

硬體描述語言

Chapter 02



Simulation, Logic System, Data Types, and Operators

Ren-Der Chen (陳仁德)

Department of Computer Science and
Information Engineering

National Changhua University of Education

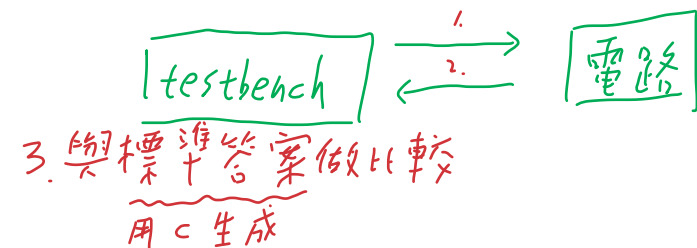
E-mail: rdchen@cc.ncue.edu.tw

Fall, 2024

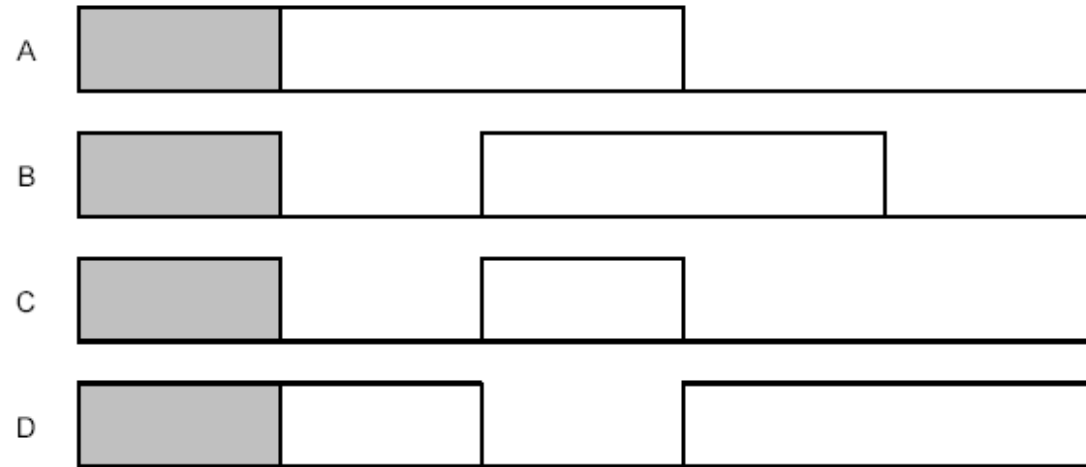
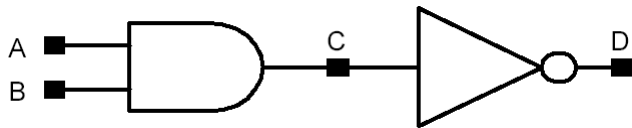
事件觸發

Event-Driven Simulation

- **Design verification** is usually achieved through digital simulation and/or formal methods.
- A design must function correctly for **all cases** of the signal patterns that could be applied to its inputs.
- Logic simulation relies on a user-defined **testbench**, and is usually done on an **event-driven** simulator.
- An **event** is said to occur when a signal undergoes **a change in value**.

