

Ch 4. Combinational logic

4.1 Introduction

4.2 Combinational circuits

- Outputs are determined from the present inputs
- Consist of input/output variables and logic gates

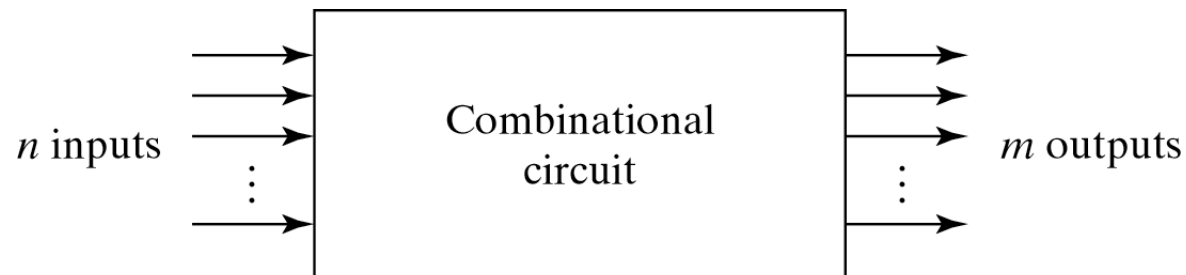


Fig. 4-1 Block Diagram of Combinational Circuit

4.3 Analysis procedure

- To determine the function of circuit
- Analysis procedure
 - Make sure the circuit is combinational or sequential
 - Obtain the output Boolean functions or the truth table

4.3 Analysis procedure

- Boolean function
 - Label all gate outputs
 - Make output functions at each level
 - Substitute final outputs to input variables
- Truth table
 - Put the input variables to binary numbers
 - Determine the output value at each gate
 - Obtain truth table

4.3 Analysis procedure

Table 4-1
Truth Table for the Logic Diagram of Fig. 4-2

A	B	C	F_2	F_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

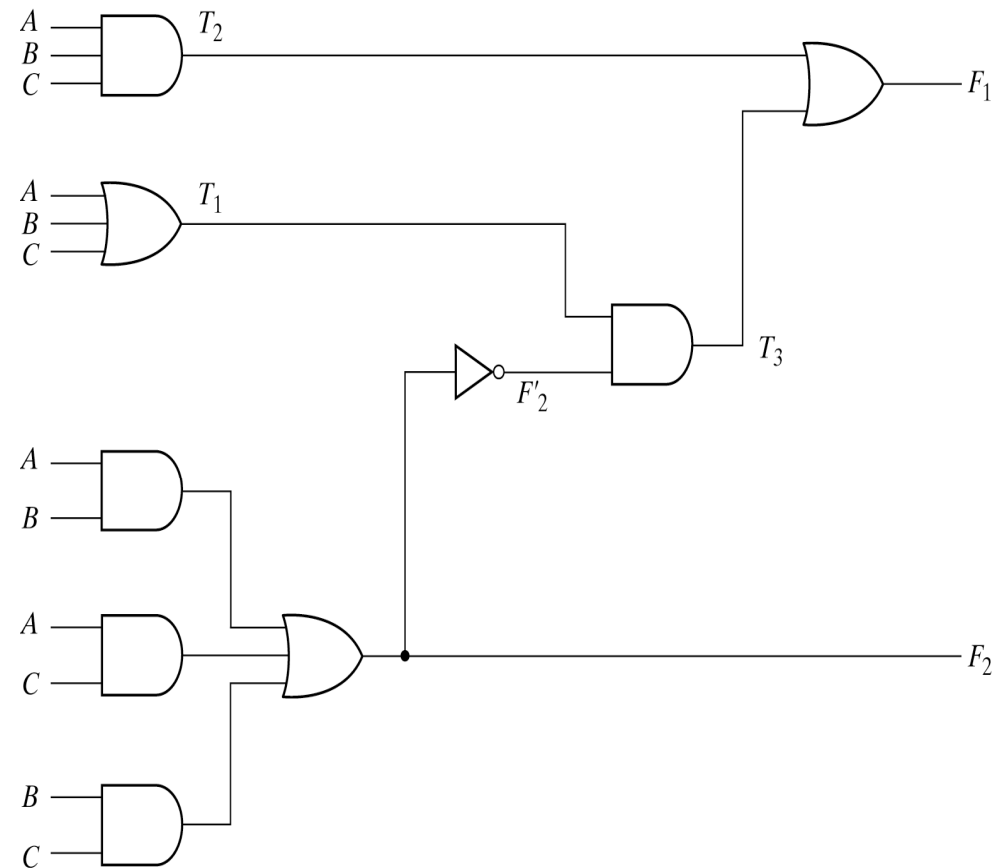


Fig. 4-2 Logic Diagram for Analysis Example

4.4 Design procedure

- Procedure to design
 - Determine the required number of input and output from specification
 - Assign a letter symbol to each input/output
 - Derive the truth table
 - Obtain the simplified Boolean functions
 - Draw the logic diagram and verify design correctness

4.4 Design procedure - Code conversion example

- BCD to excess-3 code converter
 - Excess-3 code : decimal digit+3
- Design procedure
 - 1) Determine inputs/outputs
 - Inputs : A,B,C,D (0000~1001)
 - Outputs : W,X,Y,Z (0011~1100)

4.4 Design procedure - Code conversion example

2) Derive truth table

Table 4-2
Truth Table for Code-Conversion Example

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

4.4 Design procedure - Code conversion example

3) Obtain simplified Boolean functions

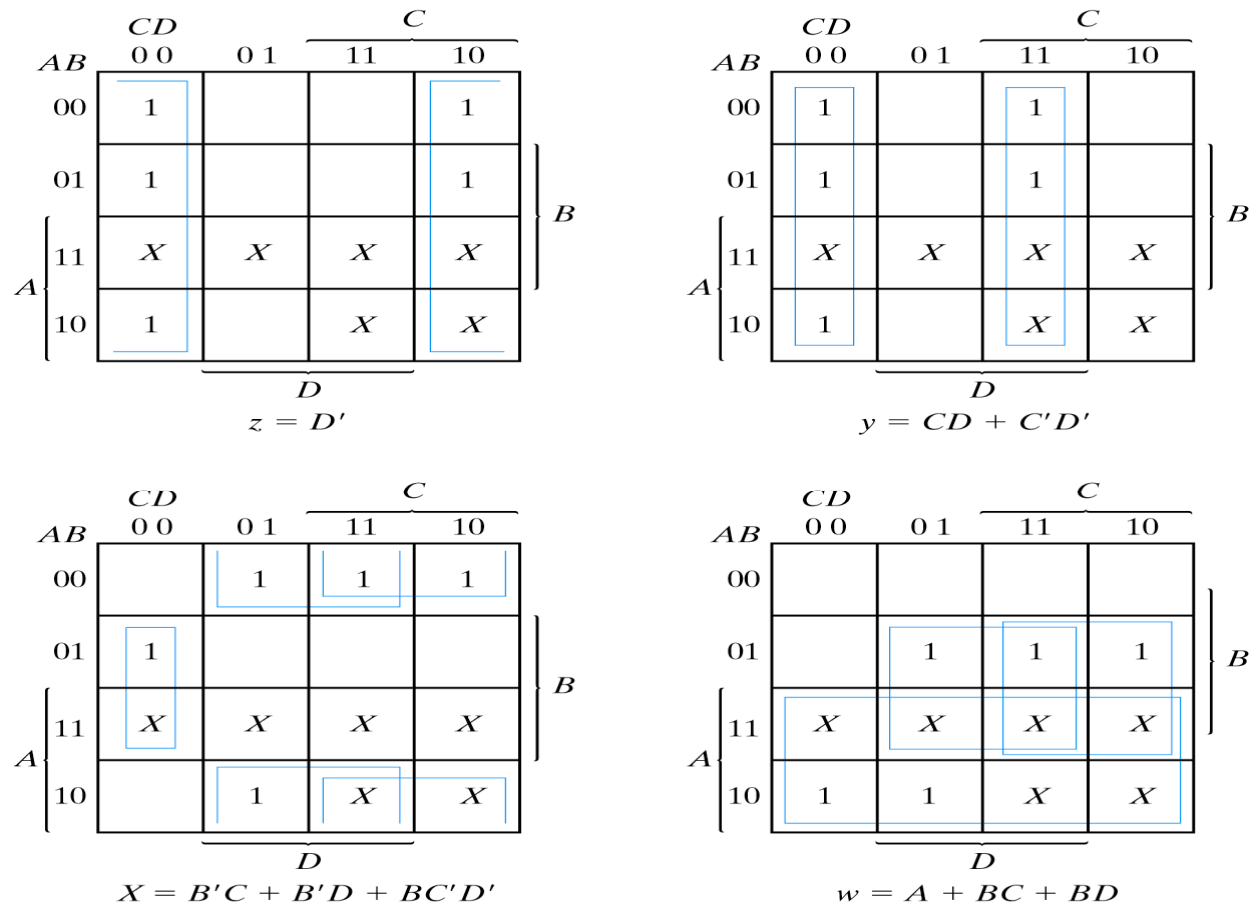


Fig. 4-3 Maps for BCD to Excess-3 Code Converter

4.4 Design procedure - Code conversion example

4) Draw the logic diagram

$$z = D'$$

$$y = CD + C'D' = CD + (C + D)'$$

$$\begin{aligned} x &= B'C + B'D + BC'D' = B'(C + D) + BC'D' \\ &= B'(C + D) + B(C + D)' \end{aligned}$$

$$w = A + BC + BD = A + B(C + D)$$

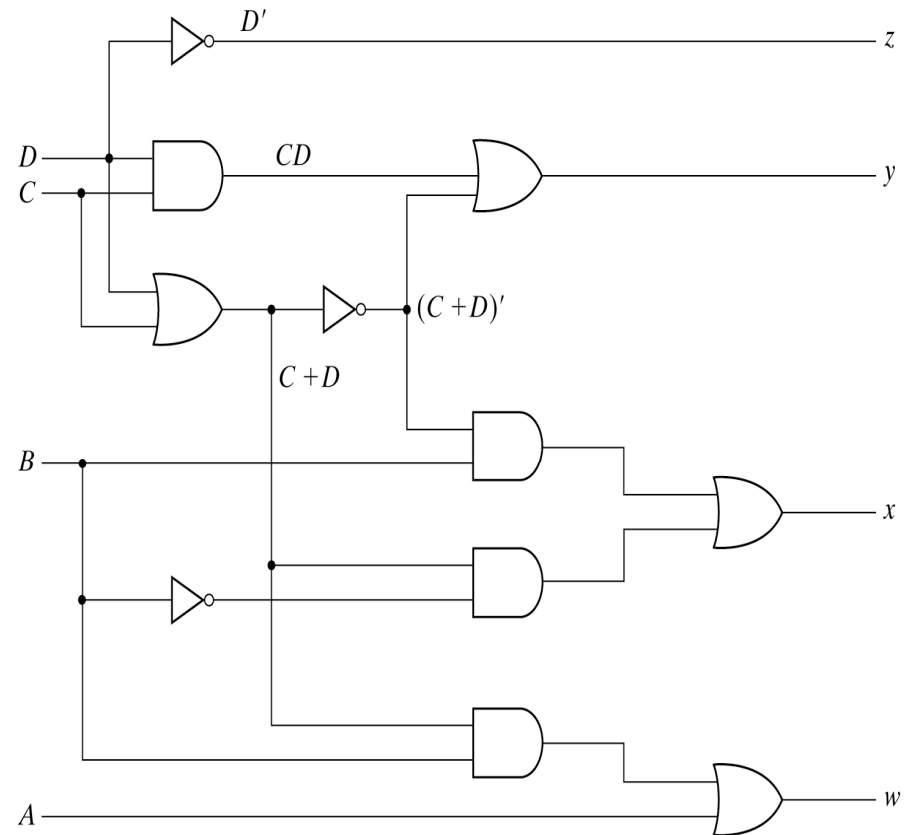


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter

4.5 Binary adder-subtractor

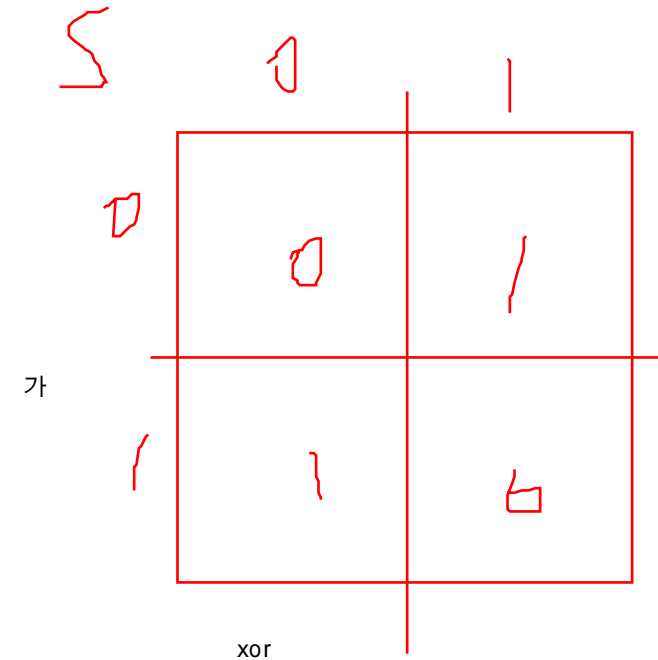
- Binary adder
 - Half adder : performs the addition of 2-bits($x+y$)
 - Full adder : performs the addition of 3-bits($x+y+z$)
 - Two half adder can be employed to a full adder
- Realization of Binary adder-subtractor
 - Half adder
 - Full adder
 - Cascade of n -full adder
 - Providing a complementing circuit

4.5 Binary adder-subtractor - Half Adder

- Sum of 2 binary inputs
- Input : X(augend), Y(addend)
Output : S(sum), C(carry)

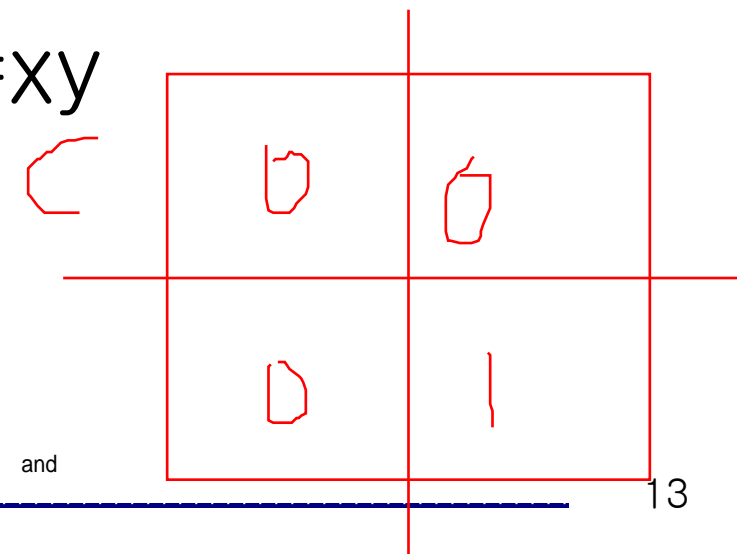
Table 4-3
Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

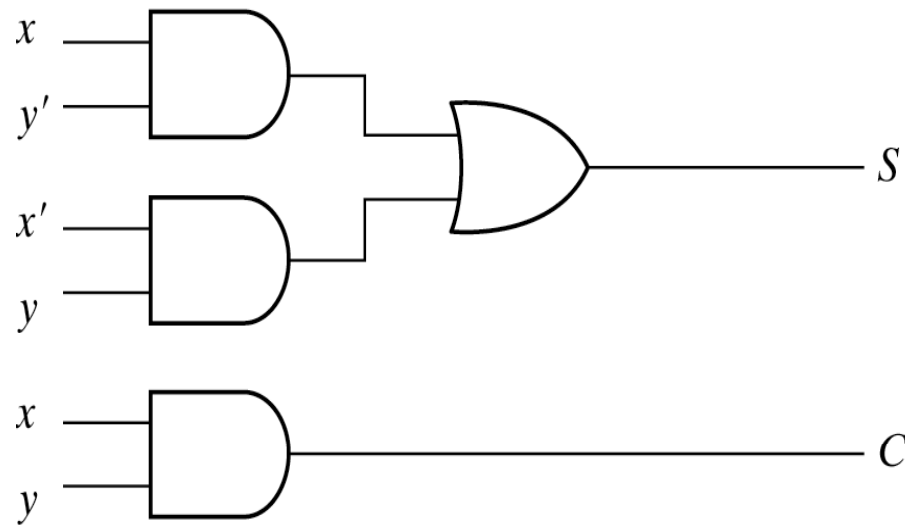


$$S = xy' + x'y$$

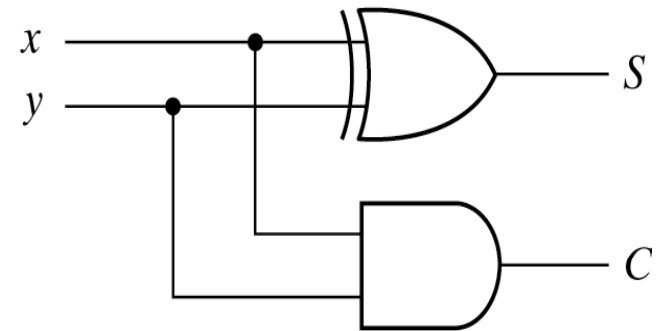
$$C = xy$$



4.5 Binary adder-subtractor - Half Adder



$$(a) \begin{aligned} S &= xy' + x'y \\ C &= xy \end{aligned}$$



$$(b) \begin{aligned} S &= x \oplus y \\ C &= xy \end{aligned}$$

Fig. 4-5 Implementation of Half-Adder

4.5 Binary adder-subtractor - Full adder

- Sum of 3 binary inputs
- Input : X,Y(2 significant bits),Z(1 carry bit)
- Output : S(sum),C(carry)

Table 4-4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

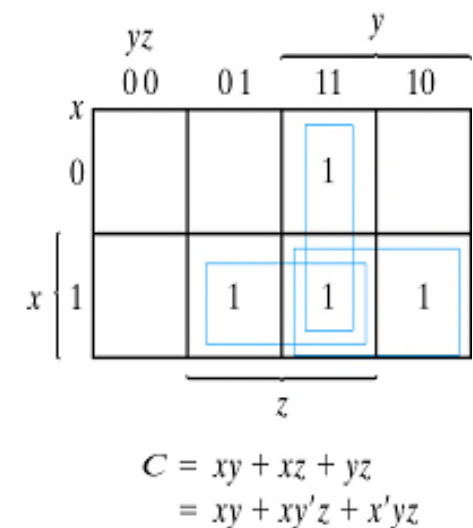
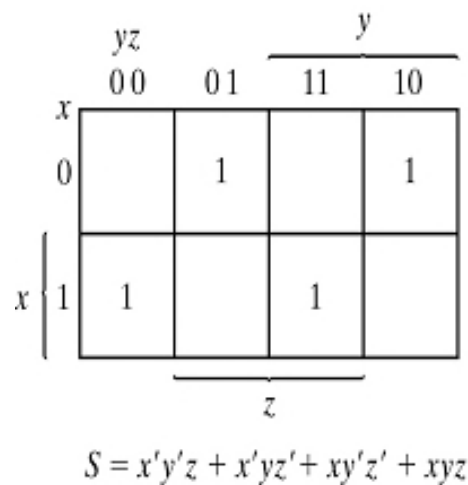


