

Manoj M Mallya

200905130

Section : C2

Roll no. : 23

DSDL – 8

Exercises :

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 // Code for 8-bit register
```

```
module l8q1 (Resetn, Clock, w, Data, Extern, R1, R2, R3, BusWires, Done);  
parameter n = 8;  
input Resetn, Clock, w, Extern;  
input [n-1:0] Data;  
output [n-1:0] BusWires ,R1, R2, R3 ;
```

```

reg [n-1:0] BusWires, R1, R2, R3;
output Done;
wire R1in, R1out, R2in, R2out, R3in, R3out, RinExt1, RinExt2;
reg [2:0] y, Y;
parameter [2:0] A = 3'b000, B = 3'b001, C = 3'b010, D = 3'b011, E = 3'b100;
// Define the next state combinational circuit for FSM
always @(w or y)
begin
case (y)
A: if (w) Y = B;
else Y = A;
B: Y = C;
C: Y = D;
D: Y = E;
E: Y = A;
endcase
end

// Define the sequential block for FSM
always @(negedge Resetn or posedge Clock)
begin
if (Resetn == 0) y <= A;
else y <= Y;
end

// Define outputs of FSM
assign RinExt1 = (y == A);
assign RinExt2 = (y == B);

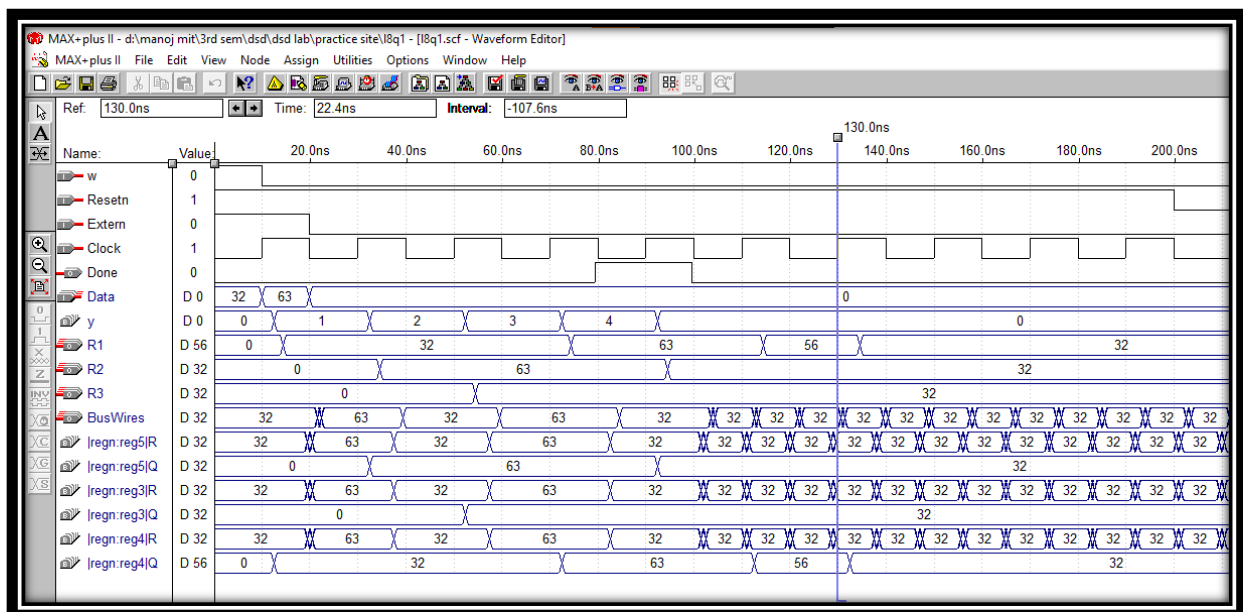
```

```

assign R3in = (y == C);
assign R1out = (y == C);
assign R2out = (y == D);
assign R1in = (y == D);
assign R3out = (y == E);
assign R2in = (y == E);
assign Done = (y == E);
always @(Extern or R1out or R2out or R3out)
if (Extern) BusWires = Data;
else if (R1out) BusWires = R1;
else if (R2out) BusWires = R2;
else if (R3out) BusWires = R3;
regn reg3 (BusWires, R3in, Clock, R3);
regn reg4 (BusWires, RinExt1 | R1in, Clock, R1);
regn reg5 (BusWires, RinExt2 | R2in, Clock, R2);
endmodule

```

Output:



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

Operation	Function performed
Load R_x , $Data$	$R_x \leftarrow Data$
Move R_x , R_y	$R_x \leftarrow [R_y]$
Add R_x , R_y	$R_x \leftarrow [R_x] + [R_y]$
Sub R_x , R_y	$R_x \leftarrow [R_x] - [R_y]$

Verilog code:

```

module l8q2(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
```

```
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 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 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 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
```

Output:

