IC Design
HW3 Tutorial

Chia-Heng Lee
Professor：Tzi-Dar Chiueh
2020/12/04

Outline

- Introduction to Verilog
  - Module
  - Value & number
  - Data type
- Ncverilog simulation & nWave tool

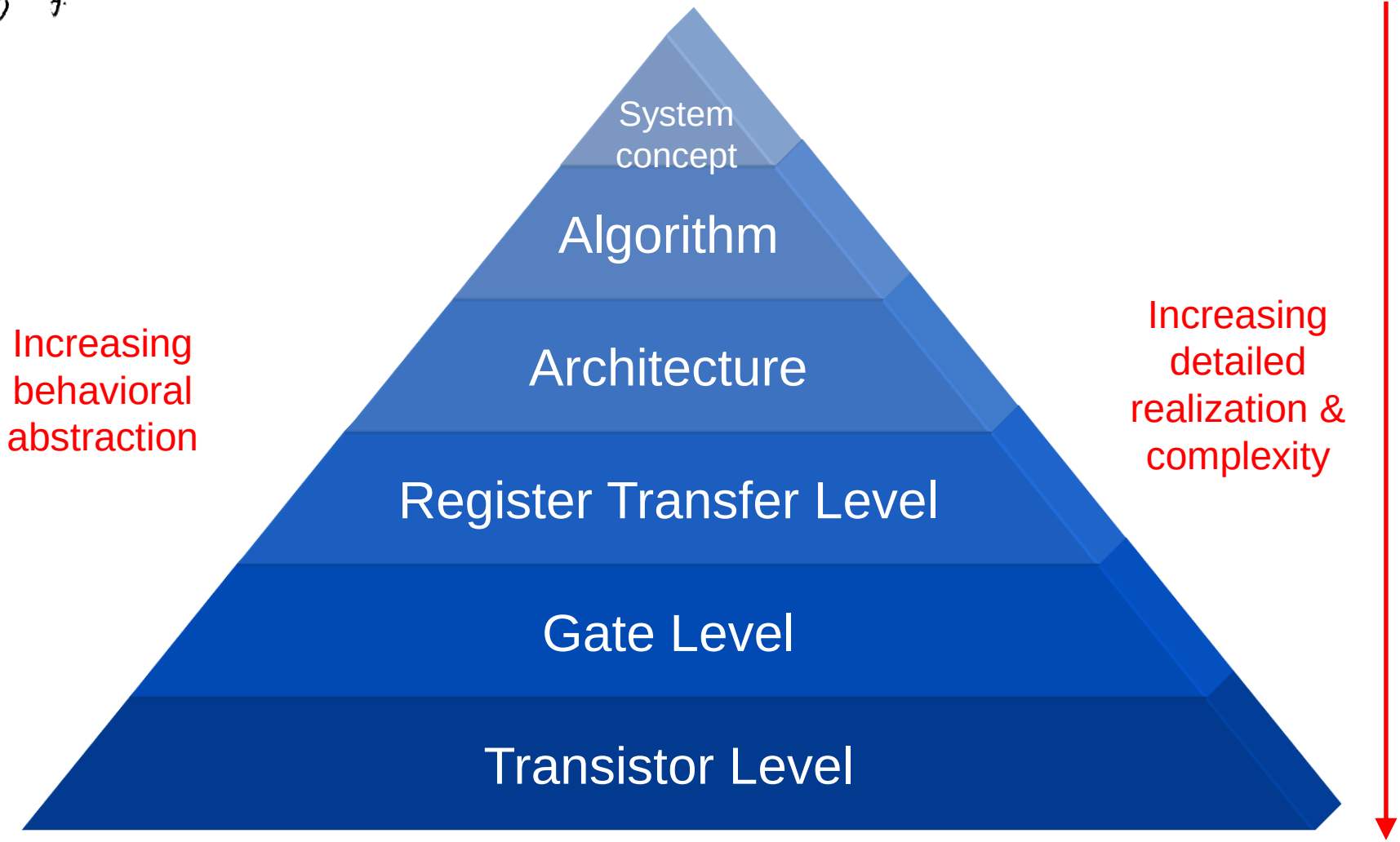
Introduction to Verilog

Module
Value & Number
Data Type

What is Verilog

- Verilog is a Hardware Description Language (HDL)
  - Describe digital electronic system at multiple levels of abstraction
  - Model the timing
  - Express the *concurrency* of the system operation
  - Test the system

