# Apuntes de Lógica Digital

Daniel Araya Román

2023-05-08

# 1. Sistemas Binarios

## 1.1 Sistemas Digitales

En los sistemas digitales electrónicos actuales, las señales emplean sólo dos valores discretos → *binarios*. Un dígito binario, llamado **bit**, que puede tomar los valores 0 y 1. Un sistema digital es una interconexión de módulos digitales. Para entender como funciona cada módulo digital, se necesiatan conocimientos básicos de circuitos digitales y de su función lógica.

Un lenguaje importante para el diseño digital es el **(HDL, Hardware Description Language)**. Sirve para simular sistemas digitales y verificar su funcionamiento antes de crearlos en hardware.

**Sistema digitales**

↓

**Manejan cantidades discretas de información**

↓

**Representadas en forma de binario**

## 1.2 Números Binarios

El número decimal 7392, contiene potencias de 10 que están implícitas en la posición de los coeficientes, e.g:

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

Un número con punto decimal se representa con una serie de coeficientes, así:

$$a_5 a_4 a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2} a_{-3}$$

los coeficientes $a_j$ son cualesquiera de los 10 dígitos (0...9); el valor del subíndice $j$ indica la posición, y la potencia de 10 que se deberá multplicar ese coeficiente. De modo que:

$$10^5 a_5 + 10^4 a_4 + 10^3 a_3 + 10^2 a_2 + 10^1 a_1 + 10^0 a_0 + 10^- 1 a_{-1} + 10^- 2 a_{-2} + 10^- 3 a_{-3}$$

El sistema binario es distinto al decimal, sus coeficientes solo pueden tener 2 valores, 0 o 1. Cada coeficiente $a_j$ se multiplica por $2^j$. 11010.11 es 26. 75 en decimal, porque:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$$

En general, un número expresado en un sistema de base **r** consiste en coeficientes que se multiplican por potencias de **r**:

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + ... + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + ... + a_{-m} \cdot r^{-m}$$

## 1.4 Números Octales y Hexadecimales

Las conversiones entre binario, octal y hexadecimal son importantes en las computadoras digitales. Puesto que $2^3 = 8$ y $2^4 = 16$, cada dígito octal corresponde a **tres** dígitos binarios, y cada dígito hexadecimal corresponde a **cuatro** dígitos binarios.

*Binario* $\rightarrow$ *octal*: agrupando los dígitos binarios de 3 en 3, de derecha a izquierda, y reemplazando cada grupo por su equivalente octal.

$$(10\,110\,001\,101\,011 \cdot 111\,100\,000\,110)_2 = (26153.7406)_8$$

*Binario* $\rightarrow$ *hexadecimal*: agrupando los dígitos binarios de 4 en 4, de derecha a izquierda, y reemplazando cada grupo por su equivalente hexadecimal.

$$(10\,1100\,0110\,1011 \cdot 1111\,0010)_2 = (2C6B.F2)_{16}$$

Cuando se habla de binario es más deseable expresarlo en términos de números octales o hexadecimales, porque son más compactos y fáciles de leer. Así $(111\,111\,111\,111)_2$ este número en binario de 12 bits, se puede escribir como $(7777)_8$ en octal o $(FFF)_{16}$ en hexadecimal.

| Decimal (base 10) | Binary (base 2) | Octal (base 8) | Hexadecimal (base 16) |
|---|---|---|---|
| 0 | 0000 | 00 | 0 |
| 1 | 0001 | 01 | 1 |
| 2 | 0010 | 02 | 2 |
| 3 | 0011 | 03 | 3 |
| 4 | 0100 | 04 | 4 |
| 5 | 0101 | 05 | 5 |
| 6 | 0110 | 06 | 6 |
| 7 | 0111 | 07 | 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 |

Table 1: Números en diferentes bases.

## 1.5 Complementos

En las computadoras digitales se usan complementos para simplificar la operación de resta y para efectuar manipulaciones lógicas. Existen dos tipos de complementos para cada sistema de base $r$: el *complemento a la base* y el *complemento a la base disminuida.* El primero se denomina complemento a $r$, mientras que el segundo es el complemento a $(r-1)$. Si se sustiyue el valor de la base $r$ en los nombres obtenemos que los dos tipos son el complemento a 2 y el complemento a 1.

### 1.5.1 Complemento a la base

El complemento a $r$ de un número $N$ de $n$ dígitos en base $r$ se define como:

$$r^n - N, \text{ para } N \neq 0, \text{ y } 0 \text{ para } N = 0.$$

Por ejemplo, el complemento a 10 de 1234 es $10^4 - 1234 = 8766$. De forma similar, el complemento a dos se forma dejando como están todos los ceros menos significativos y el primer uno, y sustituyendo los unos por ceros y los ceros por unos en las demás posiciones a la izquierda.

El complemento a dos de 1101100 es 0010100.
El complemento a dos de 0110111 es 1001001.

### 1.5.2 Complemento a la base disminuida

De igual manera, dado un número $N$ en base $r$ que tiene $n$ dígitos, el complemento a $(r-1)$ de $N$ se define como:

$$(r^n - 1) - N.$$

Con números decimales, $r = 10$ y $r - 1 = 9$, así el complemento a nueve de $N$ es $(10^n - 1) - N$. El $10^n$ representado por $n$ *nueves*. Por ejemplo, si $n = 4$, tenemos $10^4 = 10,000$ y $10^4 - 1 = 9999$. El complemento a nueve se consigue restando cada dígito a nueve. *e.g*:

Complemento a nueve de 546700 es 999999 - 546700 = 453299.
Complemento a nueve de 012398 es 999999 - 012398 = 987601.

Ahora con números binarios, $r = 2$ y $r - 1 = 1$, así el complemento a uno de $N$ es $(2^n - 1) - N$. En este caso $2^n$ se representa con un número binario que consite en un uno seguido de $n$ ceros.

$$n = 4, \quad 2^4 = 10000_2$$

Por otro lado $2^n - 1$ es un número binario representado por $n$ unos.

$$n = 4, \quad 2^4 - 1 = 1111_2$$

El complemento a uno se consigue invirtiendo cada dígito. El restar dígitos binarios a uno podemos tener $1 - 1 = 0$ y $1 - 0 = 1$. Cambiando el bit de 0 a 1 o de 1 a 0. *e.g*:

Complemento a uno de 101101 es 111111 - 101101 = 010010.
Complemento a uno de 011010 es 111111 - 011010 = 100101.

El complemento a $(r-1)$ de los números octales y hexadecimales se obtiene restando cada dígito a 7 y F, respectivamente.

### 1.5.3 Resta con complementos

La resta de dos números de $n$ dígitos sin signo, $M-N$, en base $r$ se realiza así:

1. $M + (r^n - N) = M - N + r^n$
2. Si $M \geq N$, la suma produce acarreo final. Quedando $M - N$.
3. Si $M < N$, la suma no produce acarreo final. Quedando $r^n - (N - M)$.

## 1.6 Números binarios con signo

Por limitaciones de hardware, las computadoras deben de representar todo con dígitos binarios. Por lo tanto, los números binarios con signo se representan con un bit en la posición más significativa que se usa para indicar el signo del número. la convención es que el bit sea cero si el número es positivo y uno si es negativo.

Por ejemplo la cadena de bits 01001 se considera como 9 (binario sin signo) o +9 (binario con signo). La cadena de bits 11001 se considera como 25 (binario sin signo) o -9 (binario con signo). A esto se le llama *convención de magnitud con signo*. Así:

$$(+ \text{ o } -) \rightarrow (0 \text{ o } 1)$$

## 1.7 Códigos binarios

Existe una analogía directa entre:

1. Señales binarias
2. Elementos binarios
3. Dígitos binarios

Los códigos binarios solo cambian los símbolos, no el significado de los elementos.

Un código binario de $n$ bits, es un grupo de $2^n$ combinaciones de $0's$ y $1's$. *****
Cada combinación representa a un elemento del conjunto codificado.

Las combinaciones de códigos de $n - bits$ se representan así:

$$C_e = [0 \quad ... \quad 2^n - 1]$$

El número mínimo para codificar $2^n$ elementos es $n$ bits. No existe un número máximo.

### 1.7.1 Código BCD

El código BCD → ***Binary-Coded Decimal*** es un código que almacena los dígitos decimales representandos de forma de dígitos binarios. Las computadoras solo entienden con valores binarios. Es posible crear distintos códigos binarios para representar $2^n$ combinaciones.

| Decimal | BCD |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

Table 2: Código BCD

Las combinaciones (1010 → 1111) no se usan y por lo tanto carecen de significado, en el código BCD.

Un ejemplo de código BCD comparado con binario y decimal:

$$(185)_{10} = (10111001)_2 = (0001\,1000\,0101)_{BCD}$$

Es importante reiterar que $BCD \neq Binario$, BCD = Números decimales representados en forma de bits.

### 1.7.2 Otros códigos decimales

**(BCD, 2421) → (Códigos ponderados):**

Asigna cada posición de bit ↔ Factor de ponderación (peso).

Cada dígito pueda evaluarse sumando los pesos de todos los $1's$ de la combinación codificada. Pesos de (BCD): $8, 4, 2, 1$.

**(2421, excess-3) → (Códigos autocompletadores):**

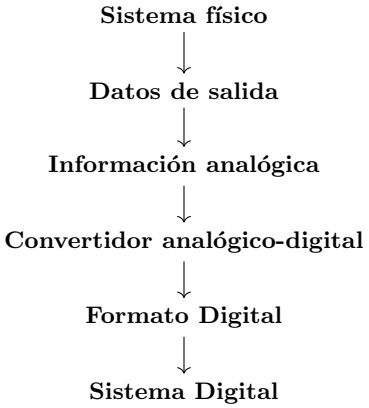
Complemento a 9 → número decimal → se obtiene intercambiando los $0's$ → $1's$ y los $1's \rightarrow 0's$.

### 1.7.3 Código Gray

Normalmente los datos de salida de **sistemas físicos** producen cantidades continuas. Por lo que se ocupa:

- Convertir las cantidades continuas en cantidades discretas.

- Convertir las cantidades discretas en cantidades digitales.
- Conviene usar el código Gray para este propósito.




La ventaja de usar el código Gray es que la diferencia entre dos números cualesquiera es únicamente de 1 bit.

$$(Binario) : (0111\,1000),\ (Gray) : (0100\,1100)$$

Una aplicación del código Gray es en los datos analógicos se representan mediante el cambio continuo en la posición de un eje.

### 1.7.4 Código ASCII

El código ASCII consta de 7-bits para su codificación, por lo que tiene un conjunto de 128 combinaciones distintas. Con $b_1 \rightarrow b_7$ siendo $b_7$ el bit más significativo, por ejemplo: letra A: 100 0001, (columna 100, fila 0001).

Contiene 94 caracteres imprimibles y 34 caracteres no imprimibles. Estos no imprimibles son caracteres de control.

- 26 letras minúsculas y 26 letras mayúsculas. (52)
- 10 dígitos decimales. (10)
- 32 caracteres especiales. (32)

### 1.7.5 Tipos de caracteres de control

1. **Creadores de Formato** Controlan la forma de imprimir, controles conocidos $\rightarrow$ máquinas de escribir.
   - (BS): Retroceso
   - (HT): Tabulador Horizontal
   - (CR): Retorno de carro

2. **Separadores de Información** Separan datos en divisiones como párrafos, líneas, páginas, etc.
    - (RS): Separador de Registros
    - (FS): Separador de Archivos
3. **Controladores de Información** Transmisión de datos entre terminales remotas.
    - (STX): Inicio de Texto
    - (ETX): Fin de Texto

    Encuadran un mensaje entre líneas telefónicas.

(8 bits) → (1 byte) → (1 carácter ASCII),
Se almacena 1 Byte por carácter ASCII.

### 1.7.6 Código para detectar errores

Para poder detectar errores en la comunicación de datos, se agrega un bit extra. Este bit indica la paridad:

**(Paridad):** Se refiere a la cantidad de bits 1's en un byte. Si la cantidad de bits 1's es par, se le asigna un 0, si es impar, se le asigna un 1. Es más común la paridad par.

El bit de paridad se transmite,
el receptor verifica la paridad del byte recibido,
si la paridad es correcta, se asume que no hay error,
si la paridad es incorrecta, se asume que hay un error.

## 1.8 Almacenamiento Binario y Registros

La información binaria → debe existir físicamente,
medio de almacenamiento en bits → registro,
un registro de → $n$ bits.

El estado de un registro es una tupla de $n$ bits;
el contenido → función, la interpretación → información.

Un registro de 16 bits:

1100001111001001

Un registro puede:

- **Almacenar datos**

elementos discretos de información.

- **Almacenar instrucciones**

misma configuración de bits, distinta interpretación.

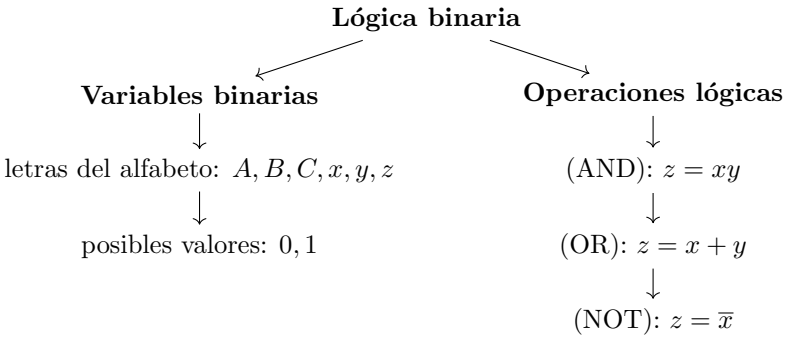
Manipulación de variables binarias → circuitos lógicos digitales.

Transferencia de registros → operación básica en sistemas digitales.

## 1.9 Lógica Binaria

**Variables** $\rightarrow$ 2 valores discretos,
**Operaciones** $\rightarrow$ 3 operaciones lógicas.

Se habla en términos de bits,
en valores de (0's y 1's), álgebra booleana.




### 1.9.1 Distintas interpretaciones

aritmética binaria:

$$1 + 1 = 10$$

lógica binaria:

$$1 + 1 = 1$$

### 1.9.2 Compuertas lógicas

Son dispositivos que operan 1 o más entradas binarias para producir una salida binaria.

(AND), (OR), (NOT)

# 2. Álgebra booleana y compuertas lógicas

## 2.1 Definiciones básicas

En el álgebra booleana, al igual que en todos los sitemas matemáticos deductivos, se define con un conjunto de elementos, un conjunto de operadores y varios axiomas. Un conjunto de elementos es cualquier colección de objetos con alguna propiedad en común.

### 2.1.1 Conjunto de elementos

Si $S$ es un conjunto y $x$ y $y$ son ciertos objetos, entonces $x \in S$, se denota que $x$ es un miembro del conjunto $S$, $S$, y $y \notin S$, denota que $y$ no es un miembro del conjunto $S$. $A = \{1, 2, 3, 4\}$, denota que estos elementos son miembros del conjunto $A$.

### 2.1.2 Conjunto de operadores

Un operador binario definido sobre un conjunto $S$ de elementos es una regla que asigna a cada par de elementos de $S$ un elemento único de $S$. Por ejemplo, $a * b = c$. Se designa $*$ como operador binario si especifica una regla para encontrar $c$ a partir del par $(a, b)$ y además si $a, b, c \in S$. Por contraparte no se designa operador binario si se descubre que $a, b \in S$, pero $c \notin S$.

### 2.1.3 Conjunto de axiomas

1. **Cerradura**. Un conjunto $S$ es cerrado respecto a un operador binario si, por cada par de elementos de $S$, el operador especifica una regla para obtener un elemento único de $S$. Por ejemplo, el conjunto de números naturales $N = \{1, 2, 3, ...\}$ el operador binario más $(+)$. Pero no es cerrado respecto al operador binario menos $(-)$, por las reglas de la resta aritmética.

2. **Ley asociativa**. Se dice que un operador binario $*$ sobre un conjunto $S$ es asociativo si:

$$(x * y) * z = x * (y * z) \text{ para todos } x, y, z \in S$$

3. **Ley conmutativa**. Se dice que un operador binario $*$ sobre un conjunto $S$ es conmutativo si

$$x * y = y * x \text{ para todos } x, y \in S$$

4. **Elemento identidad**. Se dice que un conjunto $S$ tiene un elemento de identidad respecto a una operación binaria $*$ sobre $S$ si existe un elemento $e \in S$ con la propiedad

$$e * x = x * e = x \text{ para todos } x \in S$$

5. **Inverso**. Se dice que un conjunto $S$, que tiene el elemento de identidad $e$ respecto a un operador $*$, tiene un inverso si, para todo $x \in S$, existe un elemento $y \in S$ tal que

$$x * y = e$$

6. **Ley distributiva**. Si $*$ y $\cdot$ son dos operadores binarios sobre un conjunto $S$, decimos que $*$ es distributivo sobre $\cdot$ si

$$x * (y \cdot z) = (x * y) \cdot (x * z)$$

## 2.2 Definición axiomática del álgebra booleana

En 1854, **George Boole** introdujo un tratamiento sistemático de la lógica. En 1938 **C. E. Shannon** introdujo un álgebra booleana de dos valores también llamada **álgebra de conmutación**.

Este álgebra es una estructura algebraica definida por un conjunto de elementos $B$, junto con dos operadores binarios, $+$ y $\cdot$, y seis postulados que introdujo **Huntington**.

1. **Cerradura**
   Cerradura respecto al operador $+$.
   Cerradura respecto al operador $\cdot$.

2. **Elemento identidad**
   Elemento identidad con respecto a $+$, designado por 0: $x + 0 = 0 + x = x$.
   Elemento identidad con respecto a $\cdot$, designado por 1, $x \cdot 1 = 1 \cdot x = x$.

3. **Conmutativa**
   Conmutativa respecto a $+$: $x + y = y + x$.
   Conmutativa respecto a $x$: $x \cdot y = y \cdot x$.

4. **Distributiva**
   $\cdot$ es distributivo sobre $+$: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
   $+$ es distributivo sobre $\cdot$: $x + (y \cdot z) = (x + y) \cdot (x + z)$.

5. **Complemento**
   Para cada elemento $x \in B$, existe un elemento un elemento $x' \in B$, tal que $x + x' = 1$ y $x \cdot x' = 0$.

6. **Dualidad**
   Existen al menos dos elementos $x, y \in B$, tales que $x \neq y$.

## 2.3 Funciones booleanas

El álgebra booleana se ocupa de variables binarias y operaciones lógicas. Una función booleana, es descrita por una expresión algebraica que consta de variables binarias, contantes 0 y 1, y símbolos lógicos de operación. Por ejemplo:

$$F_1 = x + y'z$$

La función $F_1$ es igual a 1 si $x$ es igual a 1 o si tanto $y'$ como z son iguales a 1. En los demás casos, $F_1$ es igual a 0.

Se puede representar una función booleana en una **tabla de verdad**. Una tabla de verdad es una lista de *combinaciones* de unos y ceros asignados a las variables binarias y una columna que muestra el valor de la función para cada combinación. El número de filas de la tabla es de $2^n$, donde $n$ es el número de variables de la función. Contando de 0 hasta $2^n - 1$.

| $x$ | $y$ | $z$ | $F_1$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Table 3: Tabla de verdad de $F_1 = x + y'z$

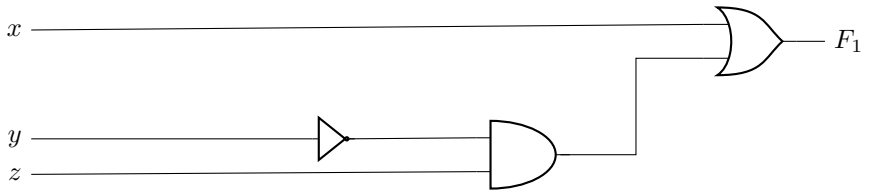



Figure 1: Circuito lógico de $F_1 = x + y'z$

Una función booleana se puede transformar de una expresión algebraica a un diagrama de circuitos hecho con compuertas lógicas. Solo hay una forma de representar una función booleana en una tabla de verdad. Sin embargo, la función en forma algebraica, puede expresarse de varias maneras. Manipulando una expresión booleana se puede obtener una expresión más simple de la misma función y así reducir el número de compuertas lógicas del circuito.

Consideremos ahora esta función booleana y su posible simplificación:

$$\begin{aligned} F_2 &= x'y'z + x'yz + xy' + xy' \\ &= x'z(y' + y) \\ &= x'z + xy \end{aligned}$$

La función de tres terminos y ocho literales se reduce a únicamente dos términos y cuatro literales. Ambas realizan la misma función, pero es preferible la forma simplificada porque requiere menos compuertas lógicas.

### 2.3.1 Manipulación algebraica

Se define una literal como una sola variable dentro de un término. Si se reduce el número de términos, el número de literales, o ambas, en una expresión booleana, podría obtenerse un circuito más simple. Las funciones de hasta cinco variables se pueden simplificar con el método del mapa. Se describirá más adelante.

### 2.3.2 Complemento de una función

El complemento de una función $F$ es $F'$, se obtiene intercambiando los ceros por unos y unos por ceros en el valor de $F$. Esto se puede deducir algebraicamente usando el teorema de **DeMorgan**. Además este se puede extender a tres o más variables. Así:

$$\begin{aligned} (A + B + C)' &= (A + x)' \\ &= A'x' \\ &= A'(B + C)' \\ &= A'(B'C') \\ &= A'B'C' \end{aligned}$$

El teorema de DeMorgan se puede generalizar de la siguiente manera:

$$(A + B + C + D + ... + F)' = A'B'C'D'...F'$$
$$(ABCD...F)' = A' + B' + C' + D' + ... + F'$$

Otro procedimiento más sencillo para obtener el complemento de una función consiste en obtener el dual de la función y complementar cada literal. Esto es en consecuencia del teorema de DeMorgan. El dual de una función se obtiene intercambiando los operadores **AND** y **OR**, y unos y ceros. Ejemplo:

$F_1 = x'yz' + x'y'z$
El dual de $F_1$ es $(x' + y + z')(x' + y' + z)$
Complementando cada literal: $(x + y' + z)(x + y + z') = F_1'$

## 2.4 Formas canónicas y estándar

| | | | Minitérminos | | Maxitérminos | |
|---|---|---|---|---|---|---|
| $x$ | $y$ | $z$ | Términos | Designación | Términos | Designación |
| 0 | 0 | 0 | $x'y'z'$ | $m_0$ | $x+y+z$ | $M_0$ |
| 0 | 0 | 1 | $x'y'z$ | $m_1$ | $x+y+z'$ | $M_1$ |
| 0 | 1 | 0 | $x'yz'$ | $m_2$ | $x+y'+z$ | $M_2$ |
| 0 | 1 | 1 | $x'yz$ | $m_3$ | $x+y'+z'$ | $M_3$ |
| 1 | 0 | 0 | $xy'z'$ | $m_4$ | $x'+y+z$ | $M_4$ |
| 1 | 0 | 1 | $xy'z$ | $m_5$ | $x'+y+z'$ | $M_5$ |
| 1 | 1 | 0 | $xyz'$ | $m_6$ | $x'+y'+z$ | $M_6$ |
| 1 | 1 | 1 | $xyz$ | $m_7$ | $x'+y'+z'$ | $M_7$ |

Table 4: Minitérminos y maxitérminos para tres variables binarias

Una variable binaria podría aparecer en su forma normal ($x$) o en su forma complementada ($x'$). Ahora si se considerada dos variables binarias $x$ y $y$ que se combinan con una operación AND. Se pueden obtener cuatro combinaciones posibles: $x'y'$, $x'y$, $xy'$ y $xy$. Cada uno de estos cuatro términos AND es un *minitérmino, o producto estándar*. De igual manera se puede combinar $n$ variables para formar $2^n$ minitérminos. Se enumeran del 0 a $2^n - 1$. Cada minitérmino se obtiene de un término AND de las n variables, se designa un símbolo $m_j$, donde $j$ denota el equivalente decimal del número binario que representa el minitérmino.

Asimismo, $n$ variables que forman un término OR, llamado *maxitérmino o suma estándar*. Cabe decir que cada maxitérmino es el complemento de su minitérmino correspondiente y viceversa. Así, expresar las combinaciones 001, 100 y 111 como $x'y'z'$, $xy'z'$ y $xyz$, respectivamente. Puesto que cada uno de estos minitérminos hace que $f_1 = 1$, se tiene:

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

### 2.4.1 Forma canónica: suma de minitérminos

Esto ilustra una propiedad importante del álgebra booleana: cualquier función booleana se puede expresar como *una suma de minitérminos*. Ahora podemos hacer lo mismo pero con maxitérminos, de modo qué el complemento de $f_2$ se lee:

$$f_2' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

Ejemplo de suma de minitérminos:

Expresar la función booleana $F = A + B'C$ como la suma de minitérminos.
Por lo tanto:

$$A = A(B + B') = AB + AB'$$

Aún le falta una variable:

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

Al segundo término, B'C , le falta una variable:

$$B'C = B'C(A + A') = AB'C + A'B'C$$

Al combinar todo se tiene:

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C \end{aligned}$$

Como AB'C aparece dos veces, se puede simplificar:

$$\begin{aligned} F &= ABC + ABC' + AB'C + AB'C' + A'B'C \\ &= A'B'C + AB'C' + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

En ocasiones conviene expresar la función booleana, de la siguiente notación:

$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

### 2.4.2 Forma canónica: producto de maxitérminos

Si obtenemos el complemento de $f_2'$, se obtiene $f_2$:

$$\begin{aligned} f_2 &= (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \end{aligned}$$

Este ejemplo ilusta la segunda propiedad del álgebra booleana, cualquier función booleana se puede expresar como un *producto de maxitérminos*. Se dice que las funciones booleanas expresadas como suma de minitérminos o producto de maxitérminos están en **forma canónica**.

Ejemplo de producto de maxitérminos:

Expresar la función booleana $F = xy + x'z$ en forma de producto de maxitérminos. Por lo tanto:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

La función tiene tres variables, $x$, $y$ y $z$. A cada término OR le falta una variable; por tanto:

$$x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$
$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$
$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Después de combinar todos los términos y eliminar los que se repiten; se tiene:

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0 \cdot M_2 \cdot M_4 \cdot M_5 \end{aligned}$$

Una forma cómoda de expresar esta función es:

$$F(x, y, z) = \prod(0, 2, 4, 5)$$

## 2.5 Conversión entre formas canónicas

El complemento de una función expresado como la suma de minitérminos es igual a la suma de los minitéminos que faltan en la función original. Por ejemplo:

$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

Su complemento se expresa como

$$F'(A, B, C) = \sum(0, 2, 3) = m_0 + m_2 + m_3$$

Ahora si determinamos el complemento de $F'$, se obtiene $F$:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \prod(0, 2, 3)$$

Por lo que se puede ver la relación entre minitérminos y maximínimos:

$$m_j' = M_j$$

Este ejemplo ilustra la conversión de una función expresada como la suma de minitérminos y su equivalente como el producto de maxitérminos.

Si tenemos:

$$F = xy + x'z$$

La suma de minitérminos:

$$F(x, y, z) = \sum(1, 3, 6, 7)$$

El producto de maxitérminos:

$$F(x, y, z) = \prod(0, 2, 4, 5)$$

## 2.6 Formas estándar

Las formas canónicas son formas básicas al leer una función de su tabla de verdad, pero casi nunca son las que tienen el número mínimo de literales, porque cada minitérmino o maxitérmino debe contener, por definición, todas las variables, complementadas o sin complementar.

Otra manera de expresar las funciones booleanas es en forma estándar. Existen dos tipos de forma estándar: *la suma de productos y el producto de sumas.* La suma de productos contiene términos AND, llamados *términos de producto*, ejemplo:

$$F_1 = y' + xy + x'yz'$$

Por lo consecuente se puede crear un circuito con una implementación de dos niveles. El producto de sumas contiene términos OR, llamados *términos de suma*, ejemplo:

$$F_2 = x(y' + z)(x' + y + z)$$

El tipo estándar produce una estructura de compuertas de dos niveles.

