EE303 Midterm Soultion&Criteria

1. [Total 30 Pts.] Variants of the problems provided in the course materials.

(a) [10 Pts.] Add (-12) + (-7) in binary, using the 2's complements to represent negative numbers and a word length of 5 bits (including sign). Indicate whether the 5-bit result is correct.

Soultion)

$(-12)_{10} = (10100)_2$ in 2's compliment

$(-7)_{10} = (11001)_2$ in 2's compliment

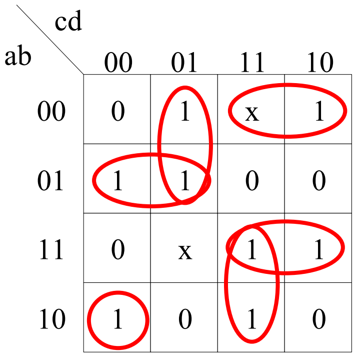
```
        10100
+       11001
-----------------------                ,        5-bit result is incorrect due to overflow
      ~~1~~01101
```

Criteria)

- No partial credit if the 5 bit result is miscalculated
- +5 point credit for correct calculation and wrong verification

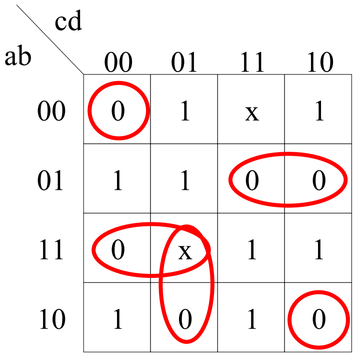
(b) [10 Pts.] Derive the most simplified expression in **(i)** sum-of-products and **(ii)** product-of-sums forms with the Karnaugh map below.

Soultion)



ab \ cd	00	01	11	10
00	0	1	x	1
01	1	1	0	0
11	0	x	1	1
10	1	0	1	0

(i) y = a'c'd + a'bc' + acd +abc +a'b'c + ab'c'd'



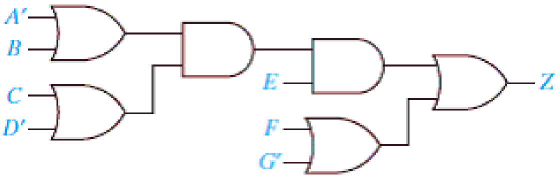
ab \ cd	00	01	11	10
00	0	1	x	1
01	1	1	0	0
11	0	x	1	1
10	1	0	1	0

(ii) y = (a + b' + c')(a' + c + d')(a' + b' + c)(a + b + c +d)(a' + b + c' +d)

Criteria)

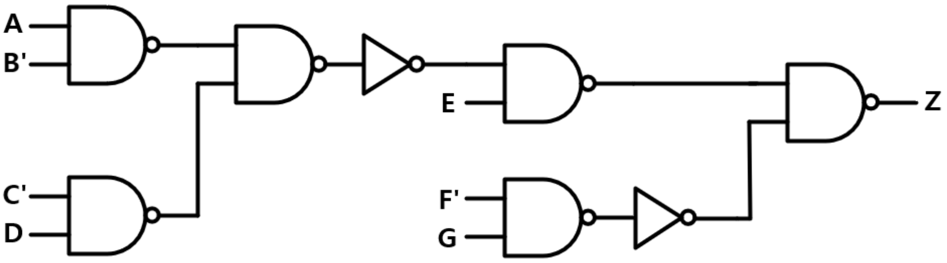
- No partial credit if the expression is not in its simplest form
- For (i), other forms of simplest expressions are also valid for full credit

(c) [10 Pts.] Convert the circuit shown below **(i)** to all NAND gates, by adding bubbles and inverters where necessary, and **(ii)** to all NOR gates (an inverter at the output is allowed).

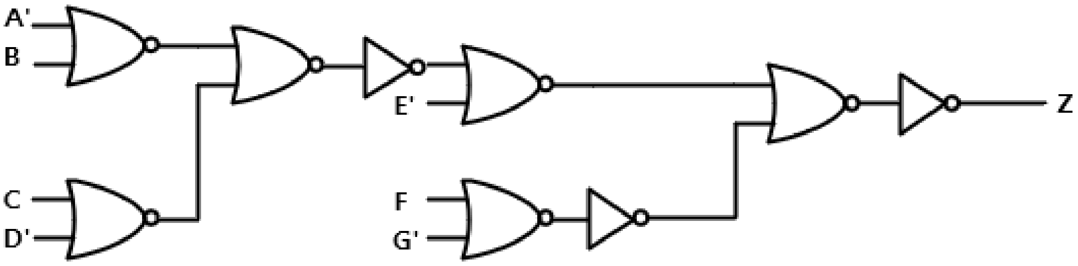


Solution)

(i)



(ii)



