100-1 Under-Graduate Project
Verilog

Speaker: 洪輝舜
Adviser: Prof. An-Yeu Wu
Date: 2011/9/27

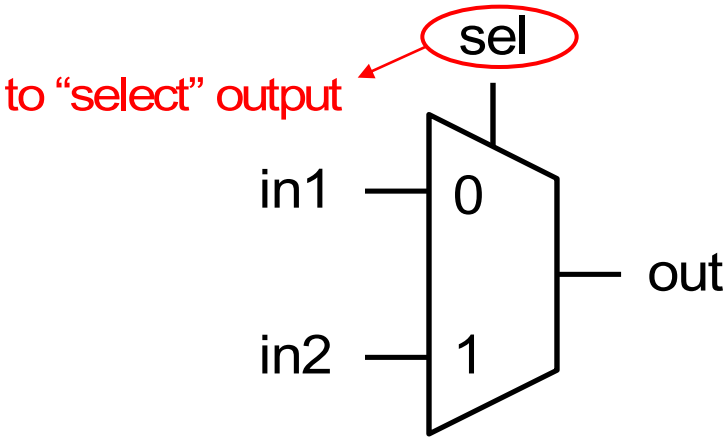
Outline

- Basic Concept of Verilog HDL
- Gate Level Modeling
- Simulation & Verification
- Summary

What is Verilog HDL ?

- Hardware Description Language
- Mixed level modeling
  - Behavioral
    - Algorithmic (like high level language)
    - Register transfer (Synthesizable)
  - Register Transfer Level (RTL)
    - Describing a system by the flow of data and control signals within and between functional blocks
    - Define the model in terms of cycles, based on a defined clock
  - Structural
    - Gate (AND, OR ……)
    - Switch (PMOS, NOMS, JFET ……)

An Example
1-bit Multiplexer

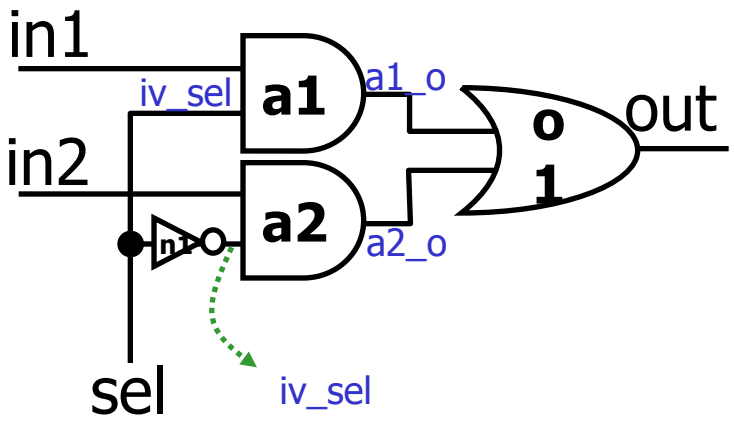


sel	in1	in2	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

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

out = (sel’ in1) + (sel in2)

Gate Level Description




```
module mux2(out,in1,in2,sel);
    output  out;
    input   in1,in2,sel;

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

Gate Level: you see only netlist (gates and wires) in the code

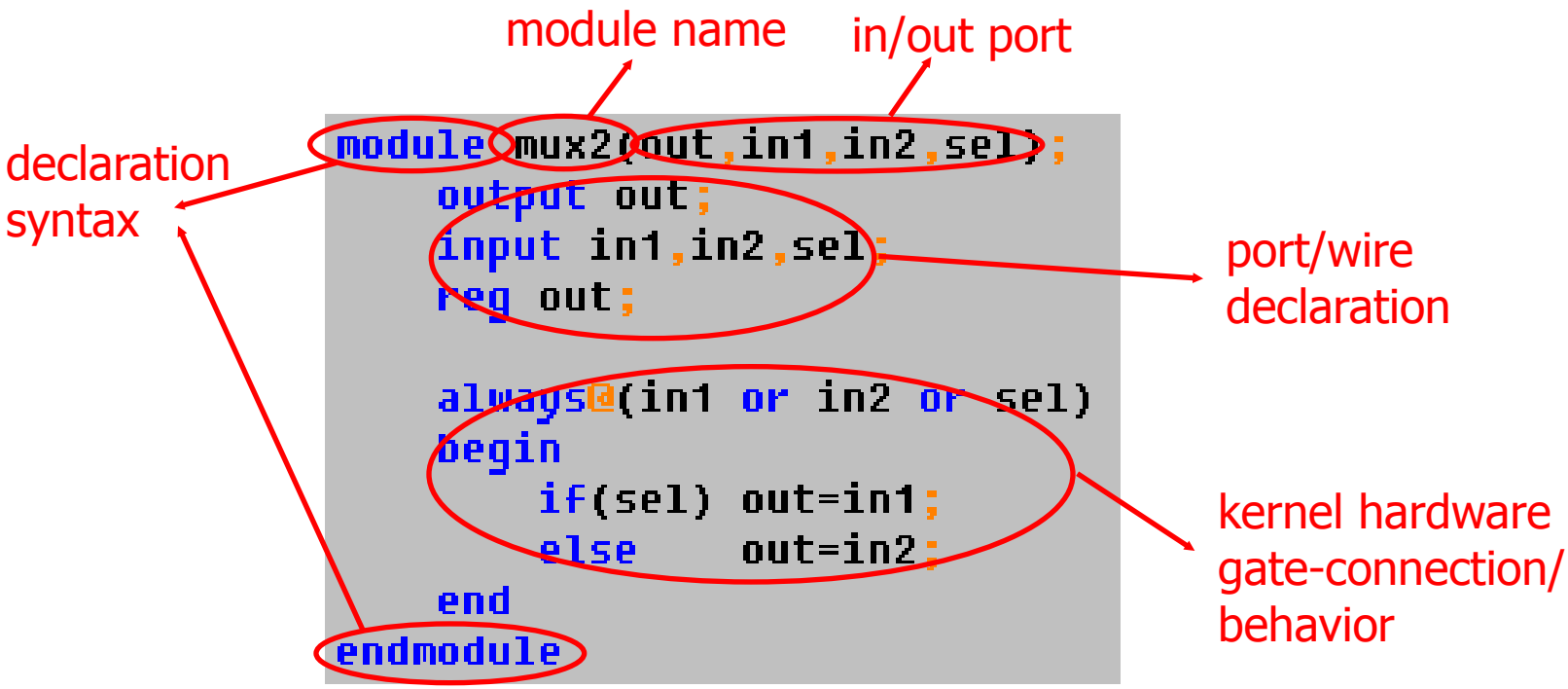
Verilog Language Rules

- Verilog is a **case sensitive** language (with a few exceptions)
- Identifiers (space-free sequence of symbols)
  - upper and lower case letters from the alphabet
  - digits (*0, 1, ..., 9*) (cannot be the first character)
  - underscore (_)
  - *$* symbol (only for system tasks and functions)
  - Max length of 1024 symbols
- Terminate lines with semicolon ;
- Single line comments:
  - // A single-line comment goes here
- Multi-line comments:
  - /* Multi-line comments like this */

```
/*  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
```

Verilog HDL Syntax



Module

- Basic building block in Verilog.
- Module
  1. Created by “declaration” (can’t be nested)
  2. Used by “instantiation“
- Interface is defined by ports
- May contain instances of other modules
- All modules run concurrently

Instances

- A module provides a template from which you can create actual objects.
- When a module is invoked, Verilog creates a unique object from the template.
- Each object has its own name, variables, parameters and I/O interface.

Module Instantiation

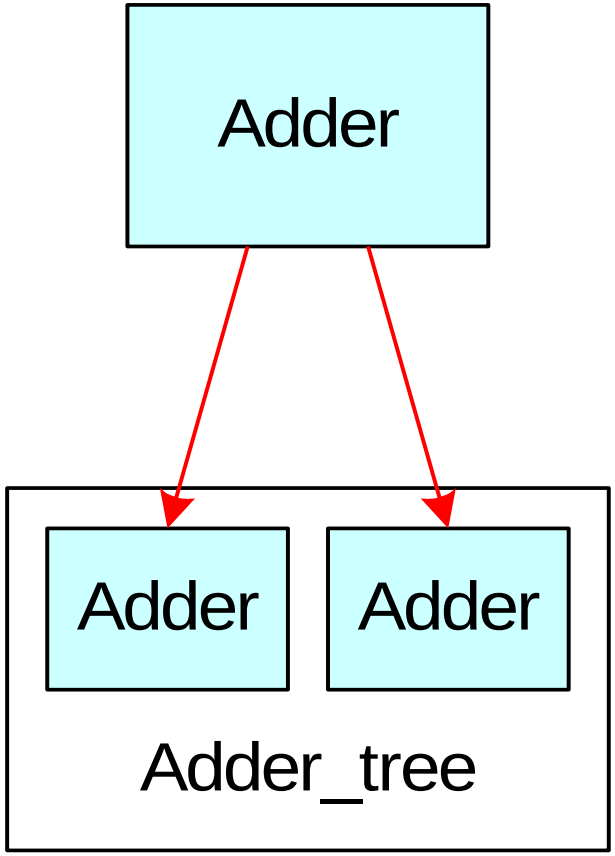
```
module adder(out,in1,in2);
    output  out;
    input   in1,in2,sel;

    assign  out=in1 + in2;
endmodule
```

instance example

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

    adder   add_0 (out0,in1,in2);
    adder   add_1 (out1,in3,in4);
endmodule
```



