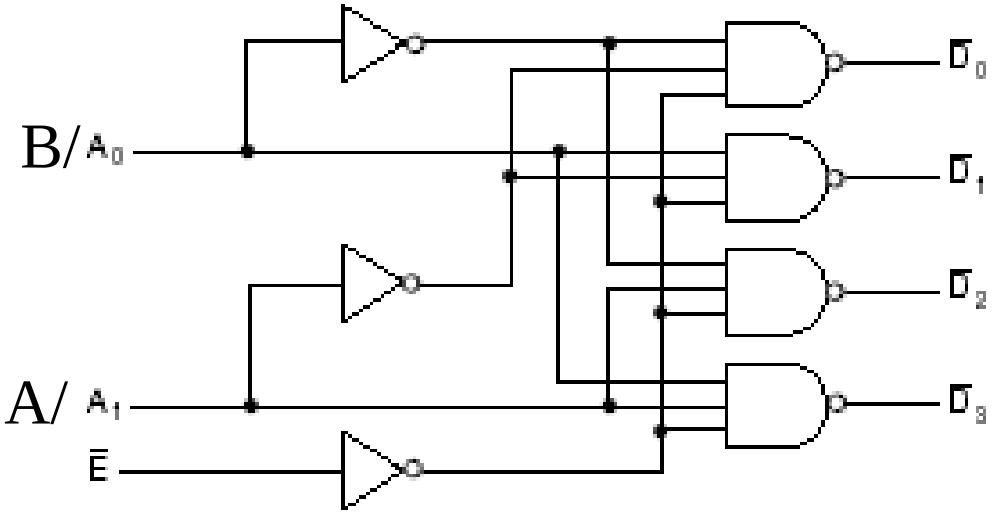
Back to HDL…..

# 2-to-4 Line Decoder




(a) Logic diagram

| Ē | $A_1$ | $A_0$ | $\bar{D}_0$ | $\bar{D}_1$ | $\bar{D}_2$ | $\bar{D}_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | X | X | 1 | 1 | 1 | 1 |

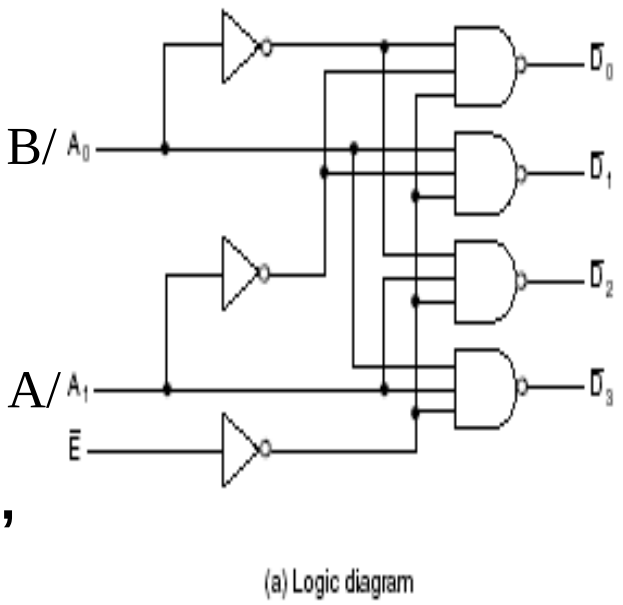
(b) Truth table

$$\bar{D}_0 = \overline{\bar{E}\,\bar{A}_1\,\bar{A}_0}$$
$$\bar{D}_1 = \overline{\bar{E}\,\bar{A}_1\,A_0}$$
$$\bar{D}_2 = \overline{\bar{E}\,A_1\,\bar{A}_0}$$
$$\bar{D}_3 = \overline{\bar{E}\,A_1\,A_0}$$

(c) Logic Equations

Fig.3-14 A 2–to–4-Line Decoder

```
//Gate - level description of a 2-to-4 line decoder
module decoder_g1 ( A,B,E,D);
    input A, B, E;
    output [0:3]D;
    wire Anot, Bnot, Enot;
    not
        n1(Anot, A),
        n2(Bnot, B),
        n3( Enot, E);
    nand
      n4 (D[0], Anot, Bnot, Enot),
      n5 (D[1], Anot, B, Enot),
      n6 (D[2], A, Bnot, Enot),
      n7 (D[3], A, B, Enot);
endmodule
```




