Lab2: Design of Arithmetic Logic Unit(ALU)

Instructor: Lih-Yih Chiou

Speaker: Jay

Date: 2024/02/29

EE

LPHPLAB VLSI Design LAB

Outline

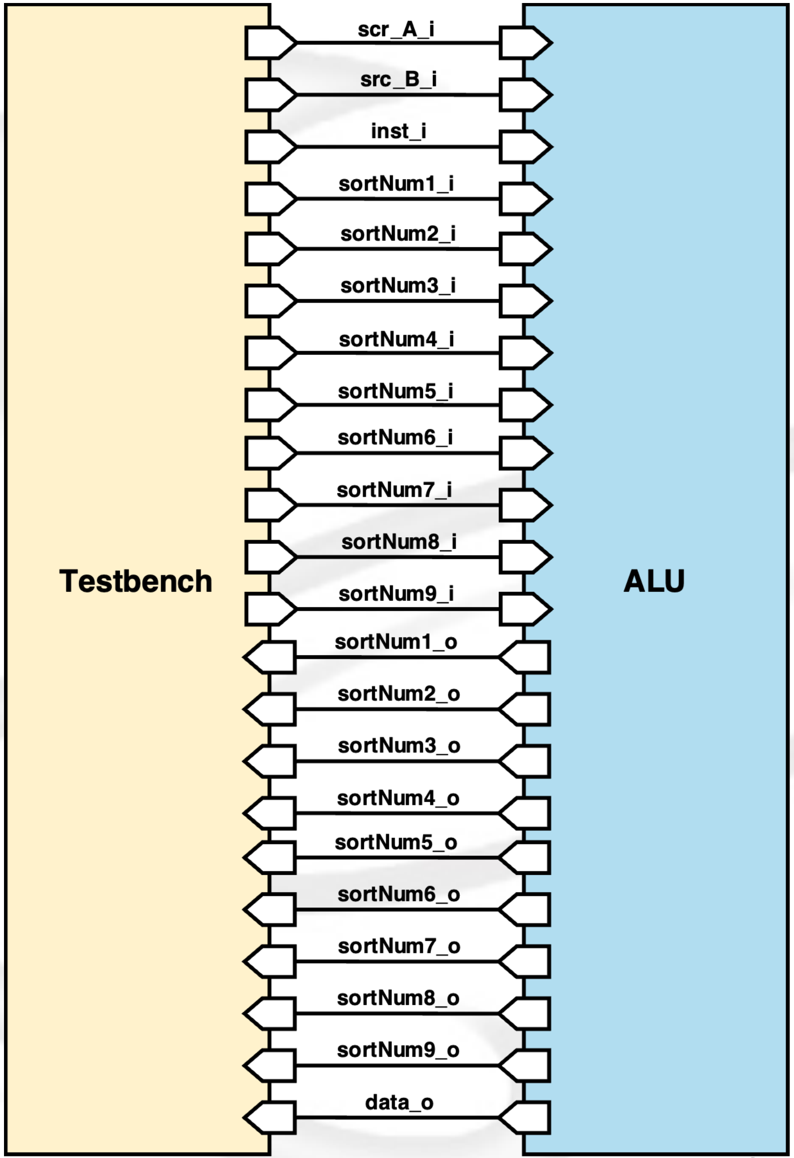
- Introduction
- Hardware description
  - Block diagram
  - Instruction
  - I/O Information
- Lab2 Implementation
  - Data format
  - Saturation & Rounding
  - Piecewise linear approximate for the tanh function
- Criteria
  - Grading policy
  - Simulation Result
  - Requirement & file format

Introduction

- Arithmetic logic unit (ALU)
  - Is one of the components of a computer processor.
- In this lab, you're going to design an ALU with some special instructions. Use the ALU to compute input data to get the correct results .

Hardware description

- Block Diagram



Hardware description

Instruction

Operation	inst_i	Desc.	Note
Signed Addition (Fixed Point)	3’b000	data_o = scr_A_i + scr_B_i	Output saturation and rounding is needed. **(Rounding policy: round to the nearest, ties to even)**
Signed Subtraction (Fixed Point)	3’b001	data_o = scr_A_i - scr_B_i	
Signed Multiplication (Fixed Point)	3’b010	data_o = scr_A_i * scr_B_i	
GeLU	3’b011	data_o = GeLU(scr_A_i)	Output is needed; only scr_A_i is used. **(Rounding policy: round to the nearest, ties to even)**
CLZ	3’b100	Count leading zero bits	Only scr_A_i is used.
Sort nine numbers	3’b101	Sort nine numbers in ascending order.	

Hardware description

□ I/O Information

Signal	I/O	width	Desc.
src_A_i	I	16	For instruction 000~011, signed input data with 2's complement representation. (6-bit signed integer + 10-bit fraction) For instruction 100, 16-bit number.
src_B_i	I	16	
inst_i	I	3	Operation code select which operation to be executed
sortNum*_i	I	8	Unsigned 8-bit number
sortNum*_o	O	8	Unsigned 8-bit number
data_o	O	16	For instruction 000~011, signed input data with 2's complement representation. (6-bit signed integer + 10-bit fraction) For instruction 100, 16-bit number. For instruction 101, don't care.

