Verilog

Author : 2004 Chia-Hao Lee (matchbox@si2lab.org)

2006revised Yi-Min Lin (ymlin@si2lab.org)

2008revised Chien-Ying Yu (cyyu@si2lab.org)

2008revised Chi-Heng Yang (kevin@oasis.ee.nctu.edu.tw)

2009revised Yung-Chih Chen (ycchen@oasis.ee.nctu.edu.tw)

2009revised Chia-Lung Lin (jllin@si2lab.org)

2010revised Ju-Hung Hsiao (ju0909@si2lab.org)

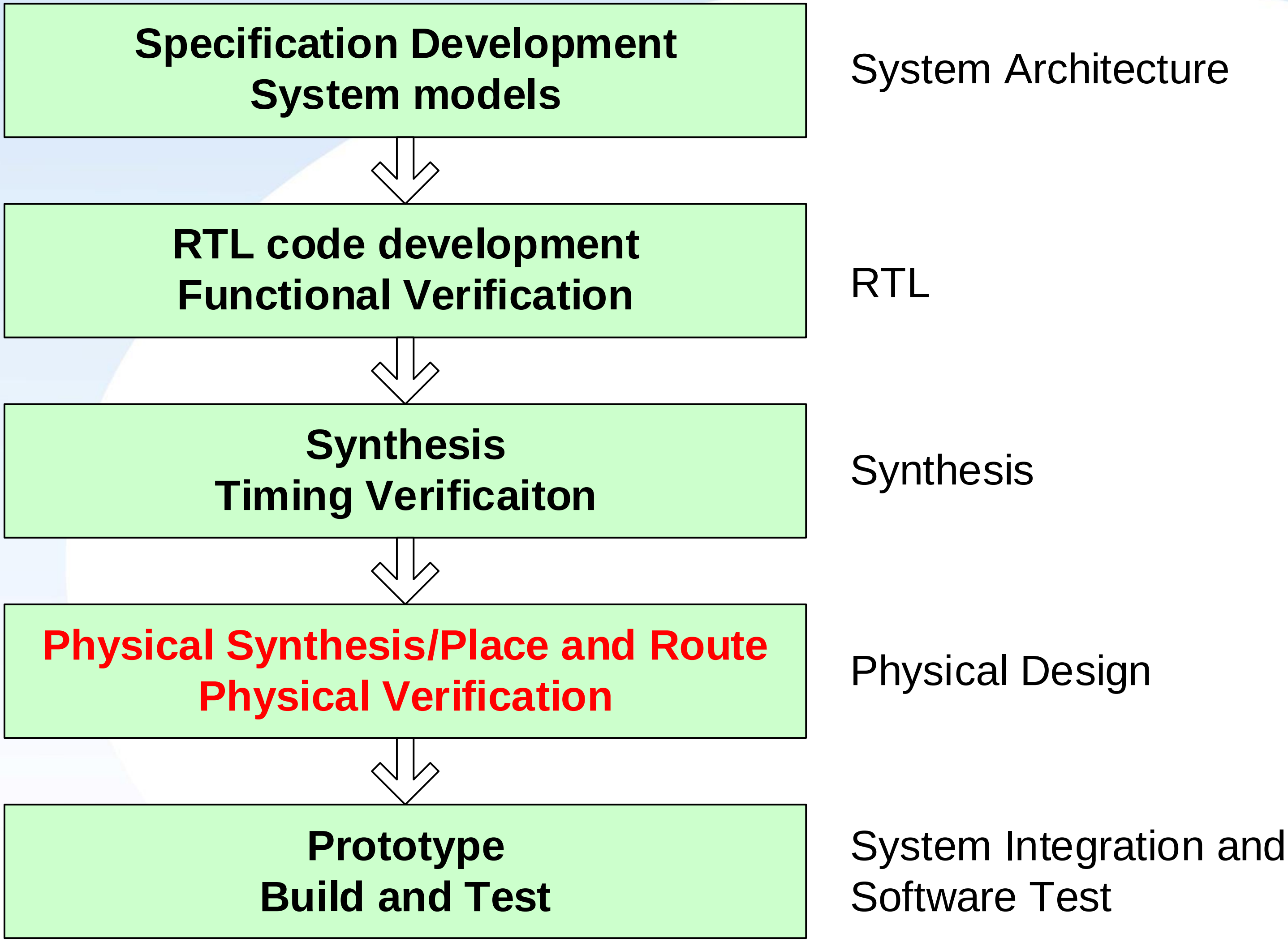
2015revised Heng-Wei Hsu (hengwzx@si2lab.org)

Lecturer : Eugene Lee (eugene@ieee.org)

Outline

- ✓ Introduction to Design Flow
- ✓ Basic Description of Verilog
- ✓ nWave

Design Flow



Specification Development
System models

RTL code development
Functional Verification

Synthesis
Timing Verificaiton

Physical Synthesis/Place and Route
Physical Verification

Prototype
Build and Test

System Architecture

RTL

Synthesis

Physical Design

System Integration and
Software Test

Cell-based ASIC

- ✓ **Cell-based IC (CBIC)**
  - – use ***pre-designed*** logic cells (known as standard cells)
  - – designers save time, money, and reduce risk
  - – each standard cell can be optimized individually
  - – all mask layers are customized
  - – custom blocks can be embedded

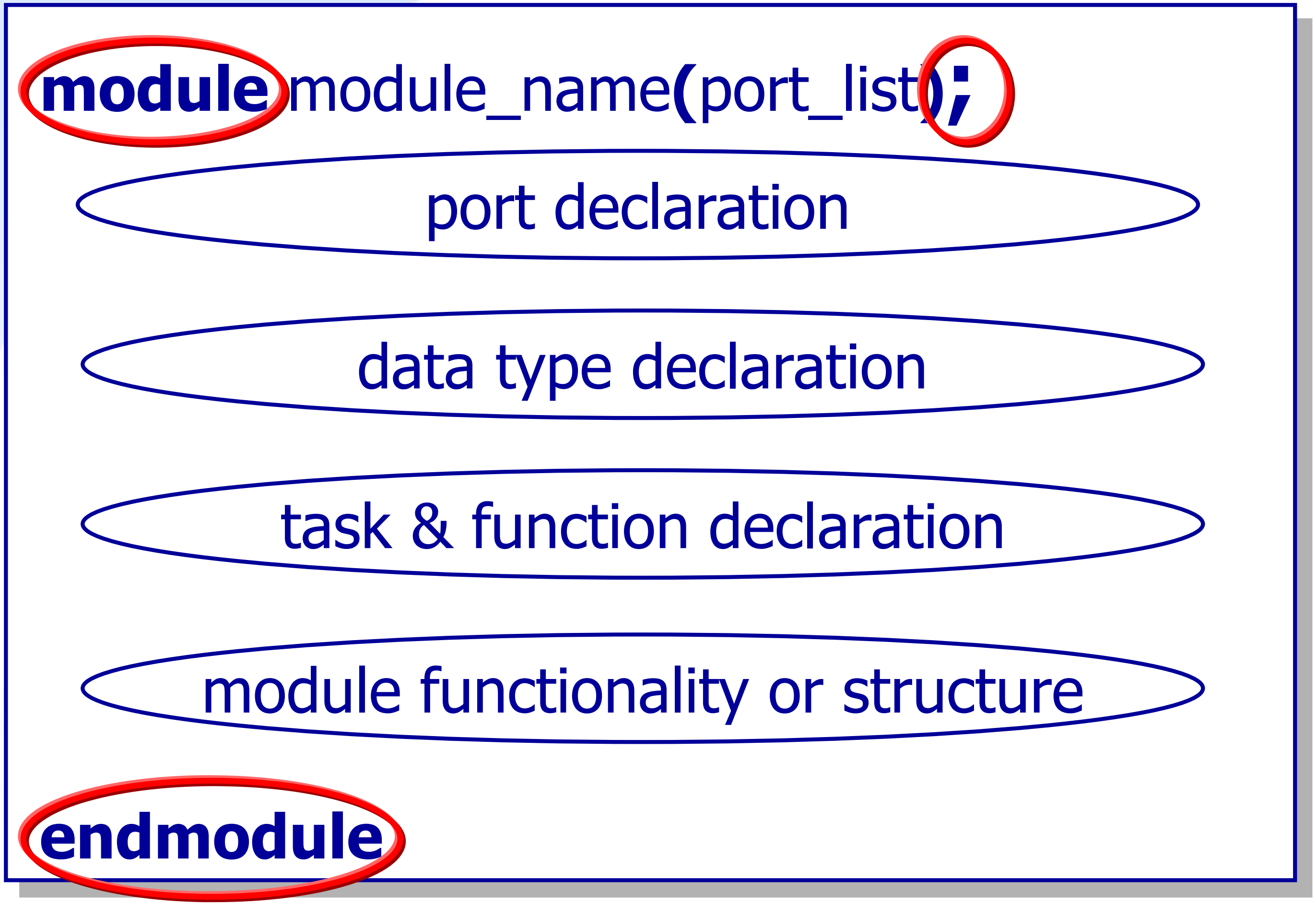
Full-Custom ASIC

- ✓ **An engineer designs some or all of the logic cells, circuits, layout specifically for one ASIC**
  - required cells/IPs are not available
  - existing cell libraries are not fast enough
  - logic cells are not small enough or consume too much power
  - technology migration (mixed-mode design)
  - ***demand long design cycle***

Basic Description of Verilog

- ✓ Lexical convention
- ✓ Data type & Port
- ✓ Gate level modeling
- ✓ Cell Library
- ✓ Data assignment
- ✓ Simulation Environment

A Module



module module_name**(**port_list**);**

port declaration

data type declaration

task & function declaration

module functionality or structure

endmodule

Lexical Convention

- ✓ **Identifiers**
  - **starts *only* with a letter or an _(underscore), can be any sequence of letters, digits, $, _ .**
  - **case-sensitive !**
    - e.g. shiftreg_a
      _bus3
      n$657
      12_reg ➔ illegal !!!!