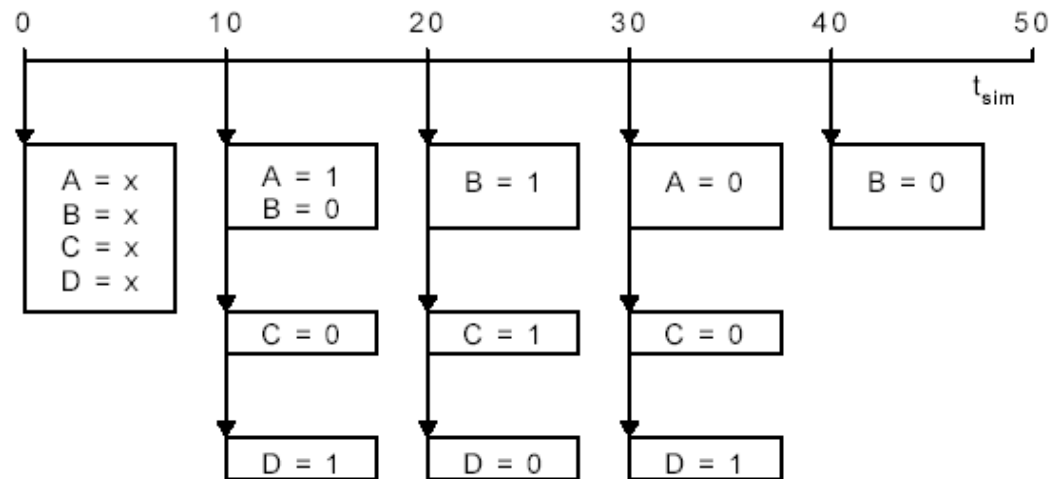


Zero-Delay Simulation

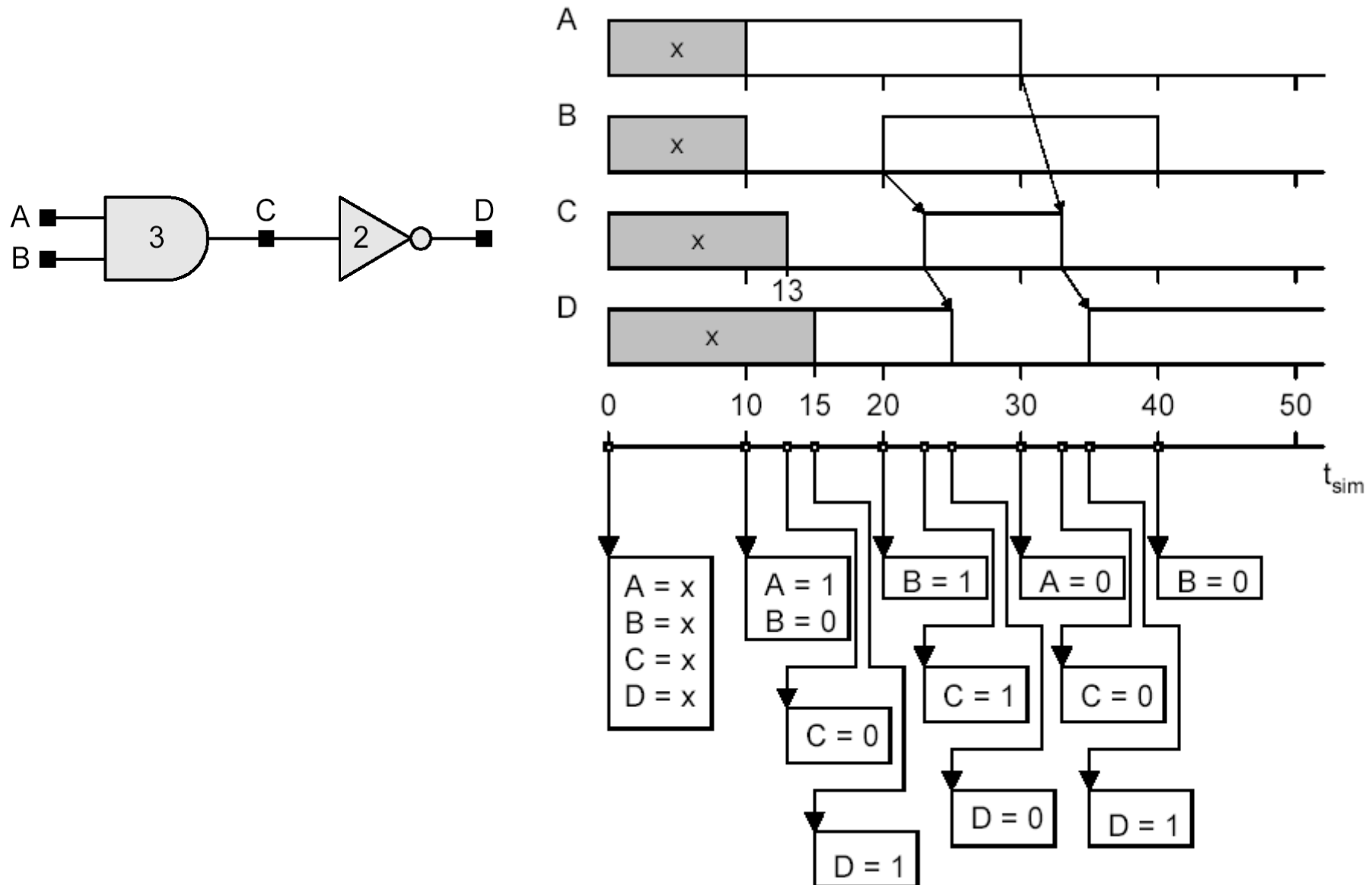


Event-time list

Signal-change list

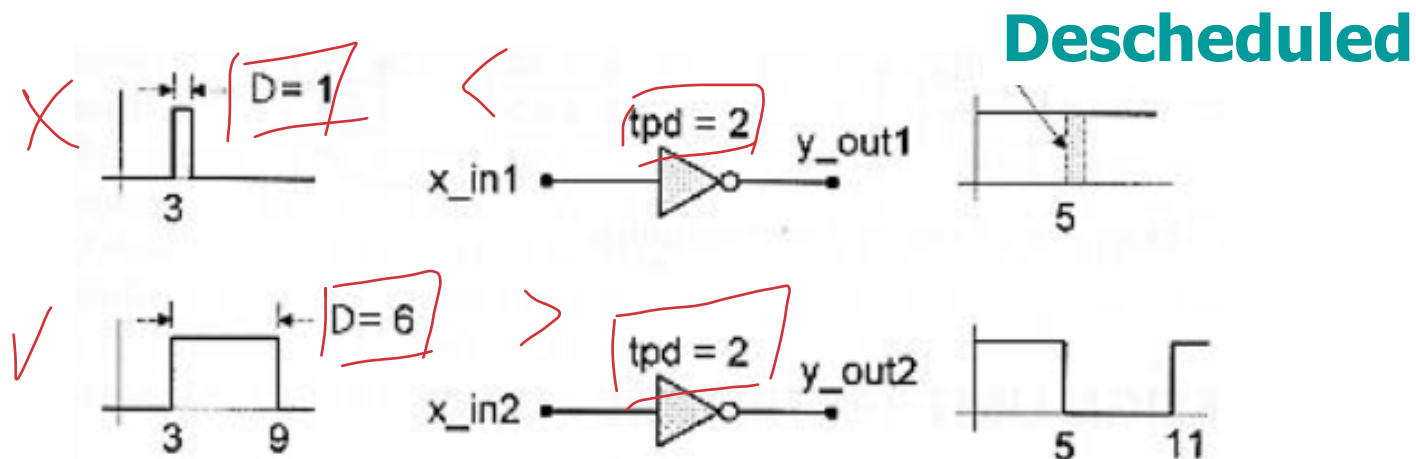


Non-Zero-Delay Simulation

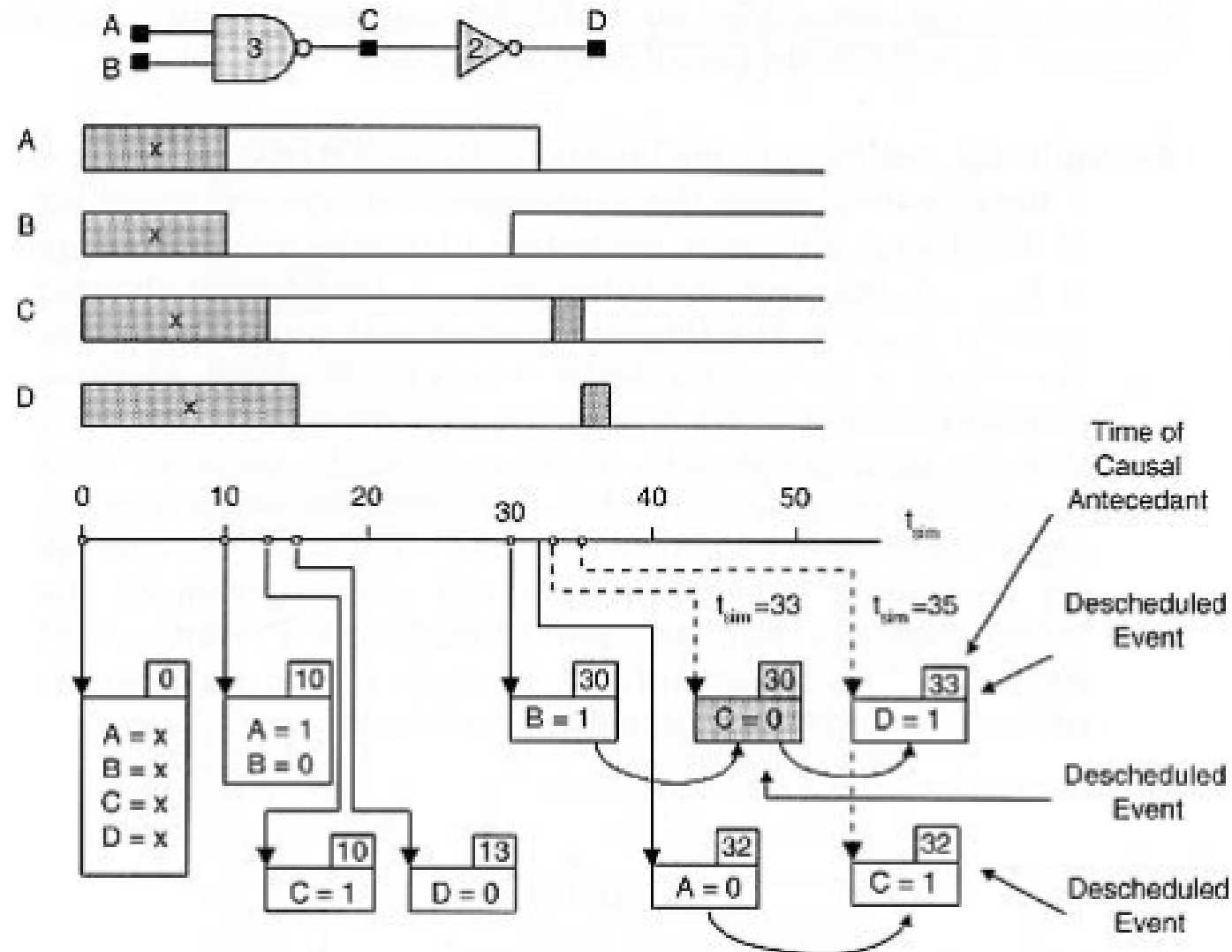


Inertial Delay

- The **propagation delay** of primitive gates obeys an “**inertial**” delay model.
- Verilog uses the propagation delay as **the minimum width of an input pulse that could affect the output**.
- Input pulse whose duration is shorter than the inertial delay will be suppressed.

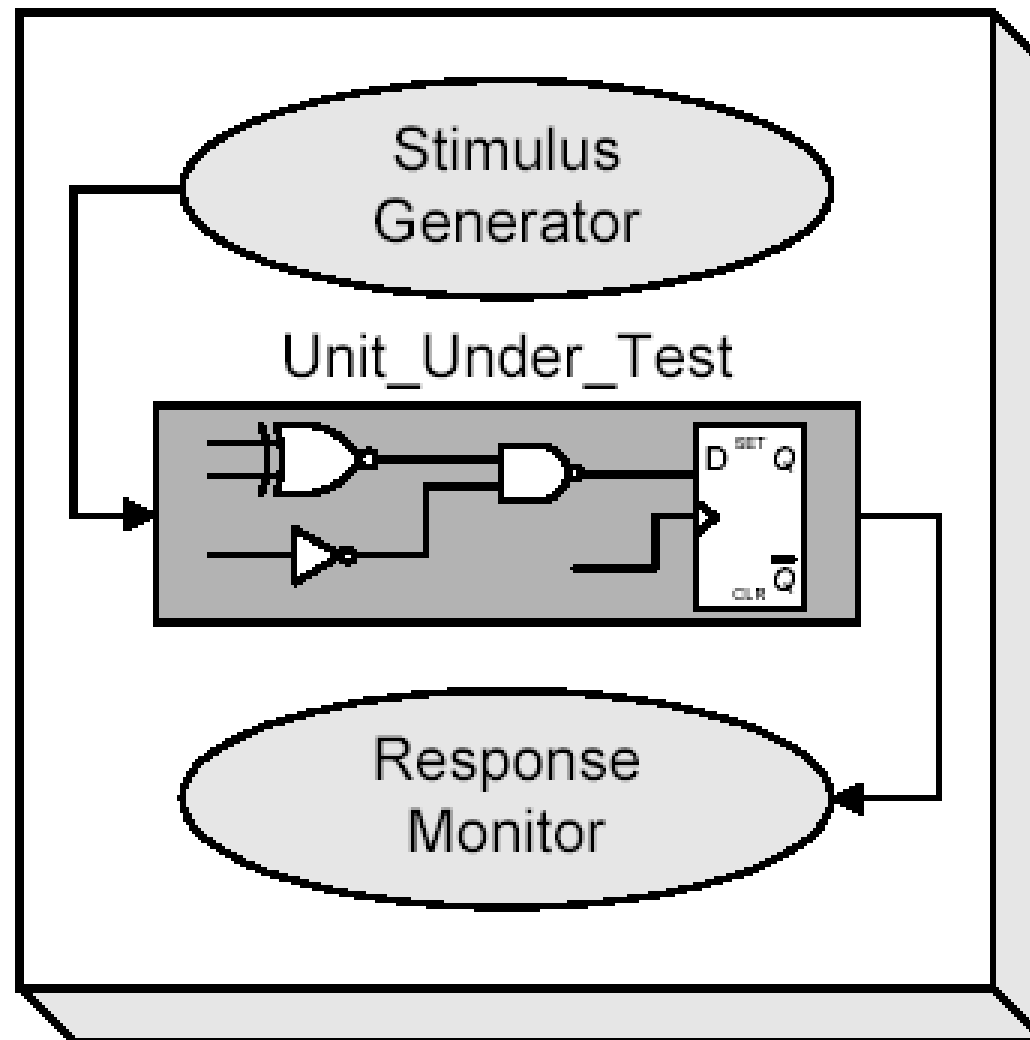


Event De-Scheduling



HDL-Based Simulation Methodology

Design_Unit_Test_Bench



NAND Latch

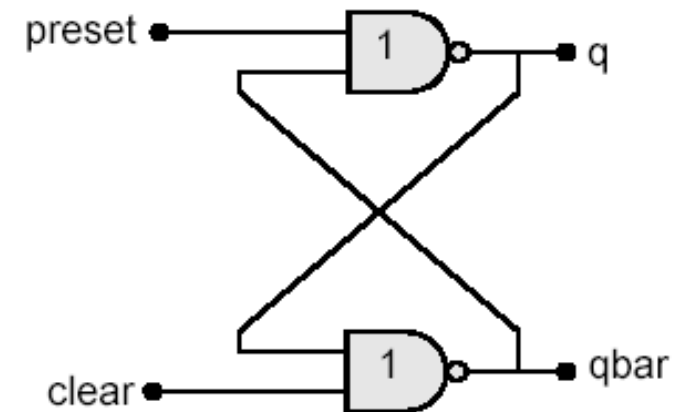
```
module nand_Latch (preset, clear, q, qbar);
```

```
    input          preset, clear;
```

```
    output         q, qbar;
```

```
    nand #1      G1(q, preset, qbar), G2(qbar, clear, q);
```

