Digital design

Ch 4. Combinational logic

4-1 Combinational circuits

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

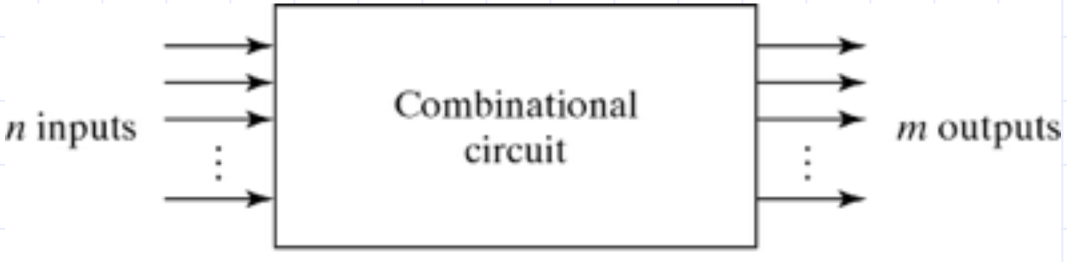


Fig. 4-1 Block Diagram of Combinational Circuit

4-2 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

Obtain 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

Obtain procedure

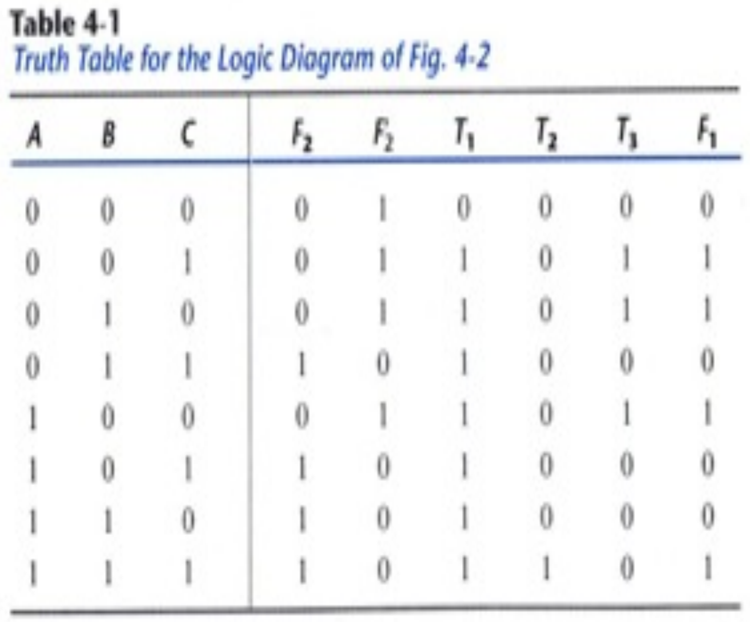


Table 4-1
Truth Table for the Logic Diagram of Fig. 4-2

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

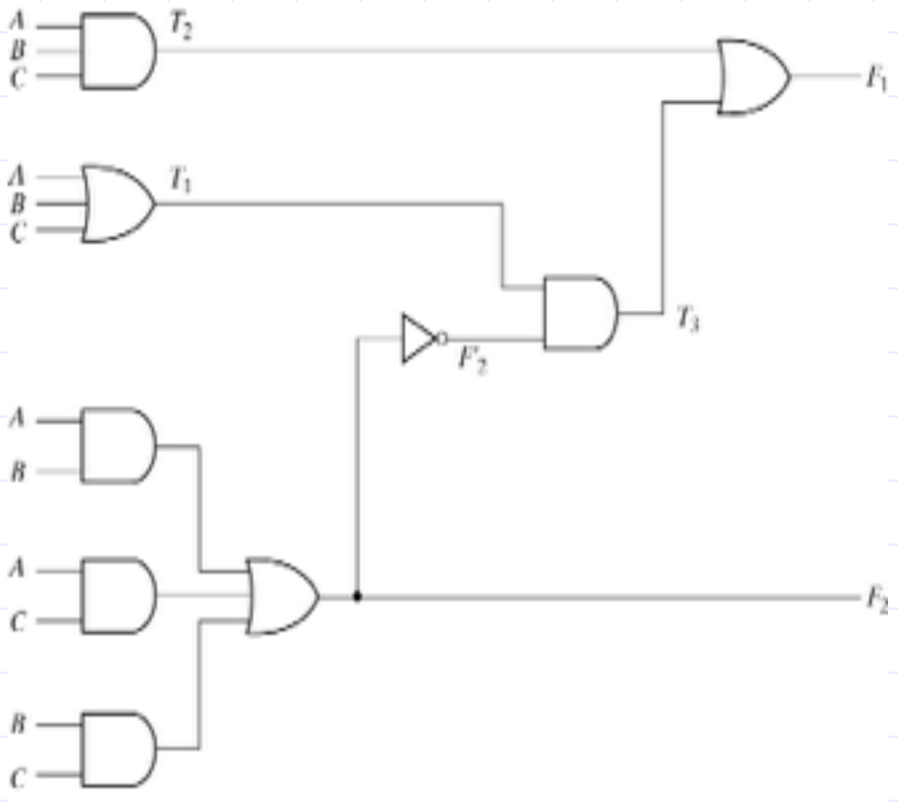


Fig. 4-2 Logic Diagram for Analysis Example

4-3 Design procedure

1. Determine the required number of input and output from specification
2. Assign a letter symbol to each input/output
3. Derive the truth table
4. Obtain the simplified Boolean functions
5. Draw the logic diagram and verify design correctness

Code conversion example

- BCD to excess-3 code converter
  - Excess-3 code : decimal digit+3

Code conversion example

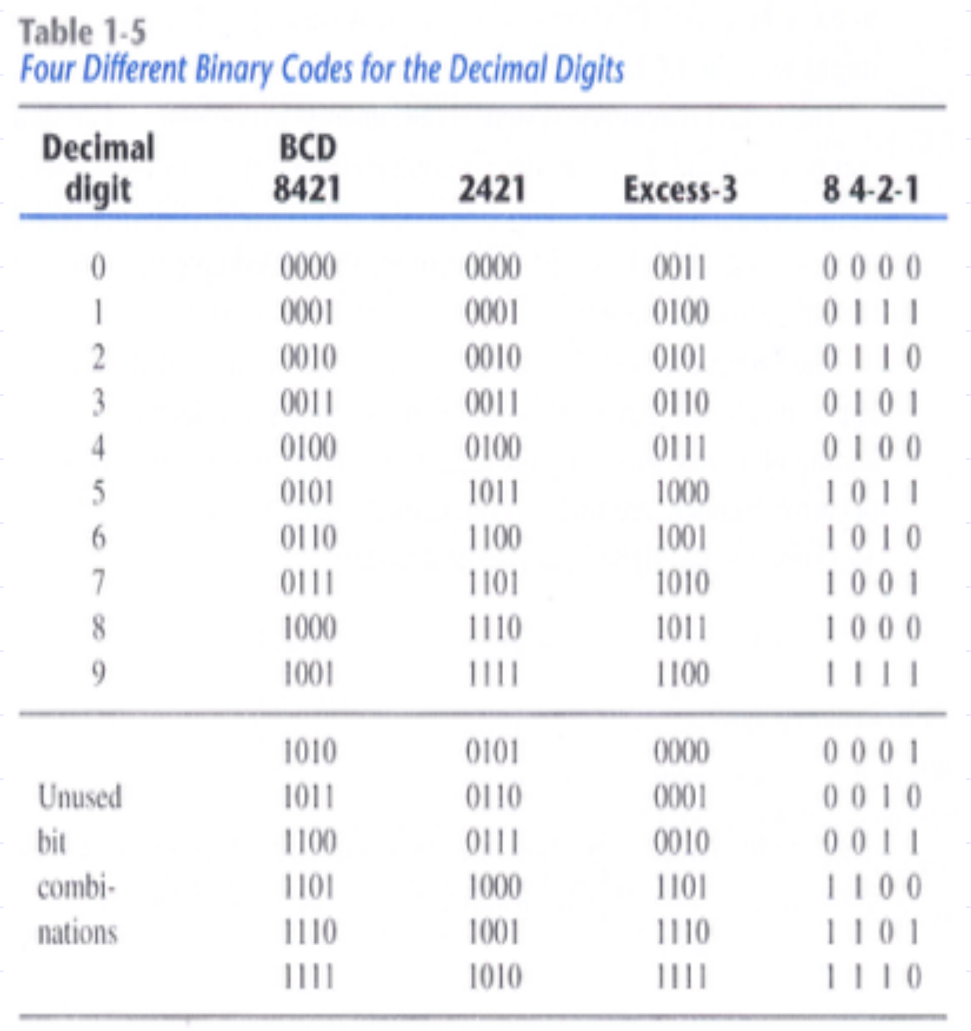


Table 1-5
Four Different Binary Codes for the Decimal Digits

Decimal digit	BCD 8421	2421	Excess-3	8 4-2-1
0	0000	0000	0011	0 0 0 0
1	0001	0001	0100	0 1 1 1
2	0010	0010	0101	0 1 1 0
3	0011	0011	0110	0 1 0 1
4	0100	0100	0111	0 1 0 0
5	0101	1011	1000	1 0 1 1
6	0110	1100	1001	1 0 1 0
7	0111	1101	1010	1 0 0 1
8	1000	1110	1011	1 0 0 0
9	1001	1111	1100	1 1 1 1
	1010	0101	0000	0 0 0 1
Unused	1011	0110	0001	0 0 1 0
bit	1100	0111	0010	0 0 1 1
combi-	1101	1000	1101	1 1 0 0
nations	1110	1001	1110	1 1 0 1
	1111	1010	1111	1 1 1 0

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)

Code converter example

2)Derive truth table

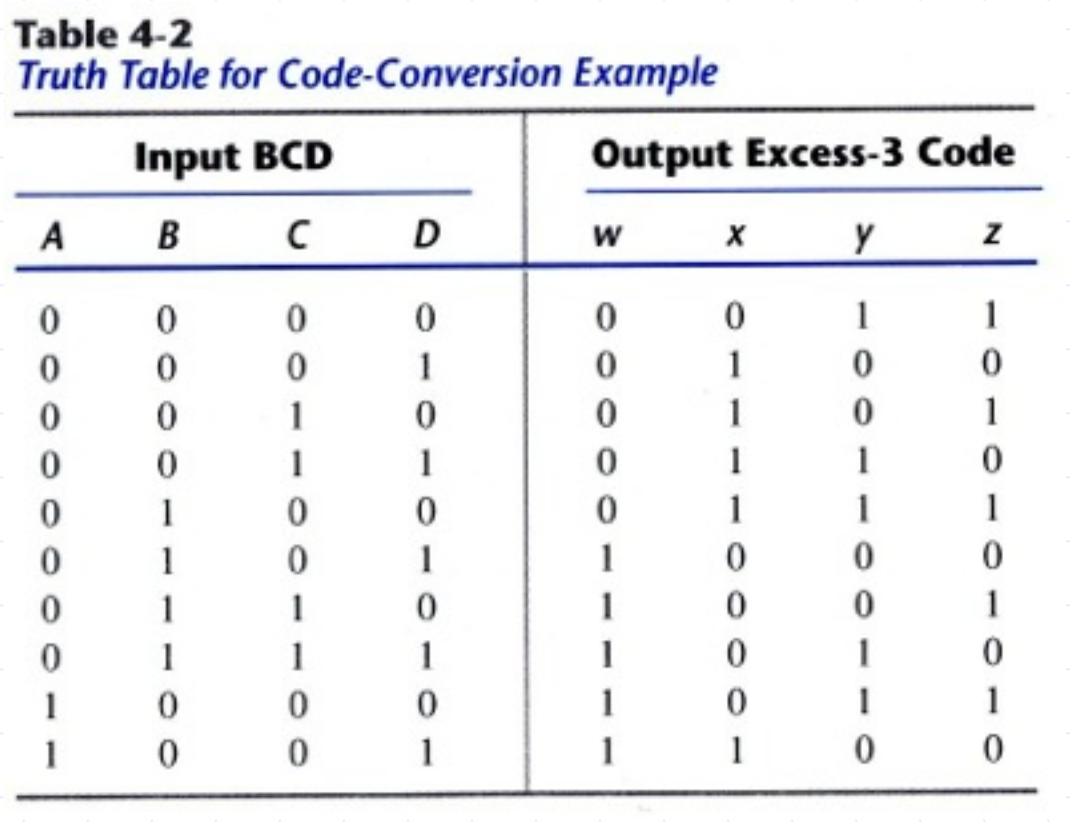


Table 4-2
Truth Table for Code-Conversion Example

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

Code converter example

3)Obtain simplified Boolean functions

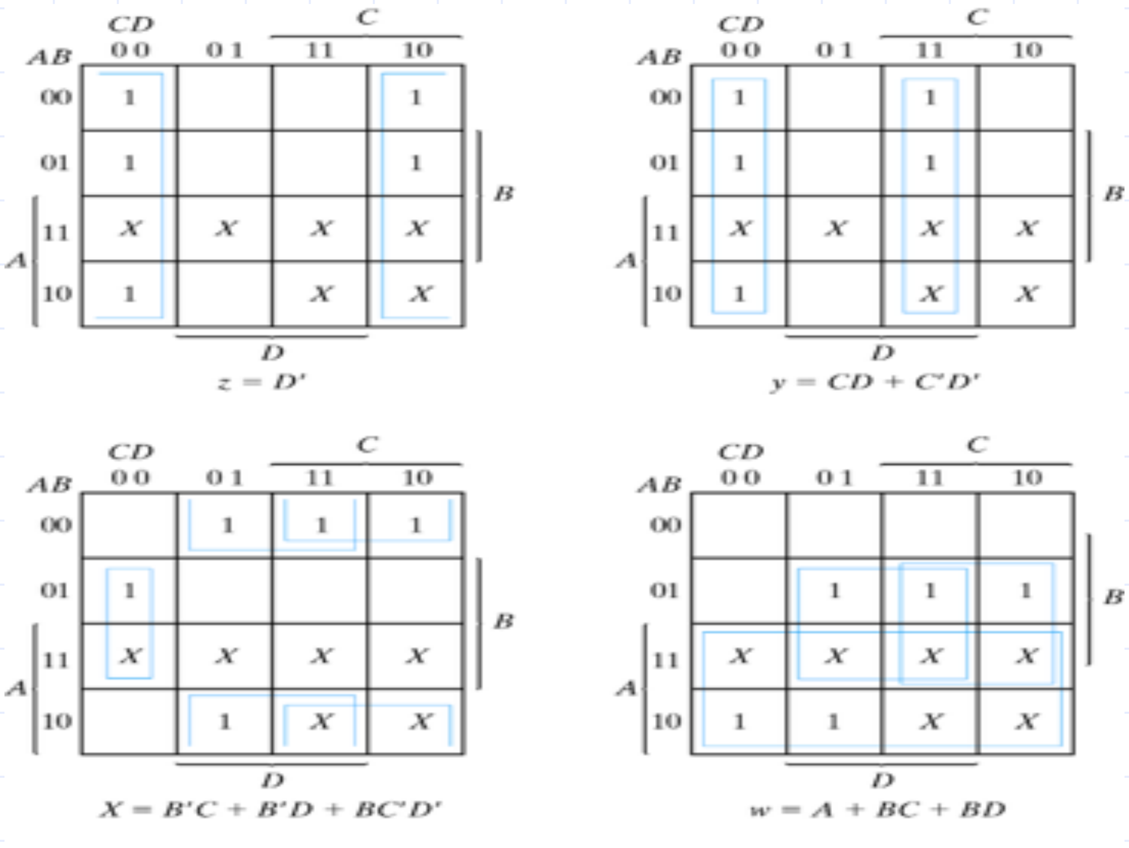


Fig. 4-3 Maps for BCD to Excess-3 Code Converter

Code converter example

4)Draw the logic diagram

$$z = D'$$

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

$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$

$$= B'(C + D) + B(C + D)'$$

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

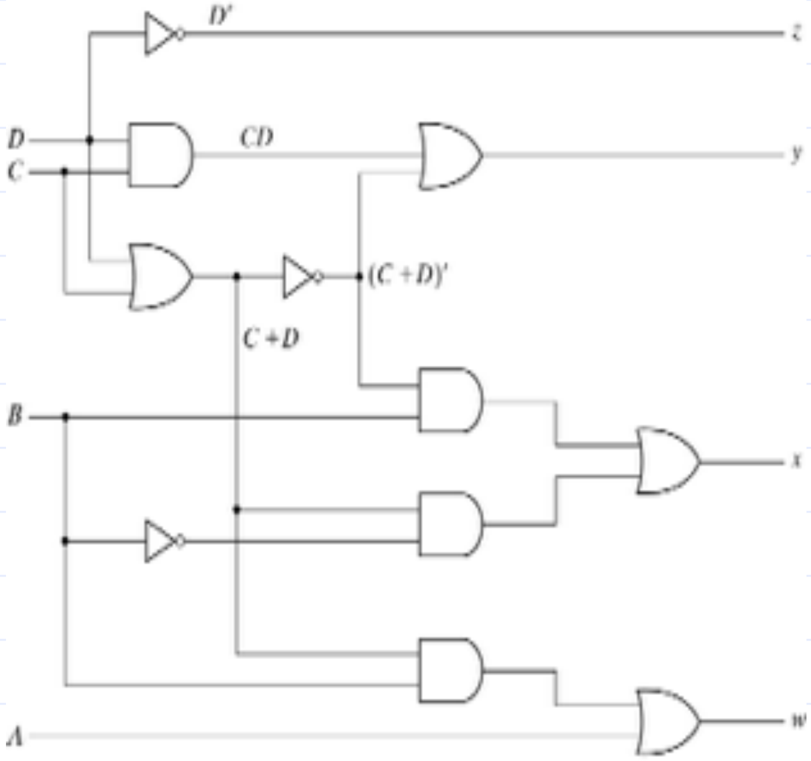


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter

4-4 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

Half Adder

- Sum of 2 binary inputs
- Input : X(augend), Y(addend)

  Output : S(sum), C(carry)

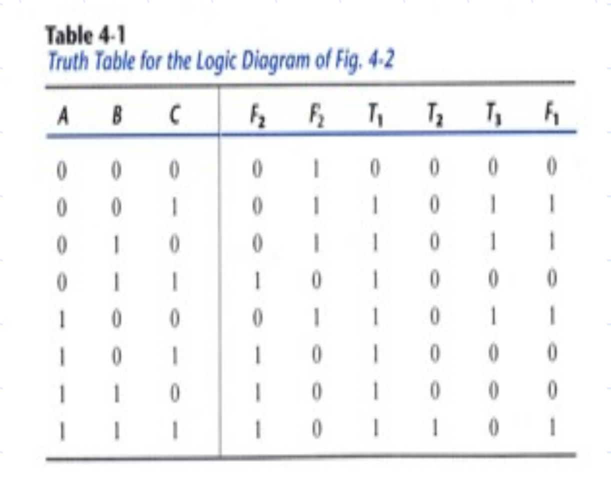


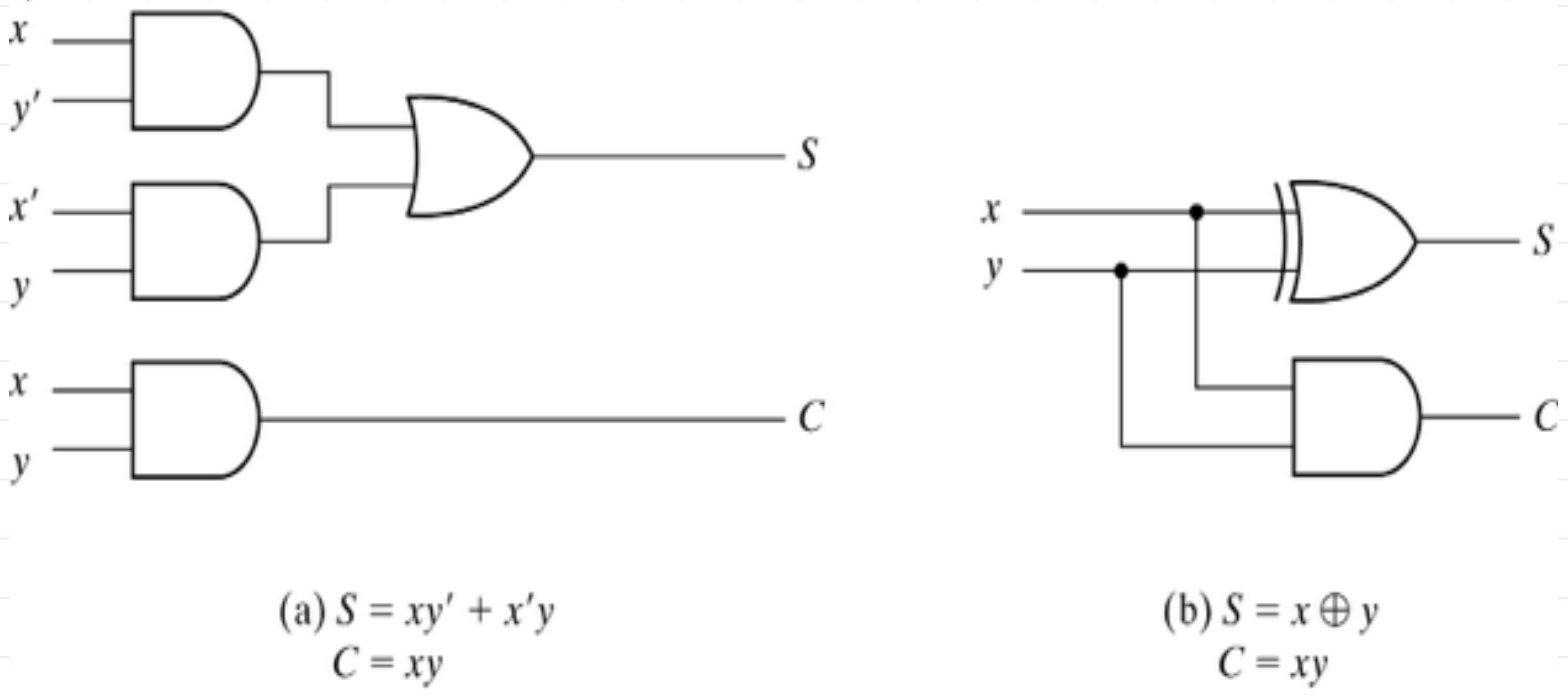
Table 4-1
Truth Table for the Logic Diagram of Fig. 4-2

A	B	C	F_2	$\overline{F}_2$	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

$S = xy' + x'y$

$C = xy$

Half adder



(a) $S = xy' + x'y$
$C = xy$

(b) $S = x \oplus y$
$C = xy$

Fig. 4-5 Implementation of Half-Adder

Full adder

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

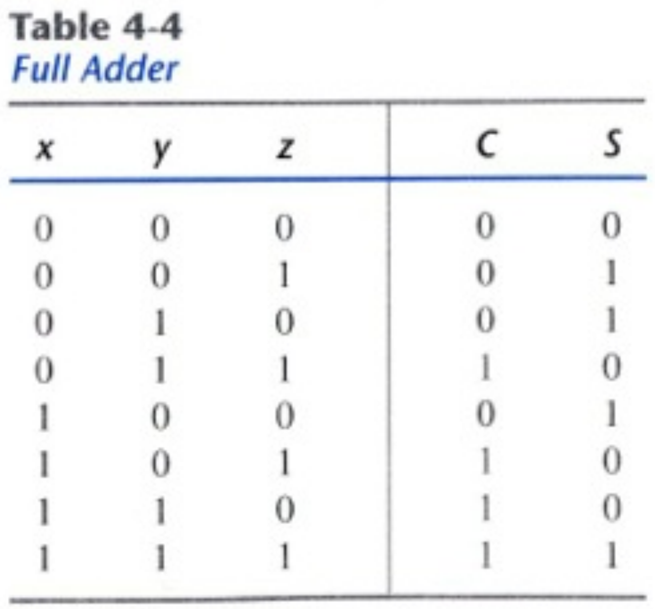
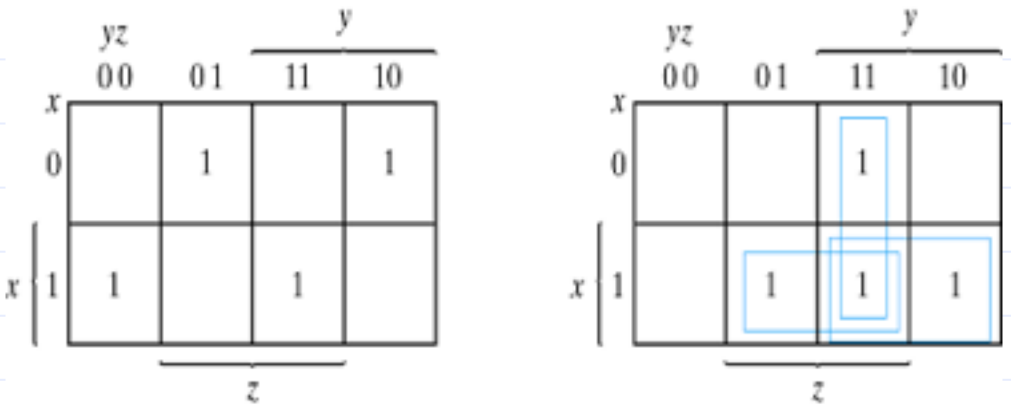


Table 4-4
Full Adder

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



$S = x'y'z + x'yz' + xy'z' + xyz$

$C = xy + xz + yz$
$\quad = xy + xy'z + x'yz$

Fig. 4-6 Maps for Full Adder

Full adder

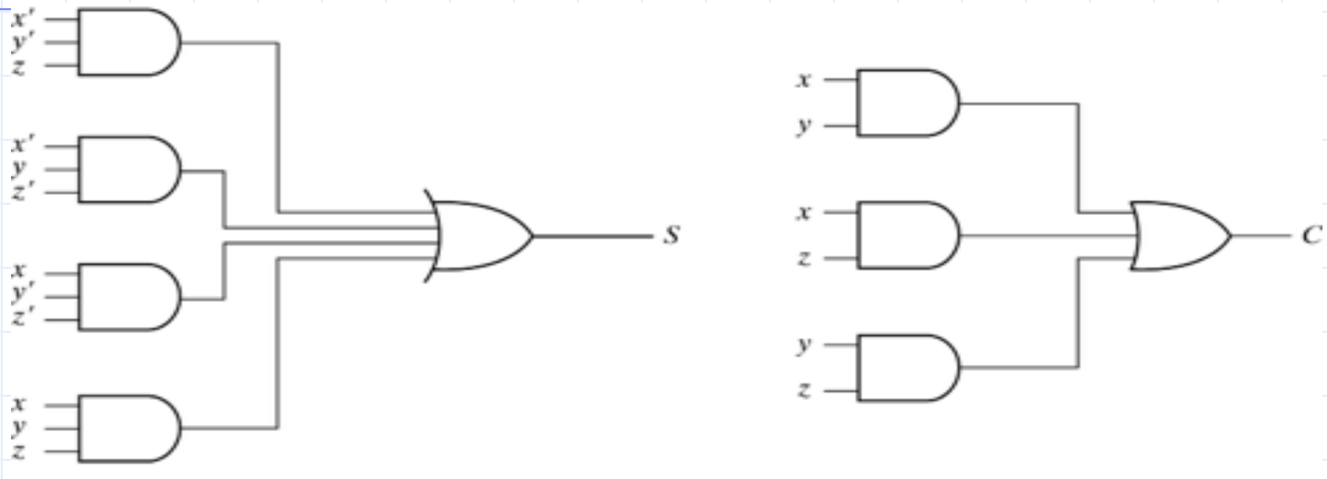


Fig. 4-7 Implementation of Full Adder in Sum of Products

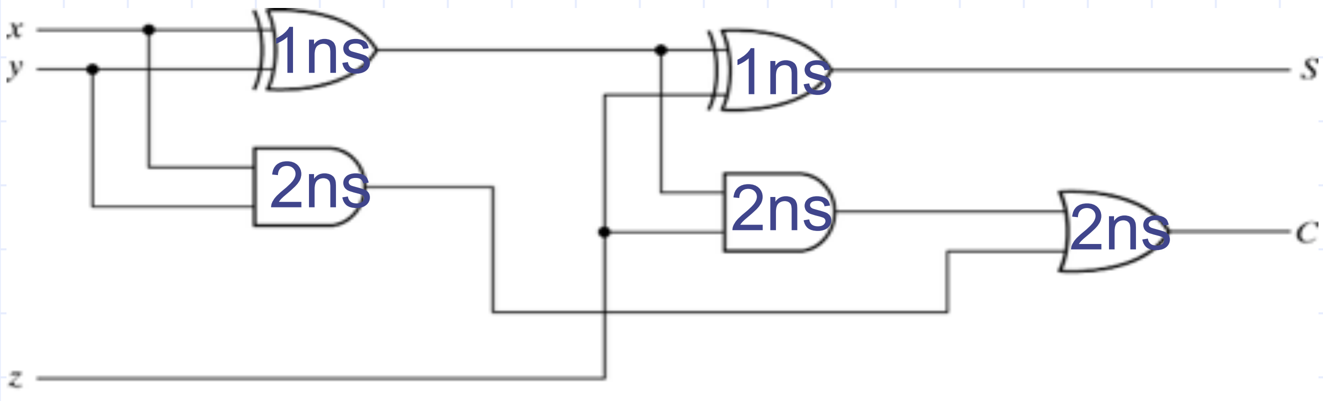


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

지연시간:

x => s: 2ns

x => c: 5ns

y => s: 2ns

y => c: 4ns

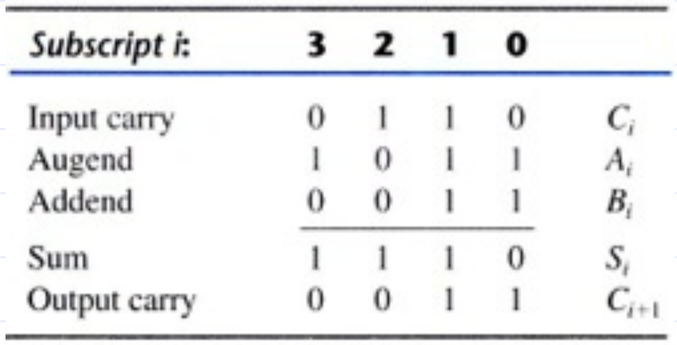
z => s: 1ns

z => c: 4ns

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}

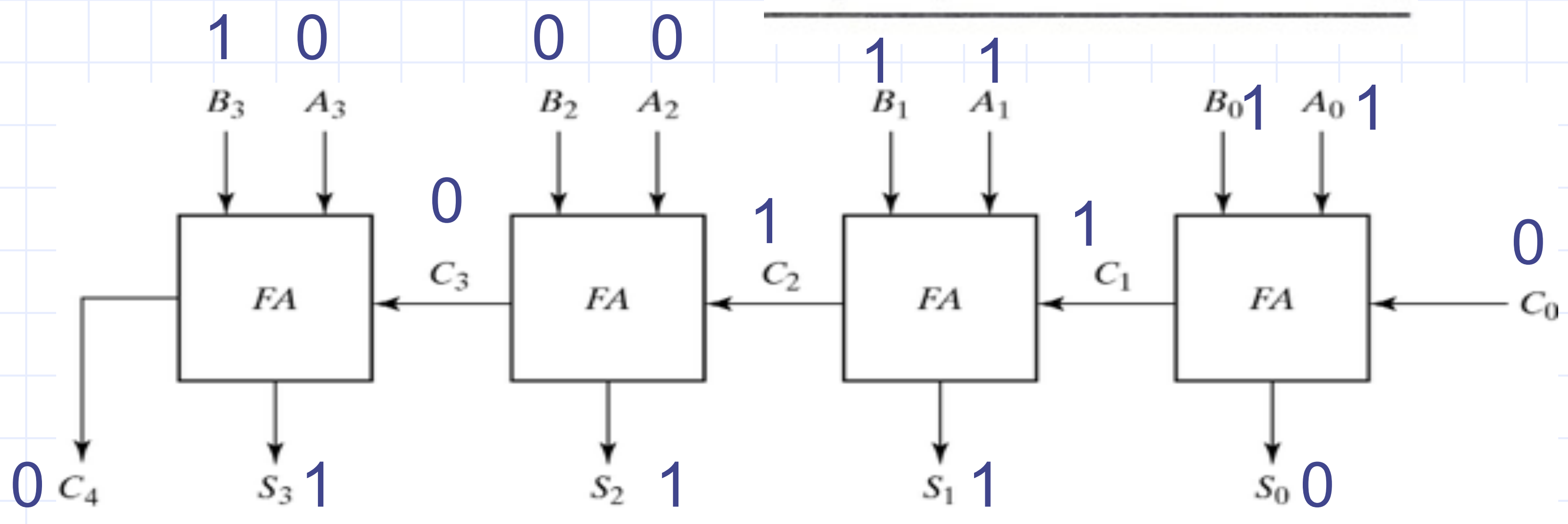


Fig. 4-9 4-Bit Adder

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

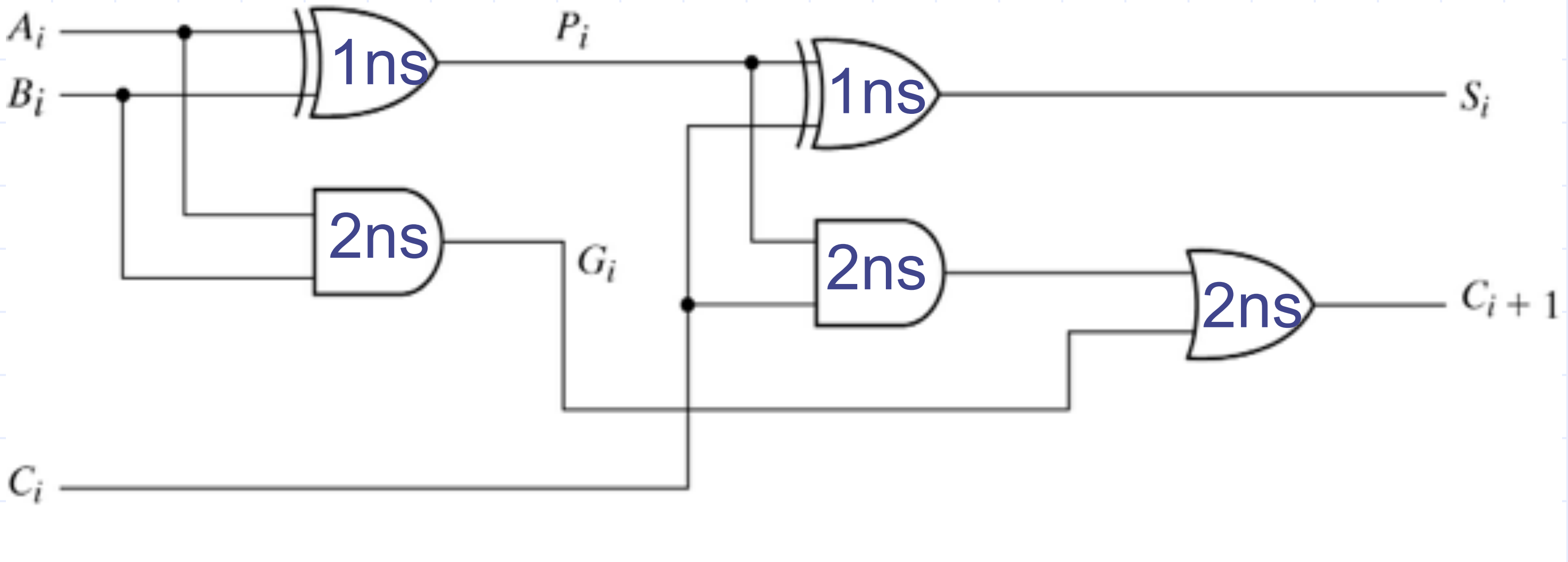


Fig. 4-10 Full Adder with P and G Shown

Carry propagation

- Carry lookahead generator

C_0 = input carry

$C_1 = G_0 + P_0C_0$

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

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

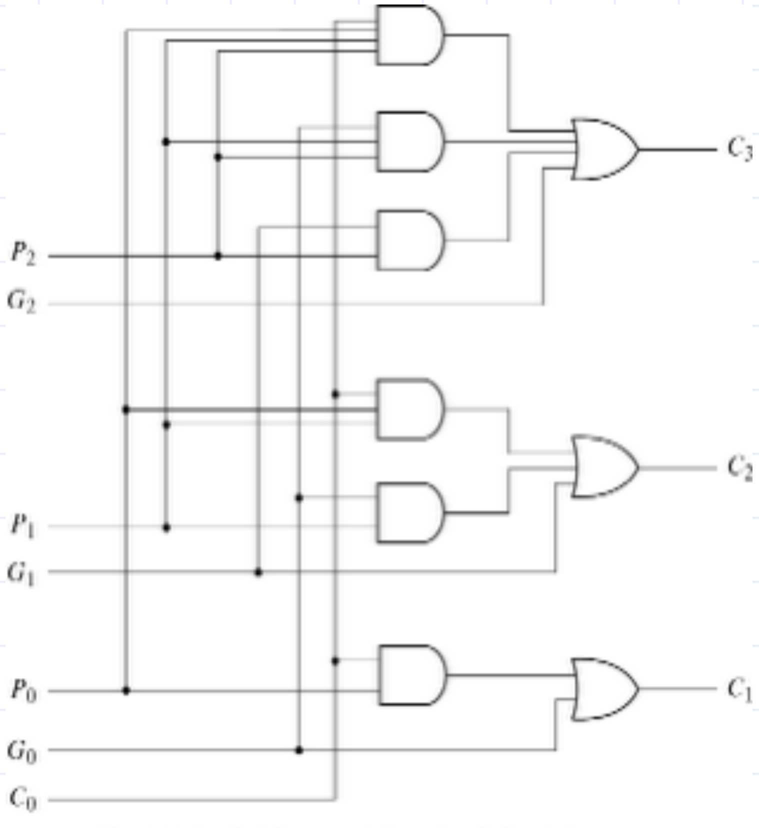


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

Carry propagation

- 4-bit adder with carry lookahead

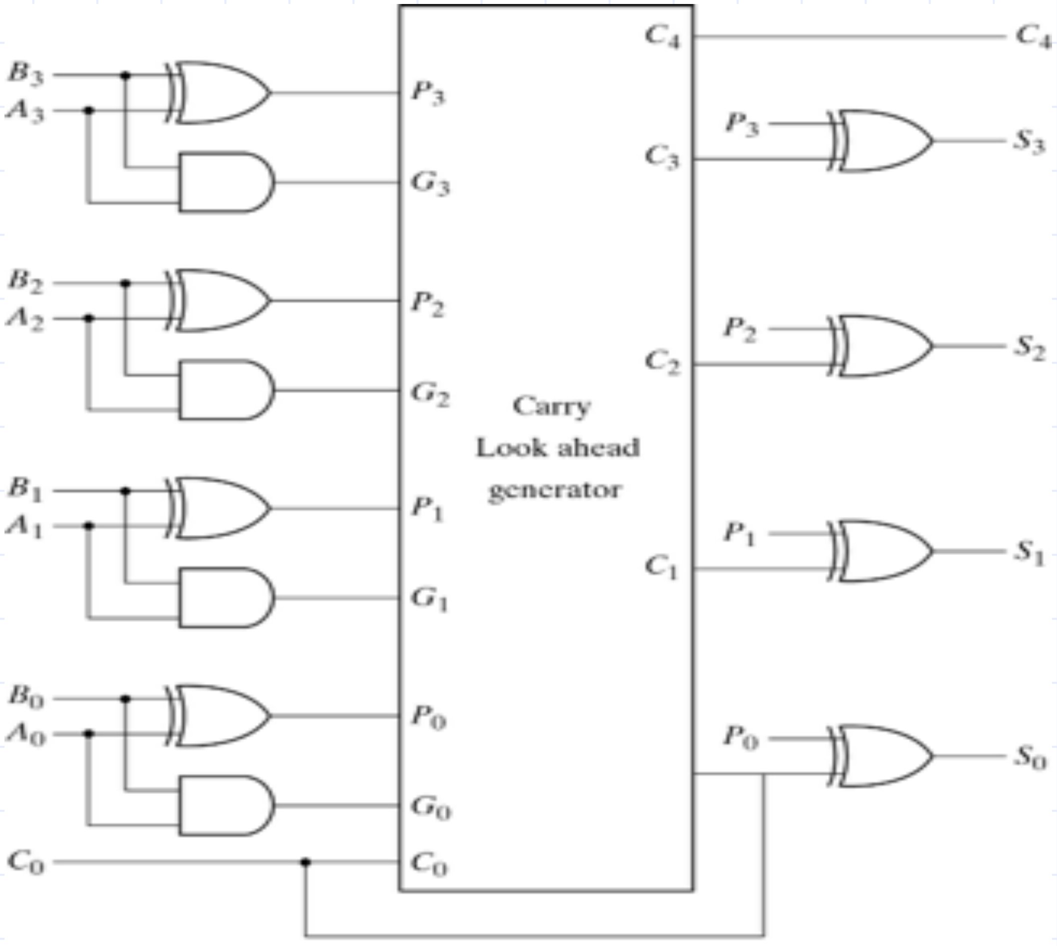


Fig. 4-12 4-Bit Adder with Carry Lookahead

Binary subtractor

- A-B = A+(2's complement of B)
- When M=0(act as adder) M=1(subtractor)

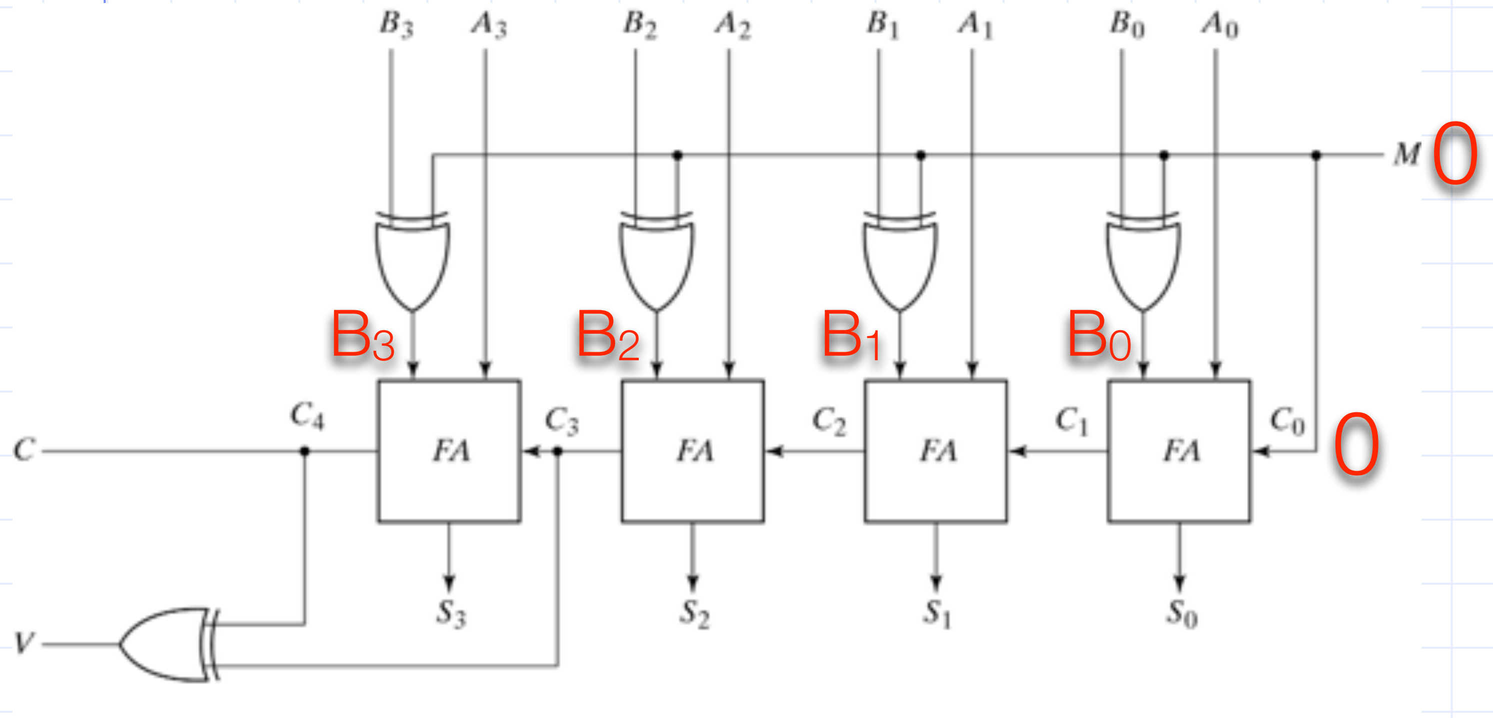


Fig. 4-13 4-Bit Adder Subtractor

$x \oplus 0 = x, \; x \oplus 1 = x'$

Binary subtractor

- A-B = A+(2's complement of B)
- When M=0(act as adder) M=1(subtractor)

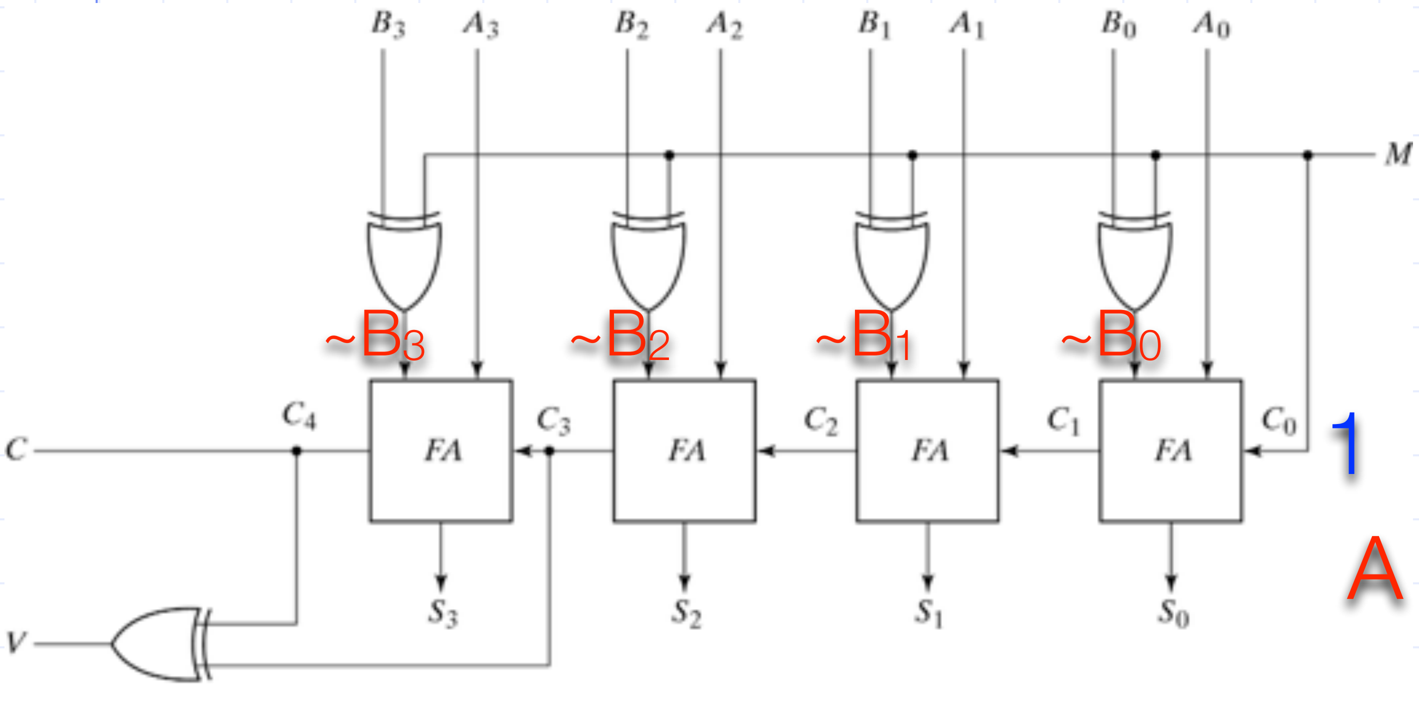


Fig. 4-13 4-Bit Adder Subtractor

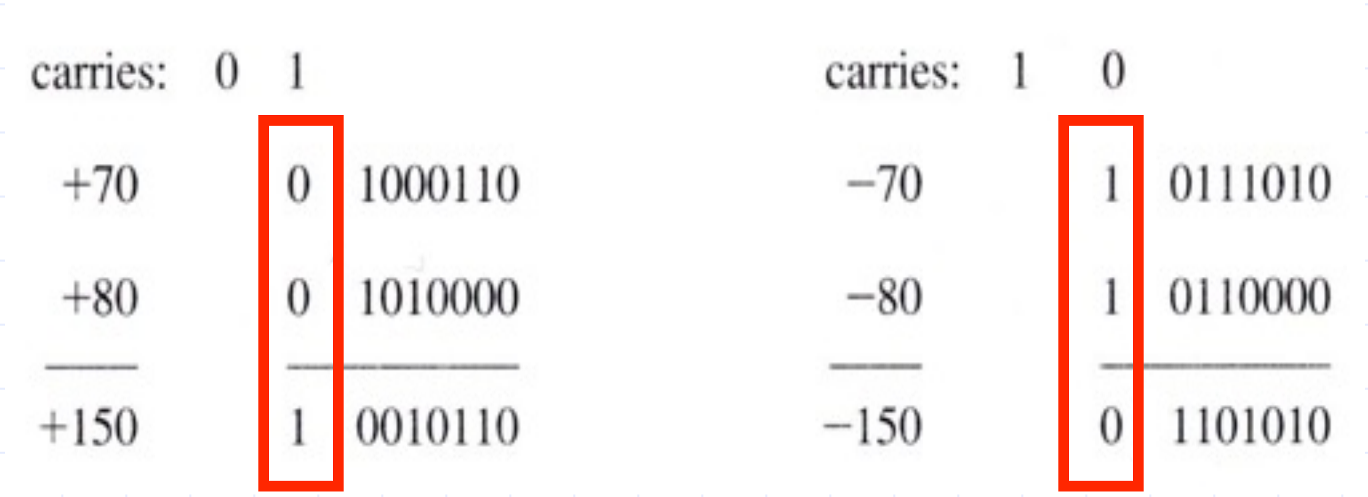
$A + \boxed{(\sim B) + 1}$

$x \oplus 0 = x,\ x \oplus 1 = x'$

Overflow

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

(examples of overflow)



```
carries:   0  1                    carries:   1   0

  +70         0  1000110             −70          1  0111010
  +80         0  1010000             −80          1  0110000
 -----        ----------            -----         ----------
 +150         1  0010110            −150          0  1101010
```

(양수)+(양수) => (음수)　　　　(음수)+(음수) => (양수)

4-5 Decimal adder

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

BCD Adder

- BCD digit output of 2-BCD digit sum
- Carry arise if output 1010~1111
- $C=K+Z_8Z_4+Z_8Z_2$

Z_8Z_4	Z_8Z_2
1100	1010
1101	1011
1110	
1111	

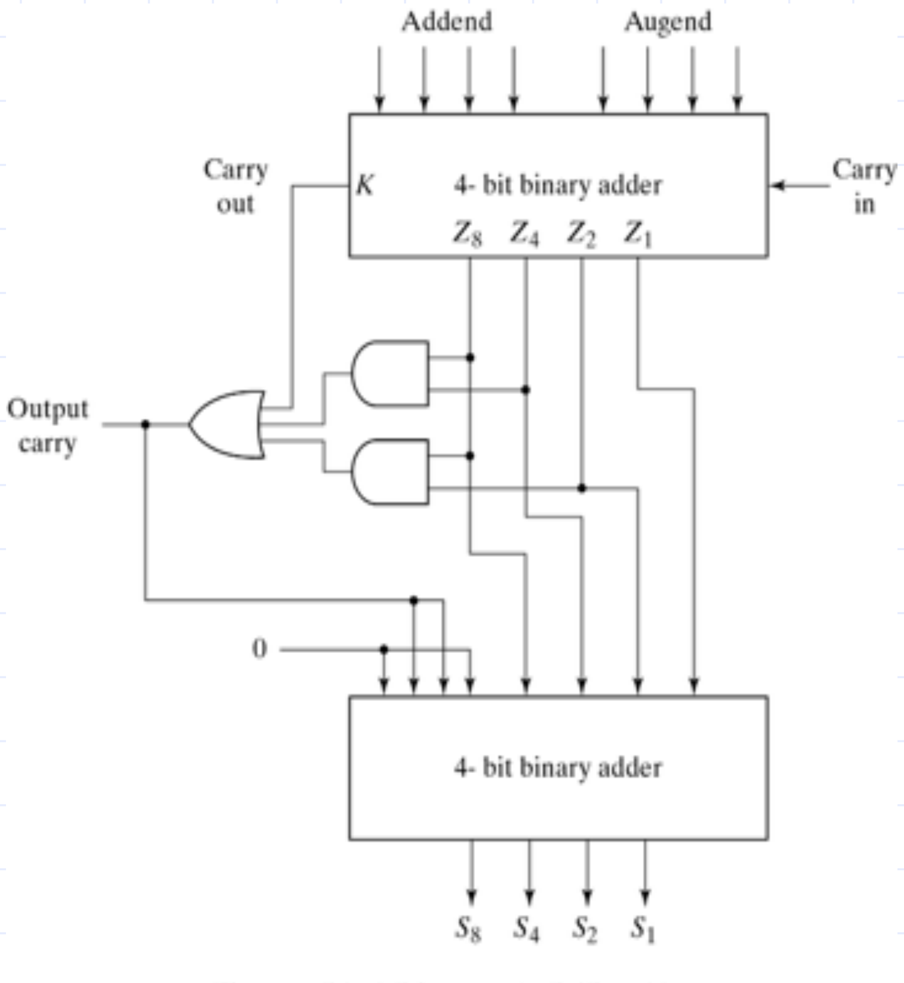


Fig. 4-14 Block Diagram of a BCD Adder

4-6 Binary multiplier

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

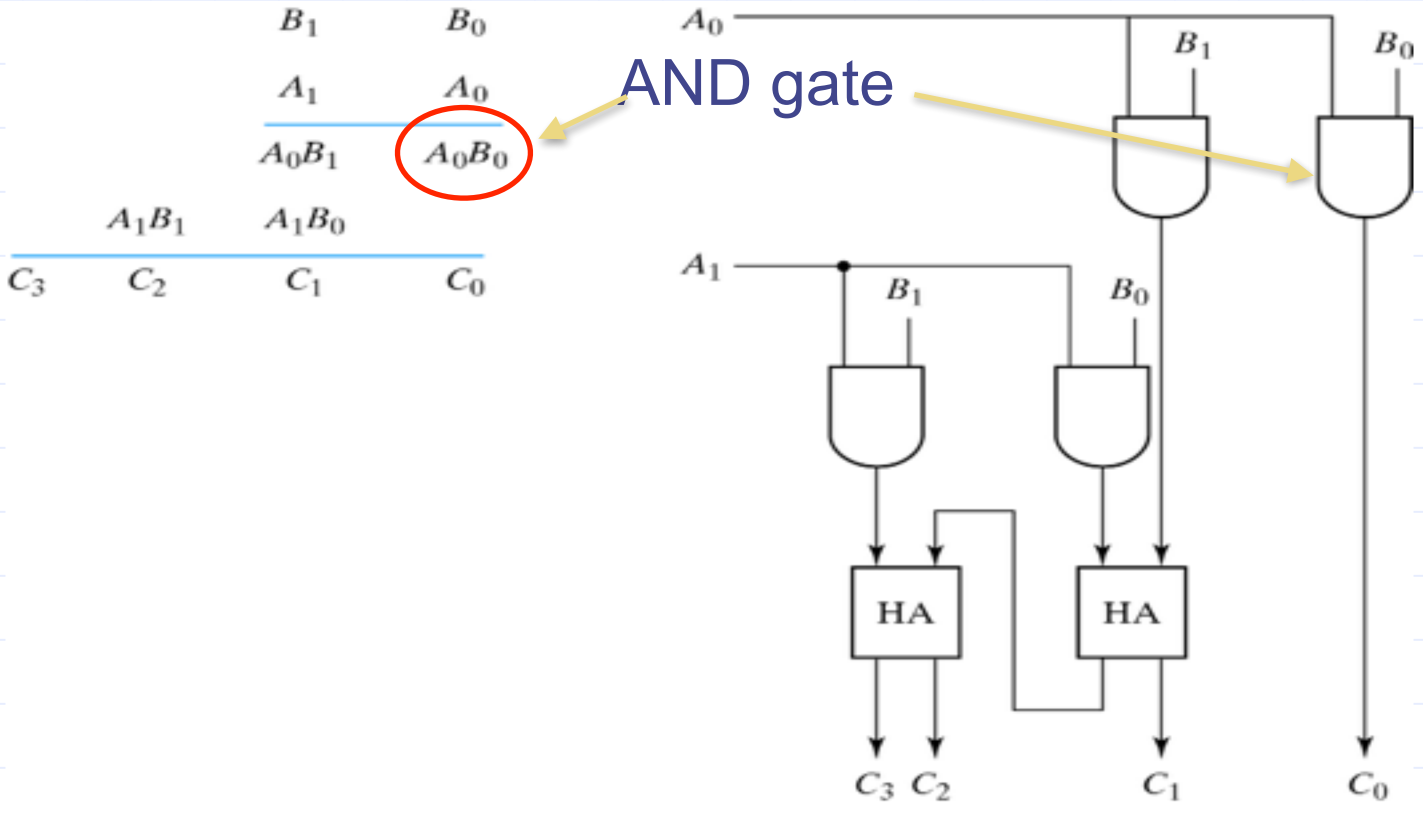


Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

Binary multiplier

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

$$
\begin{array}{cccccc}
 & & B_3 & B_2 & B_1 & B_0 \\
 & & \times & A_2 & A_1 & A_0 \\
\hline
 & & A_0B_3 & A_0B_2 & A_0B_1 & A_0B_0 \\
 & A_1B_3 & A_1B_2 & A_1B_1 & A_1B_0 & \\
A_2B_3 & A_2B_2 & A_2B_1 & A_2B_0 & & \\
\hline
\end{array}
$$

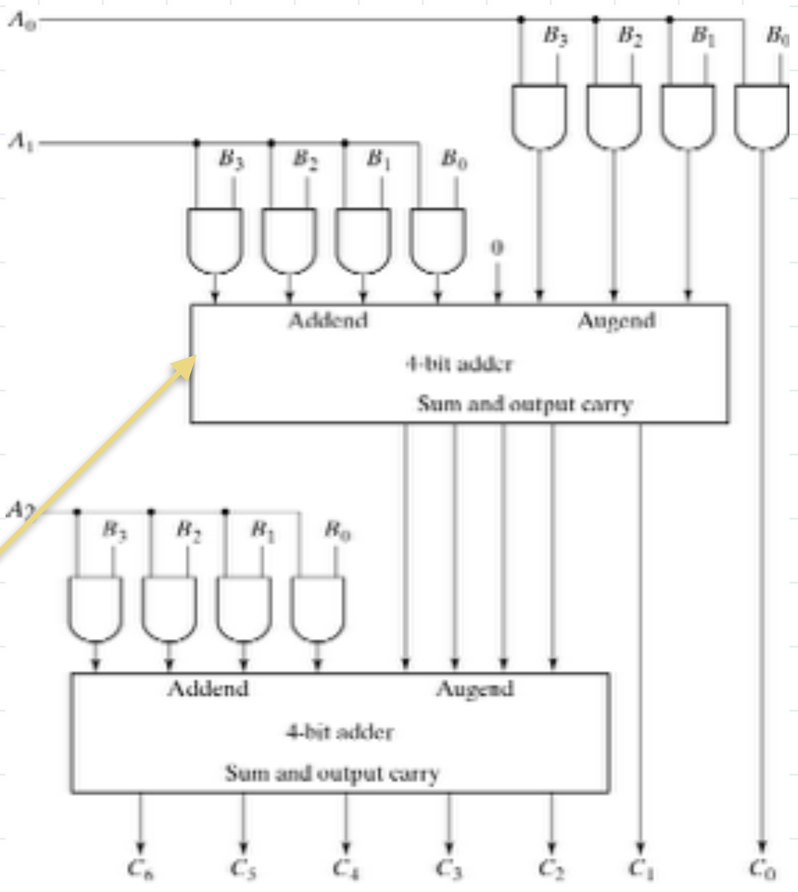


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

4-7 Magnitude comparator

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

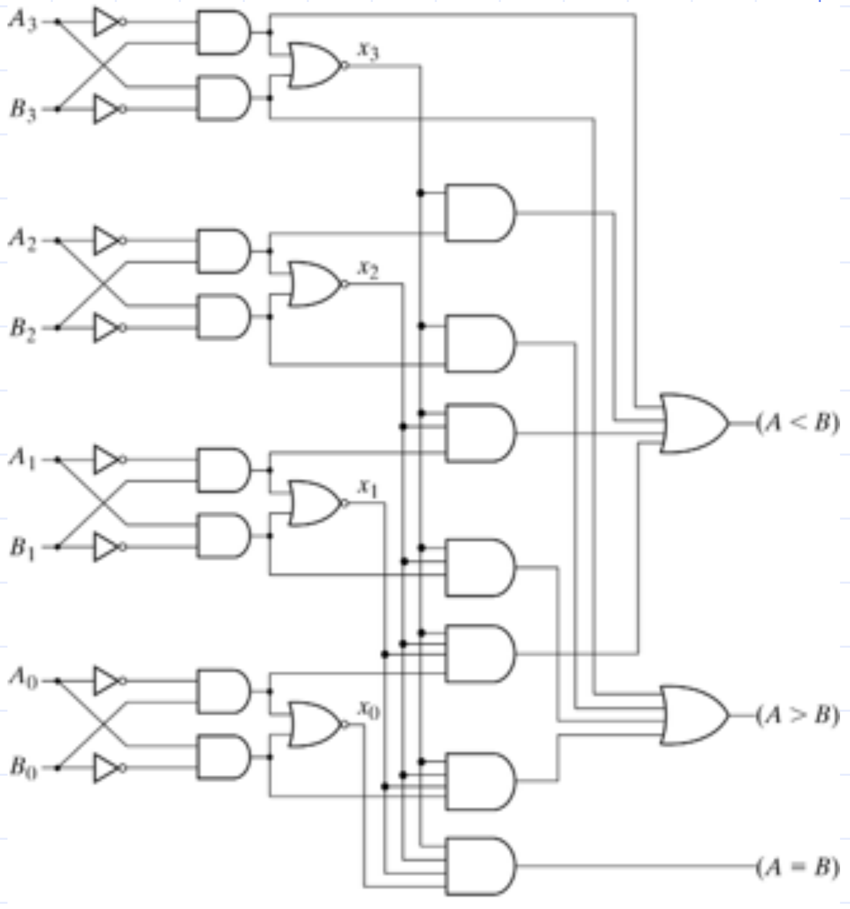


Fig. 4-17 4-Bit Magnitude Comparator

4-8 Decoders

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

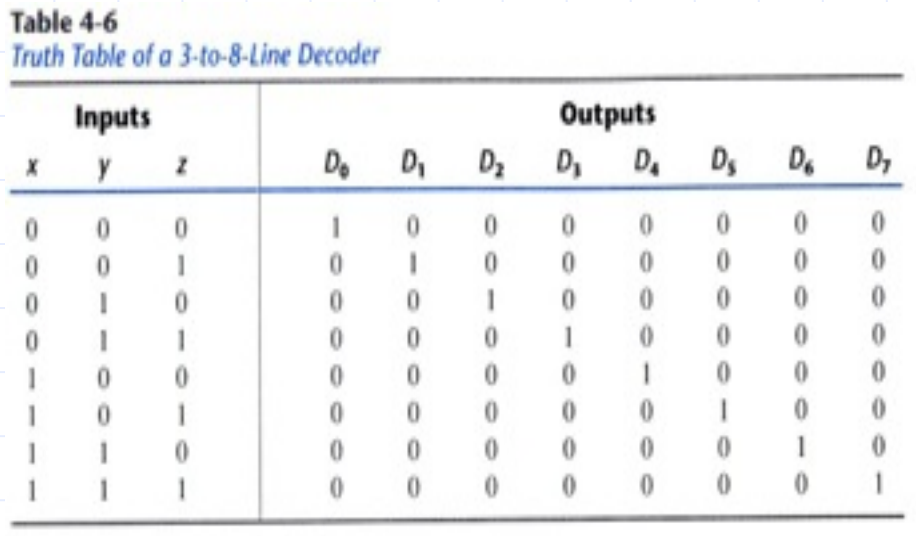


Table 4-6

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

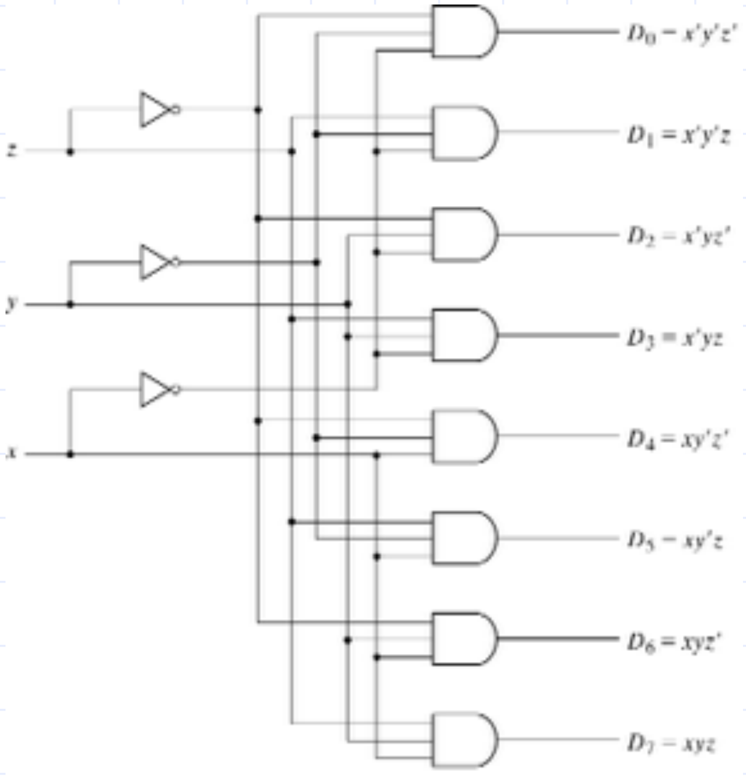
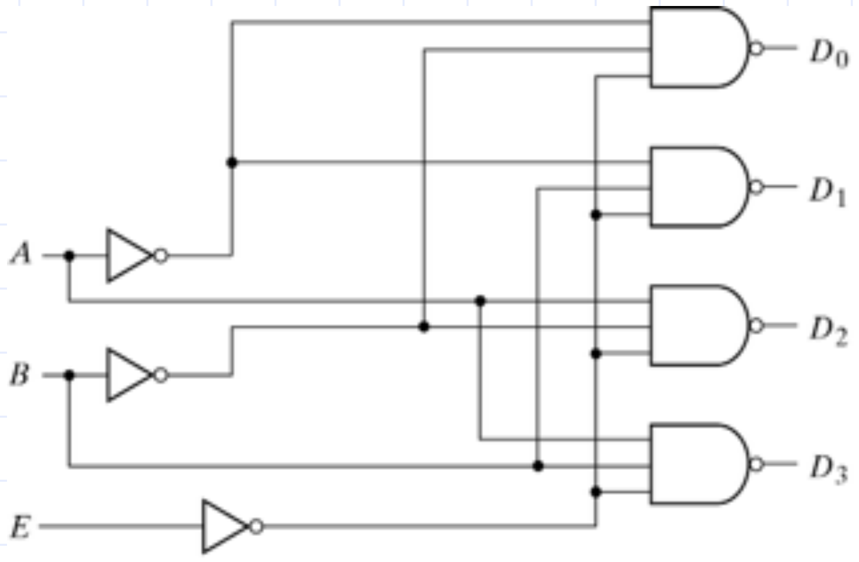


Fig. 4-18 3-to-8-Line Decoder

Decoders

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



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	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

Decoders

- Decoders with enable inputs can be a larger decoder circuit

Eg) 4x16 decoder by two 3x8 decoders

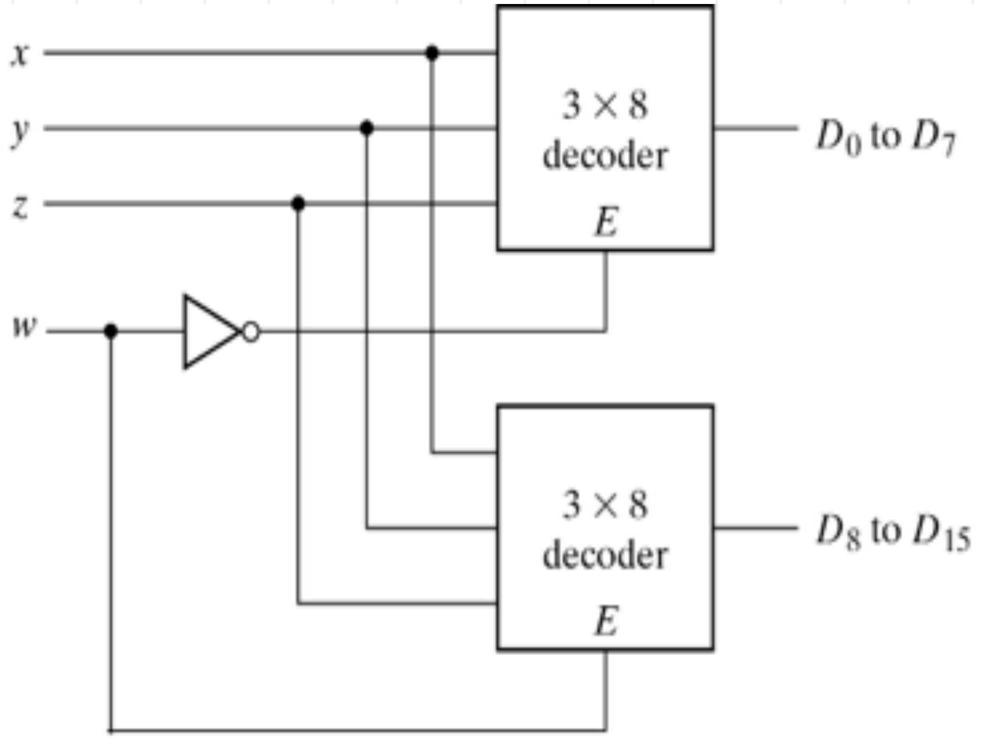


Fig. 4-20 4 × 16 Decoder Constructed with Two 3 × 8 Decoders

Decoders

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

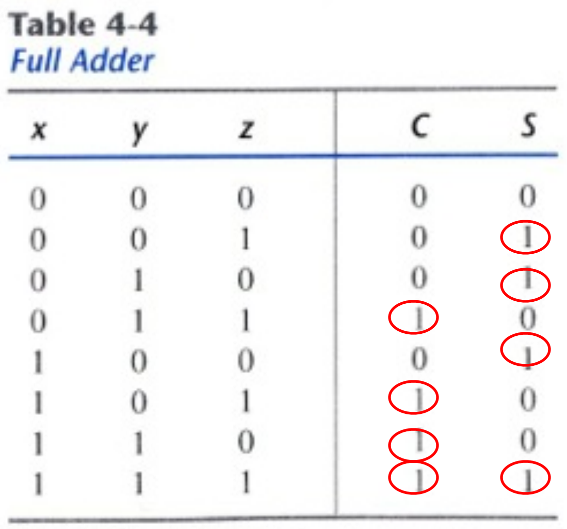


Table 4-4
Full Adder

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

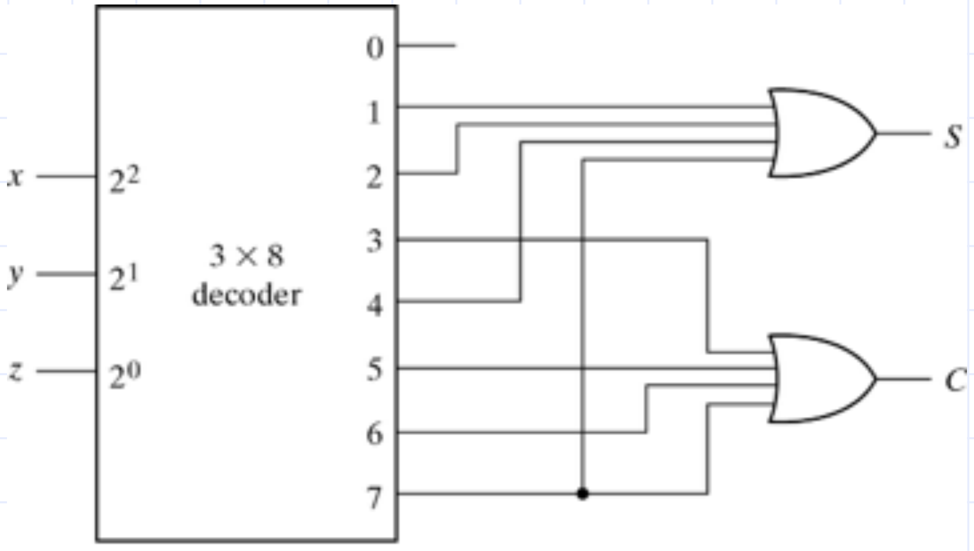


Fig. 4-21 Implementation of a Full Adder with a Decoder

4-9 Encoders

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

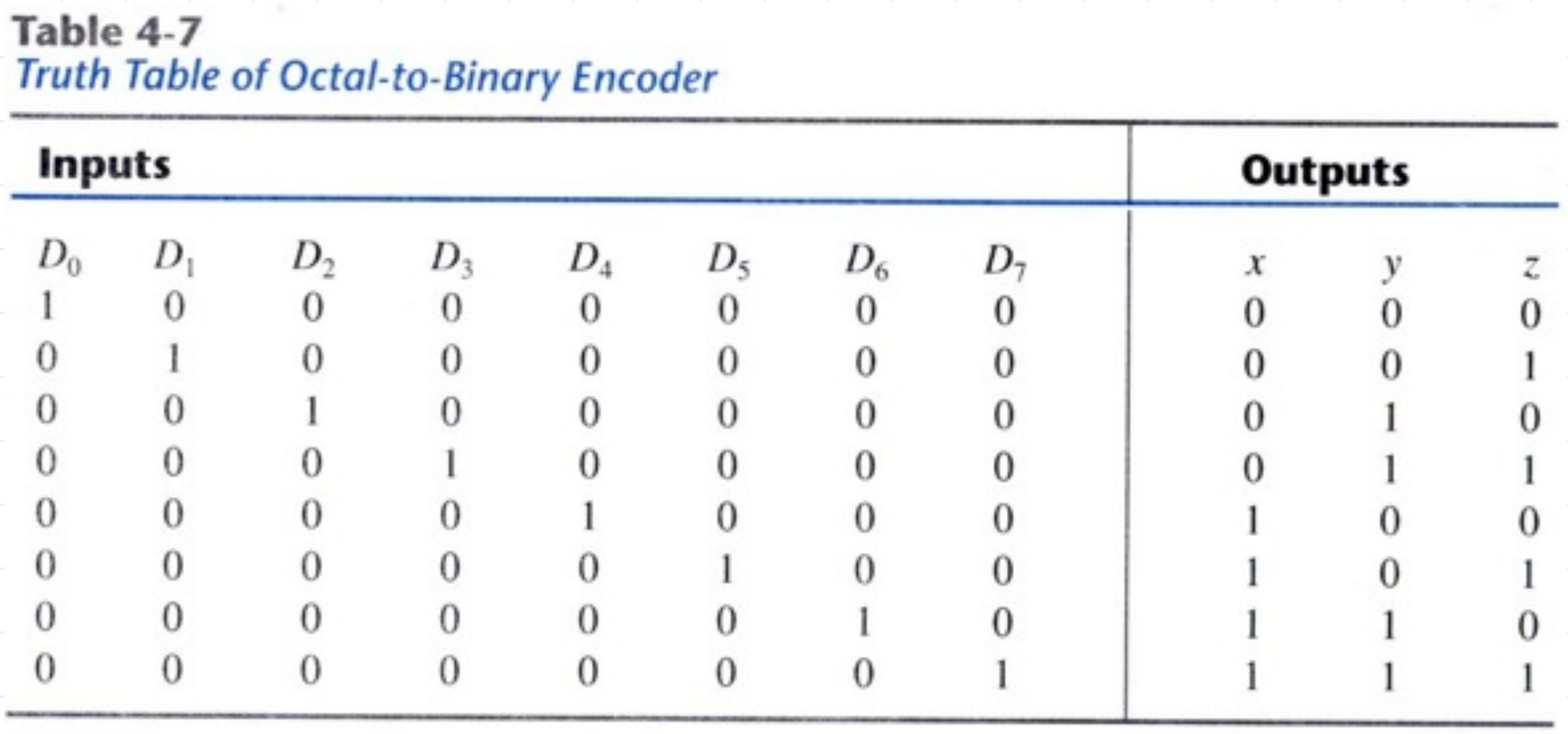


Table 4-7
Truth Table of Octal-to-Binary Encoder

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

Priority encoder

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

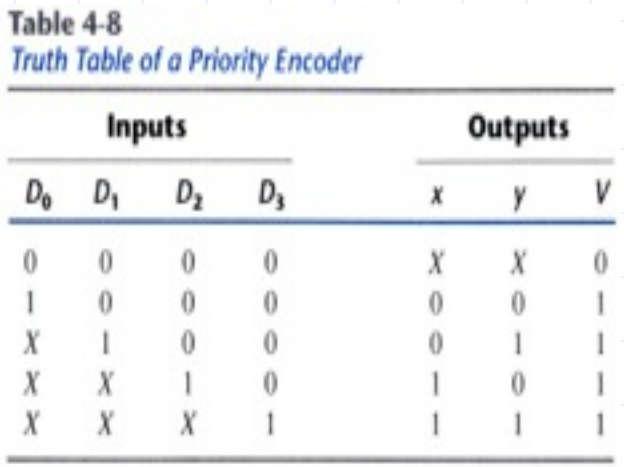


Table 4-8
Truth Table of a Priority Encoder

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)

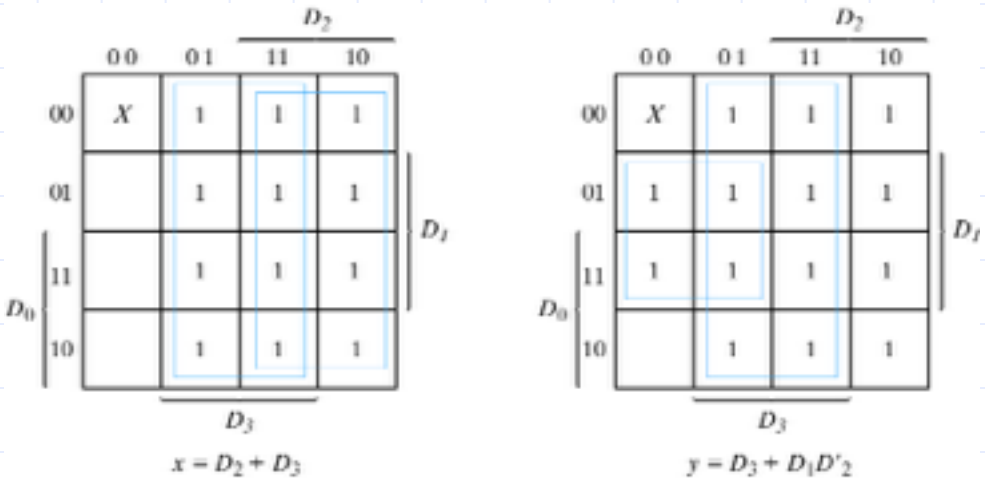


Fig. 4-22 Maps for a Priority Encoder

4-10 Multiplexers

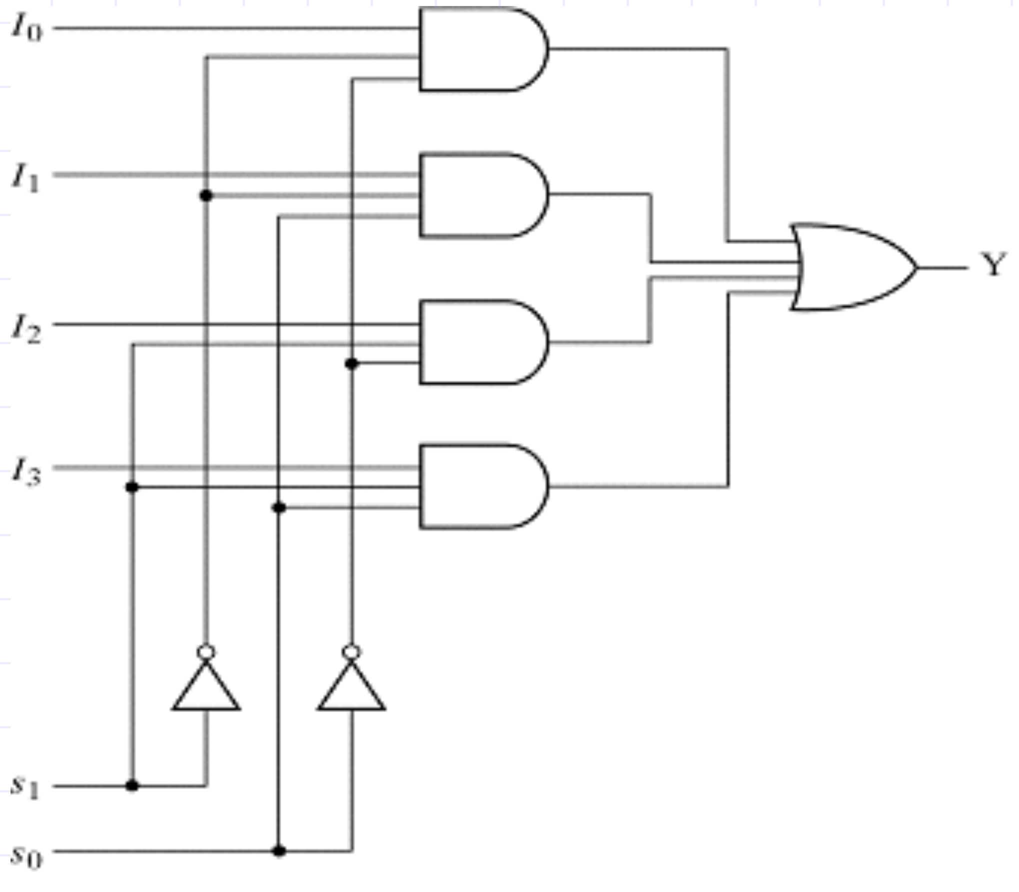
- Select a binary information from many input lines
- Selection is controlled by a set of selection lines
- 2^n input lines have n selection lines

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

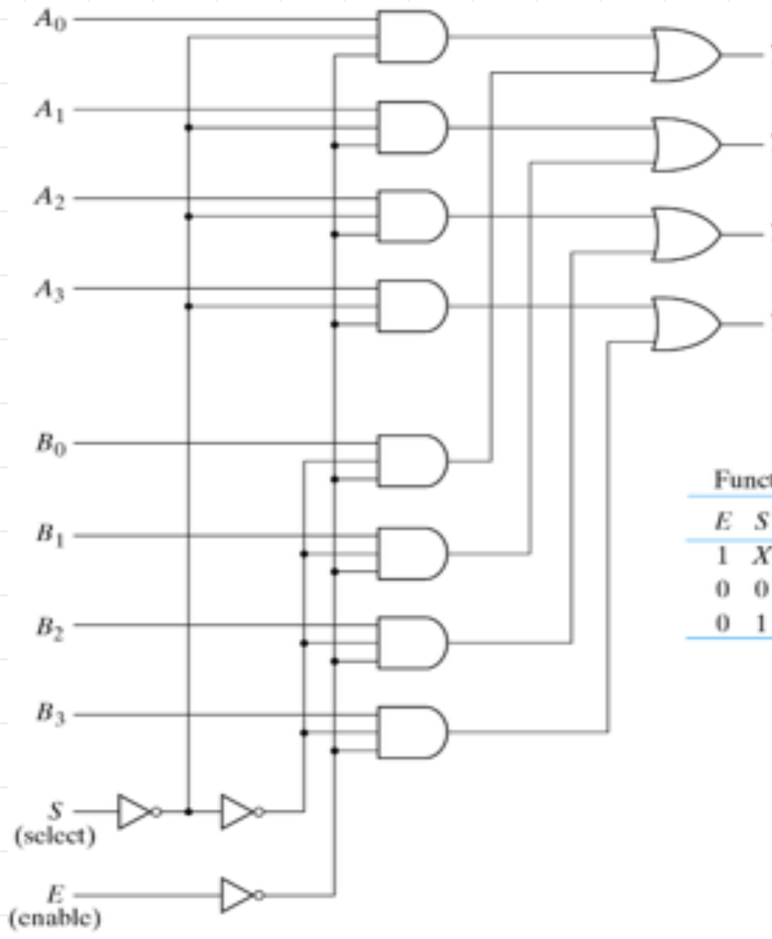
(b) Function table



(a) Logic diagram

Multiplexers

- Quadruple 2 to 1 line multiplexer



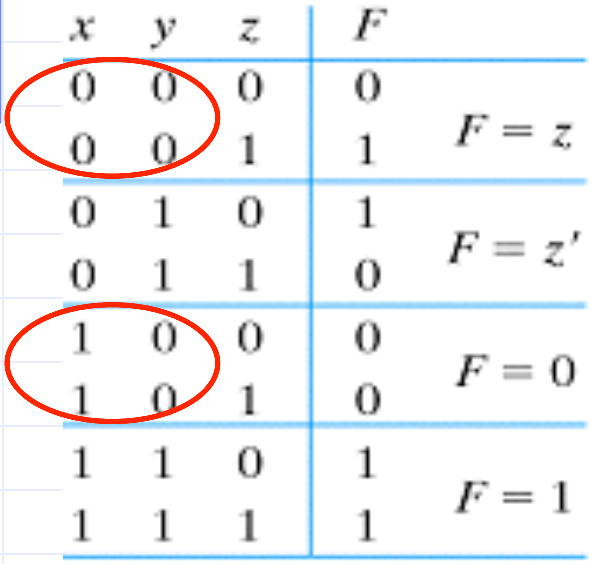
Function table

E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

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

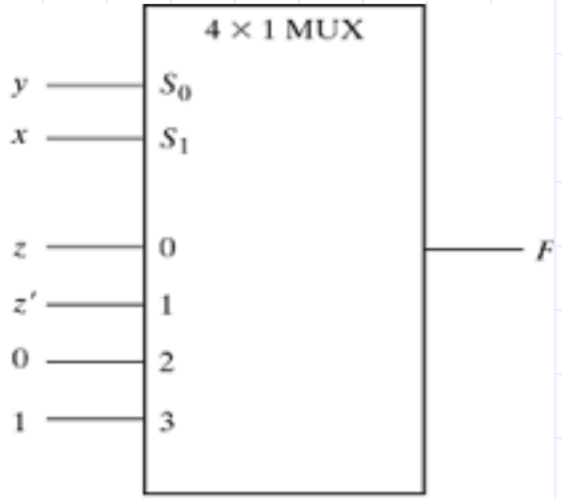
Multiplexers

- 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	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

F=xy+yz′+x′y′z

Multiplexers

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

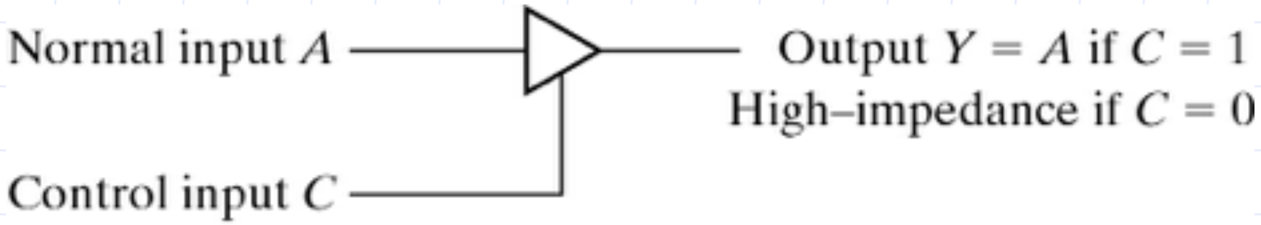
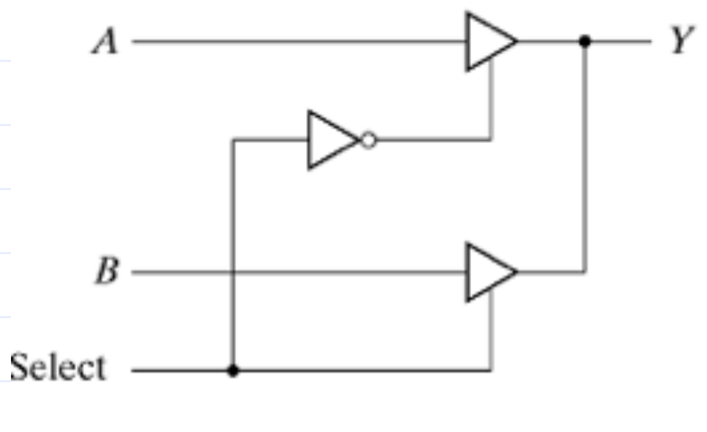


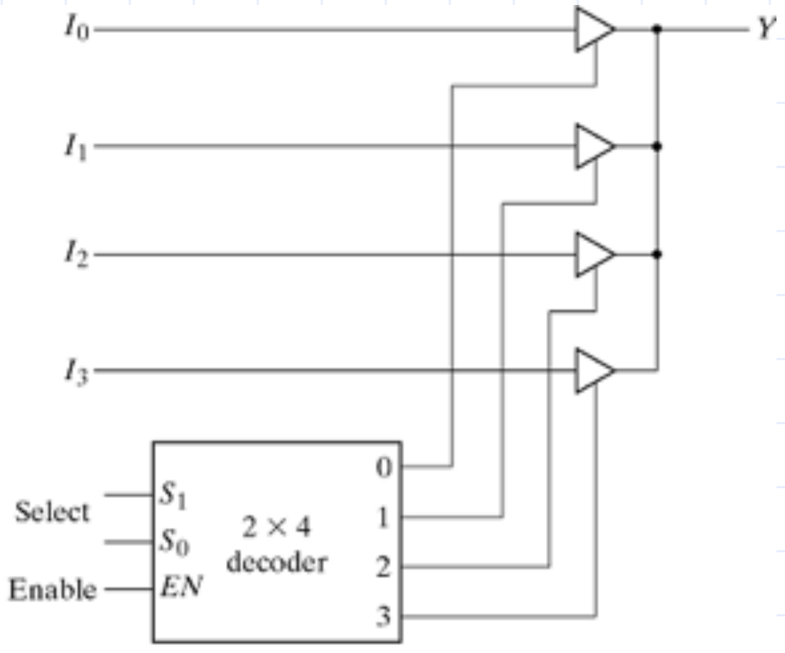
Fig. 4-29 Graphic Symbol for a Three-State Buffer

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

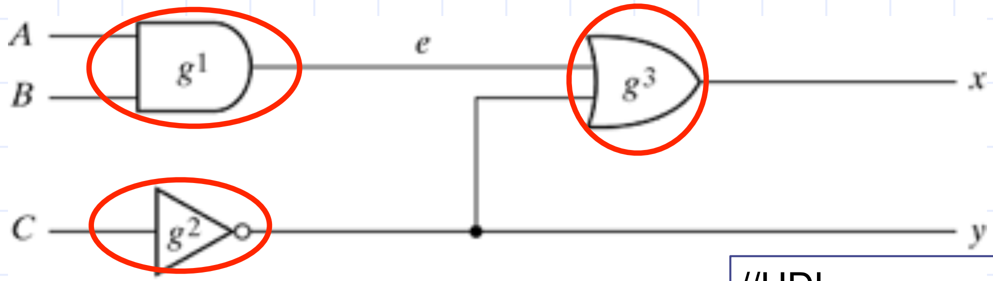
Verilog HDL

HDL(Hardware Description Language)

- HDL: 하드웨어의 설계에 사용되는 언어
- Verilog HDL은 하드웨어를 모듈 단위로 설계하며, 다양한 방법으로 하드웨어를 기술
- 현재 대부분의 디지털 하드웨어는 HDL을 사용하여 설계됨

HDL(Hardware Description Language)

- VHDL, Verilog HDL
- Module Representation



```
//HDL
module smpl_circuit(A,B,C,x,y);
   input A,B,C;
   output x,y;
   wire e;
   and g1(e,A,B);
   not g2(y, C);
   or  g3(x,e,y);
endmodule
```

HDL(Hardware Description Language)

- Gate Delays - `timescale 1ns/100ps

```
//Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
   input A,B,C;
   output x,y;
   wire e;
   and #(30) g1(e,A,B);
   or  #(20) g3(x,e,y);
   not #(10) g2(y,C);
endmodule
```

```
//Stimulus for simple circuit
module stimcrct;
reg A,B,C;
wire x,y;
circuit_with_delay cwd(A,B,C,x,y);
initial
   begin
        A = 1'b0; B = 1'b0; C = 1'b0;
     #100
        A = 1'b1; B = 1'b1; C = 1'b1;
     #100  $finish;
   end
endmodule
```

HDL(Hardware Description Language)

- Boolean Expressions

  - AND, OR, NOT는 &, |, ~로 표시

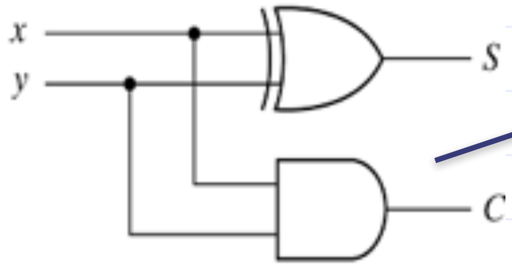
```
assign x = (A & B) | ~C);  // (A and B) or (not C)
```

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

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
```



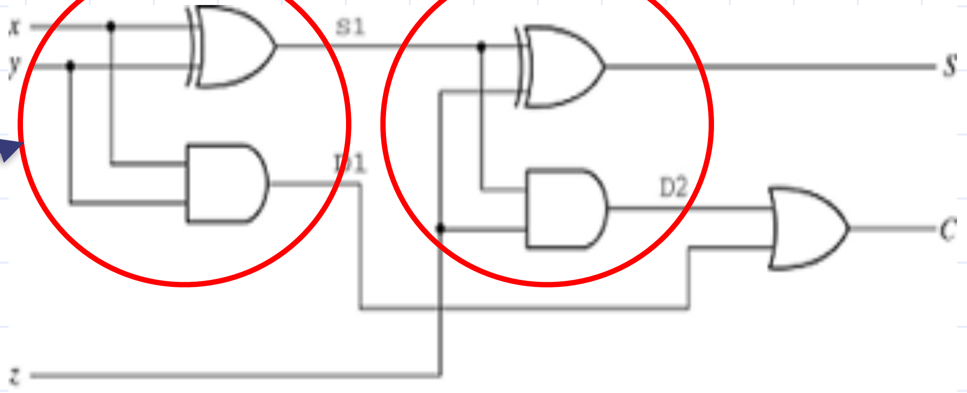


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

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
```

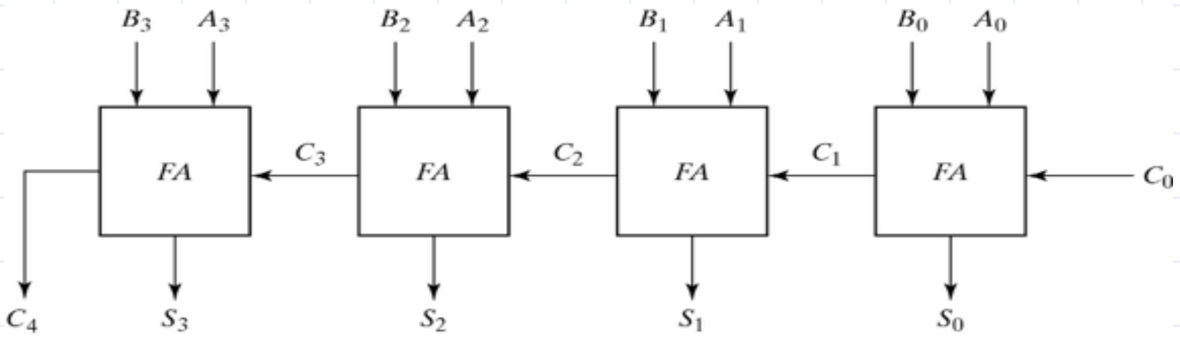


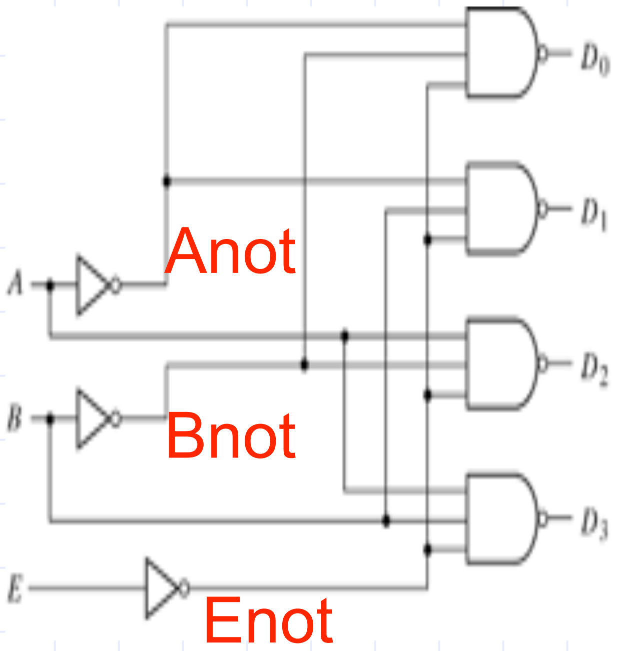
Fig. 4-9 4-Bit Adder

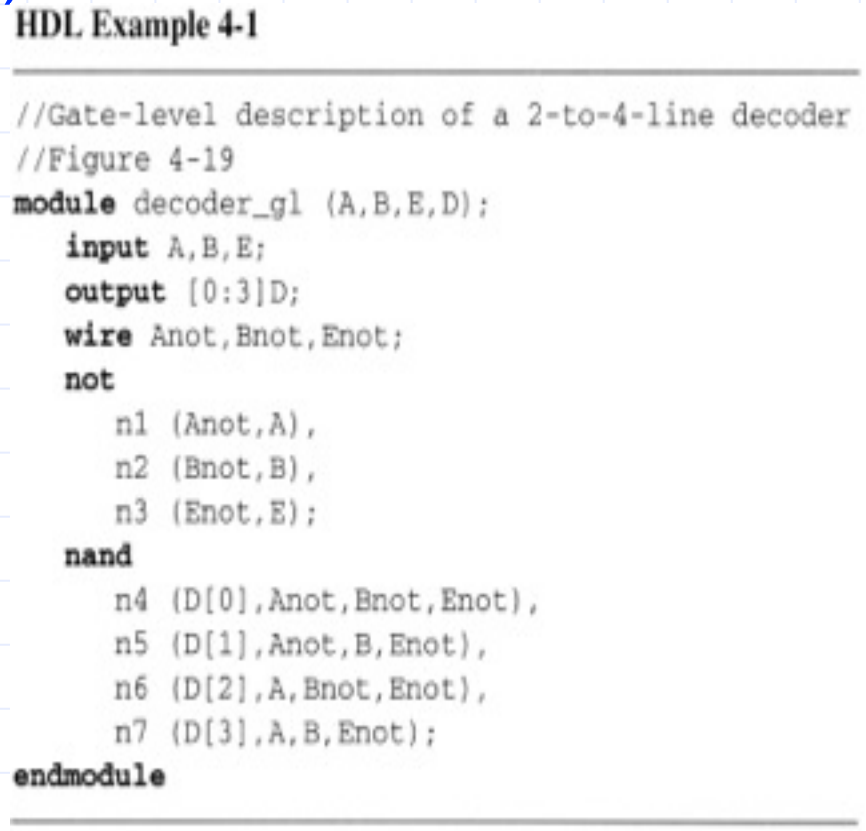
4-11 HDL for combinational circuit

- 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

Gate-level modeling

- Circuit is specified by its gates and their interconnection(연결)





HDL Example 4-1

```
//Gate-level description of a 2-to-4-line decoder
//Figure 4-19
module decoder_gl (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
```

Dataflow modeling

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

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

  out=x ? A : B means
  out=A, if x is true
     =B, if x is false

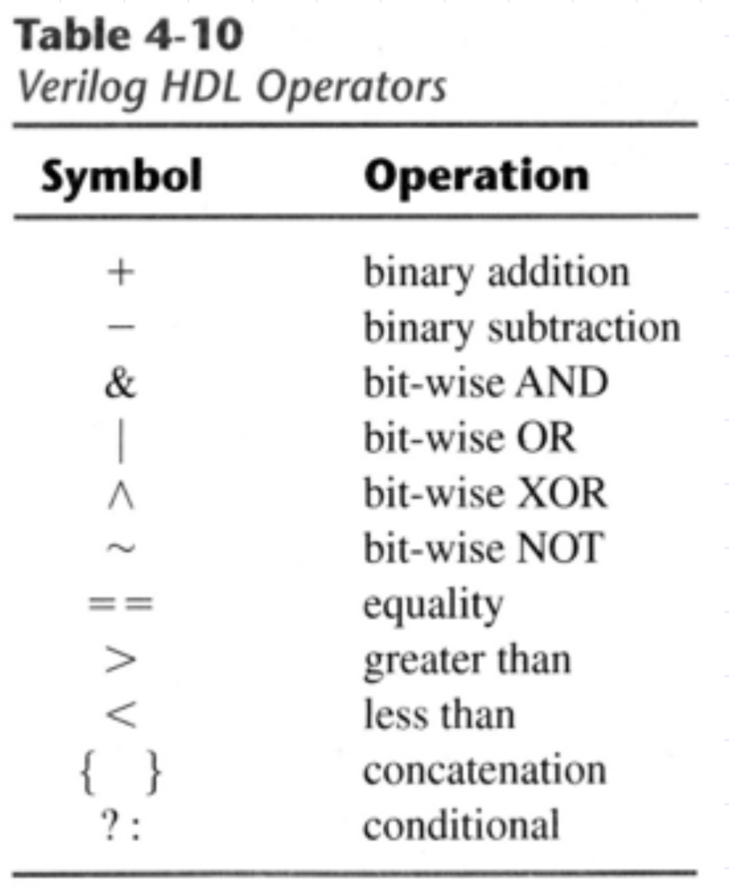
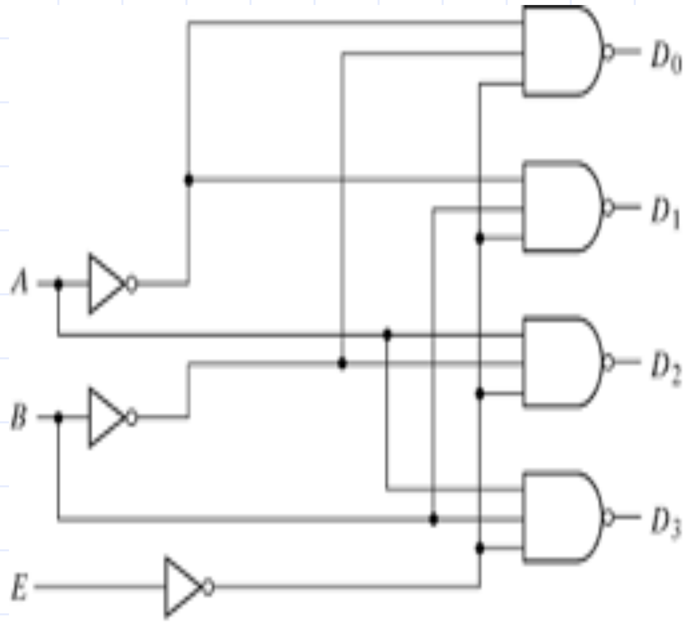


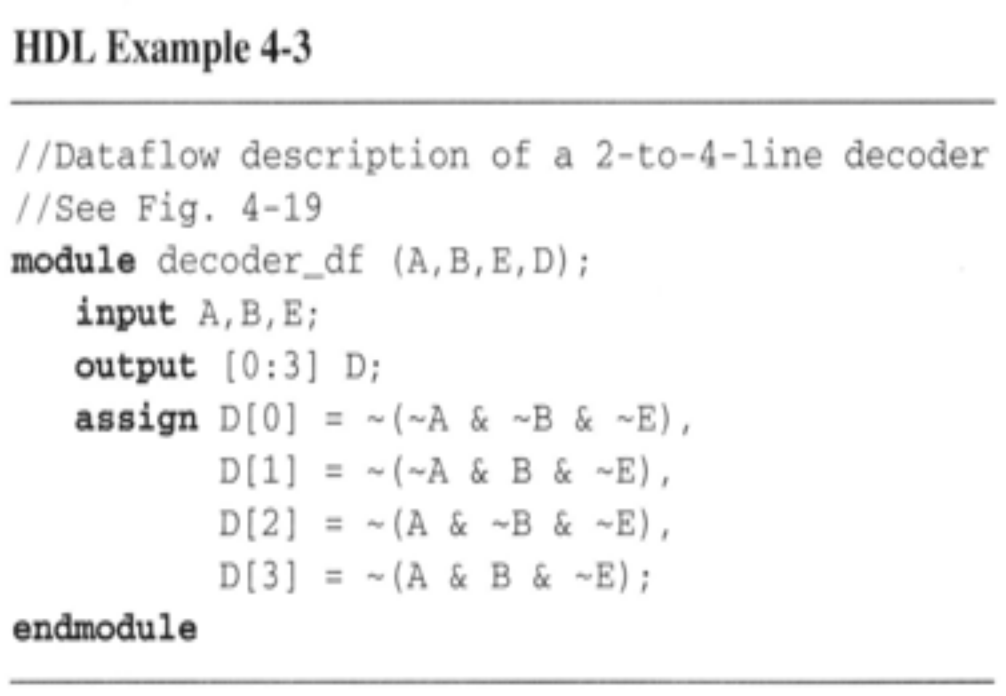
Table 4-10
Verilog HDL Operators

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

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-bit adder

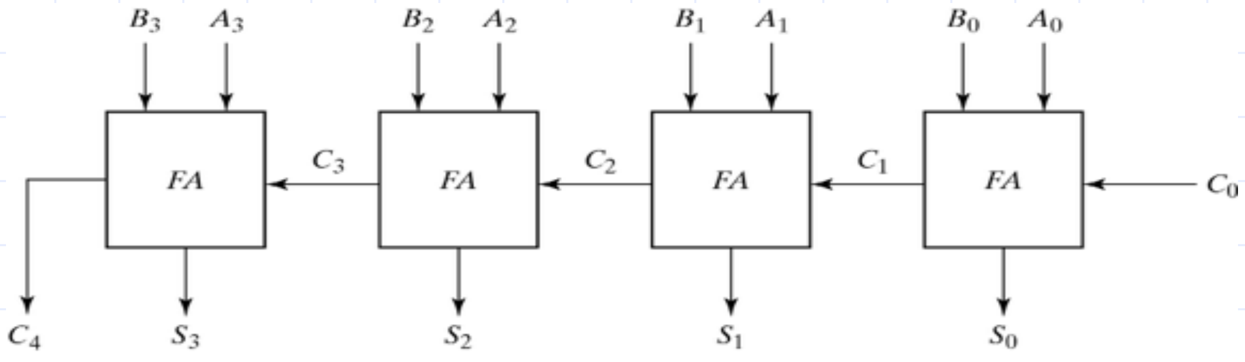


Fig. 4-9 4-Bit Adder

HDL Example 4-4

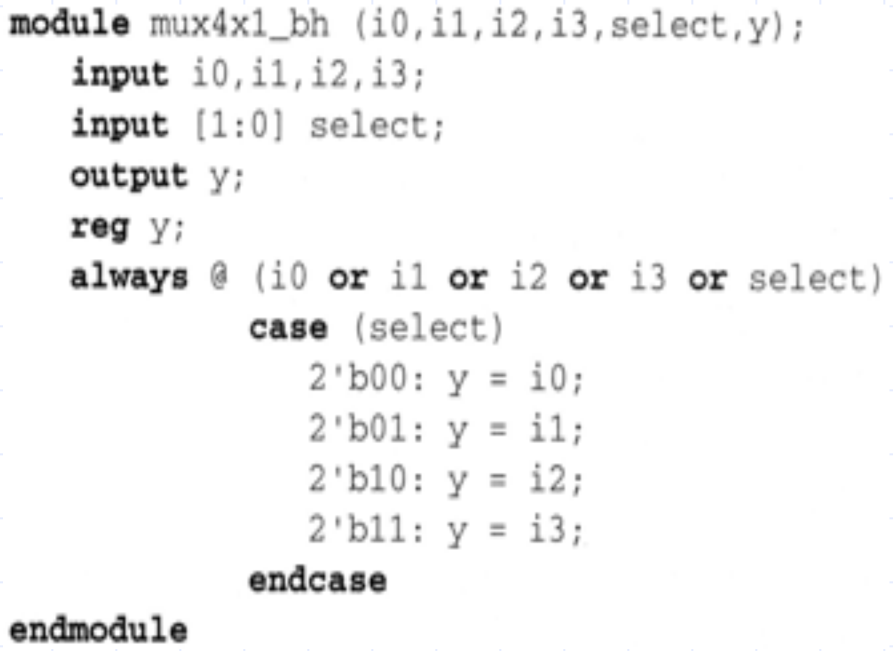
```
//Dataflow description of 4-bit adder
module binary_adder (A,B,Cin,SUM,Cout);
   input [3:0] A,B;
   input Cin;
   output [3:0] SUM;
   output Cout;
   assign {Cout,SUM} = A + B + Cin;
endmodule
```

Behavioral modeling

- Use procedural assignment statement,always

  - Target output must be the reg data type

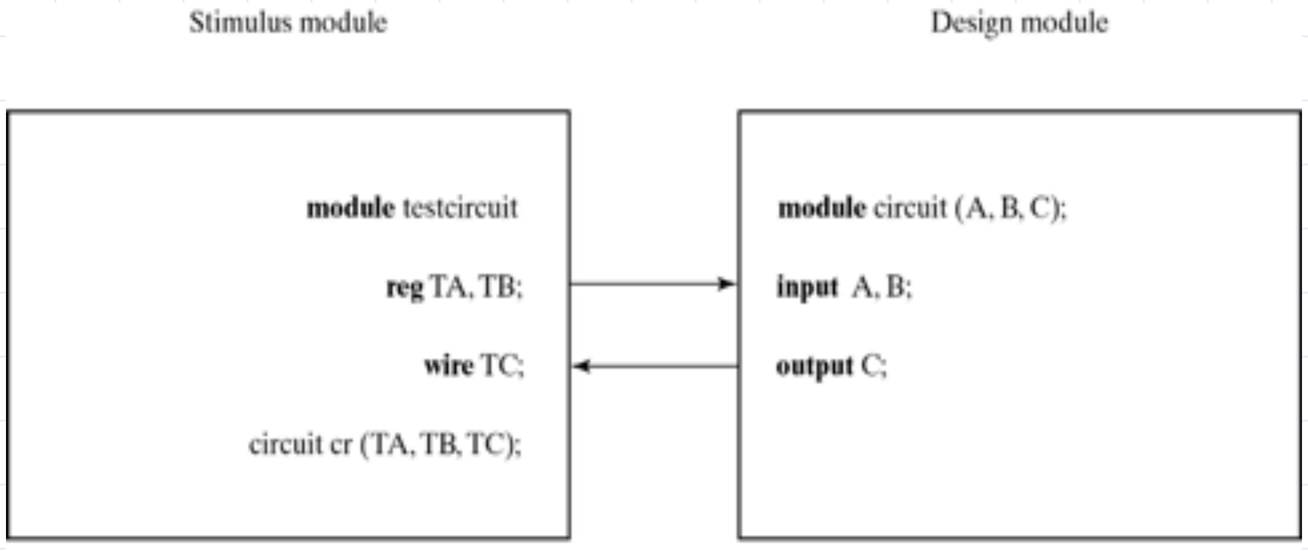
  Eg) 4 to 1 line mux



```
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
```

Writing a simple test bench

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



```
module testcircuit
reg TA, TB;
wire TC;
circuit cr (TA, TB, TC);
```

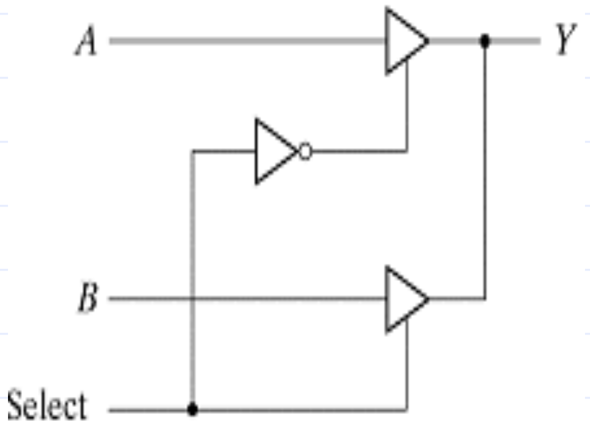
```
module circuit (A, B, C);
input A, B;
output C;
```

Fig. 4-33 Stimulus and Design Modules Interaction

System tasks

- 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);

Example of testbench



```
//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
```