CHAPTER 4

4-1.

$$*S_0 = C_0\overline{A_0}\,\overline{B_0} + \overline{C_0}A_0\overline{B_0} + \overline{C_0}\,\overline{A_0}B_0 + C_0A_0B_0$$

$$S_1 = C_1\overline{A_1}\,\overline{B_1} + \overline{C_1}A_1\overline{B_1} + \overline{C_1}\,\overline{A_1}B_1 + C_1A_1B_1$$

$$\begin{aligned}*S_1 = {} & C_0A_0\overline{A_1}\,\overline{B_1} + A_0B_0\overline{A_1}\,\overline{B_1} + C_0B_0\overline{A_1}\,\overline{B_1} \\ & + \overline{C_0}\,\overline{A_0}A_1\overline{B_1} + \overline{A_0}\,\overline{B_0}A_1\overline{B_1} + \overline{C_0}\,\overline{B_0}A_1\overline{B_1} \\ & + \overline{C_0}\,\overline{A_0}\,\overline{A_1}B_1 + \overline{A_0}\,\overline{B_0}\,\overline{A_1}B_1 + \overline{C_0}\,\overline{B_0}\,\overline{A_1}B_1 \\ & + C_0A_0A_1B_1 + A_0B_0A_1B_1 + C_0B_0A_1B_1\end{aligned}$$

$$C_1 = C_0A_0 + A_0B_0 + C_0B_0$$

$$C_2 = C_1A_1 + A_1B_1 + C_1B_1$$

$$\begin{aligned}*C_2 = {} & C_0A_0A_1 + A_0B_0A_1 + C_0B_0A_1 \\ & + C_0A_0B_1 + A_0B_0B_1 + C_0B_0B_1 + A_1B_1\end{aligned}$$

* These are the three equations for the outputs. The logic diagram consists of a sum-of-products implementation of these equations.

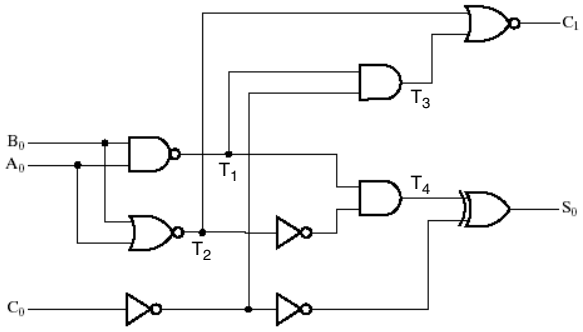
4-2.*

$$C_1 = \overline{T_3 + T_2} = \overline{T_1\overline{C_0} + T_2} = \overline{\overline{A_0B_0}\,\overline{C_0} + \overline{A_0 + B_0}} = \overline{(\overline{A_0} + \overline{B_0})\overline{C_0} + \overline{A_0}\,\overline{B_0}} = (A_0B_0 + C_0)(A_0 + B_0)$$

$$C_1 = A_0B_0 + A_0C_0 + B_0C_0$$

$$S_0 = C_0 \oplus T_4 = C_0 \oplus T_1\overline{T_2} = C_0 \oplus \overline{A_0B_0}(A_0 + B_0) = C_0 \oplus (\overline{A_0} + \overline{B_0})(A_0 + B_0) = C_0 \oplus A_0\overline{B_0} + \overline{A_0}B_0$$

$$S_0 = A_0 \oplus B_0 \oplus C_0$$



4-3.*(5-3)

Unsigned	1001 1100	1001 1101	1010 1000	0000 0000	1000 0000
1's Complement	0110 0011	0110 0010	0101 0111	1111 1111	0111 1111
2's Complement	0110 0100	0110 0011	0101 1000	0000 0000	1000 0000

4-4.(5-4)

a)	11010	b)	11110	c)	1111110	d)	101001
+	01111	+	10010	+	0000010	+	111011
	01001		10000		0000000		100100

4-5.

a)	11010	b)	11110	c)	1111110	d)	101001
+	01111	+	00010	+	0000010	+	000011
	01001		00000		0000000		101100

4-6.*

+36	=	0100100	36		0100100
- 24	=	1101000	+(–24)	+	1101000
- 35	=	1011101			10001100
			= 12	=	0001100
			–35		1011101
			- (–24)	+	0011000
			= –11	=	1110101

4-7.

a)			**b)**			**c)**			**d)**		
	100111	-25		001011	11		110001	-15		101110	-18
+	111001	-7	+	100110	-26	+	101110	-18	+	001001	9
	100000	-32		110001	-15		011111	-33		110111	-9
							Overflow				

4-8.+

a) $H = D$

$G = C \oplus D$

$F = \overline{B}C + \overline{B}D + B\overline{C}\,\overline{D}$

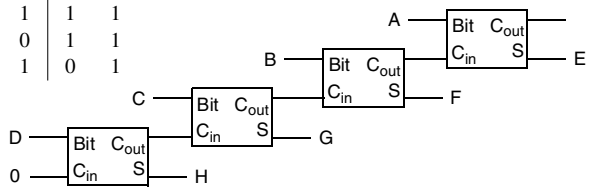
$E = \overline{A}B + A\overline{B}\,\overline{C}\,\overline{D} + \overline{A}C + \overline{A}D$

b)

Bit	Cin	S	Cout
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	1

$S = Bit \oplus Cin$

$Cout = Bit + Cin$



A	B	C	D	E	F	G	H
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1
0	1	0	0	1	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

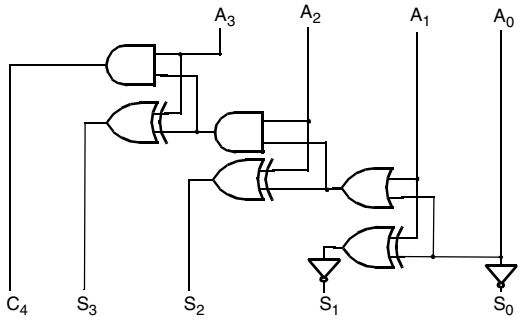
c) Using shared inverters, XOR cost = 6.

