Chapter 6

Exercise 6.1 From Exercise 5.11, we know that the single-error detector for a 2-out-of-5 code (a, b, c, d, e) is implemented by the expression:

$E(a, b, c, d, e) = abc + abd + acd + bcd + abe + ace + ade + bce + bde + cde + a'b'c'e' + a'b'd'e' + a'c'd'e' + b'c'd'e'$

Using only gates from Table 4.1 of the textbook we can generate all product terms but the OR operation of all 14 product terms must be implemented by a tree of gates. To minimize the delay in the implementation, we should use NAND gates. The generation of the product terms is done using 3 and 4-input NAND gates from Table 4.1. A 14-input NAND however is not available and should be obtained combining smaller gates. The large NAND gate may be decomposed into smaller ones as follows (for a 4-input NAND to 2-input NANDs):

$$(abcd)' = (ab)' + (cd)'$$

The possibilities are:

Network	Delay	
	t_{pLH}	t_{pHL}
A - First Level: 1 NAND-6, 1 NAND-8, Second Level: 1 OR-2	0.40+0.037L	0.64+0.019L
B - First Level: 1 NAND-4, 2 NAND-5, Second Level: 1 OR-3	0.37+0.038L	0.70+0.022L
C - First Level: 2 NAND-3, 2 NAND-4, Second Level: 1 OR-4	0.27+0.038L	0.62+0.025L

Even though the LH transition delay of network C is less than the one for network A, the HL transition delay of network C becomes worse when the output load is greater or equal to 3. For this reason we consider network A as the implementation of the 14-input NAND, since it is going to be less susceptible to output load values. The resulting circuit is presented in Figure 6.1, on page 86.

The delay of the network is obtained from the critical paths NAND-4 $\rightarrow$ NAND-6 $\rightarrow$ OR-2 or NAND-3 $\rightarrow$ NAND-8 $\rightarrow$ OR-2:

$$\begin{aligned}
T_{pLH}(net) &= max(t_{pHL}(\text{NAND-4}) + t_{pLH}(\text{NAND-6}) + t_{pLH}(\text{OR-2}), \\
&\quad t_{pHL}(\text{NAND-3}) + t_{pLH}(\text{NAND-8}) + t_{pLH}(\text{OR-2})) \\
&= max(0.12 + 0.051 \times 1 + 0.24 + 0.037 \times 1 + 0.12 + 0.037 \times L, \\
&\quad 0.09 + 0.039 + 0.24 + 0.038 + 0.12 + 0.039L) \\
&= max(0.57 + 0.037L, 0.53 + 0.037L) = 0.57 + 0.037L \\
T_{pHL}(net) &= max(t_{pLH}(\text{NAND-4}) + t_{pHL}(\text{NAND-6}) + t_{pHL}(\text{OR-2}), \\
&\quad t_{pLH}(\text{NAND-3}) + t_{pHL}(\text{NAND-8}) + t_{pHL}(\text{OR-2})) \\
&= max(0.10 + 0.037 \times 1 + 0.36 + 0.019 \times 1 + 0.20 + 0.019 \times L, \\
&\quad 0.07 + 0.038 + 0.42 + 0.019 + 0.2 + 0.019L) \\
&= max(0.72 + 0.019 \times L, 0.75 + 0.019L) = 0.75 + 0.019L
\end{aligned}$$

