Basic Logic Design via Verilog HDL

(Combinational Circuits)

Lecturer: 蔡承融 (EEII-232)
Professor: 李建模
Professor: 江介宏

Date: 2013/11/01

Outline

- Introduction to Verilog HDL
- Verilog Coding Architecture
- Syntax in Verilog HDL
  - Basic usage
  - Advanced usage
  - Module and Instance
- Gate-Level Modeling
- Testbench
- Compilation and Simulation Tools
- Preview of Lab Questions

Introduction to Verilog HDL

What is Verilog HDL?

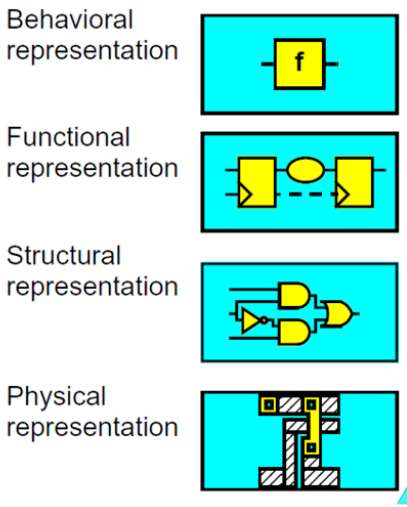
- Why using Hardware Description Language?
  - High-level programming language
    - Hardware design and modeling
    - Reduce cost and time to design hardware
    - Flexibility and convenience
- Two Popular HDLs
  - VHDL
  - Verilog

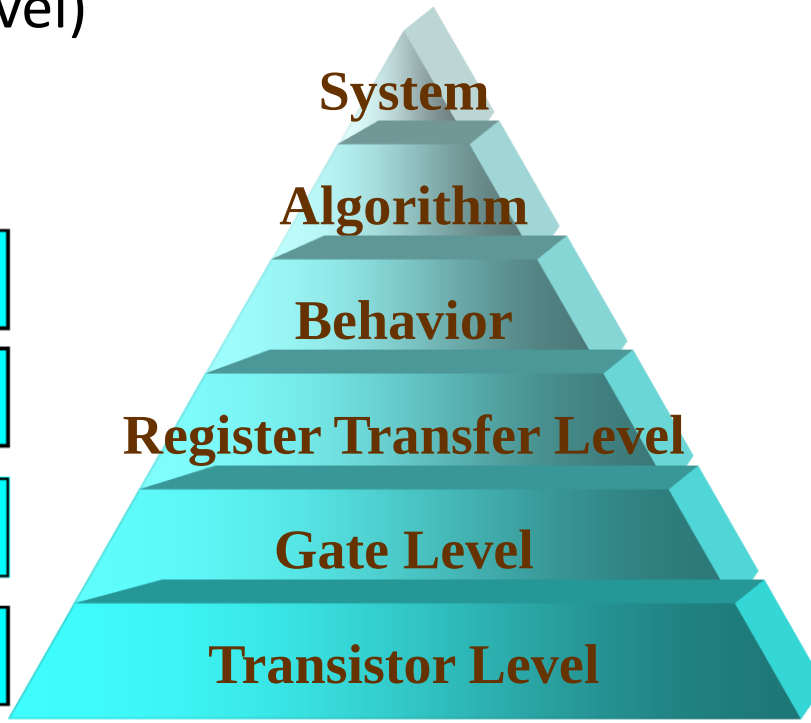
What is Verilog HDL?

- Key features of Verilog
  - Multiple levels of abstraction
    - Behavioral
    - Functional (RTL:Register Transfer Level)
    - Structural (Gate-Level)
  - Model the timing of the system
  - Express the concurrency
  - Verify the design

Levels of Abstraction

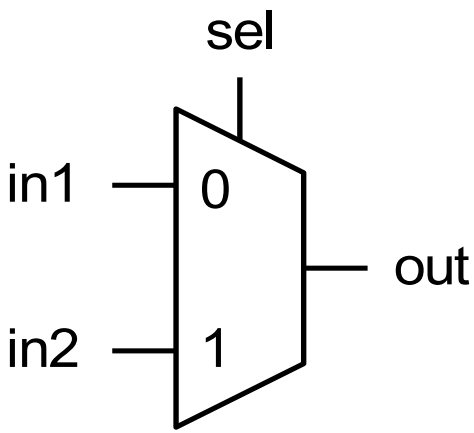
- Behavioral level
  - State the behavior or function of the design
  - Without the information of the architecture
- Functional level(Register transfer level)
  - Data flow between registers
  - Data processing
- Structural level
  - Logic gates
  - Interconnections





High-Level Description

Example: 1-bit Multiplexer



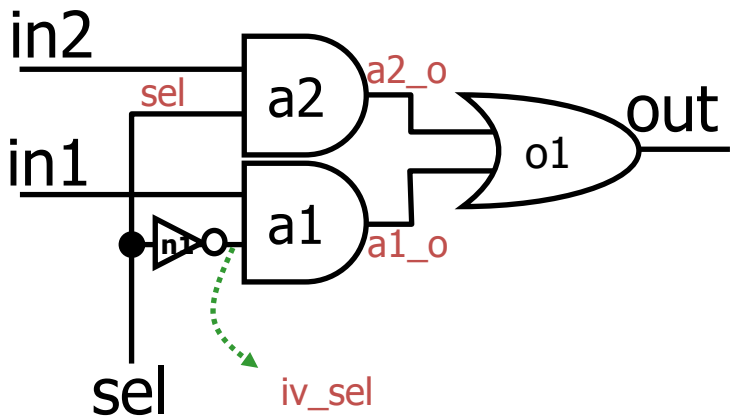
Behavior or **RTL**

```
if ( sel==0 )
  out = in1;
else
  out = in2;
```

```
out = sel'&in1 + sel&in2;
```

- High-level description
  - Describe the behavior or function of the circuit
  - Without the detail of the circuit construction

Gate-Level Description



$$out = (sel' \cdot in1) + (sel \cdot in2)$$

Gate level

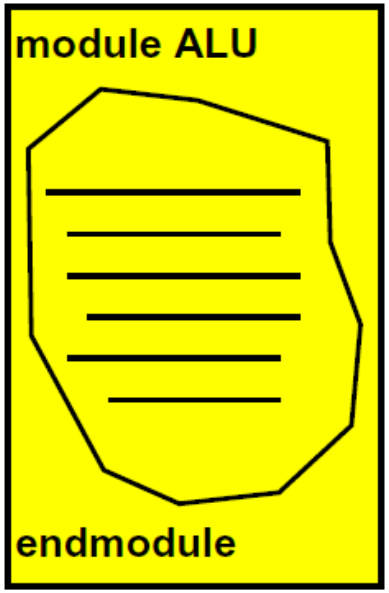
```
module mux2(out , in1 , in2 , sel);
  output out;
  input in1 , in2 , sel;
  wire iv_sel , a1_o ,s2_o;

  and a1(a1_o,in1,in_sel);
  not n1(iv_sel,sel);
  and a2(a2_o,in2,sel);
  or  o1(out,a1_o,a2_o);
endmodule
```

- Gate-level description
  - May not realize the circuit function easily
  - Contain the detail construction of the circuit

Verilog Coding Architecture

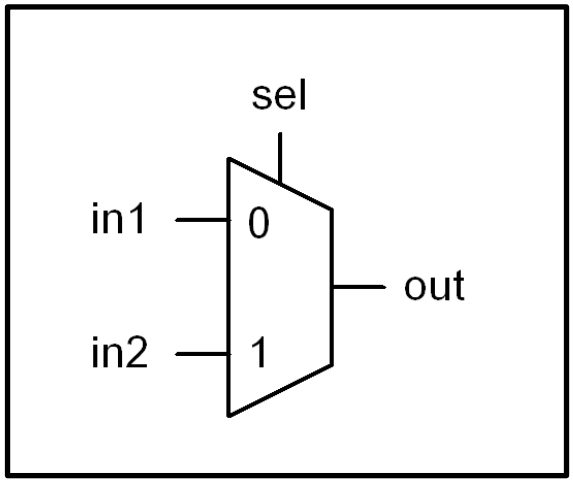
Verilog Architecture

- module / endmodule
  - Basic building block
  - Can contain instances of other modules
  - All modules run concurrently
- Module ports
  - Input/output declaration
- Wire declaration
- Kernel hardware connection



A Simple Example --- MUX (1/3)

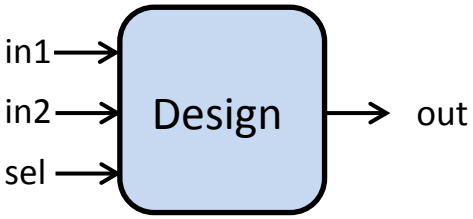
Example. 1-bit Multiplexer	Step 1. Derive truth table	Step 2. Derive the output Boolean expression



sel	in1	in2	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

out = sel'*in1 + sel*in2

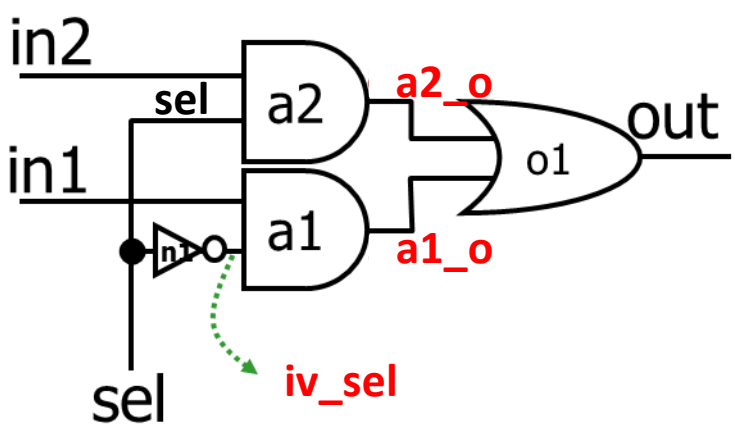
in1in2 \ sel	0	1
00	0	0
01	0	1
11	1	1
10	1	0



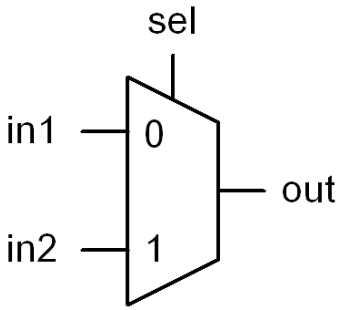
A Simple Example --- MUX (2/3)

Step3. Draw the circuit	Step 4. Specify the wire connection

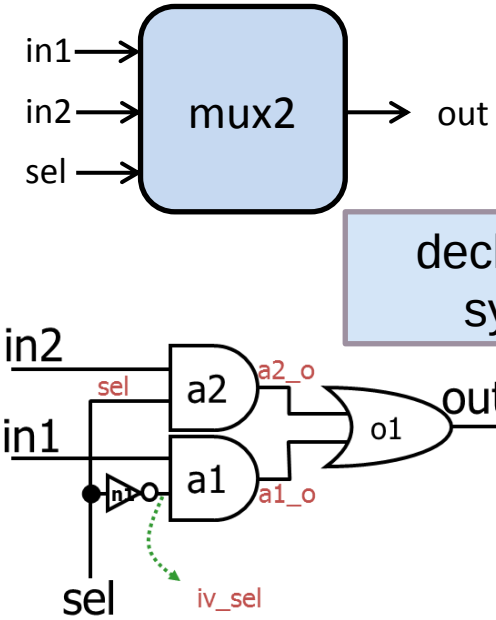
out = sel'*in1 + sel*in2



A Simple Example --- MUX (3/3)



Step 5. Verilog Coding



module name

in/out port

```
module mux2(out,in1,in2,sel);
    output out;
    input  in1,in2,sel;
    wire   iv_sel,a1_o,a2_o;

    not n1 (iv_sel,sel);
    and a1 (a1_o,in1,iv_sel);
    and a2 (a2_o,in2,sel);
    or  o1 (out,a1_o,a2_o);
endmodule
```

port/wire declaration

declaration syntax

Gate level circuit

Syntax in Verilog HDL Coding

Basic Usage

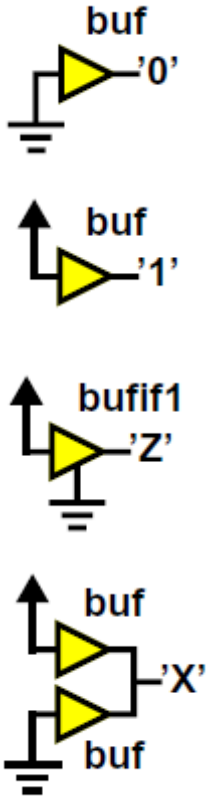
Verilog Language Rule

- Identifiers
  - Decimal digits **0~9**
  - Underscore **_**
  - Upper and lower case alphabeticals: a~z A~Z
  - **Cannot start with decimal digits**
    - **"2wire" is illegal**
- Case sensitive
  - **Avoid using this part**
- Terminate statement/declaration with semicolon "**;**"
- Comments
  - Single line: **// it's a single line comment example**
  - Multi-line: **/* When the comment exceeds single line, multi-line comment is necessary */**

```
/*  Verilog HDL module
    Half adder
*/
module adder (out0,in1,in2);
    output  [1:0]   out0;
    input           in1,in2;

    assign  out0 = in1 + in2
endmodule   //end of module
```

4-valued Logic System

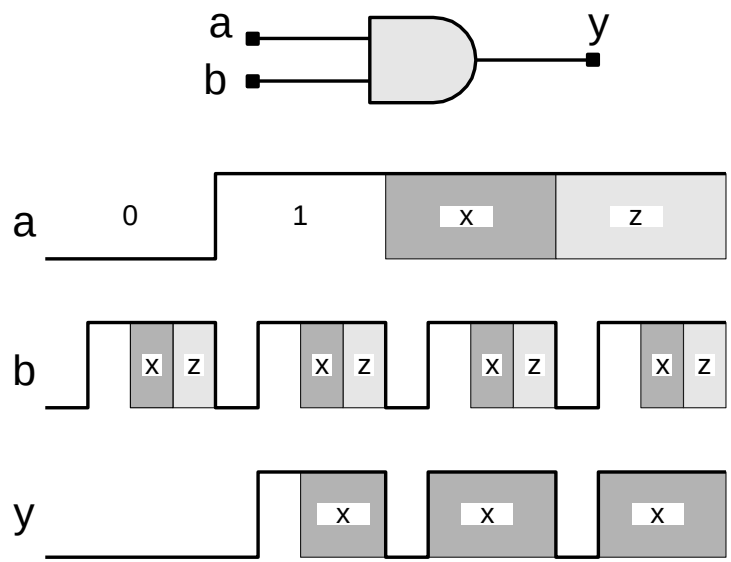
- 4-value logic system in Verilog
  - **0** represents a logic '0' or a false condition
  - **1** represents a logic '1' or a true condition
  - **z**
    - High-Z value
    - Model the wire not be driven
  - **x**
    - Models of confliction –
      un-initialized or unknown logic value
      - Initial state of registers
      - A wire is being driven to 0 and 1 simultaneously



