DSD LAB

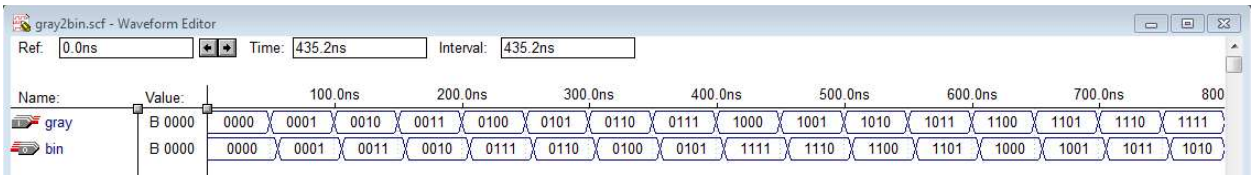
Lab 3

Q1)

Source Code:

```
module gray2bin(gray, bin);
        parameter N = 4;
        input [N-1: 0] gray;
        output [N-1: 0] bin;
        reg [N-1: 0] bin;
        integer i;
        always @(gray)
        begin
                bin[N-1] = gray[N-1];
                for(i = N-2; i >= 0; i = i-1)
                        bin[i] = bin[i+1] ^ gray[i];
        end
endmodule
```

Output Waveform:



Q2)

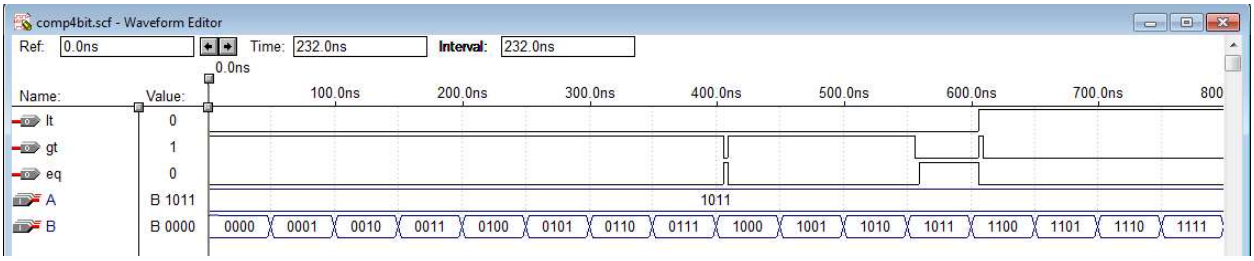
Source Code:

```
module comp4bit(A, B, lt, gt, eq);
        input [3:0] A, B;
        output lt, gt, eq;
        wire lt1, lt2, gt1, gt2, eq1, eq2;
        comp2bit s1(A[0], A[1], B[0], B[1], lt1, gt1, eq1);
        comp2bit s2(A[2], A[3], B[2], B[3], lt2, gt2, eq2);
        assign eq = eq1 & eq2;
        assign gt = gt2 | (gt1 & eq2) ;
        assign lt = lt2 | (lt1 & eq2);
endmodule

module comp2bit(a0, a1, b0, b1, lt, gt, eq);
        input a0, a1, b0, b1;
```

```
        output lt, gt, eq;
        wire i1, i0;
        assign i1 = ~(a1 ^ b1);
        assign i0 = ~(a0 ^ b0);
        assign eq = i1 & i0;
        assign gt = (a1 & ~b1) | (i1 & a0 & ~b0);
        assign lt = ~(gt | eq);
endmodule
```

Output Waveform:



Q3)

Source Code:

```
module mux16to1(W, S, out);
        input [15:0] W;
        input [3:0] S;
        wire [1:0]op;
        output out;
        mux8to1 m1(W[7:0], S[2:0], op[0]);
        mux8to1 m2(W[15:8], S[2:0], op[1]);
        mux2to1 m3(op, S[3], out);
endmodule

module mux8to1(W, S, out);
        input [7:0]W;
        input [2:0]S;
        wire [7:0]W;
        wire [2:0]S;
        output out;
        reg out;

        always @(W or S)
        begin
                case(S)
                0: out = W[0];
                1: out = W[1];
```

```
                2: out = W[2];
                3: out = W[3];
                4: out = W[4];
                5: out = W[5];
                6: out = W[6];
                7: out = W[7];
                endcase
        end
endmodule

module mux2to1(W, s, out);
        input [1:0] W;
        wire [1:0] W;
        input s;
        output out;
        reg out;

        always @(W or s)
        begin
        if(s)
                out = W[1];
        else
                out = W[0];
        end
endmodule
```

Output Waveform: