



# Introduction to Digital Design in Verilog

---

**Prepared by**

Beig Rajibul Hasan &

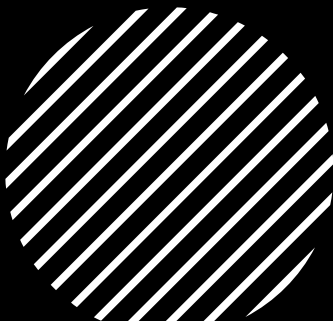
Md. Asif Hossain Bhuiyan

Lecturer, CSE, BRAC University



# Lab Policy

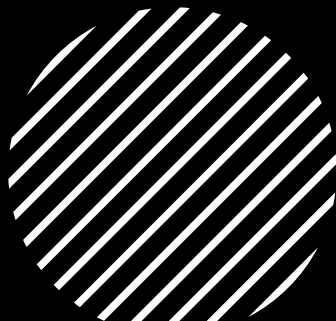
- Attendance (All 6) - 2%
- Class work - 3%
- Lab report (n-1) - 5%
- Lab test - 5%
- Project - 10%
- **Total - 25%**





# Lab Policy

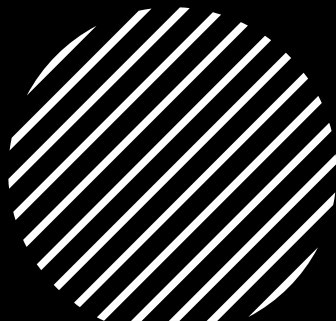
- Lab 1 (exp 1) - QUARTUS
- Lab 2 (exp 2) - QUARTUS
- Lab 3 (exp 3.1) - QUARTUS
- Lab 4 (exp 3.2) - QUARTUS
- Lab 5 (exp 4,5) - DSCH2
- Lab 6 (exp 6,7) - MICROWIND2



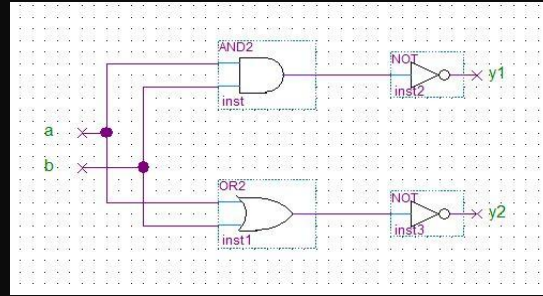


# Verilog HDL

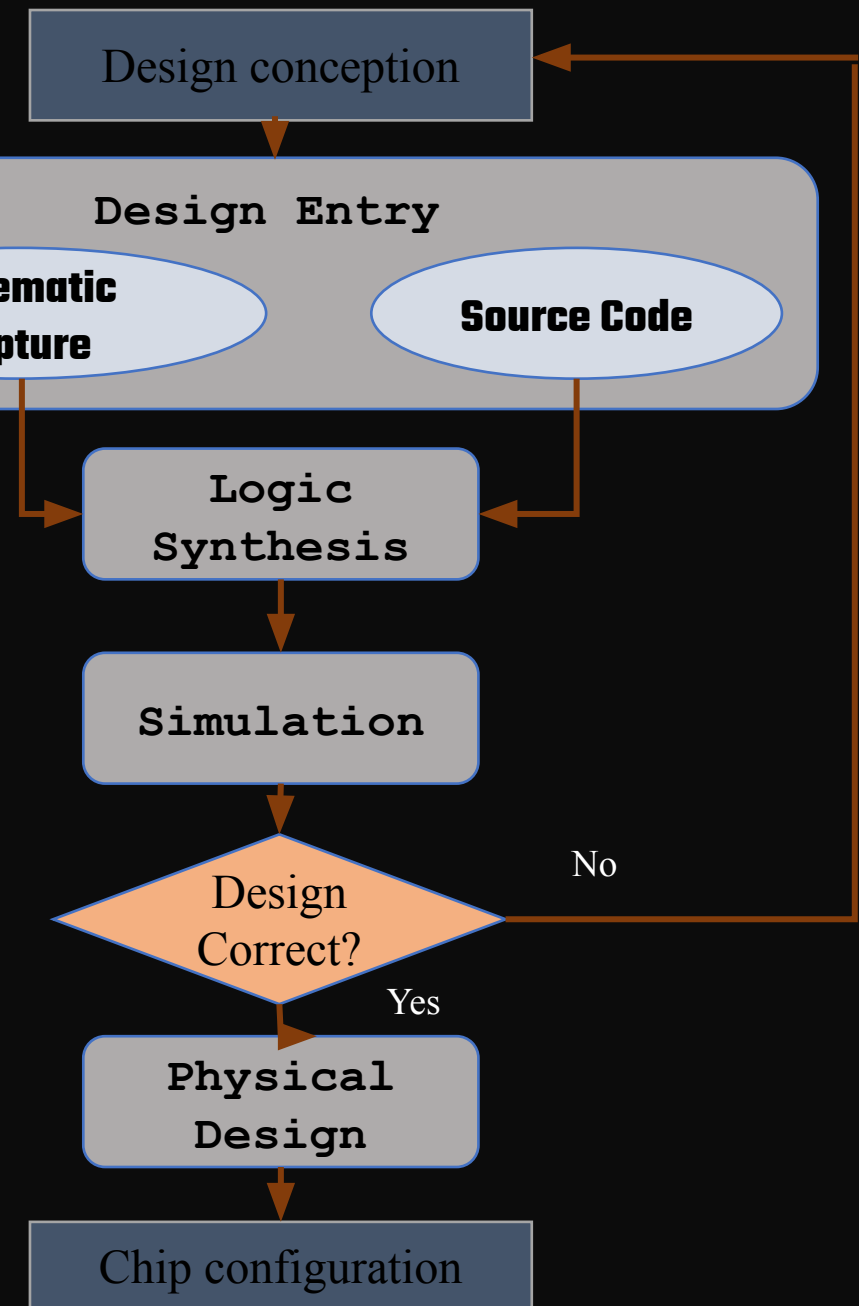
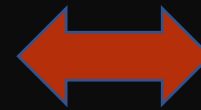
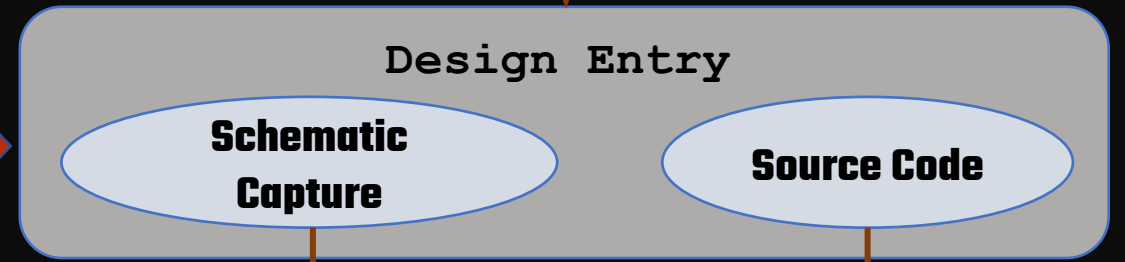
- Verilog HDL is a general-purpose **hardware description language which can describe the digital circuits** with C-like syntax.
- Most popular logic synthesis tools support Verilog HDL.
- Digital circuits can be described at the RTL of abstraction which ensures design portability.



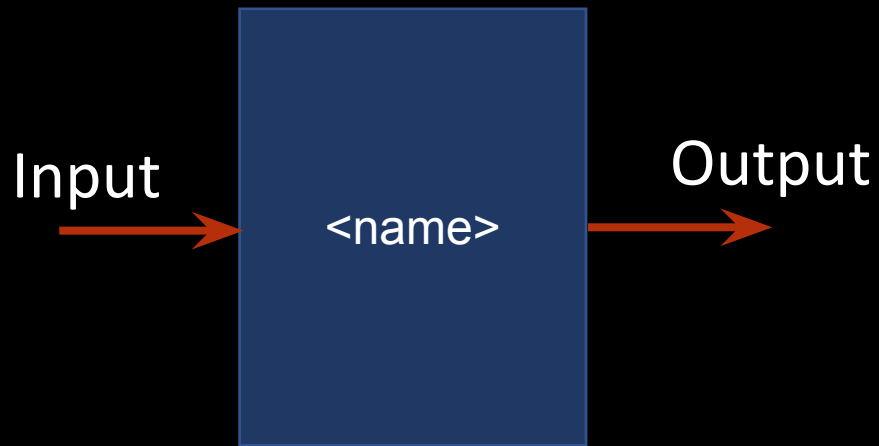
# Design Flowchart of a Typical CAD System



a	b		
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	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:

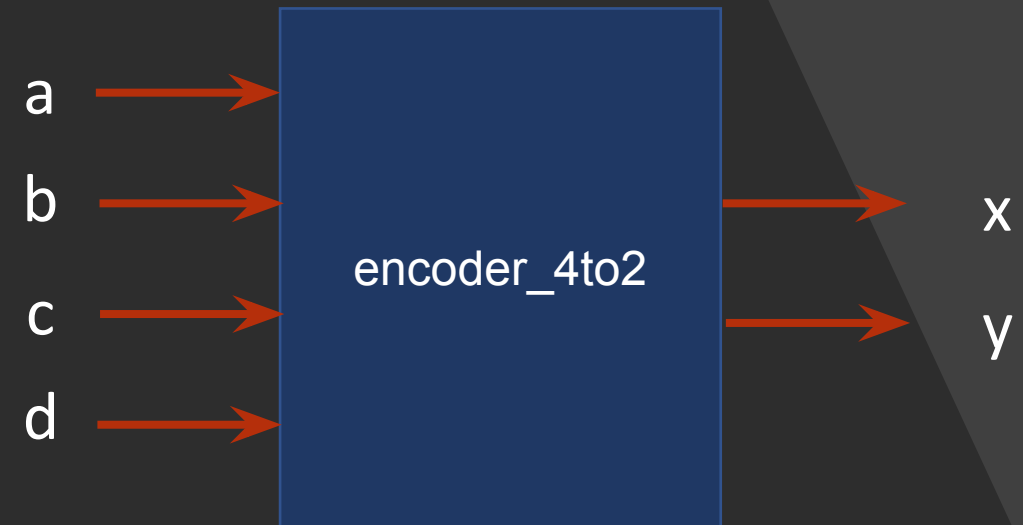
```
module <name> (<ports_list>);  
...  
// Verilog Codes //  
...  
endmodule
```

# Verilog module and ports

## Verilog module declaration

```
module encoder_4to2 (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 fulladd(s, cout, a, b, cin);  
    // A full adder verilog module  
    /*  
    This module takes three inputs a, b, cin and adds them.  
    The sum of the inputs are stored in s, the carryout is stored in cout.  
    */  
    input a, b, c;  
    output s, cout;  
    // Verilog Code //  
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

- 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

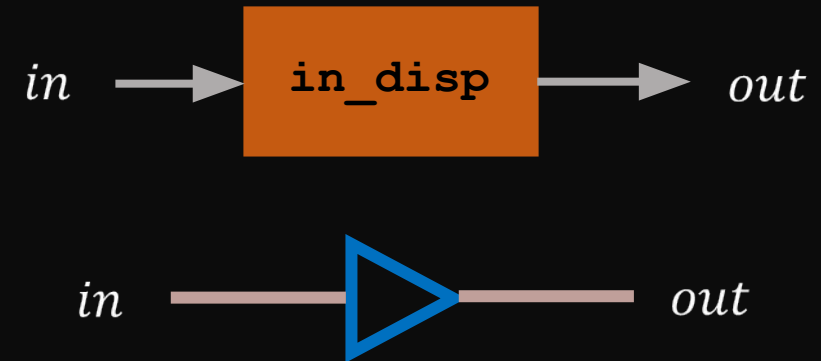
- Each individual signal/variable in Verilog can be assigned one of 4 values:

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
```



- 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>*

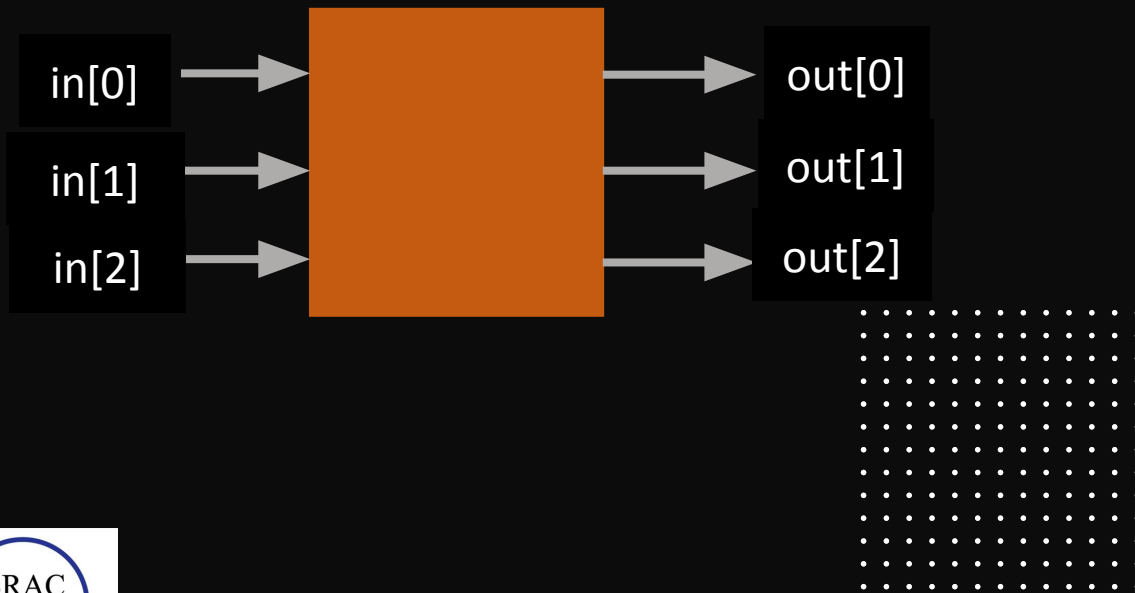
```
module in_disp(out, in);  
// This module implements a 3-bit  
buffer
```

```
    input  [2:0] in;  
    output [2:0] out;
```

```
    assign out = in;  
endmodule
```

