NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER ENGINEERING

LECTURE

Subject:

VERILOG
Hardware Description Language

Chapter2: Fundamental concepts

Lecturer: Lam Duc Khai

Agenda

1. Chapter 1: Introduction (Week1)
2. Chapter 2: Fundamental concepts (Week1)
3. Chapter 3: Modules and hierarchical structure (Week2)
4. Chapter 4: Primitive Gates – Switches – User defined primitives (Week2)
5. Chapter 5: Structural model (Week3)
6. Chapter 6: Behavioral model – Combination circuit (Week4)
7. Chapter 7: Behavioral model – Sequential circuit (Week5)
8. Chapter 8: Tasks and Functions (Week6)
9. Chapter 9: State machines (Week6)
10. Chaper 10: Testbench and verification (Week7)

Contents

Chapter 2: Fundamental concepts

- Lexical conventions
- Data types
- Verilog in Hardware Design

Lexical conventions

- **Comments :**

  - Two forms to introduce comments:

    1. //…….//
    2. /*……..*/

Ex:

```
module FlipFlop (din, clk, qout);
        input din, clk;
        output qout;
        reg qout;
// At the rising edge of clk, qout <= din  //
/* At the rising edge of clk, qout <= din  */
        always @ (posedge clk) begin
                qout <= #8 din;
endmodule
```

Lexical conventions

- **Numbers :**

  - Two forms to express numbers:

    1. 37 — 32 bit decimal 37
    2. <size>'<base_format><number>

Ex:

10'hFA	10 bits hexadecimal number FA (00_1111_1010)
1'b0	1 bit binary number 0 (0)
6'd30	6 bits decimal number (011110), decimal 30
15'o10752	15 bits octal number (001,000,111,101,010), decimal 4586
	4'b0 is equal to 4'b0000
4'b1 is equal to 4'b0001	
4'bz is equal to 4'bzzzz	
4'bx is equal to 4'bxxxx	
-8 'd 6	The two's complement of 6, held in 8 bits

Lexical conventions

- **Strings :**

  -A string is a sequence of characters enclosed by double quotes and all contained on a single line. Verilog treats strings used as operands in expressions and assignments as a sequence of eight-bit ASCII values, with one eight-bit ASCII value representing one character.

Ex:

"Hello world"

- **String variable declaration:**

  - To store the string "Hello world" requires a register 8*11, or 88 bits wide:

```
reg [8*11:1] stringvar;
initial
begin
        stringvar = "Hello world";
end
```

Lexical conventions

- **Strings manipulation**:

  - It can be manipulated with strings.

Ex:

```
module string_test;
  reg [8*11:1] stringvar;
  initial
  begin
        stringvar = “Hello”;
        $display(“%s is stored as %h”,stringvar,stringvar);
        stringvar = {stringvar,” world”};
        $display(“%s is stored as %h”,stringvar,stringvar);
  end
endmodule
```

Lexical conventions

- **Special character in string**

Escape string	Character produces by escape string
\n	New line character
\t	Tab character
\\	Slash (\) character
*	Double quote (*) character
\ddd	A character specified in 1-3 octal digits (0 <= d <= 7)
%%	Percent (%) character

Lexical conventions

- **Keywords** :

  - Keywords are used to define the language constructs. There are a lot of keywords in Verilog HDL. (Refer to Verilog books)
  - All keywords are defined in lower case
  - Do not use keywords as user's definition.

Examples :

module, endmodule	fork, join
input, output, inout	specific, endspecific
reg, integer, real, time	timescale
not, and, nand, or, nor, xor	include
parameter	undef
begin, end	nmos, pmos,…

Lexical conventions

- **System tasks and functions :**

  – They are considered part of the Verilog HDL. These system tasks and functions are divided into some categories as follows:

  + Display tasks : $display, $monitor, $strobe, $writ, $dumpfile, $dumpvars…

  + File I/O tasks : $fclose, $fdisplay, $swrite, $fread, $sdf_annotate, $readmemb, $readmemh…