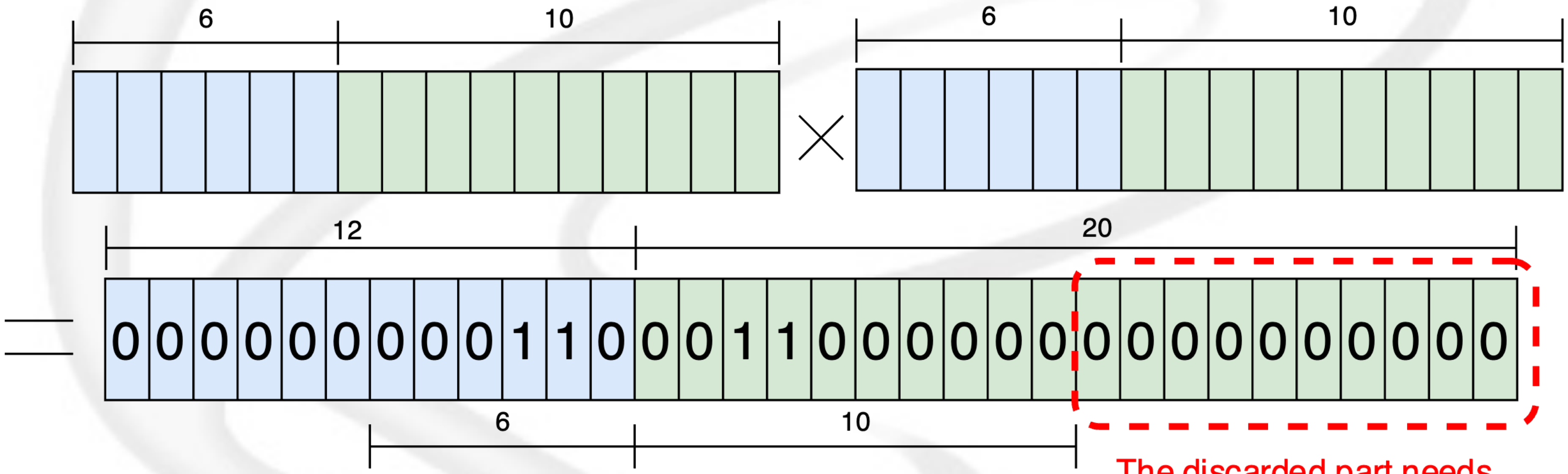
Lab2 Implementation

- Why Fixed point?
  - Floating-point format in Verilog is not synthesizable; you need to use fixed-point or IEEE 754 floating-point to represent it.
    - Ex: “a = b * 0.123;” is not synthesizable.
- Fixed point
  - Designer can decide the format.
  - It often requires less hardware resources compared to floating-point arithmetic.
  - In this lab, for instruction 000~011, the input data is in fixed-point format.

Lab2 Implementation

- Fixed point multiplication
  - Ex:1.5 * 4.125 = 000001_1000000000 * 000100_0010000000

    = 000000_000110_0011000000_0000000000
  - We only want the 6-bit integer and the 10-bit fractional part of the answer.



The discarded part needs to be rounded

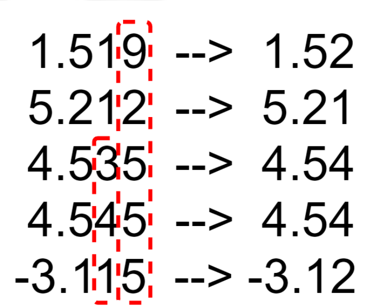
= 6.1875

Lab2 Implementation

□ Saturation & Rounding

- ➔ For instructions 3’b000~3’b010, If the output value exceeds the maximum, value of 16-bit representation, **use the maximum value as output, and vice versa.**
- ➔ For instructions 3’b010 to 3’b011, you need to round the result to the nearest value.
  - ◆ **Rounding policy: Round to the Nearest, ties to Even (RNE)**

Ex: To round to the nearest, to the second decimal place.

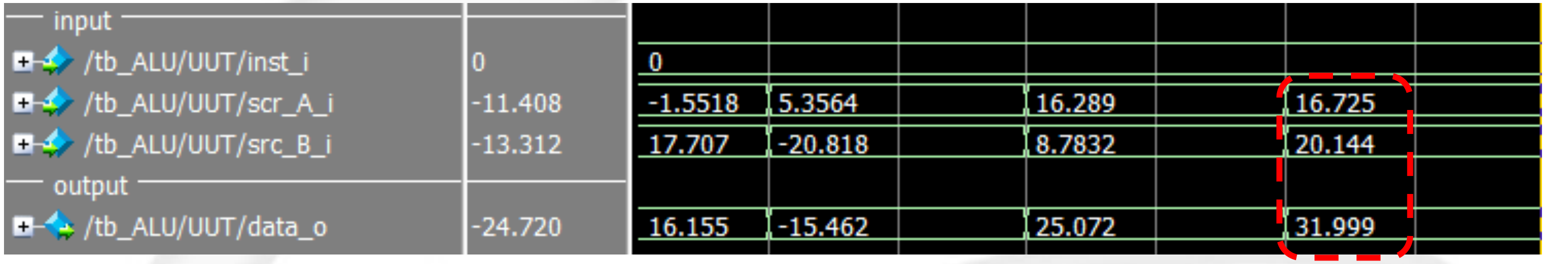


```
 1.519 -->  1.52
 5.212 -->  5.21
 4.535 -->  4.54
 4.545 -->  4.54
-3.115 --> -3.12
```

1. Greater than 5, round up
2. less than 5, round down
3. equal to 5, look at the preceding digit –
   if it's odd, round up; if it's even, round down

