REG.NO.:

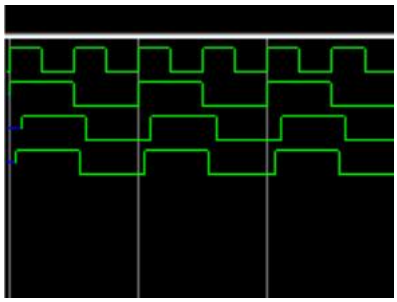
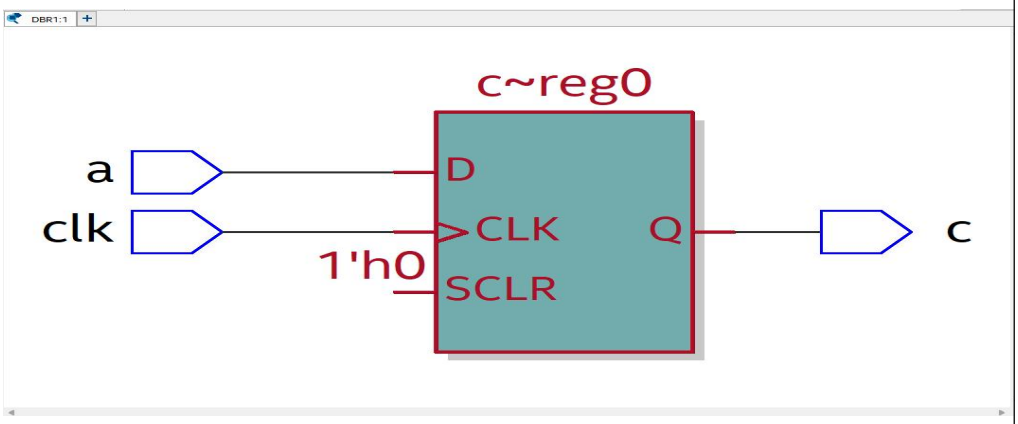
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

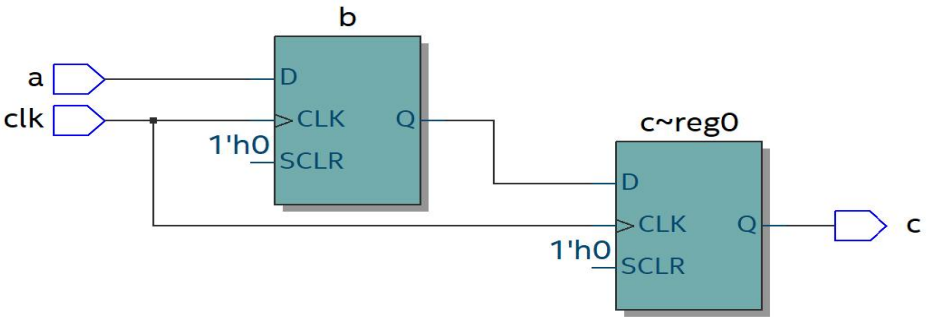
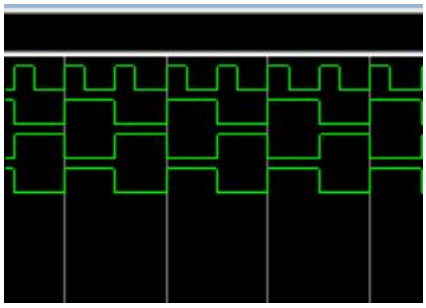
SCHOOL OF ELECTRONICS ENGINEERING
CONTINUOUS ASSESSMENT TEST - II
FALL SEMESTER 2024-2025

SLOT: D1

Programme Name & Branch : BTECH ECE
Course Code and Course Name : BECE102L Digital Systems Design
Faculty Name(s) : Nithish Kumar V, Dhanabal R, Prayline Rajabai C, Tanmaya Kumar Das and Shilpi Ruchi Kerketta
Class Number(s) : 3678, 3677, 3670, 3680, 4052
Date of Examination : 16.10.24
Exam Duration : 90 minutes **Maximum Marks: 50**

General instruction(s):