Logic System: Example

Table 8-1 in Page 231

	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X



Data Type

- Register
  - Keyword: **reg**, **integer, time, real**
  - Storage element (modeling sequential circuit)
  - Used in behavior or functional level code
  - See the provided testbench

- Net
  - Keyword: **wire**, **wand, wor, tri, triand, trior, supply0, supply1**
  - Does not store values, just represent a connection
  - Input, output **and inout** ports are default "**wire**"

Net Declaration

- Examples

Representation	Bit-length	Note
wire a1 , a2;	1	Multiple wire declarations
wire **[2:0]** b;	3	b is 3-bit wire
input in1;	1	in1 is input which is also a 1-bit wire
output **[1:0]** out;	2	out is output which is also a 2-bit wire

- Bad usage

Representation	Bit-length	Note
wire a , A;	1	**Avoid using case-sensitive declarations**
wire **[3:1]** b;	3	**Avoid using different bit representations**

Number Representation

- Format: **<size>'<base_format><number>**
- **<size>** - decimal specification of bits count
  - Default: unsized and machine-dependent but at least 32 bits
- **<base_format>** - **'** followed by arithmetic base of number
  - **d** or **D** – decimal (default if no base format given)
  - **h** or **H** – hexadecimal
  - **o** or **O** – octal
  - **b** or **B** – binary
- **<number>** - value given in base of base format
  - **_** can be used for reading clarity
  - **0** extended
  - **x** and **z** are automatically extended

A Simple Example

Example :

Represent a 8-bit decimal number(110) in 4 formats

Format	Value	Representation
Decimal	110	**8'd110**
Binary	**01101110**	**8'b01101110**
Octal	**156**	**8'o156**
Hexadecimal	**6E**	**8'h6E**

More Examples

Representation	Bit length	Value(binary)	Value(decimal)
6’b01_0111	6	010111	23
8’b0110	8	00000110	6
12’hAB	12	000010101011	171
5’o36	5	11110	30

Net Concatenation

- An easy way to group nets

Representation	Meaning	Usage
{cout, sum}	{cout, sum}	{cout, sum} = 2'b11;
{b[7:4],c[3:0]}	{b[7], b[6], b[5], b[4], c[3], c[2], c[1], c[0]}	{b[7:4],c[3:0]} = 8'd110;
{4{2'b01}}	8'b01010101	{b[7:4],c[3:0]} = {4{2'b01}};
{{8{byte[7]}},byte}	Sign extension	wire [7:0] byte; wire [15:0] byte_2; byte_2 = {{8{byte[7]}},byte};

Syntax in Verilog HDL Coding

Advanced Usages

Note :

You are forbidden to use this part in your homework!!!

Learn more from the testbench.

Operators

Arithmetic Operators	+, -, *, /, %
Relational Operators	<, <=, >, >=
Equality Operators	==, !=, ===, !==
Logical Operators	!, &&, ||
Bit-wise Operators	~, &, |, ^, ~^
Unary Reduction	&, ~&, |, ~|, ^, ~^
Shift Operators	>>, <<
Conditional Operators	?:
Concatenations	{}

Excerpts from CIC training course: Verilog_9807.pdf

Compiler Directives

- **`define**
  - `define RAM_SIZE 16
  - Define a name and give a constant value to it.
- **`include**
  - `include adder.v
  - Include the entire contents of another verilog source file.
- **`timescale**
  - `timescale 100ns/1ns
  - Sett the reference time unit and time precision of your simulation.

System Tasks

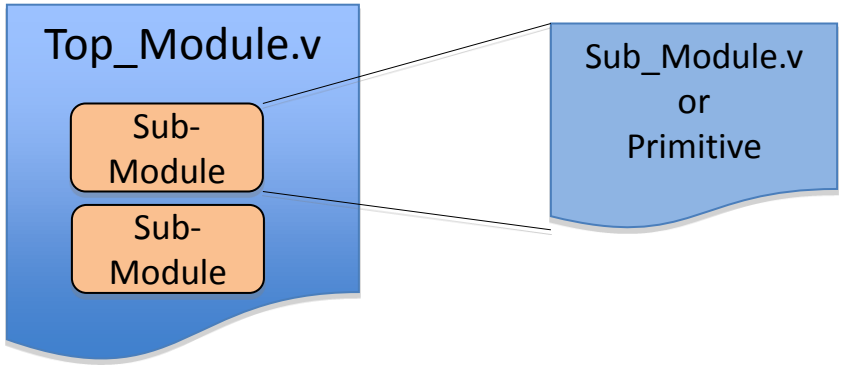
- *$monitor*
  - $monitor ($time,"%d %d %d",address,sinout,cosout);
  - Display the values of the argument list whenever any of the arguments alters except $time.
- *$display*
  - $display ("%d %d %d",address,sinout,cosout);
  - Print out the current values of the signals in the argument list
- *$finish*
  - $finish
  - Terminate the simulation

Syntax in Verilog HDL Coding

Instance: Module & Primitive

Instances

- A instance provides a template from which you can create actual objects.
- When a instance is invoked, Verilog creates a unique object from the template.
- Each object has its own name, variables, parameters and I/O interface.
- Instance can be a module or a primitive
- Primitives
  - Logic gates provided by cell library
  - and / or / nor / xor



Module and Instantiation

Module declaration

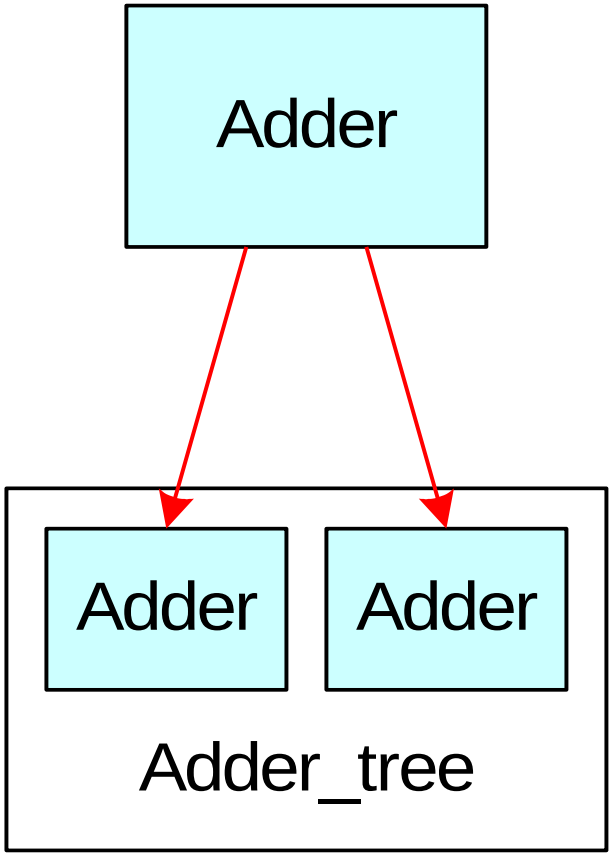
```
module adder (out,in1,in2);
  output out;
  input  in1 , in2;

  assign out = in1 + in2;
endmodule
```

Module Instantiation

```
module adder_tree (out0,out1,in1,in2,in3,in4);
  output out0 , out1;
  input  in1 , in2 , in3 , in4;

  adder adder_0 (.out(out0),.in1(in1),.in2(in2));
  adder adder_1 (.out(out1),.in1(in3),.in2(in4));
endmodule
```

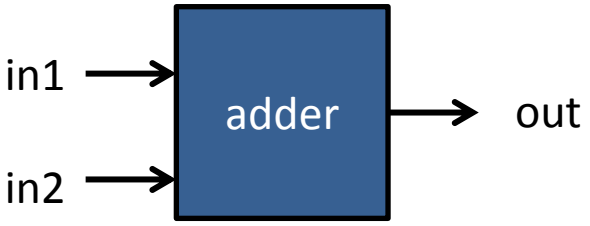
Instance name

Port connection



Port Connection

```
module adder (out,in1,in2);
  output out;
  input  in1 , in2;

  assign out = in1 + in2;
endmodule
```



- Connect module ports by *order list*
  - `adder adder_0 ( C , A , B );  // C = A + B`
- Connect module ports by *name* **(Recommended)**
  - Usage: .PortName (NetName)
  - `adder adder_1 ( .out(C) , .in1(A) , .in2(B) );`
- Not fully connected**(Avoid)**
  - `adder adder_2 ( .out(C) , .in1(A) , .in2() );`

Primitives

- Primitives are modules provided by the cell library
- The smallest modeling block for simulator
- Verilog build-in primitive gates
  - *and, or, xor, nand, nor, xnor (multiple inputs, 1 output)*
    - prim_name [inst_name](out0, in0, in1,....);
    - and an1 (abc , a , b , c)
      - abc = a & b & c
  - *not, buf (1intput, multiple outputs)*
    - prim_name [inst_name](out0, out1, ..., in0);
    - not n1 (na1 , na2 , a)
      - na1 = ~a; na2 = ~a;

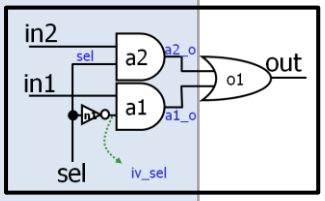
Gate-Level Modeling

Gate-Level Modeling (Review)

Gate-Level Circuit Design

Step 1. Truth table	Draw the true table.
Step 2. Boolean expression	Derive the output Boolean expression.
Step 3. Circuit	Draw the circuit based on the output Boolean expression.
Step 4. Wire specification	Specify the wire connection on the circuit drawn from step3.
Step 5. Coding	Write Verilog code according to your circuit.

sel	in1	in2	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

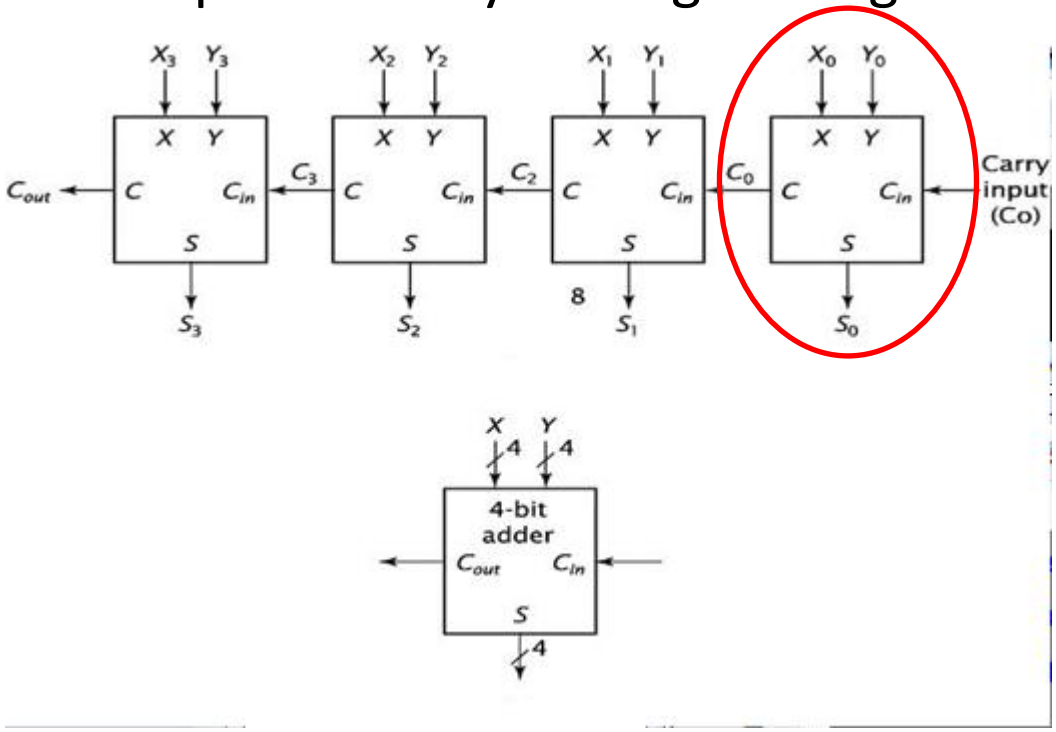



```
module mux2(out,in1,in2,sel);
    output out;
    input  in1,in2,sel;
    wire   iv_sel,a1_o,a2_o;

    not n1 (iv_sel,sel);
    and a1 (a1_o,in1,iv_sel);
    and a2 (a2_o,in2,sel);
    or  o1 (out,a1_o,a2_o);
endmodule
```

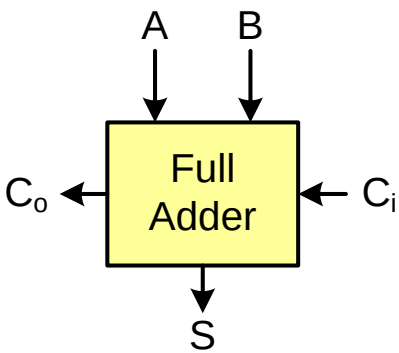
Figure out architecture before coding!!!

