**Logic Gates Overview**

Logic gates are the fundamental building blocks of digital circuits. They perform basic logical functions that are essential for digital system design. Let’s cover these one by one!

**AND Gate**

**Introduction**

The AND gate outputs 1 only if **both** inputs are 1. It represents logical multiplication.

**Truth Table:**

| **A** | **B** | **Y (A AND B)** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Verilog Code (and\_gate.sv)**

module and\_gate (

input logic A,

input logic B,

output logic Y

);

assign Y = A & B;

endmodule

**Testbench (and\_gate\_tb.sv)**

`timescale 1ns/1ps

module and\_gate\_tb;

logic A, B;

logic Y;

and\_gate dut (

.A(A),

.B(B),

.Y(Y)

);

initial begin

$monitor("A=%b B=%b | Y=%b", A, B, Y);

// Apply test vectors

A = 0; B = 0; #10;

A = 0; B = 1; #10;

A = 1; B = 0; #10;

A = 1; B = 1; #10;

$finish;

end

endmodule

**OR Gate**

**Introduction**

The OR gate outputs 1 if **any** input is 1. It represents logical addition.

**Truth Table:**

| **A** | **B** | **Y (A OR B)** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Verilog Code (or\_gate.sv)**

module or\_gate (

input logic A,

input logic B,

output logic Y

);

assign Y = A | B;

endmodule

**Testbench (or\_gate\_tb.sv)**

`timescale 1ns/1ps

module or\_gate\_tb;

logic A, B;

logic Y;

or\_gate dut (

.A(A),

.B(B),

.Y(Y)

);

initial begin

$monitor("A=%b B=%b | Y=%b", A, B, Y);

A = 0; B = 0; #10;

A = 0; B = 1; #10;

A = 1; B = 0; #10;

A = 1; B = 1; #10;

$finish;

end

endmodule

**NOT Gate**

**Introduction**

The NOT gate (inverter) outputs the **opposite** of the input.

**Truth Table:**

| **A** | **Y (NOT A)** |
| --- | --- |
| 0 | 1 |
| 1 | 0 |

**Verilog Code (not\_gate.sv)**

module not\_gate (

input logic A,

output logic Y

);

assign Y = ~A;

endmodule

**Testbench (not\_gate\_tb.sv)**

`timescale 1ns/1ps

module not\_gate\_tb;

logic A;

logic Y;

not\_gate dut (

.A(A),

.Y(Y)

);

initial begin

$monitor("A=%b | Y=%b", A, Y);

A = 0; #10;

A = 1; #10;

$finish;

end

endmodule

**XOR Gate**

**Introduction**

The XOR (Exclusive OR) gate outputs 1 only when the inputs are **different**.

**Truth Table:**

| **A** | **B** | **Y (A XOR B)** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Verilog Code (xor\_gate.sv)**

module xor\_gate (

input logic A,

input logic B,

output logic Y

);

assign Y = A ^ B;

endmodule

**Testbench (xor\_gate\_tb.sv)**

`timescale 1ns/1ps

module xor\_gate\_tb;

logic A, B;

logic Y;

xor\_gate dut (

.A(A),

.B(B),

.Y(Y)

);

initial begin

$monitor("A=%b B=%b | Y=%b", A, B, Y);

A = 0; B = 0; #10;

A = 0; B = 1; #10;

A = 1; B = 0; #10;

A = 1; B = 1; #10;

$finish;

end

endmodule

**NAND Gate**

**Introduction**

The NAND (NOT AND) gate outputs 0 only when **both** inputs are 1.

**Truth Table:**

| **A** | **B** | **Y (A NAND B)** |
| --- | --- | --- |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Verilog Code (nand\_gate.sv)**

module nand\_gate (

input logic A,

input logic B,

output logic Y

);

assign Y = ~(A & B);

endmodule

**Testbench (nand\_gate\_tb.sv)**

`timescale 1ns/1ps

module nand\_gate\_tb;

logic A, B;

logic Y;

nand\_gate dut (

.A(A),

.B(B),

.Y(Y)

);

initial begin

$monitor("A=%b B=%b | Y=%b", A, B, Y);

A = 0; B = 0; #10;

A = 0; B = 1; #10;

A = 1; B = 0; #10;

A = 1; B = 1; #10;

$finish;

end

endmodule

**NOR Gate**

**Introduction**

The NOR (NOT OR) gate outputs 1 only when **both** inputs are 0.

**Truth Table:**

| **A** | **B** | **Y (A NOR B)** |
| --- | --- | --- |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Verilog Code (nor\_gate.sv)**

module nor\_gate (

input logic A,

input logic B,

output logic Y

);

assign Y = ~(A | B);

endmodule

**Testbench (nor\_gate\_tb.sv)**

`timescale 1ns/1ps

module nor\_gate\_tb;

logic A, B;

logic Y;

nor\_gate dut (

.A(A),

.B(B),

.Y(Y)

);

initial begin

$monitor("A=%b B=%b | Y=%b", A, B, Y);

A = 0; B = 0; #10;

A = 0; B = 1; #10;

A = 1; B = 0; #10;

A = 1; B = 1; #10;

$finish;

end

endmodule

**XNOR Gate**

**Introduction**

The XNOR (Exclusive NOR) gate outputs 1 when the inputs are **the same**. It’s the opposite of XOR.

**Truth Table:**

| **A** | **B** | **Y (A XNOR B)** |
| --- | --- | --- |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Verilog Code (xnor\_gate.sv)**

module xnor\_gate (

input logic A,

input logic B,

output logic Y

);

assign Y = ~(A ^ B);

endmodule

**Testbench (xnor\_gate\_tb.sv)**

`timescale 1ns/1ps

module xnor\_gate\_tb;

logic A, B;

logic Y;

xnor\_gate dut (

.A(A),

.B(B),

.Y(Y)

);

initial begin

$monitor("A=%b B=%b | Y=%b", A, B, Y);

A = 0; B = 0; #10;

A = 0; B = 1; #10;

A = 1; B = 0; #10;

A = 1; B = 1; #10;

$finish;

end

endmodule