Q. No	Question	M
1.	**Differentiate the circuit's functionality with Truth table, schematic Circuit diagram and output wave form for the following codes in detail.** **a. module DBR1 (clk, a, c);** **input clk ; input a; output c;** **wire clk ; wire a; reg c; reg b;** **always @ (posedge clk)** **begin #10 b = a; #10 c = b; end** **Endmodule**   **b module DBR2 (clk ,a, c);** **input clk; input a; output c;** **wire clk; wire a; reg c; reg b;** **always @ (posedge clk)** **begin b <= a; c <= b; end**	**10**

	endmodule  ***Answer key:*** ***Circuit : 4 marks*** ***Truth table: 4 marks*** ***Output wave form : 2 marks*** 	
2.	**Design BCD adder using 4 bit ripple adders with internal circuits.** **Ans:** ***Circuit for BCD Adder : 2 marks*** ***Internal circuit of 4 bit adder using full adder with or without half adder:2 marks*** ***Internal circuit of full adder and half adder : 1 marks*** ***Truth table :*** ***BCD Adder TT : 1 marks;*** ***4 bit adder: 1 marks*** ***Full adder and Half adder:1 marks*** ***Flowchart /Algorithm : 1 marks*** ***Manual calculation and explanation or illustration of working principle with test cases: 1 marks***	**10**

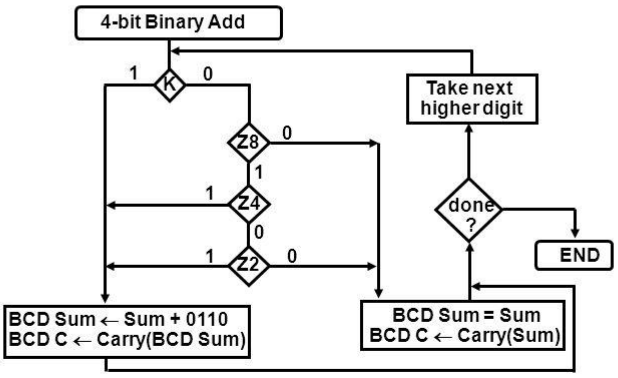
Computer Arithmetic 19 *BCD Arithmetic*

BCD ADDER

If we can convert *Binary Sums* to *BCD Sum* ,
we can use a binary adder to add two BCD numbers

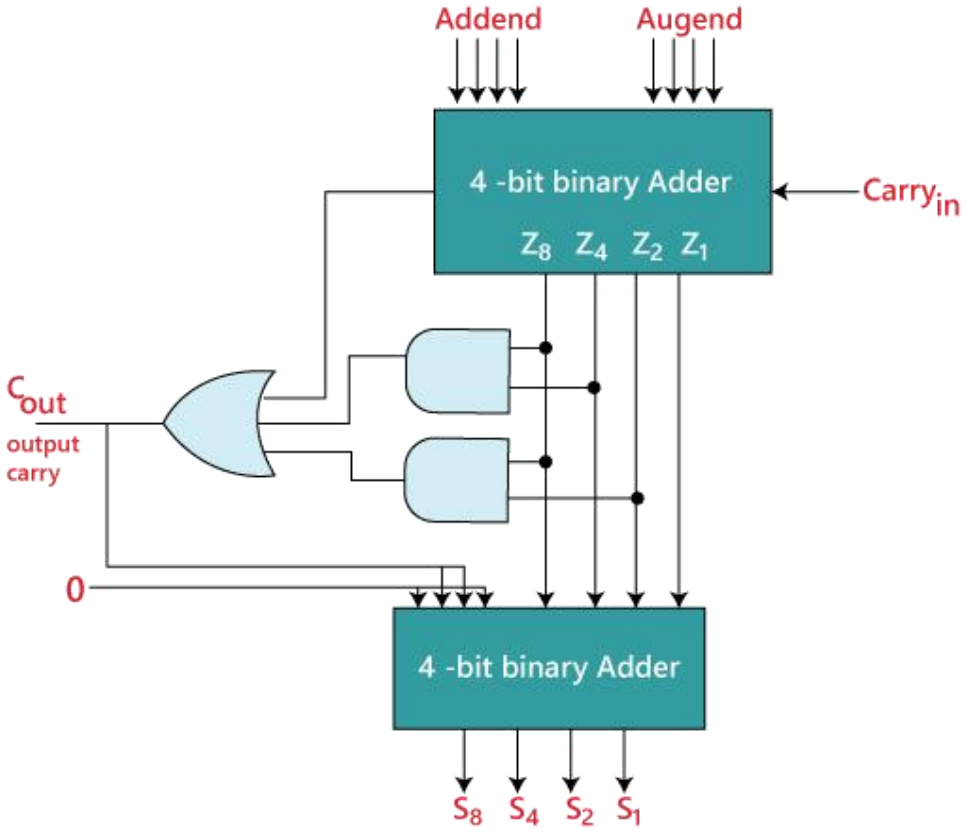
SUM =< 9
BCD Sum = Binary Sum
BCD Carry = Binary Carry