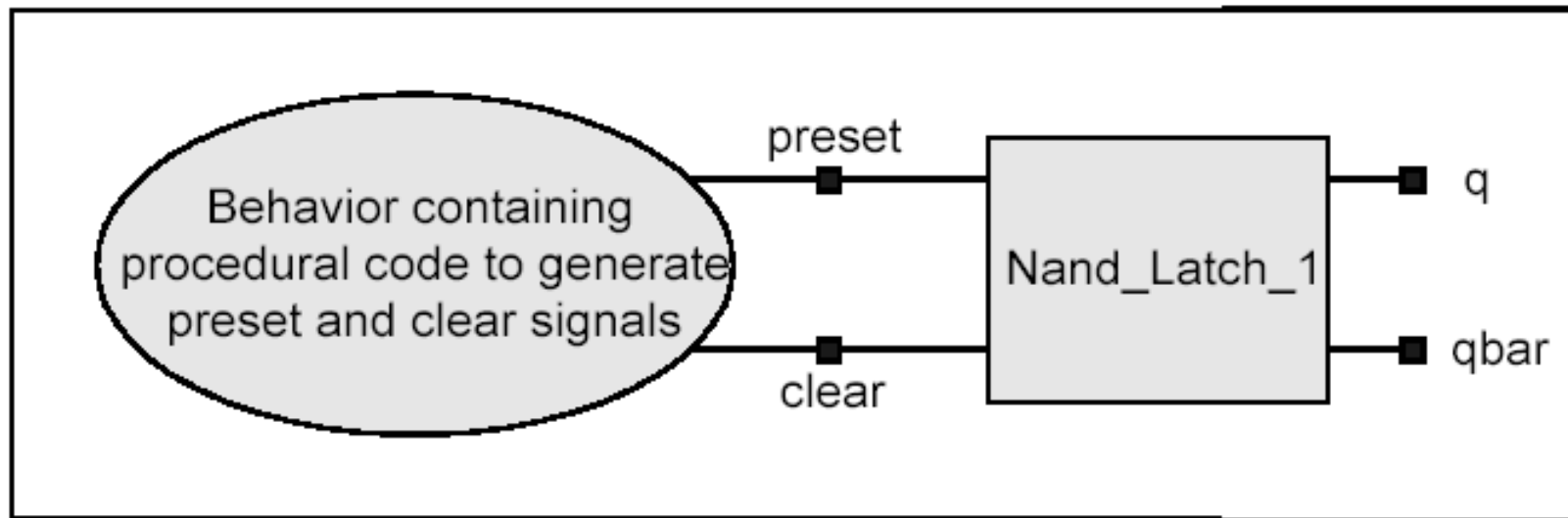
```
endmodule
```



實際驗證

Preset	Clear	q_n	q_{n+1}	$qbar_n$	$qbar_{n+1}$
0	0	-	1	-	1
0	1	-	1	-	0
1	0	-	0	-	1
1	1	-	q_n	-	$qbar_n$

Design Unit Test Bench (1/2)

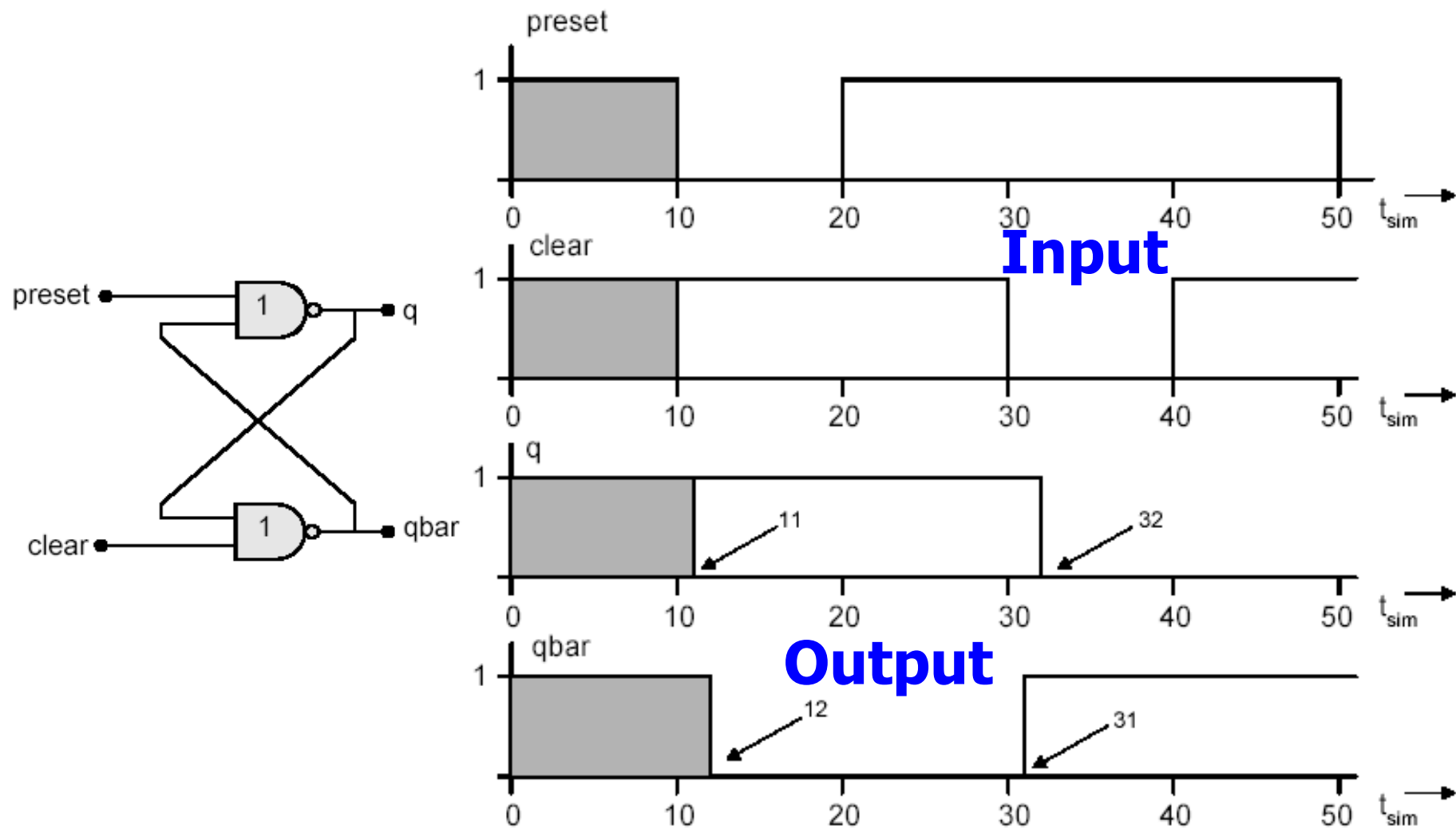


Design Unit Test Bench (2/2)

```
module nand_latch_test;  
  reg      preset, clear;  
  wire     q, qbar;  
  
  nand_Latch M1 (preset, clear, q, qbar);  
  
  initial  
    $monitor("%0t  preset = %b  clear = %b  q = %b  qbar = %b",  
              $time, preset, clear, q, qbar);  
  
  initial  
    begin  
      #10 preset = 0; clear = 1;  
      #10 preset = 1;  
      #10 clear  = 0;  
      #10 clear  = 1;  
      #10 preset = 0;  
    end  
endmodule
```

```
initial  
  #60 $finish;  
endmodule
```

