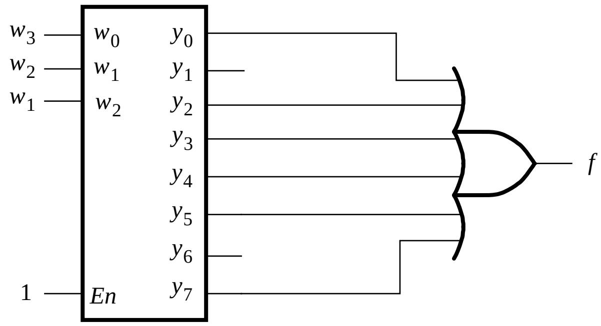
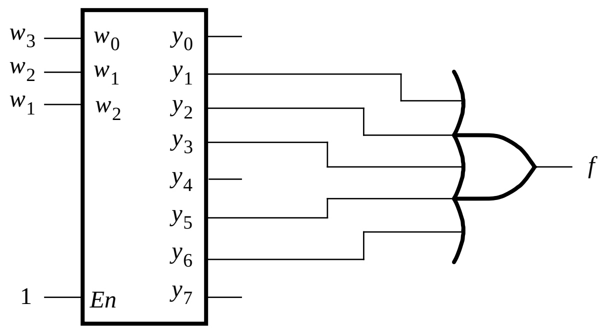
Chapter 6

6.1.



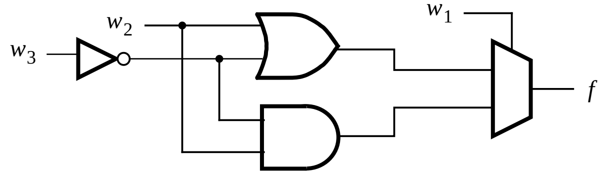
6.2.



6.3.

w_1	w_2	w_3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

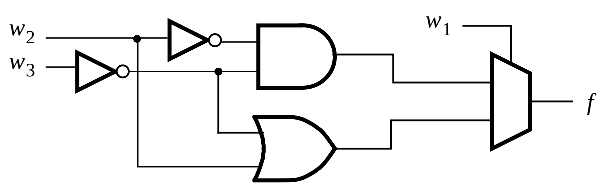
w_1	f
0	$w_2 + \overline{w}_3$
1	$w_2\overline{w}_3$



6.4.

w_1	w_2	w_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

w_1	f
0	$\overline{w}_2\overline{w}_3$
1	$w_2 + \overline{w}_3$



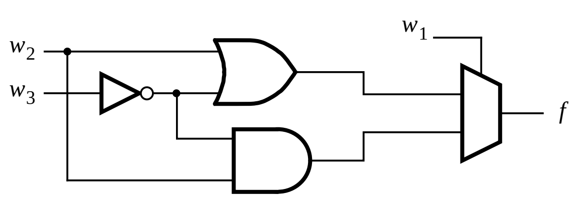
6.5. The function f can be expressed as

$$f = \overline{w}_1\overline{w}_2\overline{w}_3 + \overline{w}_1 w_2\overline{w}_3 + \overline{w}_1 w_2 w_3 + w_1 w_2\overline{w}_3$$

Expansion in terms of w_1 produces

$$f = \overline{w}_1(w_2 + \overline{w}_3) + w_1(w_2\overline{w}_3)$$

The corresponding circuit is



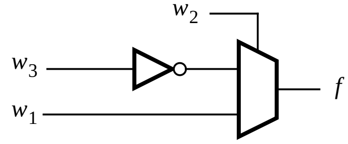
6.6. The function f can be expressed as

$$f = \overline{w}_1\overline{w}_2\overline{w}_3 + w_1\overline{w}_2\overline{w}_3 + w_1 w_2\overline{w}_3 + w_1 w_2 w_3$$

Expansion in terms of w_2 produces

$$f = \overline{w}_2(\overline{w}_3) + w_2(w_1)$$

The corresponding circuit is



6.7. Expansion in terms of w_2 gives

$$\begin{aligned} f &= \overline{w}_2(1+\overline{w}_1\overline{w}_3+w_1w_3)+w_2(\overline{w}_1\overline{w}_3+w_1w_3) \\ &= \overline{w}_1\overline{w}_2\overline{w}_3+w_1\overline{w}_2w_3+\overline{w}_2+\overline{w}_1w_2\overline{w}_3+w_1w_2w_3 \end{aligned}$$

