Laboratory of Physics III

Experience 7: Digital Electronics

University of Trento, Physics Department

Authors: Matteo Zortea, Elena Acinapura
Date: December 2020
Group: A05
Experience number: 7

Contents

Contents

Introduction

Digital electronics is a wide field that consists in the study and processing of digital signals, which are signals that assume only discrete values in time.
Digital signals have many advantages compared to analogs: for example they are less susceptible of external noise (e.g. a small change in the voltage may not affect the signal's value).
A digital signal whose value can assume only two different states is called *binary signal*. Starting from these two states one can obtain more numbers by taking a combination of multiple binary signals: this is, for example, the idea behind a computer memory, who stores binary digits to represent decimal numbers by means of *flip flops*, electronic circuits that will be presented in the following pages.
Binary signals can be compared, processed and modified by the use of *logic gates*. The logic gate is a physical implementation of a boolean operation, a function that operates on the set of binary states. Given a boolean operation one can associate a *truth table*, that is simply a list of the results of the operation for all the possible combination of the inputs. For example two of the basic boolean operations are the AND and the NOT, often represented with the symbols " & " and " $^-$ ". The corresponding truth tables are

A	B	$A\&B$
0	0	0
0	1	0
1	0	0
1	1	1

A	$\overline{A}$
0	1
1	0

Table 1: Truth tables of the AND and NOT logic operations.

One can prove that the set of operations {AND, NOT} is a functional complete set, or, in other words, every other boolean operation can be obtained by a particular combination of ANDs and NOTs only.
Two other well known functional complete sets are {NAND} and {NOR}: in both cases, the multiple use of one single gate can implement every possible boolean operation.

In the first part of this experience we first built a NOR gate and we then tried to obtain a XOR gate by the use of NANDs only. In the second part we first tried to implement an RS flip flop by using NAND gates, and then implemented a frequency divider and a counter by using pre-built flip-flops.

1 NOR logic gate

One of the most frequently used logic operation is the NOR. The NOR operation between two logical variables A and B is usually denoted by the symbol $\overline{A+B}$ which highlights the fact that the NOR operation is equivalent to a NOT operation applied to the result of an OR operation. Table 2 reports all the possible results of a binary NOR.

As a first step in this experience we assembled a NOR gate implementing the circuit in figure 1.
Suppose that both switches A and B are open: the transistor's base is then connected to ground, hence the bjt is in the interdiction state and the output is approximately 5 V. If we close at least one switch, the bjt goes in the saturation state and the output goes

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Table 2: Truth table of the NOR logic operation.

to approximately 0 V. This corresponds to the NOR logic gate's truth table in the TTL logic.

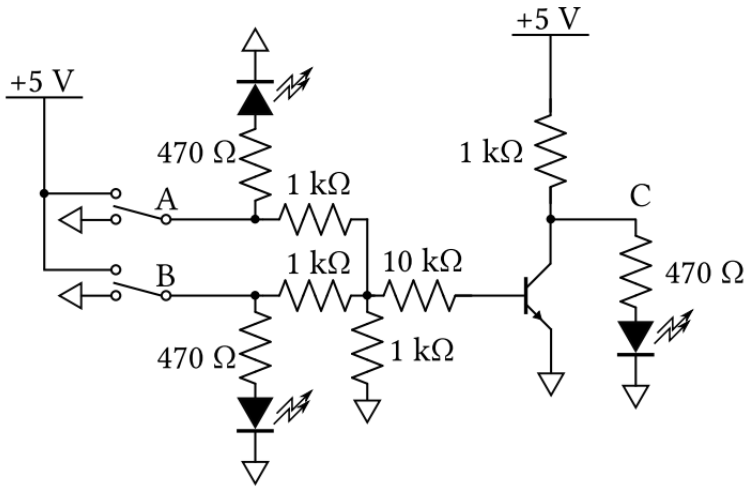


Figure 1: The reported circuit is an implementation of a NOR gate. After assembling this circuit, we used it in the following parts of the experience to obtain other logic gates.

The correct functioning was checked by attaching a led diode to the output as reported in the circuit diagram, and connecting the points A and B of the circuit to 5 V or 0 V (TTL logic) to simulate the previously reported truth table. Photos of the correct functioning are reported in figure 2.

