# EGC220 Class Notes 3/31/2023

**Baback Izadi**

Division of Engineering Programs

bai@engr.newpaltz.edu

# Taste of Verilog

Module name

Module ports

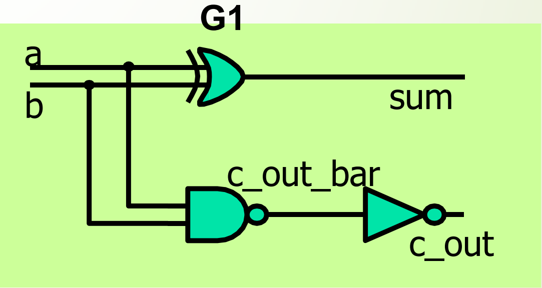
```
module Add_half ( sum, c_out, a, b );
   input      a, b;
   output     sum, c_out;
   wire       c_out_bar;

   xor (sum, a, b);
   // xor G1(sum, a, b);
   nand (c_out_bar, a, b);
   not (c_out, c_out_bar);
endmodule
```

Declaration of port modes

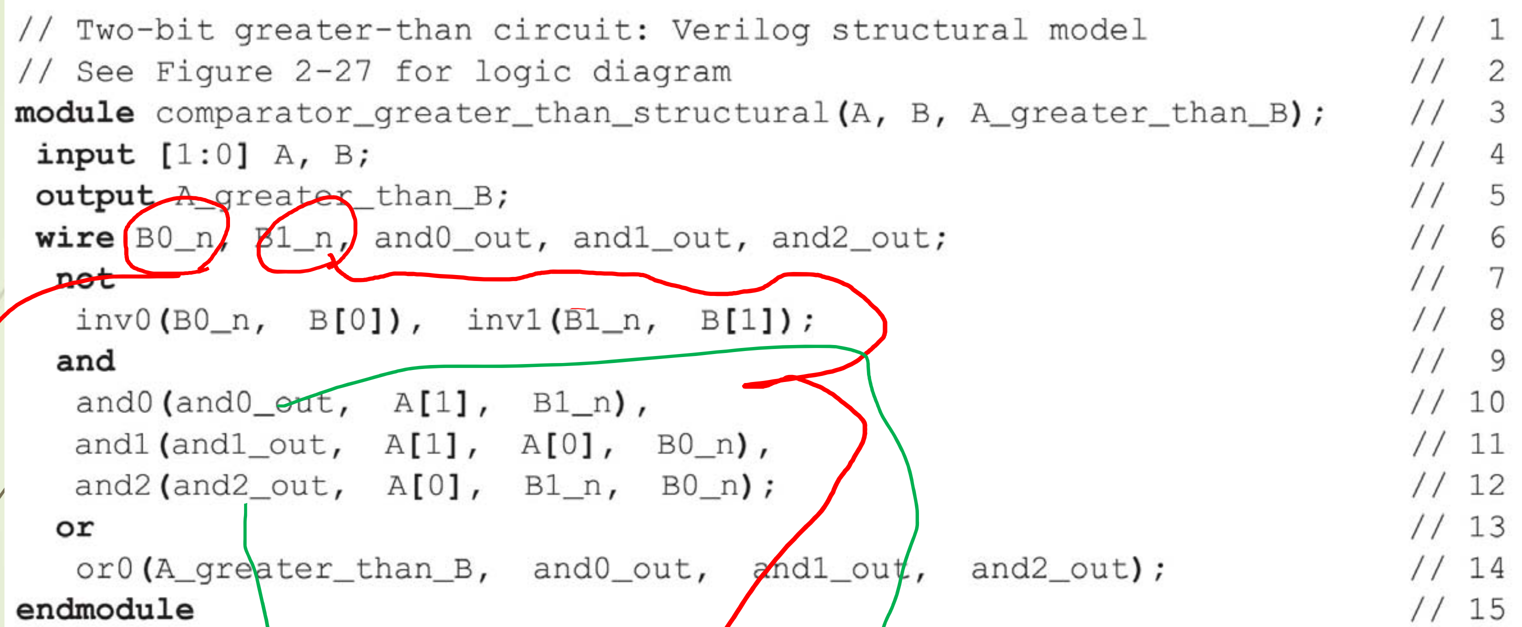
Declaration of internal signal

Instantiation of primitive gates

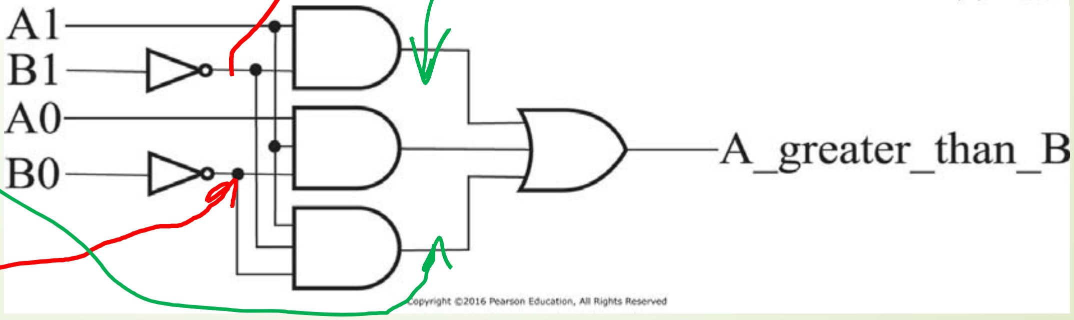
Verilog keywords



# Structural Verilog Description of Two-Bit Greater-Than Circuit



```
// Two-bit greater-than circuit: Verilog structural model
// See Figure 2-27 for logic diagram
module comparator_greater_than_structural(A, B, A_greater_than_B);
 input [1:0] A, B;
 output A_greater_than_B;
 wire B0_n, B1_n, and0_out, and1_out, and2_out;
  not
   inv0(B0_n,  B[0]),  inv1(B1_n,  B[1]);
  and
   and0(and0_out,  A[1],  B1_n),
   and1(and1_out,  A[1],  A[0],  B0_n),
   and2(and2_out,  A[0],  B1_n,  B0_n);
  or
   or0(A_greater_than_B,  and0_out,  and1_out,  and2_out);
endmodule
```



# Dissection

- **Module and Port declarations**
  - Verilog-2001 syntax
    - **module** AOI (input A, B, C, D, output F);
  - Verilog-1995 syntax

```
module AOI (A, B, C, D, F);
 input A, B, C, D;
 output F;
```