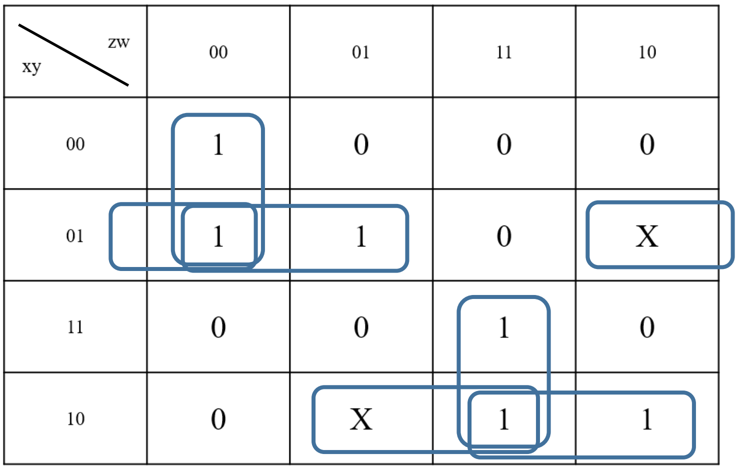
Criteria)

- No partial credit for wrong logic circuit
- No partial credit for usage of gates other than NAND, NOR, INV
- Usage of bubbles are valid for full credit as specified in the problem
- Full credit for other equivalent circuits

2. [Total 15 Pts.] Consider the function F(x,y,z,w) = Σm(0,4,5,10,11,15) with don’t care conditions d(x,y,z,w) = Σd(6,9). Justify your answers by using Karnaugh maps.

(a) [5 Pts.] Find all prime implicants.

Solution)



xy \ zw	00	01	11	10
00	1	0	0	0
01	1	1	0	X
11	0	0	1	0
10	0	X	1	1

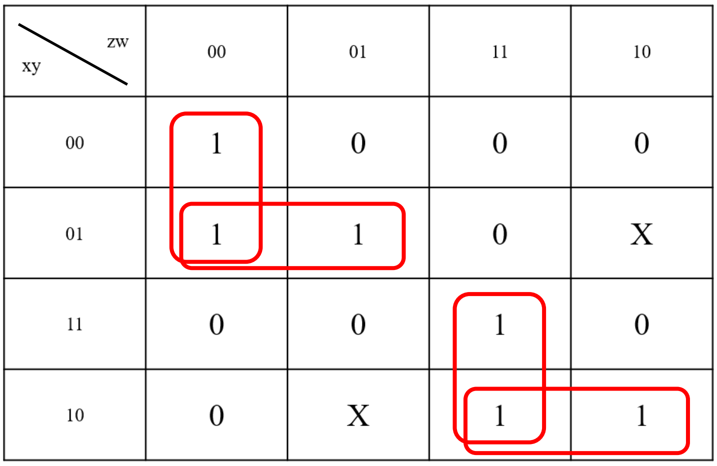
Prime implicants : x’z’w’, x’yw’, x’yz’, xzw, xy’w, xy’z

Criteria)

- Found all 6 prime implicants: +5
- Found 5 prime implicants: +4
- Found 4 prime implicants: +3
- Found 3 prime implicants: +2
- Found 1~2 prime implicants: +1
- -0.5 point for each of extra wrong answer
- Only K-map provided as an answer (and if K-map notation is correct): +2.5

(b) [5 Pts.] Find all essential prime implicants.

Solution)



xy \ zw	00	01	11	10
00	1	0	0	0
01	1	1	0	X
11	0	0	1	0
10	0	X	1	1

Essential prime implicants : x’z’w’, x’yz’, xzw, xy’z

Criteria)

- Found all 4 essential prime implicants: +5
- Found 3 essential prime implicants: +4
- Found 2 essential prime implicants: +3
- Found 1 essential prime implicants: +2
- Only K-map provided as an answer (and only if K-map notation is correct): +2.5

(c) [5 Pts.] Give the minimal sum of products expression for F.

Solution)

F = x’z’w’ + x’yz’ + xzw + xy’z

Criteria)

- Found all 4 essential prime implicants: +5
- Found 3 essential prime implicants: +4
- Found 2 essential prime implicants: +3
- Found 1 essential prime implicants: +2

3. [Total 15 Pts.] Using the Boolean Postulates and Theorems, show that

a’bc’d + (a’ + bc)(a + c’d’) + bc’d + a’bc’ = abcd + a’c’d’ + abd + abcd’ + bc’d

Note: Show each step.