  + Simulation control tasks: $finish, $stop

  + Math functions: $ln, $log10, $exp, $sqrt, $sin, $cos, $asin, $acos…

Ex:

```
$display (“Hello world”);
$finish;
```

Lexical conventions

- **System tasks and functions** (cont'd)
  - $time - returns the current simulation time
  - $display - similar to printf in C
  - $stop - stops simultion
  - $finish - ends simulation
  - $monitor – monitor simulation
  - $readmemh - load memory array from text file in hex format
  - Many more …

Lexical conventions

Compiler directives

- The scope of a compiler directive extends from the point where it is processed, across all files processed, to the point where another compiler directive supersedes it or the processing completes.
- There are some common compiler directives:
  + `celldefine
  + `endcelldefine
  + `define
  + `else
  + `elsif
  + `ifdef
  + `endif
  + `include
  + …

Lexical conventions

- **Compiler directives**
  - Example

```
`include file_name // include source code from another file
`define macro_name macro_code // substitute macro_code for macro_name
`define macro_name(par1, par2,...) macro_code // parameterized macro
`undef macro_name // undefine a macro
`ifdef macro_name1 // include source lines1 if macro_name1 is defined
        <source lines1> // the source lines1
`elsif macro_name2 // any number of elsif clauses, the first defined
        <source lines2> // macro_name includes the source lines
`else // include source lines3 when no prior macro_name defined
       <source lines3> // the source lines 3
`endif // end the construct
`ifndef macro_name // like `ifdef except logic is reversed,
                   // true if macro_name is undefined
`timescale 1ns/1ns // units/precision for time e.g. #5 : 5 ns, #5.455: 5.455 ns, #5.4557: 5.456 ns
`celldefine // marks beginning of a cell
`endcelldefine // marks end of a cell
```

Lexical conventions

Module 1 (with directive)	Module 2 (without directive)

`timescale 1 ns / 10 ps
 	
module mod1 (set);
 	
output set;
reg set;
parameter d = 1.55;
 	
initial
begin
set = 1'bz;
#d set = 1'b0;
#d set = 1'b1;
end
 	
endmodule
	module mod2 (set);
	
	output set;
	reg set;
	parameter d = 1.55;
	
	initial
	begin
	set = 1'bz;
	#d set = 1'b0;
	#d set = 1'b1;
	end
	
	endmodule

Case 1 — Run the **vsim** command in the following order:

vsim mod2 mod1

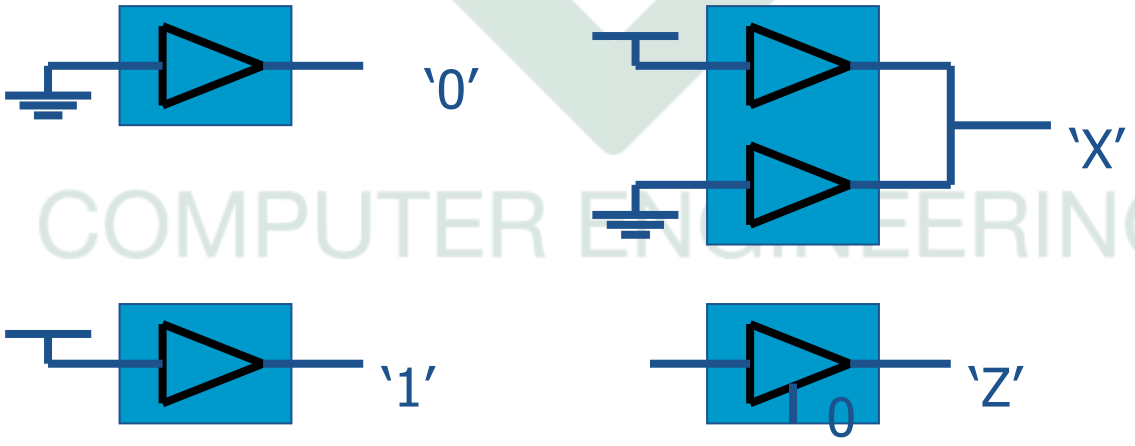
Module 1 sets the simulator resolution to 10 ps. Module 2 has no timescale directive, so the time units default to the simulator resolution, in this case 10 ps. If you looked at */mod1/set* and */mod2/set* in the Wave window, you would see that Module 1 transitions every 1.55 ns as expected (because of the 1 ns time unit in the timescale directive). However, in Module 2, *set* transitions every 20 ps. That is because the delay of 1.55 in Module 2 is read as 15.5 ps, which is rounded up to 20 ps.