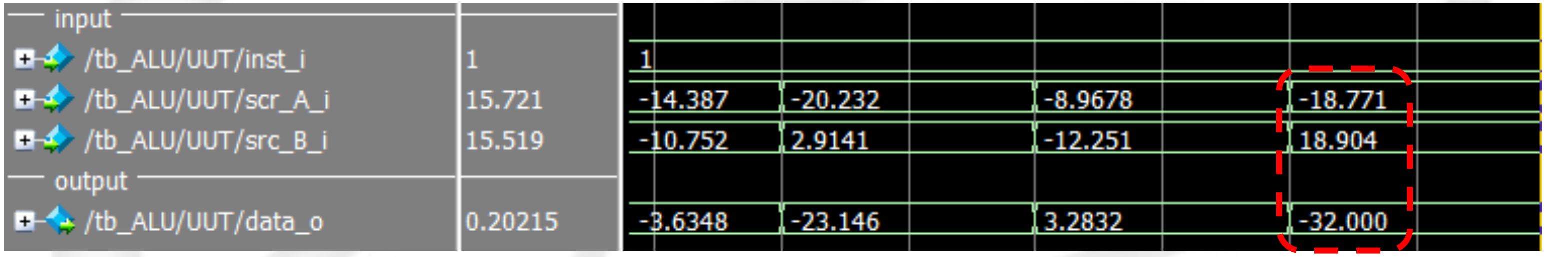
Lab2 Implementation

- Here are some examples of output saturation
  - Addition



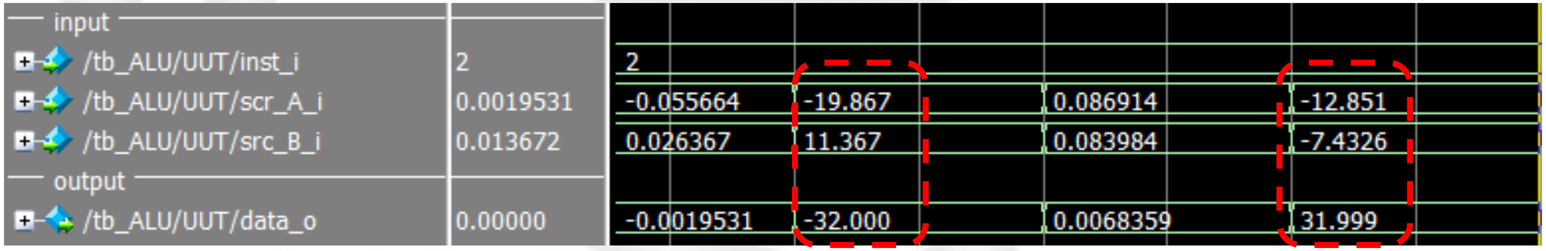
```
input
/tb_ALU/UUT/inst_i    0         0
/tb_ALU/UUT/scr_A_i   -11.408   -1.5518  5.3564   16.289   16.725
/tb_ALU/UUT/src_B_i   -13.312   17.707   -20.818  8.7832   20.144
output
/tb_ALU/UUT/data_o    -24.720   16.155   -15.462  25.072   31.999
```

  - Subtraction



```
input
/tb_ALU/UUT/inst_i    1         1
/tb_ALU/UUT/scr_A_i   15.721    -14.387  -20.232  -8.9678  -18.771
/tb_ALU/UUT/src_B_i   15.519    -10.752  2.9141   -12.251  18.904
output
/tb_ALU/UUT/data_o    0.20215   -3.6348  -23.146  3.2832   -32.000
```

  - Multiplication

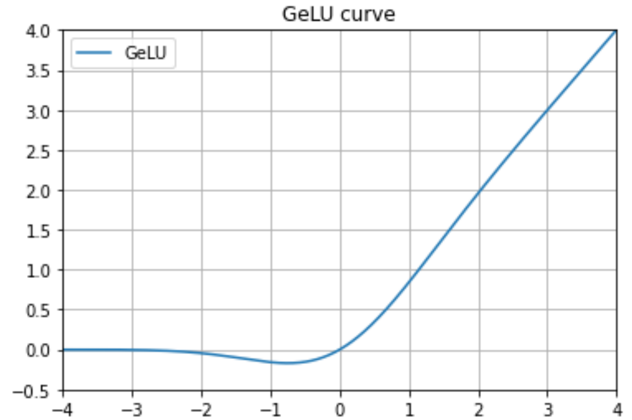


```
input
/tb_ALU/UUT/inst_i    2           2
/tb_ALU/UUT/scr_A_i   0.0019531   -0.055664   -19.867  0.086914   -12.851
/tb_ALU/UUT/src_B_i   0.013672    0.026367    11.367   0.083984   -7.4326
output
/tb_ALU/UUT/data_o    0.00000     -0.0019531  -32.000  0.0068359  31.999
```

Lab2 Implementation

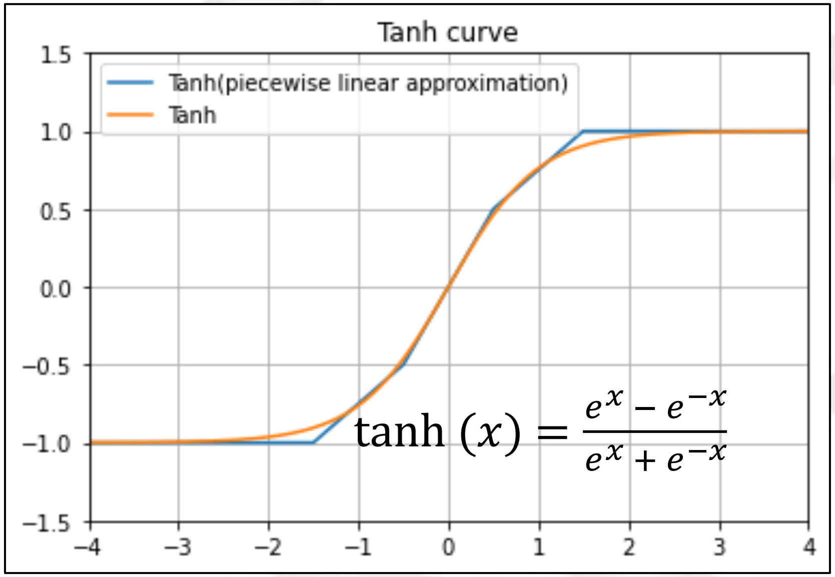
- GeLU (Gaussian Error Linear Unit)
  - Compared with ReLU, the GELU function has a non-zero gradient at x = 0, allowing the network to learn in this region.
  - We approximate the GeLU function using the Tanh function.

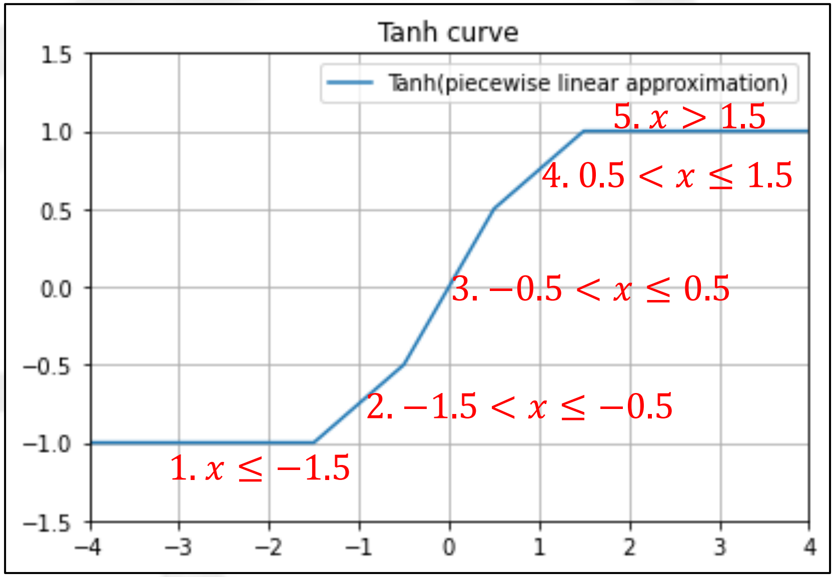
$$\mathrm{GeLU}\ (x)$$
$$\approx 0.5 * x * \left[1 + \tanh\left(\sqrt{\frac{2}{\pi}} * x * (1 + 0.044715 * x^2)\right)\right]$$
$$\approx 0.5 * x * \left[1 + \tanh\left(0.7978515625 * x * (1 + 0.044921875 * x^2)\right)\right]$$



Lab2 Implementation

- Piecewise linear approximation
  - We use piecewise linear approximation to implement the tanh function.
  - We divide the curve into 5 segments to compute the output.





