Final Project: Find the Maximum in a Sequence of Numbers

Problem 1 (40%)

The functional unit in Fig. 1 can perform different operations on two inputs based on an instruction. In this problem, the two inputs and the instruction of the functional unit are 8-bit and 3-bit, respectively. The function table of this functional unit is described in Table 1. Please implement a combinational circuit of this functional unit. Overflow is neglected here.

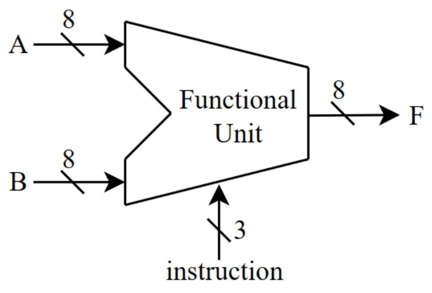
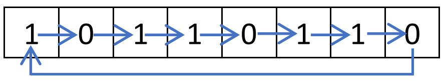


Fig. 1. The Functional Unit

An example of "rotate":
Given that A = 10110110,
right-rotate = 01011011



left-rotate = 01101101

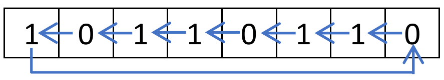


Table 1. Functional Table.

Function	Instruction	Operation (**unsigned**)
Arithmetic	3'b000	F = A + B
Arithmetic	3'b001	F = A + ~B
Logic	3'b010	F = A and B
Logic	3'b011	F = A or B
Logic	3'b100	F = A xor B
Shifter, Arithmetic	3'b101	F = (shift-right A by 1 bit) + B
Rotate, Arithmetic	3'b110	F = (right-rotate A by 1 bit) + B
Rotate, Arithmetic	3'b111	F = (left-rotate A by 1 bit) + B

Problem 2 (60%)

(1) Problem Formulation: See Fig. 2 for clearer concept about the description here. Two sequences of positive integers will be sequentially loaded into the circuit, one pair of integers per cycle. However, the length of a sequence (the number of the integers in a sequence) is not a fixed value. The two "valid" integers which are loaded into the circuit in a cycle will perform an operation and generate a result. After loading two sequences entirely, there should be many results. The circuit needs to find the maximum among these results. So, for these two sequences of integers, the circuit outputs one result (the maximum).

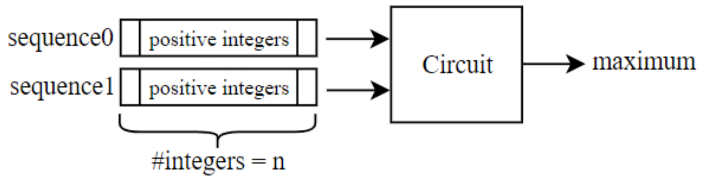


Fig. 2. Problem Formulation.

(2) I/O Description:

Table 2. I/O description.

Signal Name	I/O	Width	Description
clk	I	1	Clock (positive-edge-triggered clock is used)
rst_n	I	1	Active-low reset (synchronous reset)
start	I	1	Single-cycle active-high pulse. After this signal becomes 1, it means the circuit can start reading the integers and doing the computation.
valid	I	1	Only when this signal is 1, the input data are valid to be computed; otherwise, it is invalid input data, which you cannot use.
Data_A	I	8	A positive integer read from sequence0.
Data_B	I	8	A positive integer read from sequence1.
one_left	I	1	Single-cycle active-high pulse. This signal indicates one pair of integers is left to be read/computed.
instruction	I	3	3-bit instruction. The functional table is shown as Table 1.
maximum	O	8	The maximum among the results of the operations on several pairs of integers.
finish	O	1	Single-cycle active-high pulse. This signal is used to inform the testbench to examine your output results. It should be pulled high in the same cycle when the *maximum* is ready.

(3) Waveform

The waveform shown in Fig. 3 is an example where a sequence contains four integers. After the circuit is reset, it waits for the single-cycle *start* pulse to start reading the integers. After the single-cycle *start* pulse, data can be read and used when *valid* equals 1. **Note that *valid* might be pulled high at any cycle after the single-cycle *start* pulse, but high *valid* of continuous cycle will not be the case in this homework.** As we mentioned in (1) Problem Formulation, the number of integers to be read is not a fixed value, so *one_left* is used to indicate that one pair of integers is left to be read/computed. For example, if there are four pairs of integers to be computed, *one_left* becomes 1 for one cycle after the third pair of integers is read. Finally, when the correct *maximum* is computed, pull up a single-cycle high *finish* pulse at the same cycle. Every time after we finish all operations for two sequences, there may be another two sequences to be loaded into the circuit to do some operations, which means ***start*** **may be pulled high many times. However, *rst_n* will not be pulled down again**.

