Mano 3-9

Hardware Description Language

J.J. Shann

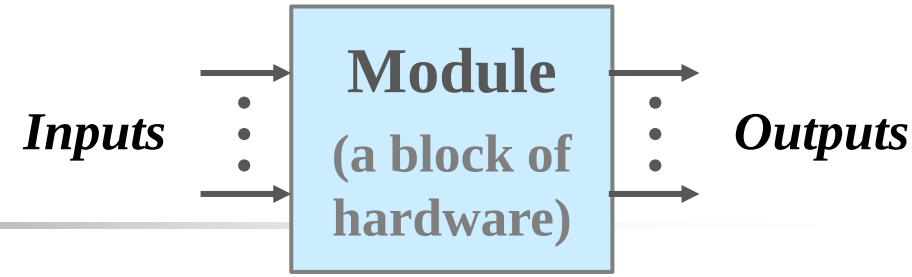
3-9 Hardware Description Language

- Hardware description language (HDL)
  - Circuit descriptions in HDL resemble code in a programming language.
  - Note: The code is intended to represent ***digital hardware***.
- Modeling styles of HDL:
  - *gate-level modeling*
  - *dataflow modeling*
  - *behavioral modeling*
- 2 leading HDLs:
  - Verilog (✓ for this course)
  - VHDL

Verilog

- Verilog: (✓ for this course)
  - was developed by Gateway Design Automation for logic simulation in 1984.
    - Gateway was acquired by Cadence in 1989.
  - was made an open standard in 1990.
  - became an IEEE standard in 1995 & was updated in 2001.
    - IEEE: the Institute of Electrical and Electronics Engineers
  - file name: **.v**
- SystemVerilog: (*)
  - was extended from Verilog in 2005
    to streamline idiosyncracies and
    to better support modeling and verification of systems.
  - file name: **.sv**

Modules

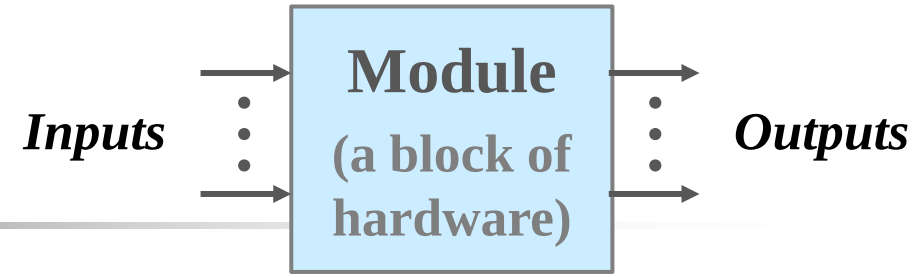


- Module:
  - a block of hardware w/ inputs and outputs
  - e.g.s: an AND gate, a MUX, a priority ckt, …
- Styles for describing module:
  - *gate-level*: use instantiations of predefined and user-defined primitive gates
  - *behavioral* model: describe what a module does
  - *structural* model: describe how a module is built from simpler pieces ⇒ an application of *hierarchy*

Design Flow of an Integrated Circuit (IC)

- Major steps in the design flow of an IC:
  - Design entry
    - creates an HDL-based description of the functionality that is to be implemented in hardware
  - Function simulation or verification
    - displays the behavior of a digital system 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

Logic Simulation



- Simulation:
  - inputs are applied to a module and outputs are checked to verify that the module operates correctly
  - E.g.: Simulation waveforms (Harris, p.170, Fig 4.1)

$$y = a'b'c' + ab'c' + ab'c$$

Figure Simulation waveforms

Logic Synthesis

- Synthesis:
  - transforms HDL code into a *netlist* describing the hardware, e.g., the logic gates and the wires connecting them.
    - *netlist*: may be a text file, or may be drawn as a schematic
  - might perform optimizations to reduce the hardware.
  - E.g.: Synthesized ckt (Harris, p.170, Fig 4.2)