Case Study: Full Adder

- What is Full Adder?
  - Basic unit of ripple-carry adder, carry-lookahead adder…
  - Refer to chapter 4.7 of your Logic Design textbook



Case Study: Full Adder



Step 1. Truth table

C_i	A	B	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Case Study: Full Adder (C_o)

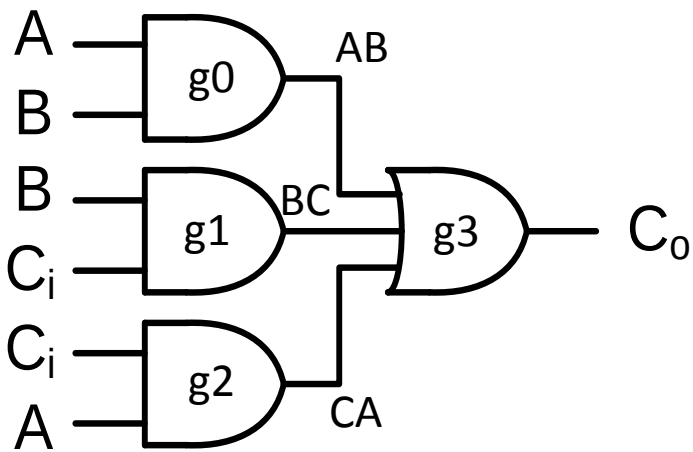
Step 2. Output expression

Step 3. Circuit
Step 4. Wire specification

C_i	A	B	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

AB \ C_i	0	1
00	0	0
01	0	1
11	1	1
10	0	1

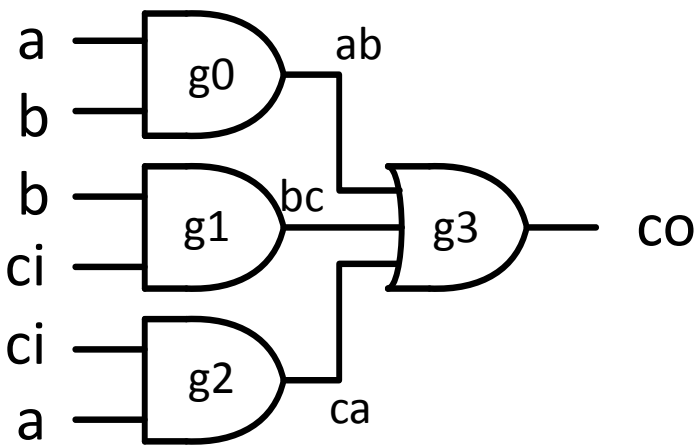
$$C_o = AB + BC_i + C_iA$$



Case Study: Full Adder (C_o)

Step 5. Coding

co = a&b + b&ci + ci&a




```
module FA_co (co, a, b, ci);
    output co;
    input  a, b, ci;
    wire   ab, bc, ca;

    and g0(ab, a, b);
    and g1(bc, b, ci);
    and g2(ca, ci, a);
    or  g3(co, ab, bc, ca);
endmodule
```

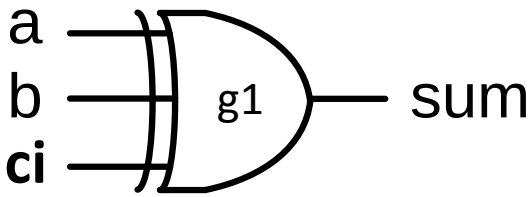
Case Study: Full Adder (Sum)

Step 2. Output expression

Step 3. Circuit
Step 4. Wire specification

C_i	A	B	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

AB \ C_i	0	1
00	0	1
01	1	0
11	0	1
10	1	0

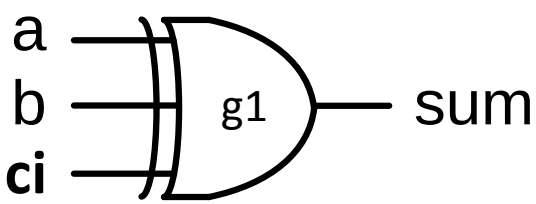


sum = a ⊕ b ⊕ c_i

Case Study: Full Adder (Sum)

Step 5. Coding

$$\mathbf{sum = a \oplus b \oplus c_i}$$



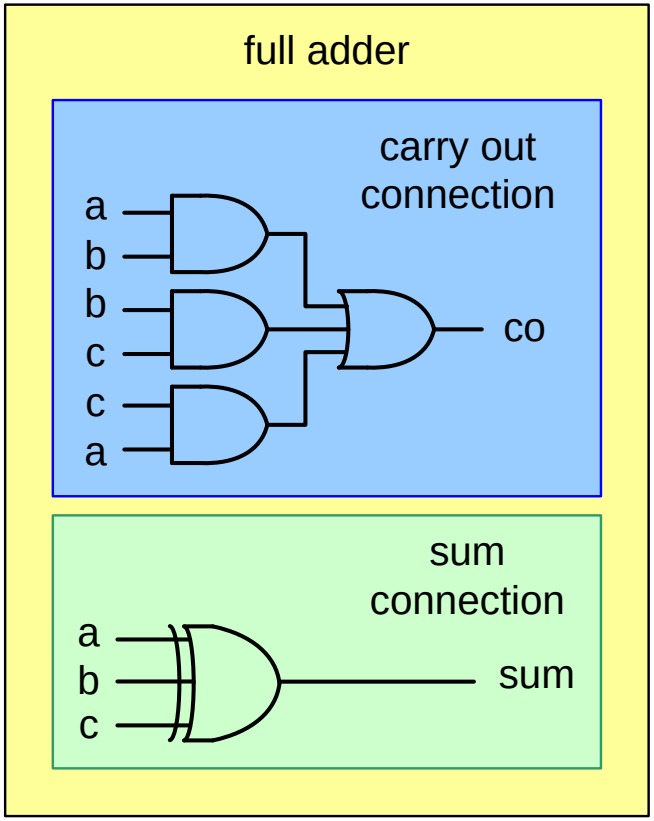

```
module FA_sum (sum,a,b,ci);

  output sum;
  input  a,b,ci;

  xor g1 (sum,a,b,ci);

endmodule
```

Case Study: Full Adder

- Full Adder Connection
  - Instance *ins_C* from FA_co
  - Instance *ins_S* from FA_sum

Connection

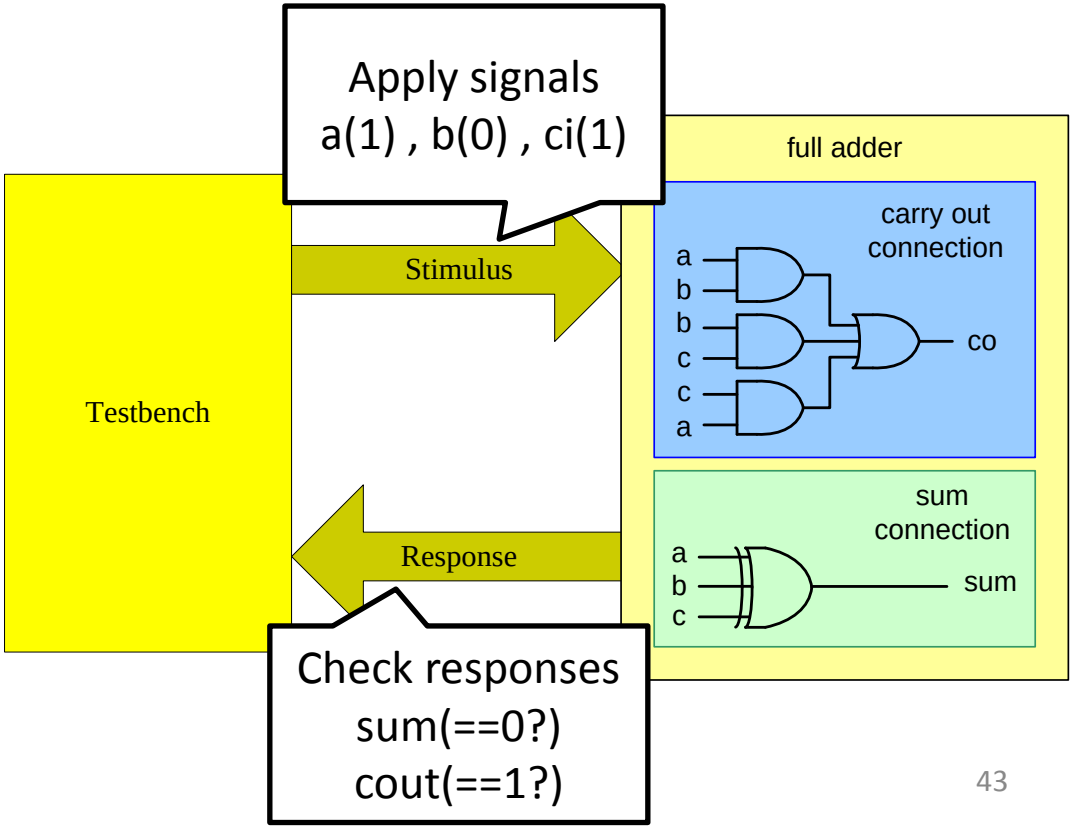
```
module FA (sum,co,a,b,ci);

  output sum,co;
  input  a,b,ci;

  FA_co  ins_C (.co(co),.a(a),.b(b),.ci(ci));
  FA_sum ins_S (.sum(sum),.a(a),.b(b),.ci(ci));

endmodule
```



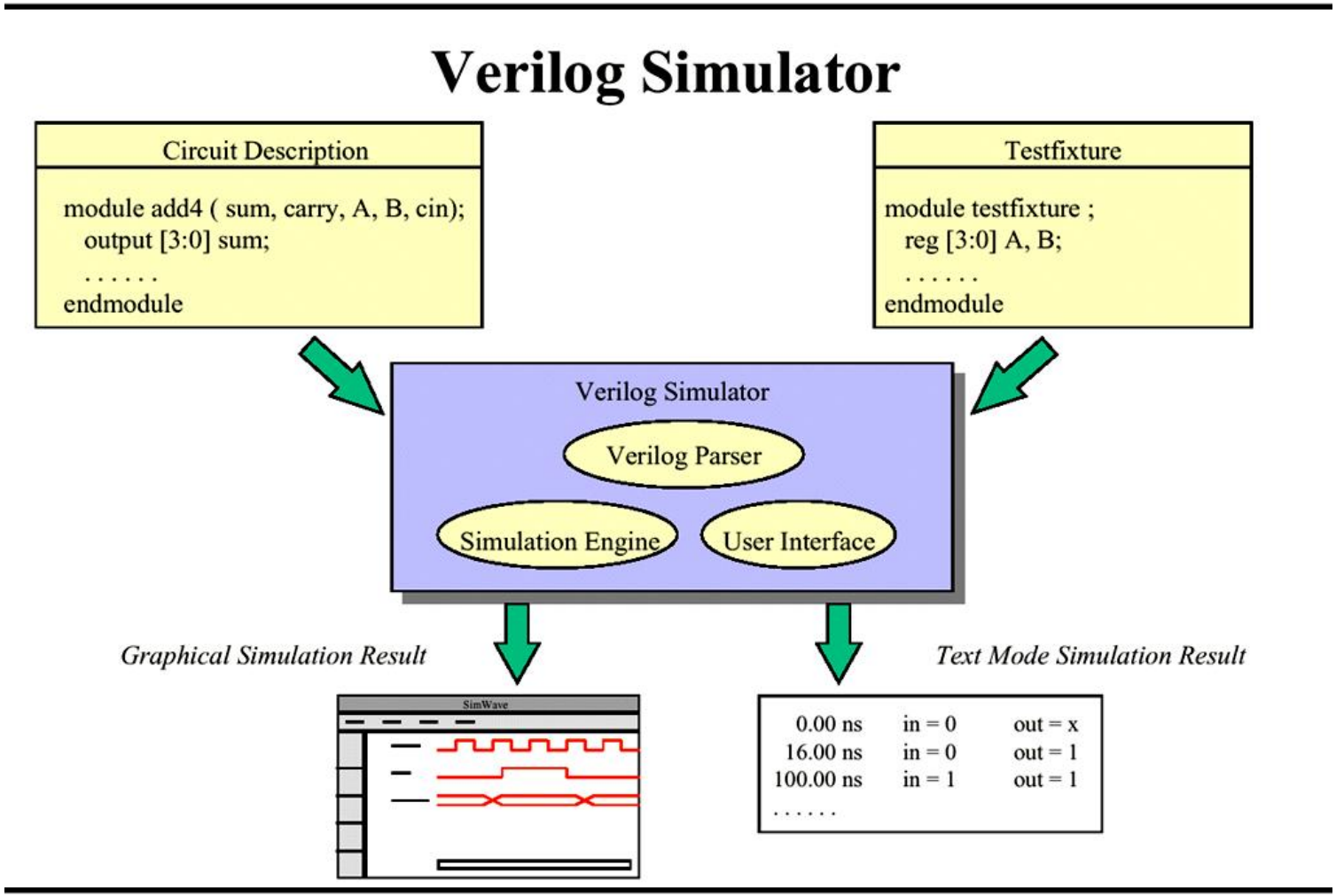
Testbench

Test Methodology

- Systematically verify the **functionality** of a model
- Procedures of simulation
  - Detect syntax errors in source codes
  - Simulate the behavior
  - Monitor results

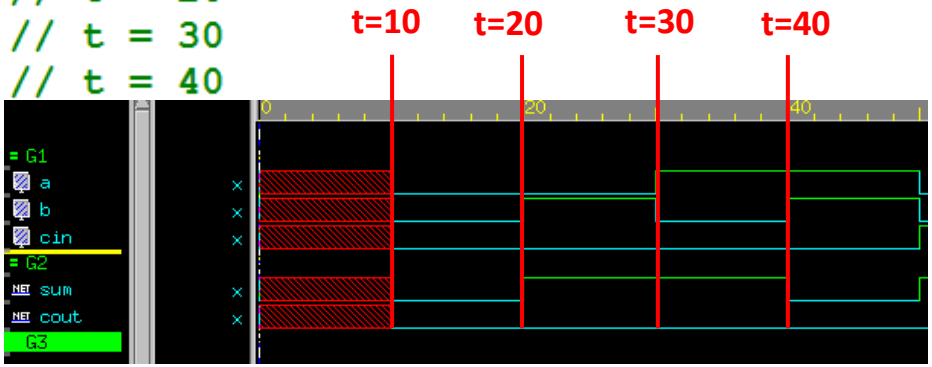


Verilog Simulator



Testbench for Full Adder

```
module test();
  reg  a , b , cin;              //signal declaration
  wire sum , cout;

  //Instance
  FA my_fa1 (.sum(sum),.co(cout),.a(a),.b(b),.ci(cin));

  initial #100 $finish;          //stop simulation

  initial begin
    $dumpvars();
    $dumpfile("FA.vcd");
  end

  initial begin                   // stimulus patterns
    #10 a = 0; b = 0; cin = 0;    // t = 10
    #10 a = 0; b = 1; cin = 0;    // t = 20
    #10 a = 1; b = 0; cin = 0;    // t = 30
    #10 a = 1; b = 1; cin = 0;    // t = 40
    #10 a = 0; b = 0; cin = 1;
    #10 a = 0; b = 1; cin = 1;
    #10 a = 1; b = 0; cin = 1;
    #10 a = 1; b = 1; cin = 1;
  end
endmodule
```



Summary

- Design module
  - Divide-and-Conquer
    - Partition the whole design into several parts
  - Derive the architecture of each sub-module
    - Make architecture figures before you write Verilog codes
  - Create hardware design in gate level
  - Connection of sub-modules

- Test-bench
  - Feed input data and compare output values at right timing slots
  - Usually describe in behavioral level
  - Not real hardware, just like software programming (e.g. C/C++)

Note

- Verilog is a platform
  - Support hardware design (design module)
  - Also support C/C++ like coding (testbench)
- How to write Verilog well?
  - Know basic concepts and syntax
  - Get good reference codes
    (a person or some code files)
  - Form a good coding style
- Hardware
  - Combinational circuits (today's topic)
  - Sequential circuits (we won't model them in this course)

Compilation and Simulation Tools

Workstations
MobaXterm
NC-Verilog
nWave

Workstations

- Why workstations?
  - Multiple Users, multiple tasking
  - Stable Operations
- How to run tasks on the workstations?
  - Operating System: Unix-like
    - Example: Linux
    - Example: Solaris
  - User Interface
    - Text Mode
    - X-Window

Workstations

- Where are the workstations?
  - http://cad.ee.ntu.edu.tw/
  - You are receiving the account and the password to the workstations.
    - **NOTE:** This account expires at the end of the course.
    - If you want to continue enjoying the resources on the workstations, contact the TA managing the IC design Lab to get more information.
      - Usually you need to attend the Special Projects held by the professors in ICS or EDA group

Welcome to IC Design Lab

cad.ee.ntu.edu.tw

Welcome to IC Design Lab.

