



University of San Carlos | Department of
COMPUTER ENGINEERING

CpE 3101L – Introduction to HDL

Unit 2: Basic Constructs in Verilog HDL

Lecture 1

Outline

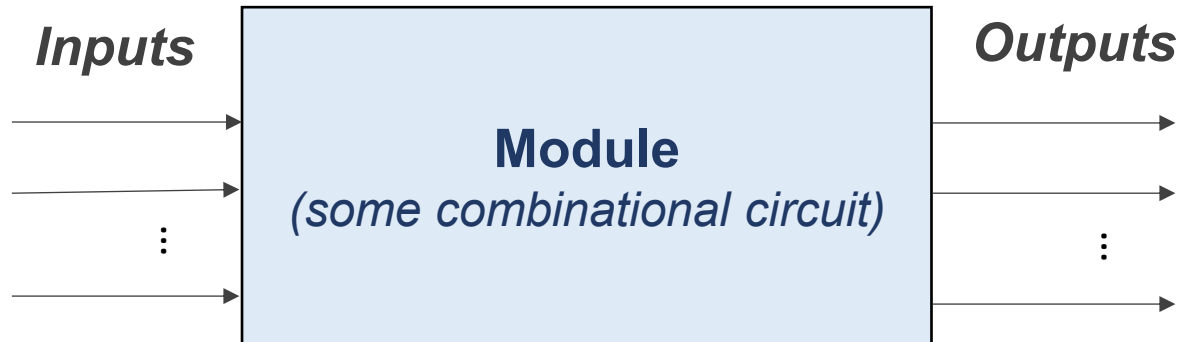
- Verilog Module
- Basic Lexical Elements and Data Types
- Module Instantiation
- Structural Modeling and Port Maps
- Gate Primitives
- Introduction to Logic Simulation
- Introduction to ModelSim*-Intel® FPGA Starter Edition

Verilog Module

- Fundamental descriptive unit in Verilog
 - Does not necessarily correspond to a physical partition in final implementation (*can be used as a logical or functional partition*)
- Module declaration made into three main parts:
 - Module name – *unique identifier to a module*
 - Ports – *external interface of a module (I/Os)*
 - Body – *functional/structural description of the module*
- The best way to understand an HDL code is ***to think in terms of hardware circuits***

Module and Port Declarations

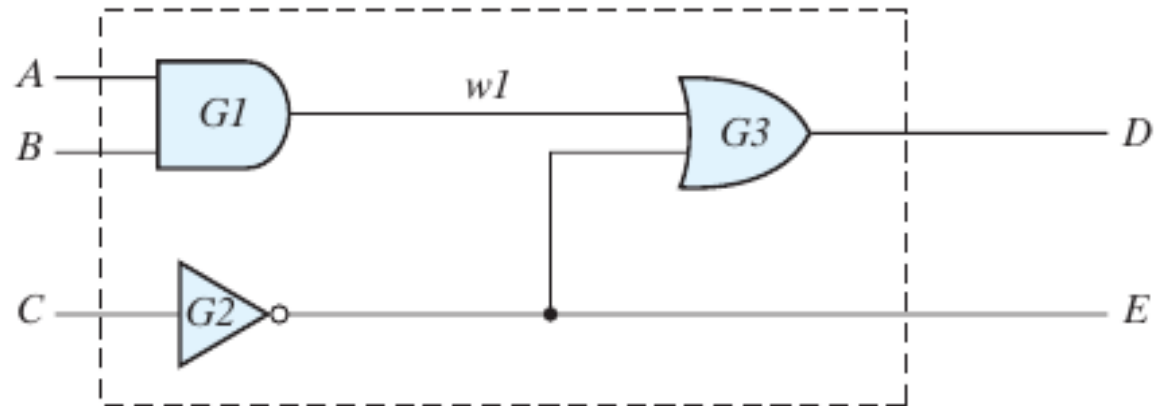
- Example:



```
module [module_name] ( [port_name1], [port_name2], ... );  
    // body;  
endmodule
```

Module and Port Declarations

- Example:



- Verilog description:

```

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

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

endmodule

```

Module name (list of ports)

Body:

- Port declaration
- Signal declaration
- Internal organization

* Statements are terminated with a *semi-colon*;

Lexical Elements

- Identifier

- Gives a **unique name to an object**
- Examples: A, G1, w1, D, etc.

