

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 9.12.

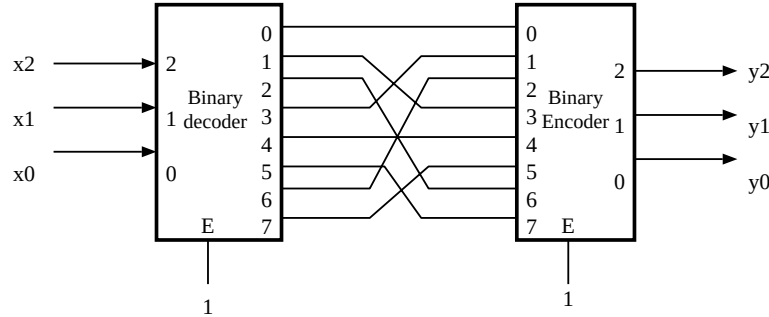


Figure 9.12: Function $y = 3x$ using decoder and encoder

Exercise 9.12 The 64-input encoder network in Figure 9.35 of the textbook consists of two levels of modules. In the first level there are eight encoders, each of them encoding part of the input vector $\underline{x}$. Since there is only one input with value 1, the outputs of all encoder modules are 0 except the one corresponding to this $x_i = 1$. Also, only the corresponding A has value 1. Naming w_j the value of the 3-bit output of the encoder that has inputs x_i , $8k \leq i \leq 8k + 7$, the output is described as:

$$w_j = \begin{cases} i \bmod 8 & \text{if } x_i = 1 \text{ for } j = \lfloor i/8 \rfloor \\ 0 & \text{otherwise} \end{cases}$$

and

$$A_j = \begin{cases} 1 & \text{if } x_i = 1 \text{ for } j = \lfloor i/8 \rfloor \\ 0 & \text{otherwise} \end{cases}$$

In the second level, there are three OR gates with eight inputs each which produce (y_2, y_1, y_0) and an eight-input encoder to encode the A outputs of the first level encoders and produce (y_5, y_4, y_3) . The connection of the OR gates produces:

$$\sum_{j=0}^2 y_j 2^j = OR(w_7, w_6, \dots, w_0) = i \bmod 8$$

Exercise 9.15

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

	$abcd$	$f(a, b, c, d)$
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 9.14. The expression can be manipulated as follows:

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

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

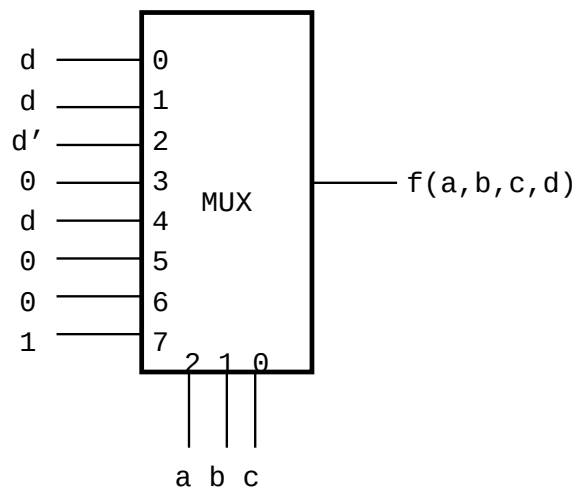


Figure 9.14: Implementation for Exercise 9.15 (a)

(b) the implementation using 4-input multiplexer is presented in Figure 9.15. The expression for this implementation is:

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

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

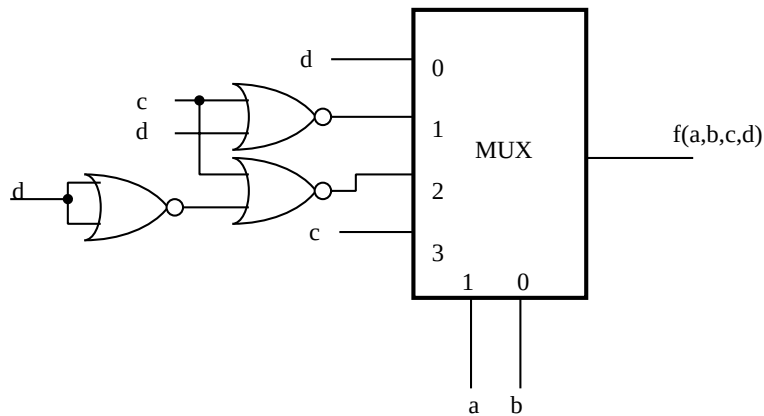


Figure 9.15: Network for Exercise 9.15 (b)

