EE 2720

Handout # 17

• Decoders

General definition: A decoder is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different. The input code has generally fewer bits than the output code, and there is a one-to-one mapping from input code words into output code words. In a one-to-one mapping, each input code word produces a different output code word.

The most common decoder circuit is an n-to-2^n decoder or binary decoder. Such a decoder has a n-bit binary input code and a 1-out-of-2^n output code; (this decoder has n inputs and 2^n outputs). A binary decoder is used when you need to activate exactly one of 2^n outputs based on an n-bit input value; (n-bit input combination).

Let me show you, as an example, a 3-to-8 decoder. This decoder has three inputs a, b, c and eight outputs $y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7$. This decoder generates all the minterms of the three input variables a, b, c. Exactly one of the output lines $y_0, y_1, \ldots, y_7$ will be 1 for each combination of the values of the input variables a, b, c. Figure 1 (on next page) shows the 3-to-8 decoder, while table 1 (on next page) shows the truth table of the 3-to-8 decoder.

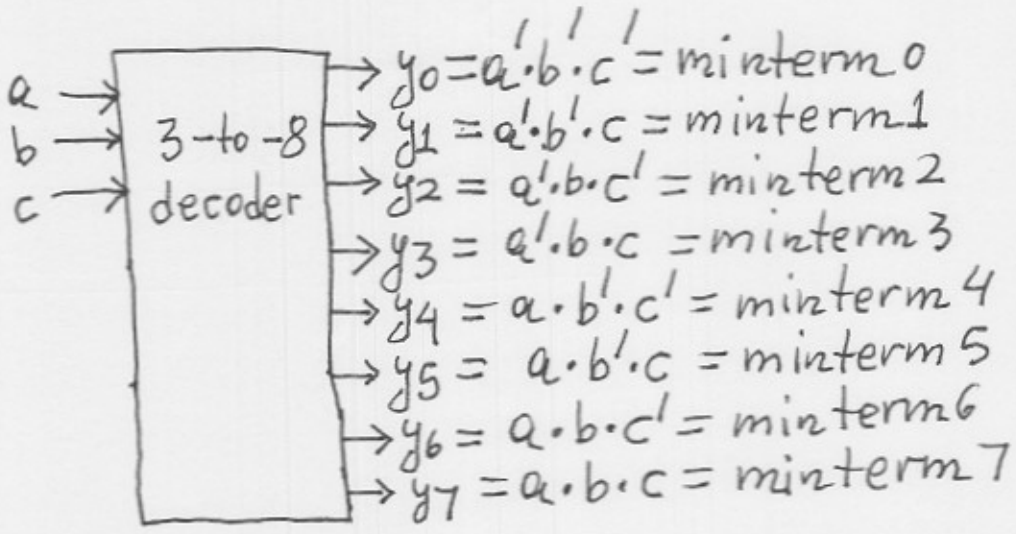


Figure 1: A 3-to-8 decoder.

a b c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0 0 0	1	0	0	0	0	0	0	0
0 0 1	0	1	0	0	0	0	0	0
0 1 0	0	0	1	0	0	0	0	0
0 1 1	0	0	0	1	0	0	0	0
1 0 0	0	0	0	0	1	0	0	0
1 0 1	0	0	0	0	0	1	0	0
1 1 0	0	0	0	0	0	0	1	0
1 1 1	0	0	0	0	0	0	0	1

Table 1: Truth table of the 3-to-8 decoder.

As another example, consider a 4-to-16 decoder which is shown in figure 2 on next page. This decoder has four inputs a, b, c, d and 16 outputs $y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14}, y_{15}$. Again this decoder generates all the minterms of the four input variables a, b, c, d and of course, like before, <u>exactly</u> one of the outputs of the decoder namely $y_0, y_1, \ldots, y_{15}$ will be 1 for each com-

bination of the values of the input variables a, b, c, d. For example, if abcd = 0000, then $y_0 = 1$ (and all other y_i's are 0), so $y_0 = a' \cdot b' \cdot c' \cdot d' = \text{minterm } 0$, if abcd = 0001, then $y_1 = 1$ and all other y_i's are 0, so $y_1 = a' \cdot b' \cdot c' \cdot d = \text{minterm } 1$, if abcd = 0010, then $y_2 = 1$ and all other y_i's are 0, so $y_2 = a' \cdot b' \cdot c \cdot d' = \text{minterm } 2$, etc...., if abcd = 1111, then $y_{15} = 1$ and all other y_i's are 0 so $y_{15} = a \cdot b \cdot c \cdot d = \text{minterm } 15$. Figure 2 follows:

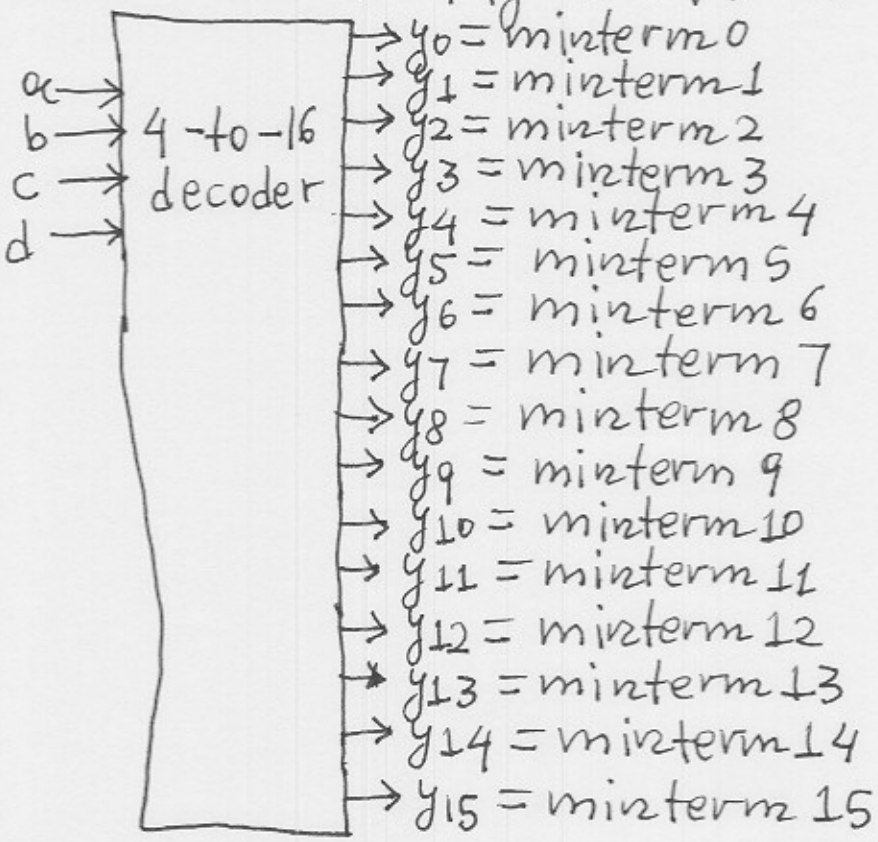


Figure 2: A 4-to-16 decoder. Here I do not provide algebraic expressions for the minterms, but I am sure you can easily derive them. I gave you four of these at the top of this page anyway. So if you want, get the rest.
You can easily get a truth table for the 4-to-16 decoder, so do it if you want.

