Logic Gate Puzzles

A Fun Resource for the Computing Workshop

Answer Key

Introduction

This section contains some logic puzzles that help you cement your understanding of how using gates together can check inputs and provide outputs according to the questions you specify. The challenges start with simple examples and move to more challenging questions. Every puzzle has an answer provided in the answer key.

Each puzzle's objective is to solve the question asked, using as few gates as possible. There are many possible solutions, but the solutions focus on one straightforward answer, and sometimes a more clever answer where it exists. All puzzles have similar rules.

- The solution must only use OR gates, AND gates, and NOT gates
- There must be a single output to check for the result (that is, you can't ask the user to check three separate AND gates to see if AB, AC, or BC are "true")

Some other parameters:

- You can connect signals to more than one gate at a time if you want to
- You can connect inputs to each other if it helps
- You may not have to build a circuit to handle every case

Your first move should be to understand the potential combinations of inputs and how to test them. Don't think about testing all of them at once. Start small. There may not be more than one thing to test, but if you need to, figure out the smallest test that helps you out, and then move on to the next one. You may have to figure out how the results of some tests must be compared: you may need to add AND, NOT, and OR comparisons to some of the results of individual tests.

Puzzle 1

Which circuit will create the following truth table?

Truth Table

A	B	OUT
0	0	0
1	0	1
0	1	1
1	1	1

Answer

This circuit is a single OR gate. On the one hand, there's not much to design, but on the other, it's the first puzzle of the set. This one isn't supposed to be *hard*, but it should put your brain in the way of thinking to tackle the rest of these puzzles.

Puzzle 2

Design a circuit that tests to see if A is TRUE (has an output of 1) and B is FALSE (has an output of 0). You do not need to check to see if A is FALSE and B is TRUE.

Answer

Since there are two *required* conditions, an OR gate won't help us. AND gates check if both inputs are equal to 1, but an AND gate will only be useful if it can check if B is 0. Another way to express the puzzle is "A AND NOT B". Because a NOT gate will convert B from "off" to "on", then the AND gate will work as we want it to.

The solution is the same for the reverse (check for $B = 1$ and $A = 0$), provided the NOT gate converts the A input instead of the B input.

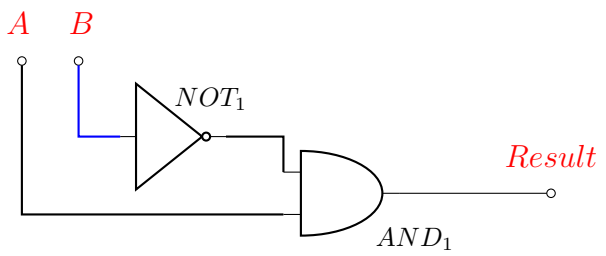

Figure 1: Using discrete gates to test for A AND NOT B.

Puzzle 3

Design a circuit that tests to see if A is *different from* B, no matter what value A has. *Hint: the solution to Puzzle 1 is a huge chunk of the answer.*

Answer

This challenge asks you to make an XOR gate. Use the solution from Puzzle 1, twice, swapping the NOT gate in one of the (A AND NOT B) or (B AND NOT A) blocks. Then take the result from each block and merge the result with an OR gate.

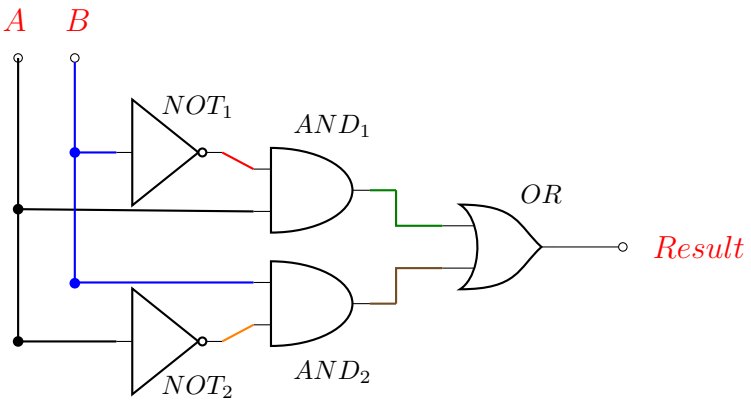

Figure 2: Using discrete gates to make an XOR gate. One pair of NOT and AND gates solves for (A AND NOT B), while the other pair solves for (B AND NOT A). The OR gate returns true if either condition is true.