- ✓ **Comments**
  - **single line : //**
  - **multiple line : /* … */**
    - e.g. // single line comment
    - e.g. /* multiple line
      comment */

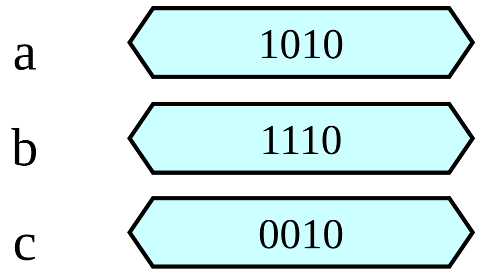
Lexical Convention (cont.)

- ✓ **Number Specification**
  - – **<size>’<base><value>**
    - <size> is the length of desired value in bits.
    - <base> can be b(binary), o(octal), d(decimal) or h(hexadecimal).
    - <value> is any legal number in the selected base.
    - When <size> is *smaller than <value>, then left-most bits of <value>*are truncated
    - When <size> is *larger than <value>, then left-most bits are filled,* based on the value of the left-most bit in <value>.
      - ◆ Left most '0' or '1' are filled with '0', 'Z' are filled with 'Z' and 'X‘ with 'X'
    - Default size is 32-bits decimal number
    - e.g. 4′d10 ➔ 4-bit, 10, decimal, store as 4’b1010
    - e.g. 6′hca ➔ 6-bit, store as 6′b001010 (truncated, not 11001010!)
    - e.g. 6′ha ➔ 6-bit, store as 6′b001010 (filled with 2-bit ‘0′ on left!)
    - e.g. 8′b0 ➔ 0000 0000
    - e.g. 12′b1111_0010_1011 ➔ 1111 0010 1011

Data Type

✓ Signed & Un-signed

- Verilog-1995
  - Signed : integer
  - Unsigned : reg, time, and all net data type.
- Verilog-2001
  - allows reg variables and all net data types to be declared using the reserved keyword **signed**



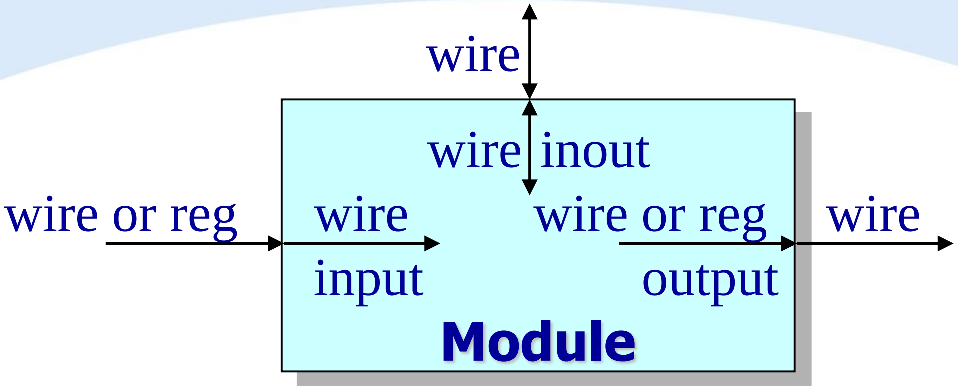
```
module verilog2001(a, b, c);

output signed [3:0] b;
output [3:0] c;
input  signed [3:0] a;

assign b = a >>> 2;
assign c = a >> 2;

endmodule
```

Port



- ✓ **Port declaration**
  - input : input port
  - output : output port
  - inout : bidirectional port
- ✓ **Port connection**
  - input : only wire can be assigned to represent this port in the module
  - output : wire or reg can be assigned to represent this port out of module
  - inout : register assignment is forbidden neither in module nor out of module

Module Connection

- ✓ **Modules connected by port order (implicit)**
  - – Here order should match correctly. Normally it is not a good idea to connect ports implicit. Could cause problem in debug, when any new port is added or deleted.
  - – e.g. :

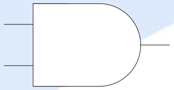
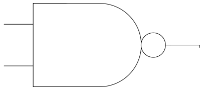
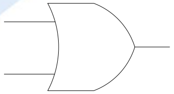
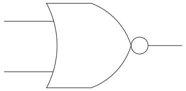
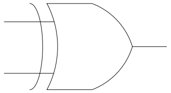
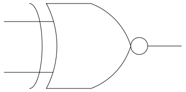
```
FA  U01( A, B, CIN, SUM, COUT );
```

- ✓ **Modules connect by name (explicit)**
  - – Here name should match correctly.
  - – e.g. :

```
FA U01(  .a(A), .b(B), .cin(CIN), .sum(SUM), .cout(COUT) );
```

Gate-Level Modeling

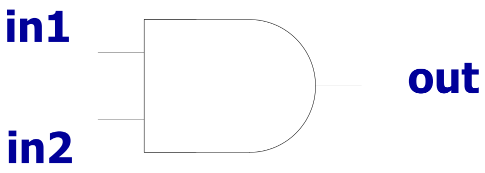
✓ **Primitive logic gate**

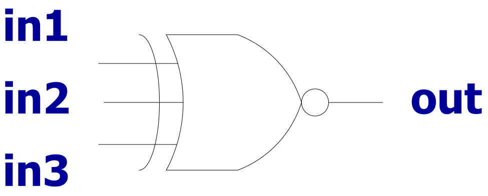
- and 
- nand 
- or 
- nor 
- xor 
- xnor 

The gates have *one scalar output and multiple scalar inputs.* The 1st terminal in the list of gate terminals is an output and the other terminals are inputs.

-- can use without instance name → e.g. and(out, in1, in2) ;

-- can use with multiple inputs → e.g. xor(out, in1, in2, in3) ;



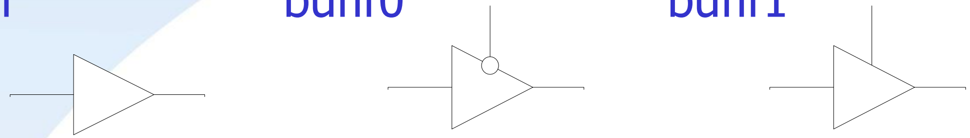


Gate-Level Modeling (cont.)

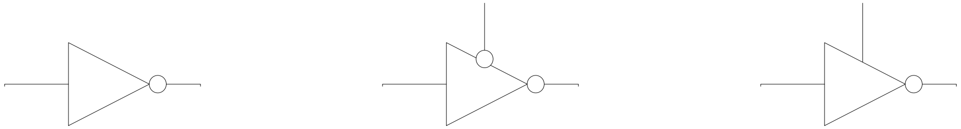
✓ **Primitive logic gate**

– buf/not gates

- buf bufif0 bufif1

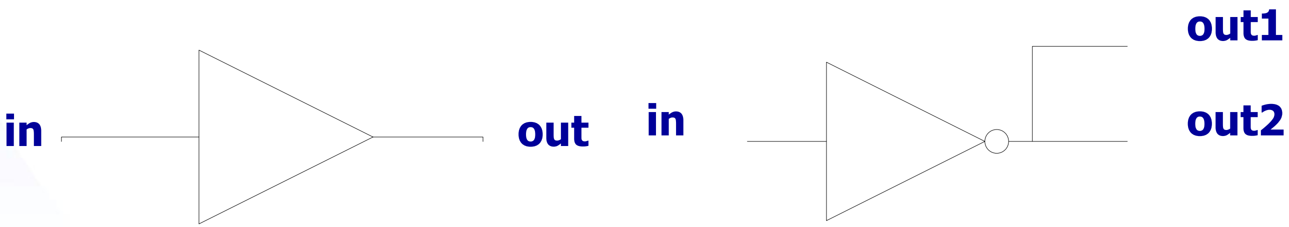


- not notif0 notif1

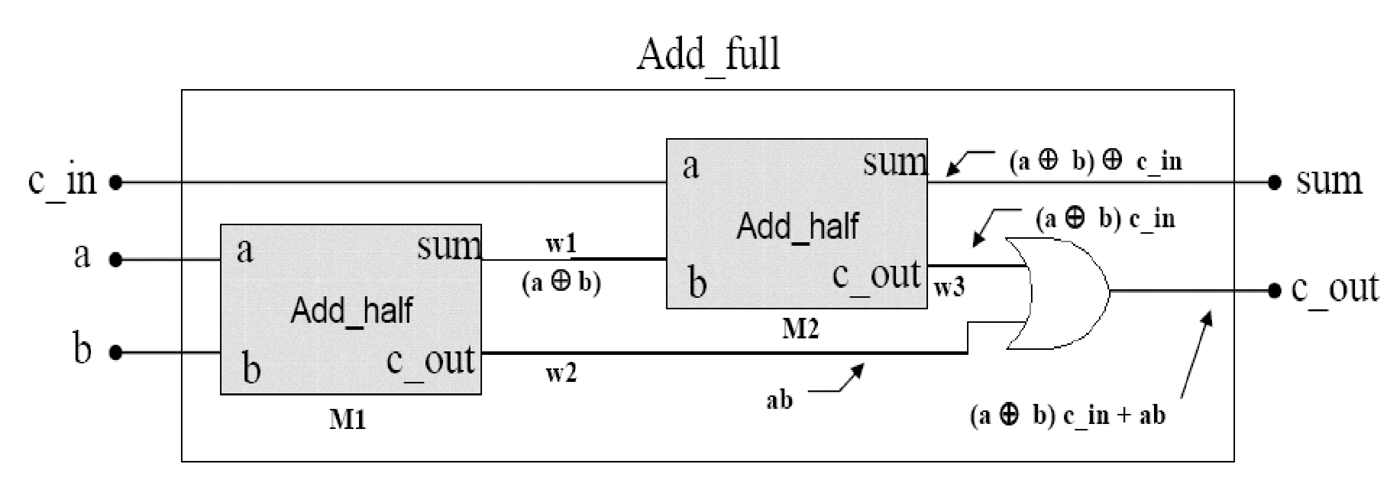


-- can use without instance name ➔ e.g. buf(out, in) ;