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

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

- Composed of **letters, digits, underscore character (_), dollar sign (\$)**
- \$ is usually used with a system task or function
- **First character** of an identifier must be a **letter** or **underscore**
- **Tip:** It is good practice to give an object a descriptive name
- Example: mem_addr_en (*for memory address enable signal*)
- Verilog is a **case-sensitive language**
- Data_bus, data_bus, and DATA_BUS refer to 3 different objects

Lexical Elements

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

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

endmodule
```

- **Keywords**

- **Predefined lowercase identifiers** used to describe Verilog constructs
- Examples: *module*, *endmodule*, *input*, *output*, *wire*, *and*, *or*, *not*

- **White space**

- Used to separate identifiers and can be used freely in Verilog
- But should not appear within the text of a keyword, a user-specified identifier, an operator, or a representation of a number
- Includes **space**, **tab**, **newline characters (enter)**
- Used for formatting and making it more readable

Lexical Elements

- Comments

- For documentation purposes
- Ignored by software
- No effect on simulation

- One-line comment starts with `//`:

`// This is a comment.`

- Multiple-line comment is encapsulated between `/*` and `*/`

`/* This is comment line 1.`

`This is comment line 2.`

`This is comment line 3. */`

```
//
// Sample 1 for Unit 2
//
module GateLevel (A, B, C, D, E);
    input    A, B, C;
    output   D, E;
    wire     w1;

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


Four-value System

- Used in most data types:
- **0**: for “logic 0” or a **false** condition
- **1**: for “logic 1” or a **true** condition
- **z**: for the **high-impedance state** (*disconnected*)
 - z value corresponds to the output of a tri-state (three-state) buffer
- **x**: for an **unknown** value
 - x value is usually used in modeling and simulation
 - Represents a value that is not 0, 1, or z (*uninitialized input or output conflict*)

Data Type Groups

- Net group

- Represent **physical connections** between hardware components
- Used as **outputs of continuous assignments** and as **connection signals** between different modules
- Most commonly used data type: **wire**

```
wire p0, p1;    // two 1-bit signals
```

- When a collection of signals is grouped into a **bus**, this can be represented using a **1D array (vector)**:

```
wire [7:0]      data1, data2;    // 8-bit data
wire [31:0]     addr;            // 32-bit address
wire [0:7]      reverse_data;    // ascending index should be avoided
```

- **2D array** is sometimes needed to represent a memory:

```
wire [3:0] mem_32x4 [31:0];    // 32x4 memory (32 words by 4 bits wide)
```

Data Type Groups

- Variable group
 - Represent **abstract storage** in behavioral modeling
 - Used in the **outputs of procedural assignments**
 - 5 data types in this group: *reg, integer, real, time, realtime*
 - The inferred circuit may or may not contain physical storage components
 - The most common in this group and is **synthesizable**: *reg*
 - Can only be used in modeling and simulation (*not synthesizable and shouldn't be used in design entry*): *integer, real, time, realtime*

Number Representation

- An **integer constant** in Verilog can be represented in various formats
- **General form:** ~~[sign]~~[size]' [base] [value]
- *Base* refers to the base of the number:
 - B or b: binary
 - O or o: octal
 - H or h: hexadecimal
 - D or d: decimal
- *Value* specifies the value of the number in the specified base
 - The **underscore character** (`_`) can be included for readability/clarity

Number Representation

- **General form:** ~~[sign]~~[size] ' [base] [value]
- *Size* specifies the **number of bits in a number** (*optional*)
 - The number is known as a *sized number* when a [size] term exists and is known as an *unsized number* otherwise.
- *Sized number* – specifies the number of bits explicitly
 - If the size of the value is smaller, zeros are padded in front (or z or x or MSB is padded)
- *Unsized number* – actual size depends on the host computer but must be at least 32 bits

Number Representation

- General form: ~~[sign]~~ [size] ' [base] [value]