## 2.7 Otras operaciones lógicas

Hay $2^2n$ funciones para n variables binarias, en el caso de dos variables $n = 2$, existe 16 posibles funciones booleanas.

| Funciones booleanas | Símbolo operador | Nombre | Comentarios |
|---|---|---|---|
| $F_0 = 0$ | | Nula | Constante binaria 0 |
| $F_1 = xy$ | $x \cdot y$ | AND | $x$ y $y$ |
| $F_2 = xy'$ | $x/y$ | Inhibición | x, pero no y |
| $F_3 = x$ | | Transferencia | $x$ |
| $F_4 = x'y$ | $y/x$ | Inhibición | y, pero no x |
| $F_5 = y$ | | Transferencia | $y$ |
| $F_6 = xy' + xy$ | $x \oplus y$ | OR exclusivo | $x$ o $y$, pero no ambos |
| $F_7 = x + y$ | $x + y$ | OR | $x$ o $y$ |
| $F_8 = (x + y)'$ | $x \downarrow y$ | NOR | No OR |
| $F_9 = xy + x'y'$ | $(x \oplus y)'$ | Equivalencia | $x$ es igual a $y$ |
| $F_{10} = y'$ | $y'$ | Complemento | No $y$ |
| $F_{11} = x + y'$ | $x \subset y$ | Implicación | Si y, entonces x |
| $F_{12} = x'$ | $x'$ | Complemento | No $x$ |
| $F_{13} = x' + y$ | $x \supset y$ | Implicación | Si x, entonces y |
| $F_{14} = (xy)'$ | $x \cdot y$ | NAND | No AND |
| $F_{15} = 1$ | | Identidad | Contante binaria 1 |

## 2.8 Circuitos integrados

Un circuito integrado (CI) es un cristal semiconductor de silicio. llamado *chip*, que contiene componentes electrónicos para construir compuertas digitales. Las diversas compuertas se interconectan dentro del chip para crear el circuito requerido.

### 2.8.1 Niveles de integración

Los CI digitales se clasifican según la complejidad de sus circuitos, la cual se mide por el número de compuertas que contiene. Los CI se clasifican en cuatro categorías:

1. **SSI** (small-scale integration). Contiene hasta diez compuertas.
2. **MSI** (medium-scale integration). Contiene ente diez y mil compuertas.
3. **LSI** (large-scale integration). Contiene miles de compuertas.
4. **VLSI** (very large-scale integration). Contiene cientos de miles de compuertas.

### 2.8.2 Familias de lógica digital

Los CI también se clasifican por su funcionamiento lógico. El circuito básico en cada tecnología es una compuerta NAND, NOR o inversor. Las familias de lógica digital más populares son:

- **TTL** lógica transistor-transistor.
- **ECL** lógica acoplada por emisor.
- **MOS** metal-óxido-semiconductor.
- **CMOS** metal-óxido-semiconductor complementario.

TTL ha estado en operación por mucho tiempo y se le considera **estándar**. ECL resulta ventajoso en sistemas que deben operar a **alta velocidad**. MOS es apropiado para circuitos que requieren una **densidad elevadad de componentes** y CMOS es preferible en sistemas que requieren **bajo consumo de energía**. **CMOS** se ha convertido en la familia lógica dominante, mientras que TTL y ECL ha decaído.

### 2.8.3 Diseño asistido por computadora (CAD)

El diseño de sistemas digitales con circuitos VLSI contienen millones de transistores. En general es imposible desarrollar y verificar sistemas tan complejos sin la ayuda de herramientas computarizadas. Estas herramientas (CAD) consisten en programas de software que ayudan a desarrollar hardware digital automatizando el proceso de diseño. El diseñador cuenta con diversas opciones para crear la implementación física de un circuito digital en silicio. Por ejemplo:

- (**ASIC**, application-specific integrated circuit): circuito integrado para una aplicación específica.
- (**FPGA**, field-programmable gate array): arreglo de compuertas programable en campo.
- (**PLD**, programmable logic device): un dispositivo de lógica programable.

Un adelanto importante en el diseño de sistemas digitales es el uso de un lenguaje de descripción de hardware (HDL). Este representa diagramas de lógica y otra información en forma textual. Sirve para simular un sistema antes de construirlo, a fin de verificar la funcionalidad y la operación.

# 3. Minimización en el nivel de compuertas

## 3.1 El método del mapa

La complejidad de las compuertas de lógica digital que implementan una función booleana está relacionada directamente con la complejidad de la expresión algebraica de la función. aunque la representación como una tabla de verdad es única, hay muchas maneras de expresarla algebraicamente.

El método del mapa ofrece un procedimiento para simplificar funciones booleanas. Se podría considerar como una versión pictórica de una tabla de verdad. Se conoce como mapa de **Karnaugh** o mapa K.

Es un diagrama hehco de cuadrados, cada uno representando un minitérmino de la función. Al reconocer diversos patrones, el usuario puede deducir expresiones algebraicas alternas para la misma función y luego escoger la más simple. Las expresiones están expresadas en una de las dos formas estándar, suma de productos o producto de sumas. Esto produce un diagrama de circuito con el mímino de compuertas y el mímino de entradas a cada compuerta. La expresión más simple no es única.

### 3.1.1 Mapas de dos variables

Para un mapa de dos variables, hay cuatro minitérminos $m_0, m_1, m_2, m_3$, por tanto el mapa consiste en cuatro cuadrados, uno para cada minitérmino. Si se marca los cuadrados cuyos minitérminos pertenecen a una función dada, el mapa se convertirá en otra forma útil de representar cualquiera de las 16 funciones booleanas de dos variables.

### 3.1.2 Mapas de tres variables

En un mapa de tres variables hay ocho minitérminos para tres variables binarias; por tanto el mapa consta de ocho cuadrados.

$$m_0 = 000, m_1 = 001, m_2 = 011, m_3 = 010$$
$$m_4 = 110, m_5 = 111, m_6 = 101, m_7 = 100$$

Advertir que los minitérminos no están acomodados en sucesión binaria, sino en código Gray. La característica es que sólo un bit cambia de valor entre dos columnas adyacentes.

El número de cuadrados adyacentes que es posible combinar **siempre** debe ser una potencia de 2, como 1, 2, 4, 8. Al aumentar el número de cuadrados adyacentes que se combinan, se reduce el número de literales del término producto obtenido.

- Un cuadrado representa un minitérmino $\rightarrow$ término con tres literales.
- Dos cuadrados adyacentes representan dos minitérminos $\rightarrow$ término con dos literales.

- Cuatro cuadrados adyacentes representan cuatro minitérminos → término con un literal.
- Ocho cuadrados adyacentes representan ocho minitérminos → término constante 1.

### 3.1.3 Mapas de cuatro variables

El mapa para funciones booleanas de cuatro variables presentan 16 minitérminos, por tanto el mapa consta de 16 cuadrados. El término correspondiente para cada cuadrado de obtiene de la concatenación del número de fila con el número de la columna. Por ejemplo, los números de la tercera fila (11) y la segunda columna (01) al concatenarse dan el número binario 1101, que es equivalente binario al 13 decimal, representando al minitérmino $m_{13}$.

Para el proceso de simplificación podemos tener en cuenta:

- Un cuadrado representa un minitérmino → término con cuatro literales.
- Dos cuadrados adyacentes representan dos minitérminos → término con tres literales.
- Cuatro cuadrados adyacentes representan cuatro minitérminos → término con dos literales.
- Ocho cuadrados adyacentes representan ocho minitérminos → término con una literal.
- Dieciseis cuadrados adyacentes representan dieciseis minitérminos → constante 1.

### 3.1.4 Mapas de cinco variables

El uso de mapas de más de cuatro variables no es tan simple. Un mapa de cinco variables ocupa 32 cuadrados, y uno de seis variables ocupa 64 cuadrados. Por lo cual se complica progresivamente. La mejor forma de visualizar estos mapas es imaginar que los mapas están uno encima del otro. Cualesquier dos curadrados que queden uno encima del otro se consideran adyacentes. Una alternativa es utilizar programas de computadora escritos específicamente para facilitar la simplificacion de funciones booleanas de cinco o más variables.

