

Fig. 3-1 Block Diagram of Combinational Circuit

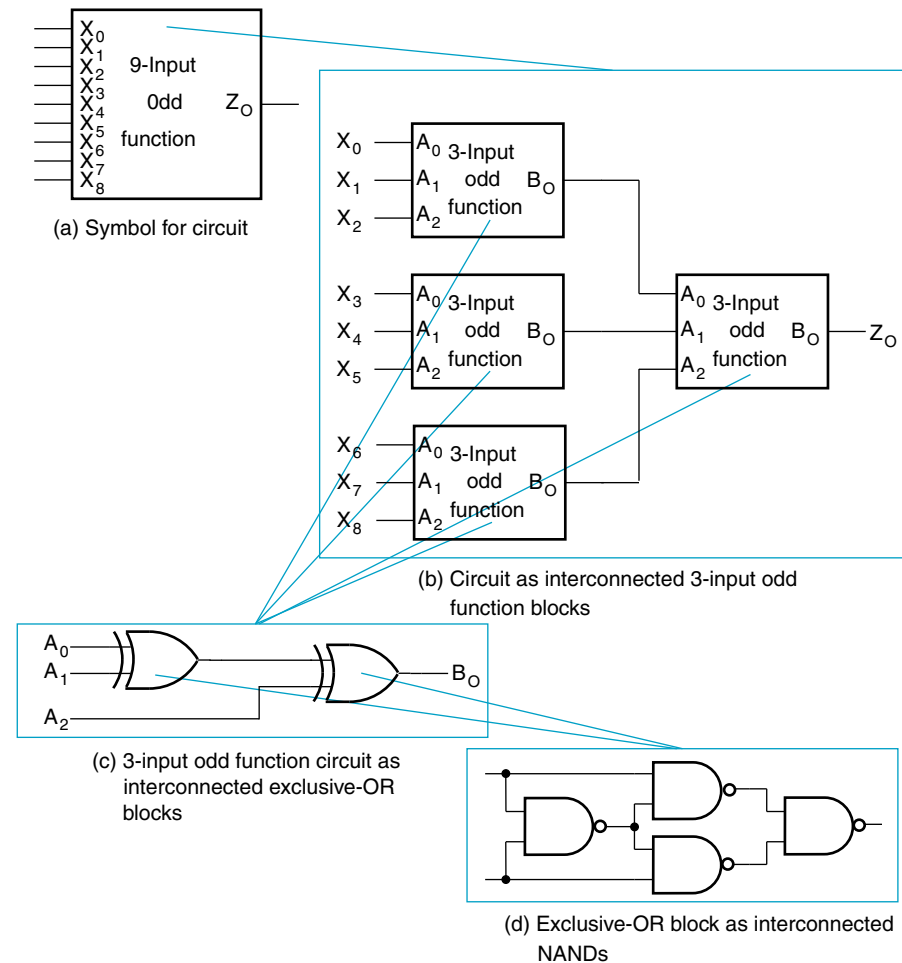


Fig. 3-2 Example of Design Hierarchy and Reusable Blocks

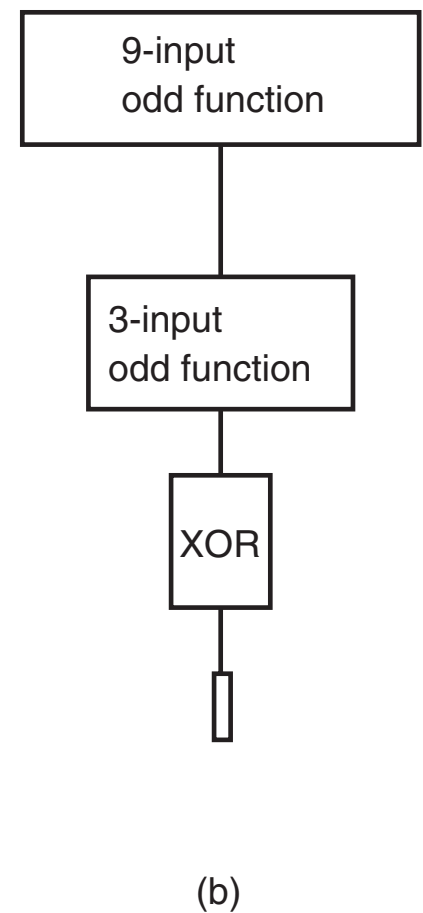
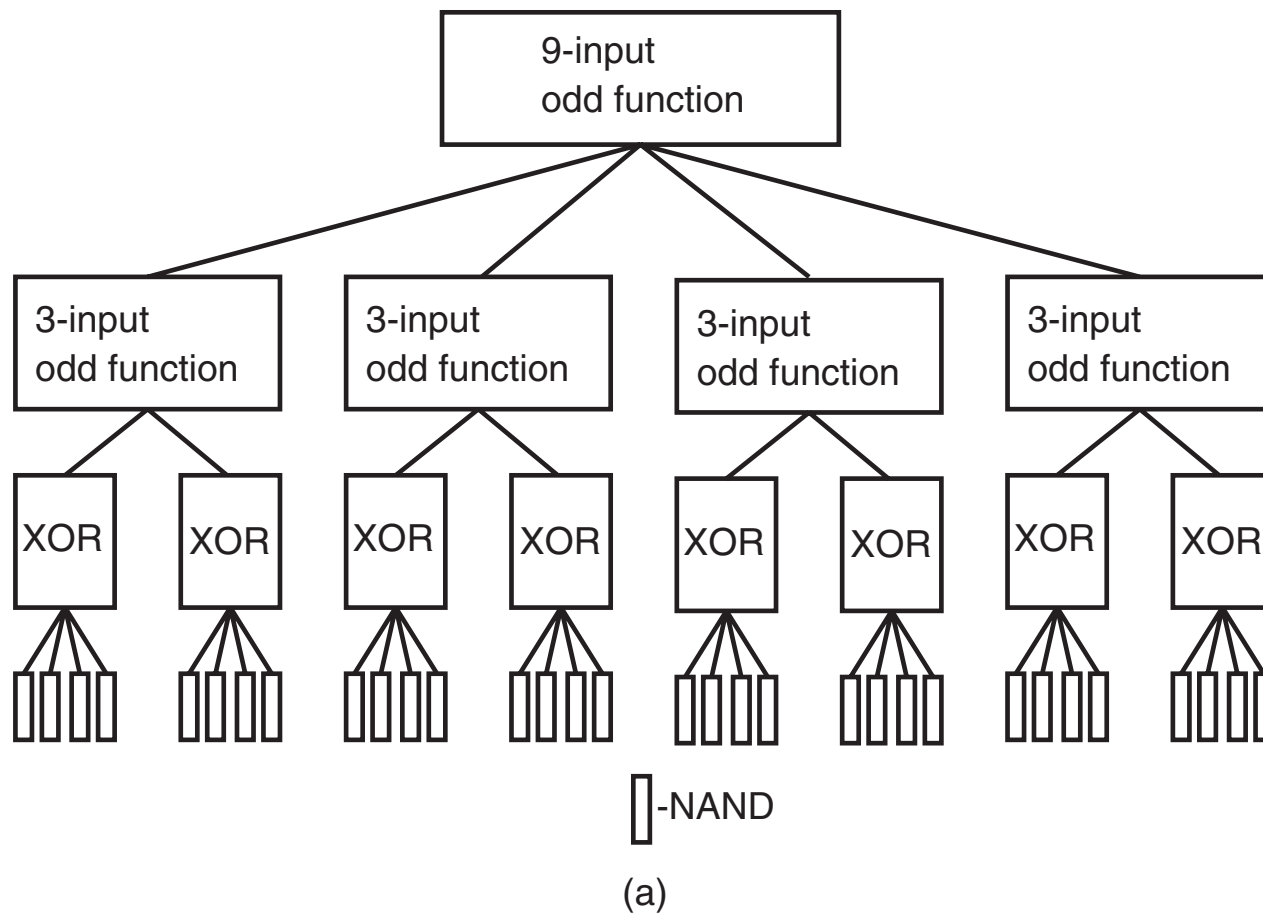