(a) Logic diagram

| Ē | $A_1$ | $A_0$ | $\bar{D}_0$ | $\bar{D}_1$ | $\bar{D}_2$ | $\bar{D}_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | X | X | 1 | 1 | 1 | 1 |

(b) Truth table

$\bar{D}_0 = \overline{\bar{E}\,\bar{A}_1\,\bar{A}_0}$

$\bar{D}_1 = \overline{\bar{E}\,\bar{A}_1\,A_0}$

$\bar{D}_2 = \overline{\bar{E}\,A_1\,\bar{A}_0}$

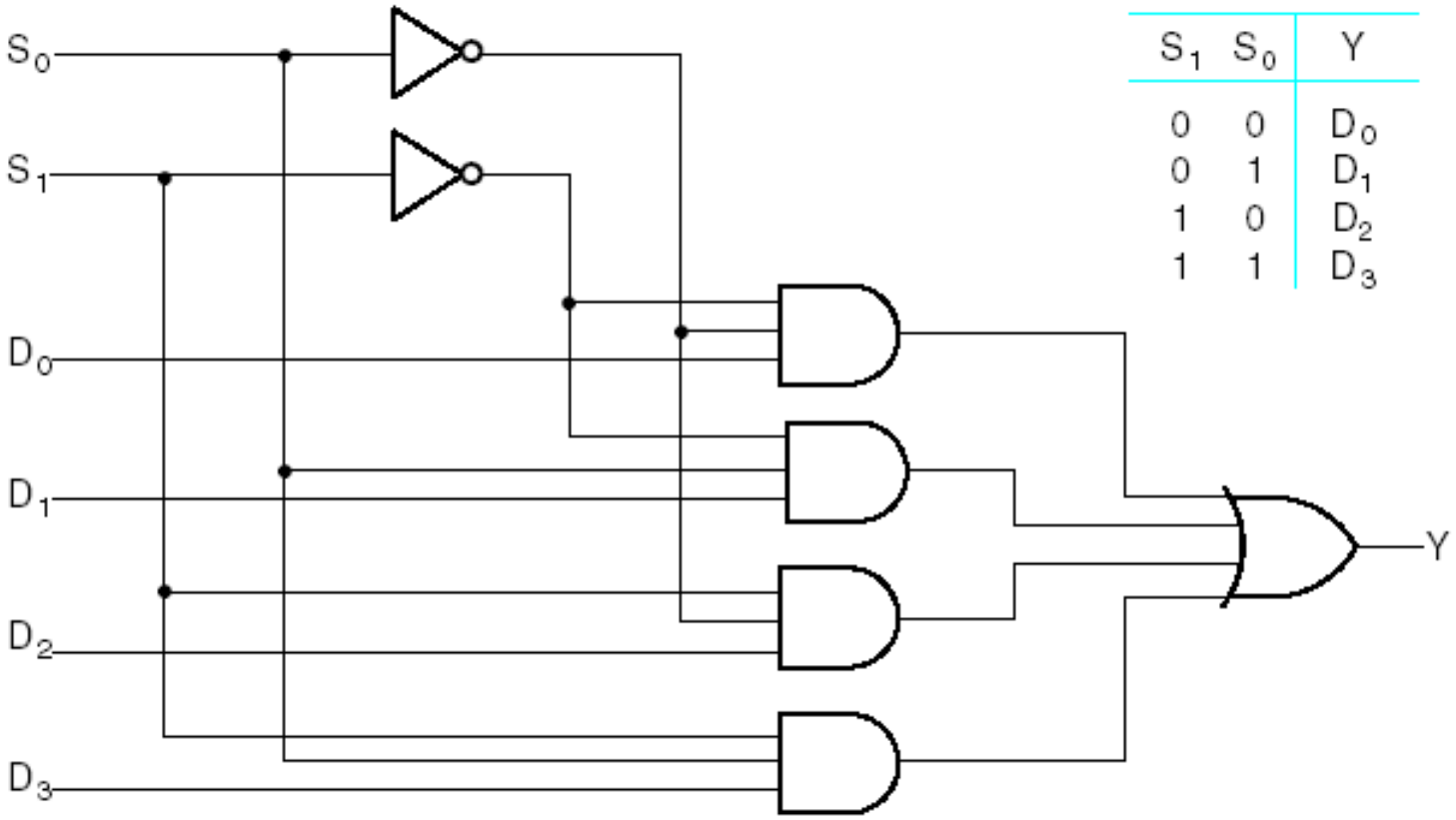
$\bar{D}_3 = \overline{\bar{E}\,A_1\,A_0}$