- **Wires:** Continuous assignment to an internal signal

# A Simple Dataflow Design

& AND    ^ XOR

| OR

~ NOT

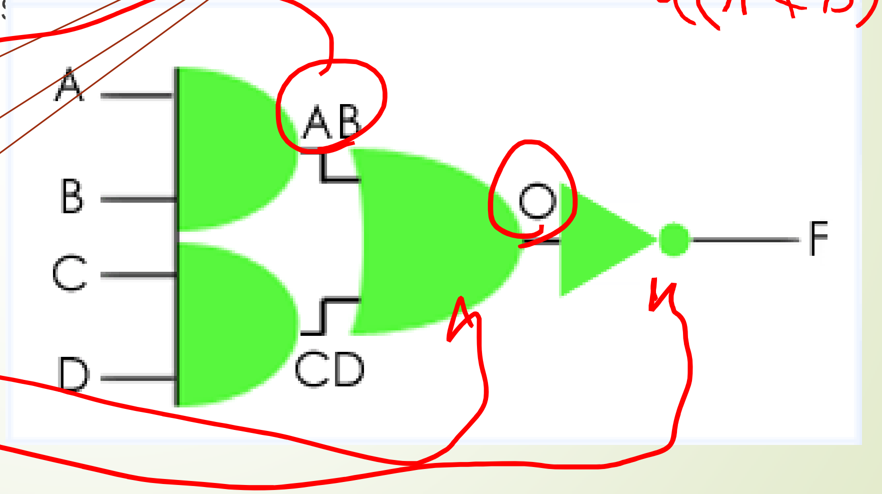
Put equation

$F = \overline{AB + CD}$

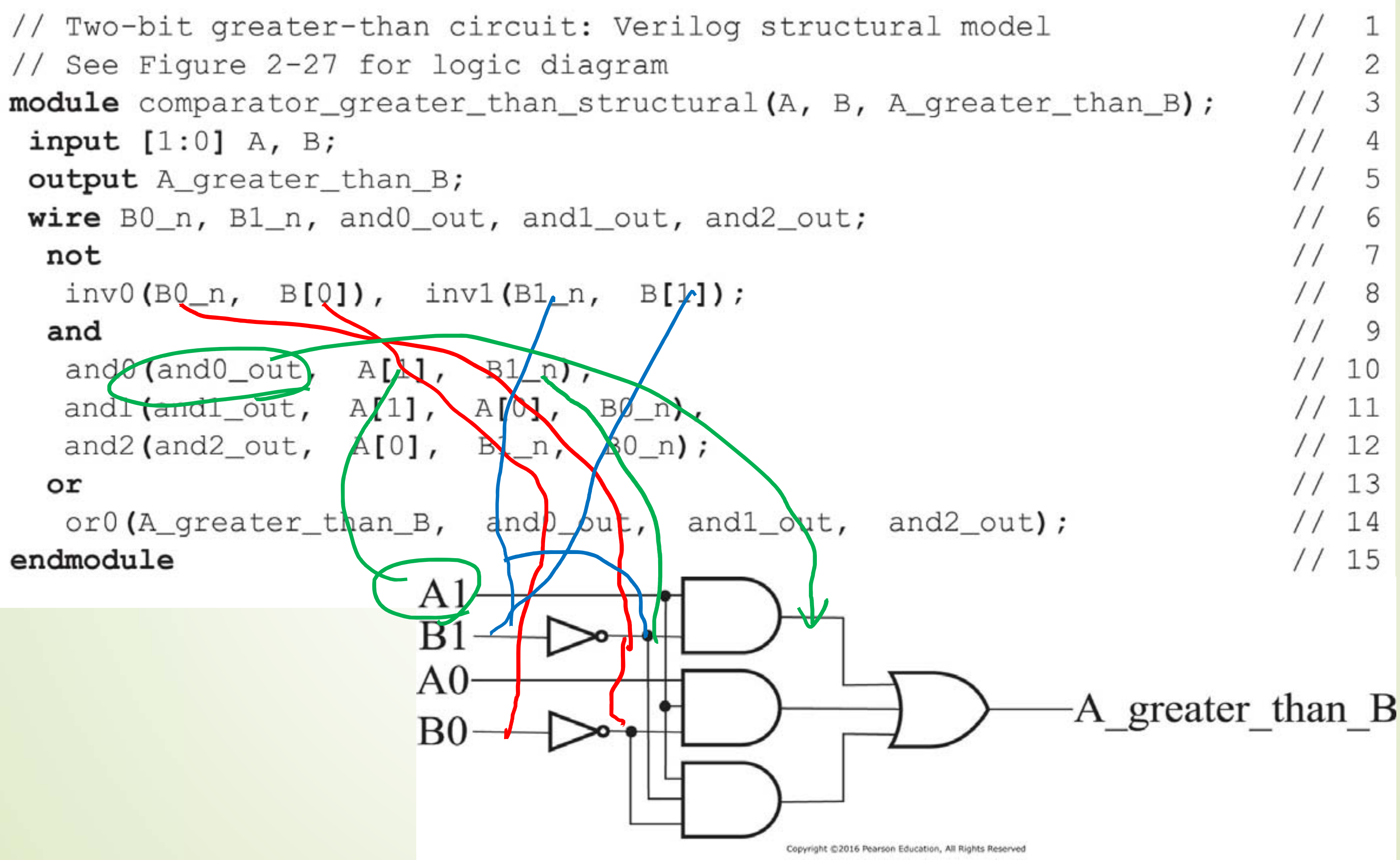
**Continuous Assignments**

assign F=~((A&B)|(C&D));

```
// Verilog code for AND-OR-INVERT gate
module AOI (input A, B, C, D, output F);
  wire F;  // the default
  wire AB, CD, O;  // necess
  assign AB = A & B;
  assign CD = C & D;
  assign O = AB | CD;
  assign F = ~O;
endmodule
// end of Verilog code
```



