

# Introducción

# Representaciones numéricas

- Representaciones analógicas

una cantidad se representa mediante un indicador proporcional que varía en forma continua

- Termómetro
- Velocímetro

- Representaciones digitales

- las cantidades se representan no mediante indicadores que varían en forma continua, sino mediante símbolos llamados dígitos.

- La representación digital es el resultado de asignar un número de precisión limitada a una cantidad que varía en forma continua. (REDONDEAR)

análogica ≡ continua  
digital ≡ discreta (paso por paso)

# Ejemplo.

- ¿Cuáles de las siguientes cantidades son analógicas y cuáles son digitales?
  - (a) Un interruptor de diez posiciones.
  - (b) La corriente que fluye a través de un contacto eléctrico.
  - (c) La temperatura de una habitación.
  - (d) Granos de arena en la playa.
  - (e) El medidor de combustible de un automóvil.

# SISTEMAS DIGITALES Y ANALÓGICOS

- Un **sistema digital** es la combinación de dispositivos diseñados para manipular información lógica/cantidades físicas que se representan en forma digital.
  - EJ. Computadoras, calculadoras digitales, equipos de audio y video digital, sistema telefónico.
- Un **sistema analógico** contiene dispositivos que manipulan cantidades físicas que se representan en forma analógica. las cantidades pueden variar sobre un intervalo continuo de valores.
  - Ej. amplificadores de audio, los equipos de grabación y reproducción de cintas magnéticas, interruptor regulador de luz.

# Pros de los Sistemas digitales

- los sistemas digitales son más fáciles de diseñar
- Es fácil almacenar información
- Es más fácil mantener la precisión y la exactitud en todo el sistema
- La operación puede programarse
- Los circuitos digitales son más resistentes al ruido
- Pueden fabricarse más circuitos digitales en los chips de CI

# Limitaciones

- El mundo real es analógico. El procesamiento de las señales digitales lleva tiempo.

Para aprovechar las técnicas digitales al procesar entradas y salidas analógicas, se deben seguir los siguientes pasos:

1. Convertir la variable física en una señal eléctrica (analógica).
2. Convertir la señal eléctrica (analógica) a su forma digital.
3. Procesar (operar con) la información digital.
4. Convertir las salidas digitales nuevamente a la forma analógica del mundo real.

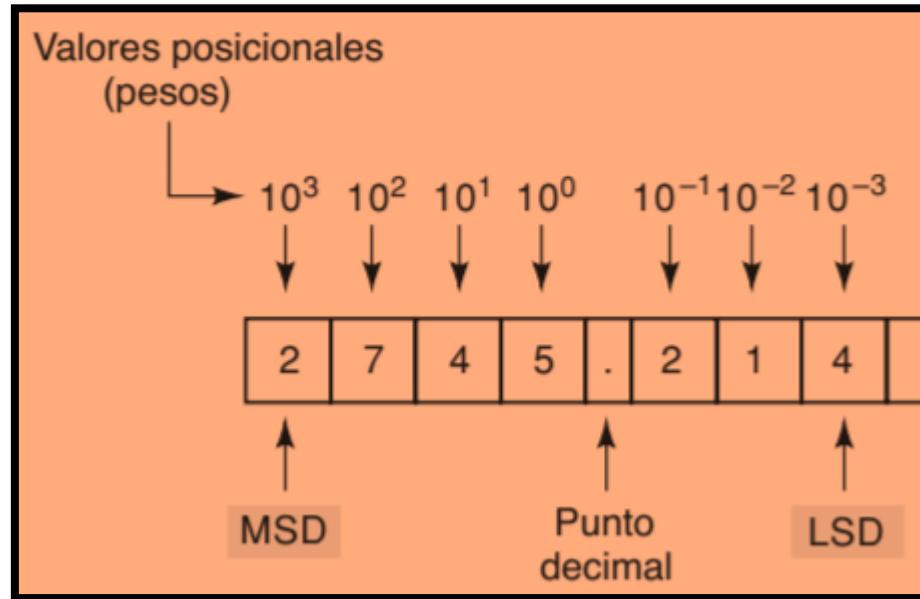
A large, irregularly shaped circle filled with a dark blue gradient, centered on a white background. The circle is surrounded by a textured, splattered border in shades of blue and white, resembling paint or liquid droplets.

# SISTEMAS NUMÉRICOS DIGITALES

# Sistema decimal.

Ejemplo 2745.214

$$(2 \times 10^{+3}) + (7 \times 10^{+2}) + (4 \times 10^1) + (5 \times 10^0) \\ + (2 \times 10^{-1}) + (1 \times 10^{-2}) + (4 \times 10^{-3})$$

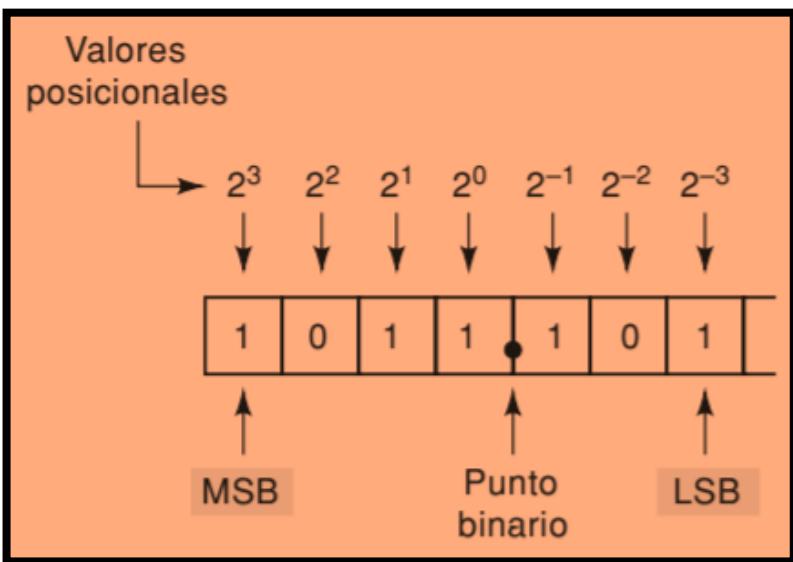


# Sistema binario

- sólo hay dos símbolos o posibles valores de dígitos: **0** y **1**. sin embargo se puede usar para representar cualquier cantidad que pueda representarse en decimal o en otros sistemas numéricos.

- Ejemplo.

1101.101



$$\begin{aligned}1011.101_2 &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\&\quad + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\&= 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 \\&= 11.625_{10}\end{aligned}$$

Conversión de binario a decimal

# Secuencia de conteo binario

Pesos	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	Equivalent decimal
	0	0	0	0	0
	0	0	0	1	1
	0	0	1	0	2
	0	0	1	1	3
	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
	1	0	0	0	8
	1	0	0	1	9
	1	0	1	0	10
	1	0	1	1	11
	1	1	0	0	12
	1	1	0	1	13
	1	1	1	0	14
	1	1	1	1	15

↑  
LSB

¿Cuál es el mayor número que se puede representar si se utilizan nueve bits?

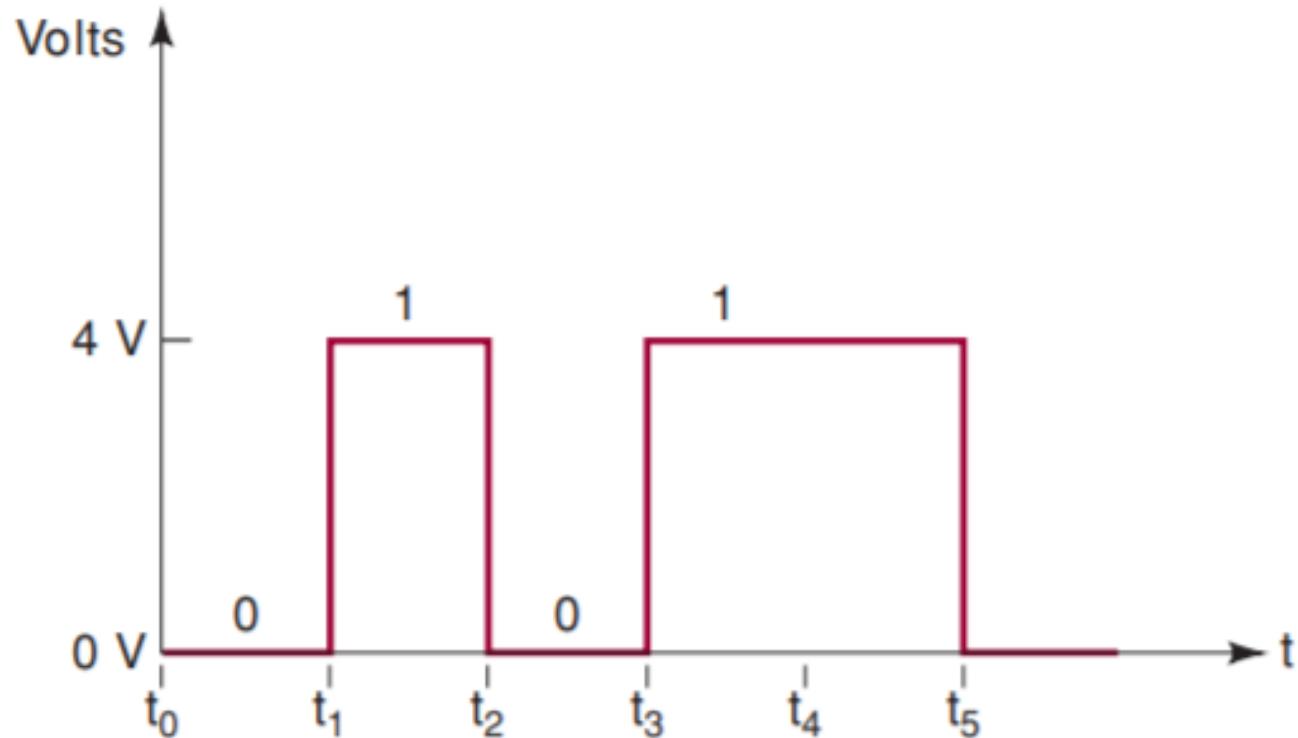
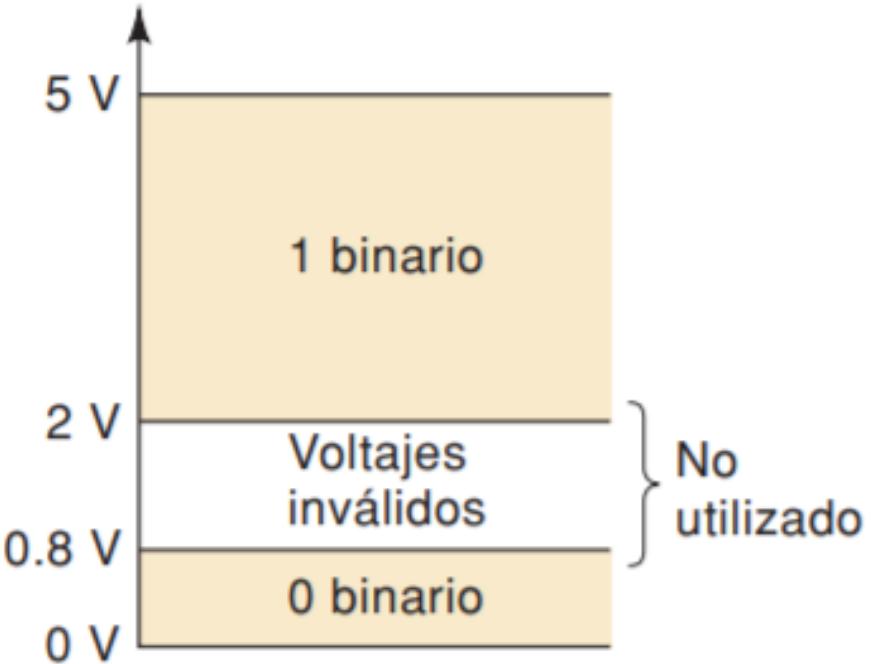
¿Cuál es el equivalente decimal de **11010112**?

¿Cuál es el binario que sigue al **10111<sub>2</sub>** en la secuencia de conteo?

¿Cuál es el mayor valor decimal que se puede representar si se utilizan 12 bits?

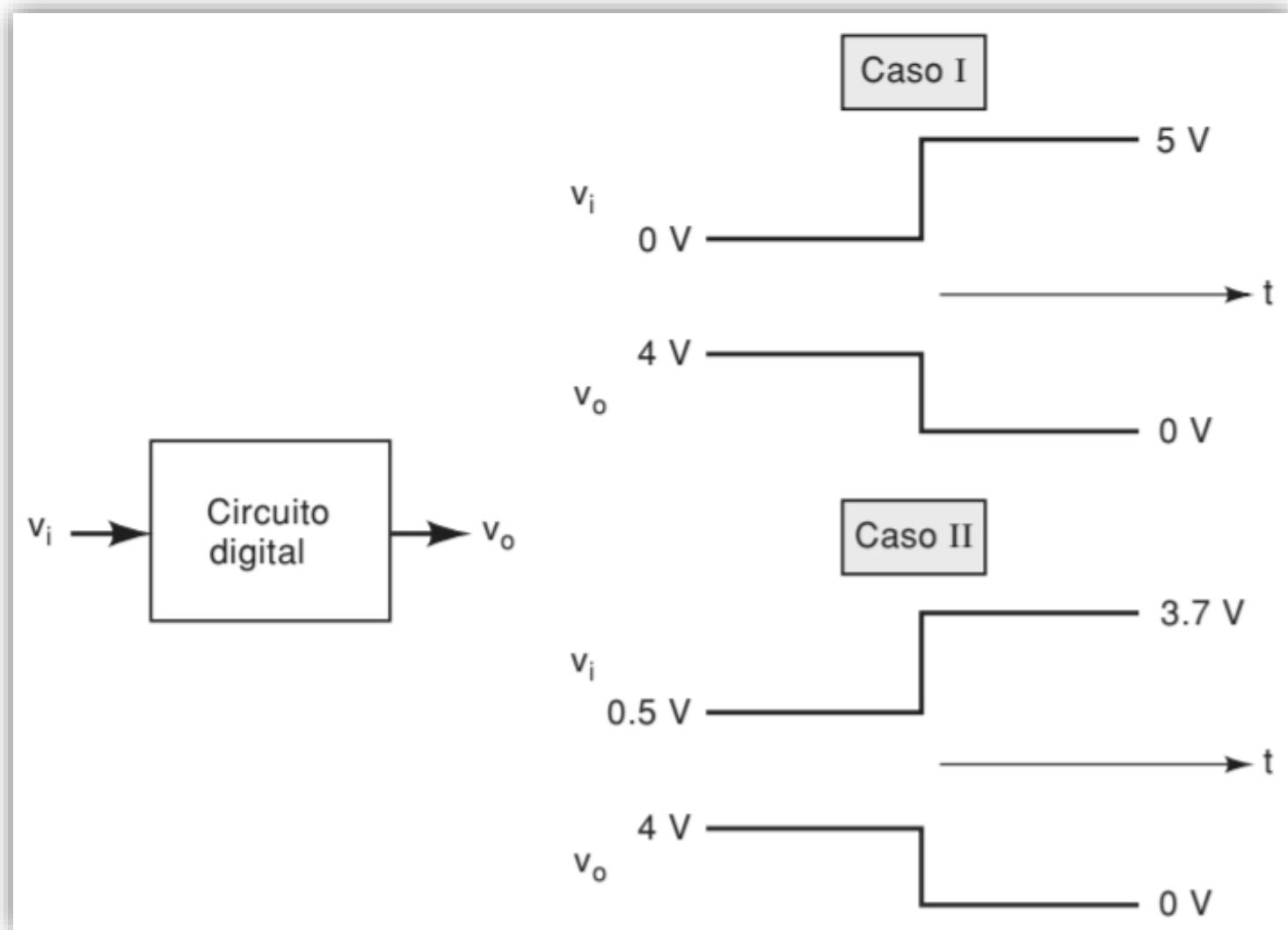
# REPRESENTACIÓN DE CANTIDADES BINARIAS

- En los sistemas digitales electrónicos la información binaria se representa mediante voltajes (o corrientes) que están presentes en las entradas y salidas de los diversos circuitos.
- cero Volts (0 V) podrían representar el 0 binario y 5 V podrían representar el 1 binario.
- debido a las variaciones en los circuitos, el 0 y el 1 se representan mediante intervalos de voltaje. Normalmente todas las señales de entrada y salida se encuentran dentro de alguno de estos intervalos, excepto durante las transiciones de un nivel a otro.



En los sistemas analógicos el valor exacto de un voltaje es importante. Por ejemplo, si el voltaje analógico es proporcional a la temperatura medida por un transductor, 3.6 V representaría una temperatura distinta que 4.3 V. En otras palabras, la magnitud del voltaje lleva información importante.

# CIRCUITOS DIGITALES/CIRCUITOS LÓGICOS

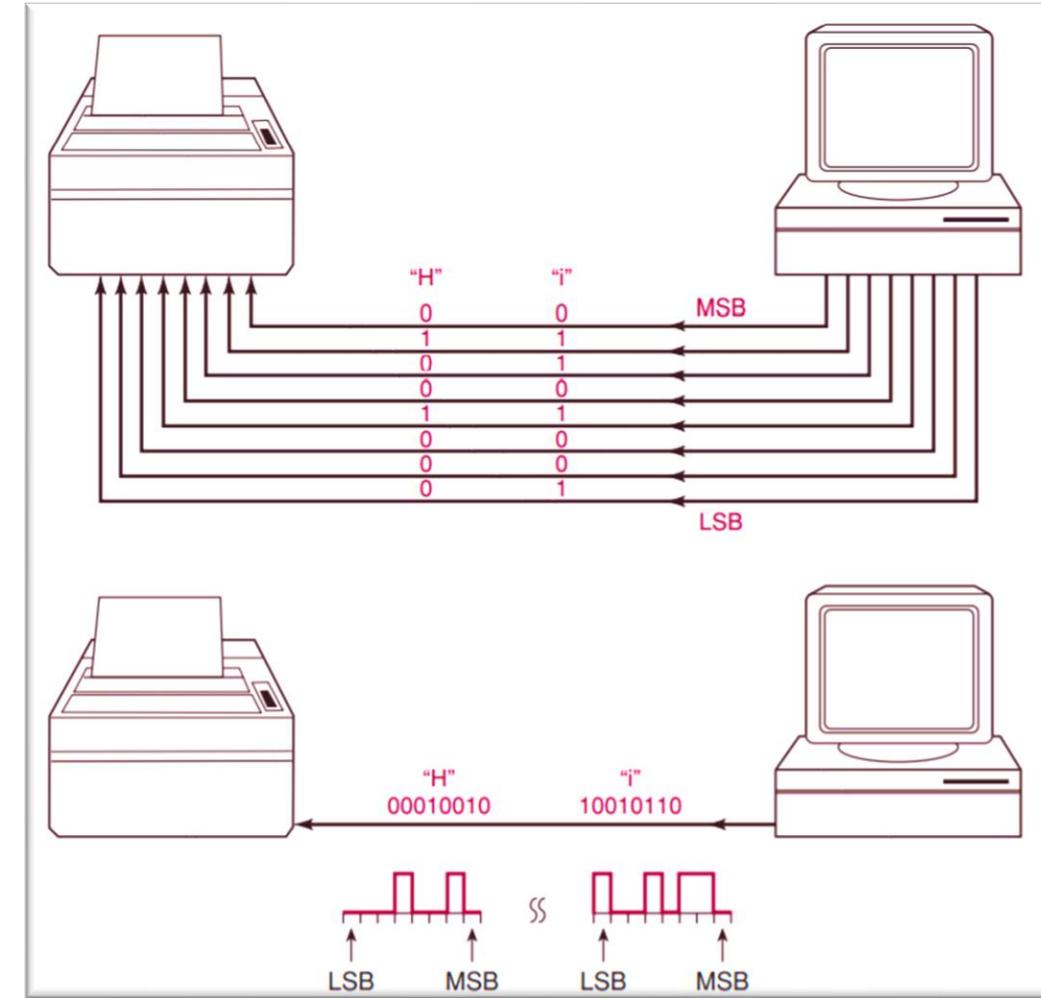


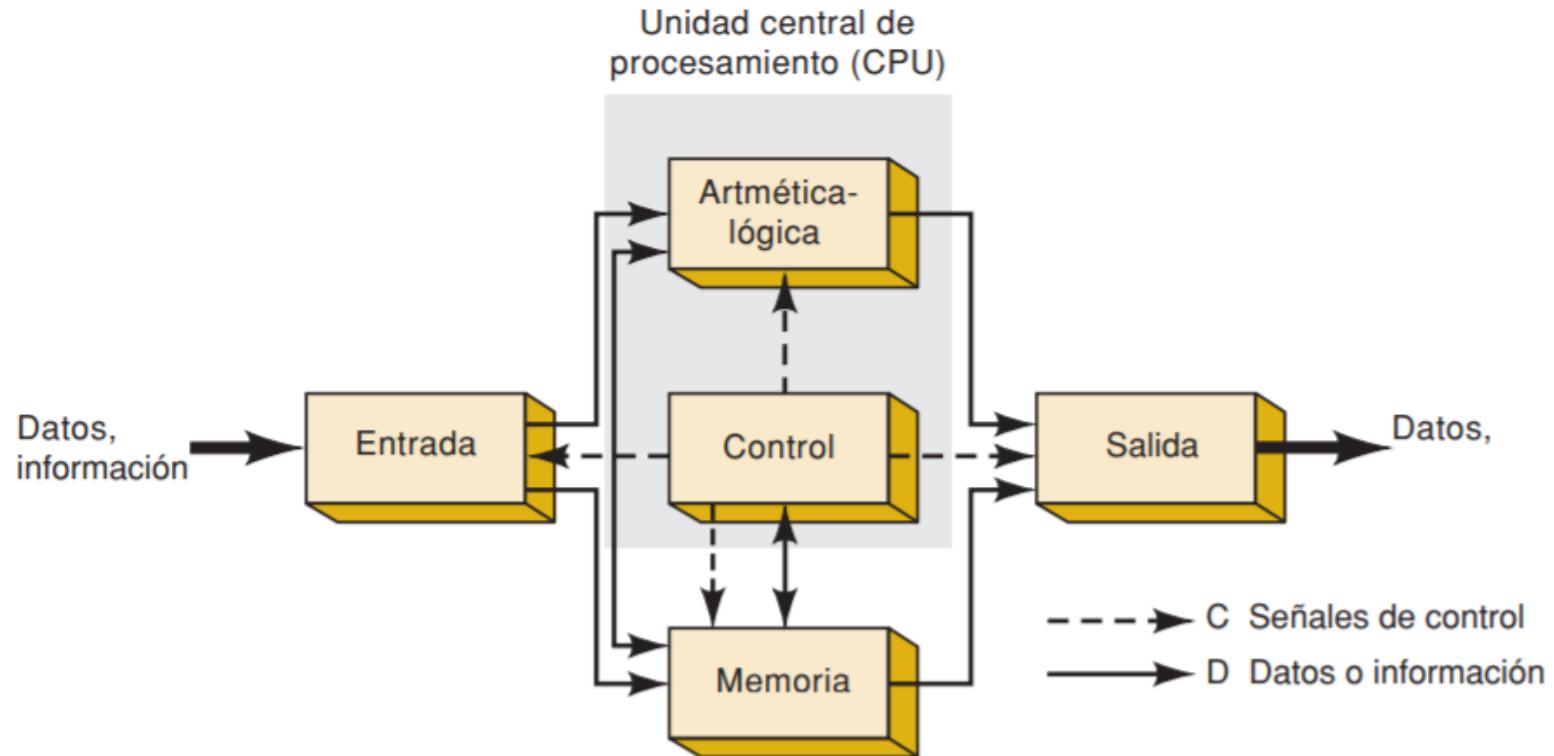
1. Verdadero o falso: El valor exacto de un voltaje de entrada es imprescindible para un circuito digital.
2. ¿Puede un circuito digital producir el mismo voltaje de salida para distintos valores del voltaje de entrada?
3. Un circuito digital también se conoce como circuito \_\_\_\_\_.
4. Un gráfico que muestra cómo cambian una o más señales digitales a través del tiempo se llama \_\_\_\_\_.

