Dept. Electronics Engineering, National Chiao Tung University

Introduction to Verilog HDL

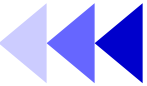
Chun-Wei Ku

sjerry@twins.ee.nctu.edu.tw

Dept. Electronics Engineering

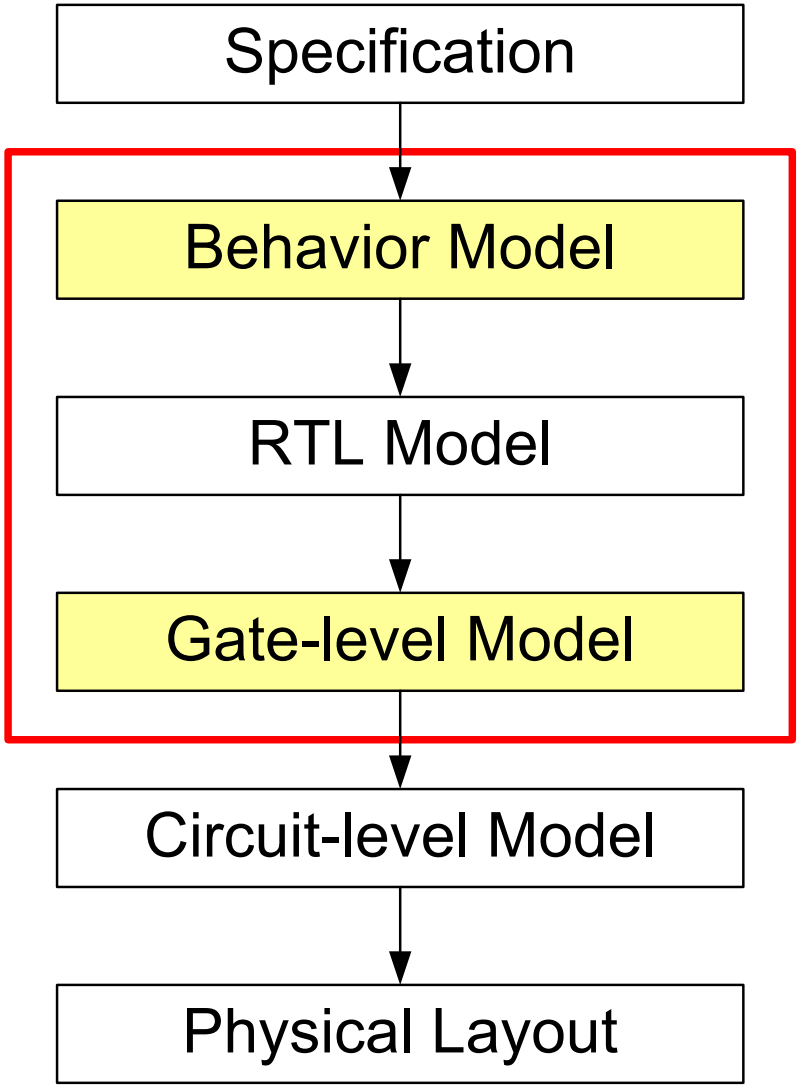
National Chiao-Tung University

What is HDL?



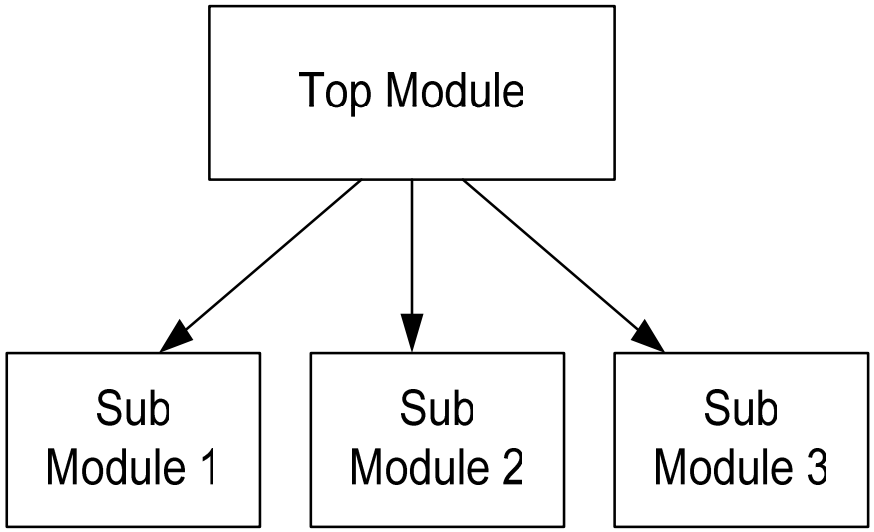
- Hardware Description Language
  - Verilog-1995, Verilog-2001
  - VHDL
- Design complexity increases in the past 20 years
  - Gate counts increase
  - Complex algorithm in communication and multi-media
  - Design time and time-to-market pressure
- Why we use Verilog?
  - Easy for beginners
  - Adopted by more than 90% companies

