Chapter 9

Exercise 9.1 (a) Implement a 10 output (decimal) decoder for 2-out-of-5 code using NAND gates. The 2-out-of-5 code represents decimal digits as shown in the following table:

i	$x_4x_3x_2x_1x_0$
0	00011
1	11000
2	10100
3	01100
4	10010
5	01010
6	00110
7	10001
8	01001
9	00101

Each output z_i of the decoder is activated when the input has the 2-out-of-5 code that represents the value i in decimal and the *enable* input $E = 1$. The expressions for the outputs are:

$$\begin{aligned}
z_0 &= x_4'x_3'x_2'x_1x_0E \\
z_1 &= x_4x_3x_2'x_1'x_0'E \\
z_2 &= x_4x_3'x_2x_1'x_0'E \\
z_3 &= x_4'x_3x_2x_1'x_0'E \\
z_4 &= x_4x_3'x_2'x_1x_0'E \\
z_5 &= x_4'x_3x_2'x_1x_0'E \\
z_6 &= x_4'x_3'x_2x_1x_0'E \\
z_7 &= x_4x_3'x_2'x_1'x_0E \\
z_8 &= x_4'x_3x_2'x_1'x_0E \\
z_9 &= x_4'x_3'x_2x_1'x_0E
\end{aligned}$$

Part of the gate network for the circuit is shown in Figure **??**.

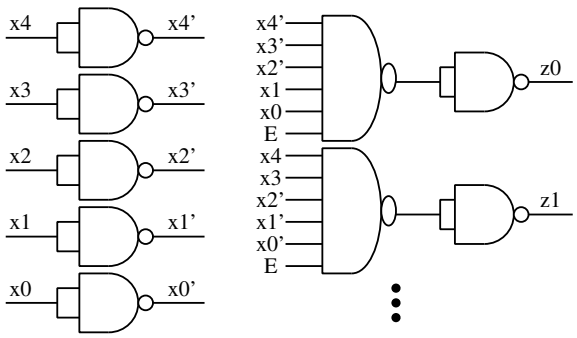


Figure 9.1: Network for Exercise 9.1 (a)

(b) Implement a 10 output (decimal) decoder using NAND gates for a 4-bit Gray code. The code table is shown next.

i	$g_3g_2g_1g_0$
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101

The expressions for the ten outputs z_0 to z_9 are:

$$\begin{aligned}
z_0 &= g_2'g_1'g_0'E \\
z_1 &= g_2'g_1'g_0E \\
z_2 &= g_2'g_1g_0E \\
z_3 &= g_2'g_1g_0'E \\
z_4 &= g_2g_1g_0'E \\
z_5 &= g_2g_1g_0E \\
z_6 &= g_3'g_2g_1'g_0E \\
z_7 &= g_3'g_2g_1'g_0'E \\
z_8 &= g_3g_0'E \\
z_9 &= g_3g_0E
\end{aligned}$$

The implementation, shown in Figure **??**, consists of six 4-input NAND gates, two 5-input NAND gates, two 3-input NAND gates, and 14 NOT gates.

(c) Implement a 10 output (decimal) decoder using NAND gates for a 2-4-2-1 code. The code table is shown next.

i	$c_3c_2c_1c_0$
0	0000
1	0001
2	0010
3	0011
4	0100
5	1011
6	1100
7	1101
8	1110
9	1111

The switching expressions for the decoder outputs are:

$$z_0 = c_3'c_2'c_1'c_0'E$$

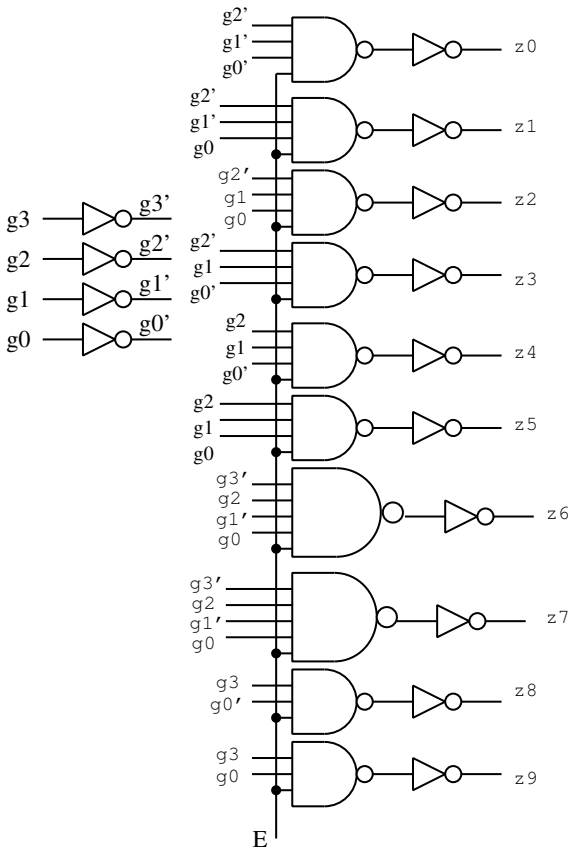


Figure 9.2: Network for Exercise 9.1 (b)

$$
\begin{aligned}
z_1 &= c_3' c_2' c_1' c_0 E \\
z_2 &= c_3' c_2' c_1 c_0' E \\
z_3 &= c_3' c_2' c_1 c_0 E \\
z_4 &= c_3' c_2 c_1' c_0' E \\
z_5 &= c_3 c_2' c_1 c_0 E \\
z_6 &= c_3 c_2 c_1' c_0' E \\
z_7 &= c_3 c_2 c_1' c_0 E \\
z_8 &= c_3 c_2 c_1 c_0' E \\
z_9 &= c_3 c_2 c_1 c_0 E
\end{aligned}
$$

Part of the gate network that implements these expressions is shown in Figure **??**.

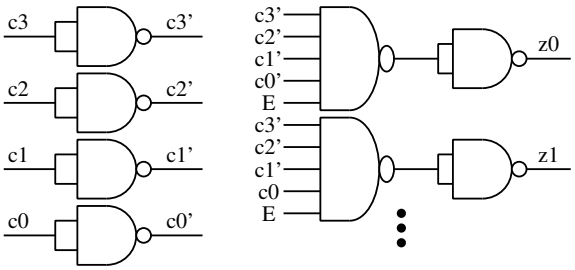


Figure 9.3: Gate network for Exercise 9.1 (c)

Exercise 9.3 The 4-bit Odd/Even parity functions have value 1 when the number of 1s in the input vector is odd/even. The implementation of these functions using a 4-input decoder and OR gates is shown in Figure **??**. The even parity is produced by output EP and the odd parity by output OP.

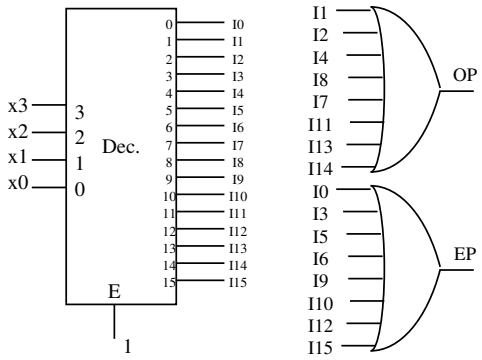


Figure 9.4: Parity functions - Exercise 9.3

Exercise 9.5 Part (a)

- number of levels in the tree of decoders = $20/4 = 5$ levels.
- the number of decoders in this tree = $1 + 16 + 16^2 + 16^3 + 16^4 = 69905$ decoders.

Part (b) using coincident decoding we need five 4-input decoders to receive the 20 inputs. We would need 2^{20} AND gates.

