

# Verilog Tutorial 1

---



INDRAPRASTHA INSTITUTE *of*  
INFORMATION TECHNOLOGY **DELHI**



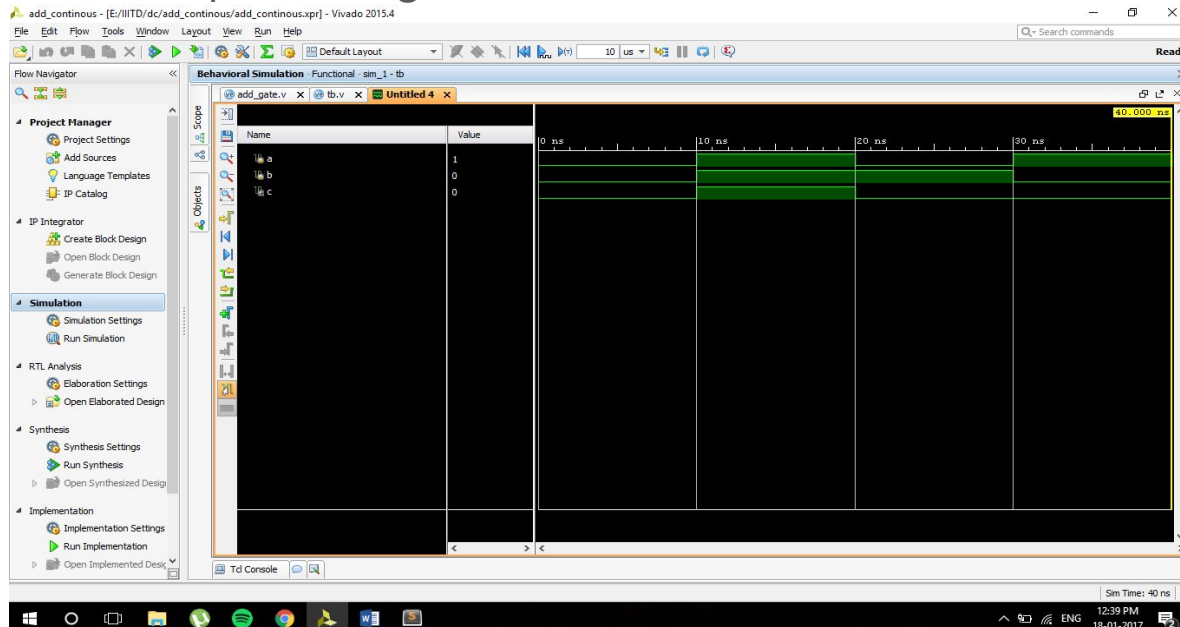
# Hardware description languages

---

- Hardware description languages (HDL) offer a way to design circuits using text-based descriptions.
- HDLs have two objectives
  - Allow for testing/verification using computer simulation » Includes syntax for timing, delays
  - Allow for synthesis » Synthesizable HDL
- Two primary hardware description languages
  - VHDL
  - Verilog

# Verilog

- Represents hardware structure and behavior
- Logic simulation: generates waveforms
- For example: And gate



# Syntax

---

- Lines that begin with `//` are comments (ignored by simulation).
- There are various keywords in verilog (keywords are case sensitive).
- In verilog, all keywords are in lower case.
- **module**: Building block in Verilog.
- **endmodule**: To terminate a module in verilog.
- Module parameters are ports which can either be **input** or **output**.
- **wire** defines internal circuit connection.
- **reg** defines a storing element.
- Every executable statement ends with a semicolon.

<http://euler.ecs.umass.edu/ece232/pdf/03-verilog-11.pdf>

<http://www.asic-world.com/verilog/syntax1.html>

# Syntax

---

- Numbers are represented as

**<size>'<radix><value>**

- Size: Number of bits
- Radix:
  - d represents decimal
  - b represents binary
- Example: 4'd15 or 4'b1111  
As 15 is denoted by 1111 in binary

# Syntax

---

## Buses/nets

- It denotes the number of registers/wires used to store your data.
- Its size is equal to the number of bits in your input/output.

## Example:

wire [3:0] a, where 3 is the MSB and 0 is the LSB.

Similarly, for wire [1:3] a, 1 will be the MSB and 3 will be the LSB.

# Modelling Styles

---

- **Gate-level Modelling:** It is useful when a circuit is a simple combinational. It is the lowest level, and implemented using switches and transistors. Designer should have knowledge about switches and transistors. For example,
  - and A1(Out, in1, in2); // This will and the values of 'in1' and 'in2' and store it in the 'out' variable.
- **Dataflow Modelling:** It is mainly used to describe combinational circuits. The basic mechanism used is the continuous assignment.
- **Behavioral Modelling:** It is the highest level of implementation. It is primarily used to model sequential circuits, but can also be used to model pure combinatorial circuits. It uses procedural assignment. Here the designer necessarily not need to know the exact hardware implementation.

