Digital Design with Verilog

Verilog

Lecture 5: Basic Concepts, Modules, and Ports

Learning Objectives

- Understand lexical conventions for
  - Operators,
  - Comments,
  - Whitespace,
  - Numbers,
  - Strings, and Identifiers.

- Define the logic value set and data types such as
  - nets,
  - registers,
  - vectors,
  - numbers,

Learning Objectives …Cont’d

- Simulation time, arrays, parameters, memories, and strings.
- Identify useful system tasks for displaying and monitoring information, and for stopping and finishing the simulation.
- Learn basic compiler directives to define macros and include files.

Lexical Conventions

Lexical Conventions

- Similar to C
  - Contains steam of tokens: comments, delimiters, numbers, strings, identifiers, and keywords.
- Whitespace
  - Blank spaces, tabs, and newlines
  - \b, \t, \n
- Whitespace ignored except in
  - Strings and token separation
- Comments
  - // -single line; /* …*/ -multiple lines

Operators and Number Specification

- Operators
  - Unary, binary and ternary
  - A = ~b; a = b&&c; a = b?c : d;

- Number specification
  - Sized numbers: <size>’<base format><number>
  - <size> -number of bits
  - <size> is specified only in decimal
  - 4’b1111; 12’habc

- Unsized numbers
  - 23456 *// default is decimal*
  - ‘hc3 *// hexadecimal number*
  - ‘o21 *// octal number*

Possible Values

- X or Z values
  - 12’h13x; 6’hx; 32’bz;
  - x or z sets 1,3,4 bits in binary, octal and hexadecimal representations respectively

- Value extension
  - If most significant bit is
    - 0,x,z then 0,x,z respectively
    - 1 then 0

- Negative numbers
  - Negative numbers can be specified by putting a minus sign before the size for a constant number.
    - -6’d3 //negative number stored as 2’s complement of 3
    - 4’d-2 // illegal specification

Underscore Characters, Question Marks

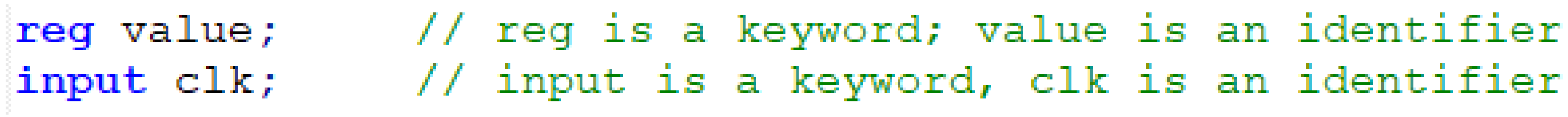
- Improving readability
  - "_" is allowed anywhere in the number, except the first character.
    - 12'b1111_1110_0101 *//Underscore*
  - "?" is used to enhance the readability in the **casex** and **casez** statements.
    - 4'b10?? *// Equivalent of a 4'b10zz*

Strings

- Strings
  - A string is a sequence of characters that are enclosed by double quotes.
  - It must be contained on a single line.
  - Strings are treated as a sequence of one-byte ASCII values.
    - “Hello Verilog World” *// is a string*
    - “a / b” *// is a string*

Keywords

- Keywords are special identifiers reserved to define the language constructs.
  - Keywords are in lowercase.



```
reg value;      // reg is a keyword; value is an identifier
input clk;      // input is a keyword, clk is an identifier
```

