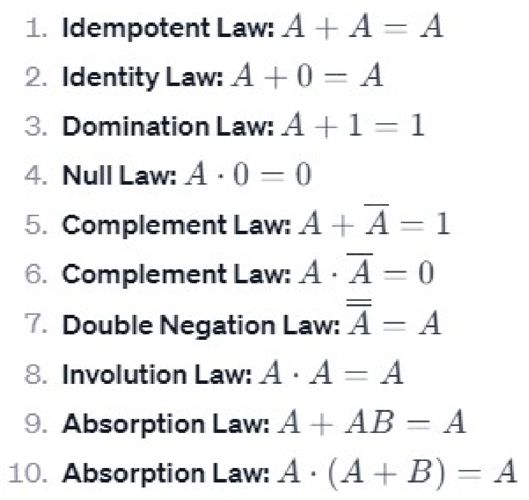
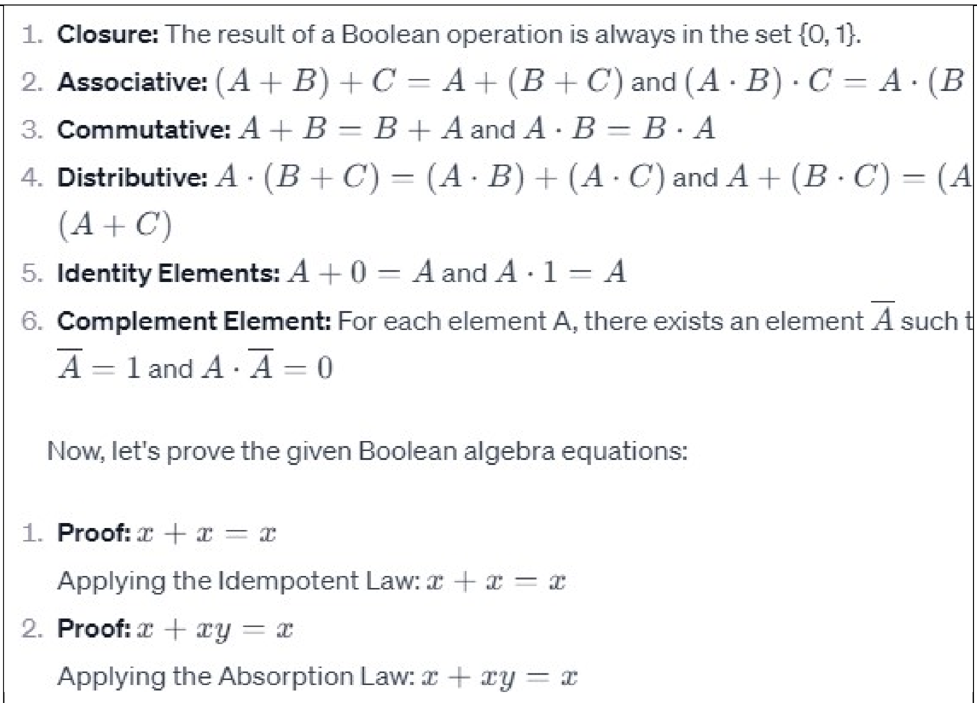
USN										

Internal Assessment Test I –Nov. 2021

Sub :	Digital Design and Computer Organization					Sub Code:	BCS302		Branch:	AIML
Date :	19122023	Duration:	90 mins	Max Marks:	50	Sem Sec:	B	Time	12.15 – 1.45PM	OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	List basic theorem of boolean algebra and its postulates Prove: x+x=x x+xy=x  1. **Idempotent Law:** $A + A = A$ 2. **Identity Law:** $A + 0 = A$ 3. **Domination Law:** $A + 1 = 1$ 4. **Null Law:** $A \cdot 0 = 0$ 5. **Complement Law:** $A + \overline{A} = 1$ 6. **Complement Law:** $A \cdot \overline{A} = 0$ 7. **Double Negation Law:** $\overline{\overline{A}} = A$ 8. **Involution Law:** $A \cdot A = A$ 9. **Absorption Law:** $A + AB = A$ 10. **Absorption Law:** $A \cdot (A + B) = A$	[10]	CO1	L4

