

Verilog 簡介-- for Lab 0

Reference:

M. Morris Mano and Michael D. Ciletti, *Digital Design*, 5th ed., 2013, Prentice Hall. (§3-9)

Design Flow of an Integrated Circuit (IC)

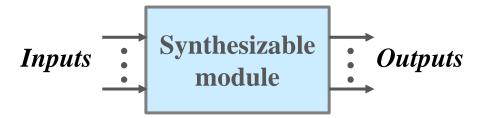
- Major steps in the design flow of an IC:
 - Design entry: Verilog HDL: Hardware Description Language
 - creates an HDL-based description of the functionality that is to be implemented in hardware
 - Function simulation or verification: ModelSim, iVerilog
 - displays the behavior of a digital system, e.g., simulation waveforms, through the use of a computer
 - Logic synthesis
 - derives a list of physical components and their interconnections, netlist, from the model of a digital system described in an HDL
 - Timing verification
 - confirms that the fabricated IC will operate at a specified speed
 - Fault simulation
 - compares the behavior of an ideal ckt w/ the behavior of a ckt that contains a process-induced flaw

Verilog Model

- Verilog model: case sensitive
 - is composed of text using keywords (100)
- Keywords:
 - are predefined *lowercase* identifiers that define the language constructs
 - E.g.s: module, endmodule, input, output, wire, and, or, not, ...
- Comments:
 - _ //: single-line comment
 - _ /* */: multiline comments
- File name: .v

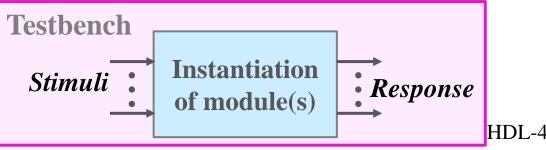
Synthesizable Modules & Testbench

- HDL code may be divided into *synthesizable modules* and a *testbench*.
- Synthesizable modules:
 - describe the hardware



- Testbench:
 - contains code to apply inputs to a module, check whether the output results are correct, and print discrepancies b/t expected and actual outputs.
 - * Testbench code is intended only for **simulation** and cannot be synthesized.

* No input/output port.

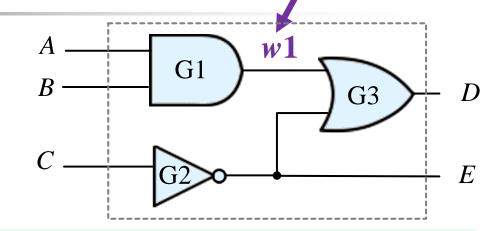


HDL Example: Gate-Level Model

Synthesizable Module:

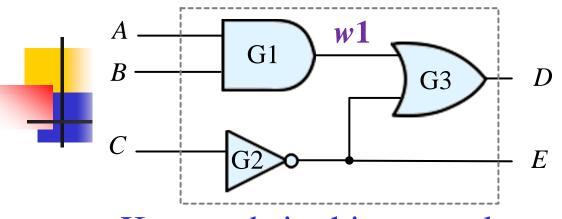
Gate-Level Model

Combinational logic
 modeled w/ primitives



```
module Simple_Circuit (A, B, C, D, E);
   output
            D, E;
   input
         A, B, C;
                          The output of a primitive gate is always
                          listed first, followed by the inputs.
   wire
           w1;
             G1(w1), A, B); //Optional gate instance name
   and
             G2(E)(C);
   not
             G3(D) w1, E);
   or
                                       Concurrence
endmodule
```

.V

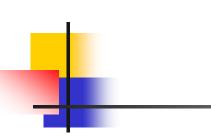


module Simple_Circuit(A, B, C, D, E);
 output D, E;
 input A, B, C;
 wire w1;

and G1(w1, A, B);
 not G2(E, C);
 or G3(D, w1, E);
endmodule

Keywords in this example:

- module: start the declaration (description) of a module
 - > is followed by a *name* and a *list of ports*
 - > The port list is the interface b/t the module and its environment.
 - > The inputs and outputs of a module may be listed in any order.
- endmodule: complete the declaration of a module
- input, output: specify which of the ports are inputs/outputs
- wire: declare internal connection
- and, or, not: primitive gates
 - > The output of a primitive gate is always listed first, followed by the inputs.



```
module Simple_Circuit(A, B, C, D, E);
  output D, E;
  input A, B, C;
  wire w1;

and G1(w1, A, B);
  not G2(E, C);
  or G3(D, w1, E);
endmodule
```

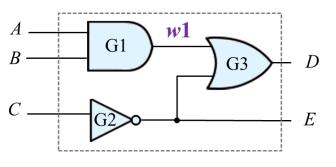
Identifiers:

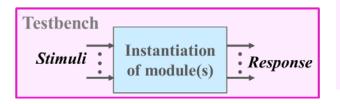
- are names given to modules, variables, and other elements
 of the language for reference in the design
- are composed of alphanumeric characters and the underscore "_ ", but can not start w/ a number.
- are case sensitive
- * Choose meaningful names for modules.
- Each statement must be terminated w/ a ";", but
 not after endmodule.

Test Bench of HDL Example

* The test bench has no input or output port ← it does not interact w/ its environment.

```
module Simple_Circuit(A, B, C, D, E);
  output D, E;
  input A, B, C;
  wire w1;
  and G1(w1, A, B);
  not G2(E, C);
  or G3(D, w1, E);
endmodule
```





```
module(t)Simple_Circuit;
  wire D, E;
                   The inputs to the ckt are declared w/ reg
                   & the outputs are declared w/ wire.
   reg A, B, C;
   //instantiate device under test
                                         an instantiation
   Simple_Circuit M1 (A, B, C, D, E);
                                         of the model to
                                         be verified
   //apply inputs one at a time
  initial
       begin
          A = 1'b0; B = 1'b0; C = 1'b0;
          #100 A = 1'b1; B = 1'b1; C = 1'b1;
      end
```

endmodule

initial #200 \$finish;

executed in sequence.

a signal generator

* The statements are

4

initial statement:

- executes the statements in its body at the *start* of simulation
- * should be used only in *testbenches* for simulation

```
module t_Simple_Circuit;
   wire D, E;
   reg A, B, C;
   //instantiate device under test
   Simple_Circuit_prop_delay M1 (A, B, C, D, E);
   //apply inputs one at a time
   initial
      begin
         A = 1'b0; B = 1'b0; C = 1'b0;
         #100 A = 1'b1; B = 1'b1; C = 1'b1;
      end
    initial #200 $finish;
endmodule
```

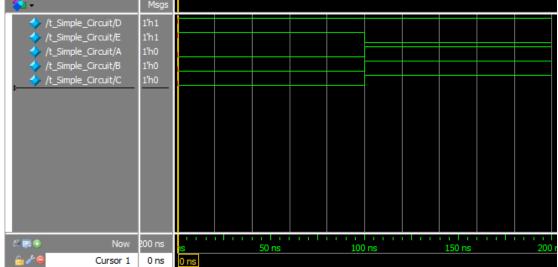
- begin, end
- \$finish: terminate the simulation
- reg: declares signals in initial statements
 - Variables of type reg retain their value until they are assigned a new value by an assignment statement