Lab2 Implementation

- Rounding the result of the GELU function
  - ➔ Round to the nearest number with ties rounding to the even number (16-bits, 6-bit integer + 10-bit fraction).
    - Round the value before the tanh…①
    - Round the value after the tanh …②
    - Round the final value …③

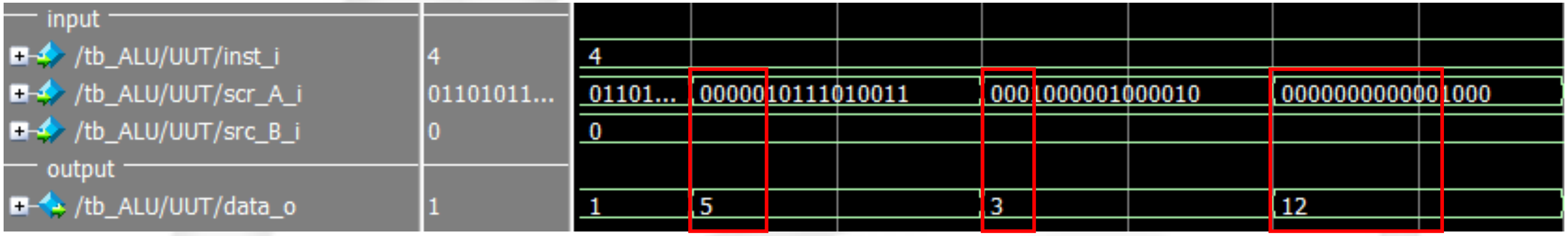
$$\mathrm{GeLU}\ (x)$$

$$\approx \boxed{0.5 * x * \left[1 + \boxed{\tanh\left(\boxed{0.7978515625 * x * (1 + 0.044921875 * x^{2})}\right)}\right]}$$

Lab2 Implementation

- □ CLZ (Count Leading Zero bits)

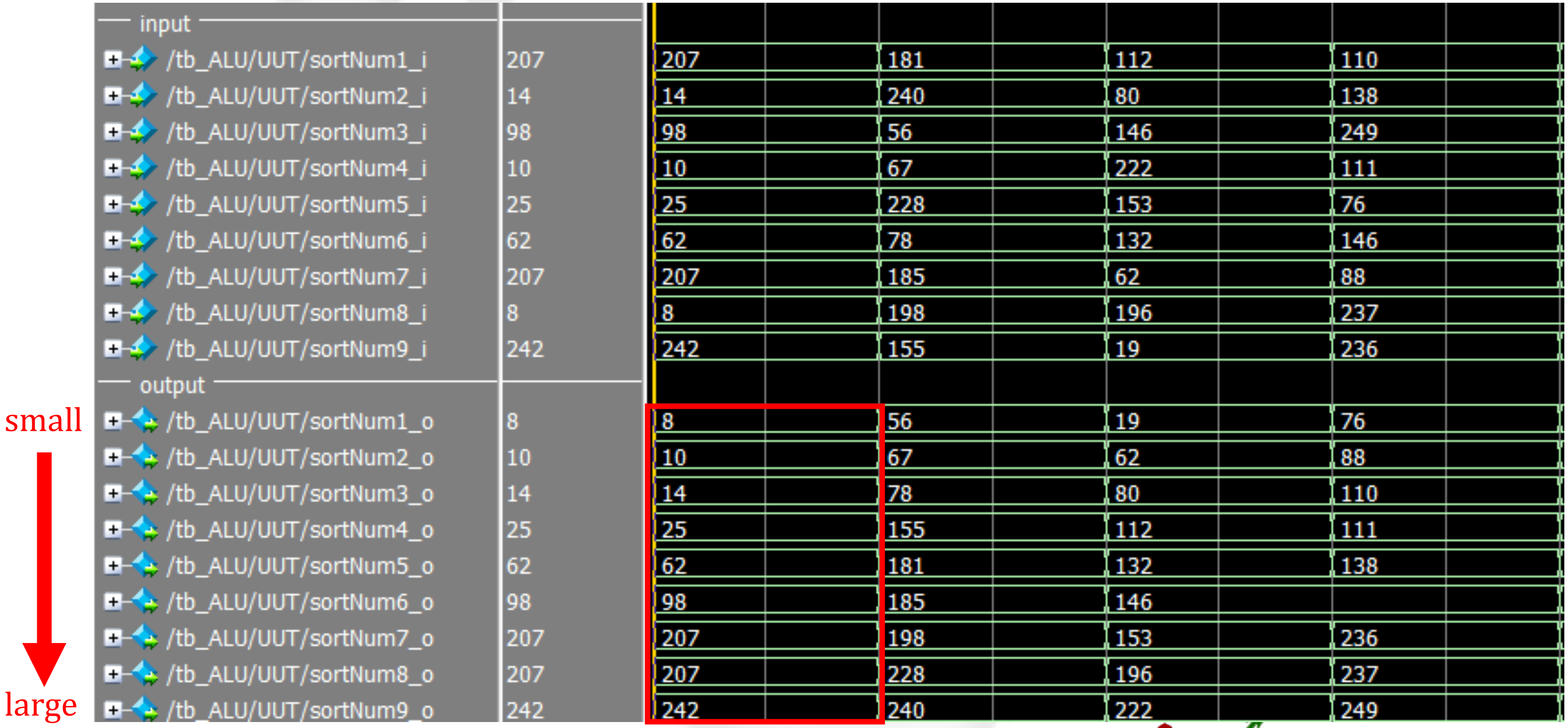
  - ➔ Ex: if input = 8’b0001_1101, then output = 3



```
input
/tb_ALU/UUT/inst_i     4            4
/tb_ALU/UUT/scr_A_i    01101011...  01101...  0000010111010011  0001000001000010  0000000000001000
/tb_ALU/UUT/src_B_i    0            0
output
/tb_ALU/UUT/data_o     1            1         5                 3                 12
```

