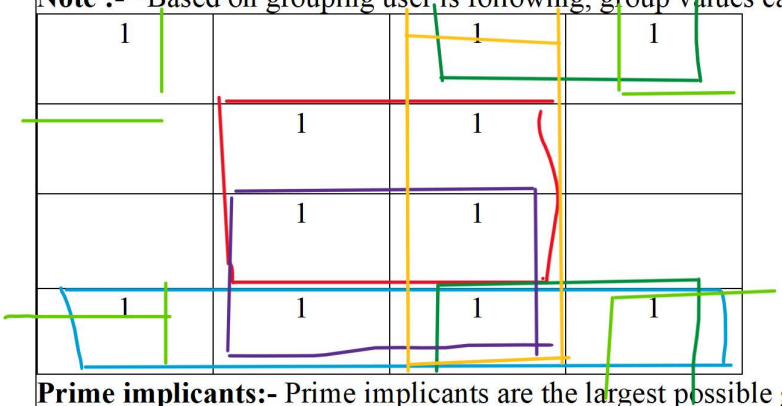
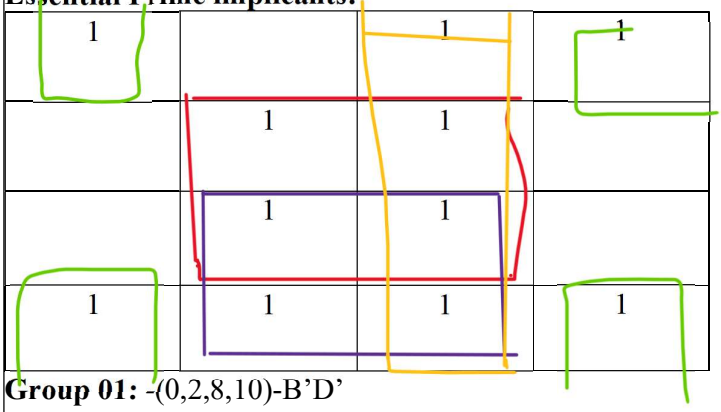
USN | | | | | | | | | | |

Internal Assessment Test 1 – November 2025

Sub:	Digital Design and Computer Organization					Sub Code:	BCS302	Branch:	AIML	
Date:	04/11/24	Duration:	90 minutes	Max Marks:	50	Sem/Sec:	III			OBE

		Answer any FIVE FULL Questions	MARKS	CO	RBT
1	a	Find all the prime implicants for the following Boolean functions, and determine which are essential. Also simplify the functions: F (w, x, y, z) = (0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15) **Evaluation Distribution:-** 2 marks for K-map, 4 marks for Grouping, 2 marks for prime implicants, 2 marks for essential prime implicants. **Note :-** *Based on grouping user is following, group values can change.  **Prime implicants:-** Prime implicants are the largest possible groups of 1's in the Karnaugh map that can be combined. Group 01:- (8,9,10,11)->AB' Group 02:- (5,7,13,15) ->BD Group 03:-(0,2,8,10)-> B'D' Group 04:- (2,3,10,11)->B'C Group 05:- (9,11,13,15)-> AD Group 06:- (3,7,11,15)->CD **Essential Prime implicants:-**  **Group 01:** -(0,2,8,10)-B'D' **Group 02:** -(5,7,13,15)->BD Group 03:- (9,11,13,15)-> AD Group 04:- (3,7,11,15)->CD	[10]	1	L3
2	a	**Give SOP and POS circuit for F(A,B,C,D) = ∑m (6,8,9,10,11,12,13,14,15)** **Evaluation distribution:-** 2 Marks for K-map, 2 marks for SOP equation, 2 marks for POS equation, 2 marks for SOP circuit, 2 marks For POS circuit. **Note :-** *Based on grouping user is following, group values can change.	[10]	2	L3

To find the Sum of Products (SOP) and Product of Sums (POS) expressions for the Boolean function $F(A,B,C,D) = \sum m(6,8,9,10,11,12,13,14,15)$, let's break down the problem step by step.

Step 1: Set up the K-map

We'll plot these minterms on a 4-variable Karnaugh Map (K-map). The variables are A, B, C, D.

Here's the K-map layout:

			1
1	1	1	1
1	1	1	1

Step 2: Identify the Groups

Group 01:- (8,9,10,11,12,13,14,15)-A

Group 02 :- (6,14)-BCD' [A'BCD' If you are keeping 6th individually]

Step 3: Sum of Products (SOP)

From the prime implicants identified in the K-map, we can now write the Sum of Products (SOP) expression by OR'ing the prime implicants together.

