Cyber-Physical Systems

Authorship: Francesc Aulí, Joaquin Garcia-Alfaro, Oriol Pujol

Preliminaries

1. **The digital world vs. the analog world**
   - 1.1. The analog-to-digital conversion process
   - 1.2. Analog signal representation
   - 1.3. Sampling, quantization, and encoding of an analog signal
   - 1.4. The Nyquist theorem

2. **The concept of digital information**
   - 2.1. Binary representation vs. decimal representation
   - 2.2. The binary system
   - 2.3. Storage units
   - 2.4. Logic gates (AND, OR, NOT)
   - 2.5. Logic gates represented by transistors
   - 2.6. Addition of binary numbers
   - 2.7. Adder, or summer, digital circuit
   - 2.8. Complementing digital circuits with a clock

Introduction

We can describe the world we live in as a continuous or analog world. This nature implies that the information we perceive (image, sound, symbols, etc.) cannot be directly represented and stored in digital format without a previous digitization step. The purpose of this chapter is to introduce the concept of analog and digital information and to see how this conversion can be carried out to process and manipulate information in our real world.

1.1. The digital world vs. the analog world

Cyber-physical systems, as we understand them today, include tools that store and process data of a single nature: digital. It is essential to convert the continuous information of the real world into a digital format to be able to deal with it conveniently by computers, cameras, sensors, and so on. Logically, we can also transform digitally stored information into analog to be able to perceive it again in our environment (see Figure 1.1.1).

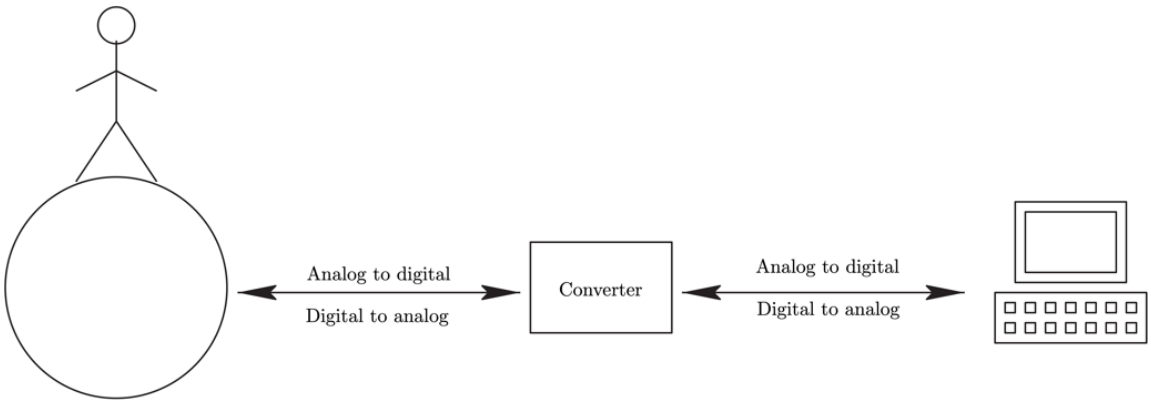

Figure 1.1.1. The analog-to-digital conversion process, or vice versa, requires specific equipment.

We find this transformation process every day when using a computer. Speech recognition programs, for example, transform sound waves into digital information to be able to recognize certain patterns (the phonemes) and later turn them into words and sentences. Similarly, reading programs convert digital information (text stored on our computer) into sound waves that we can perceive and understand.

The discretization-digitization process of an analog signal goes through three distinct steps:

- Sampling: retrieving instantaneous values of the signal at regular intervals;
- Quantization: assigning a discrete value to the continuous value retrieved in the sampling step;
- Encoding: transform to the (usually binary) encoding from the previous step.