Lab2 Implementation

- Sort 9 numbers
  - ➔ The inputs are 8-bit unsigned numbers.
  - ➔ **Avoid the use of the “for-loop” in Verilog to solve this problem.**



input					
/tb_ALU/UUT/sortNum1_i	207	207	181	112	110
/tb_ALU/UUT/sortNum2_i	14	14	240	80	138
/tb_ALU/UUT/sortNum3_i	98	98	56	146	249
/tb_ALU/UUT/sortNum4_i	10	10	67	222	111
/tb_ALU/UUT/sortNum5_i	25	25	228	153	76
/tb_ALU/UUT/sortNum6_i	62	62	78	132	146
/tb_ALU/UUT/sortNum7_i	207	207	185	62	88
/tb_ALU/UUT/sortNum8_i	8	8	198	196	237
/tb_ALU/UUT/sortNum9_i	242	242	155	19	236
output					
/tb_ALU/UUT/sortNum1_o	8	8	56	19	76
/tb_ALU/UUT/sortNum2_o	10	10	67	62	88
/tb_ALU/UUT/sortNum3_o	14	14	78	80	110
/tb_ALU/UUT/sortNum4_o	25	25	155	112	111
/tb_ALU/UUT/sortNum5_o	62	62	181	132	138
/tb_ALU/UUT/sortNum6_o	98	98	185	146	
/tb_ALU/UUT/sortNum7_o	207	207	198	153	236
/tb_ALU/UUT/sortNum8_o	207	207	228	196	237
/tb_ALU/UUT/sortNum9_o	242	242	240	222	249

Criteria

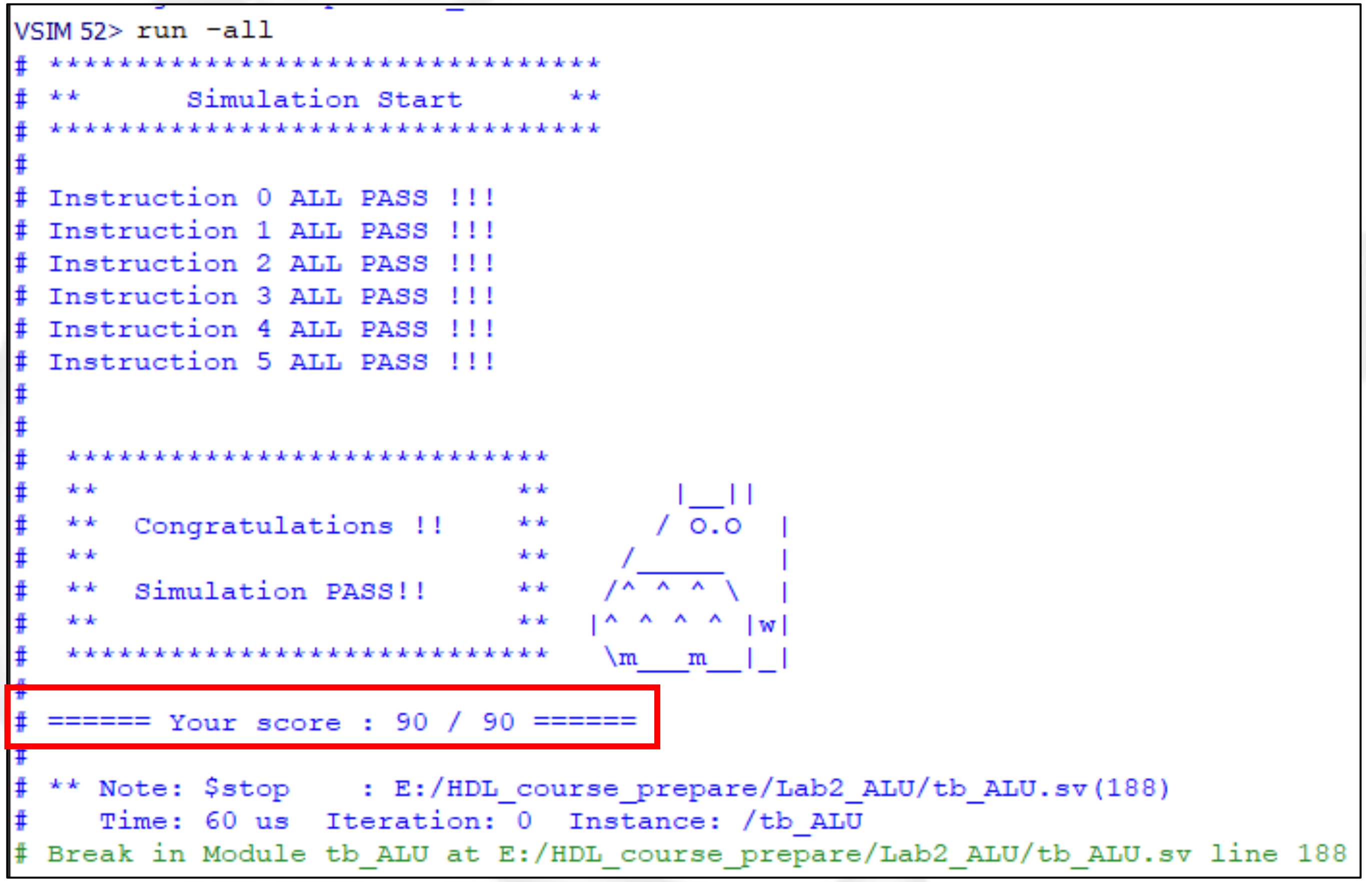
- Grading policy(100%)
  - Lab2
    - Simulation pass (90%)

Signed Addition	(16%)
Signed Subtraction	(16%)
Signed Multiplication	(16%)
GeLU	(10%)
CLZ	(16%)
Sort nine numbers	(16%)

- Report (10%)

