

# Diving into Digital Electronics

Aria Alinejad

February 12, 2024

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Abstraction Levels . . . . .	2
1.2	The transistor analogy . . . . .	3
<b>2</b>	<b>Logic gates</b>	<b>4</b>
<b>3</b>	<b>Digital adder</b>	<b>8</b>

# 1 Introduction

The difference between analog and digital electronics is not as big as one might imagine. In some sense the main difference is just what abstraction level you are working on. In this short text I will introduce the main concepts of digital electronics and show how they can be implemented to build a logical circuit.

Computers are quite mysterious and you might be wondering how one can go from the currents and voltages (the electronics you know) to building something that can solve problems. But I promise that by the end of this you will be able to make a logical circuit that can do simple calculations (more specifically an adding circuit). I will also give some insight into how the key concepts can be taken further to develop a computer.

## 1.1 Abstraction Levels

A key concept in digital electronics and engineering in general is abstraction levels. You might not know it but you are already familiar with this concept through real life. There are layers to real life just as in electronics.

### Layers of abstraction

To pass your masters you need to:

pass all of your 5 years,  
to pass a year you need to pass all your courses,  
to pass your courses you need to pass x number of tests,  
to pass your test you need to get x% right on the questions.

Circuit equivalent:

Block level,  
Logic gate level,  
Transistor level,  
Voltages level.

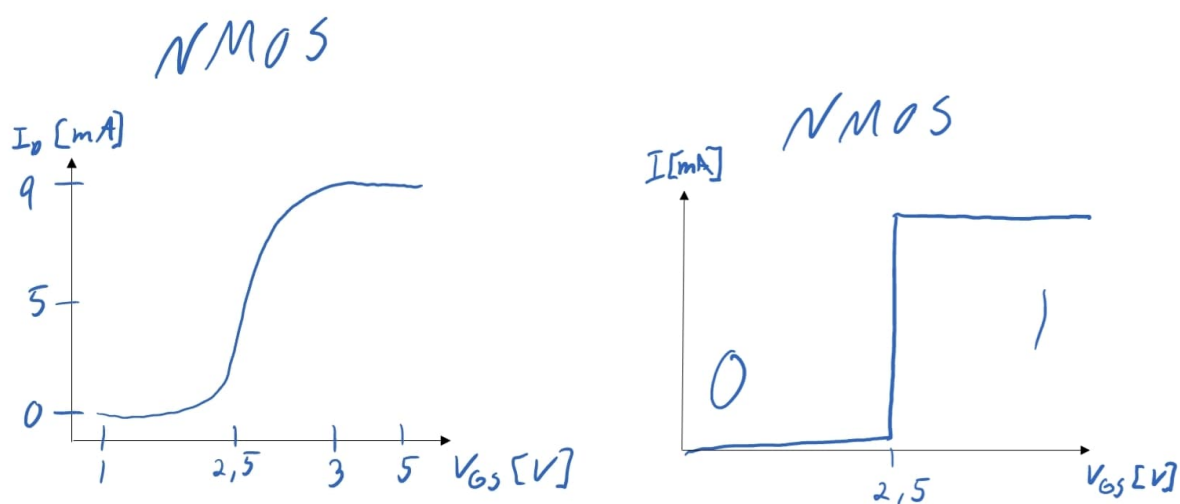
These are abstraction levels. When trying to find out if you passed your masters or not you can't sit down and think about every question you have answered on every test. Similarly, we can't start calculating the voltage in each wire to find out if the computer will work correctly or not. When deeming your test passed or failed (1 or 0) the teacher only cares if your score is above or below 40%. In the same way the transistor only cares if your voltage is above or below a certain level (high or low voltage) when it decides to pass the current or not.

When you want to know the outcome of a course you only look at what tests are passed and failed. In the same way, when looking at the outcome of digital circuit, with transistors, we only look at where there are low or high voltages, 0's or 1's. Thus, the details are abstracted away.

You are probably already familiar with the transistor and how it works in a circuit. The relationship between the current and the voltage in the NMOS transistor is shown in figure 1a.<sup>1</sup> If we simplify this we can look at it as something that passes all the current, or stops all the current based on the voltage, see figure 1b.

---

<sup>1</sup>You might think that the transistor is quite magical in comparison to a resistor, but what defines a resistor is that it has a linear relationship between current and voltage. What defines a transistor is nothing more than a relationship between current and voltage as the one shown in the figure. This relationship between current and voltage just happens to be very practical for many purposes.