-- can use with multiple outputs ➔ e.g. not(out1, out2 ,in) ;



Example



```
module ADDR_FULL(A, B, C_IN, SUM, C_OUT);
input   A, B, C_IN;
output  SUM, C_OUT;
wire   w1, w2, w3;
ADDR_HALF M1(.A(A), .B(B), .SUM(w1), .C_OUT(w2));
ADDR_HALF M2(.A(C_IN), .B(w1), .SUM(SUM), .C_OUT(w3));
or (C_OUT, w2, w3);
endmodule
```

Introduction to Standard Cell Library

- ✓ **Standard Cell Library includes varies type of cells with different driving strength for optimization.**
- ✓ **Cell Type:**
  - Combinational: AND, OR, AOI, OAI, CLKBUF, TBUF,…
  - Sequential: DFF, DFF with Set/Reset, Scan DFF, Latch…
  - I/O Cell: Input/Output PAD, Power PAD………
  - Special: Antenna-Fix Cell, Fill Cell, Delay Cell…..
- ✓ **RAM/ROM Compiler and Register File Compiler:**
  - Generate required single/dual port memory and register files

Basic Description of Verilog

- ✓ **Logic Operator**
- ✓ **Operand**
- ✓ **Delay Expression**

Assignments

- ✓ **The assignment is always active**
  - – Whenever any change on the RHS of the assignment occurs, it is evaluated and assigned to the LHS.
    - Ex:

```
wire [3:0] a, X;
assign a = b + c;     // continuous assignment
assign X = {a[3] , a[2] , a[1] , a[0]};
```

- ✓ **Net declaration assignment**
  - – An equivalent way of writing net assignment statement.
  - – Can be declared once for a specific net.
    - Ex:

```
wire [3:0]  a = b + c; // net declaration   assignment
// is equivalent to the above description
```

Note : It's not allowed which required a concatenation on the LHS

i.e. `wire [4:0] {cout , sum} =  b+ c ;` *Wrong!*

Delay Expression

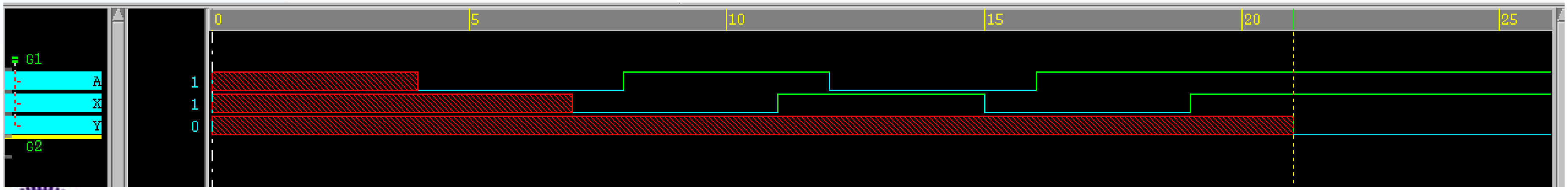
✓ Syntax for <mintypmax_expression>

- <mindelays> : <typdelays> : <maxdelays>
  - mindelays : best conditional delay
    typdelays : typical conditional delay
    maxdelays : worst conditional delay
    - ◆ +mindelays ➔ select minimum expression
      +typdelays ➔ select typical expression
      +maxdelays ➔ select maximum expression
      default : typical expression

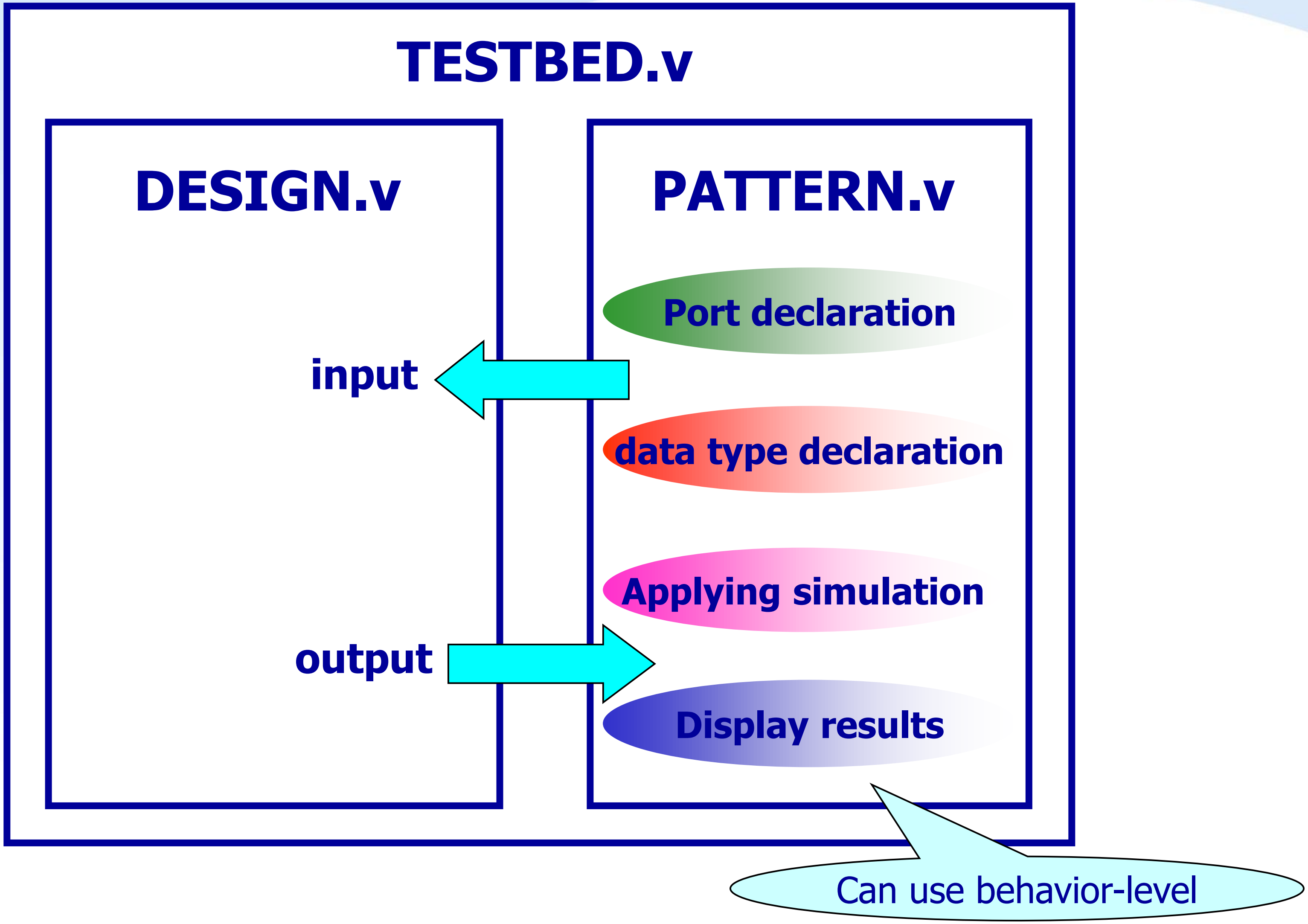
  <Example>

  ◆
```
module DELAY(A, X, Y);
  input A;
  output X, Y;
  assign #(3:4:5) X = A;
  assign #(5:7:9) Y = ~A;
endmodule
```

  ◆ verilog delay.v +mindelays ➔ select minimum delay



Simulation Environment



Can use behavior-level

Simulation Environment (cont.)

TESTBED.v

```
`timescale 1ns/10ps
`include “MUX2_1.v”
`include “PATTERN.v”

module  TESTBED;

wire out,a,b,sel,clk,reset;

MUX2_1 mux(.out(out),.a(a),.b(b),.sel(sel));
PATTERN pat(.sel(sel),.out(out),.a(a),.b(b));

endmodule
```

Simulation Environment (cont.)

PATTERN.v

```
module PATTERN(sel,out,a,b);

input out;
output a,b,sel;

reg a,b,sel,clk,reset;
integer i;
parameter cycle=10;

always #(cycle/2) clk = ~clk;

initial begin
a=0;b=0;sel=0;reset=0;clk=0;
#3 reset = 1;
#10 reset = 0;
```

```
#cycle sel=1;
for(i=0;i<=3;i=i+1)
begin
#cycle {a,b}=i;
#cycle $display( "sel=%b, a=%b, b=%b,
                 out=%b" , sel, a, b, out);
end

#cycle sel=0;
for(i=0;i<=3;i=i+1)
begin
#cycle {a,b}=i;
#cycle $display( "sel=%b, a=%b, b=%b,
                 out=%b" , sel, a, b, out);
end

#cycle $finish;
end

endmodule
```