19 >= SUM > 9
BCD Sum = Binary Sum + 0110
BCD Carry = Carry(Binary Sum + 0110)



Computer Organization *Prof. H. Yoon*

Truth table



Black diagram of a BCD adder

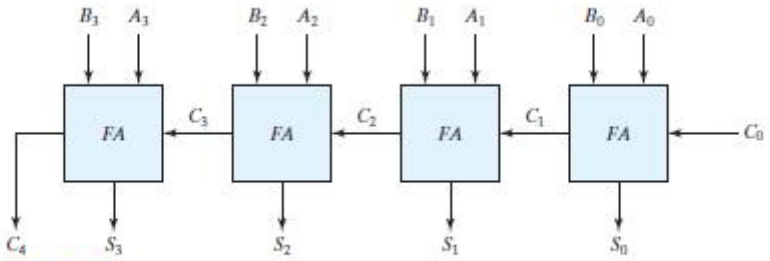


FIGURE 4.9
Four-bit adder

Truth table

Binary Sum						BCD Sum						Decimal
K	Z_8	Z_4	Z_2	Z_1		C	S_8	S_4	S_2	S_1		
0	0	0	0	0	SAME CODE	0	0	0	0	0		0
0	0	0	0	1		0	0	0	0	1		1
0	0	0	1	0		0	0	0	1	0		2
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
.	.	.	.	.		.	.	.	.	.		.
0	1	0	0	0		0	1	0	0	0		8
0	1	0	0	1		0	1	0	0	1		9
10 to 19 Binary and BCD codes are not the same												
0	1	0	1	0		1	0	0	0	0		10
0	1	0	1	1		1	0	0	0	1		11
0	1	1	0	0		1	0	0	1	0		12
0	1	1	0	1		1	0	0	1	1		13
0	1	1	1	0		1	0	1	0	0		14
0	1	1	1	1		1	0	1	0	1		15
1	0	0	0	0		1	0	1	1	0		16
1	0	0	0	1		1	0	1	1	1		17
1	0	0	1	0		1	1	0	0	0		18
1	0	0	1	1		1	1	0	0	1		19

SUM EQUALS 9 OR LESS WITH CARRY 0

Let us consider additions of 3 and 6 in BCD.

	6	0	1	1	0	← BCD for 6
+	3	0	0	1	1	← BCD for 3
	9	1	0	0	1	← BCD for 9

The addition is carried out as in normal binary addition and the sum is 1 0 0 1, which is BCD code for 9.

SUM GREATER THAN 9 WITH CARRY 0

Let us consider addition of 6 and 8 in BCD

```
    6        0  1  1  0      ← BCD for 6
+   8        1  0  0  0      ← BCD for 8
-----    ---------------
   14        1  1  1  0      ← Invalid BCD number
```

The sum 1 1 1 0 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number, as shown below

```
    6        0  1  1  0      ← BCD for 6
+   8        1  0  0  0      ← BCD for 8
-----    ---------------
   14        1  1  1  0      ← Invalid BCD number
          +  0  1  1  0      ← Add 6 for correction
          ---------------
0  0  0  1   0  1  0  0
                             ← BCD for 14
   1         4
```

After addition of 6 carry is produced into the second decimal position.

