Lab 3

Question 1

Behavioral Modelling Code

```
// Q1_VotingCounter_behavioral.v
module Q1_VotingCounter_behavioral(
    input  w, x0, x1, y0, y1, y2, z0, z1, z2, z3,
    output reg [6:0] seg
);
    integer cnt;
    always @(*) begin
        cnt = w + x0 + x1 + y0 + y1 + y2 + z0 + z1 + z2 + z3;
        case (cnt)
            0:  seg = 7'b0000000; // blank
            1:  seg = 7'b0110000;
            2:  seg = 7'b1101101;
            3:  seg = 7'b1111001;
            4:  seg = 7'b0110011;
            5:  seg = 7'b1011011;
            6:  seg = 7'b1011111;
            7:  seg = 7'b1110000;
            8:  seg = 7'b1111111;
            9:  seg = 7'b1111011;
            10: seg = 7'b1111110; // show 0
            default: seg = 7'b0000000;
        endcase
    end
endmodule
```

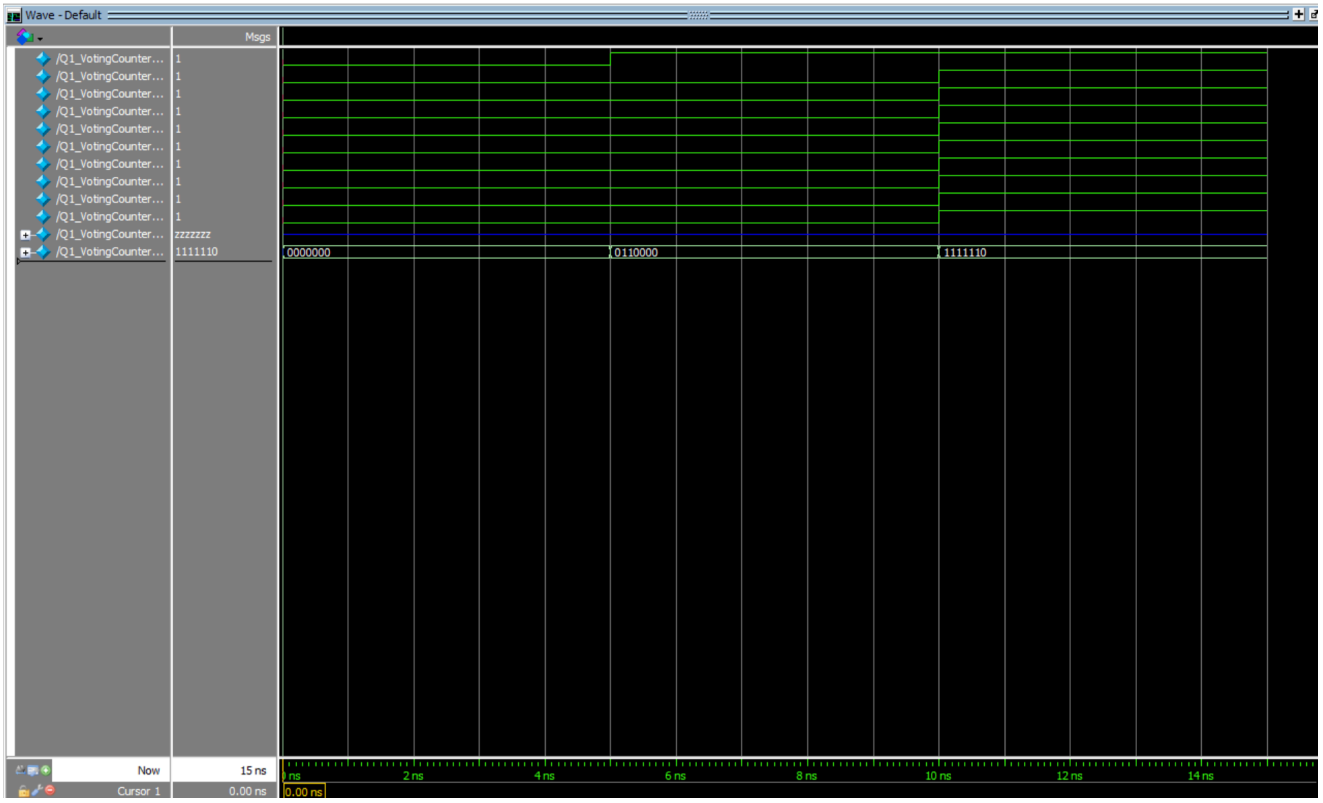
Dataflow Modelling Code

```
// Q1_VotingCounter_dataflow.v
module Q1_VotingCounter_dataflow(
    input  w, x0, x1, y0, y1, y2, z0, z1, z2, z3,
    output reg [6:0] seg   // 7-segment output: seg[6]=a ... seg[0]=g
);
    wire [3:0] sum;
    assign sum = w + x0 + x1 + y0 + y1 + y2 + z0 + z1 + z2 + z3;

    always @(*) begin
        case (sum)
            4'd0:  seg = 7'b0000000; // blank
            4'd1:  seg = 7'b0110000; // 1
            4'd2:  seg = 7'b1101101; // 2
            4'd3:  seg = 7'b1111001; // 3
            4'd4:  seg = 7'b0110011; // 4
            4'd5:  seg = 7'b1011011; // 5
            4'd6:  seg = 7'b1011111; // 6
            4'd7:  seg = 7'b1110000; // 7
            4'd8:  seg = 7'b1111111; // 8
            4'd9:  seg = 7'b1111011; // 9
            4'd10: seg = 7'b1111110; // treat 10 as 0
            default: seg = 7'b0000000; // blank
        endcase
    end
endmodule
```

