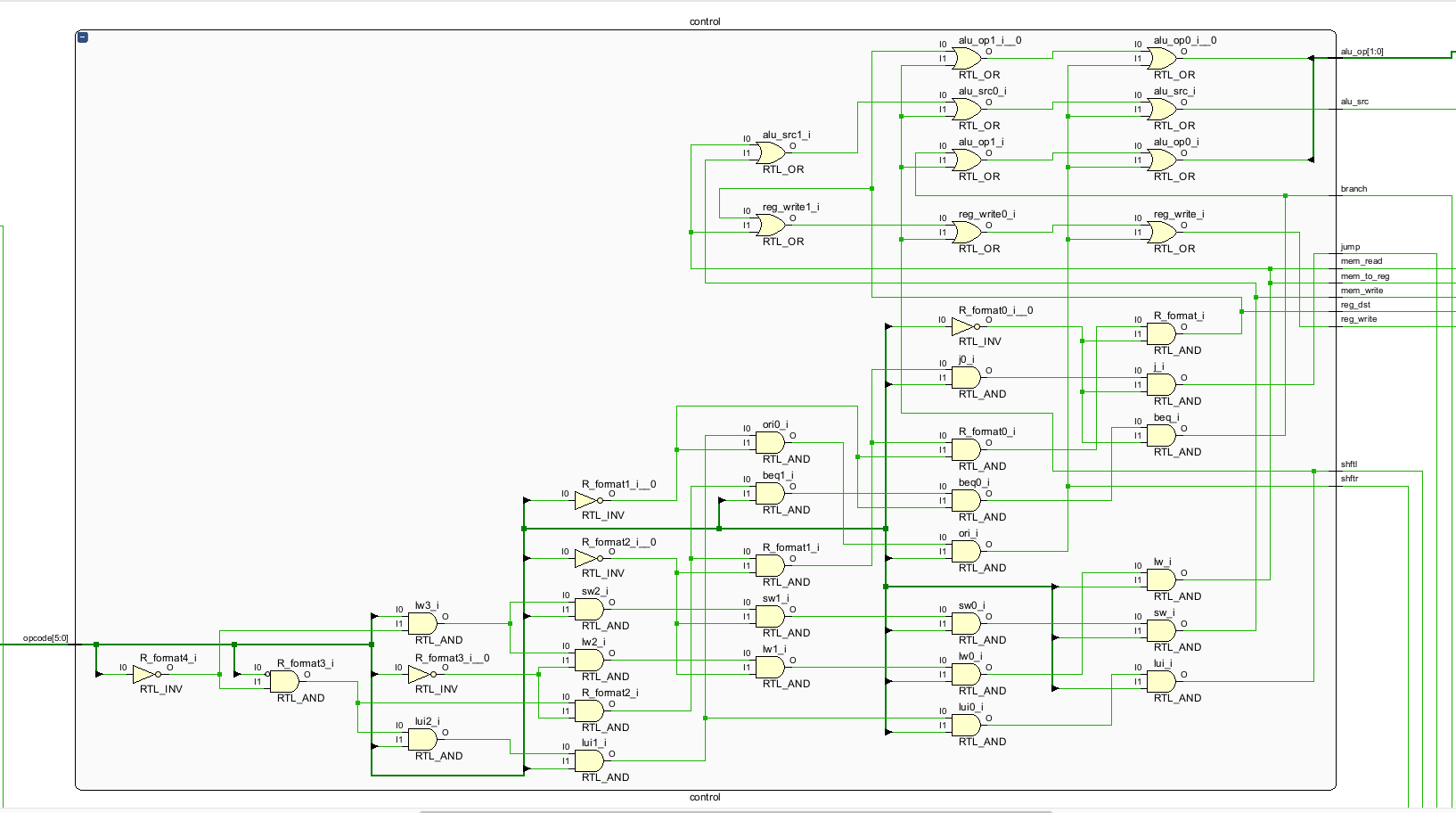


I assign each bit of operation separately according to the truth table below. For lui and ori, do “OR”(0001) operation.