Note: One can very easily obtain realizations for the 3-to-8 and 4-to-16 decoders based on inverters and AND gates. Do it if you want. It is very simple.

In general, an n-to-2^n decoder generates all 2^n minterms (or maxterms) of the n input variables. Why did I say maxterms? Refer to handout #8. Then you might remember that maxterm i = = (minterm i)'. So if we complement the outputs of the decoder, we get maxterms instead of minterms. Since a decoder can generate all 2^n minterms or maxterms of a logic function, a decoder can implement any logic function; (recall that any logic function can be written as a canonical sum which is a sum of minterms or as a canonical product which is a product of maxterms). Above I said that we can complement the outputs of a decoder to get maxterms. This is denoted by bubbles at the output as shown in figure 3 below:

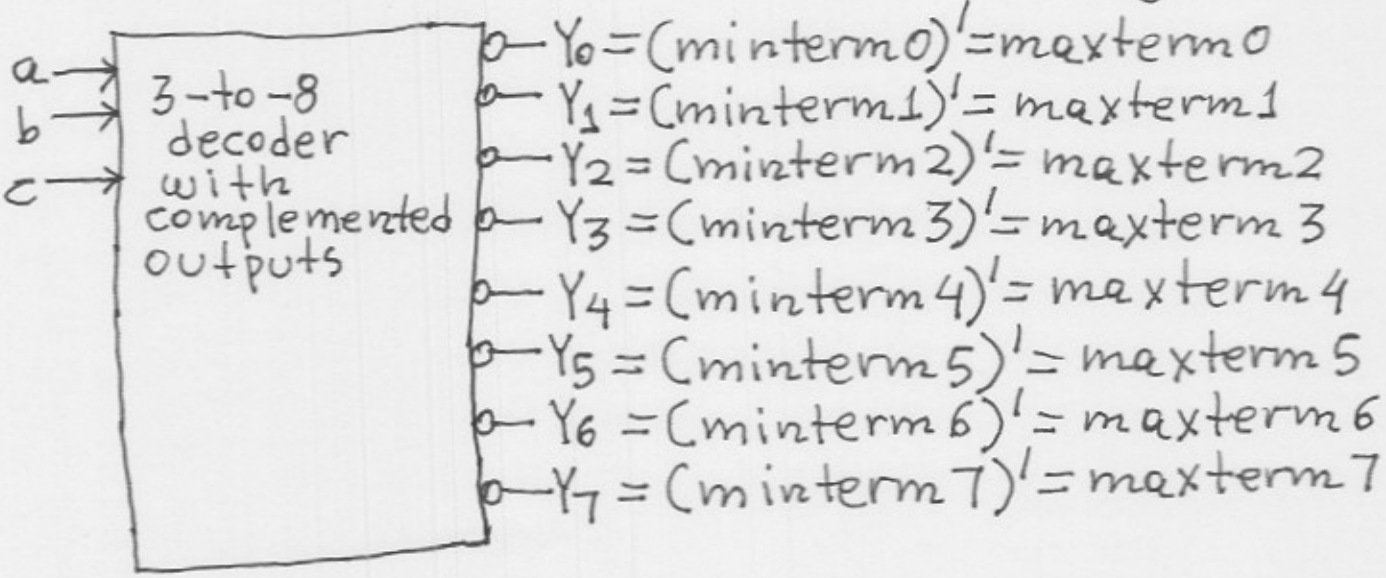


Figure 3: A 3-to-8 decoder with complemented outputs.

Note: A decoder with complemented outputs (like the one of fig. 3 of the previous page) is such that exactly one of its outputs is 0 for each combination of the values of the input variables. For example, refering to the 3-to-8 decoder of fig. 3 of previous page, if $abc=000$, then $Y_0=0$ and all other Y_is are 1, if $abc=001$, then $Y_1=0$ and all other Y_is are 1, etc...., if $abc=111$, then $Y_7=0$ and all other Y_is are 1.

The detailed implementation of the decoder of figure 3 of the previous page (or any decoder with complemented outputs) is based on NAND gates as shown in figure 4 below; (fig. 4 shows the decoder of fig. 3).

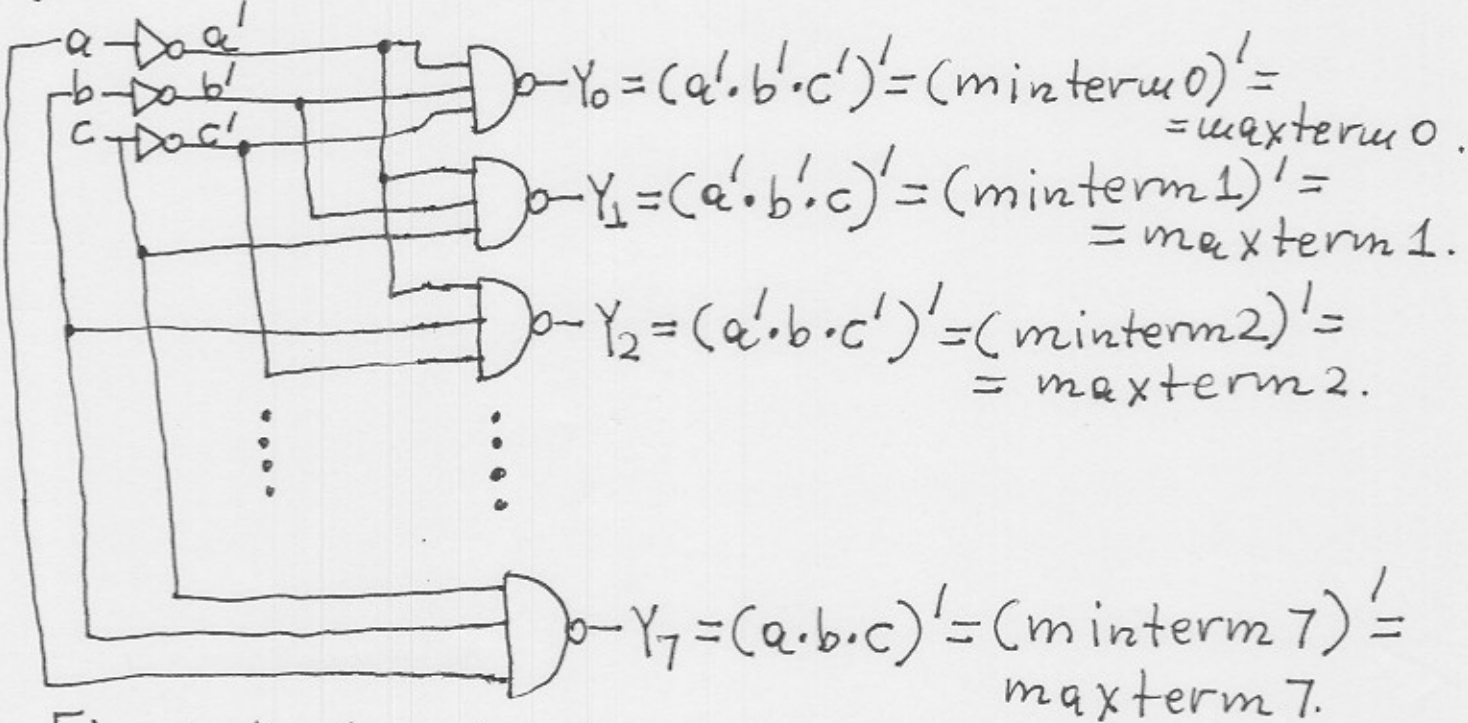


Figure 4: A realization of a 3-to-8 decoder with complemented outputs based on inverters and NAND gates. Here we can replace the inverters with NAND gates so that the realization will be based on only NAND gates.

Note: NAND gates are cheaper and faster, so they are preferred.

Note: I will call the decoders of figures 1 and 2 as decoders with uncomplemented outputs, while the decoder of figure 3 as a decoder with complemented outputs.

I will now show you examples of implementing logic functions with decoders. In the examples below, I will denote minterm i by m_i and maxterm i by M_i (just notations; nothing more).

Example 1: Using a decoder with complemented outputs and any other components that you need, implement the logic functions $F=\Sigma_{a,b,c,d}(1,2,4)$ and $G=\Sigma_{a,b,c,d}(4,7,9)$.

Answer: We have:

$F=$ minterm 1 + minterm 2 + minterm 4 $= m_1+m_2+m_4$

$=(m_1'\cdot m_2'\cdot m_4')'=$ (maxterm 1 · maxterm 2 · maxterm 4)'

$=(M_1\cdot M_2\cdot M_4)'$. Regarding function G we have

$G=$ minterm 4 + minterm 7 + minterm 9 $= m_4+m_7+m_9=$

$=(m_4'\cdot m_7'\cdot m_9')'=$ (maxterm 4 · maxterm 7 · maxterm 9)' $=$

$=(M_4\cdot M_7\cdot M_9)'$. The implementations of the functions F and G are shown in figure 5 on the next page.

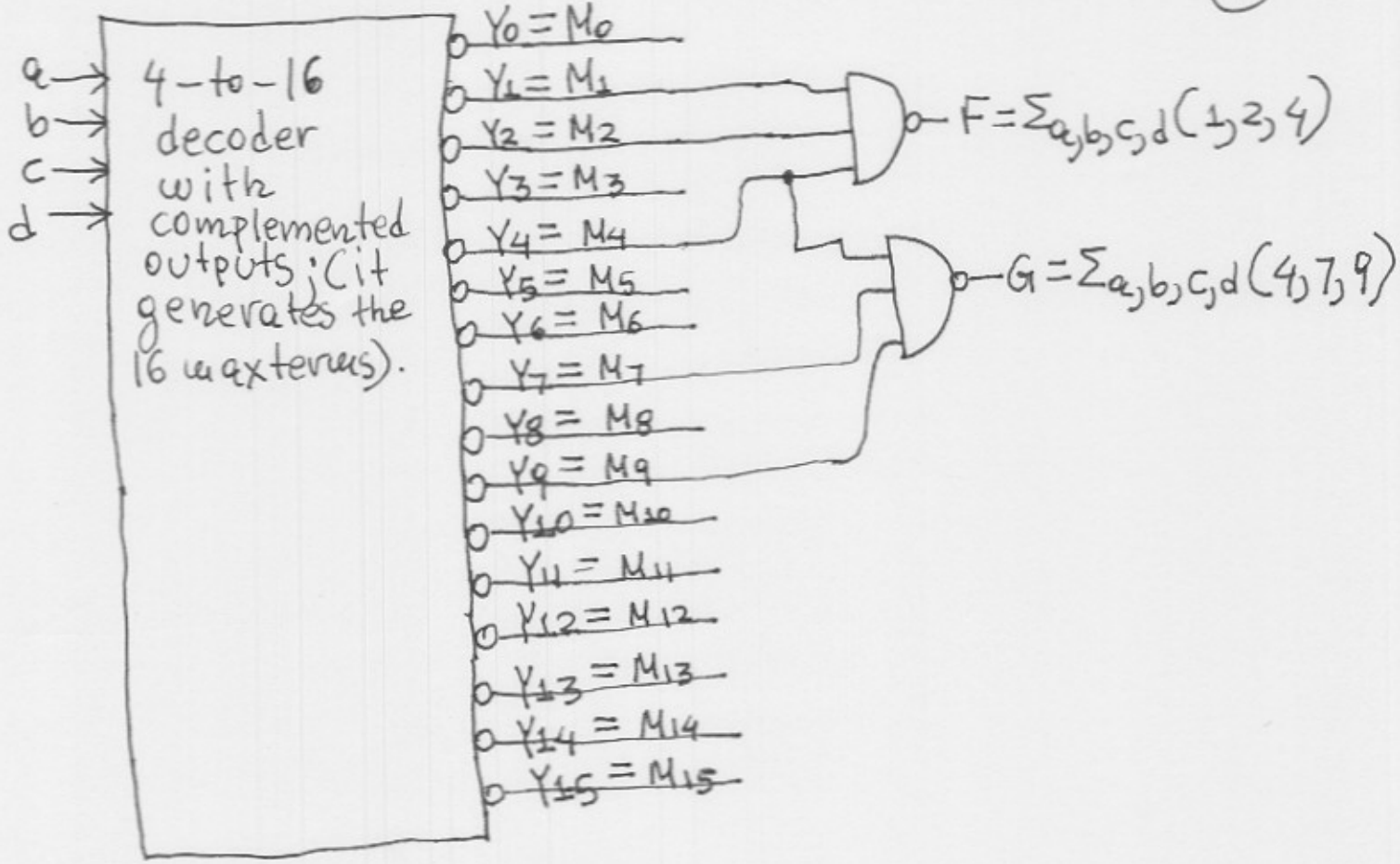


Figure 5: Implementation of the logic functions $F=\sum_{a,b,c,d}(1,2,4)$ and $G=\sum_{a,b,c,d}(4,7,9)$ using a 4-to-16 decoder with complemented outputs and two 3-input NAND gates. These are NAND-NAND implementations.

Example 2: Using a decoder with complemented outputs, and any other components that you need, implement the logic function $G=\prod_{a,b,c}(1,3,5,7)$.

Answer: We have:

$G=\prod_{a,b,c}(1,3,5,7)=$

$=\text{maxterm}1\cdot\text{maxterm}3\cdot\text{maxterm}5\cdot\text{maxterm}7$

$=M_1\cdot M_3\cdot M_5\cdot M_7$. The implementation of the function G is shown in figure 6 on the next page.

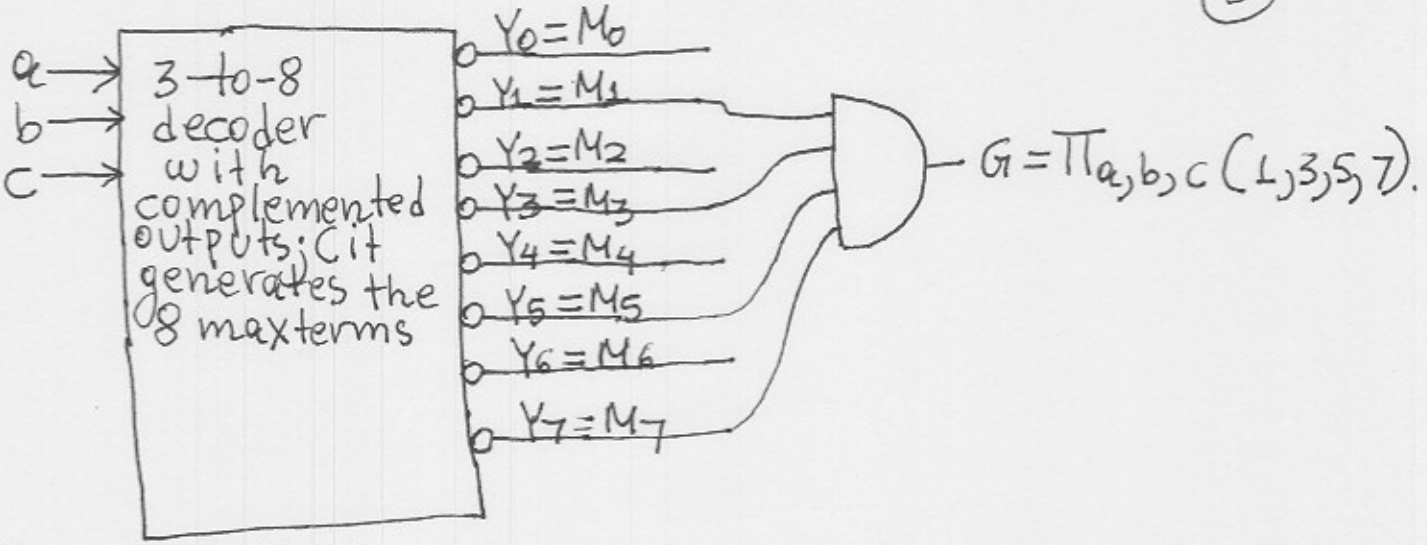


Figure 6: Implementation of the logic function $G = \Pi_{a,b,c}(1,3,5,7)$ using a 3-to-8 decoder with complemented outputs and one 4-input AND gate.

Example 3: Using a decoder with uncomplemented outputs and any other components that you need, implement the logic function $F = \Sigma_{x,y,z}(1,3,5,7)$.

Answer: we have:

$F = \Sigma_{x,y,z}(1,2,5,7) =$

minterm 1 + minterm 2 + minterm 5 + minterm 7 =

$= m_1 + m_2 + m_5 + m_7$. The implementation of F is shown in figure 7 below:

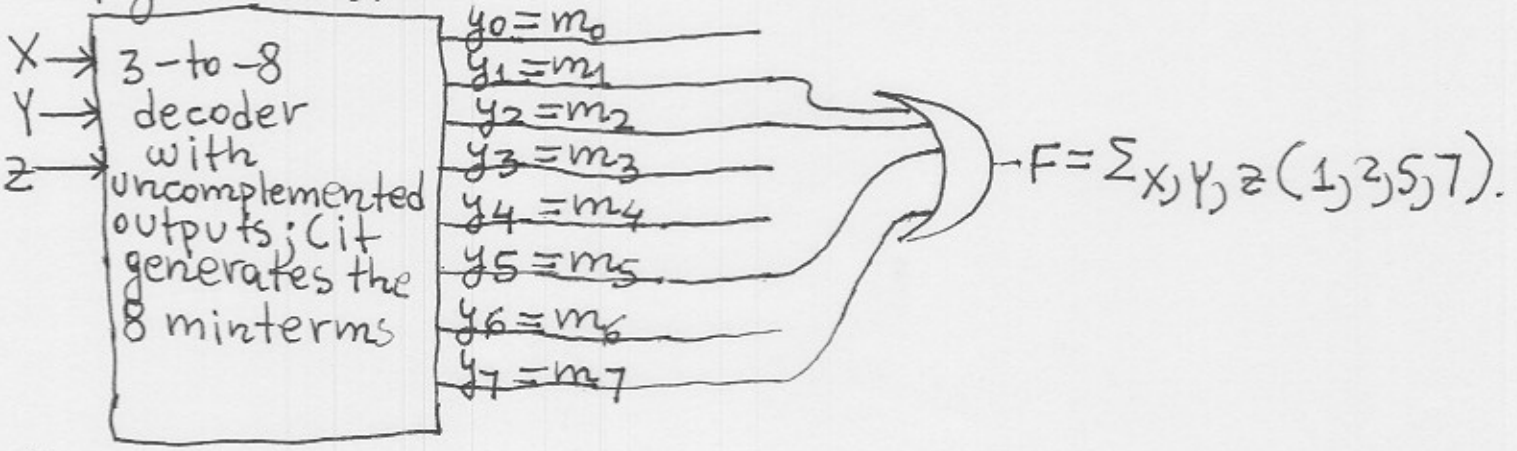


Figure 7: Implementation of $F = \Sigma_{x,y,z}(1,2,5,7)$ using a 3-to-8 decoder with uncomplemented outputs and one 4-input OR gate.

Example 4: Using a decoder with uncomplemented outputs and any other components that you need, implement the logic function $F = \Pi_{x,y,z}(1,2,5,7)$.