(c) Logic Equations

Fig.3-14 A 2–to–4-Line Decoder

```
module stimckt;
reg A,B,E;
wire [0:3]D;
  decoder_g1 dec (A, B, E, D);
  initial
    begin
    A = 1'b0; B = 1'b0; E = 1'b0;
    #100
    A = 1'b0; B = 1'b1; E = 1'b0;
    #100 $finish;
    end
endmodule
```

# 4-to-1 Multiplexer




Function table

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $D_0$ |
| 0 | 1 | $D_1$ |
| 1 | 0 | $D_2$ |
| 1 | 1 | $D_3$ |

Fig. 3-19 4-to-1-Line Multiplexer

# 4-to-1 Multiplexer

```
// 4-to-1 Line Multiplexer: Structural Verilog Description     //  1
// (See Figure 3-19 for logic diagram)                         //  2
module multiplexer_4_to_1_st_v(S, D, Y);                       //  3
   input [1:0] S;                                              //  4
   input [3:0] D;                                              //  5
   output Y;                                                   //  6

   wire [1:0] not_S;
   wire [0:3] N;
```

Function table

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $D_0$ |
| 0 | 1 | $D_1$ |
| 1 | 0 | $D_2$ |
| 1 | 1 | $D_3$ |

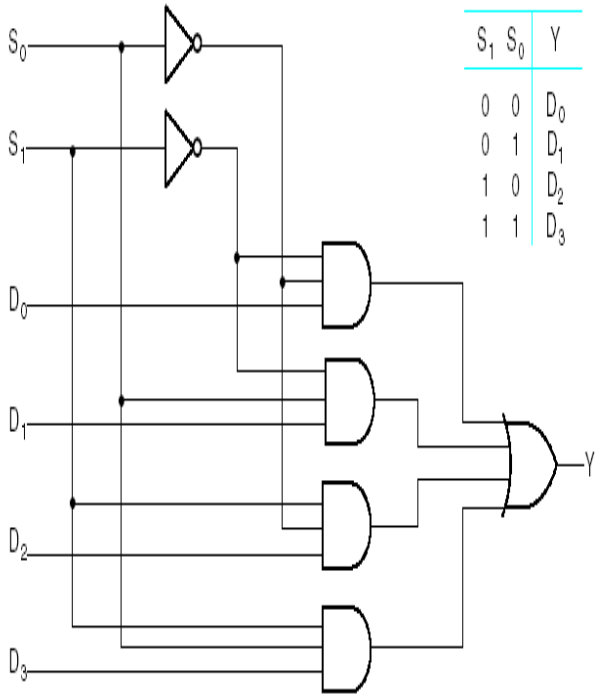



Fig. 3-19 4-to-1-Line Multiplexer

# 4-to-1 Multiplexer

```
// 4-to-1 Line Multiplexer: Structural Verilog Description     //  1
// (See Figure 3-19 for logic diagram)                         //  2
module multiplexer_4_to_1_st_v(S, D, Y);                       //  3
   input [1:0] S;                                              //  4
   input [3:0] D;                                              //  5
   output Y;                                                   //  6
                                                               //  7
   wire [1:0] not_S;                                           //  8
   wire [0:3] N;

not
    gn0(not_S[0], S[0]),
    gn1(not_S[1], S[1]);
```

