ESD - Elettronica dei Sistemi Digitali

The Digital Abstraction

Prof. Riccardo Berta

2025.09.26

Contents

1 The Digital Abstraction

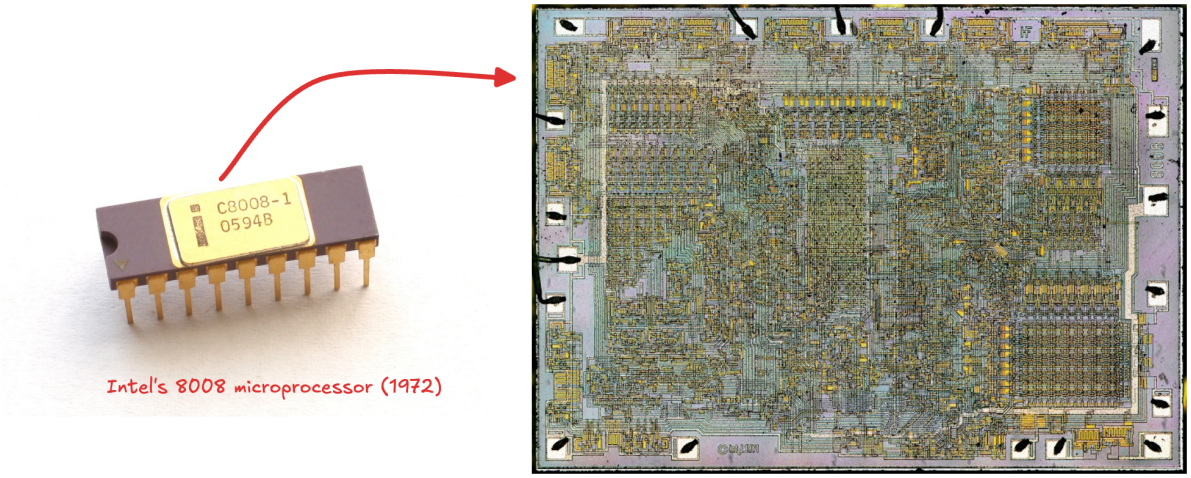
Microprocessors have revolutionized our world over the past three decades. A microprocessor is an **electronic circuit** that acts as the "brain" of a computer. It performs arithmetic, logic, control, and input/output operations by executing instructions stored in a program. Thanks to these devices, today's laptop computers have more capability than room-sized mainframes of the past. Microprocessors are not only found in personal computers: modern automobiles, for example, contain around one hundred of them, controlling everything from engine performance to safety systems. The economic impact has been equally impressive. The worldwide **semiconductor industry** has grown enormously, with sales rising from about 21 billion US dollars in 1985 to over 520 billion dollars in 2023. This growth reflects the **central role of digital technology** in every aspect of our society.

Modern digital systems are built from millions, or even billions, of **transistors**. A transistor is a tiny electronic component that can act as a **switch** or as an **amplifier**. By controlling the flow of current, it serves as the fundamental building block of all electronic circuits. However, the **complexity** of such systems is staggering:

- The first microprocessor, Intel 4004 (1971), had 2.300 transistors.
- The first 32-bit microprocessor, Motorola 68000 (1979), had 68.000 transistors.
- The first 64-bit microprocessor, MIPS R4000 (1991), had 1.35 million transistors.
- The first Pentium processor, Intel Pentium (1993), had 3.1 million transistors.
- The first dual-core processor, AMD Athlon 64 X2 (2005), had 233.2 million transistors.

- The first quad-core processor, Intel Core 2 Quad (2006), had 582 million transistors.
- The first eight-core processor, AMD FX-8150 (2011), had 1.2 billion transistors.
- The first ten-core processor, Intel Core i7-6950X (2016), had 3.2 billion transistors.
- The first twenty-eight-core processor, Intel Xeon W-3175X (2019), had 8.6 billion transistors.
- ...

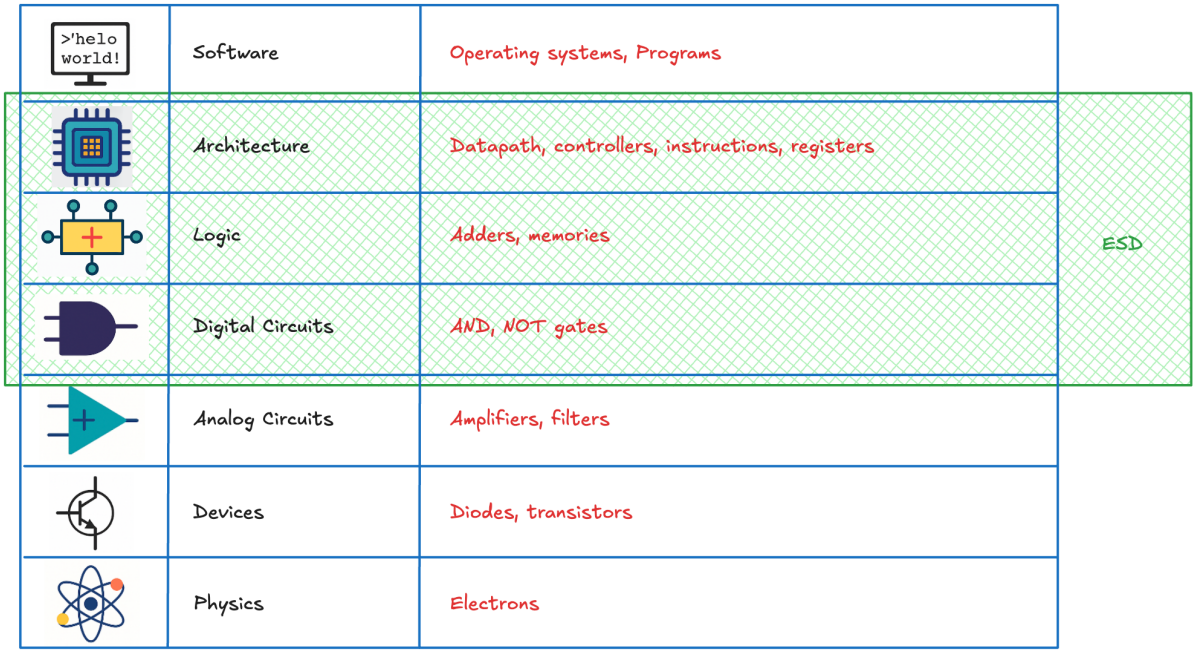
At the lowest level, the behavior of transistors could be described using **semiconductor device equations**: for example, quantum mechanics and Maxwell's equations for electromagnetic fields, or the drift–diffusion equations that model how electrons and holes move inside the semiconductor material. In principle, one could attempt to write down a huge system of partial differential equations to describe the movement of charges in every transistor. But for a processor with billions of devices, this would mean billions of coupled equations, clearly impossible for any human (or even any computer) to solve in practice. For example, this is a picture of the internal of the Intel's 8008 microprocessor (1972) is the ancestor of modern processors that you may be using right now:



It is composed of only 3.500 transistors, this should give you an idea of the scale of complexity involved in modern microprocessors. Therefore, the key challenge is to **manage complexity**. To design and understand digital systems (like microprocessors), we must learn to think in **layers of abstraction**, focusing on the essential features at each level without being overwhelmed by unnecessary details. This strategy enables engineers to build reliable and sophisticated digital systems while keeping the design process under control.

1.1 Abstraction

The most powerful idea in digital design is **abstraction**: the practice of **hiding details** when they are not important for the task at hand. Without abstraction, the enormous complexity of modern digital systems would be unmanageable. The following figure shows the stack of abstractions in a typical digital system:



At the lowest level, everything is governed by **physics**. The motion of electrons is described by quantum mechanics and Maxwell's equations. These equations are essential for understanding semiconductor behavior, but we do not need to deal with them directly when building circuits.

Instead, we step up one level to **devices**. Here, we work with transistors and diodes. A transistor has well-defined terminals and can be **modeled** as a controllable switch or amplifier. This allows us to ignore the messy details of individual electron movement. From devices, we construct **analog circuits** such as amplifiers and filters, which process continuous voltages and currents.

Narrowing further, we focus on **digital circuits**, where voltages are restricted to discrete ranges representing binary values ("0" and "1"). With these digital building blocks, we can design logic gates such as AND, OR, and NOT. Combining logic gates leads us to more **complex components**, including adders, multiplexers, and memories. At this stage, the designer can think in terms of bits and operations rather than individual transistors.

The next step is the **architecture** level, where datapaths, control units, and pipelines are organized to carry out the fundamental operations of a processor. The architecture level provides

the programmer's view of the system: instruction sets, registers, and addressing modes.

Beyond hardware, **operating systems** manage resources such as memory, files, and devices, hiding the complexity of hardware from software developers. At the very top, **application software** allows users to interact with the machine in a way that is completely independent of the underlying physics and electronics.

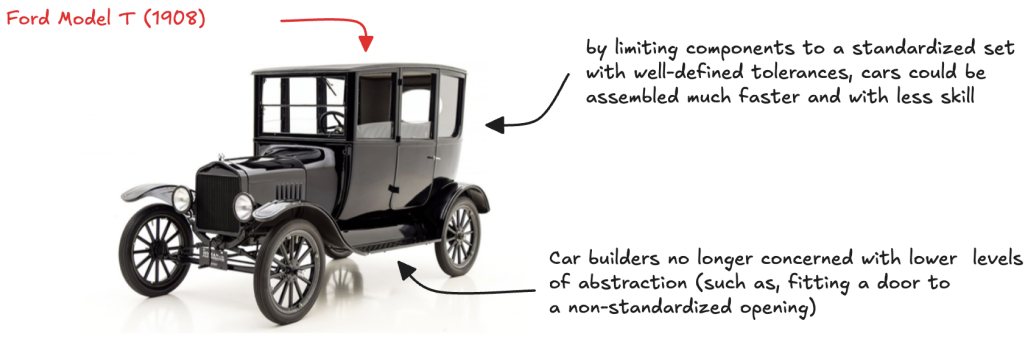
This hierarchical approach means that at each level **we can focus only on the concepts that matter**, while trusting that the lower layers will behave as specified. Abstraction is therefore the essential tool that enables engineers to design, understand, and use digital systems without drowning in unnecessary details.

1.1.1 Discipline

In order to manage complexity effectively, engineers must exercise **discipline** in their design processes. This means **intentionally restricting the design choices available**, so that they can work more productively at a higher level of abstraction. Without discipline, systems would become too complex to design, maintain, or scale.

A classic example of this principle comes from the automotive industry. Before 1908, cars were handcrafted by skilled workers. **Each vehicle was unique, and parts were often adjusted individually to fit together**. This approach was not only **time-consuming**, but also **expensive**, since production depended heavily on craftsmanship and could not easily be scaled.

Henry Ford revolutionized car manufacturing with the introduction of the Model T (1908). Instead of relying on handcrafted uniqueness, he focused on mass production.



Two key innovations were critical:

1. **Interchangeable parts**: components were manufactured according to standardized dimensions and tolerances. This meant that a door or a wheel from one car could fit perfectly onto another car of the same model.

2. **Moving assembly lines**: cars were assembled in a systematic process, where workers specialized in performing a small number of tasks repeatedly. This made production faster, cheaper, and less dependent on individual skill.

By enforcing these **restrictions**, Ford introduced discipline into car design and manufacturing. As a result, car builders no longer had to worry about **low-level issues** such as adjusting each door to a non-standard opening. Instead, they could focus on **higher-level goals**: efficiency, reliability, and cost reduction.

Ford's approach became so influential that he is often quoted with his famous saying: "Any customer can have a car painted any colour that he wants so long as it is black".

This statement illustrates the idea of limiting choices for the sake of productivity. While customers had fewer options, the standardized approach allowed the industry to achieve unprecedented levels of mass production.

In digital systems, we apply the same logic. By **restricting ourselves to well-defined design rules and abstractions**, we can manage complexity and build systems with millions or billions of transistors. Without such discipline, it would be impossible to coordinate the design of modern digital circuits, just as it would have been impossible to build affordable cars without Ford's innovations.

1.1.2 The three -y

A concrete method to apply the principles of abstraction and discipline in digital design is the use of the **three-y** framework: Hierarchy, Modularity, and Regularity. These principles provide a systematic way to design large systems by breaking them down into simpler, more manageable parts.

1. **Hierarchy** means dividing a system into modules, and then further subdividing those modules into smaller pieces until each part is easy to understand. By organizing a design into **layers of detail**, engineers can focus on one level at a time without being overwhelmed by the full complexity.
2. **Modularity** requires that each module has a clear and well-defined function, along with precise interfaces to connect with other modules. This ensures that modules can be designed, tested, and reused independently, without causing unexpected side effects when integrated into a larger system.
3. **Regularity** emphasizes uniformity and reuse. When common modules are used repeatedly, the number of distinct designs that must be created is reduced. This not only saves

design effort, but also improves reliability, since the same module can be tested and verified once and then applied many times.

The Ford Model T illustrates these three principles very clearly. A car can be broken down into major components like the chassis, engine, and seats. The engine itself contains smaller parts such as cylinders, carburetor, and cooling system. The carburetor can be further divided into fuel and air intakes, a throttle, and so forth. This recursive breakdown mirrors the idea of hierarchical decomposition in design.Consider the coupling nut that holds the fuel feed line to the intake elbow. Its diameter, thread pitch, and torque are standardized. Because its function and interface are well-defined, the nut can be manufactured, replaced, and tightened with a standard wrench. This is a perfect analogy for modularity. A standardized nut can be purchased from many different suppliers, as long as it meets the correct specification. This shows how reuse and uniformity reduce the need to reinvent components, saving time and resources.

In digital design, these same principles apply. A microprocessor, for example, is organized hierarchically into modules such as arithmetic circuits, memory units, and control logic. Each module is defined with clear interfaces, ensuring modularity. And regularity ensures that structures like logic gates or memory cells can be replicated billions of times on a chip without requiring unique designs for each instance.

By applying Hierarchy, Modularity, and Regularity, engineers can build systems of enormous complexity in a manageable, reliable, and efficient way.

1.1.3 Digital Electronics

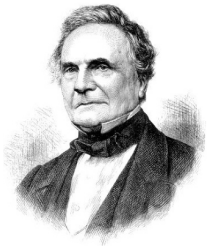
In digital electronics those concepts are applied through the decision to represent information with **discrete values** rather than **continuous ones**.

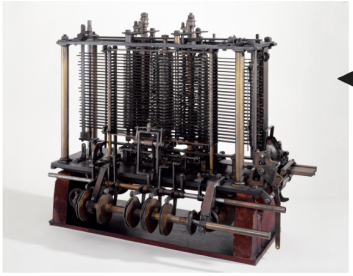
In an analog circuit, voltages can vary continuously over a range of values. While this can describe physical phenomena with great precision, it also makes circuits harder to design and more sensitive to noise. By contrast, digital circuits restrict voltages to discrete ranges that represent logical values. This restriction makes digital circuits a subset of analog circuits, but one that is much simpler to design and reason about.

This simplification is extremely powerful. By limiting ourselves to digital circuits, we can combine components into sophisticated systems that **ultimately outperform their analog counterparts** in many applications. The shift from analog to digital technologies has transformed our world: televisions, cameras, and telephones that were once analog are now almost entirely digital, with improved performance, flexibility, and reliability.

Digital systems thus represent information with **discrete-valued variables**. An early attempt to harness this idea can be found in Charles Babbage's Analytical Engine (1834–1871):

Charles Babbage's Analytical Engine (1834–1871)





A mechanical device to perform calculations. It represented numbers using discrete positions of gears, reducing continuous mechanical motion to a finite set of states that could be combined to perform complex calculations.

Although mechanical, it embodied a digital representation: gears with ten discrete positions labeled 0 through 9. This illustrates that the abstraction of information into discrete states predates electronics, even if practical digital systems only became feasible with the advent of electronic devices.

Modern electronic digital systems typically use a **binary representation**. In this scheme, a high voltage corresponds to a "1" and a low voltage corresponds to a "0". Each **binary digit** is called a **bit**. Why only two states? The answer is simplicity and robustness: it is much easier to distinguish between two voltage levels than between ten or more.

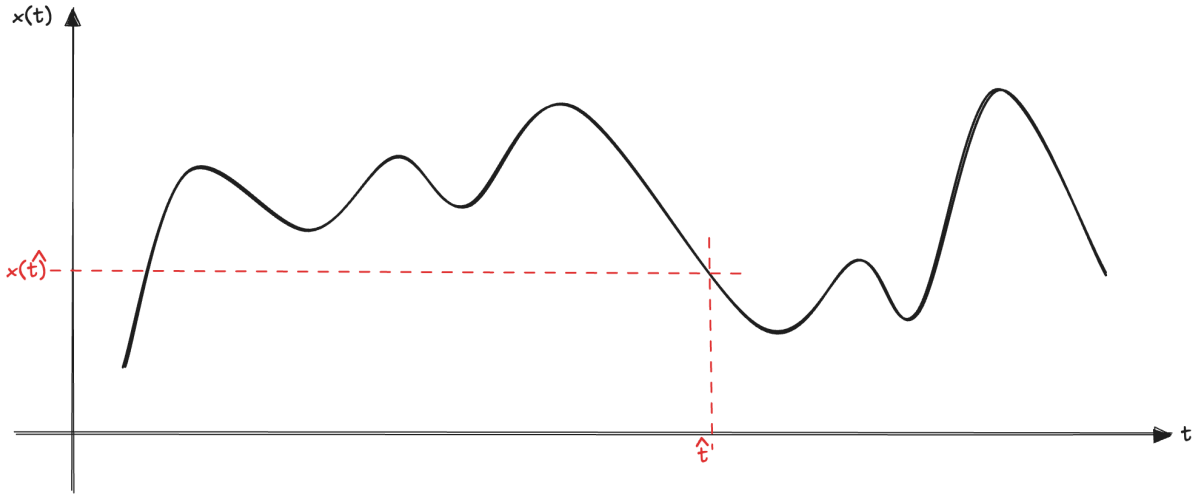
The digital abstraction, therefore, allows engineers to design, build, and understand extremely complex systems by thinking in terms of "0" and "1" rather than dealing with the infinite range of values found in the analog world.

1.2 Digitalizazion

Most physical phenomena **vary continuously over time**. For example, the temperature of a room, the intensity of light, or the force applied to an object can all take on an infinite number of values Mathematically, we can represent such a **signal** as a function:

$x(t)$

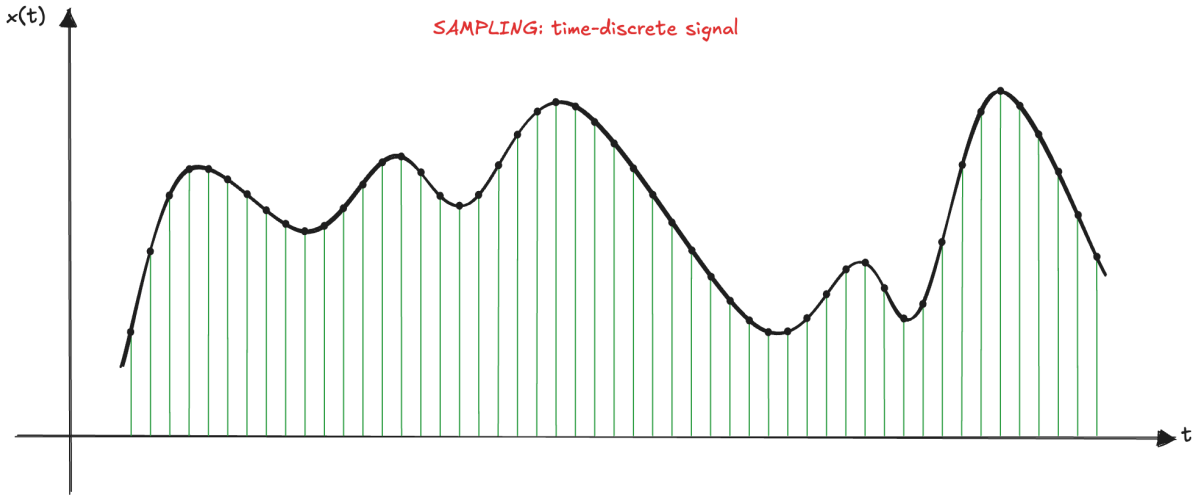
which is continuous in **time** and **amplitude**. Strictly speaking, at the quantum scale, nature is not perfectly continuous: energy, charge, and even light come in discrete packets (quanta). However, these quantum steps are so small compared to human-scale measurements that they can be safely ignored. For several practical purposes in engineering, signals like voltage, force, or temperature can be treated as continuous functions.

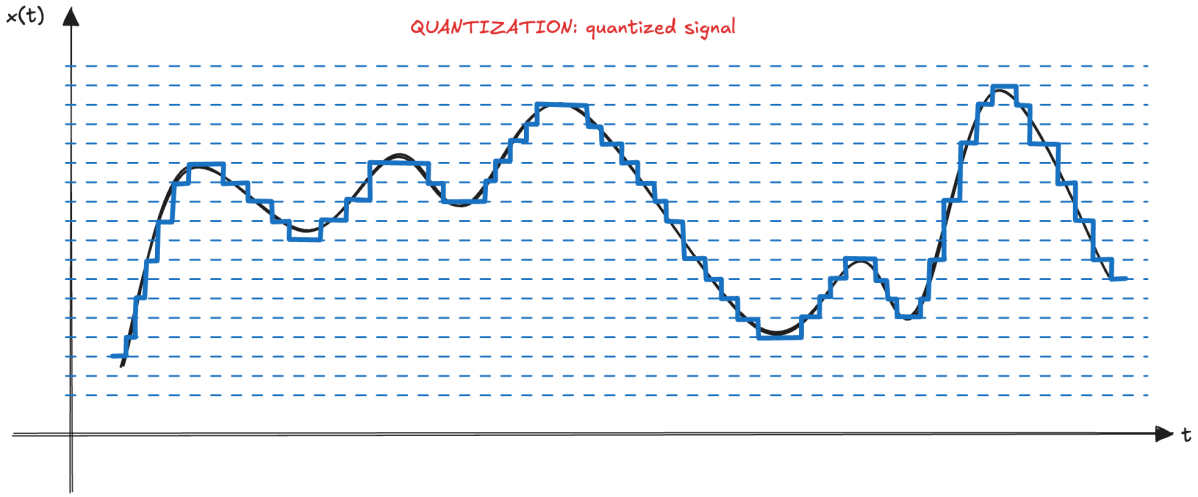


For describing nature, the analog representation seems the most natural and effective, because it preserves the continuity of the physical world. An analog voltage, for instance, can vary smoothly to reflect all possible values of a measured signal.

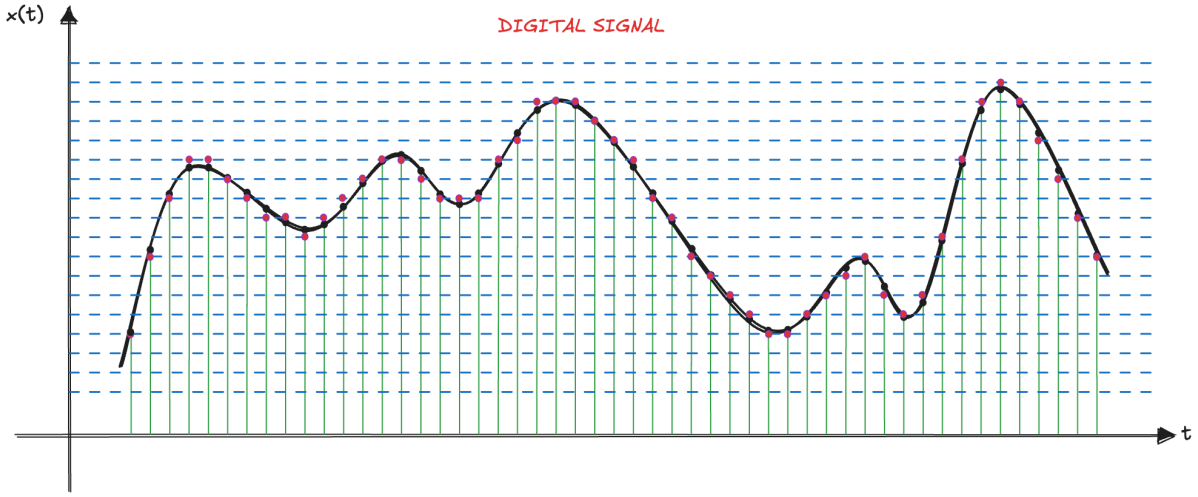
However, digital systems require us to work with discrete values in order to manage the complexity of processing and storing information. . To bridge this gap, we need a way to convert continuous quantities into digital ones. This is the role of an **Analog-to-Digital Converter** (**ADC**). In order to represent an analog signal digitally, we must digitize it in two dimensions:

- **in time**, by selecting specific instants at which the signal is measured (a process called **sampling**),
- **in amplitude**, by mapping each measured value to the nearest level in a finite set of possibilities (a process called **quantization**)





Through sampling and quantization, a continuous signal is transformed into a sequence of numbers that a digital circuit can store and process:



This digital representation makes it possible to apply algorithms, store data efficiently, and transmit information reliably, even though it always involves some degree of approximation compared to the original analog signal.

1.2.1 Sampling

Sampling is the process of reducing a continuous-time signal into a **discrete-time signal** by measuring its value at specific instants in time. A **sample** is one of these measured values. If the signal is measured every T seconds, then:

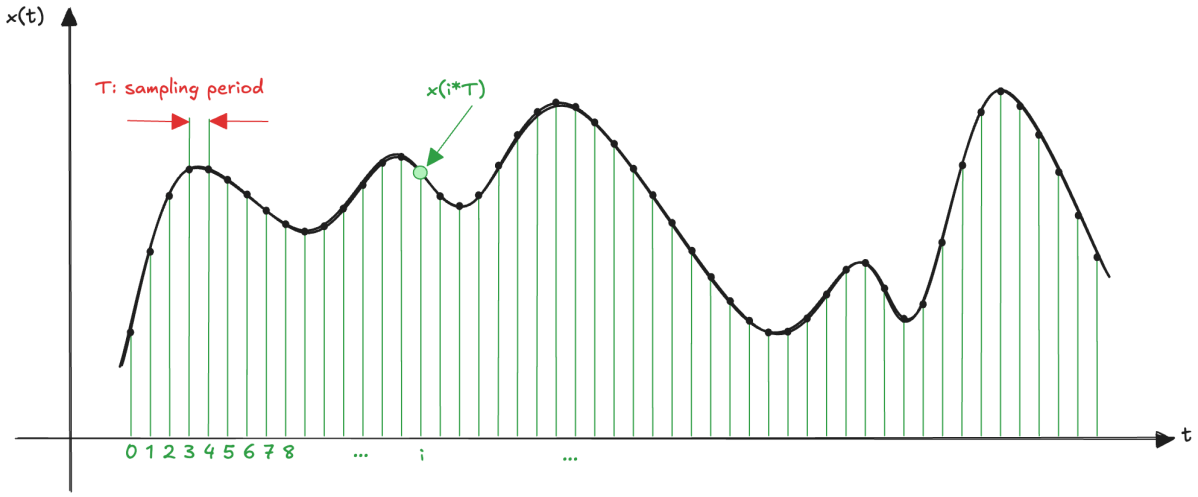
$t_i = i * T \quad \text{for } i = 0, 1, 2, \ldots$

and the corresponding samples are:

$\hat{x}[i] = x(t_i) = x(i * T)$

T is called the **sampling period** (the time between two consecutive samples) and its reciprocal is the **sampling frequency**.

$f_s = \frac{1}{T}$ [samples per second, or Hertz]



For example, if we measure the temperature of a room every T=1 second, then the sampling frequency is:

$f_s = \frac{1}{T} = \frac{1}{1\,\text{s}} = 1\,\text{Hz}$

If instead we sample every T=0.001s (1ms), the frequency becomes:

$f_s = \frac{1}{0.001\,\text{s}} = 1000\,\text{Hz}$

However, one key question arises: **which T should we use?** Choosing the right sampling period is essential. If the sampling frequency is too low, the discrete signal will not capture the variations of the continuous one, leading to loss of information (a phenomenon called aliasing). According to the **Nyquist–Shannon sampling theorem**, in order to reconstruct the signal without losing information, the sampling frequency must satisfy:

$f_s \geq 2f_{\text{max}}$

where f_{max} is the highest frequency present in the signal.

1.2.2 Quantization

The continuous range of amplitudes of a signal must be mapped to a finite set of discrete values. Formally, if the input signal has amplitude values in a range

$[x_{\min}, x_{\max}]$

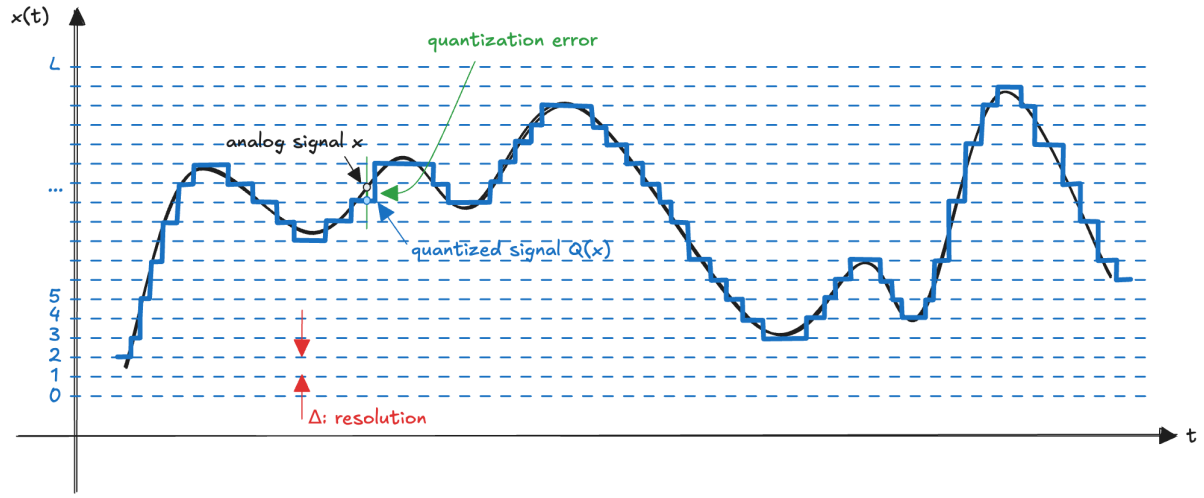
and we choose to represent it with L quantization levels, then each level corresponds to an interval of width:

$$\Delta = \frac{x_{\max} - x_{\min}}{L}$$

where Δ is called the **quantization step** size (or **resolution**).

Each measured value is then rounded to the nearest discrete level:

$$Q(x) = \text{round}\left(\frac{x - x_{\min}}{\Delta}\right) \cdot \Delta + x_{\min}$$



In this way, we introduces an error, since the quantized value is only an approximation of the true input:

$$e_q = x - Q(x)$$

The error is bounded by:

$$-\frac{\Delta}{2} \leq e_q \leq +\frac{\Delta}{2}$$

1.2.3 The amount of information

After the continuous signal is sampled in time and quantized in amplitude, the result is a **sequence of discrete** values that can be stored and processed by a digital system. But a natural

question arises: **how many discrete values are needed** to represent the information? To answer this, we must define the **amount of information** carried by a variable. This depends on the **size of the alphabet S** (the number of different symbols available, e.g. 2 for binary, 10 for decimal, 26 for letters) and on **the word length D** (how many symbols we place side by side to form one encoding).The total number N of distinct encodings we can create is:

S^D

If we want to represent a set of N distinct objects, we must ensure that:

$S^D \geq N$

In most cases, we know how many objects N we want to represent, and the size of the alphabet S (for example, binary symbols S=2) and we then need to find the **minimum word length** D, that we call **amount of information**:

$D \geq \log_S N$

When the alphabet is binary (S=2), the unit of measurement is the **bit** (**binary digit**):

$D = \log_2 N \quad [\text{bits}]$

The amount of information of a single binary variable is:

$D = \log_2 2 = 1 \text{ bit}$

To represent the 75 alphanumeric characters (uppercase, lowercase, digits, and some special characters), we need:

$\log_2 75 \approx 6.23$

which means at least 7 bits are required. This is why the original **American Standard Code for Information Interchange** (**ASCII**) was designed with 7 bits per character.

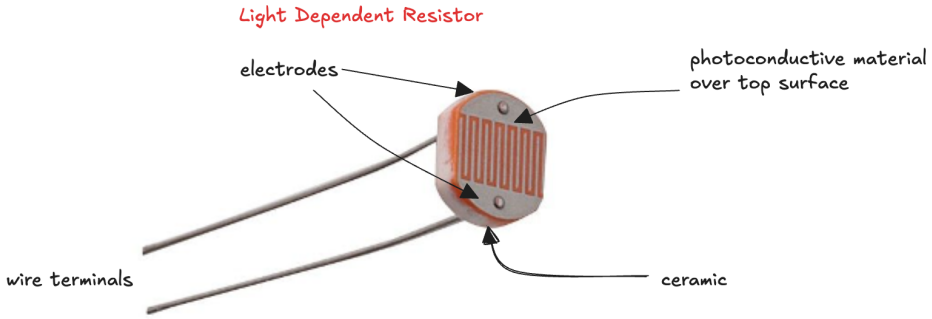
A continuous signal theoretically contains an **infinite amount of information**, since it can take an infinite number of values. However, in practice, physical limitations such as noise and measurement error restrict the effective resolution. For most continuous signals, this practical information content is limited to about 10 to 16 bits.

By quantifying information in this way, we establish a link between the abstract concept of information and the physical requirements of digital systems.

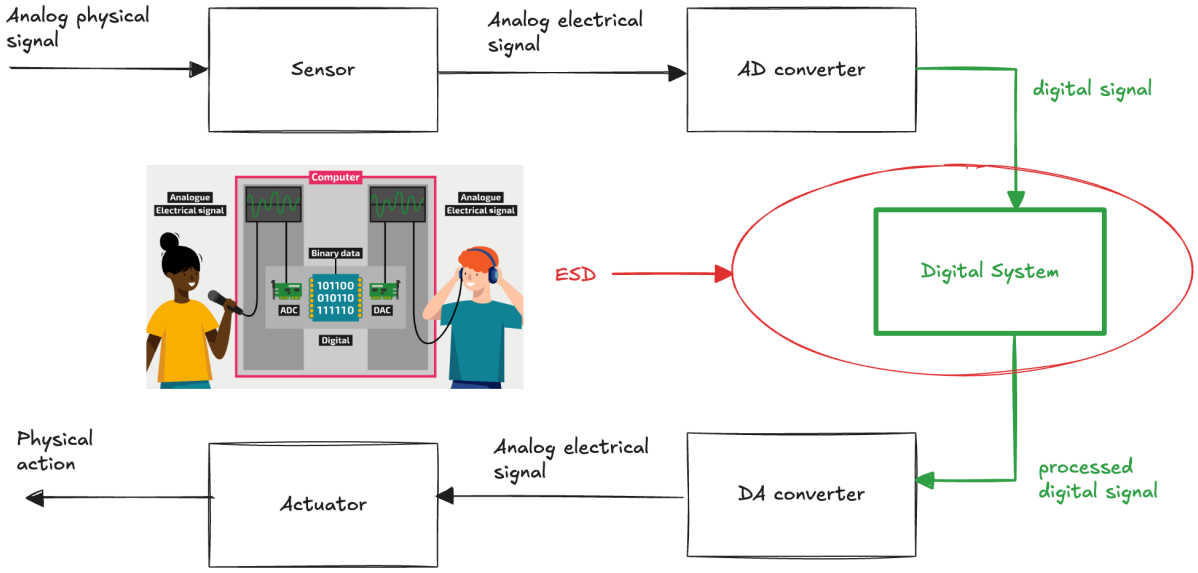
1.3 Interaction with the Physical World

To interact with **the physical world**, digital systems need devices that can sense physical phenomena and produce electrical signals that represent them. These devices are called **sensors**. A sensor converts one form of energy into another. It takes a non-electrical signal (like heat, light,

pressure, or motion) and transforms it into an electrical signal that can be processed. For example, a light sensor is made of a light-sensitive resistor, often called an LDR (Light Dependent Resistor). Its resistance decreases as the intensity of light increases. This change in resistance can be measured as a voltage, providing an electrical representation of the light level:



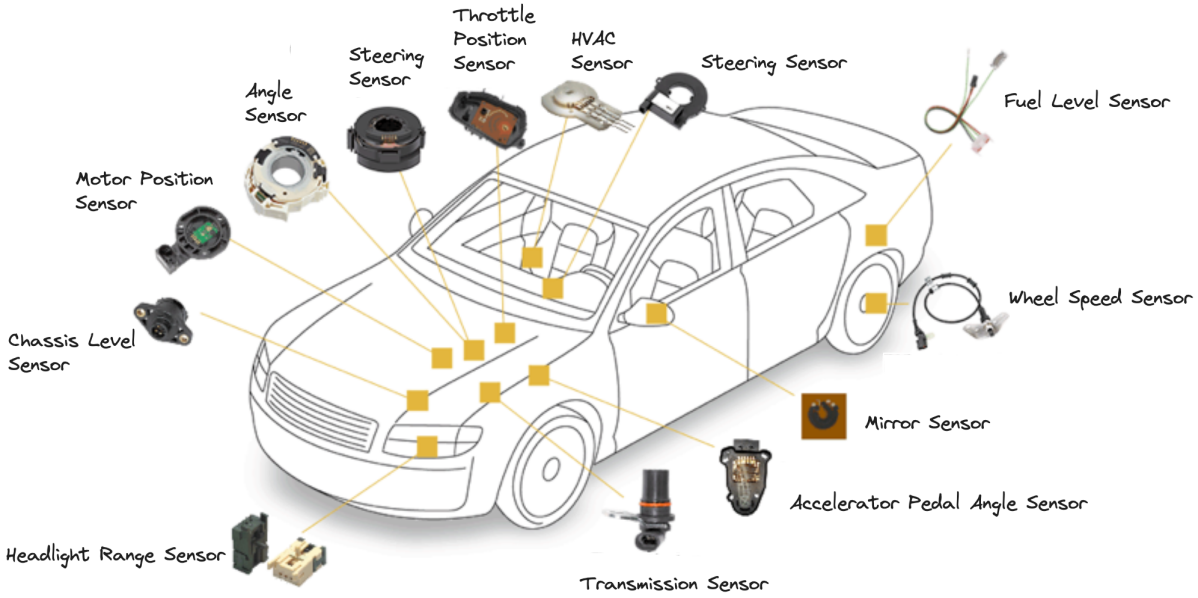
Once a sensor produces an analog signal, the system must digitize it to use it in a digital system. This involves an **Analog-to-Digital Converter (ADC)**, which transforms the continuous voltage into a sequence of digital values. These values can then be processed by a digital processing unit such as a processor or a custom designed digital circuit. Then the processed signal needs to be converted back to the real world (for example, to drive a loudspeaker, a motor, or a display) and a **Digital-to-Analog Converter (DAC)** is used to transform the digital output into a continuous analog signal again that can be used by an actuator to perform a physical action:



For example, modern cars are full of sensors:

- Temperature sensors for engine control

- Position sensors for throttle, accelerator pedal, and steering
- Speed sensors for wheels and transmission
- Light and rain sensors for headlights and wipers
- Pressure sensors for tires and fuel systems
- ....



All these sensors produce analog signals that are digitized, processed, and used by the vehicle's electronic control units to optimize performance, improve safety, and enhance comfort.

1.4 Boolean Algebra

When working with digital systems, it is often more convenient to leave behind the messy physical world of voltages, currents, and transistors, and instead **operate in the clean and abstract world of mathematics**. The most important mathematical tool for digital design is **Boolean algebra**, named to honor the mathematician **George Boole**, who first defined its principles in the mid-19th century.

George Boole, 1815–1864

Born to working-class parents and unable to afford a formal education, Boole taught himself mathematics and joined the faculty of Queen's College in Ireland. He wrote "An Investigation of the Laws of Thought" (1854), which introduced binary variables and the three fundamental logic operations: *AND*, *OR*, and *NOT*

Boolean algebra is a form of **discrete mathematics** that deals with variables that can take only two possible values. In digital electronics, these values are naturally associated with the binary digits 0 and 1, which correspond to the two allowed voltage ranges in a digital circuit. **This correspondence between mathematics and hardware is the essence of the digital abstraction**: once we agree that ”low voltage” means 0 and ”high voltage” means 1, we can reason about circuits entirely in terms of 0s and 1s, **without worrying about electrons or physical devices**.

The beauty of Boolean algebra is that it frees designers from implementation details. A Boolean variable may be physically represented by:

- a voltage level in a transistor circuit,
- the presence or absence of a gear tooth in a mechanical system,
- or even the level of liquid in a hydraulic device.

From the perspective of Boolean algebra, all of these are equivalent, as long as the system consistently distinguishes between two states. This means that **an hardware designer can use Boolean algebra to model, analyze, and optimize circuits**. For example, Boolean expressions can be simplified using algebraic laws such as De Morgan's Theorems or the Distributive Law, which help reduce the number of gates needed in a design. The same principle that allows a programmer to remain abstract also empowers a hardware engineer to manipulate logical relationships efficiently.

It is important to recognize, however, that abstraction is a two-edged sword. While Boolean algebra allows us to forget about the messy analog world, **understanding some of those details can still be useful**. For example, a circuit designer who understands transistor-level characteristics can optimize power consumption or speed.

In summary, Boolean algebra is the **mathematical foundation of digital design**. It gives us a simple language of 0s and 1s, and a set of algebraic rules, that make it possible to design and reason about digital systems at a high level of abstraction. Without it, the complexity of modern electronics (millions or billions of transistors in a single microprocessor) would be unmanageable.

1.5 Exercises

1 - An analog voltage is in the range of 0–5V. If it can be measured with an accuracy of $\pm$50 mV, at most how many bits of information does it convey?

According to the text we know:

- the analog voltage in range: 0V to 5V
- the accuracy is $\pm$50mV = $\pm$0.05V

This means the smallest resolvable interval is Δ V = 0.1V (because $\pm$0.05V corresponds to a resolution step of 0.1V)

We count how many distinct intervals fit in the range.

$N = \frac{5-0}{0.1} = 50$

So we can distinguish 50 different levels. Then we ompute the number of bits, using the amount of information definition:

$D = \log_2 N = \log_2 50 \approx 5.64$bits

The analog voltage conveys about 5.64 bits of information.

2 - A classroom has an old clock on the wall whose minute hand broke off. (a) If you can read the hour hand to the nearest 15 minutes, how many bits of information does the clock convey about the time? (b) If you know whether it is before or after noon, how many additional bits of information do you know about the time?

According to the text we know:

- one hour is divided into 4 segments of 15 minutes
- in 12 hours, the hand can point to: 12 * 4 = 48 distinct positions

The number of bits of information is: $D = \log_2 48 \approx 5.58$bits

If we also know whether it is before or after noon, that doubles the number of possibilities: 48 * 2 = 96

So the extra information is:

$D = \log_2 96 - \log_2 48 = \log_2 \left(\frac{96}{48}\right) = \log_2 2 = 1$ bit

The clock conveys about 5.58 bits and knowing AM/PM gives 1 additional bit.