	 1. **Closure:** The result of a Boolean operation is always in the set {0, 1}. 2. **Associative:** $(A+B)+C = A+(B+C)$ and $(A \cdot B) \cdot C = A \cdot (B$ 3. **Commutative:** $A+B = B+A$ and $A \cdot B = B \cdot A$ 4. **Distributive:** $A \cdot (B+C) = (A \cdot B)+(A \cdot C)$ and $A+(B \cdot C) = (A$ $(A+C)$ 5. **Identity Elements:** $A+0 = A$ and $A \cdot 1 = A$ 6. **Complement Element:** For each element A, there exists an element $\overline{A}$ such t $\overline{A} = 1$ and $A \cdot \overline{A} = 0$ Now, let's prove the given Boolean algebra equations: 1. **Proof:** $x + x = x$ Applying the Idempotent Law: $x + x = x$ 2. **Proof:** $x + xy = x$ Applying the Absorption Law: $x + xy = x$ L.H.S. X+X =(X+X).1 [∵ X.1 = X] =(X+X)(X+X') [∵ X+X' = 1] =(X+X.X') [∵ X+YZ = (X+Y)(X+Z)] =X+0 [∵ X.X' = 0] =X [∵ X+0 = X] = R.H.S. L.H.S. X+XY =X.1+XY [∵ X.1 = X] =X(1+Y) [∵ X(Y+Z) = XY+XZ] =X(Y+1) =X.1 =X [∵ X+1 = X] = R.H.S.			
2a	Explain canonical and standard form with an example **Canonical Form:** Canonical form refers to a standard or unique representation of a logical expression in Boolean algebra. It is a way of expressing Boolean functions in a standard format that is independent of any particular variable names or ordering. There are two common types of canonical forms: the Sum of Products (SOP) and the Product of Sums (POS). 1. **Sum of Products (SOP) Canonical Form:** In SOP, a Boolean expression is represented as the sum (OR) of products (AND). Each term in the sum is a product of literals (variables or their complements). Here's an example: Let's consider a Boolean function (F(A, B, C) = AB + overline{A}C). The SOP canonical	[10]	CO1	L2

form is (F(A, B, C) = AB + overline{A}C).

2. **Product of Sums (POS) Canonical Form:**

In POS, a Boolean expression is represented as the product (AND) of sums (OR). Each term in the product is a sum of literals. An example is:

Let (G(A, B, C) = (A + B)(overline{A} + C)). The POS canonical form is (G(A, B, C) = (A + B)(overline{A} + C)).

Standard Form:

The term "standard form" typically refers to a specific representation or format that is commonly used or recognized. In Boolean algebra, standard form often refers to expressions where each term has all variables present (either directly or complemented) in a consistent order.

For example, consider the Boolean expression (F(A, B, C) = AB + overline{A}C). In standard form, we might rewrite it as (F(A, B, C) = ABC + overline{A}BC), ensuring that each term includes all variables in a consistent order.

In summary, canonical form is a representation that is unique and independent of variable names, while standard form often implies a specific and commonly accepted representation with consistent variable ordering. The distinction can sometimes blur, and the terms might be used interchangeably depending on the context.