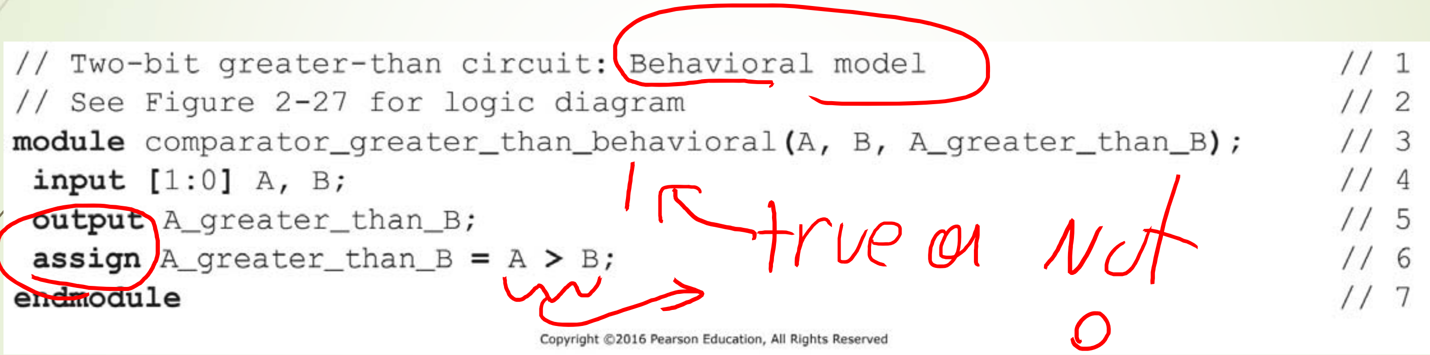
# Structural Verilog Description of Two-Bit Greater-Than Circuit



```
// Two-bit greater-than circuit: Verilog structural model
// See Figure 2-27 for logic diagram
module comparator_greater_than_structural(A, B, A_greater_than_B);
 input [1:0] A, B;
 output A_greater_than_B;
 wire B0_n, B1_n, and0_out, and1_out, and2_out;
  not
   inv0(B0_n,  B[0]),  inv1(B1_n,  B[1]);
  and
   and0(and0_out,  A[1],  B1_n),
   and1(and1_out,  A[1],  A[0],  B0_n),
   and2(and2_out,  A[0],  B1_n,  B0_n);
  or
   or0(A_greater_than_B,  and0_out,  and1_out,  and2_out);
endmodule
```

Copyright ©2016 Pearson Education, All Rights Reserved

# Verilog Description of Two-Bit Greater-Than Circuit



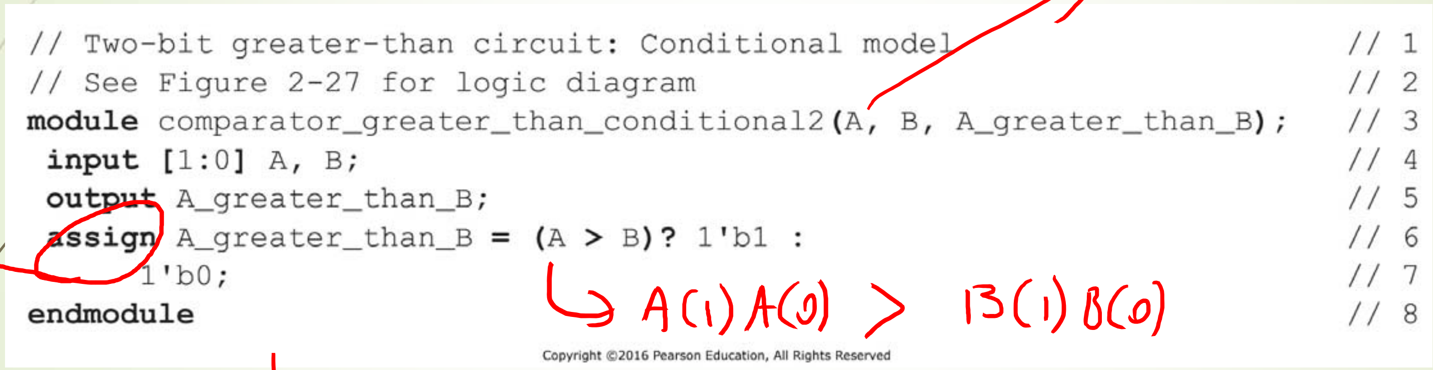
```
// Two-bit greater-than circuit: Behavioral model
// See Figure 2-27 for logic diagram
module comparator_greater_than_behavioral(A, B, A_greater_than_B);
 input [1:0] A, B;
 output A_greater_than_B;
 assign A_greater_than_B = A > B;
endmodule
```