Further expansion in terms of w_1 gives

$$\begin{aligned} f &= \overline{w}_1(w_2\overline{w}_3+\overline{w}_2\overline{w}_3+\overline{w}_2)+w_1(w_2w_3+\overline{w}_2w_3+\overline{w}_2) \\ &= \overline{w}_1w_2\overline{w}_3+\overline{w}_1\overline{w}_2\overline{w}_3+\overline{w}_1\overline{w}_2+w_1w_2w_3+w_1\overline{w}_2w_3+w_1\overline{w}_2 \end{aligned}$$

Further expansion in terms of w_3 gives

$$\begin{aligned} f &= \overline{w}_3(\overline{w}_1w_2+\overline{w}_1\overline{w}_2+\overline{w}_1\overline{w}_2+w_1\overline{w}_2)+w_3(w_1w_2+w_1\overline{w}_2+w_1\overline{w}_2+\overline{w}_1\overline{w}_2) \\ &= \overline{w}_1w_2\overline{w}_3+\overline{w}_1\overline{w}_2\overline{w}_3+w_1\overline{w}_2\overline{w}_3+w_1w_2w_3+w_1\overline{w}_2w_3+\overline{w}_1\overline{w}_2w_3 \end{aligned}$$

6.8. Expansion in terms of w_1 gives

$$f = \overline{w}_1w_2+\overline{w}_1\overline{w}_3+w_1w_2$$

Further expansion in terms of w_2 gives

$$\begin{aligned} f &= \overline{w}_2(\overline{w}_1\overline{w}_3)+w_2(w_1+\overline{w}_1+\overline{w}_1\overline{w}_3) \\ &= \overline{w}_1w_2+\overline{w}_1w_2\overline{w}_3+\overline{w}_1\overline{w}_2\overline{w}_3+w_1w_2 \end{aligned}$$

Further expansion in terms of w_3 gives

$$\begin{aligned} f &= \overline{w}_3(\overline{w}_1\overline{w}_2+w_1w_2+\overline{w}_1w_2+\overline{w}_1w_2)+w_3(w_1w_2+\overline{w}_1w_2) \\ &= \overline{w}_1\overline{w}_2\overline{w}_3+w_1w_2\overline{w}_3+\overline{w}_1w_2\overline{w}_3+\overline{w}_1w_2w_3+w_1w_2w_3 \end{aligned}$$

6.9. Proof of Shannon's expansion theorem

$$f(x_1,x_2,...,x_n)=\overline{x}_1\cdot f(0,x_2,...,x_n)+x_1\cdot f(1,x_2,...,x_n)$$

This theorem can be proved using *perfect induction*, by showing that the expression is true for every possible value of x_1. Since x_1 is a boolean variable, we need to look at only two cases: $x_1=0$ and $x_1=1$.

Setting $x_1=0$ in the above expression, we have:

$$\begin{aligned} f(0,x_2,...,x_n) &= 1\cdot f(0,x_2,...,x_n)+0\cdot f(1,x_2,...,x_n) \\ &= f(0,x_2,...,x_n) \end{aligned}$$

Setting $x_1=1$, we have:

$$\begin{aligned} f(1,x_2,...,x_n) &= 0\cdot f(0,x_2,...,x_n)+1\cdot f(1,x_2,...,x_n) \\ &= f(1,x_2,...,x_n) \end{aligned}$$

This proof can be performed for any arbitrary x_i in the same manner.