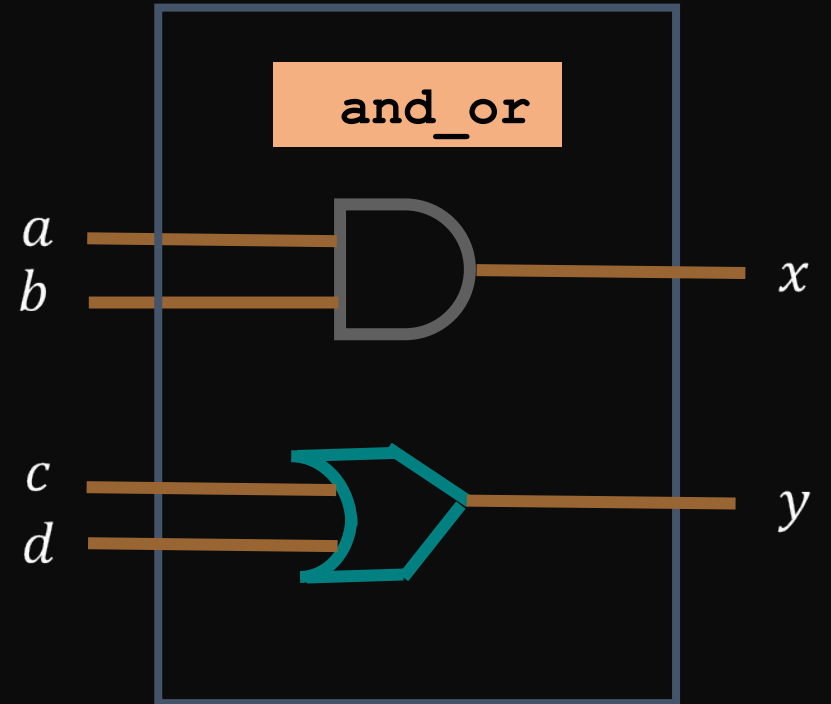


# Basic Syntax and Lexical Conventions



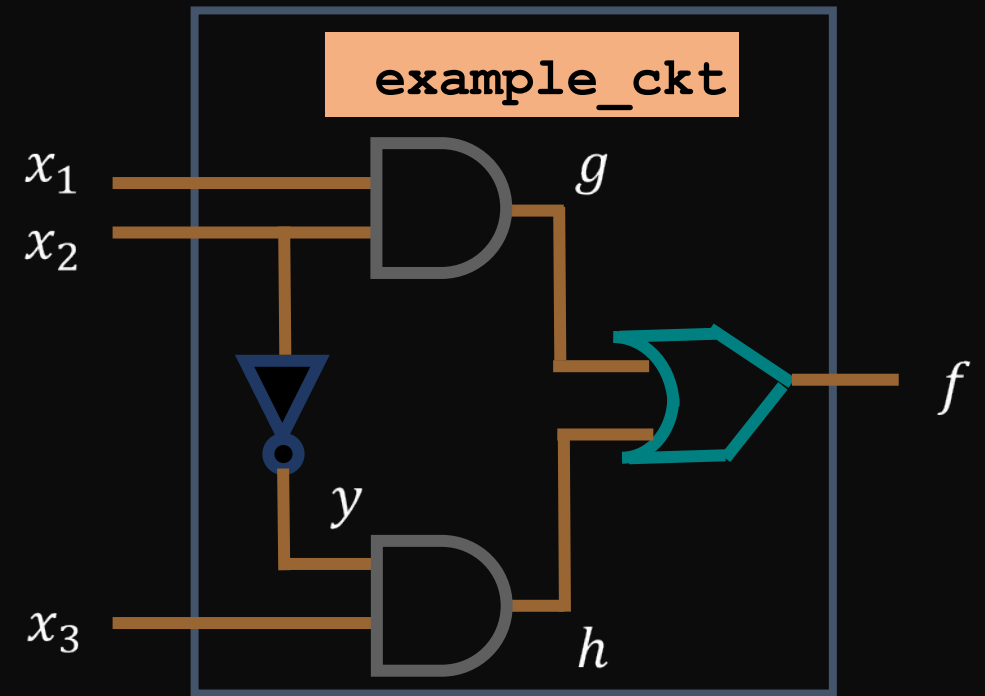
# A simple circuit

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

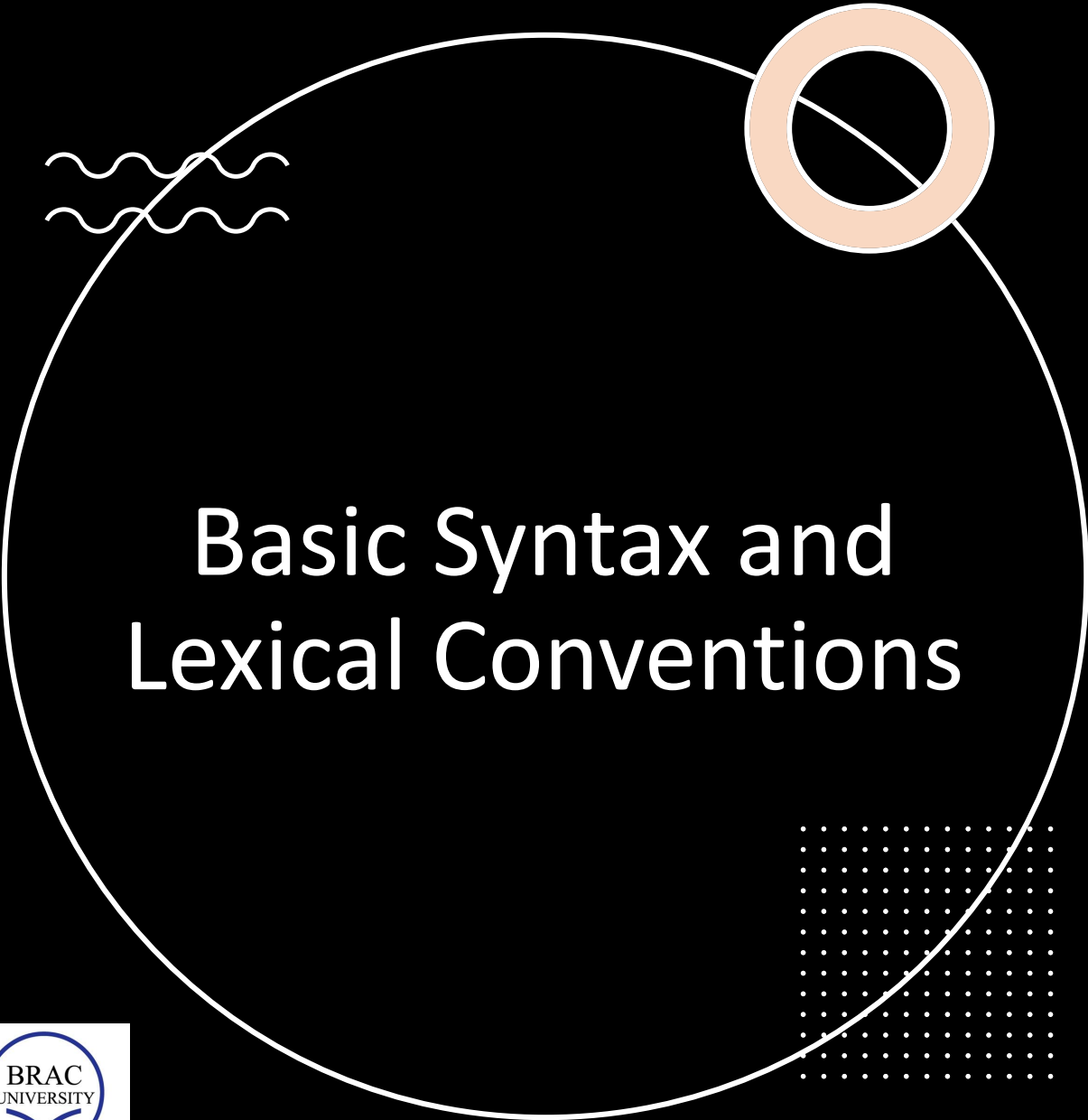


# Another simple circuit

```
module example_ckt(f, x1, x2, x3);  
  
    input x1, x2, x3;  
    output f;  
  
    wire g, y, h;  
  
    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



# Basic Syntax and Lexical Conventions

- Nets

- Nets represent connections between hardware elements. Nets are continuously driven by the outputs of the devices they are connected to.

- Nets are declared with the keyword *wire*. A net is assigned the value *z* by default.

- Registers

- In verilog **register means a variable that can hold a value**. Unlike net, a register doesn't need a driver.

- Registers are declared with the keyword *reg*. The default value of a *reg* data type is *x*.

```
wire a, b; // wire declaration
reg clock; // register declaration
```



# Basic Syntax and Lexical Conventions

- Vectors

- *wire* or *reg* type data types can also be declared as vectors (multiple bit widths). If bit width is not specified, the default is scalar (1-bit).

- The multibit nets/registers or vectors in general can be declared in Verilog using the syntax:

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