Gate input cost for a = 4 + 0 + 6 + 10 + 14 = 34

Gate input cost for b = 4 x (2 + 6 + 2) = 40

In terms of gate cost, a is the better design.

4-9.



4-10.

$B_0 = 0$ $\qquad C_0 = 0$

$S_0 = A_0 \oplus S \oplus S = A_0$

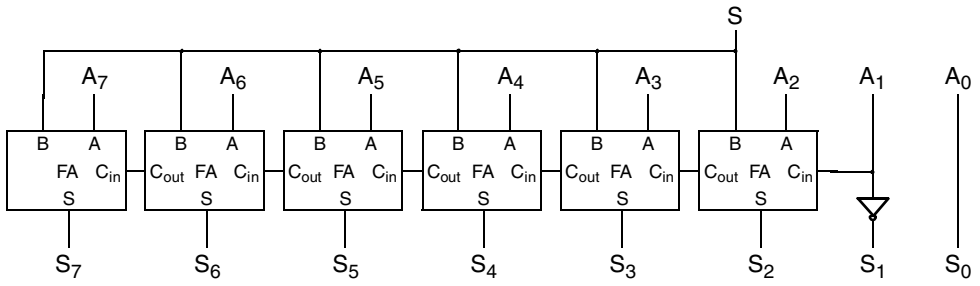
$C_1 = A_0(B_0 \oplus S) + (B_0 \oplus S)C_0 + A_0C_0 = A_0S + S \cdot S + S \cdot S = S$

$B_1 = 1$

$S_1 = A_1 \oplus \bar{S} \oplus S = \overline{A_1}$ $\qquad C_2 = A_1\bar{S} + A_1S + S \cdot \bar{S} = A_1$

$B_{2-7} = S$

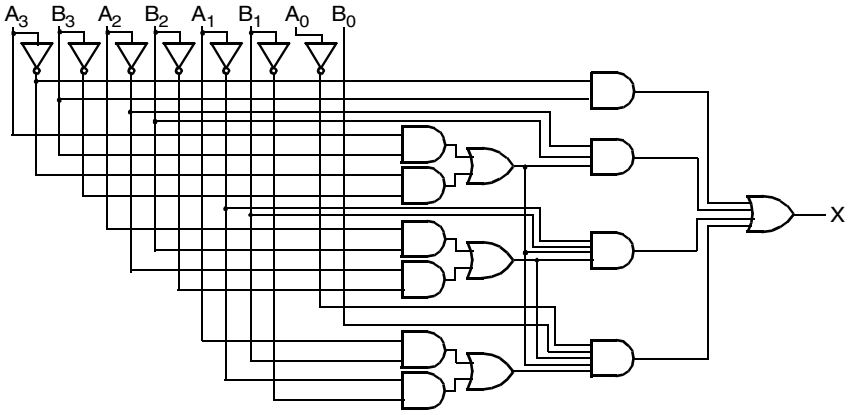
Bits 2-6 use regular full adder/subtractor logic. For bit 7, the carry logic is omitted.



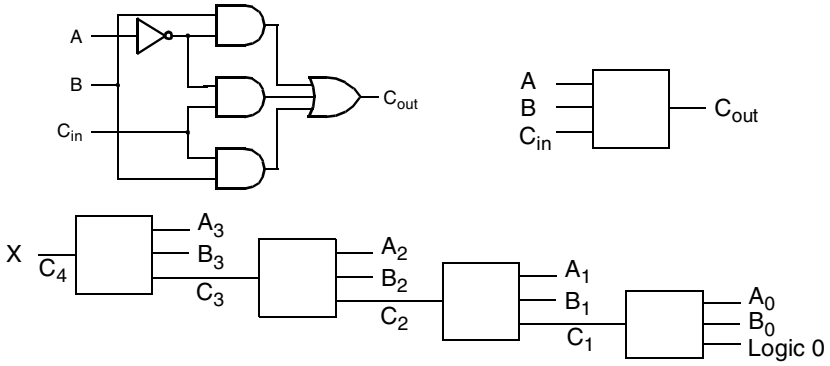
4-11.

Proceeding from MSB to LSB: $A < B$ if $A_i < B_i(\overline{A}_i B_i = 1)$ and for all j > i, $A_j = B_j(A_j B_j + \overline{A}_j \overline{B}_j = 1)$
Based on the above,

$$X = \overline{A}_3 B_3 + (A_3 B_3 + \overline{A}_3 \overline{B}_3)\overline{A}_2 B_2 + (A_3 B_3 + \overline{A}_3 \overline{B}_3)(A_2 B_2 + \overline{A}_2 \overline{B}_2)\overline{A}_1 B_1$$
$$+ (A_3 B_3 + \overline{A}_3 \overline{B}_3)(A_2 B_2 + \overline{A}_2 \overline{B}_2)(A_1 B_1 + \overline{A}_1 \overline{B}_1)\overline{A}_0 B_0$$



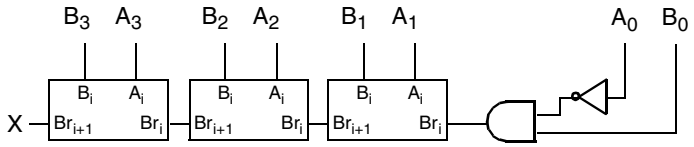
4-12.+



4-13.

In a subtractor, the sum is replaced by the difference and the carry is replaced by the borrow. The borrow at any given point is a 1 only if in the LSB direction from that point, $A < B$.