Solution)

(example)

a’bc’d + (a’ + bc)(a + c’d’) + bc’d + a’bc’
= a’bc’d + *a’a* + *a’c’d’* + *abc* + *bcc’d’* + bc’d + a’bc’
= a’bc’d + a’c’d’ + abc + bc’d + a’bc’
= a’bc’d + a’c’d’ + (*abcd* + *abcd’*) + (*abc’d* + *bc’d*) + a’bc’
= a’bc’d + a’c’d’ + (*abcd* + *abcd*) + abcd’ + abc’d + bc’d + a’bc’
= a’bc’d + a’c’d’ + abcd + abcd’ + *abd* + bc’d + a’bc’
= *a’bc’* + a’c’d’ + abcd + abcd’ + abd + bc’d
= *a’bc’d* + *a’bc’d’* + a’c’d’ + abcd + abcd’ + abd + bc’d
= a’c’d’ + abcd + abcd’ + abd + bc’d

Criteria)

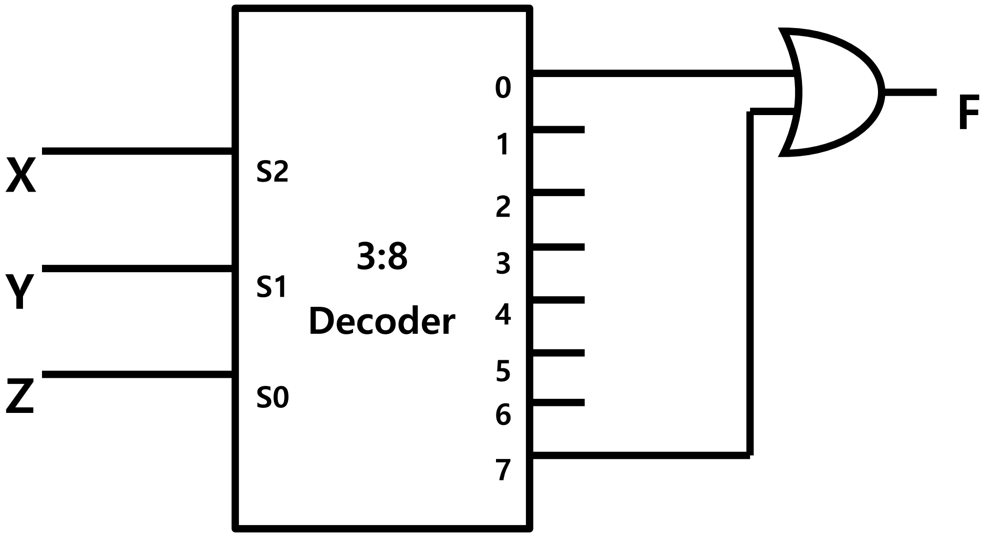
- Proved without any error: +15
- Proved but with minor error : +7.5

4. [Total 20 Pts.] Implement the following truth table as described below:

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(a) [5 Pts.] Using a 3:8 decoder and one other logic gate

Solution)

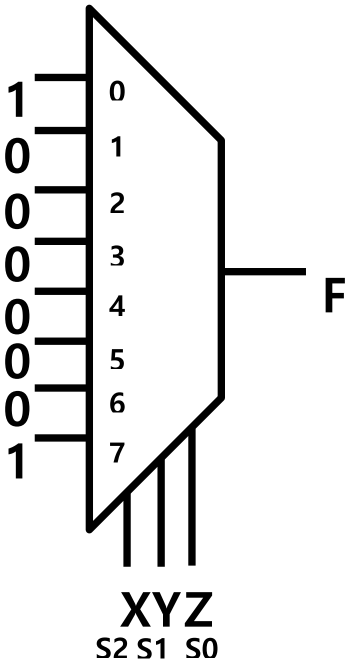


Criteria)

- Correct implementation of F with correct input order and single gate : (+5pts)
- Correct implementation of F with single gate, but incorrect input order : (+4pts)
- Correct implementation of F with correct input order but not single gate : (+1pt)
- Correct implementation of F, but with inverted inputs : (+1pt)
- Incorrect implementation of F, or using both inverted inputs and multiple gates : No points

(b) [5 Pts.] Using a 8:1 multiplexer

Solution)

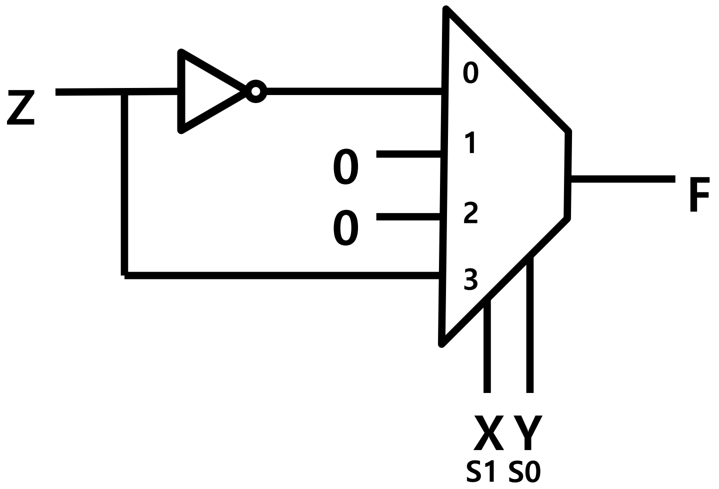


Criteria)

