# 01_introduccion_sistemas_numeracion

June 19, 2026

# 1 Electrónica Digital — Tema 1: Introducción y Sistemas de Numeración

Apuntes de estudio · Grado en Ing. Electrónica, Robótica y Mecatrónica (GIERM) · Universidad de Sevilla

Basado en las diapositivas oficiales *ED1 — Introducción a la Electrónica Digital* y en el material de fundamentos del tema.

**Qué cubre este cuaderno:** 1. Señales analógicas vs. digitales y digitalización. 2. Valores lógicos, códigos y el bit como unidad de información. 3. Sistemas de numeración: binario, octal, hexadecimal. 4. Conversiones entre bases. 5. Complemento a 1 y complemento a 2 (representación de negativos). 6. Aritmética binaria (suma, resta, multiplicación, overflow). 7. Códigos: BCD, Gray, ASCII.

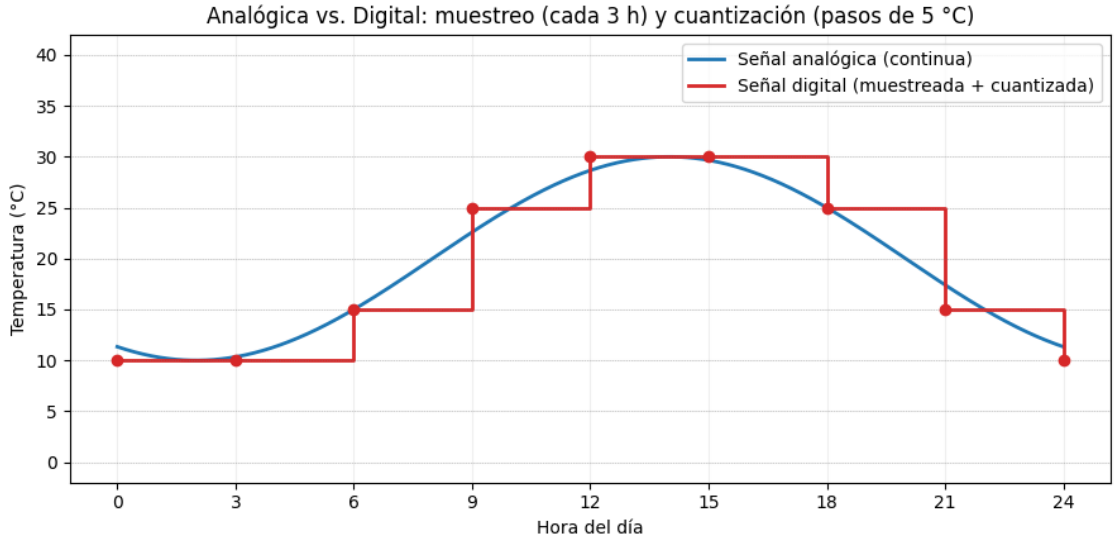
Las celdas de código son **autocontenidas** (solo `numpy` y `matplotlib`) y generan figuras para visualizar los conceptos.

## 1.1 1. Señales analógicas y digitales

| | Señal **analógica** | Señal **digital** |
|---|---|---|
| Valores | Continuos (números reales, $\in \mathbb{R}$) | Discretos, dentro de un conjunto predefinido |
| Ejemplo | Temperatura real a lo largo del día | Temperatura redondeada a $\{0, 5, 10, \ldots, 40\}$ °C |
| Ruido | Sensible | Inmune (mientras el ruido no cruce el umbral) |
| Almacenamiento | Difícil | Fácil (basta guardar ceros y unos) |

**Ventajas de las señales digitales** (diapositivas oficiales): facilidad de diseño, precisión, programabilidad, inmunidad frente al ruido, facilidad de almacenamiento, integrabilidad, transmisión y coste.

Un sistema mixto típico es **A/D → procesado digital → D/A**: el mundo físico es analógico, se convierte a digital para procesarlo y se vuelve a convertir a analógico para actuar.



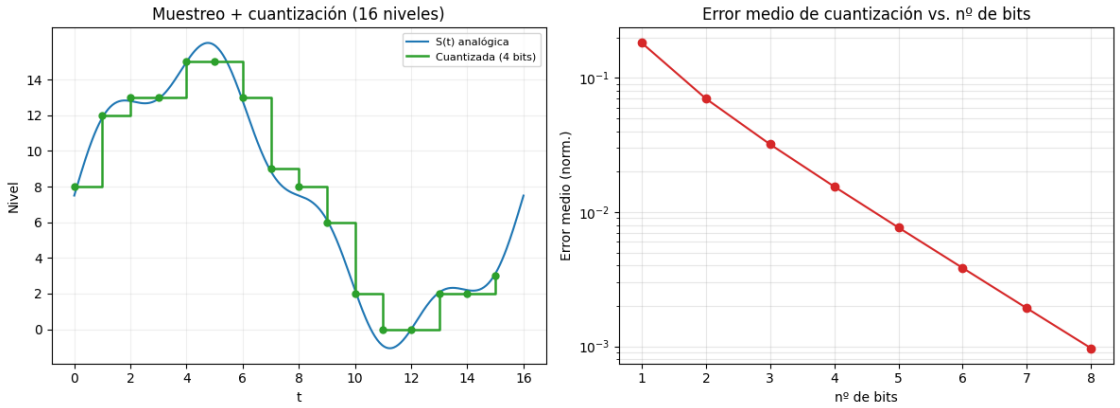

## 1.2 2. Digitalización: muestreo y cuantización

Convertir una señal analógica en digital tiene **dos pasos**:

1. **Muestreo** (sampling): tomar el valor de la señal en instantes discretos (eje horizontal → tiempo discreto).
2. **Cuantización** (quantization): aproximar cada muestra al nivel discreto más cercano (eje vertical → amplitud discreta).

Cada nivel cuantizado se codifica en **binario**. Con $n$ bits hay $2^n$ niveles. En la diapositiva oficial, una señal se digitaliza con 4 bits → 16 niveles (`0000` … `1111`), produciendo un **código binario** $C(n)$.

El **error de cuantización** es la diferencia entre el valor real y el nivel asignado: cuantos más bits, menor error (más resolución).



```
Código binario de las muestras (4 bits):
1000, 1100, 1101, 1101, 1111, 1111, 1101, 1001, 1000, 0110, 0010, 0000, 0000,
0010, 0010, 0011
```

## 1.3 3. Valores lógicos, códigos y el bit

- Una variable digital binaria toma dos valores: **0** y **1**.
- **Lógica positiva**: nivel alto $V_H \to 1$, nivel bajo $V_L \to 0$. **Lógica negativa**: al revés.
- Un **bit** (binary digit) es la unidad mínima de información.
- Un **código** establece una correspondencia *biunívoca* entre cada estado del sistema y una combinación de cifras (p. ej. `101101` → "arrancar motor").

**Regla fundamental:** con $n$ bits se forman $2^n$ combinaciones distintas.

| nº de bits ($n$) | combinaciones ($2^n$) |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 8 | 256 |
| 10 | 1024 |
| 16 | 65 536 |

## 1.4 4. Sistemas de numeración

Un número en base $b$ con dígitos $d_i$ vale (notación posicional):

$$N = \sum_i d_i \cdot b^{\,i}$$

donde $i$ es la posición (0 = el dígito menos significativo a la izquierda de la coma; negativos a la derecha de la coma).

| Sistema | Base $b$ | Dígitos válidos | Uso |
|---|---|---|---|
| Binario | 2 | 0, 1 | Lógica interna de los circuitos |
| Octal | 8 | 0–7 | Agrupa los bits de 3 en 3 |
| Decimal | 10 | 0–9 | El que usamos las personas |
| Hexadecimal | 16 | 0–9, A–F | Agrupa los bits de 4 en 4 (compacto) |

En hexadecimal: A=10, B=11, C=12, D=13, E=14, F=15.

### 1.4.1 Ejemplo resuelto — interpretar un número binario

$$\begin{aligned} 1011.01_{(2)} &= 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 1\cdot 2^{-2} \\ &= 8 + 0 + 2 + 1 + 0 + 0.25 = 11.25_{(10)} \end{aligned}$$

```
 DEC |   BIN |  OCT |  HEX
---------------------------
   0 |  0000 |    0 |    0
   1 |  0001 |    1 |    1
   2 |  0010 |    2 |    2
   3 |  0011 |    3 |    3
   4 |  0100 |    4 |    4
   5 |  0101 |    5 |    5
   6 |  0110 |    6 |    6
   7 |  0111 |    7 |    7
   8 |  1000 |   10 |    8
   9 |  1001 |   11 |    9
  10 |  1010 |   12 |    A
  11 |  1011 |   13 |    B
  12 |  1100 |   14 |    C
  13 |  1101 |   15 |    D
  14 |  1110 |   16 |    E
  15 |  1111 |   17 |    F
```

## 1.5 5. Conversiones entre bases

### 1.5.1 5.1 De cualquier base → decimal

Sumar cada dígito por su peso $b^i$ (notación posicional). Visto arriba con `1011.01`.

### 1.5.2 5.2 Decimal → binario (parte entera): divisiones sucesivas

Dividir entre 2 repetidamente; los **restos leídos de abajo arriba** son el número binario.

**Ejemplo:** $77_{(10)} \rightarrow$ **binario**

| División | Cociente | Resto |
|---|---|---|
| 77 / 2 | 38 | **1** |
| 38 / 2 | 19 | **0** |
| 19 / 2 | 9 | **1** |
| 9 / 2 | 4 | **1** |
| 4 / 2 | 2 | **0** |
| 2 / 2 | 1 | **0** |
| 1 / 2 | 0 | **1** |

Leyendo los restos de abajo arriba: $77_{(10)} = 1001101_{(2)}$.

### 1.5.3 5.3 Decimal → binario (parte fraccionaria): multiplicaciones sucesivas

Multiplicar la fracción por 2; la **parte entera** del resultado es el siguiente bit (leídos de arriba abajo).

**Ejemplo:** $0.625_{(10)} \rightarrow 0.625 \times 2 = 1.25$ (**1**), $0.25 \times 2 = 0.5$ (**0**), $0.5 \times 2 = 1.0$ (**1**) $\rightarrow 0.101_{(2)}$.

### 1.5.4 5.4 Atajo binario  octal  hexadecimal

- **Binario → octal**: agrupar los bits **de 3 en 3** desde la coma.
- **Binario → hexadecimal**: agrupar **de 4 en 4** desde la coma.

$$\underbrace{1001101}_{} = \underbrace{001}_{1}\ \underbrace{001}_{1}\ \underbrace{101}_{5} = 115_{(8)} \qquad \underbrace{0100}_{4}\ \underbrace{1101}_{D} = 4D_{(16)}$$

```
Convertir 77 a binario (divisiones sucesivas entre 2):
  77 / 2 = 38  resto 1
  38 / 2 = 19  resto 0
  19 / 2 = 9  resto 1
  9 / 2 = 4  resto 1
  4 / 2 = 2  resto 0
  2 / 2 = 1  resto 0
  1 / 2 = 0  resto 1
  -> leyendo restos de abajo arriba: 1001101
\nComprobación: 0b1001101 = 77
Octal de 77  = 115   Hex de 77 = 4D
\nAgrupación de 3 (octal):  1001101 -> 001 001 101 = 115
Agrupación de 4 (hex):    1001101 ->  0100 1101 = 4D
```

## 1.6 6. Representación de números negativos: complementos

Los circuitos solo manejan 0 y 1, así que el **signo** debe codificarse también en bits. Tres métodos clásicos (sobre $n$ bits):

### 1.6.1 6.1 Signo-magnitud

El bit más significativo (MSB) es el signo ($0 = +$, $1 = -$); el resto es la magnitud. Problema: hay **dos ceros** ($+0$ y $-0$) y la aritmética es incómoda.

### 1.6.2 6.2 Complemento a 1 (Ca1)

Se obtiene **invirtiendo todos los bits** del positivo. También tiene doble cero.

$$C_1(N) = (2^n - 1) - N \quad \equiv \quad \text{invertir cada bit}$$

### 1.6.3 6.3 Complemento a 2 (Ca2) — el estándar

Es el complemento a $1$ **+ 1**. Es el que usan los procesadores porque la resta se convierte en suma y solo hay **un cero**.

$$C_2(N) = C_1(N) + 1 = 2^n - N$$

**Método rápido del Ca2:** copiar los bits desde la derecha hasta el primer `1` incluido, e invertir el resto.

**Ejemplo (8 bits): representar** $-20$ - $+20 = 0001\,0100$ - Ca1: invertir $\rightarrow 1110\,1011$ - Ca2: $+1$ $\rightarrow 1110\,1100$

En Ca2 con $n$ bits el rango es $[-2^{n-1},\ 2^{n-1}-1]$. Con 8 bits: $[-128, +127]$.

```
+20  en 8 bits         = 00010100
Ca1 (invertir)        = 11101011
Ca2 (Ca1 + 1) = -20    = 11101100
\nTabla Ca2 en 4 bits (cómo se interpreta cada patrón):
 patrón | valor (Ca2)
----------------------
   0000 |           0
   0001 |           1
   0010 |           2
   0011 |           3
   0100 |           4
   0101 |           5
   0110 |           6
   0111 |           7
   1000 |          -8
   1001 |          -7
   1010 |          -6
   1011 |          -5
   1100 |          -4
   1101 |          -3
   1110 |          -2
   1111 |          -1
```

## 1.7 7. Aritmética binaria

### 1.7.1 7.1 Suma binaria

Reglas bit a bit (con acarreo $C$):

| a | b | C_in | Suma | C_out |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Ejemplo:** $0110\,(6) + 0101\,(5) = 1011\,(11)$.

### 1.7.2 7.2 Resta como suma en Ca2

$A - B = A + C_2(B)$. Se suma el complemento a 2 del sustraendo y se **descarta el acarreo final**.

**Ejemplo (8 bits):** $50 - 20 \rightarrow 50 + C_2(20) = 0011\,0010 + 1110\,1100 = (1)\,0001\,1110 = 30$ (se ignora el 1 de acarreo).

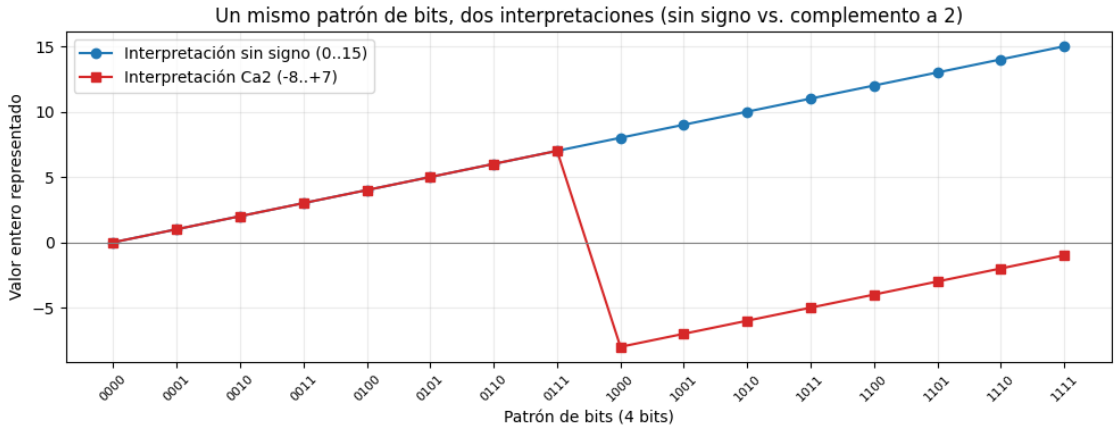
### 1.7.3 7.3 Overflow (desbordamiento)

En Ca2 hay overflow cuando se suman dos números del **mismo signo** y el resultado sale con signo contrario (el acarreo entrante y saliente del bit de signo difieren). Distinto del acarreo final en sumas sin signo.

### 1.7.4 7.4 Multiplicación binaria

Igual que la decimal pero con desplazamientos y sumas: $101 \times 11 = 1111$ ($5 \times 3 = 15$).

```
Suma 6 + 5:
  0110  (= 6)
+ 0101  (= 5)
= 1011  (= 11)
\nResta 50 - 20 como 50 + Ca2(20) en 8 bits:
  Ca2(20) = 11101100
  50 + Ca2(20) = 00011110 = 30 (descartado el acarreo)
\nMultiplicación 5 x 3:
  101 x 011 = 1111 = 15
```




## 1.8 8. Códigos binarios

No todo código binario es "el número en base 2". Según el uso se eligen códigos con propiedades concretas.

### 1.8.1 8.1 BCD (Binary Coded Decimal)

Cada **dígito decimal** se codifica por separado con 4 bits (su valor binario 0000–1001). Las combinaciones 1010–1111 **no se usan**.

$$57_{(10)} \rightarrow \underbrace{0101}_{5}\ \underbrace{0111}_{7} = 0101\,0111_{(BCD)}$$

Ventaja: conversión directa para *displays* de 7 segmentos. Inconveniente: desperdicia combinaciones y la aritmética requiere corrección.

### 1.8.2 8.2 Código Gray

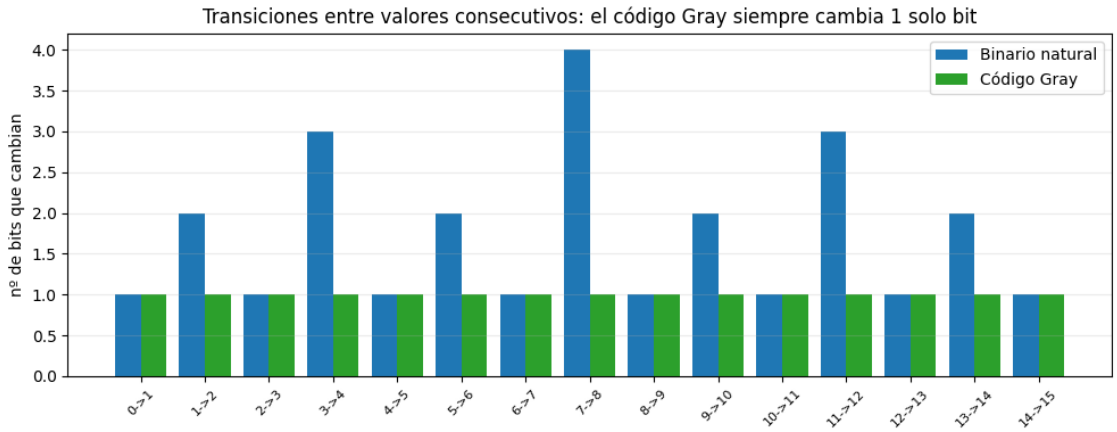
Código binario reflejado en el que **dos valores consecutivos difieren en un solo bit**. Minimiza errores en codificadores de posición (encoders) y en mapas de Karnaugh.

Conversión binario → Gray: $g_i = b_{i+1} \oplus b_i$ (el MSB se copia).

### 1.8.3 8.3 ASCII

Código de 7 bits (128 símbolos) que asigna un patrón a cada carácter: letras, dígitos, signos y caracteres de control. Ej.: `'A'` = 65 = `100 0001`, `'0'` = 48 = `011 0000`. Extensiones de 8 bits llegan a 256 símbolos.

```
BCD:
   57 -> 0101 0111
    9 -> 1001
  482 -> 0100 1000 0010
\nCódigo Gray (3 bits) - fíjate que solo cambia 1 bit entre filas consecutivas:
dec | binario |  gray
----------------------
  0 |     000 |   000
  1 |     001 |   001  (cambió 1 bit)
  2 |     010 |   011  (cambió 1 bit)
  3 |     011 |   010  (cambió 1 bit)
  4 |     100 |   110  (cambió 1 bit)
  5 |     101 |   111  (cambió 1 bit)
  6 |     110 |   101  (cambió 1 bit)
  7 |     111 |   100  (cambió 1 bit)
\nASCII de la cadena 'Dig0':
  'D' -> dec  68  bin 01000100  hex 44
  'i' -> dec 105  bin 01101001  hex 69
  'g' -> dec 103  bin 01100111  hex 67
  '0' -> dec  48  bin 00110000  hex 30
```



## 1.9 9. Resumen para el examen

- **Analógico** = continuo; **digital** = discreto (muestreo + cuantización). Con $n$ bits $\rightarrow$ $2^n$ niveles.
- **Posicional:** $N = \sum d_i b^i$. Sirve para pasar cualquier base a decimal.
- **Decimal→binario:** entero por divisiones sucesivas (restos de abajo arriba); fracción por multiplicaciones sucesivas (enteros de arriba abajo).
- **Atajos:** binario octal en grupos de 3; binario hex en grupos de 4.
- **Ca2** = Ca1 + 1 = método estándar para negativos; resta = suma del Ca2; rango $[-2^{n-1}, 2^{n-1} - 1]$; ojo al **overflow** (signos iguales $\rightarrow$ signo distinto).
- **Códigos:** BCD (cada dígito decimal en 4 bits), Gray (1 bit cambia entre consecutivos), ASCII (7 bits, caracteres).

**Errores típicos:** confundir acarreo final (sin signo) con overflow (Ca2); olvidar el `+1` del Ca2; agrupar octal/hex sin partir desde la coma; usar combinaciones inválidas en BCD (1010–1111).