Behavioral Model of Circuits

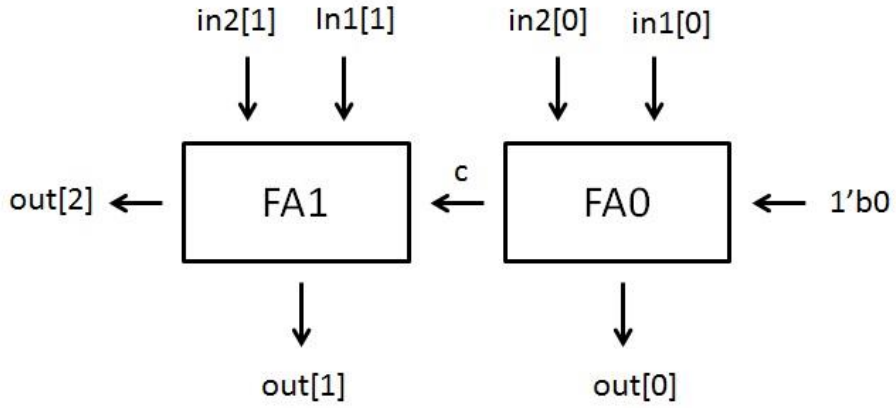


Verilog-Supported Levels of Abstraction

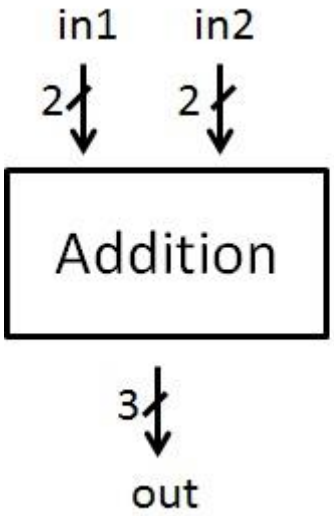
- Behavioral
  - Structural and procedural like the C programming language
  - Describe algorithm level and RTL level Verilog models
- Register Transfer Level (RTL)
  - Describe the flow of data between registers and how a design process these data.
- Structural
  - Describe gate-level and switch-level circuits.

The Verilog Module

- Basic building blocks .
- Begin with **module**,
  end with **endmodule**

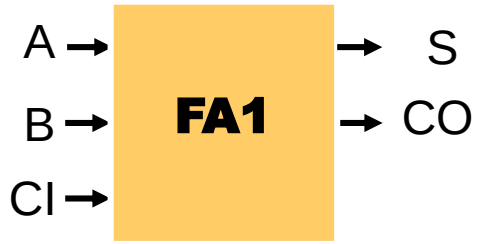



```
module <module name>(<port lists>);

        //module description

endmodule
```



Module Ports

- Modules communicate through ports
  - input
  - output
  - inout (bidirectional)




```
module FA1(CO,S,A,B,CI);
    output S,CO;
    input A,B,CI;    //module description
endmodule
```

4-Value Logic System

- 0
  - zero, false, low
- 1
  - one, true, high
- Z
  - high impedance, floating
- X
  - unknown, occurs at un-initialized storage elements or un-resolvable logic conflicts

Value and Number

- [[<size>]'<radix>]<value>
  - Size
    - The size in bits
    - Default size is 32 bits
  - Radix
    - b (binary), o (octal), d (decimal), h (hexadecimal)
    - Default radix is decimal
  - Value
    - Any legal number in selected radix, including "x" and "z"
- Radix and value are case-insensitive

Value and Number – Examples

```
4'b1001   // 4-bit binary
5'd3      // 5-bit decimal
12'h7ff   // 12-bit hexadecimal
3'bx10    // 3-bit binary with
             unknown MSB
4'b101x   // 4-bit binary with
             unknown LSB
12'hx     // 12-bit unknown
-8'd6     // phrase as -(8'd6)
```

- underline usage
  - 16'b0001_0101_0001_1111
  - 32'h12ab_f001
- X and Z is sign-extended
  - Ex. 12-bit a

```
a = 'h x;   // yields xxx
a = 'h 3x;  // yields 03x
a = 'h 0x;  // yields 00x
```

Data Type Classes

- wire
  - **wire** [MSB:LSB] *variables*;
  - input, inout, output are default to be wire.
  - Used to describe combinational circuit!
- reg
  - **reg** [MSB:LSB] *variables*
  - Used to describe combinational or sequential circuit.

Assign a value to wire

- wire
  - “assign “

```
wire a;
assign a = 1’b1;
```

  - Output port

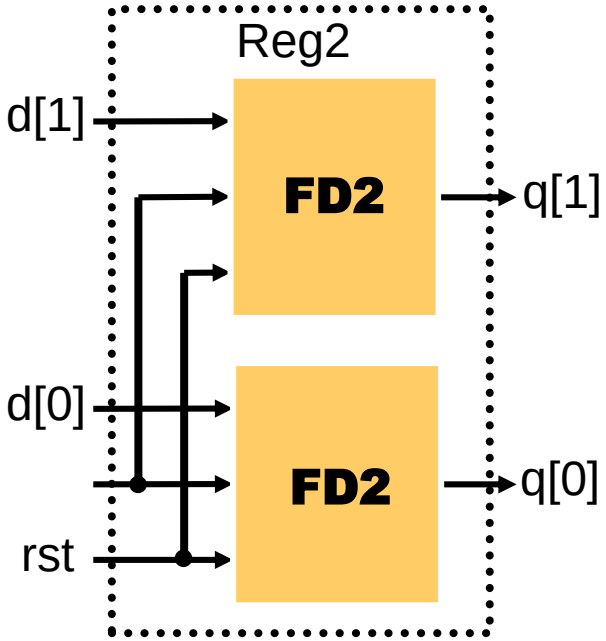
```
wire a;
wire b;
assign b = 1’b0;
NOT n0(a, b);
```