- Correct implementation of F with 8:1 Mux without any gates : (+5pts)
- Correct implementation of F with 8:1 Mux, but with incorrect input order : (+4pts)
- Correct implementation of F with 8:1 Mux, but used other gates : (+1pt)
- Correct implementation of F with 8:1 Mux, but with inverted inputs : (+1pt)
- Incorrect implementation of F or using both other gates with inverted inputs : No points

(c) [5 Pts.] Using a 4:1 multiplexer and one inverter

Solution)

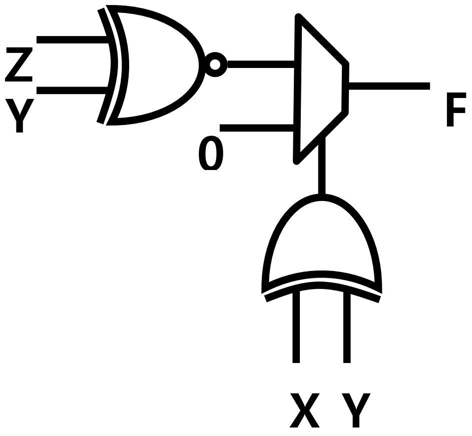


Criteria)

- Correct implementation of F with 4:1 Mux with single inverter : (+5pts)
- Correct implementation of F with 4:1 Mux, but with incorrect input order : (+4pts)
- Correct implementation of F with 4:1 Mux, but used other gates or used more than one inverter : (+1pt)
- Correct implementation of F with 4:1 Mux, but with inverted inputs : (+1pt)

(d) [5 Pts.] Using a 2:1 multiplexer and two other logic gates

Solution)



Criteria)

- Correct implementation of F with 2:1 Mux with only two gates : (+5pts)
- Correct implementation of F with 2:1 Mux, but with incorrect input order : (+4pts)
- Correct implementation of F with 2:1 Mux with only two gates, but with inverted inputs : (+1pt)
- Correct implementation of F with 2:1 Mux, but used more than two gates : (+1pt)
- Correct implementation of F with 2:1 Mux, but with inverted inputs : (+1pt)
- Incorrect implementation of F, or using both inverted inputs and more than 3 gates : No points

5. [Total 30 Pts.] Carry-lookahead adders make use of the generate and propagate signals to precompute whether or not stage *i* would output a carry. Similarly, an adder can be designed by using a propagate (P_i), a generate (G_i) and a delete (D_i) signal. Its truth table is shown below. The first two rows of the table correspond to the condition where the carry-out signal gets suppressed (deleted) at c_{i+1}, independent of the value at c_i.

x_i	y_i	c_i	s_i	c_{i+1}	Carry Status
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

(a) [15 Pts.] Derive the expressions for D_i, P_i and G_i in terms of x_i and y_i from the truth table above.

Solution)

- D_i is “high” when the two inputs x_i, y_i is “low” and it is independent of the value c_i. Therefore, $D_i = x_i' \cdot y_i'$
- P_i is “high” when the two inputs x_i, y_i is “low & high” or “high & low” and it is independent of the value c_i. Therefore, $P_i = x_i \oplus y_i$
- G_i is “high” when the two inputs x_i, y_i is “high” and it is independent of the value c_i. Therefore, $G_i = x_i \cdot y_i$

Criteria)

- Derive all expressions: 15 pts
- Derive two expressions among them: 10 pts
- Derive one expression only: 5 pts
- None of expressions are derived: 0 pts

(b) [15 Pts.] Derive the expressions for s_i and c_{i+1} in terms of D_i, P_i, G_i, and c_i.

Solution)

D_i	P_i	G_i	C_i	S_i	C_(i+1)
1	0	0	0	0	0
1	0	0	1	1	0
0	1	0	0	1	0
0	1	0	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	1

1) Derive s_i in terms of D_i, P_i, G_i, and c_i.

By using K-amp, You can draw the table below.

S_i		G_i·C_i			
		00	01	11	10
D_i·P_i	00	X	X	1	0
	01	1	0	X	X
	11	X	X	X	X
	10	0	1	X	X

You should minimize the logic function as much as possible.
So, There is only one solution.

$$\therefore s_i = P_i c_i' + P_i' c_i$$

2) Derive c_{i+1} in terms of D_i, P_i, G_i, and c_i.

By using K-amp, You can draw the table below.