The SOP expression is:

$$F(A,B,C,D) = A + BCD'$$

Step 4: Product of Sums (POS)

To find the Product of Sums (POS) expression, we need to write the K-map as groups of 0's instead of 1's. First, we identify the 0's in the K-map:

$AB\backslash CD$	00	01	11	10
00	1	1	1	1
01	1	1	1	0
11	0	0	0	0
10	0	0	0	0

Now we need to group the 0's in the K-map:

- Group 1: (0,1,4,5)->A+C
- Group 2: (1,5,7,9)A+D'
- Group 3: (0,1,2,3)A+B

Step 5: Final POS Expression

To find the POS expression, we combine the groups we just identified. The POS expression is:

$$F(A,B,C,D) = (A + C)(A + D')(A + B)$$

3	a	**3A digital system is to be designed in which the months of the year is given as input in four-bit form. The month January is represented as '0000' and so on. The outputs of the system should be '1' if the month contains 30 days, else 0. Consider the excess number in the input beyond '1011' as don't care conditions. Find the following: -** 1. Give the truth table and simplify by using the K-map. 2. Boolean expression in ∑m and ∏M form 3. Implement the simplified equation using NAND-NAND gates. **Evaluation Distribution:-** 4 (2+2) marks for k-map and simplification, 2(1+1) marks for SOP and POS form, 4 for NAND-NAND gates. **3.(1)** **Truth Table**	[10]	2	L3

Month	No of days	A	B	C	D	X

		January	31	0	0	0	0	0			
		February	28/29	0	0	0	1	0			
		March	31	0	0	1	0	0			
		April	30	0	0	1	1	1			
		May	31	0	1	0	0	0			
		June	30	0	1	0	1	1			
		July	31	0	1	1	0	0			
		August	31	0	1	1	1	0			
		September	30	1	0	0	0	1			
		October	31	1	0	0	1	0			
		November	30	1	0	1	0	1			
		December	31	1	0	1	1	0			
				1	1	0	0	X			
				1	1	0	1	X			
				1	1	1	0	X			
				1	1	1	1	X			

**Note :- ***Based on grouping user is following, group values can change.

K-map

		1	
	1		
X	X	X	X
1			1

Group 01:- (8,10,12,14)-> AD'
Group 02:- (5,13)->BC'D
Group 03:- 3 -> A'B'CD
F(A,B,C,D) = AB'D'+BC'D+A'B'CD

3. (2)

F(A,B,C,D) = ∑m(3,5,8,10) + d(12,13,14,15) and ∏M
F(A,B,C,D) = ∏M (0,1,2,4,6,7,9,11) + d(12,13,14,15)

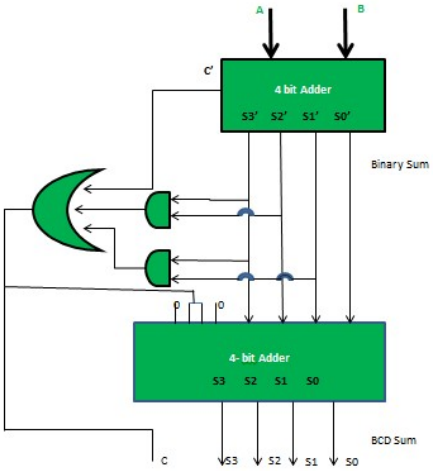
4	a	**Write short notes on the BCD Adder.** **Evaluation Distribution: -**At least 5 components (such as Definition, working principal, logical diagram, advantages, disadvantages, applications etc.), 2 marks for each component. A **BCD Adder** (Binary Coded Decimal Adder) is a digital circuit used to add two decimal numbers that are represented in **Binary Coded Decimal (BCD)** format. In BCD representation, each decimal digit (0–9) is represented by its 4-bit binary equivalent. **Need for a BCD Adder:** • Ordinary binary addition does not always give a valid BCD result. • A valid BCD digit must be in the range **0000 (0)** to **1001 (9)**. • When the binary sum exceeds 1001 (i.e., decimal 9), it becomes an **invalid BCD number**, and a correction must be applied. **Working Principle:** 1. **Step 1: Add the two BCD digits** (including any carry from a previous stage) using a 4-bit binary adder. 2. **Step 2: Check for correction condition.** o If the 4-bit sum ≤ 1001 (9), it is a valid BCD result.	[10]	1	L2

- If the sum > 1001 (i.e., 1010–1111) or a carry out from the 4-bit adder occurs, then **add 6 (0110)** to the sum to correct it.