Criteria

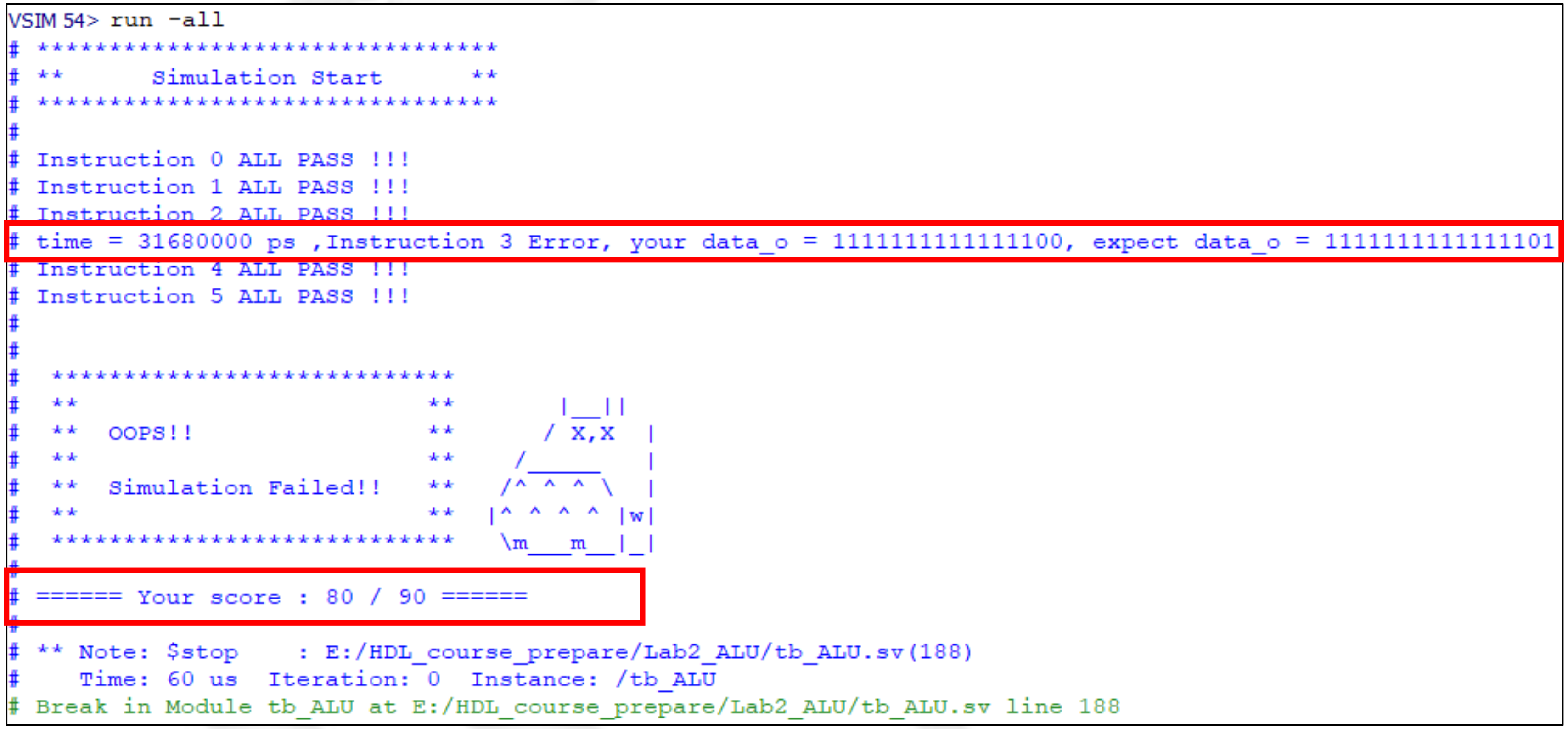
- Simulation result
  - Pass



```
VSIM 52> run -all
# ********************************
# **      Simulation Start      **
# ********************************
#
# Instruction 0 ALL PASS !!!
# Instruction 1 ALL PASS !!!
# Instruction 2 ALL PASS !!!
# Instruction 3 ALL PASS !!!
# Instruction 4 ALL PASS !!!
# Instruction 5 ALL PASS !!!
#
#
#  ****************************
#  **                        **       |__||
#  **  Congratulations !!    **      / O.O  |
#  **                        **    /______  |
#  **  Simulation PASS!!     **   /^ ^ ^ \  |
#  **                        **  |^ ^ ^ ^ |w|
#  ****************************   \m___m__|_|
#
# ====== Your score : 90 / 90 ======
#
# ** Note: $stop    : E:/HDL_course_prepare/Lab2_ALU/tb_ALU.sv(188)
#    Time: 60 us  Iteration: 0  Instance: /tb_ALU
# Break in Module tb_ALU at E:/HDL_course_prepare/Lab2_ALU/tb_ALU.sv line 188
```

Criteria

- Simulation result
  - Failed



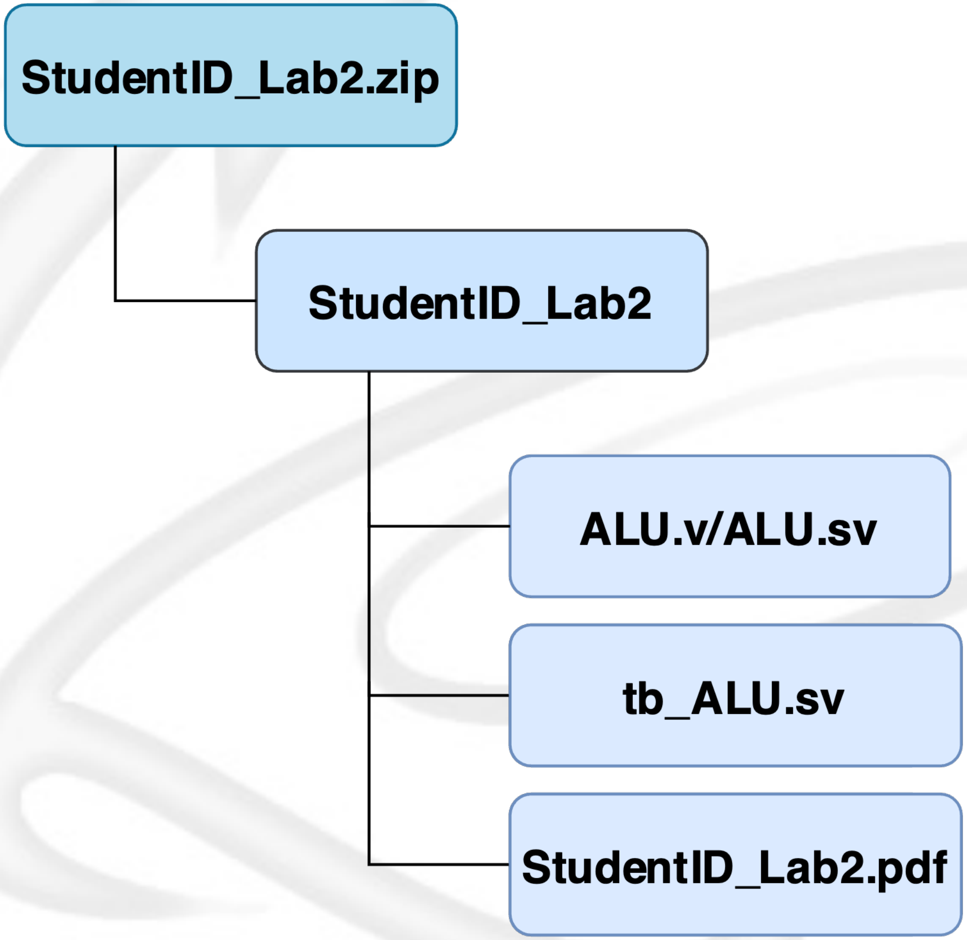
```
VSIM 54> run -all
# ********************************
# **      Simulation Start      **
# ********************************
#
# Instruction 0 ALL PASS !!!
# Instruction 1 ALL PASS !!!
# Instruction 2 ALL PASS !!!
# time = 31680000 ps ,Instruction 3 Error, your data_o = 1111111111111100, expect data_o = 1111111111111101
# Instruction 4 ALL PASS !!!
# Instruction 5 ALL PASS !!!
#
#
#  ****************************
#  **                        **       |__||
#  **  OOPS!!                **      / X,X  |
#  **                        **    /______  |
#  **  Simulation Failed!!   **   /^ ^ ^ \  |
#  **                        **  |^ ^ ^ ^ |w|
#  ****************************   \m___m__|_|
#
# ====== Your score : 80 / 90 ======
#
# ** Note: $stop    : E:/HDL_course_prepare/Lab2_ALU/tb_ALU.sv(188)
#    Time: 60 us  Iteration: 0  Instance: /tb_ALU
# Break in Module tb_ALU at E:/HDL_course_prepare/Lab2_ALU/tb_ALU.sv line 188
```