Only borrow logic is needed to produce X, so the difference logic is discarded in contraction.The remaining equation for borrow into the i + 1 position is: $Br_{i+1} = \overline{A}_i B_i + \overline{A}_i Br_i + B_i Br_i$ for i = 0,1,2,3. $Br_0 = 0$ giving $Br_1 = \overline{A}_0 B_0$ When the borrow $Br_4 = 1$, then $A < B$. Thus, $X = Br_4$. The resulting circuit using the borrow logic is:



4-14.+

This problem requires two decisions: Is A > B? Is A = B? Two "carry" lines are required to build an iterative circuit, G_i and E_i. These carries are assumed to pass through the circuit from right to left with $G_0 = 0$ and $E_0 = 1$. Each cell has inputs A_i, B_i, G_i, and E_i and outputs G_{i+1} and E_{i+1}. Using K-maps, cell equations are:

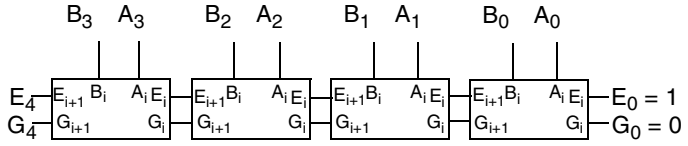
$$E_{i+1} = \overline{A}_i\overline{B}_i\, E_i + A_i B_i\, E_i$$

$$G_{i+1} = A_i\overline{B}_i\, E_i + (A_i + \overline{B}_i)\, E_i$$

Using multilevel circuit techniques, the cost can be reduced by sharing terms:

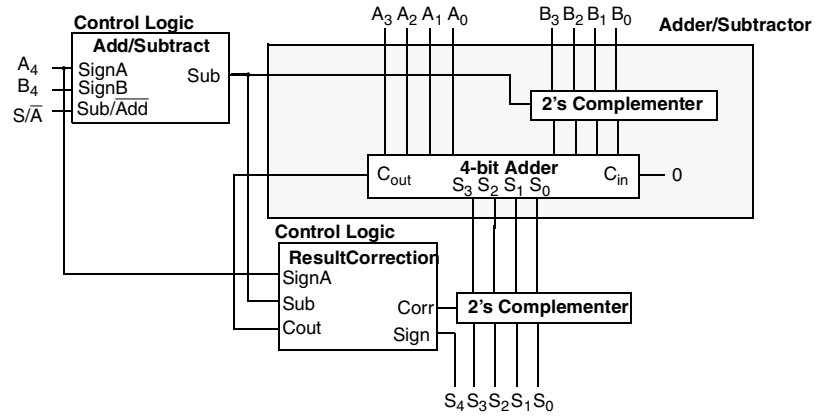
$$E_{i+1} = \overline{(A_i\overline{B}_i + \overline{A}_i B_i)}\, E_i$$

$$G_{i+1} = (A_i\overline{B}_i + \overline{(\overline{A}_i B_i)}\, G_i$$



4-15.+

Circuit Diagram:



Control Logic Truth Tables

Inputs				Inputs					
$S/\overline{A}$	SignA	SignB	Sub	Sub	SignA	Cout	Corr	Sign	Over-flow
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	0	0	1
0	1	0	1	0	1	0	0	1	0
0	1	1	0	0	1	1	0	1	1
1	0	0	1	1	0	0	1	1	0
1	0	1	0	1	0	1	0	0	0
1	1	0	0	1	1	0	1	0	0
1	1	1	1	1	1	1	0	1	0

$\text{Sub} = S/\overline{A} \oplus \text{SignA} \oplus \text{SignB}$

$\text{Corr} = \text{Sub}\ \overline{\text{Cout}}$

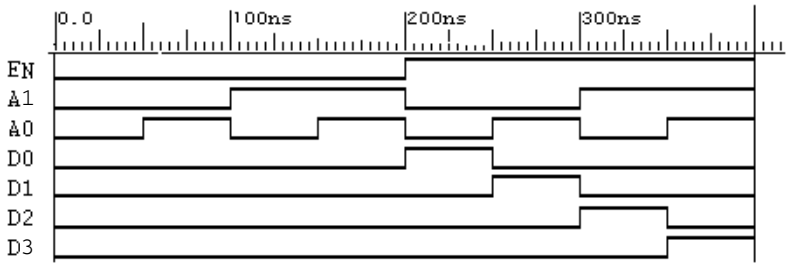
$\text{Sign} = \text{SignA} \oplus \text{Corr}$

$\text{Overflow} = \overline{\text{Sub}}\ \text{Cout}$

4-16.*

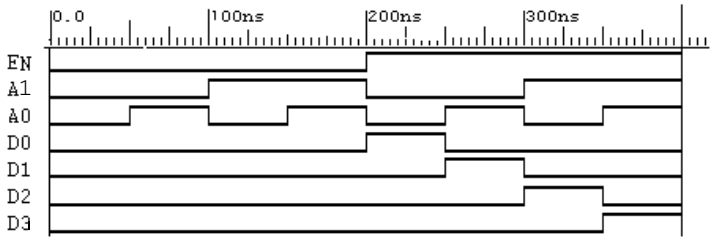
	S	A	B	C_4	S_3	S_2	S_1	S_0
a)	0	0111	0111	0	1	1	1	0
b)	1	0100	0111	0	1	1	0	1
c)	1	1101	1010	1	0	0	1	1
d)	0	0111	1010	1	0	0	0	1
e)	1	0001	1000	0	1	0	0	1

4-17.

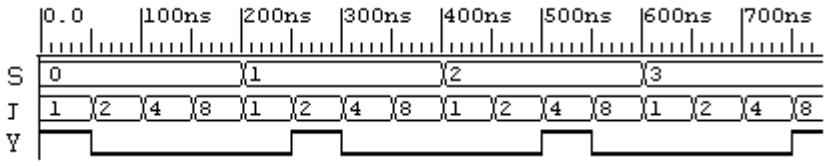


4-18.

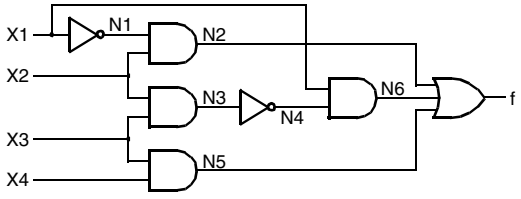
```
entity decoder_2_to_4 is
  port(EN: in std_logic;
       A: in std_logic_vector(0 to 1);
       D: out std_logic_vector(0 to 3));
end decoder_2_to_4;
...
signal not_A: std_logic_vector(0 to 1);
begin
  g0: NOT1 port map (A(0), not_A(0));
  g1: NOT1 port map (A(1), not_A(1));
  g2: AND3 port map (not_A(0), not_A(1), EN, D(0));
  g3: AND3 port map (A(0), not_A(1), EN, D(1));
  g4: AND3 port map (not_A(0), A(1), EN, D(2));
  g5: AND3 port map (A(0), A(1), EN, D(3));
end structural_1;
```



4-19.



4-20.*

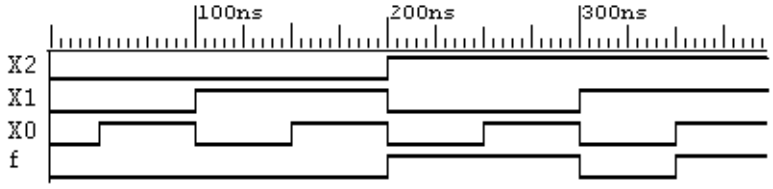


4-21.

```
-- Figure 4-40: Structural VHDL Description
library ieee;
use ieee.std_logic_1164.all;
entity nand2 is
   port(in1, in2: in std_logic;
        out1 : out std_logic);
end nand2;

architecture concurrent of nand2 is
begin
   out1 <= not (in1 and in2);
end architecture;

library ieee;
use ieee.std_logic_1164.all;
entity nand3 is
   port(in1, in2, in3 : in std_logic;
        out1 : out std_logic);
end nand3;

architecture concurrent of nand3 is
begin
   out1 <= not (in1 and in2 and in3);
end concurrent;

library ieee;
use ieee.std_logic_1164.all;
entity nand4 is
   port(in1, in2, in3, in4: in std_logic;
        out1 : out std_logic);
end nand4;
-- The code above this point could be eliminated by using the library, func_prims.

library ieee;
use ieee.std_logic_1164.all;
entity fig440 is
   port(X: in std_logic_vector(0 to 2);
        f: out std_logic);
end fig440;
architecture structural_2 of fig440 is

component NAND2
   port(in1, in2: in std_logic;
        out1: out std_logic);
end component;
```

```
component NAND3
   port(in1, in2, in3: in std_logic;
        out1: out std_logic);
end component;

signal T: std_logic_vector(0 to 4);
begin
       g0: NAND2 port map (X(0),X(1),T(0));
       g1: NAND2 port map (X(0),T(0),T(1));
       g2: NAND2 port map (X(1),T(0),T(2));
       g3: NAND3 port map (X(2),T(1),T(2),T(3));
       g4: NAND2 port map (X(2),T(2),T(4));
       g5: NAND2 port map (T(3),T(4),f);
end structural_2;
```

$$F = X_0X_2 + \overline{X}_1X_2$$

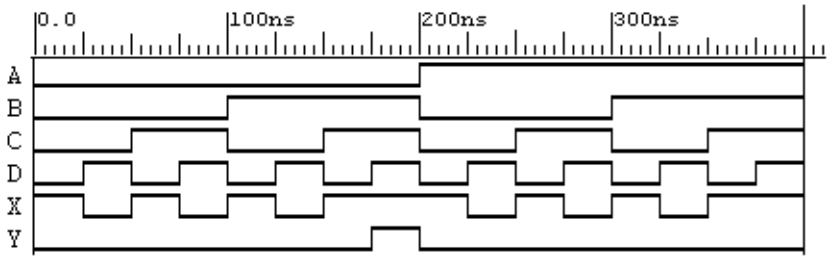


4-22.

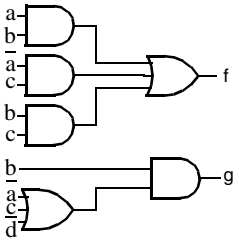
```
begin
   g0: NOT_1 port map (D, x1);
   g1: AND_2 port map (B, C, x2);
   g2: NOR_2 port map (A, x1, x3);
   g3: NAND_2 port map (x1, x3, x4);
   g4: OR_2 port map (x1, x2, x5);
   g5: AND_2 port map (x4, x5, X);
   g6: AND_2 port map (x3, x5, Y);
end structural_1;
```

$X = \overline{D} + BC$

$Y = \overline{A}\ BCD$



4-23.

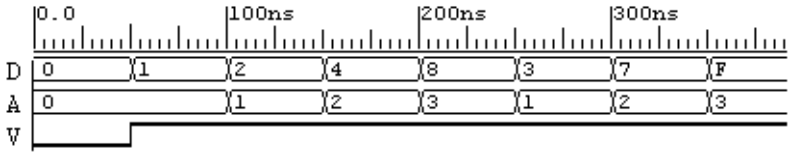


4-24.*

```
begin
    F <= (X and Z) or ((not Y) and Z);
end;
```

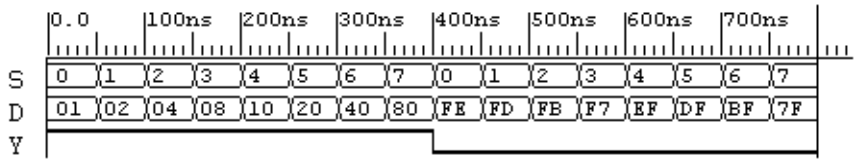
4-25.+

```
library IEEE;
use IEEE.std_logic_1164.all;

entity priority_4 is
      port (
            D: in STD_LOGIC_VECTOR (3 downto 0);
            A: out STD_ULOGIC_VECTOR (1 downto 0);
            V: out STD_LOGIC
            );
end priority_4;

architecture priority_4_arch of priority_4 is
begin
      V <= '0' when D ="0000" else '1';
      A <=  "11" when D(3) = '1' else
            "10" when D(2) = '1' else
            "01" when D(1) = '1' else
            "00" when D(0) = '1' else "00";

end priority_4_arch;
```



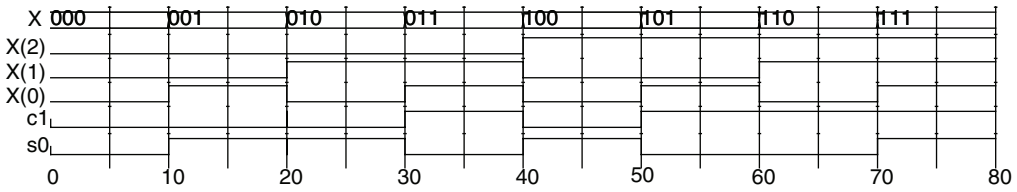
4-26.

```
library ieee;
use ieee.std_logic_1164.all;
entity multiplexer_8_to_1 is
      port(S: in std_logic_vector(2 downto 0);
            D: in std_logic_vector(7 downto 0);
            Y: out std_logic);
end multiplexer_8_to_1;

architecture function_table of multiplexer_8_to_1 is
signal not_S: std_logic_vector(0 to 1);
signal N: std_logic_vector(0 to 3);
begin
     with S select
      Y <=   D(0) when "000"
            D(1) when "001"
            D(2) when "010"
            D(3) when "011"
            D(4) when "100"
            D(5) when "101"
            D(6) when "110"
            D(7) when "111"
            'X';
end function_table;
```

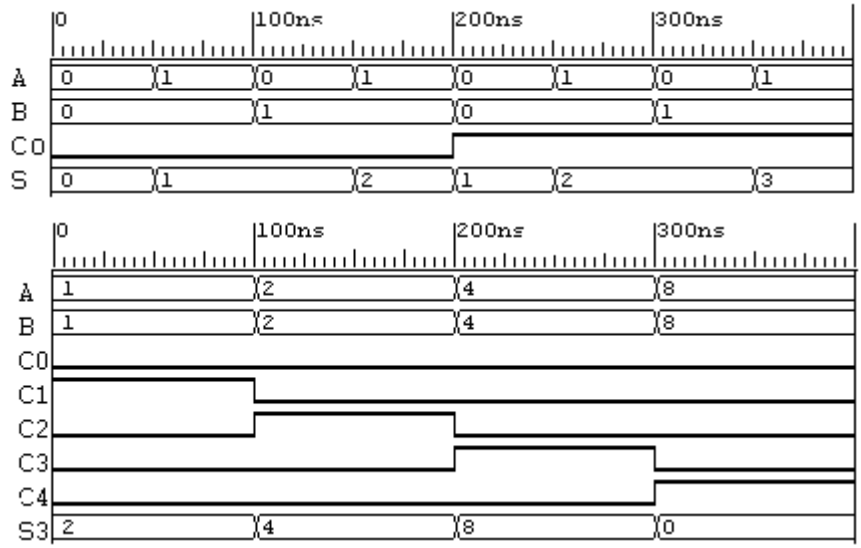


4-27.

```
-- Full Adder: Structural VHDL Description
-- (See Figure 4-28 for logic diagram)
library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all, lcdf_vhdl.func_prims.all; --X is input vector (A0, B0, C0).
entity full_adder_st is
   port(X: in std_logic_vector(0 to 2);
    C1, S0: out std_logic);
end;
architecture logic of full_adder_st is
component NOT1
   port(in1: in std_logic;
        out1: out std_logic);
end component;
component NAND2
   port(in1, in2: in std_logic;
        out1: out std_logic);
end component;
component NOR2
   port(in1, in2: in std_logic;
   out1: out std_logic);
end component;
component AND2
   port(in1, in2: in std_logic;
        out1: out std_logic);
end component;
component XOR2
   port(in1, in2: in std_logic;
        out1: out std_logic);
end component;
signal S: std_logic_vector(0 to 6); -- S(0 to 6) is the vector of six gate output signals
                                    -- from upper left to lower right.
--The following is the circuit netlist.
begin
   g0: NAND2 port map (X(1), X(0), S(0));
   g1: NOR2 port map (X(1), X(0), S(1));
   g2: NOT1 port map (X(2), S(2));
   g3: AND2 port map (S(0), S(2), S(3));
   g4: NOT1 port map (S(1), S(4));
   g5: NOT1 port map (S(2), S(5));
   g6: AND2 port map (S(0), S(4), S(6));
   g7: NOR2 port map (S(1), S(3), Z(1));
   g8: XOR2 port map (S(6), S(5), Z(0));
end logic;
```

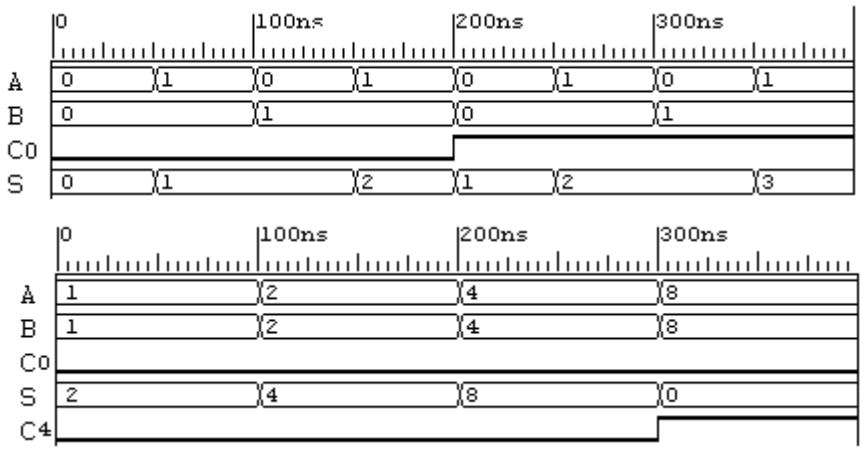


4-28.