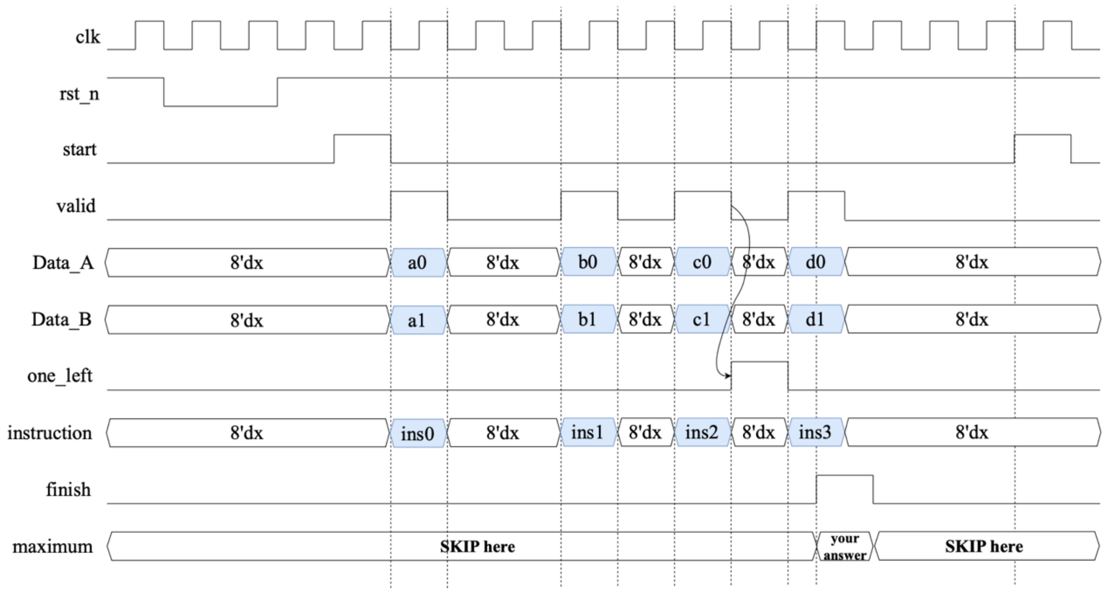


Fig. 3. An example of the waveform

(4) Circuit Structure

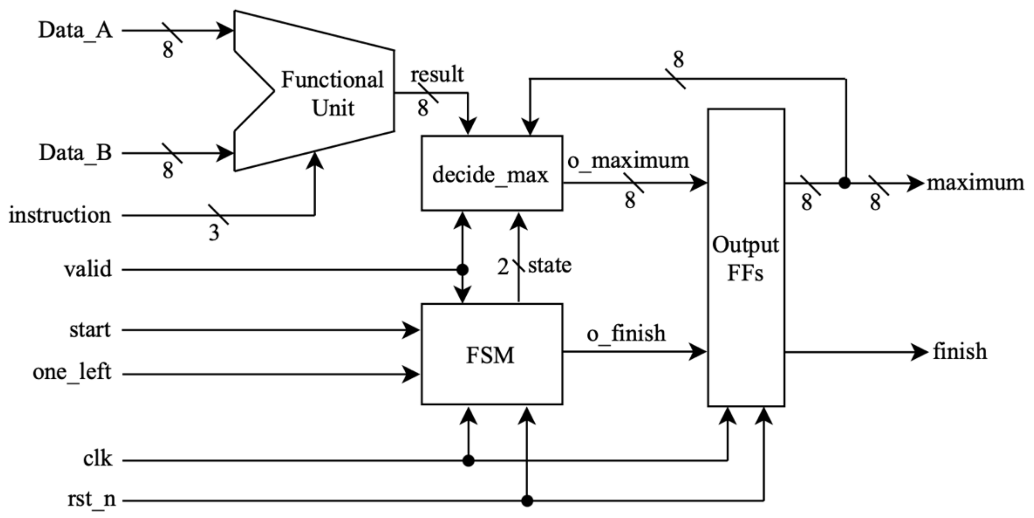


Fig. 4. The circuit structure

The circuit structure is shown in Fig. 4. It is a sequential circuit, which you would practice how to design with an FSM (finite state machine). **The design must be a synchronous circuit with a positive-edge-triggered clock.**

Two integers read from sequence0 and sequence1 are loaded into the circuit through *Data_A* and *Data_B*, respectively. These two integers will perform an operation based on *instruction*, and then get a result. *Data_A*, *Data_B*, and *instruction* are read in the same cycle. **You must use the functional unit designed in Problem 1**

as a part of the implementation of Problem 2. For the remaining components in the circuit, you can refer to the circuit structure in Fig. 4 or design other structures by yourself. Hint: FSM may be a necessary component in the circuit.