```
module example(a, b);  
input b;  
output a;  
wire [7:0] in1, in2; // 8-bit wire type  
variables  
reg [0:31] base_clk; // 32-bit reg type  
variable  
assign a = in1[7] * in2[2];  
endmodule
```

# Verilog Representations of Digital Circuits

Verilog allows designers to describe a digital circuit in several ways. Among them two fundamental representations are: *structural representation* and *behavioral representation*.

- Structural representation

- Structural representation is to use Verilog's **gate-level primitives to describe the digital circuit**. Various gate-level primitives are included in Verilog such as: *and* gate, *or* gate, *not* gate, *nand* gate, *nor* gate etc.

- Behavioral representation

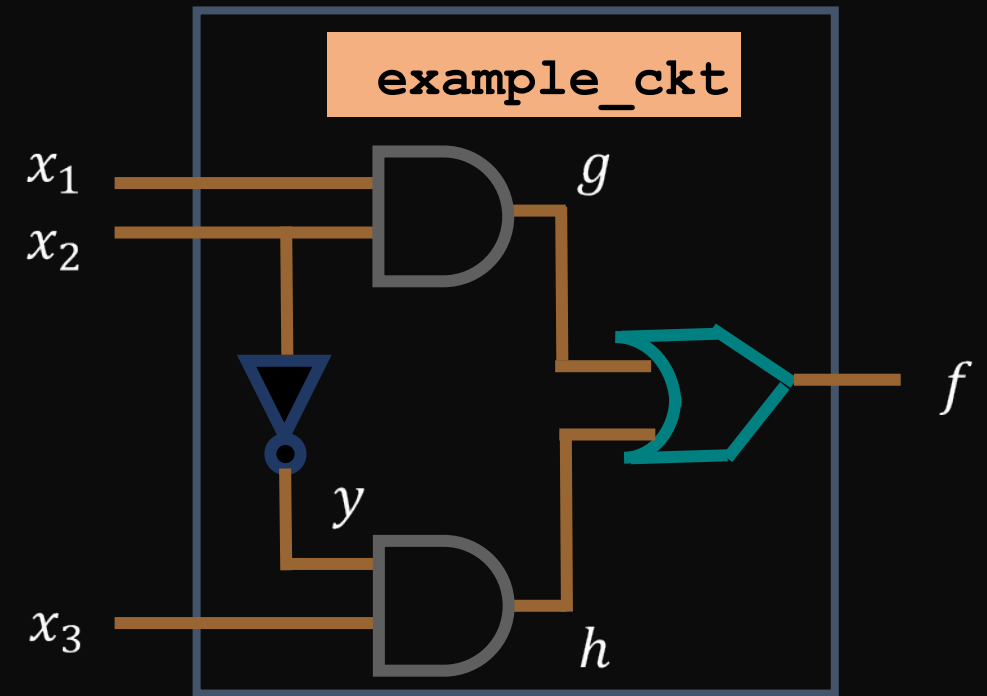
- Using gate-level primitives can be tedious while designing larger circuits. Instead, the designers use **more abstract expressions and programming constructs** to describe the circuit. This is called the behavioral representation of the digital circuit.

# Structural representation

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

    wire g, y, h;

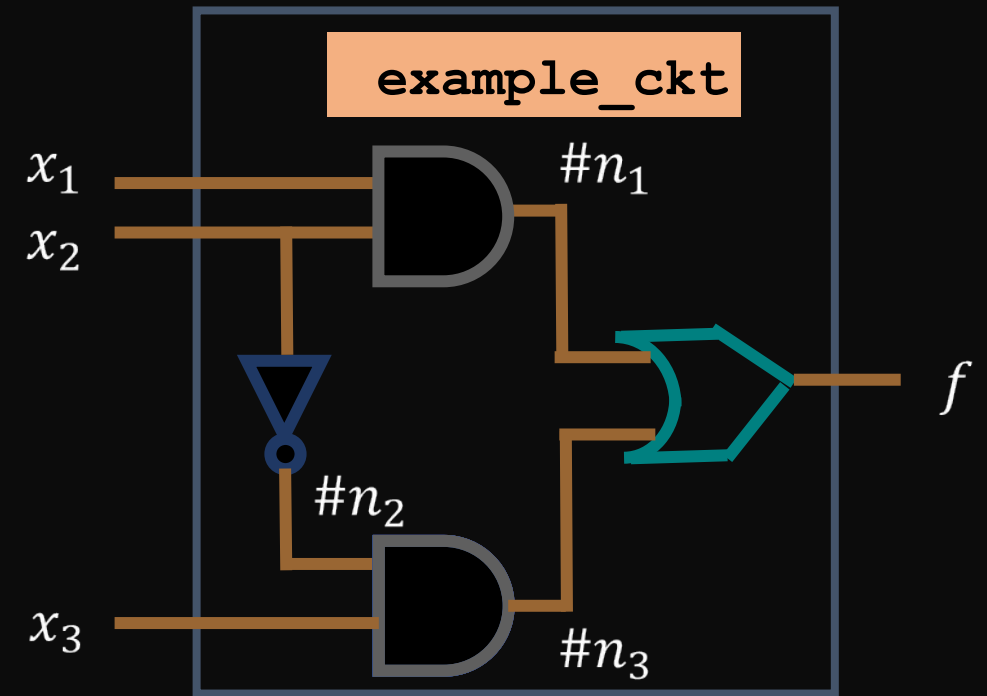
    and(g, x1, x2);
    not(y, x2);
    and(h, y, x3);
    or(f, g, h);
endmodule
```