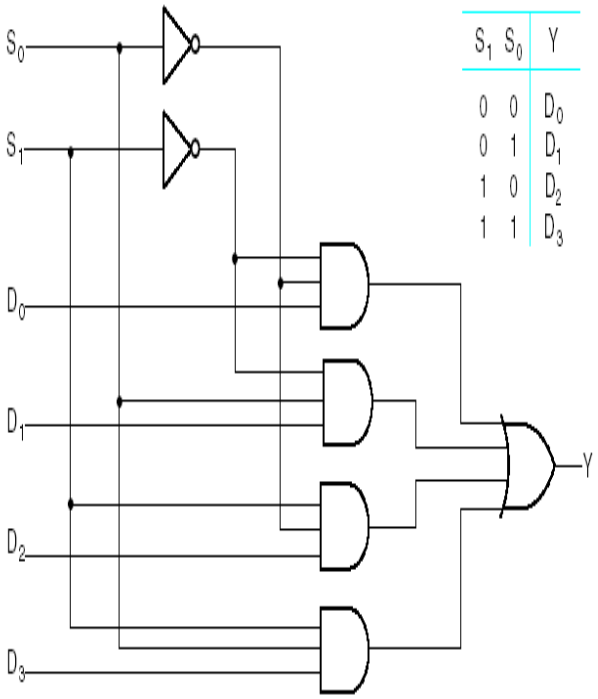



Fig. 3-19 4-to-1-Line Multiplexer

# 4-to-1 Multiplexer

```
// 4-to-1 Line Multiplexer: Structural Verilog Description     //  1
// (See Figure 3-19 for logic diagram)                         //  2
module multiplexer_4_to_1_st_v(S, D, Y);                       //  3
   input [1:0] S;                                              //  4
   input [3:0] D;                                              //  5
   output Y;                                                   //  6

   wire [1:0] not_S;
   wire [0:3] N;

not
    gn0(not_S[0], S[0]),
    gn1(not_S[1], S[1]);

and
   g0(N[0], not_S[1], not_S[0], D[0]),
   g1(N[1], not_S[1], S[0], D[1]),
   g2(N[2], S[1], not_S[0], D[2]),
   g3(N[3], S[1], S[0], D[3]);

or go(Y, N[0], N[1], N[2], N[3]);

endmodule
```

Function table

| $S_1$ | $S_0$ | Y |
|---|---|---|
| 0 | 0 | $D_0$ |
| 0 | 1 | $D_1$ |
| 1 | 0 | $D_2$ |
| 1 | 1 | $D_3$ |

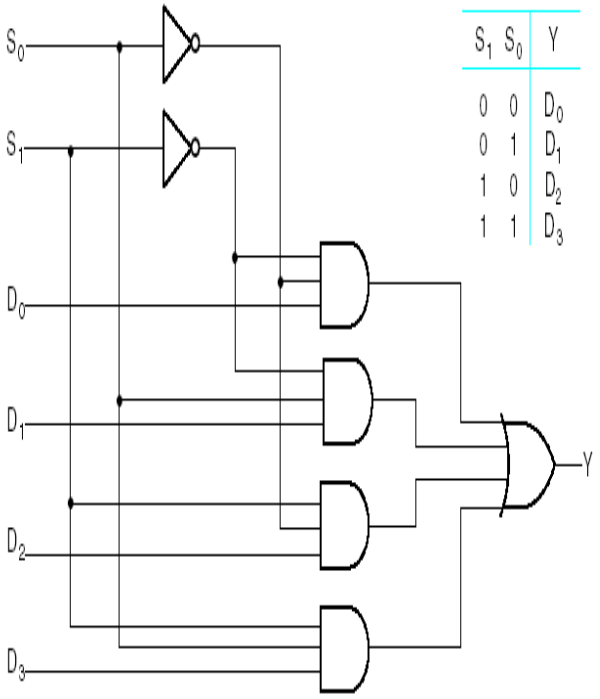



Fig. 3-19 4-to-1-Line Multiplexer

# DATA FLOW MODELLING

# Dataflow modelling

- Another level of abstraction is to model dataflow.
- Dataflow modeling uses a number of operators that act on operands to produce desired results.
- Verilog HDL provides about 30 operator types.
- In dataflow models, signals are continuously assigned values using the **`assign`** keyword.

# Dataflow Modeling

- For ex: assign Y = (A &B) | (C &D)
- **`assign`** can be used with Boolean expressions.
  - **Verilog uses `&` (and), `|` (or), `^` (xor) and `~` (not)**
- Logic expressions and binary arithmetic are also possible.

