# DSD LAB

# Week 8 Assignment Submission

**Swamiraju Satya Praveen Varma**

**200905044**

**Batch B1**

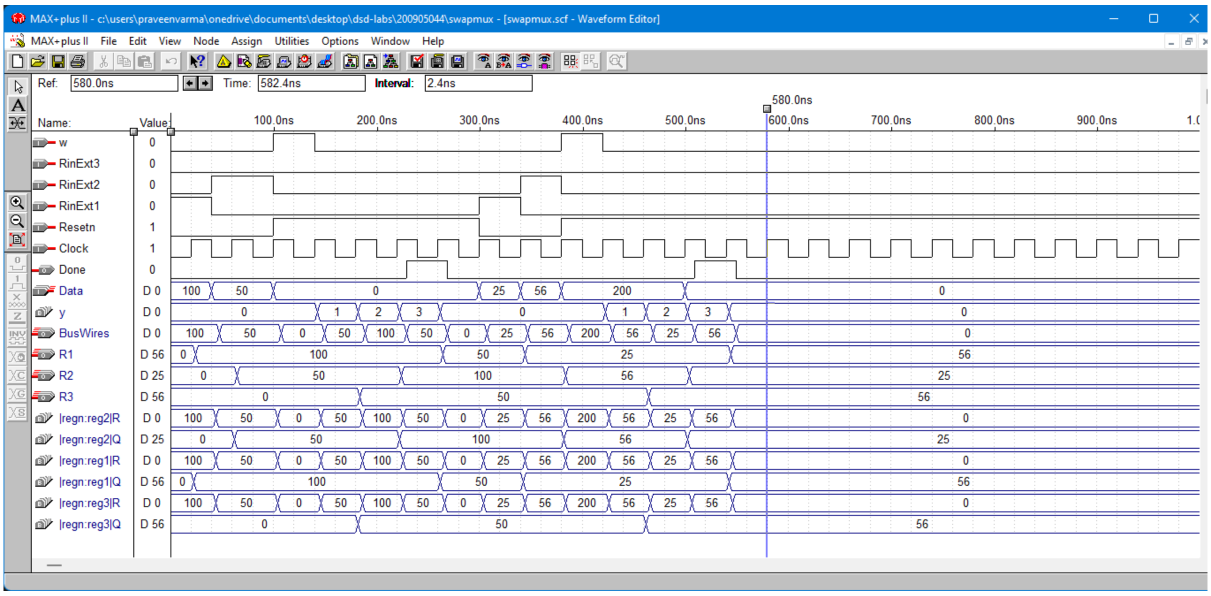
**10**

1. **Write and simulate the Verilog code to swap the contents of two registers using multiplexers.**

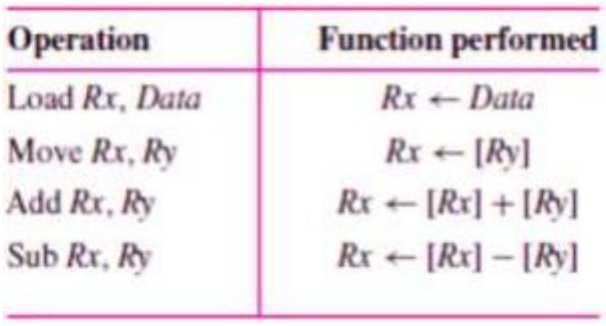
   **Verilog code:**

```
module regn(R, L, Clock, Q);

        parameter n = 8;
        input [n-1:0] R;
        input L, Clock;
        output[n-1:0] Q;
        reg [n-1:0] Q;

        always @(posedge Clock)
                if (L)
                        Q <= R;
endmodule


module swapmux (Resetn, Clock, w, Data, RinExt1, RinExt2, RinExt3, BusWires,
Done,R1,R2,R3);

        parameter n = 8;
        input Resetn, Clock, w, RinExt1, RinExt2, RinExt3;
        input [n-1:0] Data;
        output [n-1:0] BusWires;
        reg [n-1:0] BusWires;
        output Done;
        output [n-1:0] R1, R2, R3;
        reg [n-1:0] R1,R2,R3;
        wire R1in, R2in, R3in;
        reg [2:1] y, Y;
        parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10, D = 2'b11;

        // Define the next state combinational circuit for FSM
        always @(w or y)
                case (y)
                        A: if (w)
                                        Y = B;
                          else
                                        Y = A;
                        B: Y = C;
                        C: Y = D;
                        D: Y = A;
                endcase

        always @(negedge Resetn or posedge Clock)
                if (Resetn == 0)
                        y<=A;
                else
                        y<=Y;

                // Define control signals
                assign R3in = (y == B);
                assign R2in = (y == C);
                assign R1in = (y == D);
                assign Done = (y == D);

                // Instantiate registers
                regn reg1 (BusWires, RinExt1 | R1in, Clock, R1);
                regn reg2 (BusWires, RinExt2 | R2in, Clock, R2);
                regn reg3 (BusWires, RinExt3 | R3in, Clock, R3);

                // Define the multiplexers
                always @(y or Data or R1 or R2 or R3)
                if (y == A)
                        BusWires = Data;
                else if (y == B)
                        BusWires = R2;
                else if (y == C)
                        BusWires = R1;
                else
                        BusWires = R3;
endmodule
```