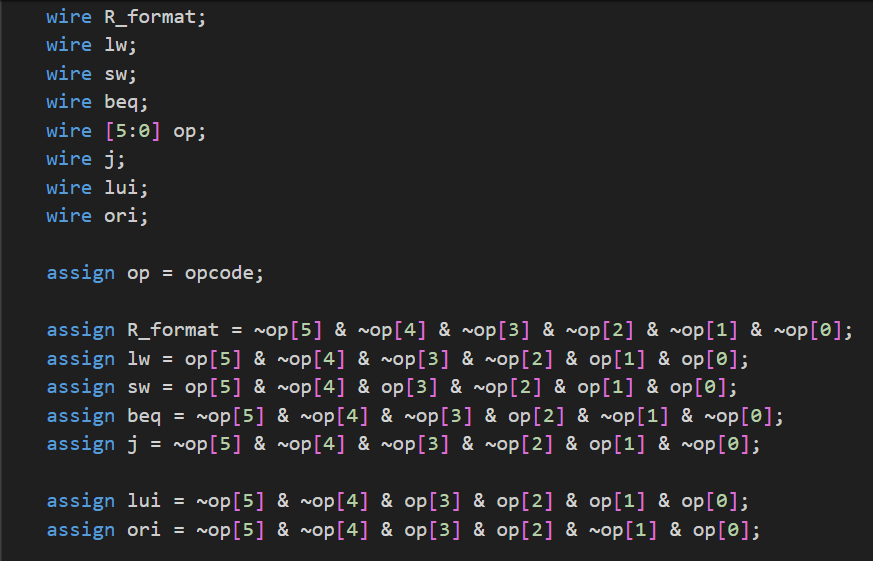




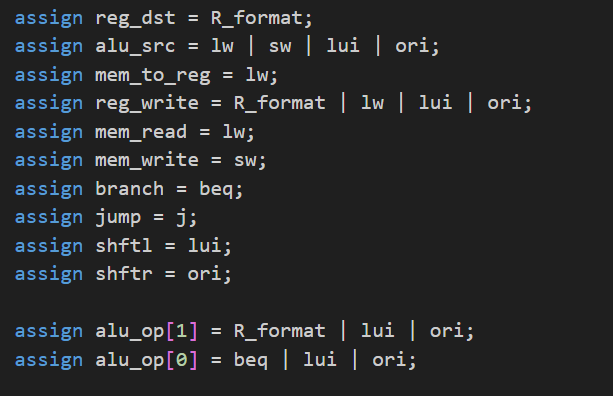
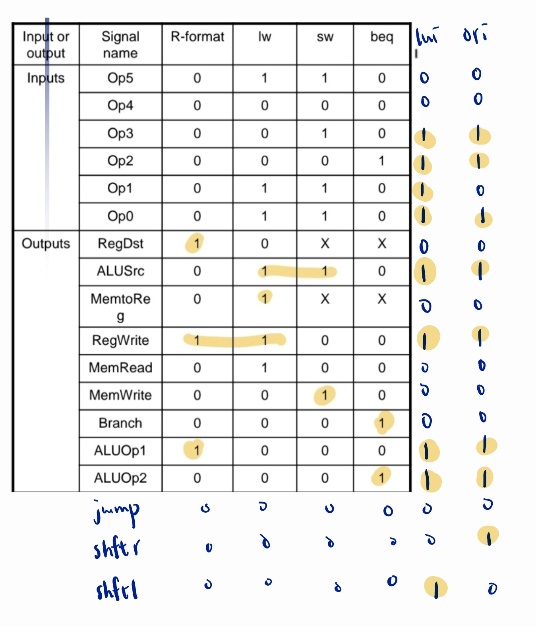
## main control



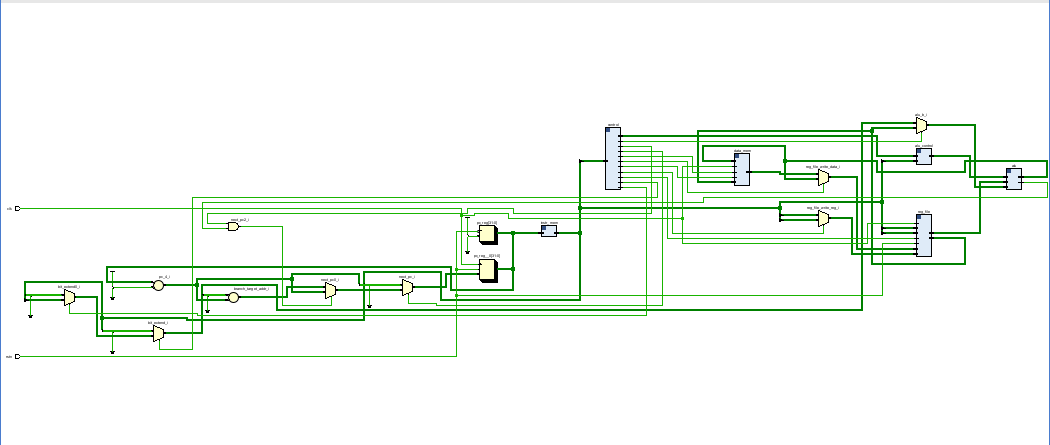
First, I determine whether the opcode is R-format, lw, sw, beq, j, lui, ori.