Answer: We have: $F = \Pi_{x,y,z}(1,3,5,7) = \Sigma_{x,y,z}(0,3,4,6) =$
$= \text{minterm } 0 + \text{minterm } 3 + \text{minterm } 4 + \text{minterm } 6 =$
$= m_0 + m_3 + m_4 + m_6$. The implementation of F is shown in figure 8 below:

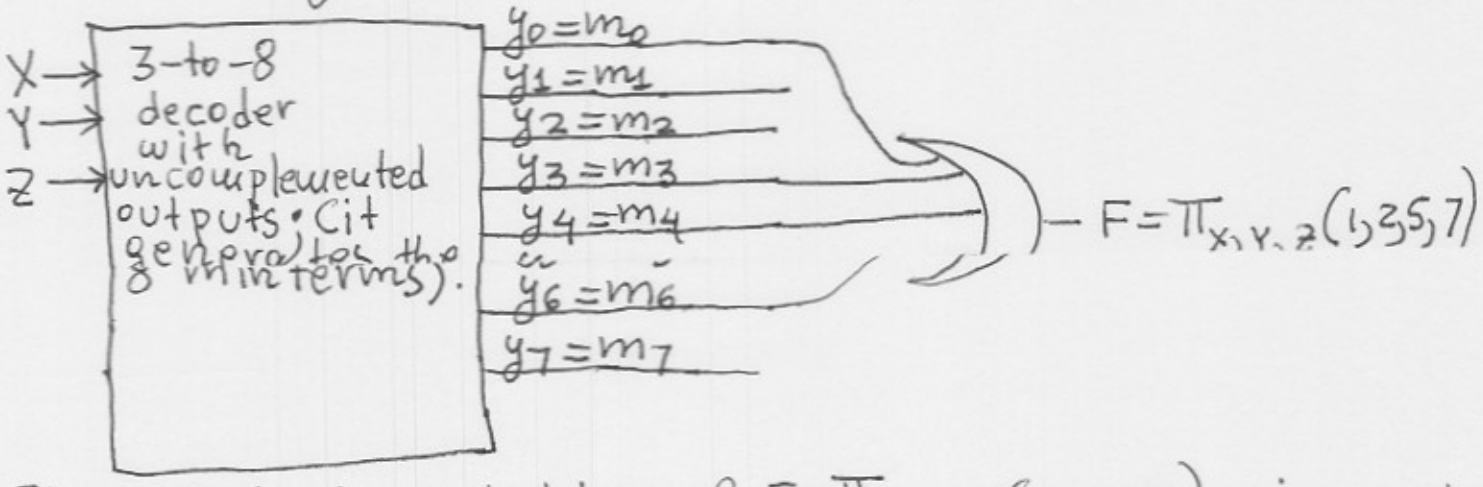


Figure 8: Implementation of $F = \Pi_{x,y,z}(1,2,5,7)$ using a 3-to-8 decoder with uncomplemented outputs and one 4-input OR gate.

Example 5: Using a decoder with uncomplemented outputs and any other components that you need, implement the logic function $F = (a+b)\cdot(a+c)\cdot(b+c)$

Answer: Since a decoder with uncomplemented outputs generates the minterms, we have to express F as a canonical sum which means a sum of minterms; (F is not in canonical sum neither in canonical product form). Since F is given as a product-of-sums expression, it is probably easier to obtain the canonical product from which we can very easily get the canonical sum. We have:

$F=(a+b)\cdot(a+c)\cdot(b+c)=(a+b+c\cdot c')\cdot(a+c+b\cdot b')\cdot(b+c+a\cdot a')=$
($c\cdot c'=0$, $b\cdot b'=0$, $a\cdot a'=0$)

$=(a+b+c)\cdot(a+b+c')\cdot(a+\cancel{c}+b)\cdot(a+c+b')\cdot$
$(b+\cancel{c}+a)\cdot(b+c+a')=(a+b+c)\cdot(a+b+c')\cdot$
$\cdot(a+b'+c)\cdot(a'+b+c)=\Pi_{a,b,c}(0,1,3,4)=$
$=\Sigma_{a,b,c}(3,5,6,7)$. We have now obtained the canonical sum for F so we can show the implementation. The implementation of F is shown in figure 9 below:

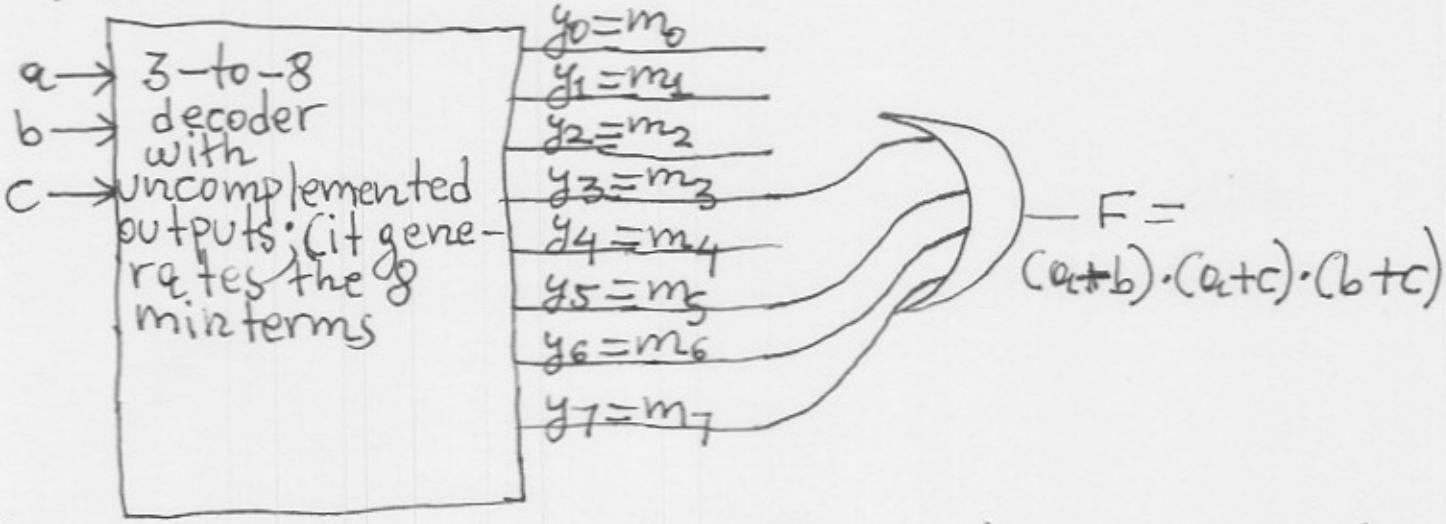


Figure 9: Implementation of $F=(a+b)\cdot(a+c)\cdot(b+c)$ using a 3-to-8 decoder with uncomplemented outputs and one 4-input OR gate.

Note: In the above derivation, I used theorem (T8').

Example 6: Repeat the problem of example 5 but now use a decoder with complemented outputs.