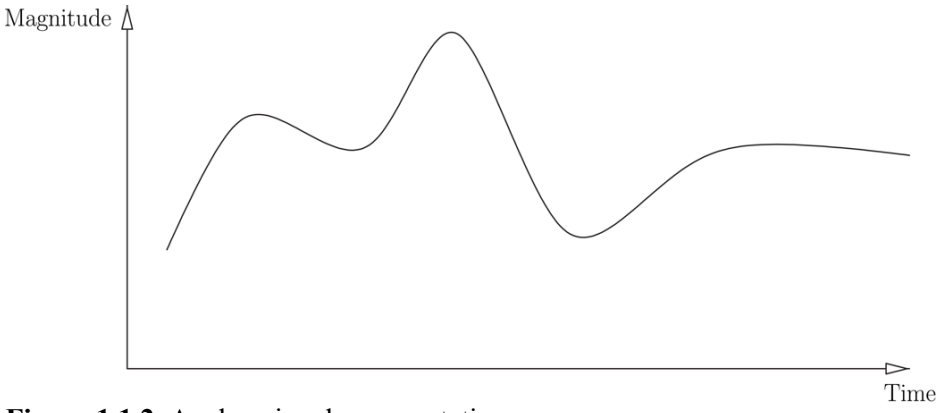

Figure 1.1.2. Analog signal representation.

Let us start with an example, to better understand how sampling an analog signal works. Figure 1.1.2 depicts an analog signal representation, for instance, an analog signal representing sound. It shows what a continuous representation looks like. If we want to store the signal without any digitization step, we would need infinite values. The sampling process is hence responsible for taking values of this continuous signal at regular time intervals. Figure 1.1.3a depicts the idea, in which vertical lines represent this sampling of the same continuous signal shown in Figure 1.1.2.

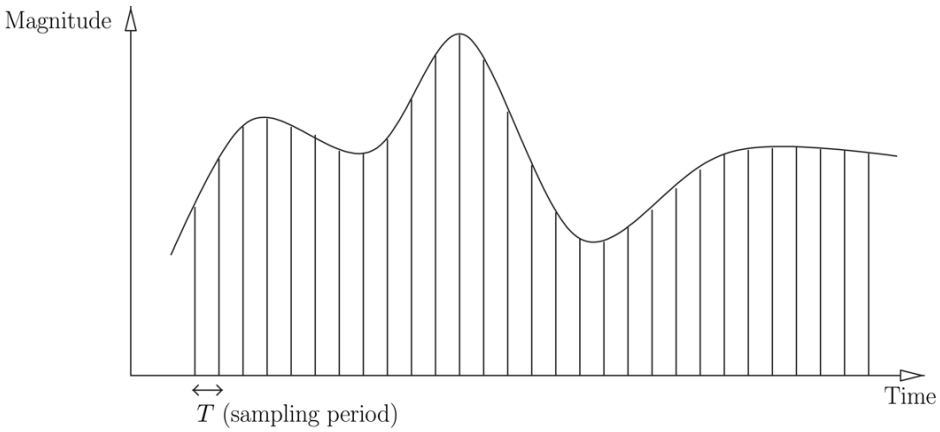

Figure 1.1.3a. Sampling of an analog signal.

What we achieve with the sampling process is to discretize the signal in time (or in space, in case we deal with images). This way, instead or requiring infinite points to store the signal, we would only need those points identified within the sampling period, i.e., within an interval. The quantization process does the same but on the magnitude axis. Notice that if we do not discretize this axis, we would encounter the same problem: we would have infinite representation points. To quantize this signal, then, we divide the magnitude axis into different intervals and approximate the value we get from the sampling to the discretized point closest to the original (see Figure 1.1.3b).