Last Updated:08/29/2011

最新公告：

- All workstations will be off at 18:00 on 09/24/2011 for maintenance. (08/29/2011)
- The new workstations are installed, please refer to workstation list. (08/01/2011)
- New quota policy try: PhD student: 10 GB, Master student: 5 GB, Others: 1 GB. (07/26/2011)
- The account of userd92, user97, and userb95 will be remove on 09/19/2011.
  Please backup personal data by yourself. Contact TA if you have any problems. (07/26/2011)
- TA Office Hours: Tue. 10:30 am ~ 12:00 pm and Fri. 10:30 am ~ 12:00 pm. (07/18/2011)
- New version Cadence tools: IUS, MMSIM, and Conformal.
  New version Synopsys tools: Synthesis, CosmosScope, PrimePower, PrimeTime, VCS, and Vera.
  New version Mentor tools: Calibre and Eldo.
  Please refer to tool_list. (02/16/2011)
- The new disk array is installed, please contact TA if your folder have any problem. (09/28/2009)
- The summer course information is announced. Please refer to summer course (08/03/2008)
- The raid1 data is recovered, please contact TA if you have any problem. (05/21/2009)
- New PCB fabrication machine is arrival. Please refer to PCB rules (10/31/2008)
- LAB 231 will no longer support ftp after July 31, 2006. Please use secure ftp afterward.
  Please refer to SFTP.
- New version of Debussy and Verdi are available. Old version will be out of work by the end of July 2006.
  Debussy: /usr/spring_soft/CIC/debussy.cshrc
  Verdi: /usr/spring_soft/CIC/verdi.cshrc

一般公告：

1. 可連線主機一覽表
2. 帳號命名規則
3. 上線軟體資源一覽表
4. Q & A
5. Sample .cshrc files: /home/raid1_1/.cshrc
6. Cell library: /home/raid1_1/cic/

IC設計實驗室可供連線工作 ×

cad.ee.ntu.edu.tw/ws_list.htm

IC設計實驗室可供連線工作站一覽表

Oct. 02, 2009

IP	NAME	TYPE	CPU	CPU CLOCK	MEMORY	OS
140.112.20.60	cad17	**IBM X3550**	**Intel Xeon 64**	2.4 GHz * 16	20 G	**RHEL 5**
140.112.20.61	cad18	SUN Blade 2500	UltraSPARC	1.28 GHz*2	8 G	Solaris 10
140.112.20.62	cad19	SUN Blade 2000	UltraSPARC	1.2 GHz * 2	8 G	Solaris 10
140.112.20.63	cad20	SUN Blade 2000	UltraSPARC	1.2 GHz * 2	8 G	Solaris 10
140.112.20.64	cad21	SUN Blade 2500	UltraSPARC	1.28 GHz * 2	8 G	Solaris 10
140.112.20.65	cad22	SUN Blade 2500	UltraSPARC	1.28 GHz * 2	8 G	Solaris 10
140.112.20.66	cad23	SUN Blade 2500	UltraSPARC	1.28 GHz * 2	8 G	Solaris 9
140.112.20.67	cad24	SUN Fire 280R	UltraSPARC	1.2 GHz * 2	4 G	Solaris 9
140.112.20.68	cad25	SUN Fire 280R	UltraSPARC	1.2 GHz * 2	4 G	Solaris 9
140.112.20.69	cad26	SUN Fire 280R	UltraSPARC	1.2 GHz * 2	4 G	Solaris 9
140.112.20.70	**cad27**	**IBM x260**	**Intel Xeon 64**	**3.2 GHz * 4**	**8 G**	**RHEL 4**
140.112.20.71	**cad28**	**IBM x260**	**Intel Xeon 64**	**3.2 GHz * 4**	**8 G**	**RHEL 4**
140.112.20.72	**cad29**	**IBM e336**	**Intel Xeon 64**	**3.2 GHz**	**5 G**	**RHEL 4**
140.112.20.73	**Cad30**	**IBM X3650**	**Intel Xeon 64**	**2 GHz * 16**	**12 G**	**SUSE 11**
140.112.20.74	cad31	SUN Blade 2000	UltraSPARC	1.015 GHz * 2	8 G	Solaris 10
140.112.20.75	cad32	SUN Blade 2000	UltraSPARC	1.015 GHz * 2	8 G	Solaris 8
140.112.20.76	cad33	SUN Blade 2000	UltraSPARC	1.2 GHz * 2	8 G	Solaris 9
140.112.20.77	**cad34**	**SUN V20z**	**AMD Opteron**	**2.2 GHz * 2**	**4 G**	**RHEL 4**
140.112.20.78	**cad35**	**SUN V20z**	**AMD Opteron**	**2.4 GHz * 2**	**4 G**	**RHEL 4**
140.112.20.79	**cad36**	**SUN V20z**	**AMD Opteron**	**2.4 GHz * 2**	**4 G**	**RHEL 4**
140.112.20.80	**cad37**	**SUN V20z**	**AMD Opteron**	**2.4 GHz * 2**	**4 G**	**RHEL 4**
140.112.20.81	**cad38**	**ACER Altos R700**	**Intel Xeon 32**	**3.0 GHz * 2**	**6 G**	**RHEL 4**
140.112.20.82	**cad39**	**SUN V20z**	**AMD Opteron**	**2.4 GHz * 2**	**4 G**	**RHEL 4**
140.112.20.83	**cad40**	**IBM e326**	**AMD Opteron**	**2.4 GHz * 2**	**4 G**	**RHEL 4**
140.112.20.84	**cad41**	**Fujitsu RX300 S4**	**Intel Xeon 64**	**2 GHz * 8**	**10 G**	**RHEL 4**
140.112.20.85	**cad42**	**Fujitsu RX300 S4**	**Intel Xeon 64**	**2 GHz * 8**	**10 G**	**RHEL 4**

Rules of Workstations

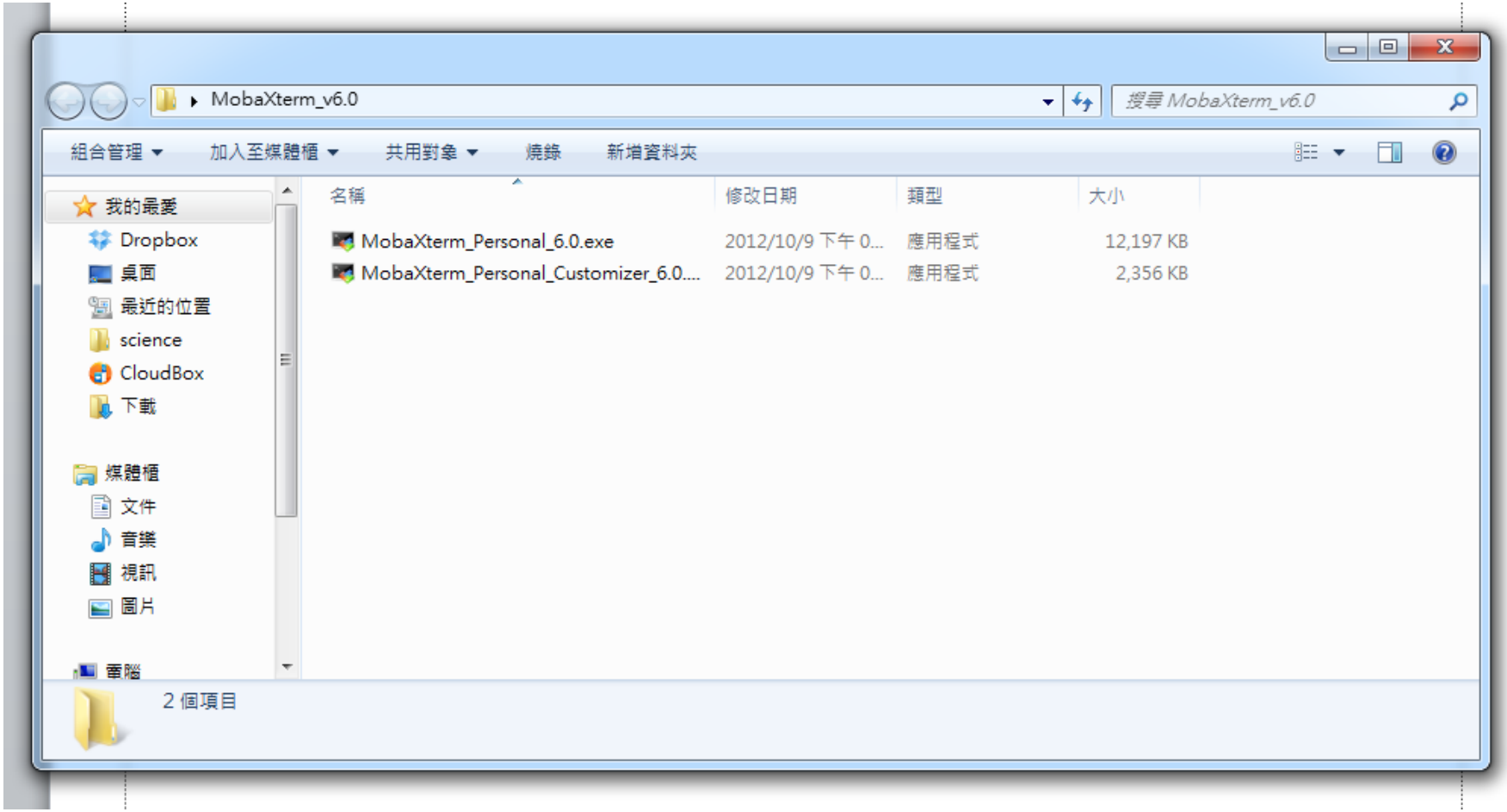
- 實驗室規則：
  - 1. 請勿任意 reboot 主機，破壞系統或做出對系統有害之行為，或將帳號借予他人使用，否則若經查獲，立即刪除帳號，並交與教授處理。
  - 2. 硬碟使用空間以**大學部 1G**、碩士班 5G、博士班 10G 為限，若需超過，請填寫「系統維護申請表」，申請更改 QUOTA。
  - 3. 請愛護使用實驗室儀器設備，保持桌面整潔，座椅用畢歸位，垃圾自行帶走。
  - 4. 借閱本實驗室之軟體、書籍、使用手冊時，請按時歸還。
  - 5. **個人資料請自行備份，並於畢業時將個人目錄下的檔案清理乾淨。**
  - 6. **大學部同學帳號預設期限為一學期**，碩士班同學為兩年，博士班同學為四年，若需要延長，請填寫系統維護申請表。
  - 7. 其它注意事項請參閱本實驗室內的實驗室公布欄與實驗室網頁 http://cad.ee.ntu.edu.tw。