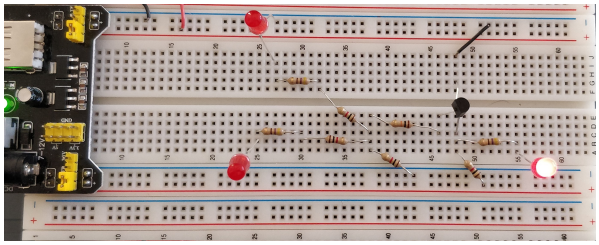
2 XOR logic gate

Another frequently used logic operation is XOR, whose symbol is $\oplus$. The X in XOR stands for "exclusive", because the XOR of two boolean variable is 1 if one of them is 1, but 0 if both are 1. Its truth table is reported in table 3.

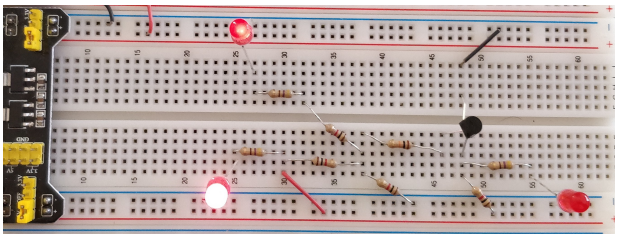
A	B	A $\oplus$ B
0	0	0
0	1	1
1	0	1
1	1	0

Table 3: Truth table of the XOR logic operation.

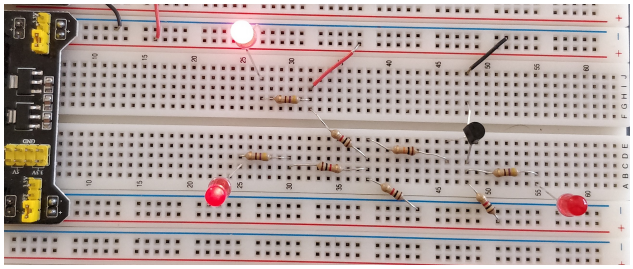
In this experience our goal was to implement a XOR logic gate with means of only 4 NAND logic gates. To accomplish this task, we need to rewrite the XOR operation in terms only of NAND operations in the following way:



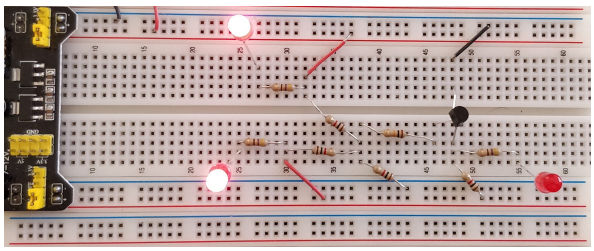
(a) $A = 0$, $B = 0$



(b) $A = 0$, $B = 1$



(c) $A = 1$, $B = 0$



(d) $A = 1$, $B = 1$

Figure 2: Visualization of the states of an NOR gate. The two leftmost leds represent the inputs A and B, while the rightmost one represents the output.

$$A \oplus B = (A + B)\overline{AB} \tag{1}$$
$$= A\overline{AB} + BAB \tag{2}$$
$$= \overline{\overline{A\overline{AB} + B\overline{AB}}} \tag{3}$$
$$= \overline{\overline{A\overline{AB}} \cdot \overline{B\overline{AB}}} \tag{4}$$

Translating it in terms of logic gates, we have to implement the circuit in figure 3.

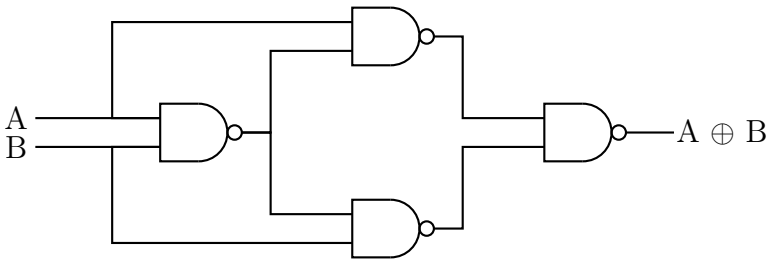
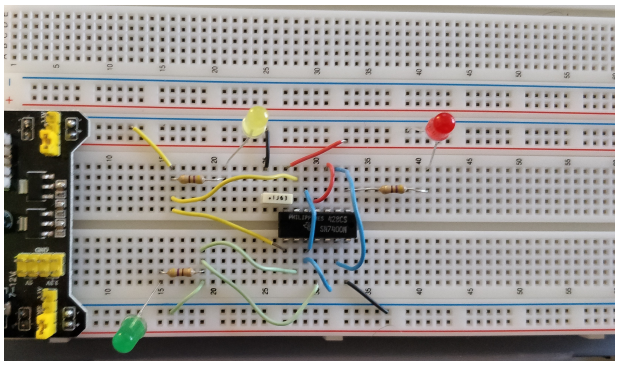


Figure 3: Implementation of a XOR logic gate with 4 NAND.