Puzzle 4

Say you had three inputs A, B, and C, and you want to know when a majority of them are ON or TRUE. The question is simple enough: with three inputs, use connected logic gates in a way that can tell when the majority of inputs (here, that's at least two of the three) have a 1 as their input value.

So a line over the letter indicates "NOT true", and no line indicates "true".

Setting Up

Start by taking a minute to think of the way to test if some pair of inputs are both 1. We will need to test each possible pair of inputs. That may seem like a big deal, but we can break it down into small chunks.

There's a way to represent these conditions in a compact way:

- If all of the inputs are off, then the condition can be represented by: $\overline{ABC}$.
- When all inputs are on, the shorthand would be ABC.
- When only A is off, the condition is represented by $\overline{A}BC$.

We must first find any combination of the three signals that has two or more of them reporting "yes".

How many combinations are there in total?

1. Each input can be 1 or 0 – there are exactly two possible states
2. There are three inputs in total, and each can be on or off
3. Each input is not affected by any other input (the inputs are not tied together in any way)

Now we need a truth table. The truth table will have $2 \times 2 \times 2$ (that is, 2^3) possible combinations. So let's make a table by counting from 0 to 7 in binary. We will assign one of the inputs to each column (2^0, 2^1, and 2^2).

A	B	C	Symbol
0	0	0	$\overline{A}\overline{B}\overline{C}$
0	0	1	$\overline{A}\overline{B}C$
0	1	0	$\overline{A}B\overline{C}$
0	1	1	$\overline{A}BC$
1	0	0	$A\overline{B}\overline{C}$
1	0	1	$A\overline{B}C$
1	1	0	$AB\overline{C}$
1	1	1	ABC

To cover every possible case, half of of each column will be "true" and half will be "false. For the left-most column ("A"), the first four column values are "off" and the next four are "on". The right-most column alternates "on-off" four times, and the middle column alternates off and on every other row.

The table shows us all possible conditions for these three inputs. Now add a results column, showing the answer our logic circuit should produce. Remember, if any two inputs in the row are true, then the result column will be true. Otherwise, the result is false.

A	B	C	Symbol	Result
0	0	0	$\overline{A}\overline{B}\overline{C}$	0
0	0	1	$\overline{A}\overline{B}C$	0
0	1	0	$\overline{A}B\overline{C}$	0
0	1	1	$\overline{A}BC$	1
1	0	0	$A\overline{B}\overline{C}$	0
1	0	1	$A\overline{B}C$	1
1	1	0	$AB\overline{C}$	1
1	1	1	ABC	1

The fourth, sixth, seventh, and eighth rows all have two or more inputs set to "on". The results ("R") column is 1/"true" for these four rows, and 0/"false" for the rest.

The logic can be expressed like this:

(A AND B) OR (A AND C) OR (B AND C) OR (A AND B AND C).

So if any of these individual conditions is true, the result is true. Note that all three values do not have to be true for the result to be true. Any two true inputs gives the same result as all three inputs being on. So there is no need to specifically test for all three inputs being true.

Answer 1

How can we build a logic circuit to produce this result? Let's start with the example we started above. If we use an AND gate for each of the three combinations of inputs that could produce a result of "true", we can then put each of the results into an OR gate, because the output of any AND gate tells us that we should produce a result of "yes". We have three outputs, and to this point we haven't discovered that there are three-input gates (heh) so we must use two OR gates to capture all three possible "majority yes" voting conditions.

Follow the logic and see if you can convince yourself that the logic is correct.