Port Connection

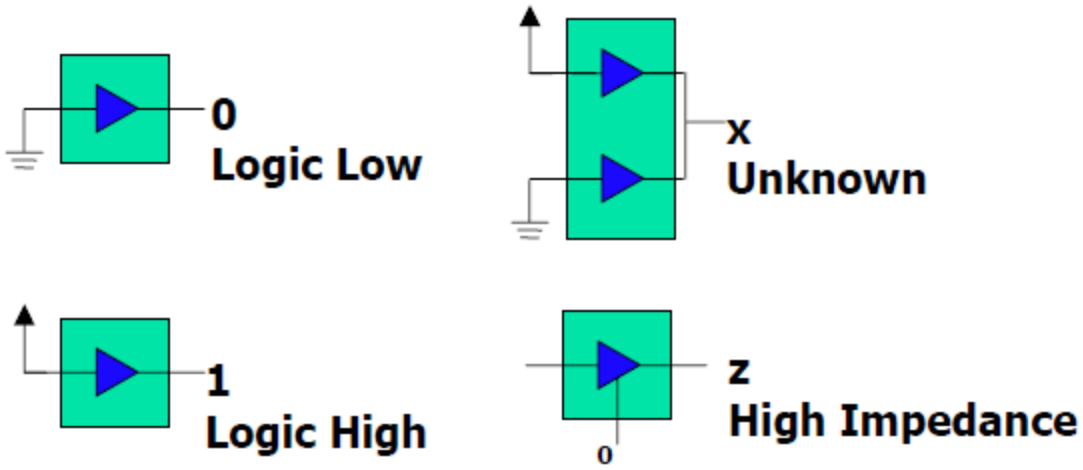
```
module FA1 (CO,S,A,B,CI);
    output  CO,S;
    input   A,B,CI;

    assign  {CO,S} = A+B+CI;
endmodule
```

- Connect module port by order list
  - FA1 fa1(c_o, sum, a, b, c_i);
- Connect module port by name *.PortName(NetName)*
  - FA1 fa2(.A(a), .B(b), .CO(c_o),.CI(c_i), .S(sum));
  - Recommended

Value Sets

- 4-value logic system in Verilog
  - **Four values: 0, 1, x or X, z or Z // Not case sensitive here**



Register and Net

- Registers
  - Keyword : **reg,** integer, time, real
  - Event-driven modeling
  - Storage element (modeling sequential circuit)
  - Assignment in “always” block
  - Default value is X
- Nets
  - Keyword : **wire,** wand, wor, tri
    triand, trior, supply0, supply1
  - Doesn’t store value, just a connection
  - input, output, inout are default “wire”
  - Can’t appear in “always” block assignment
  - Default value is Z

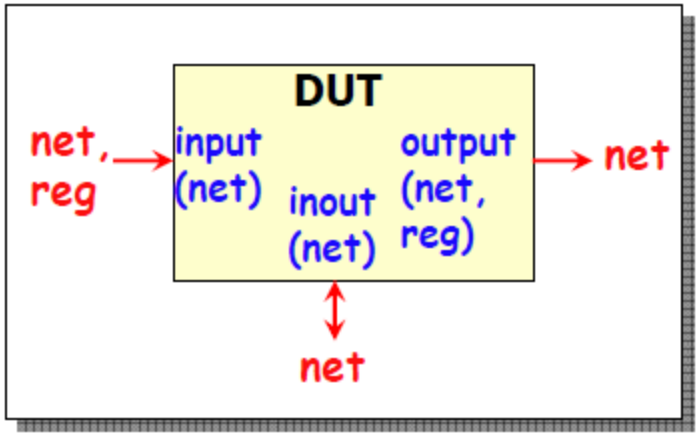
Wire & Reg

- reg
  - A variable in Verilog
- Use of “reg” data type is not exactly synthesized to a really register.
- Use of wire & reg
  - When use “wire” ➔ usually use “assign” and “assign” **does not** appear in “always” block
  - When use “reg” ➔ only use “a=b” , always appear in “always” block

```
module test(a,b,c,d);
input a,b;
output c,d;
reg d;
assign c=a;
always @(b)
  d=b;
endmodule
```

Choosing the Correct Data Types

- An **input or inout port must be a net.**
- An **output port can be a register data type.**
- A signal assigned a value in a procedural block must be a register data type.



Number Representation

- Format: <size>'<base_format><number>
  - <size> - decimal specification of number of bits
    - default is unsized and machine-dependent but at least 32 bits
  - <base format> - ' followed by arithmetic base of number
    - <d> <D> - decimal - default base if no <base_format> given
    - <h> <H> - hexadecimal
    - <o> <O> - octal
    - <b> <B> - binary
  - <number> - value given in base of <base_format>
    - _ can be used for reading clarity
    - If first character of sized, binary number 0, 1, x or z, will extend 0, 1, x or z (defined later!)

Number Representation

- Examples:
  - 6′b010_111 gives 010111
  - 8′b0110 gives 00000110
  - 4′bx01 gives xx01
  - 16′H3AB gives 0000001110101011
  - 24 gives 0...0011000
  - 5′O36 gives 11110
  - 16′Hx gives xxxxxxxxxxxxxxx
  - 8′hz gives zzzzzzzz

Net Concatenations

- A easy way to group nets

Representation	Meanings
{cout, sum}	{cout, sum}
{b[7:4],c[3:0]}	{b[7], b[6], b[5], b[4], c[3], c[2], c[1], c[0]}
{a,b[3:1],c,2'b10}	{a, b[3], b[2], b[1], c, 1'b1, 1'b0}
{4{2'b01}}	8'b01010101
{{8{byte[7]}},byte}	Sign extension

Compiler Directives

- `*define*
  - `define RAM_SIZE 16
  - Defining a name and gives a constant value to it.
- `*include*
  - `include adder.v
  - Including the entire contents of other verilog source file.
- `*timescale*
  - `timescale 100ns/1ns
  - Setting the reference time unit and time precision of your simulation.