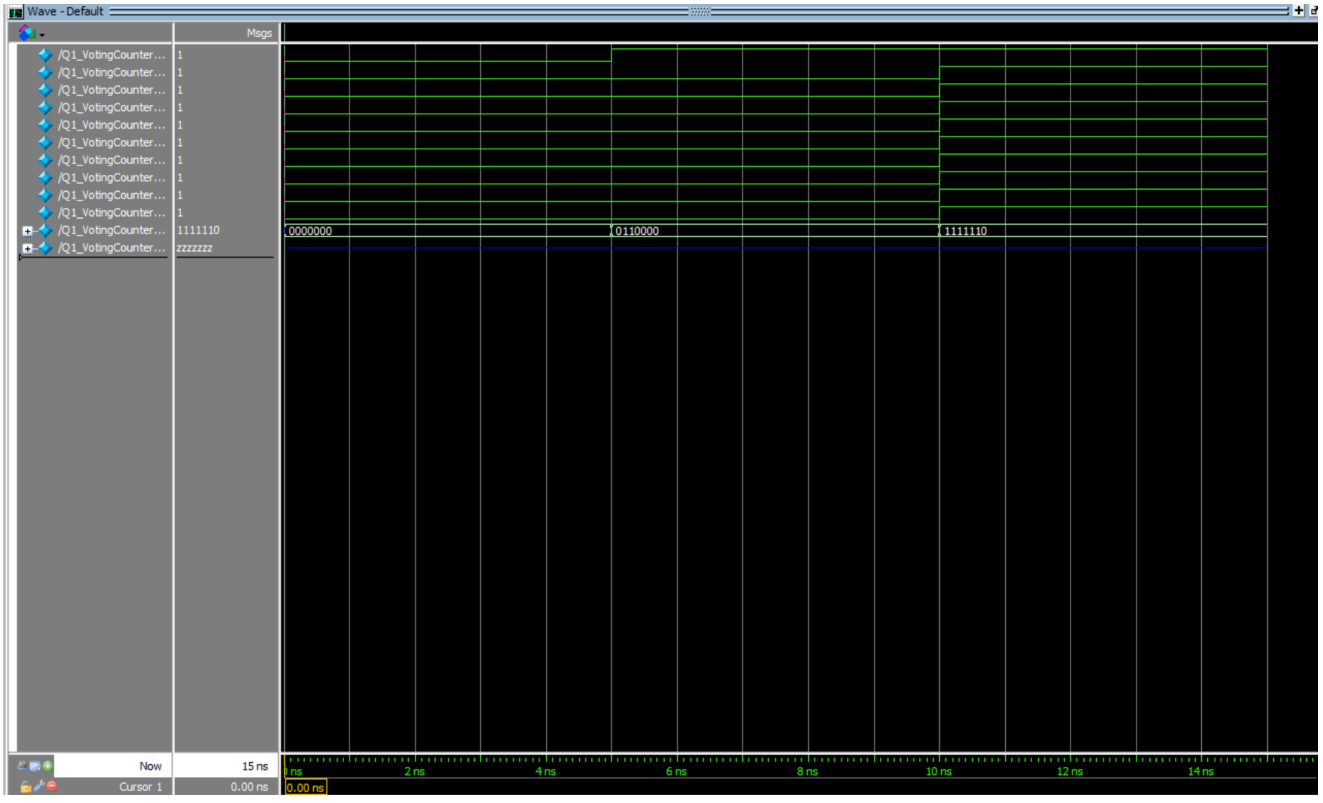
Testbench

```
// Q1_VotingCounter_tb.v
`timescale 1ns/1ps
module Q1_VotingCounter_tb;
    reg w, x0, x1, y0, y1, y2, z0, z1, z2, z3;
    wire [6:0] seg_df, seg_bh;

    Q1_VotingCounter_dataflow   dut1 (w,x0,x1,y0,y1,y2,z0,z1,z2,z3,seg_df);
    //Q1_VotingCounter_behavioral dut2
(w,x0,x1,y0,y1,y2,z0,z1,z2,z3,seg_bh);

    initial begin
        $display("w x0 x1 y0 y1 y2 z0 z1 z2 z3 | seg_df   seg_bh");
        // Test 0 votes
        {w,x0,x1,y0,y1,y2,z0,z1,z2,z3}=10'b0000000000; #5;
        $display("%b %b %b %b %b %b %b %b %b %b | %b %b",
                 w,x0,x1,y0,y1,y2,z0,z1,z2,z3,seg_df,seg_bh);

        // Test 1 vote
        {w,x0,x1,y0,y1,y2,z0,z1,z2,z3}=10'b1000000000; #5;
        $display("%b %b %b %b %b %b %b %b %b %b | %b %b",
                 w,x0,x1,y0,y1,y2,z0,z1,z2,z3,seg_df,seg_bh);

        // Test 10 votes
        {w,x0,x1,y0,y1,y2,z0,z1,z2,z3}=10'b1111111111; #5;
        $display("%b %b %b %b %b %b %b %b %b %b | %b %b",
                 w,x0,x1,y0,y1,y2,z0,z1,z2,z3,seg_df,seg_bh);

        $finish;
    end
endmodule
```