Fig. 3-3 Diagrams Representing the Hierarchy for Figure 3-2

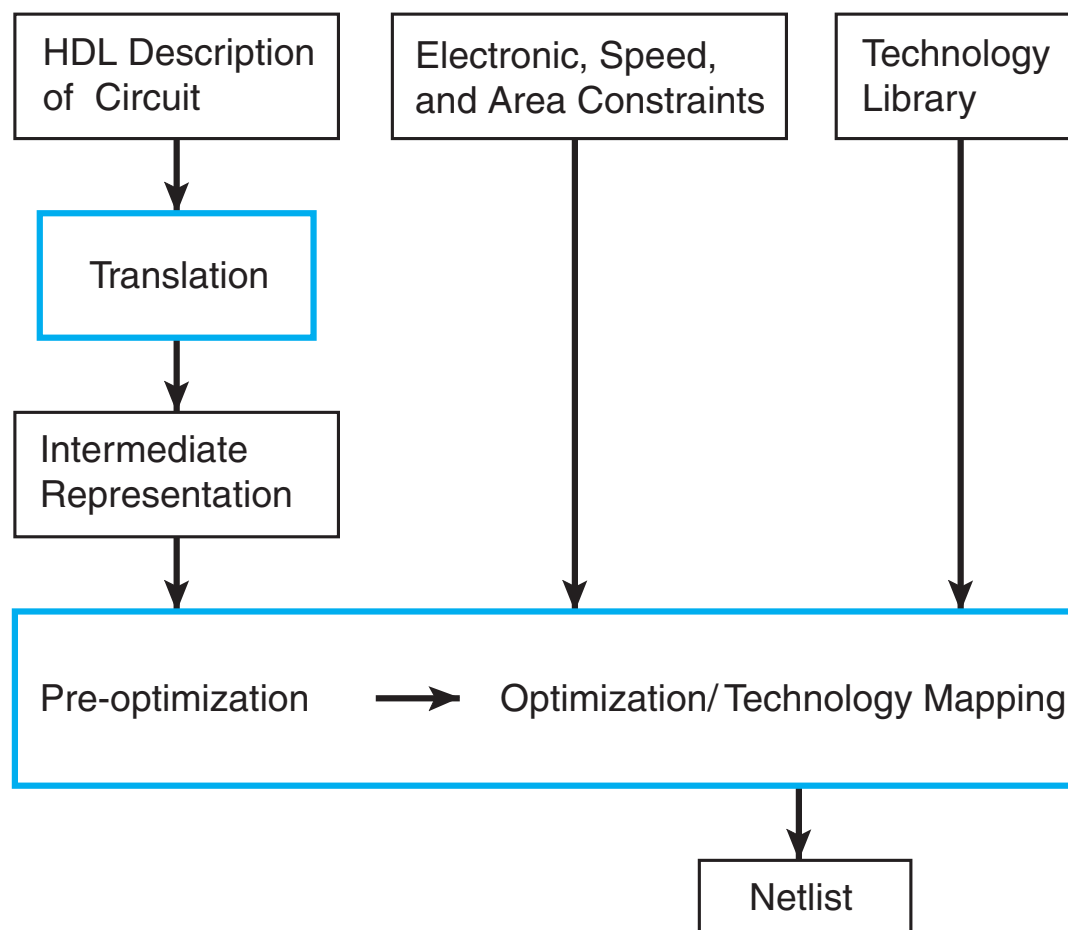


Fig. 3-4 High-Level Flow for Synthesis Tool

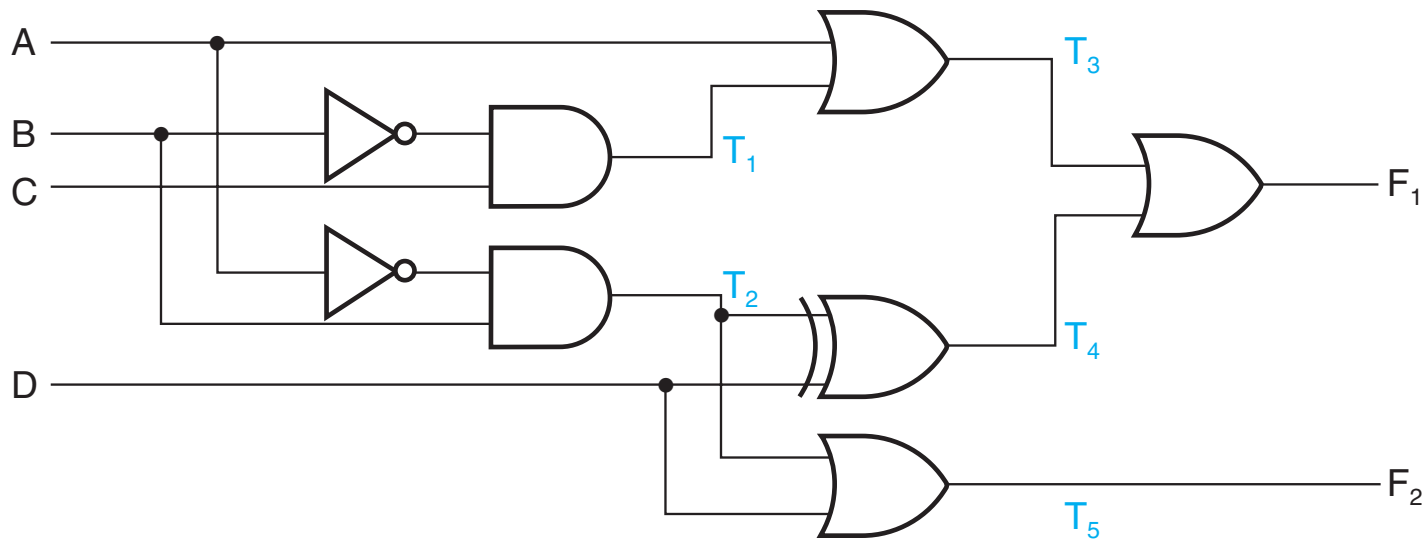


Fig. 3-5 Logic Diagram for Analysis Example

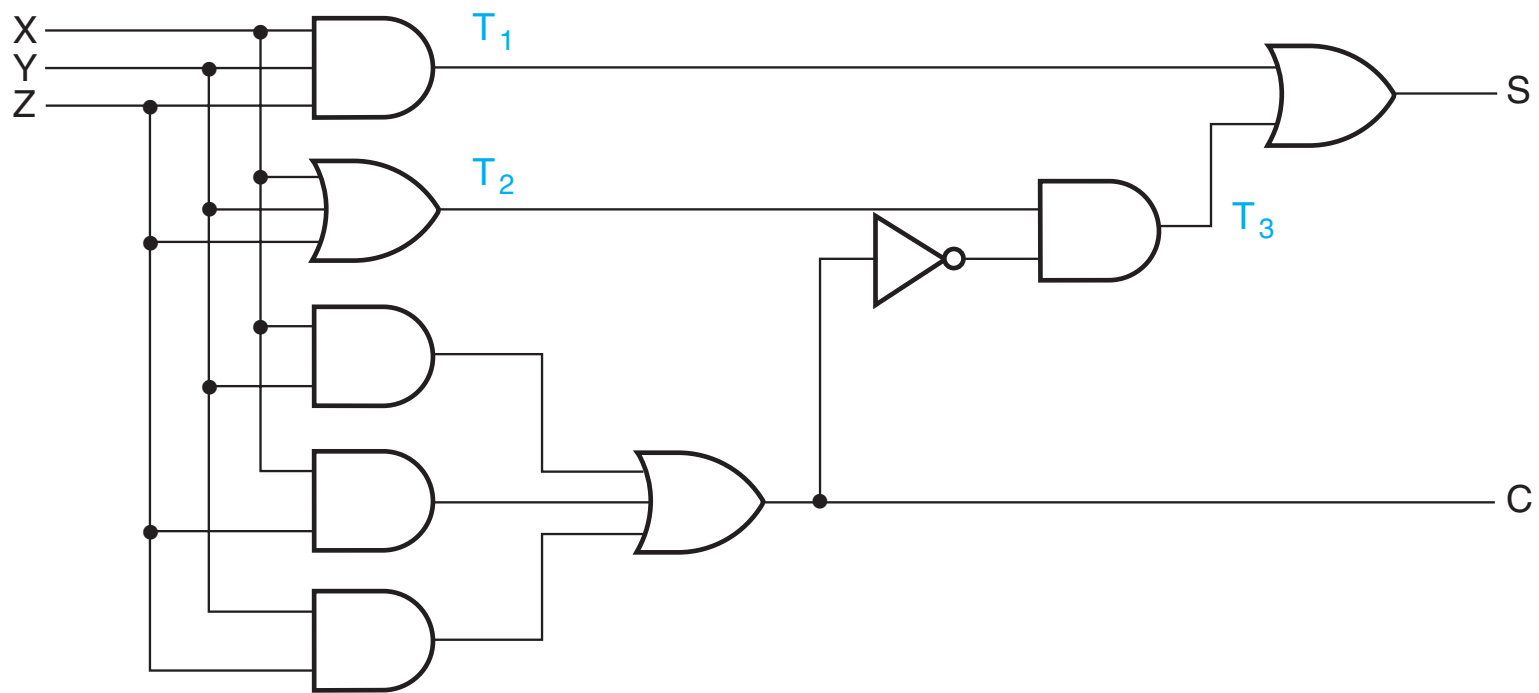


Fig. 3-6 Logic Diagram for Binary Adder

TABLE 3-1
Truth Table for Binary Adder

X	Y	Z	C	\bar{C}	T₁	T₂	T₃	S
0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	1	1	1
0	1	0	0	1	0	1	1	1
0	1	1	1	0	0	1	0	0
1	0	0	0	1	0	1	1	1
1	0	1	1	0	0	1	0	0
1	1	0	1	0	0	1	0	0
1	1	1	1	0	1	1	0	1

Table 3-1 Truth Table for Binary Adder

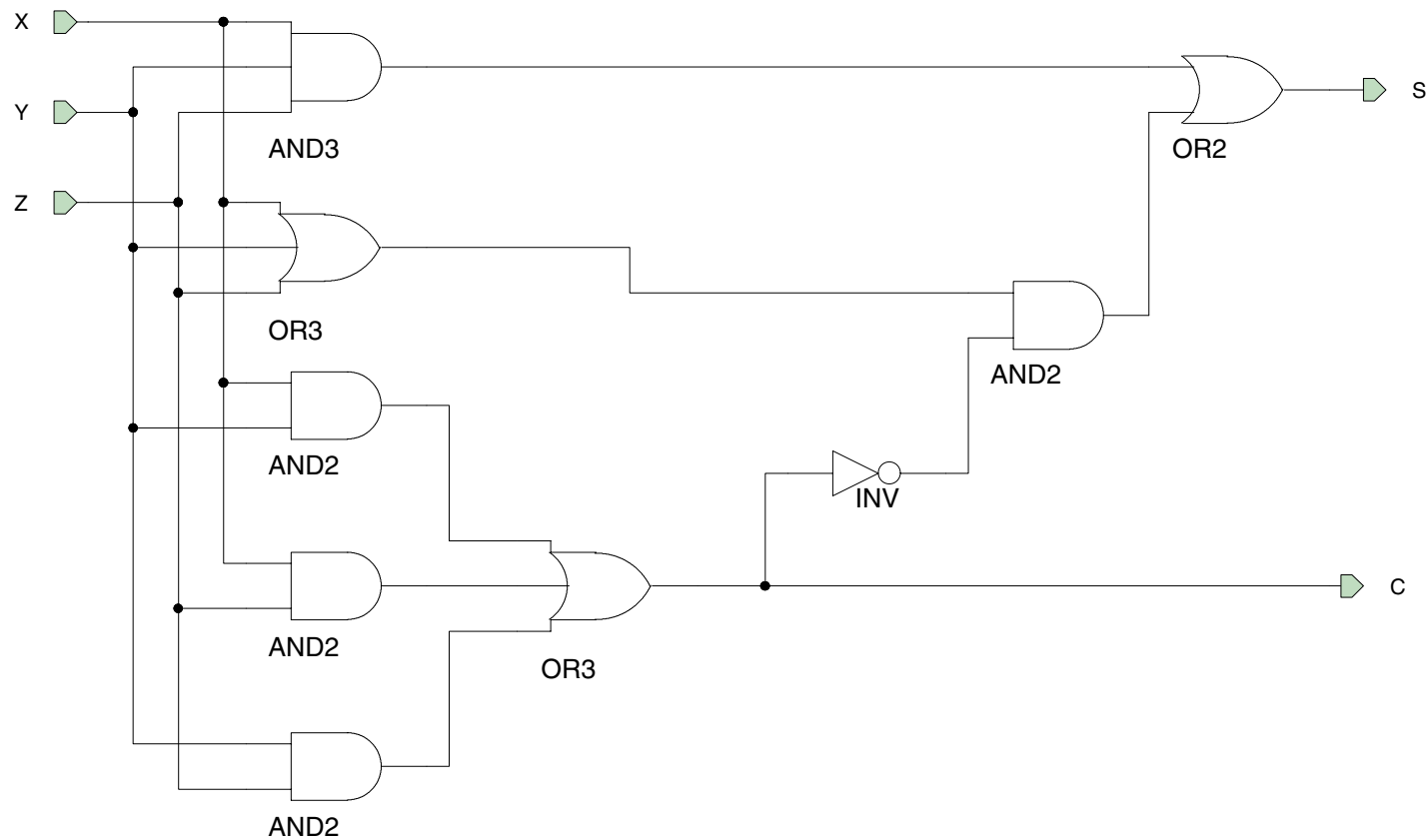


Fig. 3-7 Xilinx Foundation Schematic for Binary Adder in Figure 3-6

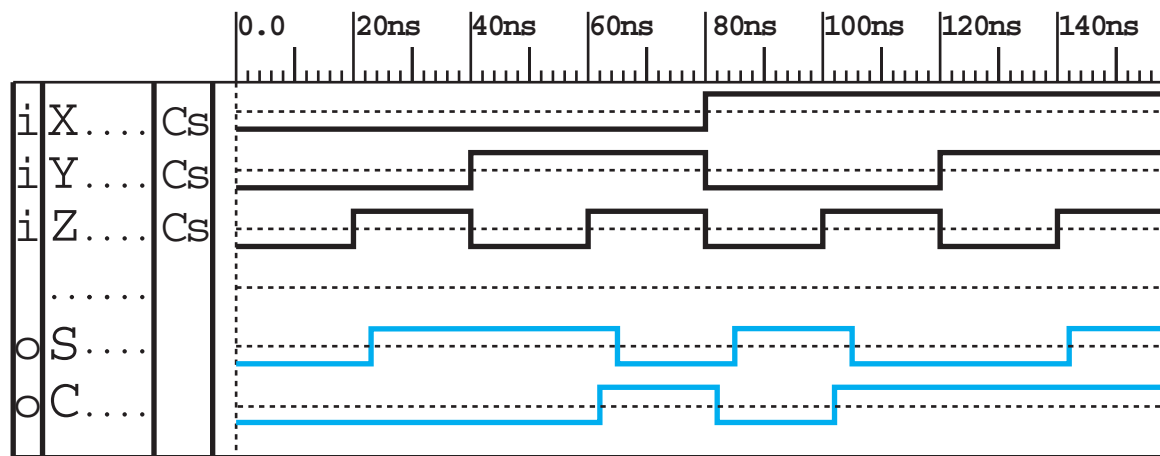
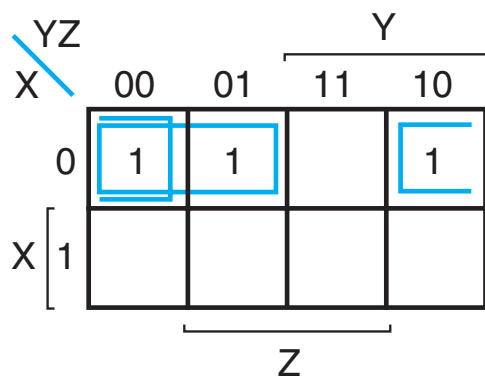


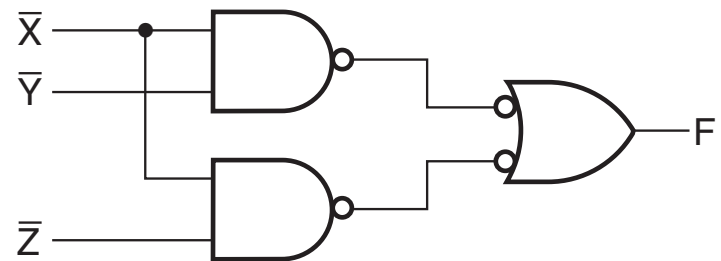
Fig. 3-8 Waveforms for the Binary Adder Schematic in Figure 3-7

X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

(a) Truth table



(b) Map $F = \bar{X}\bar{Y} + \bar{X}\bar{Z}$



(c) Logic diagram

Fig. 3-9 Solution to Example 3-1

TABLE 3-2
Truth Table for Code Converter Example

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Table 3-2 Truth Table for Code Converter Example