Data types

- **Value set :**

  - Four basic values:

    1. 0 – represents a logic zero, logic low, ground or false condition.
    2. 1 – represents a logic one, logic high, power or true condition.
    3. x – represents an unknown logic value.
    4. z – represents a high-impedance, unconnected, tri-state.



Data types

➢ Nets data types

- Nets data types present the physical connections between devices. A net does not store a value, it must be driven by a gate or continuous assignment. If a net variable has no driver, then it has a high-impedance value (z).

- Net data type can be declared by following keywords : wire, wand, wor, supply0, supply1, …
- Cannot be assigned in an *initial* or *always* block
- *Net* data type represent physical connections between structural entities.
- A *net* must be driven by a driver, such as a gate or a continuous assignment.
- Verilog automatically propagates new values onto a *net* when the drivers change value.

Ex:

- wire w1, w2; // declares 2 wires
- wand w;

Data types

➢ Net data types (cont'd)

- Used in structural modeling and continuous assignment
- Types of nets:
  - wire, tri : default
  - wor, trior : wire-ORed
  - wand, triand : wire-ANDed
  - trireg : with capacitive storage
  - tri1 : pull high
  - tri0 ; pull low
  - supply1 ; power
  - supply0 ; ground

Data types

Registers

- Registers present abstract storage elements. Registers are data types that hold the assigned values until a new value assigned to it. A new value can be assigned to registers only by using procedural assignments.
- Register data type can be declared by using “reg” keyword.
- Don’t confuse reg assignments with the combinational continuous assign statement! (more soon)
- Reg should only be used with always blocks (sequential logic)

Ex:

- reg a; // a scalar register
- reg[3:0] v; // a 4-bit vector register made up of (from most to least significant) v[3], v[2], v[1] and v[0]
- reg [1:4] b; // a 4-bit vector register
- reg signed [0:3] signed_reg; // 4-bit signed register with a range of -8 to +7
- reg signed [0:3] signed_mem [99:0] // 100 words with a range of -8 to +7

Data types

- **Register (cont'd)**
  - A variables used in behavioral description
  - A storage device or a temporary variable
  - Types of register:
    - reg : unsigned integer variables of varying bit width
    - integer : 32-bit signed integer
    - real : signed floating-point
    - time : 64-bit unsigned integer

Data types

➢ Variable declarations

- **Declaring a net**
  wire [<range>] <net_name> [<net_name>*];
  Range is specified as [MSb:LSb]. Default is one bit wide