# Blocking vs Non-blocking statements

---

- **Blocking Statements:** A blocking statement must be executed before the execution of the statements that follow it in a sequential block.
- **Non Blocking Statements:** Nonblocking statements allow you to schedule the assignments without blocking the procedural flow.

A = B;                      B = A;

Here both the values will be the value of B, as this runs sequentially.

A <= B;                      B <= A;

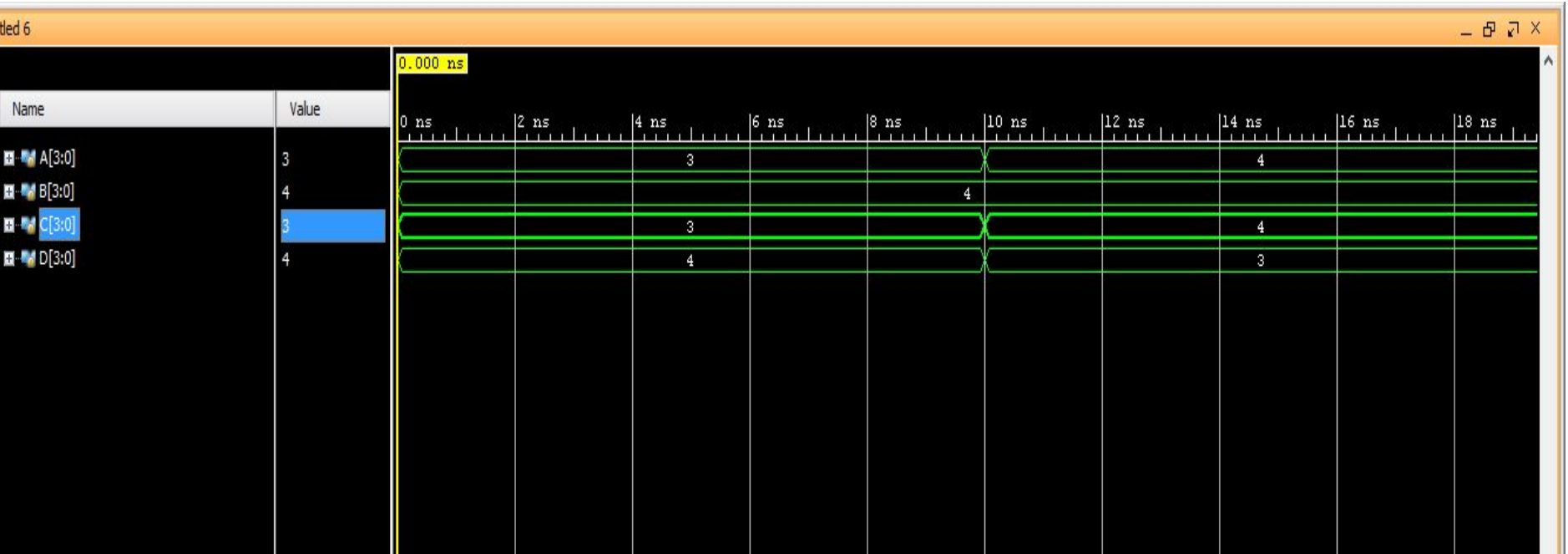
In this case both the values will swap, as it runs on the same clock cycle.



# Example 1

```
add1.v
E:/IITD/dc/add/add.srcs/sources_1/new/add1.v
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21
22
23 module add1();
24     reg [3:0] A, B, C, D;
25     initial begin
26         A = 4'd3;
27         B = 4'd4;
28
29         #10
30         A = B;      // This will start to execute after 10 cycles
31         B = A;      // After this statement both A and B will be equal to the value of B.
32     end
33
34     initial begin
35         C = 4'd3;
36         D = 4'd4;
37
38         #10
39         C <= D;    // This statement will start to execute after 10 cycles.
40         D <= C;    // After this statement C and D values will swap.
41     end
42
43 endmodule
44
45
46
47
```

# Simulation for example on the previous slide



# Example 2

---

```
module main();
```

```
    reg [1:0] A, B, C, D, E, F;
```

```
    initial begin
```

```
        A = #1 2'd1; // This will be executed after 1 cycle
```

```
        B = #2 2'd2; // This will be executed after 3 cycle
```

```
        C = #2 2'd3; // This will be executed after 5 cycle
```

```
    End
```

```
    initial begin
```

```
        D <= #1 2'd1; // This will be executed after 1 cycle
```

```
        E <= #2 2'd2; // This will be executed after 2 cycle
```

```
        F <= #2 2'd3; // This will be executed after 2 cycle
```