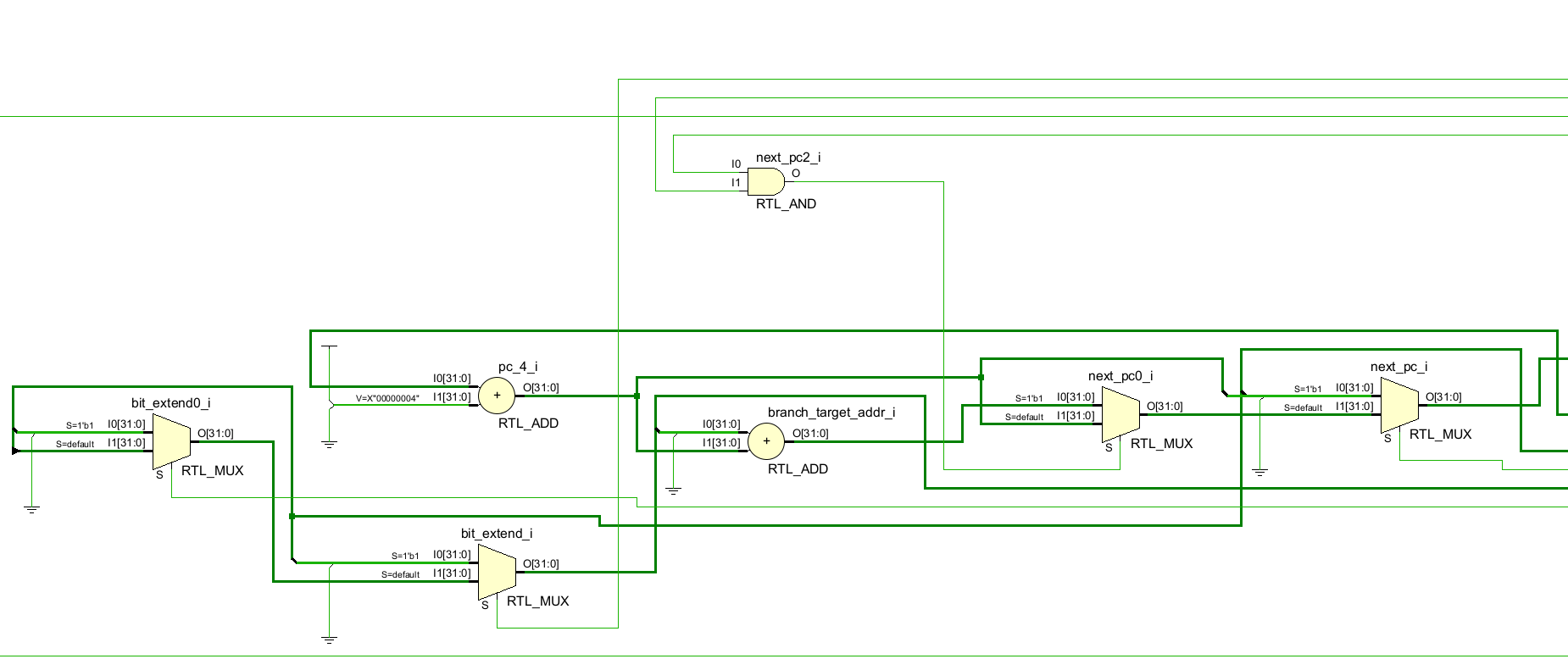
Then, assign output based on the truth table below. I implement “shftr” (shift right and extend with zero) and “shftl” (shift left and extend with zero) for ori and lui respectively. Use alu\_op 2’b11 for both lui and ori.