3. **Step 3: Generate carry to the next decimal digit** if a correction was made.

Logic:- $C' + S3'.S2' + S3'.S1' = 1$

Block Diagram:



Advantages:

- Provides **accurate decimal arithmetic** in digital systems.
- Ensures compatibility between **binary processing** and **decimal display**.
- **Decimal Precision:** BCD adders guarantee that when adding decimal numbers, they do not make mistakes since the process is conducted on digits that are Binary Coded Decimal direct (0-9).
- **Simplified Decimal Arithmetic:** When it comes to decimal arithmetic operations, BCD adders offer computerized systems with an easier way out making them fit for fields.
- **Common Display Compatibility:** The common display technologies such as 7-segment displays are directly compatible with BCD numbers
- **Mistake Recognition:** Just a simple addition is all that is required by such devices so as to find out the parity of invalid BCDs (for instance those larger than digit 9), making it easier for FEC systems. In this manner it forms part of an error detection system and correction scheme that ensure precision results.
- **High-Efficiency Circuit Design:** BCD adders facilitate the creation of efficient, optimized circuits specifically designed for decimal arithmetic, which results in speedier processing times and less complicated digital circuits.

These benefits show how critical BCD adders are in processing decimal arithmetic using digital logic well and correctly.

Disadvantages of BCD Adder

- **Memory Misallocation:** In comparison to binary digits, the BCD figures take up more memory to portray comparable values, hence generating greater memory use within BCD operational systems.

		• **Restricted Set of Values:** BCD adders are constrained to only decimal digits (0-9) hence cannot carry out direct arithmetic on values that are beyond this range without extra conversion circuitry thus restricting their versatility in some applications. • **Lower Speed of Arithmetic Operations:** Since they require BCD correction and manage decimal numbers, BCD adders may have lower operational speeds than binary ones affecting the overall performance of digital systems. • **Compatibility concerns:** BCD arithmetic could be at odds with some techniques or algorithms especially those that are improved to perform better in binary arithmetic; hence you get such compatibility problems when using both types of arithmetic within a system. • **High Circuit Complexity:** BCD adders are more complicated than binary adders owing to BCD correction logic requirements that make sure valid BCD outputs are produced. This increased complexity can also lead to bigger circuit sizes as well as more difficult designs.			

5	a	**Design a Verilog code to implement 2:1 and 4:1 multiplexer**
		Evaluation Distribution: - 5(2*2.5) marks,2.5 marks for each multiplexer, in each multiplexer 0.5 marks for equations, 2 marks for code for each multiplexer.
		Verilog Code for 2:1 multiplexer
		The logic equation:
		$$Y = S?I1:I0$$
		or equivalently,
		$$Y = (\bar{S} \cdot I0) + (S \cdot I1)$$
		Module code
		`module mux2to1_gate (`
		`  input I0, I1, S,`
		`  output Y`
		`);`
		`  wire Sbar, a1, a2;`
		
		`  not (Sbar, S);`
		`  and (a1, I0, Sbar);`
		`  and (a2, I1, S);`
		`  or  (Y, a1, a2);`
		`endmodule`
		
		TESTBENCH
		`module tb_mux2to1;`
		`  reg I0, I1, S;`
		`  wire Y;`
		
		`  mux2to1 uut (.I0(I0), .I1(I1), .S(S), .Y(Y));`
		`  initial begin`
		`  $display("S I1 I0 | Y");`
		`  $monitor("%b  %b  %b  | %b", S, I1, I0, Y);`
		`  I0=0; I1=0; S=0; #10;`
		`  I0=0; I1=1; S=0; #10;`
		`  I0=1; I1=0; S=1; #10;`
		`  I0=1; I1=1; S=1; #10;`
		`  $stop;`
		`  end`
			[05]	3	L3

```
endmodule
```

Verilog Code for 4:1 multiplexer

The logic equation:

$$Y = (\bar{S1}\bar{S0}I0) + (\bar{S1}S0I1) + (S1\bar{S0}I2) + (S1S0I3)$$

Module file

```
module mux4to1_gate (
   input I0, I1, I2, I3,
   input S1, S0,
   output Y
);
   wire S1bar, S0bar;
   wire and0, and1, and2, and3;

   not (S1bar, S1);
   not (S0bar, S0);

   and (and0, I0, S1bar, S0bar);
   and (and1, I1, S1bar, S0);
   and (and2, I2, S1, S0bar);
   and (and3, I3, S1, S0);

   or (Y, and0, and1, and2, and3);
endmodule
```