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

# Behavioral representation

```
module example_ckt(f, x1, x2, x3);  
    input x1, x2, x3;  
    output f;  
    assign f = (x1 & x2) | (~x2 & x3);  
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



# Verilog Operators

Verilog supports a large number of operators for carrying out different types of operations. Verilog operators can be broadly classified into the following categories:

- ☐ Bitwise operators
- ☐ Logical operators
- ☐ Arithmetic operators
- ☐ Relational operators
- ☐ Shift operators
- ☐ Condition operator



# Verilog Operators

## Bitwise operators

Operator	Operation
$\sim A$	This will produce 1's complement of A
$-A$	This will produce 2's complement of A
$A \& B$	Bitwise AND
$A   B$	Bitwise OR
$A \wedge B$	Bitwise XOR
$A \wedge \sim B / A \sim \wedge B$	Bitwise XNOR



# Verilog Operators

## Logical operators

These operators work on single or multi bit operands but generate 1 bit result i.e., *True* or *False*.

Operator	Operation
!A	NOT A(!A) produces “1(True)” only if all bits of A are 0 else !A gives “0(False)”
A&&B	The result of A&&B is “1(True) if both A and B are nonzero
A  B	A  B gives “1(True)” unless both A and B are zero.



# Verilog Operators



## Arithmetic operators

Operator	Operation
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





# Verilog Operators

## Relational operators

Operator	Operation
$A == B$	1 ( <i>True</i> ) if A is equal to B, 0 ( <i>False</i> ) otherwise
$A != B$	1 ( <i>True</i> ) if A is not equal to B, 0 ( <i>False</i> ) otherwise
$A > B$	1 ( <i>True</i> ) if A is greater than B, 0 ( <i>False</i> ) otherwise
$A < B$	1 ( <i>True</i> ) if A is less than B, 0 ( <i>False</i> ) otherwise

# Verilog Operators

Operator type	Symbol	Operation performed
Shift right	>>	Shift right logical (division by 2)
Shift left	<<	Shift left logical (multiplication by 2)
Condition	$D = A ? B : C$	D is equal to B if A is True, otherwise D is equal to C

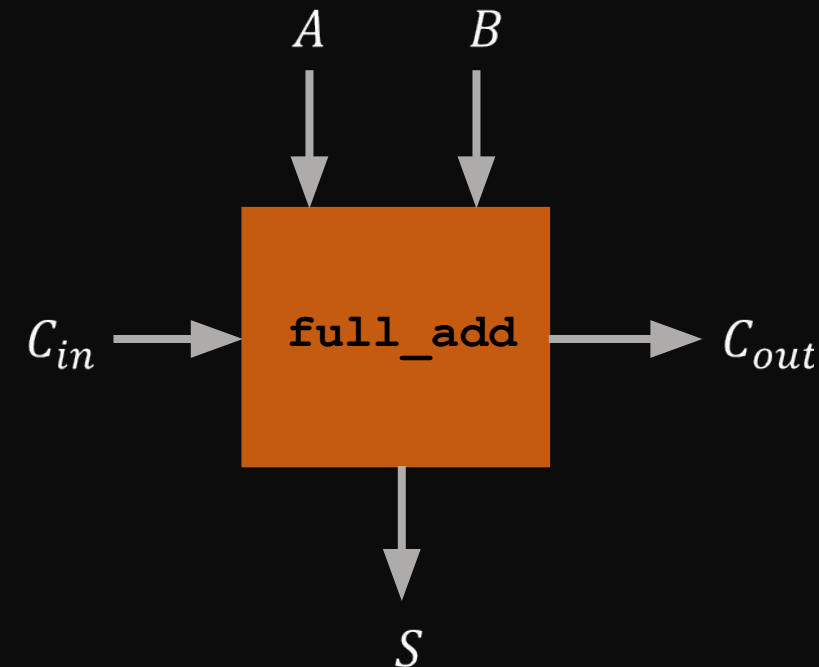
# Concurrent Statements

In any HDL, concurrent statement means the code may include a number of statements and each represent a part of the circuit.

## What concurrent means:

Concurrent is used because the statements are considered in parallel and the ordering of statements in the code doesn't matter. Most frequently used concurrent statements in Verilog are the continuous assignments.