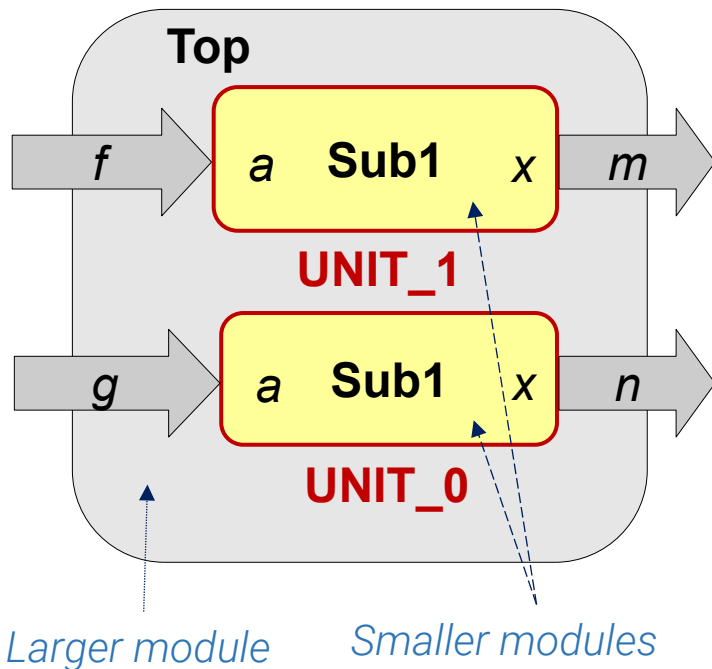
number	stored value	comment
5'b11010	11010	_ ignored
5'b11_010	11010	
5'o32	11010	
5'h1a	11010	
5'd26	11010	
5'b0	00000	0 extended
5'b1	00001	0 extended
5'bz	zzzzz	z extended
5'bx	xxxxx	x extended
5'bx01	xxx01	x extended
-5'b00001	11111	2's complement of 00001
'b11010	000000000000000000000000000011010	extended to 32 bits
'hee	000000000000000000000000011101110	extended to 32 bits
1	000000000000000000000000000000001	extended to 32 bits
-1	111111111111111111111111111111111	extended to 32 bits

Module Instantiation

- A digital system is usually composed of several smaller subsystems.
- Larger modules can be built by instantiating smaller modules in the body of the larger module.
- This modularity is especially helpful in creating larger designs which can be broken down into smaller and more manageable components.
- This type of HDL description is called structural modeling

Module Instantiation

- The **instance name** corresponds to a **unique name** for a **specific instance** of a certain module
 - Different names for different instances of the same module*



```
// Sub1 module
module Sub1 (a, x);
  input  a;
  output x;

  not N1 (x, a);
endmodule
```

*Module Declaration
of Sub1
(Sub1 is an existing
module)*

```
// Top module
module Top (f, m, g, n);
  input  f, g;
  output m, n;

  Sub1  UNIT_1 (f, m);
  Sub1  UNIT_0 (g, n);
endmodule
```

*Instantiate two
Sub1 modules*

Declaration vs. Instantiation

- Declaration

- Declaration of a Verilog module specifies input-output behavior of the hardware module it represents

- Instantiation

- Just a copy (*an instance*) of an existing (*smaller*) module to be reused in the larger module

*Module declaration of Top
(with module name, ports,
body)*

```
// Top module
module Top (f, m, g, n);
    input    f, g;
    output   m, n;

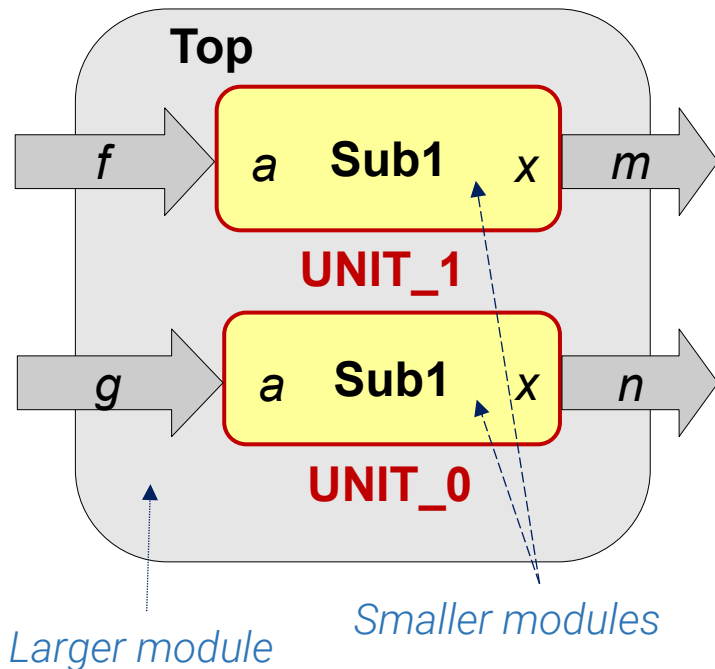
    Sub1    UNIT_1 (f, m);
    Sub1    UNIT_0 (g, n);

endmodule
```

