Name：- Meet Soni

ID No. :- 21EL035

Division :- 04

Year :- 2023-24

Subject:- Digital System Design (3EL42)

Branch :- Electronics (EL)

Assignment :- 1

Q-1 :-Write a Verilog code for 2*4 decoder.

Dataflow:-

```
module decoder_2x4_dataflow (
    input wire enable,
    input wire [1:0] select,
    output wire [3:0] output
);

    assign output[0] = (enable && (select == 2'b00)) ? 1'b1 : 1'b0;
    assign output[1] = (enable && (select == 2'b01)) ? 1'b1 : 1'b0;
    assign output[2] = (enable && (select == 2'b10)) ? 1'b1 : 1'b0;
    assign output[3] = (enable && (select == 2'b11)) ? 1'b1 : 1'b0;

endmodule
```

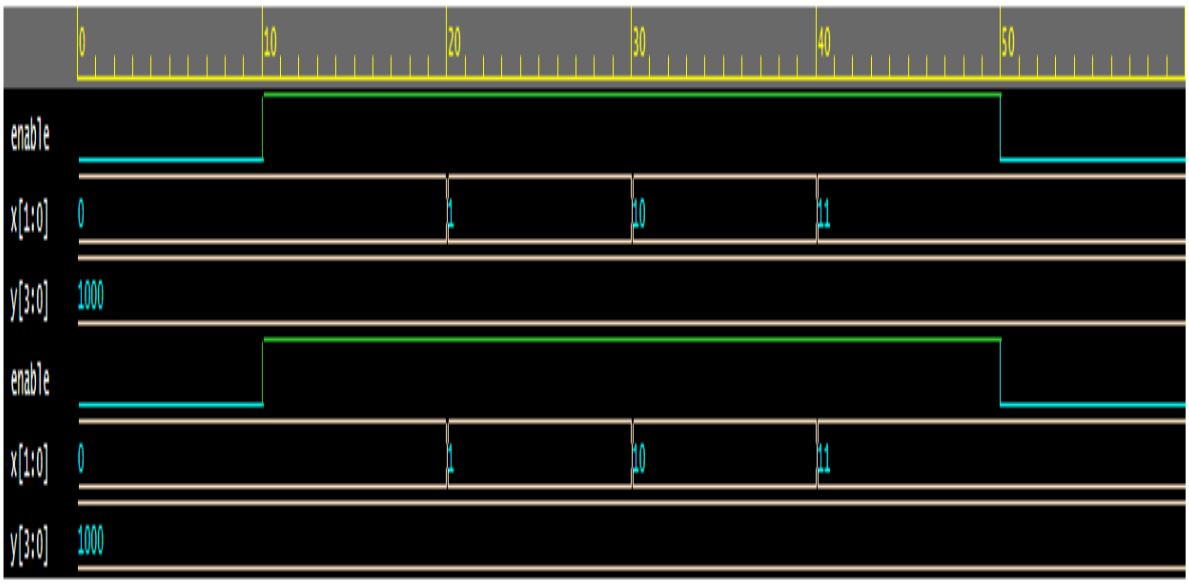
Behavioral:-

```
module decoder_2x4_behavioral (
    input wire enable,
    input wire [1:0] select,
    output wire [3:0] output
);

    always @(enable or select) begin
        case (select)
            2'b00: output = enable ? 4'b0001 : 4'b0000;
            2'b01: output = enable ? 4'b0010 : 4'b0000;
            2'b10: output = enable ? 4'b0100 : 4'b0000;
            2'b11: output = enable ? 4'b1000 : 4'b0000;
            default: output = 4'b0000;
        endcase
    end

endmodule
```