Behavioral Modelling Output



Dataflow Modelling Output



Question 2

Behavioral Modelling Code

```
// Q2_Function_behavioral.v
module Q2_Function_behavioral(
    input A, B, C, D,
    output reg F
);
    always @(*) begin
        if ((A && ~B) || (B && (C || D)))
            F = 1;
        else
            F = 0;
    end
endmodule
```

Dataflow Modelling Output

```
// Q2_Function_dataflow.v
module Q2_Function_dataflow(
    input A, B, C, D,
    output F
);
    assign F = (~B & A) | (B & (C | D));
endmodule
```

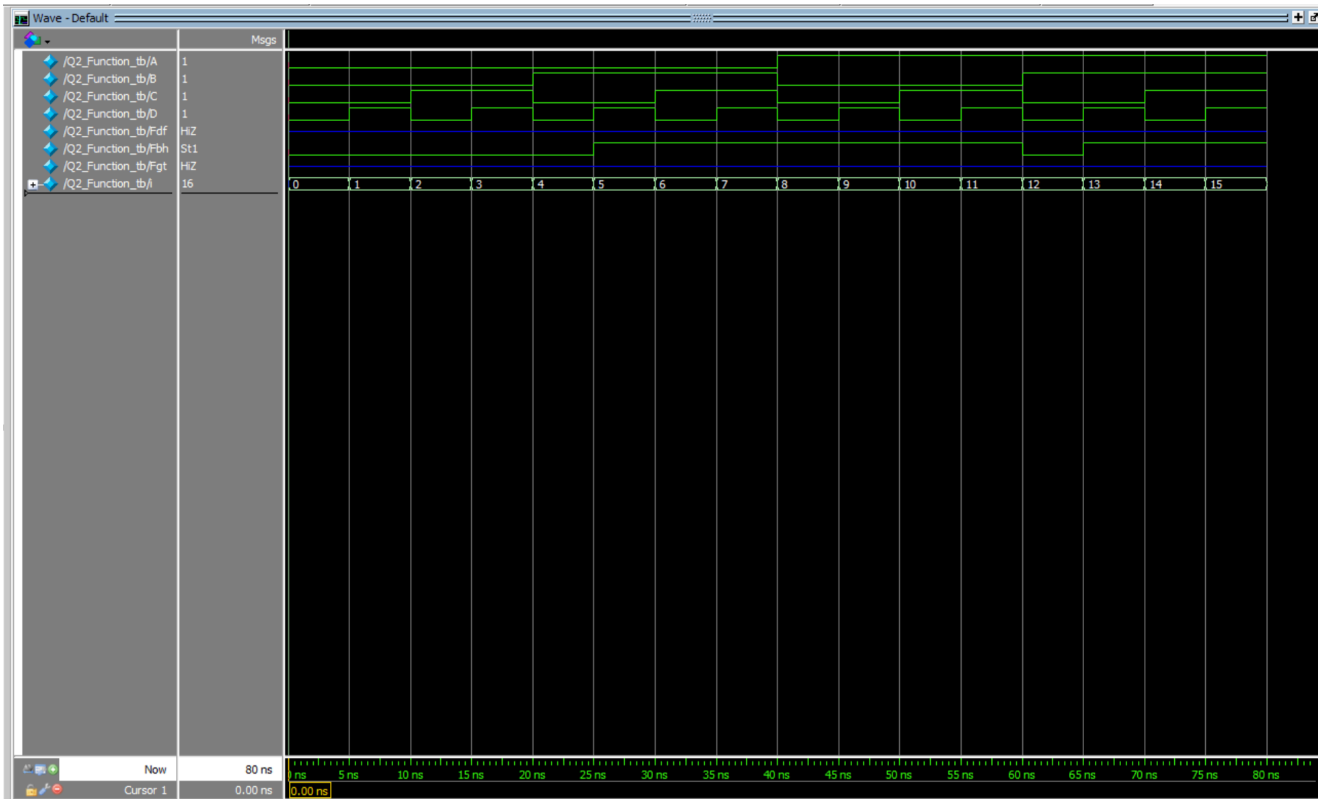
Gate Modelling Code

```
// Q2_Function_gate.v
module Q2_Function_gate(
    input A, B, C, D,
    output F
);
    wire nB, t1, t2, t3, AandnB, BandC, BandD, n1, n2, n3;

    nand (nB, B, B);            // ~B
    nand (t1, A, nB); nand (AandnB, t1, t1);
    nand (t2, B, C); nand (BandC, t2, t2);
    nand (t3, B, D); nand (BandD, t3, t3);

    nand (n1, AandnB, AandnB);
    nand (n2, BandC, BandC);
    nand (n3, BandD, BandD);
    nand (F, n1, n2, n3);
endmodule
```