System Tasks

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

Gate Level Modeling

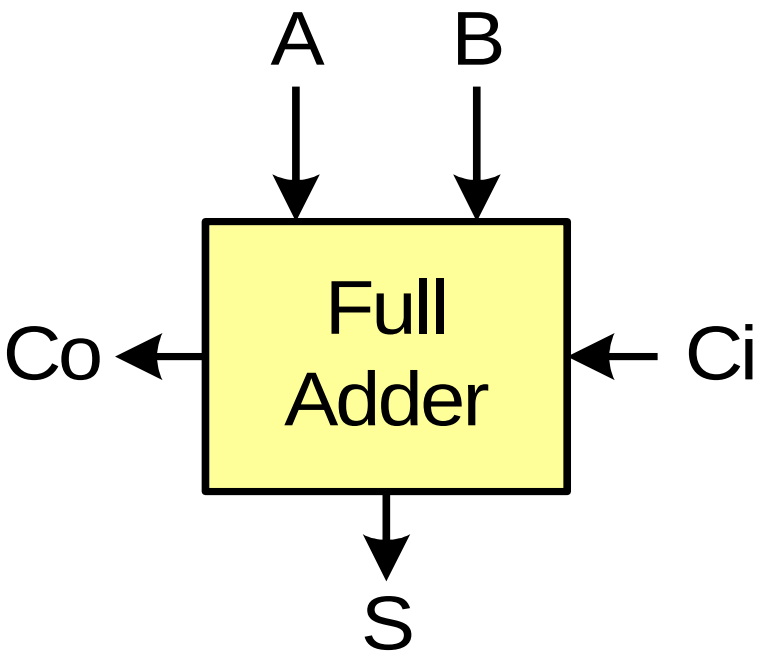
Gate Level Modeling

- Steps
  - Develop the boolean function of output
  - Draw the circuit with logic gates/primitives
  - Connect gates/primitives with net (usually wire)
- HDL: Hardware **Description** Language
  - Figure out architecture first, then write code.

Primitives

- Primitives are modules ready to be instanced
- Smallest modeling block for simulator
- Verilog build-in primitive gate
  - *and, or, not, buf, xor, nand, nor, xnor*
  - prim_name inst_name(output, in0, in1,....);
    - EX. and g0(a, b, c);
- User defined primitive (UDP)
  - building block defined by designer

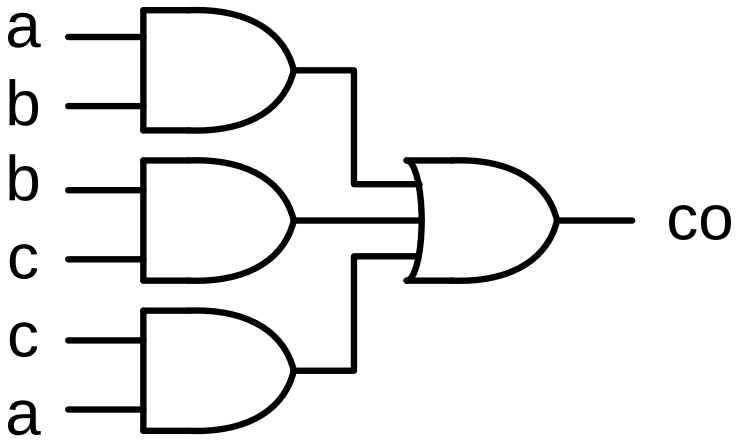
Case Study
1-bit Full Adder



Ci	A	B	Co	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
1-bit Full Adder

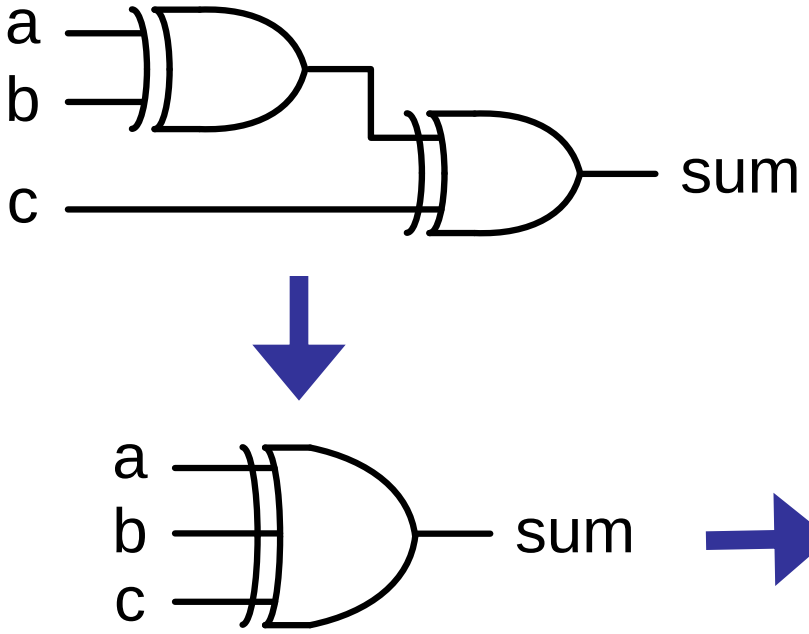
- co = (a • b) + (b • ci) + (ci • a);




```
30
31 module FA_co ( co, a, b, ci );
32
33   input   a, b, ci;
34   output  co;
35   wire    ab, bc, ca;
36
37   and g0( ab, a, b );
38   and g1( bc, b, c );
39   and g2( ca, c, a );
40   or  g3( co, ab, bc, ca );
41
42 endmodule
43
```