Se puede demostrar que cualesquier $2^k$ cuadrados adyacentes, para $k = (0, 1, 2, ..., n)$, en un mapa de $n$ variables, representan un área que produce un término de $n - k$ literales, con $n > k$, si $n = k$, el área representa una constante 1.

## 3.2 Implicantes primos

Al escoger los cuadrados adyacentes hay que asegurarse de que se cubran todos los minitérminos de la función. Evitar cubrir términos redundantes cuyos términos ya están cubiertos por otros. Este procedimiento se podría hacer de

manera sistemática con el término de **implicante primo e implicante primo esencial**.

Un implicante primo es un término producto que se obtiene combinando el número máximo posible de cuadrados adyacentes en el mapa. Si un minitérmino de un cuadrado está cubierto sólo por un implicante primo, se dice que es un **implicante primo esencial**.

## 3.3 Condiciones de indiferencia

En la práctica hay algunas aplicaciones en las que la función no está definida para ciertas combinaciones de las variables. Las funciones con salidas no especificadas para ciertas combinaciones de entradas se llaman **funciones incompletamente especificadas**. Conviene usar estas condiciones de indiferencia en el mapa para simplificar aún más la expresión booleana. Se puede representar de la siguiente manera:

$$F(w, x, y, z) = \sum(1, 3, 7, 11, 15) + d(0, 2, 5)$$

Estas condiciones de indiferencia se pueden marcar bien con 0 o 1, dependiendo de la aplicación.

## 3.4 Funciones NAND y NOR

Muchos circuitos digitales digitales se construyen con compuertas NAND y NOR en lugar de compuertas AND y OR. Ya que son más fáciles de fabricar con componentes electrónicos. Se dice que la compuerta NAND es una compuerta universal porque cualquier sistema digital puede implementarse con ella. La compuerta NOR es otra compuerta universal. Se puede demostrar que cualquier función booleana se puede implementar con compuertas NOR.

## 3.5 Función XOR

La función OR exclusivo (XOR), denotada por el símbolo $\oplus$, es una operación lógica que efectúa la siguiente operación:

$$x \oplus y = xy' + x'y$$

Es igual a 1 si sólo $x$ es igual a 1 o sólo $y$ es igual a 1, pero no si ambas son 1. El NOR exclusivo, también llamado equivalencia, realiza la siguiente operación:

$$(x \oplus y)' = xy + x'y'$$

Es igual a 1 si tanto x como y son 1 o si ambas son 0.

Se cumplen las identidades siguientes para la operación XOR:

$$\begin{gathered} x \oplus 0 = x \\ x \oplus 1 = x' \\ x \oplus x = 0 \\ x \oplus x' = 1 \\ x \oplus y' = x' \oplus y = (x \oplus y)' \end{gathered}$$

También puede demostrarse que la operación XOR es tanto conmutativa como asociativa; es decir,

$$x \oplus y = y \oplus x$$

y

$$x \oplus (y \oplus z) = (x \oplus y) \oplus z$$

### 3.5.1 Función impar

La operación XOR con tres o más variables se convierte en una función booleana ordinaria sustituyendo el símbolo $\oplus$ por su expresión booleana equivalente. Por ejemplo:

$$\begin{aligned} A \oplus B \oplus C &= (AB' + A'B)C' + (AB + A'B')C \\ &= AB'C' + A'BC' + ABC + A'B'C \\ &= \sum(1, 2, 4, 7) \end{aligned}$$

La función XOR de tres variables es una función impar, es decir, es igual a 1 si el número de 1's en la función es impar.

### 3.5.2 Generación y verificación de paridad

Las funciones XOR son muy útiles en los sistemas que requieren códigos para detectar y corregir errores. Un bit de paridad es un bit adicional que se incluye con el mensaje binario de modo que el número total de unos se impar o par. Se detecta un error si la paridad recibida no corresponde con la transmitida. El transmior usaría un **circuito generador de paridad** y el receptor un **circuito verificador de paridad**.

| Mensaje de tres bits | | | Bit de Paridad |
|---|---|---|---|
| $x$ | $y$ | $z$ | $P$ |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Table 5: Tabla de verdad para de un generador deparidad par

# 4. Lógica Combinacional

Un circuito consiste en variables de entrada, compuertas lógicas y variables de salida. Las $n$ variables de entrada provienen de una fuente externa, las $m$ variables de salidas van a un destino externo. Si se incluyen registros de almacenamiento entonces se considera un circuito secuencial. Con $n$ variables de entrada hay $2^n$ posibles combinaciones de entradas binarias. Para cada una hay un posible valor de salida. Por tanto es posible especificar una tabla de verdad con los valores de salida para cada combinación de entradas.

Se presentará circuitos combinacionales estándar más importantes, como los sumadores, restadores, comparadores, decodificadores, codificadores, multiplexores, desmultiplexores. Estos componentes se fabrican como circuitos MSI (Medium Scale Integration) y también se usan como celdas estándar en circuitos VSLI complejos tanto como ASIC.

## 4.1. Procedimiento de análisis

Se refiere a deducir la función lógica que realiza el circuito. Este proceso parte de un diagrama lógico dado y culmina en conjunto de funciones booleanas, una tabla de verdad o una posible explicación del funcionamiento del circuito.

- El primer paso es asegurarse que el circuito sea combinacional y no secuencial. Lo cual se puede asegurar si no hay elementos de almacenamiento o trayectorias de retroalimentación.
- Una vez asegurado se procede a obtener las funciones booleanas de salida o la tabla de verdad.

### 4.1.1 Pasos para obtener las funciones booleanas de salida

1. Rotular con símbolos arbitrarios todas las salidas de compuerta que son función de variables de entrada. Determinar esto para cada salida de compuerta.
2. Rotular con símbolos arbitrarios las compuertas que son función de variables de entrada y de compuertas previamente rotuladas.
3. Repetir el paso 2 hasta obtener las salidas del circuito.
4. Obtener las funciones booleanas de salida en términos de las variables de entrada.

### 4.1.2 Deducción de la tabla de verdad

1. Determinar el número de variables de entrada del circuito, paran $n$ entradas, forme $2^n$ posibles combinaciones y bosqueje una lista de 0 a $2^n - 1$.
2. Rotular las salidas de las compuertas selectas con símbolos arbitrarios.

3. Obtener la tabla de verdad para las salidas de las compuertas que son función únicamente de las variables de entrada.

4. Obtener la tabla de verdad para las salidas de las compuertas que son función de valores previamente definidos.

## 4.2. Procedimiento de diseño

Parte de la especificación del problema y culmina en un diagrama lógico de circuitos o un conjunto de funciones booleanas a partir de las cuales se puede obtener un diagrama lógico.

### 4.2.1. Pasos para obtener el diagrama lógico

1. Deducir el número requerido de entradas y salidas; asignar símbolos a cada una.

2. Deducir la tabla de verdad que define la relación entre las entradas y las salidas.

3. Obtener las funciones booleanas simplificadas para cada salida en función de las entradas.

4. Dibujar el diagrama lógico y verificar que el diseño sea correcto.

La tabla de verdad de un circuito combinacional consta de columnas de entrada y columnas de salida. Las columnas de entrada se obtinen de los $2^n$ números binarios para las $n$ variables de entrada. Los valores binarios de salida se deducen de las especificaciones planteadas. Tales especificaciones suelen ser incompletas, y cualquier interpretación errónea podría dar pie a una tabla de verdad incorrecta.

