Chapter 9

Exercise 9.2: Implement a BCD decoder using an Excess-3 decoder, a 2-input binary decoder and a NOR gate.

The relation between BCD code and the Excess-3 code is:

x	0	1	2	3	4	5	6	7	8	9	10	11	12
z (Ex-3)	-	-	-	0	1	2	3	4	5	6	7	8	9
y (BCD)	0	1	2	3	4	5	6	7	8	9	-	-	-

where x is the radix-2 representation of the input vector and z and y are the indices of the outputs of the decoders with value 1.

From the table we see that for x between 3 and 9, the output of the Excess-3 decoder can be relabeled to give some of the outputs of the BCD decoder. Since for x between 0 and 2, no output of the Excess-3 decoder has value 1, it is necessary to decode these values separately. It's possible to do this using a 2-input binary decoder that has as inputs the bits x_1 and x_0 (the least significant bits) and making the enable input active when $x \leq 3$.

The corresponding network is shown in Figure 9.1, on page 164.

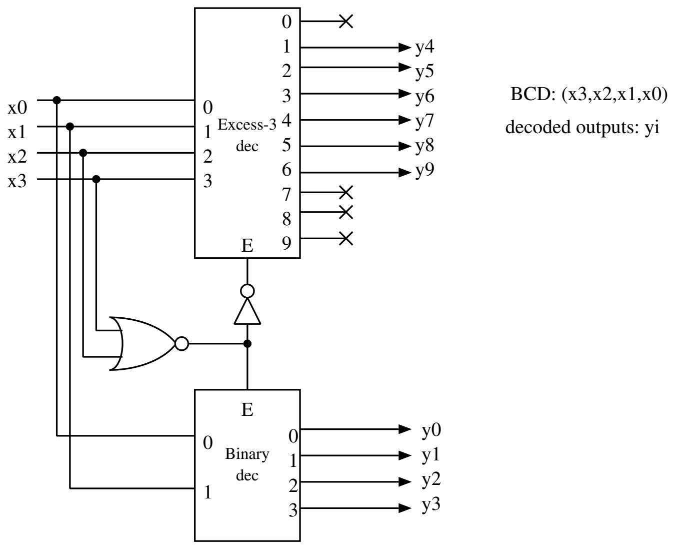


Figure 9.1: BCD decoder - Exercise 9.2

Exercise 9.8: (a) From Figure 9.34 of the textbook we get the following table:

BCD $b_3b_2b_1b_0$	7-segment display $abcdefg$
0000	0000001
0001	1001111
0010	0010010
0011	0000110
0100	1001100
0101	0100100
0110	0100000
0111	0001111
1000	0000000
1001	0001100

The implementation of this code converter using a decoder and OR gates is shown in Figure 9.2. The implementation using a decoder and an encoder is not efficient, and for this reason it is not shown. Two 4-bit encoders or a large 7-bit encoder should be used, and since there is only a small set of 7-segment codes, many of the encoder inputs would not be used, or would require OR gates to combine two or more decoder outputs.

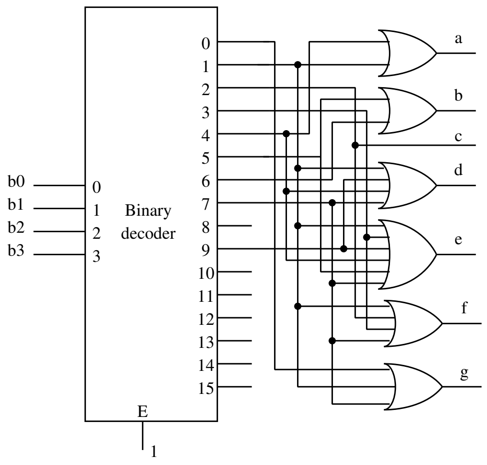
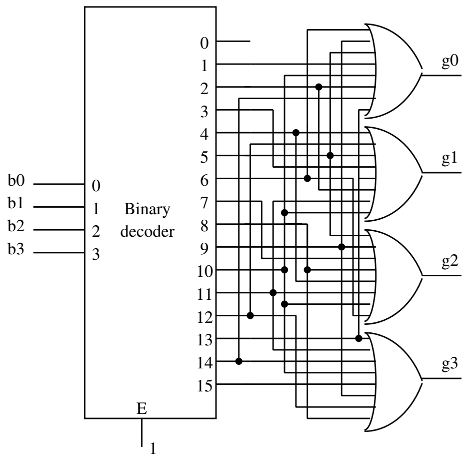


Figure 9.2: Network of Exercise 9.8 (a)

(b) Four-bit binary to 4-bit Gray code. The function table is:

binary $b_3b_2b_1b_0$	Gray $g_3g_2g_1g_0$
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Although the implementation by a gate network is quite simple, we show two different implementations in Figure 9.3. One uses a decoder and OR gates, and the other uses a decoder and encoder.



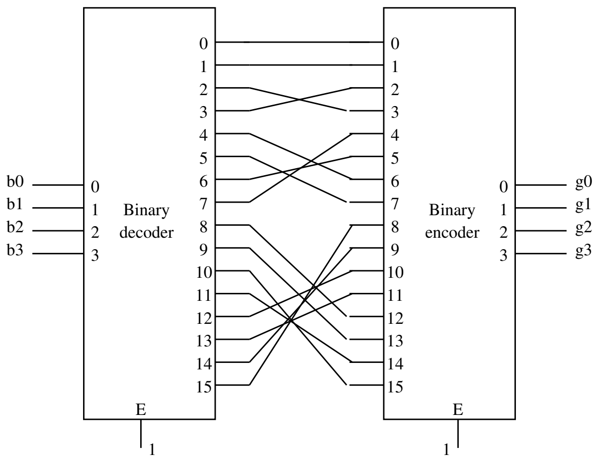


Figure 9.3: Binary to Gray-code converter - Exercise 9.8 (b)

(c) BCD to 2-out-of-5 code converter. The function table of the system follows:

BCD	2-out-of-5
$b_3b_2b_1b_0$	$c_4c_3c_2c_1c_0$
0000	00011
0001	11000
0010	10100
0011	01100
0100	10010
0101	01010
0110	00110
0111	10001
1000	01001
1001	00101

Two implementations of this code converter, one using a decoder and OR gates, and another using a decoder and encoder are shown in Figure 9.4.

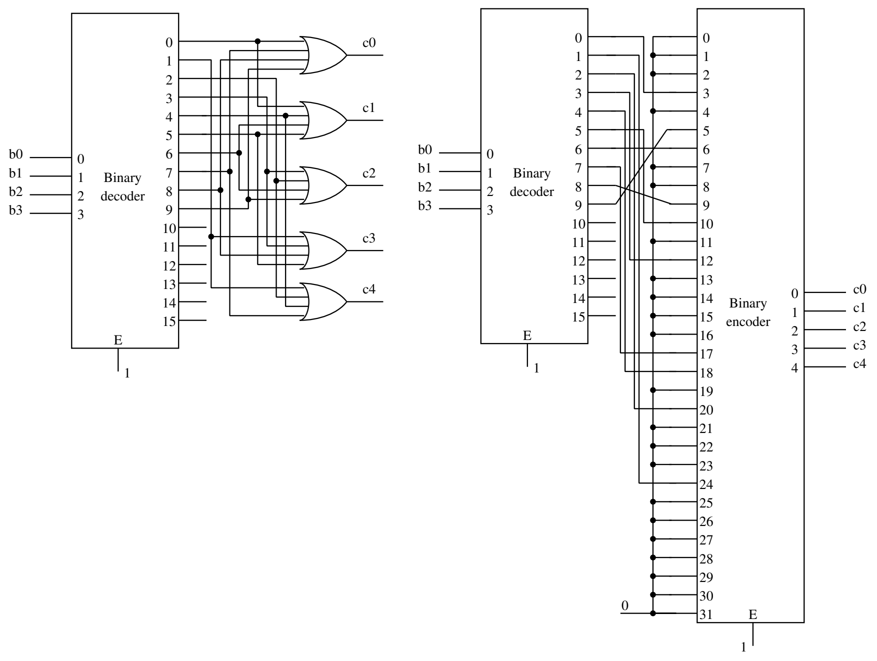


Figure 9.4: Network of Exercise 9.8 (c)

Exercise 9.16 The implementation of an 8-input multiplexer using a 3-input binary decoder and NAND gates is shown in Figure 9.5. The selection lines $s = (s_2, s_1, s_0)$ are decoded and used to make $z = i_j$, such that $j = s = s_2.2^2 + s_1.2 + s_0$.

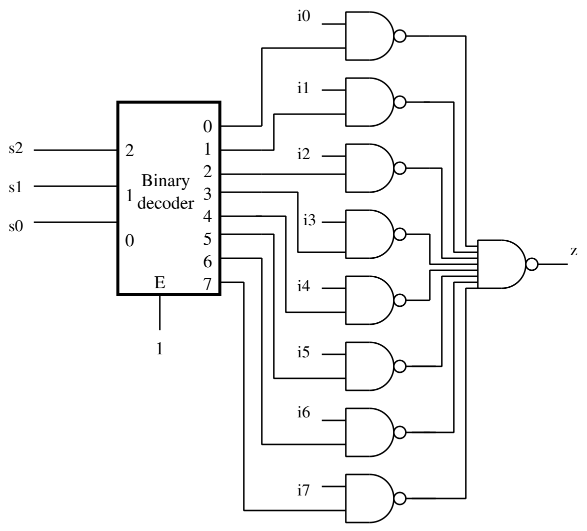


Figure 9.5: Network for Exercise 9.16

Exercise 9.20 Calling the output of stage 1 $w1$ and of stage 2 $w2$ then:

$$w1_j = \begin{cases} x_j & \text{if } s_0 = 0 \\ x_{(j+1) \bmod 8} & \text{if } s_0 = 1 \end{cases}$$

which indicates that this stage rotates 0 or 1 position left depending on the value of s_0.