*Module body with
two Sub1 module
instantiations
(UNIT_1 and
UNIT_0)*

Port Maps in Structural Modeling

- The **port mapping** describes the **connection of the ports of the smaller module** to internal signals or even the ports of the larger module.



```
// Sub1 module
module Sub1 (a, x);
    input    a;
    output   x;

    not N1 (x, a);
endmodule
```

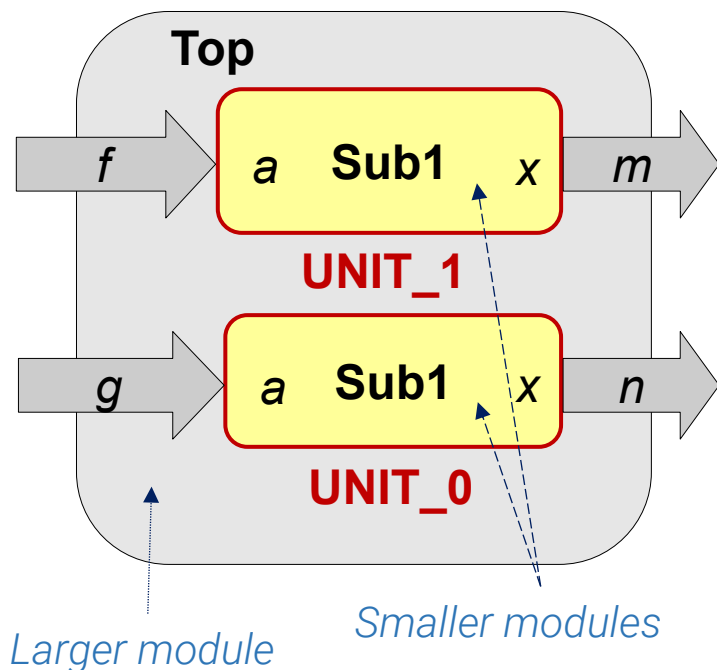
```
// Top module
module Top (f, m, g, n);
    input    f, g;
    output   m, n;

    Sub1    UNIT_1 (f, m);
    Sub1    UNIT_0 (g, n);
endmodule
```

*Port mapping along
with instantiation*

Implicit Port Maps

- Connection by ordered list (*connection by position*)
 - Associate the ports and external signals by **ordered list**
 - Port names of the smaller module (*sub1*) are omitted in the port map
 - Signals of the larger module (*top*) are listed in the same order as the smaller module's port declaration.



```
// Sub1 module
module Sub1 (a, x);
  input  a;
  output x;

  not N1 (x, a);
endmodule
```

*Port list of
smaller module*

```
// Top module
module Top (f, m, g, n);
  input  f, g;
  output m, n;

  Sub1  UNIT_1 (f, m);
  Sub1  UNIT_0 (g, n);
endmodule
```

*Same order as
smaller module's
port list*

Implicit port mapping

Wires as Internal Signals

- Example:

```
// Sub1 module
module Sub1 (a, x);
  input  a;
  output x;

  not N1 (x, a);
endmodule
```

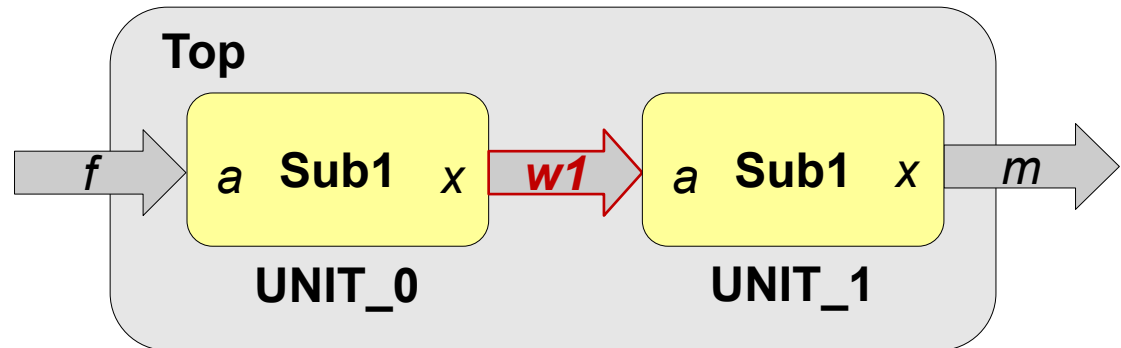
Port list of smaller module

```
// Top module
module Top (f, m);
  input  f;
  output m;
  wire  w1;

  Sub1  UNIT_1 (w1, m);
  Sub1  UNIT_0 (f, w1);
endmodule
```