Answer: Since a decoder with complemented outputs generates the maxterms, we have to use the canonical product expression for F that we obtained in example 5 above. What we got is

$F = \Pi_{a,b,c}(0,1,2,4)$. The implementation of F is shown in figure 10 below:

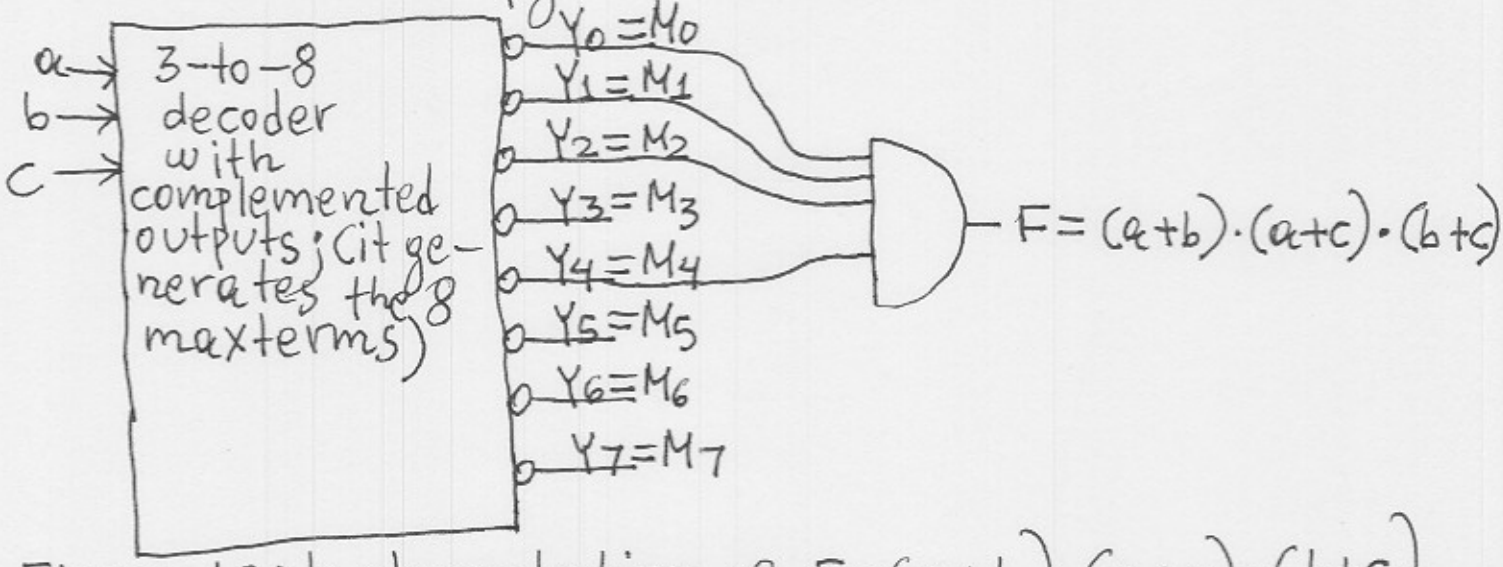


Figure 10: Implementation of $F = (a+b)\cdot(a+c)\cdot(b+c)$ using a 3-to-8 decoder with complemented outputs and one 4-input AND gate.

An interesting decoder follows. This is not an n-to-2^n decoder. This decoder is called the Seven-Segment Decoder.

• Seven-Segment Decoder

A seven-segment decoder is a 4-input 7-output logic circuit. Its inputs are named A, B, C, D and represent the ten BCD digits; (0, 1, ..., 8, 9). Its outputs are a, b, c, d, e, f, g. Consider the seven-segment display shown in figure 11 below:

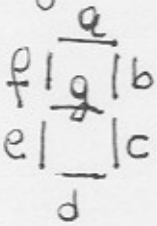


Figure 11: The seven-segment display.

With the seven-segment display of fig. 11 of the previous page we can represent all the ten decimal digits $0, 1, 2 \ldots, 9$, by illuminating the appropriate segments a, b, c, d, e, f, g. For example, if we illuminate segments a, b, c, d, e, f we represent digit 0, if we illuminate segments b, c we represent digit 1, if we illuminate all seven segments we represent digit 8 etc. If an output between the outputs a, b, c, d, e, f, g is 1, (output of the decoder circuit I mean), then the segment with the same name gets illuminated. For example, when output $a=1$, then segment a gets illuminated, when output $b=1$, then segment b gets illuminated, when output $g=1$, then segment g gets illuminated etc. The ten decimal digits $0, 1, 2 \ldots, 9$ composed out of the seven segments a, b, c, d, e, f, g are shown in figure 12 below:

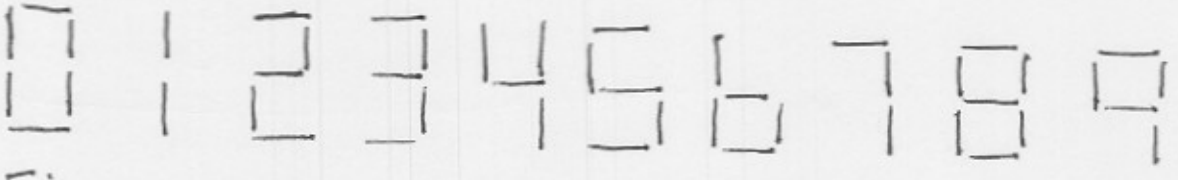


Figure 12: The ten decimal digits composed out of the seven segments a, b, c, d, e, f, g.

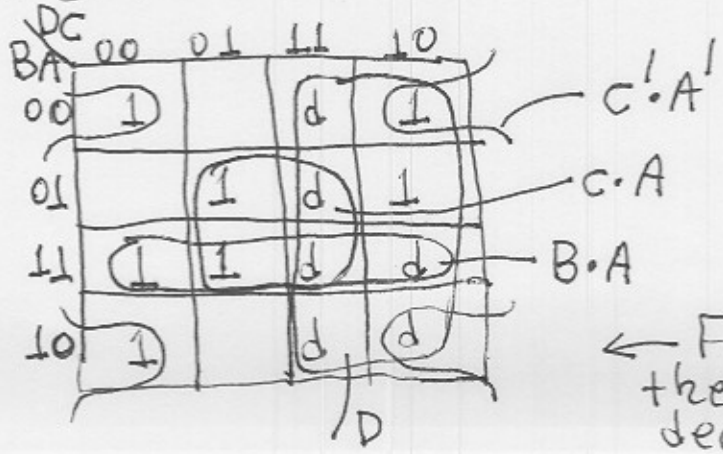
Note: The seven-segment decoder is used in watches, calculators, and instruments to display decimal data. The truth table for the seven-segment decoder describing the relationship between inputs A, B, C, D and outputs a, b, c, d, e, f, g is shown in table 2 on the next page.