A large, irregularly shaped circle filled with a dark blue gradient. The circle is surrounded by a white border that is heavily textured with blue and grey splatters, giving it a hand-painted or digital noise effect.

# Transmission de datos digitales

# TRANSMISIÓN de datos binarios EN PARALELO Y EN SERIE





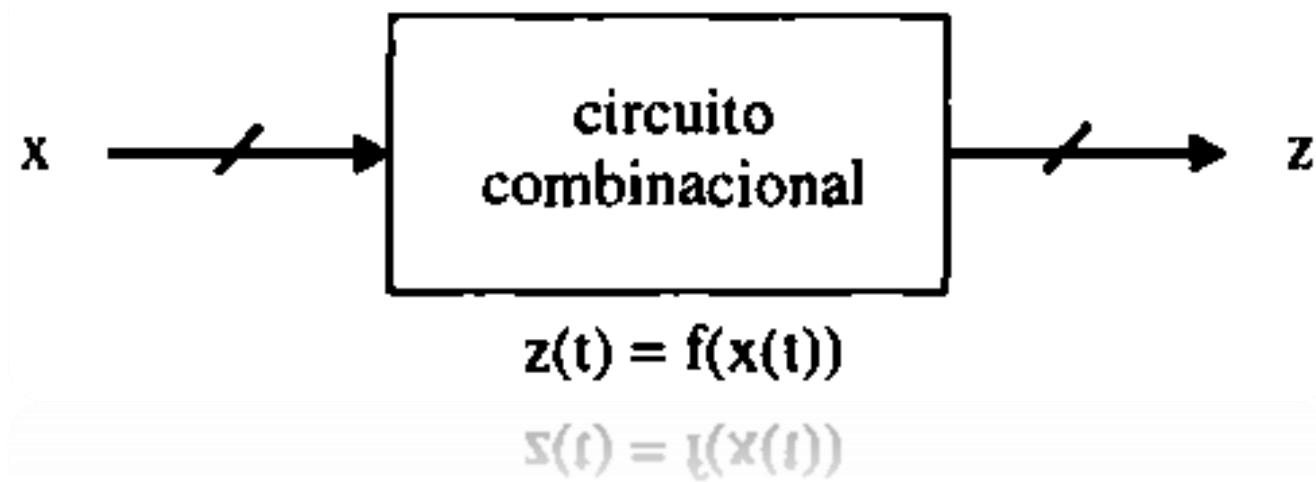
A large, irregularly shaped circle filled with a dark blue gradient. The circle is surrounded by a white border with scattered dark blue and grey splatters and dots of varying sizes, resembling ink splatters or a hand-painted effect.

# Circuitos Combinacionales

- Un circuito digital combinacional es aquel que implementa funciones de conmutación cuyas salidas en un instante,  $t$ , dependen sólo del valor de las entradas en ese mismo instante .
- Este circuito consta de compuertas lógicas interconectadas entre sí sin que haya lazos de realimentación.

Existen dos enfoques principales :

- si es conocido el circuito y se desea establecer cuál es la operación que realiza (análisis).
- si se plantea el problema contrario, conocida la función hay que obtener el circuito (diseño).



# Análisis de circuitos

- El objetivo principal es obtener una representación de la función de conmutación que se implementa. A este objetivo se le llama ***análisis lógico*** del circuito.
- En algunos casos es posible, además, obtener una descripción verbal de la operación del circuito "hace la suma", "compara números", etc). Aunque el análisis lógico es el objetivo principal no es el único aspecto que debe contemplar un buen análisis de un circuito.

# Otros aspectos que se deben considerar:

- El coste del circuito .  
(número de compuertas lógicas y conexiones entre compuertas del circuito)
- Un análisis de parámetros eléctricos .  
(rendimiento del circuito en cuanto a márgenes de ruido, potencia disipada, etc .)
- Un análisis temporal .  
(dado un patrón de entradas, determinar la forma de onda de las señales de salida considerando los retrasos de propagación de las compuertas lógicas . El análisis temporal sirve para verificar si el circuito realiza correctamente la función de conmutación o si, por el contrario, existen fenómenos transitorios como por ejemplo "azares", así como para calcular los valores máximos y mínimos de los tiempos de propagación que determinan la velocidad de operación del circuito)

# Diseño de circuitos combinacionales

- Dada una función de conmutación, obtener un circuito que la realice.
- Cualquier expresión lógica de la función sería válida para obtener el circuito .

El **objetivo del proceso de diseño** consiste en obtener un circuito óptimo respecto a algún criterio.

EJ. criterios a optimizar en los circuitos digitales combinacionales.

- 1 .- Minimizar el número compuertas que deben atravesar las señales de entrada hasta alcanzar la salida, considerando que sólo se usan puertas AND y OR (o bien sólo NAND o sólo NOR).
  
- 2 .- Minimizar el coste de la función en cuanto :
  - a) al número de puertas lógicas de que consta el circuito
  - b) al número de conexiones total del circuito.

# Suma de productos o producto de sumas

el **primer criterio** conduce a buscar expresiones que den lugar a circuitos en dos niveles de compuertas . Expresiones de tipo suma de productos (sp) o producto de sumas (ps) son las más apropiadas para cumplir este requisito .

el **segundo criterio**, reducción del coste, significa reducir el número de términos productos (en expresiones sp) o reducir el número de términos sumas (en expresiones ps) y, además, buscar términos productos o sumas con el menor número de literales posible. Existen expresiones que cumplen estos requisitos que son las llamadas sumas mínima de la función (expresión sp) o producto mínimo (ps) .

# Teorema para obtener la suma/el producto mínimo

"La suma mínima (producto mínimo) de una función de conmutación está formada por el conjunto mínimo de implicantes primas con el menor número de literales posible que cubren completamente la función".

Una **implicante prima** es un término producto o implicante (término suma) cuyos mintérminos (maxtérminos) son todos mintérminos (maxtérminos) de la función y, además, no existe otra implicante de la función que contenga a todos los mintérminos (maxtérminos) de dicho término producto (suma) .

Una **implicante** (implicada) se dice que es esencial si obligatoriamente pertenece a la solución óptima, ya que sólo ella cubre a algún mintérmino (maxtérmino) de la función.

Por lo tanto, el objetivo del teorema del proceso de diseño de una función de conmutación es encontrar una expresión formada por el conjunto mínimo de implicantes (implicadas) primas que cubran completamente la función .



# ÁLGEBRA Y FUNCIONES DE CONMUTACIÓN

# ÁLGEBRA DE CONMUTACIÓN

- El álgebra de conmutación es un sistema matemático compuesto por un conjunto de dos elementos :

$B = \{0, 1\}$ , y dos operaciones OR (+) y AND ( $\bullet$ ) definidas en B

El álgebra de conmutación cumple los postulados del álgebra de Boole.

+	0	1
0	0	1
1	1	1

OR

·	0	1
0	0	0
1	0	1

AND

# Postulados del algebra de boole

P1 . Ley de identidad : Existen elementos identidad (0 para la operación "+" y 1 para la operación "·") de forma que para cualquier elemento  $x$ , se cumple :

$$x+0=x$$

$$x \cdot 1=x$$

P2. Ley conmutativa : Para cualesquiera dos elementos  $x$  e  $y$ , se cumple :

$$x+y=y+x$$

$$x \cdot y=y \cdot x$$

P3 . Ley distributiva : Dados tres elementos  $x$ ,  $y$ ,  $z$  se cumple :

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

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

P4. Ley del complemento : Para todo elemento  $x$  existe un elemento  $x'$  tal que:

$$x+x'=1$$

$$x \cdot x'=0$$

# Teoremas del álgebra de conmutación

- T1 . LEY DE IDEMPOTENCIA :  $X + X = X$   $X \cdot X = X$
- T2. LEY DE UNICIDAD DEL COMPLEMENTO : EL ELEMENTO X DEL POSTULADO CUARTO ES ÚNICO .
- T3. LEY DE LOS ELEMENTOS DOMINANTES :  $X + 1 = 1$   $X \cdot 0 = 0$
- T4. LEY INVOLUTIVA :  $\overline{\overline{X}} = X$
- T5 . LEY DE ABSORCIÓN :  $X + X \cdot Y = X$   $X \cdot (X + Y) = X$
- T6 . LEY DEL CONSENSO :  $X + \overline{X} \cdot Y = X + Y$   $X \cdot \overline{(X + Y)} = X \cdot Y$
- T7. LEY ASOCIATIVA :  $X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$   $X + (Y + Z) = (X + Y) + Z$
- T8. LEY DE DE MORGAN :  $\overline{XY} = \overline{X} + \overline{Y}$   $\overline{X + Y} = \overline{X} \cdot \overline{Y}$
- T9. LEY DE DE MORGAN GENERALIZADA :  $\overline{XYZ\dots} = \overline{X} + \overline{Y} + \overline{Z} + \dots$   $\overline{X + Y + Z\dots} = \overline{X} \cdot \overline{Y} \cdot \overline{Z} \cdot \dots$
- T10. LEY DEL CONSENSO GENERALIZADO :  $X \cdot Y + \overline{X} \cdot Z + Y \cdot Z = X \cdot Y + \overline{X} \cdot Z$

# Demostración del teorema de idempotencia del álgebra de conmutación

- Basándonos en los postulados del álgebra de Boole tenemos:

P1. Identidad:  $x + 0 = x \quad x \cdot 1 = x$

P2. Comutativa:  $x + y = y + x \quad x \cdot y = y \cdot x$

P3. Distributiva:  $x + (y \cdot z) = (x + y) \cdot (x + z) \quad x \cdot (y + z) = x \cdot y + x \cdot z$

P4. Complemento:  $x + \bar{x} = 1 \quad x \cdot \bar{x} = 0$

LEY DE IDEMPOTENCIA :

$$x + x = x$$

$$x \cdot x = x$$

$$x + x = (x + x) \cdot 1 = (x + x)(x + \bar{x}) = x + x\bar{x} = x + 0 = x$$

$$x \cdot x = x \cdot x + 0 = x \cdot x + x \cdot \bar{x} = x \cdot (x + \bar{x}) = x \cdot 1 = x$$

# FUNCIONES DE CONMUTACIÓN

Son funciones que se definen sobre el conjunto  $B = \{0, 1\}$  del álgebra de conmutación. Estrictamente se definen como :

$$F: B \times \dots \times B \times B = B^N \rightarrow B$$

Ejemplo.

$$B^n: (x_1, x_2, \dots, x_n)$$



$$f(x, y, z)$$



$$\begin{aligned} f(0,0,0) &= 0, & f(0,0,1) &= 1, & f(0,1,0) &= 0, \\ f(0,1,1) &= 1, & f(1,0,0) &= 0, \\ f(1,0,1) &= 0, & f(1,1,0) &= 1, & f(1,1,1) &= 1. \end{aligned}$$

# REPRESENTACIÓN DE FUNCIONES

- **Tablas de verdad.**
- En una tabla se representan dos columnas. En la primera de ellas se escriben todas las combinaciones de las variables de entrada en orden binario . En la otra columna se anota el valor que toma la función para cada combinación de las variables de entrada

x	y	z	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

# Mapa de Karnaugh.

Las variables de la función se dividen en dos grupos . Uno de ellos se sitúa en el eje horizontal de una tabla y el otro en el eje vertical .

Las combinaciones de cada grupo de variables se escriben en el orden del código Gray . Así, disponemos de una cuadrícula en cuyas celdas se anota el valor de la función para la combinación de las variables asignada .

La propiedad principal es que dos celdas geométricamente adyacentes también corresponden a códigos lógicos adyacentes .

c \ d	a	b		
00	00	01	11	10
01	0	0	0	0
11	1	1	0	0
10	0	0	1	1
f	0	1	1	1

## Expresiones o fórmulas.

- Se utiliza una expresión algebraica para representar las funciones . Se combinan las variables con los operadores **NOT** , **AND** y **OR**.
- Aquellas combinaciones de las variables que hagan 1 (ó 0) la expresión serán las combinaciones en que la función es 1 (ó 0) .
- Existen formas canónicas y estándares . Tanto unas como otras tienen en común que son fórmulas compuestas únicamente por suma de productos / producto de sumas . En las formas canónicas, además, se cumple que los **productos** son siempre **mintérminos** y las **sumas** son **maxtérminos** .

– Suma de productos:

$$f(a, b, c, d) = \bar{a} \bar{c} d + a c + b c \bar{d}.$$

– Producto de sumas:

$$f(a, b, c, d) = (c + d) (\bar{a} + c) (a + \bar{c} + \bar{d}) (a + b + \bar{c}).$$

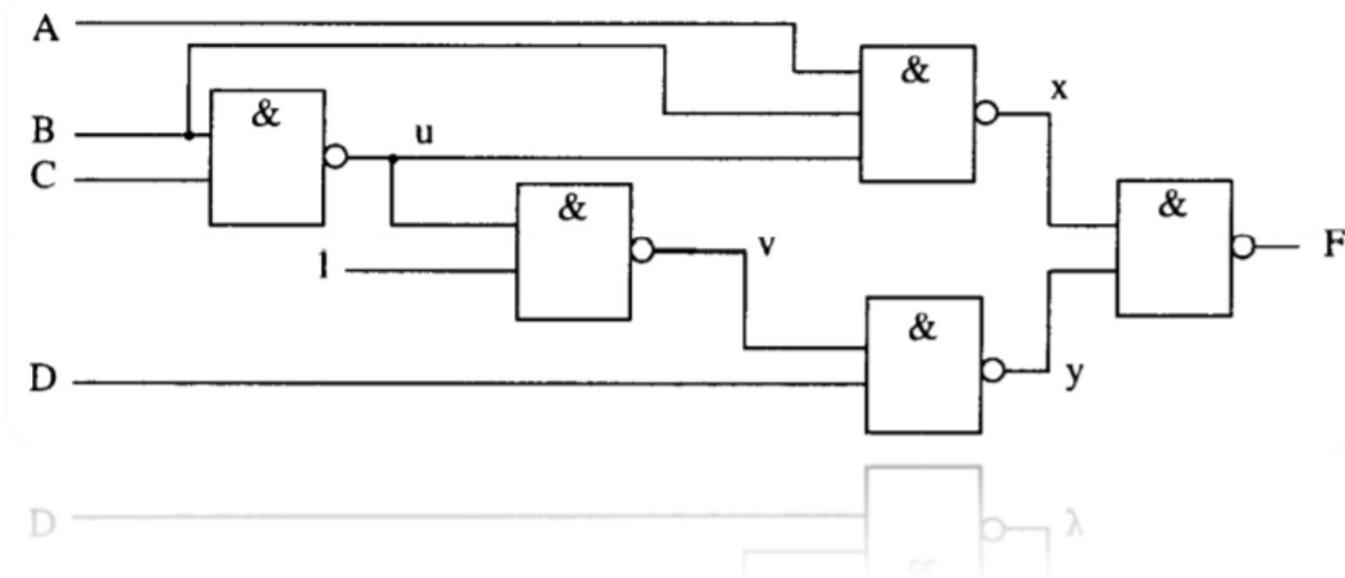
– Suma de mintérminos:

$$\begin{aligned} f(a, b, c, d) &= \bar{a} \bar{b} \bar{c} d + \bar{a} b \bar{c} d + \bar{a} b c \bar{d} + a \bar{b} c \bar{d} + a \bar{b} c d + a b c \bar{d} + a b c d \\ &= m_1 + m_5 + m_6 + m_{10} + m_{11} + m_{14} + m_{15} = \Sigma (1, 5, 6, 10, 11, 14, 15). \end{aligned}$$

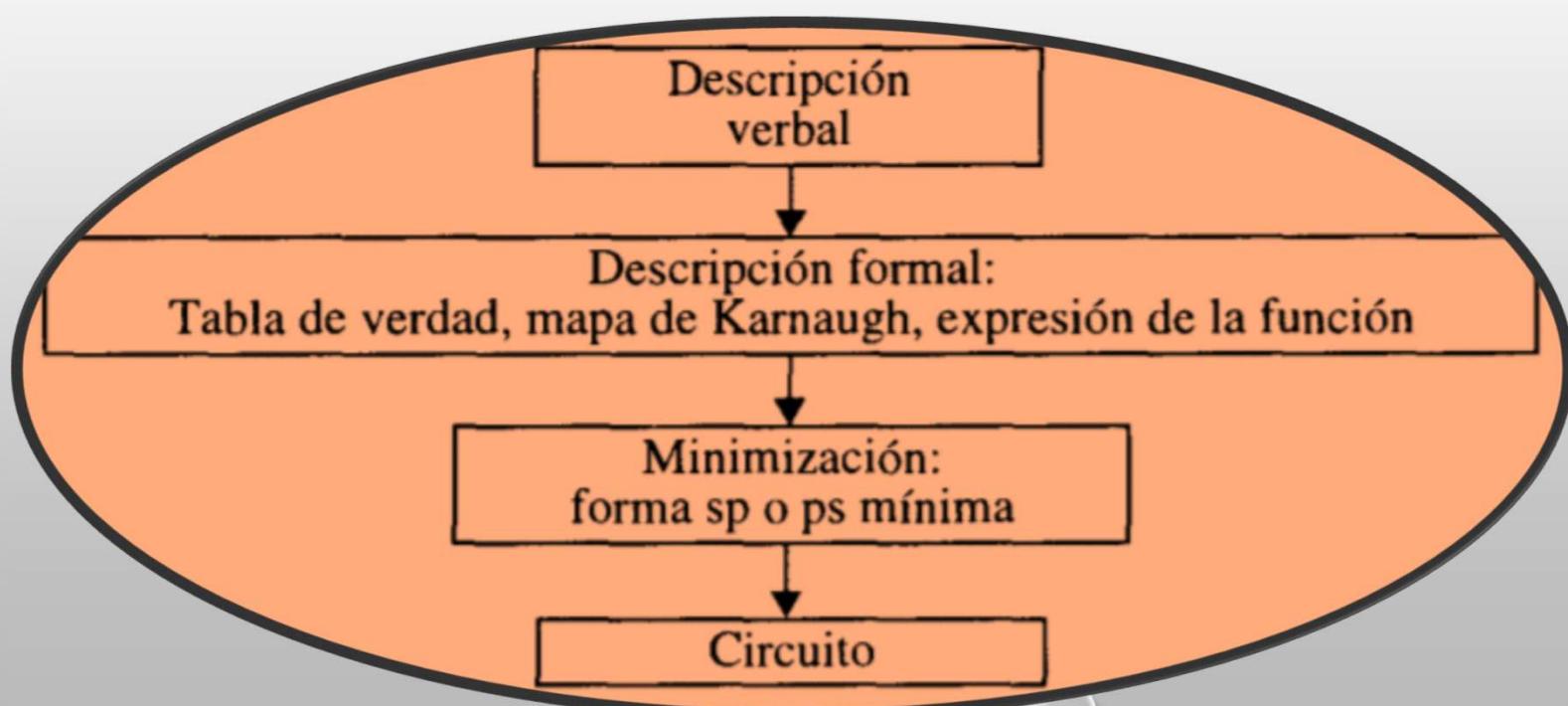
– Producto de maxtérminos:

$$\begin{aligned} f(a, b, c, d) &= (a + b + c + d) (a + b + \bar{c} + d) (a + b + \bar{c} + \bar{d}) (a + \bar{b} + c + d) \\ &\quad (a + \bar{b} + \bar{c} + \bar{d}) (\bar{a} + b + c + d) (\bar{a} + b + c + \bar{d}) (\bar{a} + \bar{b} + c + d) (\bar{a} + \bar{b} + c + \bar{d}) = \\ &= M_0 M_2 M_3 M_4 M_7 M_8 M_9 M_{12} M_{13} = \Pi (0, 2, 3, 4, 7, 8, 9, 12, 13). \end{aligned}$$

Encuentre la función de salida del siguiente circuito lógico combinacional

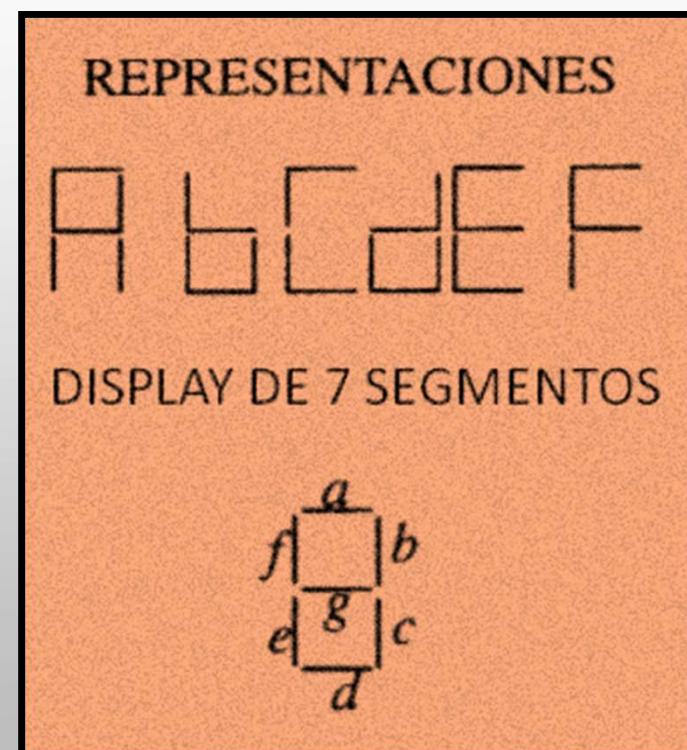


# PASOS DEL PROCESO DE DISEÑO



# Ejemplo

- Se desean visualizar las siguientes representaciones utilizando un display de 7 segmentos . Diseñe un circuito de tres entradas que encienda correctamente el segmento g.