C_(i+1)		G_i·C_i			
		00	01	11	10
D_i·P_i	00	X	X	1	1
	01	0	1	X	X
	11	X	X	X	X
	10	0	0	X	X

You should minimize the logic function as much as possible.
In this case There are two possible solutions.

$$\therefore c_{i+1} = G_i + P_i c_i \text{ or } c_{i+1} = G_i + D_i' c_i$$

Criteria)

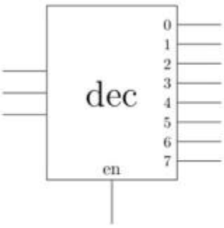
- Derive s_i in terms of D_i, P_i, G_i, and c_i with minimized logic form: 7.5 pts
- Derive s_i in terms of D_i, P_i, G_i, and c_i without minimized logic form: 2 pts
- Derive c_{i+1} in terms of D_i, P_i, G_i, and c_i with minimized logic: 7.5 pts
- Derive c_{i+1} in terms of D_i, P_i, G_i, and c_i without minimized logic: 2 pts
- None of expressions are derived or All of expressions are incorrect: 0 pts

6. [Total 30 Pts.] **Given:** F(x,w,y,z) = xwy z' + xwy' + xzy

(a) [10 Pts.] Expand the expression into its canonical form, i.e. as a sum of minterms.

(b) [20 Pts.] Use one 3-to-8 decoder and one OR gate to implement the function from the part (a). The OR gate can have any number of inputs. Use block-level diagrams for the decoder shown below (i.e. There is no need to show the details inside). Clearly label all inputs and outputs.

Note: Make sure to assign the enable input of the decoder below.



Solution)

(a) **F(x, w, y, z) = x w z' y + x w (z + z') y' + x (w + w') z y**

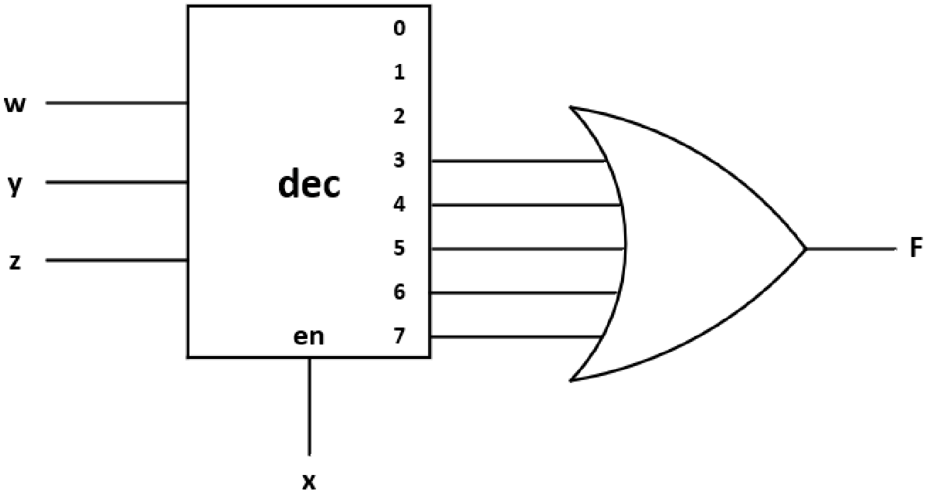
= x w z' y + x w z y' + x w z' y' + x w z y + x w' z y

= ∑(11, 12, 13, 14, 15)

(b) **F(x, w, y, z) = x (w z' y + w z y' + w z' y' + w z y + w' z y)**

➔ **Use w, y, z as input of decoder. Then, use x as enable signal**

➔ **x = 0 → F = 0, x = 1 → F = D3 + D4 + D5 + D6 + D7**



Criteria)

a) Proper explanation : 5pts, Right answer : 5pts

b) If prob 6-(a) is wrong : 0pts, no OR gates : 0pts

7. [Total 10 Pts.] Implement the following Boolean function using only two-input NAND gates. No gate may be used as a NOT gate. The function is in minimum SOP form. All inputs are available both uncomplemented and complemented.

$$f = abc + ac'd'e' + a'd'e + ce + cd$$

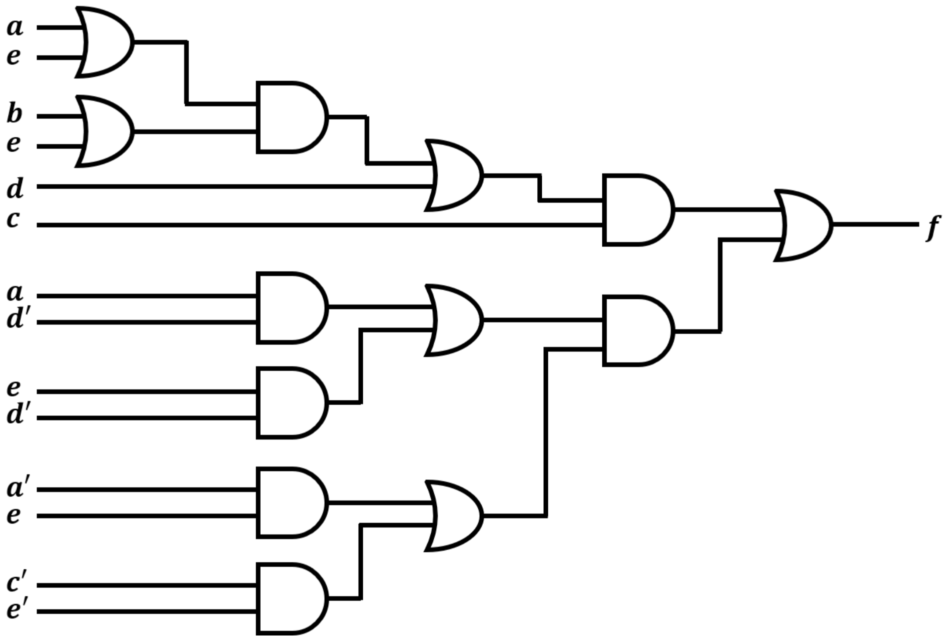
Note: Full credit for 13 gates or less.