Case Study
1-bit Full Adder

- sum = a ⊕ b ⊕ ci

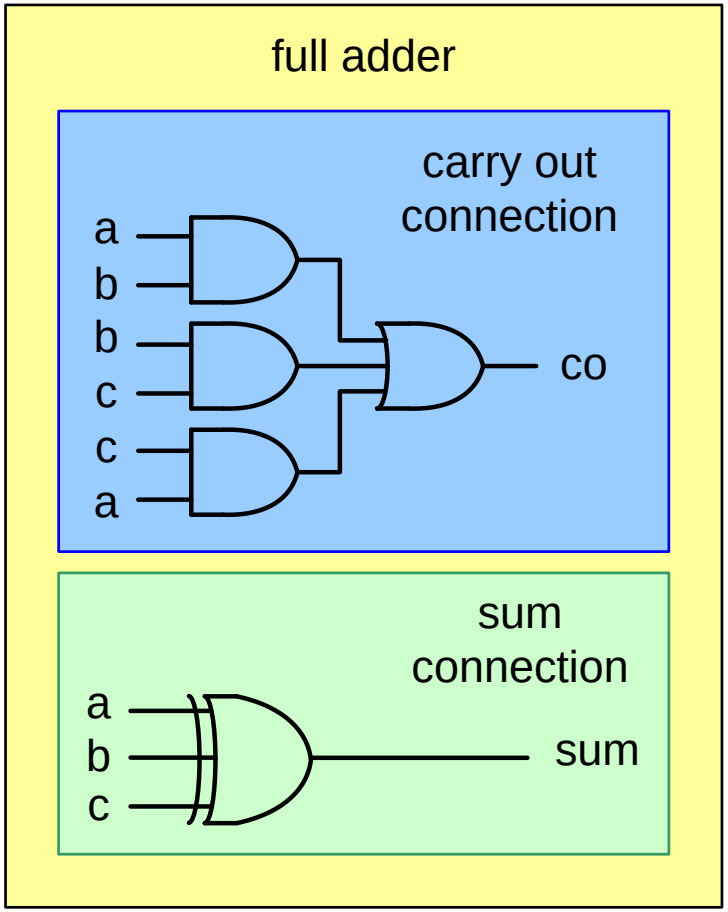



```
44 module FA_sum ( sum, a, b, ci );
45
46   input    a, b, ci;
47   output   sum, co;
48
49   xor g1( sum, a, b, ci );
50
51 endmodule
52
```

Case Study
1-bit Full Adder

- Full Adder Connection
  - Instance *ins_c* from FA_co
  - Instance *ins_s* from FA_sum

```
20
21 module FA_gatelevel( sum, co, a, b, ci);
22
23   input    a, b, ci;
24   output   sum, co;
25
26   FA_co    ins_c( co, a, b, ci );
27   FA_sum   ins_s( sum, a, b, ci );
28
29 endmodule
30
```