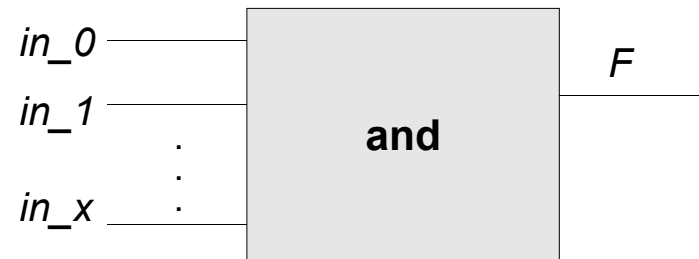
Same order as smaller module's port list

Implicit port mapping



Gate Primitives

- Verilog has **built-in modules** for modeling basic logic gates
 - Also known as **system primitives**: *and*, *nand*, *or*, *nor*, *xor*, *xnor*, *not*, *buf*
 - Primitives *not* and *buf* have *n* inputs and *n* outputs
 - The rest have *n*-input ports but with only 1 output
 - The **logic** of the primitives is based on the **four-value system** (0, 1, z, x)
- Their definitions are **predefined by Verilog**
- Will **NOT** be declared in structural modeling but **can be instantiated as modules**
- Uses **implicit port mapping**



- Instantiation example:

```
and [instance_name] (F, in_0, in_1, ..., in_x);
```

↖ *Output first in the port map*

Truth Table for Gate Primitives

- Truth table for the other gates is the same, except that the outputs are complemented.

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

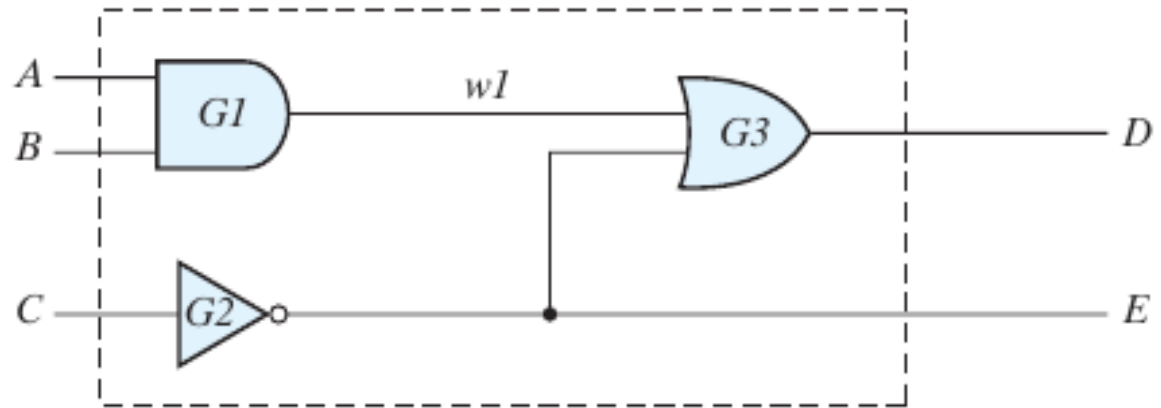
or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

not	input	output
	0	1
	1	0
	x	x
	z	x

Gate Primitives

- Example:



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

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

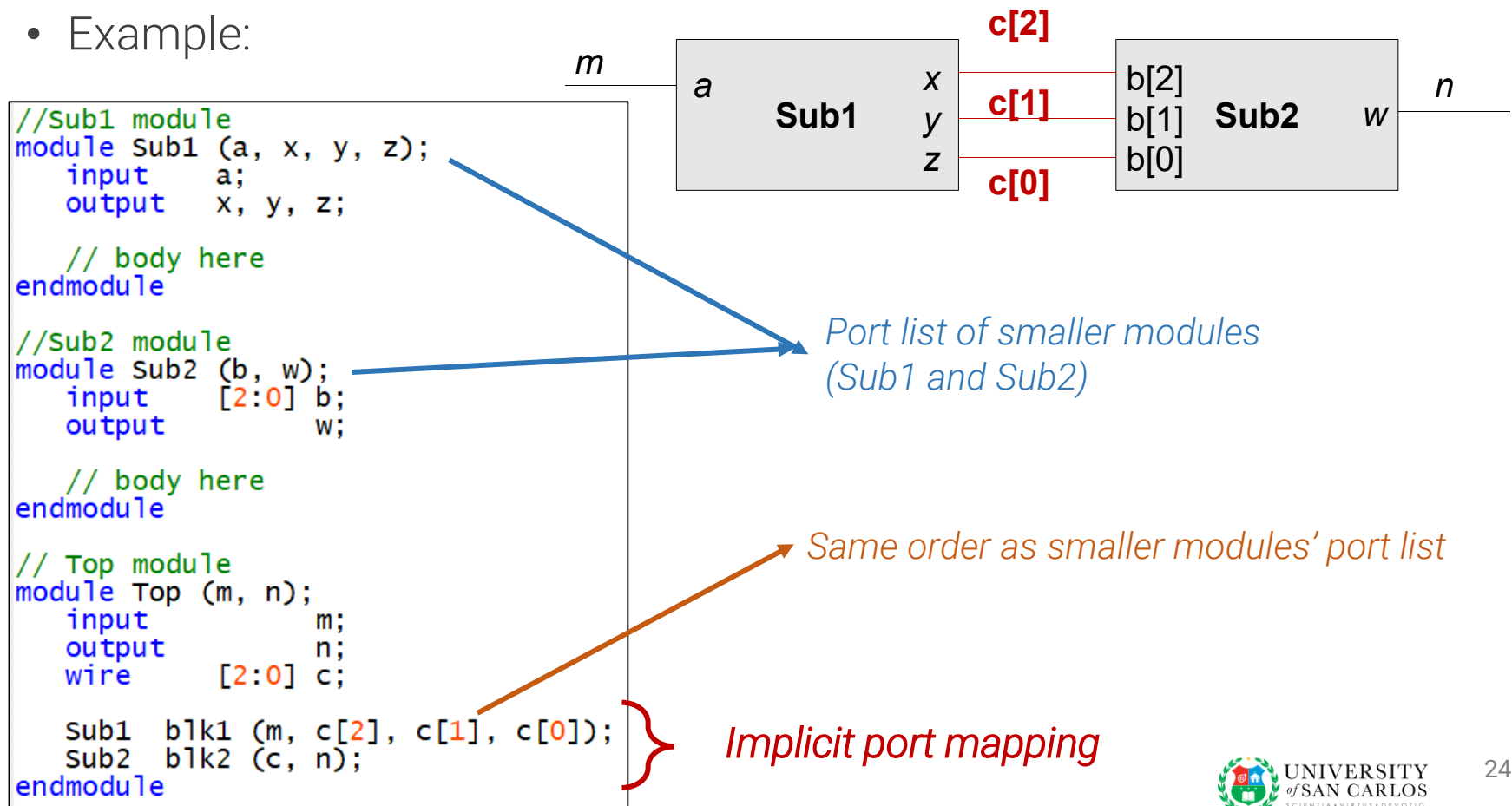
endmodule
```

*No declaration of gate primitives
(only instantiated)*

Output first in the implicit port map

Buses as Collection of Signals

- Buses are labeled with the **same name** and differentiated via an **index**.
- Example:



Explicit Port Maps

- Connection by name
 - As the number of ports increases, it may be tricky to keep track of port ordering when instantiating modules.
 - **Explicit port mapping** specifies which external signal is connected to a port.
 - Order of the port name and signal name pairs does not matter.
 - General form:

```
[module_name][instance_name]
    (
        .[port_name] ( [signal_name] ),
        .[port_name] ( [signal_name] ),
        ...
    );
```

Explicit Port Maps

```
[module_name] [instance_name]
(
    .[port_name] ( [signal_name] ),
    .[port_name] ( [signal_name] ), ...
);
```

- Example:

```
//Sub1 module
module Sub1 (a, x, y, z);
    input    a;
    output   x, y, z;

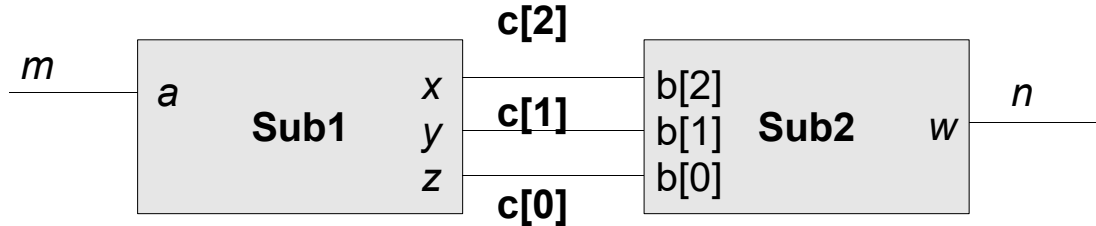
    // body here
endmodule

//Sub2 module
module Sub2 (b, w);
    input    [2:0] b;
    output   w;