Structural :-

```
module AND2 (
    input wire a,
    input wire b,
    output wire y
);
    assign y = a & b;
endmodule

module NOT (
    input wire a,
    output wire y
);
    assign y = ~a;
endmodule

module decoder_2x4_structural (
    input wire enable,
    input wire [1:0] select,
    output wire [3:0] output
);

    wire w_not_sel0, w_not_sel1;

    NOT not_sel0 (.a(select[0]), .y(w_not_sel0));
    NOT not_sel1 (.a(select[1]), .y(w_not_sel1));

    AND2 and00 (.a(enable), .b(w_not_sel1), .y(output[0]));
    AND2 and01 (.a(enable), .b(w_not_sel0), .y(output[1]));
    AND2 and10 (.a(enable), .b(select[1]), .y(output[2]));
    AND2 and11 (.a(enable), .b(select[0]), .y(output[3]));

endmodule

```

Output:-


0
10
20
30
40
50
enable
x[1:0]
0
1
10
11
y[3:0]
1000
enable
x[1:0]
0
1
10
11
y[3:0]
1000

Q-2 :- Write a Verilog code for full subtractor.

Dataflow:-

```
module full_subtractor_dataflow (
    input wire A, B, Bin,
    output wire Diff, Bout, Borr
);
    assign {Borr, Diff, Bout} = A ^ B ^ Bin;
endmodule
```

Behavioural :-

```
module full_subtractor_behavioral (
    input wire A, B, Bin,
    output wire Diff, Bout, Borr
);
    assign Diff = (A ^ B) ^ Bin;
    assign Bout = (~A & B) | (Bin & (~A ^ B));
    assign Borr = (~A & B) | (Bin & ~B);
endmodule
```

Structural :

```
module XOR2 (
    input wire a,
    input wire b,
    output wire y
);
    assign y = a ^ b;
endmodule

module AND2 (
    input wire a,
    input wire b,
    output wire y
);
    assign y = a & b;
endmodule

module NOT (
    input wire a,
    output wire y
);
    assign y = ~a;
endmodule

module full_subtractor_structural (
    input wire A, B, Bin,
    output wire Diff, Bout, Borr
);
    wire x1, x2, x3, x4, x5, x6;
    XOR2 xor1 (.a(A), .b(B), .y(x1));
    XOR2 xor2 (.a(x1), .b(Bin), .y(Diff));
    AND2 and1 (.a(~A), .b(B), .y(x2));
    AND2 and2 (.a(x2), .b(Bin), .y(x3));
    AND2 and3 (.a(~A), .b(Bin), .y(x4));
    AND2 and4 (.a(B), .b(~Bin), .y(x5));
    OR2 or1 (.a(x3), .b(x5), .y(Bout));
    OR2 or2 (.a(x4), .b(x5), .y(Borr));
endmodule
```

Output:-

```
[2023-08-23 16:18:53 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
At time 0: a=0 b=0, Bin=0, difference=0, borrow=0
At time 1: a=0 b=0, Bin=1, difference=1, borrow=1
At time 2: a=0 b=1, Bin=0, difference=1, borrow=1
At time 3: a=0 b=1, Bin=1, difference=0, borrow=1
At time 4: a=1 b=0, Bin=0, difference=1, borrow=0
At time 5: a=1 b=0, Bin=1, difference=0, borrow=0
At time 6: a=1 b=1, Bin=0, difference=0, borrow=0
At time 7: a=1 b=1, Bin=1, difference=1, borrow=1
```

Q-3 :- Write a Verilog code for 2-bit comparator.

Dataflow:-

```
module twobitcomparator(xgtyin,xltyin,x,y,xgty,xlty,xety);
    //I/O
  output xgty, xety, xlty;
  input  xgtyin, xltyin;
  input [1:0] x,y;

  assign xgty = xgtyin | (~xltyin & ((x[1] > y[1]) | ((x[1] == y[1]) &
(x[0] > y[0]))));
  assign xlty = xltyin | (~xgtyin & ((x[1] < y[1]) | ((x[1] == y[1]) &
(x[0] < y[0]))));

  assign xety = ~(xlty | xgty);

endmodule
```