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

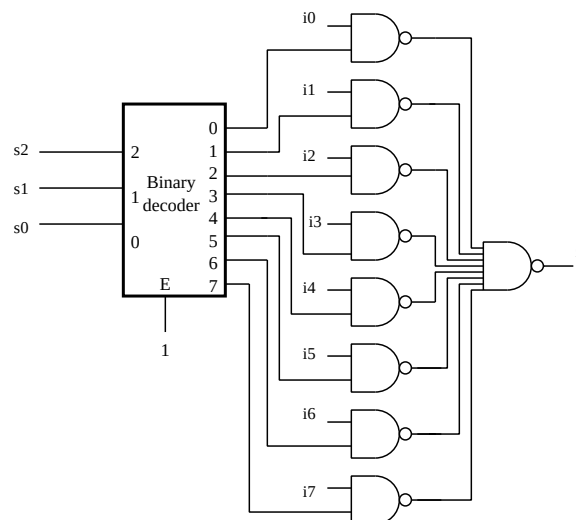


Figure 9.16: Network for Exercise 9.16

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, until $(dist \leq 10)$ and $(count = 3)$.

9.25

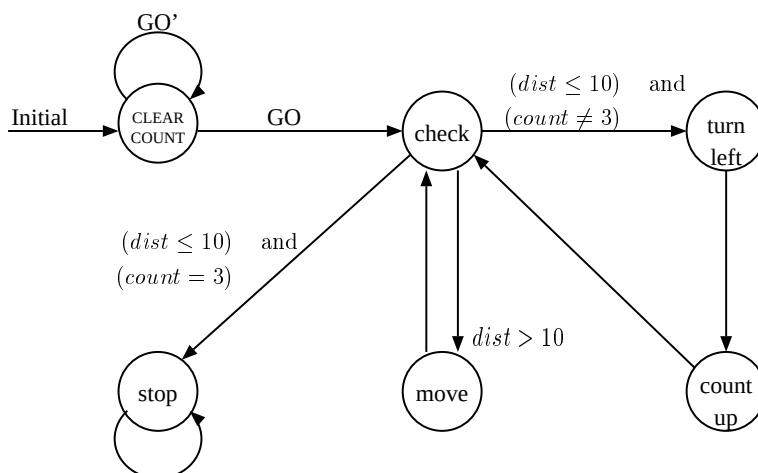


Figure 9.25: Exercise 9.25

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

n	Code A $p = (3n) \bmod 16$ $p_2 p_1 p_0$	Code B $q = (7n) \bmod 16$ $q_2 q_1 q_0$
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

Exercise 10.8 Design of a 6-bit CLA adder using gates with a maximum of 4 inputs. The section that generates *propagate* and *generate* signals doesn't need gates with more than 2 inputs. The same for the last layer of XOR gates that generates the final sum. The highest fan-in is required by the gates inside the CLG module. The expressions for the CLG outputs are:

$$\begin{aligned}
 c_6 &= g_5 + p_5g_4 + p_5p_4g_3 + p_5p_4p_3g_2 + p_5p_4p_3p_2g_1 + p_5p_4p_3p_2p_1g_0 + p_5p_4p_3p_2p_1p_0c_0 \\
 c_5 &= g_4 + p_4g_3 + p_4p_3g_2 + p_4p_3p_2g_1 + p_4p_3p_2p_1g_0 + p_4p_3p_2p_1p_0c_0 \\
 c_4 &= g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0 \\
 c_3 &= g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0 \\
 c_2 &= g_1 + p_1g_0 + p_1p_0c_0 \\
 c_1 &= g_0 + p_0c_0 \\
 P &= p_5p_4p_3p_2p_1p_0 \\
 G &= g_5 + p_5g_4 + p_5p_4g_3 + p_5p_4p_3g_2 + p_5p_4p_3p_2g_1 + p_5p_4p_3p_2p_1g_0
 \end{aligned}$$

From these expressions we observe that AND and OR gates with 5, 6 and 7 inputs are required. We decompose these gates into three gates. The implementation of the 6-bit CLA is shown in Figure 10.5. From the Figure we count 66 gates distributed in 6 levels.