nWave

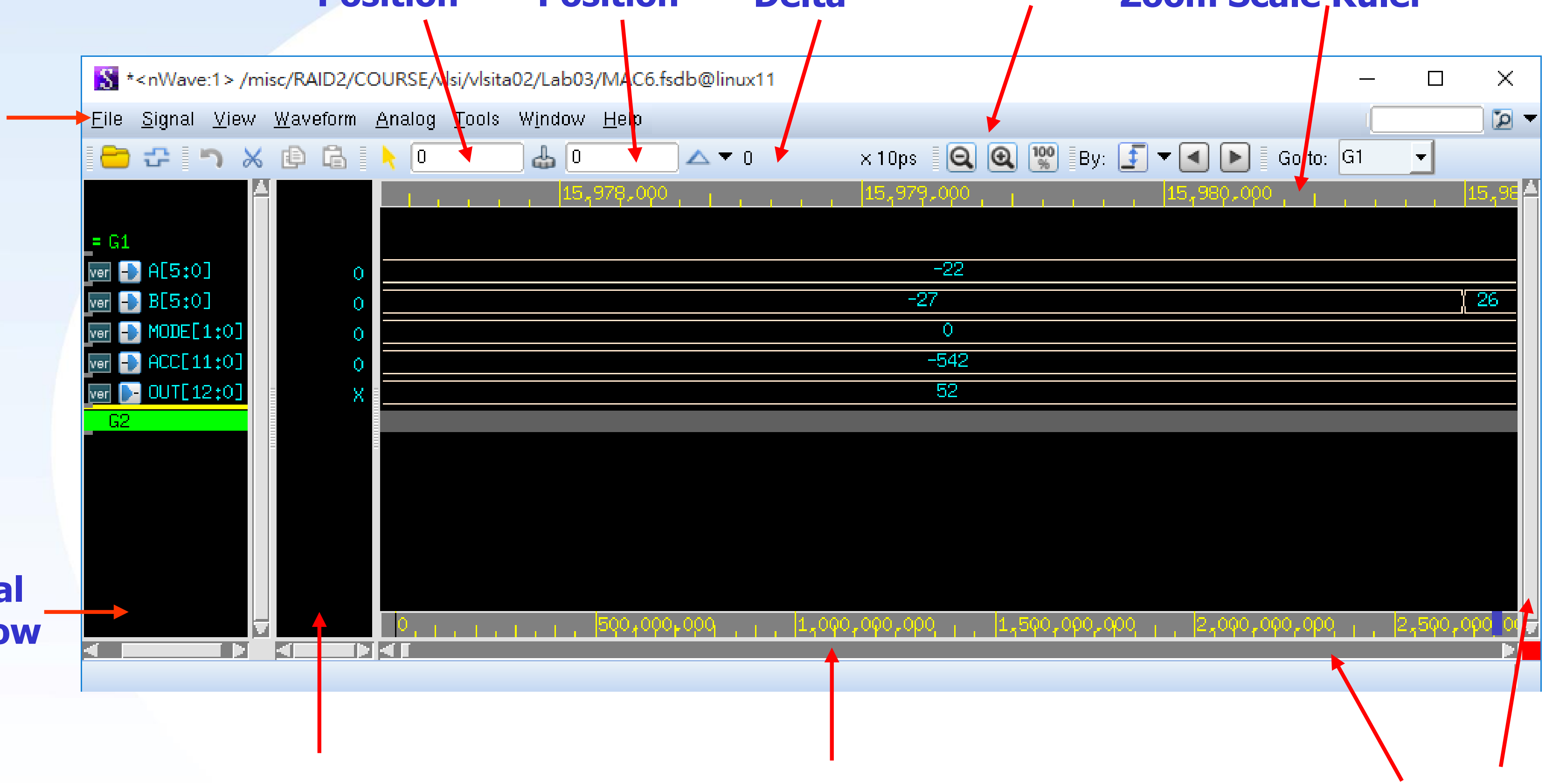
- ✓ Overview
- ✓ Import fsdb file
- ✓ Get signal
- ✓ Choose value format
- ✓ Grid on rising/falling edge
- ✓ Wave form alias
- ✓ Compare two signals
- ✓ Reload nWave

nWave

- ✓ Utilizing nTrace, nSchema and nState for debugging is just in *static* condition. So we need a powerful debugging tool, nWave, to analyze *dynamic* signal value condition.
- ✓ nWave is the most useful tool to help us debugging
- ✓ Invoke nWave
  - – By command : **nWave &**

nWave (cont.)

✓ Overview



*<nWave:1> /misc/RAID2/COURSE/vlsi/vlsita02/Lab03/MAC6.fsdb@linux11

Cursor: 0 Marker: 0 Delta: 0 Go to: G1

```
= G1
A[5:0]       0   -22
B[5:0]       0   -27   26
MODE[1:0]    0   0
ACC[11:0]    0   -542
OUT[12:0]    X   52
G2
```

Cursor Position

Marker Position

Delta

Tool Bar

Zoom Scale Ruler

Pull Down Menu

Signal Window

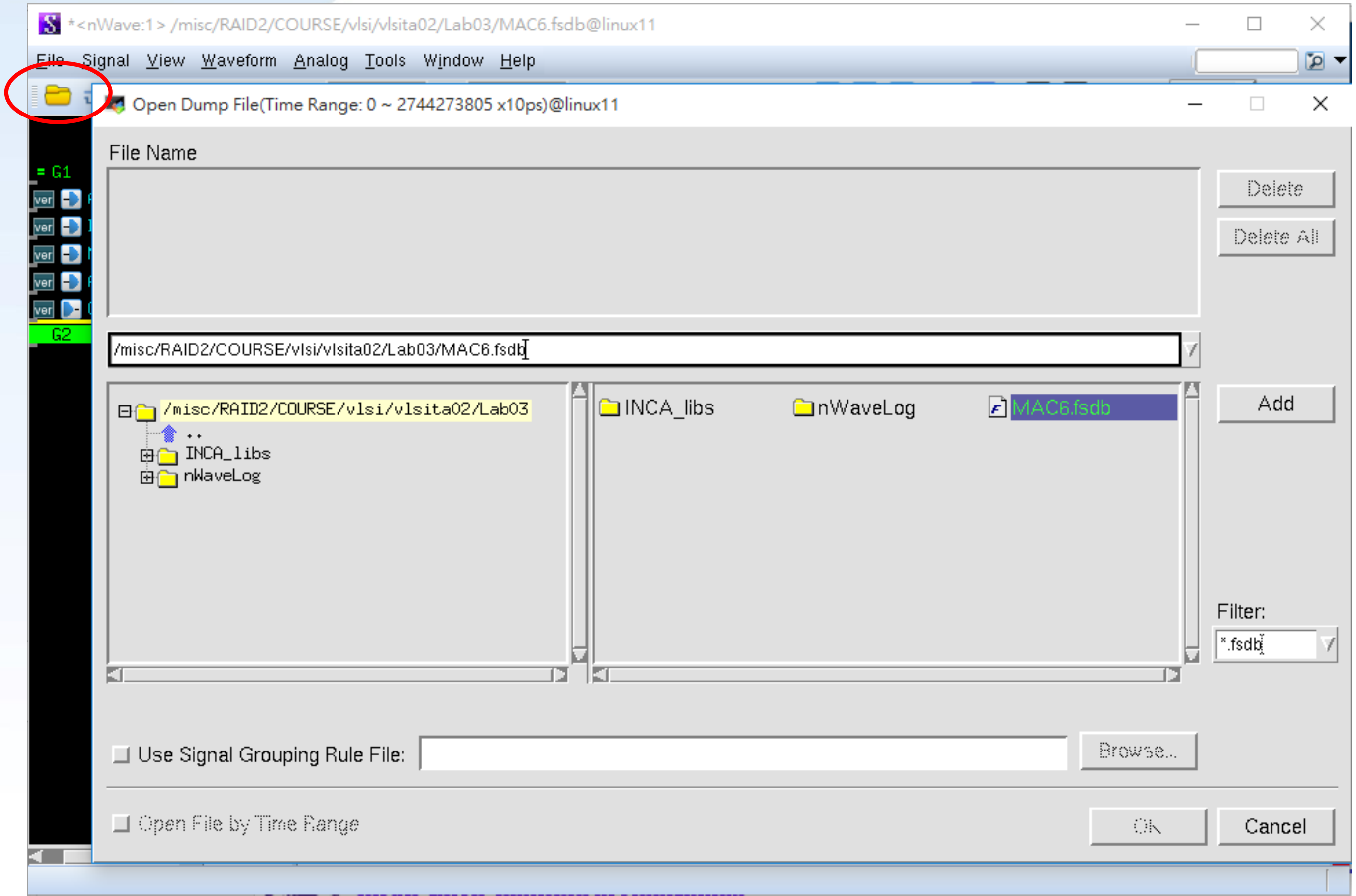
Value Window

Full Scale Ruler

Scroll Bar

nWave (cont.)

- ✓ **Open fsdb file**
  - Use ***File →Open…*** command

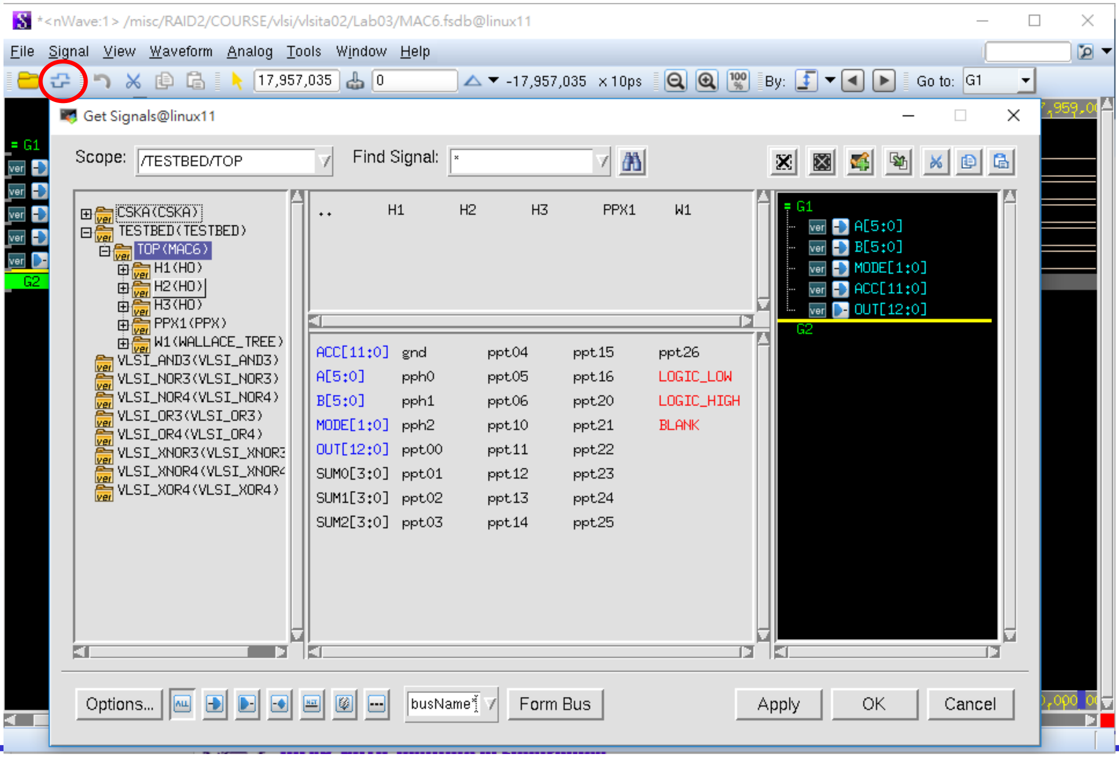


*<nWave:1> /misc/RAID2/COURSE/vlsi/vlsita02/Lab03/MAC6.fsdb@linux11
File Signal View Waveform Analog Tools Window Help
G1
G2
Open Dump File(Time Range: 0 ~ 2744273805 x10ps)@linux11
File Name
Delete
Delete All
/misc/RAID2/COURSE/vlsi/vlsita02/Lab03/MAC6.fsdb
/misc/RAID2/COURSE/vlsi/vlsita02/Lab03
..
INCA_libs
nWaveLog
INCA_libs
nWaveLog
MAC6.fsdb
Add
Filter:
*.fsdb
□ Use Signal Grouping Rule File:
Browse...
□ Open File by Time Range
OK
Cancel

nWave (cont.)