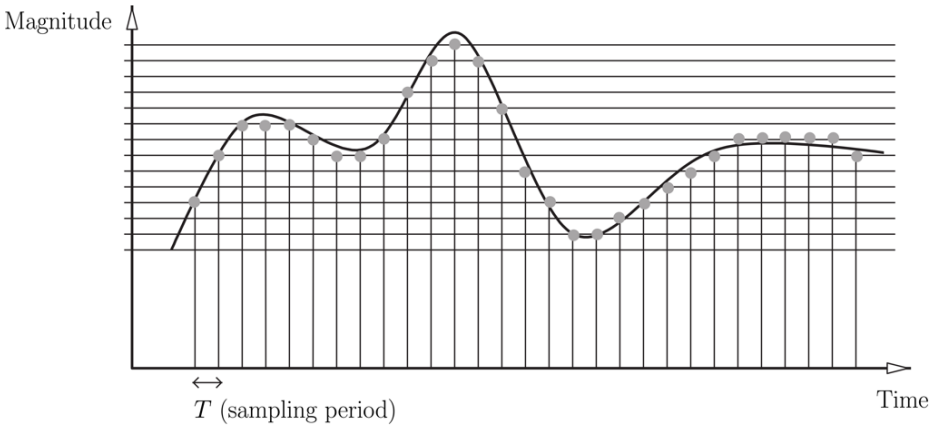

Figure 1.1.3b. Quantization of an analog signal.

The points we see circled in Figure 1.1.3b are the ones we keep. Later, during the encoding process, we assign (or map) a value to each point in the signal. In our previous example, we could carry out this coding by assigning a consecutive value to each quantization interval and sequentially saving these values to be able to treat this signal digitally. Figure 1.1.3c shows this mapping, graphically.

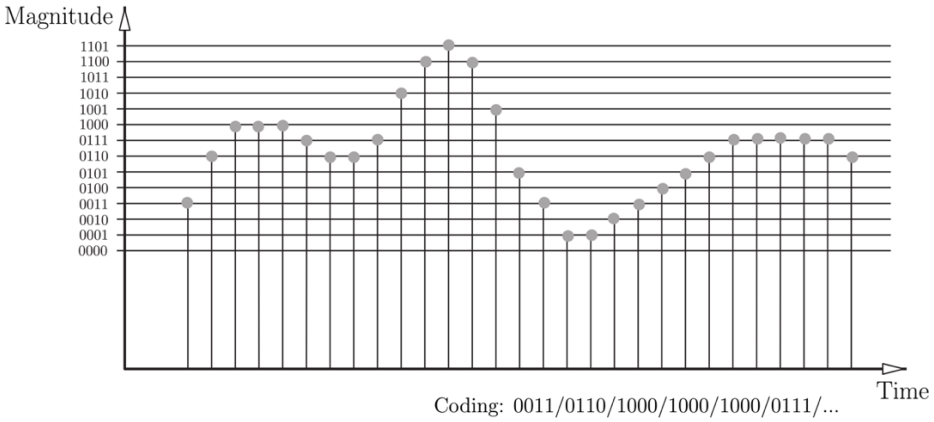

Figure 1.1.3c. Encoding of an analog signal.

By following the process shown in Figures 1.1.3a—1.1.3c, we can transform analog or continuous information of our real world into digital information, which we can process in a more efficient manner by digital computers. The definition of the precise sampling and quantization intervals would decide the quality of the signal. The smaller the interval, the better we can reproduce the original signal but also the larger to storage we need to store the information associated to the signal. It is therefore a matter of achieving a balance between quality and quantity. The tools and standards used to carry out the digitization process give some guidance to manage that balance for us.

The Nyquist theorem [1] tells us that to accurately reproduce an analog signal, the sampling period must be at least half the size of the smallest oscillation of our input signal. Figure 1.1.4 depicts the previous idea, graphically, showing us how to reproduce the original signal with the minimum number of samples.