# Solución.

- La función de salida toma los siguientes valores :

$g = 0$  , no enciende el LED .

$g = 1$  , enciende el LED .

	xy	00	01	11	10
z	0	1	0	d	1
	1	1	1	d	1

$$g = \bar{y} + z$$

	xy	00	01	11	10
z	0	1	0	x	1
	1	1	1	x	1

$$g = x + z$$

x y z	g
R → 0 0 0	1
B → 0 0 1	1
L → 0 1 0	0
D → 0 1 1	1
E → 1 0 0	1
F → 1 0 1	1

# Subsistemas combinacionales

# Características

- De acuerdo a la función que realiza estos se dividen en subsistemas de propósito específico y subsistemas de propósito general .
- Los primeros realizan funciones fijas, mientras que los segundos realizan cualquier función lógica mediante una "programación" interna o de sus entradas y salidas .
- De acuerdo a las entradas existen dos : las de control y las de datos .
  - Las primeras controlan la operación del dispositivo (Enable) .
  - las segundas corresponden a las variables independientes de las funciones que desarrollan .

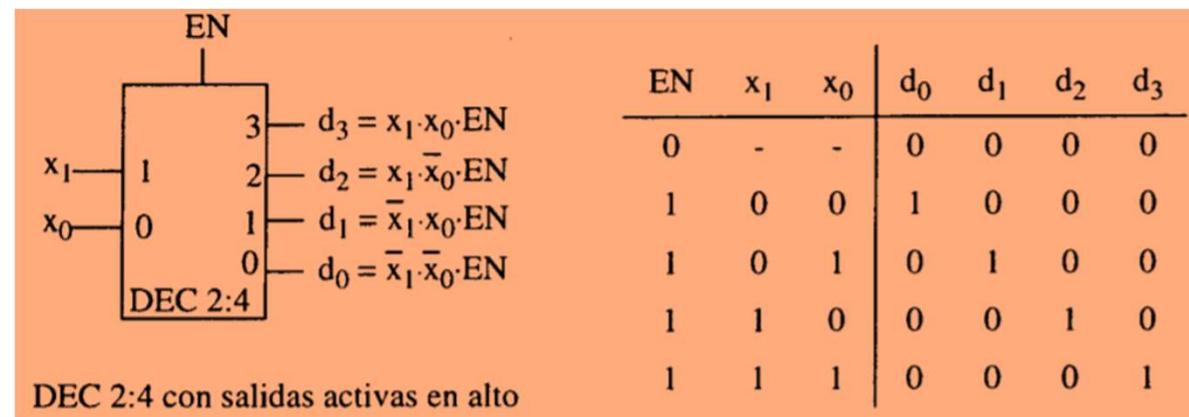
- cada entrada de dato suele poseer un peso asociado, de modo que las entradas no son intercambiables como ocurría en las compuertas .
- Respecto a las salidas, también existen las de control, que avisan de determinadas situaciones o estados en el que se encuentra el dispositivo, y las de datos, que son realmente las que dan respuesta al conjunto de entradas en cada instante .

# SUBSISTEMAS DE PROPÓSITO ESPECÍFICO

# Decodificador

- Es un dispositivo con n entradas y  $2^n$  salidas donde en función de la combinación binaria de sus entradas, una y sólo una de las salidas se activa . Convierte un código binario de entrada en código "1-entre-n" .

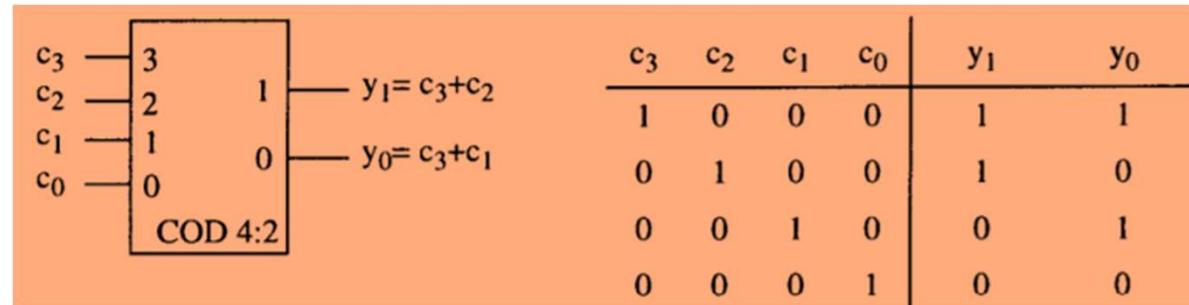
Ejemplo.



# Codificador

- Realiza la operación contraria al decodificador. Convierte el código "1-entre-n" en código binario. Un codificador completo posee  $2^n$  entradas, de las que sólo una puede estar activa, y  $n$  salidas que ofrecen la combinación binaria asociada a dicha entrada.

Ejemplo



## Codificador de prioridad.

- Es muy similar al codificador común, la diferencia es que sus entradas no necesitan estar en código "1-entre-n" ya que cada una de ellas tiene una prioridad sobre las otras, de forma que la salida es la codificación binaria asociada a la entrada de mayor prioridad que tenga el valor activo.  
Su tabla de verdad es :

$c_3$	$c_2$	$c_1$	$c_0$	$y_1$	$y_0$
1	-	-	-	1	1
0	1	-	-	1	0
0	0	1	-	0	1
0	0	0	1	0	0

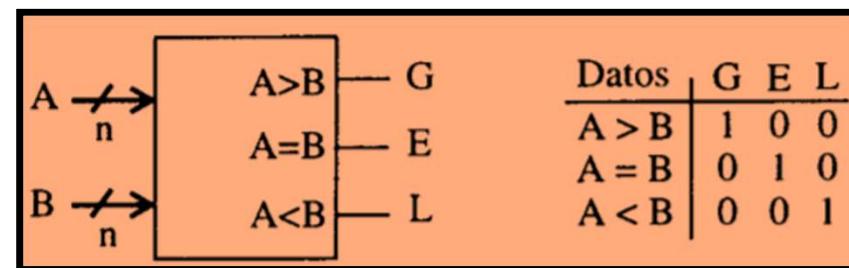
# Convertidor de código

- Consiste en un subsistema combinacional que convierte un código de entrada en otro de salida . El número de líneas de entrada y de salida depende de los códigos que se convierten. Los casos particulares en los que uno de los códigos sea "1-entre-n" son los dispositivos ya mencionados previamente.

# Comparador de magnitudes

- Es un dispositivo que compara las magnitudes de dos datos A y B de "n" bits, para dar como resultado si  $A > B$ ,  $A = B$  o  $A < B$  .

Ejemplo:

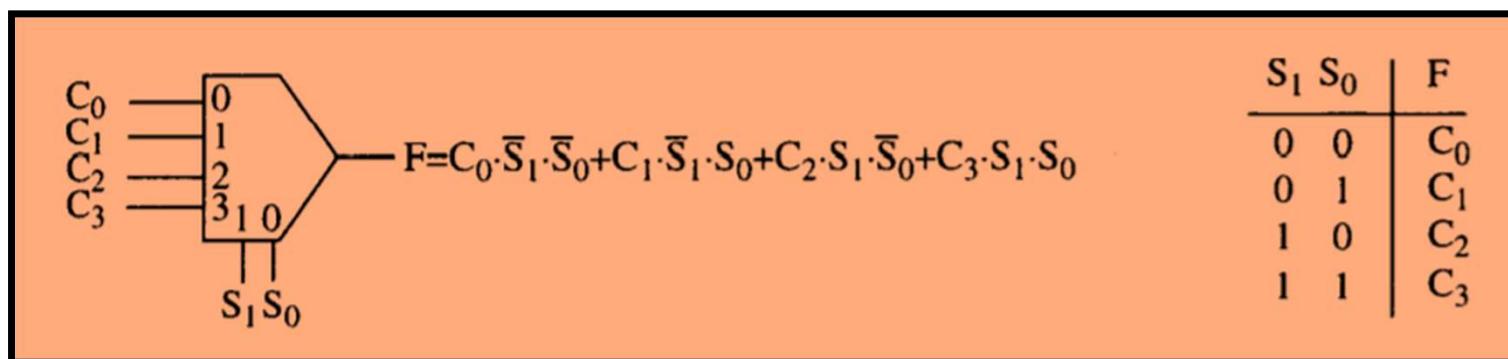


# SUBSISTEMAS DE PROPÓSITO GENERAL

# Multiplexor

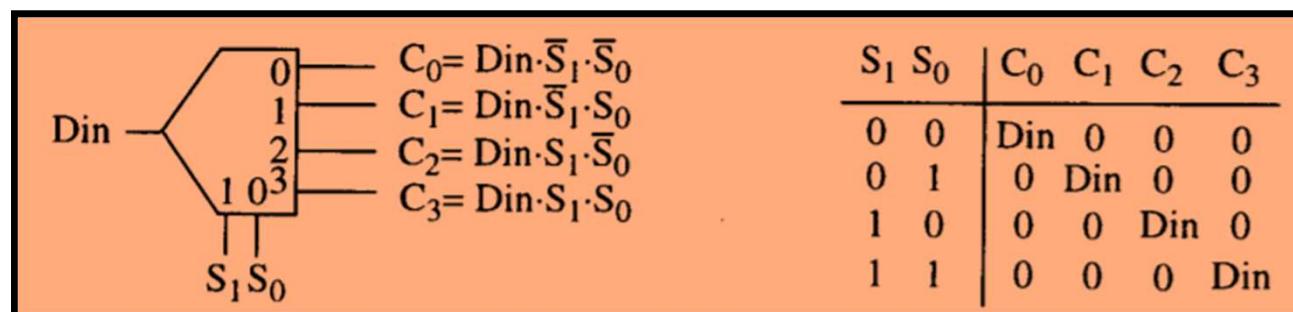
- Un MUX-n o MUX  $2^n$

Es un dispositivo de  $2^n$  canales de entrada (datos), n entradas de selección de canal y 1 salida. Su función es dejar pasar hacia la salida la información que entra por uno de sus canales de entrada, aquel que está seleccionado en función de la codificación binaria de las señales de selección. Un MUX-n es un módulo lógico universal de n variables o de  $n+1$  variables.



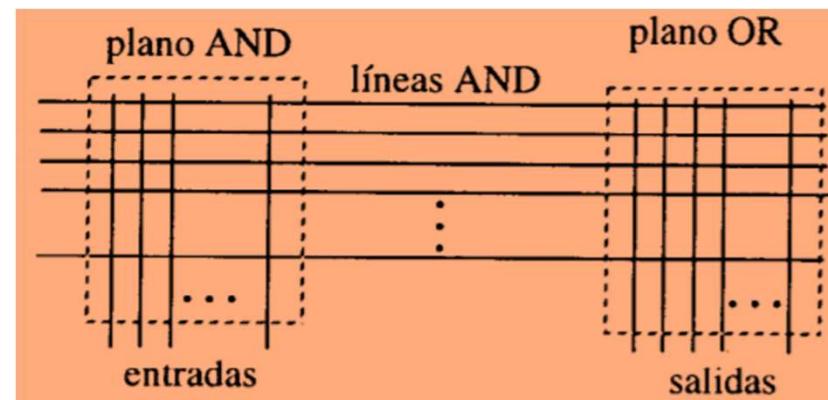
# Demultiplexor

- Realiza la función inversa al mux. Un *DEMUX-n* o *DEMUX 1 :2^n*, posee una entrada de dato, "n" entradas de selección y  $2^n$  líneas o canales de salida . Su función consiste en pasar la información de entrada de dato a una de las líneas de salida, la determinada por la combinación binaria de las señales de selección.



# Dispositivos Lógicos Programables

estructura general.



# Clasificación con base en el plano programable

plano AND	plano OR	
No programable	Programable	ROM
Programable	Programable	PLA
Programable	No programable	PAL

# ROM

- Una ROM( $2^n \times m$ ) posee ***n entradas de dirección*** y ***m salidas***, que puede verse como un dispositivo que almacena  $2^n$  palabras de  $m$  bits, de forma que para cada combinación binaria de sus ***n*** entradas se selecciona una de sus  $2^n$  palabras.
- En las ***m*** líneas de salida se lee la palabra almacenada. Del plano AND de una ROM se obtienen todos los mintérminos de las  $n$  variables de entrada, y en función de la programación del plano OR, se eligen los que interesen para realizar la función lógica que se desee . Por tanto, una ROM es un dispositivo lógico universal de ***n*** variables para ***m*** funciones.

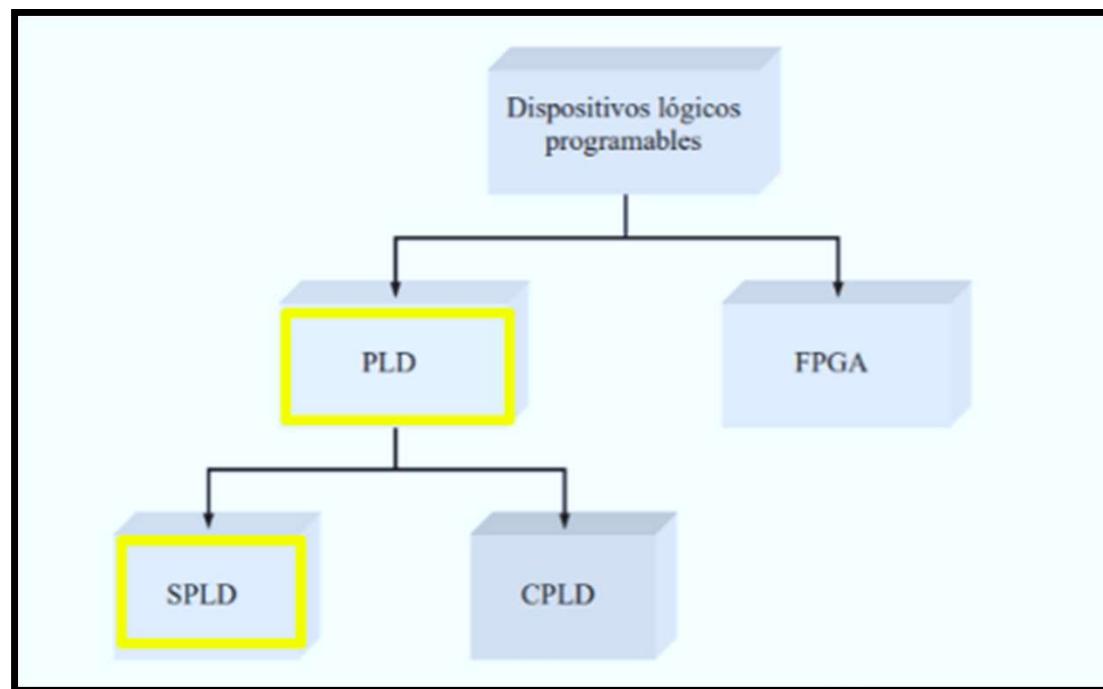
# PLA

- Un PLA es un subsistema con **n** entradas, **m** salidas y **p** términos productos (salidas del plano AND) . Mediante este dispositivo pueden implementarse m funciones lógicas de n variables expresadas en sumas de productos si para ello no se superan los p términos ANDs disponibles .

# PAL

- En este dispositivo cada salida es la OR de un conjunto determinado de líneas AND, no estando compartidas ninguna de ellas por otra salida . La implementación de una función con este dispositivo es similar al caso anterior, salvo que en el PAL cada función de salida se trata independientemente de las otras

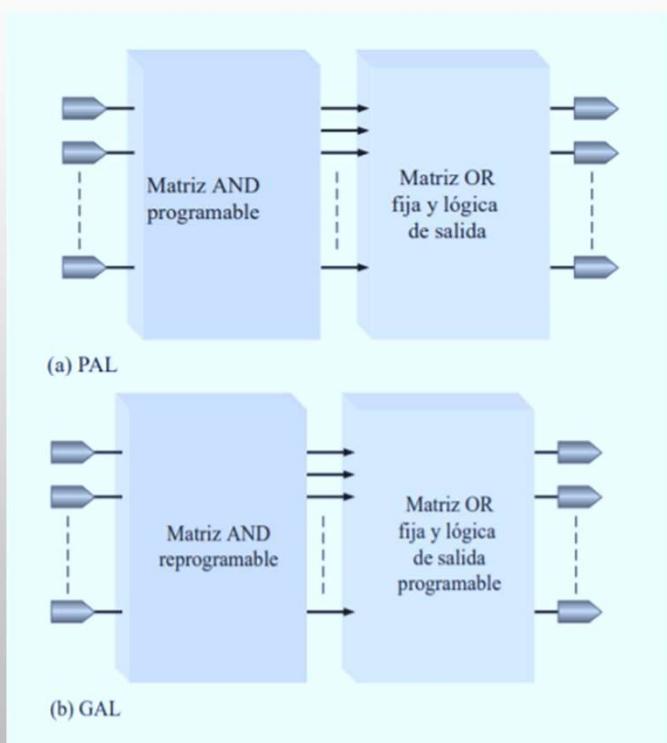
- Todos los casos anteriores corresponden a los SPLD , faltaría mencionar el dispositivo que se utilizará en el transcurso del semestre, GAL.



# PAL ----GAL

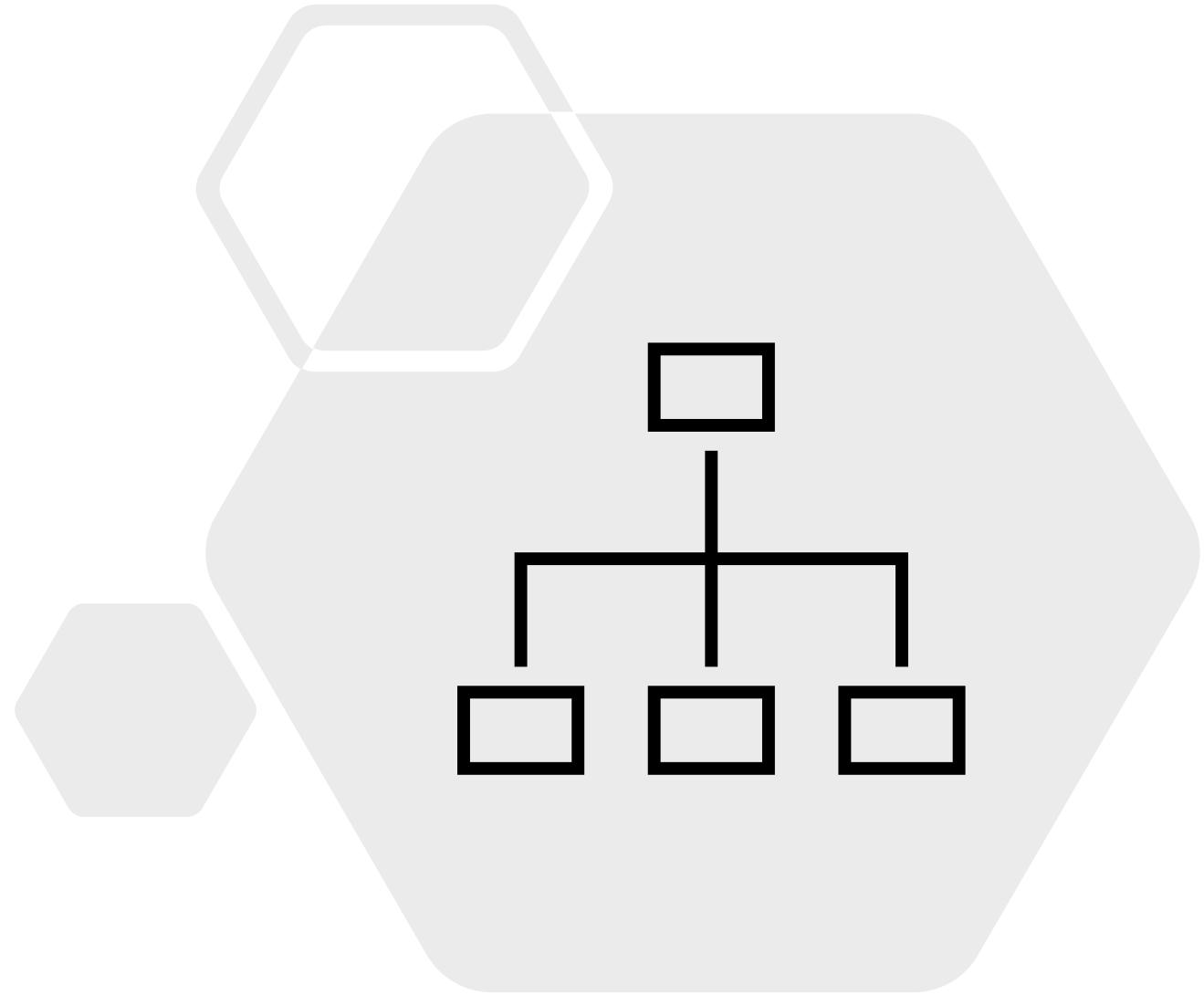
- los tres tipos de dispositivos entran en SPLD (Simple Programmable Logic Device).
- Generalmente, un SPLD puede reemplazar a diez CI de función fija y sus interconexiones, dependiendo del tipo de funciones y del SPLD específico.
- La mayoría de los SPLD pertenecen a una de dos posibles categorías: PAL y GAL.

- Una PAL (Programmable Array Logic, matriz lógica programable) es un dispositivo que se puede programar una vez. Consta de una matriz programable de puertas AND y una matriz fija de puertas OR.
- Los PLA (Programmable Logic Array) al igual que los PAL, estos disponen de dos planos diferenciados: AND y OR. Los PLA tienen ambos planos programables lo que hace que su estructura sea ideal para implementar funciones lógicas como sumas de productos, por el contrario, hace que el dispositivo tenga mayor tamaño y menor velocidad.
- Una GAL (Generic Array Logic, matriz lógica genérica) es un dispositivo que es básicamente una PAL que puede reprogramarse muchas veces. Consta de una matriz reprogramable de puertas AND y de una matriz fija de puertas OR con salidas programables (es la versión mejorada de los PLA).



- Encapsulado típico de un SPLD, que normalmente dispone de entre 24 y 28 pines.

< VHDL >



# VHSIC - Hardware Description Language

Lenguaje de descripción de hardware para Circuitos Integrados de muy Alta Velocidad

Es un acrónimo compuesto: la V es el acrónimo de VHSIC (Very High Speed Integrated Circuit) y HDL es otro acrónimo de Hardware Description Language.