✓ Get signal

- Use ***Signal*** **➔** ***Get Signals...*** command
- Directly Drag & Drop signals from nTrace or nSchema to nWave
- Select objects and use ***Add Select Set To Wave*** in nSchema

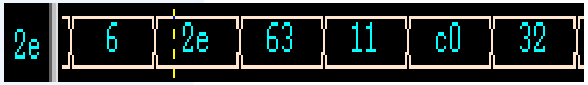

*<nWave:1> /misc/RAID2/COURSE/vlsi/vlsita02/Lab03/MAC6.fsdb@linux11
File Signal View Waveform Analog Tools Window Help
17,957,035
0
-17,957,035 x 10ps
By:
Go to: G1
Get Signals@linux11
G1
G2

Scope: /TESTBED/TOP

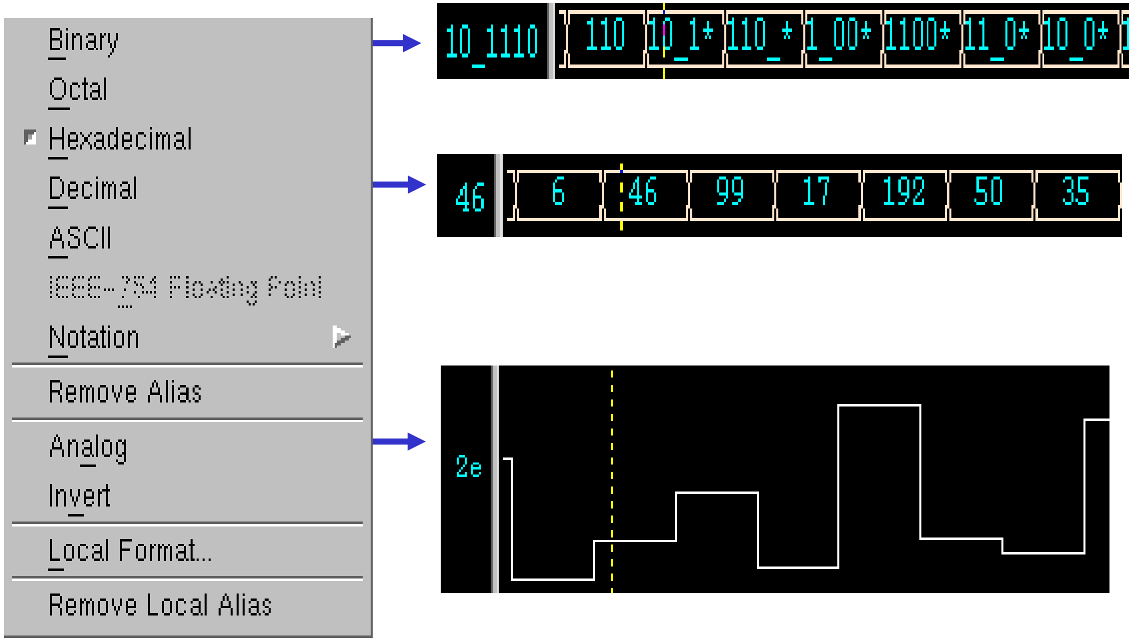
Find Signal: *

nWave (cont.)

- ✓ **Choose value format**
  - Waveform -> Signal Value Radix/Notation



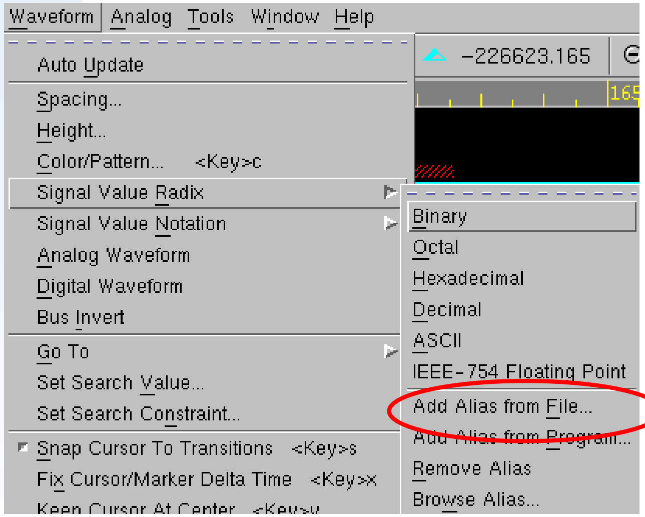
Default : Hexadecimal



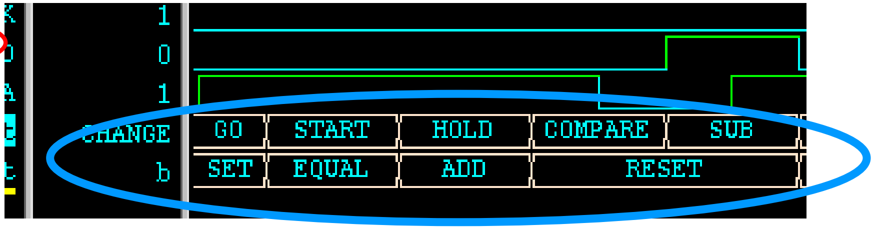
nWave (cont.)

✓ Waveform Alias

- Sometimes you want some specific value to represent some specific meaning
  - ***Waveform ➔ Signal Value Radix ➔ Add Alias from File***



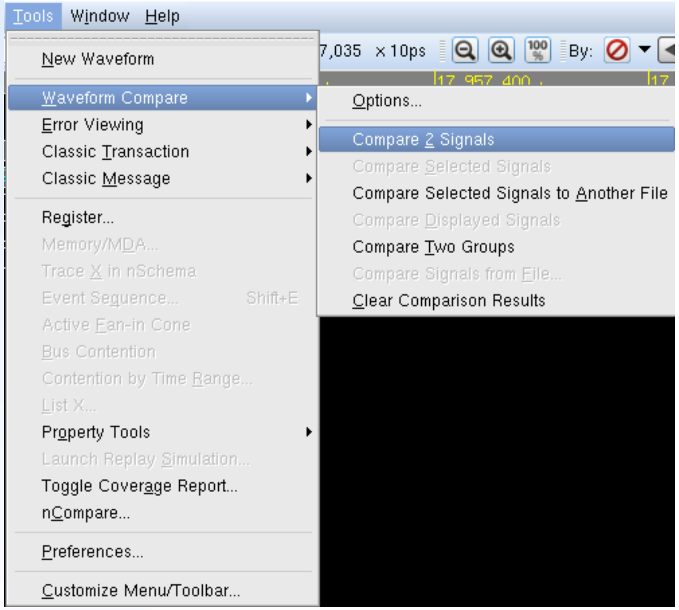
START	8’d0
HOLD	8’d1
RESET	8’d2
SUB	8’d3
…	
…	

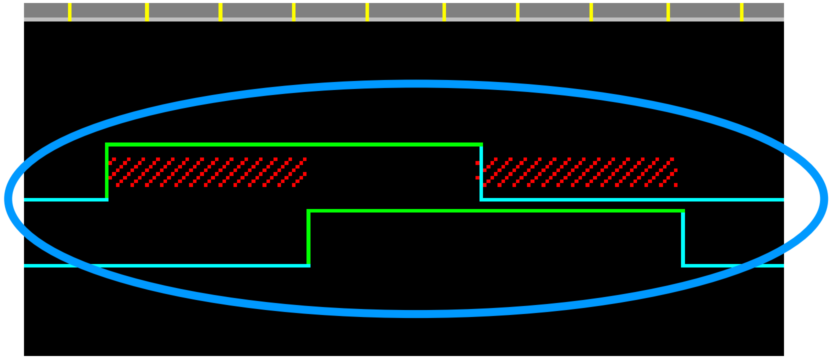


nWave (cont.)

✓ Compare two signals

- Compare two different signals to check your function
  - *Tool ➔ Waveform Compare ➔ Compare 2 Signals*
- If the signals have different values, it will sign red marks.