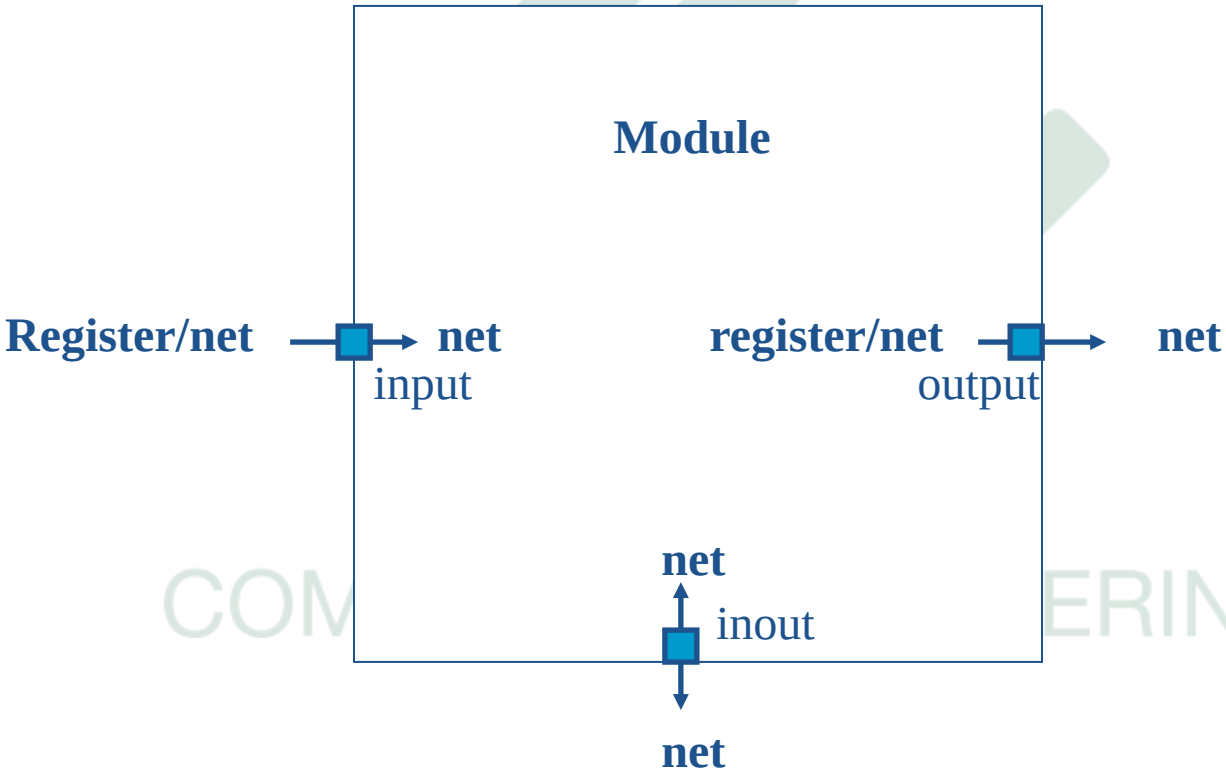
- **Declaring a register**
  reg [<range>] <reg_name> [<reg_name>*];

- **Declaring memory**
  reg [<range>] <memory_name> [<start_addr> : <end_addr>];

- **Examples**
  reg r; // 1-bit reg variable
  wire w1, w2; // 2 1-bit wire variable
  reg [7:0] vreg; // 8-bit register
  reg [7:0] memory [0:1023]; a 1 KB memory

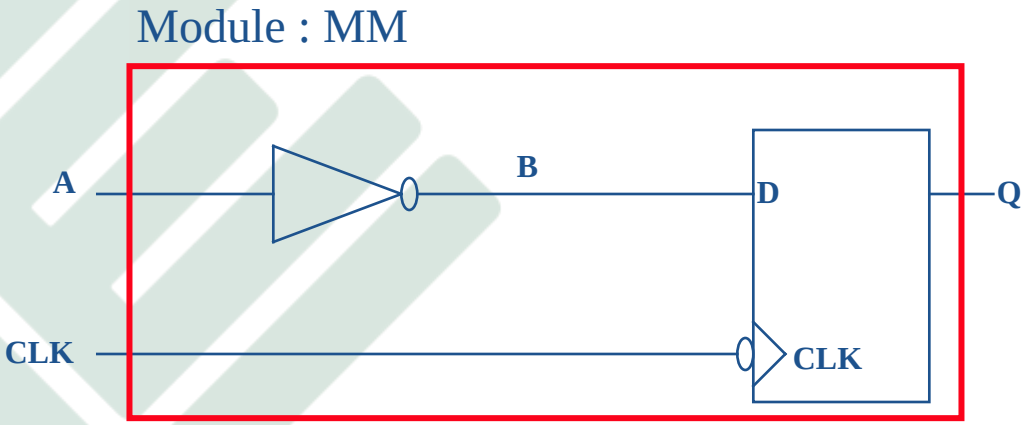
Port and Data types

- Correct data types for ports



Data types

- Example




```
module MM (Q,A,CLK) ;
output Q ;
input A,CLK;
wire B
reg Q ;
      assign B = ! A;
      always @ (negedge CLK)
                  Q <= B;
endmodule
```

Data types

- **Signed objects**:

Ex:

wire signed [3:0] signed_wire; // range -8 <-> +7

reg signed [3:0] signed_reg; // range -8 <-> +7

reg signed [3:0] signed_mem [99:0] // 100 words range -8 <-> +7

- A signed value will not cross hierarchical boundaries. If you want a signed value in other modules in a hierarchy, you must declare them in each of the modules where signed arithmetic is necessary.