6.10. Derivation using $\overline{f}$:

$$\begin{aligned} \overline{f} &= \overline{w}\overline{f}_{\overline{w}}+w\overline{f}_w \\ f &= \left(\overline{\overline{w}\overline{f}_{\overline{w}}+w\overline{f}_w}\right) \\ &= \left(\overline{\overline{w}\overline{f}_{\overline{w}}}\right)\cdot\left(\overline{w\overline{f}_w}\right) \\ &= (w+f_{\overline{w}})(\overline{w}+f_w) \end{aligned}$$

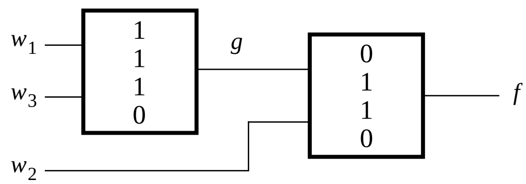
6.11. Expansion in terms of w_2 gives

$$f = \overline{w}_2(\overline{w}_1 + \overline{w}_3) + w_2(w_1 w_3)$$

Letting $g = \overline{w}_1 + \overline{w}_3$, we have

$$f = \overline{w}_2 g + w_2 \overline{g}$$

The corresponding circuit is



6.12. Expansion of f in terms of w_2 gives

$$\begin{aligned} f &= \overline{w}_2(\overline{w}_1 + \overline{w}_3) + w_2(w_1 w_3) \\ &= w_2 \oplus (\overline{w}_1 + \overline{w}_3) \\ &= w_2 \oplus \overline{w_1 w_3} \end{aligned}$$

The cost of this multilevel circuit is 2 gates + 4 inputs = 6.

6.13. Using Shannon's expansion in terms of w_2 we have

$$\begin{aligned} f &= \overline{w}_2(\overline{w}_3 + \overline{w}_1 w_4) + w_2(w_3 \overline{w}_4 + w_1 w_3) \\ &= \overline{w}_2(\overline{w}_3 + \overline{w}_1 w_4) + w_2(w_3(w_1 + \overline{w}_4)) \end{aligned}$$

If we let $g = \overline{w}_3 + \overline{w}_1 w_4$, then

$$f = \overline{w}_2 g + w_2 \overline{g}$$

Thus, two 3-LUTs are needed to implement f.

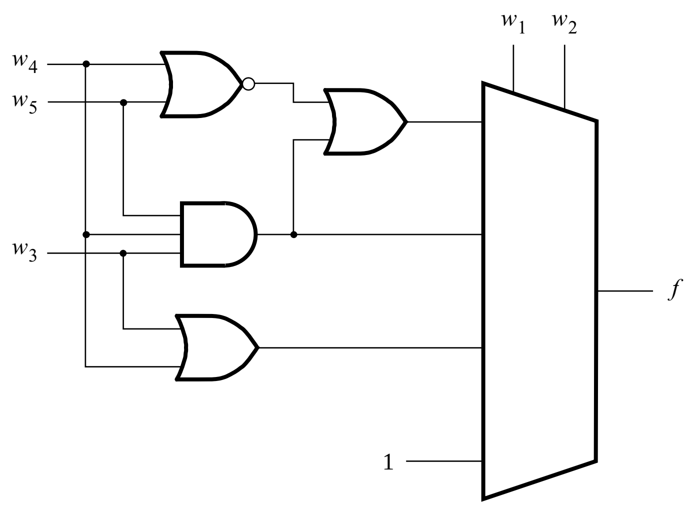
6.14. Any number of 5-variable functions can be implemented by using two 4-LUTs. For example, if we cascade the two 4-LUTs by connecting the output of one 4-LUT to an input of the other, then we can realize any function of the form

$$\begin{aligned} f &= f_1(w_1, w_2, w_3, w_4) + w_5 \\ f &= f_1(w_1, w_2, w_3, w_4) \cdot w_5 \end{aligned}$$

6.15. Shannon's expansion with respect to w_1 and w_2 gives

$$\begin{aligned} f &= \overline{w}_1\overline{w}_2\overline{w}_4\overline{w}_5 + w_1w_2 + w_1w_3 + w_1w_4 + w_3w_4w_5 \\ &= \overline{w}_1\overline{w}_2 f_{\overline{w}_1\overline{w}_2} + \overline{w}_1 w_2 f_{\overline{w}_1 w_2} + w_1\overline{w}_2 f_{w_1\overline{w}_2} + w_1w_2 f_{w_1w_2} \\ &= \overline{w}_1\overline{w}_2(\overline{w}_4\overline{w}_5 + w_3w_4w_5) + \overline{w}_1w_2(w_3w_4w_5) + w_1\overline{w}_2(w_3+w_4) + w_1w_2(1) \end{aligned}$$