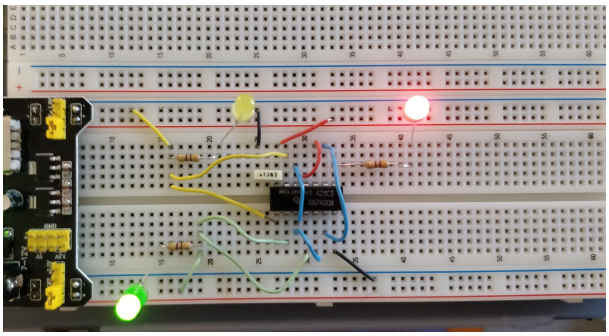
We created the circuit represented in figure 3 with the 74xx00 circuit, which is composed of 4 NAND gates. We used three leds to visualize the state of the variable A, B and A $\oplus$ B. Pictures in figure 4 show that the circuit behaves as the table of truth 3 predicts.

3 RS flip-flop

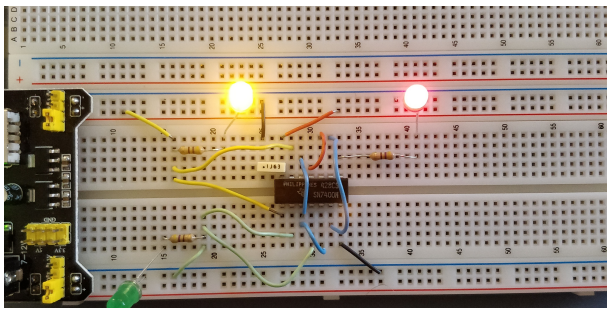
Another very important and useful digital circuit that we studied is the RS flip-flop. Figure 5 reports an implementation of a RS flip-flop.
In order to carry on the circuit analysis, let's define two boolean variables $A \equiv \bar{S}$ and



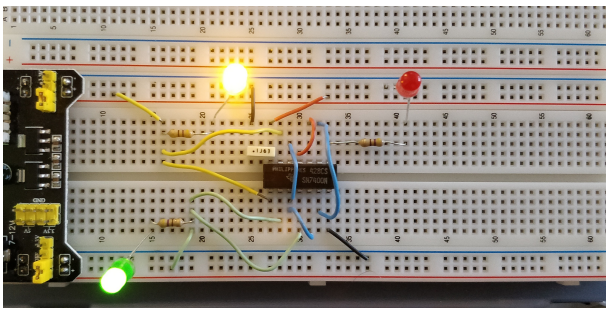
(a) $A = 0$, $B = 0$



(b) $A = 0$, $B = 1$



(c) $A = 1$, $B = 0$



(d) $A = 1$, $B = 1$

Figure 4: Visualization of the states of an XOR gate. The green and yellow leds represent the inputs A and B, while the green one represents the output.

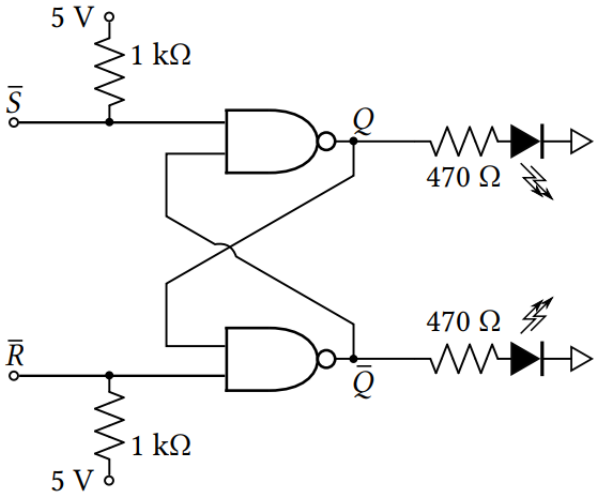


Figure 5: Circuit of a RS flip-flop. The SET command brings Q in the high state. The RESET, on contrary, in the low state. In both cases the state is maintained even if the command is released, until the opposite command is activated

$B \equiv \bar{R}$ that correspond to the gates' inputs, and then we will translate the results in therms of $\bar{S}$ and $\bar{R}$.