FSM controls the states of the circuit. **When designing FSM, you can refer to the state diagram in Fig. 5 or design other state diagram by yourself.**

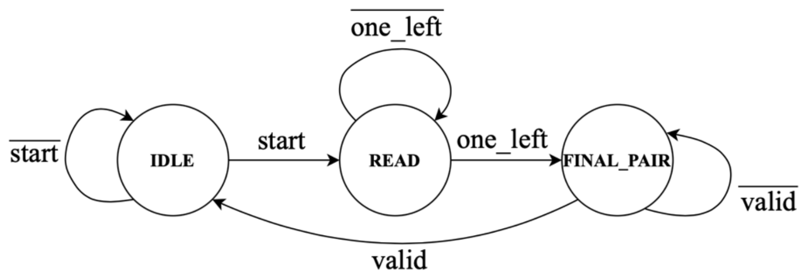

$\overline{\text{one\_left}}$
$\overline{\text{start}}$
IDLE
start
READ
one_left
FINAL_PAIR
$\overline{\text{valid}}$
valid

Fig. 5. State diagram of FSM.

※ State Description:

IDLE: The circuit does not do anything and stay still.

READ: Read the pairs of the integers until there remains one pair of the integers to be read/computed.

FINAL_PAIR: Read the final pair of the integers.

decide_max block decides the value of *o_maximum* based on the state of FSM. In *READ* and *FINAL_PAIR* states, valid *result* is compared with *maximum*, and update *maximum* if necessary. You have to think about the conditions where *maximum* should be unchanged and the conditions where *maximum* should be reset to zero. **Actually, you can combine the behavior of decide_max block into your FSM**, that is, decide *o_maximum* in FSM when writing the Verilog code.

Files (See the tutorial of workstation in *workstation.pdf*)

We provide these five files and one folder in this homework:

1. Makefile 2. project_1.v 3. tb1.v 4. project_2.v 5. tb2.v 6. data/

You can download the zip file and unzip it to get them from eeclass. These files need to be in the same directory. Simply speaking, after unzipping, it should be a folder which is named “Logic_Design”, and don’t change the relative paths of the files. Then upload it to the workstation. You should do the homework in Logic_Design/, that is, type “cd Logic_Design” on the workstation and start your work.

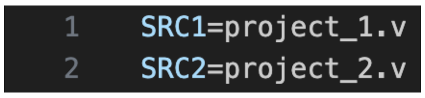
You need to write your Verilog code with project_1.v and project_2.v for problem 1 and 2, respectively. You need to finish “TODO” in project_1.v and project_2.v. The ports declarations are already written, and the instantiation of Functional Unit is already written in project_2.v. **Note that you should change the filenames of project_1.v and project_2.v to <stu_id>_1.v and <stu_id>_2.v, respectively. For example, 111222333_1.v and 111222333_2.v for problem1 and problem2, respectively.**

We will use tb1.v and tb2.v with some hidden cases to examine your designs of Problem 1 and Problem 2, respectively.

Simulation

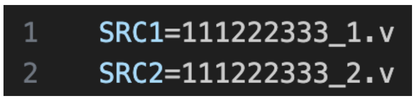
In Makefile, you should modify these two lines. You need to change “project_1.v” and “project_2.v” to the corresponding filenames which you name previously..

For example, if you change project_1.v and project_2.v to 111222333_1.v and 111222333_2.v, respectively, then your Makefile should be modified like:



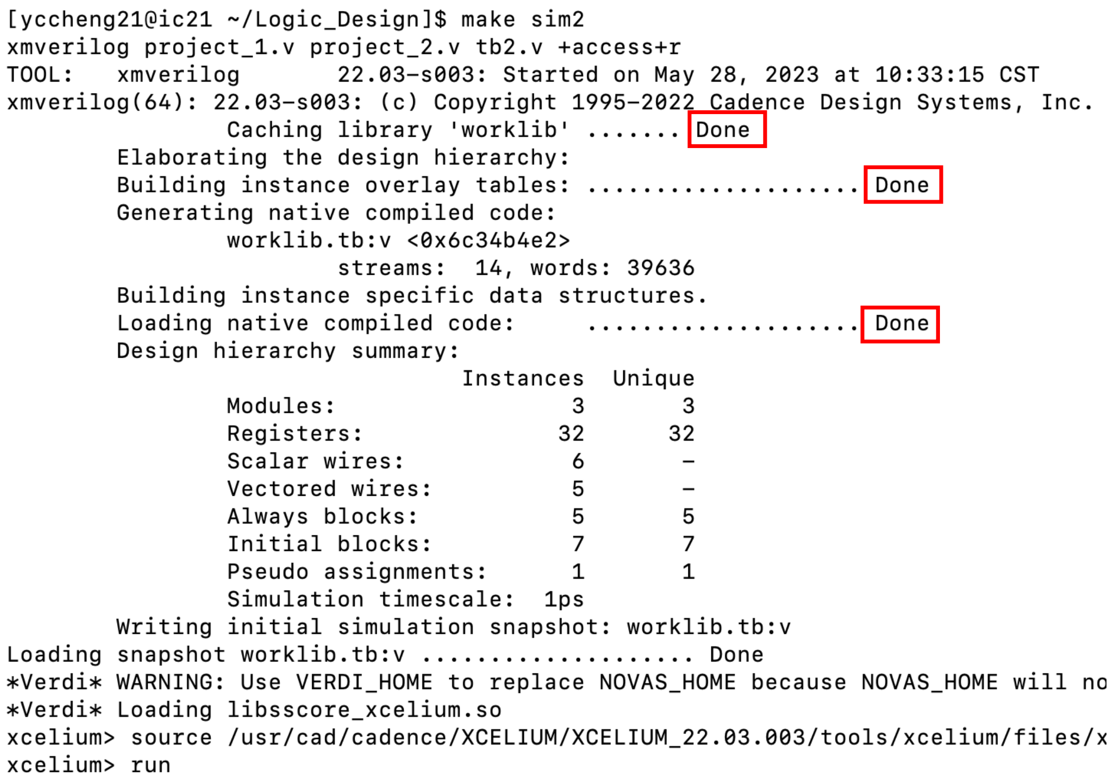
```
SRC1=project_1.v
SRC2=project_2.v
```

$\rightarrow$



```
SRC1=111222333_1.v
SRC2=111222333_2.v
```

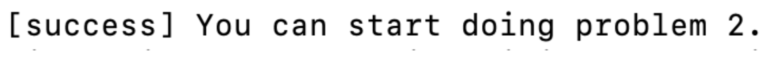
You need to see this view, and then your simulation will start successfully, otherwise, your Verilog file may have some syntax or other errors, which fails to do the simulation.



```
[yccheng21@ic21 ~/Logic_Design]$ make sim2
xmverilog project_1.v project_2.v tb2.v +access+r
TOOL:   xmverilog       22.03-s003: Started on May 28, 2023 at 10:33:15 CST
xmverilog(64): 22.03-s003: (c) Copyright 1995-2022 Cadence Design Systems, Inc.
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Building instance overlay tables: .................... Done
        Generating native compiled code:
                worklib.tb:v <0x6c34b4e2>
                        streams:  14, words: 39636
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                                  Instances  Unique
                Modules:                 3       3
                Registers:              32      32
                Scalar wires:            6       -
                Vectored wires:          5       -
                Always blocks:           5       5
                Initial blocks:          7       7
                Pseudo assignments:      1       1
                Simulation timescale:  1ps
        Writing initial simulation snapshot: worklib.tb:v
Loading snapshot worklib.tb:v .................... Done
*Verdi* WARNING: Use VERDI_HOME to replace NOVAS_HOME because NOVAS_HOME will no
*Verdi* Loading libsscore_xcelium.so
xcelium> source /usr/cad/cadence/XCELIUM/XCELIUM_22.03.003/tools/xcelium/files/x
xcelium> run
```

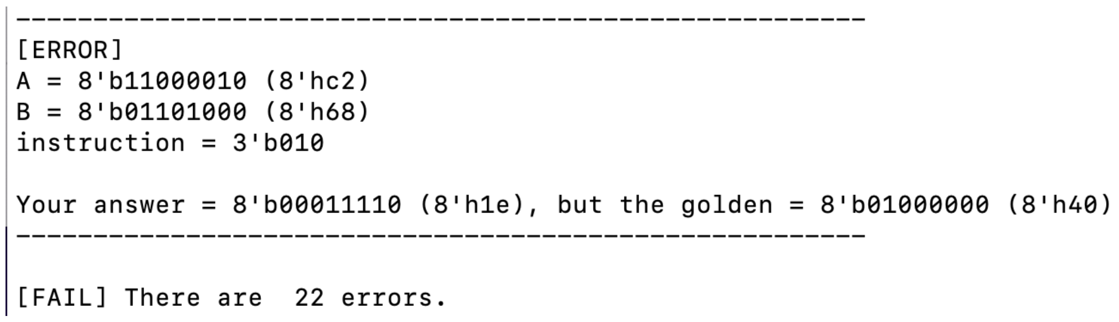
1. To do the simulation of problem1, type “make sim1”

   If you pass all the patterns, you will see:



```
[success] You can start doing problem 2.
```

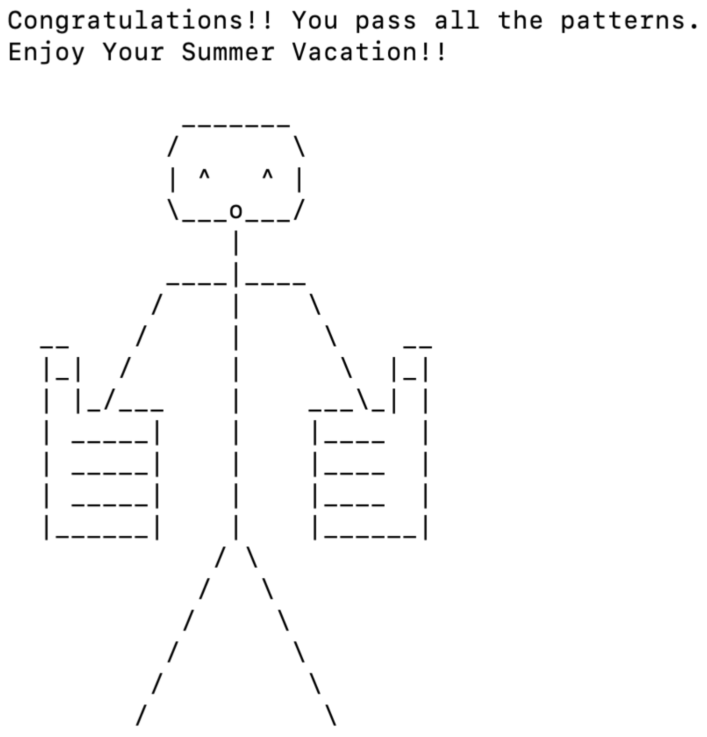
   If there are some errors, you will see:



```
-------------------------------------------------------
[ERROR]
A = 8'b11000010 (8'hc2)
B = 8'b01101000 (8'h68)
instruction = 3'b010

Your answer = 8'b00011110 (8'h1e), but the golden = 8'b01000000 (8'h40)
-------------------------------------------------------

[FAIL] There are  22 errors.
```

2. To do the simulation of problem2, type “make sim2”

   If you pass all the patterns, you will see:



```
Congratulations!! You pass all the patterns.
Enjoy Your Summer Vacation!!

           _______
          /       \
          | ^   ^ |
          \___o___/
              |
          ____|____
         /    |    \
  __    /     |     \    __
  |_|  /      |      \  |_|
  | |_/___    |    ___\_| |
  | _____|    |    |____  |
  | _____|    |    |____  |
  | _____|    |    |____  |
  |______|    |    |______|
             / \
            /   \
           /     \
          /       \
         /         \
        /           \
```

   If there are some errors, you will see different types of error descriptions.

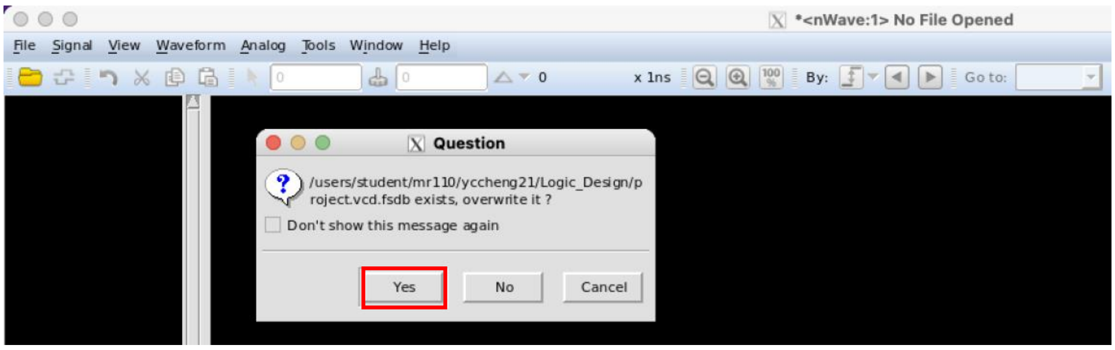
Waveforms

Because Problem 1 is a combinational circuit, it's useless to see the waveform. So, only Problem 2 has the waveform after you do the simulation.