Lab2 Requirement & file format

- You must finish ALU.v/.sv and pass all patterns
- For Lab2, you need to submit
  - ALU.v / ALU.sv
  - tb_ALU.sv
  - StudentID_Lab2.pdf
- Deadline:2024/03/07 08:59 (No late submission)

Lab2 Requirement & file format

- Homework submission must follow the file hierarchy!



StudentID_Lab2.zip

StudentID_Lab2

ALU.v/ALU.sv

tb_ALU.sv

StudentID_Lab2.pdf

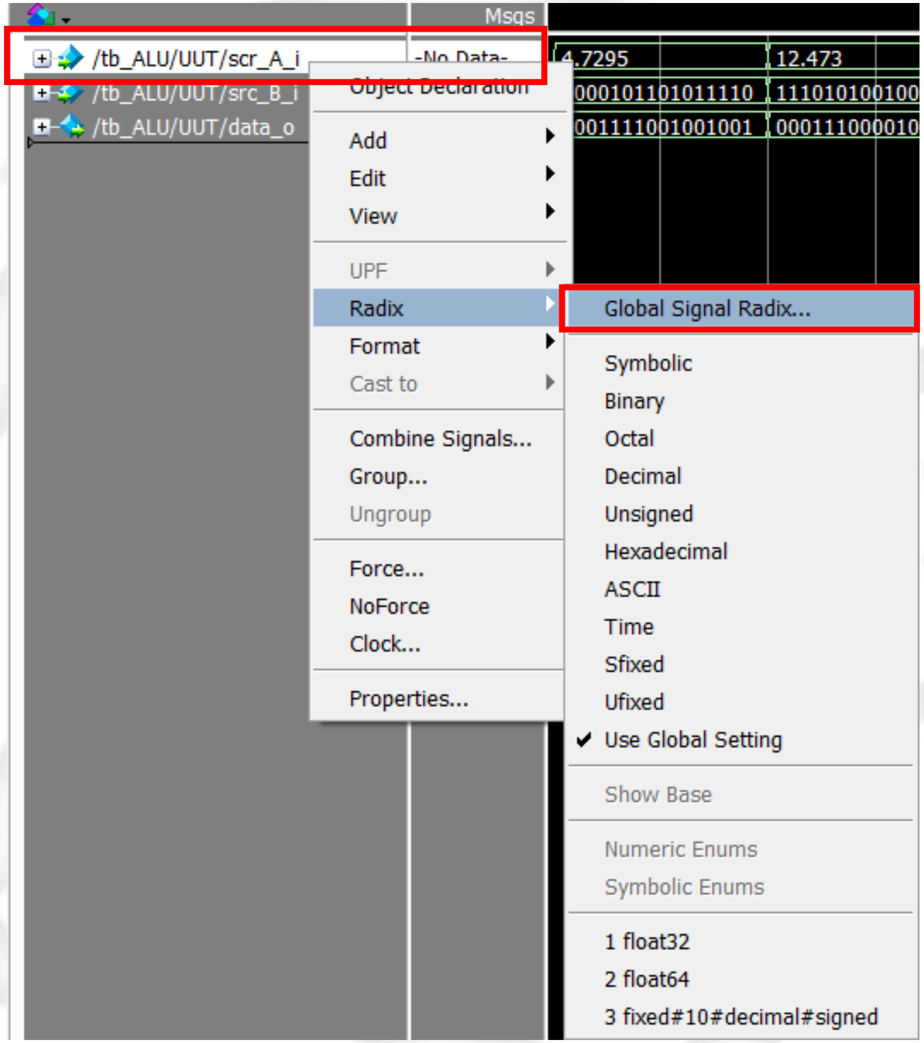
Thanks for listening

How to view fixed-point format in ModelSim

How to view fixed-point format in ModelSim

- Step1

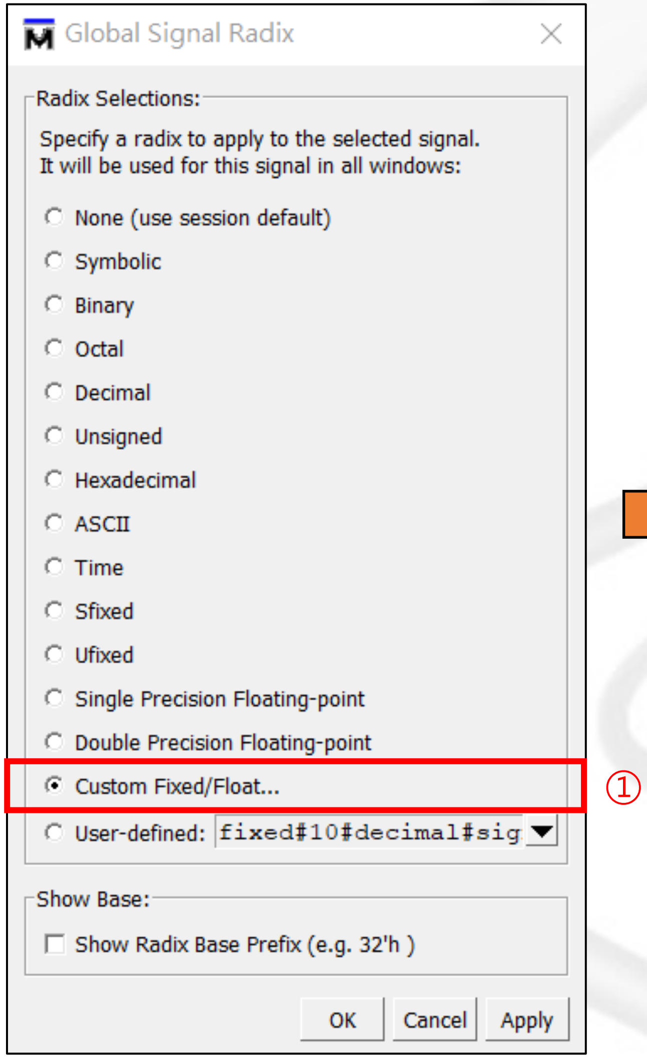
① Right Click



②

How to view fixed-point format in ModelSim

Step2



Global Signal Radix

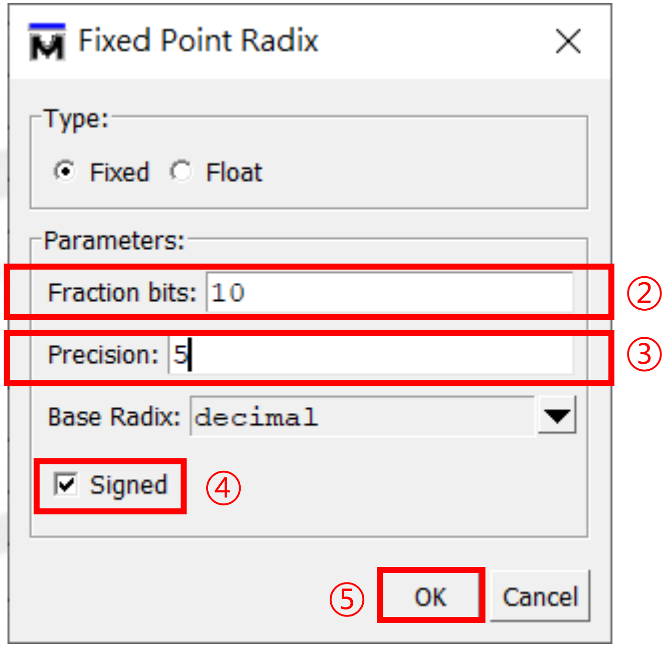
Radix Selections:

Specify a radix to apply to the selected signal.
It will be used for this signal in all windows:

○ None (use session default)
○ Symbolic
○ Binary
○ Octal
○ Decimal
○ Unsigned