Exercise 9.7

From the figure, we get the following expressions for the circuit outputs:

$$\begin{aligned}
D &= (I_8' + I_9')' = I_8 I_9 \\
C &= ((I_8' + I_9')'.(I_4' + I_5' + I_6' + I_7'))' \\
&= (I_8 I_9.(I_4' + I_5' + I_6' + I_7'))' \\
B &= (I_8 I_9.(I_2' I_4 I_5 + I_3' I_4 I_5 + I_6' + I_7'))' \\
&= (I_8 I_9 (I_6' + I_7') + I_9 I_8 I_7 I_6 I_5 I_4 (I_3' + I_2'))' \\
A &= ((I_8 I_9).(I_1' I_2 I_4 I_6 + I_3' I_4 I_6 + I_5' I_6 + I_7') + I_9')' \\
&= (I_9' + I_9 I_8 I_7' + I_9 I_8 I_7 I_6 I_5' + I_9 I_8 I_7 I_6 I_5 I_4 I_3' + I_9 I_8 I_7 I_6 I_5 I_4 I_3 I_2 I_1')'
\end{aligned}$$

From these expression we get the following table:

I_9'	I_8'	I_7'	I_6'	I_5'	I_4'	I_3'	I_2'	I_1'	D'	C'	B'	A'	Output(decimal)
1	-	-	-	-	-	-	-	-	1	0	0	1	9
0	1	-	-	-	-	-	-	-	1	0	0	0	8
0	0	1	-	-	-	-	-	-	0	1	1	1	7
0	0	0	1	-	-	-	-	-	0	1	1	0	6
0	0	0	0	1	-	-	-	-	0	1	0	1	5
0	0	0	0	0	1	-	-	-	0	1	0	0	4
0	0	0	0	0	0	1	-	-	0	0	1	1	3
0	0	0	0	0	0	0	1	-	0	0	1	0	2
0	0	0	0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Consequently, if $Z = (D', C', B', A')$ represents a decimal digit z in BCD, we get

$$z = \begin{cases} i & \textbf{if } (I_i' = 1 \text{ for some } i > 0) \textbf{ and } (I_j' = 0 \text{ for all } j > i); \\ 0 & \text{otherwise} \end{cases}$$

which corresponds to a priority encoder function.

Exercise 9.9: The implementation is shown in Figure **??**, on page **??**. The outputs of the decoder are labeled according to the Gray code and connected to the corresponding inputs of the encoder.

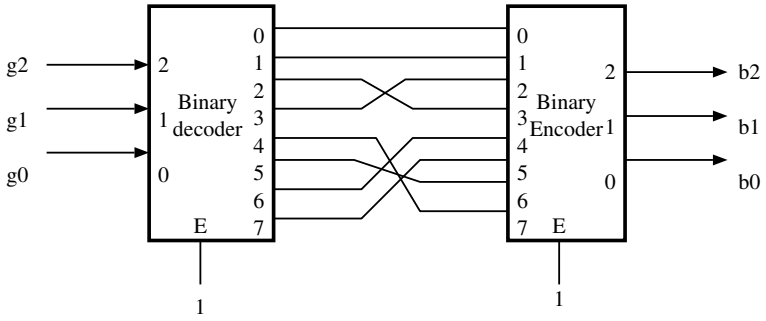


Figure 9.5: Code converter Binary-Gray, Exercise 9.9

Exercise 9.11

Input: $x \in \{0,1,2,3,4,5,6,7\}$, represented in binary by the vector $\underline{x} = \{x_2, x_1, x_0\}, x_i \in \{0,1\}$.
Output: $y \in \{0,1,2,3,4,5,6,7\}$, represented in binary by the vector $\underline{y} = \{y_2, y_1, y_0\}, y_i \in \{0,1\}$
Function: $y = (3x) \bmod 8$
The function table for the system is:

x	0	1	2	3	4	5	6	7
y	0	3	6	1	4	7	2	5

The implementation of this system using a binary decoder and a binary encoder is shown in Figure **??**.