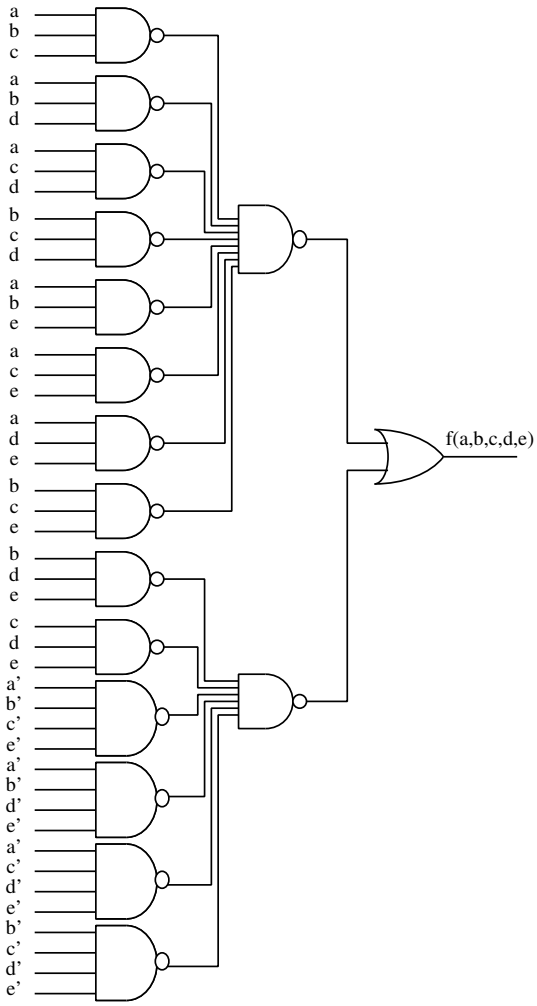

Figure 6.1: Single-error detector for 2-out-of-5 code

Exercise 6.3 A high-level specification for this system is:
Input: x is a decimal digit represented in BCD.
Output: two BCD digits y and z.
Function: $10y + z = 3x$.
From this specification we define the following switching functions:

$x_3x_2x_1x_0$	$y_3y_2y_1y_0$	$z_3z_2z_1z_0$
0000	0000	0000
0001	0000	0011
0010	0000	0110
0011	0000	1001
0100	0001	0010
0101	0001	0101
0110	0001	1000
0111	0010	0001
1000	0010	0100
1001	0010	0111

The simplified switching expressions are obtained from K-maps (not shown):

$$\begin{aligned}
y_3 &= y_2 = 0 \\
y_1 &= x_3 + x_2x_1x_0 \\
y_0 &= x_2x_1' + x_2x_0' \\
z_3 &= x_2x_1x_0' + x_2'x_1x_0 \\
z_2 &= x_3 + x_2x_1'x_0 + x_2'x_1x_0' \\
z_1 &= x_2x_1'x_0' + x_2'x_1'x_0 + x_2'x_1x_0' \\
z_0 &= x_0
\end{aligned}$$

The $(NAND, NAND)$ network is shown in Figure 6.2.

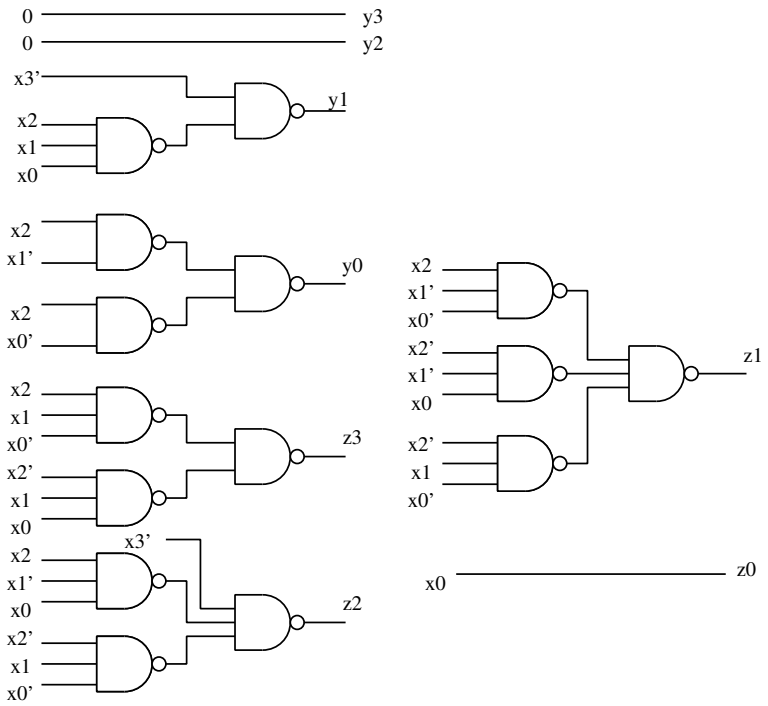

Figure 6.2: Network of Exercise 6.3

Exercise 6.5: The modification of the network of Example 4.6 is shown in Figure 6.3. Since we are asked to use 4 complex gates 2-AND/NOR2, the best solution is to use them on the level that has y_i and w_i as inputs. The fourth one should not be used to generate z_2 since this output is composed of 3 products and the complex gate is able to handle only 2. More logic is required to generate the third product and combine it with the output of the complex gate (that would take care of 2 products). For this reason, it is more interesting to use the fourth complex gate to generate z_1 and keep the same structure of gates that was used in Example 4.6 to generate z_2. Although the output of AN3 corresponds to z_0 we didn't use its output as the network output to avoid the influence of the z_0 output load on the delay of the other outputs.