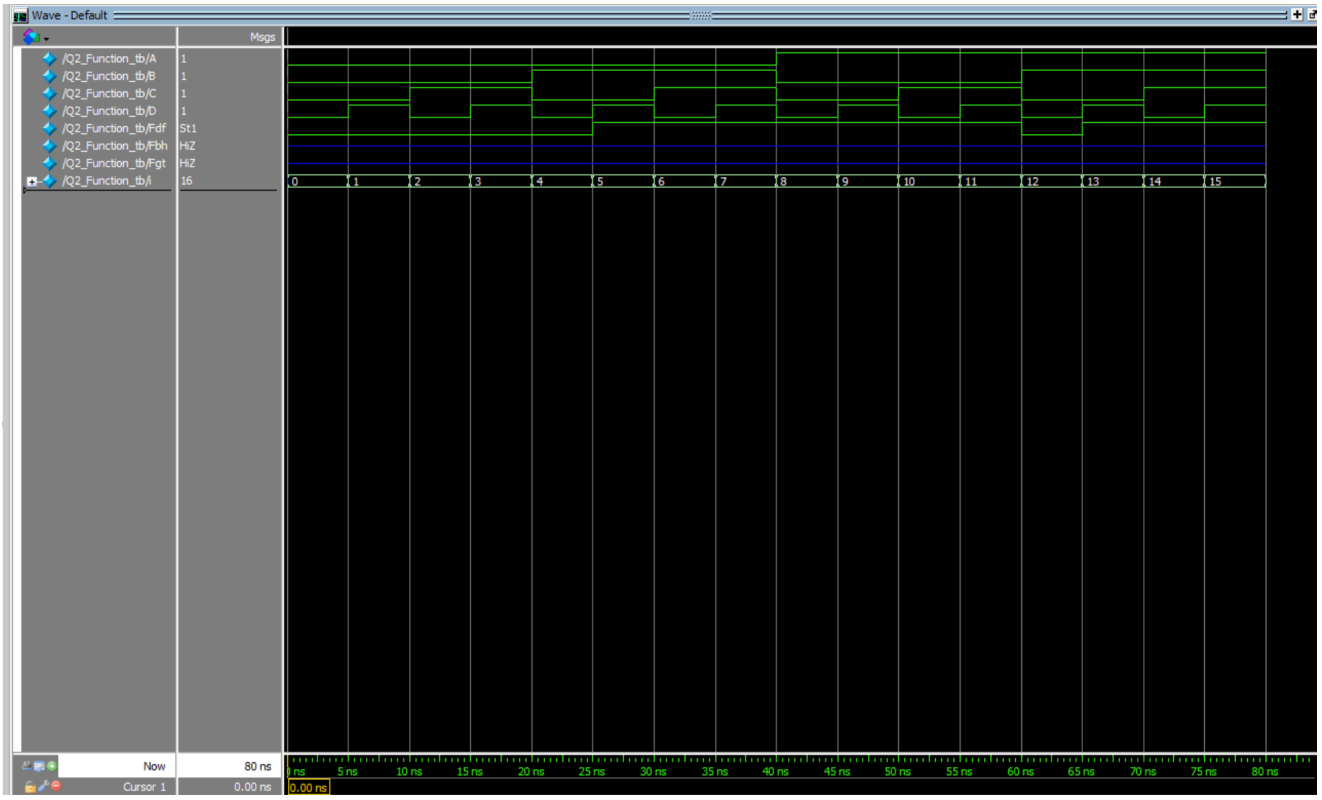
Testbench

```
// Q2_Function_tb.v
`timescale 1ns/1ps
module Q2_Function_tb;
    reg A,B,C,D;
    wire Fdf,Fbh,Fgt;

    Q2_Function_dataflow   dut1 (A,B,C,D,Fdf);
    //Q2_Function_behavioral dut2 (A,B,C,D,Fbh);
    //Q2_Function_gate       dut3 (A,B,C,D,Fgt);

    integer i;
    initial begin
        $display("A B C D | DF BH GT");
        for (i=0; i<16; i=i+1) begin
            {A,B,C,D}=i; #5;
            $display("%b %b %b %b | %b  %b  %b", A,B,C,D,Fdf,Fbh,Fgt);
        end
        $finish;
    end
endmodule
```

