

Introduction to Verilog

Programming Warm-up

Prepared by

Beig Rajibul Hasan

Lecturer, CSE, BRAC University

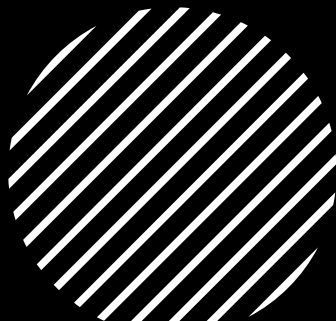
Contact: rajib.hasan@bracu.ac.bd





Verilog HDL

- Verilog HDL is a general-purpose **hardware description language which can describe the digital circuits** with C-like syntax.
- Most popular logic synthesis tools support Verilog HDL.
- Digital circuits can be described at the RTL of abstraction which ensures design portability.



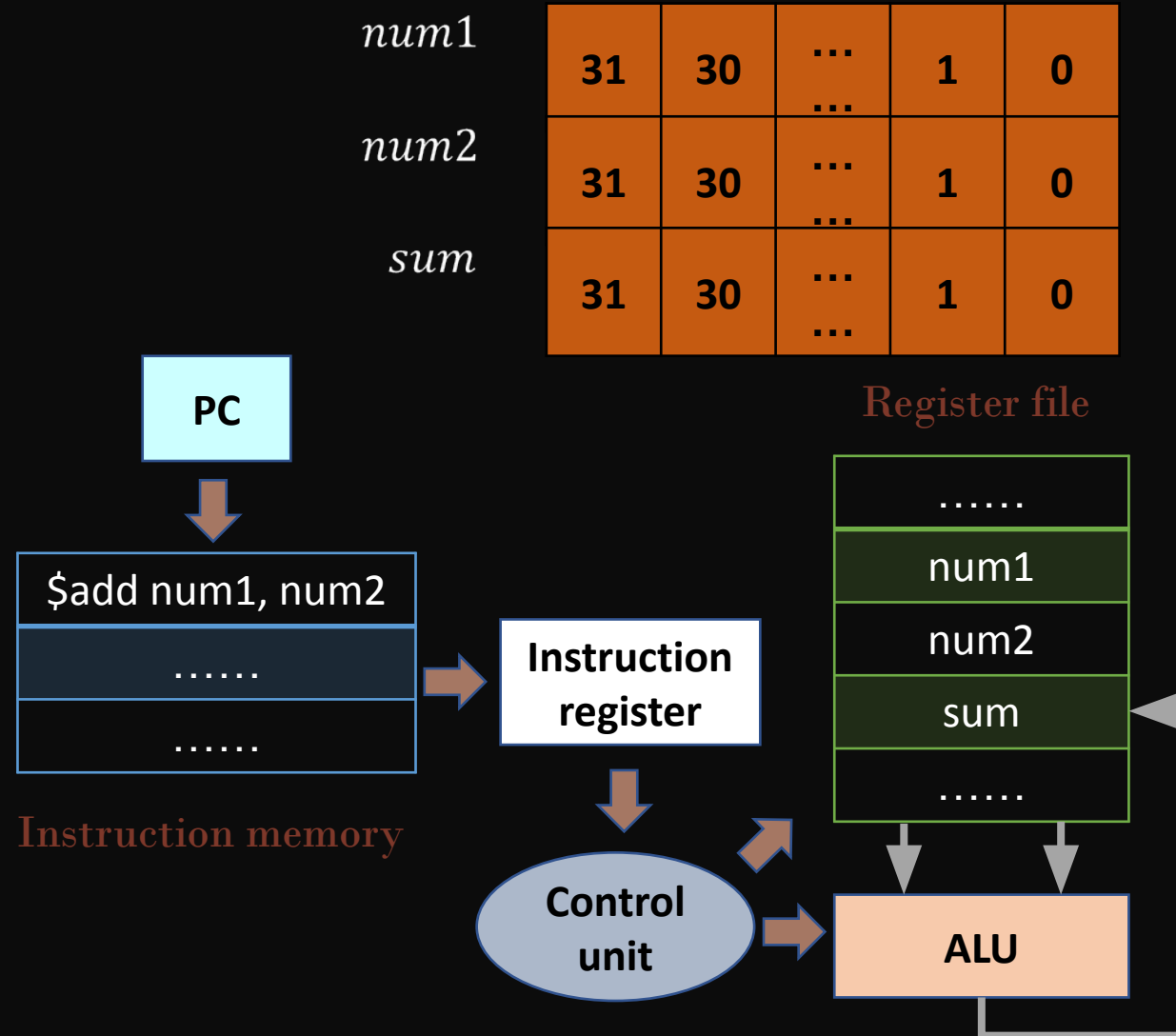
Execution details of a simple C code

```
#include <stdio.h>
int main() {
    int num1, num2, sum;

    printf("Enter two integers: ");
    scanf("%d %d", &num1, &num2);

    // calculating sum
    sum = num1 + num2;

    printf("%d + %d = %d", num1, num2, sum);
    return 0;
}
```