The solution given is very thorough since it checks each of the carry connections between adjacent cells transferring 0 and 1. In contrast a test applying C0 = 1 and A = 15 with B = 0 would allow a whole variety of incorrect connections between cells that would not be detected.

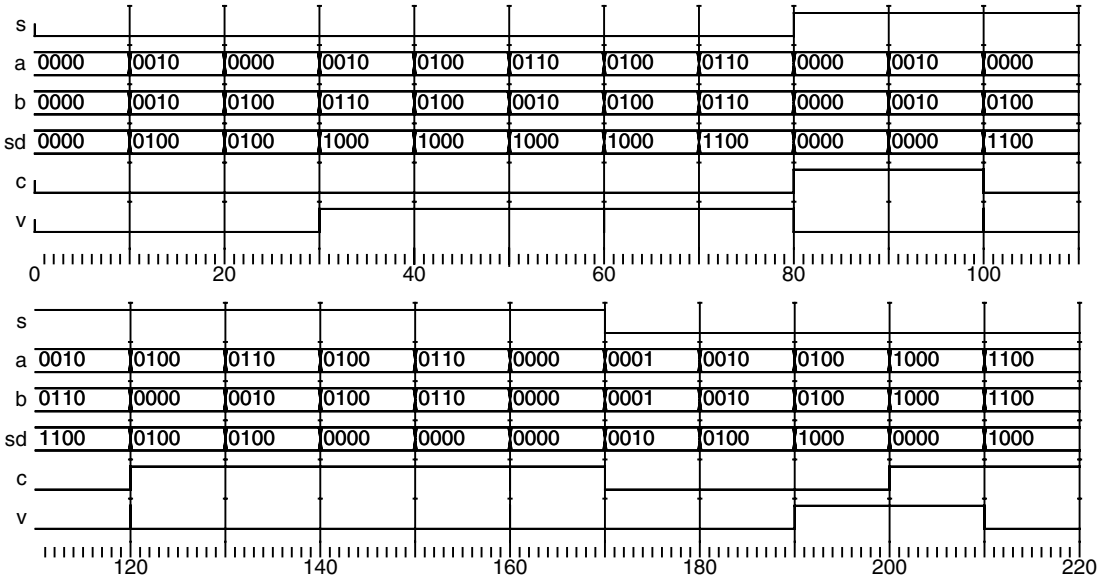
4-29.*



The solution given is very thorough since it checks each of the carry connections between adjacent cells transferring 0 and 1. In contrast a test applying C0 = 1 and A = 15 with B = 0 would allow a whole variety of incorrect connections between cells that would not be detected.

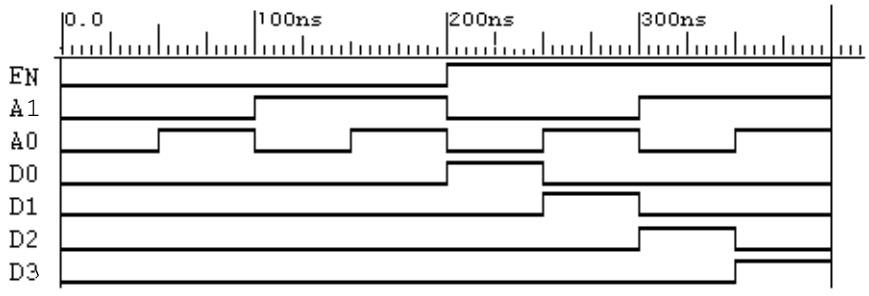
4-30.+

```
-- prob_4-30 adder-subtractor with overflow detection
library ieee;
use ieee.std_logic_1164.all, ieee.std_logic_unsigned.all;
entity addsubov is
 port(A,B: in std_logic_vector(3 downto 0);
      S: in std_logic;
      SD: out std_logic_vector(3 downto 0);
      C, V: out std_logic);
end addsubov;
architecture behavior of addsubov is
signal Y: std_logic_vector(3 downto 0);
signal temp4: std_logic_vector(3 downto 0);
signal temp2: std_logic_vector(4 downto 3);
signal CI: std_logic_vector(4 downto 3);
begin
 with S select
  Y(2 downto 0) <= B(2 downto 0) when '0',
                   not B(2 downto 0) when '1',
                   "XXX" when others;
 with S select
  Y(3) <= B(3) when '0',
          not B(3) when '1',
          'X' when others;
 temp4 <= ('0' & A(2 downto 0)) + ('0' & Y(2 downto 0)) + ("000" & S);
 CI(3) <= temp4(3);
 temp2 <= ('0' & A(3)) + ('0' & Y(3)) + ('0' & CI(3));
 CI(4) <= temp2(4);
 SD <= (temp2(3) & temp4(2 downto 0));
 C <= CI(4);
 V <= CI(4) xor CI (3);
end behavior;
```



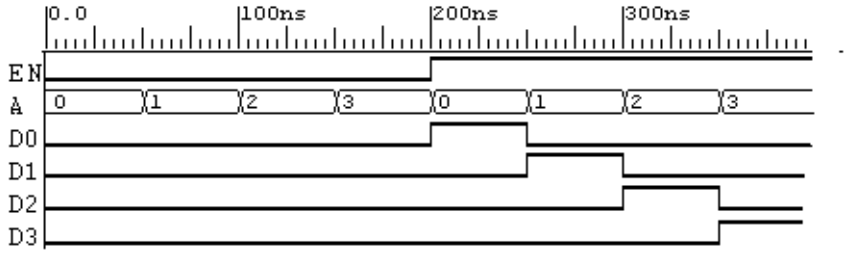
The solution given is very thorough since it checks each of the carry connections between adjacent cells transferring 0 and 1. In contrast a test applying C0 = 1 and A = 15 with B = 0 would allow a whole variety of incorrect connections between cells that would not be detected.