1 true or Not 0

= (chock)? true result; False_result

# Conditional Dataflow Verilog Description of Two-Bit Greater-Than Circuit

A1, A0, B1, B0



```
// Two-bit greater-than circuit: Conditional model
// See Figure 2-27 for logic diagram
module comparator_greater_than_conditional2(A, B, A_greater_than_B);
 input [1:0] A, B;
 output A_greater_than_B;
 assign A_greater_than_B = (A > B)? 1'b1 :
        1'b0;
endmodule
```

↳ A(1) A(0) > B(1) B(0)

input w;

input [3:0] X;

X(3), X(2), X(1), X(0)

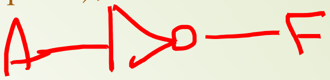
# A Design Hierarchy

- **Module Instances**
  - MUX_2 module contains references to each of the lower level modules

```
// Verilog code for 2-input multiplexer
 module MUX2 (input SEL, A, B, output F);      // 2:1 multiplexer
// wires SELB and FB are implicit
// Module instances...
 INV G1 (SEL, SELB);
 AOI G2 (SELB, A, SEL, B, FB);
 INV G3 (.A(FB), .F(F));    // Named mapping
endmodule
// end of Verilog code
```

1 2 3 (FB , F)

```
// Verilog code for 2-input multiplexer
module INV (input A, output F);   // An inverter
  assign F = ~A;
endmodule

module AOI (input A, B, C, D, output F);
  assign F = ~((A & B) | (C & D));
endmodule
```



AB + CD

F = (SEL)’. A + (SEL).B
SELB = (SEL)’
F=(SELB).A + (SEL).B
1. Invert SEL and get SELB
2. Use AOI and get F’
3. Invert F’ and get F

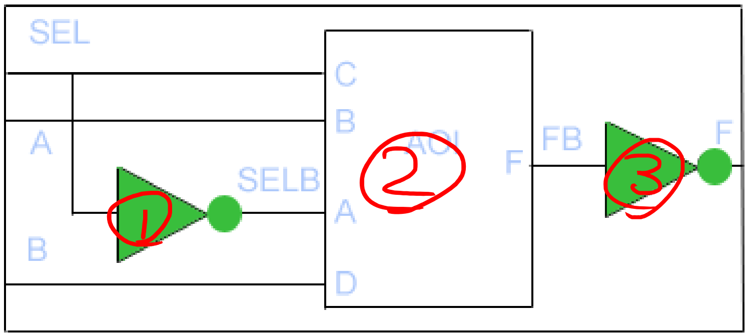
| SEL | F |
|---|---|
| 0 | A |
| 1 | B |

A SELB B

F = $\overline{SEL}$ . A + SEL . B

C D

MUX_2



# Another Example

Data flow

✓ 2x4 decoder

or

✓ 1-out-of-4 decoder

```
module decoder (A,B, D0,D1,D2,D3);
input A,B;
output D0,D1,D2,D3;
assign  D0 =(~A)&(~B);
assign  D1 = ~A&B;
assign  D2 = A&~B;
assign  D3 = A&B;
endmodule
```