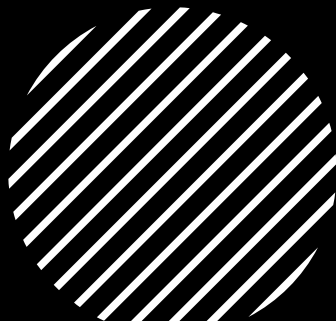




Basic building block of Verilog

- A module is the basic building block of Verilog. A **module consists of port declaration** and Verilog codes to perform the desired functionality.
- Ports are means for the Verilog module to communicate with other modules or interfaces. **Ports can be of 3 types, such as: *input*, *output*, *inout*.**
- A typical Verilog module declaration:

```
module <name> (<ports_list>);  
...  
// Verilog Codes //  
...  
endmodule
```

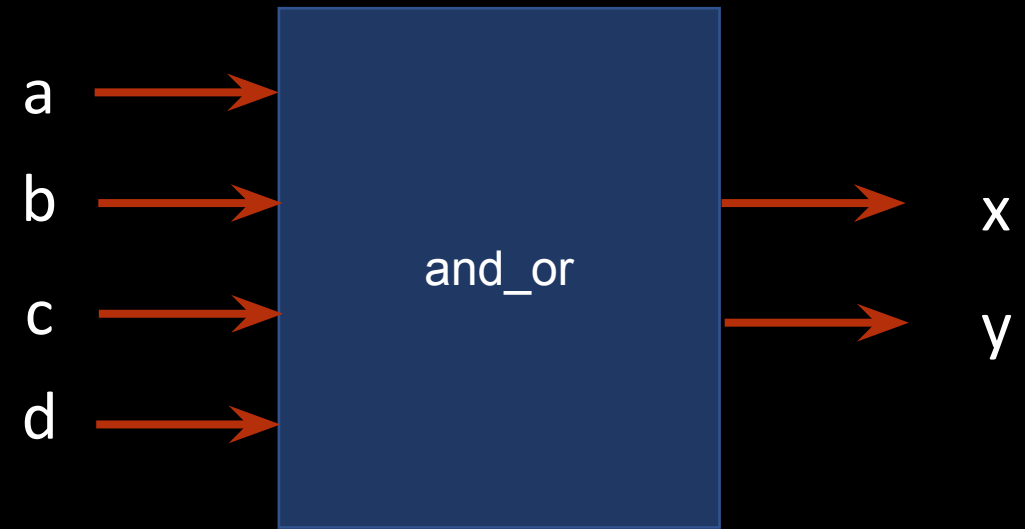


Verilog module and ports

Verilog module declaration

```
module and_or (x, y, a, b, c, d);  
    input a, b, c, d;  
    output x, y;  
    ...  
    // Verilog Code //  
    ...  
endmodule
```

Logic Synthesis



Basic Syntax and Lexical Conventions

- Documentation in Verilog code

- Documentation can be included in Verilog code by writing comments. A **short comment begins with a double slash (//)**. A **long comment spans multiple lines and is contained inside /* and */**.

```
module and_or(x, y, a, b, c, d);  
// A Verilog module  
/*  
This module takes four inputs a, b, c, d and produces two outputs x, y.  
The outputs are defined as: x = (a & b) and y = (c | d).  
*/  
input a, b, c, d;  
output x, y;  
  
// Verilog Code for the desired functionality//  
  
endmodule
```

Basic Syntax and Lexical Conventions

- White space

- White space characters such as *SPACE* and *TAB* are ignored by the Verilog compiler. Although multiple statements can be written in a single line, placing each statement in a single line and using *indentation* within blocks of code are good ways to increase readability of the code.

- Number specification

- There are two types of number specifications found in verilog, *sized* and *unsized*.

- *sized* numbers are represented as: *<size> ' <base format> <number>*. Supported formats are:

Base format	Illustration
d	decimal
b	binary
h	hexadecimal
o	octal

Basic Syntax and Lexical Conventions

- If a number is specified **without a base format**, it is treated as a **decimal number** by the Verilog compiler.
- *unsized* numbers are specified without a size specification. *unsized* numbers are assigned a **specific number of bits which is simulator and machine-specific (at least 32 bits)**.