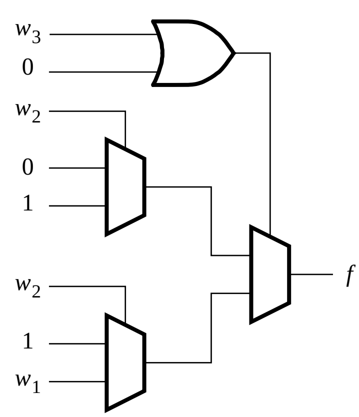
Since only uncomplemented inputs are available, the term $\overline{w}_4\overline{w}_5$ has to be implemented as $\overline{w_4 + w_5}$. The resulting circuit is



6.16. Using Shannon's expansion in terms of w_3 we have

$$\begin{aligned} f &= \overline{w}_3(w_2) + w_3(w_1 + \overline{w}_2) \\ &= \overline{w}_3(w_2) + w_3(\overline{w}_2 + w_2w_1) \end{aligned}$$

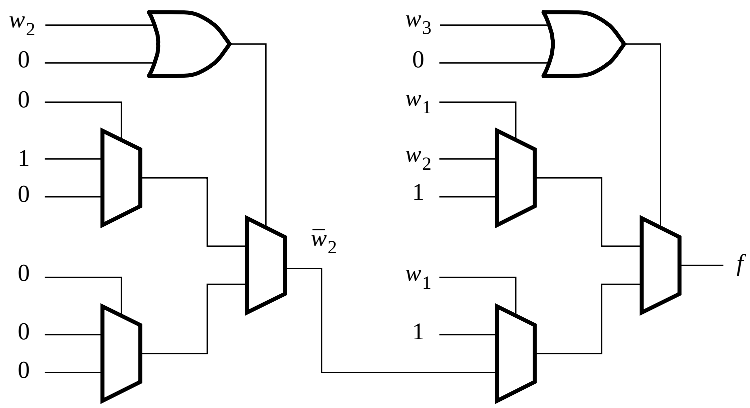
The corresponding circuit is



6.17. Using Shannon's expansion in terms of w_3 we have

$$f = w_3(\overline{w}_1 + w_1\overline{w}_2) + \overline{w}_3(w_1 + \overline{w}_1 w_2)$$

The corresponding circuit is



6.18. The code in Figure P6.2 is a 2-to-4 decoder with an enable input. It is not a good style for defining this decoder. The code is not easy to read. Moreover, the VHDL compiler often turns **if** statements into multiplexers, in which case the resulting decoder may have multiplexers controlled by the En signal on the output side.

6.19.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_19 IS
    PORT ( w   : IN    STD_LOGIC_VECTOR(1 TO 3) ;
           f   : OUT   STD_LOGIC ) ;
END prob6_19 ;

ARCHITECTURE Behavior OF prob6_19 IS
BEGIN
    WITH w SELECT
        f <= '0' WHEN "001",
             '0' WHEN "110",
             '1' WHEN OTHERS ;
END Behavior ;
```

6.20.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_20 IS
    PORT ( w  : IN   STD_LOGIC_VECTOR(1 TO 3) ;
           f  : OUT  STD_LOGIC ) ;
END prob6_20 ;

ARCHITECTURE Behavior OF prob6_20 IS
BEGIN
    WITH w SELECT
        f <= '0' WHEN "000",
             '0' WHEN "100",
             '0' WHEN "111",
             '1' WHEN OTHERS ;
END Behavior ;
```

6.21.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_21 IS
    PORT ( w  : IN   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           y  : OUT  STD_LOGIC_VECTOR(1 DOWNTO 0) ) ;
END prob6_21 ;

ARCHITECTURE Behavior OF prob6_21 IS
BEGIN
    WITH w SELECT
        y <= "00" WHEN "0001",
             "01" WHEN "0010",
             "10" WHEN "0100",
             "11" WHEN OTHERS ;
END Behavior ;
```

6.22.
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_22 IS
    PORT ( w  : IN   STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           y  : OUT  STD_LOGIC_VECTOR(2 DOWNTO 0) ) ;
END prob6_22 ;
```