# Simple Circuit Boolean Expression

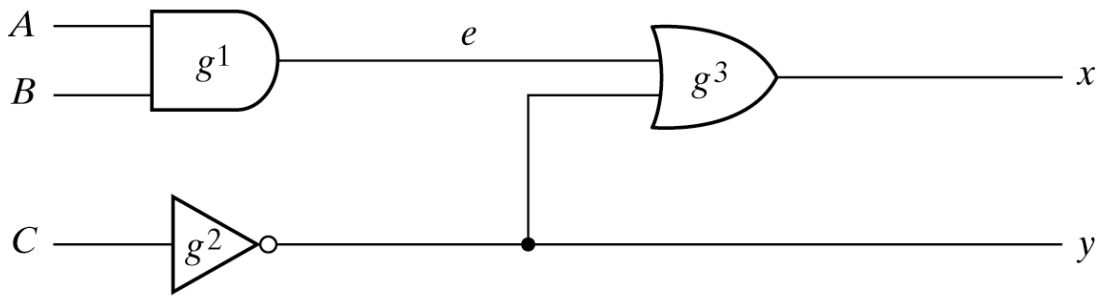



Fig. 3-37 Circuit to Demonstrate HDL

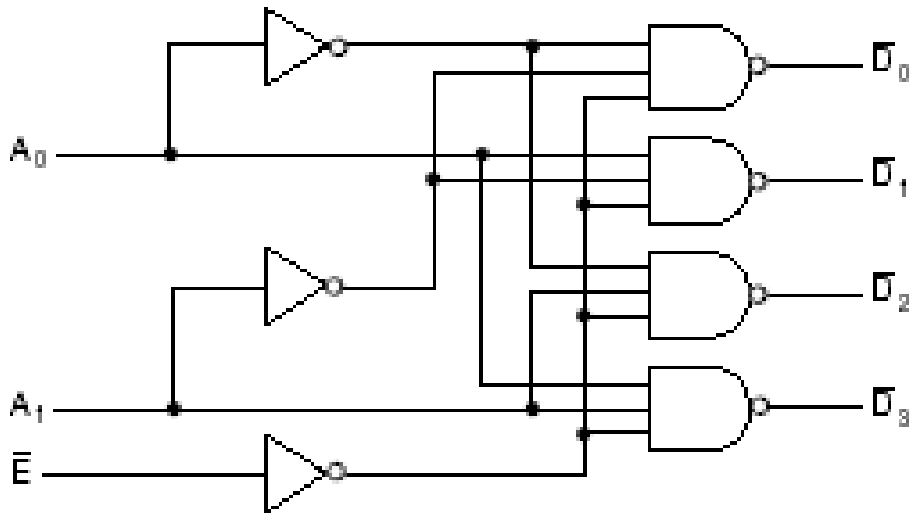
$$x = A.B + \overline{C}$$

$$y = \overline{C}$$

# Boolean Expressions

```
//Circuit specified with Boolean
equations
module circuit_bln (A,B,C,x,y);
   input A,B,C;
   output x,y;
   assign x = (A & B)|(~C);
   assign y = ~C ;
endmodule
```

# 2-to-4 Line Decoder




(a) Logic diagram

| $E$ | $A_1$ | $A_0$ | $\overline{D}_0$ | $\overline{D}_1$ | $\overline{D}_2$ | $\overline{D}_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | X | X | 1 | 1 | 1 | 1 |

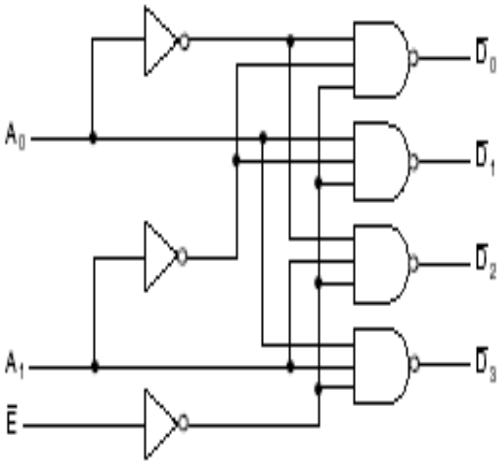
(b) Truth table

$$\overline{D}_0 = \overline{E\ \overline{A}_1\ \overline{A}_0}$$
$$\overline{D}_1 = \overline{E\ \overline{A}_1\ A_0}$$
$$\overline{D}_2 = \overline{E\ A_1\ \overline{A}_0}$$
$$\overline{D}_3 = \overline{E\ A_1\ A_0}$$

(c) Logic Equations

# Dataflow Modeling

```
//Dataflow description of a 2-to-4-line decoder
module decoder_df (A,B,E,D);
   input A,B,E;
   output [0:3] D;
   assign D[0] = ~(~A & ~B & ~E),
          D[1] = ~(~A & B & ~E),
          D[2] = ~(A & ~B & ~E),
          D[3] = ~(A & B & ~E);
endmodule
```