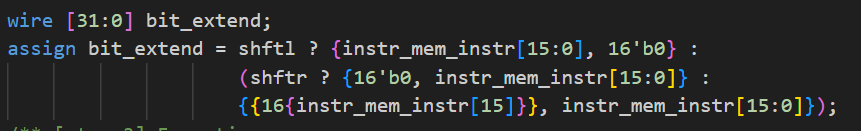
## Overall

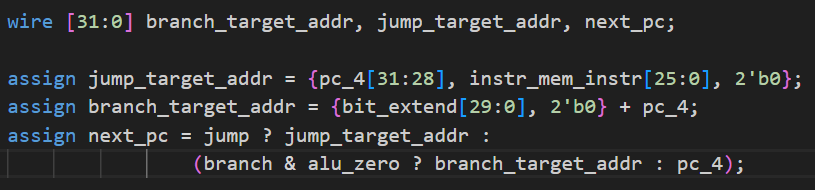


### bit\_extend and next\_pc

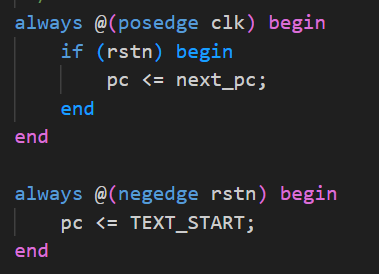
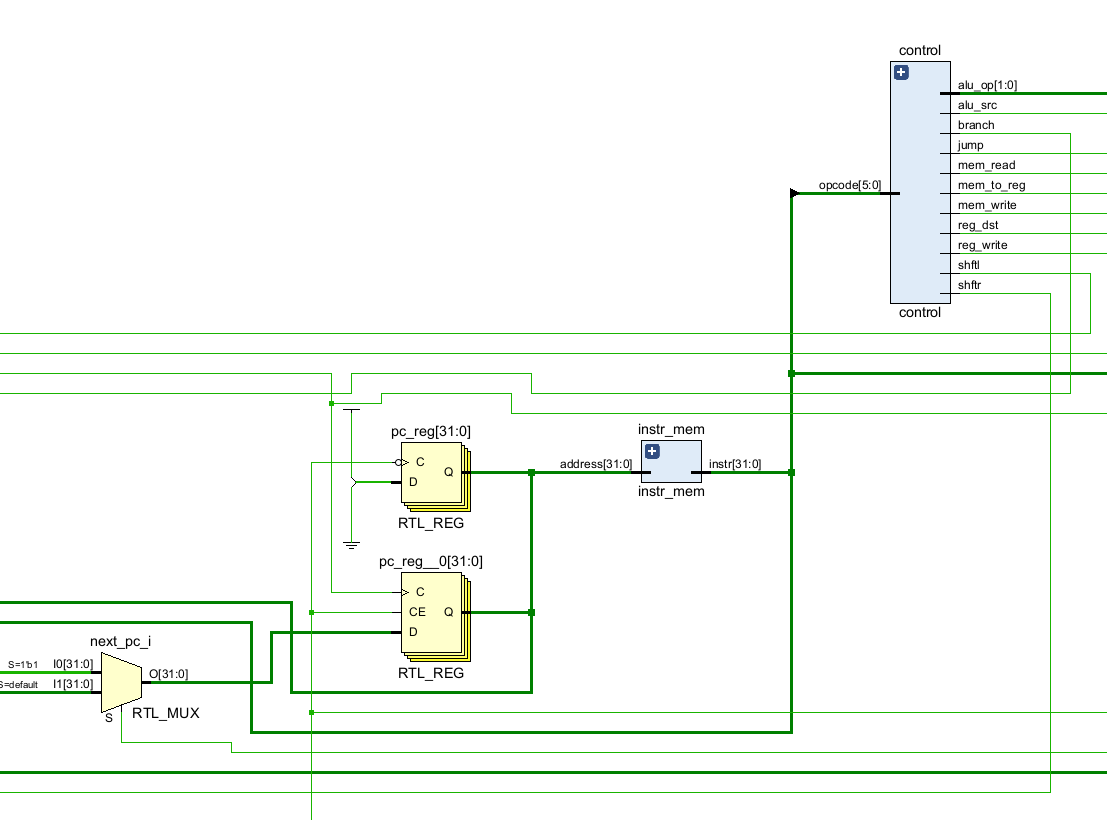