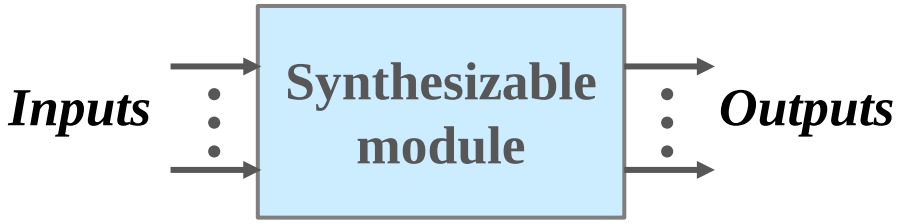
$$y = a'b'c' + ab'c' + ab'c$$

$$= b'c' + ab'$$

Figure Synthesized circuit

Synthesizable Modules & Testbench

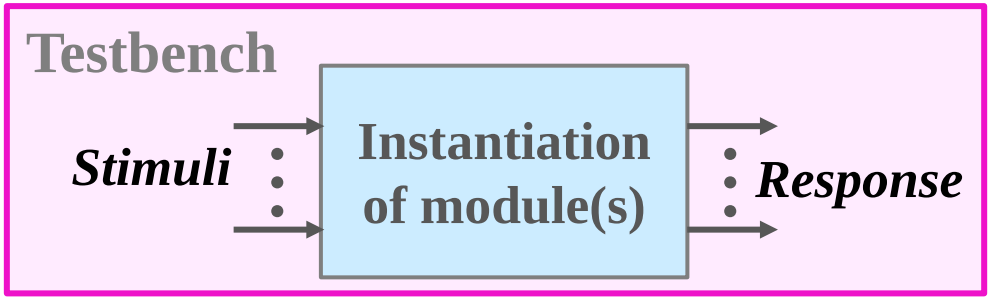
- **HDL code** may be divided into *synthesizable modules* and a *testbench*.
- Synthesizable modules:
  - describe the *hardware*



- Testbench: for *simulation*
  - contains code to apply inputs to a module, check whether the output results are correct, and print discrepancies b/t expected and actual outputs.
    - E.g.: a command to print results on the screen during simulation

* Testbench code is intended only for simulation and cannot be synthesized.

* No input/output port.



A. Module Declaration

- HDL description forms:
  - a netlist of interconnected gates
  - Boolean logic equations
  - truth tables
  - an abstract behavioral model

- Gate-level modeling:
  - use instantiations of *predefined* and *user-defined primitive gates*

Verilog Model

- Verilog model:
  - 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
- Verilog is *case sensitive*.

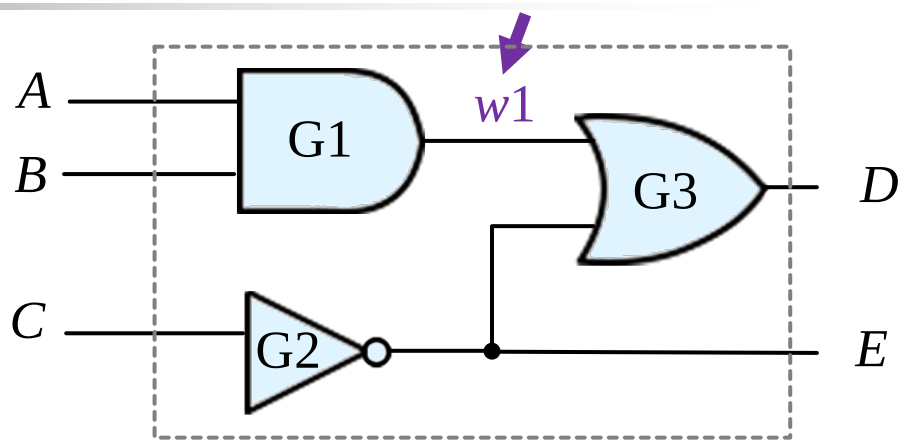
Module

- Module:
  - refers to the text enclosed by the keyword pair **module** … **endmodule**
  - is the fundamental descriptive unit
- HDL description of *combinational* logic:
  - a schematic connection of gates
  - a set of Boolean equations
  - a truth table
  - …

HDL Example: Gate-Level Model

- HDL Example:

  Gate-Level Model
  - Combinational logic modeled w/ primitives
  - IEEE 1364-1995 Syntax