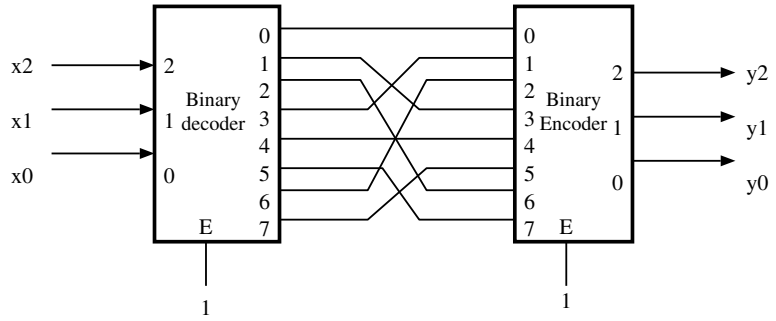


Figure 9.6: Function $y = 3x$ using decoder and encoder

Exercise 9.15

$f(a,b,c,d) = oneset(1,3,4,9,14,15)$

	$abcd$	$f(a,b,c,d)$
0	0000	0
1	0001	1
2	0010	0
3	0011	1
4	0100	1
5	0101	0
6	0110	0
7	0111	0
8	1000	0
9	1001	1
10	1010	0
11	1011	0
12	1100	0
13	1101	0
14	1110	1
15	1111	1

(a) the implementation using 8-input multiplexer is presented in Figure **??**. The expression can be manipulated as follows:

$$f(a,b,c,d) = a'b'c'd + a'b'cd + a'bc'd' + ab'c'd + abcd' + abcd$$

$$f(a,b,c,d) = m_0(a,b,c)d + m_1(a,b,c)d + m_2(a,b,c)d' + m_4(a,b,c)d + m_7(a,b,c)(d' + d)$$

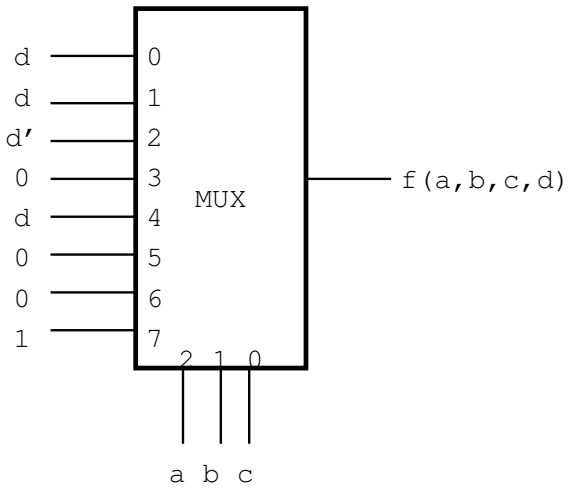


Figure 9.7: Implementation for Exercise 9.15 (a)

(b) the implementation using 4-input multiplexer is presented in Figure **??**. The expression for this implementation is:

$$f(a,b,c,d) = m_0(a,b)(c'd + cd) + m_1(a,b)c'd' + m_2(a,b)c'd + m_3(a,b)(cd + cd')$$

$$f(a,b,c,d) = m_0(a,b)d + m_1(a,b)(c+d)' + m_2(a,b)(c+d')' + m_3(a,b)c$$

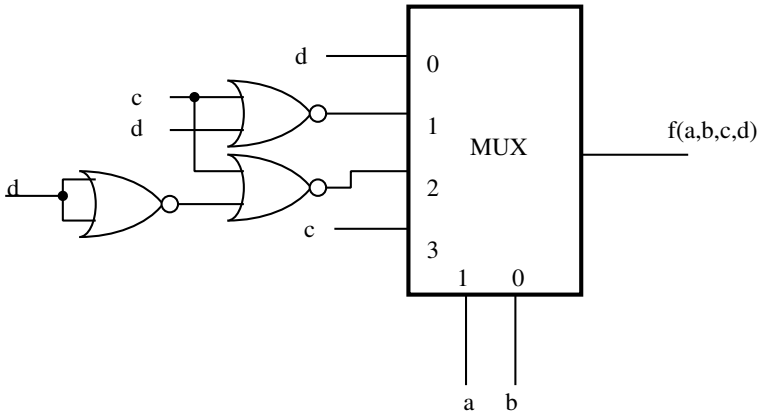


Figure 9.8: Network for Exercise 9.15 (b)

Exercise 9.17

Part (a) The specification of an $n-$bit simple shifter is given on page 265 of the textbook. A description of an 8-bit shifter is easily obtained from there. The block diagram of the circuit is shown in Figure **??**.