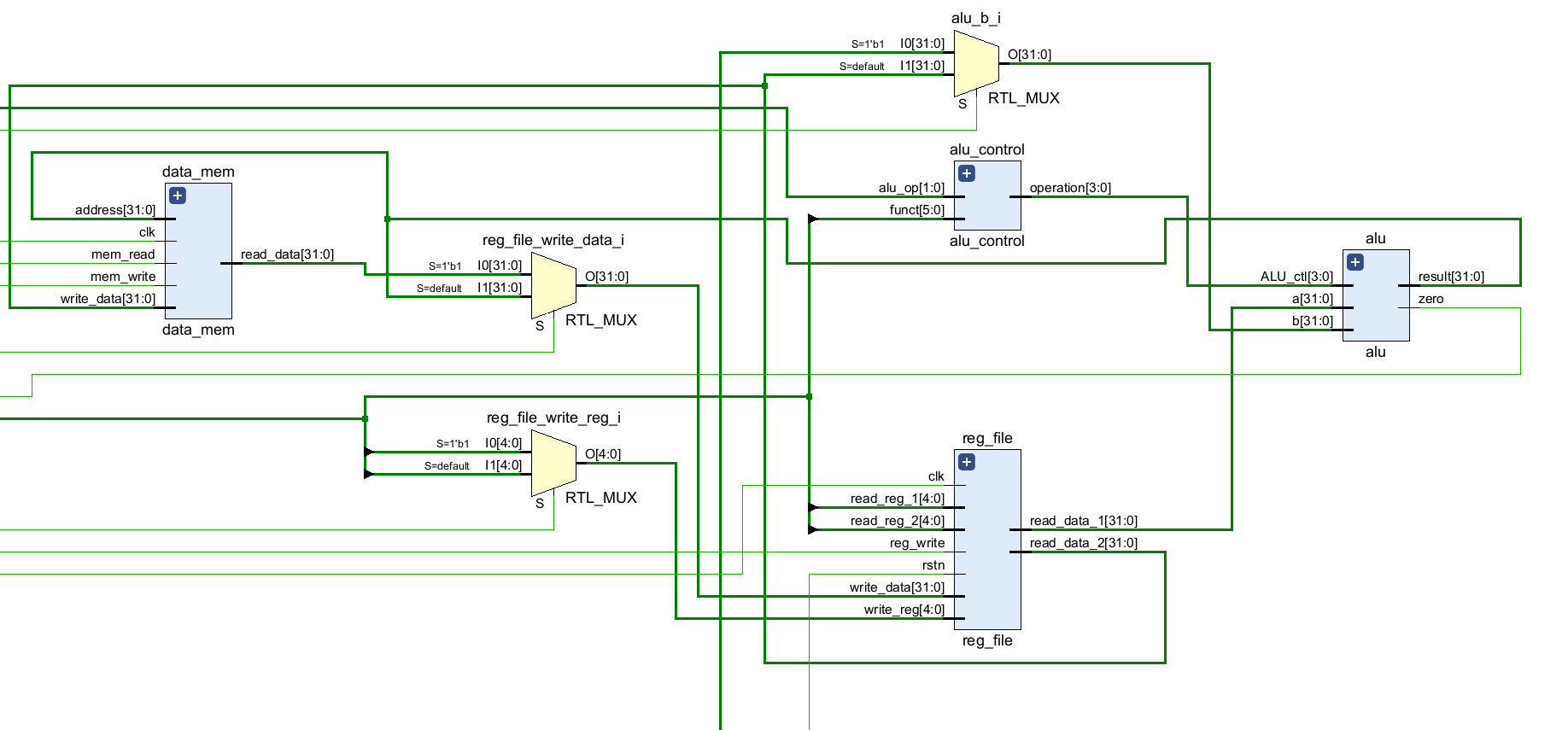
extend right 16 bit with zero when lui (shftl); extend left 16 bit with zero when ori (shftr); extend with sign bit when lw, sw, beq.





