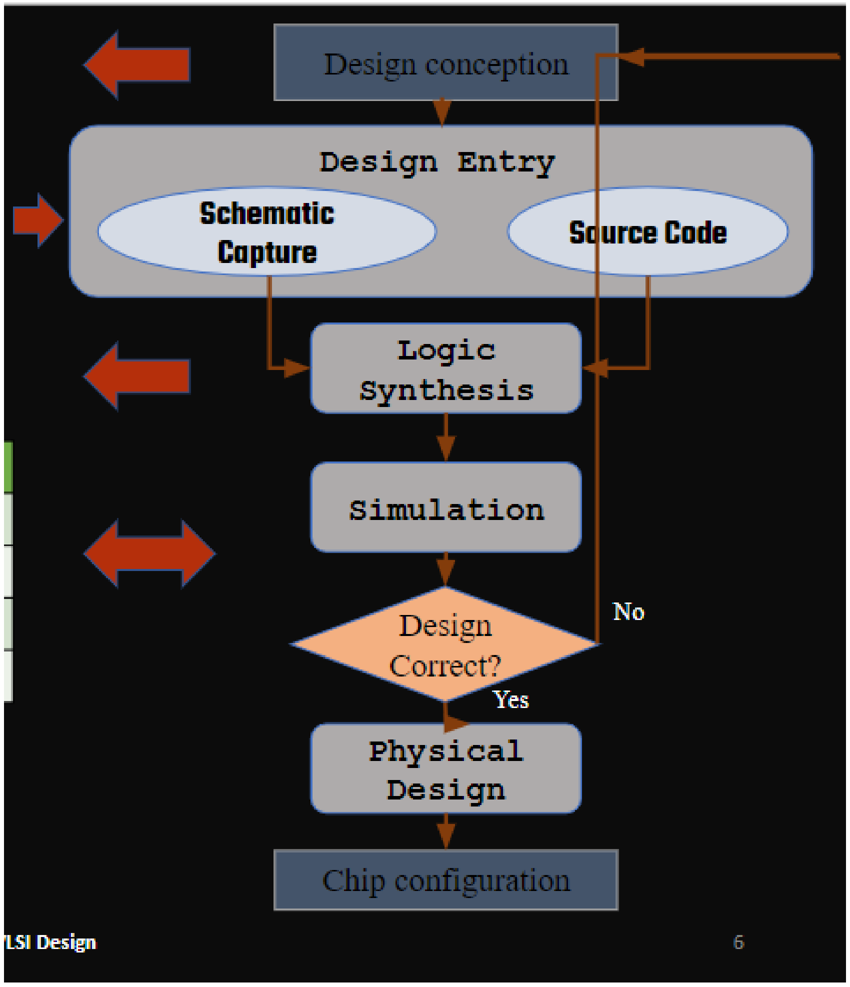
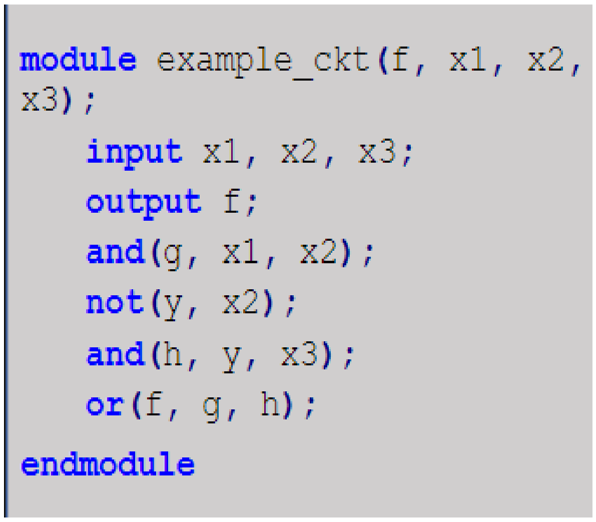
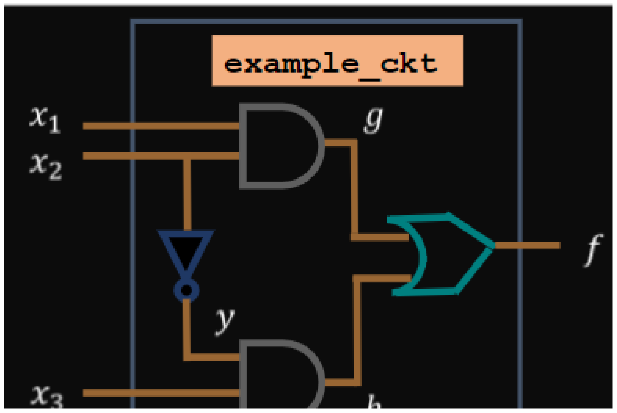


Structural Representation

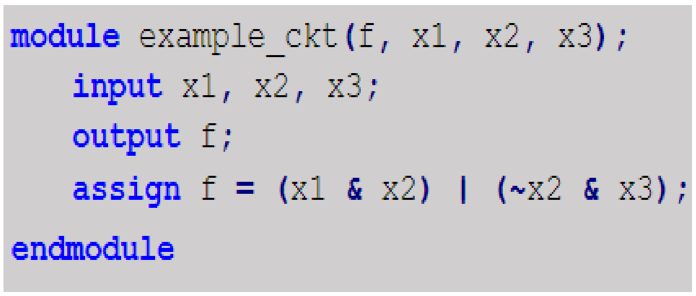


```
module example_ckt(f, x1, x2,
x3);
    input x1, x2, x3;
    output f;
    and(g, x1, x2);
    not(y, x2);
    and(h, y, x3);
    or(f, g, h);
endmodule
```