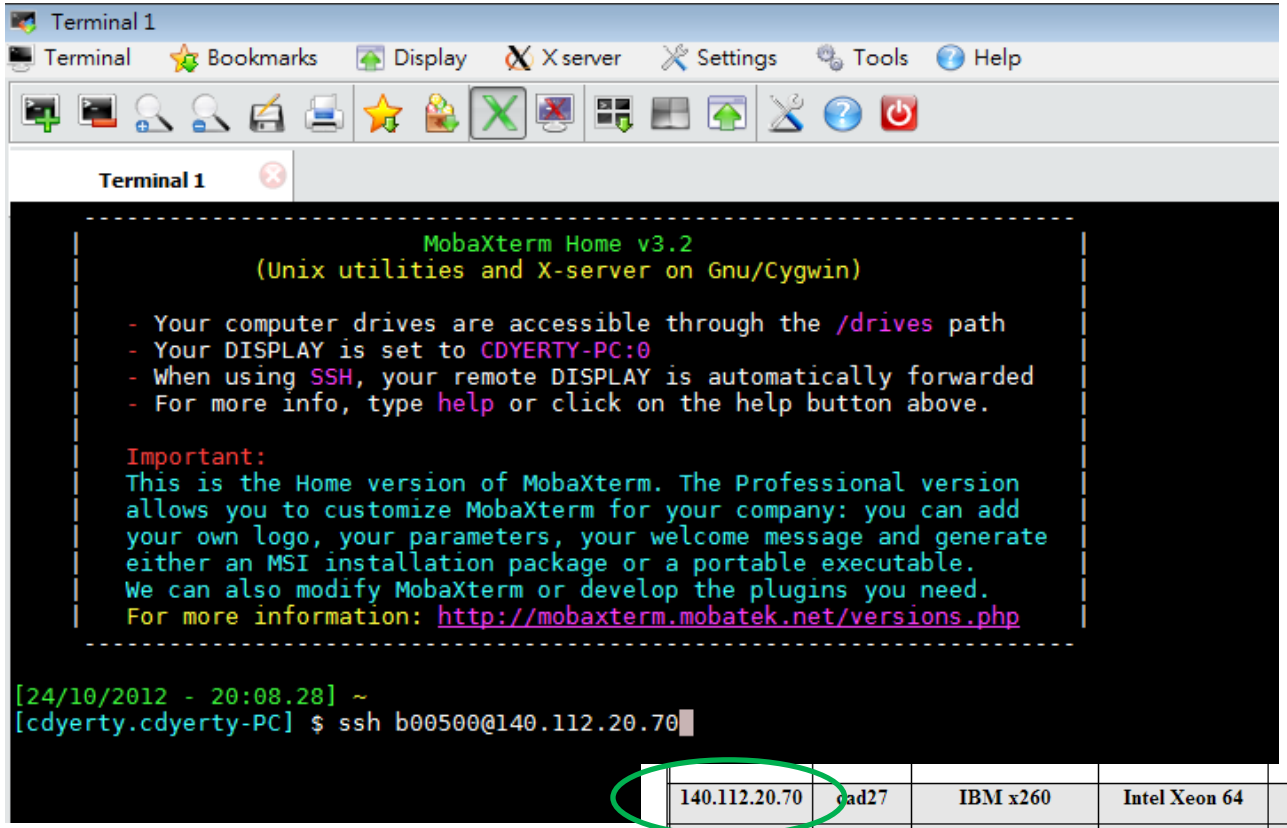
Rules of Workstations

- 帳號命名規則：
  - 學號之「系所代碼」為 **943** 與 **901** 同學：將學號中「系所代碼」去掉，即為帳號。
    - 例：學號為「b99901001」→帳號為「b99001」
  - 其它系所代碼同學，將學號中英文字母後第一個數字去掉即為帳號。
    - 例：學號為「r90942001」→帳號為「r0942001」

Workstations

- How can I connect to the workstations?
  - **putty** download site:
    - http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
  - **pietty** download site:
    - http://ntu.csie.org/~piaip/pietty/
  - **MobaXterm** download site:
    - http://mobaxterm.mobatek.net/download.html
  - In the following examples, we will use **MobaXterm**.
    - MobaXterm_v6.0.zip

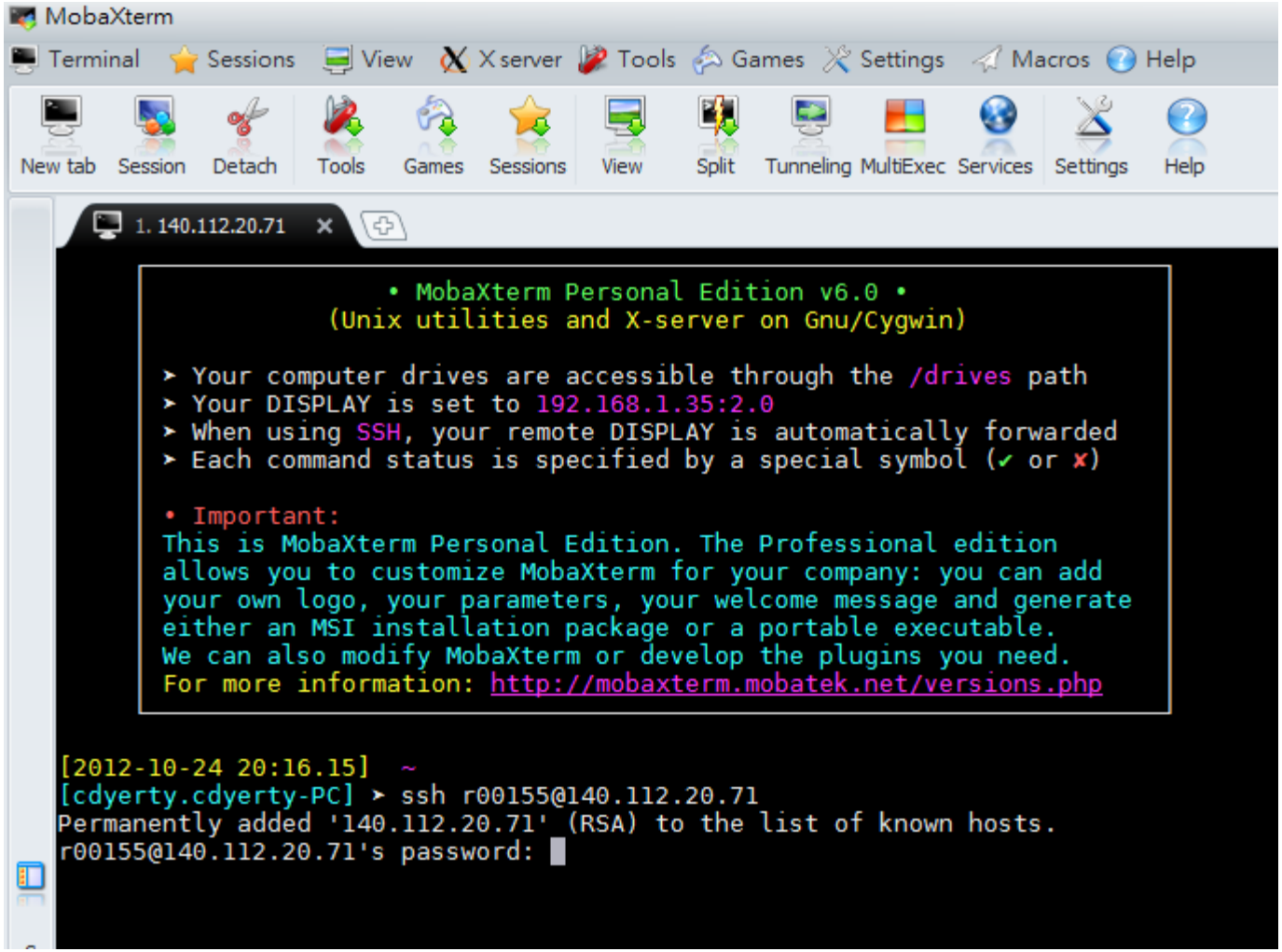

MobaXterm_v6.0
搜尋 MobaXterm_v6.0
組合管理
加入至媒體櫃
共用對象
燒錄
新增資料夾
我的最愛
Dropbox
桌面
最近的位置
science
CloudBox
下載
媒體櫃
文件
音樂
視訊
圖片
電腦
名稱
修改日期
類型
大小
MobaXterm_Personal_6.0.exe
2012/10/9 下午 0...
應用程式
12,197 KB
MobaXterm_Personal_Customizer_6.0....
2012/10/9 下午 0...
應用程式
2,356 KB
2 個項目



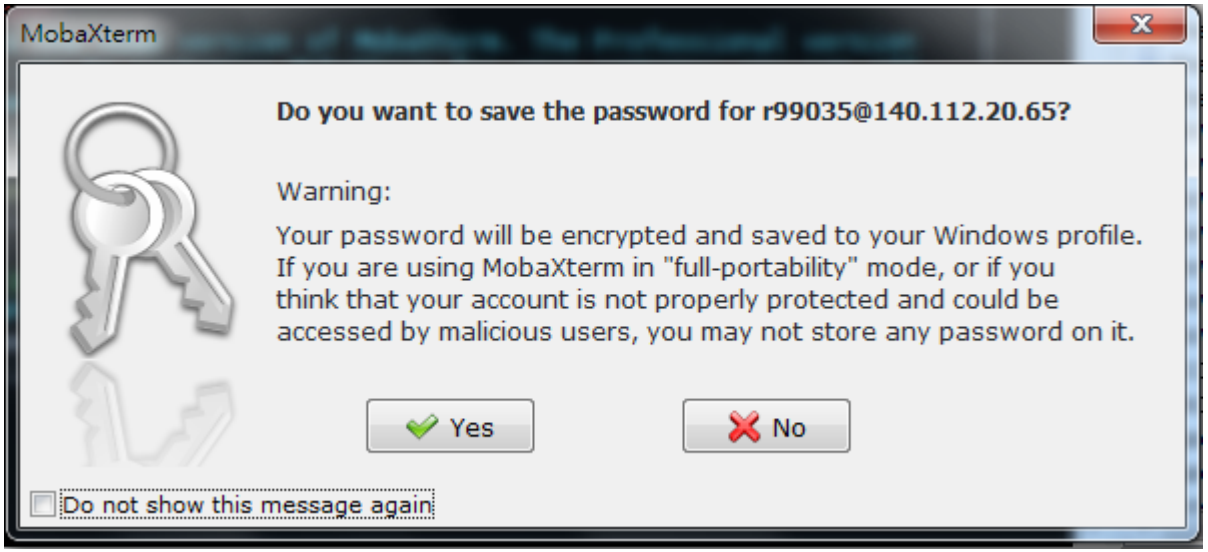
140.112.20.70	cad27	IBM x260	Intel Xeon 64	3.2 GHz * 4	8 G	RHEL 4
140.112.20.71	cad28	IBM x260	Intel Xeon 64	3.2 GHz * 4	8 G	RHEL 4
140.112.20.72	cad29	IBM e336	Intel Xeon 64	3.2 GHz	5 G	RHEL 4
140.112.20.73	Cad30	IBM X3650	Intel Xeon 64	2 GHz * 16	12 G	SUSE 11

- ssh user_account@work_station_ip
  - Ex: ssh b00500@140.112.20.70

Account for b00901500 **IP address for cad27**



- Enter your password



- Save the password if you want

Workstations

- The first time you login the workstation, type the following command
  - mv .cshrc cshrc

```
[2012-10-24 20:34.06]  ~
[cdyerty.cdyerty-PC] ➤ ssh r00155@140.112.20.70
Last login: Wed Oct 24 20:17:36 2012 from xdn20o129.ee.ntu.edu.tw
##############################################################################

# For more information, please refer to http://cad.ee.ntu.edu.tw

# Please backup personal files by yourself.

# The new disk array is installed, please contact TA if your folders
  have any problem. (09/28/2009)

# The account of userd93, user98, and userb96 will be remove
  on 09/10/2012. Please backup personal data by yourself.
  Contact TA if you have any problems. (07/02/2011)

# All workstations will be off at 18:00 on 08/10/2012 for maintenance.
  (07/26/2012)

##############################################################################
[r00155@cad27 ~]$ mv .cshrc cshrc
```

- **To make sure that you can login with FTP(Important!!!)**

Workstations

- Basic instructions
  - Change Password: **passwd**
  - Log out: **logout**
  - Show processes: **ps** (processes and PIDs)
  - Delete a process: **kill -9** *PID*

- “**passwd**” – Change your password

```
1. 140.112.20.71   4. 140.112.20.70   7. 140.112.20.70
[r00155@cad27 ~]$ passwd
Changing password for user r00155.
Changing password for r00155
(current) UNIX password:
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[r00155@cad27 ~]$
```

- “**ps**” – Show process
- “**kill -9**” – Delete process

```
1. 140.112.20.71   4. 140.112.20.70   7. 140.112.20.70
cad27:/home/raid3_1/user00/r00155% ps
  PID TTY          TIME CMD
 7969 pts/1    00:00:00 tcsh
 8593 pts/1    00:00:01 Novas
 8608 pts/1    00:00:04 common_shell_ex
 8651 pts/1    00:00:00 ps
cad27:/home/raid3_1/user00/r00155% kill -9 8593
cad27:/home/raid3_1/user00/r00155%
```

Workstations

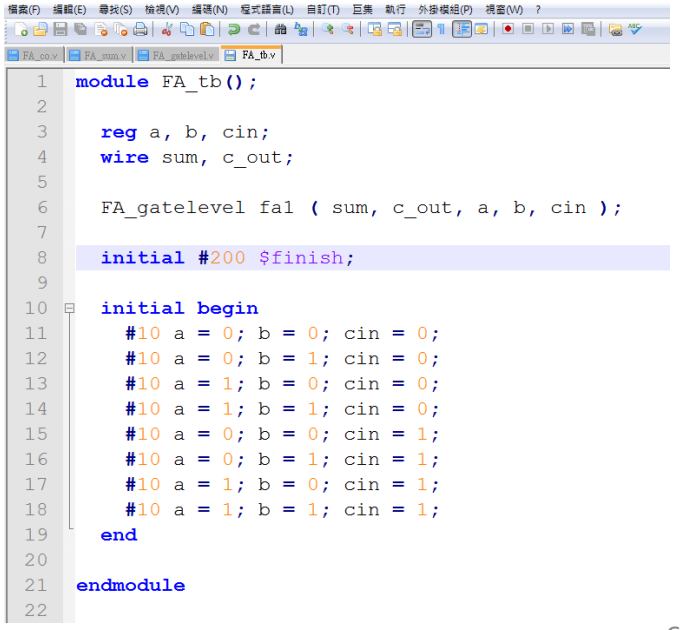
- Useful commands
  - **ls** : list files
  - **ls –a** : list all files (including the hidden files)
  - **ls –aux** : list all files with detailed information
  - **cp** : copy files from one folder/directory to another one
    - cp filename1 filename2
  - **cp –r** : copy the whole folder to another
  - **mkdir** : create a folder
  - **pwd** : display your current path

Workstations

- More useful commands
  - **cd** : change folder
  - **ps** : display process status by process identification number and name
  - **kill -9** *PID*: terminate a running process
    - kill -9 1234
  - **rm** : delete files
  - **rm –r** : remove the whole folder
  - **quota –v** : show disk space
  - **tar** : pack and compress files
    - **-cvf** : for creating compressed file
    - **-xvf** : for extracting compressed file
  - **mv** : move or rename file
  - **exit** : turn the terminal off
  - **logout**

Useful program

- NotePad ++
  - Source code editor and Notepad replacement that supports several languages
  - Running in the MS Windows environment
  - http://notepad-plus-plus.org/



Verilog-XL and NC-verilog

- Verilog-XL
  - Designed by Phil Moorby, the father of verilog
  - Interpreter of verilog
  - Designed for syntax checking and simulation
- NC-verilog
  - Designed by Cadence
  - Inherited from Verilog-XL
- In the following examples, we use NC-verilog.