(a) Continuous plot.

(b) Quantized plot.

Figure 1: Voltage vs. current plot of an arbitrary NMOS transistor.

## 1.2 The transistor analogy

The year is 1895. Lets now think of the electronics circuit as the roads in the city, see figure 2. As of now we only have resistors, that is speed bumps. But this can only control the speed of the cars. We want to control the flow of traffic, for this we need logic. In the world of cars we need police officers.

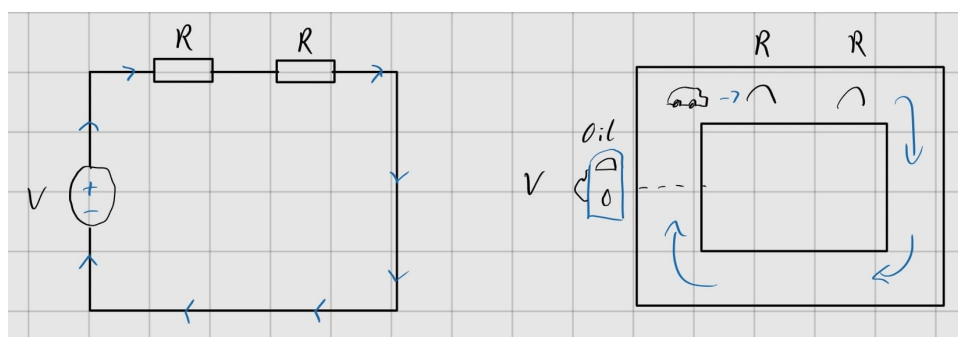


Figure 2: Depiction of a simple circuit using a road as an analogy.

Think of the NMOS transistor as a road that is controlled by a *Nice* cop, see figure 3. If there is a cop present (high voltage, **1**) you can pass. But if there is no cop (low voltage, **0**) all traffic is stopped. We call this the NMOS road.

At the PMOS road there is a *Problematic* cop, see figure 4. If this cop is present (**1**) no one can pass, so the traffic only flows if there is no cop (**0**) present.

The cops can only get to their post if they have fuel (that is voltage), but at their post they can't move, they have no current. Thus it is the cops voltage and not their current that determines if you can go or not. Furthermore the cars on the main road naturally need fuel, voltage, to be able to drive, current. No voltage means no current, but no current doesn't mean no voltage (think fuel and driving ;).

**What is the take away:**

- We need transistors to make logic, to control the flow.
- The NMOS road passes current when there is high voltage at the gate(1).
- The PMOS road passes current when there is low voltage at the gate(0).
- There is no flow of current between the gate and the other channels.
- It is the Voltage, not the current at the gate that determines the flow.
- No Voltage  $\rightarrow$  No Current, No Current  $\nrightarrow$  No Voltage.

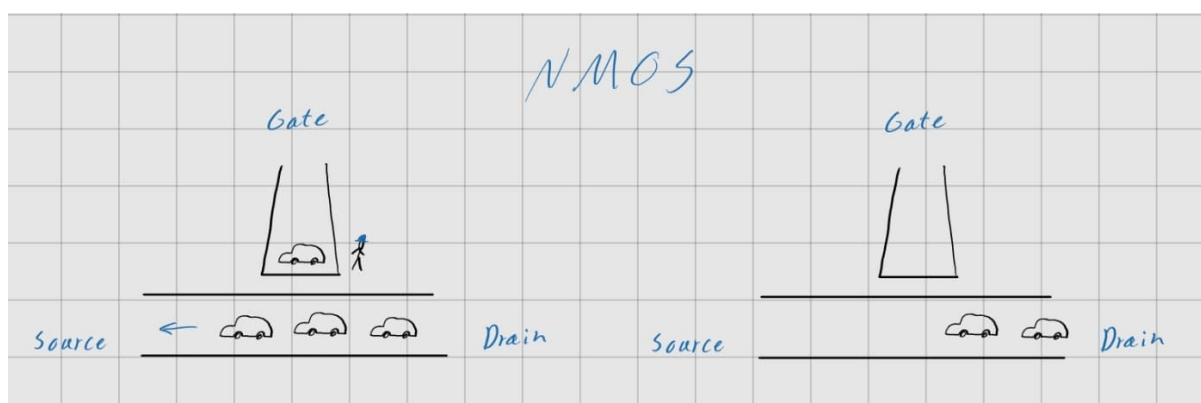


Figure 3: NMOS transistor depicted using road analogy.