Fig. 4-6 Maps for Full Adder

4.5 Binary adder-subtractor - Full adder

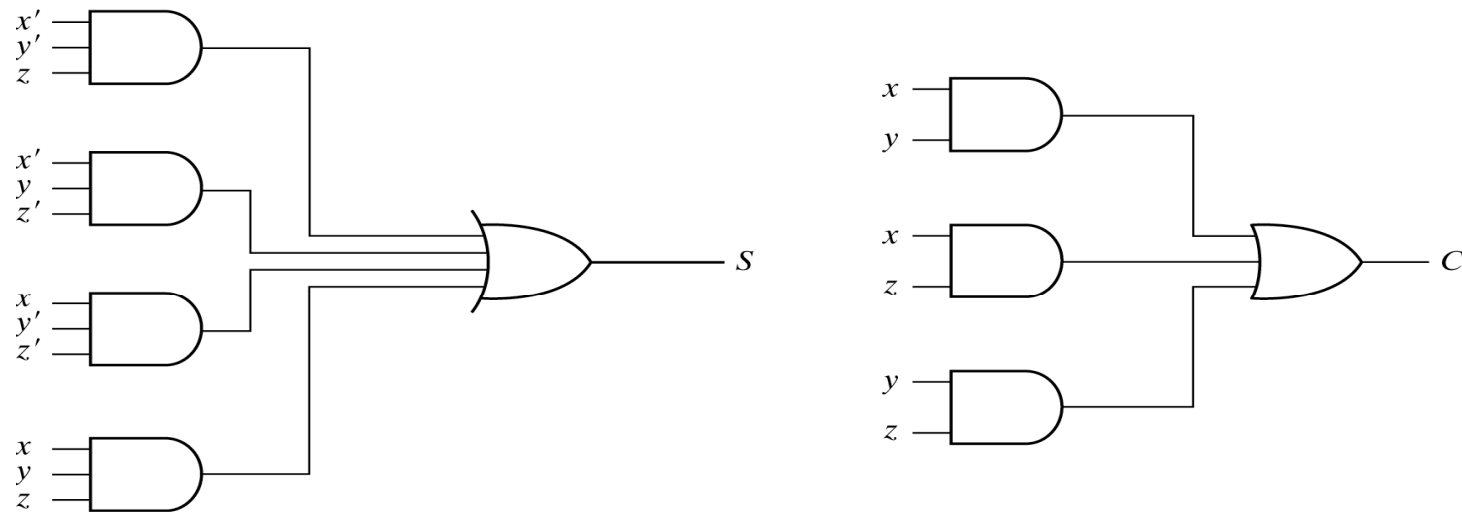


Fig. 4-7 Implementation of Full Adder in Sum of Products

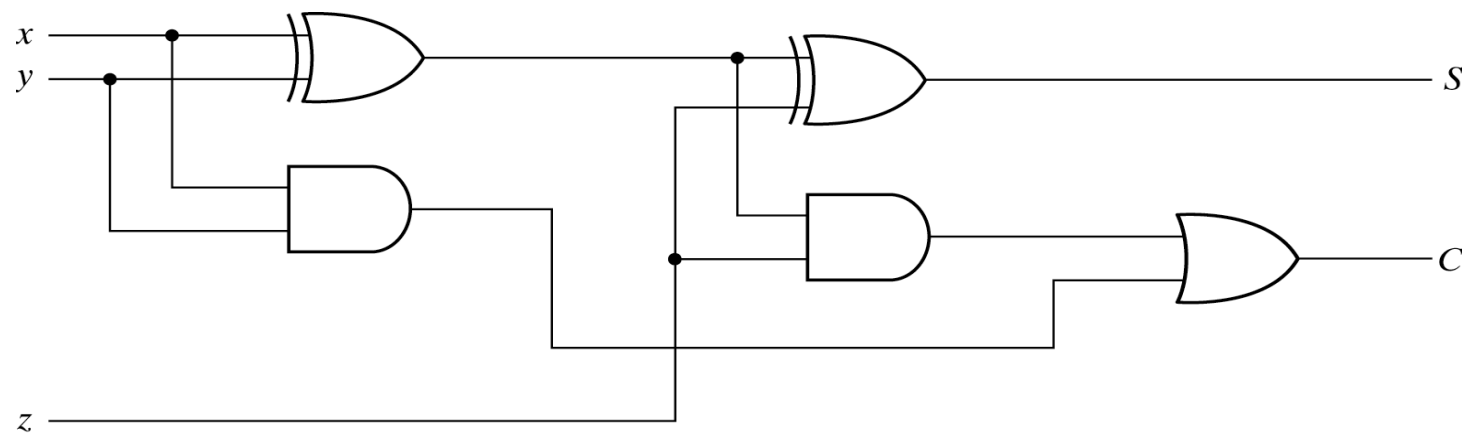


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

4.5 Binary adder-subtractor - Binary adder

● Sum of two n-bit binary numbers

– 4-bit adder

A=1011, B=0011

Subscript i:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

(delay가 , 3)

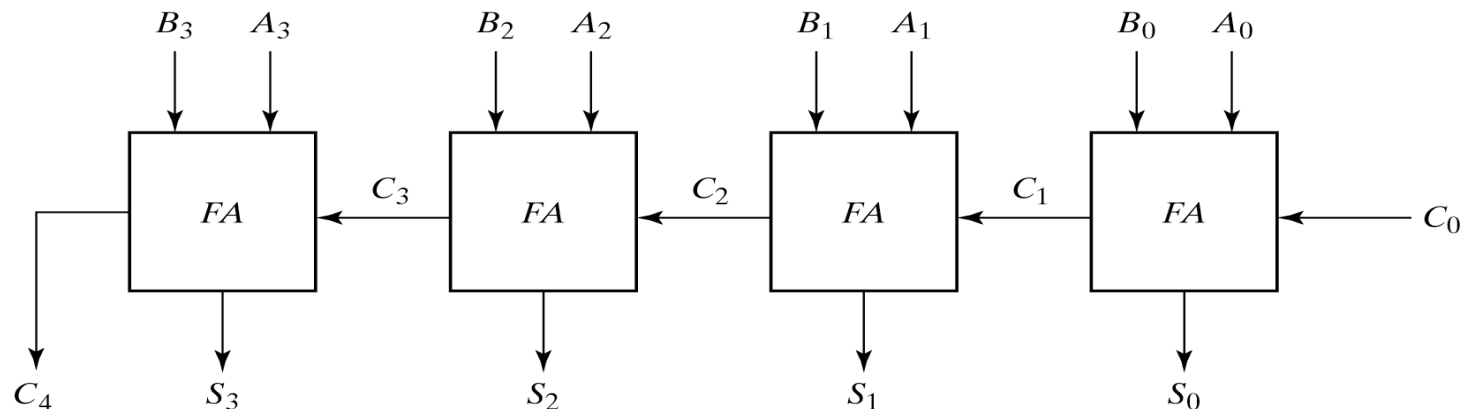


Fig. 4-9 4-Bit Adder

4.5 Binary adder-subtractor - Carry propagation

- Rising of delay time(carry delay)
- One solution is **carry lookahead**
- All carry is a function of P_i, G_i and C_0

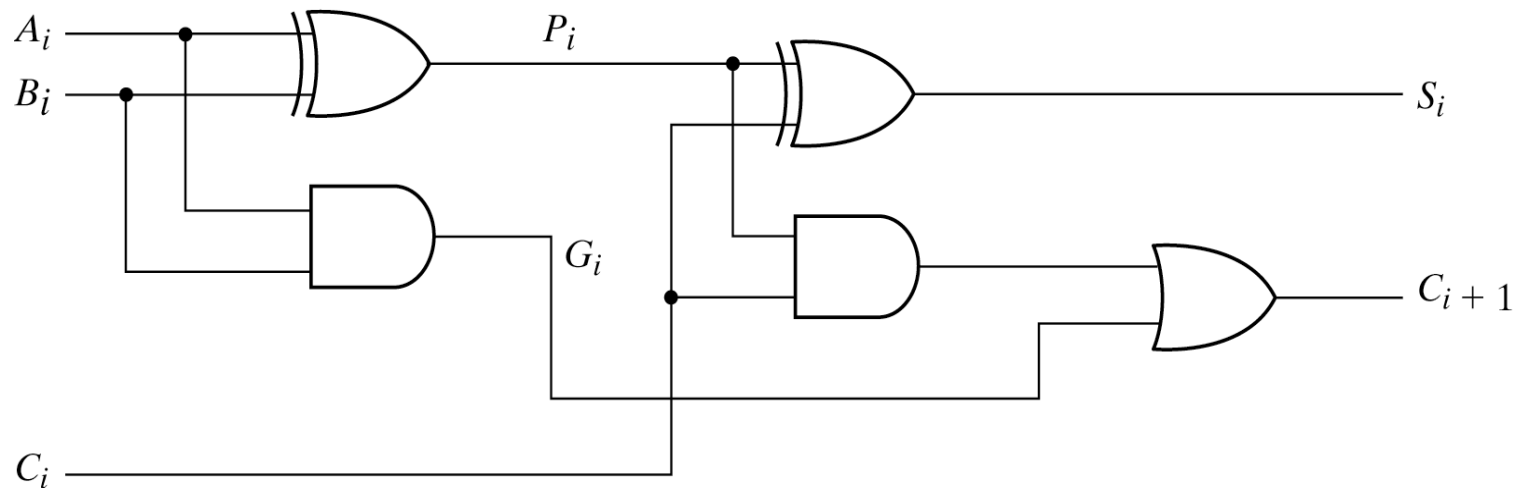


Fig. 4-10 Full Adder with P and G Shown

4.5 Binary adder-subtractor - Carry propagation

● Carry lookahead generator

C_0 = input carry

$$C_1 = G_0 + P_0C_0$$

$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

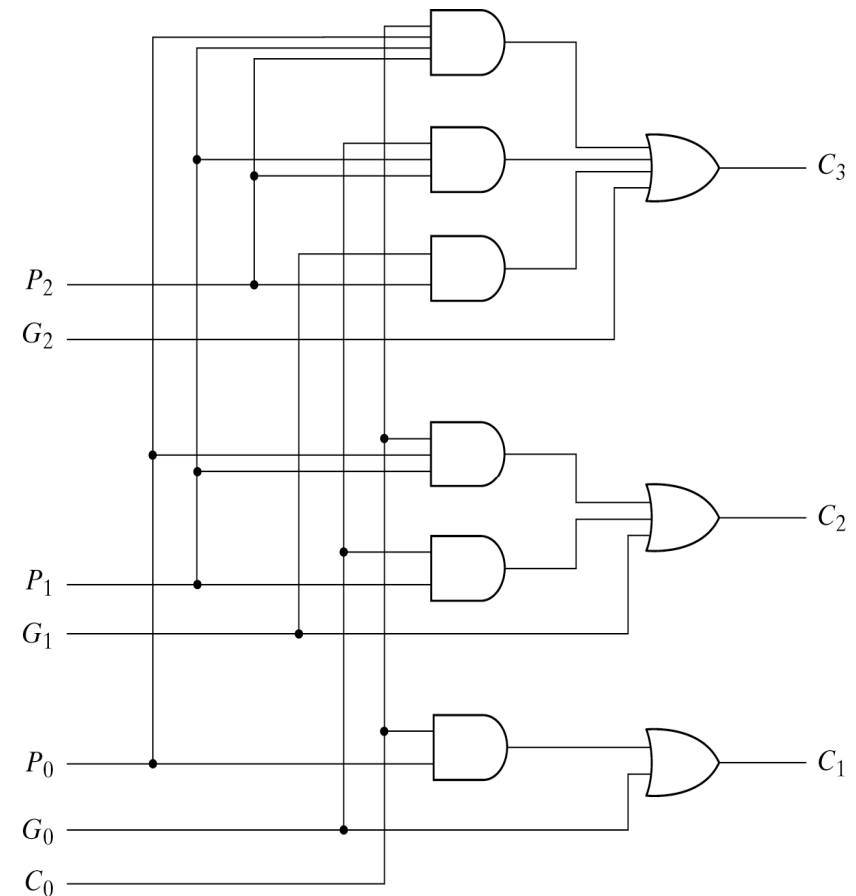


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

4.5 Binary adder-subtractor - Carry propagation

4-bit adder with carry lookahead

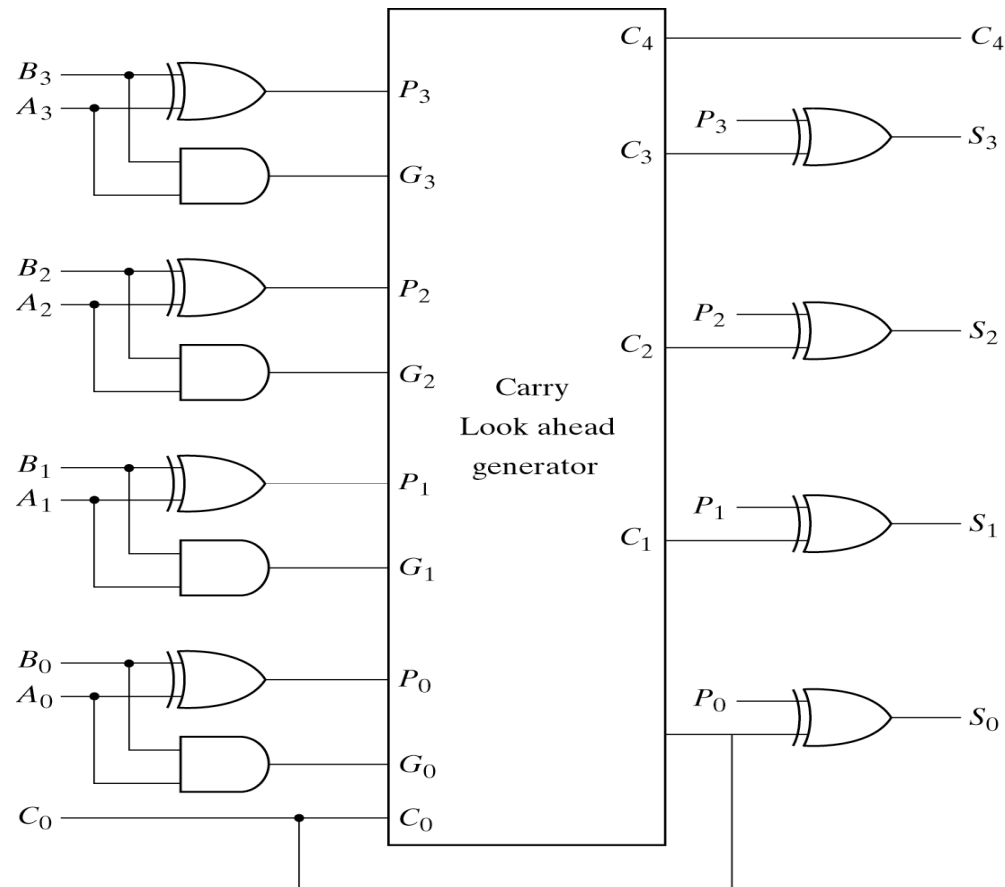


Fig. 4-12 4-Bit Adder with Carry Lookahead

4.5 Binary adder-subtractor - Overflow

- Sum of n digit number occupies $n+1$ digit
- Occurs when two numbers are same sign

