Experiment 8

Introduction to Verilog HDL Programming for Gate-Level Description and Dataflow Modeling of Combinational Logic

Objective

- By the end of this lab, students will be able to:
- Understand Gate-Level Modeling
- Use Verilog Primitives Effectively
- Model Combinational Circuits Using Assign Statements
- Develop and Simulate Combinational Circuits
- Analyze and Interpret Simulation Results

Introduction to Verilog HDL

Verilog HDL (Hardware Description Language) is a widely used language for designing and modeling digital electronic systems. It allows engineers and students to describe the behavior and structure of digital circuits at various levels of abstraction, including gate-level, dataflow, and behavioral modeling**.** Using Verilog, complex combinational and sequential circuits can be efficiently implemented, simulated, and verified before actual hardware realization.

In this lab, students will focus on gate-level and dataflow modeling of combinational circuits using Verilog. They will explore Verilog primitives**,** continuous assign statements**,** and understand how logic gates such as AND, OR, NOT, NAND, NOR, XOR, and XNOR can be represented in code. This hands-on approach reinforces the connection between circuit theory and digital design using HDL**.**

The lab exercises are performed using Vivado Design Suite, a professional FPGA design software developed by Xilinx. Vivado provides an integrated environment for writing, simulating, and synthesizing Verilog modules. It allows the creation of modular designs, where each module represents a functional block of the circuit, making complex designs easier to manage and test.

By the end of this lab, students will be able to write Verilog modules for basic combinational circuits, simulate their functionality in Vivado, and understand the use of **modules, ports, and operators** to describe digital systems effectively.

Module Declaration

The language reference manual for the Verilog HDL presents a syntax that describes precisely the constructs that can be used in the language. In particular, a Verilog model is composed of text using keywords, of which there are about 100. Keywords are predefined lowercase identifiers that define the language constructs. Examples of keywords are **module**, **endmodule**, **input**, **output**, **wire**, **and**, **or**, and **not**. For clarity, keywords will be displayed in boldface in the text in all examples of code and wherever it is appropriate to call attention to their use. Any text between two forward slashes (*//*) and the end of the line is interpreted as a comment and will have no effect on a simulation using the model. Multiline comments begin with / * and terminate with * /. Blank spaces are ignored, but they may not appear within the text of a keyword, a user-specified identifier, an operator, or the representation of a number. **Verilog is case sensitive, which means that uppercase and lowercase letters are distinguishable (e.g., not is not the same as NOT).** The term *module* refers to the text enclosed by the keyword pair **module** . . . **endmodule**. A module is the fundamental descriptive unit in the Verilog language. It is declared by the keyword **module** and must always be terminated by the keyword **endmodule**.

Table 1:Logic Gates and Their Verilog Equivalent Symbols

Gate Type	Logic Function	Verilog Primitive (Symbol)	Example Usage
AND Gate	Output is 1 only if all inputs are 1	and	and G1 (Y, A, B);
OR Gate	Output is 1 if any input is 1	or	or G2 (Y, A, B);
NOT Gate (Inverter)	Output is the complement of input	not	not G3 (Y, A);
NAND Gate	Complement of AND output	nand	nand G4 (Y, A, B);
NOR Gate	Complement of OR output	nor	nor G5 (Y, A, B);
XOR Gate	Output is 1 if inputs are different	xor	xor G6 (Y, A, B);
XNOR Gate	Output is 1 if inputs are same	xnor	xnor G7 (Y, A, B);
BUFFER	Output follows the input (used for driving signals)	buf	buf G8 (Y, A);
Tri-State Buffer	Enables/disables output based on control signal	bufif1 / bufif0	bufif1 G9 (Y, A, EN);

Boolean Expressions

Boolean equations describing combinational logic are specified in Verilog with a continuous assignment statement consisting of the keyword **assign** followed by a Boolean expression. To distinguish arithmetic operators from logical operators, Verilog uses the symbols (&), (/), and (&)

for AND, OR, and NOT (complement), respectively. Verilog HDL operators list is given in the Table 2.

