CMSC 250: Digital Circuits

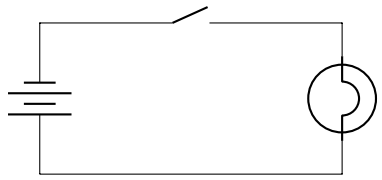
Justin Wyss-Gallifent

February 14, 2023

1 Digital Circuits and Gates 2
2 Circuit Diagrams 4
3 Input-Output Tables for Circuit Diagrams 5
4 Boolean Expressions for Circuit Diagrams 7
5 Circuit Diagrams for Boolean Expressions 7
6 Boolean Expression for a Given Input-Output Table 9
7 Equivalent Circuits 10

1 Digital Circuits and Gates

Consider the following digital circuit which represents a battery (on the left) connected to a bulb (on the right) with a switch (at the top).



It's obvious that when the switch is closed, the bulb is lit.

Now consider the following two circuits:

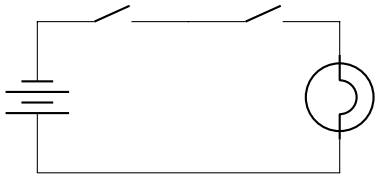


Figure 1: Series

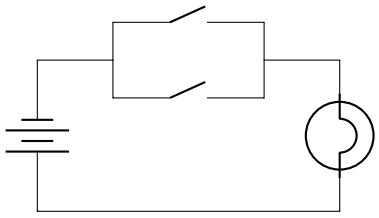


Figure 2: Parallel

In the first figure the bulb will be lit only when both swiches are closed (think AND) whereas in the second figure the bulb will be lit if either or both of the switches are closed (think OR).

It's obvious that we could combine these. For example in the following picture there are three switches and the bulb will be lit if either both the top switches is closed or the bottom switch is closed, or both of these are true:

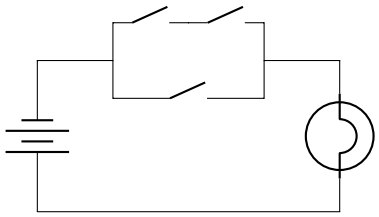


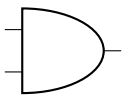
Figure 3: Combination

It's obvious that these diagrams could get really messy very quickly so what we'll do is strip away the unnecessary details and focus on the control mechanisms in a more abstract way.

Instead of saying "open" we'll use the value 0 and instead of saying "closed" we'll use the value 1. For purposes of AND, OR, and NOT, treat 0 as False and 1 as True.

In addition we can think of 0 as "off" (no power) and 1 as "on" (power!).

Definition 1.0.1. An *AND gate* is a control mechanism in which there are two inputs (each either 0 or 1) and one output (either 0 or 1). The output will be 1 iff both inputs are 1. We draw this as follows:

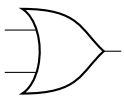


In this sense:

$$\text{output} = \text{input1} \wedge \text{input2}$$

□

Definition 1.0.2. An *OR gate* is a control mechanism in which there are two inputs (each either 0 or 1) and one output (either 0 or 1). The output will be 1 iff one or both inputs are 1. We draw this as follows:



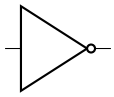
In this sense:

$$\text{output} = \text{input1} \vee \text{input2}$$

□

To round things out:

Definition 1.0.3. A *NOT gate* is a control mechanism in which there is one input (either 0 or 1) and one output (either 0 or 1). The output will be 1 iff the input is 0. We draw this as follows:



In this sense:

$$\text{output} = \sim\text{input1}$$

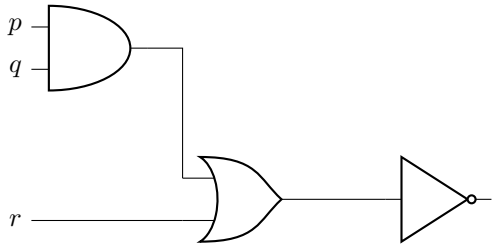
□

Note 1.0.1. Actually constructing not gates in circuits is a little more challenging than just using switches and we won't cover it here.