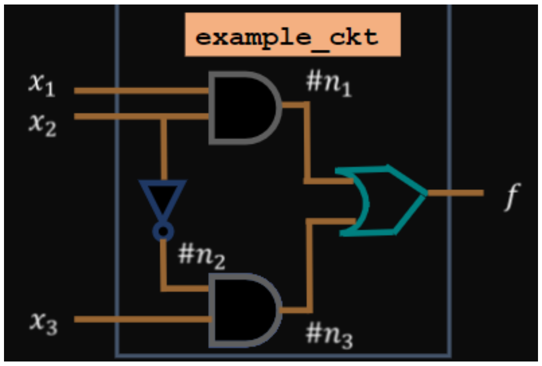


and(g,x1,x2); g=x1&x2

Behavioral Representation



```
module example_ckt(f, x1, x2, x3);
   input x1, x2, x3;
   output f;
   assign f = (x1 & x2) | (~x2 & x3);
endmodule
```



Bitwise operators:

☐ ~A=This will produce 1's complement of A
☐ -A=This will produce 2's complement of A
☐ A&B=Bitwise AND
☐ A|B=Bitwise OR
☐ A^B=Bitwise XOR
☐ A^~B / A~^B=Bitwise XNOR
☐ !A=NOT A(!A)produces "1(True)" only if all bits of A are 0 else !A gives "0(False)"
☐ A && B=The result of A && B is "1(True)" if both A and B are nonzero
☐ A || B=A || B gives "1(True)" unless both A and B are zero
☐ A+B=Addition of two single or multi bit numbers
☐ A-B=Subtraction of two single or multi bit numbers
☐ A*B=Multiplication of two single or multi bit numbers
☐ A/B=Division of of two single or multi bit numbers
☐ A%B=This returns the remainder of the integer division A/B
☐ A==B =1(True) if A is equal to B,0(False)otherwise
☐ A!=B =1(True) if A is not equal to B,0(False)otherwise
☐ A>B =1(True) if A is greater than B,0(False)otherwise
☐ A<B =1(True) if A is less than B,0(False)otherwise
☐ >> =Shift right logical (division by 2)
☐ << =Shift left logical(multiplication by 2)

- ☐ {} =join bits
- ☐ {n{m}} =Duplicate m,n times
- ☐ D=A?B:C = D is equal to B if A is true,otherwise D is equal to C

```
module example(a, b);
input b;
output a;
wire [7:0] in1, in2; // 8-bit wire type
variables
reg [0:31] base_clk; // 32-bit reg type
variable
assign a = in1[7] * in2[2];
endmodule
```

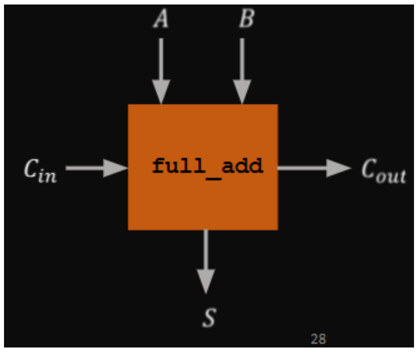
wire a,b; //wire declaration,changes continuously
reg clock; //register declaration,does not changes continuously

wire[7:0] in1,in2 //8 bit wire type variables wire[7]=MSB
reg[0:31] base_clk; //32 bit wire type variables

reg[7:0] mem[15:0]; //mem is an array has 16 column
reg[7:0] mem[7:0][15:0]; //mem is an array has 8 row 16 column
parameter n=5; //declaring constant

```
module full_add(S, Cout, A, B, Cin);
// This module implements a 1-bit full adder
   input A, B, Cin;
   output S, Cout;

   assign S = A ^ B ^ Cin;
   assign Cout = (A & B) | (Cin & (A ^ B));
endmodule
```



```
always @ (sensitivity_list)
[begin]
     [procedural assignment statements]
     [if-else statements]
     [case statements]
     [while, repeat and for loops]
[end]
```

```
always @(*)
     if(expression1)
     begin
        statement;
     end
     else if(expression2)
     begin
        statement;
     end
     else
     begin
          statement
     end
end
```

```
module mux2to1(A, B, S, Z);

     input A, B, S;
     output reg Z;

     always @(A, B, S)
     begin
    if(S == 0)
          Z = A;
    else
          Z = B;
     end

endmodule
```

4-Bit Ripple Carry Adder

```
module fulladd4(S0, S1, S2, S3, Cout, A0, A1, A2, A3, B0, B1, B2, B3, Cin);
    input A0, A1, A2, A3, B0, B1, B2, B3, Cin;
    output S0, S1, S2, S3, Cout;
    wire Cout0, Cout1, Cout2;
    fulladd stage0 (S0, Cout0, A0, B0, Cin);
    fulladd stage1 (S1, Cout1, A1, B1, Cout0);
    fulladd stage2 (S2, Cout2, A2, B2, Cout1);
    fulladd stage3 (S3, Cout, A3, B3, Cout2);
endmodule

module fulladd(S, Cout, A, B, Cin);
// This module implements a 1-bit full adder
   input A, B, Cin;
   output S, Cout;

   assign S = A ^ B ^ Cin;
   assign Cout = (A & B) | (Cin & (A ^ B));
endmodule
```