. . . con't

```
ARCHITECTURE Behavior OF prob6_22 IS
BEGIN
    y <= "000" WHEN w = "00000001" ELSE
         "001" WHEN w = "00000010" ELSE
         "010" WHEN w = "00000100" ELSE
         "011" WHEN w = "00001000" ELSE
         "100" WHEN w = "00010000" ELSE
         "101" WHEN w = "00100000" ELSE
         "110" WHEN w = "01000000" ELSE
         "111" ;
END Behavior ;
```

6.23. First define a set of intermediate variables

$$\begin{aligned}
i_0 &= \overline{w}_7\overline{w}_6\overline{w}_5\overline{w}_4\overline{w}_3\overline{w}_2\overline{w}_1 w_0 \\
i_1 &= \overline{w}_7\overline{w}_6\overline{w}_5\overline{w}_4\overline{w}_3\overline{w}_2 w_1 \\
i_2 &= \overline{w}_7\overline{w}_6\overline{w}_5\overline{w}_4\overline{w}_3 w_2 \\
i_3 &= \overline{w}_7\overline{w}_6\overline{w}_5\overline{w}_4 w_3 \\
i_4 &= \overline{w}_7\overline{w}_6\overline{w}_5 w_4 \\
i_5 &= \overline{w}_7\overline{w}_6 w_5 \\
i_6 &= \overline{w}_7 w_6 \\
i_7 &= w_7
\end{aligned}$$

Now a traditional binary encoder can be used for the priority encoder

$$\begin{aligned}
y_0 &= i_1 + i_3 + i_5 + i_7 \\
y_1 &= i_2 + i_3 + i_6 + i_7 \\
y_2 &= i_4 + i_5 + i_6 + i_7
\end{aligned}$$

6.24.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_24 IS
    PORT ( w  : IN   STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           y  : OUT  STD_LOGIC_VECTOR(2 DOWNTO 0) ;
           z  : OUT  STD_LOGIC ) ;
END prob6_24 ;

ARCHITECTURE Behavior OF prob6_24 IS
BEGIN
    y <= "111" WHEN w(7) = '1' ELSE
         "110" WHEN w(6) = '1' ELSE
         "101" WHEN w(5) = '1' ELSE
         "100" WHEN w(4) = '1' ELSE
         "011" WHEN w(3) = '1' ELSE
         "010" WHEN w(2) = '1' ELSE
         "001" WHEN w(1) = '1' ELSE
         "000" ;
    z <= '0' WHEN w="00000000" ELSE '1' ;
END Behavior ;
```

6.25.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_25 IS
    PORT ( w  : IN   STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           y  : OUT  STD_LOGIC_VECTOR(2 DOWNTO 0) ;
           z  : OUT  STD_LOGIC ) ;
END prob6_25 ;

ARCHITECTURE Behavior OF prob6_25 IS
BEGIN
    PROCESS ( w )
    BEGIN
        IF w(7) = '1' THEN
            y <= "111" ;
        ELSIF w(6) = '1' THEN
            y <= "110" ;
        ELSIF w(5) = '1' THEN
            y <= "101" ;
        ELSIF w(4) = '1' THEN
            y <= "100" ;
        ELSIF w(3) = '1' THEN
            y <= "011" ;
        ELSIF w(2) = '1' THEN
            y <= "010" ;
        ELSIF w(1) = '1' THEN
            y <= "001" ;
        ELSE
            y <= "000" ;
        END IF ;
        IF w = "00000000" THEN
            z <= '0' ;
        ELSE
            z <= '1' ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

6.26.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY if2to4 IS
    PORT ( w   : IN   STD_LOGIC_VECTOR(1 DOWNTO 0) ;
           En  : IN   STD_LOGIC ;
           y   : OUT  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END if2to4 ;
```