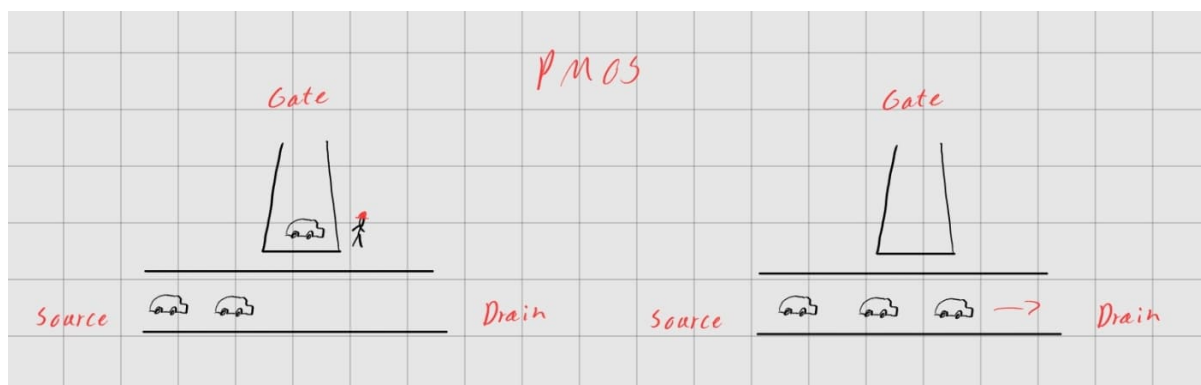


Figure 4: PMOS transistor depicted using road analogy.

## 2 Logic gates

Now we have this neat transistor that we can play around with. We have a way to control the flow of current, but how do we make a logic? Well logic is about input and output. The language of digital circuits is voltage,<sup>2</sup> 0's and 1's. Thus, whatever the circuit, we want something that connects the output to high or low voltage depending upon the inputs (if they are connected to high or low voltages). 0's and 1's out based on the 0's and 1's in.

<sup>2</sup>Why is it so? Well it is the voltage at the gate of the transistor that determines the flow of current. Transistors control the flow and make the logic. Thus what matters in a logical circuit is the high and low voltages, 0's and 1's.

Lets start simple with an inverter (NOT-gate). It does exactly what it sounds like, if it takes inn **0** then it gives out **1**, and vise versa. Lets break it down, **0**  $\rightarrow$  1, and **1**  $\rightarrow$  **0**, see table 1. We start with what we know has to be there, an input and an output:

Table 1: Truth table for NOT-gate.

In	Out
0	1
1	0

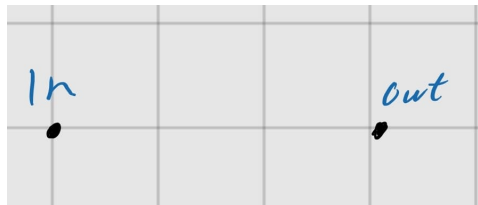


Figure 5: NOT-gate step 1.

We want *out* to be connected to low voltage (**0**) when the input is high voltage (**1**). So we add a NMOS transistor (the component that allows flow on high input):

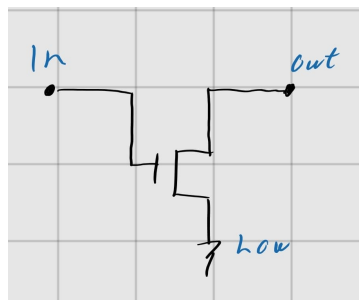


Figure 6: NOT-gate step 2.

Furthermore we want *out* to be connected to high voltage (**1**) when the input is low voltage (**0**). So we use a PMOS transistor (the component that allows flow on low input):

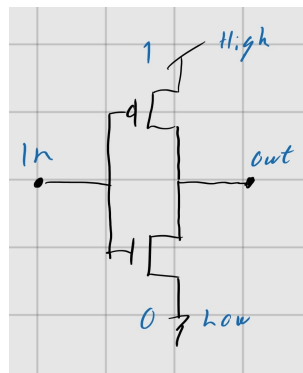


Figure 7: Finished NOT-gate.