Identifiers

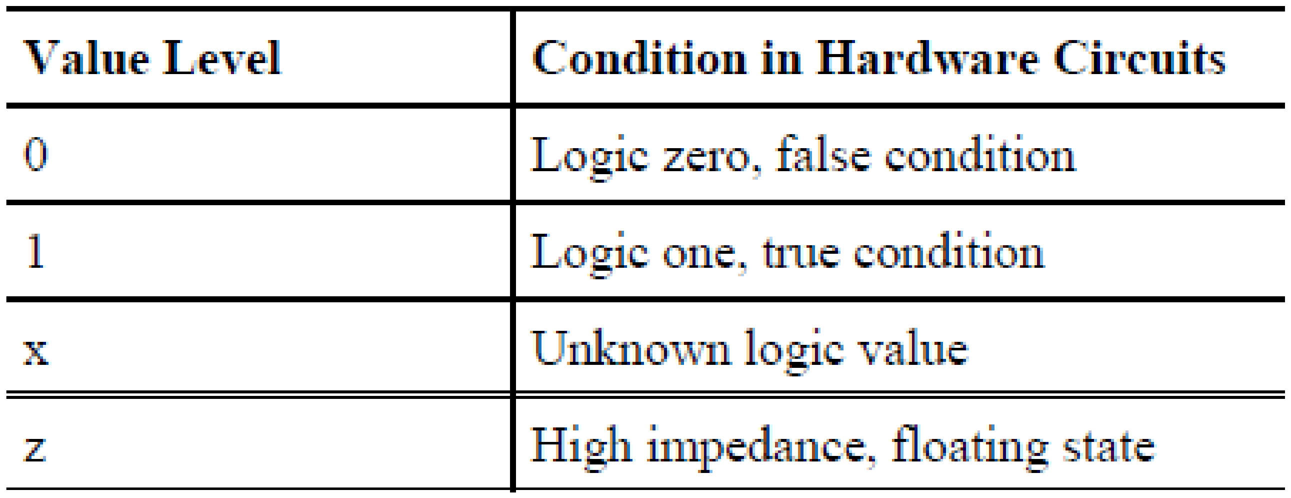
- Identifiers are names given to objects so that they can be referenced in the design.
  - Identifiers made of alphanumeric characters, the underscore (_) and the dollar ($) sign and starts with alphabets or underscore.
  - Identifiers are case sensitive.
  - The dollar sign as first character is reserved for system tasks.
    - Ex : $monitor

Data Types

Value Set

- Verilog support four values and eight strengths to model the functionality of real hardware.

- The four value levels are:



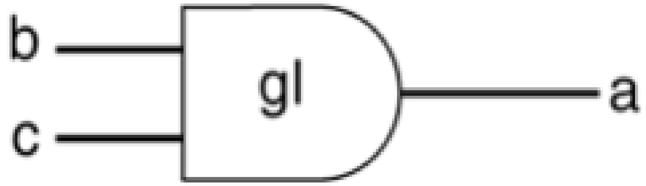
Value Level	Condition in Hardware Circuits
0	Logic zero, false condition
1	Logic one, true condition
x	Unknown logic value
z	High impedance, floating state

  - signal strength----discussed later.

Nets

- Represent connection between hardware elements.

- Declared by keyword **wire,** by default one-bit.
    - wire a;
    - wire b, c;
    - wire d = 1’b0;
    - wand, wor, tri, triand, trior, and trireg are other forms of net.
    - **Wire and net often used interchangeably.**



Registers

- Represent data storage elements
  - Retain value until another value is placed onto them.

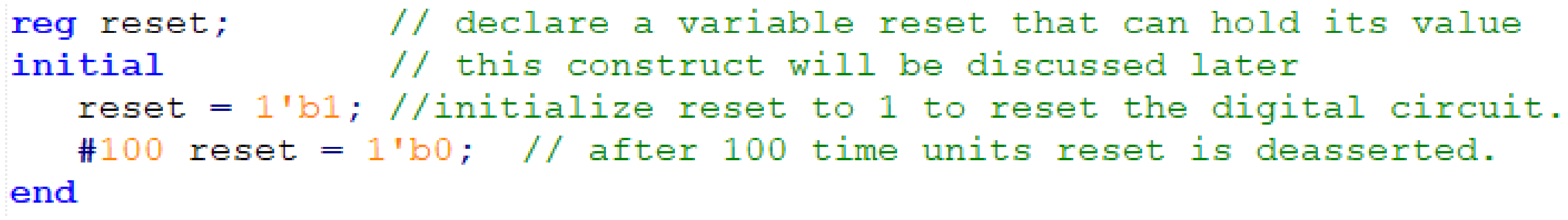
  - Do not confuse the term registers in Verilog with hardware registers built from edge-triggered flip-flops in real circuits.

- A variable that stores value
  - Unlike a net, it needs no driver

- Default value for a ‘reg’ variable is x

- Declared using keyword **reg**

Registers



```
reg reset;       // declare a variable reset that can hold its value
initial          // this construct will be discussed later
   reset = 1'b1; //initialize reset to 1 to reset the digital circuit.
   #100 reset = 1'b0;  // after 100 time units reset is deasserted.
end
```

Vectors

- Nets or reg data types can be declared as vectors (multiple bit widths). If bit width is not specified, the default is scalar (1-bit).
  - wire [7:0] busA, busB;
  - reg [16:0] address;
- [high#:low#] or [low#:high#], but the most significant bit is the left number in the square bracket
  - reg [0:40] addr1; wire [15:12] addr2