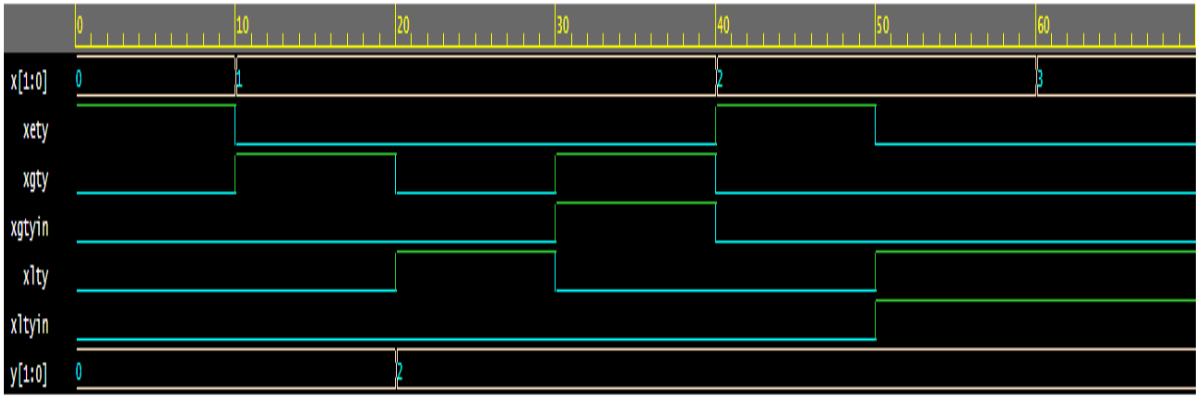
Behavioural :-

```
module comparator_2bit_behavioral (
    input wire [1:0] A,
    input wire [1:0] B,
    output wire equal,
    output wire A_greater,
    output wire B_greater
);
    assign equal = (A[1:0] == B[1:0]);
    assign A_greater = (A[1] && !B[1]) || (A[1] == B[1] && A[0] > B[0]);
    assign B_greater = (B[1] && !A[1]) || (B[1] == A[1] && B[0] > A[0]);
endmodule
```

Structural :-

```
module AND2 (
    input wire a,
    input wire b,
    output wire y
);
    assign y = a & b;
endmodule

module OR2 (
    input wire a,
    input wire b,
    output wire y
);
    assign y = a | b;
endmodule

module NOT (
    input wire a,
    output wire y
);
    assign y = ~a;
endmodule

module comparator_2bit_structural (
    input wire [1:0] A,
    input wire [1:0] B,
    output wire equal,
    output wire A_greater,
    output wire B_greater
);
    wire a0_b0, a1_b1, a0_b0_not, a1_b1_not, a0_eq_b0, a1_eq_b1;

    XOR2 xor0 (.a(A[0]), .b(B[0]), .y(a0_b0));
    XOR2 xor1 (.a(A[1]), .b(B[1]), .y(a1_b1));
    NOT not0 (.a(a0_b0), .y(a0_b0_not));
    NOT not1 (.a(a1_b1), .y(a1_b1_not));
    AND2 and0 (.a(a0_b0_not), .b(a1_b1_not), .y(equal));
    AND2 and1 (.a(a0_b0_not), .b(A[1]), .y(a0_eq_b0));
    AND2 and2 (.a(a1_b1_not), .b(A[0]), .y(a1_eq_b1));
    OR2 or0 (.a(a0_eq_b0), .b(a1_eq_b1), .y(A_greater));
    OR2 or1 (.a(a0_b0), .b(a1_b1_not), .y(B_greater));
endmodule
```

Output:-



Q-4:- Write a Verilog code for 3 bit binary to gray convertor.

Dataflow:-

```
module bin_to_gray_converter(
    input wire [2:0] binary,
    output wire [2:0] gray
);
    assign gray[2] = binary[2];
    assign gray[1] = binary[2] ^ binary[1];
    assign gray[0] = binary[1] ^ binary[0];
endmodule

module comparator_bin_to_gray_dataflow (
    input wire [2:0] A,
    input wire [2:0] B,
    output wire equal,
    output wire A_greater,
    output wire B_greater
);

    wire [2:0] gray_A, gray_B;

    bin_to_gray_converter bin_to_gray_A (.binary(A), .gray(gray_A));
    bin_to_gray_converter bin_to_gray_B (.binary(B), .gray(gray_B));

    assign equal = (gray_A == gray_B);
    assign A_greater = (gray_A > gray_B);
    assign B_greater = (gray_A < gray_B);
endmodule
```