Here it is important to notice that we need to connect to a low voltage to get a low voltage out. Connecting the output to nothing does not give it low voltage, but rather leaves it unchanged. If we don't connect anything it has whatever voltage it previously had, which might be unknown.

### What is 0 volt?

Think of the voltage as the height of a mountain. The height is calculated as meters above a given starting point. This starting point is defined by us, thus it can be sea level or maybe the foot of the mountain. What could be 0 meters, could just as easily be 200 metres based on what we define as our zero. Voltage is just like this, zero has no other significance than that we say that it is zero. Physically 0V and 5V can be the exact same, just with different definitions of zero, just like the height of a mountain. It is natural that we need to connect a point to 5V if we want it to have 5V, thus it should be just as intuitive that we need to connect it to 0V if we want it to have that.

- Voltage is about difference in energy between two points, just like height is about difference in position.

Next we can try to make something that is less straightforward. We want something that connects the output to **1** only if both inputs, *A* and *B*, are connected to **0** (NAND-gate). Lets break it down, **00** → **1**, **01** → **1**, **10** → **1** and **11** → **0**. We can write this in a table to make it more readable, see table 2. We again start with what we know that we need to have, input and output:

Table 2: Truth table for NAND-gate.

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0



Figure 8: NAND-gate step 1.

Furthermore, want *out* to be connected to low voltage when both inputs are high. Connecting when the input is high is the job of the NMOS transistor. If we think of the policemen again, we want to make a road that only connects when both *nice* policemen are present. Thus, two NMOS transistors in series will do the job:

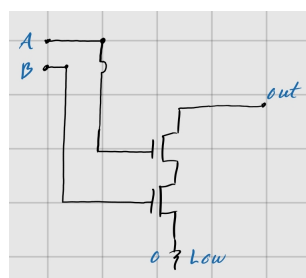


Figure 9: NAND-gate step 2.

Furthermore, want the *out* to be connected to high if any of the inputs are low. Since we want a connection on low input we need to use a PMOS (*Problematic cop*). And we want to be able to pass even if one road is blocked. Thus having the option of choosing several roads does the job. We connect the PMOS transistors in parallel:

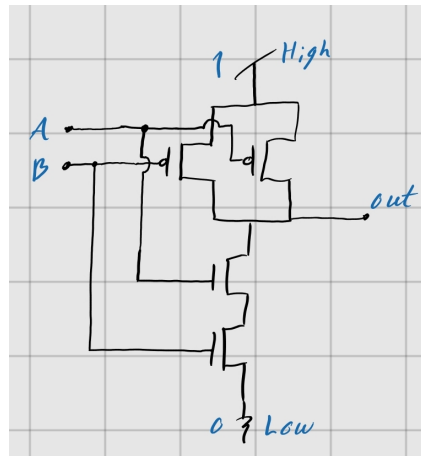


Figure 10: Finished NAND-gate.

This is now a gate that only gives us **0** if both inputs are **1**, that is what I call logic right there! The NAND-gate and the inverter are essential building blocks in making logic. When we make bigger logical circuits with many different gates it is common to abstract away the details of how the transistors are placed. Figure 11 and 12 show how the NOT and NAND gates can be drawn on gate level. In digital design there is a limited number of building blocks, just like LEGO's. Figure 13 shows all of the gates with their truth tables that describe how they act, see if you can make them out of transistors with the same technique as you used to draw the NOT and the NAND-gates.<sup>3</sup>

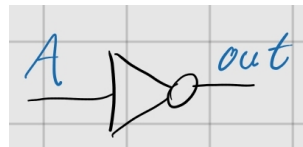


Figure 11: Gate level drawing of NOT-gate.

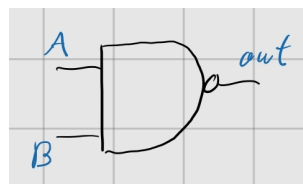


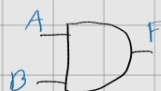
Figure 12: Gate level drawing of NAND-gate.

<sup>3</sup>In fact the AND-gate can be made by simply putting a NOT-gate after the NAND-gate, look at the truth tables and see if this makes sense.