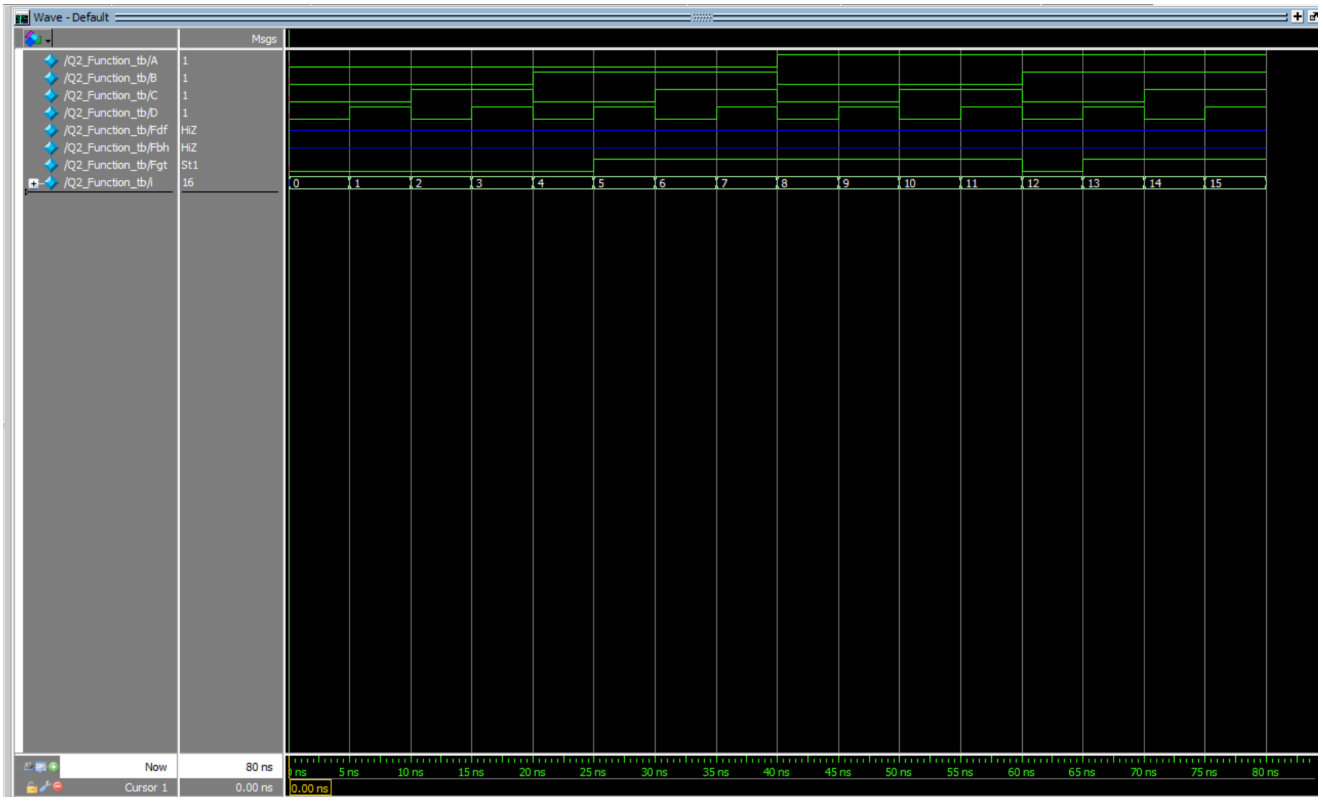
Behavioral Modelling Output



Dataflow Modelling Output



Gate Modelling Output



Question 3

```
// PMOS_not.v
// Functional PMOS: when gate=0, output = 1 (pulls up), else 0
module PMOS_not(
    input  gate,
    output out
);
    not g1 (out, gate);
endmodule
```

```
// NMOS_buf.v
// Functional NMOS: when gate=1, output = 1 (pulls down), else 0
module NMOS_buf(
    input  gate,