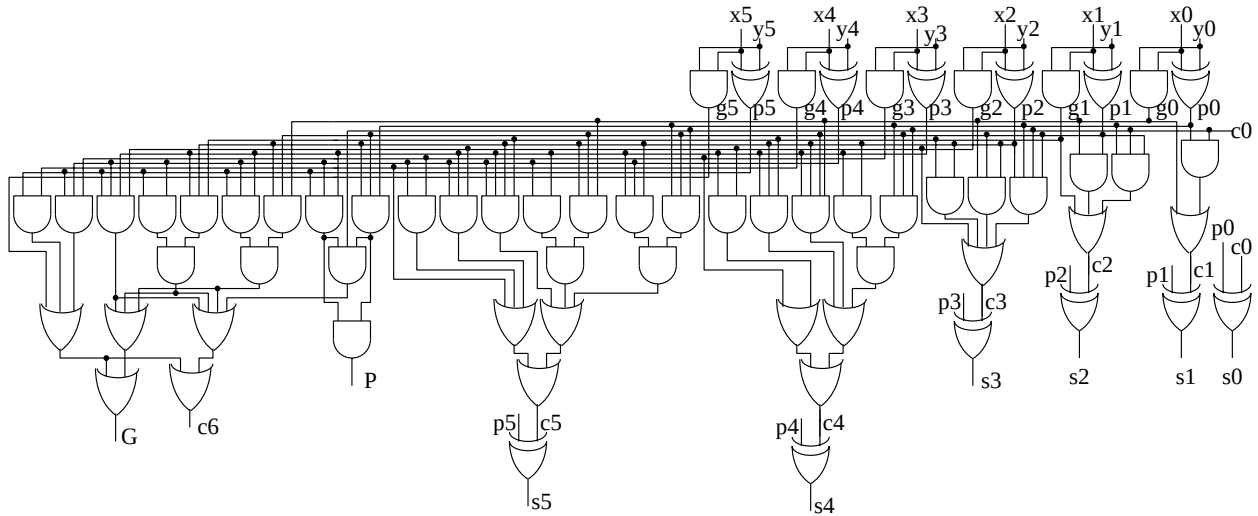


Figure 10.5: 6-bit CLA using gates of at most 4 inputs - Exercise 10.8

Exercise 10.9 Design of 64-bit adders.

a) a Ripple Carry Adder (RCA) using 4-bit adder modules is shown in Figure 10.6. This implementation of a 64-bit adder needs 16 adder modules (CLA-4). The delay is:

$$T_{CRA} = \delta_{CLA-xy-c4} + 14 * \delta_{CLA-c0-c4} + \max\{\delta_{CLA-c0-c4}, \delta_{CLA-c0-s3}\}$$

where $\delta_{CLA-xy-c4}$ is the delay to propagate the signal from input to the carry output of the 4-bit CLA, and $\delta_{CLA-c0-c4}$ and $\delta_{CLA-c0-s3}$ are the delays from the input c_0 to the carry out c_4 and the sum output bit s_3 , respectively.

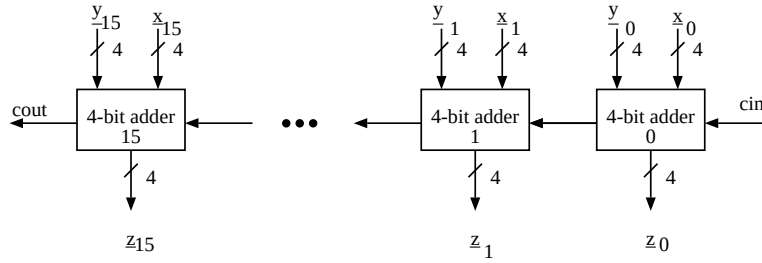


Figure 10.6: Carry Ripple Adder (CRA) - Exercise 10.9

Based on a NAND-NAND implementation of the 4-bit CLG that is part of the 4-bit CLA we obtain:

$$\begin{aligned}
 \delta_{CLA-xy-c4} &= t_{LH}(XOR - 2) + t_{HL}(NAND - 5) + t_{LH}(NAND - 5) \\
 &= 0.30 + 0.036 \times 5.1 + 0.34 + 0.019 \times 1 + 0.21 + 0.038L \\
 &= 1.05 + 0.038L \\
 \delta_{CLA-c0-s3} &= t_{LH}(XOR - 2) + t_{HL}(NAND - 4) + t_{LH}(NAND - 4) \\
 &= 0.3 + 0.036L + 0.21 + 0.038 \times 1 + 0.34 + 0.019 \times 2 \\
 &= 0.93 + 0.036L \\
 \delta_{CLA-c0-c4} &= t_{HL}(NAND - 5) + t_{LH}(NAND - 5) \\
 &= 0.21 + 0.038 \times 1 + 0.34 + 0.019L \\
 &= 0.59 + 0.019L
 \end{aligned}$$

Based on the fact that the input load of c_0 is 4, we obtain the delay of the CRA as:

$$\begin{aligned}
 T_{CRA} &= 1.07 + 0.019 \times 4 + 14(0.59 + 0.019 \times 4) + 0.93 + 0.036L \\
 &= 11.4 + 0.036L
 \end{aligned}$$

b) a Carry-lookahead Adder (CLA) using 4-bit carry-lookahead adder modules (CLA-4) and 4-bit carry-lookahead generator modules (CLG-4) is presented in Figure 10.7. Note that this design is an extension of the design showed in Figure 10.8 of the textbook.