2b Explain digital logic gates with graphic symbol, boolean function and truth table. [10] CO1 L2

Digital logic gates are fundamental building blocks of digital circuits, performing logical operations on binary inputs to produce binary outputs. There are several types of basic logic gates, each with its own graphic symbol, Boolean function, and truth table. Here are some common logic gates:

1. **AND Gate:**
- **Graphic Symbol:**
```

|     |
| AND  |
||
```
- **Boolean Function:** (Y = A cdot B)
- **Truth Table:**
```
| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
```

2. **OR Gate:**
- **Graphic Symbol:**
```

|      |
|  OR  |
||
```
- **Boolean Function:** (Y = A + B)
- **Truth Table:**
```
| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |
```

3. **NOT Gate (Inverter):**
- **Graphic Symbol:**
```

|      |
|  NOT |
||
```
- **Boolean Function:** (Y = overline{A})
- **Truth Table:**
```
| A | Y |
|---|---|
| 0 | 1 |
| 1 | 0 |
```

4. **NAND Gate:**
- **Graphic Symbol:**
```

|      |
| NAND |
||
```
- **Boolean Function:** (Y = overline{A cdot B})
- **Truth Table:**
```
| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
```

5. **NOR Gate:**
- **Graphic Symbol:**
```
```

| |
| NOR |
||
```
- **Boolean Function:** (Y = overline{A + B})
- **Truth Table:**
```
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0
```

These gates form the basis for designing complex digital circuits. The Boolean functions and truth tables provide a clear understanding of how each gate processes its inputs to produce an output.

3 Implement the logic operations with NAND gates and implement the following boolean function with NAND gates: F(x,y,z)=(1,2,3,4,5,7) [10] CO1 L4

NAND gates are universal gates, meaning that any other logic gate or boolean function can be implemented using only NAND gates. Here's how you can implement the boolean function ( F(x, y, z) ) using NAND gates:

The boolean function ( F(x, y, z) ) is specified by the set ({1, 2, 3, 4, 5, 7}). The numbers in this set correspond to the minterms in the truth table for ( F(x, y, z) ). Let's create the truth table:

```
x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Now, let's implement this boolean function using only NAND gates. Each row in the truth table where (F = 1) corresponds to a minterm that needs a NAND gate. The idea is to take the inputs for which (F = 1), feed them through NAND gates, and then combine the outputs of these NAND gates using additional NAND gates.

Here's one possible implementation using NAND gates:

```plaintext
F(x, y, z) = (x NAND x) NAND (y NAND y) NAND (z NAND z) NAND (x NAND y) NAND
(x NAND z) NAND (y NAND z) NAND (x NAND y NAND z)
```

In this expression:
- (x NAND x), (y NAND y), and (z NAND z) ensure that the minterms where only one variable is true are considered.
- (x NAND y), (x NAND z), and (y NAND z) handle the minterms where two variables are true.
- (x NAND y NAND z) takes care of the minterm where all three variables are true.

	This expression consists of a series of NAND gates cascaded to implement the specified boolean function (F(x, y, z)).			
4(a)	Simplify the following boolean functions to minimize number of literals x+x’y xy+x’z+yz Let's simplify the given Boolean functions: 1. **(x + x'y):** Applying the Absorption Law ((A + AB = A)), we can simplify this expression: [x + x'y = x(1 + y) = x] So, (x + x'y) simplifies to (x). 2. **(xy + x'z + yz):** This expression has three terms, and it's not as straightforward as the first one. We can simplify it using Boolean algebra laws. [xy + x'z + yz] We can factor out (y) from the first and third terms: [y(x + z) + x'z] Now, we can apply the Absorption Law ((A + AB = A)) to the first term: [y + x'z] This expression doesn't simplify further. So, the simplified forms are: 1. (x + x'y) simplifies to (x). 2. (xy + x'z + yz) doesn't simplify further.	[5]	CO1	L3
4 (b)	Express the boolean function F=A+B’C as sum of minterms To express the boolean function (F = A + B'C) as the sum of minterms, we first need to identify the minterms for which the function is true (evaluates to 1). The boolean function (F = A + B'C) is already in a form where each term is a minterm. The minterms can be identified by looking at the combinations of inputs (A, B, C) that make each term true. Here are the minterms for (F = A + B'C): 1. (A = 1, B' = 0, C = 0) (corresponding to the term (A)) 2. (A = 0, B' = 1, C = 0) (corresponding to the term (B'C)) So, the boolean function (F = A + B'C) can be expressed as the sum of minterms: [F = m1 + m2] where (m1) corresponds to the minterm (A) and (m2) corresponds to the minterm (B'C).	[05]	CO1	L3
5	Simplify the boolean function F(w,x,y,z)=(1,3,7,11,15) which has the dont care conditions (w,x,y,z)=(0,2,5) To simplify the boolean function (F(w, x, y, z) = {1, 3, 7, 11, 15}) with don't care conditions (	[10]	CO1	L3