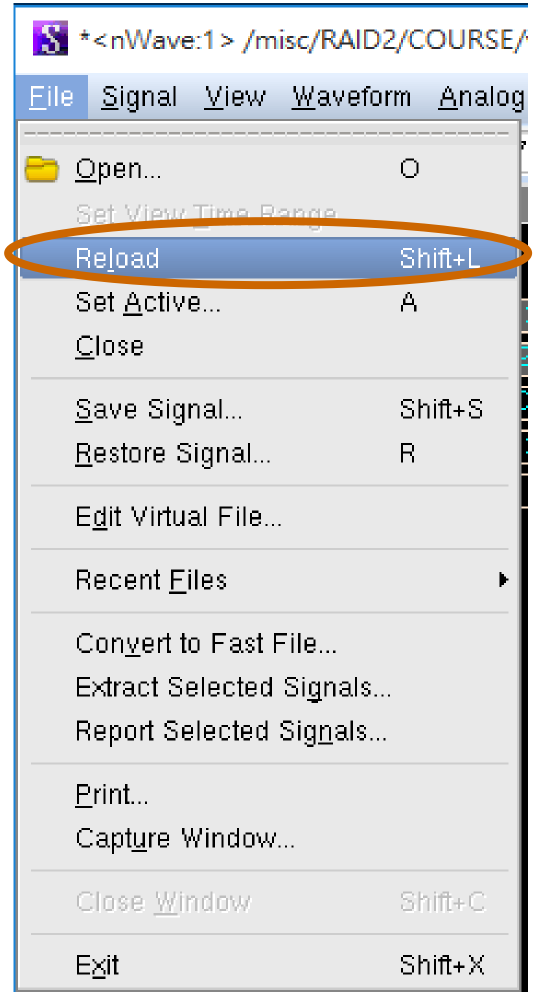




If you want to compare the same in two different fsdb files, you can drag from one and drop to another.

nWave (cont.)

- ✓ **Reload nWave**
  - Update fsdb file in Debussy database
    - ***File➔ Reload***



*<nWave:1> /misc/RAID2/COURSE/

Appendix

- ✓ Logical Operator
- ✓ Cell Library: Data Sheet
- ✓ Simulation Environment

Logical Operator

✓ Binary Operator Precedence

```
– !     ~                                  highest precedence
  *     /     %
  +     -
  <<    >>    <<<    >>>
  <     <=    >      >=
  ==    !=    ===    !==
  &
  ^     ^~
  |
  &&
  ||
  ? :                                      lowest precedence
```

– a < size -1 && b != c && index != last_one ➔ worse
(a < size -1) && (b != c) && (index != last_one) ➔ better

Logical Operator (cont.)

- Bit-wise Operator
  - NOT: A = ~ B;
  - AND: A = B & C;
  - OR: A = B | C;
  - XOR: A = B ^ C;
  - e.g. 4’b1001 | 4’b1100 ➔ 4’b1101
- Logical Operators: return 1-bit true/false
  - NOT: A = ! B;
  - AND: A = B && C;
  - OR: A = B || C;
  - e.g. 4’b1001 || 4’b1100 ➔ true, 1’b1

Logical Operator (cont.)

✓ **Logical Operators** -- the result is one bit

- ‘&&’ (and) , ‘||’ (or) , ‘!’ (not)
  - 0 : logical result → 0
    1 : logical result → 1
    x : either operand is ambiguous
  - The precedence of “&&” is greater than that of “||”
    - ◆ a & &b; a | | b → incorrect
    - ◆ a && b; a || b → correct

✓ **Bitwise Operators**

- ‘~’ (invert) , ‘&’ (and) , ‘|’ (or) , ‘^’ (exclusive or) , ‘~^’ (exclusive nor

~	
0	1
1	0
x	x

&	0	1	x
0	0	0	0
1	0	1	x
x	0	x	x

|	0	1	x
0	0	1	x
1	1	1	1
x	x	1	x

^	0	1	x
0	0	1	x
1	1	0	x
x	x	x	x

^~	0	1	x
0	1	0	x
1	0	1	x
x	x	x	x

✓ **Reduction Operators**

- ‘&’ (and) , ‘~&’ (nand) , ‘|’ (or) , ‘~|’ (nor) ,
- ‘^’ (exclusive or) , ‘~^’ (exclusive nor)
  - a bit-wise operation on a single operand to produce a single bit result

Logical Operator (cont.)

✓ Arithmetic Operators

- +(add) , -(sub) ,*(mul), /(div) , %(mod)

<Example>

```
module ARI_OP (A, B, A_ADD_B, A_SUB_B, A_MUL_B, A_DIV_B, A_MOD_B);
  input  [7:0] A, B;
  output [7:0] A_ADD_B, A_SUB_B, A_MUL_B, A_DIV_B, A_MOD_B;
  //A = 8'd3;
  //B = 8'd4;
  assign A_ADD_B = A + B; //A_ADD_B = 7
  assign A_SUB_B = A - B;  //A_SUB_B = -1(8'b1111_1111)
  assign A_MUL_B = A * B; //A_MUL_B = 12
  assign A_DIV_B = A / B;   //A_DIV_B = 0
  assign A_MOD_B = A % B; //A_DIV_B = 3
endmodule
```

Logical Operator(cont.)

- Conditional Description
  - if else
  - case endcase
  - ? : → c = sel ? a : b; // if (sel==1'b1)

```
                        //     c = a;
                        // else
                        //     c = b;
```

- Relational and equality(conditional)
  - <=, <, >, >=, ==, !=
  - i.e. if((a<=b) && (c==d) || (e>f))

Logical Operator (cont.)

✓ Relational Operators

- < (less than) , <= (less than or equal)
- > (larger than) , >= (larger than or equal)
  - 0 : false
    1 : true
    x : unknown bits in the operands
  - Have the same precedence

  <Example>

```
module REL_OP(A, B, C, REL1, REL2, REL3, REL4, REL5);
  input signed [3:0] A, B, C;
  output      REL1, REL2, REL3, REL4, REL5;
  //A = 4'b1110;
  //B = 4'b0001;
  //C = 4'b00x1;
  assign REL1 = A<B;    //REL1 = 1
  assign REL2 = A>B;    //REL2 = 0
  assign REL3 = A<=B;  //REL3 = 1
  assign REL4 = A>=B;  //REL4 = 0
  assign REL5 = A>C;    //REL5 = x
Endmodule
```

Logical Operator (cont.)

✓ Equality Operators

- === (equal) , !== (not equal) – synthesis tool doesn’t support!!
- == (equal) , != (not equal)
- Error information:

Error: /~/EQU_OP.v:4: case equality (===) is not supported by synthesis. (VER-189)

```
<Example>
module EQU_OP(A, B, EQU1, EQU2, EQU3, EQU4);
  input [3:0] A, B;
  output EQU1, EQU2, EQU3, EQU4;
  //A = 4’b01x0;
  //B = 4’b01x0;
  assign EQU1 = (A===B) ? 1:0; //EQU1 == 1
  assign EQU2 = (A!==B) ? 1:0; //EQU2 == 0
  assign EQU3 = (A==B) ? 1:0; //EQU3 == x
  assign EQU4 = (A!=B) ? 1:0; //EQU4 == x
endmodule
```

Logical Operator (cont.)

<Example> (for logical and bitwise operators)

```
module LOG_BIT_OP(A, B, LOG1, LOG2, LOG3, BIT1, BIT2, RED1, RED2);
  input [3:0] A, B;
  output      LOG1, LOG2, LOG3;
  output [3:0] BIT1, BIT2;
  output      RED1, RED2;
  //A = 4'b0000;
  //B = 4'b1010;
  assign LOG1 = A && B; //LOG1 == 0
  assign LOG2 = A || B;  //LOG2 == 1
  assign LOG3 = !A;      //LOG3 == 1
  assign BIT1 = A & B;   //BIT1 == 0000
  assign BIT2 = A | B;   //BIT2 == 1010
  assign RED1 = &A;      //RED1 == 0
  assign RED2 = |B;      //RED2 == 1
endmodule
```

Logical Operator (cont.)

✓ Shift Operators

- logical shift : << (left) , >> (right)
- arithmetic shift : <<< (left),>>> (right) – current synthesis tools support
  - a << n : a shifts left n bits , and fills in zero bits.
    b >> n : b shift right n bits , and fills in zero bits.
  - a <<< n : a shifts left n bits , and fills in zero bits.
    b >>> n : b shifts right n bits , and fills in signed bits.

<Example>

```
module SHI_OP(A, B, D, L_SHI_LEFT,L_SHI_RIGHT, A_SHI_LEFT, A_SHI_RIGHT, XZ_SHIFT);
  input [7:0] A;
  input signed [7:0] B;
  //input unsigned [7:0] C; <- Syntax Error
  input [7:0] D;
  output [7:0] L_SHI_LEFT,L_SHI_RIGHT, A_SHI_LEFT, A_SHI_RIGHT, XZ_SHIFT;
  //A = 8'b11010111;
  //B = 8'b11010111;
  //D = 8'b000x01zz;
  assign L_SHI_LEFT = A << 3; //L_SHI_LEFT == 10111000
  assign L_SHI_RIGHT = A >> 5; //L_SHI_RIGHT == 00000110
  assign A_SHI_LEFT = B <<< 3; //A_SHI_LEFT == 10111000
  assign A_SHI_RIGHT = B >>> 5; //B_SHI_RIGHT == 11111110
  assign XZ_SHIFT = D << 2; //XZ_SHIFT == 0x01zz00
endmodule
```

Logical Operator (cont.)

- ✓ **Conditional Operator**
  - <result> = <cond_expr> ? <true_expr> : <false_expr>
    - If <cond_expr> is false , <result> = <false_expr>
      If <cond_expr> is true , <result> = <true_expr>

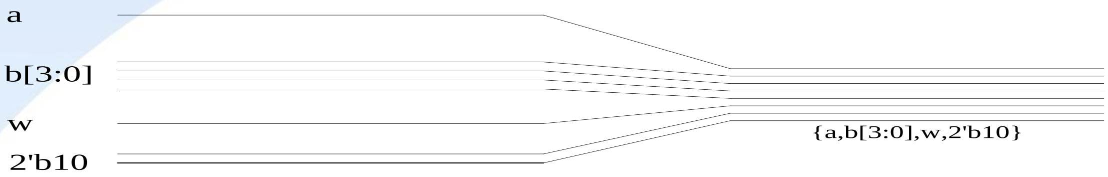
<Example>

```
module CON_OP(BUS, DRIVE_BUS, DATA);
  input [3:0] DATA;
  input       DRIVE_BUS;
  output[3:0] BUS;
  //DRIVE_BUS = 1’b1;
  //DATA = 4’b1010;
  assign BUS = DRIVE_BUS ? DATA:4'b0000;
  //DRIVE_BUS = 1 => BUSA = DATA
endmodule
```

Logical Operator (cont.)