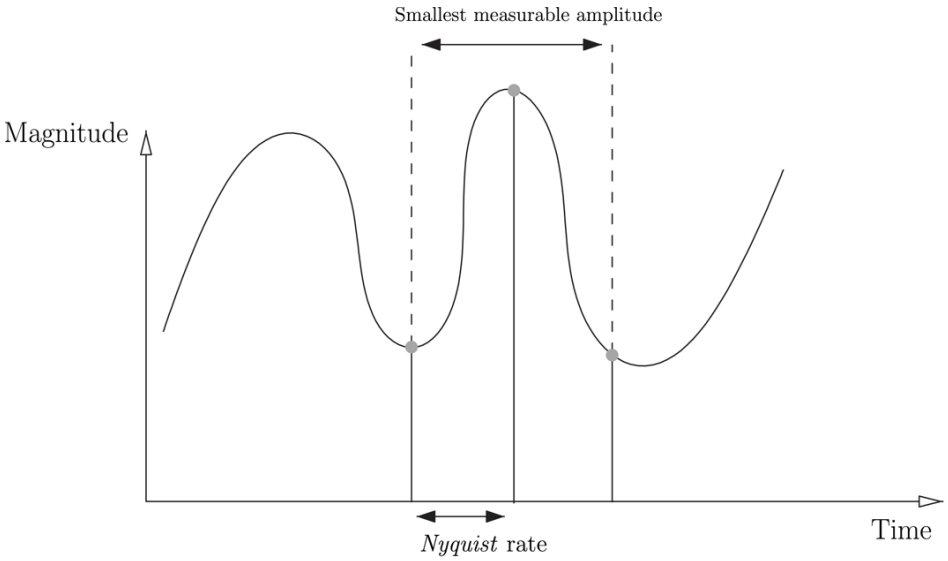

Figure 1.1.4. The Nyquist theorem.

The Nyquist theorem

The Nyquist theorem [1], also known as the Nyquist–Shannon sampling theorem [2] or simply the sampling theorem, can be summarized with the following statement: *to accurately reproduce a signal, the sample rate must be twice the highest frequency*.

Another way to talk about this interval is with what we call the sampling frequency, which is the inverse of the period and we measure it in Hz:

$$\frac{1}{T} = f_{sampling}$$

In many standards, we find that instead of talking about the sampling period we are told about its frequency, which is why it is important to differentiate these two concepts.

With the quantization process, we do not have any theory that ensures an exact reproduction of the original, but as research has been carried out, new and increasingly complex techniques have appeared that allow the original information to be represented with high quality and using the least space possible.

A proper example of the above observation are the JPEG and JPEG2000 standards (see https://jpeg.org/jpeg2000/), which use optimization and data compression methods to drastically reduce the size of digital information.

1.2. The concept of digital information

In the previous section, we have seen how to discretize an analog signal. It relies on a series of transformations and encoding into digital numbers that we can manipulate, later on. Although in our daily life we are used to dealing with base 10 numbers, in the digital world we manipulate information in base 2.

Two main reasons lead us to store information in base 2. First, we have the fact that storing information in base 2 is much more economical than in base 10. Consider the following example: imagine that we need to represent any number between 0 and 999 in base 10. To do this, we need 10 times 3 different digits (that is, 30 different digits). In base 2, on the other hand, representing 1024 (that is, 210) only costs us 20 digits (see Figure 1.2.1). This saving, when it comes to storing a lot of information, can become very significant.