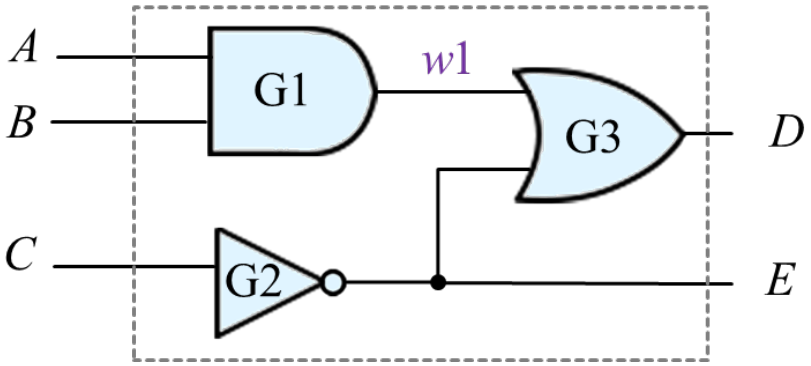


.V

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

    and      G1(w1, A, B);   //Optional gate instance name
    not      G2(E, C);
    or       G3(D, w1, E);
endmodule
```

The output of a *primitive gate* is always listed first, followed by the inputs.

* Concurrence




```
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.

Gate Delays

* Compiler directives start w/ the (`) back quote, or grave accent, symbol.

- Propagation delay: (in Verilog)
  - is specified in terms of *time units* and by the symbol # .
  - E.g.: and #(30) G1(w1, A, B);
- `timescale: a compiler directive
  - is made for association a time unit w/ physical time
  - is specified before the declaration of a module and applies to all numerical values of time in the code that follows
  - E.g.: `timescale 1ns/100ps

the unit of measurement for time delays

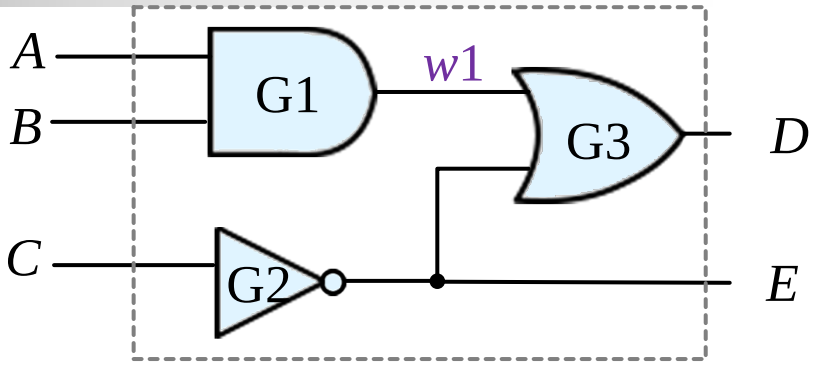
the precision for which the delays are rounded off

  - If no timescale is specified, a simulator may display dimensionless values or default to a certain time units, usually 1 ns (= 10^{-9} s).

HDL Example: Propagation Delays

- HDL Example:

  Gate-Level Model w/ Propagation Delays



.v

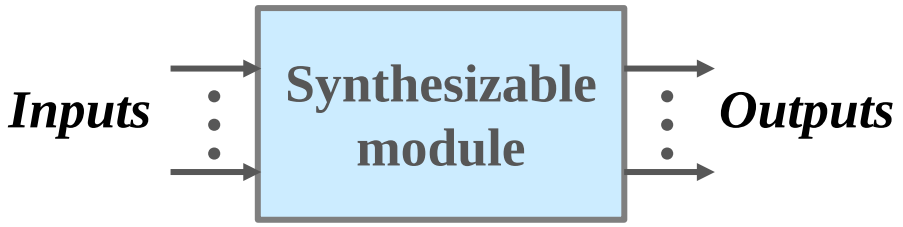
```
module Simple_Circuit_prop_delay (A, B, C, D, E);
    output   D, E;
    input    A, B, C;
    wire     w1;

    and      #(30) G1(w1, A, B);
    not      #(10) G2(E, C);
    or       #(20) G3(D, w1, E);
endmodule
```

Propagation delays of the primitive gates

– **#***n*: waits for *n* time units

Testbench

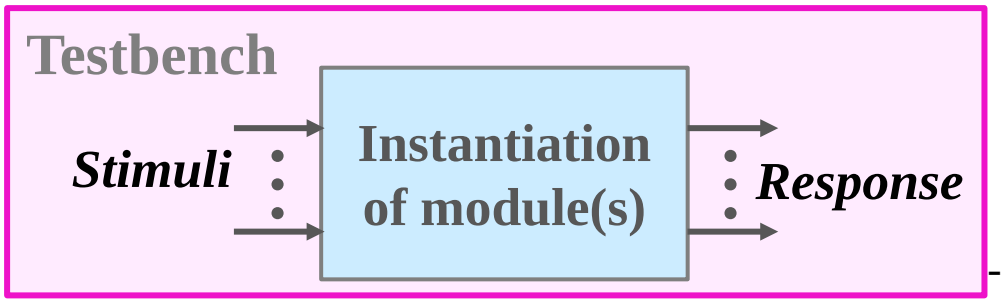
- Recall: HDL code may be divided into *synthesizable modules* and a *testbench*.
- Synthesizable modules:
  - describe the hardware



- 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.