{(w, x, y, z) = (0, 2, 5)}), we can use a Karnaugh map. A Karnaugh map is a graphical method used for simplifying boolean expressions.

Here is the truth table based on the given minterms and don't care conditions:

```
| w | x | y | z | F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 |   |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 |   |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |   |
| 0 | 1 | 0 | 1 |   |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |   |
| 1 | 0 | 0 | 0 |   |
| 1 | 0 | 0 | 1 |   |
| 1 | 0 | 1 | 0 |   |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |   |
| 1 | 1 | 0 | 1 |   |
| 1 | 1 | 1 | 0 |   |
| 1 | 1 | 1 | 1 | 1 |
```

Now, let's create a Karnaugh map for this truth table:

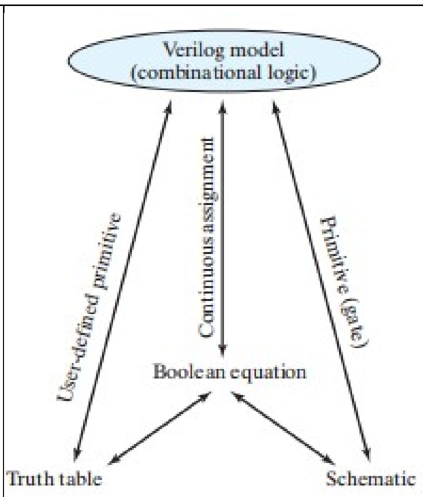
```
| 00 | 01 | 11 | 10 |
______|____|____|____|____|
00   |    |    |    |    |
______|____|____|____|____|
01   |    |    |    |    |
______|____|____|____|____|
11   |    |    |    |    |
______|____|____|____|____|
10   |    |    |    |    |
______|____|____|____|____|
```

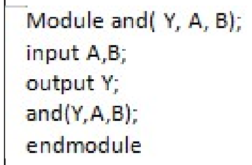
Now, fill in the cells corresponding to the minterms with 1:

```
| 00 | 01 | 11 | 10 |
______|____|____|____|____|
00   |    |    |    |    |
______|____|____|____|____|
01   |    |    |    |    |
______|____|____|____|____|
11   | 1 | 1 | 1 |    |
______|____|____|____|____|
10   |    |    |    |    |
______|____|____|____|____|
```

Group adjacent 1s in powers of 2 (1, 2, 4, 8, ...). In this case, you can form two groups: one with 1s in cells (1, 3) and another with 1s in cell (7). The don't care condition cells (0, 2, 5) can