DCBA	a	b	c	d	e	f	g	Comments
0000	1	1	1	1	1	1	0	representing digit 0
0001	0	1	1	0	0	0	0	representing digit 1
0010	1	1	0	1	1	0	1	representing digit 2
0011	1	1	1	1	0	0	1	representing digit 3
0100	0	1	1	0	0	1	1	representing digit 4
0101	1	0	1	1	0	1	1	representing digit 5
0110	0	0	1	1	1	1	1	representing digit 6
0111	1	1	1	0	0	0	0	representing digit 7
1000	1	1	1	1	1	1	1	representing digit 8
1001	1	1	1	0	0	1	1	representing digit 9
1010	d	d	d	d	d	d	d	
1011	d	d	d	d	d	d	d	
1100	d	d	d	d	d	d	d	
1101	d	d	d	d	d	d	d	
1110	d	d	d	d	d	d	d	
1111	d	d	d	d	d	d	d	

d means don't care

Table 2: Truth table for the seven-segment decoder. For input combinations DCBA 1010, 1011, ..., 1111, the outputs a, b, c, d, e, f, g are don't cares because input combinations 1010, 1011, ..., 1111 do not represent BCD digits, so we don't care what the values of the outputs are for these input combinations.

We are now going to obtain simplified sum-of-products expressions for a, b, c, d, e, f, g using Karnaugh maps. These Karnaugh maps are shown in figures 13, 14, 15, 16, 17, 18, 19.



← Figure 13: Karnaugh map for the output a of the seven-segment decoder.

The karnaugh map of fig. 13 of the previous page gives the following simplified sum-of-products expression for the output a shown in equation (1) below:

$$a = D + C' \cdot A' + B \cdot A + C \cdot A \quad (1)$$

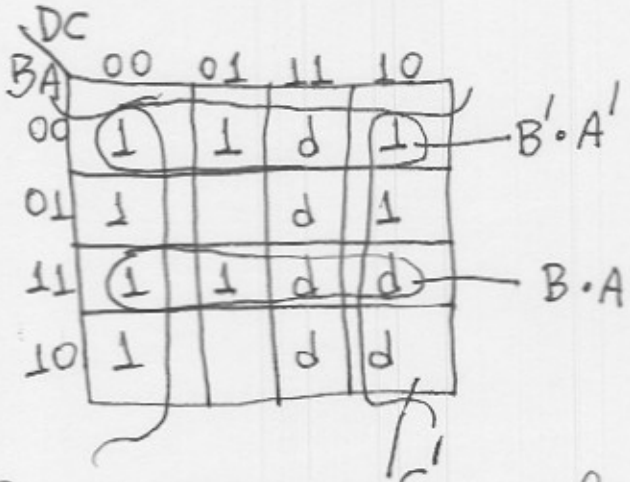


Fig. 14: Karnaugh map for the output b of the seven-segment decoder

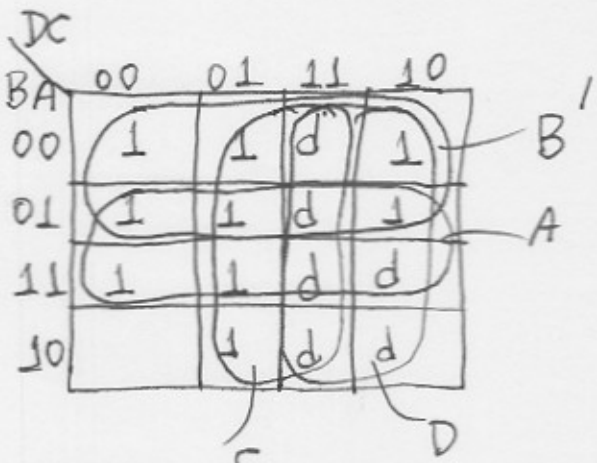


Fig. 15: Karnaugh map for the output c of the seven-segment decoder.

From the karnaugh maps of figures 14, 15, we get the following simplified sum-of-products expressions for the outputs b and c shown in equations (2), (3) below:

$$b = C' + B \cdot A + B' \cdot A' \quad (2)$$

$$c = C + D + A + B' \quad (3)$$

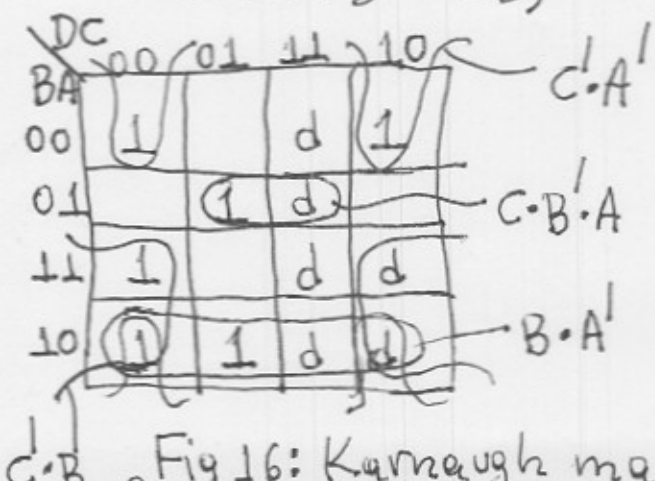


Fig 16: Karnaugh map for the output d of the seven-segment decoder.

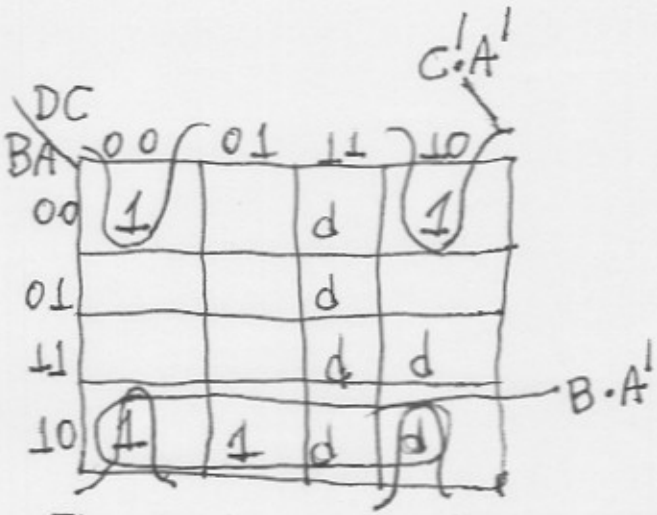


Fig. 17: Karnaugh map for the output e of the seven-segment decoder

From the karnaugh maps of figures 16, 17, we get the following simplified sum-of-products expressions for the outputs d and e shown in equations (4), (5) below:

$$d = C' \cdot B + B \cdot A' + C \cdot B' \cdot A + C' \cdot A' \quad (4)$$

$$e = B \cdot A' + C' \cdot A' \quad (5)$$

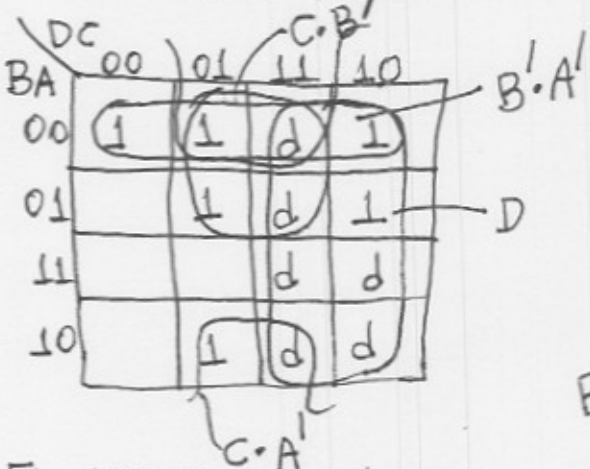


Fig 18: Karnaugh map for the output f of the seven-segment decoder.