4-31.*(Errata: Replace “E” with “EN”.)

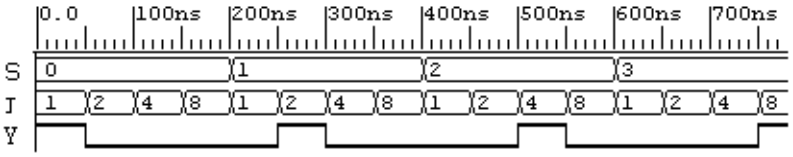


4-32.(4-47)

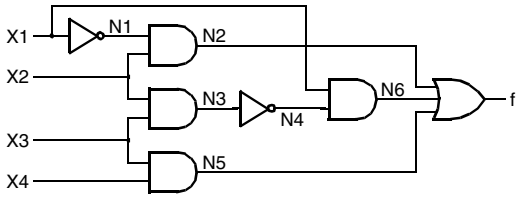
```
module decoder_2_to_4_st(A, EN, D);
  input [1:0] A;
  input EN;
  output [3:0] D;
  wire [1:0] not_A;
  not
    go(not_A[0], A[0]),
    g1(not_A[1], A[1]);
      nand
    g2(D[0], not_A[0], not_A[1], EN),
    g3(D[1],A[0], not_A[1], EN),
    g4(D[2], not_A[0], A[1], EN),
    g5(D[3], A[0],A[1], EN);
endmodule

end structural_1;
```



4-33.

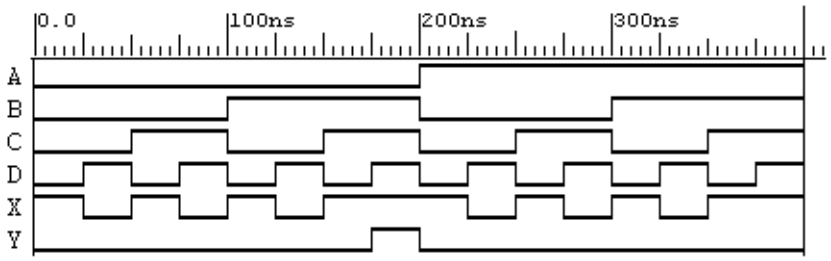


4-34.*



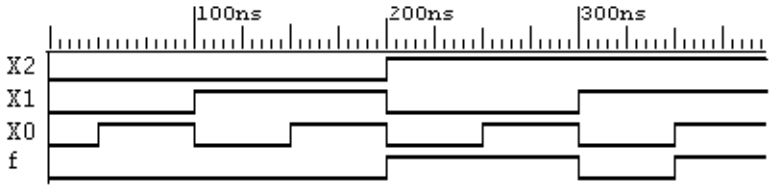
4-35.

```
module circuit_4_50(A, B, C, D, X, Y);
    input A, B, C, D;
    output X, Y;

    wire n1, n2, n3, n4, n5;

    not
        go(n1, D);

    nand
        g1(n4, n1, n3);

    and
        g2(n2, B, C),
        g3(X, n4, n5),
        g4(Y, n3, n5);

    or
        g5(n5, n1, n2);

    nor
        g6(n3, n1, A);

endmodule
```

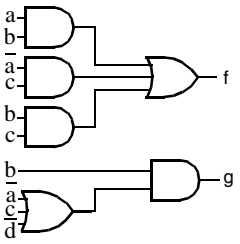


4-36. (Errata: Replace “4-33” with “4-21”)

```
module circuit_4_51(X, F);
    input [2:0] X;
    output F;

    wire [0:4] T;

    nand
        g0(T[0],X[0],X[1]),
        g1(T[1],X[0],T[0]),
        g2(T[2],X[1],T[0]),
        g3(T[3],X[2],T[1],T[2]),
        g4(T[4],X[2],T[2]),
        g5(F,T[3],T[4]);
endmodule
```



4-37.(Errata: Replace “4-34” with “4-33.”)

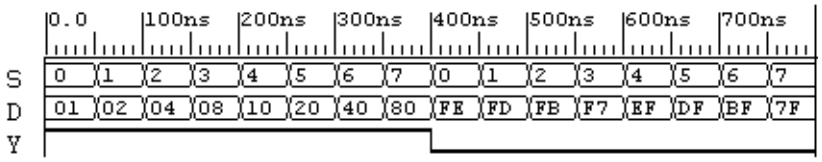


4-38.*

```
module circuit_4_53(X, Y, Z, F);
  input X, Y, Z;
  output F;
  assign F = (X & Z) | (Z & ~Y);
endmodule
```

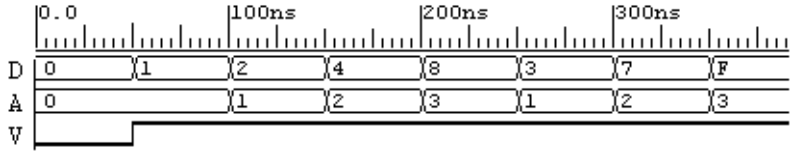
4-39.

```
module multiplexer_8_to_1_cf_v(S, D, Y);
  input [2:0] S;
  input [7:0] D;
  output Y;

  assign Y = (S == 3'b000) ? D[0] :
             (S == 3'b001) ? D[1] :
             (S == 3'b010) ? D[2] :
             (S == 3'b011) ? D[3] :
             (S == 3'b100) ? D[4] :
             (S == 3'b101) ? D[5] :
             (S == 3'b110) ? D[6] :
             (S == 3'b111) ? D[7] : 1'bx ;
endmodule
```