Las funciones de salida se simplifican con cualquier método disponible, así como mapas de karnaugh, manipulación algebraica, o un programa de computadora.

## 4.3. Sumador-restador binario

Es un circuito combinacional que realiza operaciones aritméticas de suma y resta con números binarios.

### 4.3.1 Semisumador

Necesita dos entradas binarias y dos salidas binarias. Las variables de entrada designan los bits sumandos; las salidas, la suma y el acarreo. Las funciones booleanas simplificadas para las dos salidas se obtienen directamente de la tabla de verdad. Estas son:

$$S = x'y + xy' \rightarrow x \oplus y$$
$$C = xy$$

La tabla de verdad sería:

| $x$ | $y$ | $C$ | $S$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Table 6: Semisumador

### 4.3.2 Sumador completo

Necesita tres entradas binarias y dos salidas binarias. Las variables de entrada designan los bits sumandos y el acarreo de entrada; las salidas, la suma y el acarreo de salida. Las funciones booleanas simplificadas para las dos salidas se obtienen directamente de la tabla de verdad. Estas son:

$$S = x'y'z + xyz' + xy'z' + xyz \rightarrow x \oplus y \oplus z$$
$$C = xy + xz + yz$$

La tabla de verdad sería:

| $x$ | $y$ | $z$ | $C$ | $S$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Table 7: Sumador completo

### 4.3.3 Sumador binario

Produce la suma aritmética de dos números binarios. Es posible construirlo con sumadores completos dispuestos en cascada, conectando el acarreo de salida de un sumador completo al acarreo de entrada del siguiente.

### 4.3.4 Restador binario

La forma más conveniente de efectuar la resta de números binarios sin signo es utilizando complementos. La resta de $A - B$ se efectúa obteniendo el complemento a dos de $B$ y sumándolo a $A$. El complemento a dos de un número binario se obtiene calculando el complemento a uno y sumándole 1 al par de bits menos significativo. El complemento a uno se implementa con inversores y el 1 se suma a través de un acarreo de entrada.

Las operaciones de suma y resta se pueden combinar en un solo circuito que tiene un sumador binario compartido. Esto se realiza con la inclusión de una compuerta XOR con cada sumador completo. Agregando una entrada de control $M$ para decidir si se efectúa una suma o una resta, se obtiene un sumador-restador binario.

### 4.3.5 Sumador decimal

Un sumador decimal requiere como mínimo nueve entradas y cinco salidas, ya que se requieren cuatro bits para codificar cada dígito decimal y el circuito necesita un acarreo de entrada y uno de salida. Se puede implementar utilizando el código BCD.

## 4.4. Multiplicador binario

La multiplicación de números binarios se efectúa igual que la de números decimales. El multiplicando se multiplica por cada bit del multiplicador, comenzando por el menos significativo. Cada una de estas multiplicaciones forma un producto parcial. Los productos parciales se suman para obtener el producto final.

Para construir un multiplicador se requiere:

- Multiplicador de $J$ bits.
- Multiplicando de $K$ bits.
- $(J \times K)$ compuertas AND.
- $(J - 1)$ sumadores de $K$ bits.

Se obtiene un producto de $(J + K)$ bits.

## 4.5 Comparador de magnitudes

La comparación de dos números es una operación que determina si un número es mayor que, menor que o igual a otro número. *Un comparador de magnitudes* es un circuito combinacional que compara dos números, $A$ y $B$, y determina sus magnitudes relativas. El resultado de la comparación se especifica con tres variables binarias que indican si $A > B \vee A = B \vee A < B$.

Si queremos comparar los siguientes dos números binarios:

$$A = A_3A_2A_1A_0$$
$$B = B_3B_2B_1B_0$$

Cada letra con subíndice representa uno de los dígitos del número. Los dos números son iguales si todos los pares de dígitos significativos son iguales.

$$A_3 = B_3$$
$$A_2 = B_2$$
$$A_1 = B_1$$
$$A_0 = B_0$$

Se puede implementar con una función XNOR así:

$$x_i = A_iB_i + A_i'B_i' \quad \text{para } i = 0, 1, 2, 3$$

donde $x_i = 1$ únicamente si los dos bits de la posición $i$ son iguales. Para que exista la condición de igualdad, las $x_i$ deben ser todas 1. Lo que implica una operación AND de todas las variables.

$$(A = B) = x_3x_2x_1x_0$$

Para determinar si $A > B \vee A < B$, se inspeccionan las magnitudes relativas de pares de dígitos significativos, comenzando por el más significativo. Si los dos dígitos son iguales, se comparará el siguiente par de dígitos menos significativos; así sucesivamente hasta encontrar un par de dígitos distinto.

$$\text{Si } A = 1 \wedge B = 0 \quad \text{entonces} \quad A > B,$$
$$\text{Si } A = 0 \wedge B = 1 \quad \text{entonces} \quad A < B$$

Esta comparación sucesiva se expresa lógicamente con las dos funciones booleanas:

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0',$$
$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

Los símbolos $(A > B) \wedge (A < B)$ son variables binarias de salida que valen 1 cuando $A > B$ y $A < B$, respectivamente.

## 4.6. Decodificadores

En los sistemas digitales, las cantidades discretas de información se representan con códigos binarios. Un código binario de $n$ bits puede representar hasta $2^n$ cantidades distintas.

Un *decodificador* es un circuito combinacional que convierte la información binaria de $n$ **líneas de entrada** a un máximo de $2^n$ **líneas de salida.**

Aunque podría tener un número menor de líneas de salida, así:

$$(dec) : n \text{ a } m \text{ líneas de salida, donde} \quad m \leq 2^n$$

Un decodificador podría operar con salidas complementadas o no complementadas. Podría tener una entrada de habilitación que debe satisfacer una condición lógica dada para habilitar el circuito. Un *decodificador* con entrada de habilitación puede funcionar como **desmultiplexor**. Un *desmultiplexor* es un circuito que recibe información de una sola línea y la dirige a una de $2^n$ líneas de salida.

Dado que se obtienen operaciones de decodificador y desmultiplexor con el mismo circuito, decimos que un decodificador con entrada de habilitación es un *decodificador/desmultiplexor*. Es posible conectar los decodificadores con entradas de habilitación unos con otros para formar un circuito decodificador más grande.

## 4.7. Codificadores

Un codificador es un circuito digital que efectúa la operación inversa de un decodificador. Un codificador tiene $2^n$ (o menos) líneas de entrada y $n$ líneas de salida. Estas últimas generan el valor código binario correspondiente al valor de entrada. Un ejemplo es de octal a binario. Tiene ocho entradas, y tres salidas que generan el número binario correspondiente. Se supone que sólo una entrada es igual a 1 en cualquier momento dado.

| Entradas | | | | | | | | Salidas | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | $x$ | $y$ | $z$ |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

Table 8: Codificador octal a binario

Se puede implementar con compuertas OR cuyas salidas se determinan directamente de la tabla de verdad.