HDL Design Flow



Verilog HDL supported

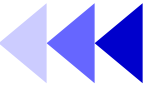
Module (1/2)



`include "file_name.v"

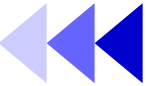
- Similar to C/C++ function
  - top module => main()
- Using ***`include*** to combine modules in different files
- A module has four parts
  - ***Port declaration***
  - ***Data type declaration***
  - ***Task & function declaration***
  - ***Module structure***

Module (2/2)



```
// Module Declaration
module FullAdder_1bit ( a, b, cin, sum, cout);

// Port Declaration
input     a, b, cin;
output    sum, cout;

// Data Type Declaration
wire      sum, cout, tmp;

// Module Structure
xor       #(5, 5)  ( tmp, a, b );
xor       #(5, 5)  ( sum, tmp, b );
……

endmodule
```

Data Type



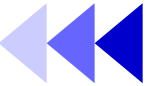
- Declaration Syntax

  <data type> [msb:lsb] <data name1>, <data name2>;

- Example

```
wire  [7:0]    a;        // 8-bit wire a with msb=a[7] and lsb=a[0]
wire  [0:7]    b;        // 8-bit wire b with msb=a[0] and lsb=a[7]
wire           c, d;     // 1-bit wire c and 1-bit wire d
reg   [15:0]   x;        // 16-bit data storage element
integer        i;        // 32-bit signed integer i
real           cycle;    // real number cycle
```

Port



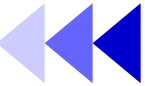
- Declaration Syntax

  <input / output / inout> [msb:lsb] <data name1>;

- Port connection & Module Hierarchy

```
module ADD_R ( A, B, CI, S, CO);
endmodule

// connected by ordering
ADD_R  #(16)  FA0( x, y, 1'b0, z, overflow);

// connected by name
 ADD_R  #(16)  FA1( .A(x), .B(y), .S(z), .CI(1'b0), .CO(overflow) );
```

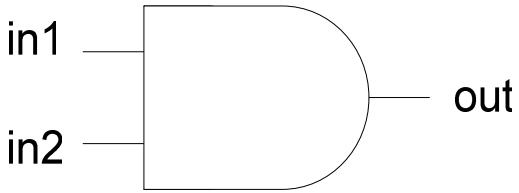
Gate-level Design (1/2)

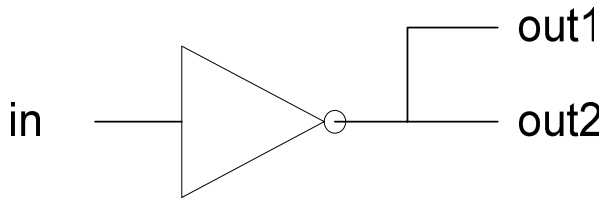


- Logic Gate Syntax

  <gate> #([Tplh], [Tphl]) (out1, …, in1, in2, …)