○ Hexadecimal
○ ASCII
○ Time
○ Sfixed
○ Ufixed
○ Single Precision Floating-point
○ Double Precision Floating-point
● Custom Fixed/Float... ①
○ User-defined: fixed#10#decimal#sig

Show Base:

□ Show Radix Base Prefix (e.g. 32'h)



Fixed Point Radix

Type:

● Fixed ○ Float

Parameters:

Fraction bits: 10 ②

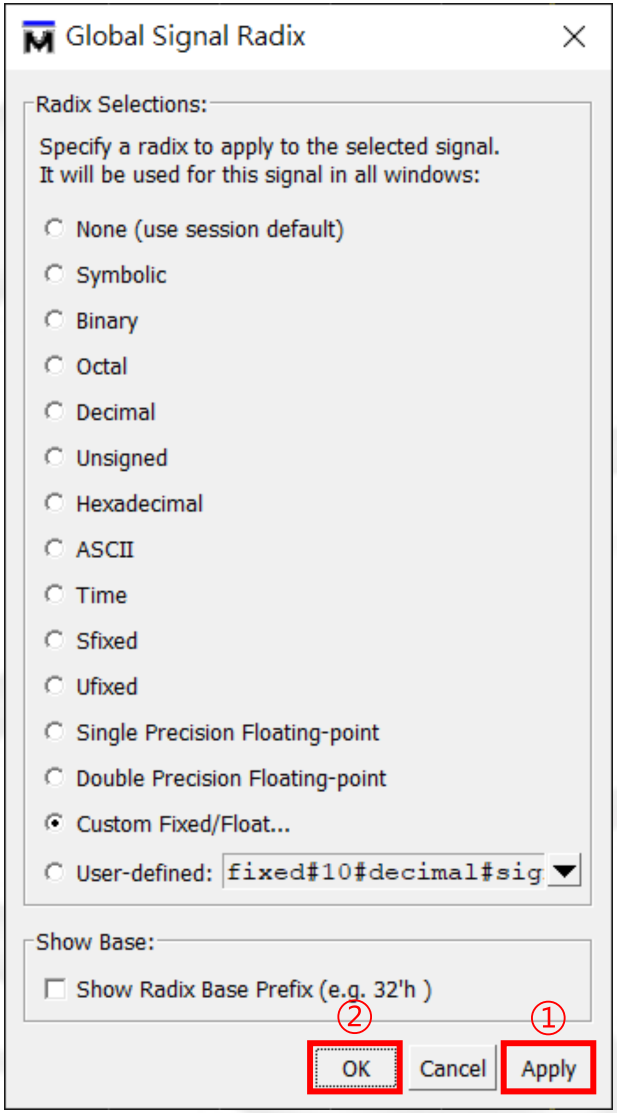
Precision: 5 ③

Base Radix: decimal

☑ Signed ④

How to view fixed-point format in ModelSim

- Step3



Global Signal Radix

Radix Selections:

Specify a radix to apply to the selected signal.
It will be used for this signal in all windows:

- ○ None (use session default)
- ○ Symbolic
- ○ Binary
- ○ Octal
- ○ Decimal
- ○ Unsigned
- ○ Hexadecimal
- ○ ASCII
- ○ Time
- ○ Sfixed
- ○ Ufixed
- ○ Single Precision Floating-point
- ○ Double Precision Floating-point
- ● Custom Fixed/Float...
- ○ User-defined: fixed#10#decimal#sig

Show Base:

- □ Show Radix Base Prefix (e.g. 32'h)