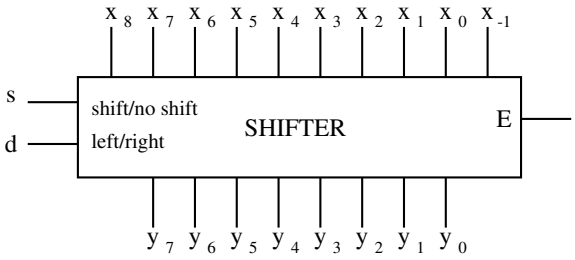


Figure 9.9: Block diagram of 8-bits shifter

To generate each output, a 4-input multiplexer is used as shown in Figure **??**. The selection inputs are defined according to following table:

c_1c_0	Operation	s	d	E
00	LEFT	shift	L	1
01	RIGHT	shift	R	1
10	NO SHIFT	no shift	-	1
11	DISABLED	-	-	0

Considering $shift = L = 1$ and $no\_shift = R = 0$ (same convention used in page 266 of the textbook) we obtain the following Kmaps:

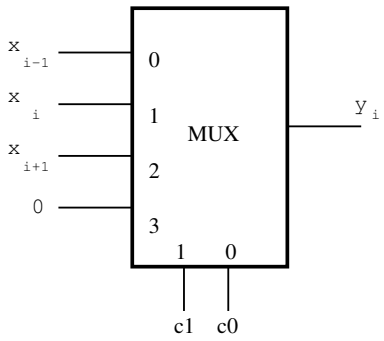
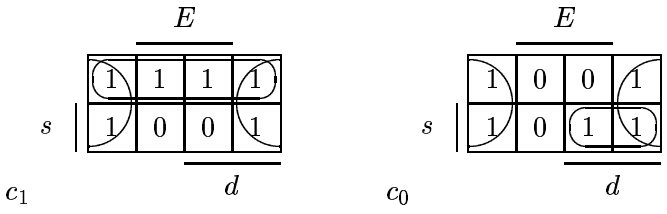


Figure 9.10: Circuit for each output of the 8-bit simple shifter



that correspond to the following expressions:

$$\begin{aligned} c_1 &= E' + s' \\ c_0 &= sd + E' \end{aligned}$$

Part (b) An 8-bit bidirectional 3-shifter is specified as:
Inputs:
$\underline{x} = (x_{10}, x_9, x_8, x_7, \ldots, x_0, x_{-1}, x_{-2}, x_{-3})$, with $x_i \in \{0, 1\}$.
$s \in \{0, 1, 2, 3\}$
$d \in \{L, R\}$
$E \in \{0, 1\}$
Output: $\underline{y} = (y_7, y_6, \ldots, y_1, y_0)$, with $y_j \in \{0, 1\}$.
Function:

$$y_i = \begin{cases} x_{i-s} & \text{if } (d = L) \text{ and } (E = 1) \\ x_{i+s} & \text{if } (d = R) \text{ and } (E = 1) \\ 0 & \text{otherwise} \end{cases} \tag{9.1}$$

Let us assume that $L = 1$ and $R = 0$. Each output is generated by an 8-input multiplexer as shown in Figure **??**. The table for the control inputs $c_2c_1c_0$ is:

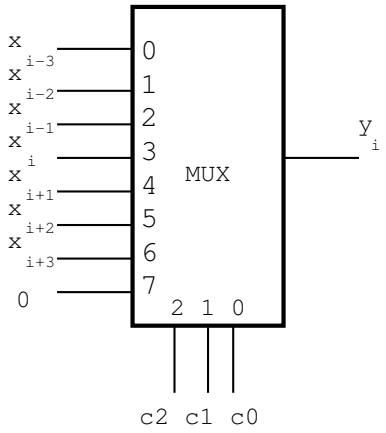
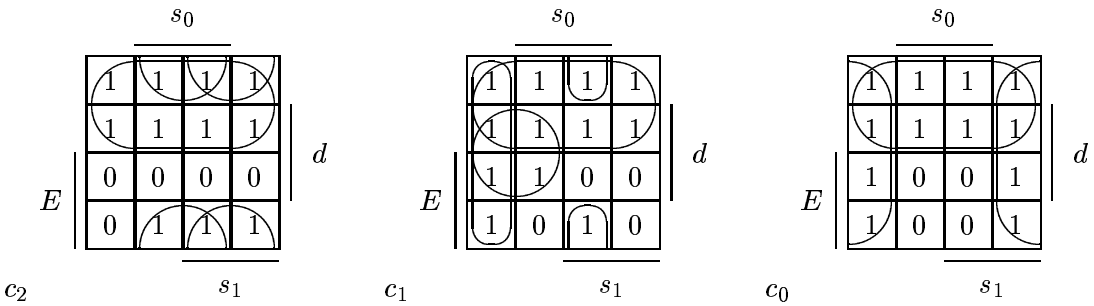


Figure 9.11: Circuit for each output of the 8-bit bidirectional shifter

Eds	000	001	002	003	010	011	012	013	100	101	102	103	110	111	112	113
$c_2c_1c_0$	111	111	111	111	111	111	111	111	011	100	101	110	011	010	001	000

The values on the table are mapped into the following Kmaps, considering that s is represented in binary code:



The switching expressions are:

$$
\begin{aligned}
c_2 &= E' + s_1 d' + s_0 d' \\
c_1 &= E' + d s_1' + E s_1 s_0 + s_1' s_0' \\
c_0 &= E' + s_0'
\end{aligned}
$$

Exercise 9.19 A n-bit p-shifter has $(n + 2p)$ input bits and n output bits. It can shift right or left (direction input d) and the distance of shifting can vary from 0 to p.

The implementation of a 32-bit 3-shifter using four 8-bit 3-shifters is presented in Figure **??**. The circuit on the top is a 32-bit 3-shifter that shifts to the left only. The circuit on the bottom of the figure is a bi-directional 32-bit 3-shifter. The input of the circuit was named i_{31} to i_0, and the output z_{31} to z_0.