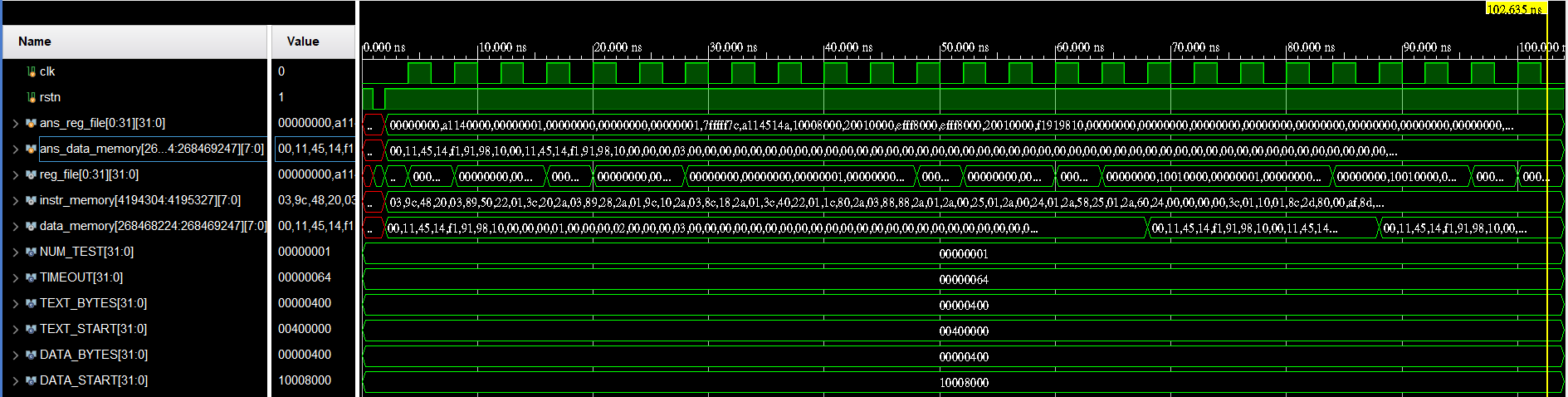
### other parts



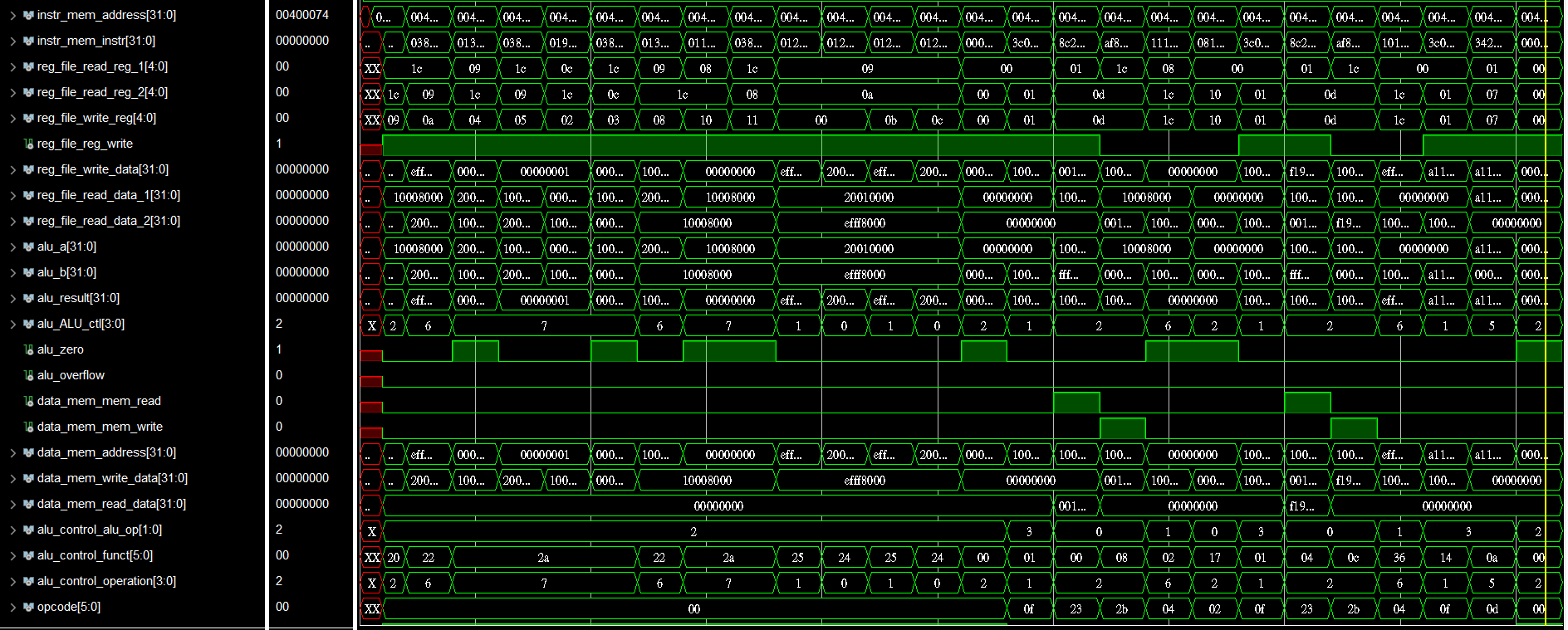


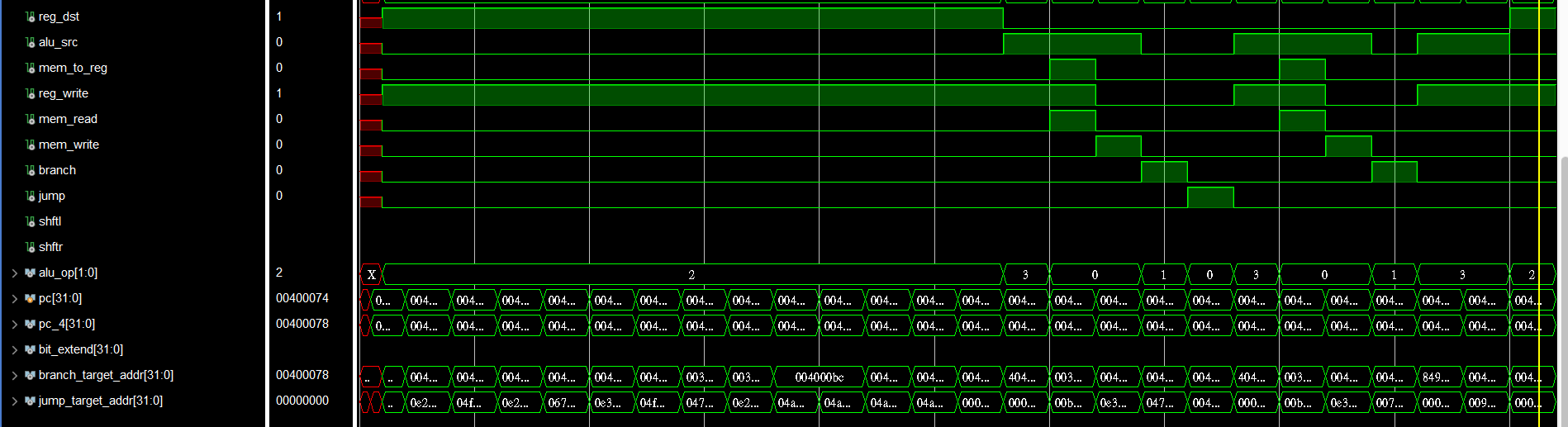