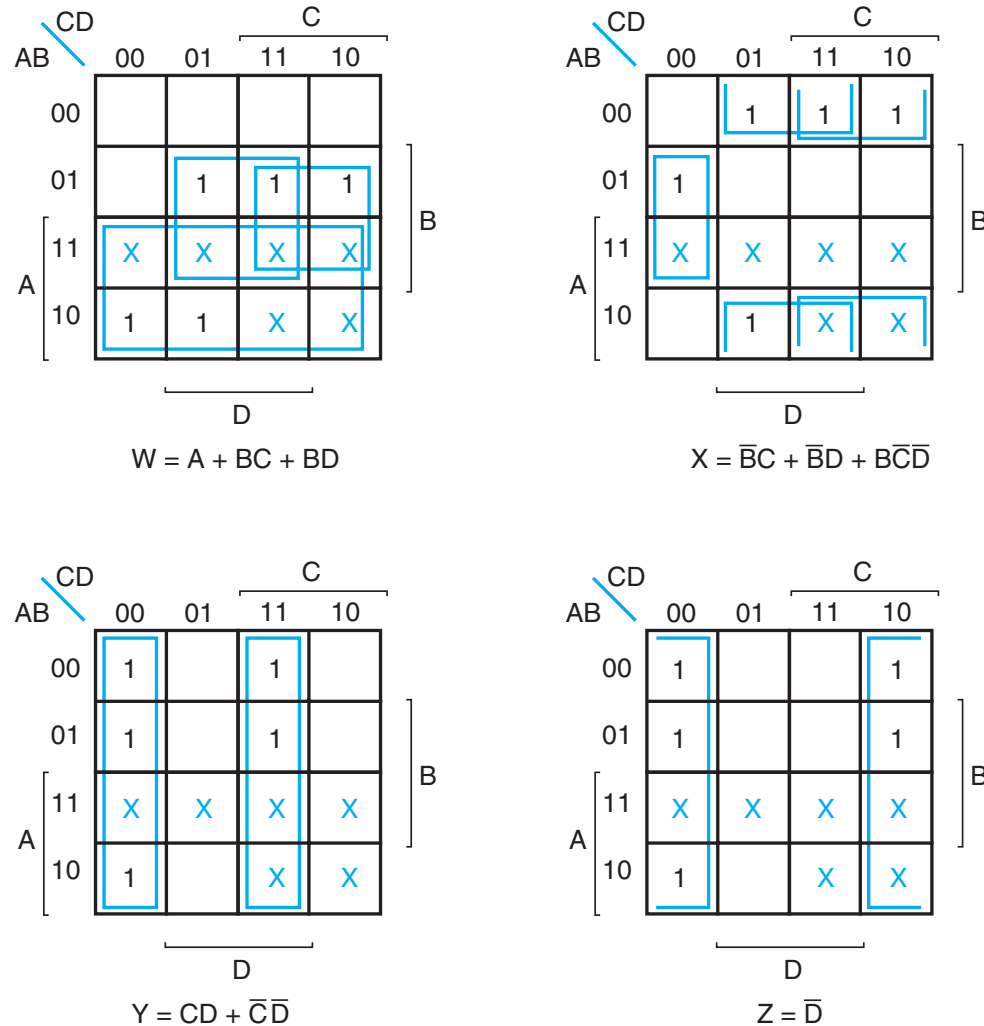


Fig. 3-10 Maps for BCD-to-Excess-3 Code Converter

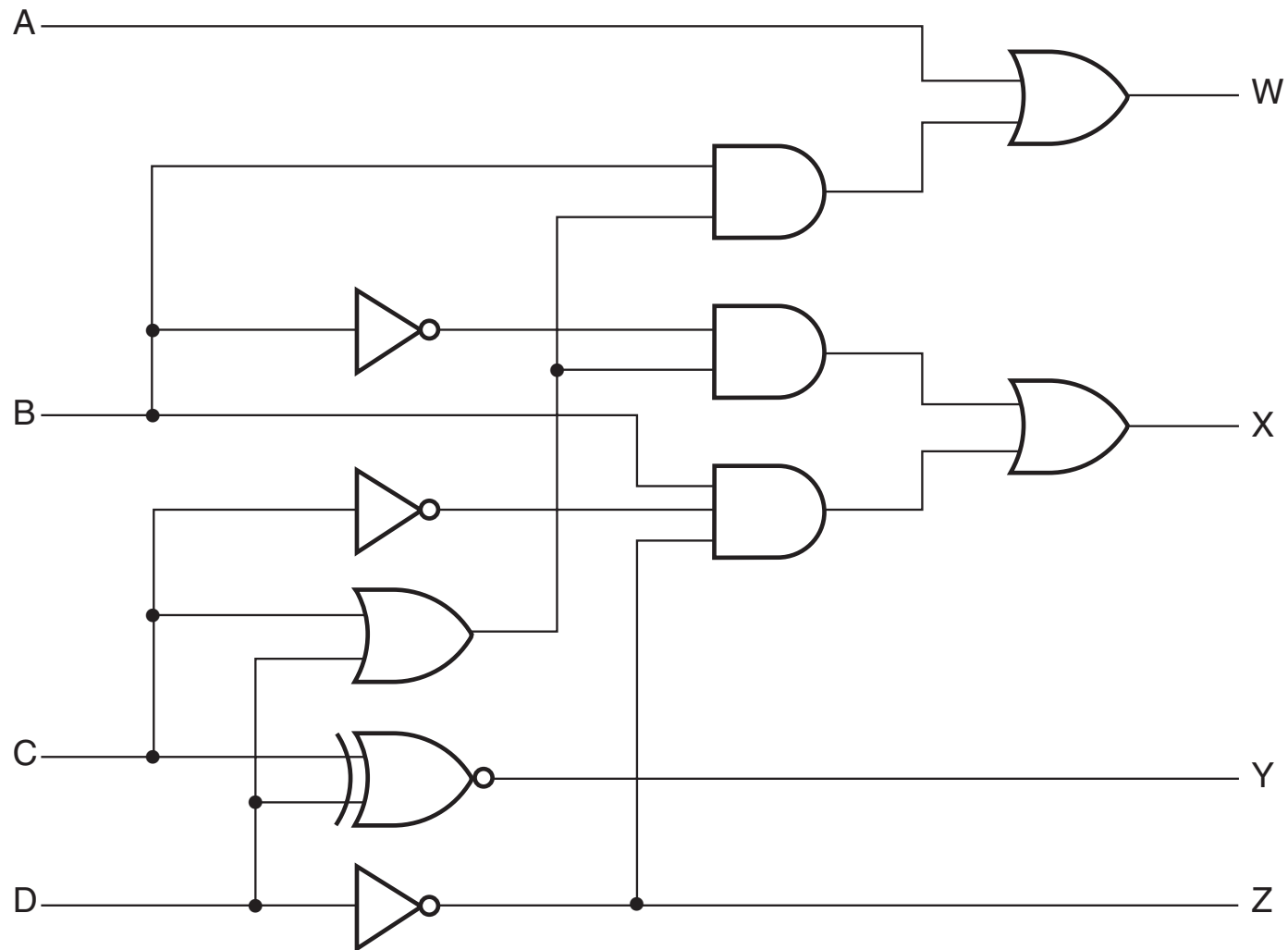


Fig. 3-11 Logic Diagram of BCD-to-Excess-3 Code Converter

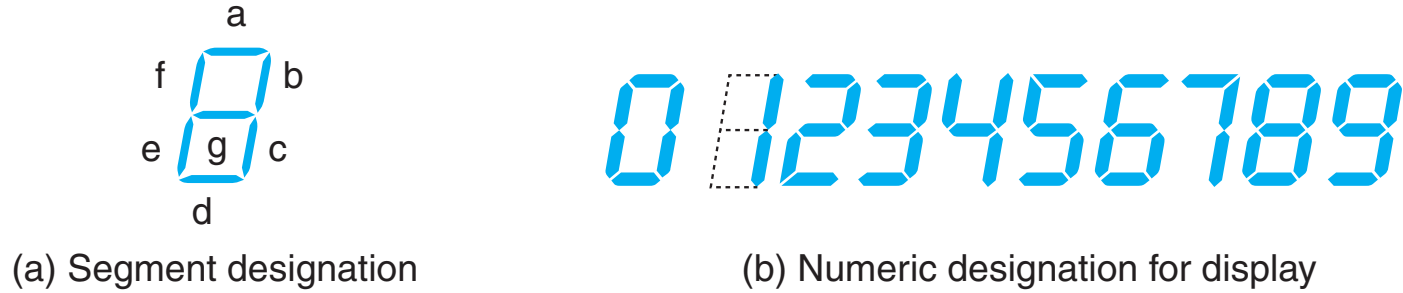


Fig. 3-12 Seven-Segment Display

TABLE 3-3
Truth Table for BCD-to-Seven-Segment
Decoder

BCD Input				Seven-Segment Decoder						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
All other inputs				0	0	0	0	0	0	0

Table 3-3 Truth Table for BCD-to-Seven-Segment Decoder

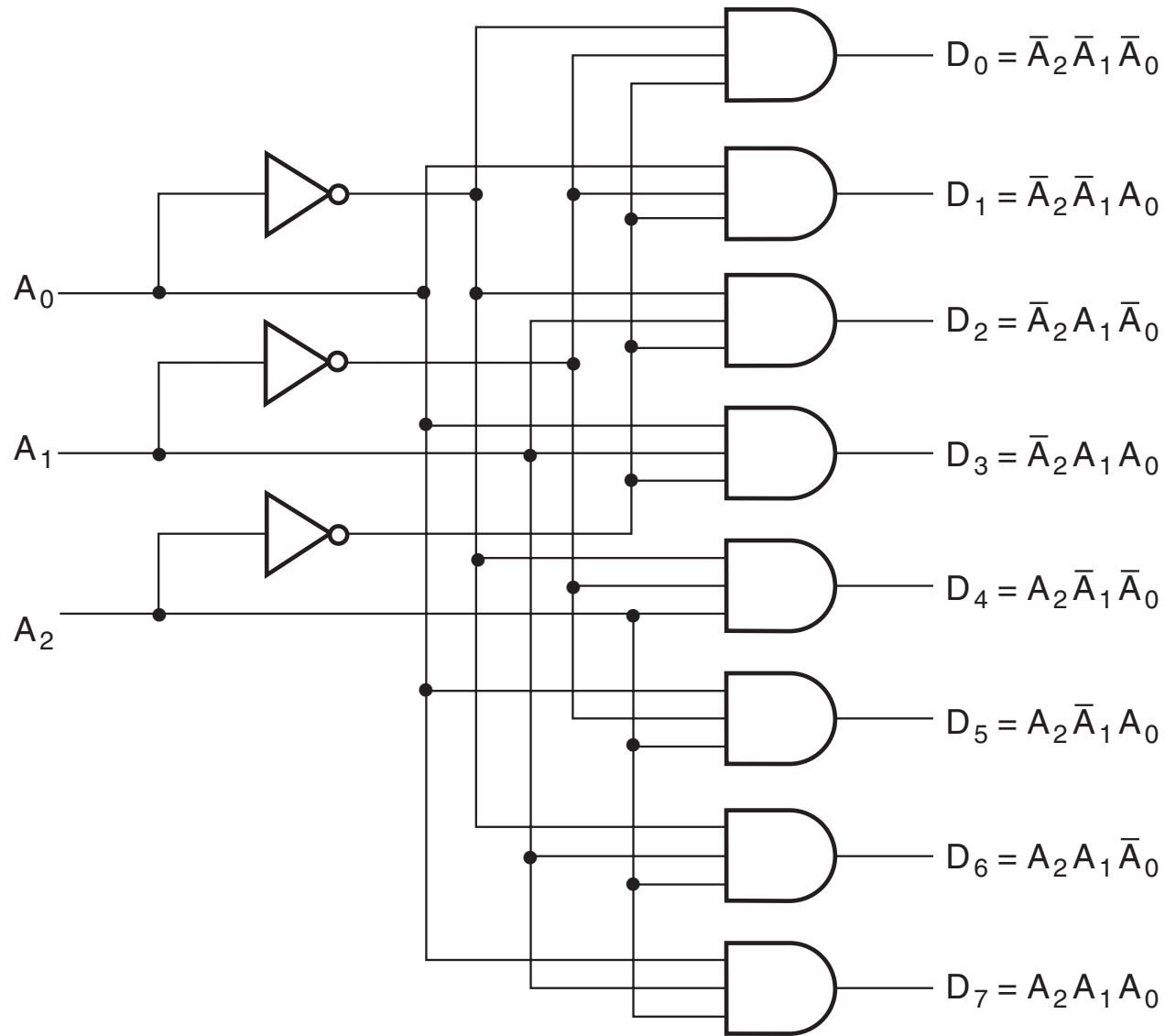
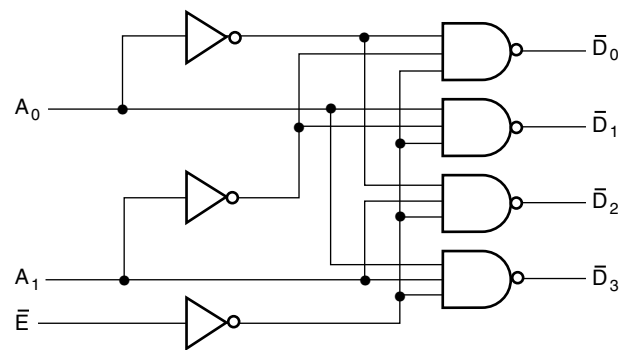


Fig. 3-13 3-to-8-Line Decoder



(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

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

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

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

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

(c) Logic Equations

Fig.3-14 A 2-to-4-Line Decoder

TABLE 3-4
Truth Table for 3-to-8-Line Decoder

Inputs			Outputs							
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Table 3-4 Truth Table for 3-to-8-Line Decoder