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

### 4.7.1 Codificador con prioridad

Un codificador con prioridad es un circuito codificador que incluye una función de prioridad. Funciona de tal manera que si dos o más entradas son 1 al mismo tiempo, la salida prioritaria tendrá precedencia.

| Entradas | | | | Salidas | | |
|---|---|---|---|---|---|---|
| $D_0$ | $D_1$ | $D_2$ | $D_3$ | $x$ | $y$ | $V$ |
| 0 | 0 | 0 | 0 | X | X | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| X | 1 | 0 | 0 | 0 | 1 | 1 |
| X | X | 1 | 0 | 1 | 0 | 1 |
| X | X | X | 1 | 1 | 1 | 1 |

Table 9: Codificador con prioridad

La salida V es un indicador de un bit válido que adquiere el valor de 1 cuando una o más entradas son 1. Si todas las entradas son 0, la entrada no será válida y V será 0. En tal caso, las otras dos salidas no se inspeccionarán y se especifican como condiciones de indiferencia.

Cuanto más alto sea el subíndice de una entrada, mayor prioridad tendrá esa entrada. La entrada $D_3$ es la de mayor prioridad, así que si es 1, la salida $xy$ será 11, sin importar el valor de las demas entradas.

Las expresiones booleanas simplificadas se obtienen por medio de mapas, la condición para la salida $V$ es una función OR de todas las variables de entrada.

$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

## 4.8. Multiplexores

Un multiplexor es un circuito combinacional que selecciona información binaria de una de muchas líneas de entrada y la envía a una sola línea de salida. La selección de la línea de entrada se controla con un conjunto de líneas de selección. Normalmente hay $2^n$ líneas de entrada y $n$ líneas de selección cuyas combinaciones determinan cuál entrada se selcciona. En diagramas de bloques es común rotular los multiplexores como $MUX$.

Los multiplexores se denominan como *selectores de datos*, pues seleccionan una de varias entradas y dirigen la información binaria a la línea de salida.

Las compuertas AND y los inversores del multiplexor semejan un circuito decodificador, de hecho, decodifican las líneas de selección de entrada. **En general un multiplexor de $2^n$ líneas a 1 se construye a partir de un decodificador de $n$ líneas a $2^n$ líneas**, una para cada compuerta AND. Las salidas de las compuertas AND se conectan a una compuerta OR.

El tamaño del multiplexor se especifica por el número de líneas de entrada de datos que tiene ($2^n$) y la única línea de salida. El número de líneas de selección de entrada es $n$. Podrían tener una entrada de habilitación que controla el funcionamiento de la unidad.

### 4.8.1 Multiplexor de 2-1

El siguiente ejemplo muestra un multiplexor de 2-1 que tiene dos entradas de datos $D_0$ y $D_1$ y una entrada de selección $S$. La entrada de selección determina cuál de las dos entradas de datos se envía a la salida. Si $S = 0$, la entrada de datos $D_0$ se envía a la salida. Si $S = 1$, la entrada de datos $D_1$ se envía a la salida.

| $S$ | $D_0$ | $D_1$ | $O$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Table 10: Tabla de verdad de un multiplexor de 2-1

**Multiplexor de 2-1 simplificado**

| $S$ | $O$ |
|---|---|
| 0 | $D_0$ |
| 1 | $D_1$ |

Table 11: Tabla de verdad de un multiplexor de 2-1

### 4.8.2 Multiplexor de 4-1

El siguiente ejemplo muestra un multiplexor de 4-1 que tiene cuatro entradas de datos $D_0$, $D_1$, $D_2$ y $D_3$ y dos entradas de selección $S_0$ y $S_1$. Las entradas de selección determinan cuál de las cuatro entradas de datos se envía a la salida.

| $S_0$ | $S_1$ | $O$ |
|---|---|---|
| 0 | 0 | $D_0$ |
| 0 | 1 | $D_1$ |
| 1 | 0 | $D_2$ |
| 1 | 1 | $D_3$ |

Table 12: Tabla de verdad de un multiplexor de 4-1

### 4.8.3 Multiplexor de 8-1

El siguiente ejemplo muestra un multiplexor de 8-1 que tiene ocho entradas de datos $D_0$, $D_1$, $D_2$, $D_3$, $D_4$, $D_5$, $D_6$ y $D_7$ y tres entradas de selección $S_0$, $S_1$ y $S_2$. Las entradas de selección determinan cuál de las ocho entradas de datos se envía a la salida.

| $S_0$ | $S_1$ | $S_2$ | $O$ |
|---|---|---|---|
| 0 | 0 | 0 | $D_0$ |
| 0 | 0 | 1 | $D_1$ |
| 0 | 1 | 0 | $D_2$ |
| 0 | 1 | 1 | $D_3$ |
| 1 | 0 | 0 | $D_4$ |
| 1 | 0 | 1 | $D_5$ |
| 1 | 1 | 0 | $D_6$ |
| 1 | 1 | 1 | $D_7$ |

Table 13: Tabla de verdad de un multiplexor de 8-1

## 4.9. Desmultiplexores

Un desmultiplexor es un circuito combinacional que selecciona información binaria de una línea de entrada y la envía a una de $2^n$ líneas de salida. La selección de la línea de salida se controla con un conjunto de líneas de selección. Se puede rotular a los desmultiplexores como *DMUX*. El desmultiplexor solo funciona cuando la entrada de habilitación es 1. Cuando es 0, todas las líneas de salida son 0.

Es utilizado cuando se requiere enviar información a varios dispositivos, es similar a un decodificador pero la diferencia es que el decodificador es usado para seleccionar una de $2^n$ líneas de salida, mientras que el desmultiplexor se

utiliza para enviar la información a una de $2^n$ líneas de salida. También es llamado como un *distribuidor de datos*, pues distribuye la información binaria de una línea de entrada a una de varias líneas de salida.

### 4.9.1 Desmultiplexor de 1-2

En el siguiente ejemplo, muestra un desmultiplexor de 1-2 que tiene una entrada de datos $D$ y una entrada de selección $S$. La entrada de selección determina cuál de las dos líneas de salida, $O_0$ u $O_1$, se envía a la salida.

| $D$ | $S$ | $O_0$ | $O_1$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Table 14: Tabla de verdad de un desmultiplexor de 1-2

### 4.9.2 Desmultiplexor de 1-4

En el siguiente ejemplo, muestra un desmultiplexor de 1-4 que tiene una entrada de datos $D_{in}$ y dos entradas de selección $S_0$ y $S_1$. La entrada de datos se envía a una de las cuatro líneas de salida, $O_0$, $O_1$, $O_2$ o $O_3$, dependiendo del valor de las entradas de selección.

| $S_0$ | $S_1$ | $D_{in}$ | $O_0$ | $O_1$ | $O_2$ | $O_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

Table 15: Tabla de verdad de un desmultiplexor de 1-4

### 4.9.3 Desmultiplexor de 1-8

En el siguiente ejemplo, muestra un desmultiplexor de 1-8 que tiene una entrada de datos $D_{in}$ y tres entradas de selección $S_0$, $S_1$ y $S_2$. La entrada de datos se envía a una de las ocho líneas de salida, $O_0$, $O_1$, $O_2$, $O_3$, $O_4$, $O_5$, $O_6$ o $O_7$, dependiendo del valor de las entradas de selección.

| $S_0$ | $S_1$ | $S_2$ | $O_0$ | $O_1$ | $O_2$ | $O_3$ | $O_4$ | $O_5$ | $O_6$ | $O_7$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | $D_{in}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | $D_{in}$ | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | $D_{in}$ | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | $D_{in}$ | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | $D_{in}$ | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | $D_{in}$ | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $D_{in}$ | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $D_{in}$ |

Table 16: Tabla de verdad de un desmultiplexor de 1-8

# 5. Lógica secuencial sincrónica