```
module full_add(S, Cout, A, B, Cin);  
  // This module implements a 1-bit full adder  
  input A, B, Cin;  
  output S, Cout;  
  
  assign S = A ^ B ^ Cin;  
  assign Cout = (A & B) | (Cin & (A ^ B));  
endmodule
```



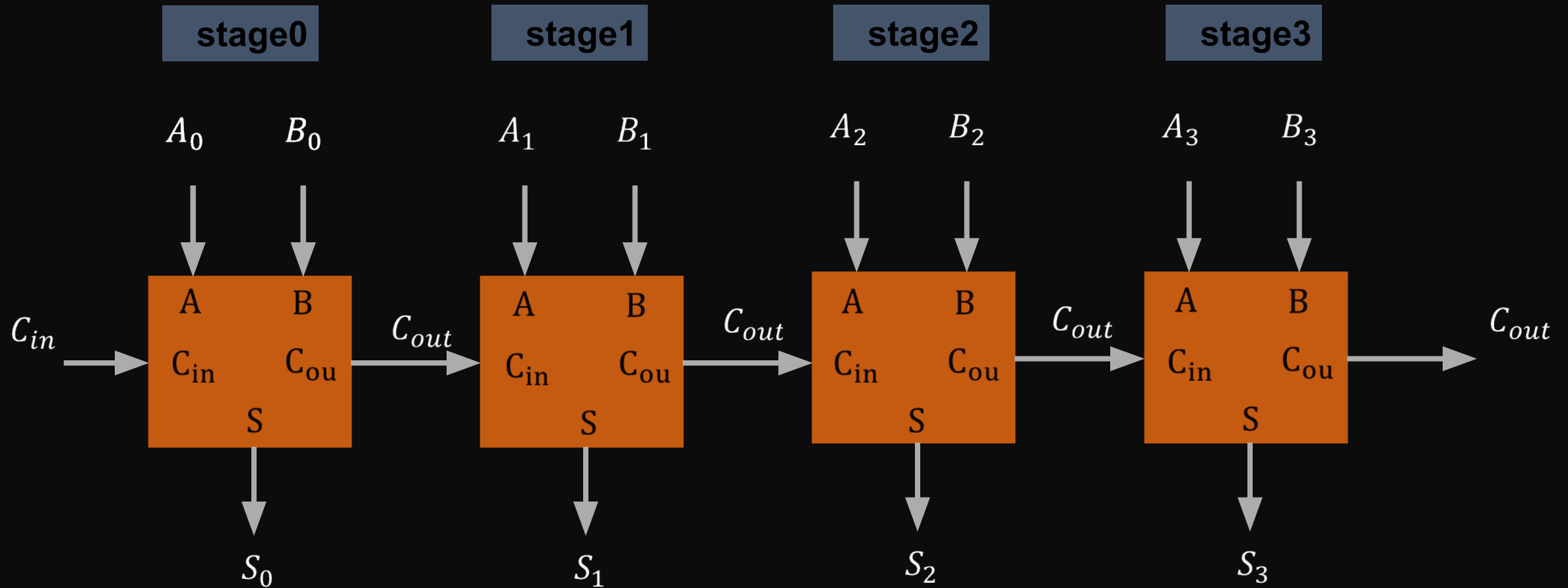
# Subcircuits in Verilog

- A Verilog module can be included as a subcircuit in another module.
- Both module must be defined in the same file or else Verilog compiler must be told where each module is located.
- The general form of module instantiation expression is

***<module\_name> <instance\_name> ( <port\_name[expressions]> )***

- In the above definition
  - ▢ *module\_name* is the name of the module of the child circuit that is to be included in the parent circuit.
  - ▢ *instance\_name* can be any legal Verilog identifiers.
  - ▢ *port\_name* basically is the list of ports that specify the connections that will be passed to the subcircuit.

# 4-bit Ripple Carry Adder



# 4-bit Ripple Carry Adder

```
module fulladd4(S0, S1, S2, S3, Cout, A0, A1, A2, A3, B0, B1, B2, B3, Cin);  
    input A0, A1, A2, A3, B0, B1, B2, B3, Cin;  
    output S0, S1, S2, S3, Cout;  
    wire Cout0, Cout1, Cout2;  
    fulladd stage0 (S0, Cout0, A0, B0, Cin);  
    fulladd stage1 (S1, Cout1, A1, B1, Cout0);  
    fulladd stage2 (S2, Cout2, A2, B2, Cout1);  
    fulladd stage3 (S3, Cout, A3, B3, Cout2);  
endmodule  
  
module fulladd(S, Cout, A, B, Cin);  
    // This module implements a 1-bit full adder  
    input A, B, Cin;  
    output S, Cout;  
  
    assign S = A ^ B ^ Cin;  
    assign Cout = (A & B) | (Cin & (A ^ B));  
endmodule
```

# Text & References

#	Title	Author(s)	Edition
1.	CMOS VLSI Design	N.H.E. Weste, D. Harris & A. Banerjee	4 <sup>th</sup> ed.
2.	Fundamentals of Digital Logic with Verilog Design	Stephen Brown & Zvonko Vranesic	2 <sup>nd</sup> /3 <sup>rd</sup> ed.
3.	Verilog HDL (A guide to Digital Design and Synthesis)	Samir Palnitkar	1 <sup>st</sup> ed.

*Thank You*