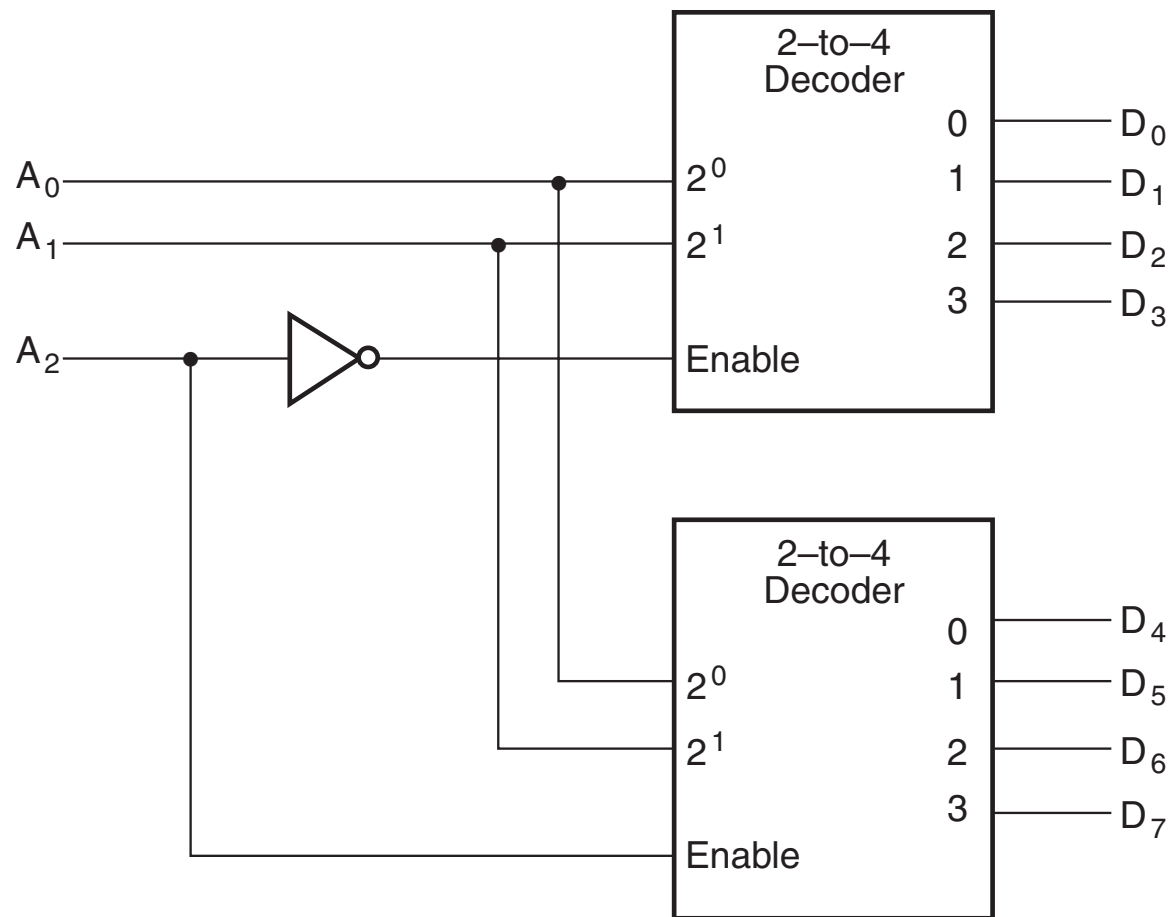


Fig. 3-15 A 3-to-8 Decoder Constructed with Two 2-to-4 Decoders

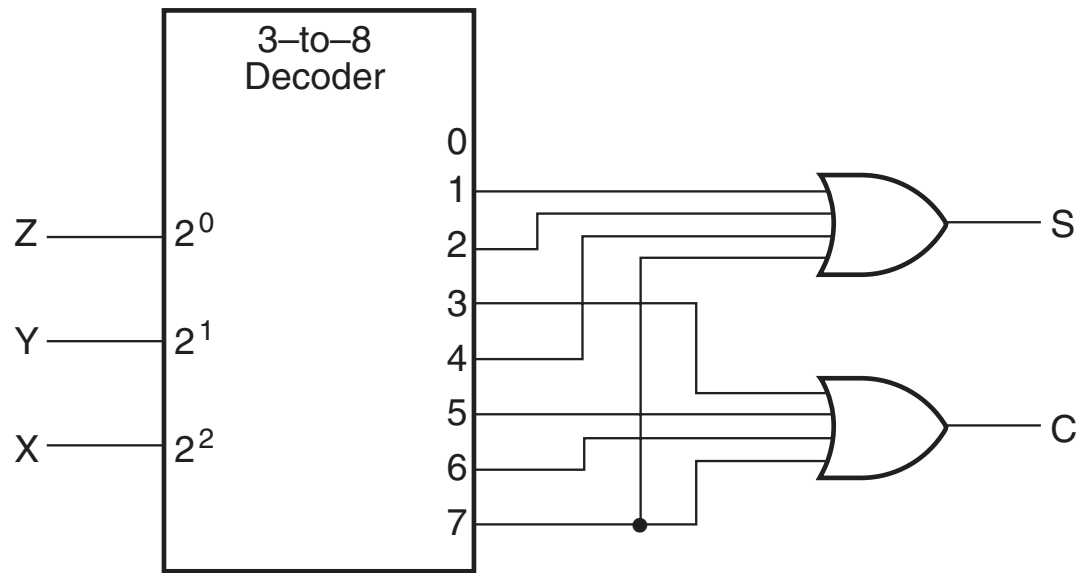


Fig. 3-16 Implementing a Binary Adder Using a Decoder

TABLE 3-5
Truth Table for Octal-to-Binary Encoder

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Table 3-5 Truth Table for Octal-to-Binary Encoder

TABLE 3-6
Truth Table of Priority Encoder

Inputs				Outputs		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Table 3-6 Truth Table of Priority Encoder