□

2 Circuit Diagrams

Now for the fun part. We can start creating new circuit diagrams by putting these together.



Example 2.1. Observe, for example, that if $p = 1$, $q = 1$, and $r = 0$ then the AND gate outputs $1 \wedge 1 = 1$ which feeds into the OR gate along with $r = 0$ which then outputs $1 \vee 0 = 1$ which then feeds into the NOT gate which then outputs $\sim 1 = 0$. So if all three input switches are closed then the ouput switch is open.

□

There are certain rules for these diagrams, they are the following:

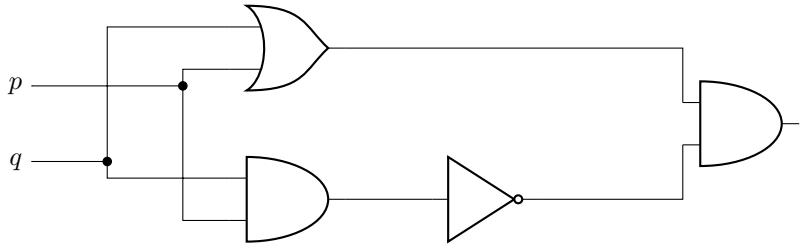
1. Never combine two input wires.
2. A single input wire can be split partway and used as input for two separate gates.
3. An output wire can be used as input.

4. No output of a gate can eventually feed back into that gate.

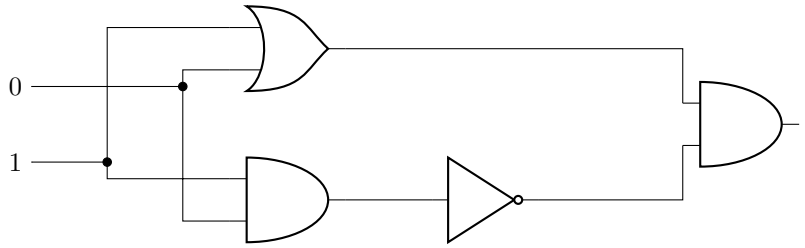
3 Input-Output Tables for Circuit Diagrams

In order to see how these circuit diagrams function we can draw output tables. To draw an output table we trace each possible combination of inputs through the circuit and create a table which shows the results.

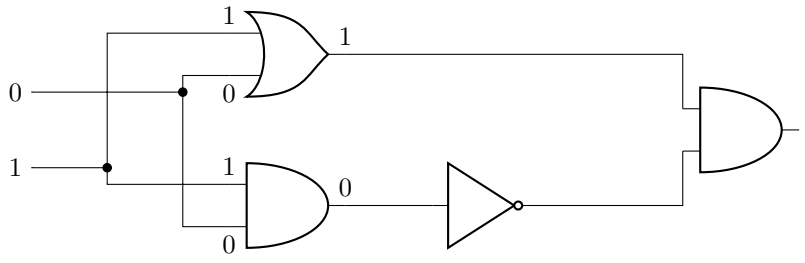
Consider this example. In the following when two wires meet at a solid point then the wire is meeting/splitting whereas if there is no solid point they are not meeting.



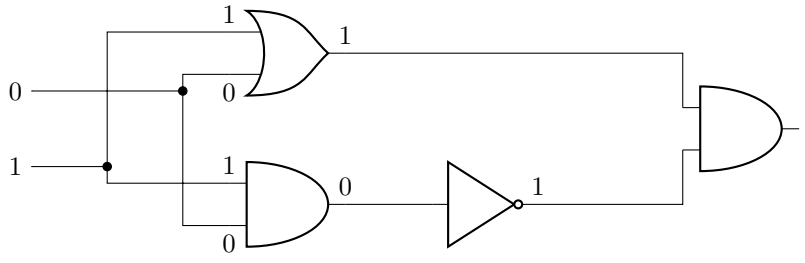
Example 3.1. Let's look at the input $p = 1$ and $q = 0$. We can follow them through the logic gates step by step.