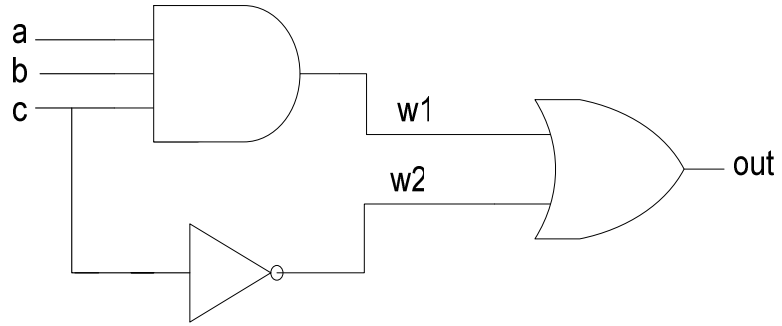
  – gate: and, or, xor, nand, nor, xnor, not, buf
  – Tplh: rise time delay (0 to 1)
  – Tphl: fall time delay (1 to 0)
  – in/out: (out_port,in_port1,in_port2, …)





Gate-level Design (2/2)

```
module Comb_logic(a, b, c, out);

input       a, b, c;
output      out;
wire        out, w1, w2;

and   #(0.2, 0.2)  ( w1, a, b, c);
not   #(0.1, 0.1)  ( w2, c);
or    #(0.2, 0.2)  ( out, w1, w2)

endmodule
```



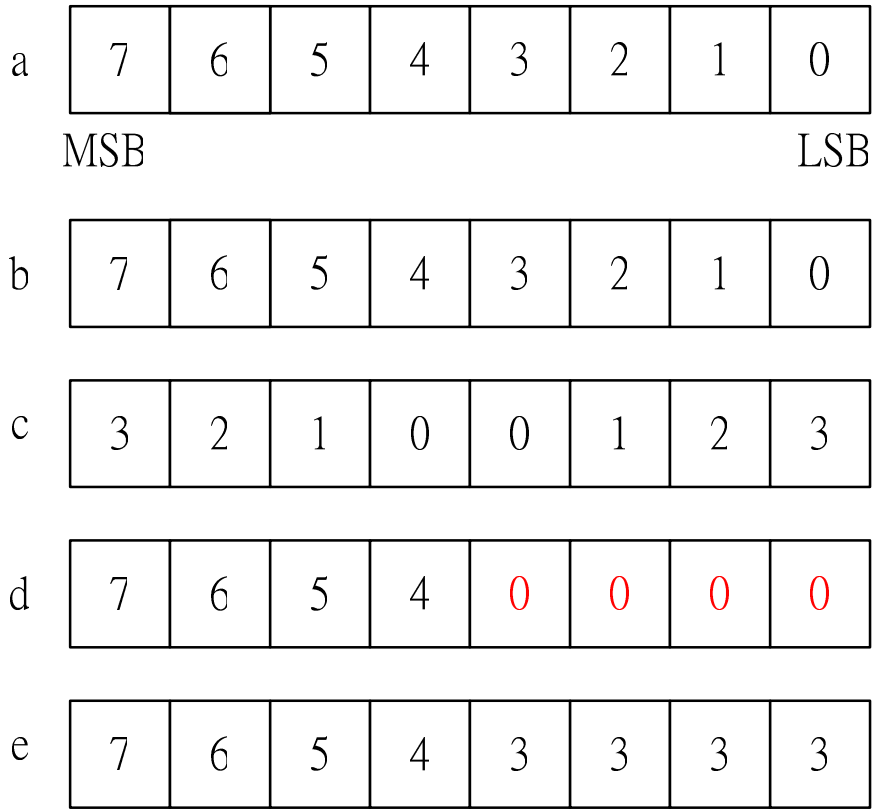
out = abc + c'

critical path delay = 0.4ns

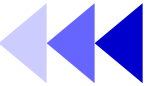
Data Assignment (1/2)

- Bit Assignment
  - example

```
wire   [7:0]        a, b, c, d, e;
b = a;
c[7:4] = a[3:0];
c[3:0] = {a[0], a[1], a[2], a[3]};
d[7:0] = {a[7:4], 4'b0};
e[7:0] = {a[7:4], {4{a[3]} } };
```



Data Assignment (2/2)



- Number Format

  <bit_number>’<base><value>

```
4’b0          // 4-bit 0000
4’b1001       // 4-bit 1001
8’b1111_1111  // 8-bit 1111 1111
16’hfe12      // 16-bit 1111 1110 0001 0010
9’o457        // 12-bit 100 101 111
8’d220        // 8-bit decimal 220
-8’d12        // 8-bit decimal -12
```