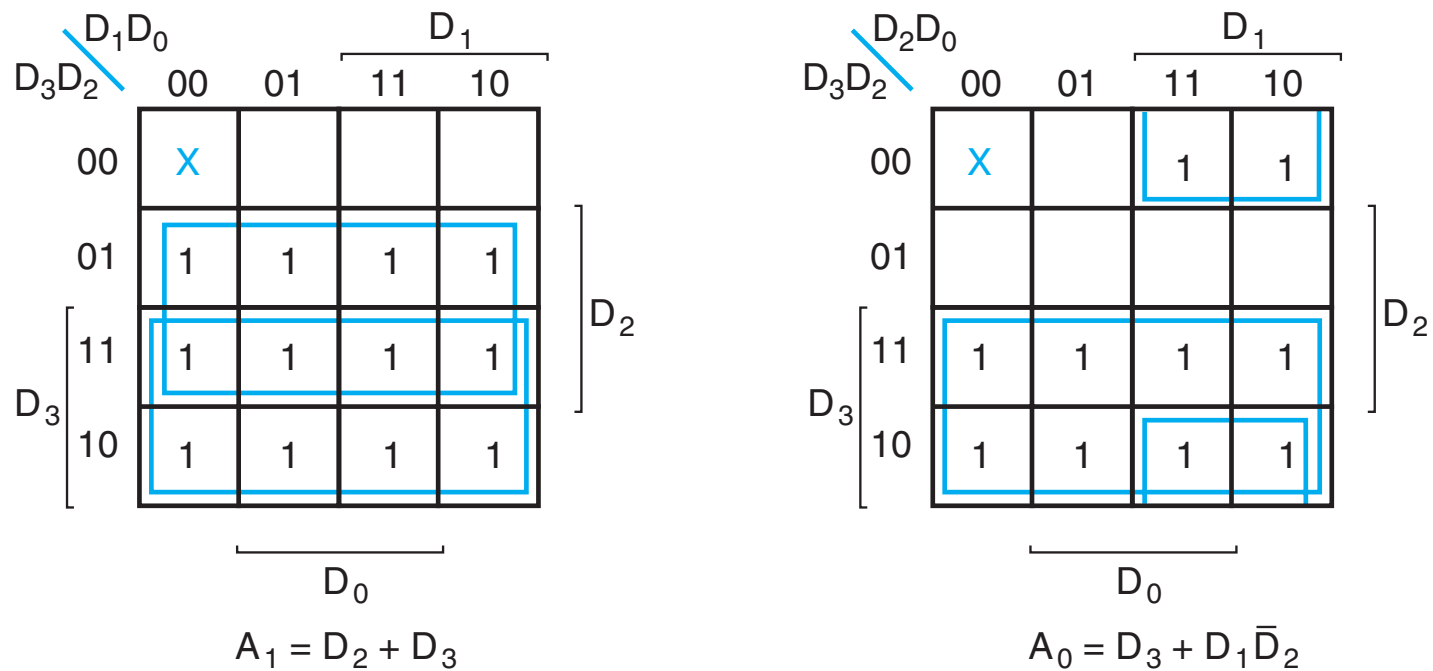


Fig. 3-17 Maps for Priority Encoder

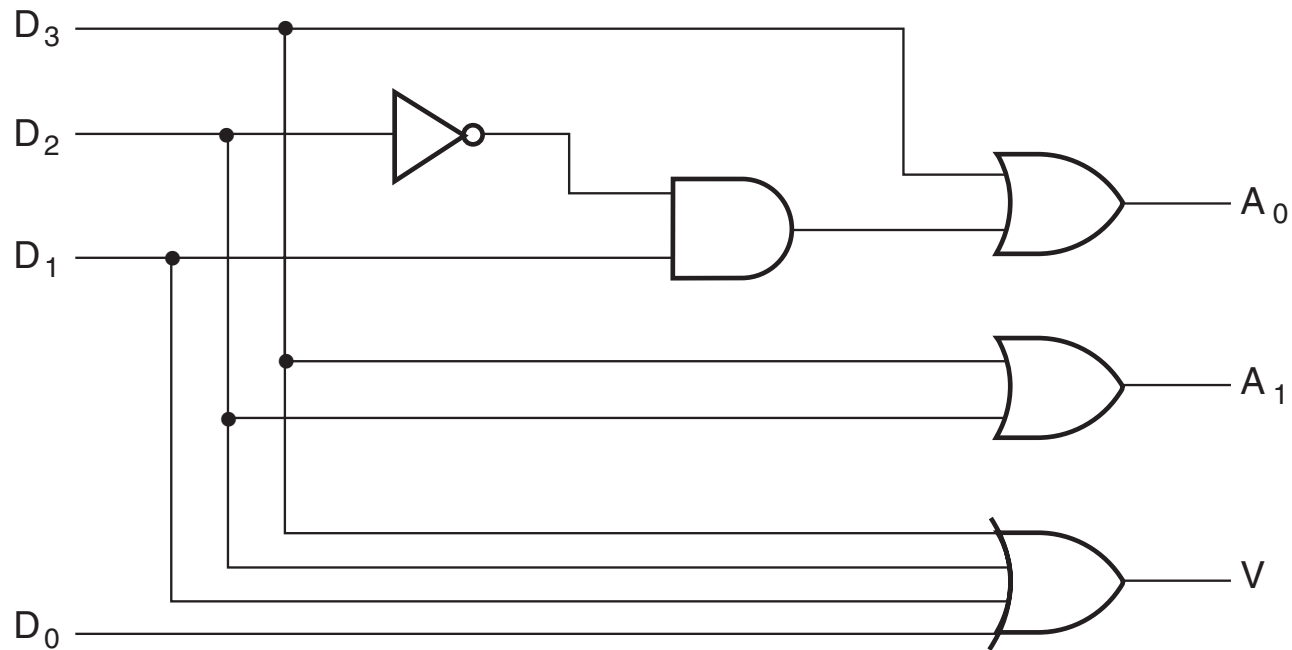


Fig. 3-18 Logic Diagram of a 4-Input Priority Encoder

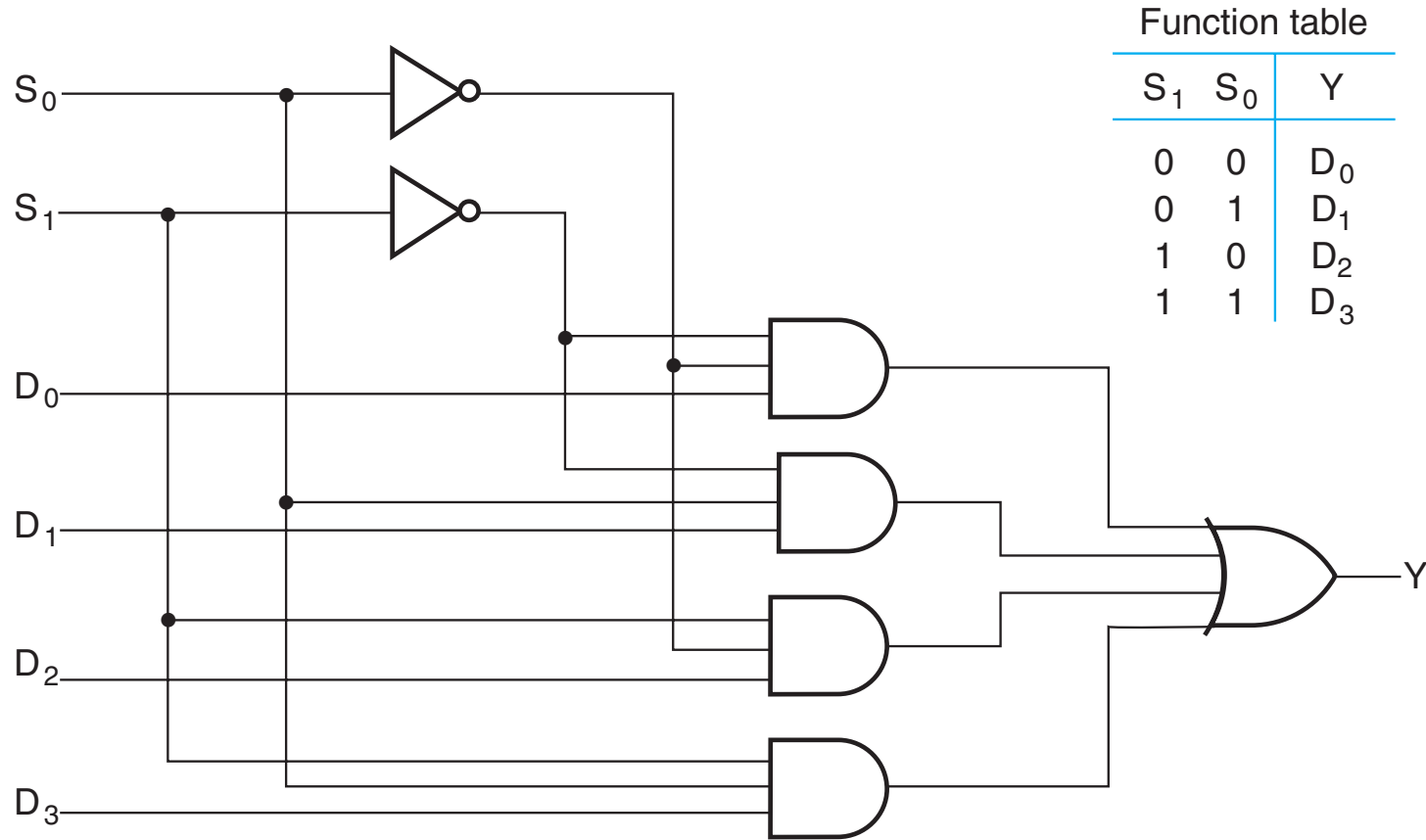


Fig. 3-19 4-to-1-Line Multiplexer

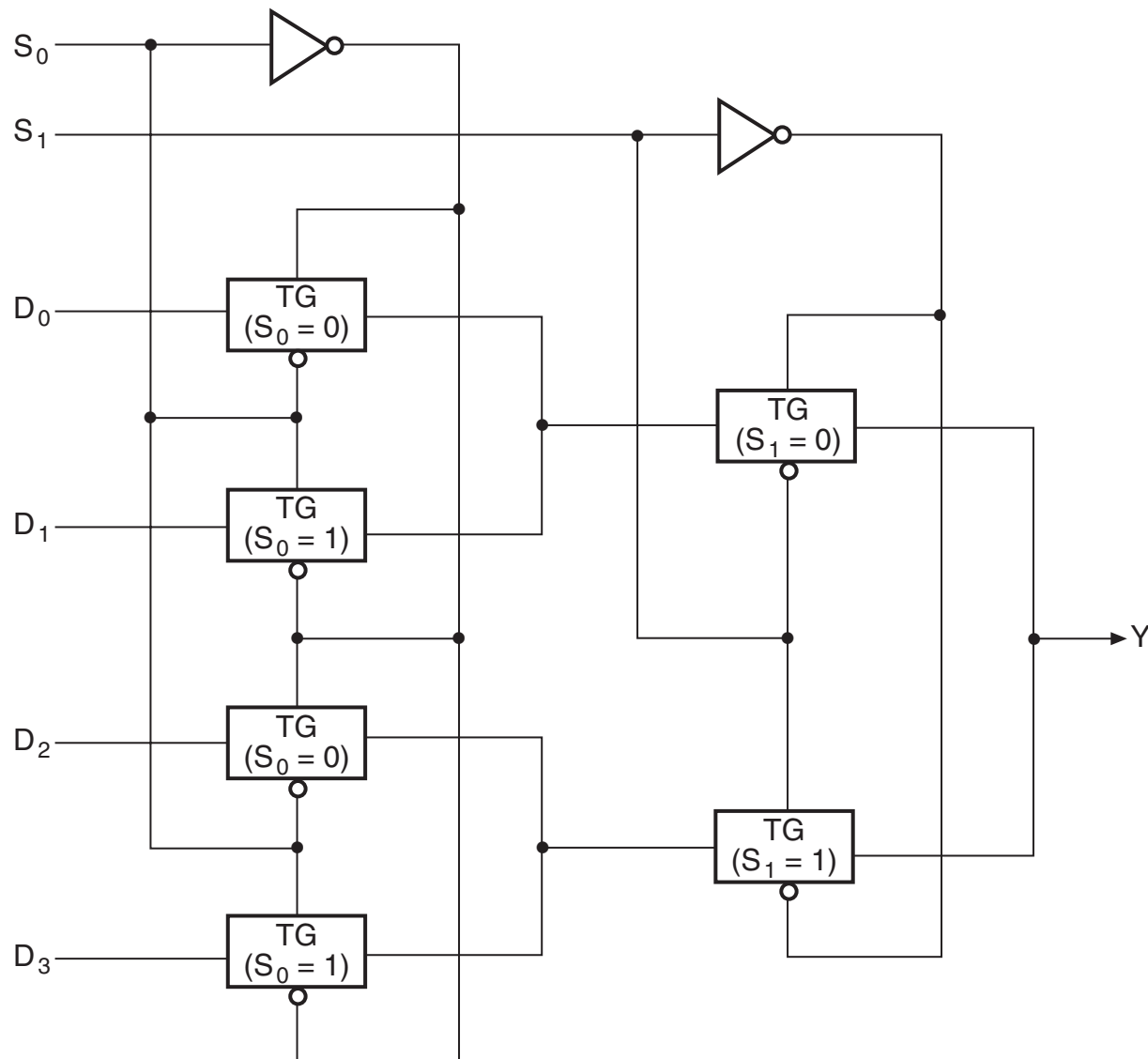


Fig. 3-20 4-to-1-Line Multiplexer with Transmission Gates

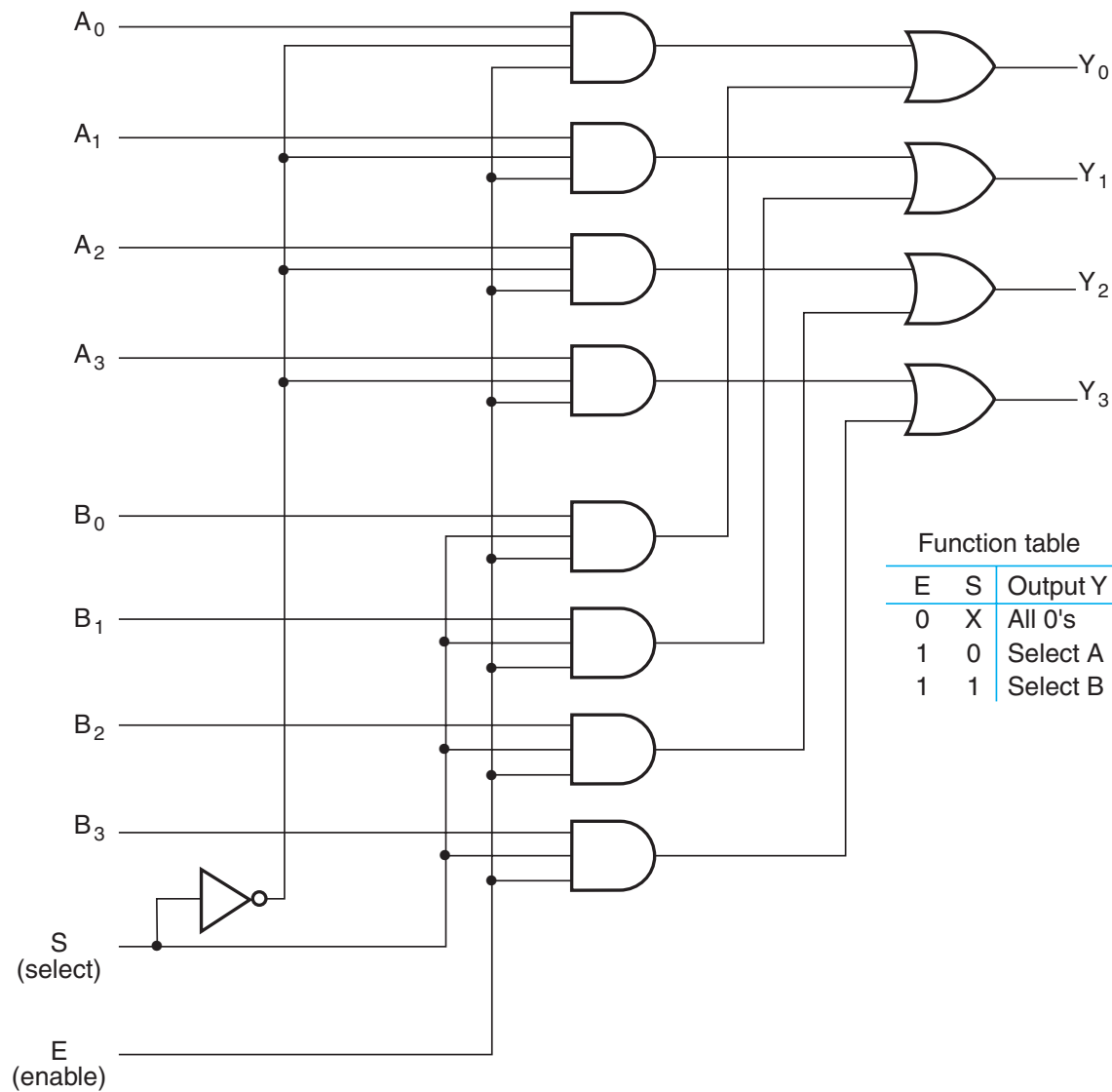
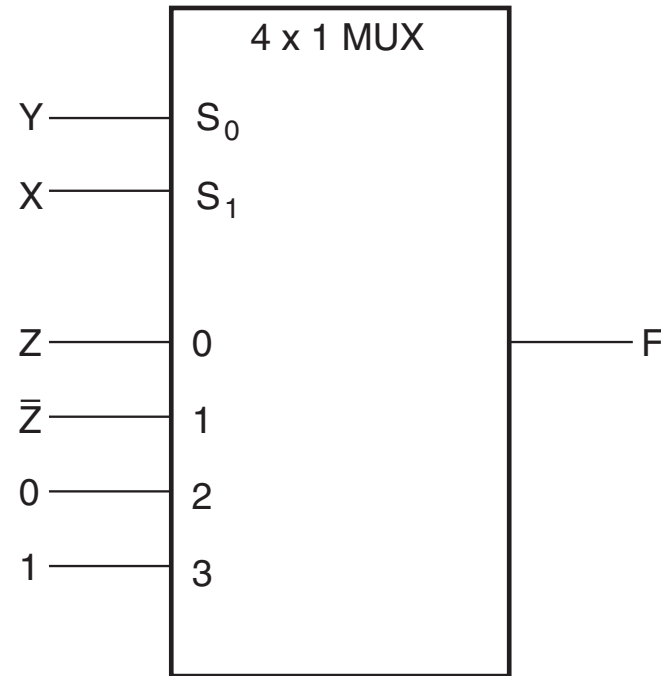


Fig. 3-21 Quadrate 2-to-1-Line Multiplexer

X	Y	Z	F	
0	0	0	0	$F = Z$
0	0	1	1	
0	1	0	1	$F = \bar{Z}$
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

Fig. 3-22 Implementing a Boolean Function with a Multiplexer

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = \bar{D}$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

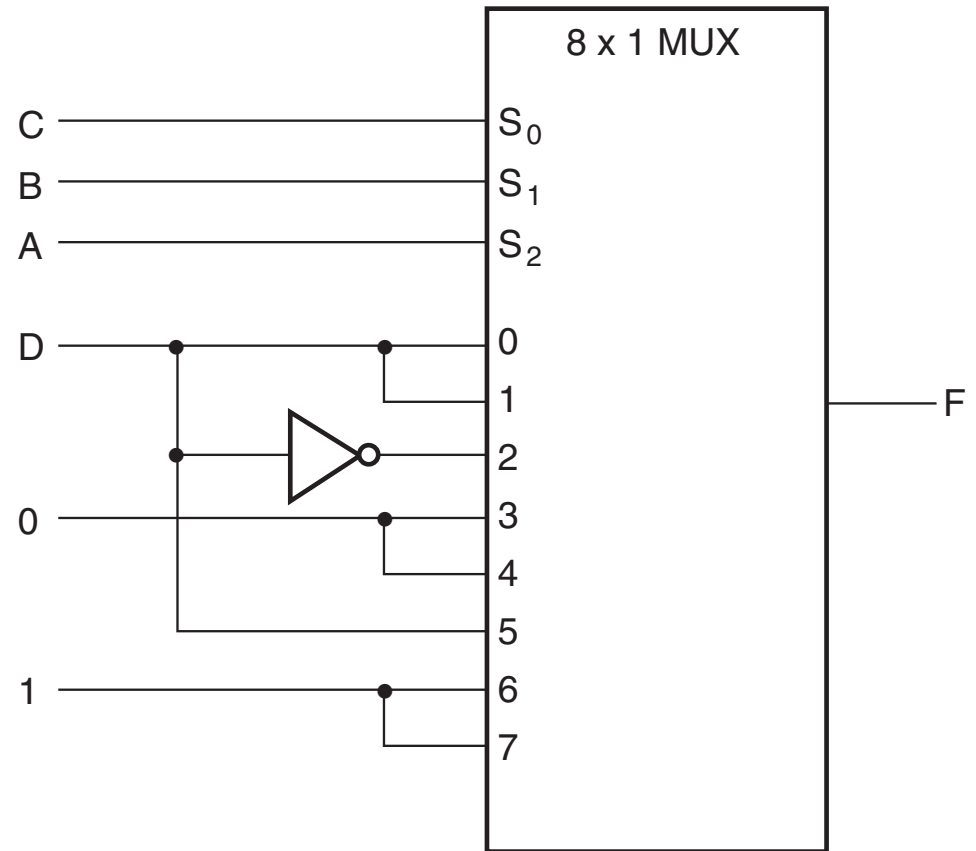


Fig. 3-23 Implementing a Four-Input Function with a Multiplexer

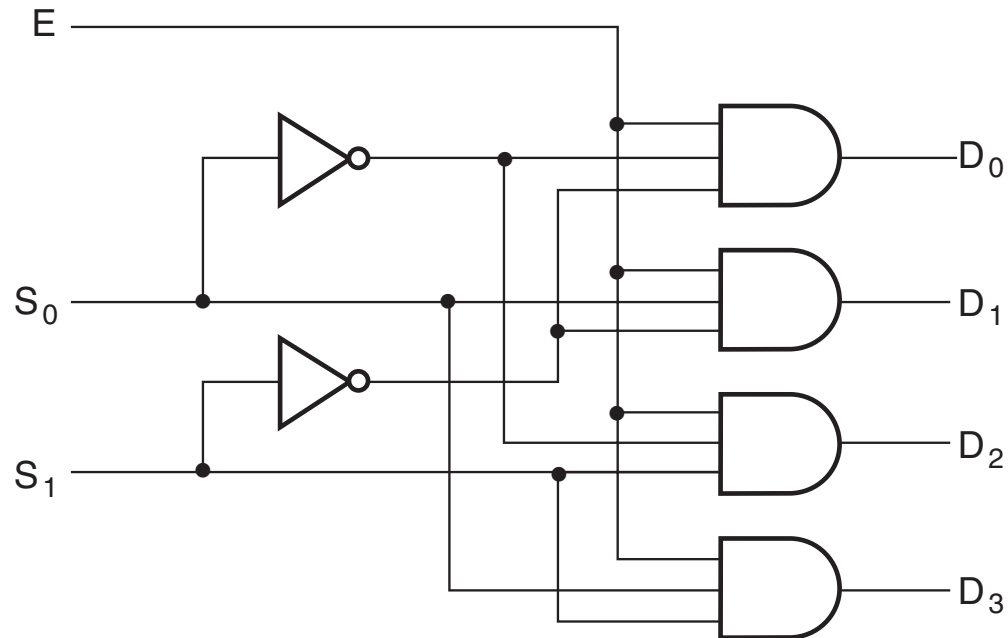


Fig. 3-24 1-to-4-Line Demultiplexer

TABLE 3-7
Truth Table of Half Adder

Inputs		Outputs	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 3-7 Truth Table of Half Adder

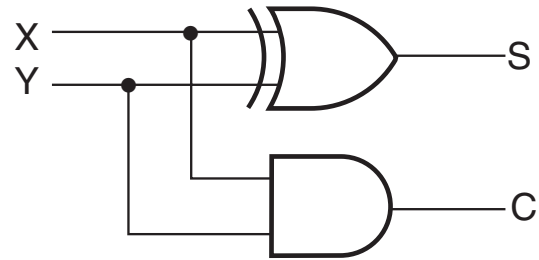
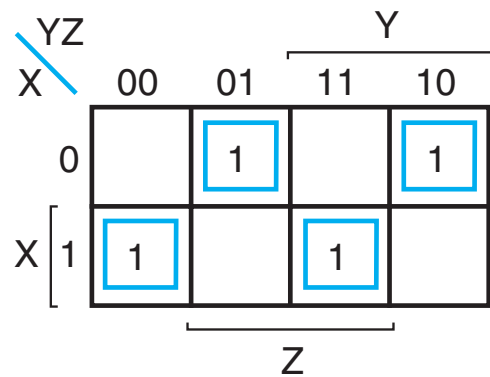