$\bar{A}B$

$AB$

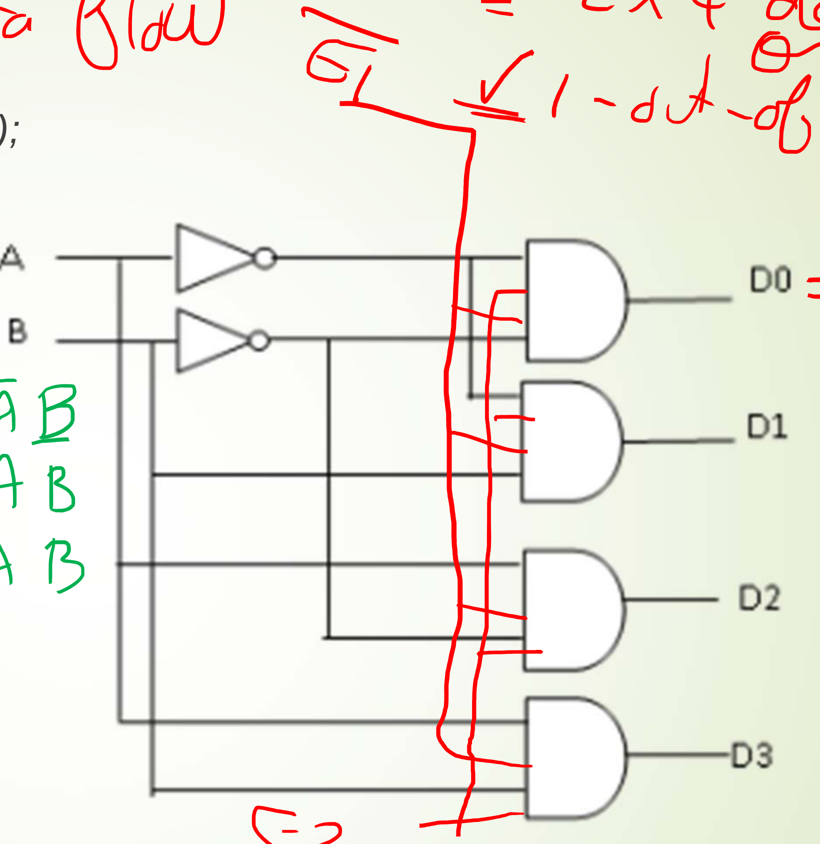
$AB$

$\overline{E_1}\,\bar{A}\bar{B}$

$\overline{E_1}\,\bar{A}B$

$\overline{E_1}\,AB$

$\overline{E_1}\,AB$



$D_0 = \bar{A}\bar{B}$

Figure 6. Logic diagram of 2-to-4 decoder

$D_0 = \sim E \,\&\, \sim A \,\&\, \sim B$

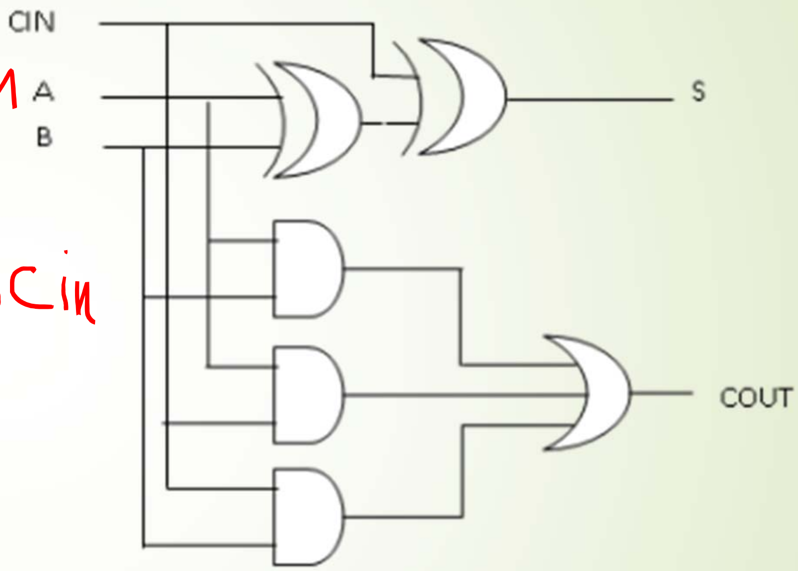
$D_1 = \sim A \,\&\, B \,\&\, \sim E_1 \,\&\, E_2$

# Hierarchical representation of Adder

```
module fulladder (A,B,CIN, S,COUT);
input A,B,CIN;
output S,COUT;
assign S = A ^ B ^ CIN;
assign COUT = (A & B) |(A & CIN) |
(B & CIN);
endmodule
```

$A \oplus B \oplus C_{in}$

$AB + AC_{in} + BC_{in}$



Logic diagram of fulladder

```
module four_bit_adder (CIN, X3,X2,X1,X0, Y3,Y2,Y1,Y0,
S3,S2,S1,S0,COUT); , sign, V) ;

input   CIN, X3, X2, X1, X0, Y3, Y2, Y1, Y0; , ~~sign~~ ;

output      S3, S2, S1, S0, COUT; , sign, V;

wire     C1, C2, C3;

fulladder FA0 (X0, Y0, CIN, S0, C1);      optional

fulladder FA1 (X1, Y1, C1, S1, C2);

fulladder FA2 (X2, Y2, C2, S2, C3);

fulladder FA3 (X3, Y3, C3, S3, COUT);
assign  sign=S3;
endmodule
assign V=C3 ~~and~~ Cout
```