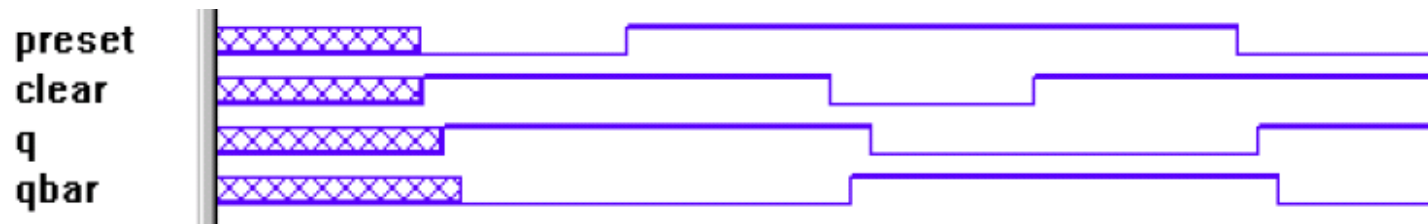
Expected Result



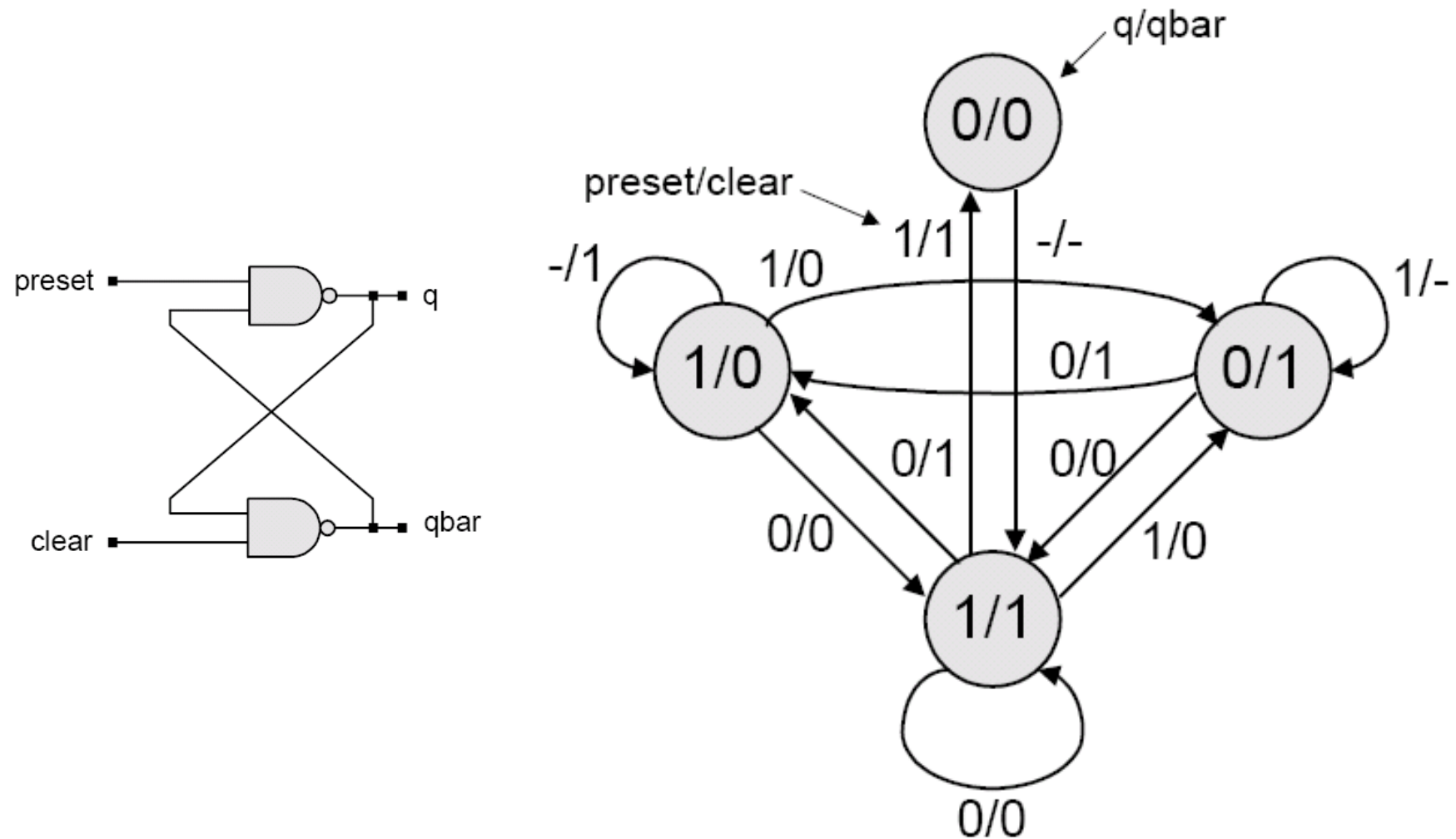
Simulation Results

```
# 0  preset = x  clear = x  q = x  qbar = x
# 10 preset = 0  clear = 1  q = x  qbar = x
# 11 preset = 0  clear = 1  q = 1  qbar = x
# 12 preset = 0  clear = 1  q = 1  qbar = 0
# 20 preset = 1  clear = 1  q = 1  qbar = 0
# 30 preset = 1  clear = 0  q = 1  qbar = 0
# 31 preset = 1  clear = 0  q = 1  qbar = 1
# 32 preset = 1  clear = 0  q = 0  qbar = 1
# 40 preset = 1  clear = 1  q = 0  qbar = 1
# 50 preset = 0  clear = 1  q = 0  qbar = 1
# 51 preset = 0  clear = 1  q = 1  qbar = 1
# 52 preset = 0  clear = 1  q = 1  qbar = 0
```

Simulation waveform

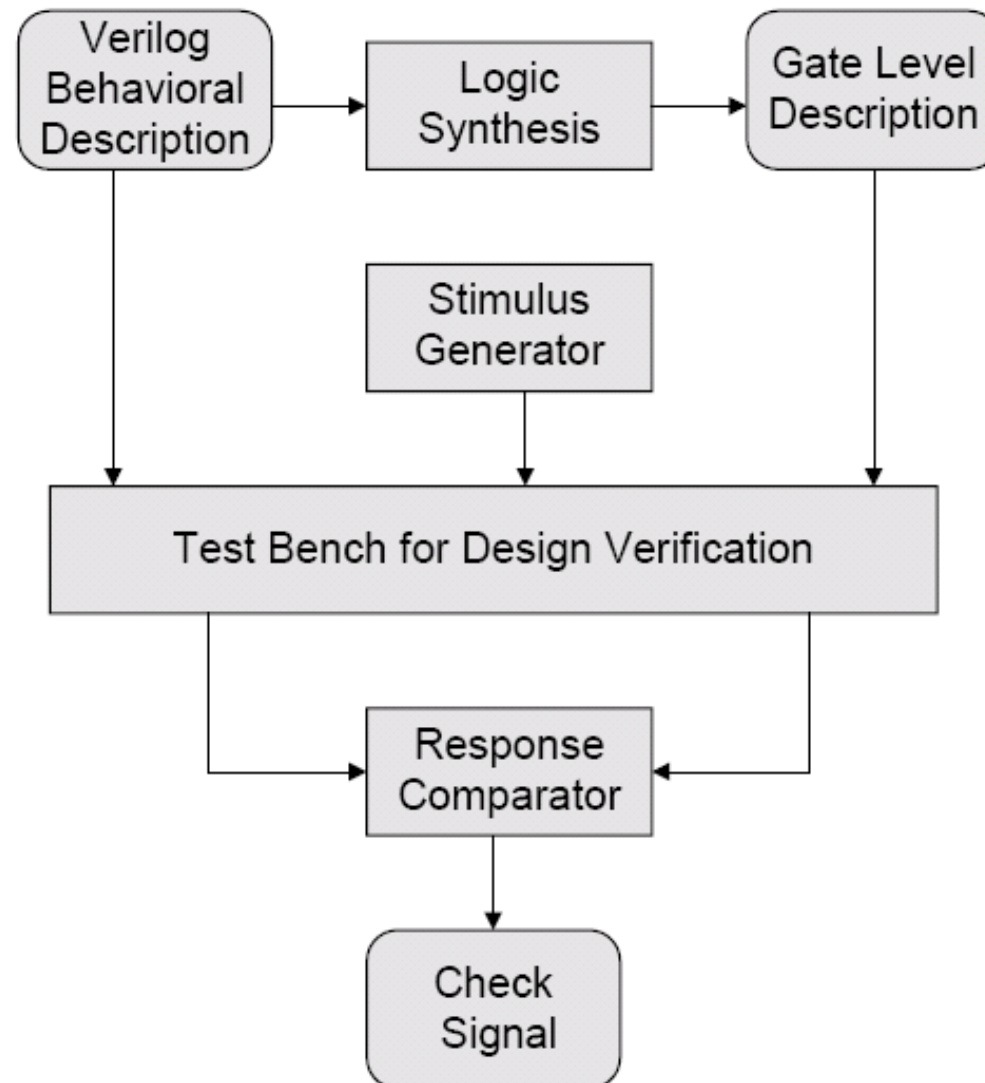


Model Verification



State diagram

Test Methodology

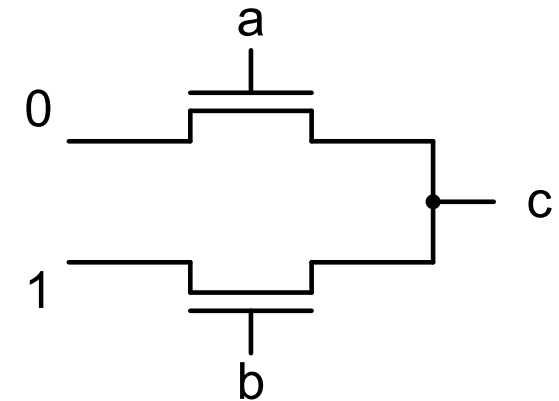


Variables

- Verilog uses **variables** to represent values of **signals** in a physical circuit.
 - **Variables**: value may change during simulation
 - **Constants**: value is fixed during simulation
- Two kinds of variables
 - **Nets**: connectivity variables
 - **Registers**: data storage variables

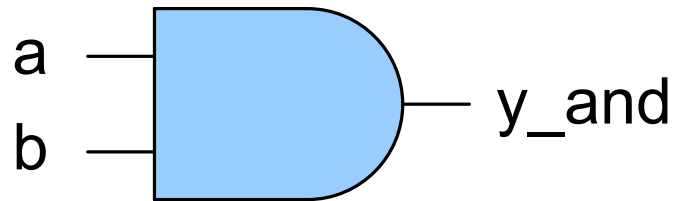
Logic Value Set

- {0, 1} is not enough
 - If $a = b = 1$, what is the value of c ?
 - If $a = b = 0$, what is the value of c ?

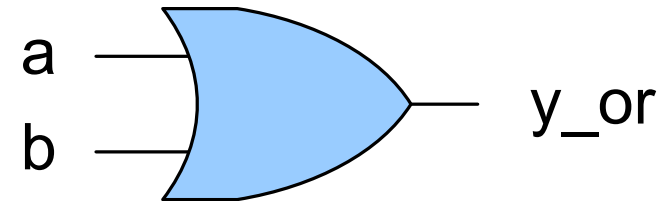


- In Verilog
 - 0 and 1: logical 0 and logical 1
 - x: unknown 無法判斷
 - z: high impedance 斷路, 高阻抗
- Example
 - $a = b = 1$, $c = \text{x}$ 皆通
 - $a = b = 0$, $c = \text{z}$ 皆不通

Four-Valued Relations



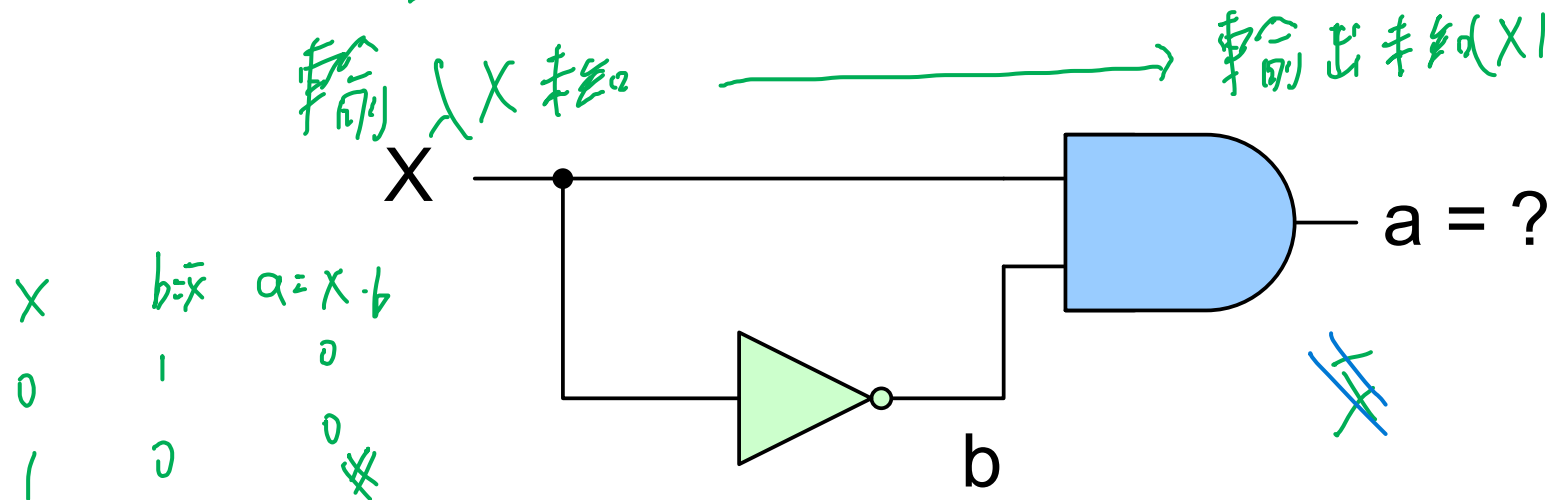
AND		<i>a</i> 0	1	x	z
<i>b</i>	0	0	0	0	0
	1	0	1	x	x
	x	0	x	x	x
	z	0	x	x	x



OR		<i>a</i> 0	1	x	z
<i>b</i>	0	0	1	x	x
	1	1	1	1	1
	x	x	1	x	x
	z	x	1	x	x

X / X-bar Problem

- $b = \bar{X}$
- In Verilog simulation, there is no such thing as “X-bar”
- $b = X$
- $a = X$ (while it should be 0)



Data Types (1/3)

Nets (Connectivity)		Registers (Storage)
wire	supply1	reg
tri	supply0	integer
wand	tri1	real
wor	tri0	time
triand	trireg	realtime
trior		

Data Types (2/3)

- Data types 連接
 - ! ● Structural connectivity: nets
 - Data storage: **reg**
 - Procedural computation: **integer**, **real**, **time**, and **realtime**
- All **register** variables are static – their values exist throughout simulation – subject to **assignments** made under program flow.
- All members of **nets** establish **connectivity** within the structure of a design.

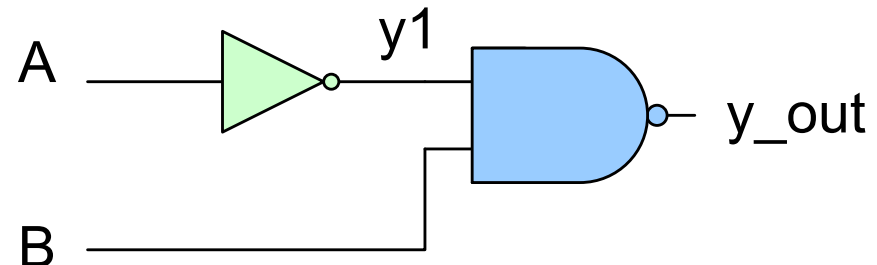
Data Types (3/3)

- Objects of a given type are assigned value
 - Explicitly – by **behavioral statements**
 - Implicitly – driven by a **gate**
- A net is assigned value
 - Explicitly – by a continuous assignment statement (**assign**) or **force ... release** procedural continuous statement
 - Implicitly – as the output terminal of a primitive or connected to an output port of a module
- *A register variable is assigned value only within a behavior (explicitly).*

Net Data Types

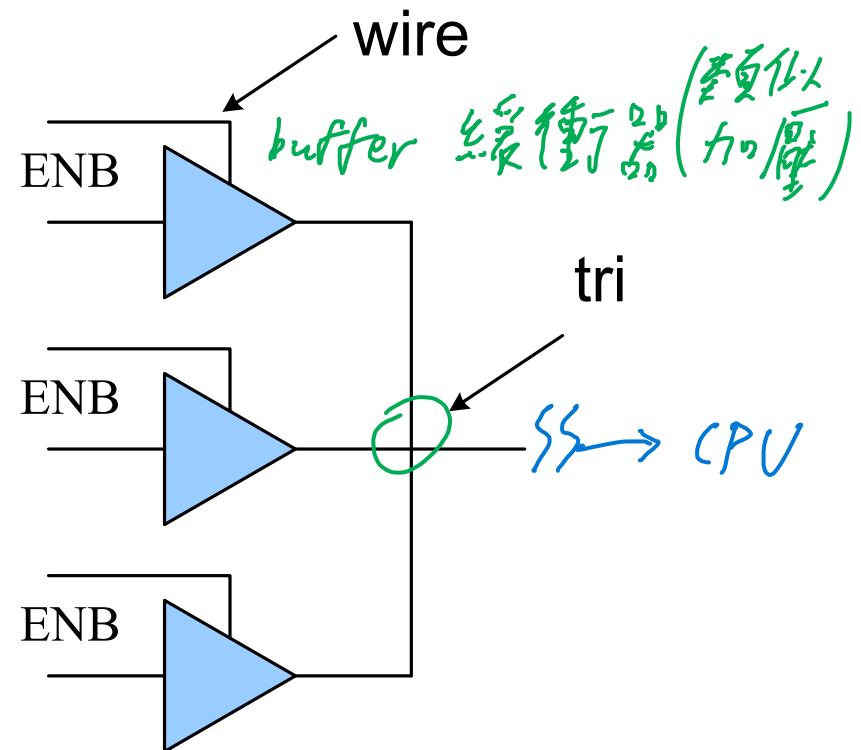
- A **net** is used to create **connectivity** within a module.
- The logic value of a net is determined by its driver.

```
module connect_y1 (y_out, A, B);  
    output y_out;  
    input  A, B;  
    wire   y1;  
    not    (y1, A);  
    nand   (y_out, y1, B);  
endmodule
```



Semantics of Verilog Nets (1/2)

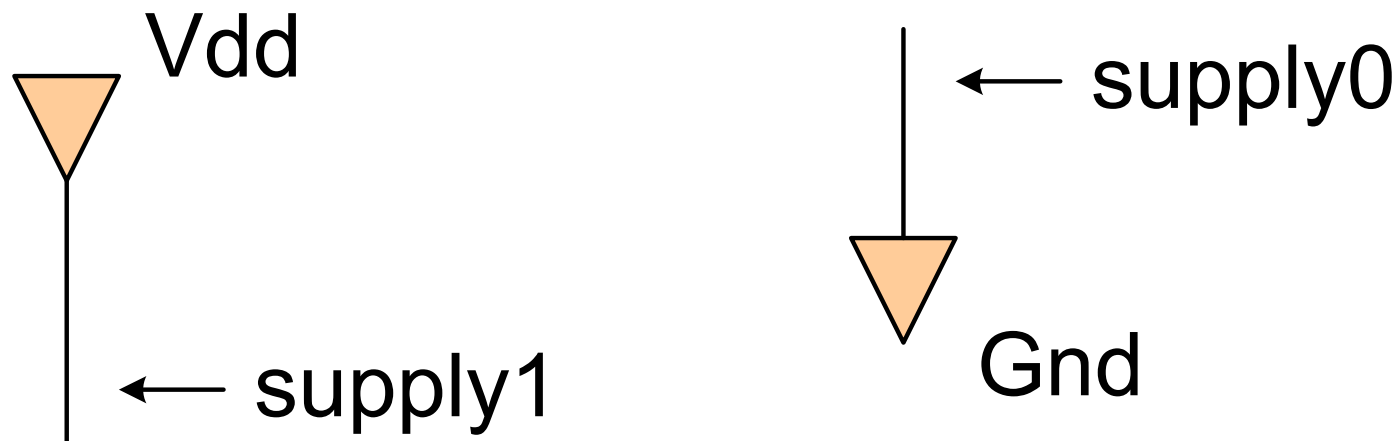
- **wire** 兩種狀態 : 0, 1
 - Establishes **connectivity**, with no logical behavior or functionality implied.
- **tri** 三種狀態 : 0, 1, Z
 - Has the same functionality as **wire**, but it will be tri-stated in hardware.



tri-state buffer

Semantics of Verilog Nets (2/2)

- **supply0** *接地*
 - A global net connected to ground.
- **supply1** *高電位*
 - A global net connected to power supply.



