# A9 - Transistors, Gates, and Circuits

Consider the following Boolean expression `(a . b) + (a . c)`. Show that this is equivalent to the following Boolean expression `a . (b + c)`.

The most straight forward way to show that these expressions are the same is to build a truth table for each expression and show that they are equivalent. You should create one truth table with columns for the three inputs `a, b, c`, the sub-expression `(a . b)` and `(a . c)`, and the expression `(a . b) + (a . c)`.

**Truth table 1**

| Aa (a and b) or (a and c) | a | b | c | (a and b) | (a and c) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

You should then create a second truth table with columns for the three inputs `a, b, c`, the sub-expression `(b + c)`, and the expression `a . (b + c)`.

**Truth table 2**

| Aa a and (b or c) | a | b | c | (b or c) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |

| a and (b or c) | a | b | c | (b or c) |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Imagine that you have designed a circuit that uses `N` expressions of the form `(a . b) + (a . c)`. We replace these expressions with expressions of the form `a . (b + c)`. How many fewer transistors will we use when building this circuit? Briefly explain.

- 2N transistors. Because we get rid of 1 AND (a and c) gate, which has 2 transistors, therefore N expression will save us 2N transistors.

---

Consider the following Boolean expression `(a + b) . (a + c)`.

Provide a simpler expression (fewer gates) that is equivalent. Show that your expression is equivalent by building truth tables for both expressions in the same way as in the first question.

**Truth table 1**

| (a or b) and (a or c) | a | b | c | (a or b) | (a or c) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

## A + (B * C) = (A + B) * (A + C)

**Truth table 2**

| A + (B * C) | a | b | c | A | (B * C) |
|---|---|---|---|---|---|

| A + (B * C) | a | b | c | A | (B * C) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

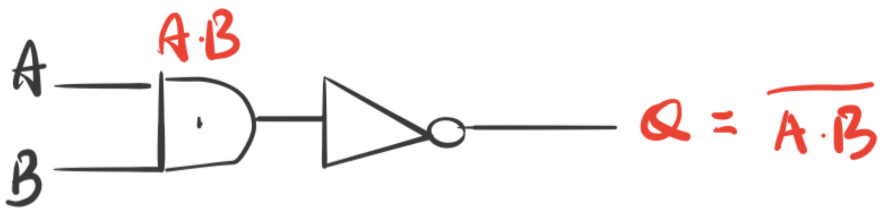
Imagine that you have designed a circuit that uses `N` expressions of the form `(a + b) . (a + c)`. We replace each of these with your solution to the previous question. How many fewer transistors will we use when building this circuit? Briefly explain.

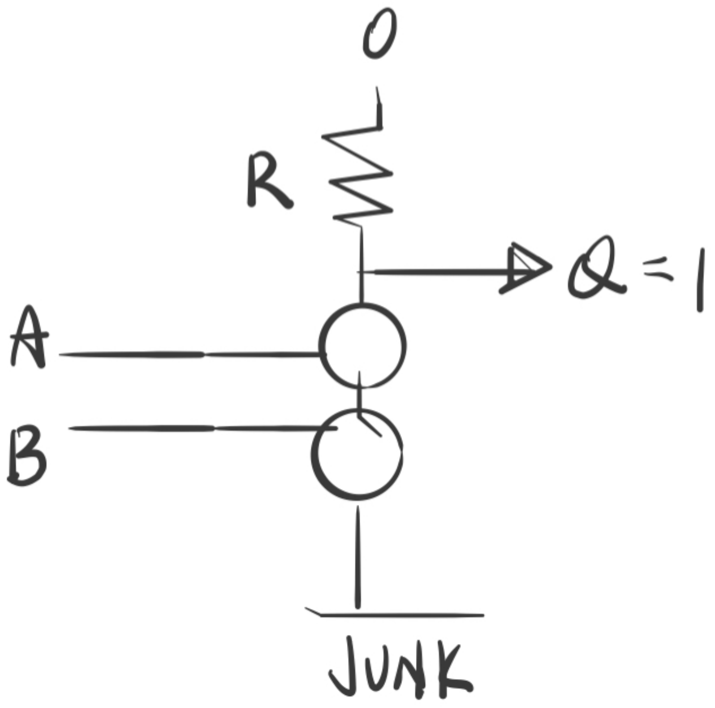
- 2N transistors. Because we get rid of 1 OR gate, which has 2 transistors, therefore N expression will save us 2N transistors.

---

The NAND operator (Not And) has two inputs and one output. The value of the output is 0 if an only if both inputs are 1. This, you can see, is the Not of an And operator. Now, recall that an AND gate and OR gate can each be constructed from two transistors while a NOT gate can be constructed from one transistor and a resistor.