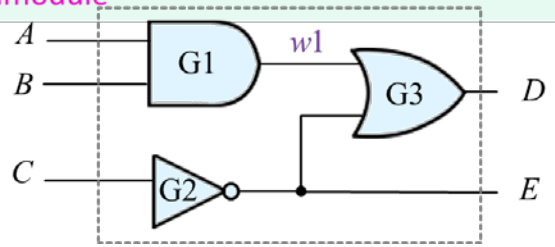


HDL Examples: Test Bench

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

- HDL Example: Testbench

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

    and      #(30) G1(w1, A, B);
    not      #(10) G2(E, C);
    or       #(20) G3(D, w1, E);
endmodule
```




```
module t_Simple_Circuit_prop_delay;
  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
```

The inputs to the ckt are declared w/ **reg** & the outputs are declared w/ **wire**.

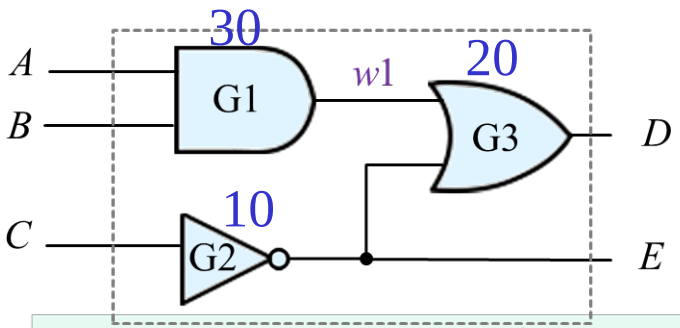
an instantiation of the model to be verified

a signal generator
* The statements are executed ***in sequence***.

.V

```
module t_Simple_Circuit_prop_delay;
    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
```

- initial statement:
  - executes the statements in its body at the start of simulation
  - * should be used only in testbenches for simulation
- 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_prop_delay(A, B, C, D, E);
    output   D, E;
    input    A, B, C;
    wire     w1;

    and      #(30) G1(w1, A, B);
    not      #(10) G2(E, C);
    or       #(20) G3(D, w1, E);
endmodule
```

```
module t_Simple_Circuit_prop_delay;
    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:

B. Boolean Expressions

- Boolean expressions of ***combinational*** logic:
  - are specified in Verilog w/ a ***continuous assignment statement*** consisting of the keyword **assign** followed by a Boolean expression
- Logic operators:

Operator	bitwise	logical
AND	&	&&
OR	|	||
NOT	~	!

* If the operands are scalar, the results of bitwise and logical operators will be identical;

  if the operands are vectors, the results will not necessarily match.

  - E.g.: A =1010, B = 0000

    A & B = 0000

    A && B = 0

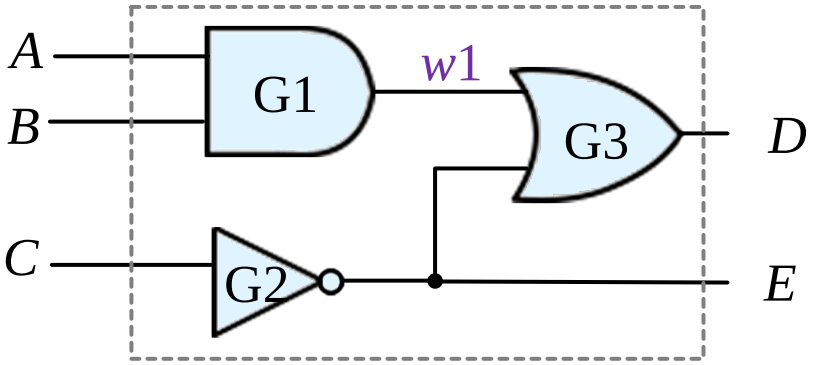
Example

- Boolean expressions of combinational logic:
  - ***continuous assignment statement*** : consists of the keyword **assign** followed by a Boolean expression
- Logic operators:

Operator	bitwise	logical
AND	&	&&
OR	|	||
NOT	~	!

- E.g.:

```
assign D = (A && B) || (!C);
```



HDL Example: Boolean Equations

- HDL Example: Combinational Logic Modeled w/ Boolean Equations

E = A + BC + B'D

F = B'C + BC'D'

Operator	bitwise	logical
AND	&	&&
OR	|	||
NOT	~	!

.V

```
module Circuit_Boolean_CA (E, F, A, B, C, D);
   output    E, F;
   input     A, B, C, D;

   assign  E = A || (B && C) || ((!B) && D);
   assign  F = ((!B) && C) || (B && (!C) && (!D));
endmodule
```

– assign statement: describe *combinational* logic

* The simulator detects when the test bench changes a value of one or more of the inputs and updates the vales of the outputs.

C. User-Defined Primitives (Truth Tables)

- *system primitives*:
  - e.g.s: and, or, not, … (in Verilog)

- *user-defined primitives* (UDPs):
  - The user can create additional primitives by defining them in tabular form ⇒ truth table
  - are declared w/ the keyword pair

    primitive … **endprimitive**
  - A UDP can be instantiated in the construction of other modules, just as the system primitives are used.

General Rules for Defining a UDP

- General rules for defining a UDP:
  - It is declared w/ the keyword **primitive**, followed by a name and port list.
  - There can be **only one output**, and it must be **listed first** in the port list and declared w/ keyword **output**.
  - There can be any # of inputs. The order in which they are listed in the **input** declaration must conform to the order in which they are given values in the table that follows.
  - The truth table is enclosed within the keywords **table** and **endtable**.
  - The values of the inputs are listed in order, ending w/ a colon (**:**). The output is always the last entry in a row and is followed by a semicolon (**;**).
  - The declaration of a UDP ends w/ the keyword **endprimitive**.

HDL Example: User-Defined Primitive

- HDL Example: User-Defined Primitive

Define UDP_02467

$D(A, B, C)$

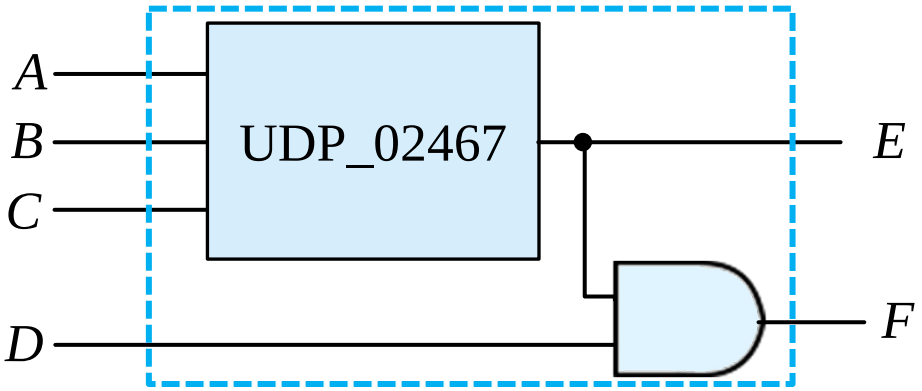
$= \Sigma m(0, 2, 4, 6, 7)$

.V

```
primitive UDP_02467 (D, A, B, C);
   output    D;
   input     A, B, C;

// Truth table for D = f(A, B, C) = Σm(0, 2, 4, 6, 7);
   table
//     A  B  C  :  D   // Column header comment
       0  0  0  :  1;
       0  0  1  :  0;
       0  1  0  :  1;
       0  1  1  :  0;
       1  0  0  :  1;
       1  0  1  :  0;
       1  1  0  :  1;
       1  1  1  :  1;
  endtable
endprimitive
```

HDL Example: (con't)

– Construct module w/ UDPs:



.v

```
// Verilog model: Circuit instantiation of Circuit_UDP_02467
module Circuit_with_UDP_02467 (e, f, a, b, c, d);
   output    e, f;
   input     a, b, c, d;

   UDP_02467  M0 (e, a, b, c);
   and  (f, e, d);        //Option gate instance name omitted
endmodule
```