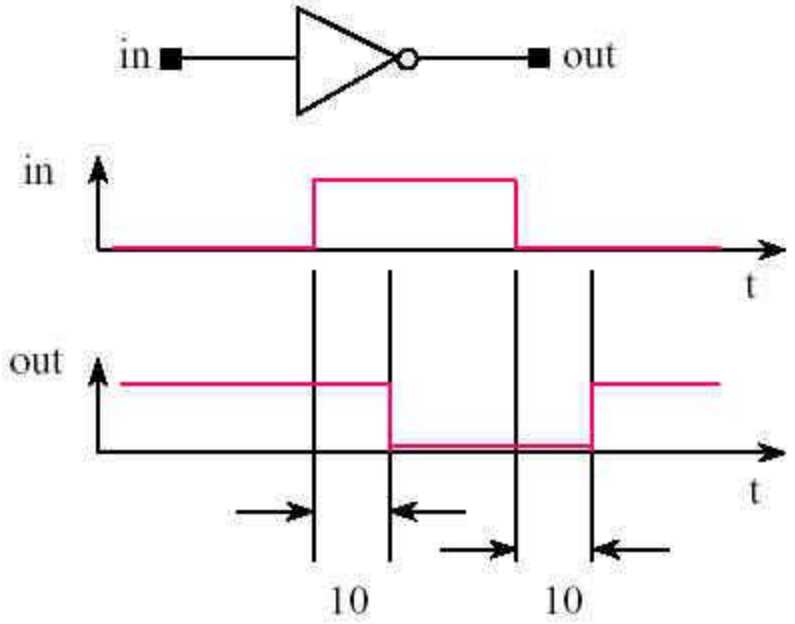


Timing and Delays

Delay Specification in Primitives

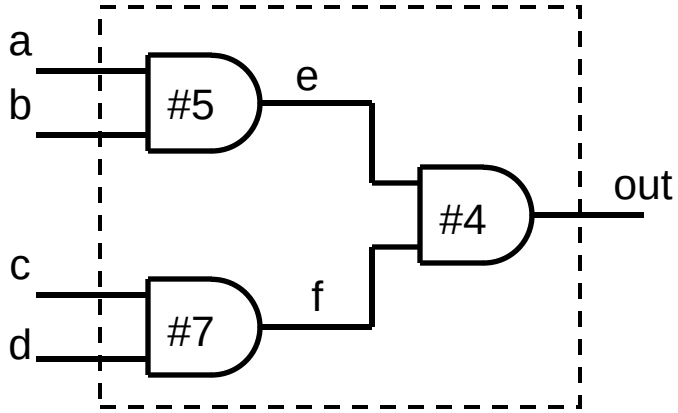
- Delay specification defines the propagation delay of that primitive gate.

not #10 (out,in);



Types of Delay Models

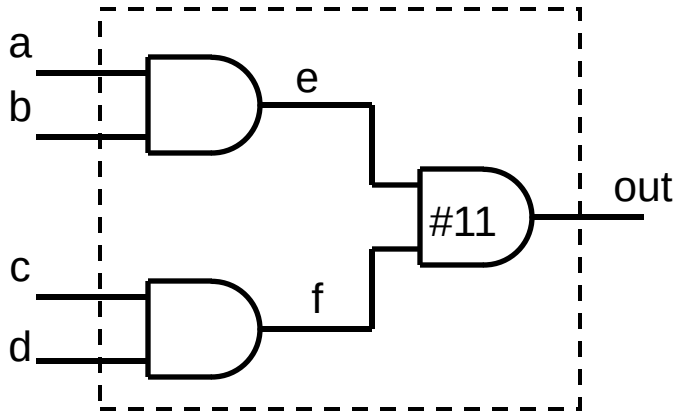
- Distributed Delay
  - Specified on a per element basic
  - Delay value are assigned to individual in the circuit




```
module  and4(out, a, b, c, d);
  … …
  and  #5  a1(e, a, b);
  and  #7  a2(f, c, d);
  and  #4  a3(out, e, f);
endmodule
```

Types of Delay Models

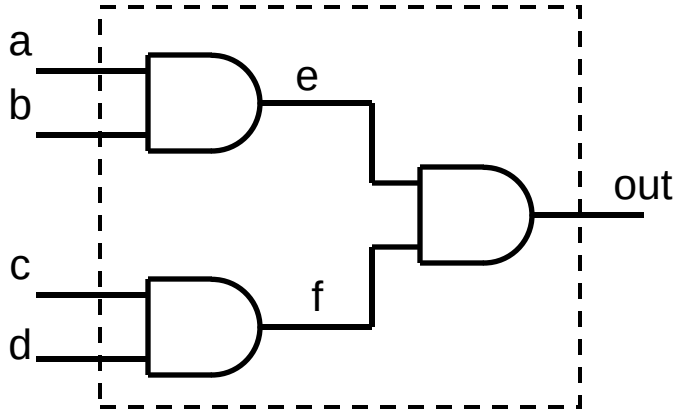
- Lumped Delay
  - They can be specified as a single delay on the output gate of the module
  - The cumulative delay of all paths is lumped at one location




```
module  and4(out, a, b, c, d);
  … …
  and          a1(e, a, b);
  and          a2(f, c, d);
  and  #11     a3(out, e, f);
endmodule
```

Types of Delay Models

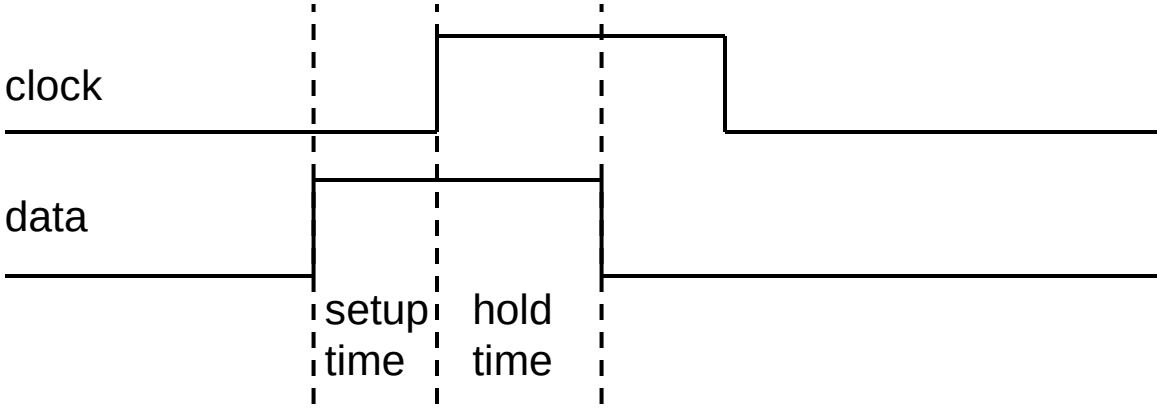
- Pin-to-Pin Delay
  - Delays are assigned individually to paths from each input to each output.
  - Delays can be separately specified for each input/output path.



