DDCO BCS302 – Verilog Programs (Experiments 1–8)

EXPERIMENT 1

```
module f(a,b,c,d,e);
input a,b,c,d;
output e;
assign e = (a & b) | (c & d);
endmodule
```

EXPERIMENT 2 – FULL ADDER

```
module fa(a,b,cin,sum,cout);
input a,b,cin;
output sum,cout;
assign sum = a ^ b ^ cin;
assign cout = (a & b) | (b & cin) | (a & cin);
endmodule

module fourbitadder(a,b,sum,cout);
input [3:0] a,b;
output [3:0] sum;
output cout;
wire c1,c2,c3;
fa g0(a[0],b[0],1'b0,sum[0],c1);
fa g1(a[1],b[1],c1,sum[1],c2);
fa g2(a[2],b[2],c2,sum[2],c3);
fa g3(a[3],b[3],c3,sum[3],cout);
endmodule
```

EXPERIMENT 2 – FULL SUBTRACTOR

```
module fs(a,b,bin,diff,bout);
input a,b,bin;
output diff,bout;
assign diff = a ^ b ^ bin;
assign bout = (~a & b) | (~a & bin) | (b & bin);
endmodule

module fourbitsub(a,b,diff,bout);
input [3:0] a,b;
output [3:0] diff;
output bout;
wire b1,b2,b3;
```

```
fs g0(a[0],b[0],1'b0,diff[0],b1);
fs g1(a[1],b[1],b1,diff[1],b2);
fs g2(a[2],b[2],b2,diff[2],b3);
fs g3(a[3],b[3],b3,diff[3],bout);
endmodule
```

EXPERIMENT 3 – OR GATE

```
module structural(a,b,y);
input a,b;
output y;
or g1(y,a,b);
endmodule

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

module beh(a,b,y);
input a,b;
output reg y;
always @(*) begin
if(a==0 && b==0) y=0;
else y=1;
end
endmodule
```

EXPERIMENT 4 – HALF ADDER

```
module ha(a,b,sum,carry);
input a,b;
output sum,carry;
assign sum = a ^ b;
assign carry = a & b;
endmodule

module hs(a,b,diff,bout);
input a,b;
output diff,bout;
assign diff = a ^ b;
assign bout = ~a & b;
```

```
endmodule
```

EXPERIMENT 5 – BCD ADDER

```
module bcd(a,b,cin,sum,carry);
input [3:0] a,b;
input cin;
output reg [3:0] sum;
output reg carry;
reg [4:0] temp;
always @(*) begin
temp = a + b + cin;
if(temp > 9) begin
temp = temp + 6;
carry = 1;
end else carry = 0;
sum = temp[3:0];
end
endmodule
```

EXPERIMENT 6 – MULTIPLEXERS

```
module mux2_1(I,sel,y);
input [1:0] I;
input sel;
output reg y;
always @(*) y = sel ? I[1] : I[0];
endmodule

module mux4_1(I,sel,y);
input [3:0] I;
input [1:0] sel;
output reg y;
always @(*) begin
case(sel)
2'b00:y=I[0];
2'b01:y=I[1];
2'b10:y=I[2];
default:y=I[3];
endcase
end
endmodule
```

```
module mux8_1(I,sel,y);
input [7:0] I;
input [2:0] sel;
output reg y;
always @(*) y = I[sel];
endmodule
```

EXPERIMENT 7 – DEMULTIPLEXERS

```
module demux1_2(I,sel,y);
input I,sel;
output reg [1:0] y;
always @(*) begin
y=0; y[sel]=I;
end
endmodule

module demux1_4(I,sel,y);
input I;
input [1:0] sel;
output reg [3:0] y;
always @(*) begin
y=0; y[sel]=I;
end
endmodule

module demux1_8(I,sel,y);
input I;
input [2:0] sel;
output reg [7:0] y;
always @(*) begin
y=0; y[sel]=I;
end
endmodule
```

EXPERIMENT 8 – FLIP FLOPS

```
module srff(clk,s,r,q);
input clk,s,r;
output reg q;
always @(posedge clk) begin
if(s==0 && r==0) q<=q;
```

```
else if(s==0 && r==1) q<=0;
else if(s==1 && r==0) q<=1;
else q<=1'bx;
end
endmodule

module jkff(clk,j,k,q);
input clk,j,k;
output reg q;
always @(posedge clk) begin
case({j,k})
2'b00:q<=q;
2'b01:q<=0;
2'b10:q<=1;
2'b11:q<=~q;
endcase
end
endmodule

module dff(clk,d,q);
input clk,d;
output reg q;
always @(posedge clk)
q<=d;
endmodule
```