Declarations

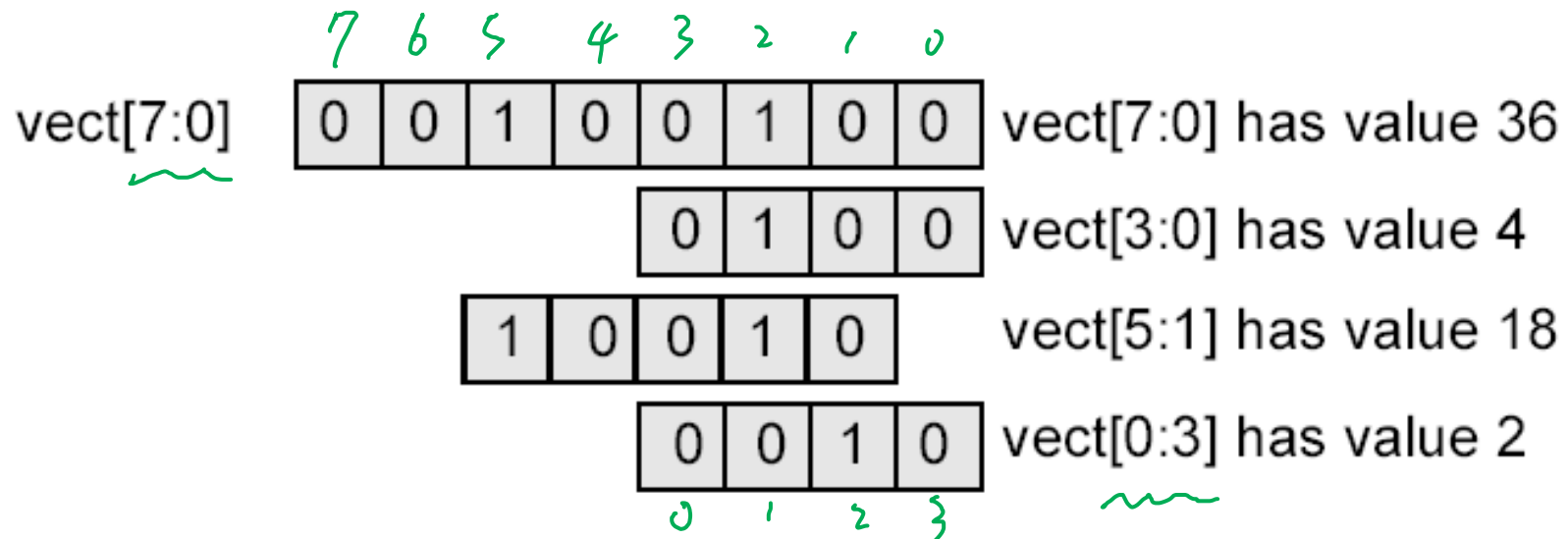
```
wire [7:0] data_bus;           // data_bus[7] is the MSB 2進制 最大有效位元
wire [0:3] control_bus;      // control_bus[0] is the MSB
wire  y1, z_5;
wire  A = B + C, D = E + F;
// Multiple declarations and continuous assignments
```

References

`data_bus[5]` // Access to bit 5 (bit-select)

`data_bus[3:5]` // Access to bits 3 to 5 (part-select)

`data_bus[k+2]` // Access bit k+2 (bit-select)



Undeclared Nets

- Nets that are not explicitly declared will default implicitly to **wire**.

```
module nand_latch (preset, clear, q, qbar);  
    input          preset, clear;  
    output         q, qbar;  
    wire           preset, clear, q, qbar; // Optional  
                                           可省去宣告  
  
    nand # 1  
        g1 (q, preset, qbar),  
        g2 (qbar, clear, q);  
endmodule
```

Example

```
module assign_y1 (A, B, y_out);
```

```
    input      A, B;
```

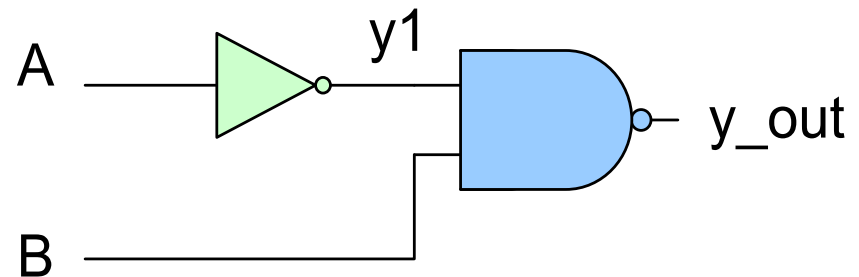
```
    output     y_out;
```

```
    wire       y1;
```

```
    assign     y1 = ~A;
```

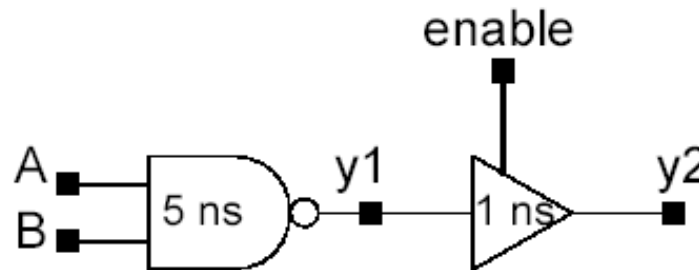
```
    assign     y_out = ~ (y1 & B);
```

```
endmodule
```



1. An event on the RHS causes the RHS to be evaluated.
2. A change on the RHS of a continuous assignment causes an event to be scheduled for the LHS.
3. The LHS of a continuous assignment must be a **net**.

Example



```

module ...
    ...
    nand #5 (y1, A, B);
    bufif1 #1 (y2, enable, y1);
endmodule

```

Handwritten green text: delay

```

module ...
    ...
    assign #5 y1 = A ~& B;
    assign #1 y2 = enable ? y1 : 1'bz;
endmodule

```

Handwritten red text: 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2

1. A = 1 and B = 1 cause y1 = 0 after 5 ns
2. enable = 0 causes y2 = z after 1 ns

Data Type - reg (1/2)

- A **reg** models the feature of hardware that allows a logic value to be stored in a flip-flop or latch.
- A **reg** is an **abstraction** of a hardware storage element, but *it does not correspond directly to physical memory*.
- A **reg** variable is treated as an unsigned value.
- Example
 - **reg** A_Bit_Register;
 - **reg** [31:0] A_Word;

Data Type - reg (2/2)

- A **reg** may be assigned value only within a **procedural statement**.
- A **reg** may never be the output of a pre-defined primitive gate or the target of a continuous assignment.
- Example
 - The indicated values of A and B are stored in **memory** immediately when the statements assigning values to A and B execute.

```
reg A, B;
```

```
initial
```

最初行に記す

```
begin
```

```
    A = 0;
```

```
    B = 1;
```

```
end
```

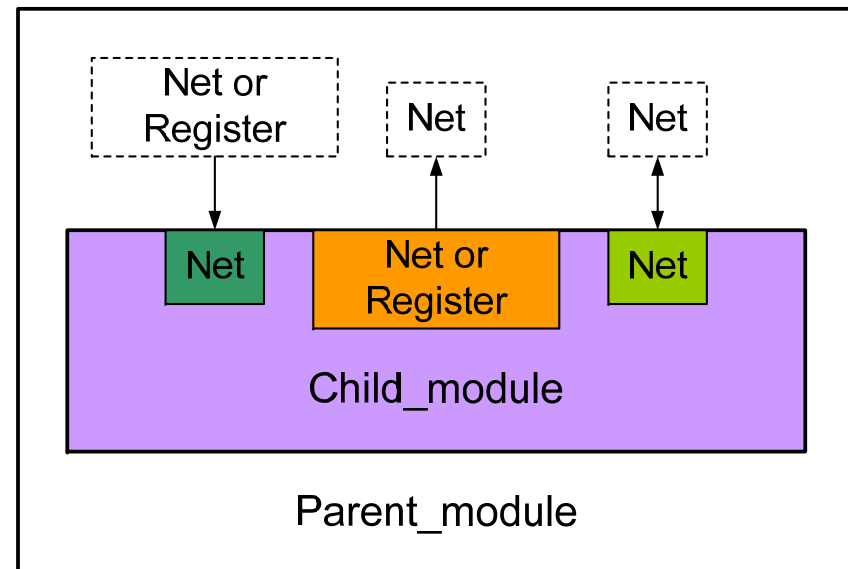
Register Variable

- In simulation, all register variables have an initial value “X”.
- This value may be changed by subsequent execution of a **procedural statement**.
- Verilog has no mechanism for undeclared register variables.
- All references to an identifier that has not been declared are assumed to be references to a **net**.
- Error occurs if such a reference is made within a behavior.

Passing Variables Through Ports

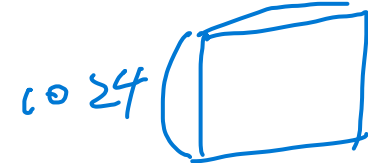
- The value of a register variable may be changed only within a **behavior**.
- The external environment may not alter a register variable.

Variable	Port mode		
	Input	Output	Inout
Net	Yes	Yes	Yes
Register	No	Yes	No



Two-Dimensional Arrays (Memories)

- No true two-dimensional arrays with access to each array element.
- But an extension of the declaration of a register variable to provide a “memory” (multiple addressable cells of the same word size).



- Ex: a memory having 1,024 words (32-bit)

- `reg [31:0] cache_memory [0:1023]`

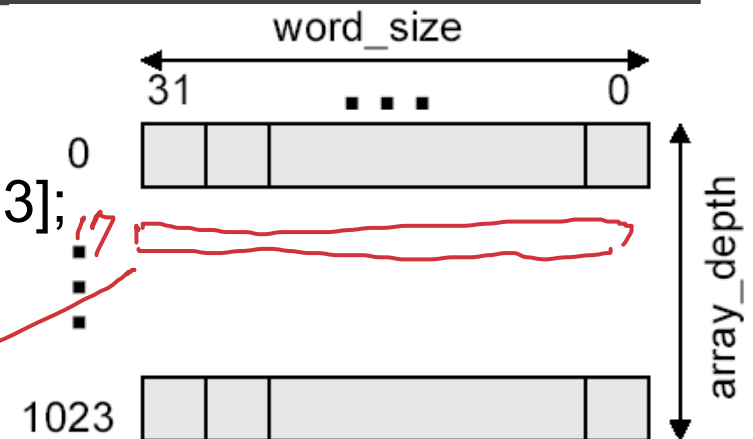
二维阵列 [1024][32]

- Rules for implementing memories
 - Bit-select and part-select are **not valid** with memories.
 - Reference may be made to only a **word** of memory.