Similarly,

$$w2_j = \begin{cases} w1_j & \text{if } s_1 = 0 \\ w1_{(j+2) \bmod 8} & \text{if } s_1 = 1 \end{cases}$$

Finally,

$$y_j = \begin{cases} w2_j & \text{if } s_2 = 0 \\ w2_{(j+4) \bmod 8} & \text{if } s_2 = 1 \end{cases}$$

and this corresponds to rotating 0 or 4 positions.

Consequently, the module rotates left $s = 4s_2 + 2s_1 + s_0$ positions with $0 \leq s \leq 7$. For example, if the input vector is $(x_7, x_6, ..., x_0) = 11010110$ the output vector, for a rotation of 2 to the left is: 01011011.

Exercise 9.22
Considering the network formed by the decoder and the multiplexer we have that

$$z = \begin{cases} 1 & \textbf{if } (w, d, e) = (f, g, h) \\ 0 & \textbf{otherwise} \end{cases} \tag{9.1}$$

where w is the output of the first multiplexer. An expression for this output is:

$$w = a'b'c + bc' + abc = ab + bc' + a'b'c$$

The gate network that implements the network in Figure 9.38 of the textbook is shown in Figure 9.6. The equality comparator that generates z is implemented using XOR and NOR gates (as proposed in the hint). The gate network to generate w is implemented with AND and OR gates.

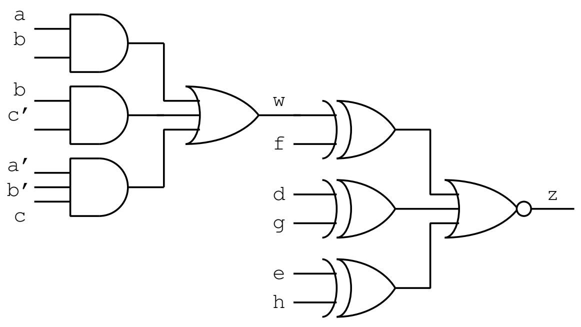


Figure 9.6: Network for Exercise 9.22

Exercise 9.26. The codewords of both systems are presented in the following table:

n	Code A $p = (3n) \bmod 16$ $p_2p_1p_0$	Code B $q = (7n) \bmod 16$ $q_2q_1q_0$
0	0000	0000
1	0011	0111
2	0110	1110
3	1001	0101
4	1100	1100
5	1111	0011
6	0010	1010
7	0101	0001
8	1000	1000
9	1011	1111
10	1110	0110
11	0001	1101
12	0100	0100
13	0111	1011
14	1010	0010
15	1101	1001

(a) the design of an A-to-B converter using one 8-input multiplexer and one 2-input XOR gate is shown in Figure 9.7. Observe that:

$$\begin{aligned} q_0 &= p_0 \\ q_1 &= p_1 \\ q_2 &= p_2 \oplus p_0 \end{aligned}$$

and q_3 is easily implemented using an 8-input multiplexer from the following function table:

$p_3p_2p_1p_0$	q_3
0000	0
0001	1
0010	1
0011	0
0100	0
0101	0
0110	1
0111	1
1000	1
1001	0
1010	0
1011	1
1100	1
1101	1
1110	0
1111	0

(b) the code converter designed using one 4-input decoder and one 16-input encoder is shown in Figure 9.8.

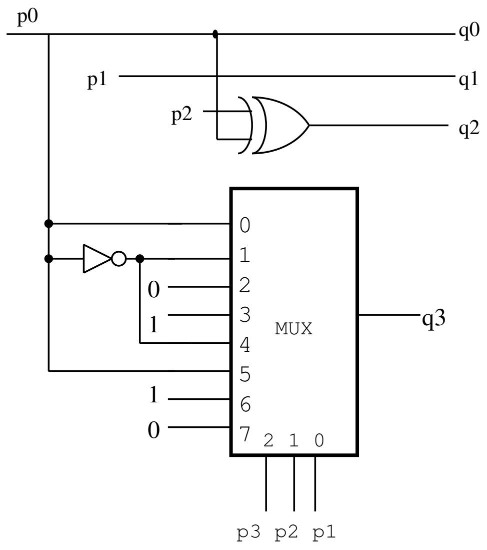


Figure 9.7: Code converter - Exercise 9.26 (a)

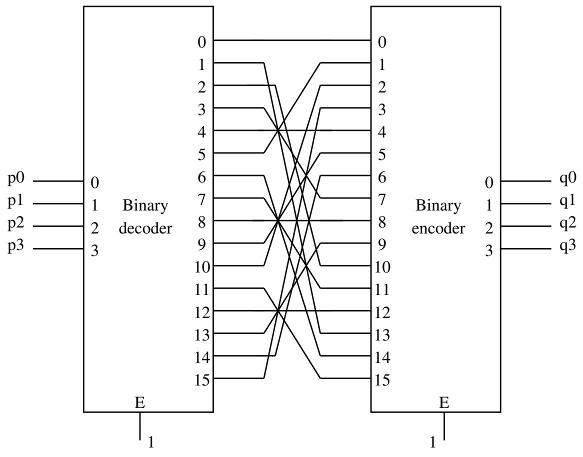


Figure 9.8: Code converter - Exercise 9.26 (b)