# Experimental Result

## Show the waveform screen shot of the test we provided.



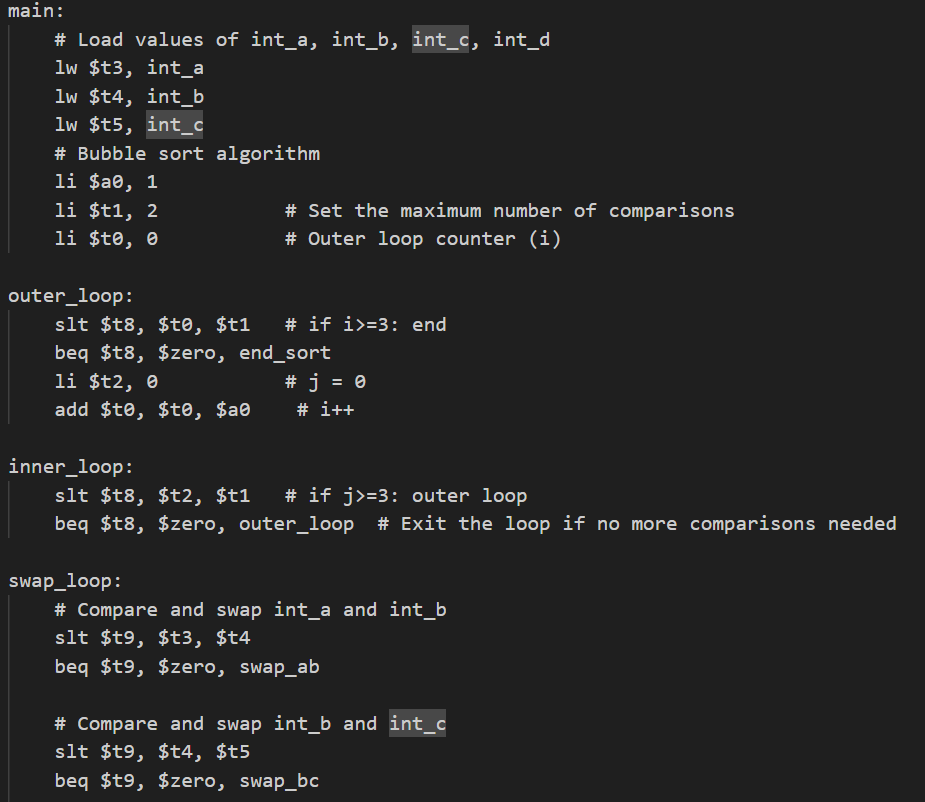
signal from *single\_cycle.v* :