    // body here
endmodule

// Top module
module Top (m, n);
    input    m;
    output   n;
    wire     [2:0] c;

    Sub1    blk1 (m, c[2], c[1], c[0]);
    Sub2    blk2 (c, n);
endmodule
```



```
// Top module with explicit port map
module Top (m, n);
    input    m;
    output   n;
    wire     [2:0] c;

    Sub1    blk1 (
        .y (c[1]),
        .a (m),
        .x (c[2]),
        .z (c[0])
    );

    Sub2    blk2 (
        .b (c),
        .w (n)
    );
endmodule
```

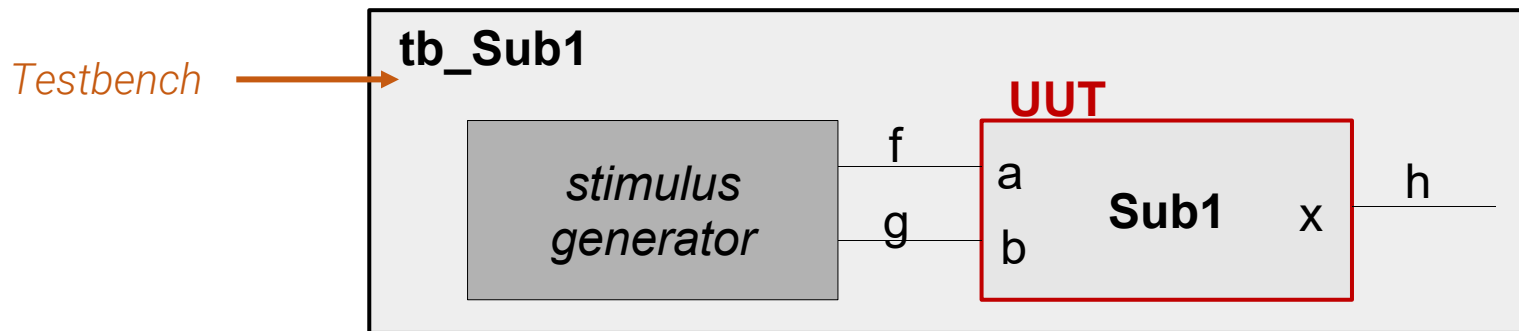
.self_port (external_wire)

Introduction to Logic Simulation

- After design entry (*HDL description is written*), it can be **simulated** in a host computer:
 - To **verify the behavior** of the circuit operation
 - **Predicts how the hardware will behave** before it is actually fabricated
 - **Detects functional errors** in a design without having to physically create and operate the circuit
- **Simulation** is usually performed within the same HDL framework
 - A **simulator** interprets the HDL description
 - Either **produces readable output** (*such as a time-ordered sequence of input and output signal values*), or **displays waveforms of the signals**

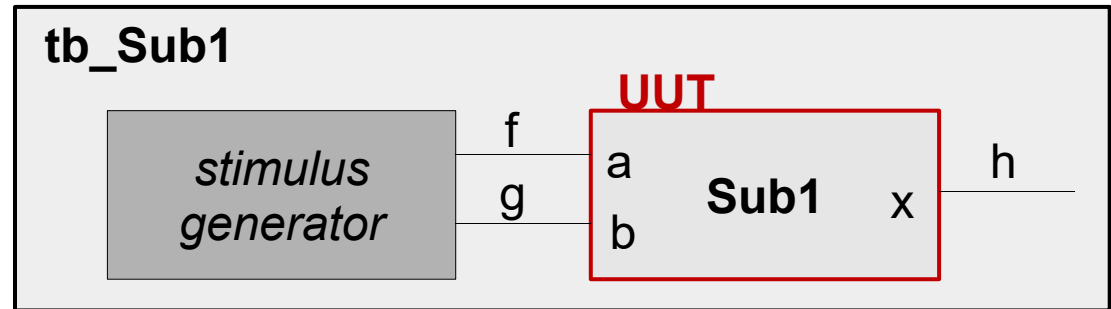
Testbench

- A special module **testbench** (also written in HDL) is created to test the HDL-based hardware module
 - Testbench has no ports (this won't be externally interfaced)
- The module to be tested is called the **Unit Under Test (UUT)**
 - UUT will be instantiated in the testbench module
 - UUT will be supplied with inputs (*stimuli or test vectors*) so that the simulator can generate an output response



UUT Instantiation in a Testbench

- Within the testbench:
- No port list
- Inputs to the UUT are declared as **reg**
- Outputs of the UUT are declared as **wire**
- Instantiate UUT
- Connect signals and UUT ports through a port map (*implicit or explicit*)
- Generate stimuli