. . . con't

```
ARCHITECTURE Behavior OF if2to4 IS
BEGIN
    PROCESS ( En, w )
    BEGIN
        IF En = '0' THEN
            y <= "0000" ;
        ELSE
            IF w = "00" THEN
                y <= "0001" ;
            ELSIF w = "01" THEN
                y <= "0010" ;
            ELSIF w = "10" THEN
                y <= "0100" ;
            ELSE
                y <= "1000" ;
            END IF ;
        END IF ;
    END PROCESS ;
END Behavior ;

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
PACKAGE if2to4_package IS
    COMPONENT if2to4
        PORT ( w    : IN    STD_LOGIC_VECTOR(1 DOWNTO 0) ;
               En   : IN    STD_LOGIC ;
               y    : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
    END COMPONENT ;
END if2to4_package ;


LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.if2to4_package.all ;

ENTITY h3to8 IS
    PORT ( w    : IN    STD_LOGIC_VECTOR(2 DOWNTO 0) ;
           En   : IN    STD_LOGIC ;
           y    : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END h3to8 ;

ARCHITECTURE Structure OF h3to8 IS
    SIGNAL EnableTop, EnableBot : STD_LOGIC ;
BEGIN
    EnableTop <= w(2) AND En ;
    EnableBot <= (NOT w(2)) AND En ;

    Decoder1: if2to4 PORT MAP ( w( 1 DOWNTO 0 ), EnableBot, y( 3 DOWNTO 0 ) ) ;
    Decoder2: if2to4 PORT MAP ( w( 1 DOWNTO 0 ), EnableTop, y( 7 DOWNTO 4 ) ) ;
END Structure ;
```

. . . con't

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
PACKAGE h3to8_package IS
    COMPONENT h3to8
        PORT ( w    : IN    STD_LOGIC_VECTOR(2 DOWNTO 0) ;
               En   : IN    STD_LOGIC ;
               y    : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
    END COMPONENT ;
END h3to8_package ;
```

6.27.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.h3to8_package.all ;

ENTITY h6to64 IS
    PORT ( w    : IN    STD_LOGIC_VECTOR( 5 DOWNTO 0 ) ;
           En   : IN    STD_LOGIC ;
           y    : OUT   STD_LOGIC_VECTOR(63 DOWNTO 0 ) ) ;
END h6to64 ;

ARCHITECTURE Structure OF h6to64 IS
    SIGNAL Enables : STD_LOGIC_VECTOR( 7 DOWNTO 0 ) ;
BEGIN
    root : h3to8 PORT MAP ( w( 5 DOWNTO 3 ), En, Enables ) ;
    leaf0: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 0 ), y( 7 DOWNTO 0 ) ) ;
    leaf1: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 1 ), y( 15 DOWNTO 8 ) ) ;
    leaf2: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 2 ), y( 23 DOWNTO 16 ) ) ;
    leaf3: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 3 ), y( 31 DOWNTO 24 ) ) ;
    leaf4: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 4 ), y( 39 DOWNTO 32 ) ) ;
    leaf5: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 5 ), y( 47 DOWNTO 40 ) ) ;
    leaf6: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 6 ), y( 55 DOWNTO 48 ) ) ;
    leaf7: h3to8 PORT MAP ( w( 2 DOWNTO 0 ), Enables( 7 ), y( 63 DOWNTO 56 ) ) ;
END Structure ;
```

6.28.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob6_28 IS
    PORT ( bcd   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           leds  : OUT   STD_LOGIC_VECTOR(1 TO 7) ) ;
END prob6_28 ;