Data types

- **Scalar and Vector**:
  - A net or reg declaration without a <*range*> specification is one bit wide; that is, it is scalar.Multiple bit net and reg data types are declared by specifying a <*range*>, and are known as vectors.

Ex:

```
wire w1, w2; // declares 2 scalar nets
reg a; // declares a scalar register
wire [3:0] addr; // declare a vector net
reg[3:0] v; // declare a vector register
```

- Vector can be declared at *[high# : low#]* or *[low# : high#]*, but the left number in the squared brackets is always the most significant bit of the vector
- Vectors can be declared only for nets and reg data types. (Vector declaration for integer, real, realtime, and time data types are illegal.)

Data types

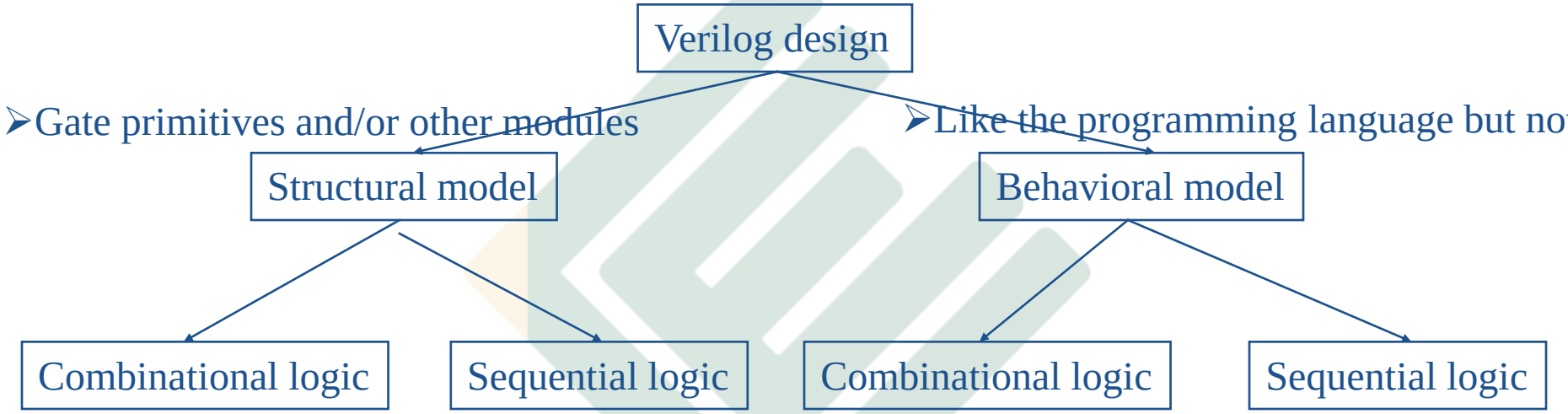
➢ Parameters:

- Parameters are constants. There are two types of parameters: module parameters and specify parameters.
- Parameters are not variables, they are used for “per instance” constants.
- For global constants, using ‘define …

+ Module parameters:

– **parameter** msb = 7; // defines msb as a constant value 7
– **parameter** e = 25, f = 9; // defines two constant numbers
– **parameter** r = 5.7; // declares r as a **real** parameter
– **parameter** byte_size = 8,
– **parameter** average_delay = (r + f) / 2;
– **parameter signed** [3:0] mux_selector = 0;
– **parameter real** r1 = 3.5e17;

Verilog in Hardware Design



- **Combinational Logic**

  ➢Logic without state variables
  ➢Examples: adders, multiplexers, decoders, encoders
  ➢No clock involved

- **Sequential Logic**

  ➢Logic with state variables
  ➢State variables: latches, flip-flops, registers, memories
  ➢Clocked
  ➢State machines, multi-cycle arithmetic, processors

END