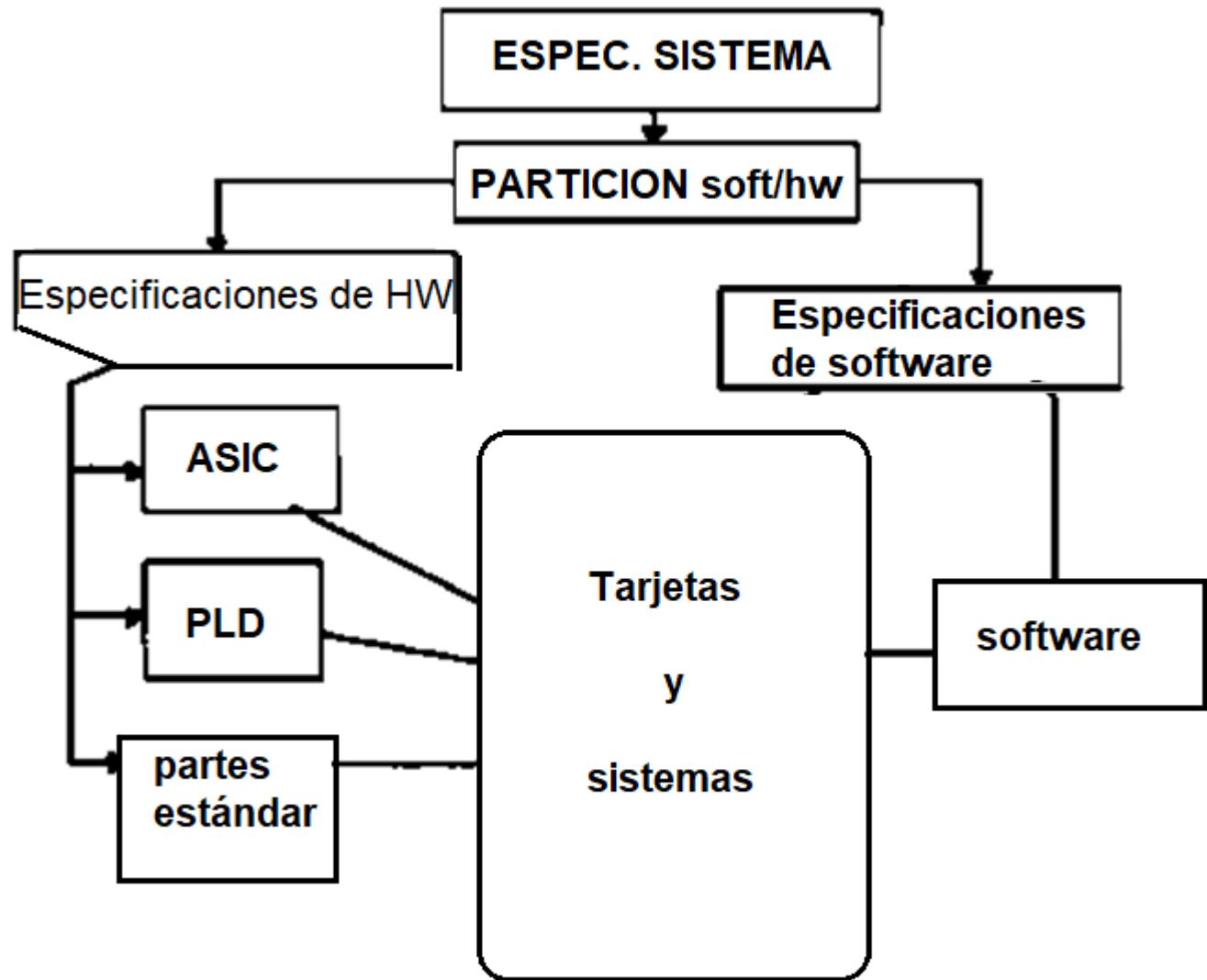
# HISTORIA

- Desarrollado a principios de los 80's
- El lenguaje VHDL se desarrolló en los 80's para describir sistemas electrónicos, buscando fueran una descripción independiente del método de implementación. Fue desarrollado bajo pedido del gobierno de USA.
- IEEE 1076 :1987, & 1993
  - IEEE 1076:1987
  - Nuevo: 1993
- Las construcciones del VHDL fueron definidas como estándar con número 1076 por la IEEE en 1987. Una nueva revisión fue definida en 1993, y se añadieron algunas nuevas capacidades, eliminando ambigüedades y manteniendo compatibilidad con la versión anterior.

# Implementación de HW

A inicios de los 90's, el VHDL fue usado principalmente para diseñar ASIC(Circuitos integrados de Aplicación Específica) complejos, usando herramientas de síntesis que automáticamente crean y optimizan la implementación.

A mediados de esos años la corriente principal del uso del VHDL se dirigió hacia la síntesis de diseño de lógica programable PLD.



# LIMITACIONES

Analógico o digital

- VHDL es un lenguaje principalmente adecuado para usarse en el diseño digital.

Estilos

- El estándar IEEE 1076 define las construcciones del lenguaje y su sintaxis sin describir ningún estilo de cómo debe ser usado en el diseño de proyectos.

- Únicamente es posible sintetizar lógica desde un subconjunto de las construcciones del VHDL.

# ESTILO DE MODELADO EN VHDL.

- ¿Los diferentes estilos de escribir código VHDL se liga con un concepto llamado **abstracción**. El nivel de abstracción define con cuánto detalle es especificado un diseño para una descripción en particular. Existen cuatro niveles de abstracción.

1

Nivel Arreglo

- Se requiere especificar información acerca de la disposición o colocado del diseño en el Silicio, especificar información detallada de tiempos o de efectos analógicos.

3

Nivel RT o RTL

- El uso de VHDL se apega a estilos estrictos para definir cada uno de los registros en el diseño y la lógica combinacional entre ellos.

2

Nivel lógico

- La información del nivel disposición y los efectos analógicos no se toman en cuenta

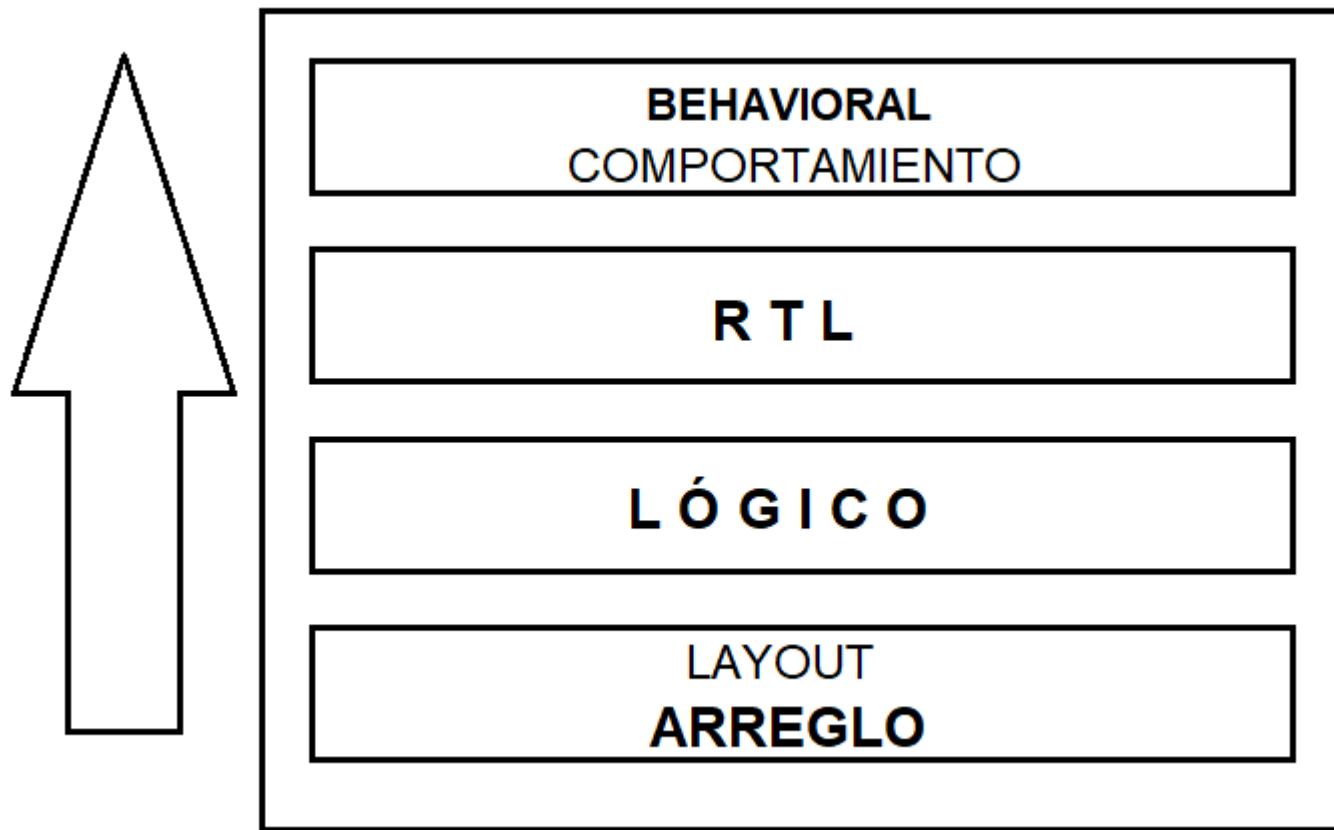
4

Nivel de Comportamiento o funcional.

- Este nivel usa VHDL para describir el funcionamiento del diseño sin especificar detalles de la arquitectura entre registros. El código comportamiento puede contener tanta información de tiempos como el diseñador lo requiera para representar su función.

# Comportamiento vs RTL

- De lo anterior podemos ver que existen al menos dos estilos diferentes de escribir VHDL
  - BEHAVIORAL vs RTL

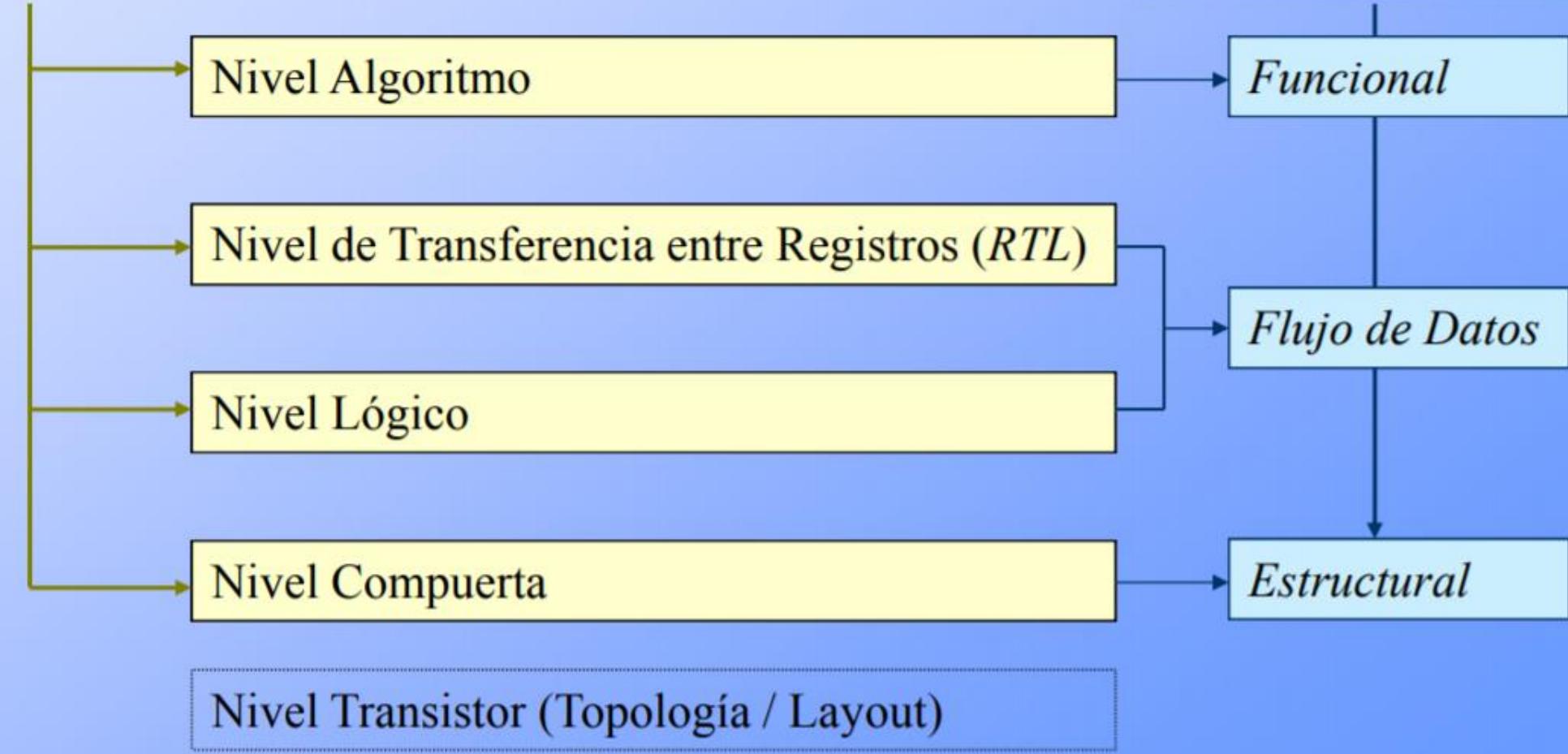


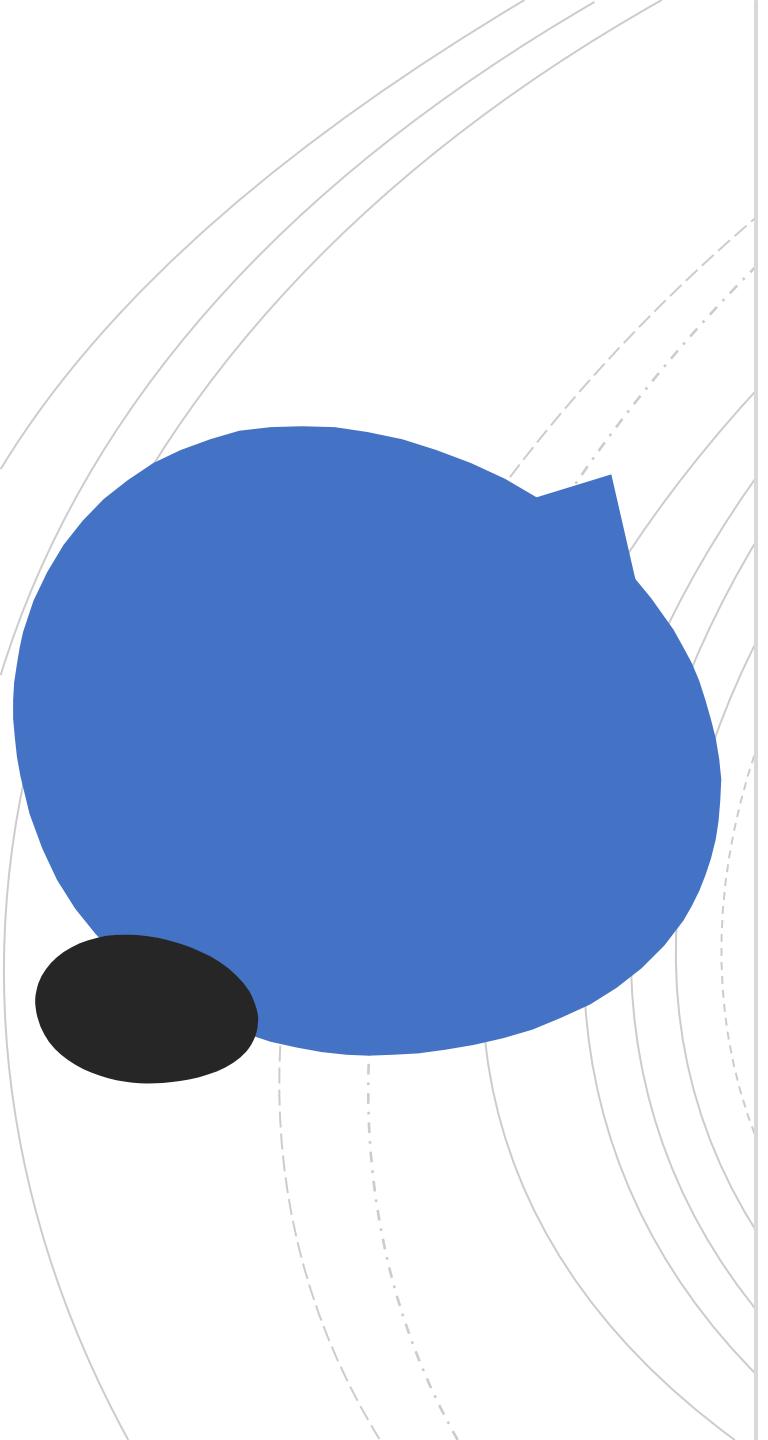
# ¿Cuándo usar uno u otro?

- La mayoría de código escrito en VHDL lo es a nivel RTL por la sencilla razón de que prácticamente todas las herramientas de síntesis requieren que así sea escrito. A este nivel el diseñador mantiene control sobre la arquitectura de los registros que intervienen en su diseño.
- Las herramientas de síntesis-comportamiento automáticamente generan arquitecturas de registros y lógica desde una descripción 'VHDL comportamiento'. Actualmente ya están disponibles para prueba varias de estas herramientas y lo común es usarlas cuando se implementan algoritmos para aplicaciones de procesamiento de señales.

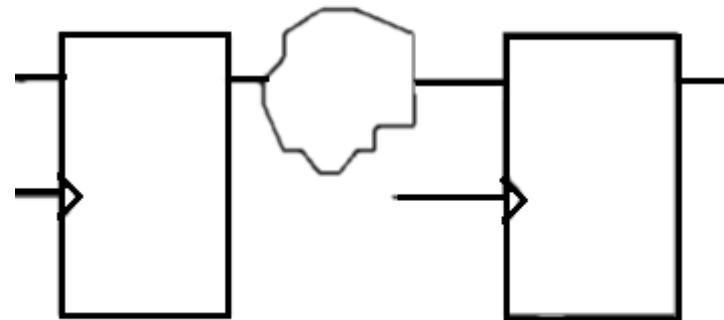
## Niveles de *Descripción* utilizados

## Estilo de descripción o Modelización

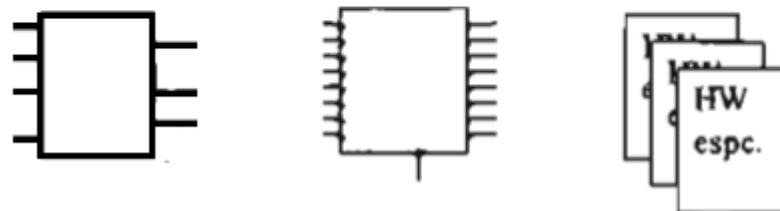




VHDL - RTL  
PARA SÍNTESIS

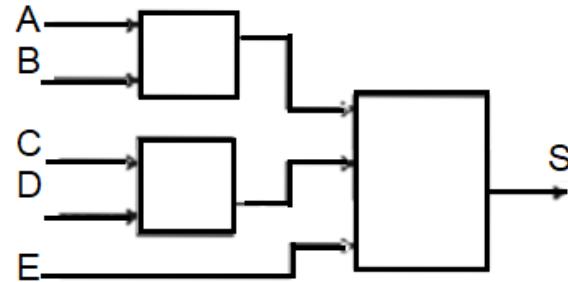


VHDL - BEHAVIORAL  
PARA ESTÍMULOS, PARTES  
ESTANDAR, ESPECIFICACIONES

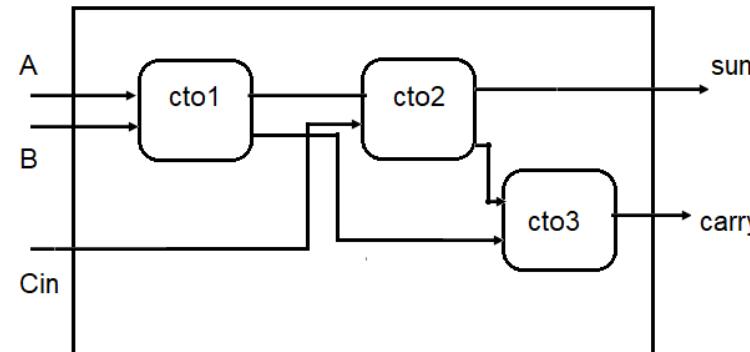


# Conceptos principales en el diseño por VHDL

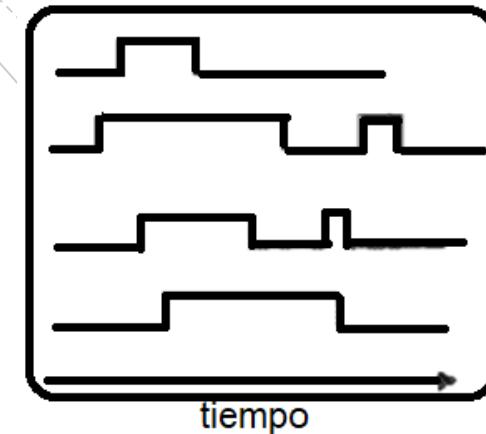
## Concurrencia



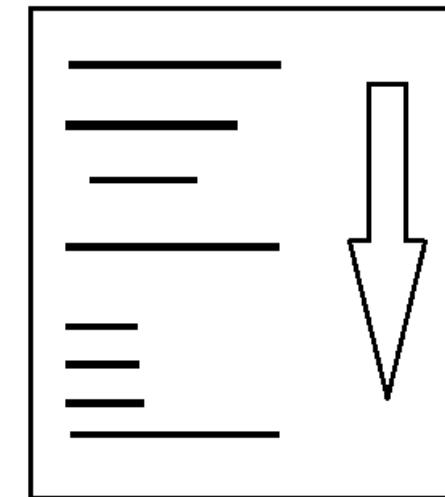
## Estructura



Tiempo



## Secuencial



LIBRARY	ENTITY	ARCHITECTURE
Contiene una lista de todas las bibliotecas que se utilizarán en el diseño  EJ. ieee, std, work, etc.	Especifica los pines de E / S del circuito	Contiene el código VHDL ,propriamente dicho, que describe cómo el circuito debe comportarse (función).