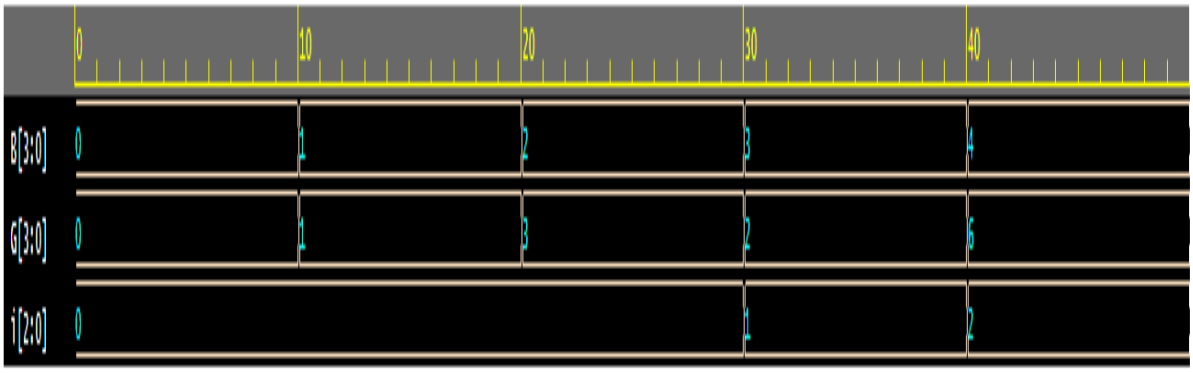
Behavioural :-

```
module comparator_bin_to_gray_behavioral (
    input wire [2:0] A,
    input wire [2:0] B,
    output wire equal,
    output wire A_greater,
    output wire B_greater
);

    wire [2:0] gray_A, gray_B;

    assign gray_A[2] = A[2];
    assign gray_A[1] = A[2] ^ A[1];
    assign gray_A[0] = A[1] ^ A[0];

    assign gray_B[2] = B[2];
    assign gray_B[1] = B[2] ^ B[1];
    assign gray_B[0] = B[1] ^ B[0];

    assign equal = (gray_A == gray_B);
    assign A_greater = (gray_A > gray_B);
    assign B_greater = (gray_A < gray_B);
endmodule
```

Structural :-

```
module XOR2 (
    input wire a,
    input wire b,
    output wire y
);
    assign y = a ^ b;
endmodule

module bin_to_gray_converter(
    input wire [2:0] binary,
    output wire [2:0] gray
);
    wire g1, g2;

    assign gray[2] = binary[2];
    XOR2 xor1 (.a(binary[2]), .b(binary[1]), .y(g1));
    XOR2 xor2 (.a(binary[1]), .b(binary[0]), .y(g2));
    assign gray[1] = g1;
    assign gray[0] = g2;
endmodule

module comparator_bin_to_gray_structural (
    input wire [2:0] A,
    input wire [2:0] B,
    output wire equal,
    output wire A_greater,
    output wire B_greater
);

    wire [2:0] gray_A, gray_B;

    bin_to_gray_converter bin_to_gray_A (.binary(A), .gray(gray_A));
    bin_to_gray_converter bin_to_gray_B (.binary(B), .gray(gray_B));

    wire [2:0] xor_outputs_A, xor_outputs_B;

    XOR2 xor0_A (.a(gray_A[2]), .b(gray_B[2]), .y(xor_outputs_A[2]));
    XOR2 xor1_A (.a(gray_A[1]), .b(gray_B[1]), .y(xor_outputs_A[1]));
    XOR2 xor2_A (.a(gray_A[0]), .b(gray_B[0]), .y(xor_outputs_A[0]));

    XOR2 xor0_B (.a(gray_B[2]), .b(gray_A[2]), .y(xor_outputs_B[2]));
    XOR2 xor1_B (.a(gray_B[1]), .b(gray_A[1]), .y(xor_outputs_B[1]));
    XOR2 xor2_B (.a(gray_B[0]), .b(gray_A[0]), .y(xor_outputs_B[0]));

    assign equal = &(xor_outputs_A);
    assign A_greater = (xor_outputs_A[2] & ~xor_outputs_A[1]) | (xor_outputs_A[1] & ~xor_outputs_A[0]);
    assign B_greater = (xor_outputs_B[2] & ~xor_outputs_B[1]) | (xor_outputs_B[1] & ~xor_outputs_B[0]);
endmodule