1. To watch the waveform of problem2, type "make wave", and you will see this view:

```
logDir = /users/student/mr110/yccheng21/Logic_Design/nWaveLog

                                   Verdi (R)

                 Version S-2021.09 for linux64 - Aug 30, 2021

                    Copyright (c) 1999 - 2021 Synopsys, Inc.
   This software and the associated documentation are proprietary to Synopsys,
 Inc. This software may only be used in accordance with the terms and conditions
 of a written license agreement with Synopsys, Inc. All other use, reproduction,
   or distribution of this software is strictly prohibited.  Licensed Products
     communicate with Synopsys servers for the purpose of providing software
    updates, detecting software piracy and verifying that customers are using
    Licensed Products in conformity with the applicable License Key for such
  Licensed Products. Synopsys will use information gathered in connection with
    this process to deliver software updates and pursue software pirates and
                                   infringers.

 Inclusivity & Diversity - Visit SolvNetPlus to read the "Synopsys Statement on
            Inclusivity and Diversity" (Refer to article 000036315 at
                        https://solvnetplus.synopsys.com)
rcfile = /users/student/mr110/yccheng21/Logic_Design/novas.rc
guiConfFile (read)= /users/student/mr110/yccheng21/Logic_Design/novas.conf (working directory)
guiConfFile (write)= /users/student/mr110/yccheng21/Logic_Design/novas.conf (working directory)
```

press Enter, and then you can continue entering other commands in the shell. Meanwhile, the GUI tool used to watch the waveform, nWave starts, which will be shown like:

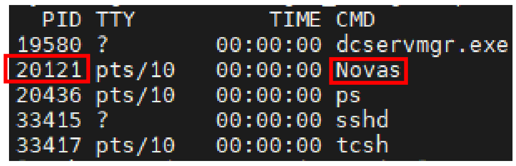

*<nWave:1> No File Opened
File Signal View Waveform Analog Tools Window Help
x 1ns
By:
Go to:
Question
Don't show this message again
Yes
No
Cancel

/users/student/mr110/yccheng21/Logic_Design/project.vcd.fsdb exists, overwrite it ?

However, if you encounter that nWave is flickering:

Step1. type "ps -u your_account_number", e.g., ps -u acc123. It will show the process status on your account.

Step2. You can see a *number* in PID column corresponding to "Novas" command. type "kill -9 *number*" to kill it. For the following example, you should type “kill -9 20121”



```
  PID TTY          TIME CMD
19580 ?        00:00:00 dcservmgr.exe
20121 pts/10   00:00:00 Novas
20436 pts/10   00:00:00 ps
33415 ?        00:00:00 sshd
33417 pts/10   00:00:00 tcsh
```

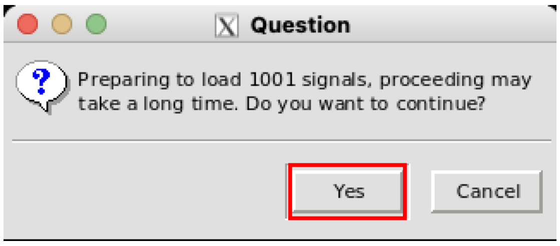
Step3. type “make clean”

Step4. type “make sim2” to simulate problem2 again.

Step5. type “make wave” if you want to watch the waveform.

2. Press this button to select the signals to be seen.

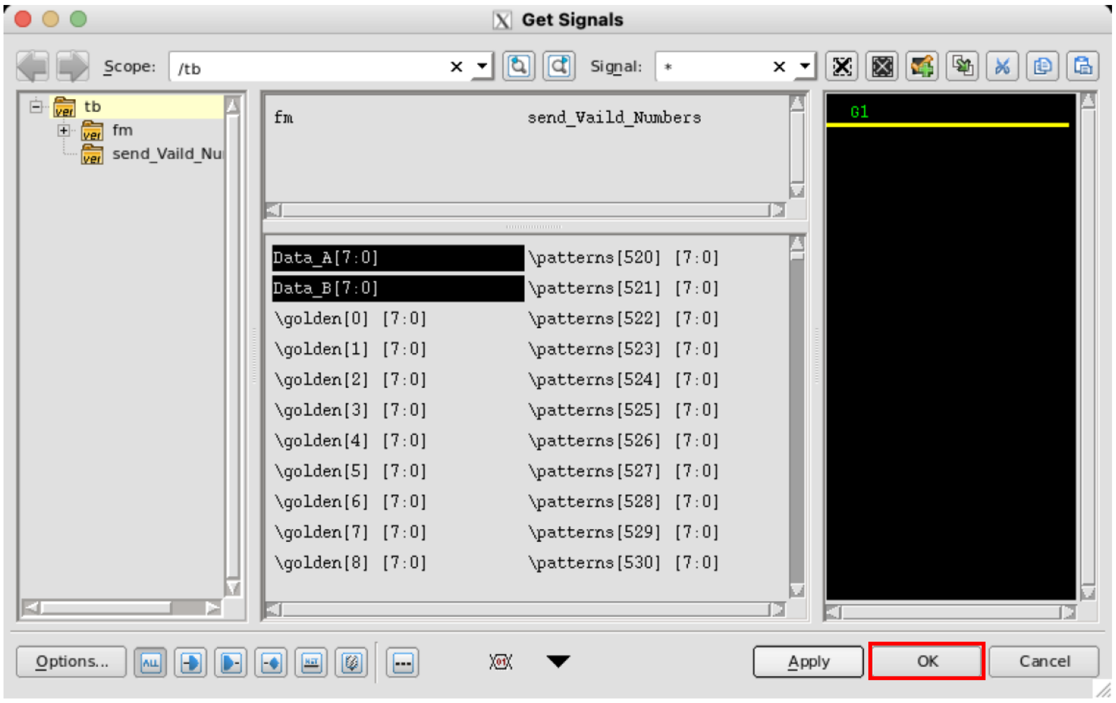
Then you will see this, choose Yes.