(a) Logic diagram

| $\bar{E}$ | $A_1$ | $A_0$ | $\bar{D}_0$ | $\bar{D}_1$ | $\bar{D}_2$ | $\bar{D}_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | X | X | 1 | 1 | 1 | 1 |

(b) Truth table

$$\bar{D}_0 = \overline{E\,\bar{A}_1\,\bar{A}_0}$$
$$\bar{D}_1 = \overline{E\,\bar{A}_1\,A_0}$$
$$\bar{D}_2 = \overline{E\,A_1\,\bar{A}_0}$$
$$\bar{D}_3 = \overline{E\,A_1\,A_0}$$

(c) Logic Equations

Fig.3-14 A 2–to–4-Line Decoder

# 4 bit Full Adder

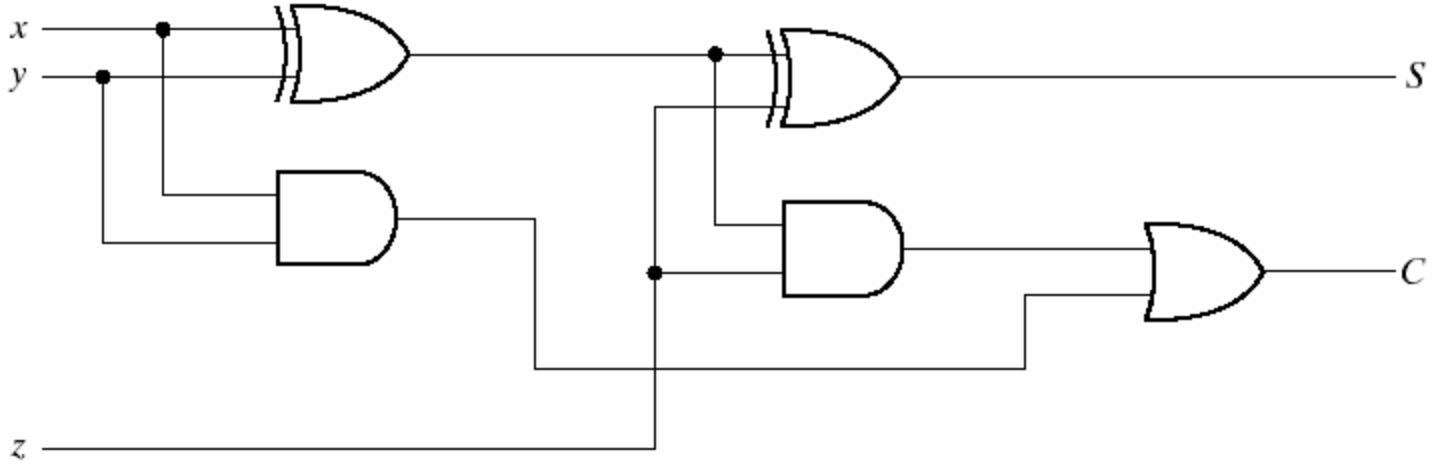



Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

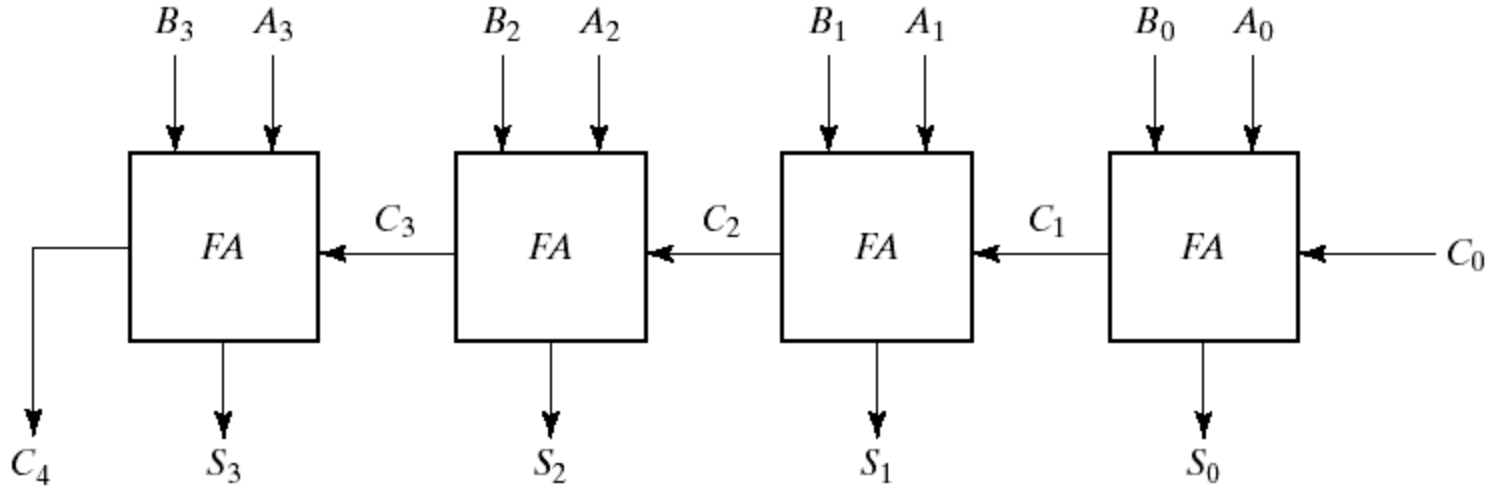



Fig. 4-9 4-Bit Adder

# Dataflow Modeling

```
//Dataflow description of 4-bit adder
module binary_adder (A,B,Cin,SUM,Cout);
   input [3:0] A,B;
   input Cin;
   output [3:0] SUM;
   output Cout;
   assign {Cout,SUM} = A + B + Cin;
endmodule
```

# Dataflow Modeling

- The addition logic of 4 bit adder is described by a single statement using the operators of addition and concatenation.
- The plus symbol (+) specifies the binary addition of the 4 bits of *A* with the 4 bits of *B* and the one bit of *Cin*.
- The target output is the concatenation of the output carry *Cout* and the four bits of *SUM*.