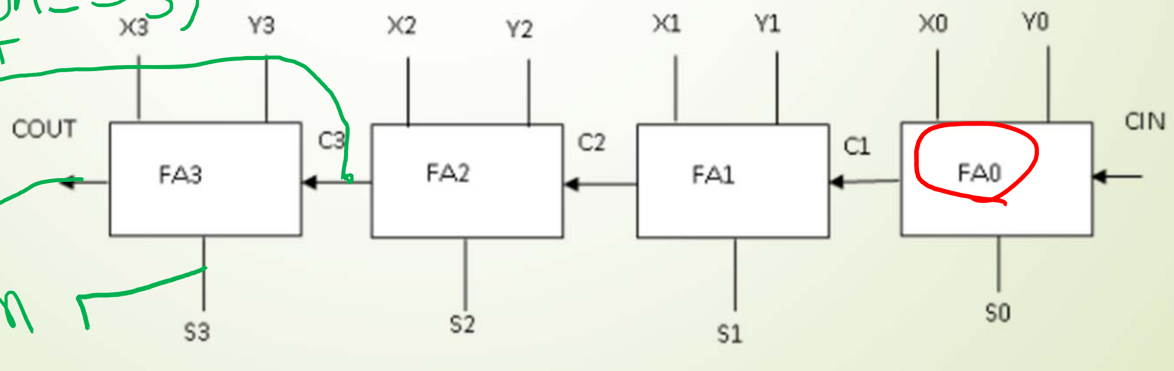


Figure 9. Four bit Full Adder

```
module adder_4 (A, B, CIN, S ,COUT);
input [3:0] A,B;
input CIN;
output [3:0] S;
output COUT;
wire  [4:0] C;
full _adder  FA0 (B(0), A(0), C(0), S(0), C(1));
full _adder  FA1 (B(1), A(1), C(1), S(1), C(2));
full _adder  FA2 (B(2), A(2), C(2), S(2), C(3));
full _adder  FA3 (B(3), A(3), C(3), S(3), C(4));
assign C(0) = CIN;
assign COUT = C(4);
endmodule
```

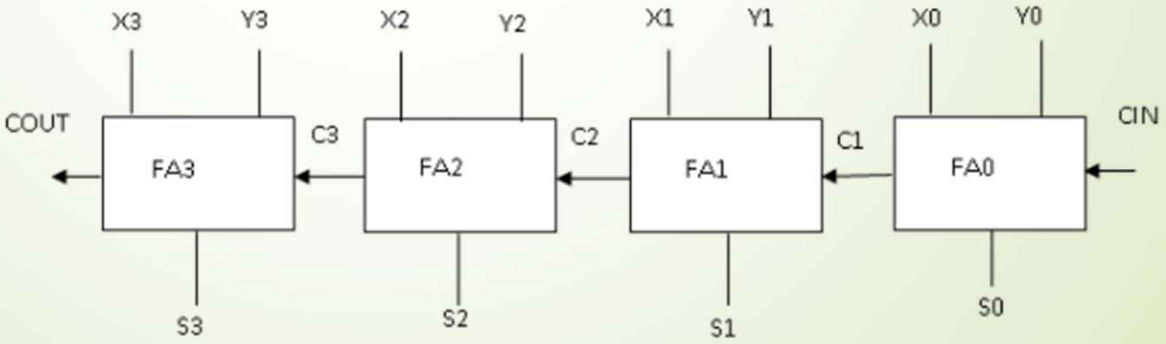


**Figure 9. Four bit Full Adder**

# Verilog Statements

Verilog has two basic types of statements

1. Concurrent statements (combinational)

   (things are happening concurrently, ordering does not matter)

   - Gate instantiations
     - **and** (z, x, y), **or** (c, a, b), **xor** (S, x, y), etc.
   - Continuous assignments
     - **assign** Z = x & y; c = a | b; S = x ^ y