Fig. 3-25 Logic Diagram of Half Adder

TABLE 3-8
Truth Table of Full Adder

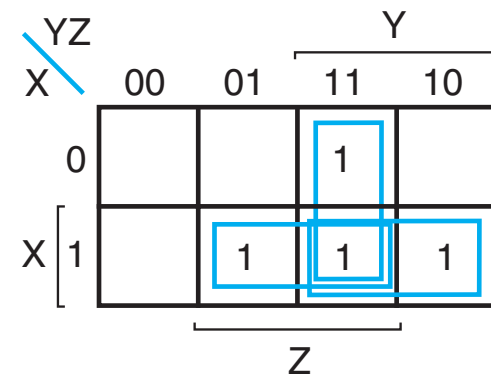
Inputs			Outputs	
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 3-8 Truth Table of Full Adder



$$S = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$$

$$= X \oplus Y \oplus Z$$



$$C = XY + XZ + YZ$$

$$= XY + Z(X\bar{Y} + \bar{X}Y)$$

$$= XY + Z(X \oplus Y)$$

Fig. 3-26 Maps for Full Adder

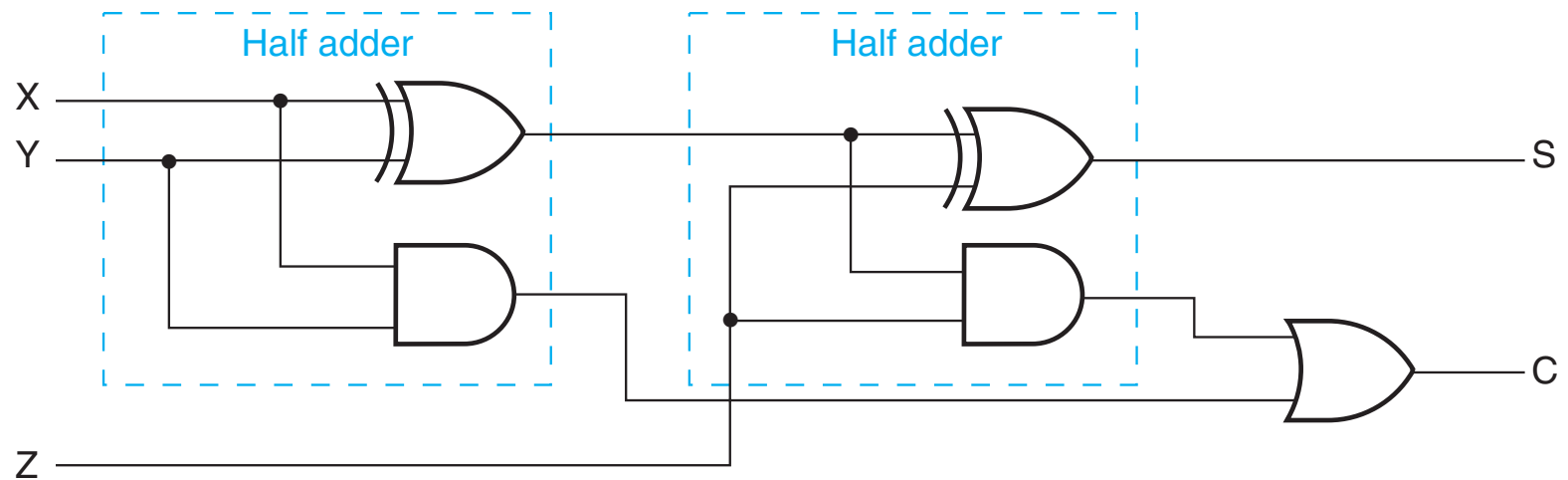


Fig. 3-27 Logic Diagram of Full Adder

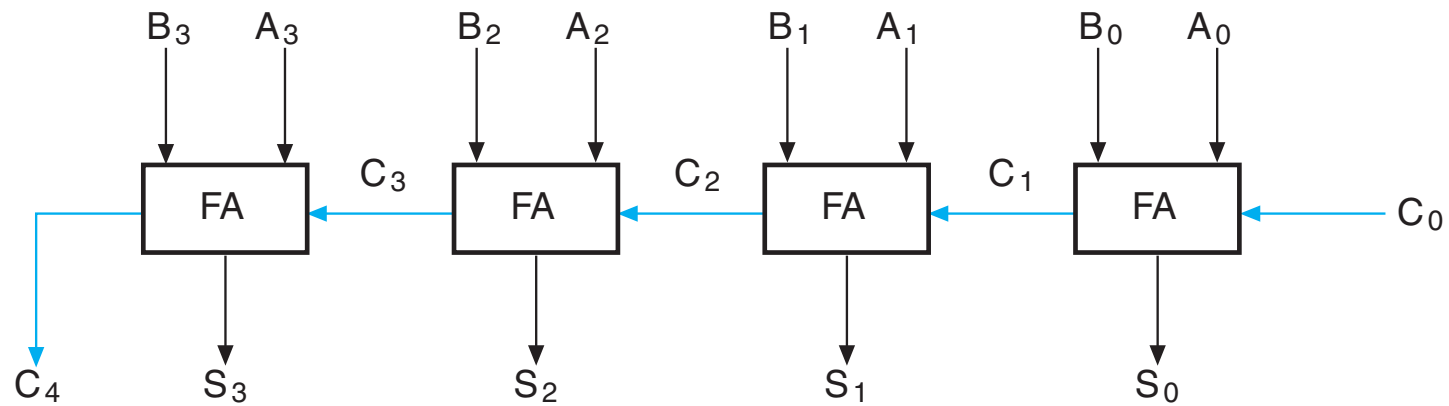


Fig. 3-28 4-Bit Ripple Carry Adder

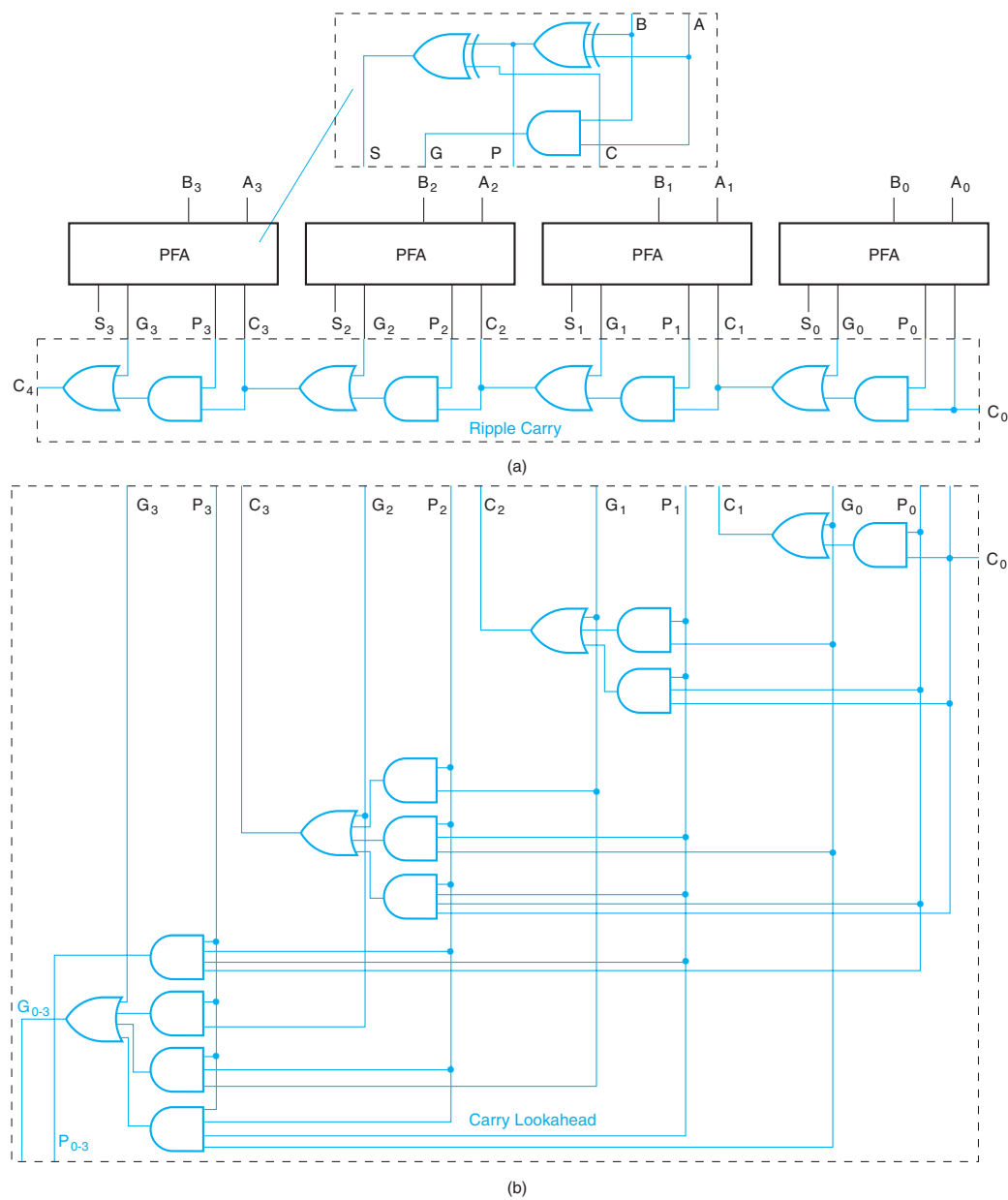


Fig. 3-29 Development of a Carry Lookahead Adder

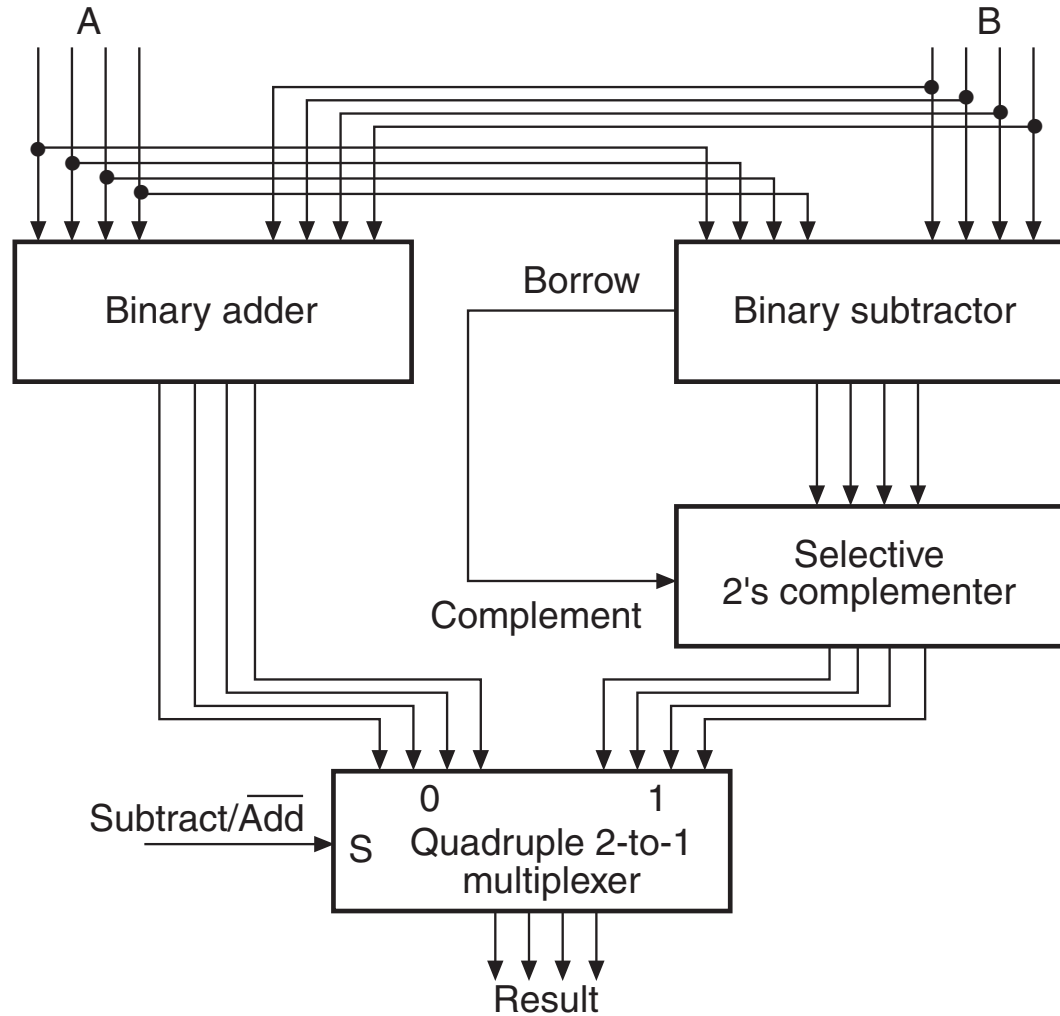


Fig. 3-30 Block Diagram of Binary Adder-Subtractor

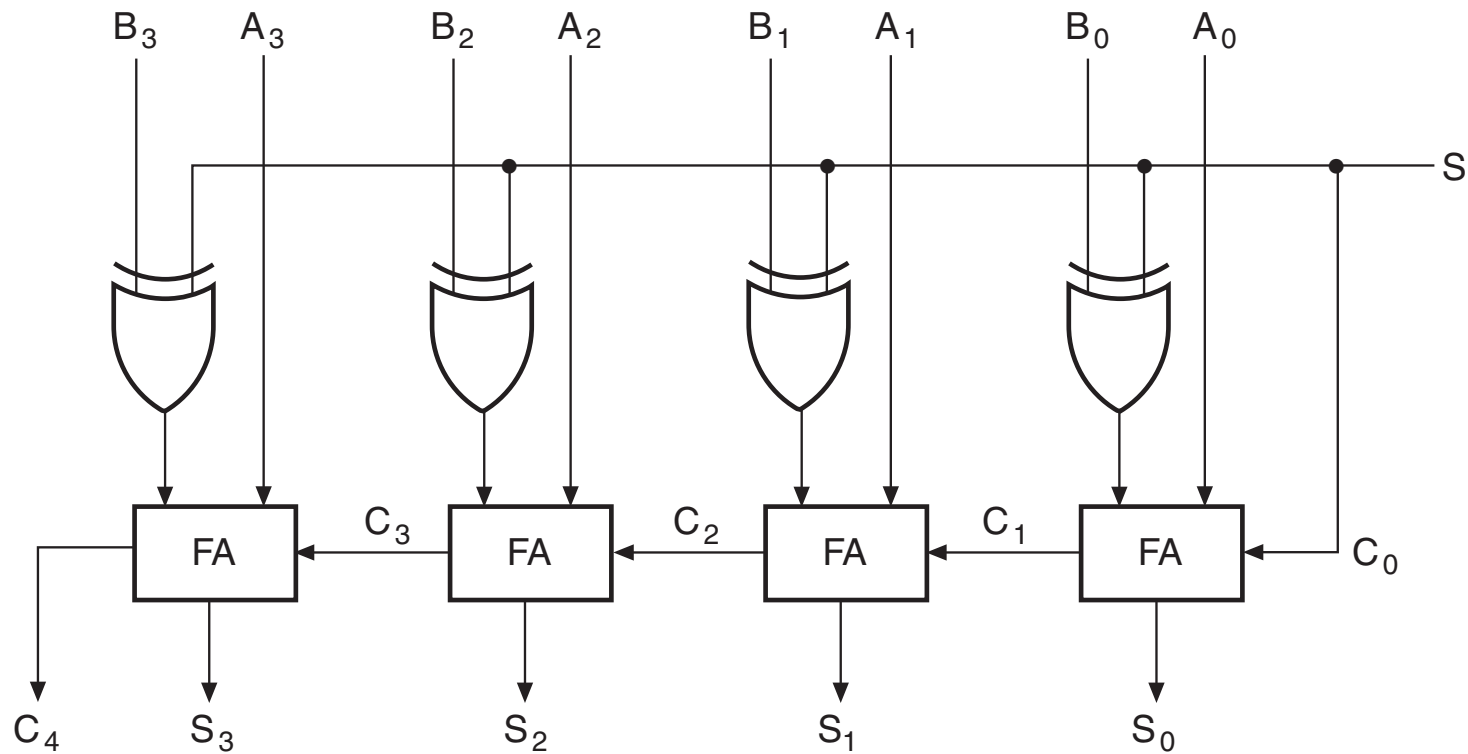


Fig. 3-31 Adder-Subtractor Circuit

TABLE 3-9
Signed Binary Numbers

Decimal	Signed 2's Complement	Signed 1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Table 3-9 Signed Binary Numbers