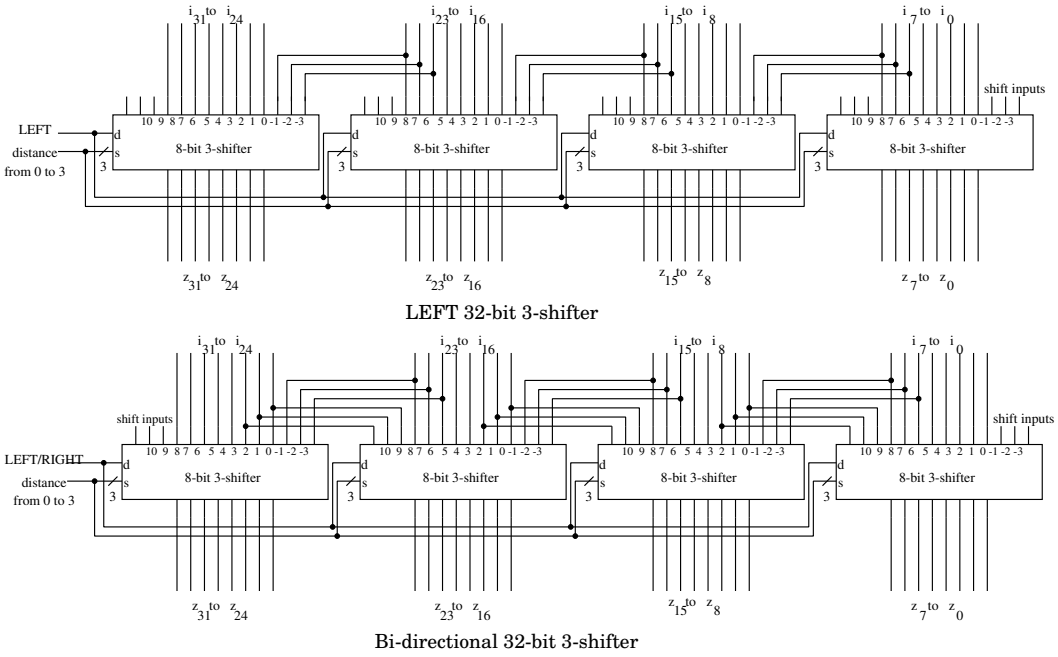


Figure 9.12: Exercise 9.19

Exercise 9.23 The network in Figure 9.39 of the textbook has a serial line (w) between the MUX and DEMUX that is described as:

$$w = \begin{cases} x_i & \text{if } i = 4a + 2b + c \text{ and } E_1 = 1 \\ 0 & \text{otherwise} \end{cases} \tag{9.2}$$

The outputs of the DEMUX are specified as:

$$y_i = \begin{cases} w & \text{if } i = 4a + 2b + c \text{ and } E_2 = 1 \\ 0 & \text{otherwise} \end{cases} \tag{9.3}$$

Combining both equations we obtain:

$$y_i = \begin{cases} x_i & \text{if } i = 4a + 2b + c \text{ and } E_1 = E_2 = 1 \\ 0 & \text{otherwise} \end{cases} \tag{9.4}$$

So, y_i is the same as x_i or is zero. This circuit is useful to allow the sharing of the single line between Mux and Demux among all pairs of (input,output): (x_0, y_0), (x_1, y_1) ... and so on. It's used in communication lines to divide the full capacity of the communication line among the many transmitters and receivers. Each pair can communicate without interference of the other pairs.

Exercise 9.25 From the circuit presented in the figure we can obtain the state diagram in Figure **??**. The binary decoder connected to the state register generates the signals S_i, which is 1 when the sequential system is at state i. The system outputs correspond to the state signals, that means, each output is active in one particular state. For this reason we used the output names as state names to make the state diagram more meaningful. In each present state we identify which input of the binary encoder is 1. This input determines the next state. For example, the origin of arcs going into state S_1 (check) are obtained by considering the input of the Binary Encoder labeled 1, which is S_0 for input GO, S_2 (always, no condition), and S_3 (always).

The state diagram shows the operation of a controller which starts to operate with a GO signal. During operation it monitors two variables: *dist* and *count* and issues movement control signals and counter control signals, until ($dist \leq 10$) and ($count = 3$).

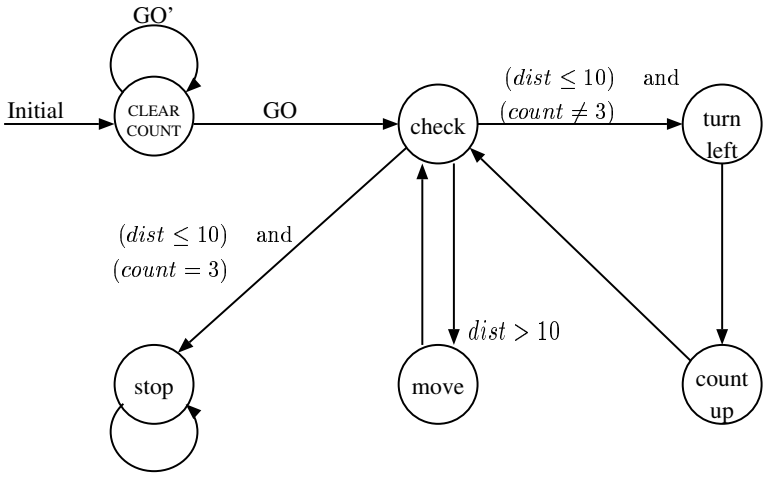


Figure 9.13: Exercise 9.25