- Every wire can be only assigned once!!!

```
wire a;
wire b;
assign b = 1’b0;
NOT n0(a, b);
assign a = 1’b0; //Wrong!!!
```

Module Instances (1/2)

- Create a higher-level system by connecting lower-level components

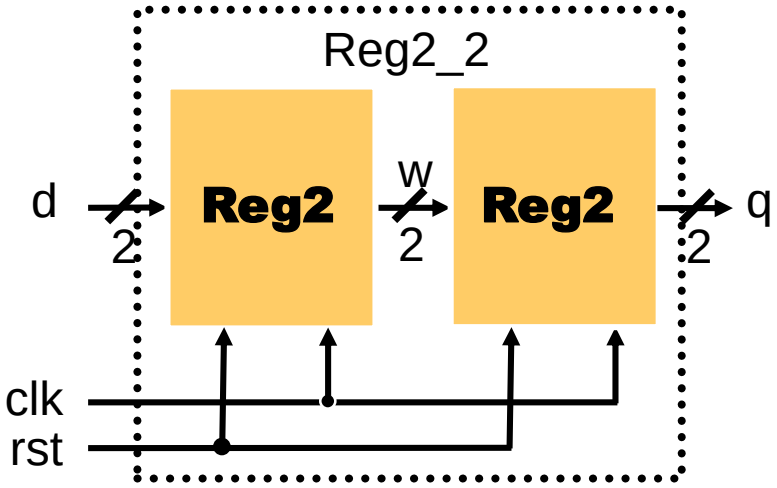



```
module Reg2 (q, d, clk, rst);

   output      [1:0] q;
   input       [1:0] d;
   input       clk, rst;

   FD2 f0(q[0], d[0], clk, rst);
   FD2 f1(.Q(q[1]), .D(d[1]),
          .CLK(clk), .RESET(rst));

endmodule
```

Module name

Instance name

Module Instances (2/2)




```
module Reg2_2 (q, d, clk, rst);

    output      [1:0] q;
    input       [1:0] d;
    input       clk, rst;

    wire        [1:0] w;

    Reg2 r0(w, d, clk, rst);
    Reg2 r1(q, w, clk, rst);

endmodule
```

Net Concatenations

Representation	Meaning
{b[3:0],c[2:0]}	{b[3] ,b[2] ,b[1] ,b[0], c[2] ,c[1] ,c[0]}
{a,b[3:0],w,3'b101}	{a,b[3] ,b[2] ,b[1] ,b[0],w,1'b1,1'b0,1'b1}
{4{w}}	{w,w,w,w}
{b,{3{a,b}}}	{b,a,b,a,b,a,b}

```
wire a;
assign a ={b[3:0],c[2:0]};
```

Standard Cell Library (lib.v)

- Choose what you need
- Compose your circuit according to I/O connections

```
module AN3(Z,A,B,C);

    output Z;
    input A,B,C;

    // netlist
    and g1(Z,A,B,C);

    // specify block, declare local
    // timing constant
    specify
        // delay parameters
        specparam Tp_A_Z = 0.275;
        specparam Tp_B_Z = 0.275;
        specparam Tp_C_Z = 0.275;
        // path delay (full connection)
        ( A *> Z ) = ( Tp_A_Z );
        ( B *> Z ) = ( Tp_B_Z );
        ( C *> Z ) = ( Tp_C_Z );
    endspecify

endmodule
```

Standard Cell Library (lib.v) - 2/2

- IV // not
- AN3
- AN4
- AN2
- EN // xnor
- EN3
- EO // xor
- EO3
- FA1 // full adder
- FD1 // DFF
- FD2
- ND2 // nand
- ND3
- ND4
- NR2 // nor
- NR3
- OR2 // or
- OR3
- OR4
- HA1 // half adder
- MUX21H // 2-to-1 MUX

Notification

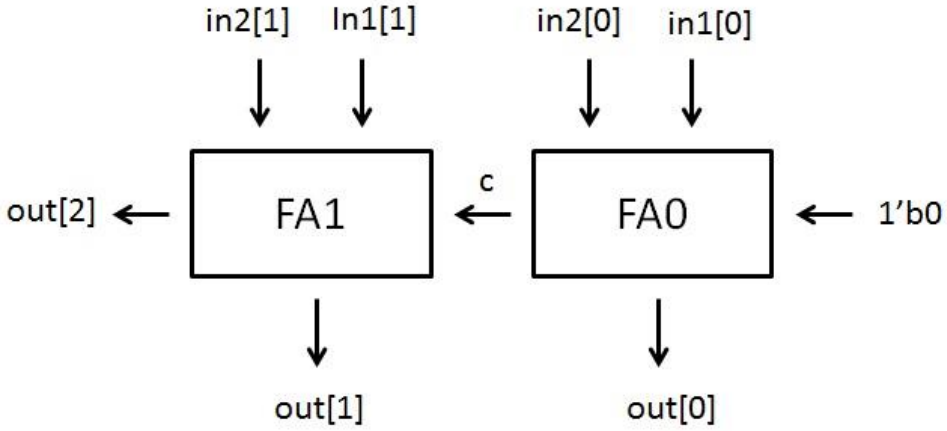
- In this HW, all the logic operation MUST consist of standard cell. You can NOT use logic operators.