Run Simulation

Example : Full Adder


```
module test();
  reg  a , b , cin;            //signal declaration
  wire sum , cout;

  //Instance
  FA my_fa1 (.sum(sum),.co(cout),.a(a),.b(b),.ci(cin));

  initial #100 $finish;        //stop simulation

  initial begin
    $dumpvars();
    $dumpfile("FA.vcd");
  end

  initial begin                // stimulus patterns
    #10 a = 0; b = 0; cin = 0;  // t = 10
    #10 a = 0; b = 1; cin = 0;  // t = 20
    #10 a = 1; b = 0; cin = 0;  // t = 30
    #10 a = 1; b = 1; cin = 0;  // t = 40
    #10 a = 0; b = 0; cin = 1;  // t = 50
    #10 a = 0; b = 1; cin = 1;  // t = 60
    #10 a = 1; b = 0; cin = 1;  // t = 70
    #10 a = 1; b = 1; cin = 1;  // t = 80
  end
endmodule
```

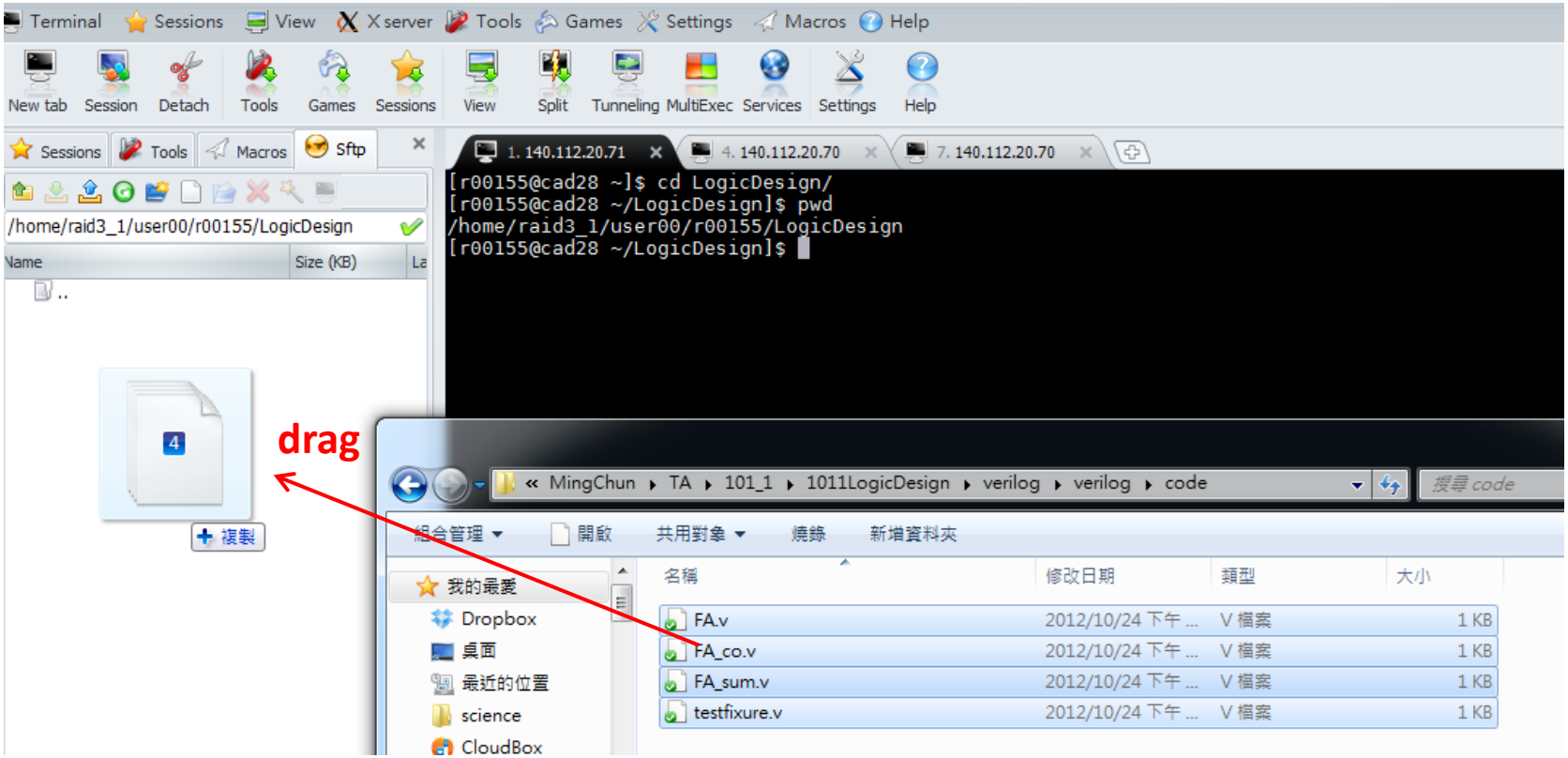
```
module FA_sum (sum,a,b,ci);

  output sum;
  input  a,b,ci;

  xor g1 (sum,a,b,ci);

endmodule
```

```
module FA_co (co, a, b, ci);
    output co;
    input  a, b, ci;
    wire   ab, bc, ca;

    and g0(ab, a, b);
    and g1(bc, b, ci);
    and g2(ca, ci, a);
    or  g3(co, ab, bc, ca);
endmodule
```

```
module FA (sum,co,a,b,ci);

  output sum,co;
  input  a,b,ci;

  FA_co  ins_C (.co(co),.a(a),.b(b),.ci(ci));
  FA_sum ins_S (.sum(sum),.a(a),.b(b),.ci(ci));

endmodule
```

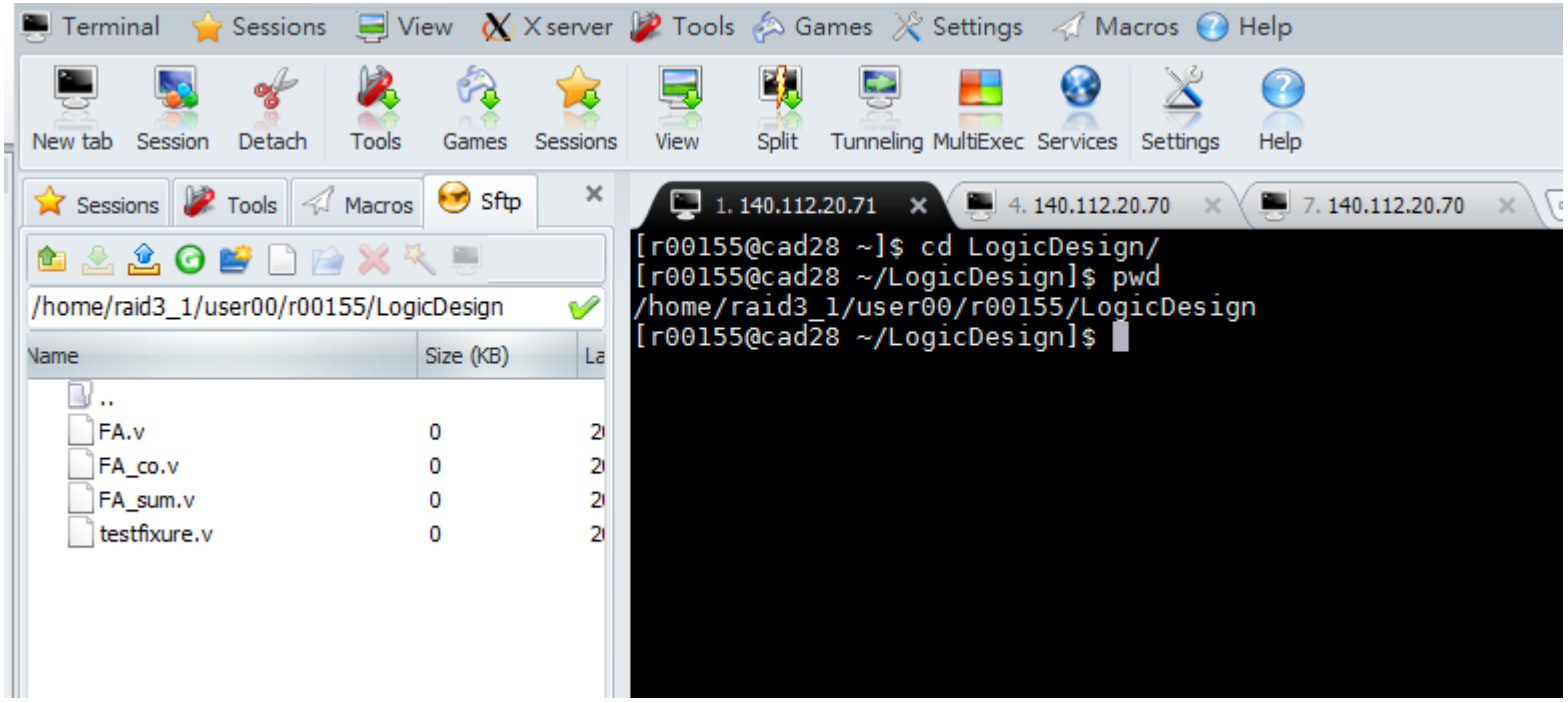
Run Simulation

Step 1
Upload the file



Run Simulation

Step 1
Upload the file



Run Simulation

Step 2
Source the source file of the tool (ncverilog & nWave)

- Remember to source the source file of the **ncverilog** and **nWave**
  - source ~cvsd/cvsd.cshrc
  - source ~cvsd/verdi.cshrc

```
1. 140.112.20.70
[r00155@cad27 ~/LogicDesign]$ source ~cvsd/cvsd.cshrc
Limit: Command not found.
 16:05:01 up 2 days, 21:09, 27 users,  load average: 2.8
Platform = LINUX
#######################################################
#         64BIT is the default mode                   #
#    If you want to run 64BIT mode,                   #
#    please set the LD_LIBRARY_PATH and SHLIB_PATH    #
#    to path of 32BIT by yourself.                    #
#######################################################
Platform = amd64
```

```
Platform = amd64
Platform = amd64
cad27:/home/raid3_1/user00/r00155/LogicDesign% source ~cvsd/verdi.cshrc
Platform = LINUX64
#######################################################
#         64BIT is the default mode                   #
#    If you want to run 64BIT mode,                   #
#    please set the LD_LIBRARY_PATH and SHLIB_PATH    #
#    to path of 32BIT by yourself.                    #
#######################################################
```

Run Simulation

Step 3
Run the simulation

- Run simulation
  - ncverilog +access+r testfixure.v FA.v FA_sum.v FA_co.v
    - +access+r : authorize to open file
    - testfixure.v : testbench should be placed at first file

```
cad27:/home/raid3_1/user00/r00155/LogicDesign% ncverilog +access+r testfixure.v FA.v FA_sum.v FA_co.v
ncverilog: 08.10-p002: (c) Copyright 1995-2008 Cadence Design Systems, Inc.
file: testfixure.v
        module worklib.test:v
                errors: 0, warnings: 0
file: FA.v
        module worklib.FA:v
                errors: 0, warnings: 0
file: FA_sum.v
        module worklib.FA_sum:v
                errors: 0, warnings: 0
file: FA_co.v
        module worklib.FA_co:v
```

Run Simulation

- How to observe the timing diagram?
  - **$dumpvars();**
  - **$dumpfile("FA.vcd");**
  - nWave (part of debussy)

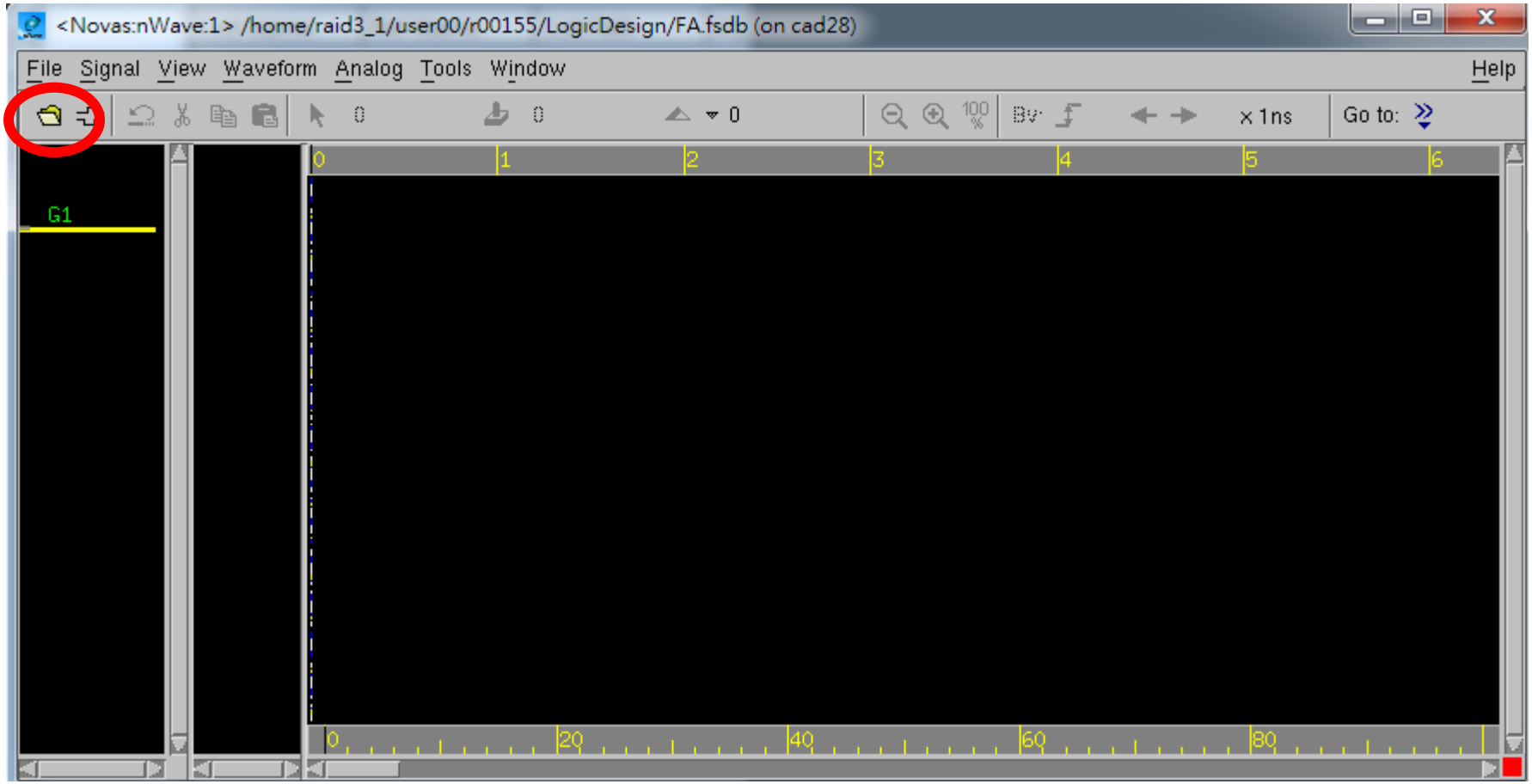
```
module test();
  reg  a , b , cin;              //signal declaration
  wire sum , cout;

  //Instance
  FA my_fa1 (.sum(sum),.co(cout),.a(a),.b(b),.ci(cin));

  initial #100 $finish;          //stop simulation

  initial begin
    $dumpvars();
    $dumpfile("FA.vcd");
  end
```

