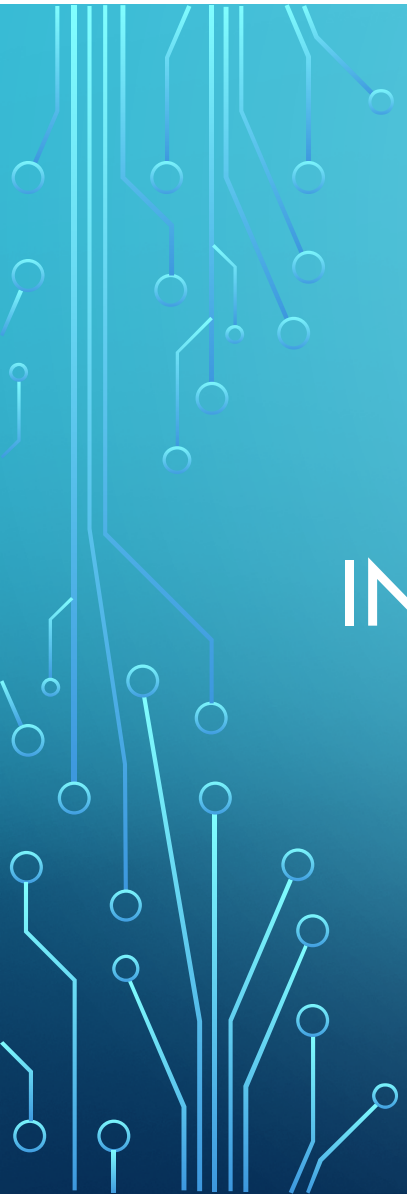


INTRODUCTION TO VERILOG

Gate Level is covered in the examples in this tutorial

VERILOG

Verilog is a hardware design language that allows digital circuits to be described at multiple levels:

- Transistor (or switch) level: the lowest, most detailed way of describing a digital circuit which gives the most accurate results but becomes unwieldy for large circuits and systems.
- Gate level: the next most abstract way of describing a digital circuit, good for low to medium levels of complexity.
- Dataflow
- Behavioral: the highest level of abstraction in describing digital circuits, appropriate for high density or complex digital circuits.

INTRODUCTION TO *VERILOG*

Example 1: The Basic Building Block = MODULE

```
// Compute the logical AND and OR of inputs A and B.
        module AND_OR(andOut, orOut, A, B);
                output logic andOut, orOut;
                input logic A, B;
                and TheAndGate (andOut, A, B);
                or TheOrGate (orOut, A, B);
        endmodule
```

Comment
Everything on a line after // is ignored.

Create a module
This line creates a digital circuit that can be used elsewhere with the name AND_OR; a module is similar to a function in other computing languages. The AND_OR circuit has four signals: andOut, orOut, A, and B. We have not yet identified which of these signals are outputs and which are inputs.

INTRODUCTION TO *VERILOG*

Example 1: The Basic Building Block = MODULE

```
// Compute the logical AND and OR of inputs A and B.
        module AND_OR(andOut, orOut, A, B);
                output logic andOut, orOut;
                input logic A, B;
                and TheAndGate (andOut, A, B);
                or TheOrGate (orOut, A, B);
        endmodule
```

Did you know?
Verilog has three types of ports: input, output, inout

Identify which signals in the module are output ports (i.e. can communicate outside the module) and what type of output they are.

Identify which signals in the module are input ports and what type of inputs those signals are.

data type "logic" and "wire" are interchangeable

INTRODUCTION TO *VERILOG*

Example 1: The Basic Building Block = MODULE

```
// Compute the logical AND and OR of inputs A and B.
        module AND_OR(andOut, orOut, A, B);
                output logic andOut, orOut;
                input logic A, B;
                and TheAndGate (andOut, A, B);
                or TheOrGate (orOut, A, B);
        endmodule
```

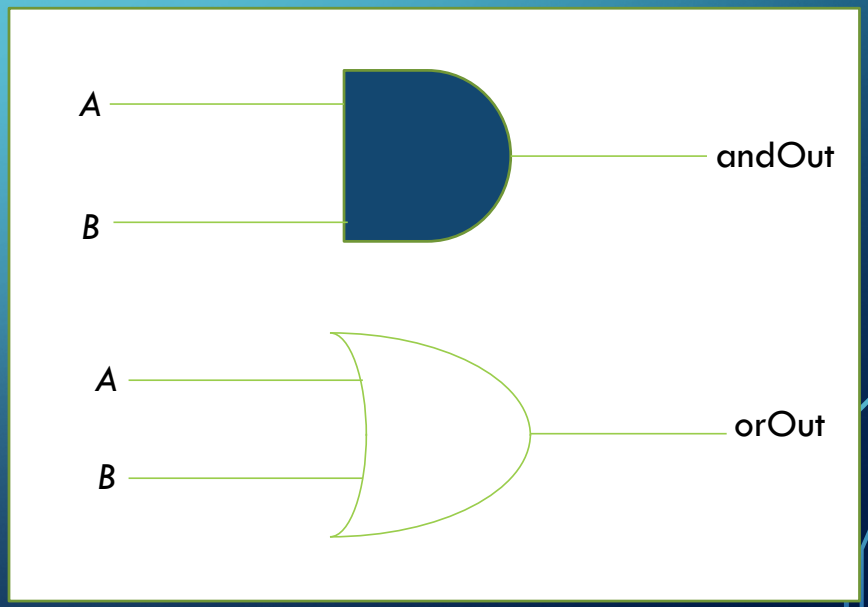
Creates an AND gate whose name is TheAndGate and which has inputs A and B and output andOut.

Creates an OR gate whose name is TheOrGate and which has inputs A and B and output orOut.