## Output waveform:



## 2. Simulate a simple processor that can perform the following functions:



| Operation | Function performed |
|---|---|
| Load *Rx*, *Data* | *Rx* ← *Data* |
| Move *Rx*, *Ry* | *Rx* ← [*Ry*] |
| Add *Rx*, *Ry* | *Rx* ← [*Rx*] + [*Ry*] |
| Sub *Rx*, *Ry* | *Rx* ← [*Rx*] − [*Ry*] |

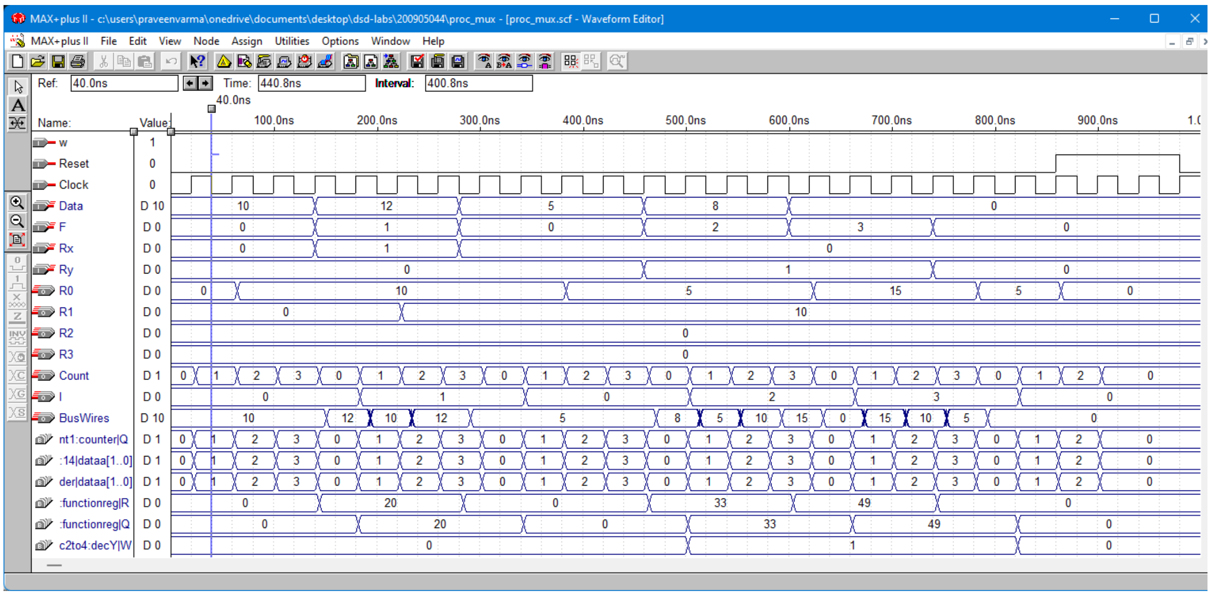
## Verilog code:

```
module dec2to4 (W, En, Y);

        input [1:0]W;
        input En;
        output [0:3] Y;
        reg [0:3] Y;

        always @(W or En)
                case ({En,W})
                        3'b100: Y = 4'b1000;
                        3'b101: Y = 4'b0100;
                        3'b110: Y = 4'b0010;
                        3'b111: Y = 4'b0001;
                        default: Y = 4'b0000;
                endcase
```

```
endmodule


module regn2(R, L, Clock, Q);

        parameter n = 4;
        input [n-1:0] R;
        input L, Clock;
        output[n-1:0] Q;
        reg [n-1:0] Q;

        always @(posedge Clock)
                if (L)
                        Q <= R;
endmodule


module regn3(R, L, Clock, Q);

        parameter n = 6;
        input [n-1:0] R;
        input L, Clock;
        output[n-1:0] Q;
        reg [n-1:0] Q;

        always @(posedge Clock)
                if (L)
                        Q <= R;
endmodule


module upcount1(Clear, Clock, Q);

        input Clear, Clock;
        output [1:0] Q;
        reg [1:0] Q;

        always @(posedge Clock)
                if (Clear)
                        Q <= 0;
                else
                        Q <= Q + 1;
endmodule
```

```
module proc_mux(Data, Reset, w, Clock, F, Rx, Ry,R0, R1, R2, R3, Count, I, BusWires);

        input [3:0] Data;
        input Reset, w, Clock;
        input [1:0] F, Rx, Ry;
        output [1:0] Count, I;
        output [3:0] BusWires, R0, R1, R2, R3;

        reg [3:0] BusWires;
        reg [3:0] Sum;
        reg [0:3] Rin, Rout;
        reg Extern, Ain, Gin, Gout, AddSub;

        //wire [1:0] Count, I;
        wire [0:3] Xreg, Y;
        wire [3:0] A, G;
        wire [1:6] Func, FuncReg, Sel;

        upcount1 counter (Reset, Clock, Count);
        assign Func = {F, Rx, Ry};
        wire FRin = w & ~Count[1] & ~Count[0];
        regn3 functionreg (Func, FRin, Clock, FuncReg);

        //defparam functionreg.n = 6;
        assign I = FuncReg[1:2];
        dec2to4 decX (FuncReg[3:4], 1'b1, Xreg);
        dec2to4 decY (FuncReg[5:6], 1'b1, Y);

        always @(Count or I or Xreg or Y)
        begin
                Extern = 1'b0; Ain = 1'b0; Gin = 1'b0;
                Gout = 1'b0; AddSub = 1'b0; Rin = 4'b0; Rout = 4'b0;

                case (Count)
                        2'b00: ; //no signals asserted in time step T0

                        2'b01: //define signals in time step T1
                                case (I)
                                        2'b00: begin //Load
                                                Extern = 1'b1; Rin = Xreg;
                                                end
                                        2'b01: begin //Move
                                                Rout = Y; Rin = Xreg;
                                                end

                                default: begin //Add, Sub
                                                Rout = Xreg; Ain = 1'b1;
                                                end
                           endcase

                        2'b10: //define signals in time step T2
                           case(I)
                                        2'b10: begin //Add
                                                Rout = Y; Gin = 1'b1;
                                                end

                                        2'b11: begin //Sub
                                                Rout = Y; AddSub = 1'b1; Gin = 1'b1;
                                                end

                                default: ; //Add, Sub
                           endcase

                        2'b11:
                                case (I)
                                        2'b10, 2'b11: begin
                                                                   Gout = 1'b1; Rin = Xreg;
                                                                   end
                                        default: ; //Add, Sub
                                endcase
        endcase
end



        regn2 reg_0 (BusWires, Rin[0], Clock, R0);
        regn2 reg_1 (BusWires, Rin[1], Clock, R1);
        regn2 reg_2 (BusWires, Rin[2], Clock, R2);
        regn2 reg_3 (BusWires, Rin[3], Clock, R3);
        regn2 reg_A (BusWires, Ain, Clock, A);

// alu
        always @(AddSub or A or BusWires)
                begin
                        if (!AddSub)
                                Sum = A + BusWires;
                else
                        Sum = A - BusWires;
        end

        regn2 reg_G (Sum, Gin, Clock, G);
        assign Sel = {Rout, Gout, Extern};

        always @(Sel or R0 or R1 or R2 or R3 or G or Data)
        begin
                if (Sel == 6'b100000)
                        BusWires = R0;

                else if (Sel == 6'b010000)
                        BusWires = R1;

                else if (Sel == 6'b001000)
                        BusWires = R2;

                else if (Sel == 6'b000100)
                        BusWires = R3;

                else if (Sel == 6'b000010)
                        BusWires = G;

                else
                        BusWires = Data;
        end
endmodule
```