Unidades fundamentales en VHDL

```
LIBRARY library_name;  
USE library_name.package_name.package_parts;
```

Al menos necesitas 3  
packages\* en el diseño

ieee.std\_logic\_1164 (de *ieee library*)

standard (de *std library*)

work (*work library*)

Library

**LIBRARY**  
declarations

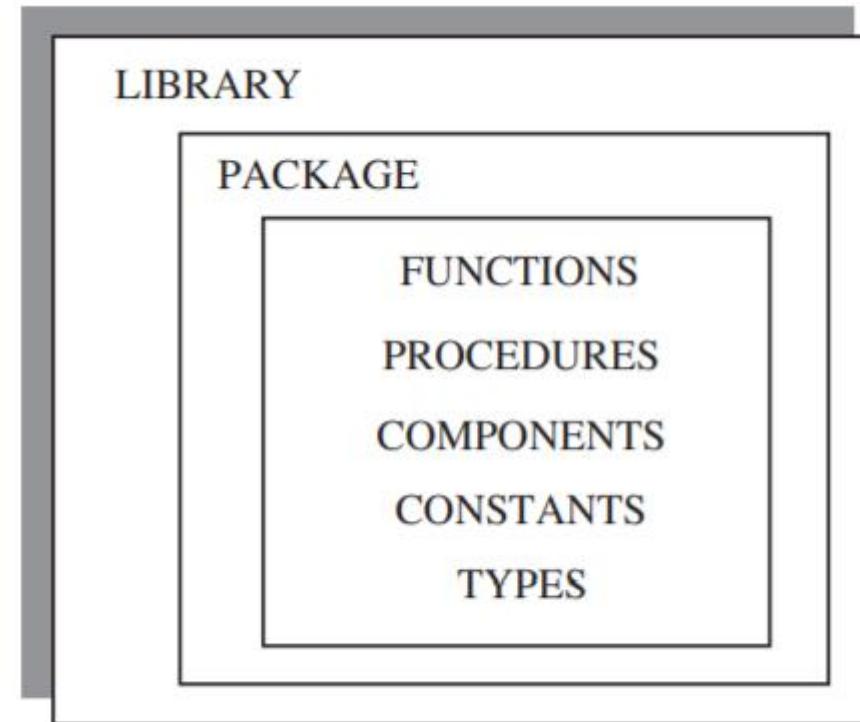
**ENTITY**

**ARCHITECTURE**



Código básico  
en VHDL

# Partes fundamentales de Library



```
ENTITY entity_name IS
  PORT (
    port_name : signal_mode signal_type;
    port_name : signal_mode signal_type;
    ...);
END entity_name;
```

# ENTITY

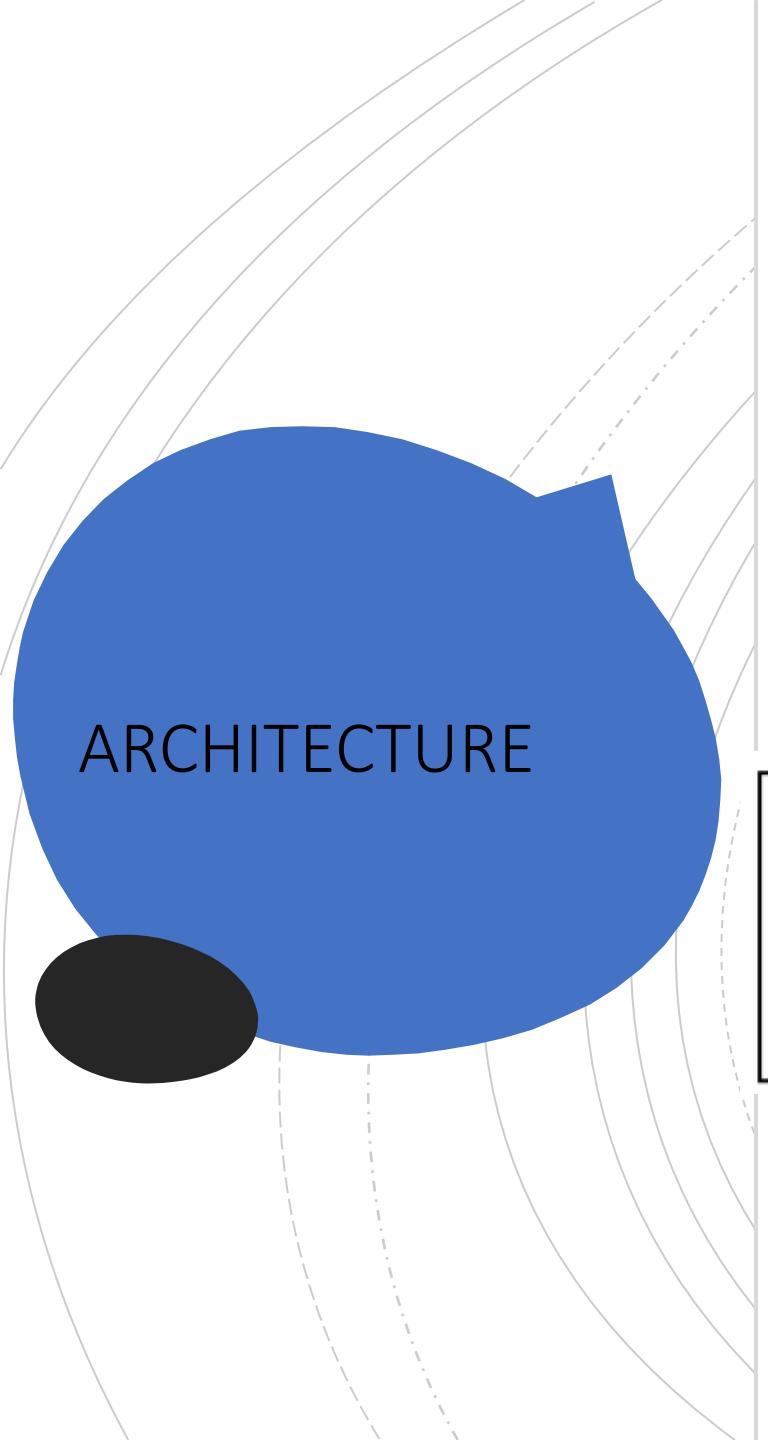
***mode*** puede ser:

IN, OUT, INOUT o BUFFER

IN y OUT son unidireccionales , INOUT is bidireccional.  
BUFFER se utiliza cuando la señal de salida debe ser  
leída de manera interna .

Los tipos de señales pueden ser BIT, STD\_LOGIC,  
INTEGER, etc.

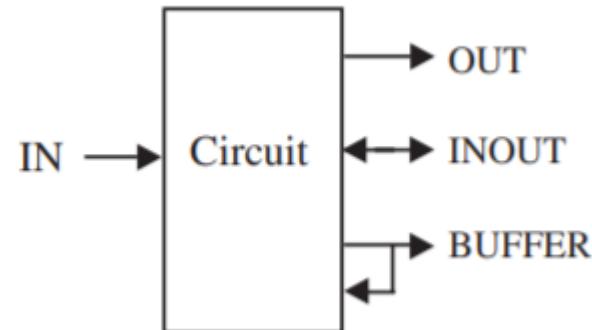
El nombre de la entidad puede llevar cualquier  
nombre , excepto palabras reservadas de VHDL



Una arquitectura tiene dos partes:  
Una parte declarativa (opcional), donde se declaran las señales y constantes (entre otras), y la parte del código (de BEGIN abajo). Como en el caso de una entidad, el nombre de una arquitectura puede ser básicamente cualquier nombre (excepto las palabras reservadas de VHDL), incluido el mismo nombre que

```
ARCHITECTURE architecture_name OF entity_name IS
  [declarations]
BEGIN
  (code)
END architecture_name;
```

# Entidad y arquitectura



```
ENTITY nand_gate IS
  PORT (a, b : IN BIT;
        x : OUT BIT);
END nand_gate;
```



```
ARCHITECTURE myarch OF nand_gate IS
BEGIN
  x <= a NAND b;
END myarch;
```

# Ejemplo

```
library ieee;
use ieee.std_logic_1164.all;

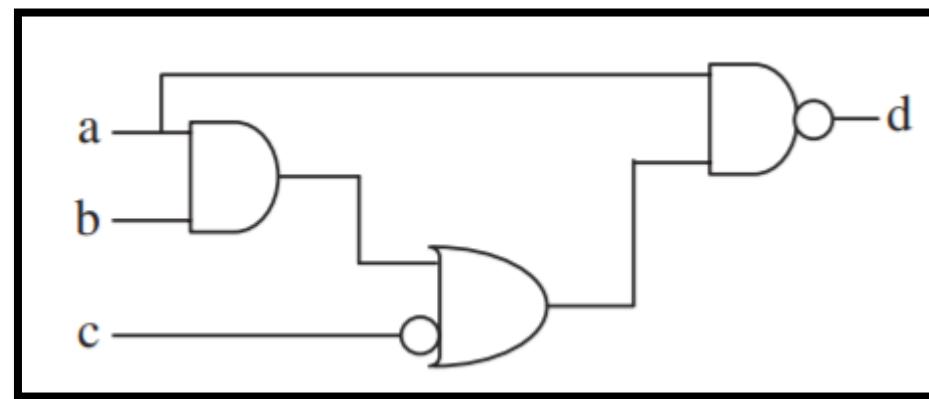
ENTITY example IS
    PORT ( a, b, clk: IN BIT;
           q: OUT BIT);
END example;

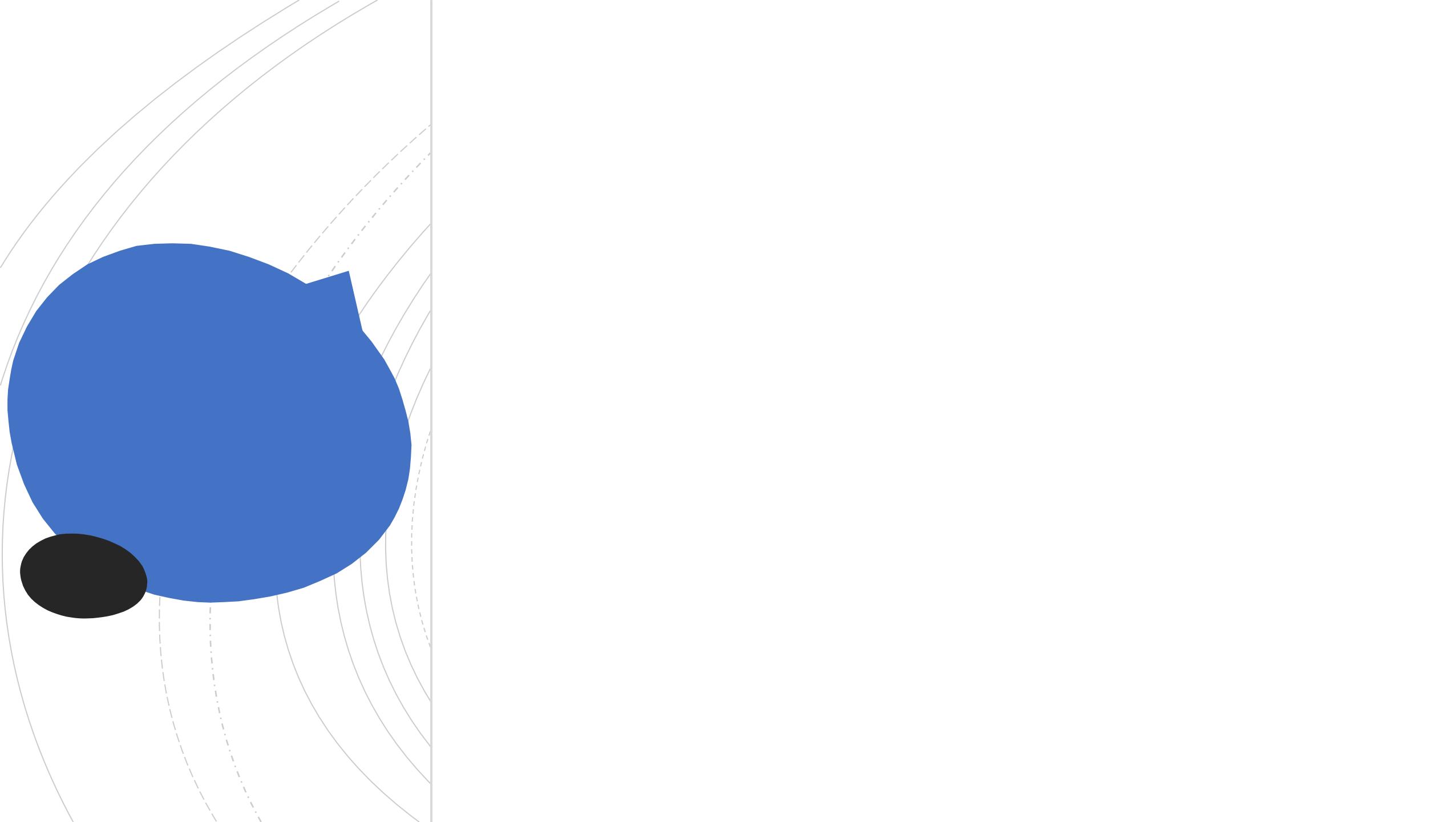
ARCHITECTURE example OF example IS
    SIGNAL temp : BIT;
BEGIN
    temp <= a NAND b;
PROCESS (clk)
BEGIN
    IF (clk'EVENT AND clk='1')
    THEN q<=temp;
    END IF;
END PROCESS;
END example;
```

# Ejercicio 1

## Circuito Combinacional en VHDL

- Escribe el Código en VHDL para el siguiente circuito. Al ser un circuito puramente Combinacional no es necesario el uso de PROCESS. Escribe la función para D usando los operadores lógicos que ya conoces (AND, OR, NAND, NOT, etc.).







Tipos de Datos

Data Types



Para escribir código VHDL de manera eficiente, es esencial saber qué tipos de datos son permitido, como especificarlos y utilizarlos.



# Tipos de datos predefinidos

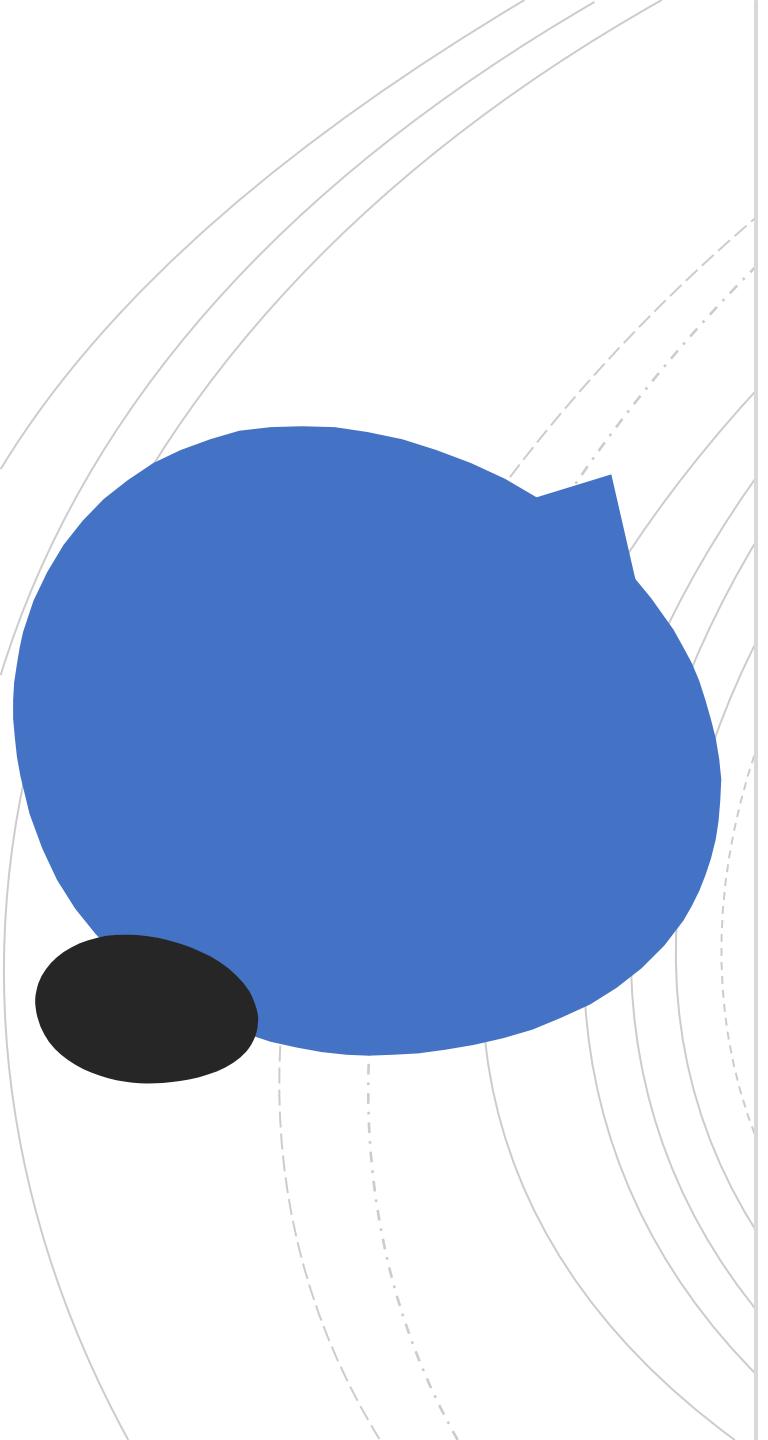
- VHDL contiene una serie de tipos de datos predefinidos, especificados a través de IEEE 1076 y estándares IEEE 1164. Más específicamente, se pueden encontrar tales definiciones de tipos de datos en los siguientes paquetes / bibliotecas:

- Package *standard* of library *std*:

BIT, BOOLEAN, INTEGER, and REAL

- Package *std\_logic\_1164* of library *ieee*:

STD\_LOGIC and STD\_ULOGIC

- 
- Package *std\_logic\_arith* of library *ieee*:  
SIGNED and UNSIGNED  
*conv\_integer(p)*, *conv\_unsigned(p, b)*, *conv\_signed(p, b)*,  
and *conv\_std\_logic\_vector(p, b)*.
  - Packages *std\_logic\_signed* and *std\_logic\_unsigned* of library *ieee*:  
STD\_LOGIC\_VECTOR  
SIGNED or UNSIGNED

**bit**

```
SIGNAL x: BIT;
SIGNAL y: BIT_VECTOR (3 DOWNTO 0);
SIGNAL w: BIT_VECTOR (0 TO 7);
```

Ejemplo:

```
x <= '1';
y <= "0111";
w <= "01110001";
```

## std\_logic

'X'	Forcing Unknown
'0'	Forcing Low
'1'	Forcing High
'Z'	High impedance
'W'	Weak unknown
'L'	Weak low
'H'	Weak high
'_'	Don't care

Ejemplo:

```
SIGNAL x: STD_LOGIC;
SIGNAL y: STD_LOGIC_VECTOR (3 DOWNTO 0) := "0001";
```

BIT  
(BIT\_VECTOR)  
|  
STD\_LOGIC  
(STD\_LOGIC\_VECTOR)

# Objetos

- Un objeto en VHDL es un elemento con nombre que contiene el valor de un tipo de datos específico.

- Hay cuatro tipos de objetos: **signal**, **variable**, **constant** y **file**.

- Una construcción conocida como ***alias*** es algo así como un objeto.
- ***File*** no puede ser sintetizable.

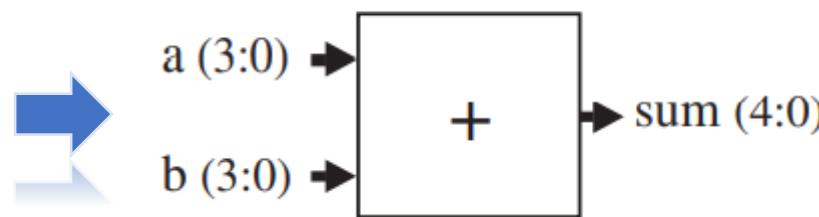
# SIGNED y UNSIGNED

(Data types IEEE numeric\_std package)

- Tipos de datos definidos en el paquete **std\_logic\_arith** de la biblioteca **ieee**.
- Tienen la apariencia de **STD\_LOGIC\_VECTOR**, pero aceptan operaciones aritméticas, que son típicas de los tipos de datos **INTEGER**

# Ejemplo del uso de datos SIGNED

Sumador de n bits



Código en VHDL

```
1
2 -----sumador ejemplo -----
3 LIBRARY ieee;
4 USE ieee.std_logic_1164.all;
5 USE ieee.std_logic_arith.all;
6 -----
7 ENTITY adder1 IS
8 PORT ( a, b : IN SIGNED (3 DOWNTO 0);
9      sum : OUT SIGNED (3 DOWNTO 0));
10 END adder1;
11 -----
12 ARCHITECTURE add OF adder1 IS
13 BEGIN
14     sum <= a + b;
15 END add;
16 -----
```

# Solución 2 usando INTEGER en la salida

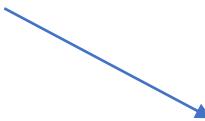
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY adder2 IS
    PORT ( a, b : IN SIGNED (3 DOWNTO 0);
           sum : OUT INTEGER RANGE -16 TO 15);
END adder2;

ARCHITECTURE adder2 OF adder2 IS
BEGIN
    sum <= CONV_INTEGER(a + b);
END adder2;
```

## Operaciones legales e ilegales entre datos de diferentes tipos

```
SIGNAL a: BIT;  
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);  
SIGNAL c: STD_LOGIC;  
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);  
SIGNAL e: INTEGER RANGE 0 TO 255;
```



```
a <= b(5);  
b(0) <= a;  
c <= d(5);  
d(0) <= c;  
a <= c;  
b <= d;  
  
e <= b;  
e <= d;
```

# Tipos de datos definidos por el usuario

- *integer* types:

```
TYPE integer IS RANGE -2147483647 TO +2147483647;  
TYPE natural IS RANGE 0 TO +2147483647;  
TYPE my_integer IS RANGE -32 TO 32;  
TYPE student_grade IS RANGE 0 TO 100;
```

- *enumerated* types:

```
TYPE bit IS ('0', '1');  
TYPE my_logic IS ('0', '1', 'Z');  
TYPE bit_vector IS ARRAY (NATURAL RANGE <>) OF BIT;  
TYPE state IS (idle, forward, backward, stop);  
TYPE color IS (red, green, blue, white);
```

La codificación de los tipos enumerados se realiza de forma secuencial y automática (a menos que sea especificado de otra manera por un atributo definido por el usuario). Ej, se necesitan dos bits (hay cuatro estados), siendo "00" asignado al primer estado (rojo), "01" al segundo (verde), "10" al siguiente (azul), y finalmente "11" hasta el último estado (blanco).

# Subtypes

UN SUBTYPE es un TYPE con una restricción. La principal razón para usar un SUBTIPO en lugar de especificar un nuevo TIPO es que, aunque las operaciones entre diferentes tipos de datos no están permitidas, están permitidos entre un SUBTYPE y su base correspondiente TYPE.

# Ejemplos