SUM EQUALS 9 OR LESS WITH CARRY 1

Let us consider addition of 8 and 9 in BCD

```
    8                    1  0  0  0   ← BCD for 8
+   9                    1  0  0  1   ← BCD for 9
-----    ---------------------------
   17    0  0  0  1      0  0  0  1   ← Incorrect BCD result
```

In this, case, result (0001 0001) is valid BCD number, but it is incorrect. To get the correct BCD result correction factor of 6 has to be added to the least significant digit sum, as shown below

```
    8                    1  0  0  0   ← BCD for 8
+   9                    1  0  0  1   ← BCD for 9
-----    ---------------------------
   17    0  0  0  1      0  0  0  1   ← Incorrect BCD result
       +  0  0  0  0      0  1  1  0   ← Add 6 for correction
         ---------------------------
         0  0  0  1      0  1  1  1   ← BCD for 17
```

Going through these three cases of BCD addition we can summarise the BCD addition procedure as follows :

1. ADD TWO BCD NUMBERS USING ORDINAY BINARY ADDITION.
2. IF FOUR-BIT SUM IS EQUAL TO OR LESS THAN 9, NO CORRECTION IS NEEDED. THE SUM IS IN PROPER BCD FORM.
3. IF THE FOUR-BIT SUM IS GREATER THAN 9 OR IF A CARRY IS GENERATED FROM THE FOUR-BIT SUM, THE SUM IS INVALID.
4. TO CORRECT THE INVALID SUM, ADD 0110_2 TO THE FOUR-BIT SUM. IF A CARRY RESULTS FROM THIS ADDITION, ADD IT TO THE NEXT HIGHER-ORDER BCD DIGIT.

Thus to implement BCD Adder Circuit we require :

- 4-BIT BINARY ADDER FOR INITIAL ADDITION
- LOGIC CIRCUIT TO DETECT SUM GREATER THAN 9 AND
- ONE MORE 4-BIT ADDER TO ADD 0110_2 IN THE SUM IF SUM IS GREATER THAN 9 OR CARRY IS 1.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given BCD Adder Truth Table.

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 3.10

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

Fig. 3.31

$$Y = S_3S_2 + S_3S_1$$

With this design information we can draw the BCD Adder Block Diagram, as shown in the Fig. 3.32.

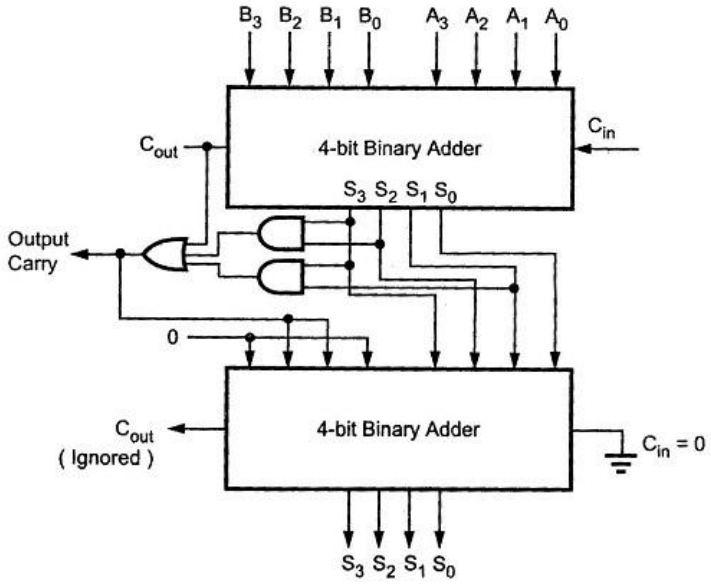
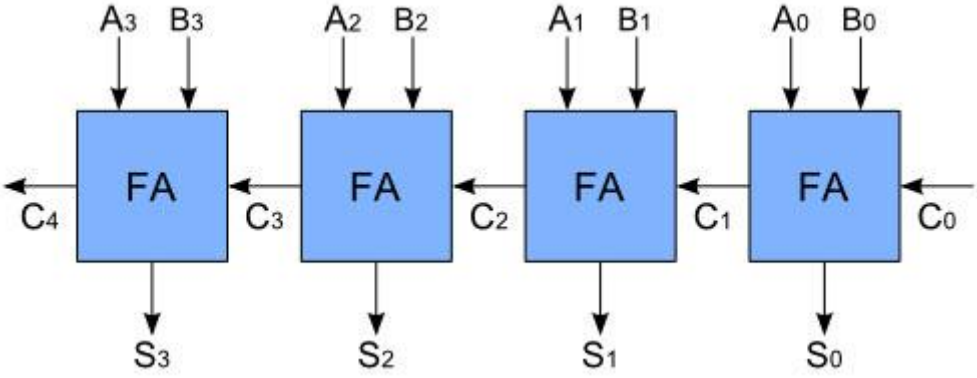
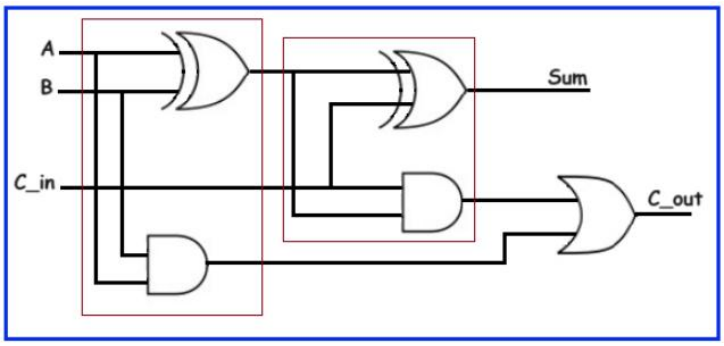