TESTBENCH

```
module tb_mux4to1;
   reg [3:0] I;
   reg [1:0] S;
   wire Y;

   mux4to1 uut (.I(I), .S(S), .Y(Y));
   initial begin
      $display("S1 S0 | I3 I2 I1 I0 | Y");
      $monitor("%b  %b  | %b  %b  %b  %b | %b",
            S[1], S[0], I[3], I[2], I[1], I[0], Y);

      I = 4'b1010;
      S = 2'b00; #10;
      S = 2'b01; #10;
      S = 2'b10; #10;
      S = 2'b11; #10;

      $stop;
   end
endmodule
```

	b	**Write the difference between combinational circuits and sequential circuits.** **Evaluation Distribution**: - At least 5 key differences. 1 marks for each.	[05]	3	L2

Aspect	**Combinational Circuit**	**Sequential Circuit**
Definition	Output depends only on the current inputs.	Output depends on both current inputs and past states (memory).

		Memory Elements	Does not require memory elements.	Requires memory elements like flip-flops or latches.			
		Timing Dependency	Output is immediate, based on input changes.	Output is dependent on clock pulses and previous states.			
		Clock Signal	No clock signal required.	Requires a clock signal to synchronize state changes.			
		Design Complexity	Simpler design without the need for memory.	More complex due to memory and clock management.			
		Speed	Faster, as outputs change instantly with inputs.	Slower due to dependency on clock cycles and past states.			
		Functionality	Performs basic logical operations without sequence dependency.	Performs operations that require sequences or timed events.			
		Examples	Adders, Subtractors, Multipl exers, Encoders.	Counters, Shift Register, Flip-Flops, State Machines.			
		Power Consumption	Generally lower power consumption.	Higher power consumption due to memory and clock circuitry.			
		Application	Used in tasks requiring direct logical operations (e.g., arithmetic).	Used in applications involving sequential operations (e.g., counters, registers).			

6	a	**Explain the following: (i) Adder (ii) Subtractor** **Evaluation Distribution**: -5 marks for each. Each evaluation on 5 components (definition, types, equations, diagram, applications etc.) 1 mark for each component. **(i) ADDER: -**An **Adder** is a **combinational logic circuit** that performs the arithmetic operation of **addition** of binary numbers. Adders are fundamental components used in **digital systems**, such as microprocessors, ALUs (Arithmetic Logic Units), and digital calculators. **Types of Adders:** **1. Half Adder** • A **Half Adder** adds two 1-bit binary numbers (A and B). • It produces two outputs: o **SUM (S) →** represents the least significant bit of the result. o **CARRY (C) →** represents the carry-out bit.	[10]	3	L2

Logic Diagram:

- **Sum = A ⊕ B** (XOR gate)
- **Carry = A · B** (AND gate)

A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

2. Full Adder

- A **Full Adder** adds **three bits**: two input bits (A, B) and an **input carry** (Cin).
- It produces:
  - **SUM (S)**
  - **CARRY (Cout)**

Logic Diagram:

- **Sum = A ⊕ B ⊕ Cin**
- **Carry = (A · B) + (Cin · (A ⊕ B))**

A	B	Cin	SUM	CARRY
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Applications:

- Used in arithmetic and logic units (ALU)
- Binary addition circuits
- Digital counters and processors

(ii) SUBTRACTOR: - A **Subtractor** is a **combinational circuit** that performs **binary subtraction**.

It subtracts one binary number from another and provides two outputs:

- **DIFFERENCE (D)**
- **BORROW (B)**

Types of Subtractors:

1. Half Subtractor

- Subtracts two binary digits: A (minuend) and B (subtrahend).
- Outputs:
  - **Difference (D)** = D=A⊕B
  - **Borrow (B)** = A'B

A	B	DIFFERENCE	BORROW
0	0	0	0
0	1	1	1

		1 0 1 0 1 1 0 0 **Logic Diagram:** • One XOR gate and one AND-NOT (inverter + AND) gate. **2. Full Subtractor** • Subtracts **three bits**: A (minuend), B (subtrahend), and **Bin** (borrow-in). • Outputs: o **Difference (D)** o **Borrow-out (Bout)** **Logic Equations:** • **Difference = A ⊕ B ⊕ Bin** • **Borrow = (¬A · B) + (Bin · ¬(A ⊕ B))**			

A	B	Bin	DIFFERENCE	BORROW
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Applications:
- Used in arithmetic logic units (ALU) for subtraction.
- Digital calculators and microprocessors.
- Binary arithmetic and control circuits.

CI **CCI** **HoD**