Rules for Implementing Memories

■ Example

- reg [31:0] cache_memory [0:1,023];
- reg [31:0] a_word_register;
- reg [7:0] instr_register;
- a_word_register = cache_memory [17];
- instr_register [k] = a_word_register [k+4];



■ Improper access *Error*

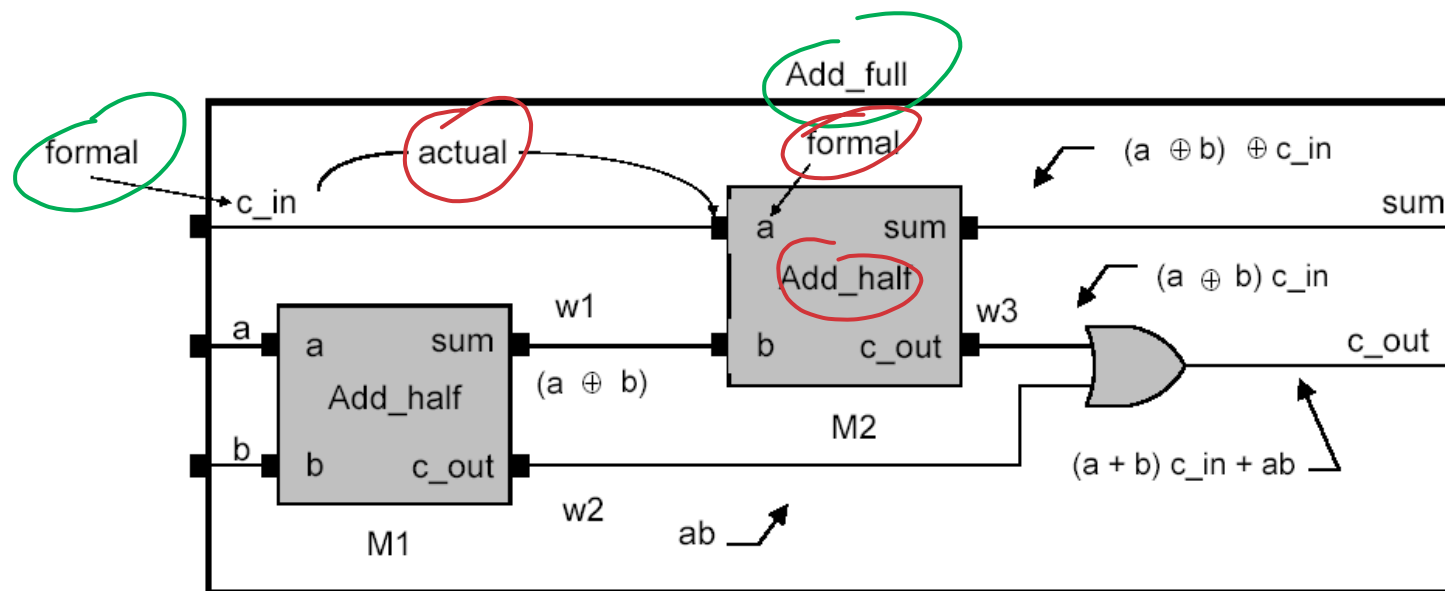
- cache_memory [0:4]; *不能一次拿多列 (row)*
- instr_register = a_word_register [4:15];
8 bit 12 bit

Data Type - integer

- Supports numeric computation in procedural code.
- Stored as a signed value in 2's complement format.
- Example
 - **integer** A1, K, Size_of_Memory;
 - **integer** Array_of_Ints [1:100];

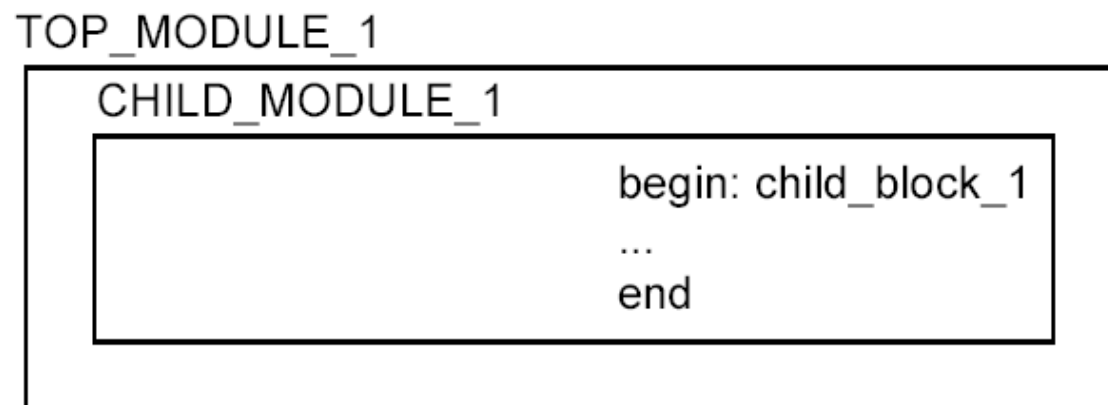
Scope of a Variable

- The **scope** of a variable is the module or named procedural (**begin ... end**) block in which it is declared.
- When a model is evaluated, the **actual** value passed through a port is substituted for the associated **formal** parameter in the model.



Variable References and Hierarchical De-Referencing

- A **variable** is referenced directly by its identifier within the scope in which it is declared.
- Hierarchical de-referencing is also supported by a variable's **hierarchical path name**.
- In a testbench, hierarchical de-referencing is used to monitor signals.



Example (1/2)

```
module test_add_rca_4 ();  
    reg [3:0] a, b;  
    reg c_in;  
    wire c_out;  
    wire [3:0] sum;  
    ...
```

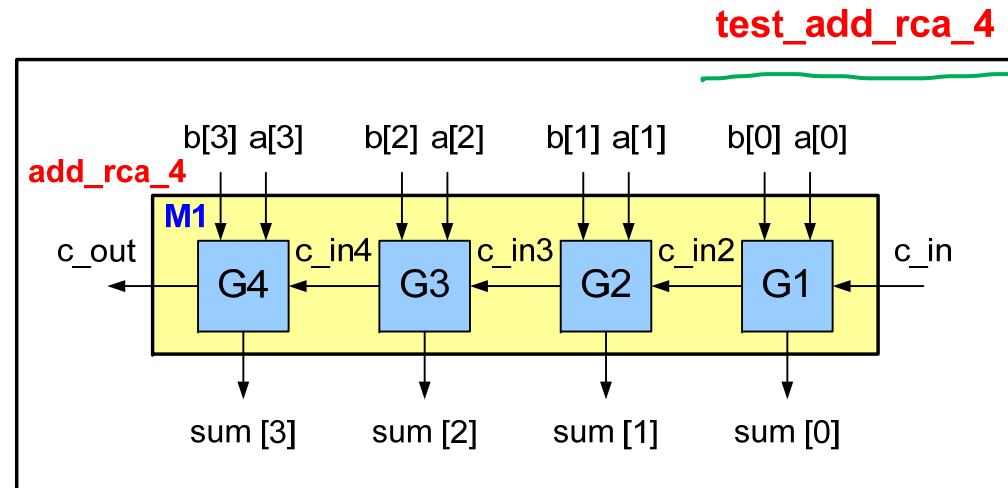
```
initial begin
```

```
    $monitor ("c_out = %b c_in4 = %b c_in3 = %b c_in2 = %b c_in =  
              %b", c_out, M1.c_in4, M1.c_in3, M1.c_in2, c_in);
```

```
end
```

```
    add_rca_4 M1 (a, b, c_in, c_out, sum);
```

```
endmodule
```



系統子數
右資料輸入
就 print

拿取電路內部變數

Example (2/2)

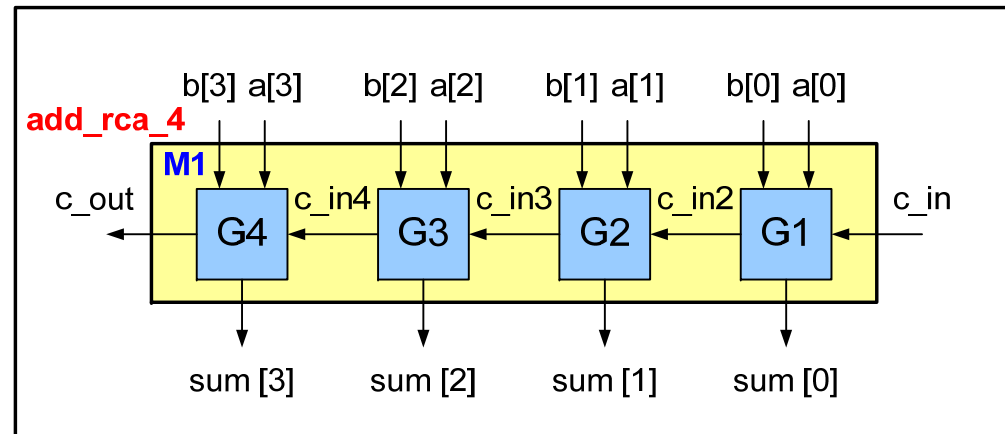
```
module add_rca_4 (a, b, c_in, c_out, sum);
```

```
  input          [3:0]    a, b;
```

```
  input          c_in;
```

```
  output         c_out;
```

```
  output         [3:0]    sum;
```



```
  wire          c_in4, c_in3, c_in2;
```

```
  add_full      G1 (a[0], b[0], c_in,  c_in2, sum[0]);
```

```
  add_full      G2 (a[1], b[1], c_in2, c_in3, sum[1]);
```

```
  add_full      G3 (a[2], b[2], c_in3, c_in4, sum[2]);
```

```
  add_full      G4 (a[3], b[3], c_in4, c_out, sum[3]);
```

```
endmodule
```