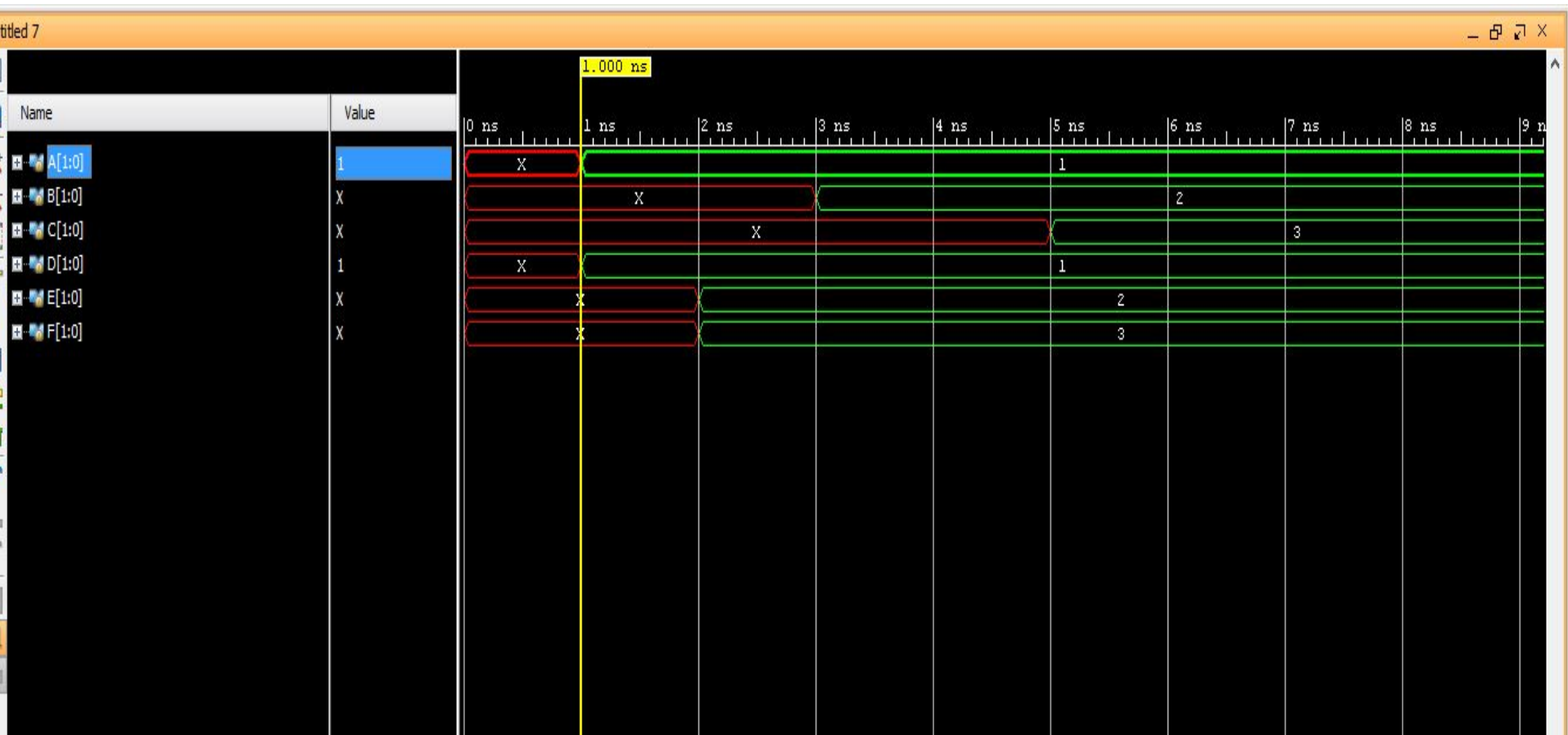
```
    end
```

```
endmodule
```

## Note:

- X in the simulation denotes an unknown logic, it may be 0 or 1.