1. Suppose that initially A is HIGH (4.5V $\sim$ 5.0V in TTL logic) and B is LOW (0.0V $\sim$ 0.5V). Then Q is LOW and $\bar{Q}$ is HIGH.

2. Now let's bring B in the HIGH state, keeping A fixed: what we obtain is that $\bar{Q}$

remains HIGH since Q was LOW an instant before, and, consequently, Q is LOW

3. Putting A in the LOW state keeping B fixed causes Q to change state to HIGH. The change of Q causes a change in $\bar{Q}$ as a consequence, to a LOW state.

4. Now, bringing again A in the HIGH state keeping B fixed, we obtain that Q is HIGH since $\bar{Q}$ was low an instant before, and, consequently, $\bar{Q}$ is LOW.

Let's consider Q as the output of the circuit. We call pin A the SET input, and pin B the RESET input. Here is a justification of the names: by assigning $A = 0$ keeping $B = 1$, we have $Q = 1$, or, in other words, we are *setting* the output. On the opposite, by assigning $A = 1$ and $B = 0$ we cause the output to be 0, or we *reset* it. The case in which both are 1, which corresponds to cases 2 and 4 in the previous description, simply keeps the output in the immediately previous configuration.
Finally, we do not want $A = 0$ and $B = 0$ at the same time, since it would cause the flip-flop to enter an indefinite state.
The fact that the flip-flop works in an inverted logic (we *set* the output when the SET pin is 0), justifies the symbol $\bar{S}$ for A and $\bar{R}$ for B.
Truth table 4 summarises the analysis.

$\bar{S}$	$\bar{R}$	Q	$\bar{Q}$
0	0	-	-
0	1	1	0
1	0	0	1
1	1	x	$\bar{x}$

Table 4: Truth table of the RS flip-flop reported in figure 5

4 D-type flip-flop

4.1 Frequency divider

We then studied a *sequential* circuit, which means a circuit whose state is determined by the transitions of some variables, rather than just by their states. An important example is the *D-type flip-flop*, whose scheme is represented in figure 6. It works as follows: it receives as input the clock, which is a TTL square wave, and a 1-bit variable D. When the clock has a *rising edge*, i.e. it has a transition from low to high, the variable D is "written" into the output Q. There is then an additional output which is always the negation of Q, so $\overline{Q}$.

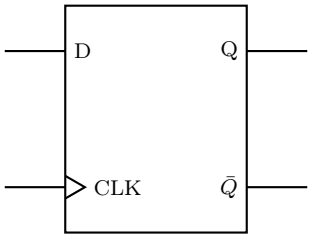


Figure 6: Scheme of a D-type flip flop.

D-type flip-flops can be used to create a frequency divider; in particular, with one such flip-flop one can divide a frequency in 2. To do that, one just needs to do connect the

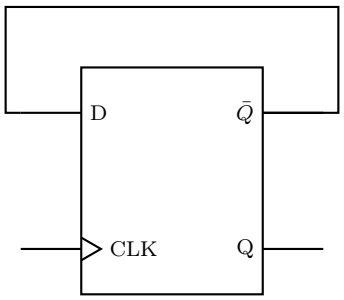


Figure 7: Implementation of a frequency divider in 2 with a D-type flip flop.

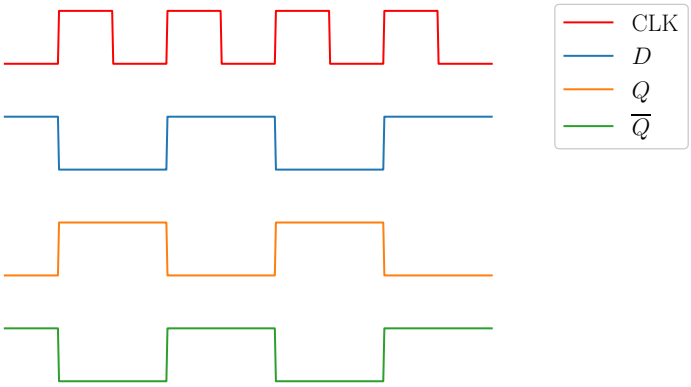


Figure 8: The value of the clock, D, Q and $\overline{Q}$ in time in a frequency divider by 2. On the rising edge of the clock, D in written into Q. The square waves are to be intended as TTL signals: when they are low they are at about 0 V, when high at about 5 V. They are represented vertically one above the other just for a comparison in time.