	be used to further simplify the expression. Now, write down the simplified expression: [F(w, x, y, z) = Sigma (1, 3, 7) + d(0, 2, 5)] Where (Sigma) denotes the sum of minterms and (d) denotes the don't care conditions. The simplified boolean expression is: [F(w, x, y, z) = bar{w}z + wxy + bar{w}x] So, (F(w, x, y, z)) is simplified to (bar{w}z + wxy + bar{w}x).			
6	**Explain 4 levels of programming abstractions provided by Verilog HDL. Use AND gate realization as an example at Gate Level, Data Flow Level and Behavioral Level respectively.** The Verilog HDL was introduced in Section 3.10. In the current section, we introduce additional features of Verilog, present more elaborate examples, and compare alternative descriptions of combinational circuits in Verilog. Sequential circuits are presented in As mentioned previously, the module is the basic building block for modeling hardware with the Verilog HDL. The logic of a module can be described in any one (or a combination) of the following modeling styles: • Gate-level modeling using instantiations of predefined and user-defined primitive gates. • Dataflow modeling using continuous assignment statements with the keyword assign . • Behavioral modeling using procedural assignment statements with the keyword always . Gate-level (structural) modeling describes a circuit by specifying its gates and how they are connected with each other. Dataflow modeling is used mostly for describing the Boolean equations of combinational logic. We'll also consider here behavioral modeling that is used to describe combinational and sequential circuits at a higher level of abstraction. Combinational logic can be designed with truth tables, Boolean equations, and schematics; Verilog has a construct corresponding to each of these "classical" approaches to design: user-defined primitives, continuous assignments, and primitives, as shown in Fig. 4.31 . There is one other modeling style, called switch-level modeling. It is sometimes used in the simulation of MOS transistor circuit models, but not in logic synthesis. We will not consider switch-level modeling. **<u>Gate-Level Modeling</u>** Gate-level modeling was introduced in Section 3.10 with a simple example. In this type of representation, a circuit is specified by its logic gates and their interconnections. Gate level modeling provides a textual description of a schematic diagram. The Verilog HDL includes 12 basic gates as predefined primitives. Four of these primitive gates are of the three-state type. The other eight are the same as the ones listed in Section 2.8. They are all declared with the lowercase keywords and, nand, or, nor, xor, xnor, not, and buf, Primitives such as and are n -input primitives	[10]	CO1	L3

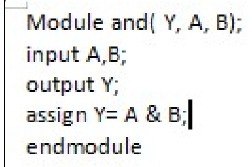




```
Module and( Y, A, B);
input A,B;
output Y;
and(Y,A,B);
endmodule
```

Dataflow Modeling

Dataflow modeling of combinational logic uses a number of operators that act on binary operands to produce a binary result. Verilog HDL provides about 30 different operators. It is necessary to distinguish between arithmetic and logic operations, so different symbols are used for each. The plus symbol 1+2 indicates the arithmetic operation of addition; the bitwise logic AND operation (conjunction) uses the symbol &. There are special symbols for bitwise logical OR (disjunction), NOT, and XOR. The equality symbol uses two equals signs (without spaces between them) to distinguish it from the equals sign used with the assign statement. The bitwise operators operate bit by bit on a pair of vector operands to produce a vector result. The concatenation operator provides a mechanism for appending multiple operands.