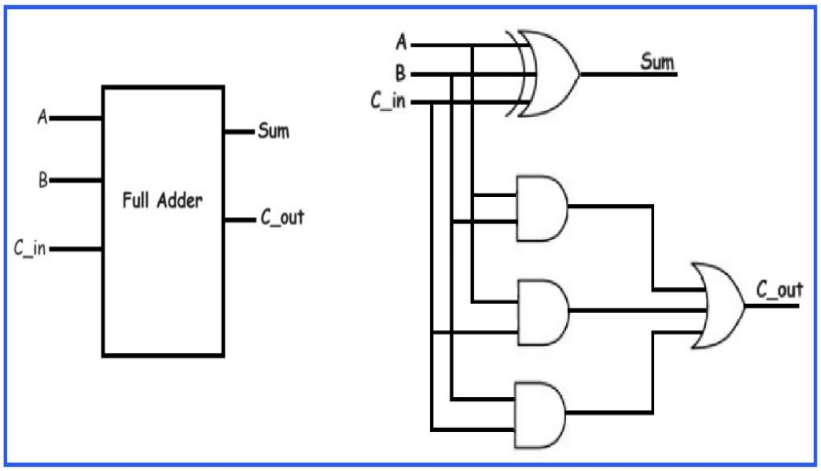
Fig. 3.32 Block diagram of BCD adder

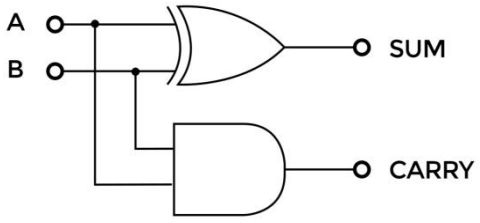
As shown in the Fig. 3.32 , the two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum. When the output carry is equal to zero (i.e. when sum ≤ 9 and C_{0UT} = 0) nothing (zero) is added to the binary sum. When it is equal to one (i.e. when sum > 9 or C_{0UT} = 1), **BINARY** 0110 is added to the binary sum through the bottom 4-bit binary adder. The output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.

4 Bit adder internal circuit diagram









A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

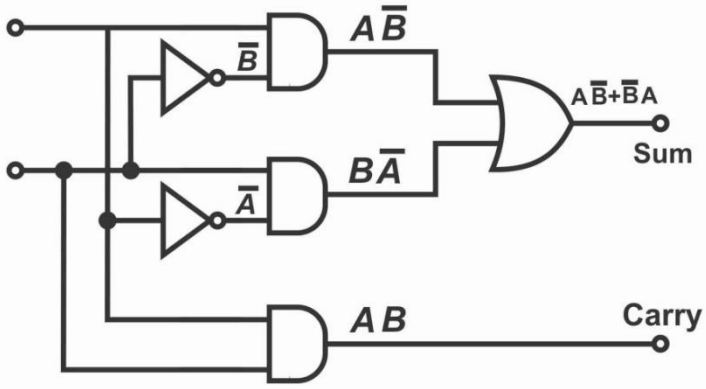


Fig 3.5 Logic gate half adder

3. **Design a 4–Bit ALU and write a verilog code for the given function table.**

S2	S1	S0	Function
0	0	0	A * B
0	0	1	A / B
0	1	0	A + B
0	1	1	A – B
1	0	0	A * * B
1	0	1	A + 1
1	1	0	A – 1
1	1	1	A >> 1