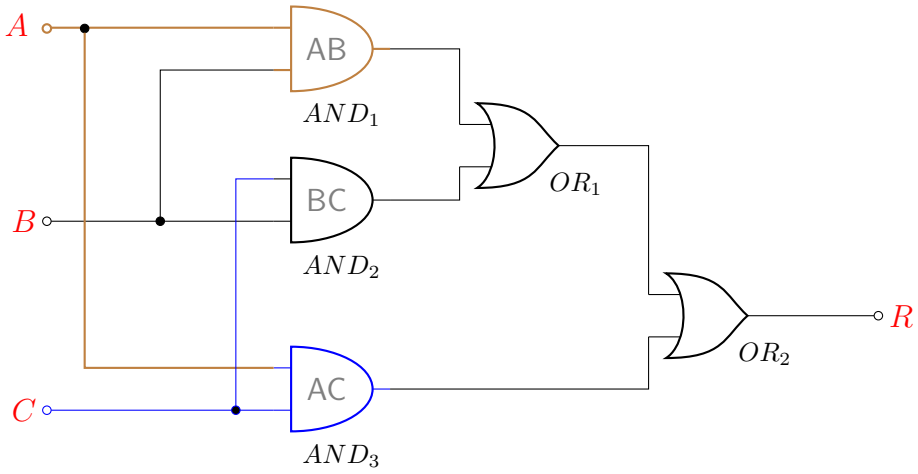

Figure 3: A majority detection circuit using three AND gates and two OR gates. Testing specifically for all three inputs being true can be ignored because the circuit only needs to check for a majority of the inputs.

Answer 2

A majority of the inputs would be 2 out of 3 or 3 out of 3. That is, the sum of the all three values must be 2 or 3. We could also use a full adder circuit and check the output of the carry bit to see if the resulting sum is at least 2. This answer is appealing because the full adder is such a fundamental building block of electronic logic, but it uses a comparatively large number of gates, and so it isn't a great answer to the question.

Answer 3

But might there be a more efficient way to do this? That is, could we get by with fewer than three AND and two OR gates?*

We can be clever and produce a slightly less straightforward-looking circuit that still accomplishes our goal, but uses fewer resources to get the correct answer. The goal is the same – figure out if any two of the inputs are "yes".

1. If (B AND C) is true, A does not matter – the result is "true"
2. If A is "true", either B OR C can be "true" to get a "true" result.

By connecting inputs B and C to both an OR gate and an AND gate, we can capture these conditions more efficiently. Let's write out the logic:

1. If (B AND C) is true, A does not matter—the result is "true" regardless of the value of A (this implies that we can use an OR gate to give a result of true if (B AND C) is true.
2. If (B OR C) is true, AND A is true, the result is also "true"

So now it should be clear that by connecting B and C to both an AND gate and an OR gate, we can capture condition (1) with an AND gate; and condition (2) with an OR gate that feeds into an input of an AND gate, with the other input tied to A. Each result connects to a second OR gate that gives the result value. In this way, we can solve the question with two AND gates and two OR gates, reducing our need for one AND gate! Not the most exciting game, true, but it's the same sort of puzzle as Sudoku.

*Would I even ask this question if the answer was "no"?

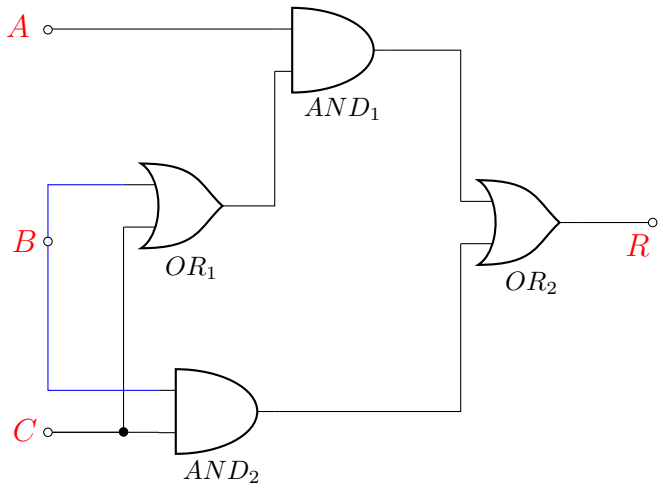

Figure 4: A majority detection circuit using only two AND gates and two OR gates.