Question

Preparing to load 1001 signals, proceeding may take a long time. Do you want to continue?

3. Select the signals and then click OK.

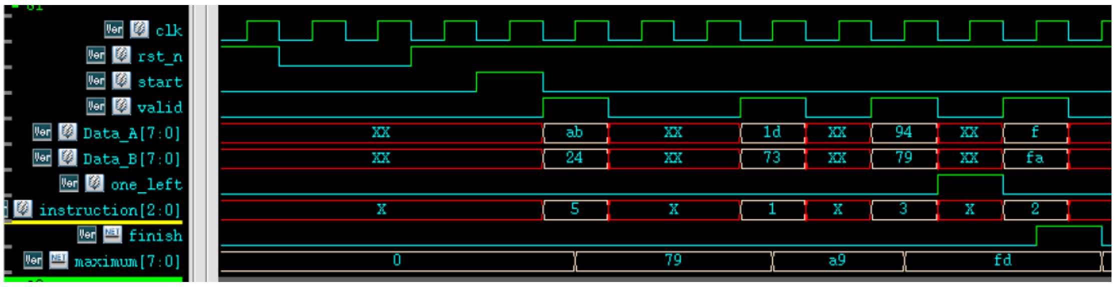


Get Signals

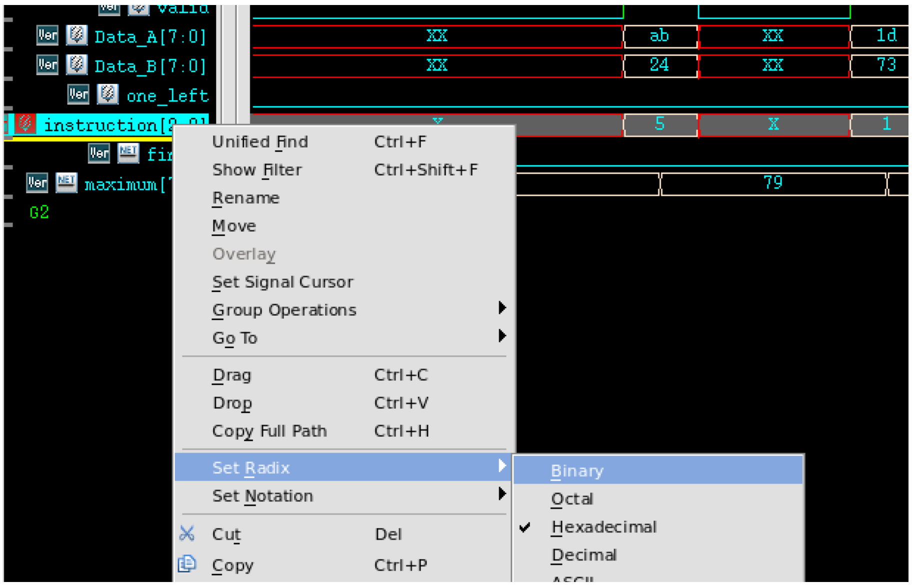
Scope: /tb   Signal: *

```
tb
  fm
  send_Vaild_Nu

fm                              send_Vaild_Numbers

Data_A[7:0]                     \patterns[520]  [7:0]
Data_B[7:0]                     \patterns[521]  [7:0]
\golden[0]  [7:0]               \patterns[522]  [7:0]
\golden[1]  [7:0]               \patterns[523]  [7:0]
\golden[2]  [7:0]               \patterns[524]  [7:0]
\golden[3]  [7:0]               \patterns[525]  [7:0]
\golden[4]  [7:0]               \patterns[526]  [7:0]
\golden[5]  [7:0]               \patterns[527]  [7:0]
\golden[6]  [7:0]               \patterns[528]  [7:0]
\golden[7]  [7:0]               \patterns[529]  [7:0]
\golden[8]  [7:0]               \patterns[530]  [7:0]

G1
```