Verilog Code in BL/DFL/SL : 5 marks
Explanation /illustration of working : 2.5 marks
Design /circuit diagram/Block diagram: 2.5 marks

Sample Verilog code:

10

```
module ALU_4bit (
   input [3:0] A, B,  // 4-bit inputs
   input [2:0] S,     // 3-bit control signal
   output reg [3:0] Y // 4-bit output
);

always @(*) begin
   case (S)
      3'b000: Y = A * B;      // A * B
      3'b001: Y = A / B;      // A / B
      3'b010: Y = A + B;      // A + B
      3'b011: Y = A - B;      // A - B
      3'b100: Y = A ** B;     // A ** B (Exponentiation)
      3'b101: Y = A + 1;      // A + 1
      3'b110: Y = A - 1;      // A - 1
```

```
      3'b111: Y = A >> 1;     // A >> 1 (Logical right shift)
      default: Y = 4'b0000;   // Default case
   endcase
end

endmodule
```

Explanation
Inputs and Outputs:
A and B are the 4-bit input operands.
S is the 3-bit control signal that selects the operation.
Y is the 4-bit output result.
Operations:
3'b000: Multiplication (A * B)
3'b001: Division (A / B)
3'b010: Addition (A + B)
3'b011: Subtraction (A - B)
3'b100: Exponentiation (A ** B)
3'b101: Increment (A + 1)
3'b110: Decrement (A - 1)
3'b111: Logical right shift (A >> 1)
This code defines a simple 4-bit ALU that performs the specified operations based on the control signals.

Or

```
module ALU (
   input [3:0] A,   // 4-bit input A
   input [3:0] B,   // 4-bit input B
   input [2:0] S,   // 3-bit select input for function
   output reg [3:0] Y, // 4-bit output
   output reg Overflow // Overflow flag
);

   always @(*) begin
      case (S)
         3'b000: begin // A * B
            {Overflow, Y} = A * B;
         end

         3'b001: begin // A / B
            if (B != 0) begin
               Y = A / B;
               Overflow = 0; // No overflow for division
            end else begin
               Y = 4'b0000; // Division by zero case
               Overflow = 1; // Set overflow flag
            end
         end
```

```
        3'b010: begin // A + B
          {Overflow, Y} = A + B;
        end

        3'b011: begin // A - B
          {Overflow, Y} = A - B;
        end

        3'b100: begin // A ** B (A raised to the power of B)
          Y = (B == 4'b0000) ? 4'b0001 : 4'b0000; // Handle A^0 = 1, A^1 = A,
others will be zero
          Overflow = 0; // No overflow
        end

        3'b101: begin // A + 1
          {Overflow, Y} = A + 1;
        end

        3'b110: begin // A - 1
          {Overflow, Y} = A - 1;
        end

        3'b111: begin // A >> 1 (right shift)
          Y = A >> 1;
          Overflow = 0; // No overflow for shift
        end

        default: begin
          Y = 4'b0000; // Default case
          Overflow = 0;
        end
      endcase
    end
endmodule
```

4. a) **Convert D Flip flop to AB Flip flop.** — **5**

A	B	Q(t+1)
0	0	Set
0	1	Q’
1	0	Reset
1	1	Q

Ans key:

Explanation and Conversion procedure:2 marks
Circuit diagram: 2 marks
Truth table/ k map simplification: 1 marks

b) **Write Structural level Verilog code for above design with Test Bench.** — **5**

Verilog code with comments : 3 marks
Test bench : 2 marks

	Expression for D
	
	The combined logic can be represented as:
	
	D=(A’B’)+(A’B)Q’+(AB’)0+(AB)QD = (A'B') + (A'B)Q' + (AB')0 + (AB)QD=(A’B’)+(A’B)Q’+(AB’)0+(AB)Q
	
	This simplifies to:
	