~~wire a, b, c;~~
~~assign a = b & c;~~

```
wire a, b, c;
AN2 an(a, b, c);
```

Example

```
module ADDER (out, in1, in2);
    output [2:0] out;
    input [1:0] in1, in2;
    wire c;

    FA1 fa0(c, out[0], in1[0], in2[0], 1'b0);
    FA1 fa1(out[2], out[1], in1[1], in2[1], c);
endmodule
```



Flow chart

tb_sorting.v

lib.v

i0.dat

i1.dat

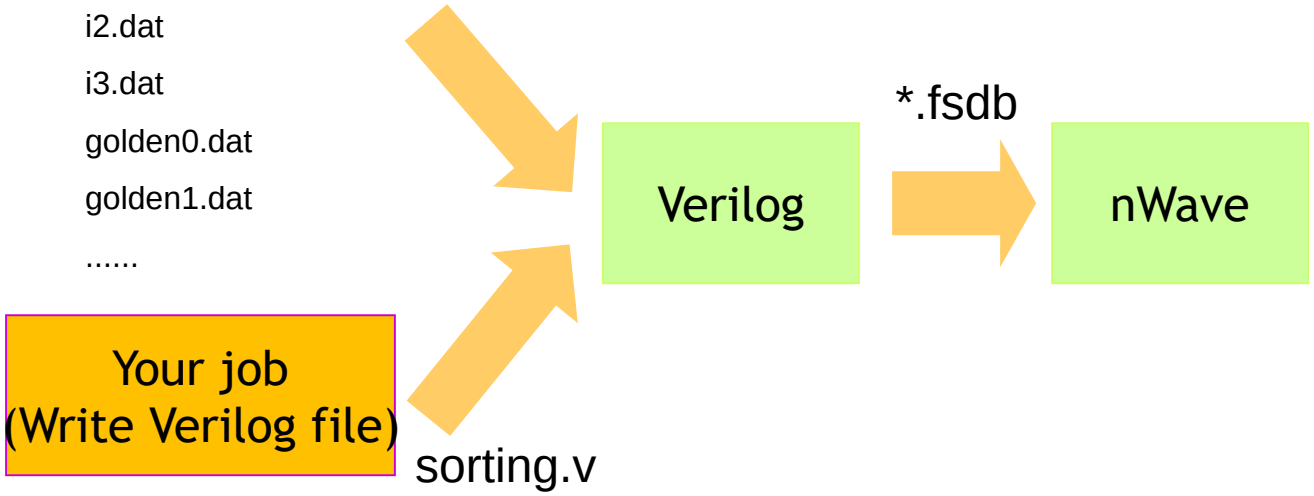
i2.dat

i3.dat

golden0.dat

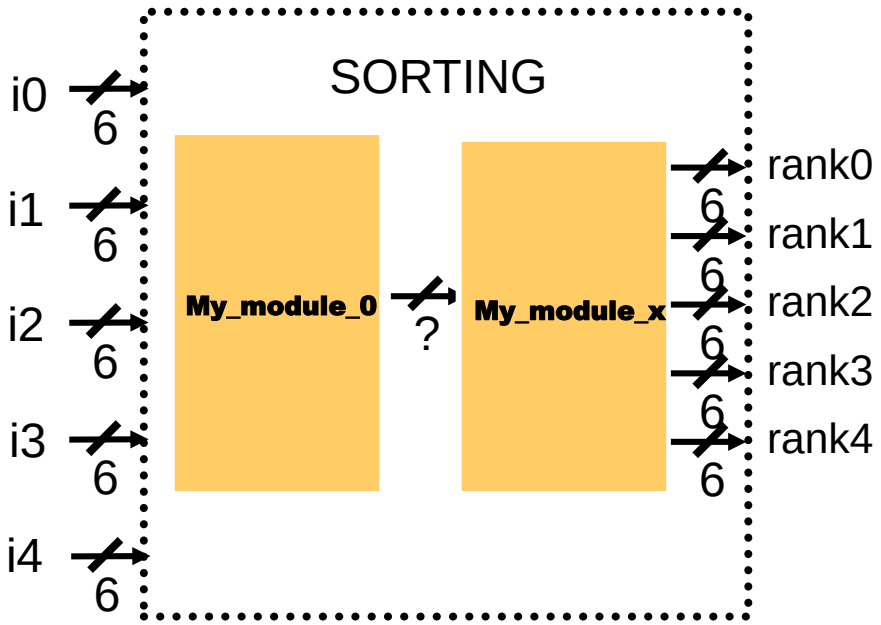
golden1.dat

.....