Path a-e-out, delay = 9
Path b-e-out, delay =9
Path c-f-out, delay = 11
Path d-f-out, delay = 11

Timing Checks

- setup and hold checks



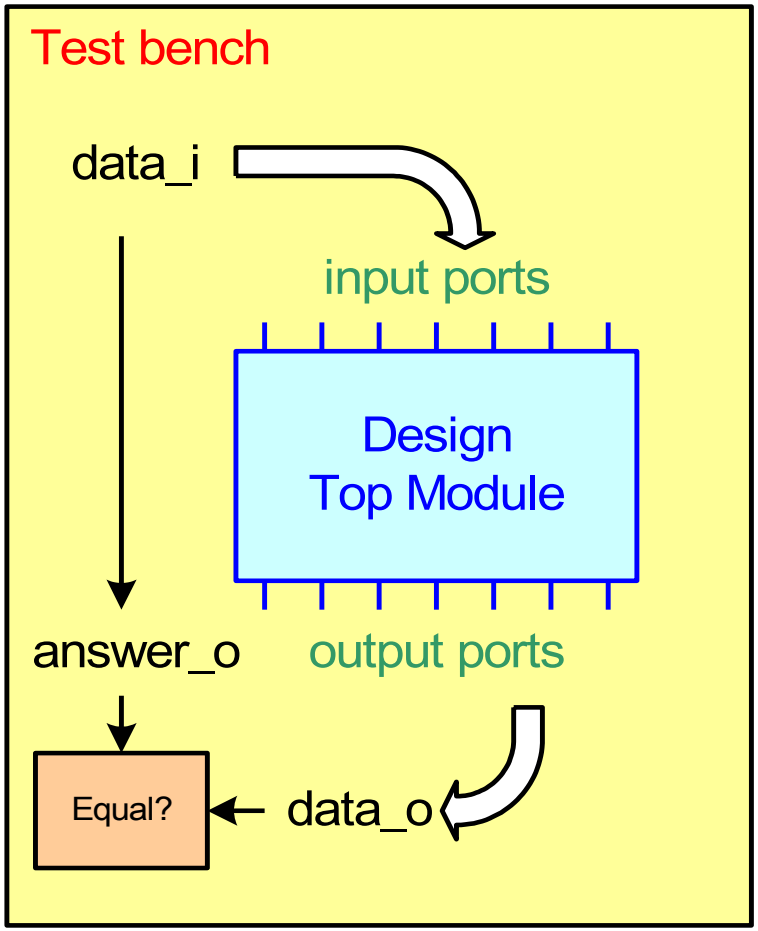

```
specify
    $setup(data, posedge clock, 3);
endspecify
```

```
specify
    $hold(posedge clock, data, 5);
endspecify
```

Simulation & Verification

Test Methodology

- Systematically verify the functionality of a model.
- Simulation:
  - (1) detect syntax violations in source code
  - (2) simulate behavior
  - (3) monitor results



Testbench for Full Adder

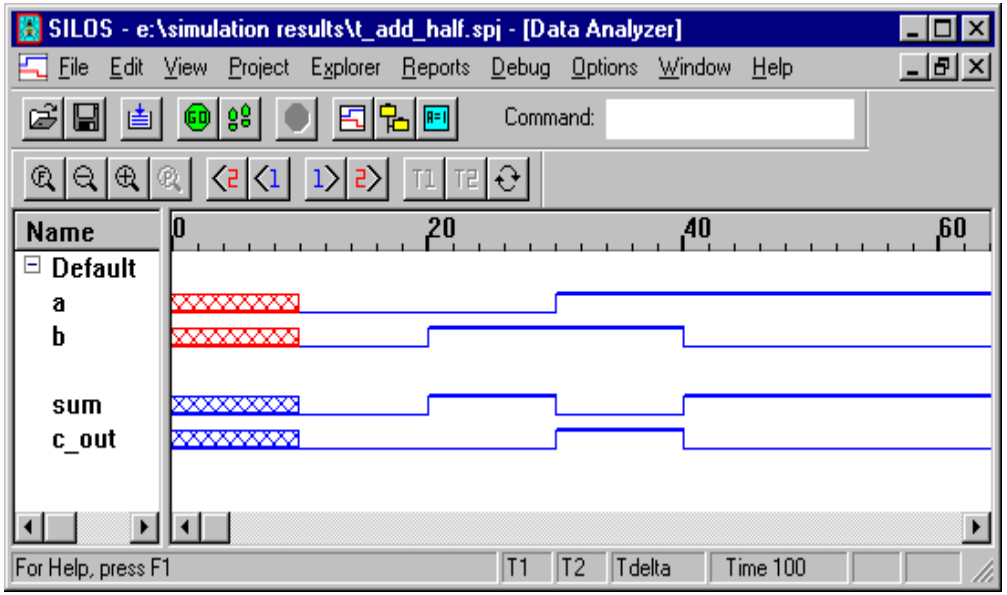
```
module t_full_add();
reg    a, b, cin;            // for stimulus waveforms
wire   sum, c_out;

full_add M1 (sum, c_out, a, b, cin); //DUT

initial #200 $finish;        // Stopwatch

initial begin                // Stimulus patterns
$fsdbDumpfile("t_full_add.fsdb");
$fsdbDumpvars;
#10 a = 0; b = 0; cin = 0; // Statements execute in sequence
#10 a = 0; b = 1; cin = 0;
#10 a = 1; b = 0; cin = 0;
#10 a = 1; b = 1; cin = 0;
#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
```