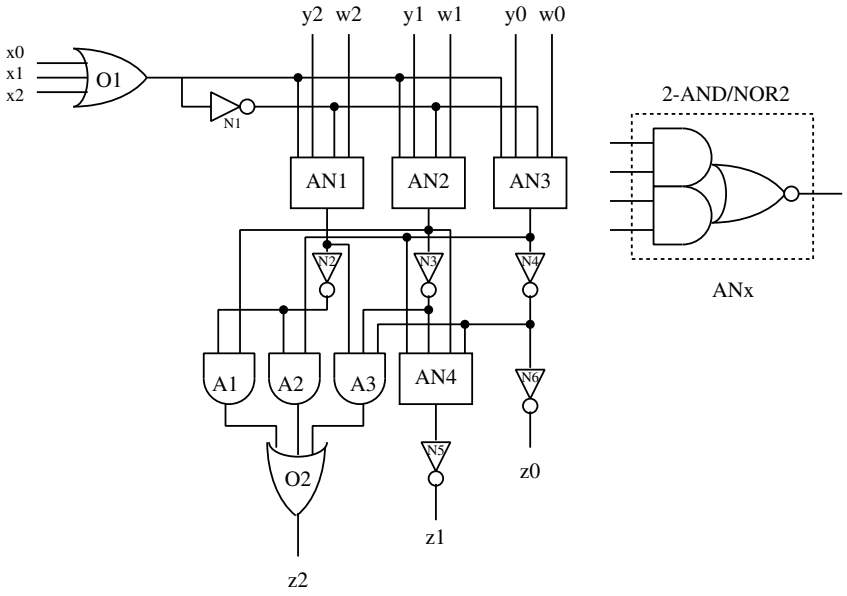

Figure 6.3: Network for Exercise 6.5

The network characteristics are:

- Load factor: 1
- Fanout factor: considering $F = 12$ we have $F(z_2) = F(z_1) = F(z_0) = 12$
- Network size: The NOT gates have size 1 and all others have size 2, thus the network has 23 equivalent gates. The size of the network on Example 4.6 was 38 equivalent gates.
- Number of levels: 6
- Network delays: consider the following table for gate delays:

gate	Identifier	Output Load	t_{pLH}(ns)	t_{pHL}(ns)
OR3	O1	4	0.27	0.43
NOT	N1/N4	3	0.13	0.10
2-AND/NOR2	AN2/AN3	3	0.40	0.18
2-AND/NOR2	AN4	1	0.25	0.13
2-AND/NOR2	AN1	2	0.32	0.16
NOT	N2/N3	2	0.10	0.08
NOT	N5	L_1	$0.02 + 0.038L_1$	$0.05 + 0.017L_1$
AND3	A3	1	0.24	0.20
OR3	O2	L_2	$0.12 + 0.038L_2$	$0.34 + 0.022L_2$

The first critical path we may consider is $O1 \rightarrow N1 \rightarrow AN1 \rightarrow N2 \rightarrow A3 \rightarrow O2$ that results in the following delays:

$$\begin{aligned} T_{pLH}(x_1, z_2) &= t_{pHL}(O1) + t_{pLH}(N1) + t_{pHL}(AN1) + t_{pLH}(N2) + t_{pLH}(A3) + t_{pLH}(O2) \\ &= 0.428 + 0.134 + 0.156 + 0.096 + 0.24 + 0.12 + 0.038L_2 = 1.17 + 0.038L_2 \\ T_{pHL}(x_1, z_2) &= t_{pLH}(O1) + t_{pHL}(N1) + t_{pLH}(AN1) + t_{pHL}(N2) + t_{pHL}(A3) + t_{pHL}(O2) \\ &= 0.272 + 0.101 + 0.32 + 0.084 + 0.2 + 0.34 + 0.022L_2 = 1.32 + 0.022L_2 \end{aligned}$$