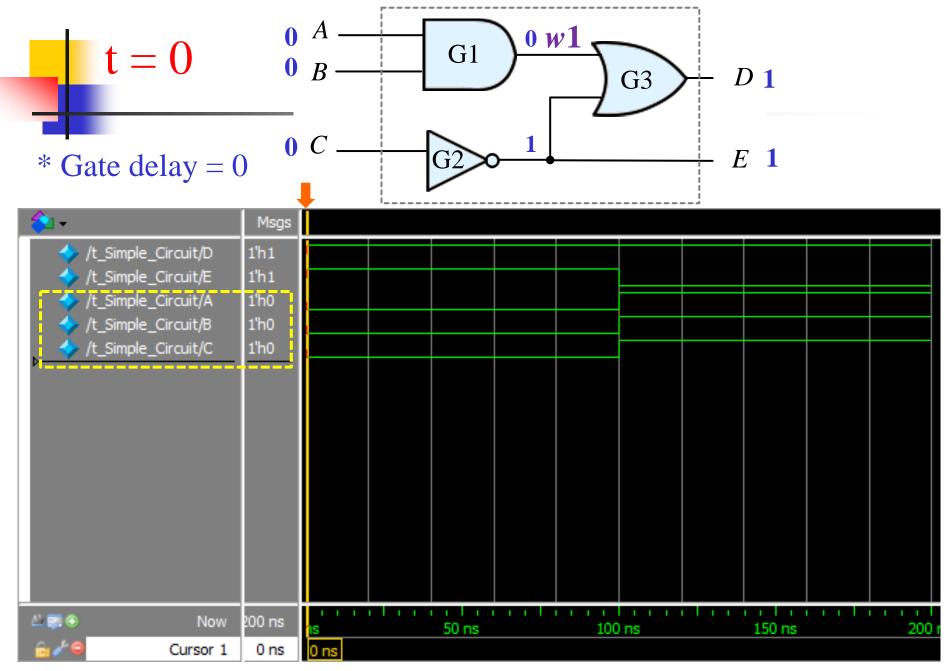
```
module Simple_Circuit(A, B, C, D, E);
  output D, E;
  input A, B, C;
  wire w1;
  and G1(w1, A, B);
  not G2(E, C);
  or G3(D, w1, E);
endmodule
```

```
module t_Simple_Circuit;
   wire D, E;
   reg A, B, C;
   //instantiate device under test
   Simple_Circuit_prop_delay M1 (A, B, C, D, E);
   //apply inputs one at a time
   initial
      begin
         A = 1'b0; B = 1'b0; C = 1'b0;
         #100 A = 1'b1; B = 1'b1; C = 1'b1;
      end
    initial #200 $finish;
endmodule
```

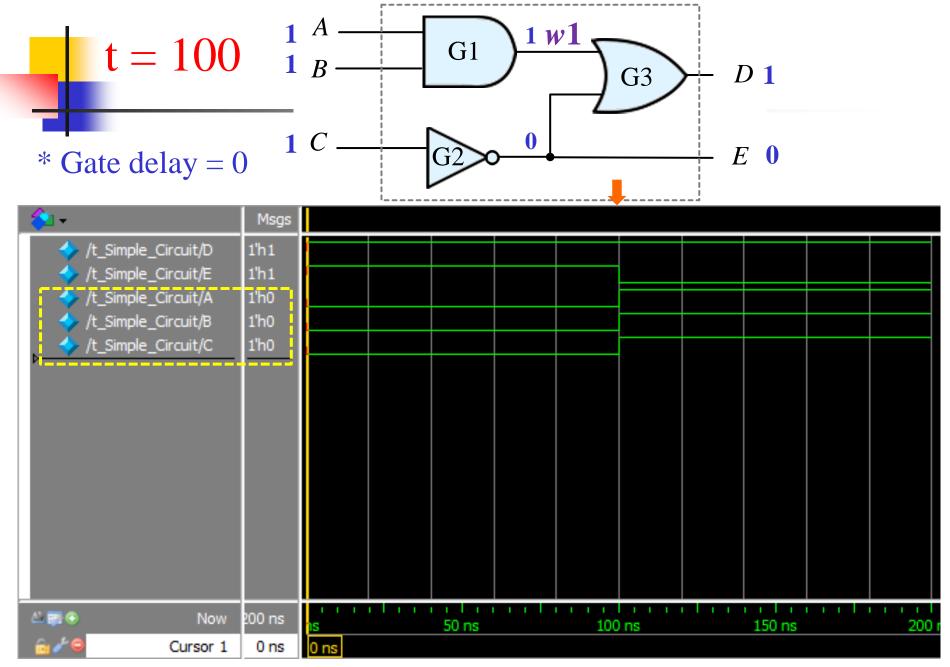
Simulation waveforms:



J.J. Shann HDL-10



J.J. Shann HDL-11



J.J. Shann HDL-12