```
SUBTYPE natural IS INTEGER RANGE 0 TO INTEGER'HIGH;  
-- NATURAL is a subtype (subset) of INTEGER.
```

```
SUBTYPE my_logic IS STD_LOGIC RANGE '0' TO 'Z';  
-- STD_LOGIC= ('X', '0', '1', 'Z', 'W', 'L', 'H', '-').  
-- my_logic= ('0', '1', 'Z').
```

```
SUBTYPE my_color IS color RANGE red TO blue;  
-- color=(red, green, blue, white)  
-- my_color=(red, green, blue).
```

```
SUBTYPE small_integer IS INTEGER RANGE -32 TO 32;  
-- subtype of INTEGER.
```

-- applies to INTEGER

SUBTYPE small\_integer IS INTEGER RANGE -32 TO 32;

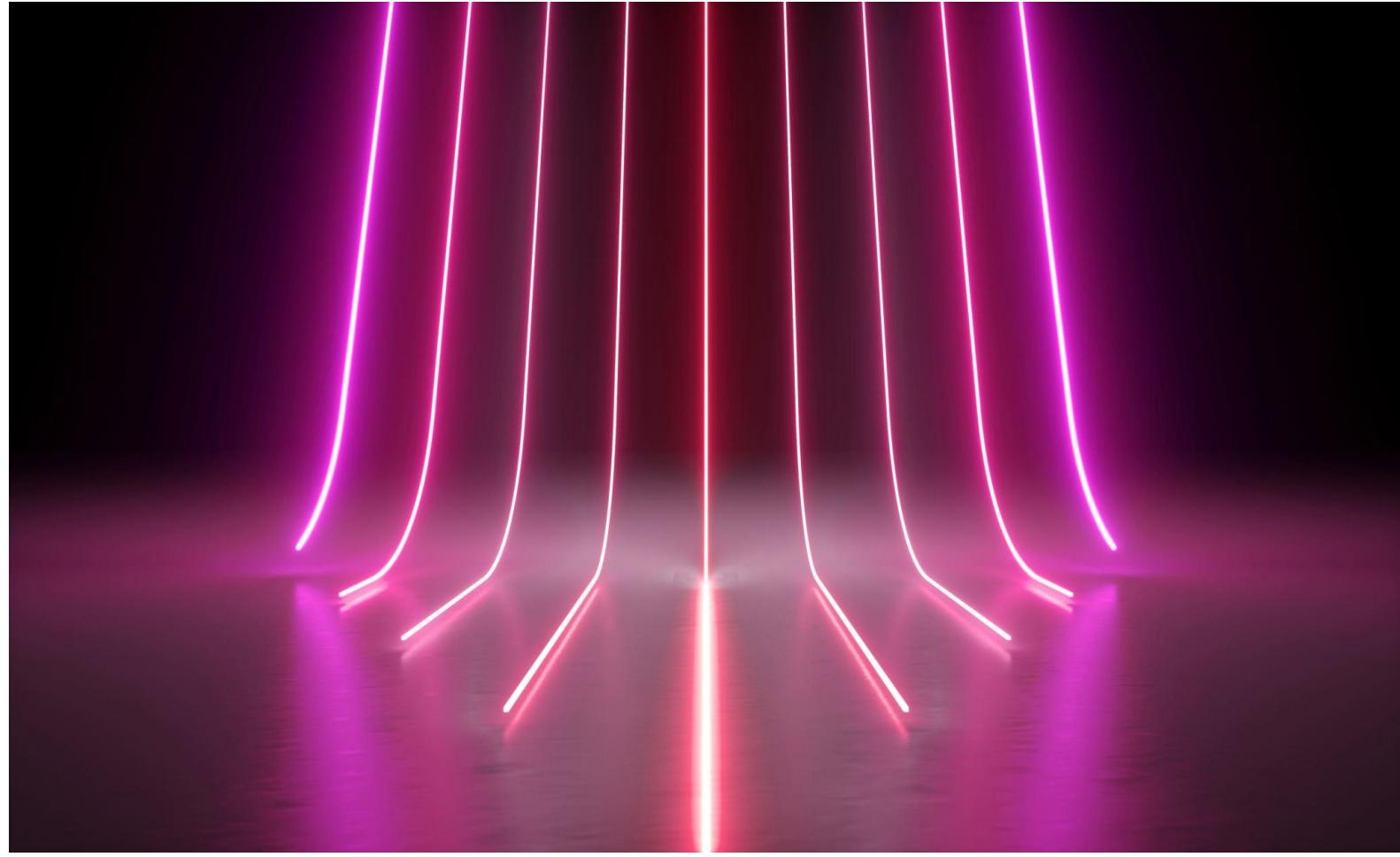
## Declaraciones ilegales entre subtipos

```
SUBTYPE my_logic IS STD_LOGIC RANGE '0' TO '1';
SIGNAL a: BIT;
SIGNAL b: STD_LOGIC;
SIGNAL c: my_logic;
```

```
b <= a;
b <= c;
```

# Tipos de datos sintetizables en VHDL

Data types	Synthesizable values
BIT, BIT_VECTOR	'0', '1'
STD_LOGIC, STD_LOGIC_VECTOR	'X', '0', '1', 'Z' (resolved)
STD_ULOGIC, STD_ULOGIC_VECTOR	'X', '0', '1', 'Z' (unresolved)
BOOLEAN	True, False
NATURAL	From 0 to +2, 147, 483, 647
INTEGER	From -2,147,483,647 to +2,147,483,647
SIGNED	From -2,147,483,647 to +2,147,483,647
UNSIGNED	From 0 to +2,147,483,647
User-defined integer type	Subset of INTEGER
User-defined enumerated type	Collection enumerated by user
SUBTYPE	Subset of any type (pre- or user-defined)
ARRAY	Single-type collection of any type above
RECORD	Multiple-type collection of any types above



Operators  
and  
Attributes

# Operadores

Tipos de operadores predefinidos

- Assignment operators
- Logical operators
- Arithmetic operators
- Relational operators
- Shift operators
- Concatenation operators

# Assignment Operators

<= asigna valor a una SIGNAL.

**:=** asigna valor a VARIABLE, CONSTANT, o GENERIC.  
Además, se utiliza **para** para asignar valores iniciales.

=> asigna valores a elementos vectoriales individuales o con otros.

## Ejemplo

```
x <= '1';
y := "0000";
w <= "10000000";
w <= (0 =>'1', OTHERS =>'0');
```

# Logical Operators

- NOT
- AND
- OR
- NAND
- NOR
- XOR
- XNOR

Ejemplos.

```
y <= NOT a AND b;      -- (a'.b)
y <= NOT (a AND b);    -- (a.b)'
y <= a NAND b;         -- (a.b)'
```

# Arithmetic Operators

- Pueden ser del Tipo INTEGER, SIGNED, UNSIGNED, or REAL.

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation
MOD	Modulus
REM	Remainder
ABS	Absolute value

# Comparison Operators

=	Equal to
/=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

# Shift Operators

- Fueron introducidos en VHDL-93.
- Su sintaxis es la siguiente: < operando izquierdo> < operación de desplazamiento> < operando derecho>. El operando izquierdo debe ser de tipo BIT\_VECTOR, mientras que el operando derecho debe ser un INTEGER.

Los tipos son:

- sll Shift left logic
- srl Shift right logic

- `s'EVENT`: Devuelve *true* cuando ocurre un evento en `s`
- `s'STABLE`: Devuelve *true* si no ocurre un evento en `s`
- `s'ACTIVE`: Devuelve *true* si `s = '1'`
- `s'QUIET <time>`: Devuelve *true* si no ha ocurrido ningún evento durante el tiempo especificado
- `s'LAST_EVENT`: Devuelve al tiempo transcurrido desde el último evento
- `s'LAST_ACTIVE`: Devuelve al tiempo transcurrido desde la última `s = '1'`
- `s'LAST_VALUE`: Devuelve al ultimo valor de `s` antes del ultimo evento.

### Ejemplos:

```
IF (clk'EVENT AND clk='1')....  
  
IF (NOT clk'STABLE AND clk='1')....  
  
WAIT UNTIL (clk'EVENT AND clk='1');  
  
IF RISING_EDGE(clk)....
```

## Signal Attributes

Los atributos de datos predefinidos y sintetizables son los siguientes:

- d'LOW: Devuelve el índice de matriz inferior
- d'HIGH: Devuelve el índice de matriz superior
- d'LEFT: Devuelve el índice de matriz más a la izquierda
- d'RIGHT: Devuelve el índice de matriz más a la derecha
- d'LENGTH: Devuelve el tamaño del vector
- d'RANGE: Devuelve el rango vectorial
- d'REVERSE\_RANGE: Devuelve el rango vectorial en orden inverso

Ejemplos:

```
SIGNAL d : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
d'LOW=0, d'HIGH=7, d'LEFT=7, d'RIGHT=0, d'LENGTH=8,  
d'RANGE=(7 downto 0), d'REVERSE_RANGE=(0 to 7).
```



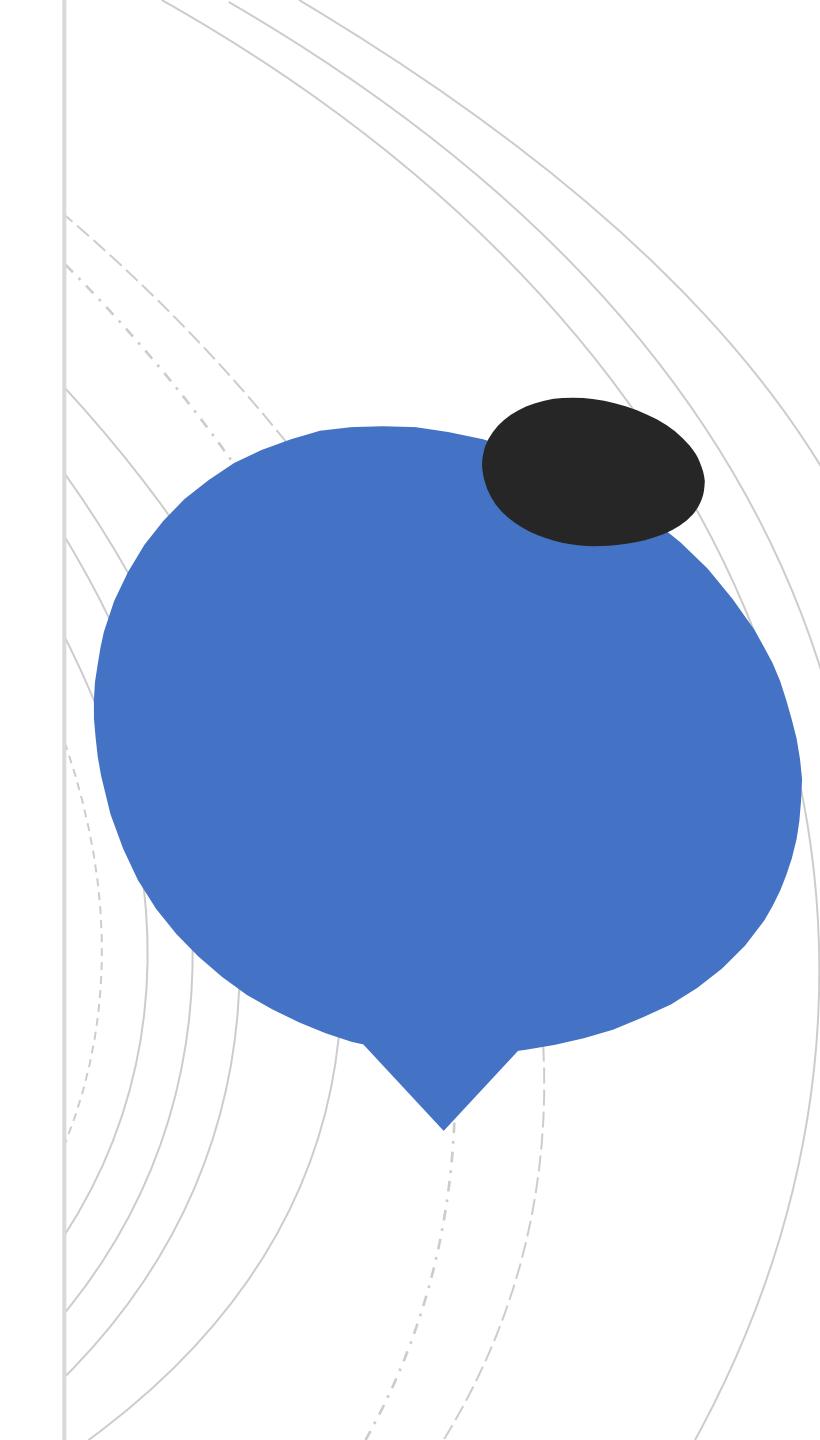
Data  
Attributes

Ejemplos:

```
SIGNAL d : STD_LOGIC_VECTOR ( 7 DOWNTO 0 );
```

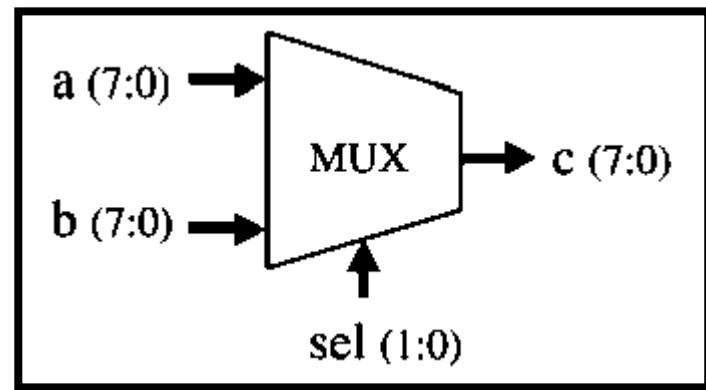


```
FOR i IN RANGE ( 0 TO 7 ) LOOP ...
FOR i IN x'RANGE LOOP ...
FOR i IN RANGE ( x'LOW TO x'HIGH ) LOOP ...
FOR i IN RANGE ( 0 TO x'LENGTH-1 ) LOOP ...
```



## Ejercicio 2

Realice el código en VHDL del siguiente diagrama:



sel	c
00	0
01	a
10	b
11	Z

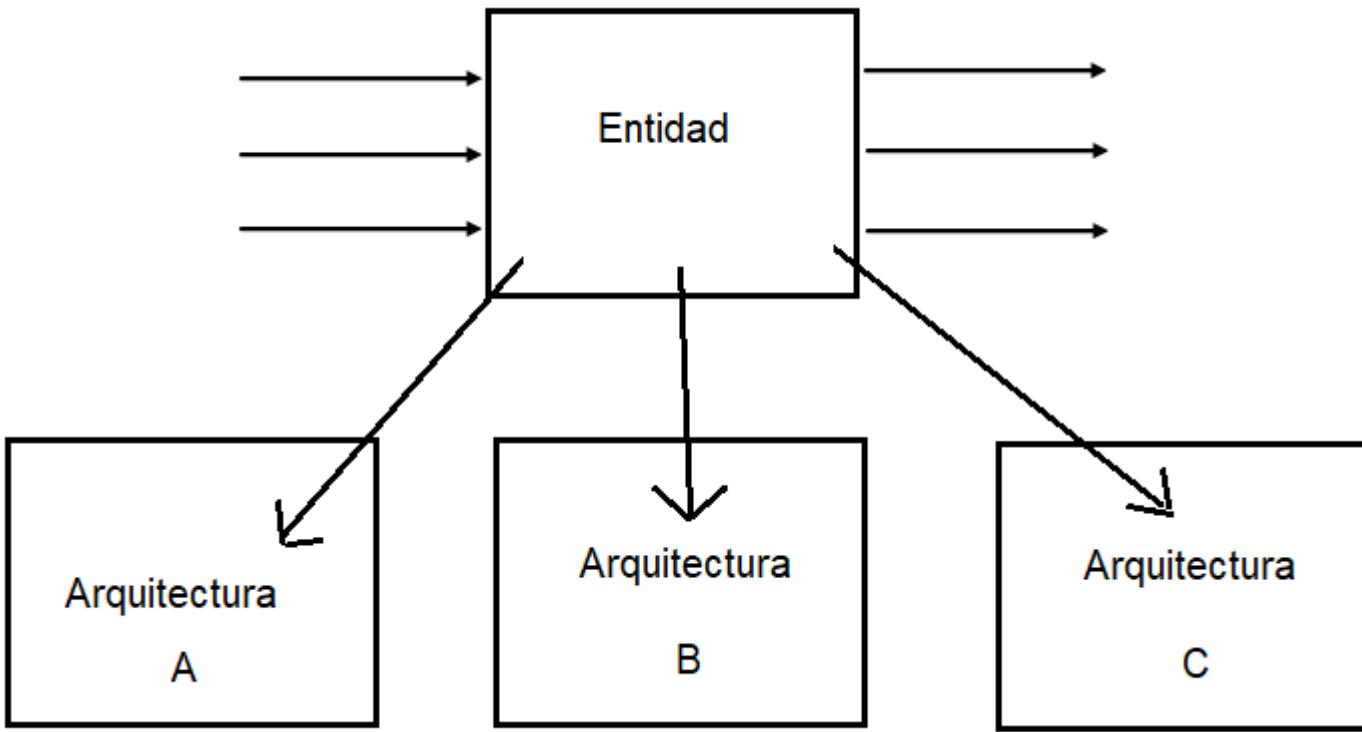
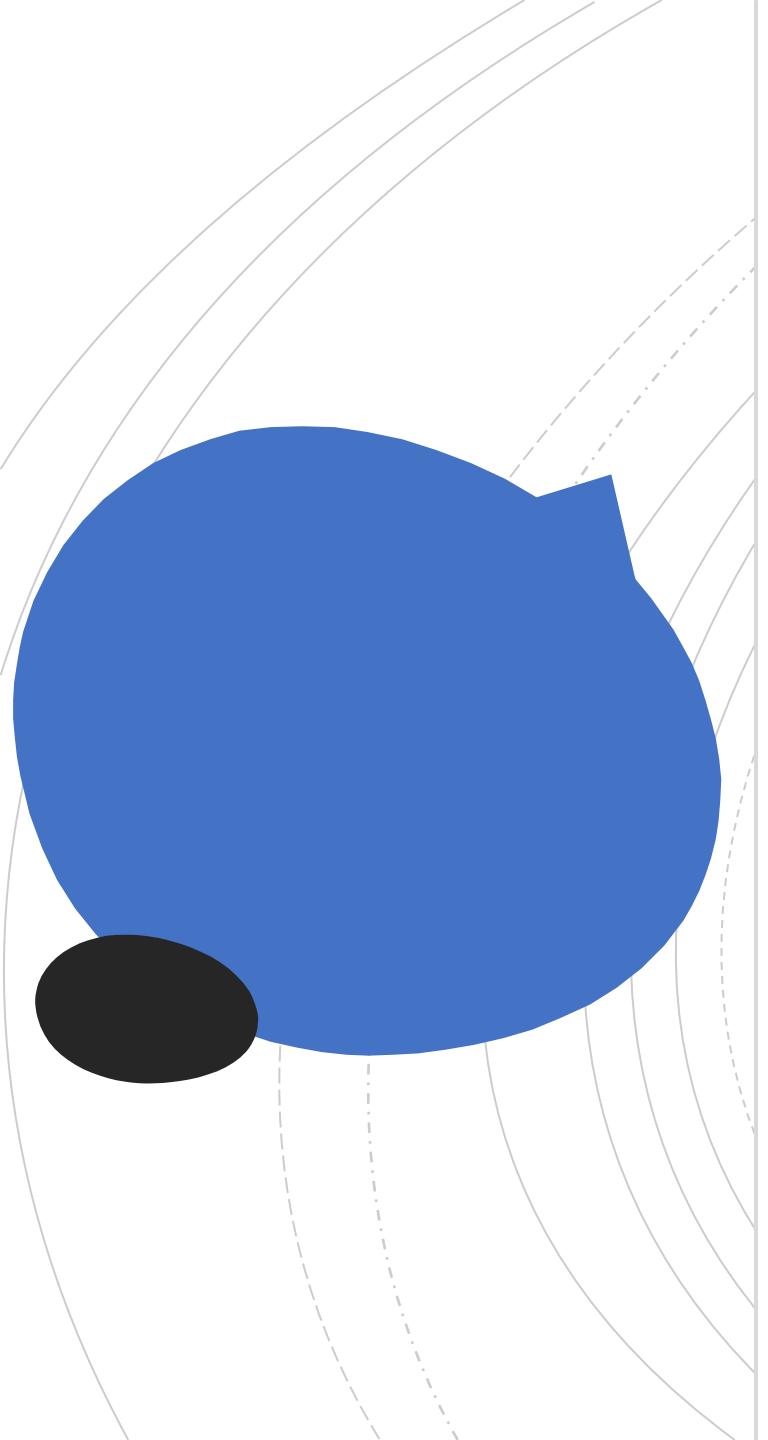
Pista. Puedes utilizar sentencias IF para la solución del problema o bien, Case - When .

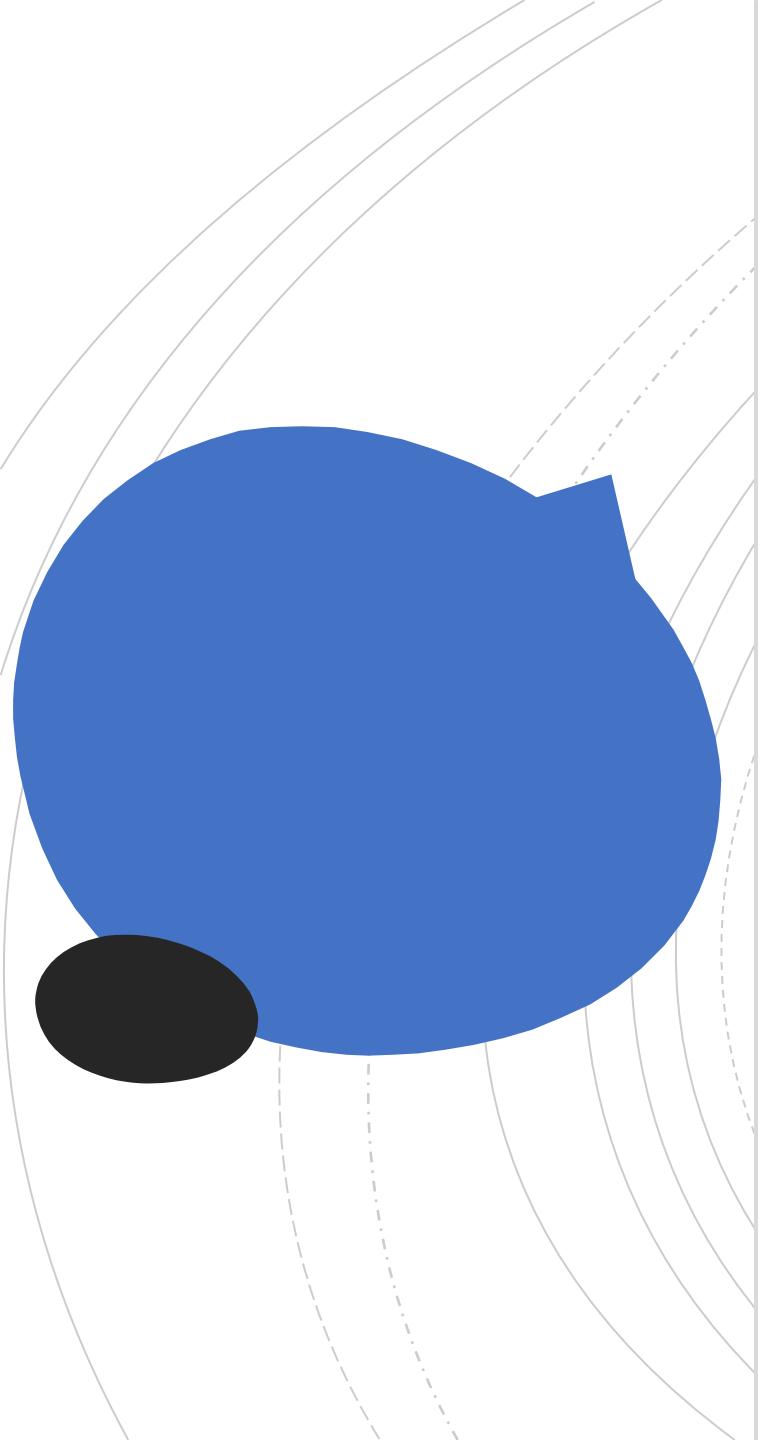
# Plantilla con IF