Vectors: Part Select

- reg[7:0] busA;
  - busA[7]; //stands for 7thbit
  - busA[2:0]; // first 3 LSBs

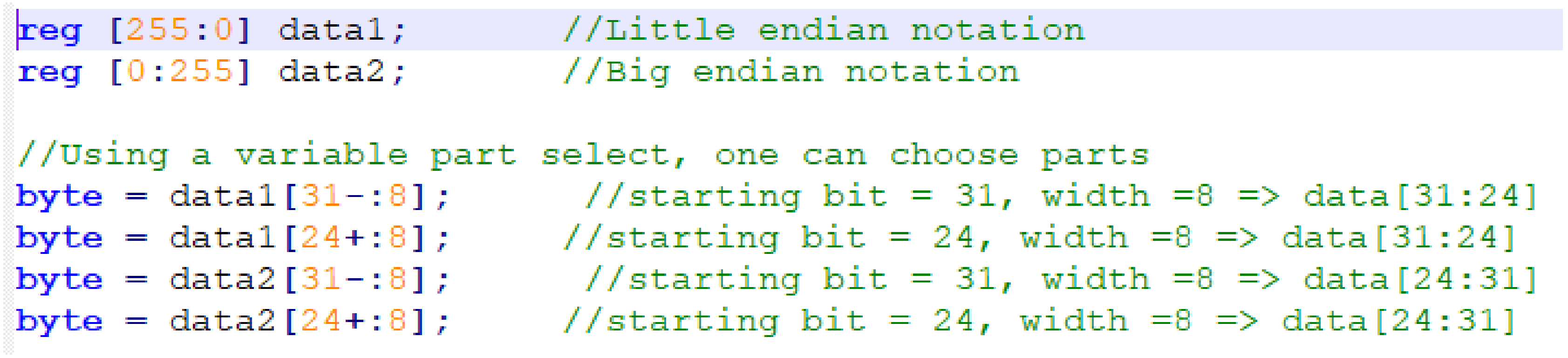
- wire [0:15] addr1;
  - addr1[2:1] is illegal;
  - addr1[1:2] is correct;

- Writing to a vector :
  - busA[2:0] = 3’d6;

- Variable Part Select

Vectors: Part Select (Contd.)

[<starting_bit>+:width] - *part-select increments from starting bit*
[<starting_bit>-:width] - *part-select decrements from starting bit*



```
reg [255:0] data1;          //Little endian notation
reg [0:255] data2;          //Big endian notation

//Using a variable part select, one can choose parts
byte = data1[31-:8];         //starting bit = 31, width =8 => data[31:24]
byte = data1[24+:8];        //starting bit = 24, width =8 => data[31:24]
byte = data2[31-:8];         //starting bit = 31, width =8 => data[24:31]
byte = data2[24+:8];        //starting bit = 24, width =8 => data[24:31]
```