```
`timescale 1 ns / 1 ps
module tb_sub1 (); // Testbench module with no ports
    reg f, g; // all inputs to UUT are reg type
    wire h; // all outputs from UUT are wire type

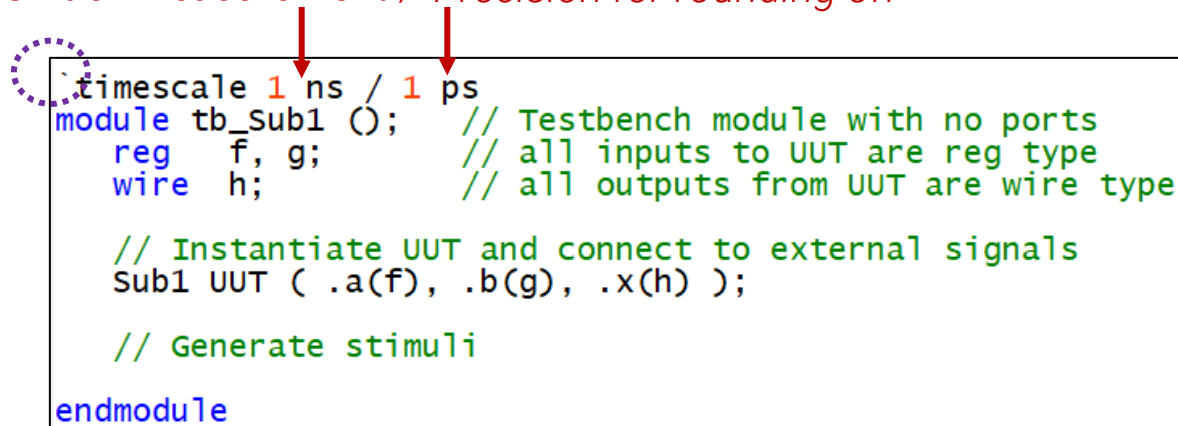
    // Instantiate UUT and connect to external signals
    Sub1 UUT ( .a(f), .b(g), .x(h) );

    // Generate stimuli
endmodule
```

Gate Delays

- All physical circuits exhibit **propagation delay** between an input and a resulting output
 - In Verilog, propagation delay of a gate is specified in terms of **time units** and by the **number symbol (#)**
 - Time delays in Verilog are **dimensionless**
 - These delays (#) should only be used in a testbench (*not synthesizable*)
- Association with physical time is made with **`timescale** compiler directive (starts with a *back quote* or *grave accent symbol*)

Unit of measurement / Precision for rounding off



```
`timescale 1 ns / 1 ps
module tb_Sub1 ();    // Testbench module with no ports
  reg  f, g;          // all inputs to UUT are reg type
  wire h;             // all outputs from UUT are wire type

  // Instantiate UUT and connect to external signals
  Sub1 UUT ( .a(f), .b(g), .x(h) );

  // Generate stimuli

endmodule
```

Stimulus Generator

- *Initial* keyword is used to generate input stimuli
 - Contains a set of statements (*block statements*) that is *evaluated sequentially* (by default, HDL statements are evaluated concurrently)
 - Block statements are enclosed by keywords *begin* and *end*
 - Runs only once
 - Should only be used in a testbench (*not synthesizable*)
- Gate delays (#) are used to separate statements in time
 - To hold stimuli values before changing them (*mimics propagation delay*)
- \$stop system task is used to terminate the simulation

*Block statement
generating stimuli
values*

```
// Generate stimuli
initial
begin
    f = 0;    g = 0;    #10
    f = 0;    g = 1;    #10
    f = 1;    g = 0;    #10
    f = 1;    g = 1;    #10

    //system task to end simulation
    $stop;
end
```

*Delays separate changes
in stimuli*

Introduction to ModelSim*-Intel® FPGA Starter Edition

- Perform Laboratory Exercise #2:



End of Unit 2