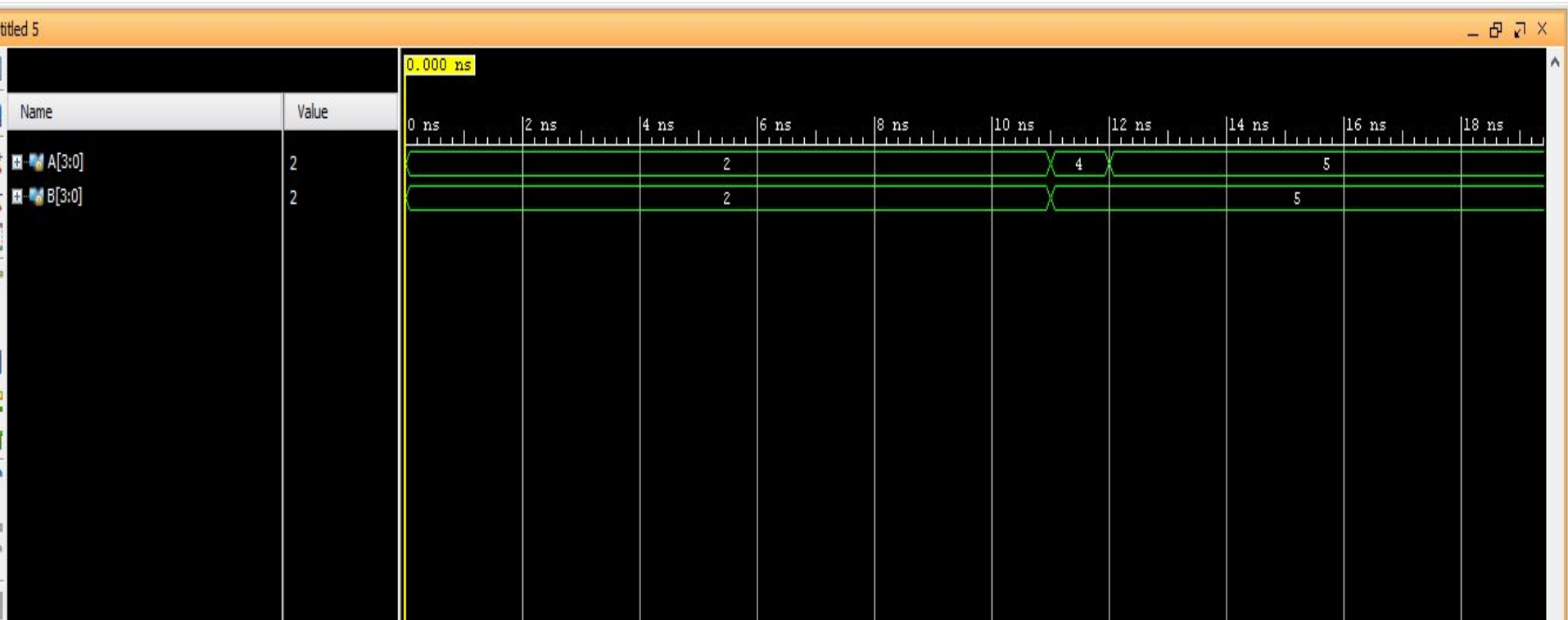
# Simulation



# Example 3

```
add1.v
E:/IIITD/dc/add/add.srcs/sources_1/new/add1.v
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////
21
22
23 module add1();
24     reg [3:0] A, B;
25     initial begin
26         A = 4'd2;
27
28         #10
29         A = #1 4'd4; // This will start to execute after 10 cycles
30                     // After this statement A will be 4 on the 11th cycle.
31         A = #1 4'd5; // After this statement A will be 5 on the 12th cycle.
32     end
33
34     initial begin
35         B <= 4'd2;
36
37         #10
38         B <= #1 4'd4; // This statement will start to execute after 10 cycles.
39                     // After this statement B will be 4 on the 11th cycle
40         B <= #1 4'd5; // After this statement B will be 5 on the 11th cycle as this statement
41                     // has also started to execute after 10 cycles (non-blocking assignment)
42     end
43
44 endmodule
45
46
47
48
```

# Simulation



# Continuous vs Procedural assignments

---

- Continuous assignment is used to drive a value on to a wire/net. This is done using **assign** keyword.

## Example:

```
module Conti_Assignment (inp1, inp2, outp);  
    input inp1, inp2;  
    Output outp;  
    wire inp1, inp2, outp;  
  
    //Net (scalar) continuous assignment  
    assign outp = inp1 | inp2;  
  
endmodule
```

# Continuous vs Procedural assignments

---

- Continuous assignment updates net, but procedural assignment update values of reg, real, integer or time variable. This is mostly done using **always** and **initial**.

## Example:

```
module Proc_Assignment (inp1, inp2, outp);  
    input inp1, inp2;  
    output outp;  
    wire inp1, inp2;  
    reg outp;
```

```
//Register procedural assignment  
    always @(inp1, inp2)  
        outp = inp1 | inp2;  
endmodule
```



# References

---

- Tutorial on verilog:  
[https://www.tutorialspoint.com/vlsi\\_design/vlsi\\_design\\_verilog\\_introduction.htm](https://www.tutorialspoint.com/vlsi_design/vlsi_design_verilog_introduction.htm)
- For functions:  
<https://www.csee.umbc.edu/portal/help/VHDL/verilog/system.html>