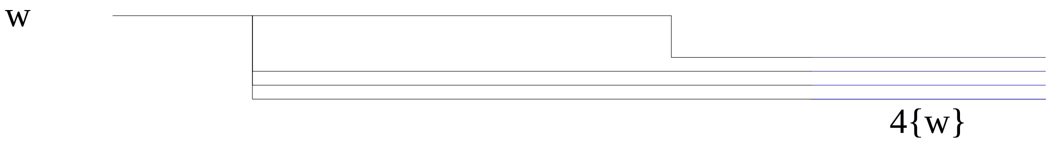
✓ Concatenations

- {a, b[3:0], w, 2’b10} = {a, b[3], b[2], b[1], b[0], w, 1’b1, 1’b0}

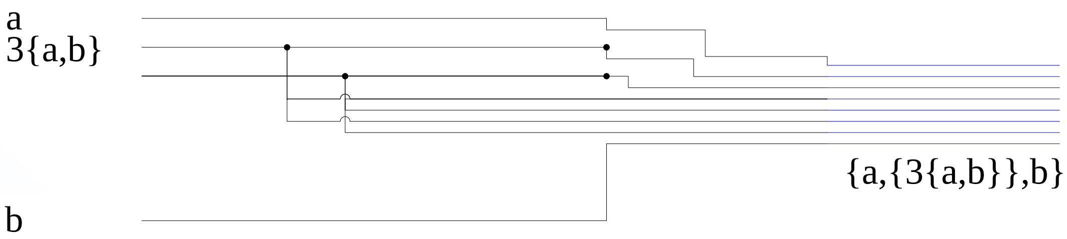


  - Unsized constant numbers are not allowed in concatenations

- {4{w}} = {w, w, w, w}



- {a,{3{a,b}},b} = {a, a, b, a, b, a, b, b}



Operand

✓ Net and register bit addressing

– declaration

- reg [MSB:LSB] vect;

– specify the single bit

- vect [index]
  - ◆ Index : constant , integer , register.

<Example>

```
module reg_op;
reg             [7:0]       vect1;
reg             [1:8]       vect2;                   //Note-1
integer                     index;
initial begin
 vect1 = 4;     //00000000000000000000000000000100 → vect1
```

vect1[7]	**vect1[6]**	**vect1[5]**	**vect1[4]**	**vect1[3]**	**vect1[2]**	**vect1[1]**	**vect1[0]**

```
 vect2 = 8'd7; //or 8'b00000111 , 8'h07
```

vect2[1]	**vect2[2]**	**vect2[3]**	**vect2[4]**	**vect2[5]**	**vect2[6]**	**vect2[7]**	**vect2[8]**

```
end
endmodule
```

» index = 2 → vect1[index] = 1 ; vect2[index] = 0

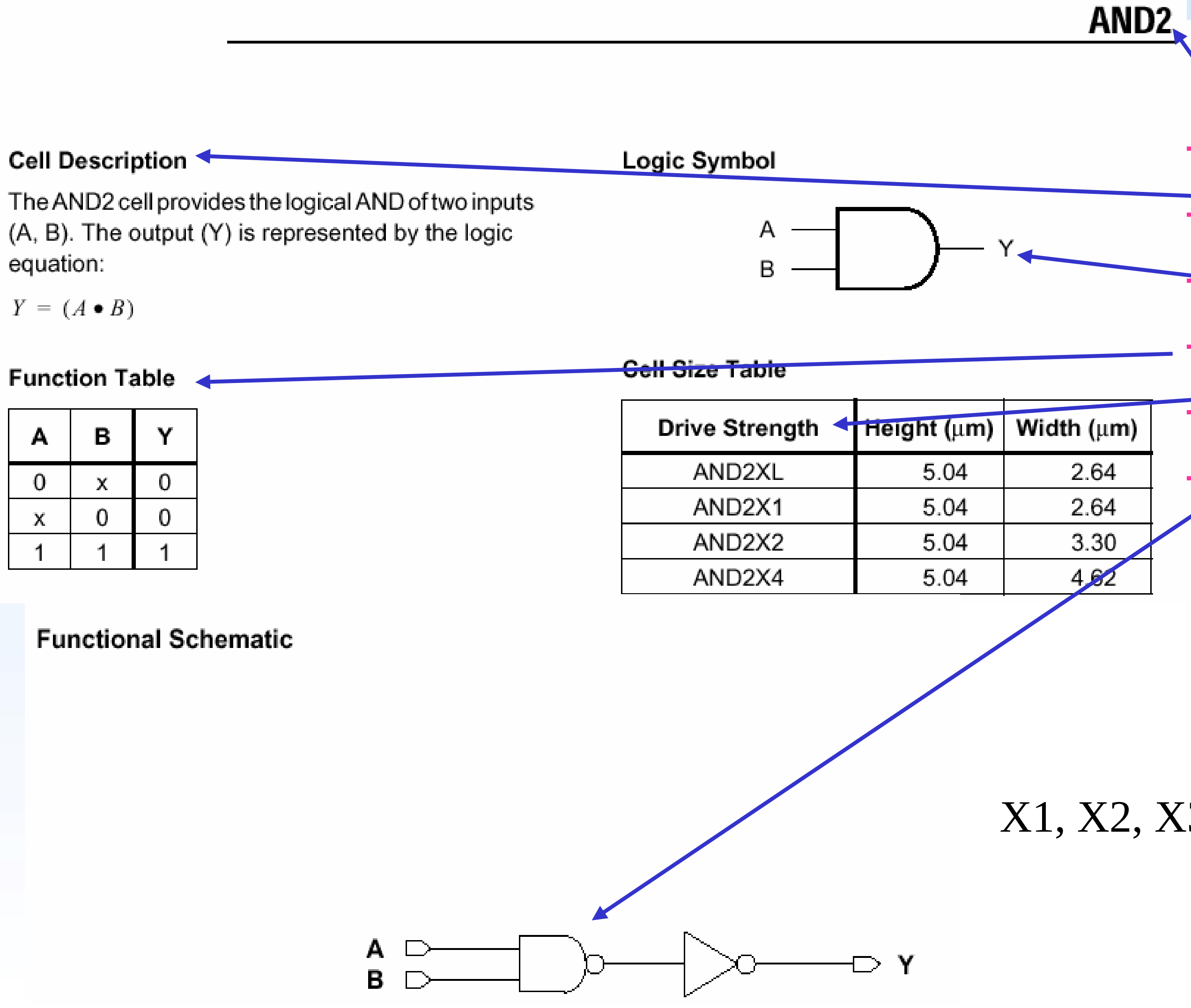
» index > 7 or index < 0 → vect1[index] = x

» vect1[3:0] = 0100 ; vect1[5:1] = 00010

» vect1[3'bx01] = x ; vect1[3'b0z1] = x;

» *Note-1 : Avoid using "MSB < LSB" (reg [2:4] vect;) or "MSB= LSB"(reg [1:1] vect) ,because there may result in problems for backend verifications.*

Cell Library: Data Sheet



AND2

Cell Description

The AND2 cell provides the logical AND of two inputs (A, B). The output (Y) is represented by the logic equation:

$Y = (A \bullet B)$

Logic Symbol

Function Table

A	B	Y
0	x	0
x	0	0
1	1	1

Cell Size Table

Drive Strength	Height (μm)	Width (μm)
AND2XL	5.04	2.64
AND2X1	5.04	2.64
AND2X2	5.04	3.30
AND2X4	5.04	4.62

Functional Schematic

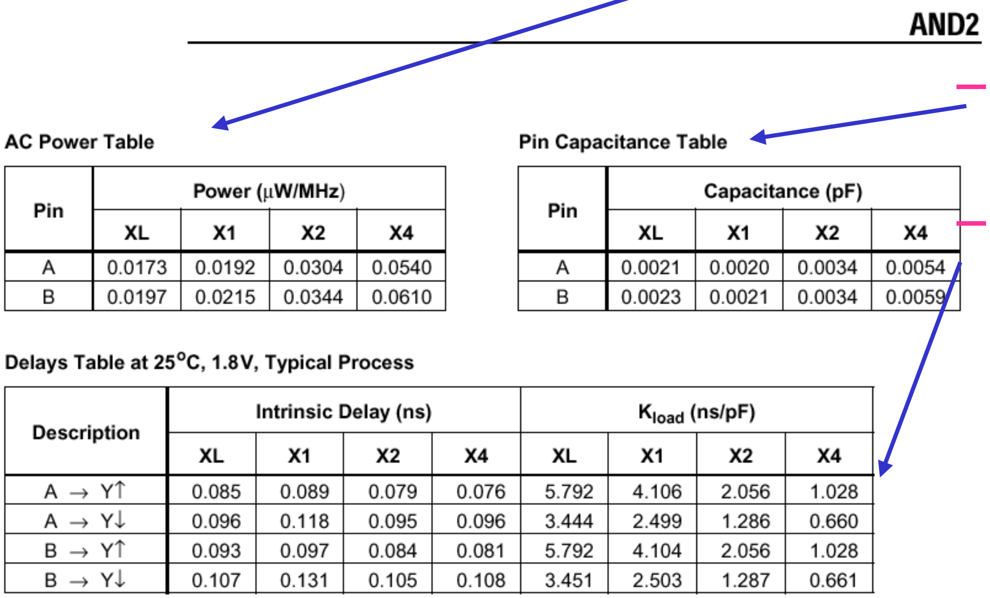
- Cell name
- Cell Description
- Logic symbol
- Function Table
- Cell Size
- Schematic

X1, X2, X3, X4: Driving Strength

Cell Library: Data Sheet (cont.)