```
1 -----
2 LIBRARY ieee;
3 USE _____ ;
4 -----
5 ENTITY mux IS
6     PORT ( __ , __ : __ STD_LOGIC_VECTOR (7 DOWNTO 0);
7             sel : IN _____ ;
8             __ : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9 END _____ ;
10 -----
11 ARCHITECTURE example OF _____ IS
12 BEGIN
13     PROCESS (a, b, __ )
14     BEGIN
15         IF (sel = "00") THEN
16             c <= "00000000";
17         ELSIF (_____ ) THEN
18             c <= a;
19         _____ (sel = "10") THEN
20             c <= __ ;
21         ELSE
22             c <= (OTHERS => '__ ');
23         END _____ ;
24     END _____ ;
25 END _____ ;
26 -----
```

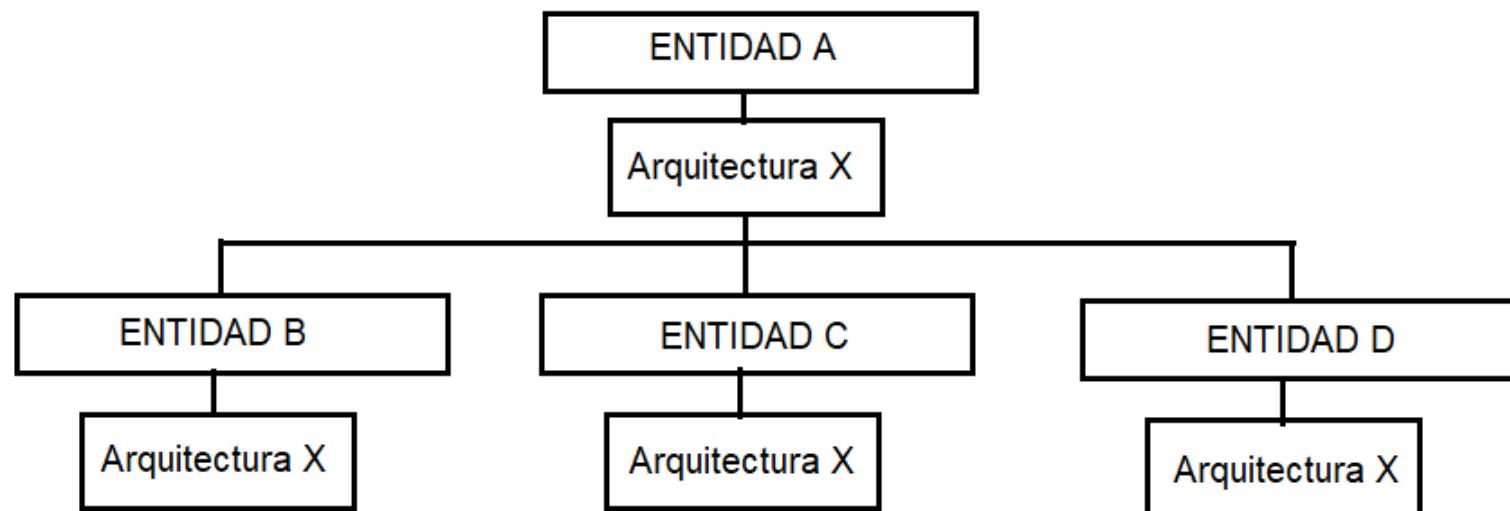


Arquitectura





Ejemplo de Arquitecturas en dispositivos CPLD o FPGA con componentes.



# Tipos de descripción en HDL

- La arquitectura es la encargada de como describir el funcionamiento, descripción o comportamiento de una entidad.

## **FLUJO DE DATOS (comportamental)**

uso de ecuaciones booleanas  
uso de when - else

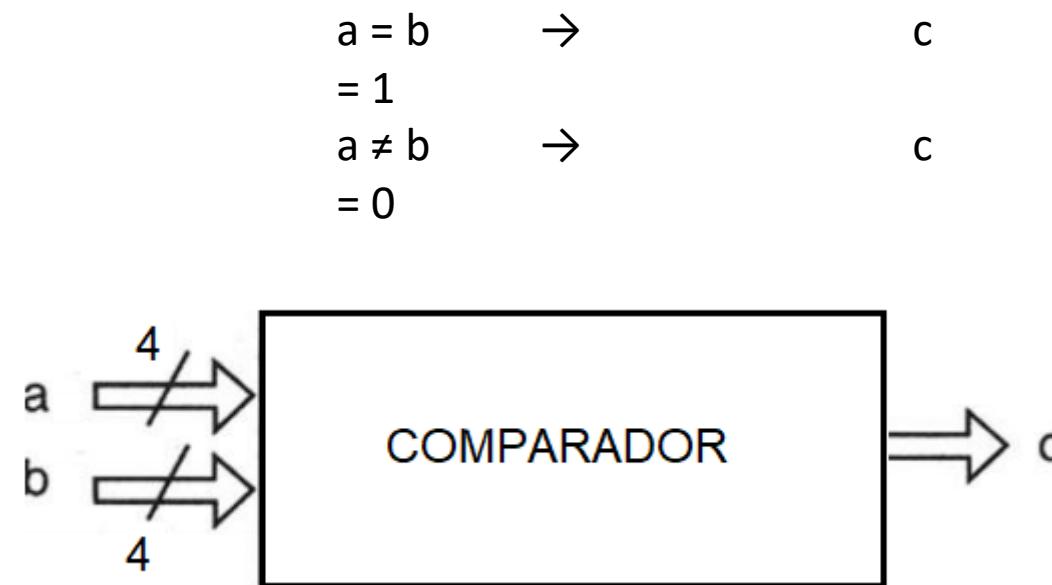
## **FUNCIONAL**

uso de Process (if –then - else)

## **JERARQUICO (Estructural)**

uso de paquetes (definidos por el usuario)  
uso de bibliotecas (definidas por el usuario)  
componentes

## Diagrama de bloques de un circuito comparador





# Solucion.

Descripción de la arquitectura Comportamental (flujo de datos)

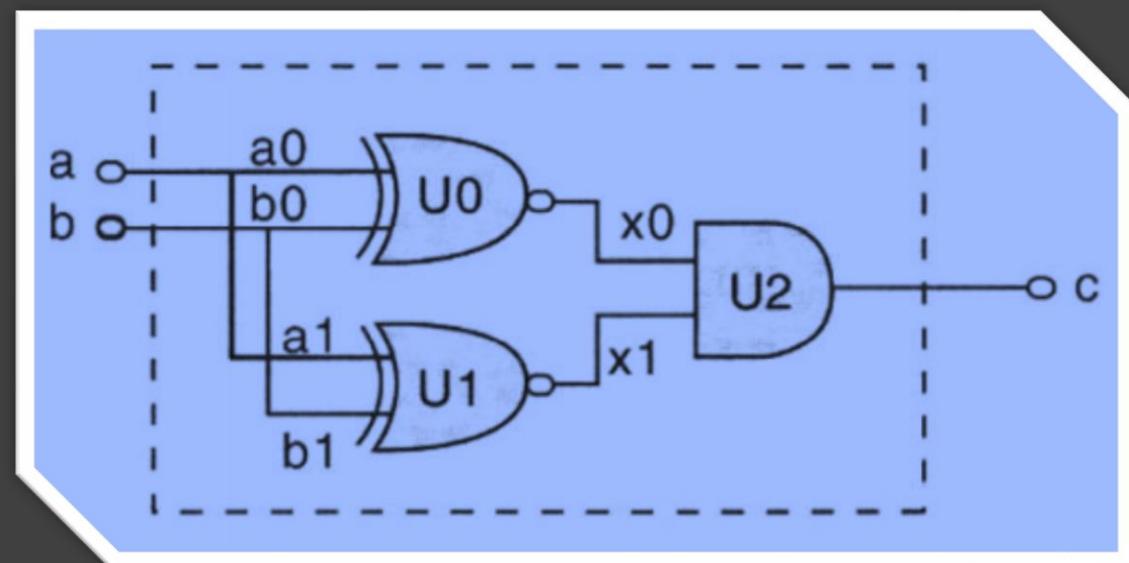
```
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY comparadorflujo IS
PORT (a,b: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
      s: OUT STD_LOGIC
);
END comparadorflujo;
ARCHITECTURE codigo OF comparadorflujo IS
BEGIN
  s <= (a(0) xor b(0)) or (a(1) xor b(1)) or (a(2) xor b(2)) or (a(3) xor b(3));
END codigo;
```

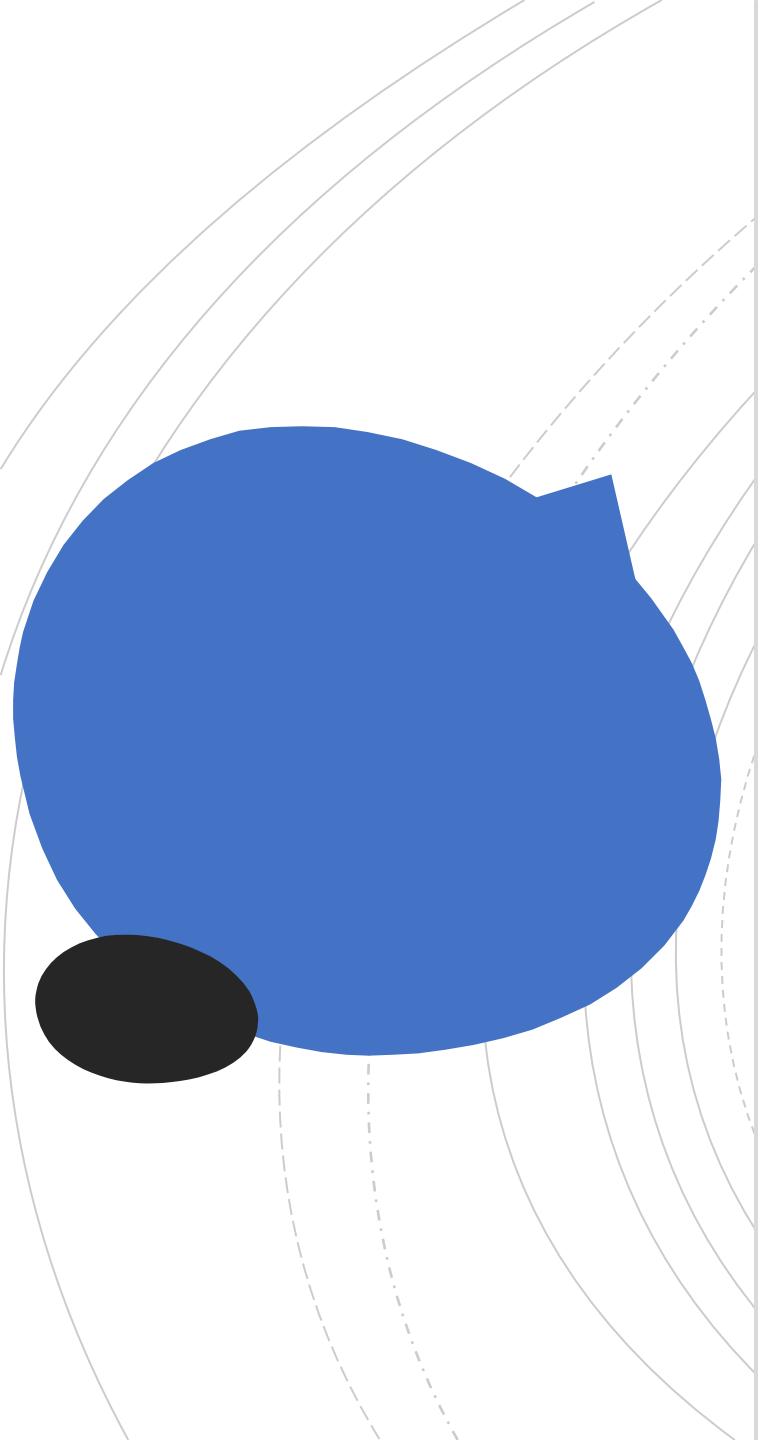
## Descripción de la arquitectura funcional

```
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY comparadorfuncional IS
    PORT (a,b: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
          s: OUT STD_LOGIC
        );
END comparadorfuncional;
ARCHITECTURE codigo OF comparadorfuncional IS
BEGIN
    PROCESS (a,b)
    BEGIN
        IF (a=b) THEN
            s <= '0';
        ELSE
            s <= '1';
        END IF;
    END PROCESS;
END codigo;
```

## Descripción de la arquitectura estructural

- una descripción estructural basa su comportamiento en modelos lógicos establecidos (compuertas, sumadores, contadores, etc.). El diseñador crea estas estructuras y las guarda para su uso posterior o tomarlas de los paquetes contenidos en las librerías de diseño del software que se esté utilizando.





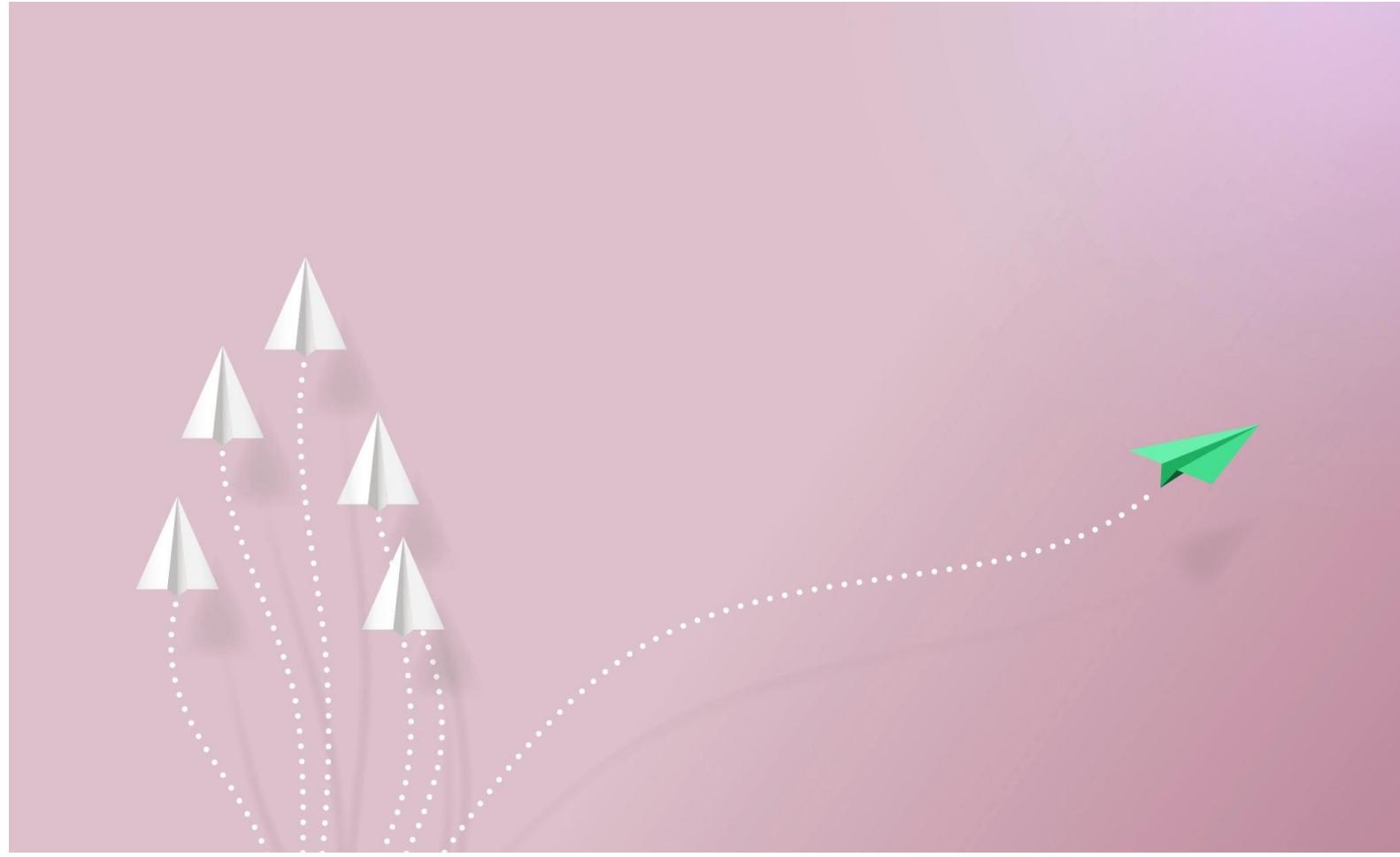
```
library ieee;
use ieee.std_logic_1164.all;

entity comp is port (
a,b: in bit_vector (0 to 1);
c: out bit);
end comp;
use work.compuerta.all ;

architecture estructural of comp is
signal x: bit_vector (0 to 1);
begin
U0: xnor2 port map (a(0), b ( 0 ) , x ( 0 )
U1: xnor2 port map (a(1), b ( 1 ) , x (1)
U2 : and2 port map (x ( 0 ) , x(1) , c) ;
end estructural;
```

Los componentes **xnor** y **and** no se declaran debido a que se encuentran en el paquete de compuertas (*creado por el diseñador*), el cual a su vez está dentro de la librería de trabajo (*work*)

un paquete consta de un conjunto de subprogramas, constantes, declaraciones, etc., con la intención de implementar algún servicio.



TABLAS DE  
VERDAD  
MEDIANTE EL  
USO DE VHDL

-----USO DE TABLA DE VERDAD-----

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

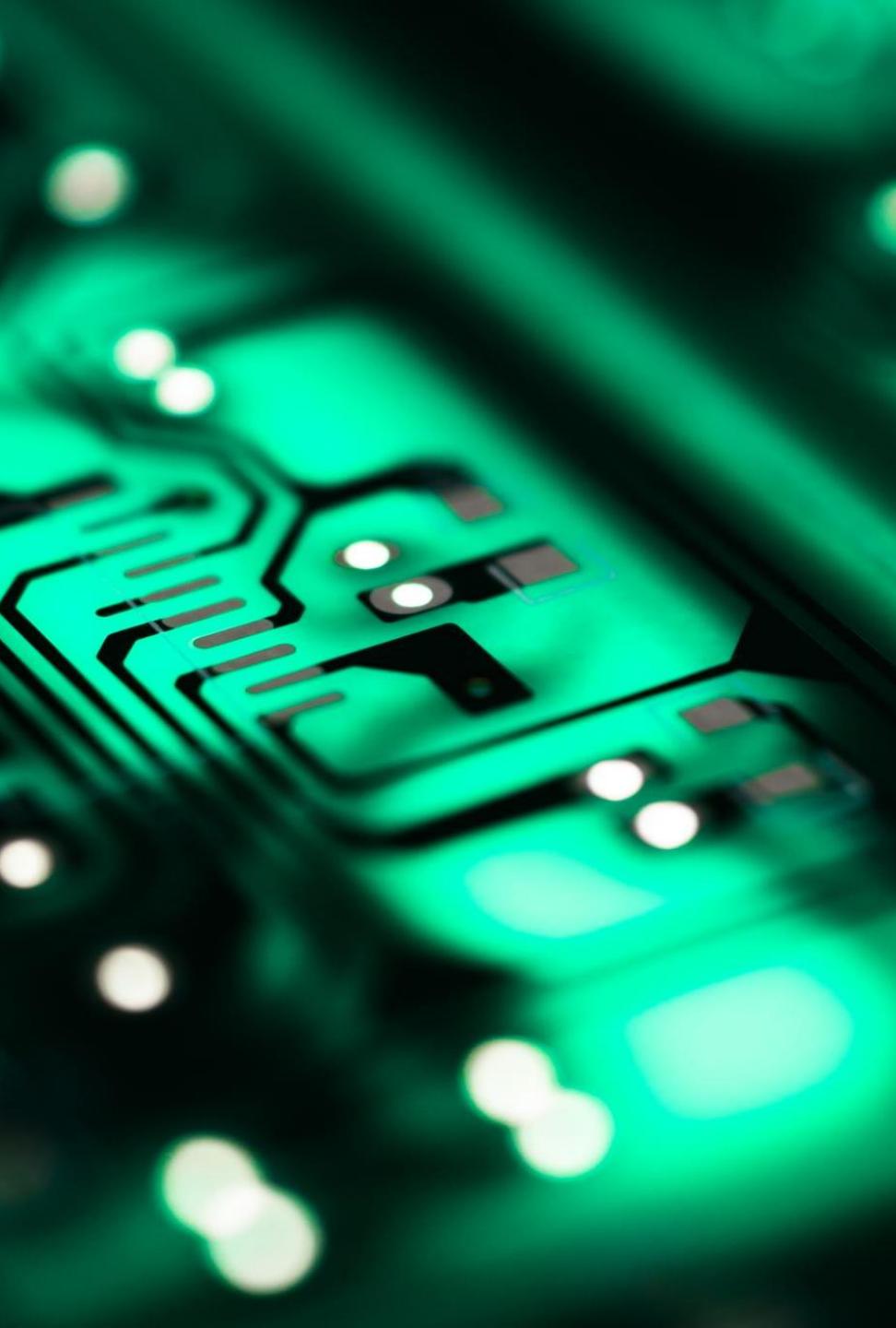
ENTITY funcion1 IS
PORT ( x, y, z : IN bit;
f1 : OUT bit);
END funcion1;
)

ARCHITECTURE tabla1 OF funcion1 IS
SIGNAL bits: BIT_VECTOR(2 DOWNTO 0);
BEGIN
    bits<= x & y & z; --concatena los bits de entrada en bit vector
    WITH bits SELECT
        y <=
            '0' WHEN "000", --Tabla de verdad
            '1' WHEN "001",
            '0' WHEN "010",
            '0' WHEN "011",
            '0' WHEN "100",
            '1' WHEN "101",
            '0' WHEN "110",
            '0' WHEN "111";
END tabla1;
```

x	y	z	f <sub>1</sub>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

# Registros

**OBJETIVO PARTICULAR.** Implementar los distintos tipos de registros con distintas funciones mediante el empleo del HDL, para la transferencia de datos.