Another path that may be considered is $O1 \rightarrow N1 \rightarrow AN2 \rightarrow N3 \rightarrow AN4 \rightarrow N5$, that results in the following delays:

$$\begin{aligned} T_{pLH}(x_1, z_1) &= t_{pHL}(O1) + t_{pLH}(N1) + t_{pHL}(AN2) + t_{pLH}(N3) + t_{pHL}(A4) + t_{pLH}(N5) \\ &= 0.428 + 0.134 + 0.184 + 0.096 + 0.128 + 0.02 + 0.038L_1 = 0.99 + 0.038L_1 \\ T_{pHL}(x_1, z_1) &= t_{pLH}(O1) + t_{pHL}(N1) + t_{pLH}(AN2) + t_{pHL}(N3) + t_{pLH}(A4) + t_{pHL}(N5) \\ &= 0.272 + 0.101 + 0.395 + 0.084 + 0.245 + 0.05 + 0.017L_1 = 1.15 + 0.017L_1 \end{aligned}$$

We can see that the path from x_1 to z_2 is still the critical path in this circuit, however, the delay was reduced when compared to Example 4.6.

Exercise 6.7 Using XOR gates it's possible to get the expressions for equality or difference:

$$SAME = x' \oplus y$$
$$DIFFERENT = x \oplus y$$

Using these expressions, we obtain the expression for each output, z_2(GREATER), z_1(EQUAL) and z_0(LESS) as:

$$\begin{aligned} z_2 &= DIFFERENT.x + SAME.c_2 \\ z_1 &= SAME.c_1 \\ z_0 &= DIFFERENT.y + SAME.c_0 \end{aligned}$$

The gate network using XOR and NAND gates is shown in Figure 6.4, on page 91.

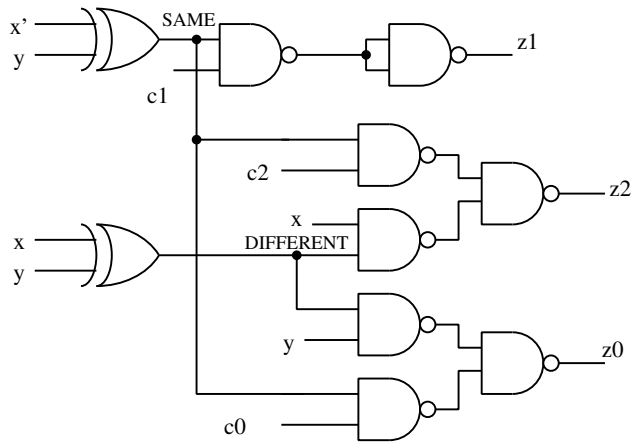

Figure 6.4: Comparator using XOR and NAND gates

Exercise 6.9 The MUX function is defined as:

$$\text{MUX}(x, y, s) = xs + ys'$$

and using this function we want to represent the following functions using muxes:

- $\text{OR}(a, b) = a + b = ab' + b = \text{MUX}(a, 1, b')$
- $\text{NOR}(a, b) = (a + b)' = a'b' = 0.b + a'b' = \text{MUX}(0, a', b)$
- $\text{NAND}(a, b, c) = \text{NAND}(a, \text{AND}(b, c))$
  $\text{AND}(b, c) = bc + 0.c' = \text{MUX}(b, 0, c)$
  $\text{NAND}(a, z) = (az)' = a' + z' = a'z + 1.z' = \text{MUX}(a', 1, z)$
  Thus $\text{NAND}(a, b, c) = \text{MUX}(a', 1, \text{MUX}(b, 0, c))$
- $\text{XOR}(a, b) = a \oplus b = a'b + ab' = \text{MUX}(a', a, b)$
- $\text{XNOR}(a, b) = a \oplus b' = ab + a'b' = \text{MUX}(a, a', b)$

Exercise 6.11 Tree of multiplexers:
Part (a) $E(a,b,c,d) = a'b + a'b'c' + bc'd + abd' + b'cd$
We use Shannon's decomposition to obtain the following four expressions:

$$\begin{aligned} E(a,b,0,0) &= a'b + a'b' + ab = a'b' + b = a' + b \\ E(a,b,0,1) &= a'b + a'b' + b = a' + b \\ E(a,b,1,0) &= a'b + ab = b \\ E(a,b,1,1) &= a'b + b' = a' + b' = (ab)' \end{aligned}$$

From these expressions we obtain the tree of multiplexers as shown in Figure 6.5.