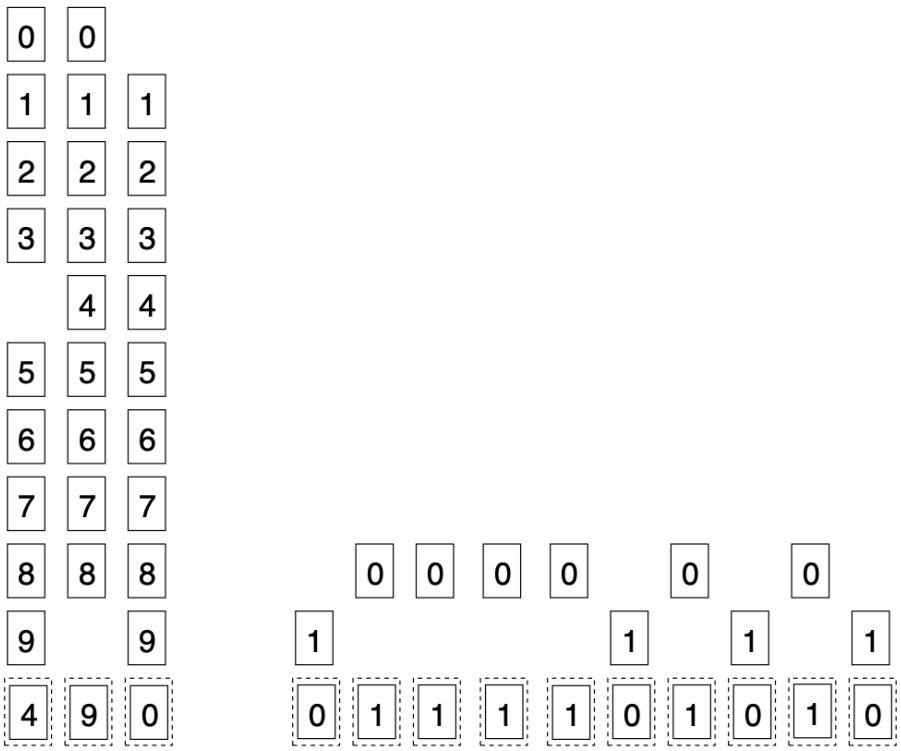

Figure 1.2.1. Binary representation vs. decimal representation.

The second reason we represent information in base 2 is due to the nature of electronic circuits. It is very easy for us to represent the symbol 0 as the lack of voltage and the symbol 1 as the voltage of a circuit. This way, electronic circuits allow us to do all kinds of operations with just these two states. Mathematically, it can be proved that the most efficient form of representation is base 3, but for practical reasons, 2 has been chosen since the loss of efficiency is minimal and its implementation in electronic circuits is much easier, technically speaking.

Figure 1.2.2 depicts how the binary system works, as well as its equivalence to the decimal system we use in our daily life. It shows that the binary representation is more efficient than the decimal representation. Naturally, all the operations we can do with one system, we can also do with the other. In other words, the two are completely equivalent. When we work with a computer, al l the internal operations are conducted in base 2, although the results can be represented to us in base 10, to ease our understanding.

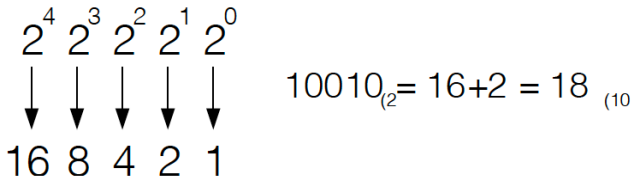

Figure 1.2.2. The binary system.

To express large bit units, we need to agree on several different measurements. As we can see in Tables 1.2.3, we take what we call a byte as the base unit. The byte is the grouping of 8 bits and, as we have already seen, a bit can only represent a symbol 0 or a symbol l. Hence, with 8 of these units we can represent 28 (i.e., 256) different values. The byte has been taken as the basis of representation for historical reasons, and by making multiples of 210 (i.e., 1024) we get the other units we so frequently use when referring to the storage capacity of our disk drive, memory, etc.

Table 1.2.3. Storage units.

Unit	Abbr.	Quantity
bit		0/1
byte		8 bits
kilobyte	KB	$2^{10} = 1024$ bytes
megabyte	MB	$2^{20} = 1024$ kilobytes
gigabyte	GB	$2^{30} = 1024$ megabytes
terabyte	TB	$2^{40} = 1024$ gigabytes
petabyte	PB	$2^{50} = 1024$ terabytes

Ternary computers

Ternary computers use three-state number systems, which are considered the most economical and efficient representation of all possible integer bases.

Balanced ternary notation which consists of *trits* (or ternary digits) can be represented with -1, zero, and one (while ordinary ternary notation can be represented with zero, one, and two) [3].

Ternary computing uses trinary logic. The Setun ternary computer, built in 1958 at the Moscow State University, was considered superior to other binary state computers of the same era. However, and due to the successful deployment of digital electronics, ternary designs diminished in significance. Some authors argue that they can potentially experience growth again in the future, due to the symmetric properties and elegance of balanced ternary notation [3].