    output out
);
    buf g1 (out, gate);
endmodule
```

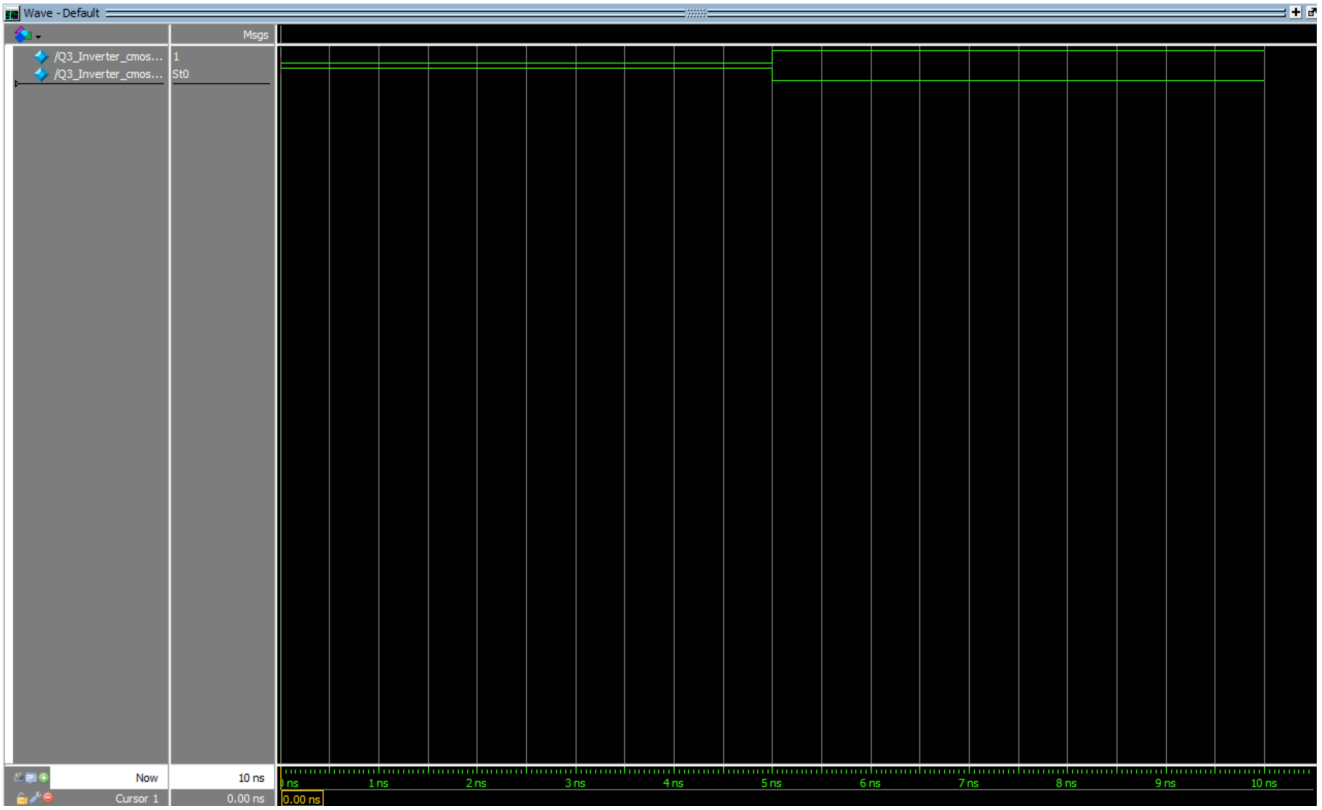
Code

```
// Q3_Inverter_cmos.v
module Q3_Inverter_cmos(
    input  A,
    output Y
);
    wire p_drive, n_drive;

    // PMOS pulls up when A=0
    PMOS_not p1 (A, p_drive);

    // NMOS pulls down when A=1
    NMOS_buf n1 (A, n_drive);

    // Output = VDD if PMOS active, GND if NMOS active
    assign Y = p_drive & ~n_drive;
endmodule
```