	D=A’B’+A’BQ’+ABQD = A'B' + A'BQ' + ABQD=A’B’+A’BQ’+ABQ
	
	Implementation
	
	1. **Use a 2-input AND gate** to determine when A’B’A'B'A’B’ is true.
	2. **Use a 2-input AND gate** to determine when A’BA'BA’B is true and connect it to a NOT gate to find Q’Q'Q’.
	3. **Use another 2-input AND gate** to determine when ABABAB is true, allowing QQQ to pass through.
	4. **Combine the outputs** of the three AND gates using an OR gate to generate DDD.
	
	module AB_FlipFlop (input wire A, input wire B, input wire clk,
	output reg Q);
	wire D; // D input for the D Flip-Flop
	wire Q_not; // Q' (complement of Q)
	not(Q_not, Q); // Invert Q to get Q'
	assign D = (A == 0 && B == 0) ? 1'b1 : // Set
	(A == 0 && B == 1) ? Q_not : // Complement
	(A == 1 && B == 0) ? 1'b0 : // Reset
	Q; // Hold // Combinational logic to determine D
	always @(posedge clk) begin Q <= D; end // D Flip-Flop
	endmodule	

5.	**Design a counter for the below sequence and write the behavioral verilog code with test Bench.** **If PQn=NQn (0-0 or 1-1) then TQn is 0** **If PQn≠NQn (0-1 or 1-0) then TQn is 1**	**10**

	Present state (PQn)				Next state (NQn)				TQn			
State	Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0	T3	T2	T1	T0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	1	1	0	0	1	0
2	0	0	1	1	0	0	1	0	0	0	0	1
3	0	0	1	0	0	1	1	0	0	1	0	0
4	0	1	1	0	0	1	1	1	0	0	0	1
5	0	1	1	1	0	1	0	1	0	0	1	0
6	0	1	0	1	0	1	0	0	0	0	0	1
7	0	1	0	0	1	1	0	0	1	0	0	0
8	1	1	0	0	1	1	0	1	0	0	0	1
9	1	1	0	1	1	1	1	1	0	0	1	0
10	1	1	1	1	1	1	1	0	0	0	0	1
11	1	1	1	0	1	0	1	0	0	1	0	0
12	1	0	1	0	1	0	1	1	0	0	0	1
13	1	0	1	1	1	0	0	1	0	0	1	0
14	1	0	0	1	1	0	0	0	0	0	0	1
15	1	0	0	0	0	0	0	0	1	0	0	0

Design using T/JK/D flip-flop (using K map) and circuit diagram: 5 marks

Code with comments : 3 Marks Test bench :2 marks

Code:

```
module counter (    input clk,    input reset,    output reg [3:0] Q );
  always @(posedge clk or posedge reset) begin
    if (reset)          Q <= 4'b0000;
```

```
    else begin          case (Q)
        4'b0000: Q <= 4'b0001;
        4'b0001: Q <= 4'b0011;
        4'b0011: Q <= 4'b0010;
        4'b0010: Q <= 4'b0110;
        4'b0110: Q <= 4'b0111;
        4'b0111: Q <= 4'b0101;
        4'b0101: Q <= 4'b0100;
        4'b0100: Q <= 4'b1100;
        4'b1100: Q <= 4'b1101;
        4'b1101: Q <= 4'b1111;
        4'b1111: Q <= 4'b1110;
        4'b1110: Q <= 4'b1010;
        4'b1010: Q <= 4'b1011;
        4'b1011: Q <= 4'b1001;
        4'b1001: Q <= 4'b1000;
        4'b1000: Q <= 4'b0000;
        default: Q <= 4'b0000;
      endcase
    end
  end
endmodule
```

Test Bench

```
module tb_counter;
  reg clk;   reg reset;   wire [3:0] Q;   // Instantiate the counter
```

```
counter uut (      .clk(clk),      .reset(reset),      .Q(Q)     );
initial begin // Clock generation
   clk = 0;        forever #5 clk = ~clk; // 10ns period
end    // Test sequence
initial begin       // Initialize reset
   reset = 1;       #10 ;       reset = 0;       // Let the counter run for a while
   #160;         // Apply reset again
   reset = 1;       #10;       reset = 0;       // Let the counter run for a while
   #160;      $finish;   // Finish simulation
   end
initial begin       $monitor("Time = %0t, Q = %b", $time, Q); end
endmodule    // Monitor the output
```

ref: https://www.youtube.com/watch?v=plhzOM8gcSA

****