To perform binary operations, we use what is called Boolean algebra, which is nothing more than a series of basic operations from which we can do all the others. The most basic logical operations are AND, OR and NOT. Tables 1.2.4a–1.2.4c show their rationale. From now on, we represent the truth with the symbol l and the false value with the symbol 0. This is how operations are conducted in all digital circuits.

Table 1.2.4a. AND logic gate.

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.2.4b. OR logic gate.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

Table 1.2.4c. NOT logic gate.

Input	Output
0	1
1	0

Digitally speaking, logical gates are implemented using electronic relays that open and close whenever a given amount of electricity shall pass by. Those relays are transistors [4]. Transistors are the physical basis on which all digital circuits are designed. Figure 1.2.5 shows how the three main logic gates (AND, OR and NOT) are implemented.

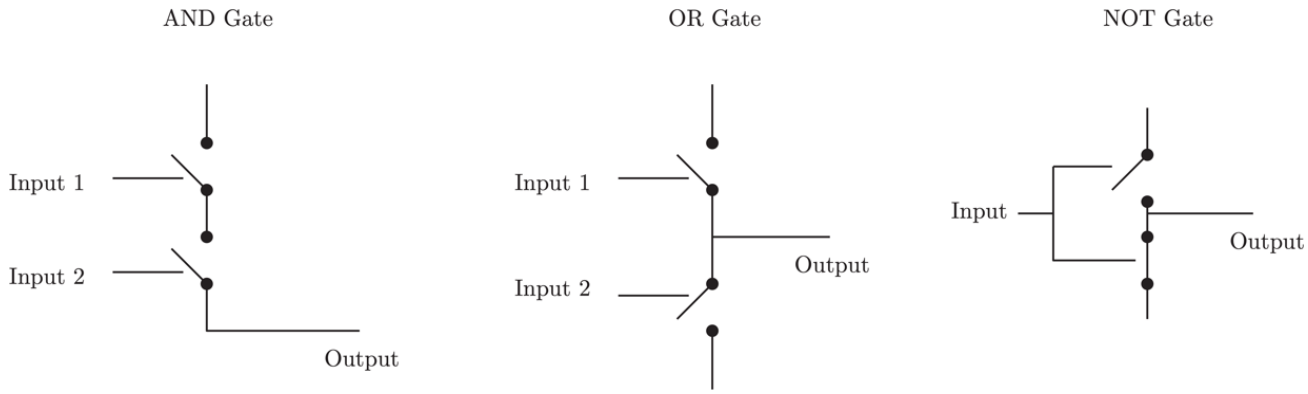

Figure 1.2.5. The three main logic gates (AND, OR, NOT), represented by transistors.

Table 1.2.6. Addition of binary numbers

Bit 1	Bit 2	Output
0	0	00
0	1	01
1	0	10
1	1	11

With these gates, we could conduct any operation. Let us see this with an example. Assume we want to compute the addition of two binary numbers (assuming one digit, each). The first thing we have to do is to build a table containing all the possible results (see Table 1.2.6). We can now design a digital circuit for each output of our previous operation (see Figure 1.2.7). Notice that the addition of two numbers, assuming one bit per number, leads to a two-bit output. Hence, we shall prepare two different circuits, one for each possible output. Likewise, if we had more inputs and more outputs, we would need to prepare a digital circuit for each combination. There exist techniques that allow us to minimize as much as possible the number of gates needed to implement those multiple combinations, to reduce the cost (in terms of transistors) as much as possible. These techniques are based on the concept of Karnaugh maps [5].