Solution)

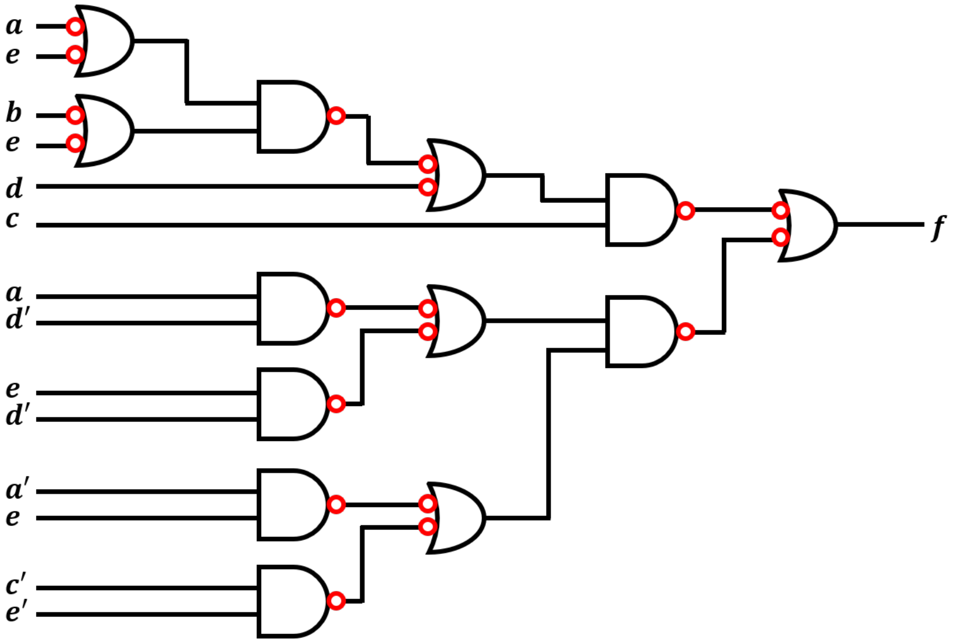
(1) 13 gates implementation (example)

$$\begin{aligned} f &= abc + ac'd'e' + a'd'e + ce + cd \\ &= c(ab + e + d) + d'(ac'e' + a'e) \\ &= c\big((a+e)(b+e) + d\big) + d'(a'e + a)(a'e + c'e') \\ &= c\big((a+e)(b+e) + d\big) + d'(a+e)(a'e + c'e') \\ &= c\big((a+e)(b+e) + d\big) + (ad' + ed')(a'e + c'e') \end{aligned}$$

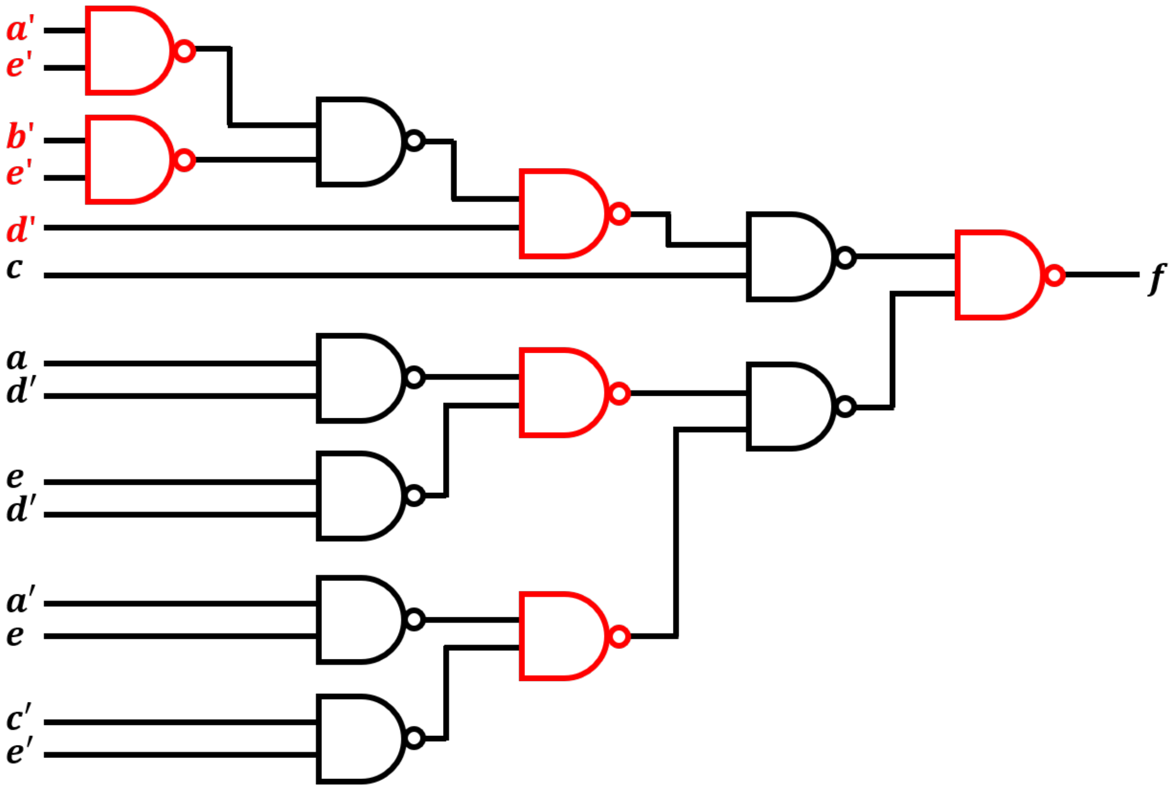
Implement this equation with AND-OR gates:



Bubbles added to convert into NAND gates:



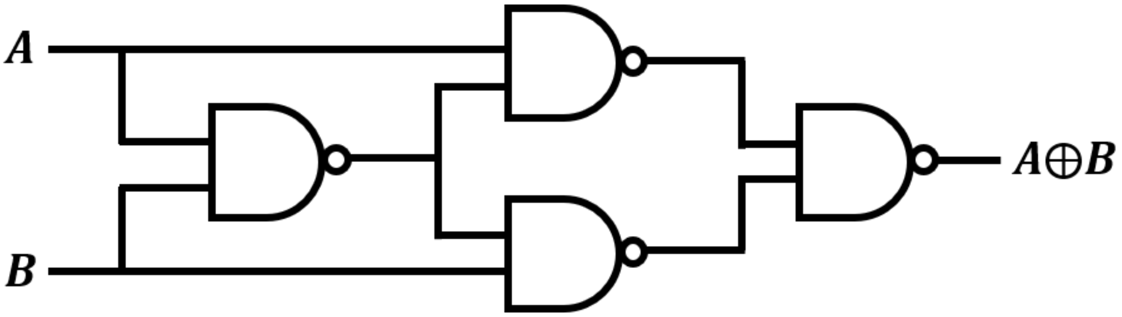
Fully converted into NAND gates:



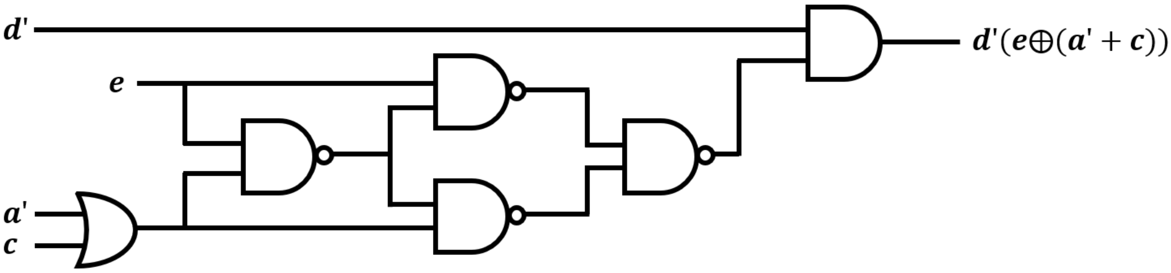
(2) 10 gates implementation (example)

$$
\begin{aligned}
f &= abc + ac'd'e' + a'd'e + ce + cd \\
&= abc + ac'd'e' + a'd'e + cde + cd'e + cd \\
&= abc + ac'd'e' + a'd'e + cd'e + cd \\
&= c(ab + d) + d'(ac'e' + a'e + ce) \\
&= c(ab + d) + d'(ac'e' + e(a' + c)) \\
&= c(ab + d) + d'(e \oplus (a' + c))
\end{aligned}
$$