ARCHITECTURE Behavior OF prob6_28 IS
BEGIN
    WITH bcd SELECT
        --        abcdefg
        leds <= "1111110" WHEN "0000",
                "0110000" WHEN "0001",
                "1101101" WHEN "0010",
                "1111001" WHEN "0011",
                "0110011" WHEN "0100",
                "1011011" WHEN "0101",
                "1011111" WHEN "0110",
                "1110000" WHEN "0111",
                "1111111" WHEN "1000",
                "1111011" WHEN "1001",
                "-------" WHEN OTHERS ;
END Behavior ;
```

6.29.

$$
\begin{aligned}
a &= w_3 + w_2 w_0 + w_1 + \overline{w}_2 \overline{w}_0 \\
b &= w_3 + \overline{w}_1 \overline{w}_0 + w_1 w_0 + \overline{w}_2 \\
c &= w_2 + \overline{w}_1 + w_0
\end{aligned}
$$

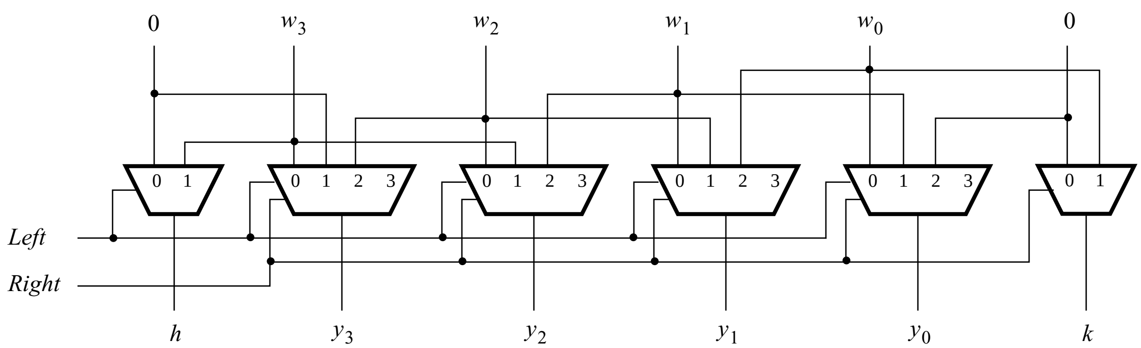
6.30.

$$
\begin{aligned}
d &= w_3 + \overline{w}_2 \overline{w}_0 + w_1 \overline{w}_0 + w_2 \overline{w}_1 w_0 + \overline{w}_2 w_1 \\
e &= \overline{w}_2 \overline{w}_0 + w_1 \overline{w}_0 \\
f &= w_3 + \overline{w}_1 \overline{w}_0 + w_2 \overline{w}_0 + w_2 \overline{w}_1 \\
g &= w_3 + w_1 \overline{w}_0 + w_2 \overline{w}_1 + \overline{w}_2 w_1
\end{aligned}
$$

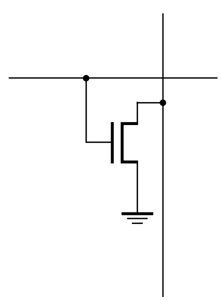
6.31. Using an arrangement similar to Figure 6.56, the desired circuit can be specified by the following truth table:

Left	*Right*	h	y_3	y_2	y_1	y_0	k
0	0	0	w_3	w_2	w_1	w_0	0
0	1	0	w_0	w_3	w_2	w_1	w_0
1	0	w_3	w_2	w_1	w_0	w_3	0

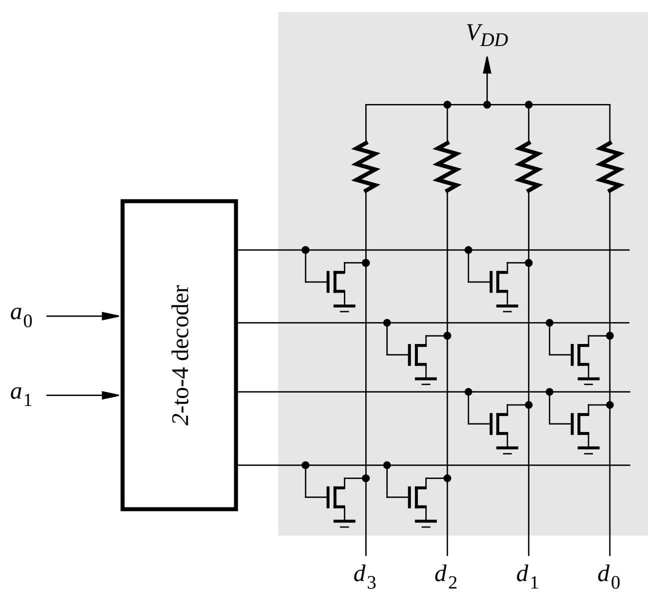
Using multiplexers, this truth table may be realized as



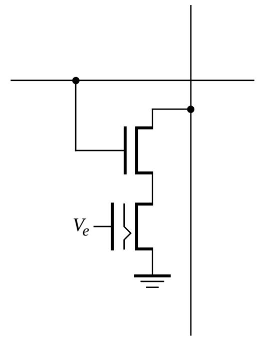
6.32. (a) Each ROM location that should store a 1 requires no circuitry, because the pull-up resistor provides the default value of 1. Each location that stores a 0 has the following cell



(b)



(c) Every location in the ROM contains the following cell



If a location should store a 1, then the corresponding EEPROM transistor is programmed to be turned off. But if the location should store a 0, then the EEPROM transistor is left unprogrammed.

(d)