A NAND gate can be constructed from two transistors and one resistor. Draw the configuration of two transistors and resistor that implement a NAND gate.





---

Build a majority-rules circuit. This is a circuit that has three inputs and one output.

The value of its output is 1 if an only if two or more of its inputs are 1; otherwise the output of the circuit is 0. For example, if the three inputs are 0, 1, 1, your circuit should output a 1. If its three inputs are 0, 1, 0, it should output a 0.

You should provide the full truth-table, the sub-expressions, the final Boolean expression, and draw the circuit.

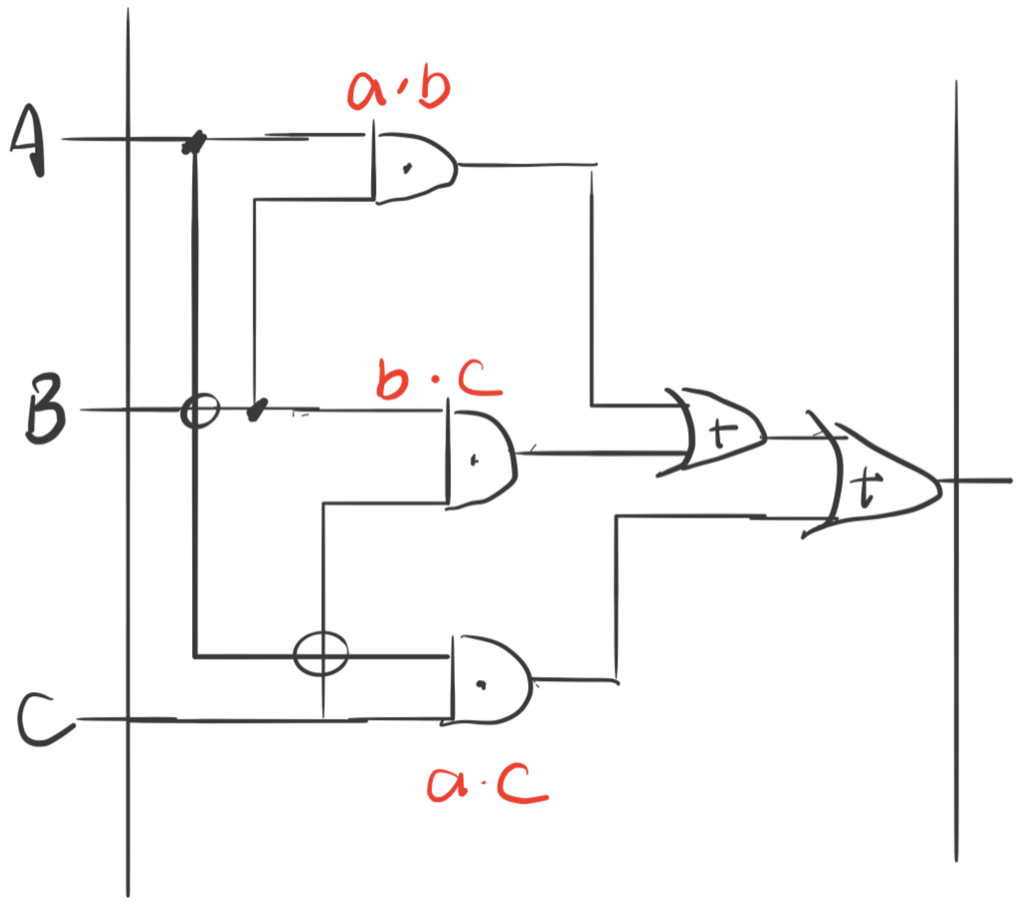
**Truth Table**

| Output | a | b | c | (a * b) | (b * c) | (a * c) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Sub expression: (a * b) + (b * c) + (a * c)

Final Boolean expression: Q = (a * b) + (b * c) + (a * c)

Circuit:



A decoder is a special type of circuit termed a control circuit that is used to route data on a computer. This circuit has `N` inputs and `2N` outputs.

The `N` input values are interpreted as a single (unsigned) binary number representing a value between `0` and `2**N-1`. The decoder's job is to determine the value represented on its `N` input lines and then to send a 1 to one output line corresponding to this value.

For example a 2-4 decoder circuit has 2 input lines and 4 output lines. If the input values are 00, then the first output line will have a value of 1, and the other output lines will have a value of 0. If the input values are 01, then the second output line will have a value of 1, and the other output lines will have a value of 0, etc.

Build a 3-8 decoder that has 3 input lines and 8 output lines. You should provide the full truth-table, the sub-expressions, the final Boolean expression, and draw the circuit.