Constants 唯讀

- Declared with the keyword **parameter**.
- The value of a constant may not be changed during **simulation**, but may be changed during **compilation**.
 - **parameter** high_index = 31; // integer
 - **parameter** width = 32, depth = 1024;
 - **parameter** byte_size = 8, byte_max = byte_size-1;
 - **parameter** a_real_value = 6.22; // real
 - **parameter** av_delay = (min_delay + max_delay)/2;
 - **parameter** initial_state = 8'b1001_0110;

Direct Substitution of Parameters

```
module modXnor (a, b, y_out);
```

```
    parameter
```

```
    input      [size-1:0]
```

```
    output     [size-1:0]
```

```
    wire       [size-1:0]
```

```
endmodule
```

```
    size = 8, delay = 15;
```

```
    a, b;
```

```
    y_out;
```

```
    #delay y_out = a ~^ b; // bitwise xnor
```



```
module param;
```

```
    wire [7:0]
```

```
    wire [3:0]
```

```
    modXnor
```

```
    modXnor
```

```
endmodule
```

```
    a1, b1, y1_out;
```

```
    a2, b2, y2_out;
```

```
    G1(a1, b1, y1_out);
```

```
    #(4, 5) G2(a2, b2, y2_out);
```

宣告電路時設定 parameter 數值
G2.size = 4, G2.delay = 5

- A module instantiation **may not** have delay associated with it.
- Parameters **may not** be associated with a primitive gate.

Operators

OPERATOR	ARGUMENT	RESULT
Arithmetic	Pair of Operands	Binary Word
Bitwise	Pair of Operands	Binary Word
Reduction	Single Operand	Bit
Logical	Pair of Operands	Boolean Value
Relational	Pair of Operands	Boolean Value
Shift	Single Operand	Binary Word
Conditional	Three Operands	Expression

Arithmetic Operators

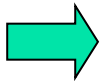
- The bit pattern stored in a register is interpreted as an **unsigned value**.
- A negative value is stored in 2's complement format.

2的補數 (負數)

SYMBOL	OPERATOR
+	Addition
-	Subtraction
*	Division
/	Multiplication
%	Modulus

Example

```
module arith1;
    reg [3:0]    A, B;
    wire [4:0]    sum, diff1, diff2, neg;
    assign        sum = A + B, diff1 = A - B, diff2 = B - A, neg = -A;
    initial begin
        #5        A = 5; B = 2; → printf
                   $display ("t_sim A B A+B A-B B-A -A");
                   $monitor ("%0t %d ...", $time, A, B, sum, diff1, diff2, neg);
        #10       $monitor ("%0t %b ...", $time, A, B, sum, diff1, diff2, neg);
    end
endmodule
```

**Simulation
result** 

# t_sim	A	B	A+B	A-B	B-A	-A
# 5	5	2	7	3	29	27
# 15	0101	0010	00111	00011	11101	11011

Bitwise Operators

- Bitwise **negation**: \sim

- $\sim(101011) = 010100$

- Bitwise **and**: $\&$

- $(010101) \& (001100) = 000100$

- Bitwise **or**: $|$

- $(010101) | (001100) = 011101$


- Bitwise **exclusive or (xor)**: \wedge

互斥或

- $(010101) \wedge (001100) = 011001$

- Bitwise **exclusive nor (xnor)**: $\sim\wedge, \wedge\sim$

- $(010101) \sim\wedge (001100) = 100110$

 If the operands do not have the same size, the **shorter** word will extend to the size of the longer word by padding bits having value "0".

Reduction Operators

- Reduction **and**: $\&$
 - $\&(010101) = 0$, $\&(010x01) = 0$
- Reduction **nand**: $\sim\&$
 - $\sim\&(010101) = 1$
- Reduction **or**: $|$
 - $|(010101) = 1$, $|(010x01) = 1$
- Reduction **nor**: $\sim|$
 - $\sim|(010101) = 0$
- Reduction **exclusive or (xor)**: \wedge
 - $\wedge(010101) = 1$
- Reduction **exclusive nor (xnor)**: $\sim\wedge$, $\wedge\sim$
 - $\sim\wedge(010101) = 0$

Logical Operators (1/2)

■ Examples

- `if (! inword) ... // true: positive integer; otherwise, false`
- `if (inword == 0) ...`
- `if ((a < size - 1) && (b != c) && (index != last_one)) ...`

- `“===”` determines whether two words match identically on a bit-bit basis, including bits that have values `“x”` or `“z”`.

- `“==”` is less sensitive, producing an `“x”` when the test is ambiguous.

SYMBOL	OPERATOR
!	Logical negation
&&	Logical and
	Logical or
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality

Logical Operators (2/2)

- “**A && B**” will return a Boolean scalar result
 - If A and B are scalars, the result is the same as obtained using “A & B”.
 - If A and B are vectors, the result returns true if both words are positive integers.
- “**A & B**” returns true if the word formed from the bitwise operation is a **positive integer**.
正整数
true
- Ex. A = 3'b**110**, B = 3'b**11x**
 - A && B = 0 (A: true, B: false)
 - A & B = 110 (true)

Relational Operators

- If the operands are nets or registers, their values are treated as **unsigned words**.
- If any bit is unknown, the relation is unknown, and the result returns “x”.

SYMBOL	OPERATOR
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Shift Operators

■ Examples

- $1001_0011 \ll 3 = 1001_1000$
- $1001_0011 \gg 3 = 0001_0010$

module shift;

reg [1:0] start, result;

initial

begin

 start = 1;

 result = (start << 1);

end

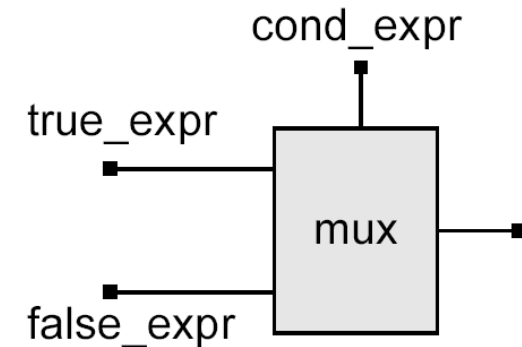
endmodule // result = 10 after execution

<<	Left shift
>>	Right shift

Conditional Operator

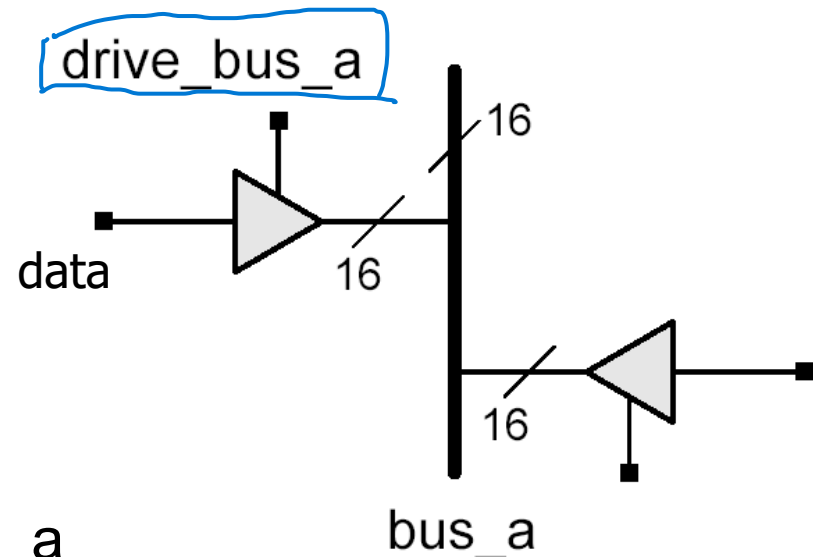
■ $Y = (A == B) ? A : B;$

- The value of A is assigned to Y if A and B are identical; otherwise, the value of B is assigned to Y.



■ **wire [15:0] bus_a = drive_bus_a ? data : 16'bz;**

- drive_bus_a = 1, set "data" on bus_a
- drive_bus_a = 0, set high impedance "z" on bus_a
- drive_bus_a = x, set "x" on bus_a



4 ^{2's} 1

4-to-1 Multiplexer

```
wire [1:0] select;
```

```
wire [15:0] driver_1, driver_2, driver_3, driver_4;
```

```
wire [15:0] bus_a = { (select == 2'b00) ? driver_1 :  
                      (select == 2'b01) ? driver_2 :  
                      (select == 2'b10) ? driver_3 :  
                      (select == 2'b11) ? driver_4 : 16'bx;
```


if
else if
else if
else if
else



Concatenation Operator

- Forms a single word from two or more operands.
- Examples
 - $A = 1011, B = 0001, \{A, B\} = 1011_0001$
 - $\{4\{a\}\} = \{a, a, a, a\}$
 - $\{0011, \{\{01\}, \{10\}\}\} = 0011_0110$

Operator Precedence

Operator Precedence	Operator	Operator Symbol
Highest	Unary	+, -, !, ~
	Multiplication, Division, Modulus	*, /, %
	Add, Subtract	+, -
	Shift	<<, >>
	Relational	<, <=, >, >=, ==, !=, ===, !==
Lowest	Conditional	? :

Reference

1. 教師自製
2. Modeling, Synthesis, and Rapid Prototyping with the Verilog HDL, Michael D. Ciletti, ISBN: 0139773983, Prentice Hall, 1999.