<i>sized</i> number representation	<i>unsized</i> number representation
5'b10001 // This is a 5-bit <i>binary</i> number	1254 // This is a 32-bit <i>decimal</i> number by default
4'hff01 // This is a 4-bit <i>hexadecimal</i> number	`h21ff // This is a 32-bit <i>hexadecimal</i> number
3'o123 // This is a 3-bit <i>octal</i> number	`o345 // This is a 32-bit <i>octal</i> number
2'd10 // This is a 2-bit <i>decimal</i> number	`b1100 // This is a 32-bit <i>binary</i> number

- Value Set

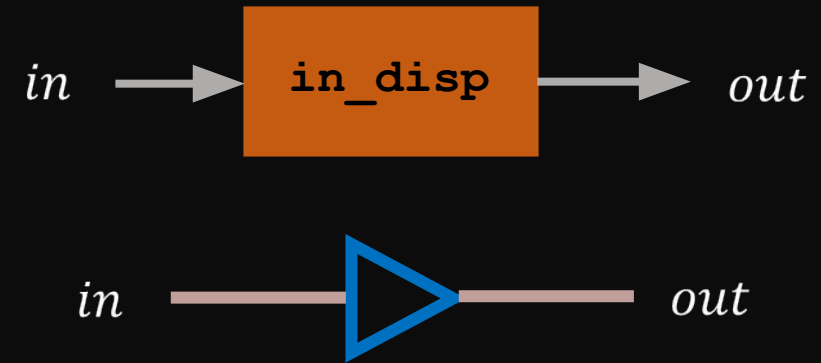
- Each individual signal/variable in Verilog can be assigned one of 4 values:

Value level	Condition in Hardware
0	logic 0 / False
1	logic 1 / True
x	undefined
z	high impedance

How to assign numerical values to the circuit nodes?

- ❖ Verilog makes use of the reserved keyword **assign** to easily store numerical values in a variable.
- ❖ The **assign** statements are **concurrent**, meaning that they are executed in parallel.

```
module in_disp(out, in);  
  
// This module implements a 1-bit buffer  
  
    input in;  
    output out;  
  
    assign out = in;  
  
endmodule
```



Basic Syntax and Lexical Conventions

Identifier Names

- Identifiers are the **names of variables** and other elements in Verilog code.
- Valid identifier can include any letter and digit as well as “_” and “\$” characters. There are two restrictions too , an identifier must not begin with a digit and it should not be a Verilog keyword. Furthermore, Verilog is case sensitive.

Identifier name	Validity
x1	Valid
x_y	Valid
1x	Invalid
+y	Invalid
x*y	Invalid
258	Invalid
ex_\$1	Valid

Basic Syntax and Lexical Conventions

- Vectors

- input or output variables can also be declared as vectors (multiple bit widths). If bit width is not specified, the default is scalar (1-bit).

- The multibit variables or vectors in general can be declared in Verilog using the syntax:

<data type> <MSB bit index : LSB bit index> <name>

```
module in_disp(out, in);  
    // This module implements a 3-bit  
    buffer  
  
    input  [2:0]in;  
    output [2:0]out;  
  
    assign out = in;  
endmodule
```