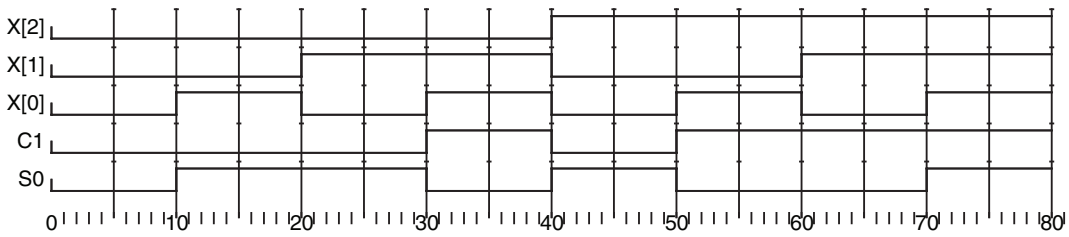
4-40.+

```
module prioencoder_4_to_1_(D, A, V);
  input [3:0] D;
  output [1:0] A;
  output V;

  assign V = D[0] | D[1] | D[2] | D[3];

  assign A[0] = D[3] ? 1'b1:(D[2] ? 1'b0:(D[1] ? 1'b1:(D[0] ? 1'b0:1'bx)));
  assign A[1] = D[3] ? 1'b1:(D[2] ? 1'b1:(D[1] ? 1'b0:(D[0] ? 1'b0:1'bx)));
endmodule
```

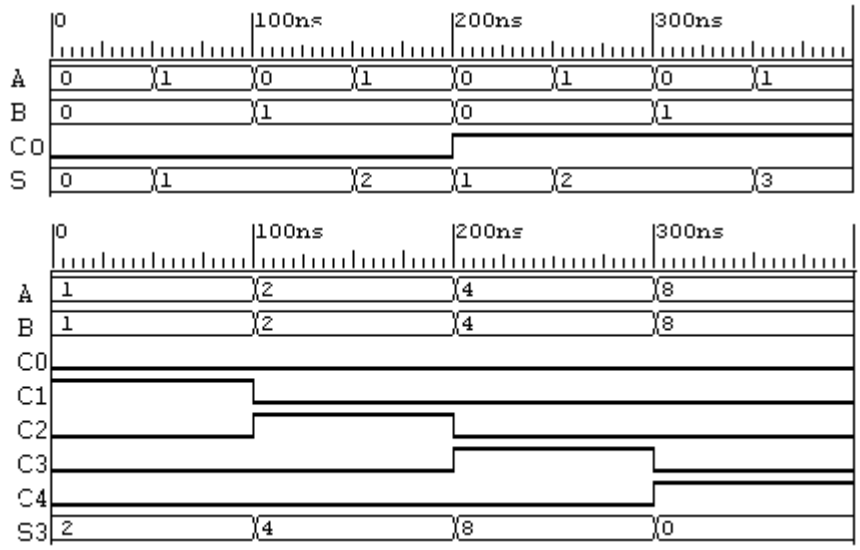


4-41.

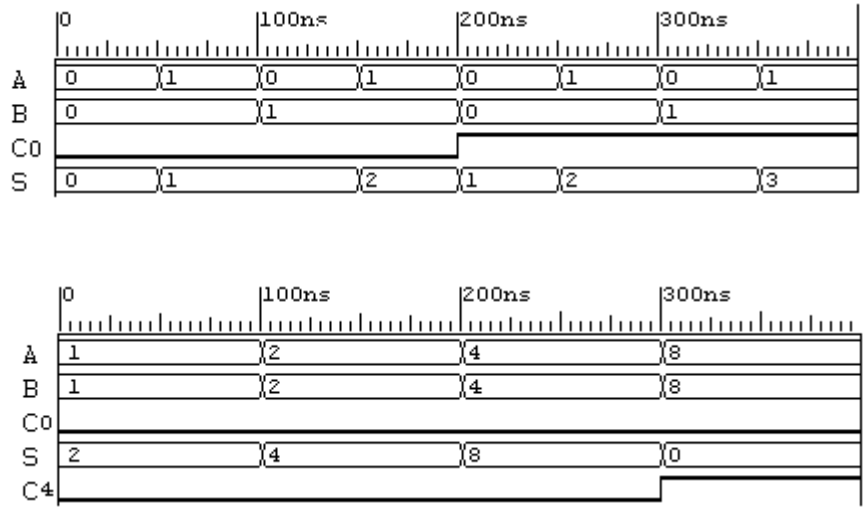
```
// Full Adder: Structural Verilog Description
// (See Figure 4-28 for logic diagram)
module full_adder_st(C1, S0, X);
  input [2:0] X; //X is the vector of inputs (A0, B0, C0).
  output C1, S0;
  wire [0:6] N; //N[0:6] is the six bit vector of gate
                //outputs from upper left to lower right.
//The netlist for the gate types:
  nand
    gna(N[0],X[1],X[0]);
  nor
    gno1(N[1],X[1],X[0]),
    gno2(C1,N[1],N[3]);
  not
    gn0(N[2], X[2]),
    gn1(N[4], N[1]),
    gn2(N[5], N[2]);
  and
    ga0(N[3], N[0], N[2]),
    ga1(N[6], N[0], N[4]);
  xor
    gx(S0, N[5], N[6]);
endmodule
```



4-42.



4-43.*



The solution given is very thorough since it checks each of the carry connections between adjacent cells transferring 0 and 1. In contrast a test applying C0 = 1 and A = 15 with B = 0 would allow a whole variety of incorrect connections between cells that would not be detected.

4-44.

```
// Adder-Subtractor Behavioral Model
module addsub_4b_v (S, A, B, SD, C4);
  input[3:0] A, B;
  input S;
  output[3:0] SD;
  output C4;

  assign {C4, SD} = S?(A - B):(A + B);
endmodule
```