**Output waveform:**


MAX+plus II - c:\users\praveenvarma\onedrive\documents\desktop\dsd-labs\200905044\proc_mux - [proc_mux.scf - Waveform Editor]
MAX+plus II File Edit View Node Assign Utilities Options Window Help
Ref: 40.0ns
Time: 440.8ns
Interval: 400.8ns
40.0ns
100.0ns 200.0ns 300.0ns 400.0ns 500.0ns 600.0ns 700.0ns 800.0ns 900.0ns
Name: Value:
w 1
Reset 0
Clock 0
Data D 10 — 10, 12, 5, 8, 0
F D 0 — 0, 1, 0, 2, 3, 0
Rx D 0 — 0, 1, 0
Ry D 0 — 0, 1, 0
R0 D 0 — 0, 10, 5, 15, 5, 0
R1 D 0 — 0, 10
R2 D 0 — 0
R3 D 0 — 0
Count D 1 — 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 0
I D 0 — 0, 1, 0, 2, 3, 0
BusWires D 10 — 10, 12, 10, 12, 5, 8, 5, 10, 15, 0, 15, 10, 5, 0
nt1:counter|Q D 1 — 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 0
:14|dataa[1..0] D 1 — 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 0
der|dataa[1..0] D 1 — 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 0
:functionreg|R D 0 — 0, 20, 0, 33, 49, 0
:functionreg|Q D 0 — 0, 20, 0, 33, 49, 0
c2to4:decY|W D 0 — 0, 1, 0

# THANK YOU!