System Tasks and Compiler Directives

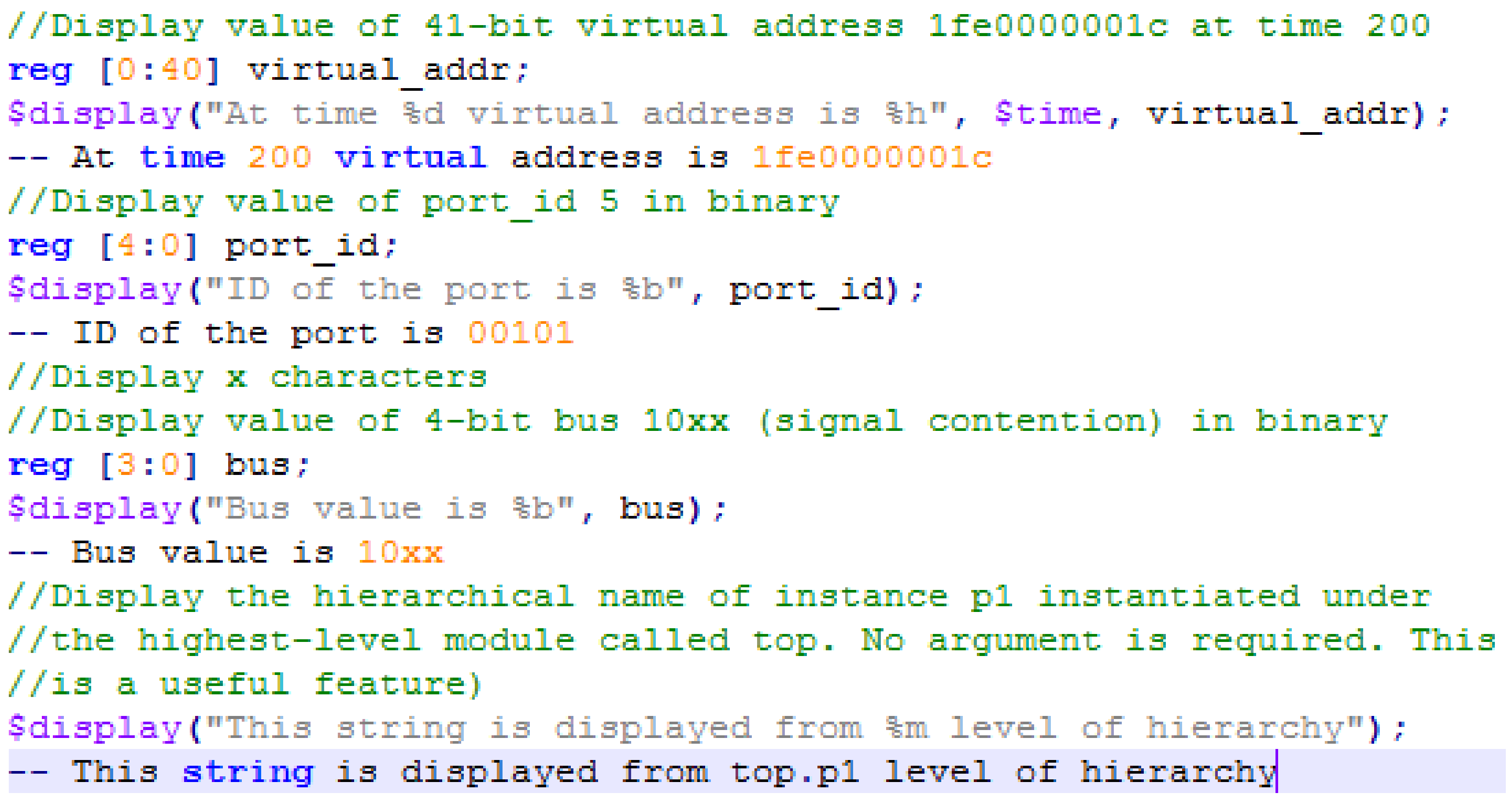
System Tasks

- Verilog provides standard system tasks for certain routine operations.
  - All system tasks appear in the form $<keyword>.
  - Operations such as displaying on the screen, monitoring values of nets, stopping, and finishing are done by system tasks.

System Tasks …cont’d

- $display
  - Like printf and used for displaying values of variables and expressions
  - Values can be printed in decimal (%d), binary (%b), string (%s), hex (%h), ASCII character (%c), octal (%o), real in scientific (%e), real in decimal (%f), real in scientific or decimal whichever is shorter(%g).
  - A $display inserts a newline at the end of the string by default.
  - A $display without any arguments produces a newline.

$display



```
//Display value of 41-bit virtual address 1fe0000001c at time 200
reg [0:40] virtual_addr;
$display("At time %d virtual address is %h", $time, virtual_addr);
-- At time 200 virtual address is 1fe0000001c
//Display value of port_id 5 in binary
reg [4:0] port_id;
$display("ID of the port is %b", port_id);
-- ID of the port is 00101
//Display x characters
//Display value of 4-bit bus 10xx (signal contention) in binary
reg [3:0] bus;
$display("Bus value is %b", bus);
-- Bus value is 10xx
//Display the hierarchical name of instance p1 instantiated under
//the highest-level module called top. No argument is required. This
//is a useful feature)
$display("This string is displayed from %m level of hierarchy");
-- This string is displayed from top.p1 level of hierarchy
```