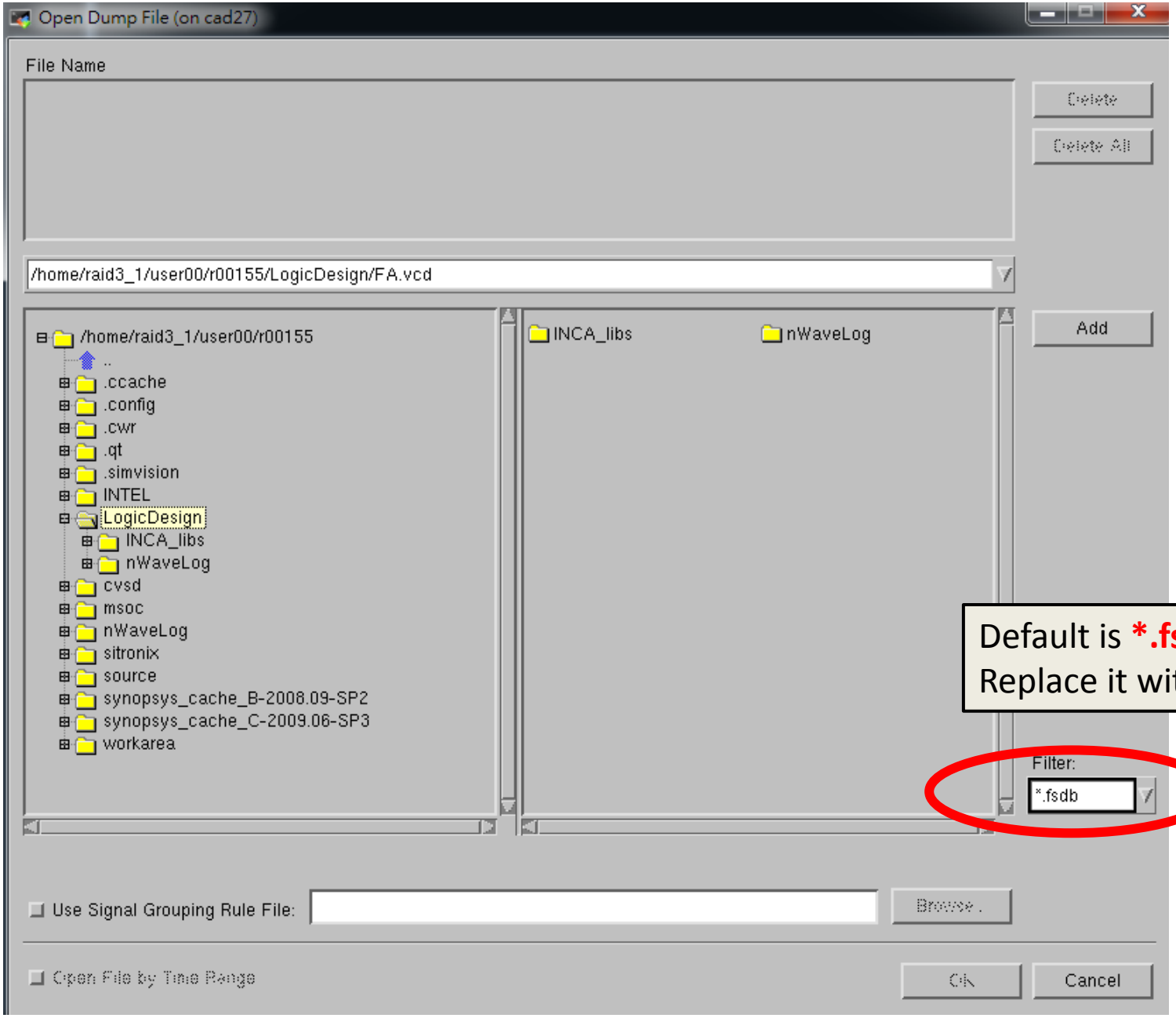
ncverilog +access+r testfixure.v FA.v FA_sum.v FA_co.v

```
1. 140.112.20.71   4. 140.112.20.70   7. 140.112.20.70
cad28:/home/raid3_1/user00/r00155/LogicDesign% ncverilog +access+r testfixure.v FA.v FA_sum.v FA_co.v
ncverilog: 08.10-p002: (c) Copyright 1995-2008 Cadence Design Systems, Inc.
Recompiling... reason: file './testfixure.v' is newer than expected.
        expected: Wed Oct 24 21:12:53 2012
        actual:   Wed Oct 24 21:13:00 2012
file: testfixure.v
file: FA.v
file: FA_sum.v
file: FA_co.v
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Building instance overlay tables: ..................... Done
        Loading native compiled code:     ..................... Done
        Building instance specific data structures.
        Design hierarchy summary:
                            Instances  Unique
                Modules:            4       4
                Primitives:         1       1
                Registers:          3       3
                Scalar wires:       4       -
                Initial blocks:     3       3
                Cont. assignments:  1       1
        Writing initial simulation snapshot: worklib.test:v
Loading snapshot worklib.test:v ..................... Done
*Novas* Loading libsscore_ius81.so
ncsim> source /usr/cadence/IUS/cur/tools/inca/files/ncsimrc
ncsim> run
Novas FSDB Dumper for IUS, Release 2012.01, Linux, 01/13/2012
Copyright (C) 1996 - 2012 by SpringSoft, Inc.
*Novas* : Create FSDB file 'FA.fsdb'
*Novas* : Begin traversing the scopes, layer (0).
*Novas* : End of traversing.
Simulation complete via $finish(1) at time 100 NS + 0
./testfixure.v:8                initial #100 $finish;                        //stop simulation
ncsim> exit
cad28:/home/raid3_1/user00/r00155/LogicDesign% nWave &
```

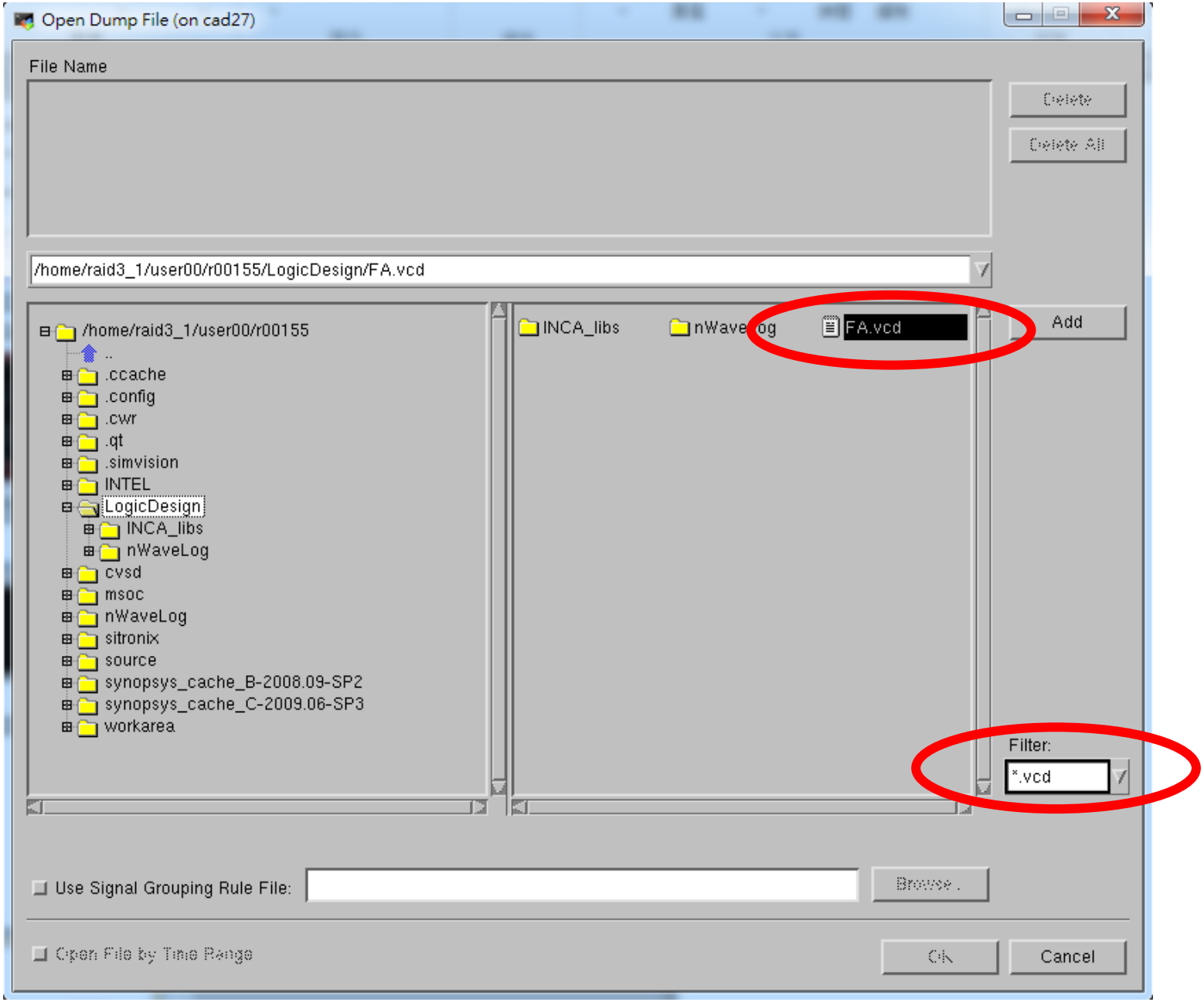
nWave & : open the tool “nWave”

