**AND GATE:**

Module AND\_gate(

Input A,

Input B,

Output Y

);

Assign Y = A & B;

End module

**OR GATE:**

Module OR\_gate(

Input A,

Input B,

Output Y

);

Assign Y = A | B;

End module

NOT GATE:

Module NOT\_gate(

Input A,

Output Y

);

Assign Y = ~A;

End module

**MULTIPLEXER 2 to 1 :**

Module mux2to1(

Input A,

Input B,

Input select,

Output Y

);

Assign Y = (select) ? B : A;

End module

**Decoder 2 to 4:**

module decoder2to4(

input [1:0] A, // 2-bit input

output [3:0] Y // 4-bit output

);

assign Y = (A == 2'b00) ? 4'b0001 :

(A == 2'b01) ? 4'b0010 :

(A == 2'b10) ? 4'b0100 :

(A == 2'b11) ? 4'b1000 : 4'b0000;

end module

**4 Bit full adder:**

Module full\_adder(

Input [3:0] A, // 4-bit input A

Input [3:0] B, // 4-bit input B

Input Cin, // Carry-in

Output [3:0] Sum, // 4-bit Sum

Output Cout // Carry-out

);

Wire [3:0] C; // Intermediate carries

// Full adder for each bit

Full\_adder\_bit fa0 (.A(A[0]), .B(B[0]), .Cin(Cin), .Sum(Sum[0]), .Cout(C[0]));

Full\_adder\_bit fa1 (.A(A[1]), .B(B[1]), .Cin(C[0]), .Sum(Sum[1]), .Cout(C[1]));

Full\_adder\_bit fa2 (.A(A[2]), .B(B[2]), .Cin(C[1]), .Sum(Sum[2]), .Cout(C[2]));

Full\_adder\_bit fa3 (.A(A[3]), .B(B[3]), .Cin(C[2]), .Sum(Sum[3]), .Cout(C[3]));

Assign Cout = C[3]; // Final carry-out

Endmodule

Module full\_adder\_bit(

Input A, B, Cin,

Output Sum, Cout

);

Assign Sum = A ^ B ^ Cin; // XOR for sum

Assign Cout = (A & B) | (Cin & (A ^ B)); // Carry-out logic

End module