Digital System Design with HDL (I)
Lecture 2

Dr. Ming Xu
Dept of Electrical & Electronic Engineering
XJTLU

In This Session

- Lexical Conventions
- Data Type Declarations

Lexical Conventions

White Space Characters

SPACE, TAB, and blank lines are ignored.

Comments

// begins a single line comment, terminated by a newline.

/* begins a multi-line comment, terminated by a */.

Case Sensitivity

Verilog is case sensitive.

Statement/Declaration Termination

with semicolon "**;**"

Lexical Conventions

Identifiers

Identifiers are the names of variables and elements

- Begin with a letter (a-z, A-Z) or an underscore _ .
- Composed of a sequence of
  - letters
  - digits (0 to 9)
  - underscore _ and $ symbol

Lexical Conventions

Logic Values

The Verilog HDL has **four** logic values:

- **0** = logic value 0
- **1** = logic value 1
- **z, Z, or ?** = tri-state (high impedance or floating)
- **x** or **X** = unknown or uninitialized

Lexical Conventions

Logic Values

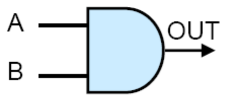
Why **x**?

- Could be drivers conflicting to a wire.
- Could be lack of initialization of a register
- Output of a gate with z inputs

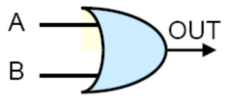
Why **z**?

- Nothing driving the signal (Tri-states)

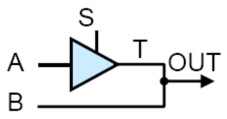
Logic Values



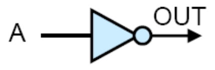
A	B	OUT
0	0	0
0	1	0
1	1	1
0	x	0
0	z	0
1	x	x
1	z	x



A	B	OUT
0	0	0
0	1	1
1	1	1
0	x	x
0	z	x
1	x	1
1	z	1



S	A	T	B	OUT
0	0	z	z	z
0	1	z	x	x
0	x	z	1	1
0	z	z	0	0
1	0	0	1	x
1	0	0	z	0
1	1	1	z	1
1	x	x	z	x
1	z	x	0	x



A	0	1	x	z
OUT	1	0	x	x

Lexical Conventions

Numbers

Representation: **[size] ['radix] value**

size

- number of **BITS** regardless of radix used
- **default** to at least 32 …

- radices
  - decimal (d or D) – **default** if no base specified!
  - hexadecimal (h or H)
  - octal (o or O)
  - binary (b or B)

Lexical Conventions

Numbers

- An _ (underscore) is ignored (used to enhance readability).
- When size is larger than value and the left-most bit of value is 0 or 1, zeros are left-extended to fill the size.
- If the left-most bit of value is X, the X is left-extended to fill the size.
- If the left-most bit of value is Z, the Z is left-extended to fill the size.

Lexical Conventions

Numbers
Examples

Number	Decimal	Binary
4’d3	3	0011
8’ha	10	00001010
5’b111	7	00111
8’b0101_1101	93	01011101
8’bx1101		xxxx1101
10	10	0…01010 (32 bits)

Data Type Declarations

Types

- *net*: represents interconnections between structural entities such as gates.
- *register*: stores a value from one assignment to the next.
- *memory: an array of vectors.*
- *parameter:* defines constants.

Data Type Declarations

Net Data Types

- A *net* represents a node in a circuit.
- Doesn’t store value, just a connection
- Input, output, inout are default “wire”
- Used when a signal is on the left-hand side of a continuous assignment.

net_type [size] *net_name* , *net_name* , ... ;

size: the range of [*msb* : *lsb*]

Data Type Declarations

Net Data Types

net_type:

- **wire:** used to connect an output of one logic element to an input of another logic element.

  Examples: wire x;

- **tri:** used for tri-state circuit nodes.

  Examples: tri [7:0] DataOut;

Data Type Declarations

Register Data Types

- A *register* is an abstraction of data storage element, which stores a value from one assignment to the next.
- Storage element (modeling sequential circuit)
- Assignment in "always" block

register_type [size] *variable_name* , *variable_name* , ... ;

size: the range of [*msb* : *lsb*]

Data Type Declarations

Register Data Types

register_types:

- **reg:** unsigned variable of any bit size to be defined
- **integer:** signed 32-bit variable

Examples:

```
reg [2:0] Count;   // 3-bit unsigned variable
integer k;         // 32-bit signed variable
```

Data Type Declarations

Register Data Types

- The original Verilog used *register* to refer to **reg** and **integer** types, which has been replaced by *variable* in Verilog 2001.
- Do not confuse the term *registers* in Verilog with hardware registers built from edge triggered flip-flops.
- In Verilog, the term *register* only means a variable that can hold a value until another value is placed onto it.

Data Type Declarations

Memory Data Types

A memory is a one-dimensional array of registers.

register_type [size] ***memory_name*** **[array_size];**

array_size: the range of [first_address : last_address]; either ascending or descending address order may be used.

Examples:

```
reg [7:0] R [3:0];        // declare four 8-bit variables
R[3]                      // access the individual variable
R[3][7]                   // access the left-most bit of R[3]
```

Data Type Declarations

Parameter Data Types

A **parameter** associates an identifier name with a constant.

parameter ***constant_name*** **=** ***value*****,** ***constant_name*** **=** ***value*****, ... ;**

Example: An n-bit adder

```
module addern (carryin, X, Y, S);
    parameter n = 32;
    input carryin;
    input [n-1:0] X, Y;
    output reg [n-1:0] S;

    always @(X, Y, carryin)
            S = X + Y + carryin;
endmodule
```