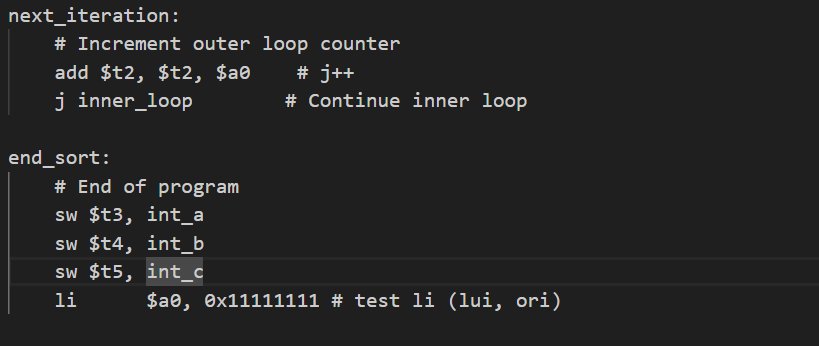


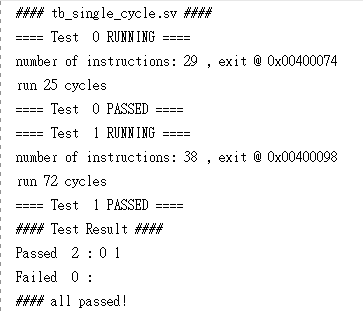


## What other cases you've tested? Why you choose them?

I’ve tested a double loop assembly code (bubble sort algorithm) to test beq and j.







# Answer the following Questions

## When does write to register/memory happen during the clock cycle? How about read?

Write happens at negedge of clock cycle, while read happens at posedge. This can ensure that the data input is stable.

## Translate the "branch" pseudo instructions ( blt , bgt , ble , bge ) in the Green Card into real instructions. Only at register can be modified, and other common registers should not be modified.

### blt $t1, $t2, Loop 🡺 slt $at, $t1, $t2; bne $at, $zero, Loop

t1 < t2 🡺 at != 0

### bgt $t1, $t2, Loop 🡺 slt $at, $t2, $t1; bne $at, $zero, Loop

t1 > t2 🡺 t2 < t1 🡺 at != 0

### ble $t1, $t2, Loop 🡺 slt $at, $t2, $t1; beq $at, $zero, Loop

t1 <= t2 🡺 t2 !< t1 🡺 at == 0

### bge $t1, $t2, Loop 🡺 slt $at, $t1, $t2; beq $at, $zero, Loop

t1 >= t2 🡺 t1 !< t2 🡺 at == 0

## Give a single beq assembly instruction that causes infinite loop. (consider that there's no delay slot)

Loop:

add $t0, $t2, $t3

beq $t0, $t0, Loop

Because $t0 always equals to $t0

## The j instruction can only jump to instructions within the "block" defined by "(PC+4)[31:28]”. Design a method to allow j to jump to the next block (block number + 1) using another j.

First, jump to the end of the current block and enter the next block, then we can jump to the start of the next block.

j current\_block\_end

current\_block\_end:

j next\_block\_start

## Why a Single-Cycle Implementation Is Not Used Today?

Because the clock period must fit the longest delay of the instruction, which may cause much idle time, violating design principle “making the common case fast.” This is inefficient when implementing various instructions, so we will improve performance by pipelining.

# Problem Encountered & Solution(optional)

*List some important problem you’ve met during this lab and the solution.*

## The main problem I encounter is that I don’t know how to implement the lui and ori instruction.

Solution: Ask friends for help and list out the truth table.

# Feedback(optional)

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

Thank you for all the patient and detailed response to the questions on Teams. I think It would be better if there is more detail regarding to which files should be added to simulation source and maybe more testcase.