AND

$$A \cdot B = F$$

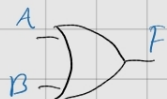
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



OR

$$A + B = F$$

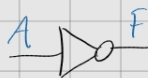
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



NOT

$$\overline{A} = F$$

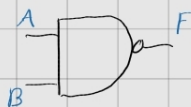
A	F
0	1
1	0



NAND

$$A \cdot B = F$$

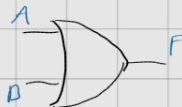
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



NOR

$$A + B = F$$

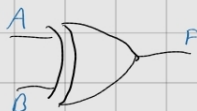
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



XOR

$$A \oplus B = F$$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



To help remember the gates and why they are useful:

- **AND:**  
When does it give 1?  
When both this AND that are 1.
- **OR:**  
When does it give 1?  
When this OR that is 1.
- **NOT:**  
When does it give 1?  
When this is NOT 1.
- **NAND:**  
When does it give 1? When AND does NOT.
- **NOR:**  
When does it give 1?  
When OR does NOT.
- **XOR:**  
When does it give 1?  
When there are an odd number of 1's.

Figure 13: Name, equation, truth table, and symbol of all logical gates.

### 3 Digital adder

Computers are all about automating tasks, and some of the first computers were used to automate calculations (calculators).<sup>4</sup> The adder is an immensely important component in most all digital electronics, and by utilizing components you know of, you will now get the chance to design a circuit that uses high and low voltages to calculate the sum of two numbers! This is the first obvious step towards making computers that can solve problems for us.

We will try to design something that is called an half-adder. To start with we clarify to our self what we want the system to do. Given two binary numbers  $A$  and  $B$  we want the circuit to output the sum.

$$A + B = out \quad (1)$$

Here the "+" symbol indicates *addition* and not logical *OR*. Since we are working in binary

<sup>4</sup>In fact, the word computer used to be the job title of a person that worked with doing a lot of calculations by hand!



the outputs should be:  $0 + 0 = 00$ ,  $1 + 0 = 01$ ,  $0 + 1 = 01$ ,  $1 + 1 = 10$ . We can again write this in a table to make it more readable, see table 3.

Table 3: Truth table for 2-bit adder.

A	B	out <sub>2</sub>	out <sub>1</sub>
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Since we have two outputs we now need two logical ports. Looking at out<sub>2</sub> we can notice that it's truth table values are the same as an AND-gate. While out<sub>1</sub> has the same values as an XOR-gate. Meaning that we can realize this 2-bit adder using an XOR-port and an AND-gate. Figure 14 shows how this can be drawn using the symbols for the gates (as shown in figure 13).

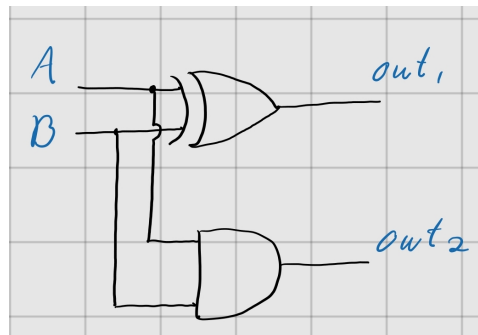


Figure 14: Realization of 2-bit adder using logical gates.

Just as these symbols abstract away the details of how the transistors are places, we can abstract even further by only drawing a box of the inputs and outputs of this 2-bit adder. We call this level of abstraction block level, see figure 15.

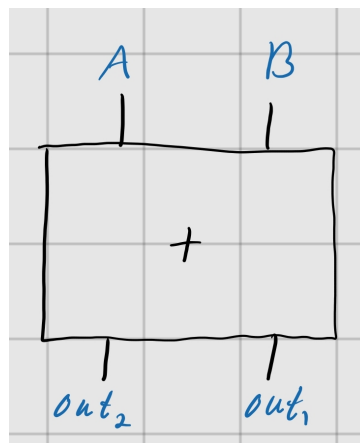


Figure 15: Block level figure of 2-bit adder.

We have now successfully designed a digital circuit that can perform addition of two numbers! The design of this can be extended, using the same logic, to make a block that can add binary numbers of arbitrary length, which is the start of the design of a calculator, or a simple computer if you will. And in the end it's all just high and low voltages. Just like a magic trick, what makes digital electronics seem impossible is simply abstracting away the details.<sup>5</sup>

---

<sup>5</sup>Many people often wonder what the real difference between engineers and magicians are... but contrary to magicians our tricks still remain magical even when we give away the secrets.