Testbench

```
// tb_Q3_Inverter_cmos.v
`timescale 1ns/1ps
module Q3_Inverter_cmos_tb;
    reg A;
    wire Y;

    Q3_Inverter_cmos dut (A, Y);

    initial begin
        $display("A | Y");
        A=0; #5; $display("%b | %b",A,Y);
        A=1; #5; $display("%b | %b",A,Y);
        $finish;
    end
endmodule
```

Output



Question 4

```
// PMOS_not.v
// Functional PMOS: when gate=0, output = 1 (pulls up), else 0
module PMOS_not(
    input  gate,
    output out
);
    not g1 (out, gate);
endmodule
```

```
// NMOS_buf.v
// Functional NMOS: when gate=1, output = 1 (pulls down), else 0
module NMOS_buf(
    input  gate,
    output out
);
    buf g1 (out, gate);
endmodule
```

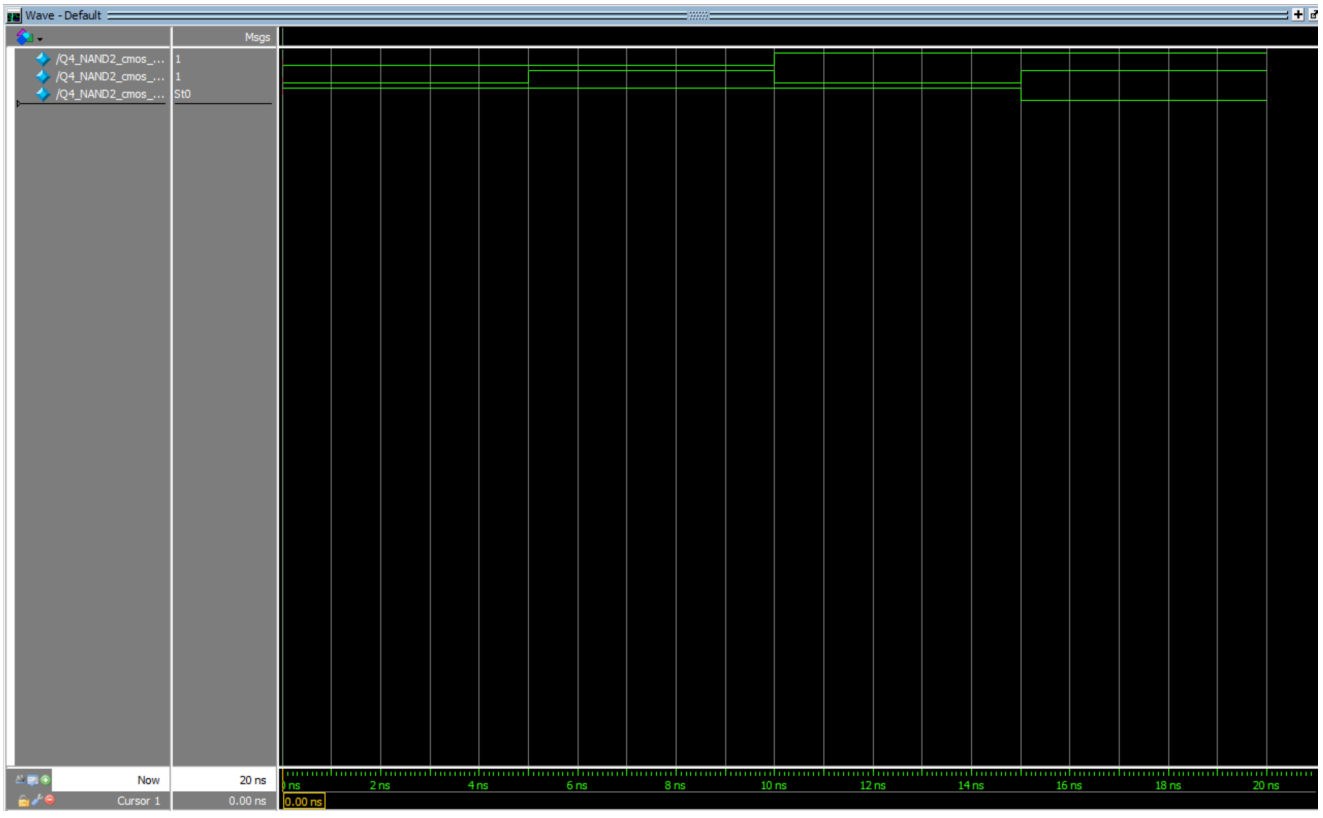
Code

```
// Q4_NAND2_cmos.v
module Q4_NAND2_cmos(
    input  A, B,
    output Y
);
    wire pA, pB;
    wire nA, nB;

    // PMOS pull-up (parallel): active if A=0 OR B=0
    PMOS_not p1 (A, pA);
    PMOS_not p2 (B, pB);

    // NMOS pull-down (series): active only if A=1 AND B=1
    NMOS_buf n1 (A, nA);
    NMOS_buf n2 (B, nB);

    // CMOS NAND = pull-up active OR (not both pull-down active)
    assign Y = (pA | pB) & ~(nA & nB);
endmodule
```