```

Output:-



Q-5 :- Write a Verilog code for BCD to excess 3 convertors.

Dataflow :-

```
module bcd_to_xs3_dataflow (
    input wire [3:0] bcd,
    output wire [3:0] xs3
);
    assign xs3[3] = 1'b0;
    assign xs3[2] = bcd[3] ^ 1'b1;
    assign xs3[1] = bcd[2] ^ 1'b1;
    assign xs3[0] = bcd[1] ^ 1'b1;
endmodule
```

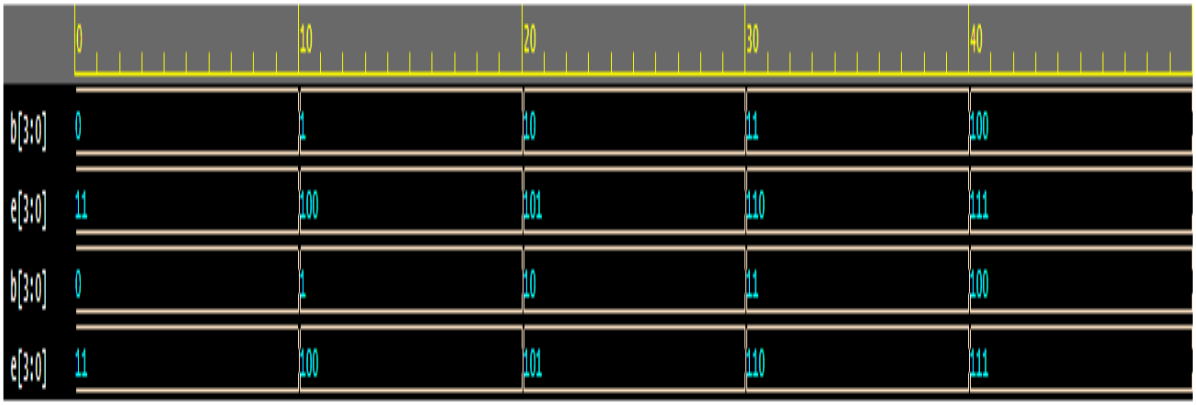
Behavioural :-

```
module bcd_to_xs3_behavioral (
    input wire [3:0] bcd,
    output wire [3:0] xs3
);
    assign xs3[3] = 1'b0;
    assign xs3[2] = ~bcd[3];
    assign xs3[1] = ~bcd[2];
    assign xs3[0] = ~bcd[1];
endmodule
```

Structural:-

```
module NOT (
    input wire a,
    output wire y
);
    assign y = ~a;
endmodule

module bcd_to_xs3_structural (
    input wire [3:0] bcd,
    output wire [3:0] xs3
);
    wire not_bcd3, not_bcd2, not_bcd1;

    NOT not_b3 (.a(bcd[3]), .y(not_bcd3));
    NOT not_b2 (.a(bcd[2]), .y(not_bcd2));
    NOT not_b1 (.a(bcd[1]), .y(not_bcd1));

    assign xs3[3] = 1'b0;
    assign xs3[2] = not_bcd3;
    assign xs3[1] = not_bcd2;
    assign xs3[0] = not_bcd1;
endmodule
```

Output:-


0
10
20
30
40
b[3:0]
0
1
10
11
100
e[3:0]
11
100
101
110
111
b[3:0]
0
1
10
11
100
e[3:0]
11
100
101
110
111