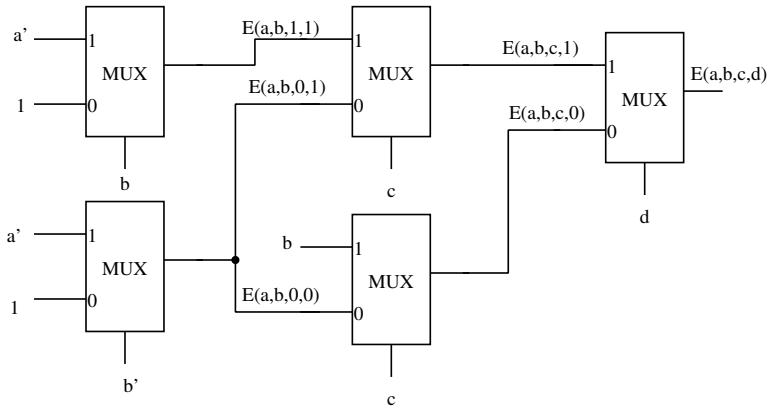

Figure 6.5: Multiplexer tree for $E(a,b,c,d) = a'b + a'b'c' + bc'd + abd' + b'cd$

Part (b) $E(a,b,c,d,e,f) = a \oplus b \oplus c \oplus d \oplus e \oplus f$
Using the same type of decomposition we get the functions shown in the next table:

(c,d,e,f)	$E(a,b,c,d,e,f)$
0000	$a \oplus b$
0001	$(a \oplus b)'$
0010	$(a \oplus b)'$
0011	$a \oplus b$
0100	$(a \oplus b)'$
0101	$a \oplus b$
0110	$a \oplus b$
0111	$(a \oplus b)'$
1000	$(a \oplus b)'$
1001	$a \oplus b$
1010	$a \oplus b$
1011	$(a \oplus b)'$
1100	$a \oplus b$
1101	$(a \oplus b)'$
1110	$(a \oplus b)'$
1111	$a \oplus b$

The straight implementation of the tree of multiplexers will look like the network shown in Figure 6.6(a). Simplifying the network by removing the repeated terms we obtain the network shown in Figure 6.6(b), that looks more like a linear array than a tree, but has the same number of levels and less muxes.