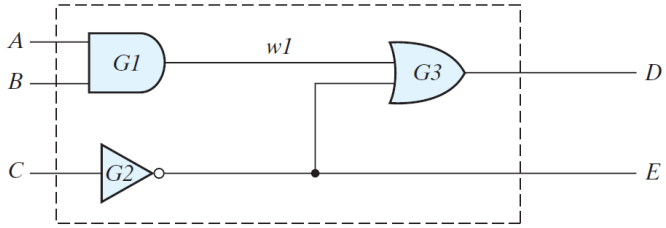
Table 2Verilog HDL Operators

Operator Type	Operator Symbol	Description / Usage	Example
Arithmetic Operators	+	Addition	Y = A + B;
	-	Subtraction	Y = A - B;
	*	Multiplication	Y = A * B;
	/	Division	Y = A / B;
	%	Modulus (remainder)	Y = A % B;
Relational Operators	==	Equal to	Y = (A == B);
	!=	Not equal to	Y = (A != B);
	<	Less than	Y = (A < B);
	>	Greater than	Y = (A > B);
	<=	Less than or equal	Y = (A <= B);
	>=	Greater than or equal	Y = (A >= B);
Logical Operators	&&	Logical AND	Y = A && B;
	`		`
	!	Logical NOT	Y = !A;
Bitwise Operators	&	Bitwise AND	Y = A & B;
	`	`	Bitwise OR
	^	Bitwise XOR	Y = A ^ B;
	~	Bitwise NOT	Y = ~A;
	~^ / ^~	Bitwise XNOR	Y = A ~^ B;
Reduction Operators	&A	AND of all bits	Y = &A;
	`	A`	OR of all bits
	^A	XOR of all bits	Y = ^A;
	~&A	NAND of all bits	Y = ~&A;
	`~	A`	NOR of all bits
	~^A / ^~A	XNOR of all bits	Y = ~^A;
Shift Operators	<<	Left shift	Y = A << 2;
	>>	Right shift	Y = A >> 2;
	<<<	Arithmetic left shift	Y = A <<< 2;
	>>>	Arithmetic right shift	Y = A >>> 2;
Concatenation / Replication	{}	Concatenate bits	Y = {A,B};
Conditional / Ternary	? :	If-else expression	Y = (A > B) ? A : B;

Note: please visit the following link for understanding how to create your first program in Xilinx Vivado

https://www.youtube.com/watch?v=sA5YEIFzCOw&t=823s

Consider the following circuit diagram.



The Verilog code for circuit is:

```
module Simple_Circuit (A, B, C, D, E);
output D, E;
input A, B, C;
wire w1;
and G1 (w1, A, B); // Optional gate instance name
not G2 (E, C);
or G3 (D, w1, E);
endmodule
```

Task 1:

Run the above code and show the test bench waveforms for it. [2]

Task 2:

Now, consider the Boolean expression given below. Write down the Verilog module for it. [2]

$$E = A + BC + B\_D$$

$$F = B'C + BC'D'$$

The equations specify how the logic values *E* and *F* are determined by the values of *A*, *B*, *C*, and *D*.

//Verilog model: Circuit with Boolean expressions

```
module Circuit_Boolean_CA (E, F, A, B, C, D);
output E, F;
input A, B, C, D;
assign E =A || (B && C) || ((!B) && D);
assign F =((!B) && C) || (B && (!C) && (!D));
endmodule
```

Run the above code and show the test bench waveforms for it.

Lab Exercise (20 Marks)

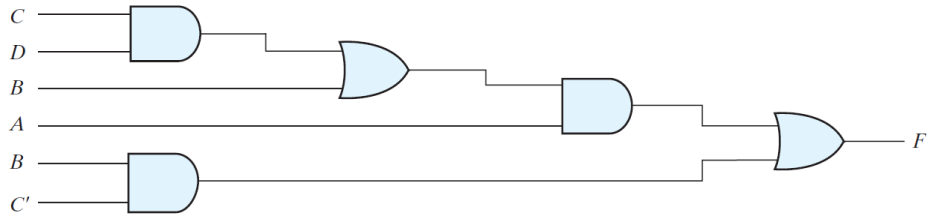
Find the syntax errors in the following declarations (note that names for primitive gates are optional): [2]

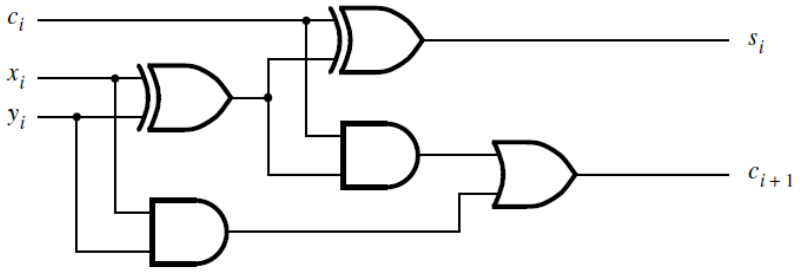
module Exmpl-3(A, B, C, D, F) // Line 1
inputs A, B, C, Output D, F, // Line 2
output B // Line 3
and g1(A, B, D); // Line 4
not (D, A, C), // Line 5
OR (F, B; C); // Line 6
endmodule; // Line 7

Draw the logic diagram of the digital circuit specified by the following Verilog description: [3]

(a) **module** Circuit_A (A, B, C, D, F);
input A, B, C, D;
output F;
wire w, x, y, z, a, d;
or (x, B, C, d);
and (y, a ,C);
and (w, z ,B);
and (z, y, A);
or (F, x, w);
not (a, A);
not (d, D);
endmodule

Write a Verilog code for gate-level description using primitives of the circuit shown in figures below. Determine the Boolean Function Equation and then write code using assign statement. Also show test bench waveforms for it. [5+5]




c_i
x_i
y_i
s_i
c_{i+1}

Using continuous assignments, write a Verilog description of the circuit specified by the following Boolean functions. Test the outputs using test bench waveforms. [5]

$$Out\_1 = (A + B')C'(C + D)$$
$$Out\_2 = (C'D + BCD + CD')(A' + B)$$
$$Out\_3 = (AB + C)D + B'C$$