input pin D with the output $\overline{Q}$ ar represented in figure 7. This configuration divides the clock's frequency in 2 for the following reason: suppose D is 1 at the beginning. When the clock has the first rising edge, D is written into Q, which becomes 1, and so $\overline{Q}$ is 0. Just a few seconds after (typically $\simeq 10$ ns), because D and $\overline{Q}$ are connected, D becomes 0. This state ($D = 0$, $Q = 1$, $\overline{Q} = 0$) remains steady until the next clock rising edge, when Q will become 0, $\overline{Q}$ will become 1 and just slightly after D will become 1. So if we visualize, for instance, the value of Q, we would see that it has a certain value, say 1, for an entire clock cycle, and the negated, 0, for the next one. The entire process is repeated over time, and it is clear that it happens with a frequency that is half the frequency of the clock. The evolution in time of the clock, D, Q and $\overline{Q}$ is represented in figure 8.
We realised the circuit in figure 7 with a 74xx74, which is composed of 2 D-type flip flops (of which we used only one). We used as clock a square TTL signal at 1 kHz and verified that the ouput Q had actually half the frequency of the clock.

But why to limit the division of the frequency to a factor of 2? In fact, by using N D-type flip-flops we can divide the original frequency by 2^N. To do this, we can just connect the output Q_0 of the first flip-flop to the clock of the second one, CLK_1, then connect Q_1 to CLK_2 and so on. By applying the same concepts that we explained for the frequency divider in 2, we can see that every flip-flop halves the frequency of its clock, which in turn has already half the frequency of the clock of the previous flip-flop. The

scheme for a frequency divider by 8 is shown as an example in figure 9.
We implemented firstly a frequency divider by 4, by using 2 flip flops, then a divider by

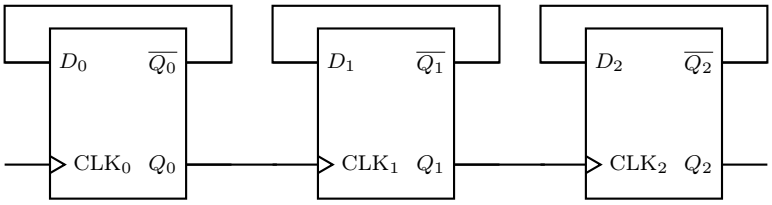


Figure 9: Implementation of a frequency divider by 8: Q_3 is a square wave with frequency that is 1/8 of that of CLK_0.

8, using 3, and finally a divider by 16, using 4 flip-flops. We verified that the output Q of the last flip flop was actually a square wave with frequency respectively 1/4, 1/8, 1/16 of the frequency of the first clock. In figure 10 is reported a photo of the implementation of the divider by 16.

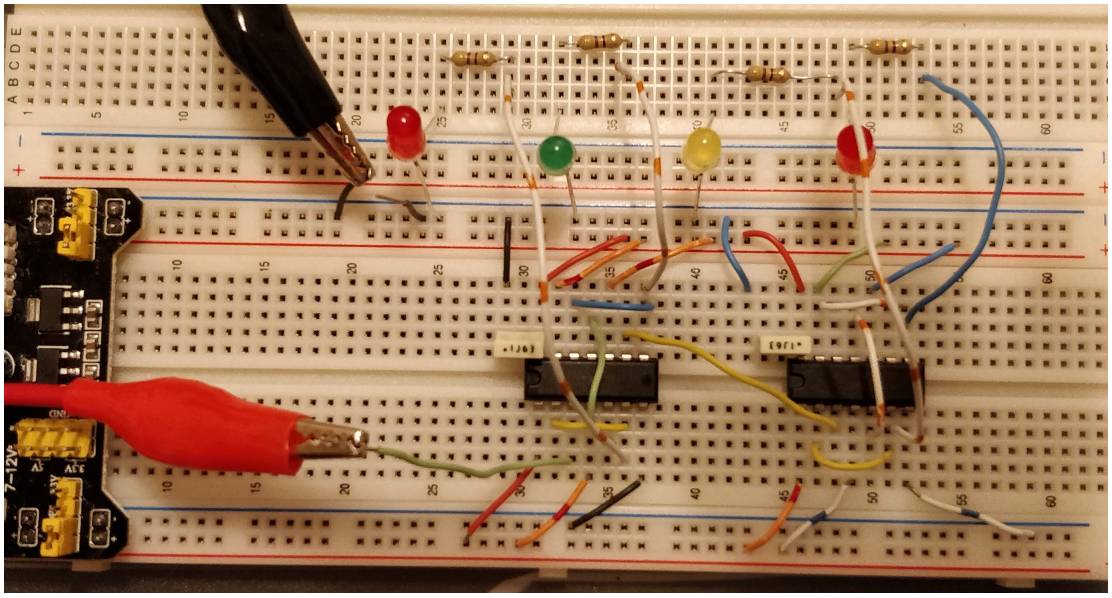


Figure 10: Implementation of the frequency divider by 16, which was also used as a counter. Leds were used to verify the correct counting.