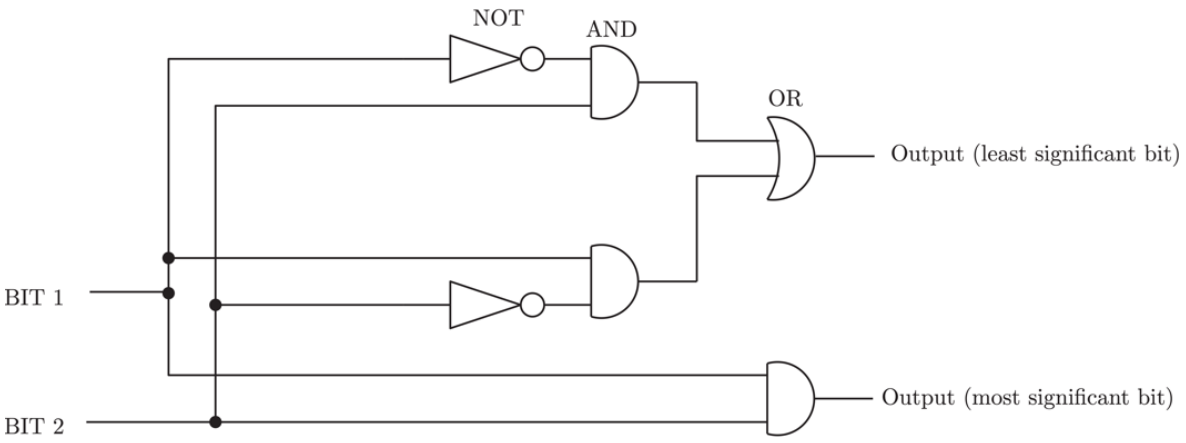

Figure 1.2.7. Adder, or summer, digital circuit.

Another problem we may face when dealing with digital circuits is knowing when we have the correct data to operate as we want. Figure 1.2.8 shows how the path from the different units that provide the information to the digital circuit (which is in charge of performing the operations) has a different length. These differences mean that, even if the data leaves units 1, 2 and 3 at the same time, it arrives at the digital circuit at different times. We might think that the solution is to make the paths that transport the data between the units and the circuit have different speeds, but this may not be feasible (technologically speaking). To solve this problem, the concept of the clock is introduced. The clock is a small circuit that gives us a signal (a voltage rise) every certain time interval, so that the digital circuit, upon receiving this signal, knows that its inputs have

Digital circuits

We refer to digital circuits as all those circuits that use a digital signal to carry out their operations [4]

the correct information and, therefore, that the can process There are several ways to make the clock's circuitry, one of which is to use a quartz crystal, which provides very precise and constant oscillations that make it ideal for this kind of situation.