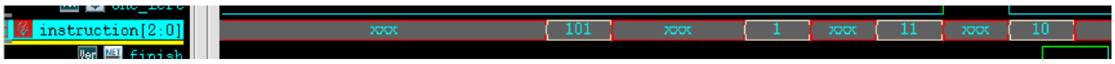
Then you will see:



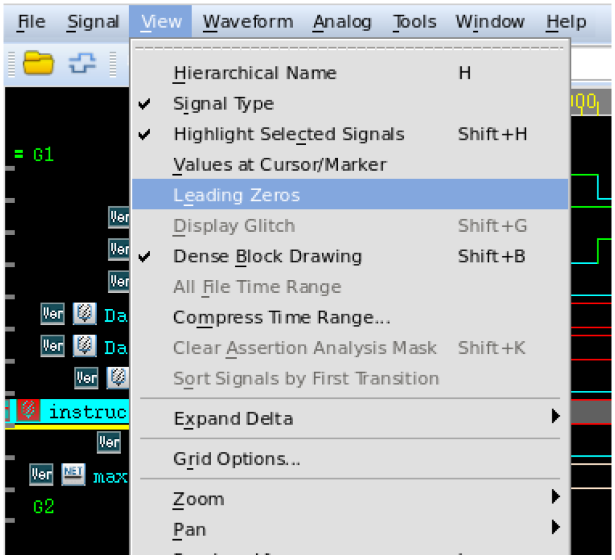
4. If you want to change the radix of the signal (The default radix is hexadecimal):
Select the signal, right-click it, choose "Set Radix", and choose the radix.



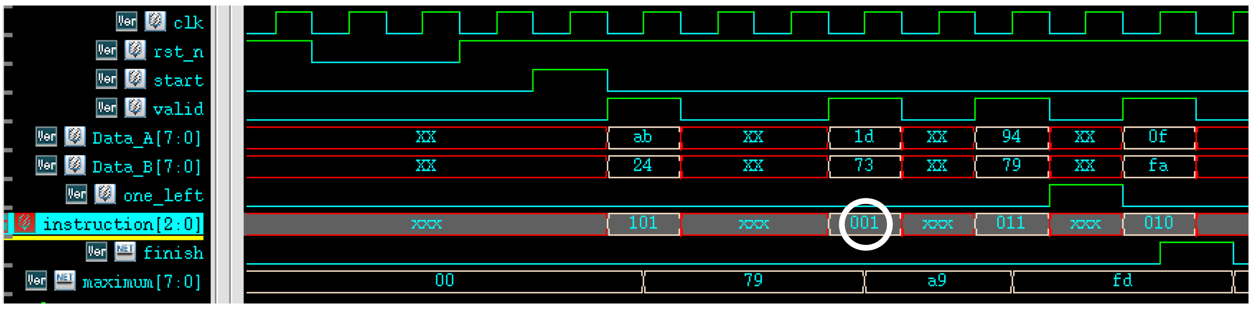
For this example, you can see that *instruction* is changed to binary numbers.



If you want to let leading zero fill the value of all signals, click "View" and click "Leading Zeros"


File Signal View Waveform Analog Tools Window Help
Hierarchical Name H
Signal Type
Highlight Selected Signals Shift+H
Values at Cursor/Marker
Leading Zeros
Display Glitch Shift+G
Dense Block Drawing Shift+B
All File Time Range
Compress Time Range...
Clear Assertion Analysis Mask Shift+K
Sort Signals by First Transition
Expand Delta
Grid Options...
Zoom
Pan

For example, for the signal, *instruction*, you will see that 1 becomes 001.



5. If you find some errors in your Verilog code while you are watching the waveform, and then you modify the Verilog code, you can press ”Shift + L” to update the waveform based on the updated Verilog code while your tool is running.

Grading

1. Problem 1 RTL simulation pass (40%) : public cases(20%), hidden cases(20%)
2. Problem 2 RTL simulation pass (60%) : public cases(30%), hidden cases(30%)
3. You have to submit these two files to EECLASS: **<stu_id>_1.v** and **<stu_id>_2.v** for problem 1 and problem 2, respectively. **If you don’t change the filenames, 10 points will be deducted.** Don’t compress these two files.
4. You **must not** use “initial” block and delay statements (e.g., #5).
5. You need to satisfy all “**must/must not**” in previous descriptions, or you will get 50% discount on your project.
6. **Plagiarism is strictly forbidden. If you do that, you and the classmates who you plagiarize will get -20 points in final score of this course on this project.**
7. **Overdue is forbidden. If so, you will get 0 points on this project.**