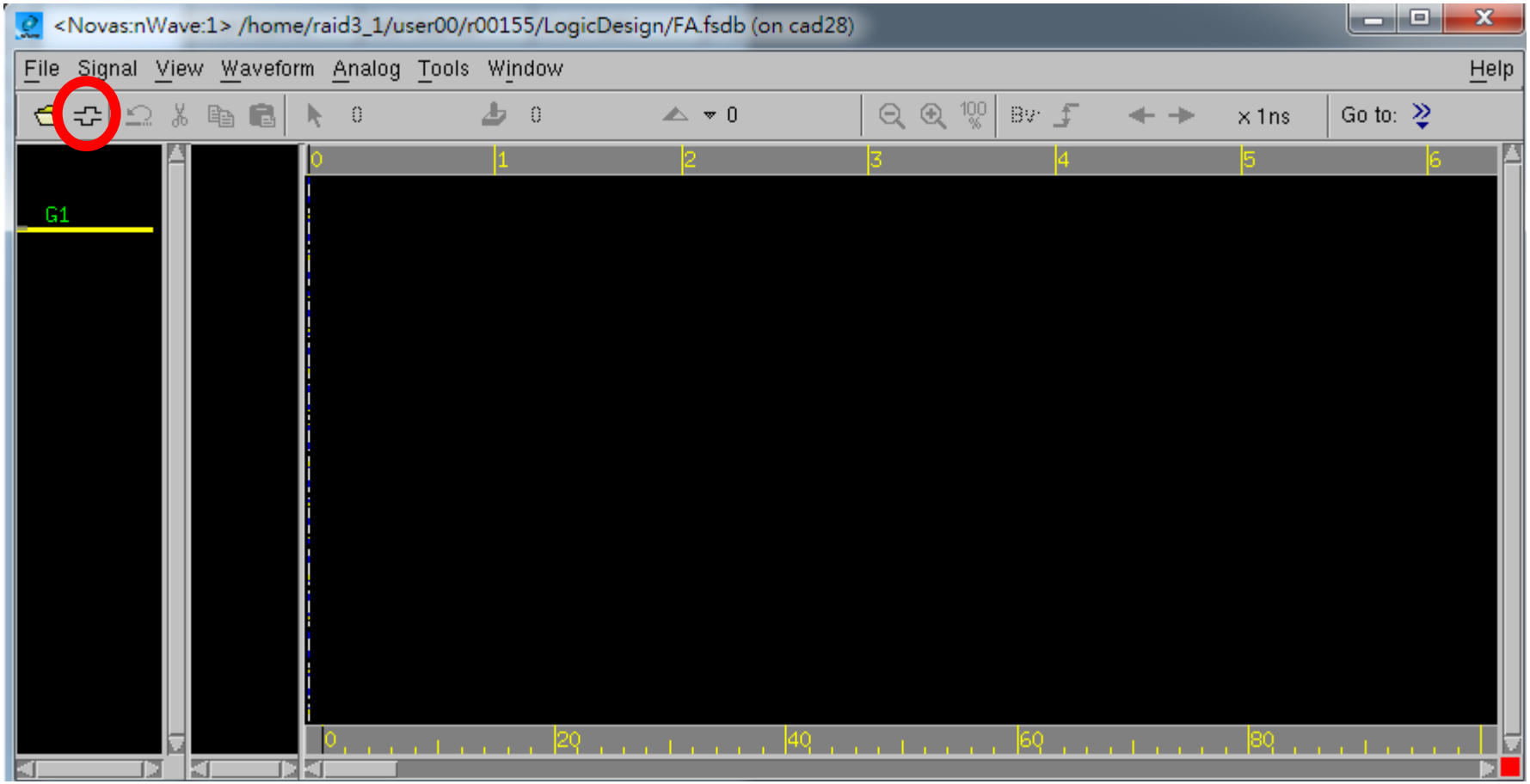
- “&” means open in background mode

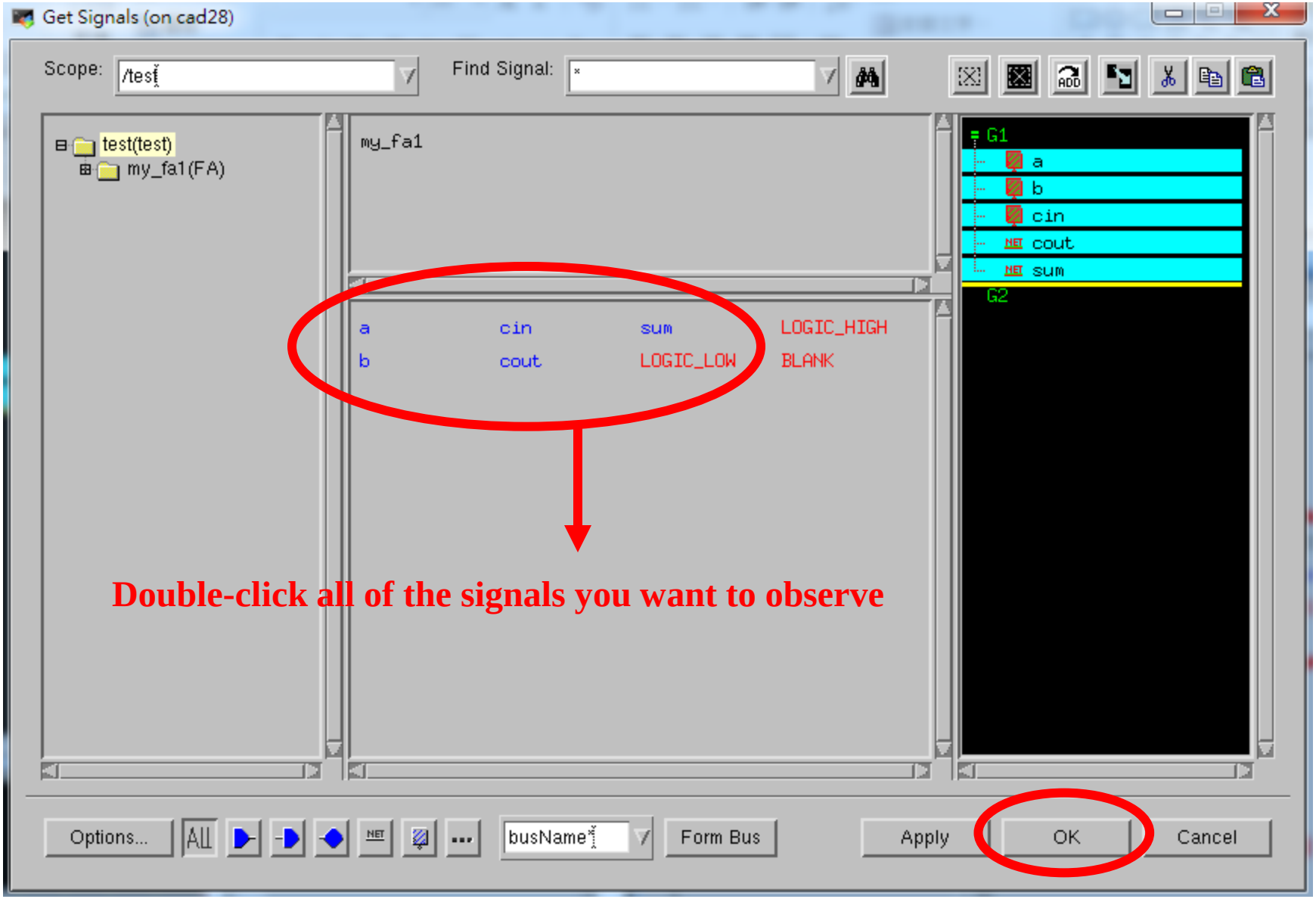

<Novas:nWave:1> /home/raid3_1/user00/r00155/LogicDesign/FA.fsdb (on cad28)
File Signal View Waveform Analog Tools Window
Help
x 1ns
Go to:
G1
0
1
2
3
4
5
6
0
20
40
60
80

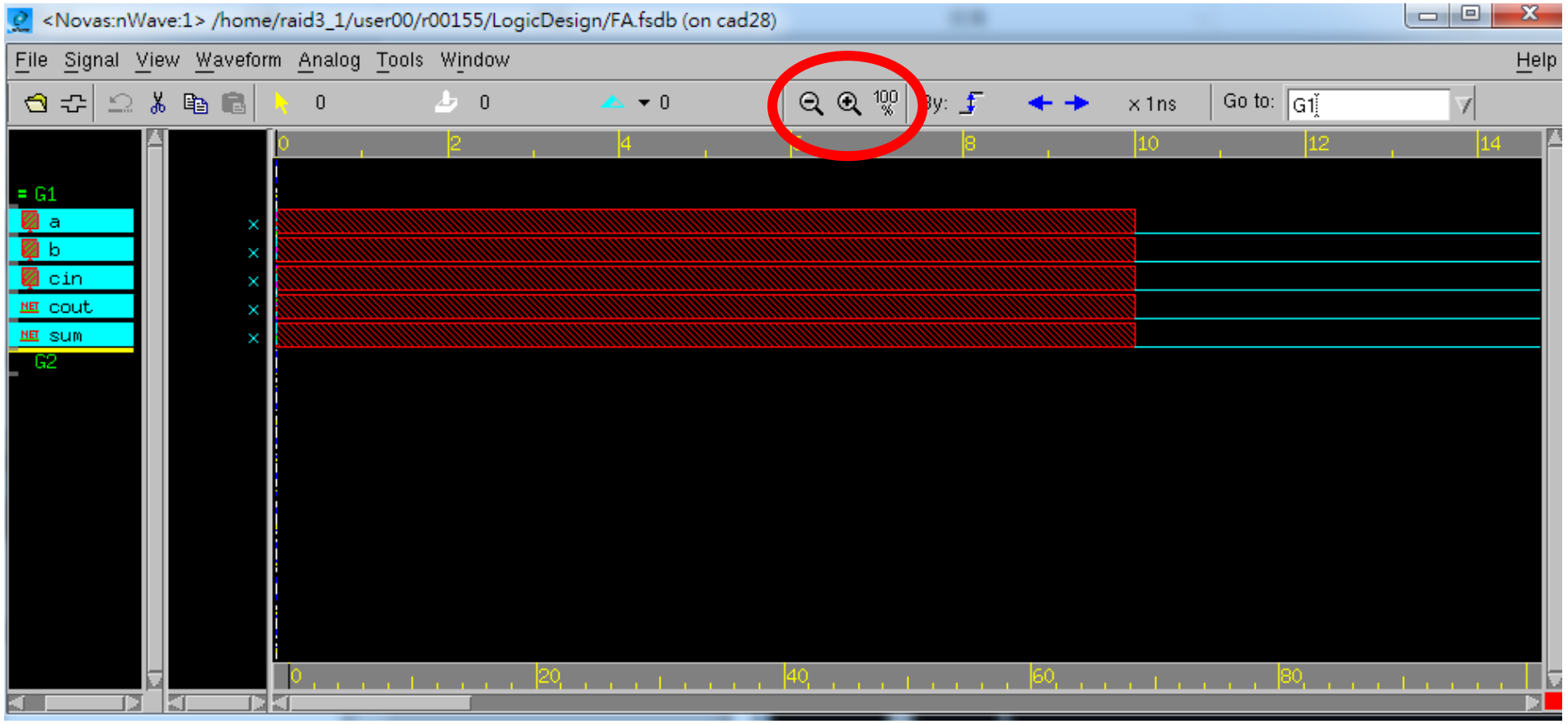


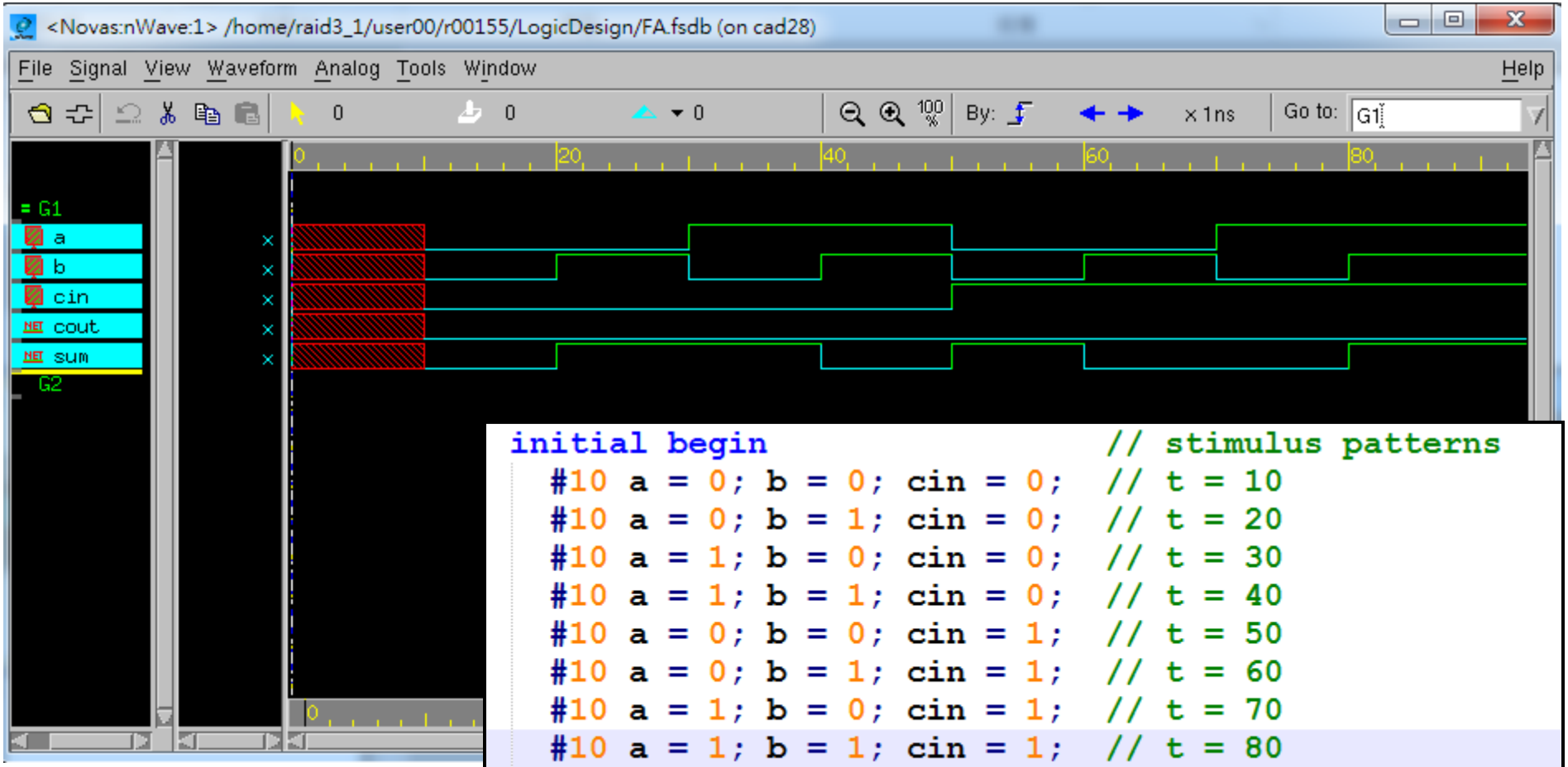
Default is ***.fsdb**
Replace it with ***.vcd**


Open Dump File (on cad27)
File Name
Delete
Delete All
/home/raid3_1/user00/r00155/LogicDesign/FA.vcd
/home/raid3_1/user00/r00155
..
.ccache
.config
.cwr
.qt
.simvision
INTEL
LogicDesign
INCA_libs
nWaveLog
cvsd
msoc
nWaveLog
sitronix
source
synopsys_cache_B-2008.09-SP2
synopsys_cache_C-2009.06-SP3
workarea
INCA_libs
nWaveLog
FA.vcd
Add
Filter:
*.vcd
Use Signal Grouping Rule File:
Browse...
Open File by Time Range
Ok
Cancel


<Novas:nWave:1> /home/raid3_1/user00/r00155/LogicDesign/FA.fsdb (on cad28)
File Signal View Waveform Analog Tools Window
Help
x 1ns
Go to:
G1
0
1
2
3
4
5
6
0
20
40
60
80


Get Signals (on cad28)
Scope:
/test
Find Signal:
test(test)
my_fa1(FA)
my_fa1
a
cin
sum
LOGIC_HIGH
b
cout
LOGIC_LOW
BLANK
G1
a
b
cin
cout
sum
G2
Double-click all of the signals you want to observe
Options...
All
busName
Form Bus
Apply
OK
Cancel


<Novas:nWave:1> /home/raid3_1/user00/r00155/LogicDesign/FA.fsdb (on cad28)
File Signal View Waveform Analog Tools Window
Help
Go to: G1
x 1ns
G1
a
b
cin
cout
sum
G2




```
initial begin                          // stimulus patterns
  #10 a = 0; b = 0; cin = 0;   // t = 10
  #10 a = 0; b = 1; cin = 0;   // t = 20
  #10 a = 1; b = 0; cin = 0;   // t = 30
  #10 a = 1; b = 1; cin = 0;   // t = 40
  #10 a = 0; b = 0; cin = 1;   // t = 50
  #10 a = 0; b = 1; cin = 1;   // t = 60
  #10 a = 1; b = 0; cin = 1;   // t = 70
  #10 a = 1; b = 1; cin = 1;   // t = 80
end
```

Preview of Lab Questions

Verilog Lab

- The lab questions (Homework) are due on **12/06/2013**
- The attendance of verilog lab is counted into grading (Participation – 2%)
- Lab questions are available on course website
- Information for the verilog lab
  - Time slots: 11/18~22, 11/25~11/29 – 18:00~20:30
  - Location: EEii-130

Creators of the Slides

- Ver. 1: *Chen-han Tsai*
- Ver. 2: *Chih-hao Chao*
- Ver. 3: *Xin-Yu Shi*
- Ver. 4: *Bo-Yuan Peng*
- Ver. 5: *Chieh-Chuan Chiu & Chieh-Chi Kao*
- Ver. 6: *Yu-Hao Chen & Ming-Chun Hsiao*