All these files should be placed under the same folder

Your Job

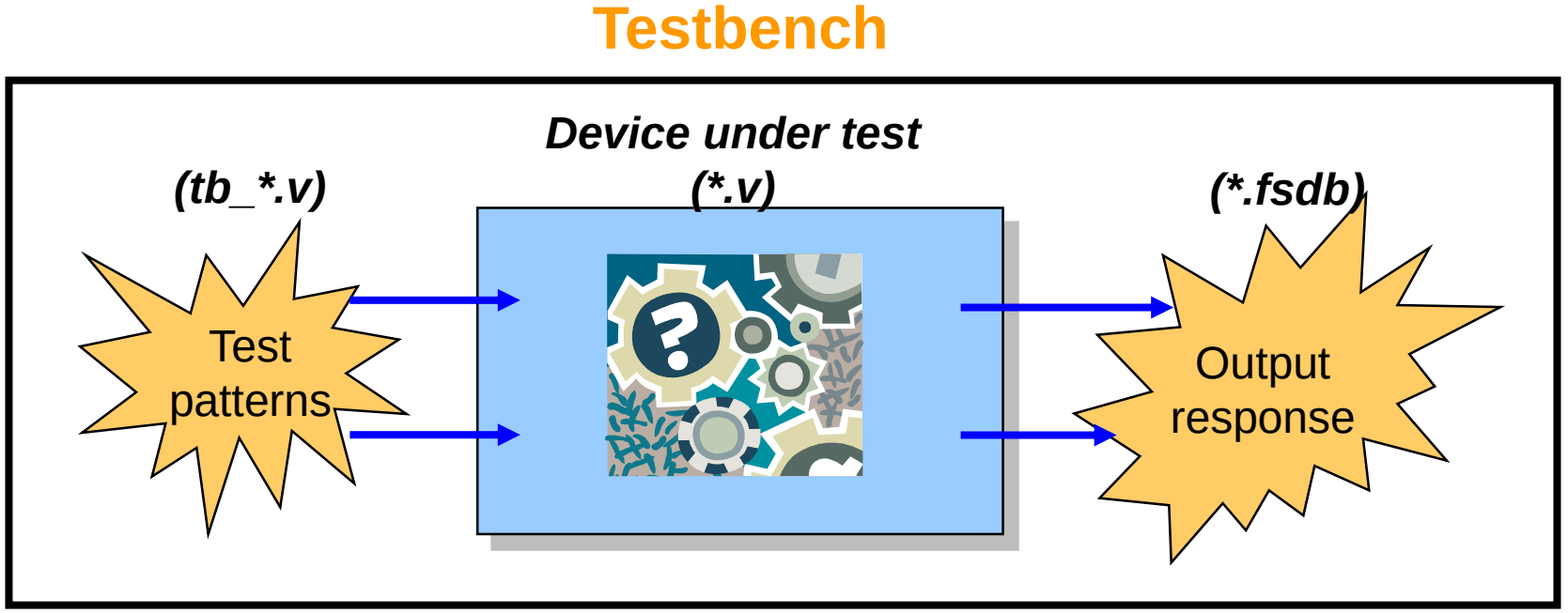



```
module sorting (rank0, rank1, rank2,
rank3, rank4, i0, i1, i2, i3, i4);

    input  [5:0] i0, i1, i2, i3, i4;

    output [5:0] rank0, rank1, rank2,
           rank3, rank4;


// Write your design here

    wire           [?:0] …;

    My_module_0 M0(?, ?, … , ?);

     ……..
    My_module_x Mx(?, ?, … , ?);

endmodule
```

Ncverilog Simulation & nWave Tool

Test and verification your circuit

- By applying input patterns and observing output responses



Compile and debug (1/4)

- Source
  - source /usr/cadence/cshrc
- Include the testbench & lib.v files to run simulation
  - *ncverilog +access+r tb_sorting.v sorting.v lib.v*

Compile and debug (2/4)

```
[r08006@cad29 ~/HW3_2020]$ ncverilog tb_sorting.v sorting.v lib.v +access+r
ncverilog: 15.20-s039: (c) Copyright 1995-2017 Cadence Design Systems, Inc.
```

```
Congratulations! Your critical path is below 5!

Simulation complete via $finish(1) at time 210 US + 0
./tb_sorting_v2.v:137                   $finish;
ncsim> exit
```

Compile and debug (3/4)

```
[r08006@cad29 ~/HW3_2020]$ ncverilog tb_sorting.v sorting.v lib.v +access+r
ncverilog: 15.20-s039: (c) Copyright 1995-2017 Cadence Design Systems, Inc.
```

```
Verdi* : End of traversing the MDAs.
There is some bug in your code. Errors exceed 100.

Simulation complete via $finish(1) at time 2121 NS + 0
./tb_sorting_v2.v:141                   $finish;
ncsim> exit
```