(examples of overflow)

carries:	0	1
+70	0	1000110
+80	0	1010000
<hr/>		
+150	1	0010110

carries:	1	0
-70	1	0111010
-80	1	0110000
<hr/>		
-150	0	1101010

4.6 Decimal adder

- Calculate binary and represent decimal in binary coded form
- Decimal adder for the BCD code

4.6 Decimal adder - BCD Adder

- BCD digit output of 2-BCD digit sum
- Carry arise if output 1010~1111
- $C = K + Z_8 Z_4 + Z_8 Z_2$

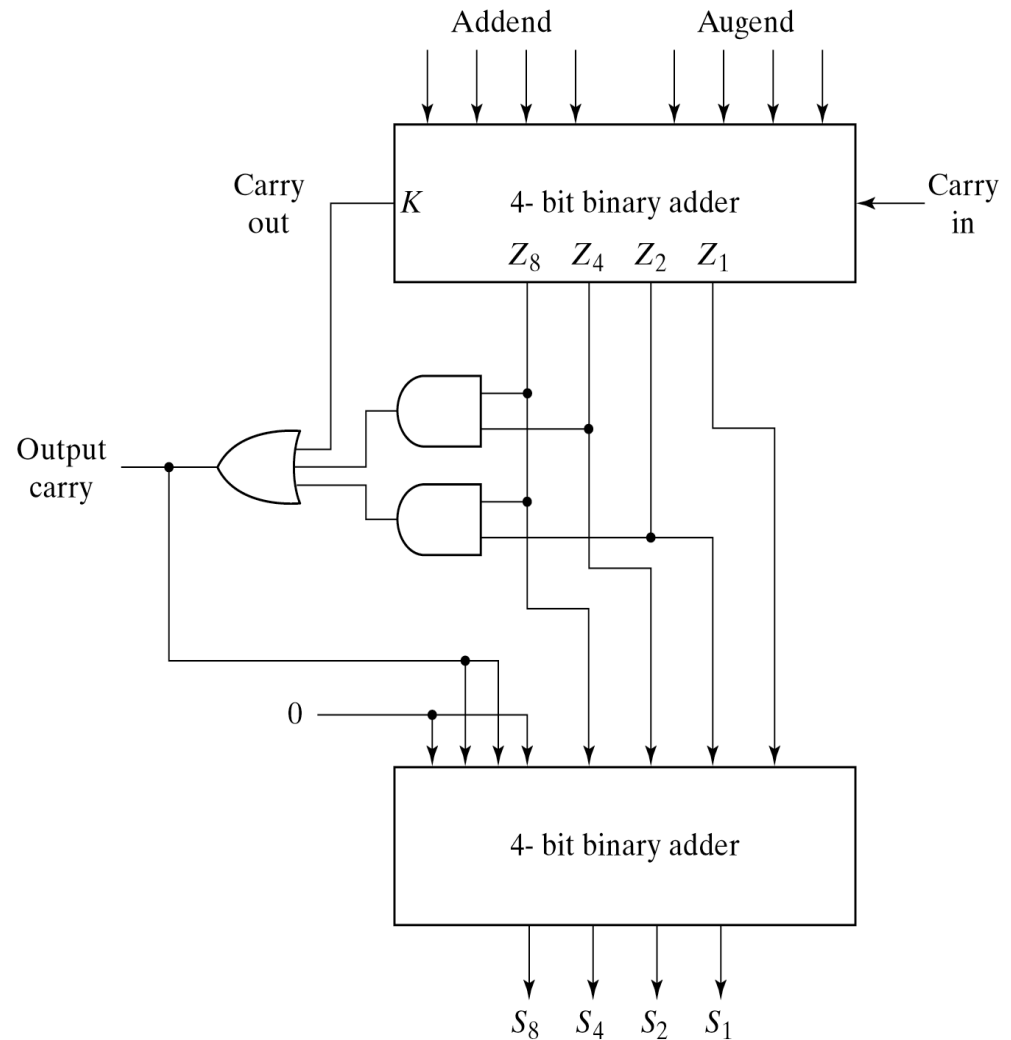
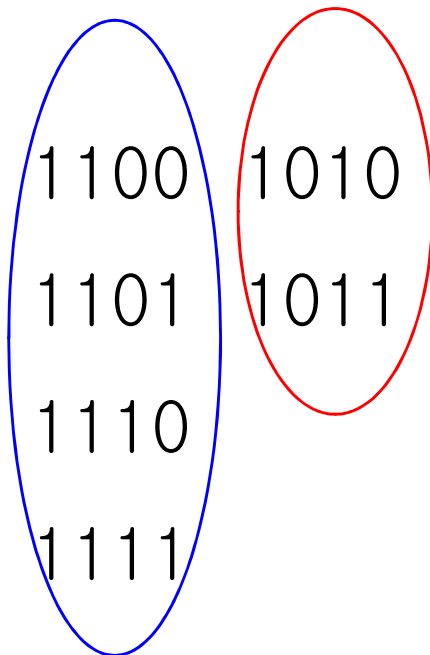


Fig. 4-14 Block Diagram of a BCD Adder

4.7 Binary multiplier

- 2bit x 2bit = 4bit(max)

		B_1	B_0
	A_1	$A_1 B_1$	$A_1 B_0$
	A_0	$A_0 B_1$	$A_0 B_0$
C_3	C_2	C_1	C_0

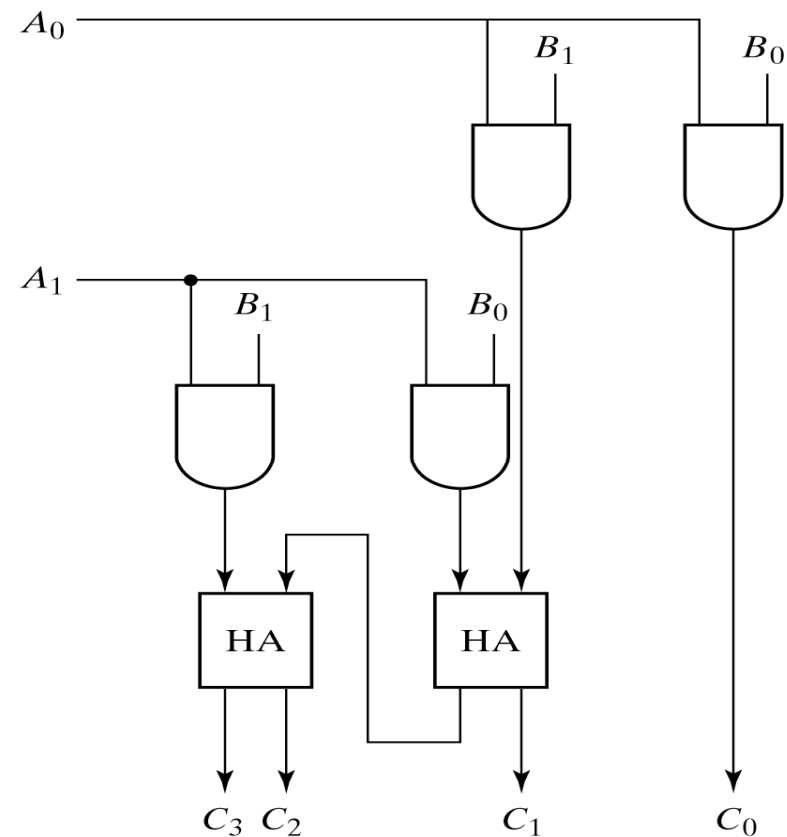


Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

4.7 Binary multiplier

- (K-bit) x (J-bit)
 - (K x J) AND gates,
(J-1) K-bit adder needed

$$\begin{array}{r} B_3 B_2 B_1 B_0 \\ \times \quad A_2 A_1 A_0 \\ \hline \end{array}$$

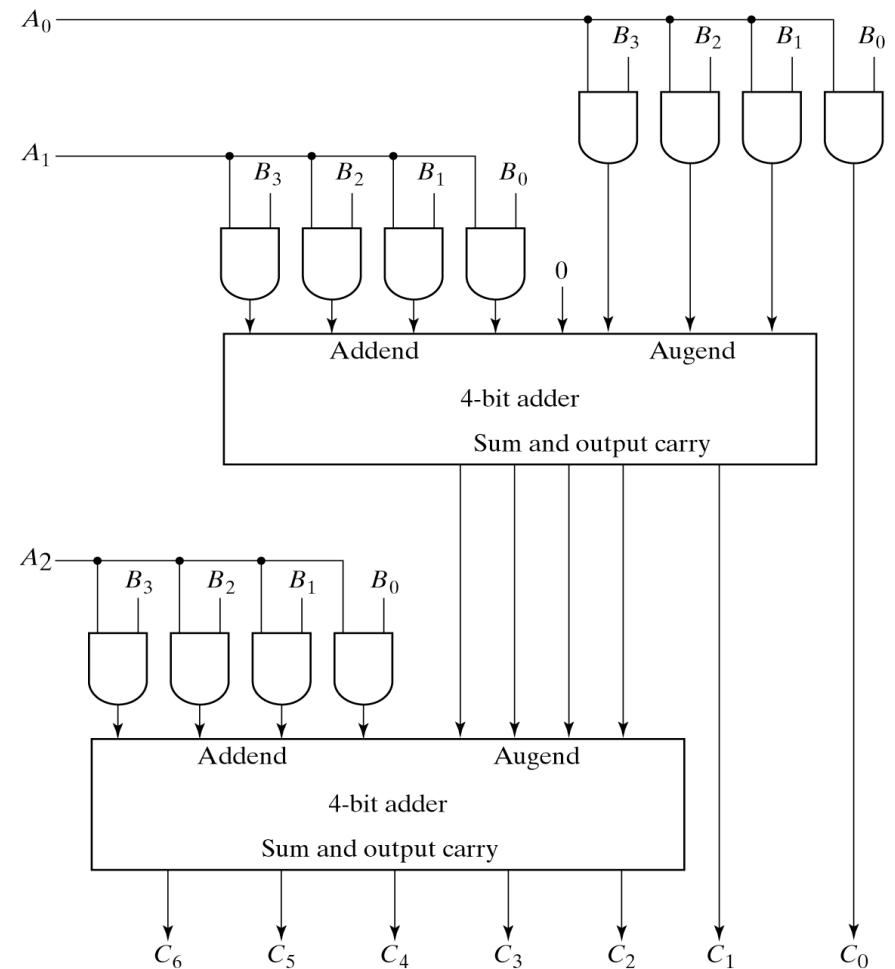


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

4.8 Magnitude comparator

- $X_i=1$ only if the pair of bits in i are equal
- $(A=B)=x_3x_2x_1x_0$
- $(A>B)=A_3B_3'+x_3A_2B_2'+x_3x_2A_1B_1'+x_3x_2x_1A_0B_0'$
- $(A<B)=A_3'B_3+x_3A_2'B_2+x_3x_2A_1'B_1+x_3x_2x_1A_0'B_0$

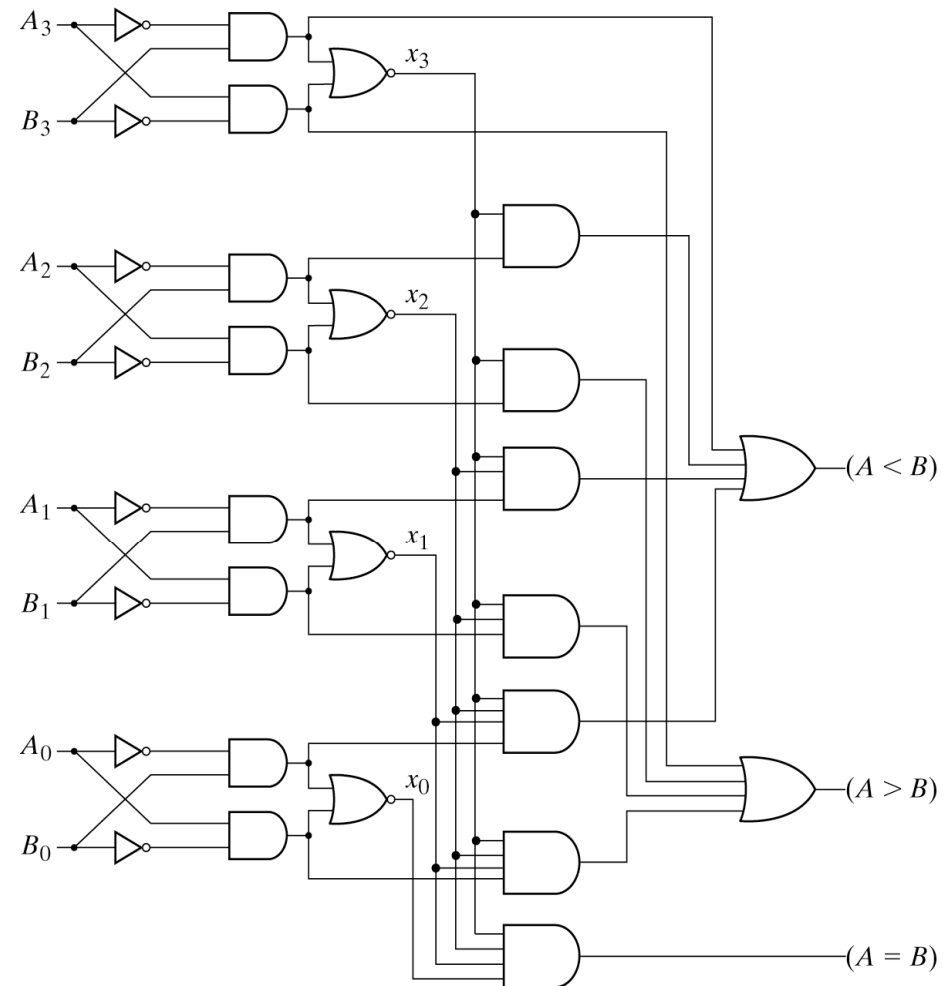


Fig. 4-17 4-Bit Magnitude Comparator

4.9 Decoders

- Generate the 2^n (or less) minterms of n input variables
 - Eg)3 to 8 line decoder

Table 4-6
Truth Table of a 3-to-8-Line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

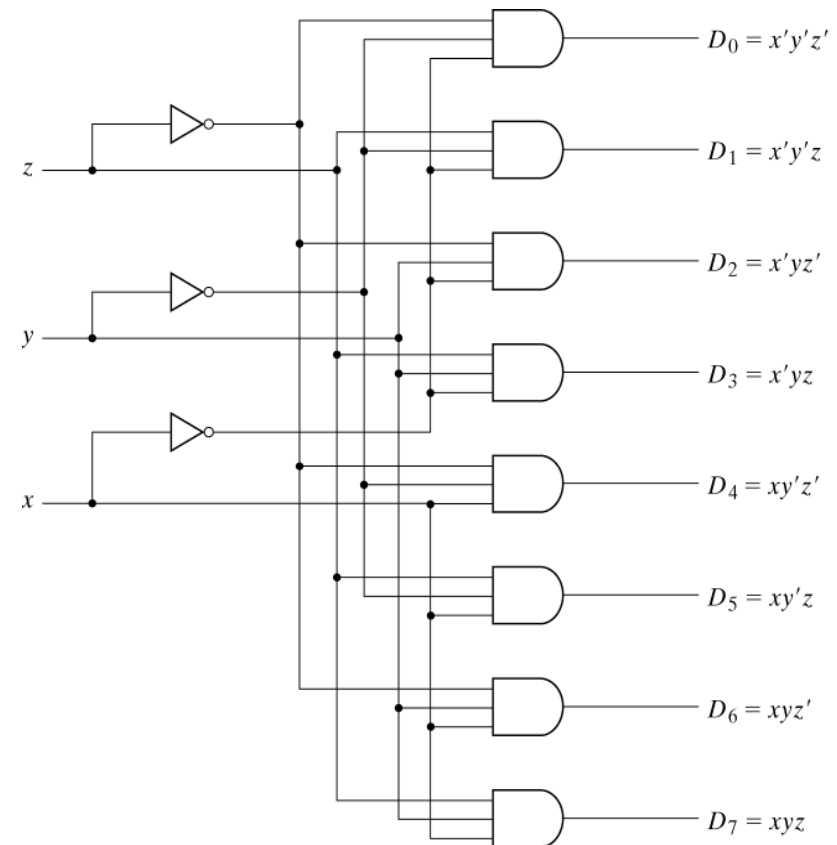
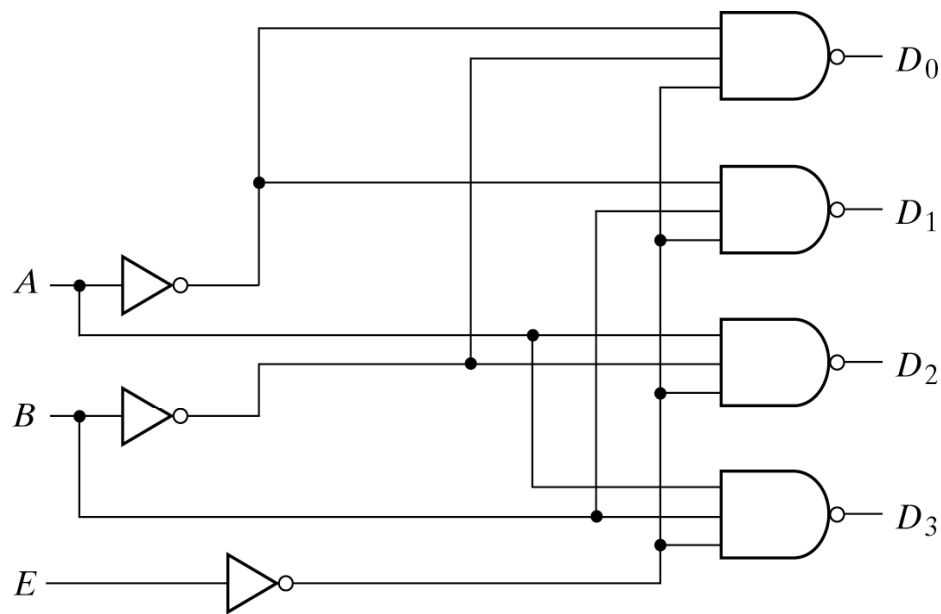


Fig. 4-18 3-to-8-Line Decoder

4.9 Decoders

- 2 to 4 line decoder with Enable input
 - Control circuit operation by E



(a) Logic diagram

<i>E</i>	<i>A</i>	<i>B</i>	<i>D</i> ₀	<i>D</i> ₁	<i>D</i> ₂	<i>D</i> ₃
1	<i>X</i>	<i>X</i>	1	1	1	1
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

(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

4.9 Decoders

- Decoders with enable inputs can be a larger decoder circuit

Eg) 4x16 decoder by two 3x8 decoders

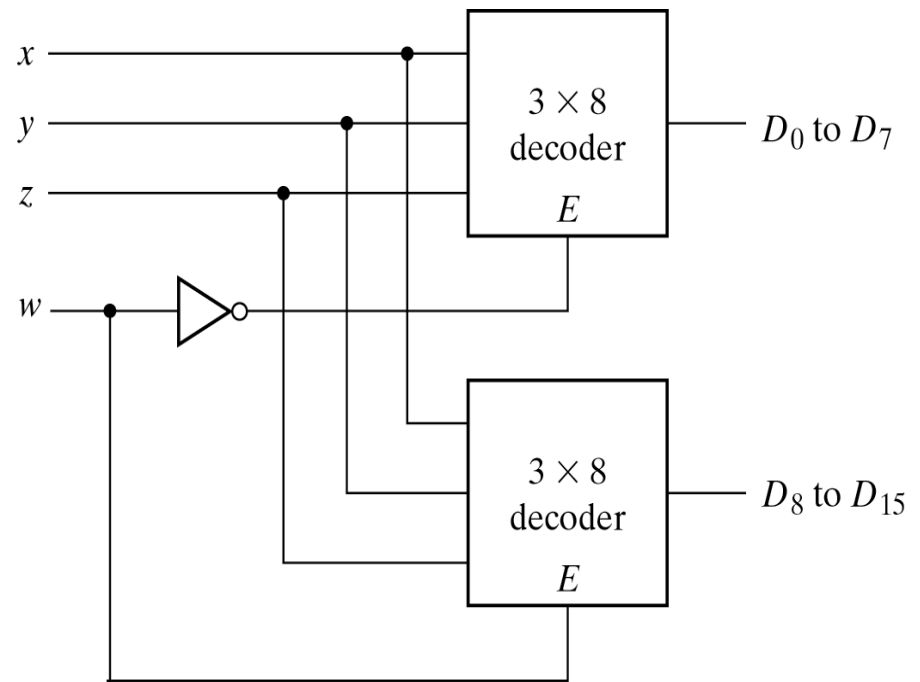


Fig. 4-20 4×16 Decoder Constructed with Two 3×8 Decoders

4.9 Decoders - Combinational logic implementation

- Combinational logic implementation
 - Any combinational circuit can be implemented with line decoder and OR gates
 - Eg)full adder

Table 4-4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

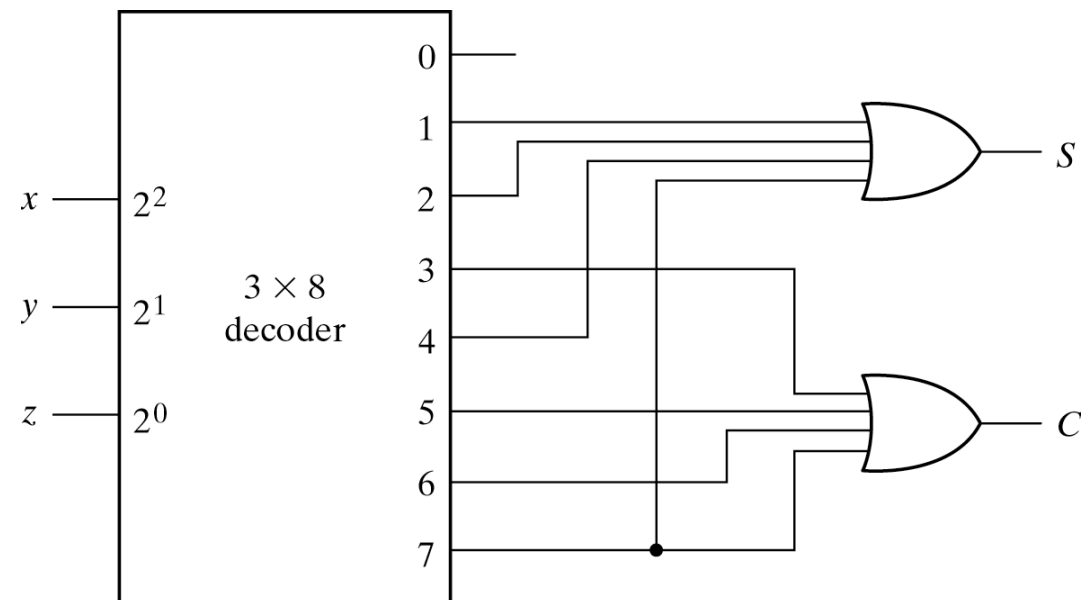


Fig. 4-21 Implementation of a Full Adder with a Decoder

4.10 Encoders

- Inverse operation of a decoder
- Generate n outputs of 2^n input values
 - Eg) octal to binary encoder

Table 4-7
Truth Table of Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

4.10 Encoders - Priority encoder

- Problem happens two or more inputs equal to 1 at the same time
- Give a priority function to circuit

Table 4-8
Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

(x100 means 0100, 1100)

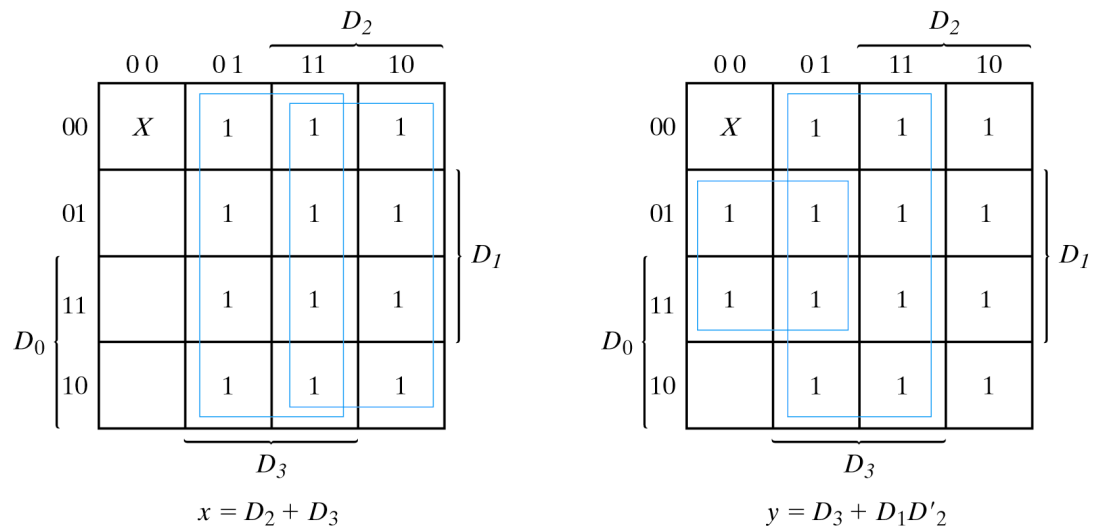


Fig. 4-22 Maps for a Priority Encoder

4.11 Multiplexers

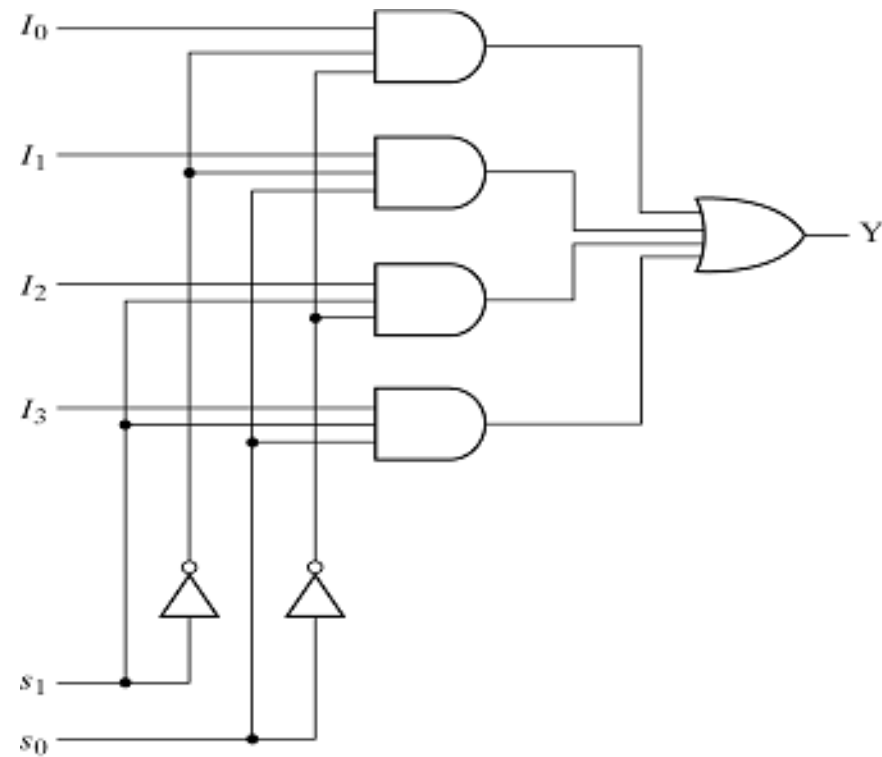
- Select a binary information from many input lines
- Selection is controlled by a set of selection lines
- 2^n input lines have n selection lines

4.11 Multiplexers

4 to 1 line multiplexer

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table



(a) Logic diagram

4.11 Multiplexers

• Quadruple 2 to 1 line multiplexer

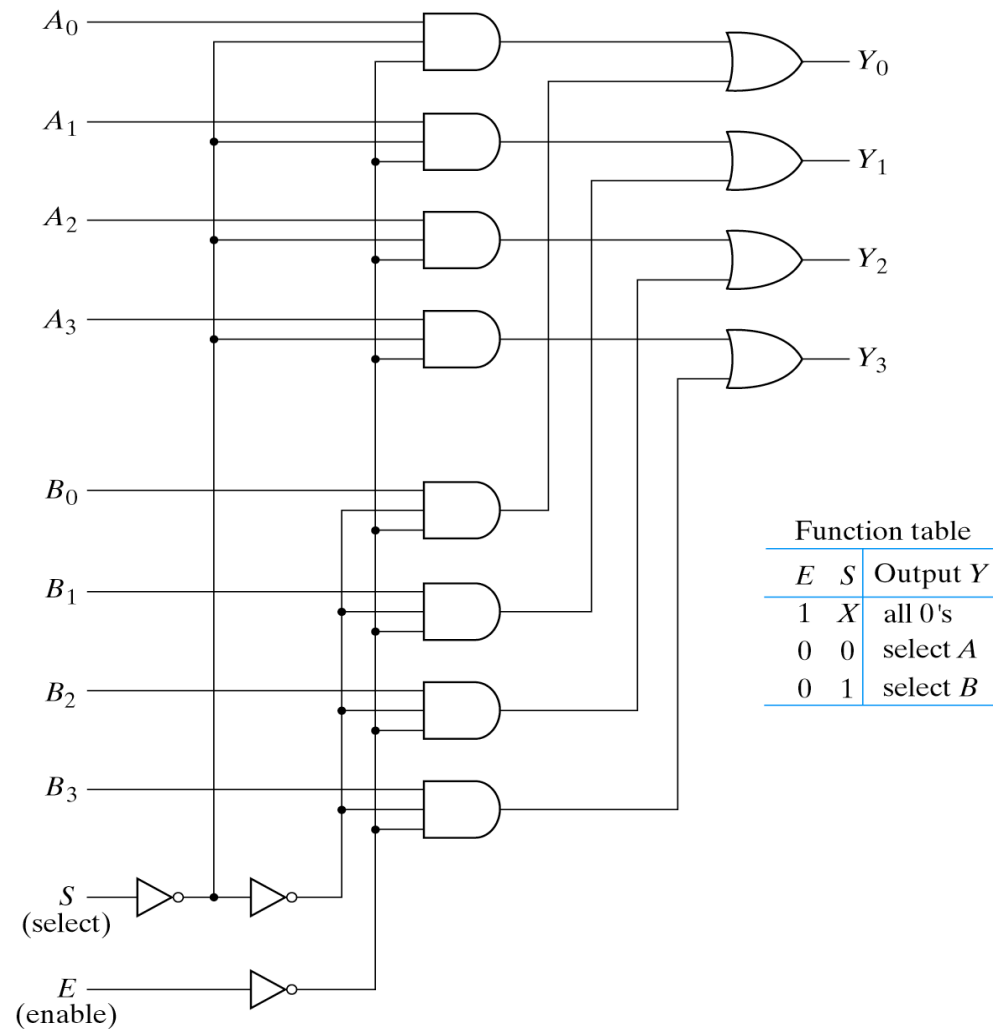


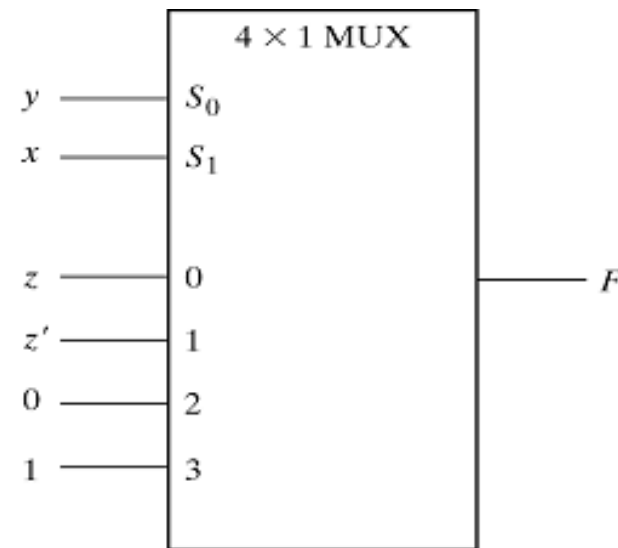
Fig. 4-26 Quadruple 2-to-1-Line Multiplexer

4.11 Multiplexers - Boolean function implementation

- Boolean function implementation
 - Minterms of function are generated in a MUX
 - n input variables, $n-1$ selection input

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

(a) Truth table



(b) Multiplexer implementation

$$F = xy + yz' + x'y'z$$

4.11 Multiplexers - Three-state gates

- Three-state gates
 - Logic 1, 0 and *high-impedance*
 - *High-impedance* behaves like an open circuit

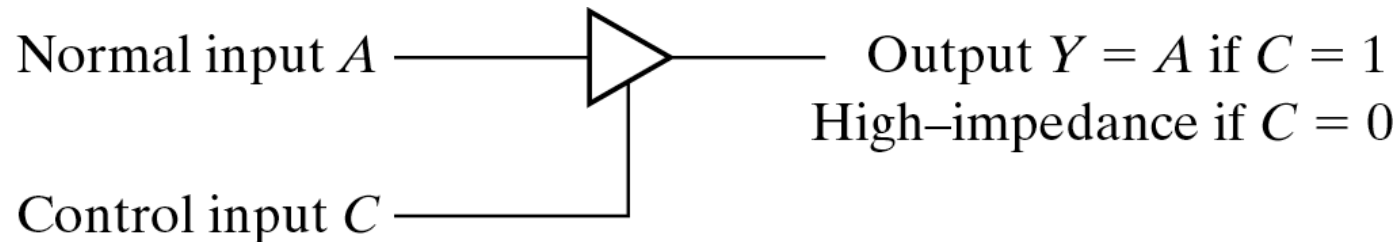
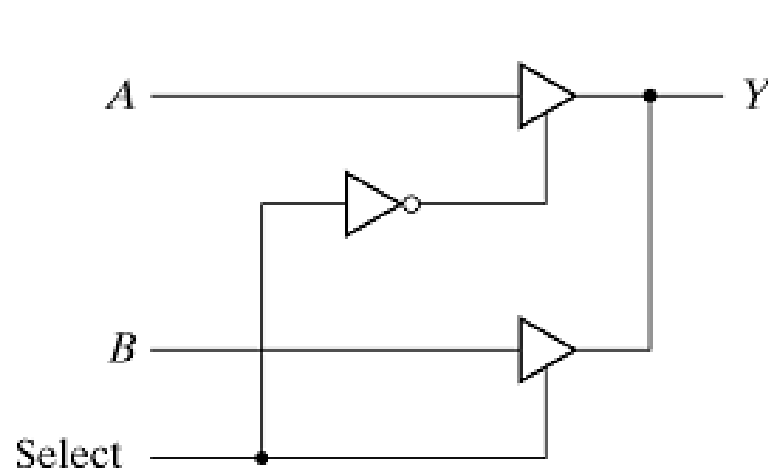


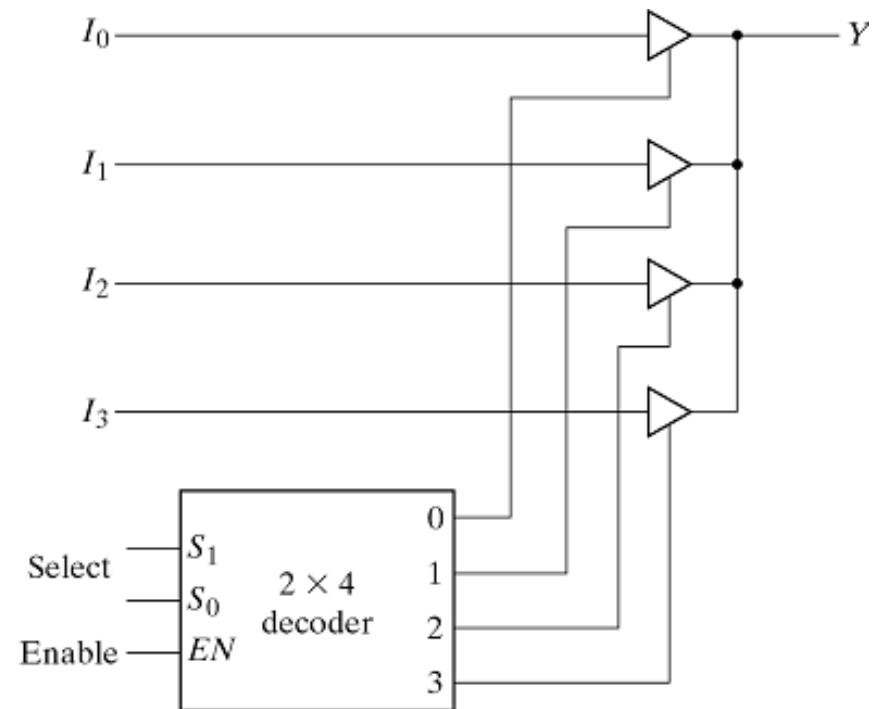
Fig. 4-29 Graphic Symbol for a Three-State Buffer

4.11 Multiplexers

• Multiplexers with three-state gates



(a) 2-to-1- line mux



(b) 4 - to - 1 line mux

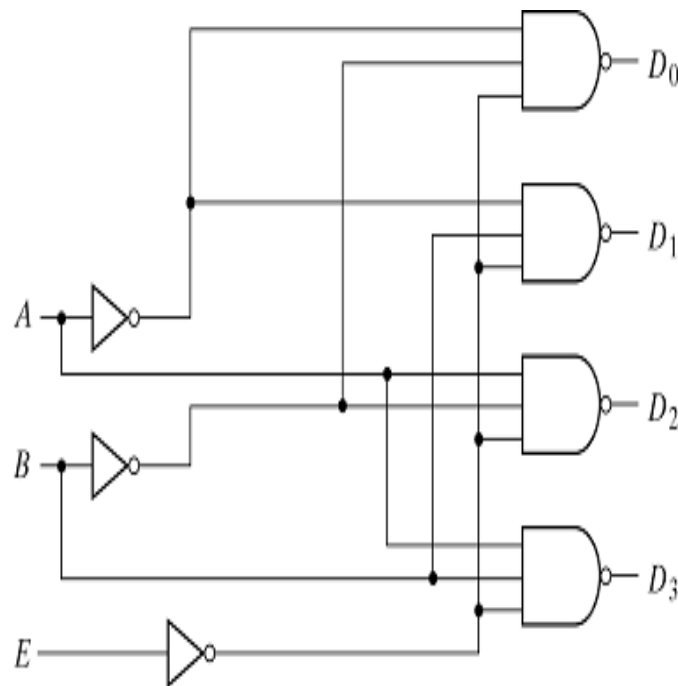
Fig. 4-30 Multiplexers with Three-State Gates

4.12 HDL for combinational circuit

- Modeling techniques:
 - Gate level modeling
 - Instantiation of gates and user defined modules
 - Dataflow modeling
 - Using continuous assignment statements-***assign***
 - Behavioral modeling
 - Using procedural assignment statements-***always***

4.12 HDL for combinational circuit - Gate-level modeling

- Circuit is specified by its gates and their interconnection



HDL Example 4-1

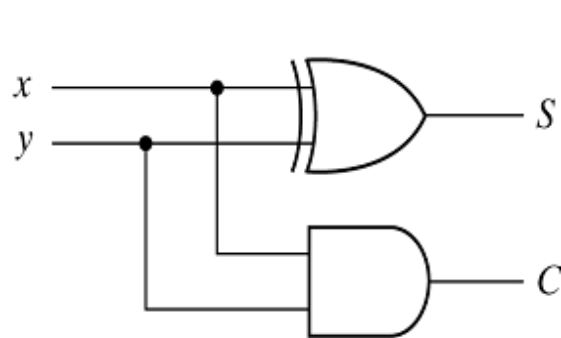
//Gate-level description of a 2-to-4-line decoder
//Figure 4-19

```
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
```

4.12 HDL for combinational circuit - Gate-level modeling

Instantiation

```
module halfadder (S,C,x,y);  
    input x,y;  
    output S,C;  
    //Instantiate primitive gates  
    xor (S,x,y);  
    and (C,x,y);  
endmodule
```



```
module fulladder (S,C,x,y,z);  
    input x,y,z;  
    output S,C;  
    wire S1,D1,D2;  
    //Instantiate the halfadder  
    halfadder HA1 (S1,D1,x,y),  
               HA2 (S,D2,S1,z);  
    or g1 (C,D2,D1);  
endmodule
```

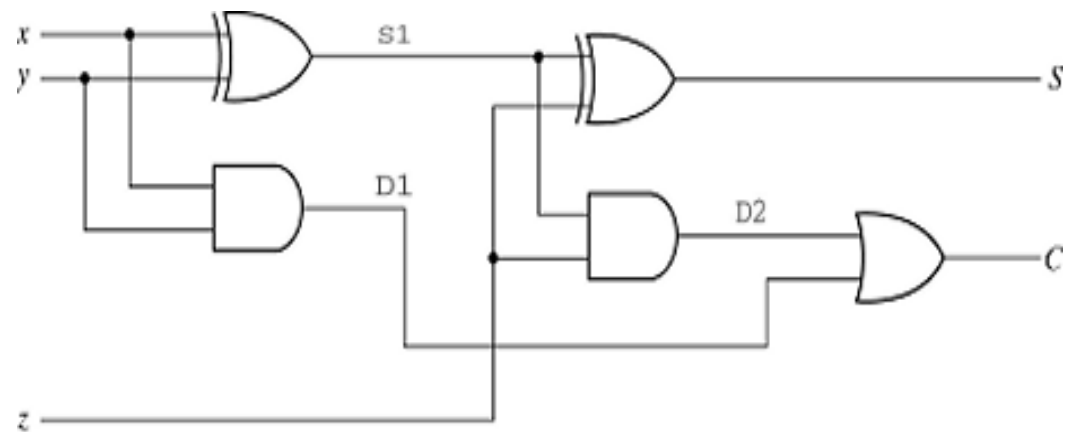


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

4.12 HDL for combinational circuit - Gate-level modeling

Instantiation in 4-bit adder

```
module _4bit_adder (S,C4,A,B,C0);  
  input [3:0] A,B;  
  input C0;  
  output [3:0] S;  
  output C4;  
  wire C1,C2,C3; //Intermediate carries  
  //Instantiate the fulladder  
  fulladder FA0 (S[0],C1,A[0],B[0],C0),  
             FA1 (S[1],C2,A[1],B[1],C1),  
             FA2 (S[2],C3,A[2],B[2],C2),  
             FA3 (S[3],C4,A[3],B[3],C3);  
  
endmodule
```

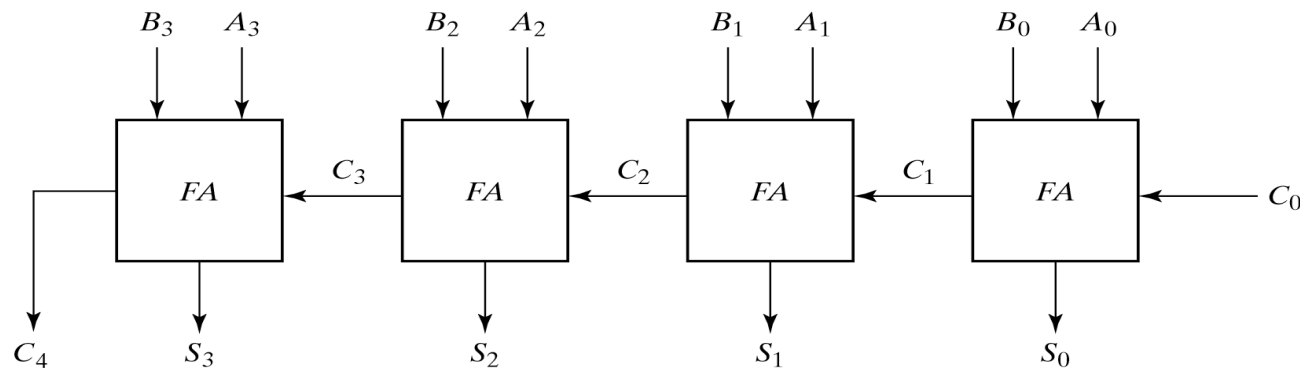


Fig. 4-9 4-Bit Adder

4.12 HDL for combinational circuit - 3 State Gate

4.12 HDL for combinational circuit - Dataflow modeling

- Assign a value to a net by using operands and operators

eg) $J=01, K=10$ can be
 $\{J, K\}=0110$

$\text{out} = x ? A : B$ means
 $\text{out} = A$, if x is true
 $= B$, if x is false

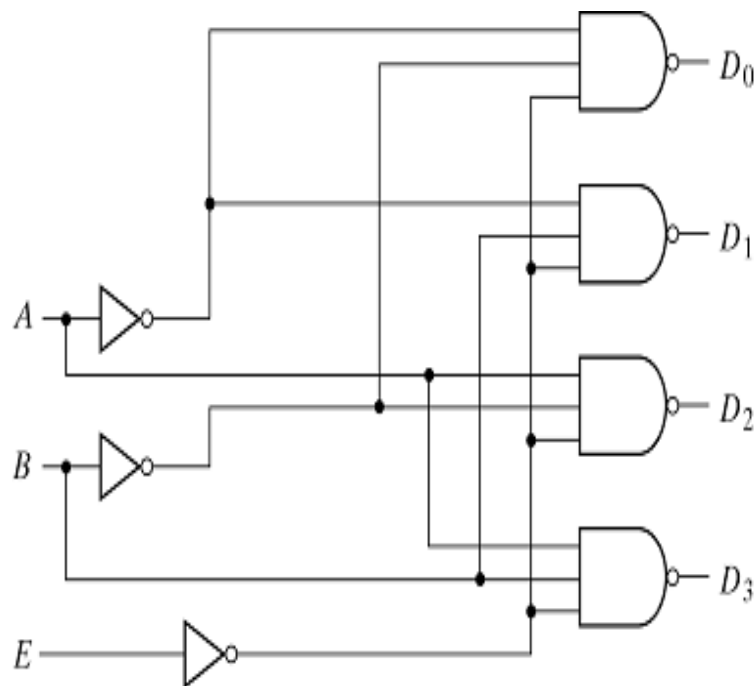
Table 4-10
Verilog HDL Operators

Symbol	Operation
+	binary addition
-	binary subtraction
&	bit-wise AND
	bit-wise OR
^	bit-wise XOR
~	bit-wise NOT
==	equality
>	greater than
<	less than
{ }	concatenation
? :	conditional

4.12 HDL for combinational circuit - Dataflow modeling

● Assignment

- 2-to-4 line decoder



HDL Example 4-3

//Dataflow description of a 2-to-4-line decoder
//See Fig. 4-19

```
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
```

4.12 HDL for combinational circuit - Dataflow modeling

● Assignment - 4-bit adder

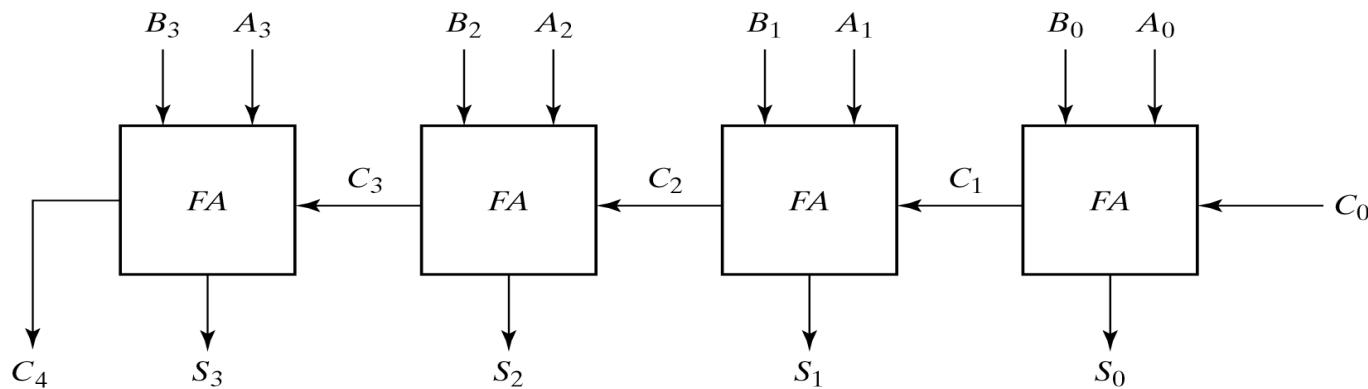


Fig. 4-9 4-Bit Adder

HDL Example 4-4

```
//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
```

4.12 HDL for combinational circuit - Behavioral modeling

- Use procedural assignment statement, ***always***

- Target output must be the *reg* data type

Eg) 4 to 1 line mux

```
module mux4x1_bh (i0,i1,i2,i3,select,y);  
    input i0,i1,i2,i3;  
    input [1:0] select;  
    output y;  
    reg y;  
    always @ (i0 or i1 or i2 or i3 or select)  
        case (select)  
            2'b00: y = i0;  
            2'b01: y = i1;  
            2'b10: y = i2;  
            2'b11: y = i3;  
        endcase  
endmodule
```


4.12 HDL for combinational circuit - Writing a simple test bench

- Test bench : Applying stimulus to test HDL and observe its response
- **reg** - inputs , **wire** - outputs

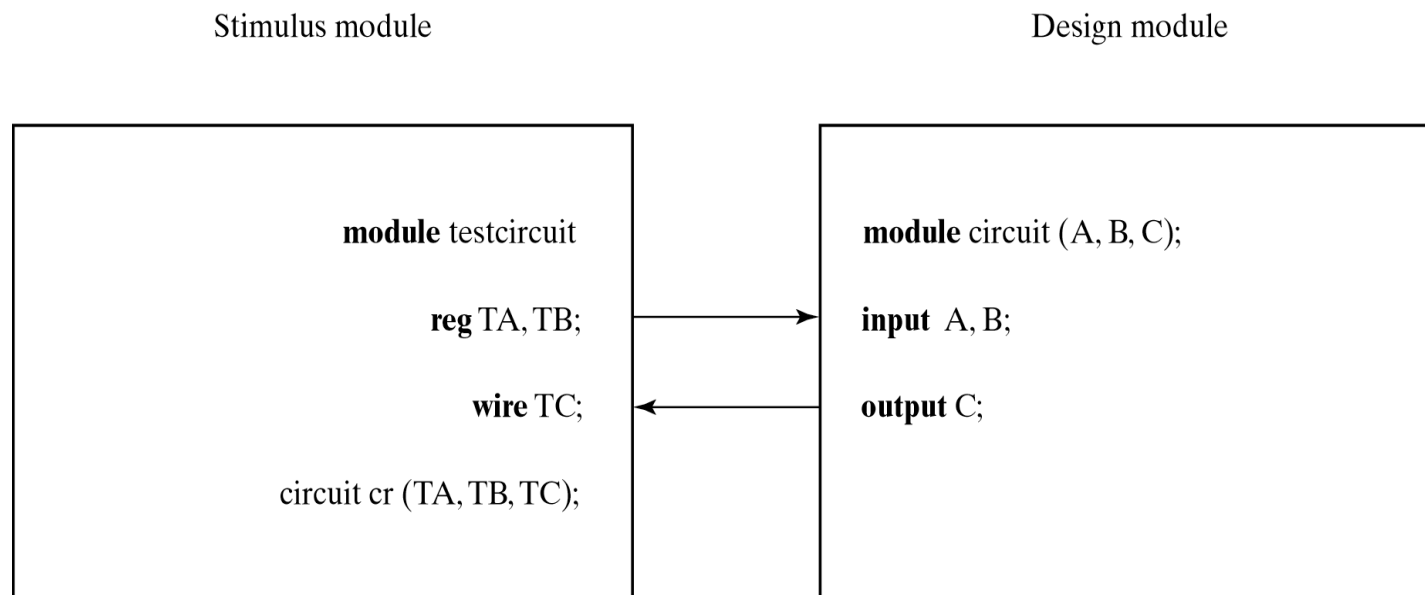


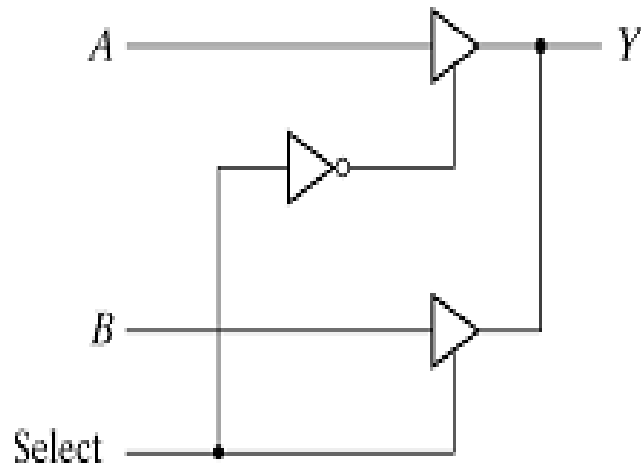
Fig. 4-33 Stimulus and Design Modules Interaction

4.12 HDL for combinational circuit - Writing a simple test bench

- System tasks : keywords that can display various outputs (begin with \$)
- \$display , \$write , \$monitor , \$time , \$finish
- Format of system tasks
 - Task name(format specification,argument list);
 - Eg) \$monitor(%d %b %b, C,A,B);

4.12 HDL for combinational circuit - Writing a simple test bench

● Example of test bench



```
//Stimulus for mux2x1_df.
module testmux;
    reg TA,TB,TS; //inputs for mux
    wire Y;       //output from mux
    mux2x1_df mx (TA,TB,TS,Y); // instantiate mux
    initial
        begin
            TS = 1; TA = 0; TB = 1;
            #10 TA = 1; TB = 0;
            #10 TS = 0;
            #10 TA = 0; TB = 1;
        end
    initial
        $monitor("select = %b A = %b B = %b OUT = %b time = %0d",
            TS, TA, TB, Y, $time);
endmodule

//Dataflow description of 2-to-1-line multiplexer
//from Example 4-6
module mux2x1_df (A,B,select,OUT);
    input A,B,select;
    output OUT;
    assign OUT = select ? A : B;
endmodule
```