Simulation Results



MODELING TIP

A Verilog simulator assigns an *initial* value of **x** to all variables.

Example: Propagation Delay

- Unit-delay simulation reveals the chain of events

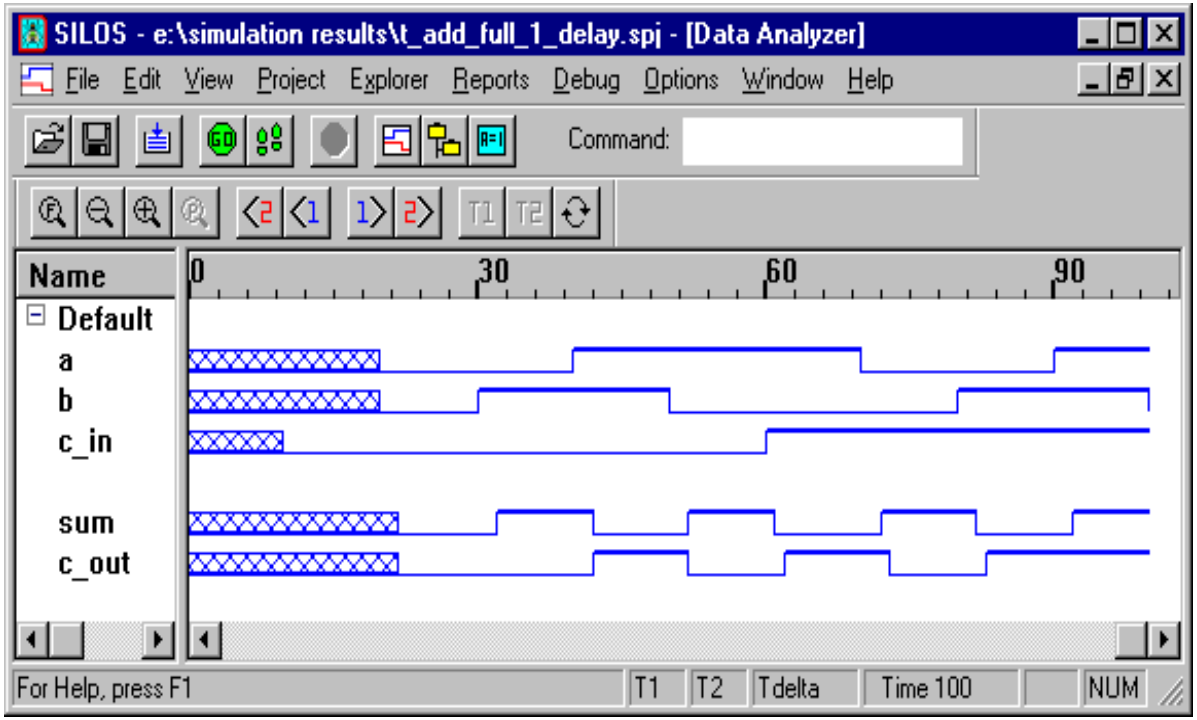
```
module Add_full (sum, c_out, a, b, c_in);
  output          sum, c_out;
  input           a, b, c_in;
  wire            w1, w2, w3;

  Add_half                  M1 (w1, w2, a, b);
  Add_half                  M2 (sum, w3, w1, c_in);
  or              #1        M3 (c_out, w2, w3);
endmodule

module Add_half (sum, c_out, a, b);
  output          sum, c_out;
  input           a, b;

  xor             #1 M1 (sum, a, b);   // single delay value format
  and             #1 M2 (c_out, a, b); // others are possible
endmodule
```

Simulation with delay



Summary

- Design module
  - Gate-level or RT-level
  - Real hardware
    - Instance of modules exist all the time
  - Each module has architecture figure
    - Plot architecture figures before you write verilog codes
- Test bench
  - Feed input data and compare output values versus time
  - Usually behavior level
  - Not real hardware, just like C/C++

Note

- Verilog is a platform
  - Support hardware design (design module)
  - Also support C/C++ like coding (test bench)
- How to write verilog well
  - Know basic concepts and syntax
  - Get a good reference (a person or some code files)
  - Form a good coding habit
    - Naming rule, comments, format partition (assign or always block)
- Hardware
  - Combinational circuits
    - 畫圖(architecture), then 連連看(coding)
  - Sequential circuits
    - register: element to store data