- AC power
  - The power consumption when cell transition
- Pin Capacitance
  - Input pin’s capacitance
- Delays Model
  - Intrinsic delay :

    delay with no load
  - Delay = Intrinsic delay+K_{load} * C_{load}



AND2

AC Power Table

Pin	Power (μW/MHz)			
	XL	X1	X2	X4
A	0.0173	0.0192	0.0304	0.0540
B	0.0197	0.0215	0.0344	0.0610

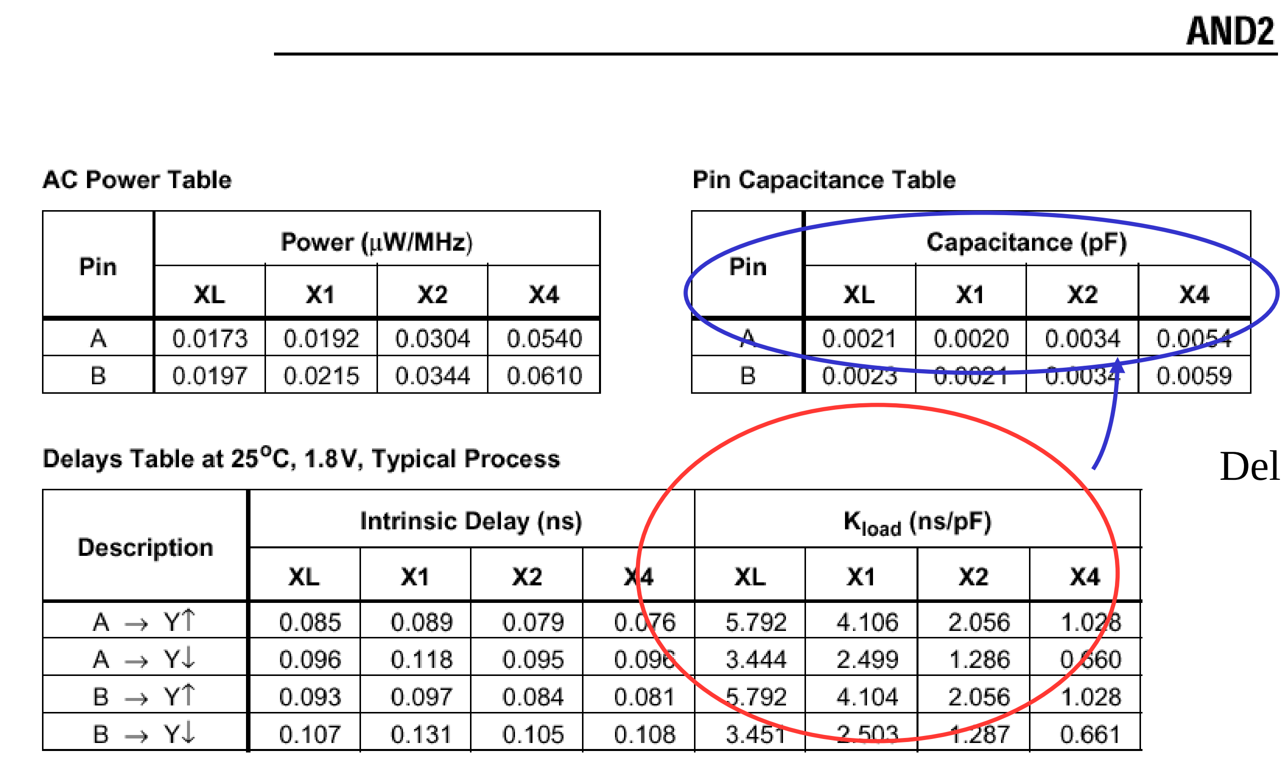
Pin Capacitance Table

Pin	Capacitance (pF)			
	XL	X1	X2	X4
A	0.0021	0.0020	0.0034	0.0054
B	0.0023	0.0021	0.0034	0.0059

Delays Table at 25°C, 1.8V, Typical Process

Description	Intrinsic Delay (ns)				K_{load} (ns/pF)			
	XL	X1	X2	X4	XL	X1	X2	X4
A → Y↑	0.085	0.089	0.079	0.076	5.792	4.106	2.056	1.028
A → Y↓	0.096	0.118	0.095	0.096	3.444	2.499	1.286	0.660
B → Y↑	0.093	0.097	0.084	0.081	5.792	4.104	2.056	1.028
B → Y↓	0.107	0.131	0.105	0.108	3.451	2.503	1.287	0.661

Cell Library: Delay Calculation



AND2

AC Power Table

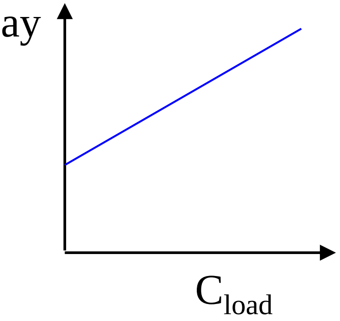
Pin	Power (µW/MHz)			
	XL	X1	X2	X4
A	0.0173	0.0192	0.0304	0.0540
B	0.0197	0.0215	0.0344	0.0610

Pin Capacitance Table

Pin	Capacitance (pF)			
	XL	X1	X2	X4
A	0.0021	0.0020	0.0034	0.0054
B	0.0023	0.0021	0.0034	0.0059

Delays Table at 25°C, 1.8V, Typical Process

Description	Intrinsic Delay (ns)				K_{load} (ns/pF)			
	XL	X1	X2	X4	XL	X1	X2	X4
A → Y↑	0.085	0.089	0.079	0.076	5.792	4.106	2.056	1.028
A → Y↓	0.096	0.118	0.095	0.096	3.444	2.499	1.286	0.660
B → Y↑	0.093	0.097	0.084	0.081	5.792	4.104	2.056	1.028
B → Y↓	0.107	0.131	0.105	0.108	3.451	2.503	1.287	0.661



- Delay = Intrinsic delay + K_{load} * C_{load}
- C_{load} is the ***total*** input capacitance of the next stage!!

Simulation Environment

✓ Dump a FSDB file for debug

- General debussy waveform generator
  - $fsdbDumpfile(“file_name.fsdb”); ➔ Dump an fsdb file
  - $fsdbDumpvars; ➔ Dump all values
- Other debussy waveform generator
  - $fsdbSwitchDumpfile(“file_name.fsdb”);

    ➔ close the previous fsdb file and create a new one and open it
  - $fsdbDumpflush (“file_name.fsdb”);

    ➔ not wait the end of simulation and Dump an fsdb file
  - $fsdbDumpMem(memory_name, begin_cell, size);

    ➔ the memory array is stored in an fsdb file
  - $fsdbDumpon; $fsdbDumpoff;

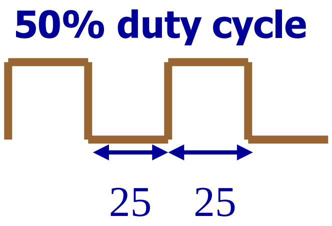
    ➔ just Dump and not Dump

Simulation Environment (cont.)

- ✓ **Clock Generation**

  (in PATTERN.v)

  ```
  initial clk = 0;
  always #25 clk = ~clk;
  ```



- ✓ **Display simulation result**
  - Texture format output
    - $display("%t, clk=%b in=%d out=%d \n", $time clk, in, out);
    - $monitor($time,"clk=%b out=%b\n",clk,out);

    format (display): %d (decimal), %b (binary), %h (hexadecimal), %o (octal), %c(ASCII),%s (strings), %v (strength), %m(hierarchical name), %t (time)

    $time : current time

    \n: new line, \t: tab, \\: backslash, \": double quote

Simulation Environment (cont.)

✓ Random number generation

- $random
- $random(seed)

e.g. in1=$random;
in2=$random(37);

✓ Control command

- $finish;

//finish the simulation

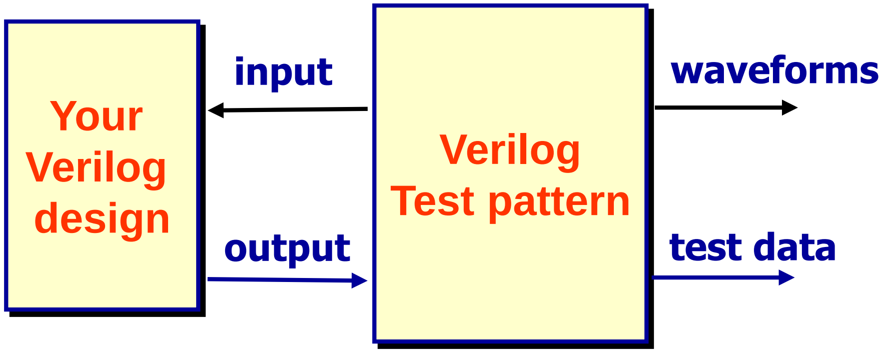
- $stop;

//stop the simulation

✓ Simulation command

– Verilog compile
  - verilog test_file_name.v
  - ncverilog test_file_name.v

– Debussy waveform generation
  - nWave &



Your Verilog design

input

output

Verilog Test pattern

waveforms

test data

– Stop the simulation and continue the simulation
  - Ctrl+z ➔Suspend the simulation at anytime you want.(not terminate yet!)
  - . ➔Continue if you stop your simulation by $stop command
  - jobs ➔Here you can see the jobs which are processing with a index on the left [JOB_INDEX]
  - kill ➔Use the command "kill %JOB_INDEX to terminate the job

Gate-level Simulation

✓ Post synthesis timing simulation

- The post synthesis design must be simulated with estimated delays from synthesis. A SDF(standard delay format) can be generated from design compiler for this purpose. Use the following command to generate the SDF file.

✓ Modify your test file

- $sdf_annotate(“the_SDF_file_name¨, the_instance_name);
- Example:

  $sdf_annotate(“ADDER.sdf”, I_ADDER);