- Concatenation of operands is expressed within braces and a comma separating the operands. Thus, {*Cout,SUM*} represents the 5-bit result of the addition operation.

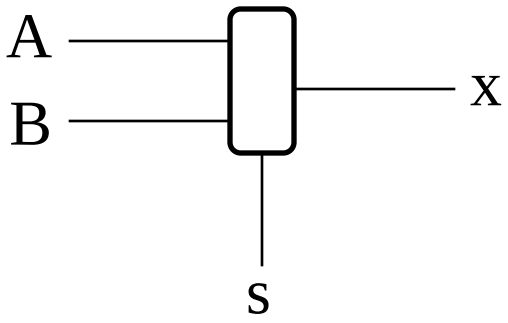
- Dataflow modelling provides means of describing combinational circuit by function rather than gate structure.
- A verilog HDL Synthesis tool can accept this module as input and can provide a netlist of a circuit equivalent.

# Dataflow Modeling

```
//Dataflow description of a 4-bit comparator.
module magcomp (A,B,ALTB,AGTB,AEQB);
   input [3:0] A,B;
   output ALTB,AGTB,AEQB;
   assign ALTB = (A < B),
          AGTB = (A > B),
          AEQB = (A == B);
endmodule
```

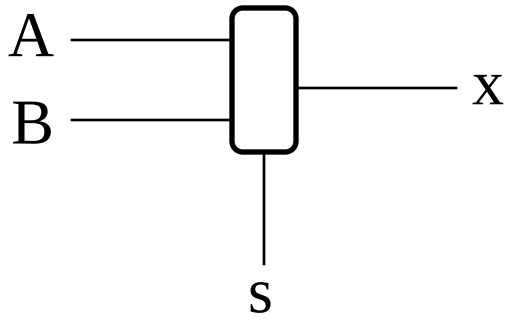
# Multiplexer

- Multiplexer is a combinational circuit where an input is chosen by a select signal.
  - Two input mux
  - output =A if select =1
  - output= B if select =0



# Two Input Multiplexor

- A two-input mux is actually a three input device.




$$x = A.s + B.\bar{s}$$

| s | A | B | x |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# Dataflow description of 2-input Mux

- Conditional operator **?:** takes three operands:

**condition? true_expression : false_expression**

```
module mux2x1_df (A,B,select,x);
    input A,B,select;
    output x;
    assign x = select ? A : B;
endmodule
```