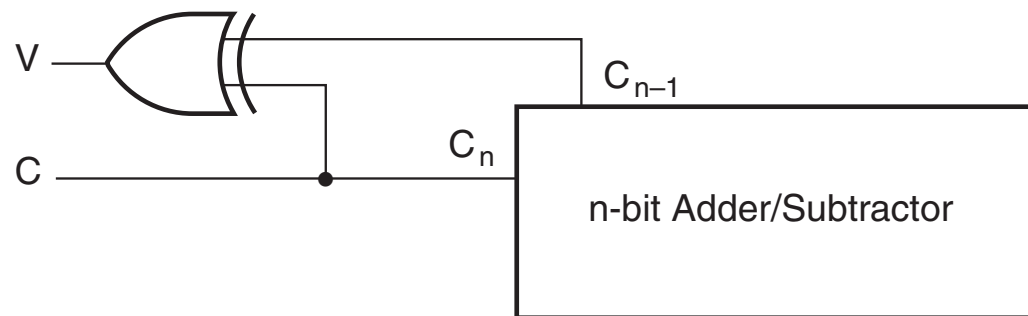


Fig. 3-32 Overflow Detection for Addition and Subtraction

		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

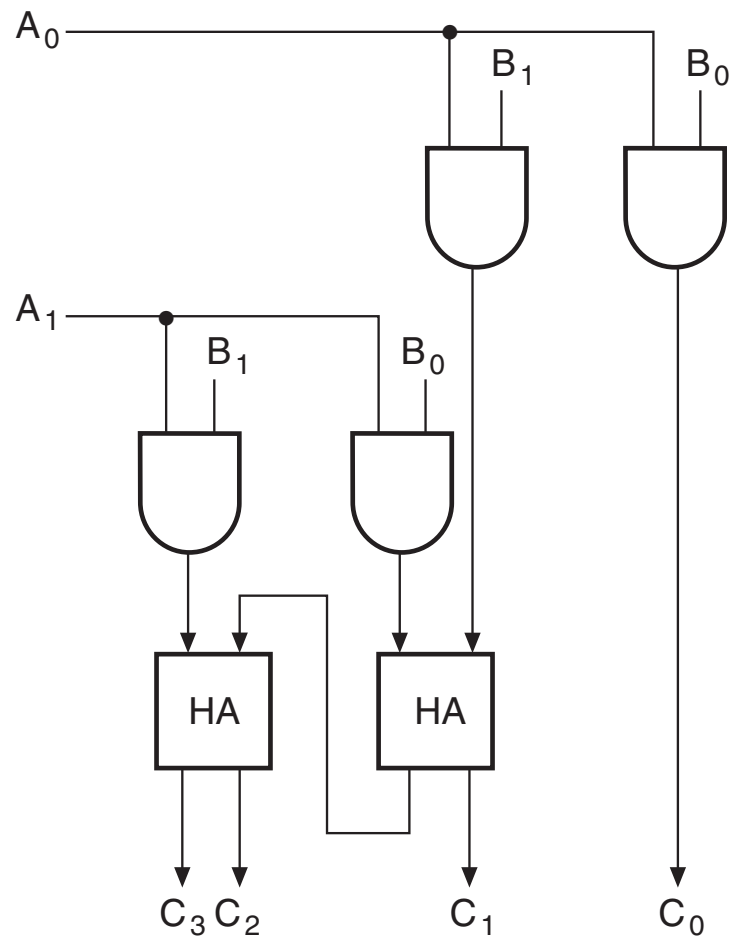


Fig. 3-33 A 2-Bit by 2-Bit Binary Multiplier

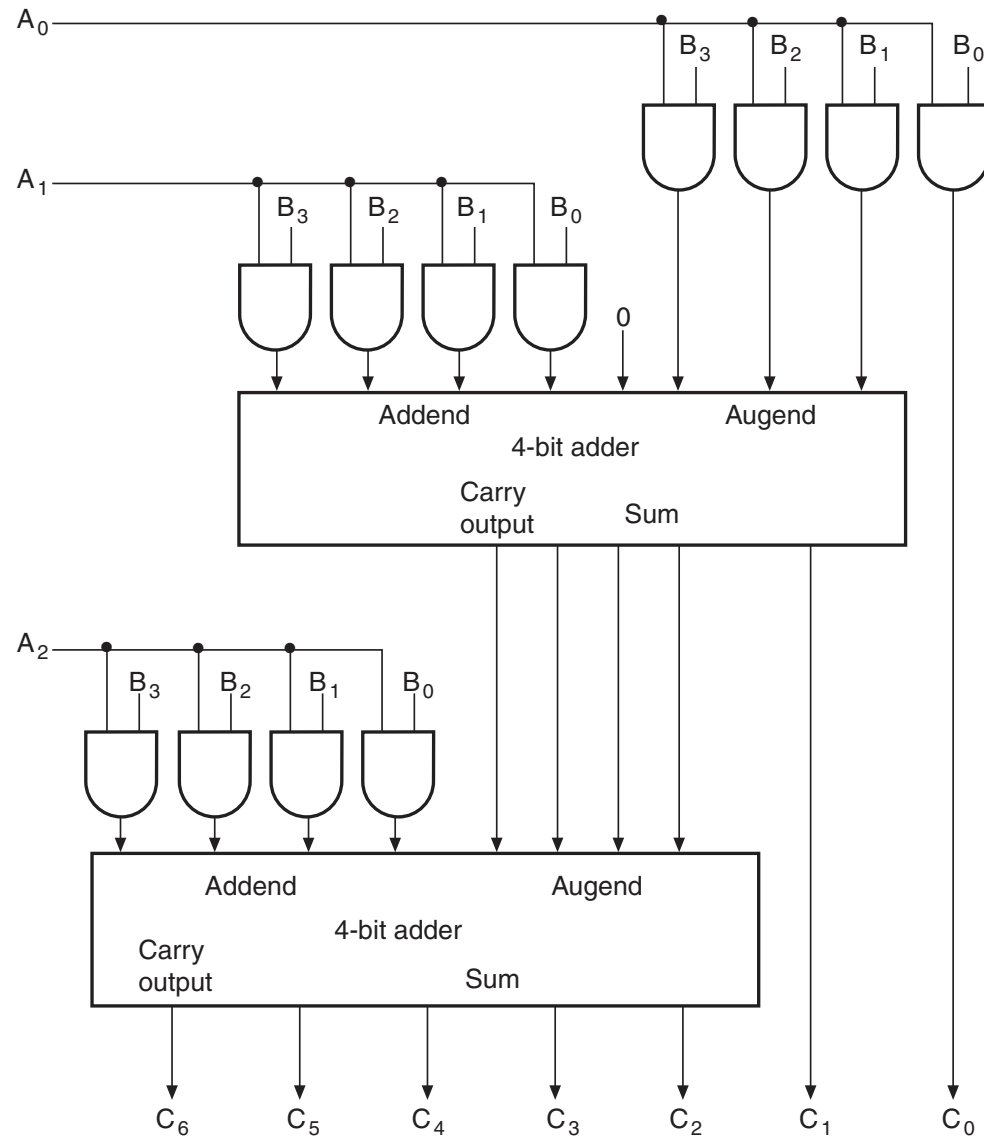


Fig. 3-34 A 4-Bit by 3-Bit Binary Multiplier

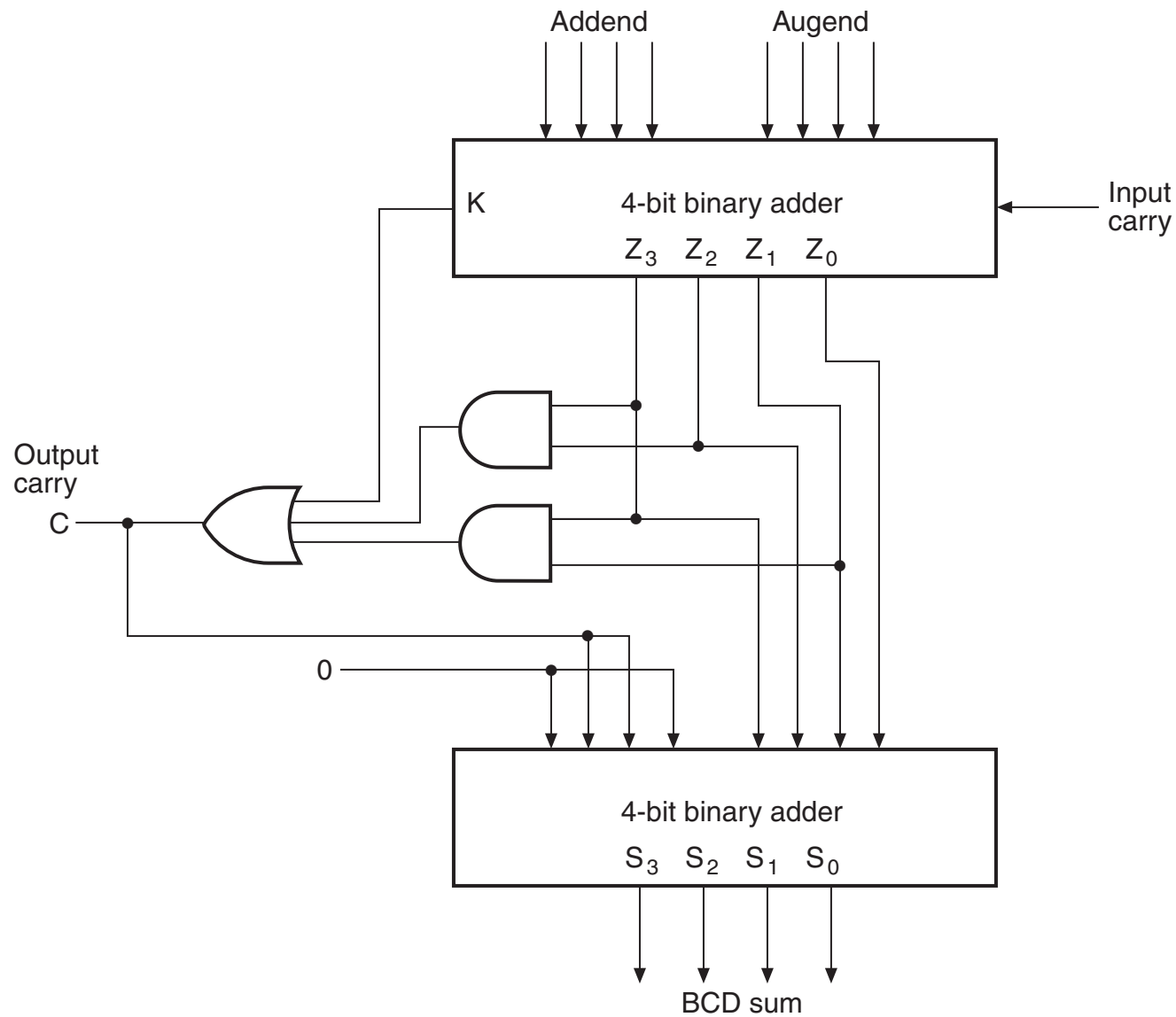


Fig. 3-35 Block Diagram of BCD Adder

```

-- 2-to-4 Line Decoder: Structural VHDL Description          -- 1
-- (See Figure 3-14 for logic diagram)                     -- 2
library ieee, lcdf_vhdl;                                    -- 3
use ieee.std_logic_1164.all, lcdf_vhdl.func_prims.all;     -- 4
entity decoder_2_to_4 is                                     -- 5
    port(E_n, A0, A1: in std_logic;                         -- 6
          D0_n, D1_n, D2_n, D3_n: out std_logic);          -- 7
end decoder_2_to_4;                                         -- 8
                                                            -- 9

architecture structural_1 of decoder_2_to_4 is              -- 10
    component NOT1                                          -- 11
        port(in1: in std_logic;                            -- 12
              out1: out std_logic);                        -- 13
    end component;                                          -- 14
    component NAND3                                          -- 15
        port(in1, in2, in3: in std_logic;                  -- 16
              out1: out std_logic);                        -- 17
    end component;                                          -- 18
    signal E, A0_n, A1_n: std_logic;                       -- 19
begin                                                       -- 20
    g0: NOT1 port map (in1 => A0, out1 => A0_n);            -- 21
    g1: NOT1 port map (in1 => A1, out1 => A1_n);            -- 22
    g2: NOT1 port map (in1 => E_n, out1 => E);              -- 23
    g3: NAND3 port map (in1 => A0_n, in2 => A1_n,           -- 24
                        in3 => E, out1 => D0_n);            -- 25
    g4: NAND3 port map (in1 => A0, in2 => A1_n,             -- 26
                        in3 => E, out1 => D1_n);            -- 27
    g5: NAND3 port map (in1 => A0_n, in2 => A1,             -- 28
                        in3 => E, out1 => D2_n);            -- 29
    g6: NAND3 port map (in1 => A0, in2 => A1,               -- 30
                        in3 => E, out1 => D3_n);            -- 31
end structural_1;                                           -- 32