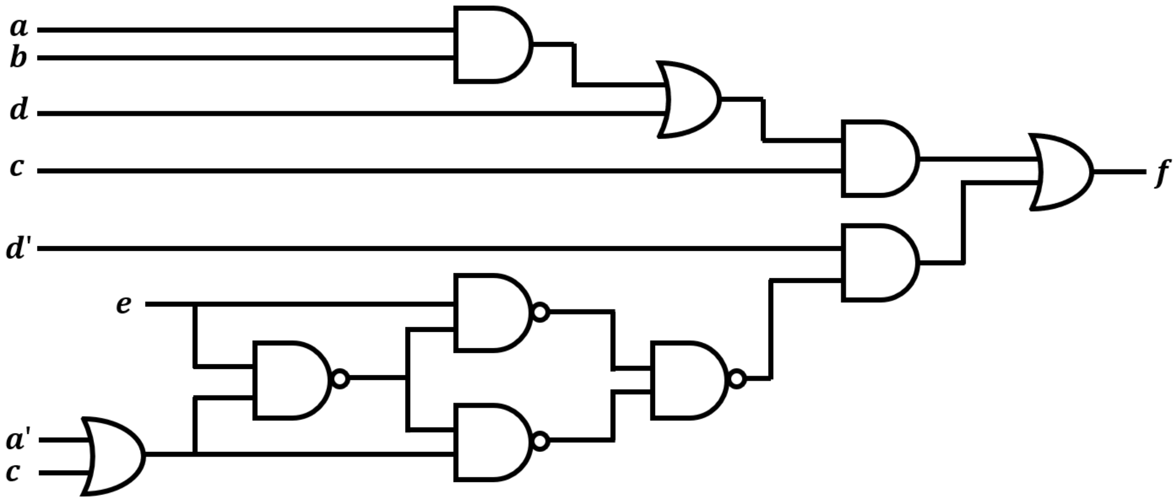
XOR gates can be made with NAND gates as:



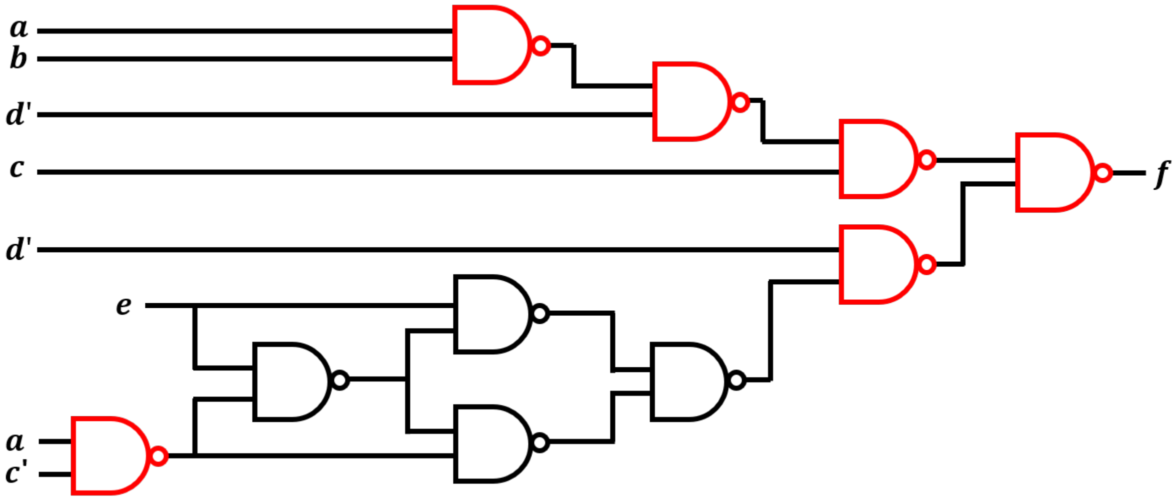
Using this implementation, $d'(e \oplus (a' + c))$ can be expressed as:



Full implementation using AND, OR, NAND gates:

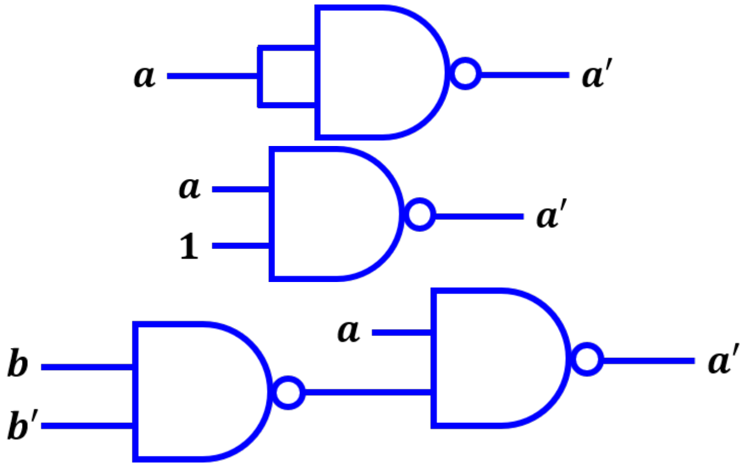


Fully converted into NAND gates:



Criteria)

- Used 13 or less gates : 10pts
- Used more than 13 gates : 5pts
- Used a NOT gate with correct implementation(13 or less gates): 2pts
- Used a NOT gate with correct implementation(More than 13 gates): 1pts
- Used 3 or more input NAND gates : 0pts
- Minor errors : -1pts each



Not gate example(These all three cases are considered as 1 NOT gate in grading)