INTRODUCTION TO *VERILOG*

Example 1: The Basic Building Block = MODULE

```
// Compute the logical AND and OR of inputs A and B.
        module AND_OR(andOut, orOut, A, B);
                output logic andOut, orOut;
                input logic A, B;
                and TheAndGate (andOut, A, B);
                or TheOrGate (orOut, A, B);
        endmodule
```



INTRODUCTION TO *VERILOG*

Logic Gates in Verilog

buf TheBuffer (OUT1, IN1)

Creates a buffer:
- **Input to the buffer is IN1**
- **Output of the buffer is OUT1**

Relationship between output and input:
- **OUT1 = IN1**

not TheInverter (OUT1, IN1)

Creates an inverter:
- **Input to the inverter is IN1**
- **Output of the inverter is OUT1**

Relationship between output and input:
- **OUT1 = $\overline{IN1}$**

INTRODUCTION TO *VERILOG*

Logic Gates in Verilog

and TheAndGate (OUT1, IN1, IN2)

Creates an AND gate:
- **Inputs to the AND gate are IN1 and IN2**
- **Output of the AND gate is OUT1**

Relationship between output and inputs:
- **OUT1 = IN1 * IN2**

and TheAndGate (OUT1, IN1, IN2, IN3, IN4)

Creates an AND gate:
- **Inputs to the AND gate are IN1, IN2, IN3, IN4**
- **Output of the AND gate is OUT1**

Relationship between output and inputs:
- **OUT1 = IN1 * IN2 * IN3 * IN4**

INTRODUCTION TO *VERILOG*

Logic Gates in Verilog

or TheORGate (OUT1, IN1, IN2)

Creates an OR gate:
- **Inputs to the OR gate are IN1 and IN2**
- **Output of the OR gate is OUT1**

Relationship between output and inputs:
- **OUT1 = IN1 + IN2**

nand TheNANDGate (OUT, IN1, IN2); // a 2-input NAND gate

nor TheNORGate (OUT, IN1, IN2, IN3); // a 3-input NOR gate

xor TheXORGate (OUT, IN1, IN2); //a 2-input XOR gate

xnor TheXNORGate (OUT, IN1, IN2); //a 2-input XNOR gate

INTRODUCTION TO *VERILOG*

Example 2: Using the Basic Building Blocks (Modules)

```
// Compute the logical NAND and NOR of inputs X and Y.
        module NAND_NOR(nandOut, norOut, X, Y);
                output logic nandOut, norOut;
                input logic X, Y;
                logic andVal, orVal;
                AND_OR aoSubmodule (.andOut(andVal), .orOut(orVal), .A(X), .B(Y));
                not n1 (nandOut, andVal);
                not n2 (norOut, orVal);
        endmodule
```

Comment
Everything on a line after // is ignored.

Create a module
This line creates another module NAND_NOR that can be used (instantiated) anywhere else in our code. Note that it uses the AND_OR module that we created in Example 1

INTRODUCTION TO *VERILOG*

Example 2: Using the Basic Building Blocks (Modules)

```
// Compute the logical NAND and NOR of inputs X and Y.
        module NAND_NOR(nandOut, norOut, X, Y);
                output logic nandOut, norOut;
                input logic X, Y;
                logic andVal, orVal;
                AND_OR aoSubmodule (.andOut(andVal), .orOut(orVal), .A(X), .B(Y));
                not n1 (nandOut, andVal);
                not n2 (norOut, orVal);
        endmodule
```

Identifies two outputs from the NAND_NOR circuit: nandOut and norOut and two inputs X and Y

Creates intermediate signals in the module NAND_NOR and calls them andVal and orVal

```
input X;
wire X;
is the same thing as:
input logic X;
```

Note that, in a module, we not only name the input and output signals when we call AND_OR, but we also specify which wires the signals andOut, orOut, A, and B are connected to.

INTRODUCTION TO *VERILOG*

The Test Bench (for simulating in ModelSim)

Once a circuit has been described using Verilog submodules, it is possible to simulate and test it in ModelSim using the test bench.

```
module NAND_NOR_testbench;
        logic X, Y;
        logic nandOut, norOut;
        initial begin // Stimulus
        X = 1; Y = 1; #10;
        X = 0; #10;
        Y = 0; #10;
        X = 1; #10;
        end
        NAND_NOR dut (.nandOut, .norOut, .X, .Y);
endmodule
```

When a test bench is called, note that there are no input or output ports in the module call

Defines the input and output signals of the type logic

Runs the sequence 11, 01, 00, 10 once and waits 10 time units in between each state (XY)

Instantiates the NAND_NOR module as the device under test (dut), hooks up the signals x and y, and enables monitoring of the output signals nandOut and norOut

INTRODUCTION TO *VERILOG*

Testing on the Altera SoC Board

```
module foo (Out1, Out2, A, B);
        output logic Out1, Out2;
        input logic A, B;
        and AndGate (Out1, A, B);
        or OrGate (Out2, A, B);
endmodule

module DE1_SoC (KEY, SW, LEDR, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5);
        input logic [3:0] KEY; // Low when pressed; High when not pressed
        input logic [9:0] SW; // Low when down; High when up
        output logic [9:0] LEDR; // Active High
        output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5; // Active Low
        foo inst1 (.Out1(HEX0[1]), .Out2(LEDR[1]), .A(SW[1]), .B(SW[2]));
endmodule
```

Creates the module foo with two outputs and two inputs

Instantiates foo with the name inst1 on the DE1_SoC board, allowing the inputs A and B to be controlled with Switch 1 and Switch 2 and sending the output Out1 to one of the segments on the first seven segment display (HEX0[1]) and the output Out2 to one of the LEDs (LEDR[1])