4.2 Counter

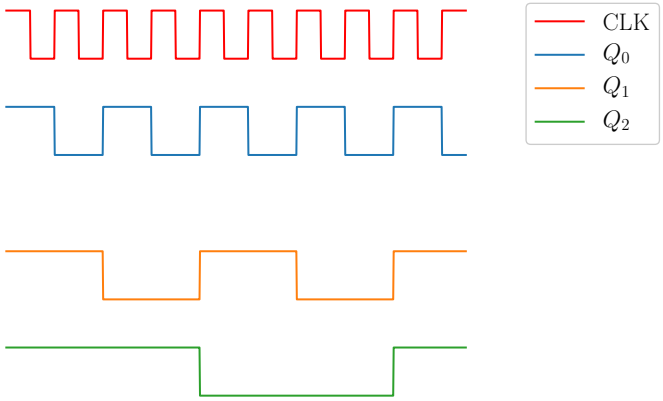
The previous circuit that implements a frequency divider can also be used to obtain a counter. Let's consider, for instance, a frequency divider by 8, whose scheme is represented in figure 9. We can visualize the state of the three Qs with some leds: when the leds are on, the corresponding Q is 1, otherwise 0. With three bits we can represent a number n so that $0 \leq n \leq 7$ by defining, for instance, its value in 10-base as

$$n_{(10)} = Q_2 * 2^2 + Q_1 * 2 + Q_0 \tag{5}$$

The sequence of the represented number can be obtained by looking at the state of the various Q_i in time, as shown in figure and table 11: we obtain a countdown! We verified this behaviour, actually not for the divider by 8 but for the divider by 16, by using leds

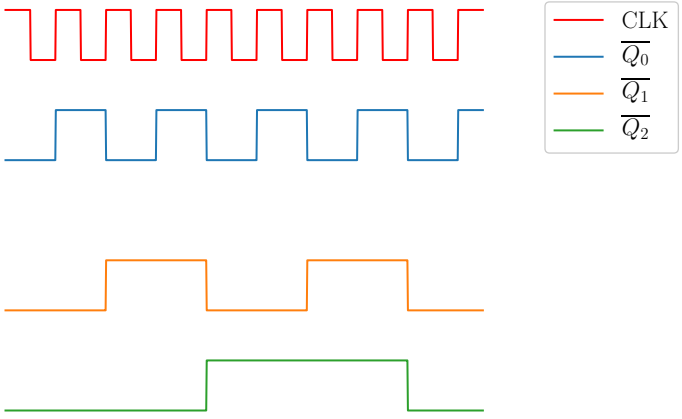
to see the states of the Qs. Here we explained the idea by using the divider by 8 as a shorter example.

One may wonder: what if I wanted a forward counter instead of a countdown? No problem, you just need to visualize the state of the $\overline{Q}$s instead of the Qs. This is visualized in figure and table 12. Again, we verified this behaviour for the frequency divider by 16.



Q_2	Q_1	Q_0	n
1	1	1	7
1	1	0	6
1	0	1	5
1	0	0	4
0	1	1	3
0	1	0	2
0	0	1	1
0	0	0	0
1	1	1	7
1	1	0	6
...			

Figure 11: Value in time of the clock and the Qs in the frequency divider by 8 (figure 9). Signals should be thought in the TTL logic. They are represented vertically one above the other just for a comparison in time.



Q_2	Q_1	Q_0	n
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7
0	0	0	0
0	0	1	1
...			

Figure 12: Value in time of the $\overline{Q}$s in the frequency divider by 8. Signals are to be intended as TTL, they are represented vertically one above the other just for a comparison in time.

Conclusion

This experience has let us understand how the fundamental logic gates can be implemented, which is something fascinating because logic gates are everywhere in today's technology. We also saw some applications of logic gates: the first one was an RS flip-flop, and we understood how it can be used in memory devices to store values. The second one was a D-type flip-flop, that can perform frequency division (something quite unfeasible with the "normal" electronics of resistors and capacitor that we knew before), and behave as a forward or backward counter. All these applications made it very clear to us how powerful the world of digital electronics can be.