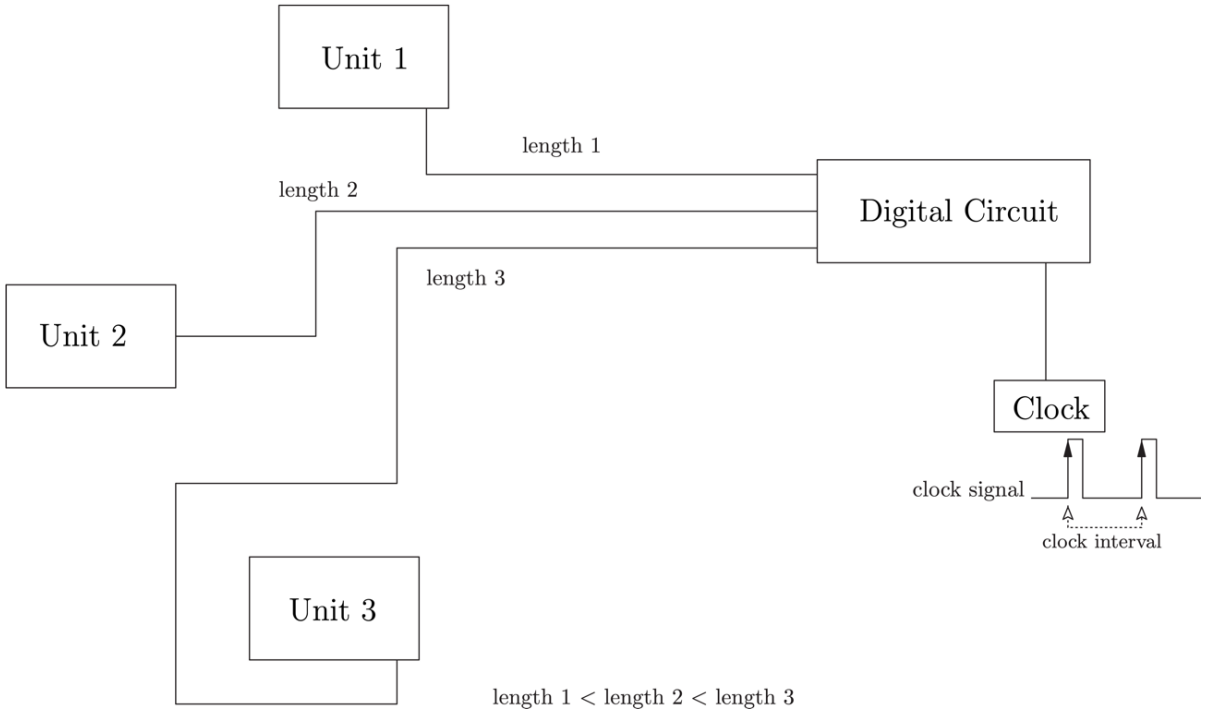

Figure 1.2.8. The digital circuit needs to know when it has the correct data to operate.

The period or interval of this clock must be equal to or greater (to leave a safety margin) than the maximum propagation time of the signal. When the clock gives us a signal, we know that data reached the input of the circuit, i.e., we can process such data correctly. Digital circuits that use a clock are called synchronous digital circuits, because of the synchrony that the clock provides them, although there are also asynchronous digital circuits which, by their nature, do not need a clock to synchronize them.

The clock period in our digital circuit is what allows us to set the maximum speed at which our device can operate. Logically, the simplest operations that our circuit can perform shall last at least one clock period. Hence, the frequency allows us to know the maximum number of operations per second that can be performed. For instance, suppose we have a microprocessor that works at 3 GHz. Since 3 GHz is exactly three times 109 Hz, we set the clock period to T = 1/(3.0 · 109 Hz), i.e., 3.3e-10 (three nanoseconds). Hence, if every clock cycle could do one operation, that means that within one second we could do more than 1 200 million operations.

In practice, most operations performed by current microprocessors are not usually following those theoretical estimations. Rather, they may need more than one clock cycle to complete the operation. Classical operations such as multiplication and division have a fairly high complexity. They cannot be designed with a low number of logic gates. They require several execution steps before reaching completion. This is where computer architecture theories are required, to find engineering strategies to reduce as much as possible the number of clock cycles that complex operations may need to complete their execution. For this reason, the computational power is measured in terms of accomplished operations, rather than in clock cycles.

Clock cycles

Measurements used to quantify the power of a CPU are, for example, the millions of floating-point operations per second it can produce (known as GFLOPS).

References

[1] Nyquist, H. (1928). *Certain topics in telegraph transmission theory.* Transactions of the American Institute of Electrical Engineers (AIEE), Volume 47, Issue 2, April, Pages 617-644, https://doi.org/10.1109/T-AIEE.1928.5055024

[2] Shannon, C. E. (1949). *Communications in the presence of noise.* Proceedings of the Institute of Radio Engineers (IRE), Volume 37, Issue 1, January, Pages 10-21, https://doi.org/10.1109/T-AIEE.1928.5055024

[3] Knuth, D. (1997). *The art of computer programming*. Volume 2. Seminumerical Algorithms, Chapter 4 (Arithmetic), Section 4.1. Positional Number Systems, Pages 207–208. Addison-Wesley, third edition, https://www-cs-faculty.stanford.edu/~knuth/taocp.html

[4] Tocci and Widmer (1998). Digital Systems: Principles and applications. 7th edition. Prentice Hall. Pearson Education International.

[5] Katz, R. H. (1994). Contemporary Logic Design. The Benjamin / Cummings Publishing Company. Pages 70–85.