```

Fig. 3-36 Structural VHDL Description of a 2-to-4 Line Decoder

```

-- 4-to-1 Line Multiplexer: Structural VHDL Description      -- 1
-- (See Figure 3-19 for logic diagram)                      -- 2
library ieee, lcdf_vhdl;                                    -- 3
use ieee.std_logic_1164.all, lcdf_vhdl.func_prims.all;      -- 4
entity multiplexer_4_to_1_st is                             -- 5
    port(S: in std_logic_vector(0 to 1);                    -- 6
          D: in std_logic_vector(0 to 3);                    -- 7
          Y: out std_logic);                                  -- 8
end multiplexer_4_to_1_st;                                   -- 9
                                                            -- 10
architecture structural_2 of multiplexer_4_to_1_st is       -- 11
    component NOT1                                           -- 12
        port(in1: in std_logic;                               -- 13
              out1: out std_logic);                           -- 14
    end component;                                           -- 15
    component AND3                                           -- 16
        port(in1, in2, in3: in std_logic;                     -- 17
              out1: out std_logic);                           -- 18
    end component;                                           -- 19
    component OR4                                             -- 20
        port(in1, in2, in3, in4: in std_logic;                 -- 21
              out1: out std_logic);                           -- 22
    end component;                                           -- 23
    signal S_n: std_logic_vector(0 to 1);                    -- 24
    signal N: std_logic_vector(0 to 3);                       -- 25
    begin                                                     -- 26
        g0: NOT1 port map (S(0), S_n(0));                     -- 27
        g1: NOT1 port map (S(1), S_n(1));                     -- 28
        g2: AND3 port map (S_n(1), S_n(0), D(0), N(0));       -- 29
        g3: AND3 port map (S_n(1), S(0), D(1), N(1));        -- 30
        g4: AND3 port map (S(1), S_n(0), D(2), N(2));        -- 31
        g5: AND3 port map (S(1), S(0), D(3), N(3));          -- 32
        g6: OR4 port map (N(0), N(1), N(2), N(3), Y);         -- 33
    end structural_2;                                         -- 34

```

Fig. 3-37 Structural VHDL Description of a 4-to-1 Line Multiplexer

```

-- 2-to-4 Line Decoder: Dataflow VHDL Description           -- 1
-- (See Figure 3-14 for logic diagram)                     -- 2
Use library, use, and entity entries from 2_to_4_decoder_st; -- 3
-- 4
architecture dataflow_1 of decoder_2_to_4 is              -- 5
-- 6
signal A0_n, A1_n: std_logic;                             -- 7
begin                                                       -- 8
    A0_n <= not A0;                                         -- 9
    A1_n <= not A1;                                         -- 10
    E <= not E_n;                                           -- 11
    D0_n <= not ( A0_n and A1_n and E );                   -- 12
    D1_n <= not ( A0 and A1_n and E );                     -- 13
    D2_n <= not ( A0_n and A1 and E );                     -- 14
    D3_n <= not ( A0 and A1 and E );                     -- 15
end dataflow_1;                                           -- 16

```

Fig. 3-38 Dataflow VHDL Description of a 2-to-4 Line Decoder

```

-- 4-to-1 Line Mux: Conditional Dataflow VHDL Description      -- 1
-- Using When-Else (See Figure 3-19 for function table)      -- 2
library ieee;                                                -- 3
use ieee.std_logic_1164.all;                                -- 4
entity multiplexer_4_to_1_we is                               -- 5
    port (S : in std_logic_vector(1 downto 0);              -- 6
          D : in std_logic_vector(0 to 3);                  -- 7
          Y : out std_logic);                                -- 8
end multiplexer_4_to_1_we;                                   -- 9
                                                                -- 10
architecture function_table of multiplexer_4_to_1_we is    -- 11
begin                                                         -- 12
    Y <= D(0) when S = "00" else                               -- 13
        D(1) when S = "01" else                               -- 14
        D(2) when S = "10" else                               -- 15
        D(3) when S = "11" else                               -- 16
            'X';                                                -- 17
end function_table;                                          -- 18

```

Fig. 3-39 Conditional Dataflow VHDL Description of a 4-to-1 Line Multiplexer Using When-Else


```

--4-to-1 Line Mux: Conditional Dataflow VHDL Description      -- 1
Using When-Else (See Figure 3-14 for logic equations)        -- 2
library ieee;                                                -- 3
use ieee.std_logic_1164.all;                                  -- 4
entity multiplexer_4_to_1_ws is                               -- 5
    port (S : in std_logic_vector(1 downto 0);              -- 6
          D : in std_logic_vector(0 to 3);                  -- 7
          Y : out std_logic);                                  -- 8
end multiplexer_4_to_1_ws;                                     -- 9
                                                                -- 10
architecture function_table_ws of multiplexer_4_to_1_ws is -- 11
begin                                                         -- 12
    with S select                                              -- 13
        Y <= D(0) when "00",                                   -- 14
           D(1) when "01",                                   -- 15
           D(2) when "10",                                   -- 16
           D(3) when "11",                                   -- 17
           'X' when others;                                    -- 18
end function_table_ws;                                       -- 19

```

Fig. 3-40 Conditional Dataflow VHDL Description of a 4-to-1
Line Multiplexer Using With-Select

```
-- 4-bit Adder: Hierarchical Dataflow/Structural
-- (See Figures 3-27 and 3-28 for logic diagrams)
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
    port (x, y : in std_logic;
          s, c : out std_logic);
end half_adder;

architecture dataflow_3 of half_adder is
begin
    s <= x xor y;
    c <= x and y;
end dataflow_3;

library ieee;
use ieee.std_logic_1164.all;
entity full_adder is
    port (x, y, z : in std_logic;
          s, c : out std_logic);
end full_adder;

architecture struc_dataflow_3 of full_adder is
    component half_adder
        port(x, y : in std_logic;
             s, c : out std_logic);
    end component;
    signal hs, hc, tc: std_logic;
begin
    HA1: half_adder
        port map (x, y, hs, hc);
    HA2: half_adder
        port map (hs, z, s, tc);
    c <= tc or hc;
end struc_dataflow_3;

library ieee;
use ieee.std_logic_1164.all;
entity adder_4 is
    port(B, A : in std_logic_vector(3 downto 0);
          C0 : in std_logic;
          S : out std_logic_vector(3 downto 0);
          C4: out std_logic);
end adder_4;
```

Fig. 3-41 Hierarchical Structural/Dataflow Description
of a 4-Bit Adder

```

architecture structural_4 of adder_4 is
    component full_adder
        port(x, y, z : in std_logic;
            s, c : out std_logic);
    end component;
    signal C: std_logic_vector(4 downto 0);
begin
    Bit0: full_adder
        port map (B(0), A(0), C(0), S(0), C(1));
    Bit1: full_adder
        port map (B(1), A(1), C(1), S(1), C(2));
    Bit2: full_adder
        port map (B(2), A(2), C(2), S(2), C(3));
    Bit3: full_adder
        port map (B(3), A(3), C(3), S(3), C(4));
    C(0) <= C0;
    C4 <= C(4);
end structural_4;

```

Fig. 3-42 Hierarchical Structural/Dataflow Description of a 4-Bit Adder (Continued)

```

-- 4-bit Adder: Behavioral Description
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity adder_4_b is
    port (B, A : in std_logic_vector(3 downto 0);
          C0 : in std_logic;
          S : out std_logic_vector(3 downto 0);
          C4: out std_logic);
end adder_4_b;

architecture behavioral of adder_4_b is
    signal sum : std_logic_vector(4 downto 0);
begin
    sum <= ('0' & A) + ('0' & B) + ("0000" & C0);
    C4 <= sum(4);
    S <= sum(3 downto 0);
end behavioral;

```

Fig. 3-43 Behavioral Description of a 4-Bit Adder

```
// 2-to-4 Line Decoder: Structural Verilog Description      // 1
// (See Figure 3-14 for logic diagram)                    // 2
module decoder_2_to_4_st_v(E_n, A0, A1, D0_n, D1_n, D2_n, D3_n); // 3
    input E_n, A0, A1;                                     // 4
    output D0_n, D1_n, D2_n, D3_n;                        // 5
                                                         // 6
    wire A0_n, A1_n;                                     // 7
    not                                           // 8
        go(A0_n, A0),                                   // 9
        g1(A1_n, A1);                                  // 10
        g2(E, E_n);                                    // 11
                                                         // 12
    nand                                           // 13
        g3(D0_n, A0_n, A1_n, E),                       // 14
        g4(D1_n, A0, A1_n, E),                       // 15
        g5(D2_n, A0_n, A1, E),                       // 16
        g6(D3_n, A0, A1, E);                          // 17
                                                         // 18
endmodule                                           // 19
```

Fig. 3-44 Structural Verilog Description of a 2-to-4 Line Decoder

```

// 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; // 9
    // 10
not // 11
    gn0(not_S[0], S[0]), // 12
    gn1(not_S[1], S[1]); // 13
    // 14
and // 15
    g0(N[0], not_S[1], not_S[0], D[0]), // 16
    g1(N[1], not_S[1], S[0], D[1]), // 17
    g2(N[2], S[1], not_S[0], D[2]), // 18
    g3(N[3], S[1], S[0], D[3]); // 19
    // 20
or go(Y, N[0], N[1], N[2], N[3]); // 21
    // 22
endmodule // 23

```

Fig. 3-45 Structural Verilog Description of a 4-to-1 Line Multiplexer

```
// 2-to-4 Line Decoder: Dataflow Verilog Description           // 1
// (See Figure 3-14 for logic diagram)                         // 2
module decoder_2_to_4_df_v(E_n, A0, A1, D0_n, D1_n, D2_n, D3_n); // 3
    input E_n, A0, A1;                                           // 4
    output D0_n, D1_n, D2_n, D3_n;                               // 5
                                                                // 6
    assign D0_n = ~(~E_n & ~A1 & ~A0);                          // 7
    assign D1_n = ~(~E_n & ~A1 & A0);                           // 8
    assign D2_n = ~(~E_n & A1 & ~A0);                          // 9
    assign D3_n = ~(~E_n & A1 & A0);                           // 10
                                                                // 11
endmodule                                                       // 12
```

Fig. 3-46 Dataflow Verilog Description of a 2-to-4 Line Decoder

TABLE 3-10
Bitwise Verilog Operators

Operation	Operator
~	Bitwise NOT
&	Bitwise AND
	Bitwise OR
^	Bitwise XOR
^~ or ~^	Bitwise XNOR

Table 3-10 Bitwise Verilog Operators


```

// 4-to-1 Line Multiplexer: Dataflow Verilog Description
// (See Figure 3-19 for logic diagram)
module multiplexer_4_to_1_df_v(S, D, Y);
    input [1:0] S;
    input [3:0] D;
    output Y;

    assign Y = (~ S[1] & ~ S[0] & D[0]) | (~ S[1] & S[0] & D[1])
               | (S[1] & ~ S[0] & D[2]) | (S[1] & S[0] & D[3]);
endmodule

```

Fig. 3-47 Dataflow Verilog Description of a 4-to-1 Line Multiplexer Using a Boolean Equation

```
// 4-to-1 Line Multiplexer: Dataflow Verilog Description
// (See Figure 3-19 for function table)
module multiplexer_4_to_1_cf_v(S, D, Y);
    input  [1:0] S;
    input  [3:0] D;
    output Y;

    assign Y = (S == 2'b00) ? D[0] :
               (S == 2'b01) ? D[1] :
               (S == 2'b10) ? D[2] :
               (S == 2'b11) ? D[3] : 1'bx ;
endmodule
```

Fig. 3-48 Conditional Dataflow Verilog Description of a 4-to-1 Line Multiplexer Using Combinations

```
// 4-to-1 Line Multiplexer: Dataflow Verilog Description
// (See Figure 3-19 for logic diagram)
module multiplexer_4_to_1_tf_v(S, D, Y);
    input [1:0] S;
    input [3:0] D;
    output Y;

    assign Y = S[1] ? (S[0] ? D[3] : D[2]) :
               (S[0] ? D[1] : D[0]) ;
endmodule
```

Fig. 3-49 Conditional Dataflow Verilog Description of a 4-to-1 Line Multiplexer Using Binary Decisions

```
// 4-bit Adder: Hierarchical Dataflow/Structural
// (See Figures 3-27 and 3-28 for logic diagrams)

module half_adder_v(x, y, s, c);
    input x, y;
    output s, c;

    assign s = x ^ y;
    assign c = x & y;

endmodule

module full_adder_v(x, y, z, s, c);
    input x, y, z;
    output s, c;

    wire hs, hc, tc;

    half_adder_v    HA1(x, y, hs, hc),
                  HA2(hs, z, s, tc);
    assign c = tc | hc;

endmodule

module adder_4_v(B, A, C0, S, C4);
    input[3:0] B, A;
    input C0;
    output[3:0] S;
    output C4;

    wire[3:1] C;

    full_adder_v    Bit0(B[0], A[0], C0, S[0], C[1]),
                  Bit1(B[1], A[1], C[1], S[1], C[2]),
                  Bit2(B[2], A[2], C[2], S[2], C[3]),
                  Bit3(B[3], A[3], C[3], S[3], C4);

endmodule
```

Fig. 3-50 Hierarchical Dataflow/Structural Description of a 4-Bit Adder

```
// 4-bit Adder : Behavioral Verilog Description
```

```
module adder_4_b_v(A, B, C0, S, C4);  
    input [3:0] A, B;  
    input C0;  
    output [3:0] S ;  
    output C4;  
  
    assign {C4, S } = A + B + C0;  
endmodule
```

Fig. 3-51 Behavioral Description of a 4-Bit Adder Using Verilog