2. Procedural statements (sequential)

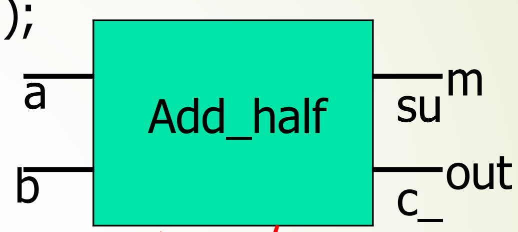
   (executed in the order written in the code)

   - **always @** - executed continuously when the event is active
   - **Initial** - executed only once (used in simulation)
   - **if then else** statements

# Behavioral Description

```
module Add_half ( sum, c_out, a, b );
   input       a, b;
   output      sum, c_out;
   reg sum, c_out;
   always @ ( a or b )
      begin
         sum = a ^ b;          // Exclusive or
         c_out = a & b;        // And
      end
endmodule
```

sensetivity list



Must be of the ‘reg’ type

Procedure assignment statements

Event control expression or sensitivity list

# Conditional Statement

- Conditional_expression ? true_expression : false expression;

**Example:**

- Assign A = (B<C) ? (D+5) : (D+2);
  - if B is less than C, the value of A will be D + 5, or else A will have the value D + 2.

- An **if-else** statement is a procedural statement.

```
//Behavioral specification
module mux2to1  (w0, w1, s, F);
input wo,w1,s;
output F;
reg F;
```

sensitivity list

```
always @ (w0,w1,s)
if (s==1) F = w1;
else F = w0;
endmodule
```

```
 always @ (w0,w1,s)
F = s ? w1: w0;
endmodule
```

| s | F |
|---|---|
| 0 | w0 |
| 1 | w1 |

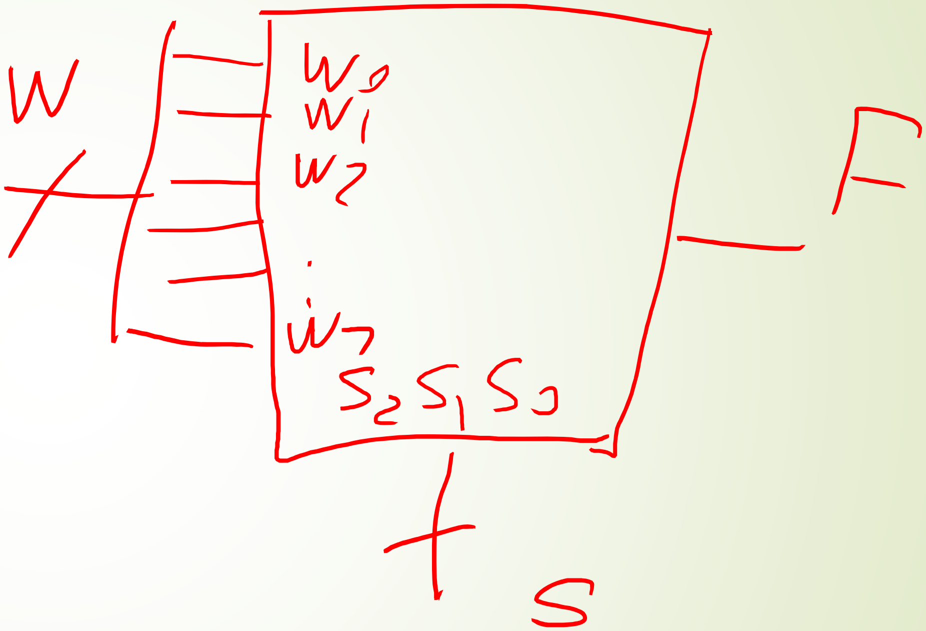
0

assign F = ~S(2)&~S(1) &~S(0)
&w0 |
----

## Problem 1

Using Verilog, design an 8 × 1 Mux.

```
//Behavioral specification
module mux8to1  (W, s, F);
Input [7:0] w
Input [2:0] s;
output F;
reg F;

 always @ (w,s)
if (s==0) F = w(0);
Elseif s== 1 F = w(1);
Elseif s== 2 F = w(2);
Elseif s== 3 F = w(3);
Elseif s== 4 F = w(4);
Elseif s== 5 F = w(5);
Elseif s== 6 F = w(6);
Else F = w(7);

endmodule
```


W
w0
w1
w2
w7
S2 S1 S0
F
S