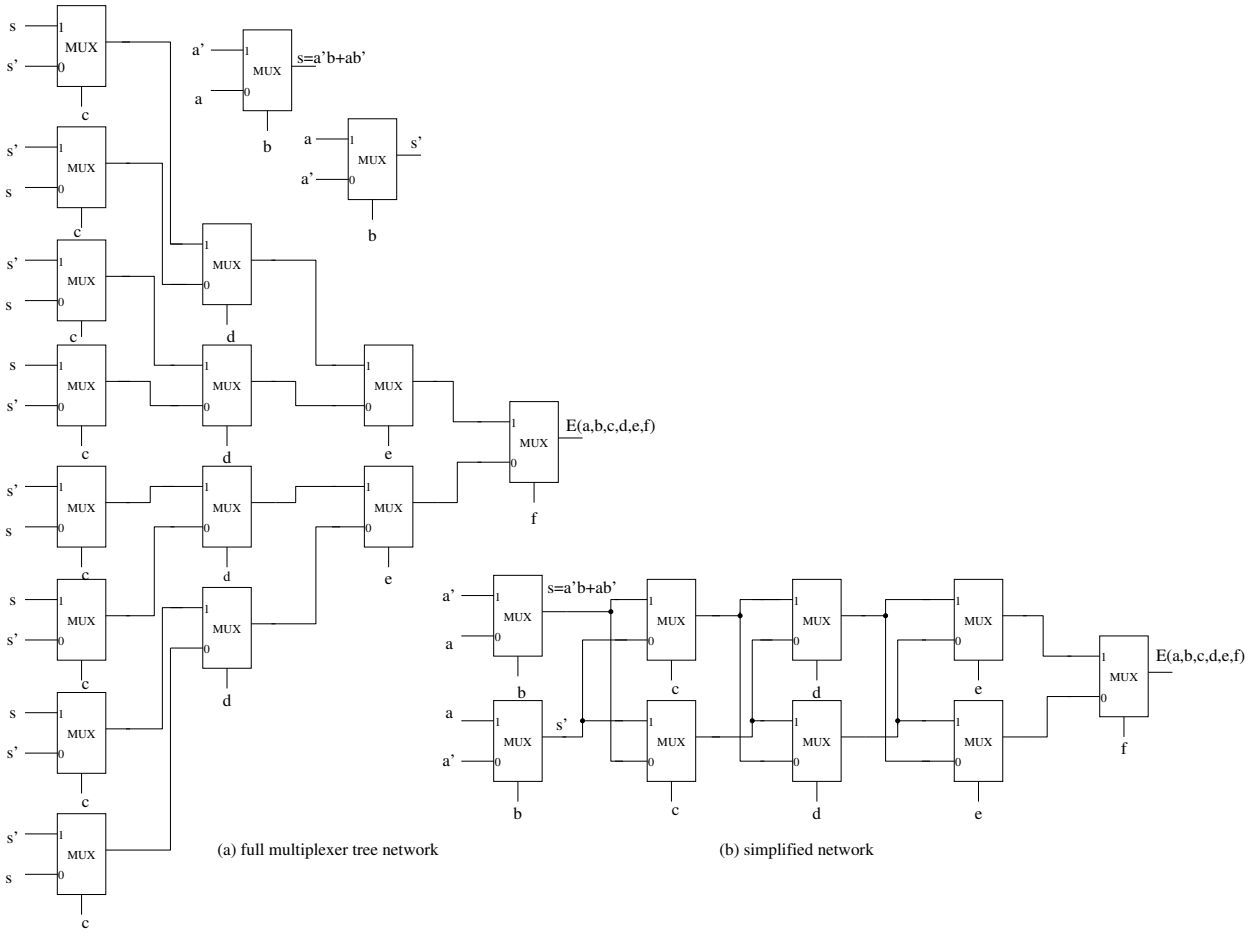

Figure 6.6: Multiplexer network for Exercise 6.11 - part (b) - $E(a,b,c,d,e,f) = a \oplus b \oplus c \oplus d \oplus e \oplus f$

A better multiplexer tree network is realized considering the implementation of the XOR and XNOR functions by multiplexers (Exercise 6.9) and the associativity of the XOR function as follows:

$$a \oplus b \oplus c \oplus d \oplus e \oplus f = [(a \oplus b) \oplus (c \oplus d)] \oplus (e \oplus f)$$

The network for this case is presented in Figure 6.7. Observe that it has only 3 levels of multiplexers in the critical path and 7 multiplexers. The previous implementation had 5 levels and used 9 multiplexers.

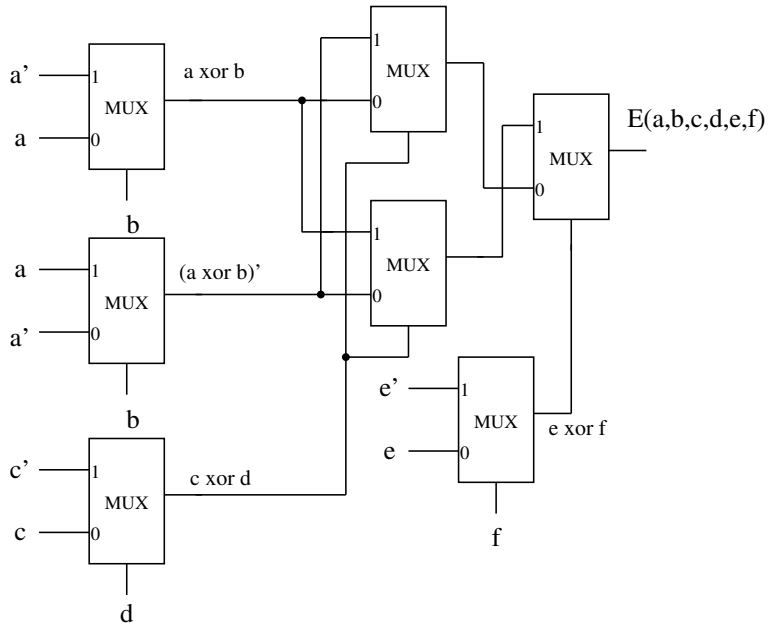

Figure 6.7: Implementation using XOR property to solve Exercise 6.11(b) - $E(a,b,c,d,e,f) = a \oplus b \oplus c \oplus d \oplus e \oplus f$