Testbench

```
// tb_Q4_NAND2_cmos.v
`timescale 1ns/1ps
module Q4_NAND2_cmos_tb;
    reg A,B;
    wire Y;

    Q4_NAND2_cmos dut (A, B, Y);

    initial begin
        $display("A B | Y");
        A=0;B=0; #5; $display("%b %b | %b",A,B,Y);
        A=0;B=1; #5; $display("%b %b | %b",A,B,Y);
        A=1;B=0; #5; $display("%b %b | %b",A,B,Y);
        A=1;B=1; #5; $display("%b %b | %b",A,B,Y);
        $finish;
    end
endmodule
```

Output



Question 5

```
// PMOS_not.v
// Functional PMOS: when gate=0, output = 1 (pulls up), else 0
module PMOS_not(
    input  gate,
    output out
);
    not g1 (out, gate);
endmodule
```

```
// NMOS_buf.v
// Functional NMOS: when gate=1, output = 1 (pulls down), else 0
module NMOS_buf(
    input  gate,
    output out
);
    buf g1 (out, gate);
endmodule
```

Code

```
// Q5_XOR_cmos.v
module Q5_XOR_cmos(
    input  A, B,
    output Y
);
    wire pA, pB, nA, nB;

    // PMOS invert
    PMOS_not p1 (A, pA); // ~A
    PMOS_not p2 (B, pB); // ~B

    // NMOS buffer
    NMOS_buf n1 (A, nA); // A
    NMOS_buf n2 (B, nB); // B

    // XOR pull-up when (A & ~B) OR (~A & B)
    assign Y = (nA & pB) | (pA & nB);
endmodule
```

Testbench

```
// tb_Q5_XOR_cmos.v
`timescale 1ns/1ps
module Q5_XOR_cmos_tb;
    reg A,B;
    wire Y;

    Q5_XOR_cmos dut (A, B, Y);

    initial begin
        $display("A B | Y");
        {A,B}=2'b00; #5; $display("%b %b | %b",A,B,Y);
        {A,B}=2'b01; #5; $display("%b %b | %b",A,B,Y);
        {A,B}=2'b10; #5; $display("%b %b | %b",A,B,Y);
        {A,B}=2'b11; #5; $display("%b %b | %b",A,B,Y);
        $finish;
    end
endmodule
```

Output