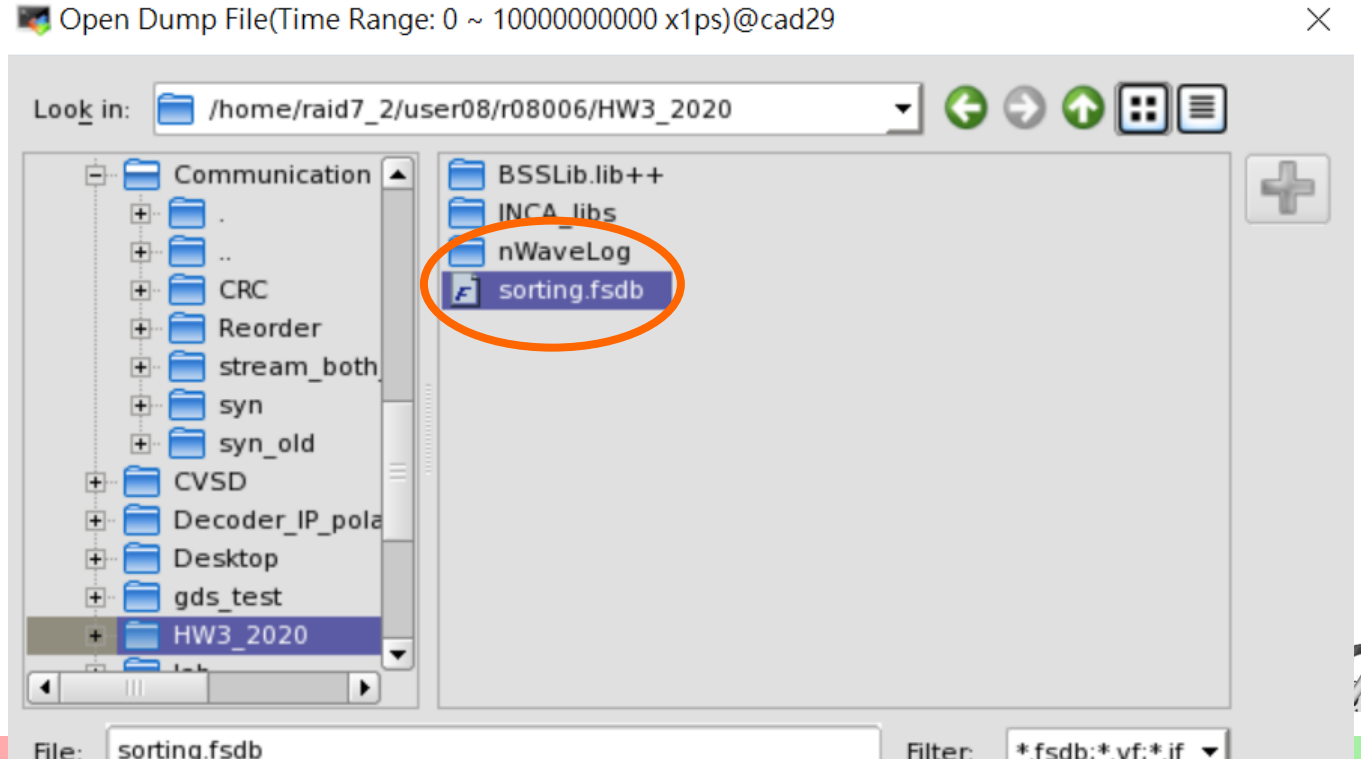
Compile and debug (4/4)

- Use tb_sorting_pattern.v as your testbench.

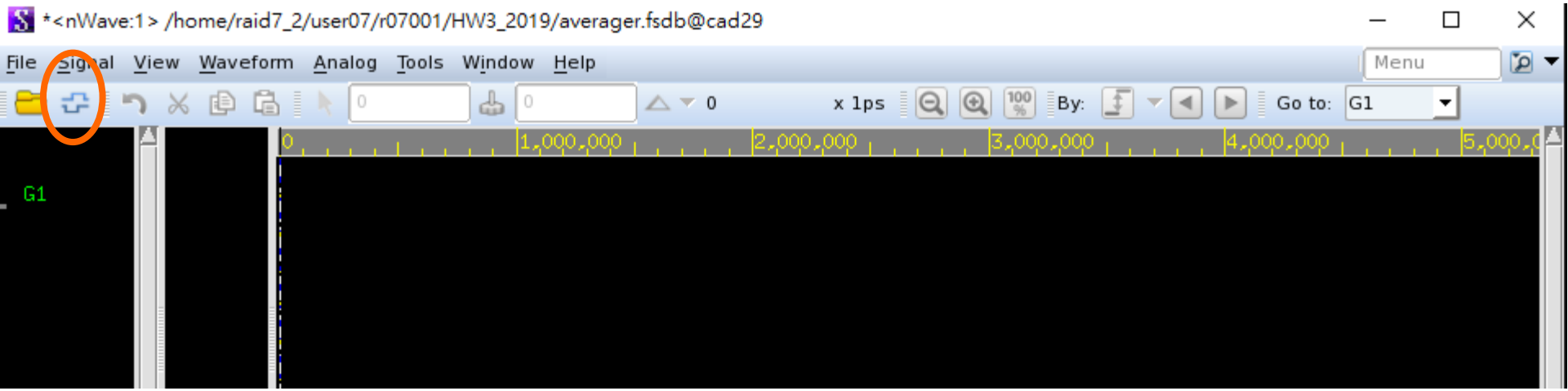
```
[r08006@cad29 ~/HW3_2020]$ ncverilog tb_sorting_pattern.v sorting.v lib.v +access+r
[CORRECT] Your output in          0 ~         999  is correct.
[CORRECT] Your output in       1000 ~        1999  is correct.
[CORRECT] Your output in       2000 ~        2999  is correct.
[ERROR] The golden in       3483 is 0x2e, 0x2d, 0x2c, 0x1b, 0x1a, but your out
put is 0xXx, 0xXx, 0xXX, 0xXX, 0xXX
[ERROR]Your output in       3000 ~        3999  has           1 errors.
[CORRECT] Your output in       4000 ~        4999  is correct.
[CORRECT] Your output in       5000 ~        5999  is correct.
[CORRECT] Your output in       6000 ~        6999  is correct.
[CORRECT] Your output in       7000 ~        7999  is correct.
[CORRECT] Your output in       8000 ~        8999  is correct.
[CORRECT] Your output in       9000 ~        9999  is correct.
--------------------------------------------------
```

Execute nWave & Open *.fsdb

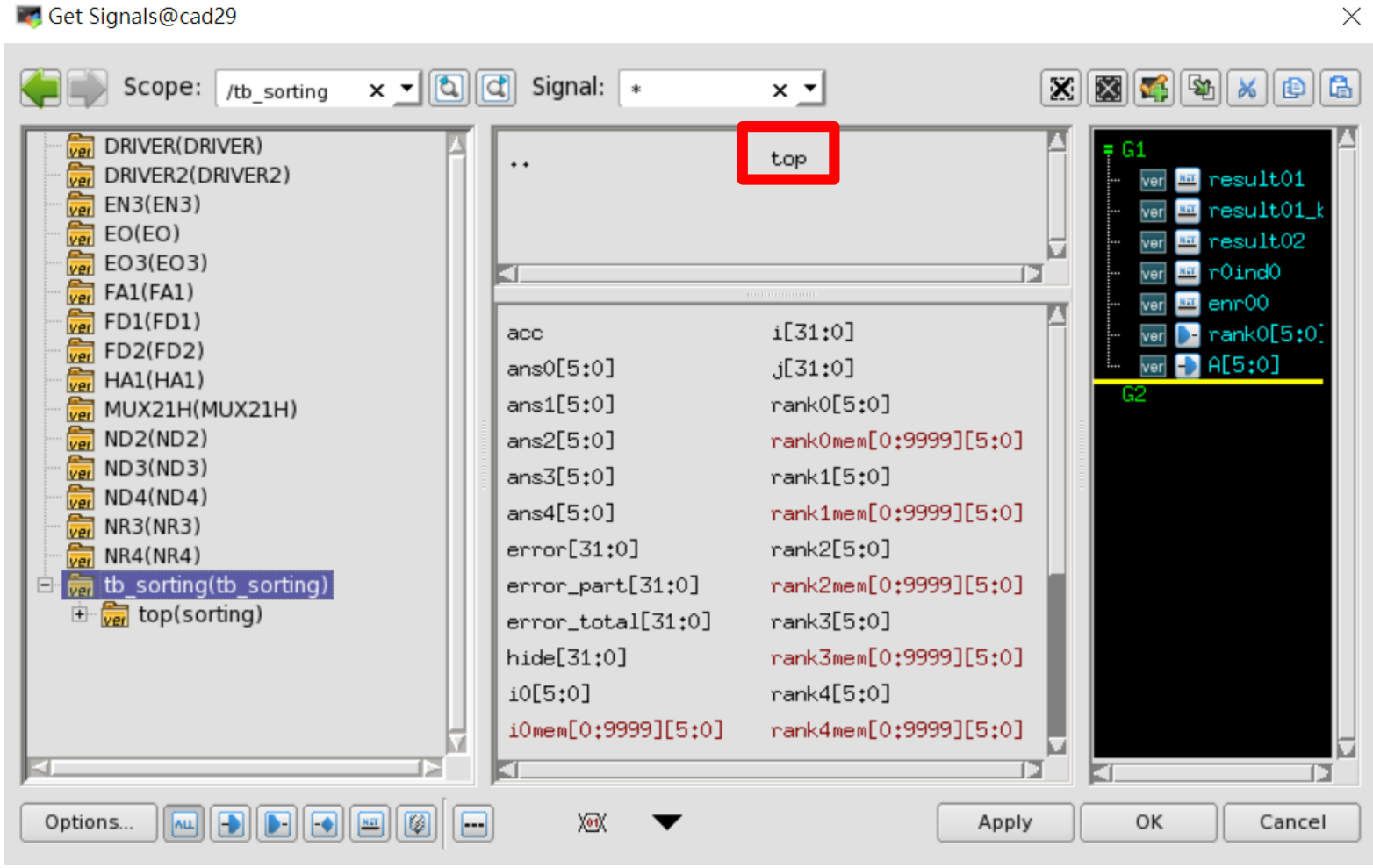
- Execute : *nWave &*
- Open waveform file:
  - File -> Open -> sorting.fsdb