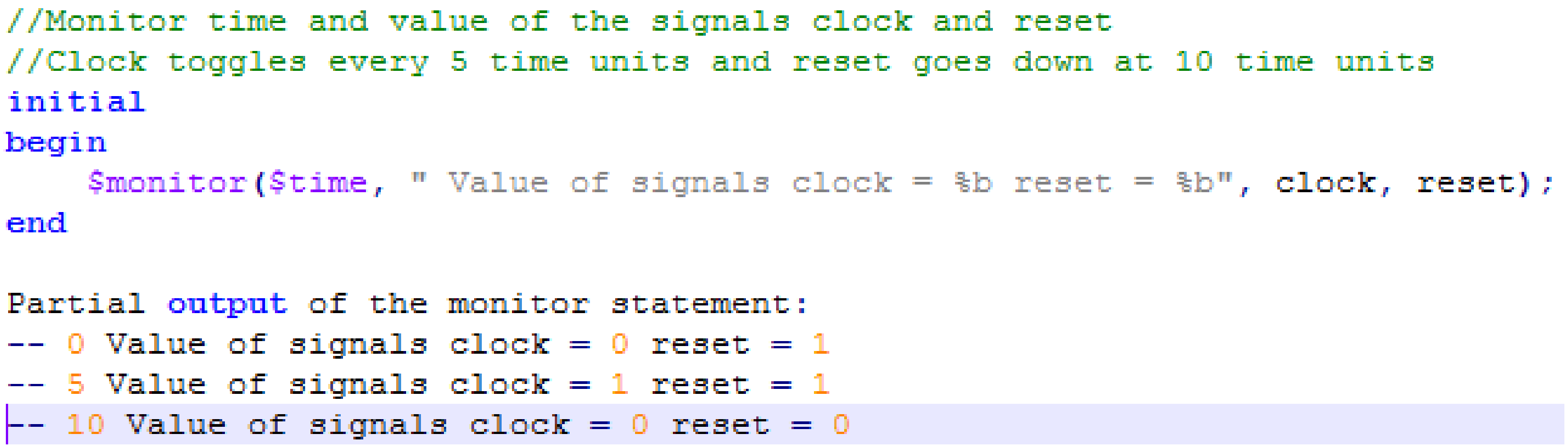
System Tasks

- $monitor
  - Monitor a signal when its value changes.
  - Only one active monitor list.
  - Format similar to the $display task is used.

- If more than one then, the last one statement will be the active one.

- Monitoring turned ON by default at start of simulation and can be controlled during the simulation using $monitoron and $monitoroff.

$monitor

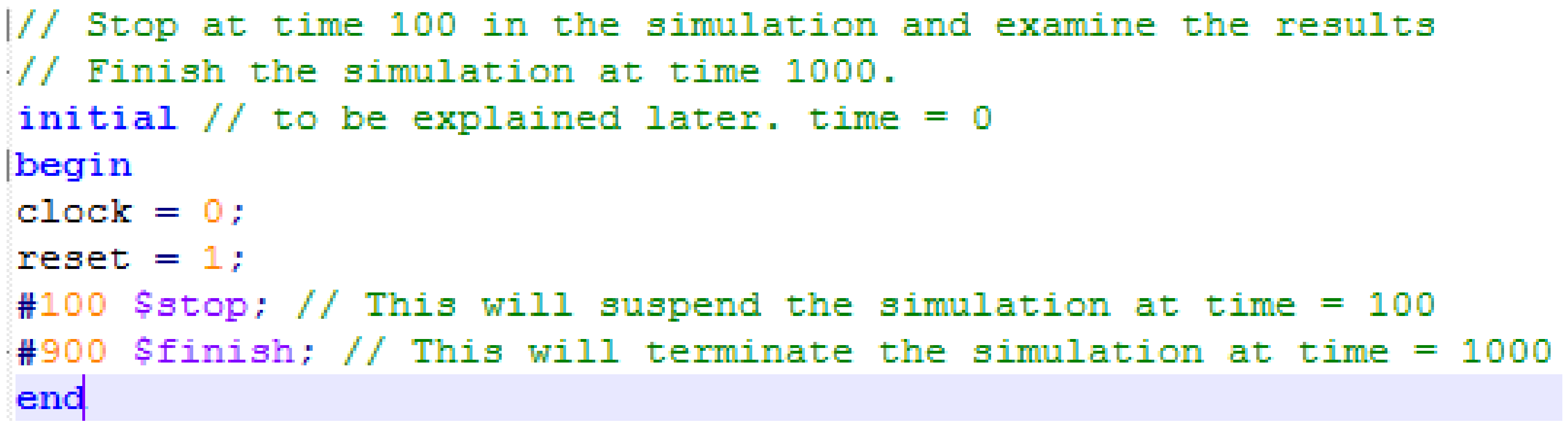


```
//Monitor time and value of the signals clock and reset
//Clock toggles every 5 time units and reset goes down at 10 time units
initial
begin
    $monitor($time, " Value of signals clock = %b reset = %b", clock, reset);
end

Partial output of the monitor statement:
-- 0 Value of signals clock = 0 reset = 1
-- 5 Value of signals clock = 1 reset = 1
-- 10 Value of signals clock = 0 reset = 0
```

System Tasks

- $stop – stops simulation and puts it into interactive mode
- $finish – terminates the simulation



```
// Stop at time 100 in the simulation and examine the results
// Finish the simulation at time 1000.
initial // to be explained later. time = 0
begin
clock = 0;
reset = 1;
#100 $stop; // This will suspend the simulation at time = 100
#900 $finish; // This will terminate the simulation at time = 1000
end
```

References

- Samir Palnitkar, “Verilog HDL”

- Reading:

  Chapter 3