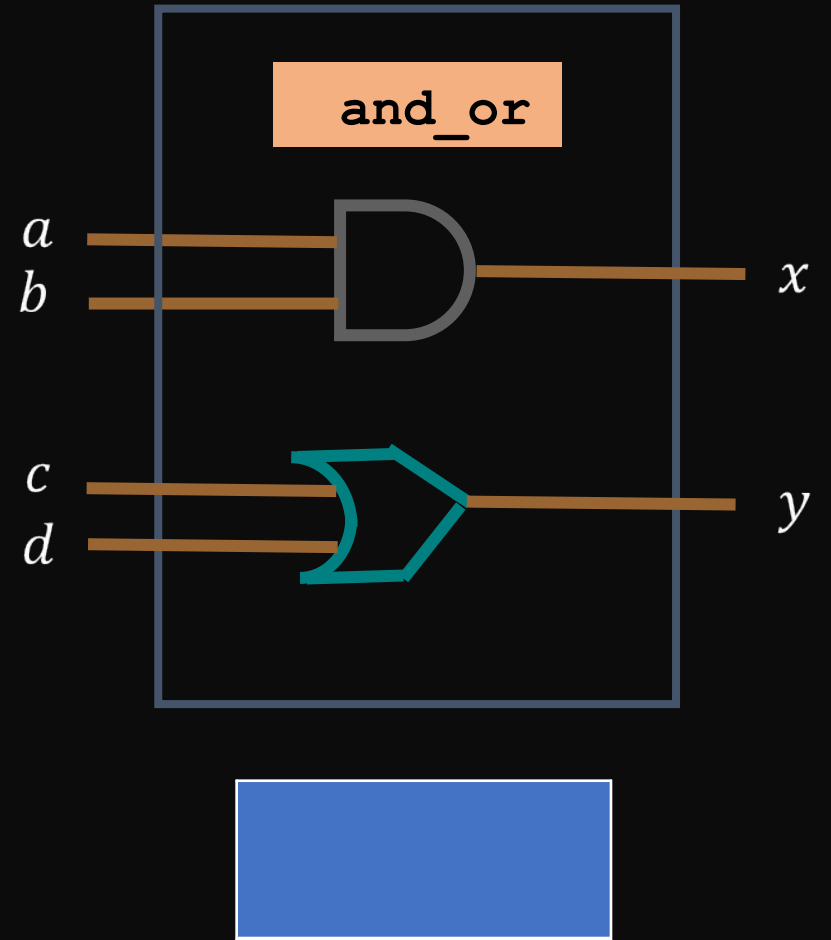
Verilog operators

Below is a list of the most frequently used operators in Verilog:

Operator	Operation
$\sim A$	This will produce 1's complement of A
$A \& B$	Bitwise AND
$A B$	Bitwise OR
$A \wedge B$	Bitwise XOR
$A + B$	Addition of two single or multibit numbers
$A - B$	Subtraction of two single or multibit numbers
$A * B$	Multiplication of two single or multibit numbers
A / B	Division of two single or multibit numbers
$A \% B$	This returns the remainder of the integer division A/B

A simple circuit

```
module and_or(x, y, a, b, c, d);  
    input a, b, c, d;  
    output x, y;  
  
    assign x = a & b;  
    assign out = c | d;  
  
endmodule
```



Another simple circuit

```
module example_ckt(f, x1, x2,  
x3);
```

```
    input x1, x2, x3;
```

```
    output f;
```

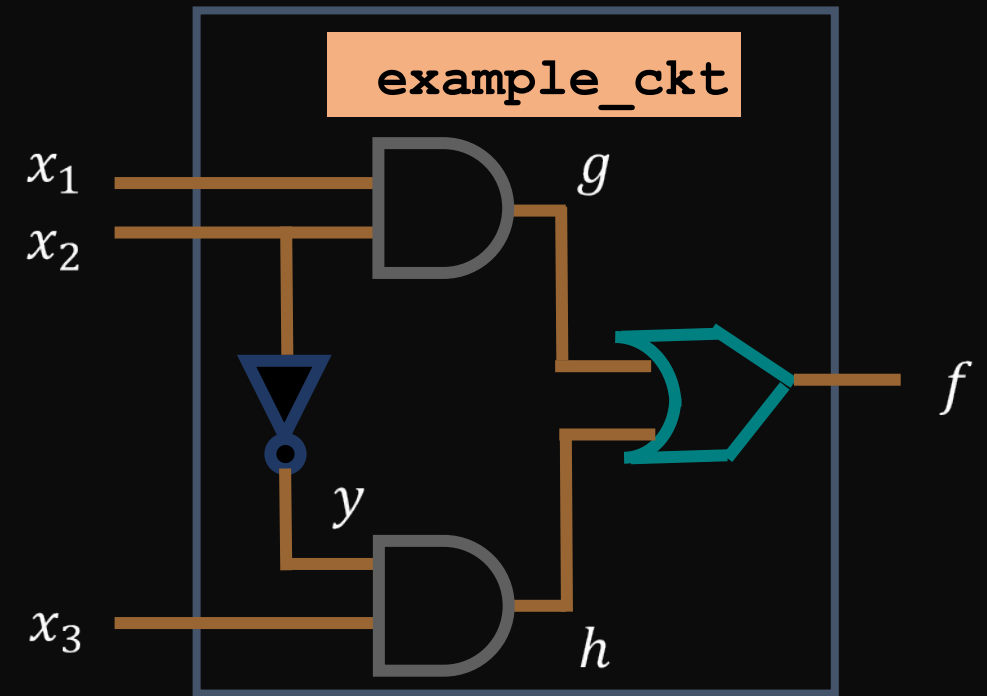
```
    assign g = x1 & x2;
```

```
    assign y = ~x2;
```

```
    assign h = y & x3;
```

```
    assign f = g | h;
```

```
endmodule
```



x1	x2	x3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Thank You