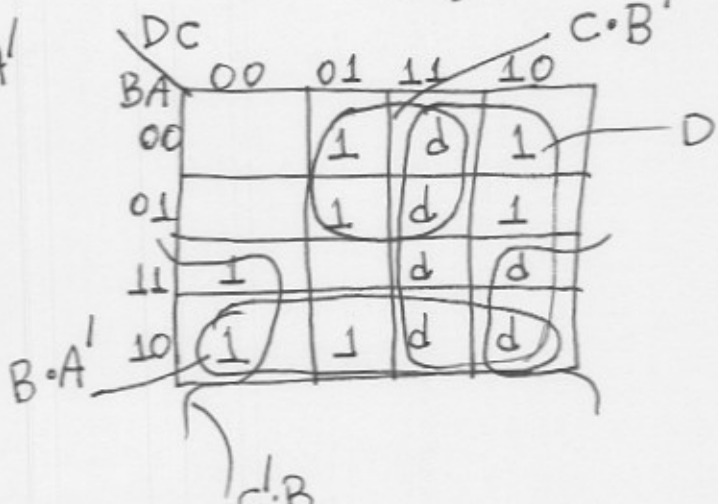


Fig. 19: Karnaugh map for the output g of the seven-segment decoder.

From the karnaugh maps of figures 18, 19 above, we get the following simplified sum-of-products expressions for the outputs f and g shown in equations (6), (7) below

$$f = D + C \cdot A' + B' \cdot A' + C \cdot B' \quad (6)$$

$$g = D + B \cdot A' + C' \cdot B + C \cdot B' \quad (7)$$

From equations (1), (2), (3), (4), (5), (6), (7), we get the following simplified AND-OR realization for the seven-segment decoder shown in figure 20 on the next page:

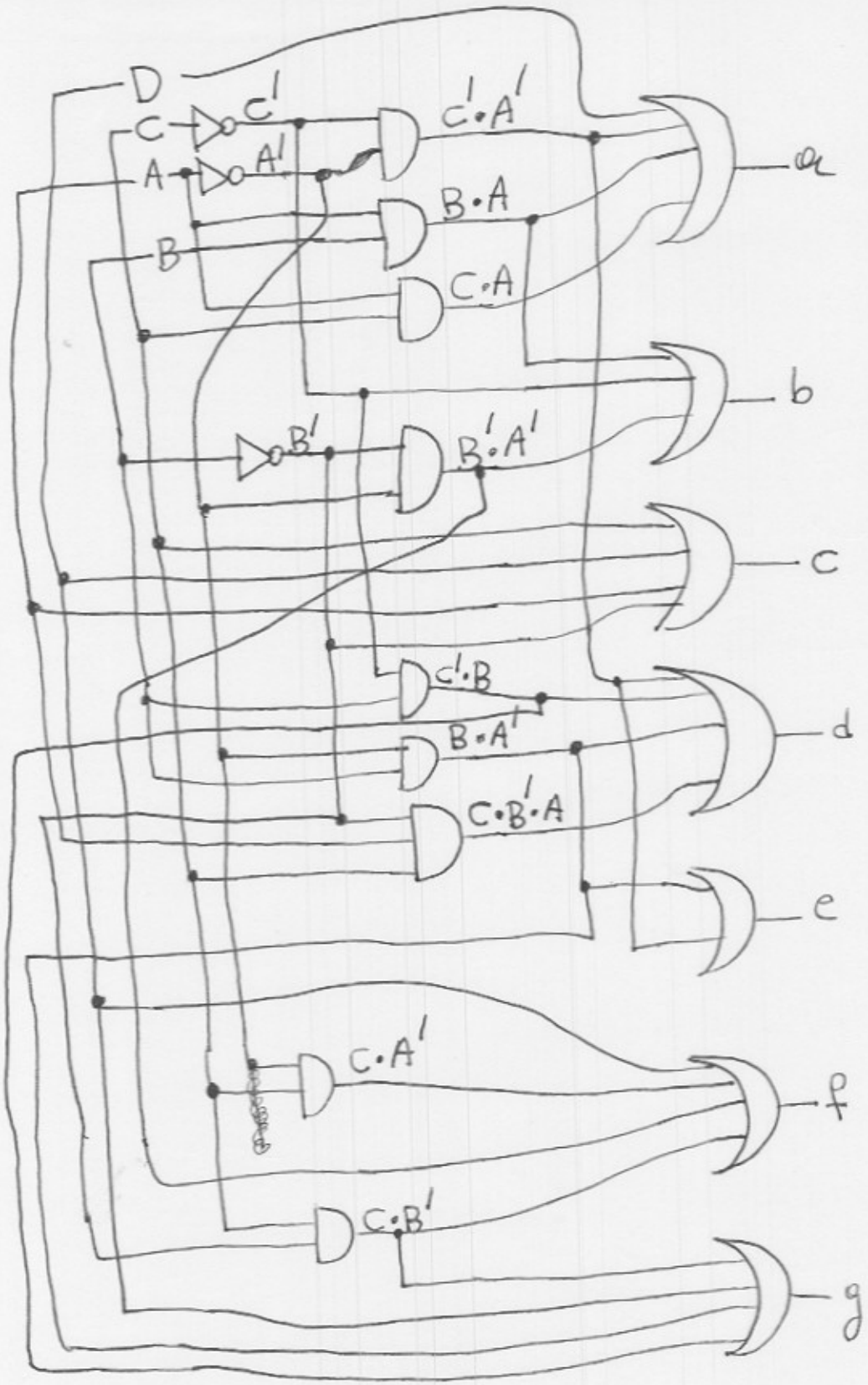


Figure 20: Simplified AND-OR realization for the seven-segment decoder. I hope I got it correct. It is a complicated figure.

Note: By applying the graphical approach (bubbles etc), you can very easily transform the logic circuit of figure 20 into one that relies only on NAND gates. Do it if you want.

Note: I am not sure if the sum-of products expressions that I got in equations (1), (2), (3), (4), (5), (6), (7) are minimal sums. I just did not check!! Please check and obtain minimal sums for the outputs a, b, c, d, e, f, g (if these are not minimal sums already). You know how to do this by now.

---

of-sums expressions for the outputs a, b, c, d, e, f, g of the seven-segment decoder; (I might assign this as a homework problem later; you know how to do it by now.).