Get signals

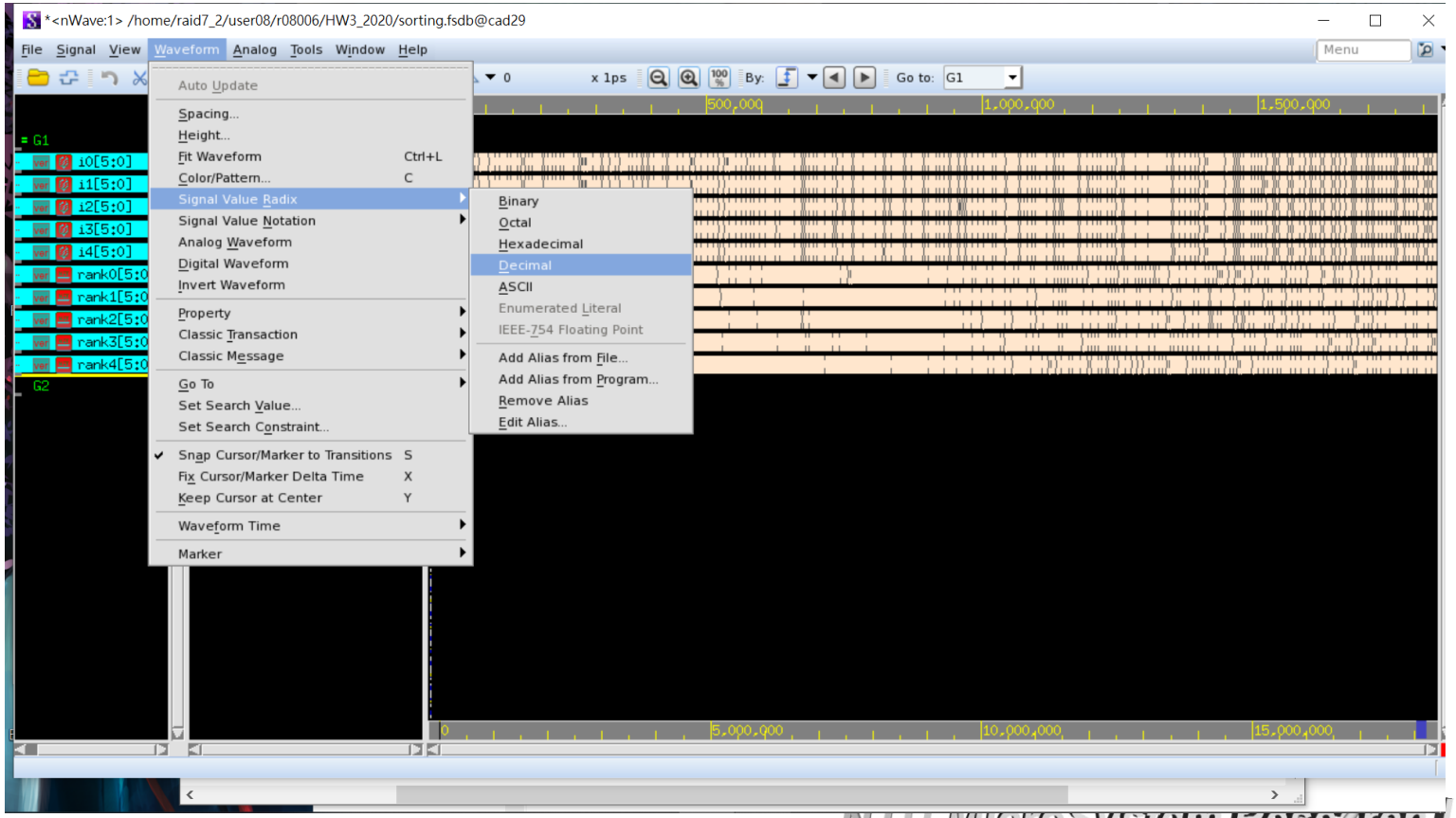

*<nWave:1> /home/raid7_2/user07/r07001/HW3_2019/averager.fsdb@cad29
File
Signal
View
Waveform
Analog
Tools
Window
Help
Menu
0
x 1ps
By:
Go to:
G1
G1
0
1,000,000
2,000,000
3,000,000
4,000,000

Find Top Module & Choose signals



Change the radix to be decimal

- Select the signal you want to change first.



Reminder

- Due to 2020/12/18 13:20
- Any further questions, please contact…
  - r08943006@ntu.edu.tw