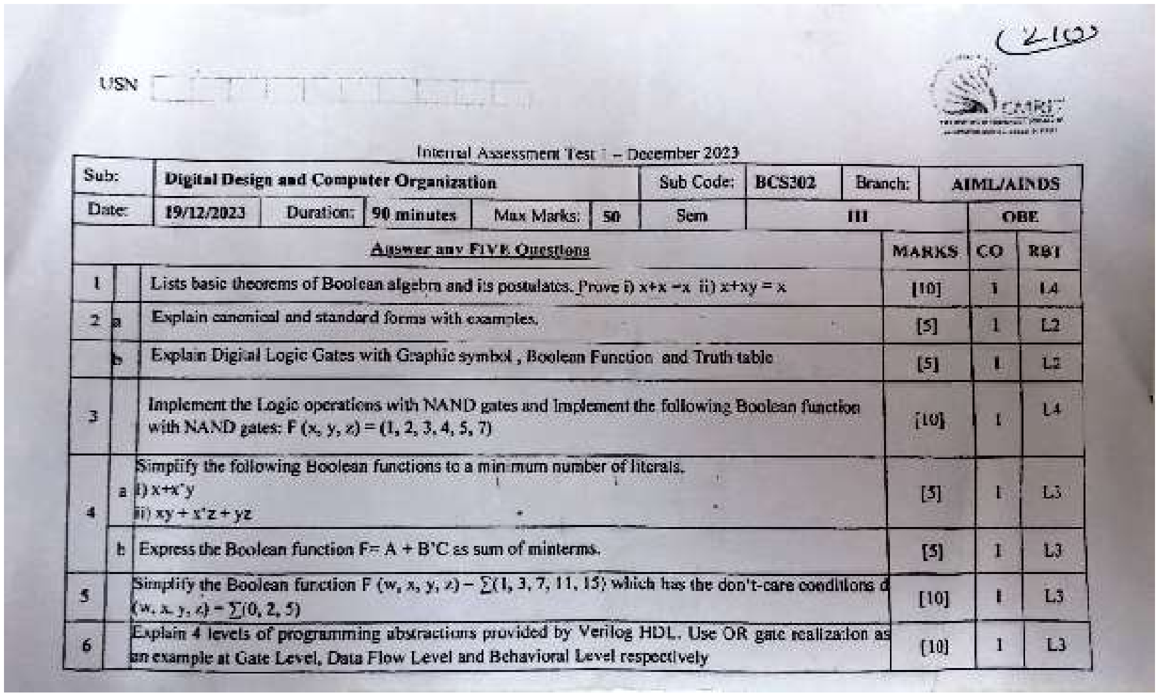


```
Module and( Y, A, B);
input A,B;
output Y;
assign Y= A & B;
endmodule
```

Behavioral Modeling

Behavioral modeling represents digital circuits at a functional and algorithmic level. It is used mostly to describe sequential circuits, but can also be used to describe combinational circuits. Here, we give two simple combinational circuit examples to introduce the subject. Behavioral modeling is presented in more detail in Section 5.6, after the study of sequential circuits. Behavioral descriptions use the keyword always , followed by an optional event control expression and a list of procedural assignment statements. The event control expression specifies when the statements will execute. The target output of a procedural assignment statement must be of the reg data type. Contrary to the wire data type, whereby the target output of an assignment may be continuously updated, a reg data type retains its value until a new value is assigned.

```
module and( Y, A, B);
input A,B;
output Y;
always @ (A or B)
begin
  if (A == 1'b1 & B == 1'b1)
begin
     Y = 1'b1;
  end
  else
     Y = 1'b0;
end
endmodule
```



(2-10)

USN

Internal Assessment Test 1 – December 2023

Sub:	Digital Design and Computer Organization				Sub Code:	BCS302	Branch:	AIML/AINDS
Date:	19/12/2023	Duration:	90 minutes	Max Marks: 50	Sem	III	OBE	

Answer any FIVE Questions

			MARKS	CO	RBT
1		Lists basic theorems of Boolean algebra and its postulates. Prove i) x+x =x ii) x+xy = x	[10]	1	L4
2	a	Explain canonical and standard forms with examples.	[5]	1	L2
	b	Explain Digital Logic Gates with Graphic symbol , Boolean Function and Truth table	[5]	1	L2
3		Implement the Logic operations with NAND gates and Implement the following Boolean function with NAND gates: F (x, y, z) = (1, 2, 3, 4, 5, 7)	[10]	1	L4
4	a	Simplify the following Boolean functions to a minimum number of literals. i) x+x'y ii) xy + x'z + yz	[5]	1	L3
	b	Express the Boolean function F= A + B'C as sum of minterms.	[5]	1	L3
5		Simplify the Boolean function F (w, x, y, z) = ∑(1, 3, 7, 11, 15) which has the don't-care conditions d (w, x, y, z) = ∑(0, 2, 5)	[10]	1	L3
6		Explain 4 levels of programming abstractions provided by Verilog HDL. Use OR gate realization as an example at Gate Level, Data Flow Level and Behavioral Level respectively	[10]	1	L3