**3 - 8 decoder Truth Table**

| Input A | Input B | Input C | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Sub - expressions:

D0 = (a~ * b~ * c~)

D1 = (a~ * b~ * c)

D2  = (a~ * b * c~)

D3 = (a~ * b * c)

D4 = (a * b~ * c~)

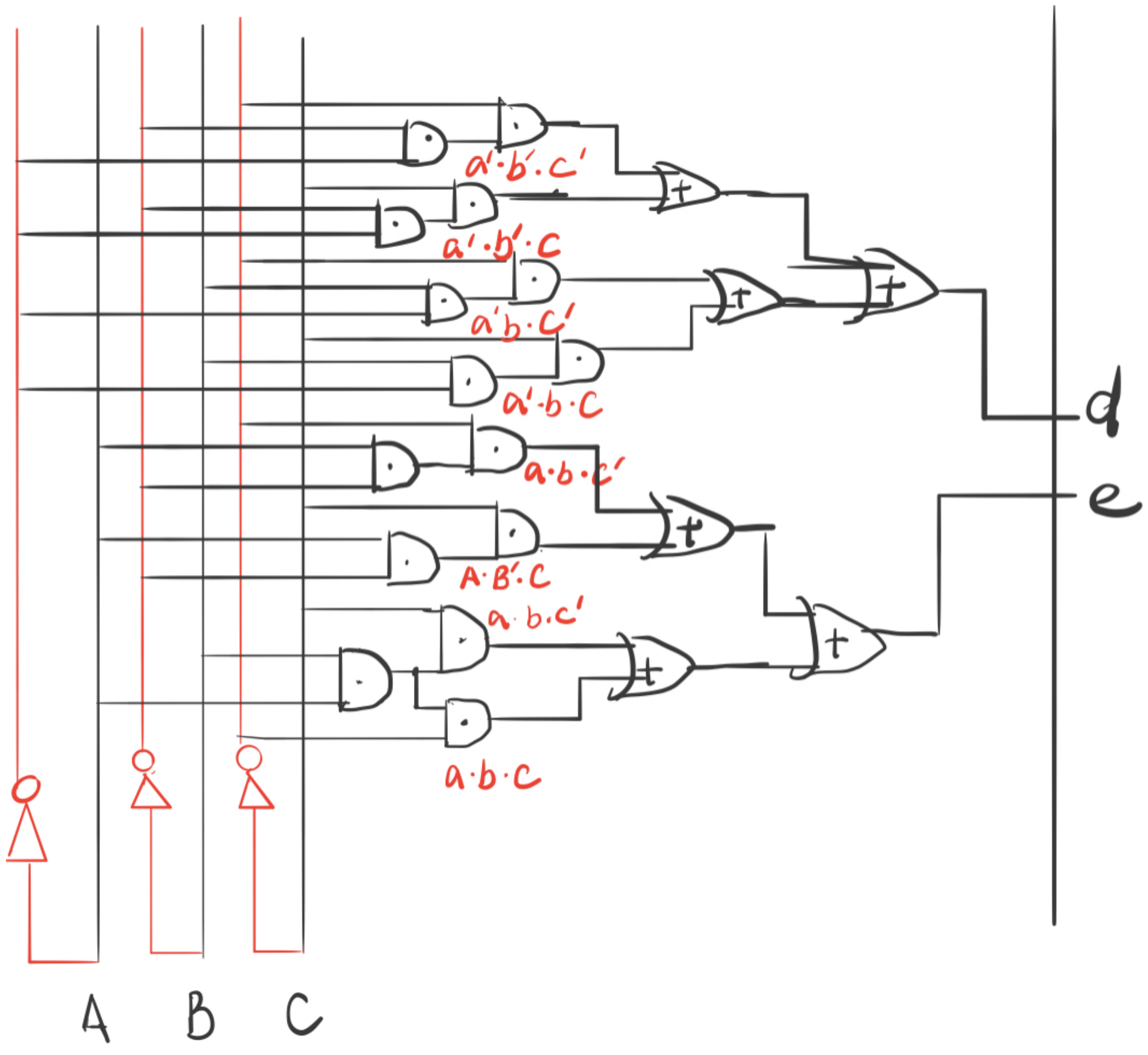
D5 = (a * b~ * c)

D6 = (a * b * c~)

D7 = (a * b * c)

Final Boolean expression: (a * b * c) + (a * b * c~) + (a * b~ * c) + (a * b~ * c~) + (a~ * b * c) + (a~ * b * c~) + (a~ * b~ * c) + (a~ * b~ * c~)

Circuit:


a'·b'·c'
a'·b'·c
a'b·c'
a'·b·C
a·b·c'
A·B'·C
a·b·c'
a·b·c
d
e
A
B
C