Observe from the Figure that 20 modules are necessary, 16 CLA modules and 4 CLG modules. The delay of this adder can be calculated considering the following module delays:

$$\begin{aligned}
 \delta_{CLA-xy-PG} &= \delta_{CLA-xy-c4} = 1.07 + 0.019L \\
 \delta_{CLG-pg-c4} &= \delta_{CLA-c0-c4} = 0.59 + 0.019L \\
 \delta_{CLG-c0-c3} &= t_{HL}(NAND - 4) + t_{LH}(NAND - 4) \\
 &= 0.21 + 0.038 \times 1 + 0.34 + 0.019L \\
 &= 0.59 + 0.019L \\
 \delta_{CLA-c0-s3} &= t_{LH}(XOR - 2) + t_{HL}(NAND - 4) + t_{LH}(NAND - 4) \\
 &= 0.3 + 0.036L + 0.21 + 0.038 \times 1 + 0.34 + 0.019 \times 2 \\
 &= 0.93 + 0.036L
 \end{aligned}$$

The load of the p , g , and c_0 inputs of the CLG is 4.

The total delay is:

$$\begin{aligned}
 T_{CLA} &= \delta_{CLA-xy-PG} + \delta_{CLG-pg-c4} + 2 \times \delta_{CLG-c0-c4} + \delta_{CLG-c0-c3} + \delta_{CLA-c0-s3} \\
 &= 1.07 + 0.019 \times 4 + 0.59 + 0.019 \times 4 + 2(0.59 + 0.019 \times 4) + 0.59 + 0.019 \times 4 \\
 &\quad + 0.93 + 0.036L \\
 &= 4.74 + 0.036L
 \end{aligned}$$

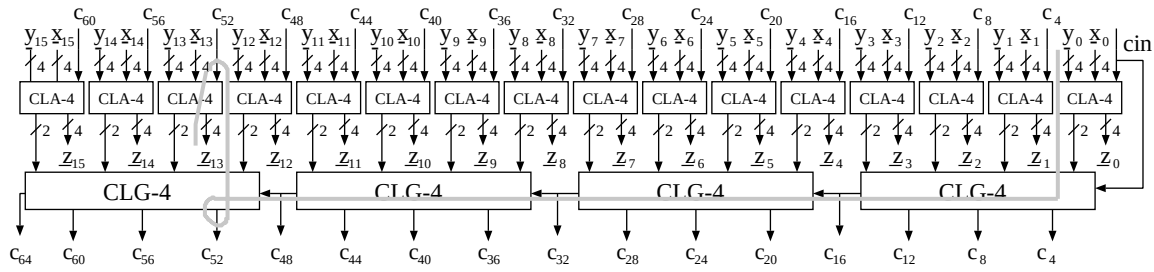


Figure 10.7: 64-bit CLA with 1 level of CLGs - Exercise 10.9

Another approach is to use one more level of CLGs, this time to generate the carries of groups of 16 bits. This scheme is presented in Figure 10.8. The propagation delay of this case is:

$$T_{CLAv2} = \delta_{CLA-xy-PG} + \delta_{CLG-pg-pg} + \delta_{CLG-pg-c3} + \delta_{CLG-c0-c3} + \delta_{CLA-c0-s3}$$

where $\delta_{CLG-pg-pg} = \delta_{CLA-c0-c4} = \delta_{CLG-pg-c3} = \delta_{CLG-c0-c3} = 0.59 + 0.019L$. Thus

$$T_{CLAv2} = 1.07 + 0.019 \times 4 + 3(0.59 + 0.019 \times 4) + 0.93 + 0.036L$$

$$T_{CLAv2} = 4.07 + 0.036L$$

which results in a faster implementation of the CLA, at the cost of one extra CLG-4 module.

Exercise 10.10:

considering radix $r = 2$

- case (a) — two's complement, $n = 7$ bits; $C = 2^7$
- case (b) — one's complement, $n = 8$ bits; $C = 2^8 - 1$
- case (c) — two's complement, $n = 5$ bits; $C = 2^5$
- case (d) — two's complement, $n = 8$ bits; $C = 2^8$

	Signed Integer - x (in decimal)	Representation - x_R (in decimal)	Bit vector
(a)	-37	91	1011011
(b)	-50	205	11001101
(c)	-5	27	11011
(d)	9	9	00001001