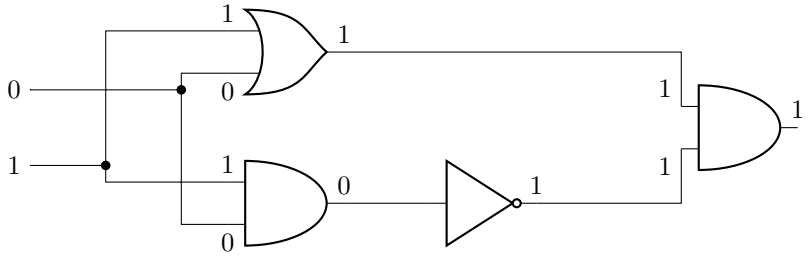
First we follow the values through their first gates and label the outputs accordingly:



Then we follow the value from our leftmost AND gate and label the output accordingly:



Then we follow the values from the OR gate and the NOT gate into the rightmost AND gate and label the output accordingly:



If we try this with the other three possibilities and put them together in a table we get the following:

p	q	Output
0	0	0
0	1	1
1	0	1
1	1	0

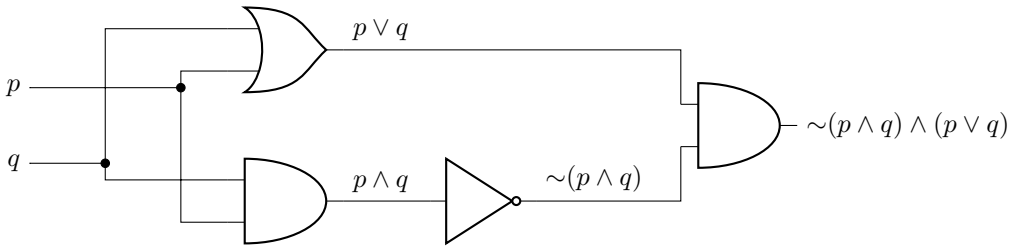
□

4 Boolean Expressions for Circuit Diagrams

Instead of following specific values through we could also trace variables through, building the result as we go.

Here is the above example. We have filled in as with before but now with variables:

Example 4.1. Consider:



Thus we see the boolean expression is:

$$\sim(p \wedge q) \wedge (p \vee q)$$

Note that this is the same as XOR, which we also see in the table above.

□

5 Circuit Diagrams for Boolean Expressions

Let's now reverse the situation. Suppose we are given a boolean expression and we wish to design a circuit which represents this.

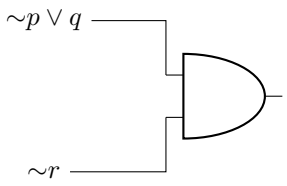
We start filling in the diagram from the right side by looking at the last operator which was applied. We proceed recursively with the subsequent operators, working our way left until we hook up the input variables. If input variables are repeated we just make sure we merge them on the left.

This is a bit confusing to write down and is much more clear with an example.

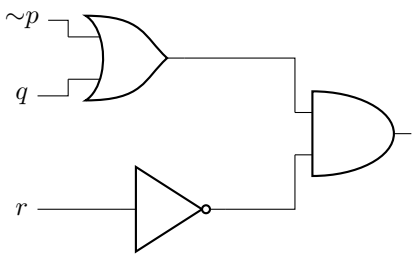
Example 5.1. Let's draw a circuit diagram for the boolean expression:

$$(\sim p \vee q) \wedge \sim r$$

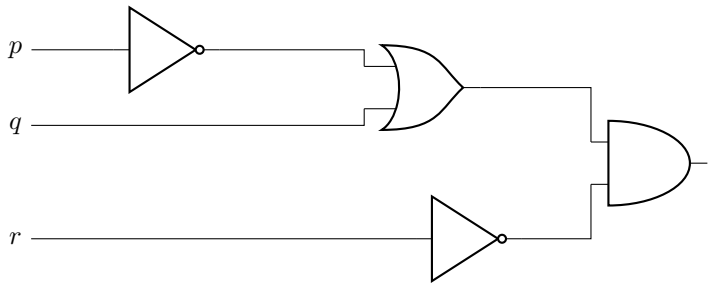
The last operator we applied was $\wedge$ so we put that on the right. The inputs are $(\sim p \vee q)$ and $\sim r$ so we can put those as inputs temporarily:



Then we do the same, recursively, for $\sim p \vee q$ and $\sim r$:



Lastly with the $\sim p$:



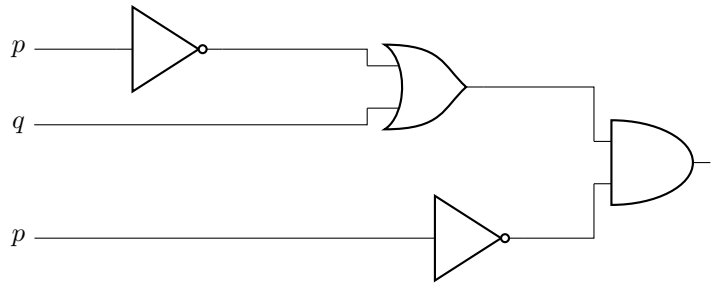
Since the input variables are all different there is no merging to do.

□

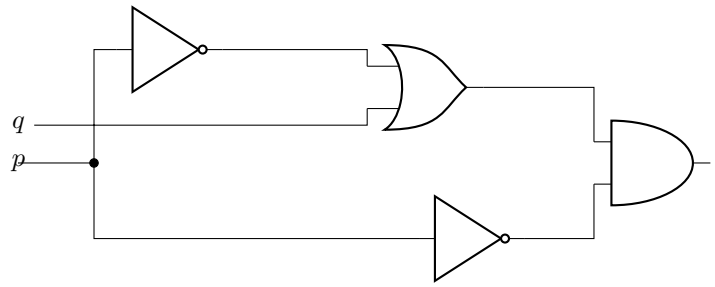
Example 5.2. Suppose the previous question had been:

$$(\sim p \vee q) \wedge \sim p$$

We would have reached:



Now we notice that there are two p on the left. Thus we merge them to become one input:



□

6 Boolean Expression for a Given Input-Output Table

Now for a more challenging and more realistic situation. Suppose we are given an input-output table and wish to design a circuit which represents the table.

Example 6.1. For example can we design a circuit which produces this:

p	q	r	Output
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

□

The systematic way we can find an appropriate boolean expression (and a circuit if we need it) is to follow these steps:

1. For each row which contains a 1 in the output column construct a $\wedge$-only statement form which is true only under the conditions of the row.
2. Then simply $\vee$ these together.

Example 6.2. In the example above the first row has $p = 0$, $q = 0$, and $r = 0$. We construct the statement form $\sim p \wedge \sim q \wedge \sim r$. Note this statement will then be true only for this row.

Likewise for the second, fourth and eighth rows we construct $\sim p \wedge \sim q \wedge r$, $\sim p \wedge q \wedge r$, and $p \wedge q \wedge r$ respectively.

The final boolean expression is then:

$$(\sim p \wedge \sim q \wedge \sim r) \vee (\sim p \wedge \sim q \wedge r) \vee (\sim p \wedge q \wedge r) \vee (p \wedge q \wedge r)$$

□

Note that we could go from here directly to a circuit but it would be fairly complex. In many cases it can be easier to first simplify the boolean expression first and then build the circuit.

7 Equivalent Circuits

Definition 7.0.1. Two circuits are said to be *equivalent* if they have boolean expressions which are logically equivalent.