# Circuitos Lógicos

- Los circuitos lógicos se clasifican en dos tipos:
  - C. Lógicos Combinacionales  
Compuertas lógicas
  - C. Lógicos Secuenciales
    - Flip - Flops

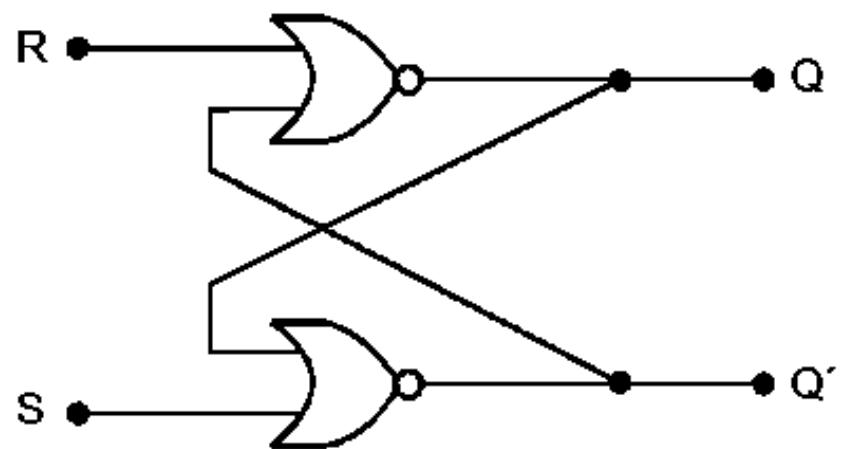
# ¿Qué es un flip flop?

- Los flip-flops son multivibradores biestables que se pueden construir a partir de compuertas, NAND y NOR. Los flip-flops se interconectan para formar circuitos lógicos secuenciales que almacenen datos, generen tiempos, cuenten y sigan secuencias.
- El MV *astable* produce una serie continua de pulsos de onda cuadrada y normalmente se utiliza como *reloj* en un sistema digital.
- El MV monoestable produce un solo pulso cuando es disparado por una fuente externa.

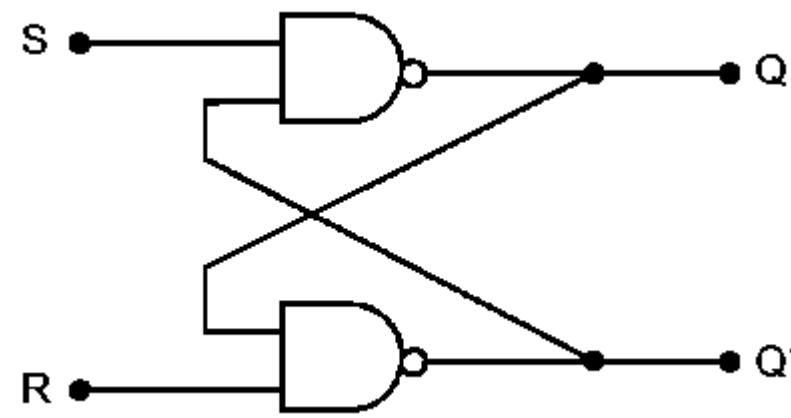
# ¿Qué es un Latch?

- Los latches se denominan, también flip-flops asíncronos\*
  - circuito de 1 bit de información, (0 ó un 1).
  - no necesitan de una señal externa de reloj para operar.
- Un latch esta en estado SET cuando la salida Q esta en nivel alto ( 1 ), y en estado RESET cuando Q esta en nivel bajo ( 0 ).
- Para almacenar un 1 lógico, se debe aplicar un pulso de disparo a la entrada SET. Para almacenar un 0 lógico, se debe de aplicar un pulso de disparo a la entrada RESET. El pulso de disparo puede ser positivo o negativo.



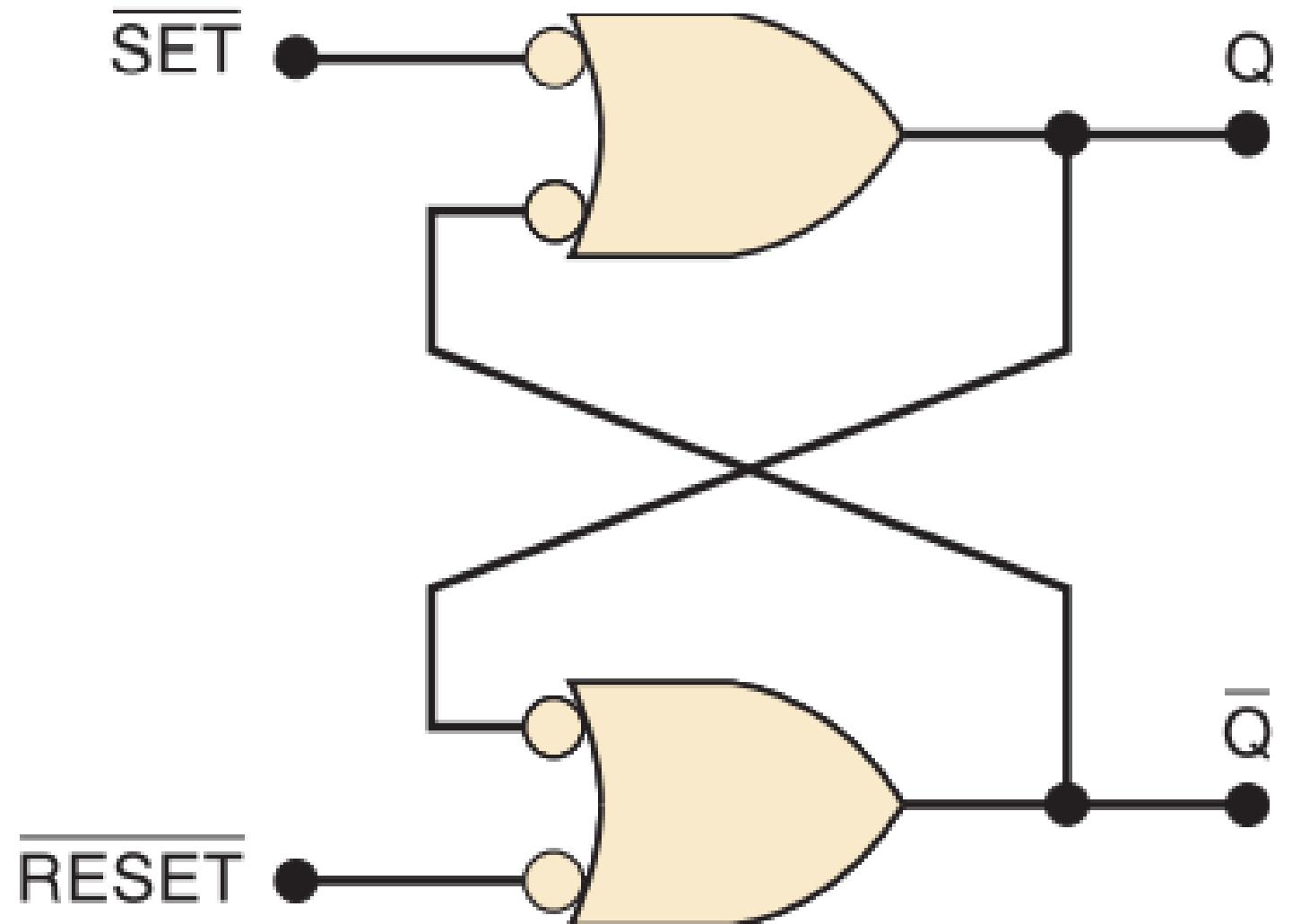


<http://tinyurl.com/y9xjtq67>



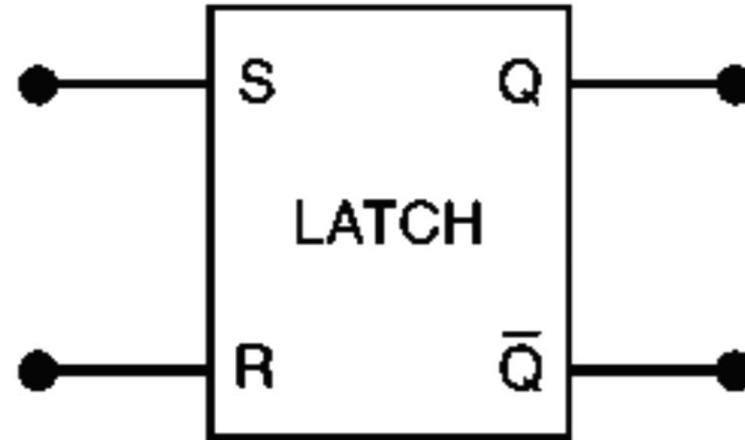
<http://tinyurl.com/yaw2ubud>

## Representación equivalente del latch NAND



# Diagrama de bloques y Tabla de funcionamiento del latch

Diagrama de bloques de un latch



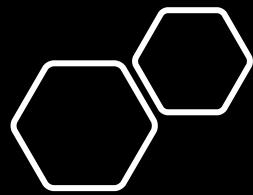
Set	Reset	Salida
1	1	Sin cambio
0	1	$Q = 1$
1	0	$Q = 0$
0	0	Inválido*

Tabla de funcionamiento de un Latch con compuertas NAND

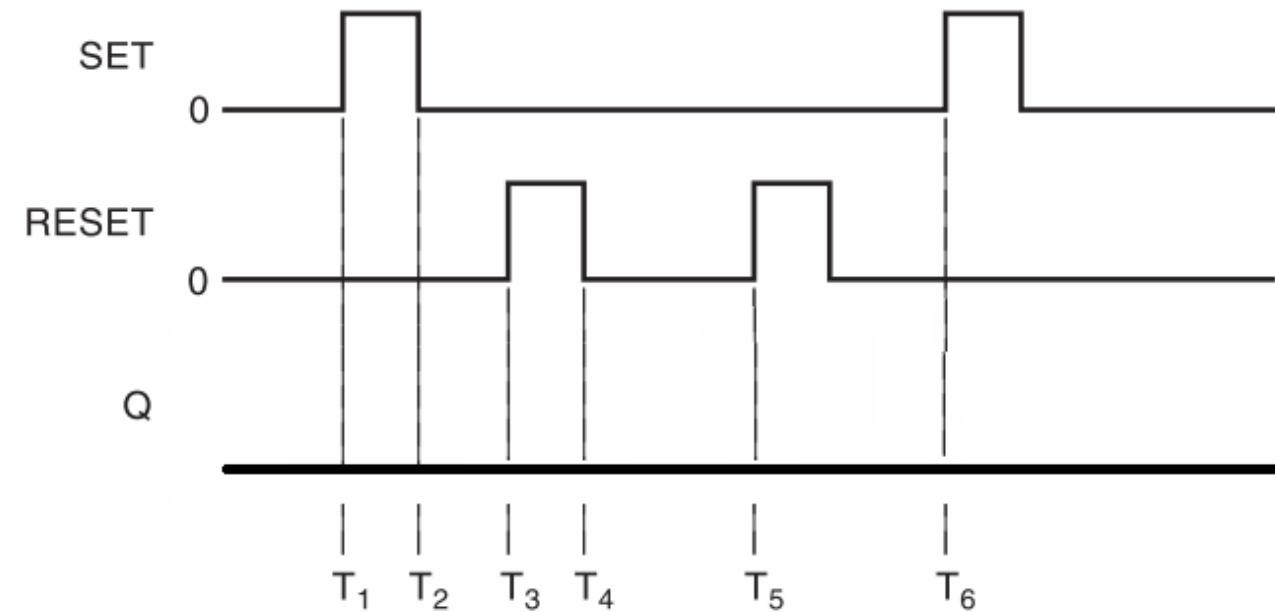
Set	Reset	Salida
0	0	Sin cambio
1	0	$Q = 1$
0	1	$Q = 0$
1	1	Inválido*

Tabla de funcionamiento de un Latch con compuertas NOR

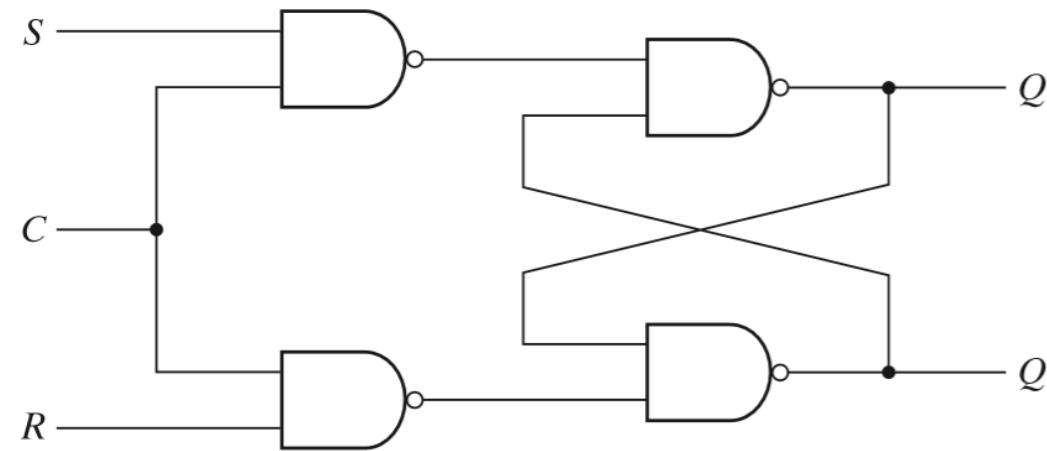
\* Produce  $Q = \bar{Q} = 1$ .



Ejercicio. Determine el diagrama de tiempo con base en el latch de compuertas NOR.



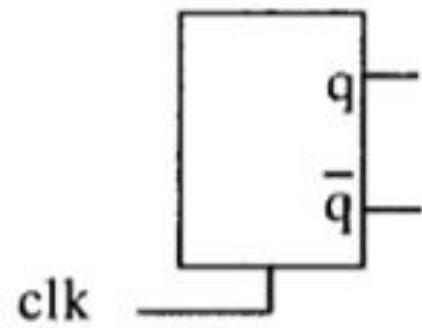
# Latch SR con entrada de control



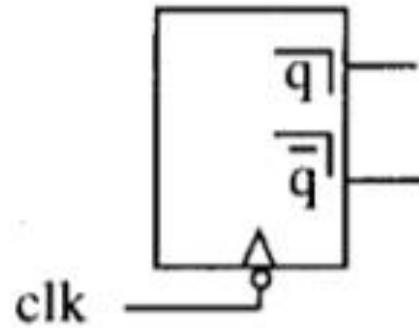
<http://tinyurl.com/y6rcommk>

# Latch vs Flip-flops

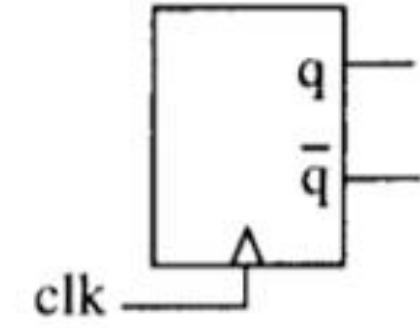
- Disparados por nivel (Latch sincrono):
  - En estos uno de los dos niveles de la señal de reloj habilita los cambios de estado ,según la tabla de estado del biestable\*, mientras que durante el otro nivel no hay cambio de estados ( $Q = q$ ).
- Disparados por flancos (flip-flops): en estos biestables los cambios de estado se producen siempre tras uno de los flancos de la señal de reloj.  
Existen dos estructuras:
  - Master-Slave en la que el biestable puede captar valores de entrada durante el nivel previo al flanco activo.
  - Edge-triggered, en el que las entradas sólo afectan en el entorno del flanco activo.



(*Latch*, nivel H)



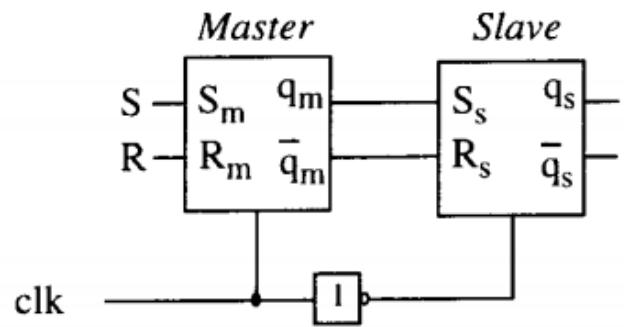
(*Master-Slave*, flanco bajada) (Edge-triggered, flanco subida)



clk

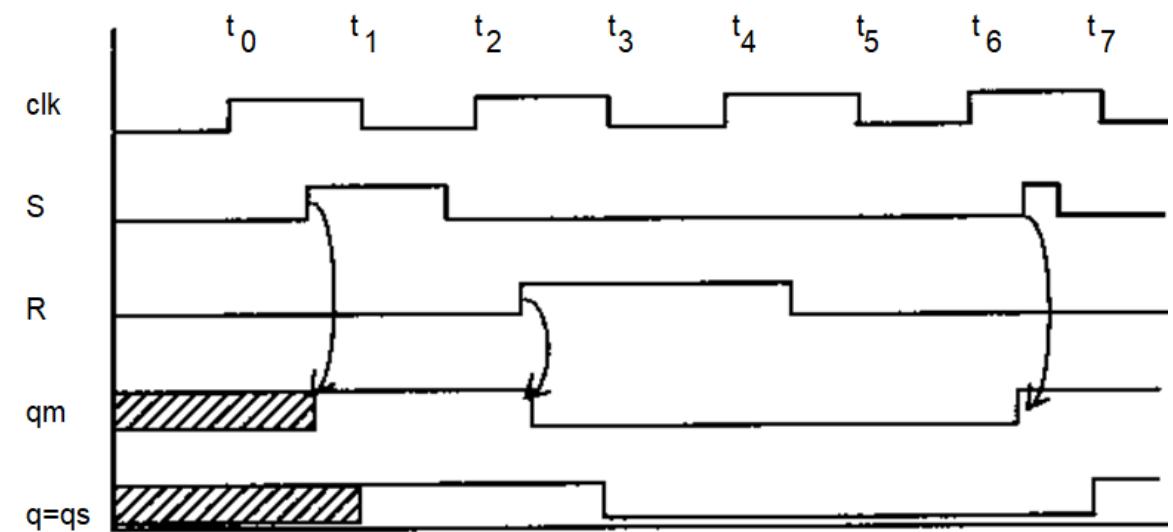
Símbolos representativos.

# Flip – Flop Master Slave

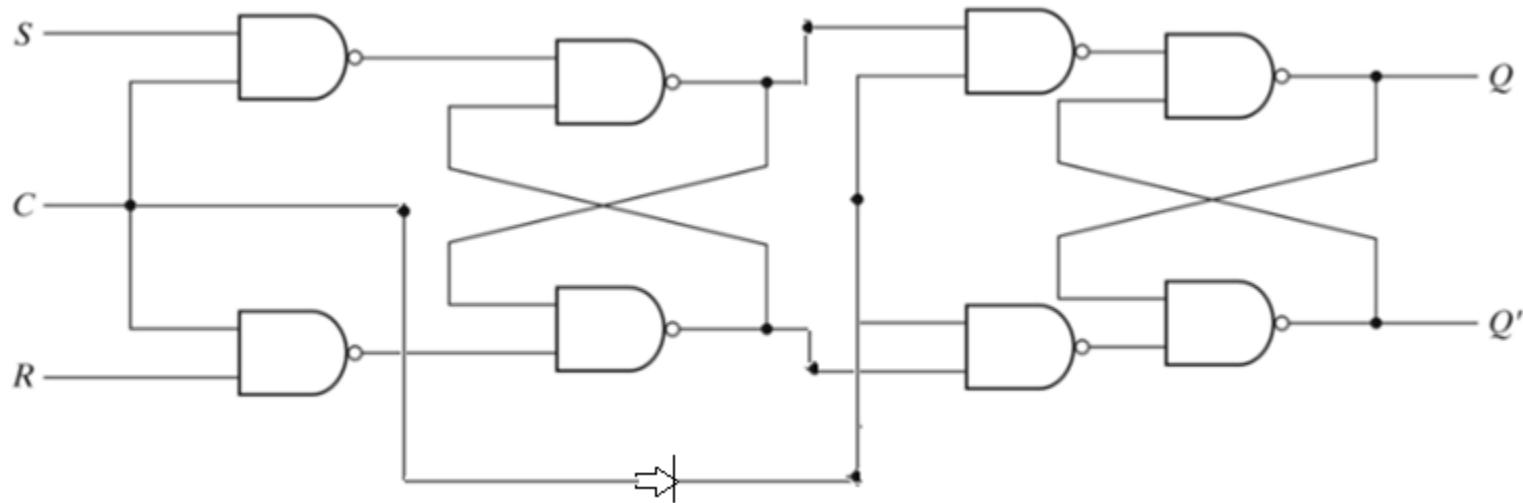


- El biestable **Master** es disparado por el nivel alto de la señal de reloj y recibe las entradas del conjunto Master-Slave .
- Sólo puede cambiar de estado cuando  $clk = 1$  y lo hará dependiendo de sus entradas de excitación. El biestable **Slave** sólo lo hará si  $clk = 0$  . Sus entradas son las salidas del **Master**, por lo que sólo se pueden dar las combinaciones  $SR=10$  (puesta a 1) y  $SR=01$  (puesta a 0) . Las salidas  $qs$  del esclavo son las salidas del conjunto Master-Slave .

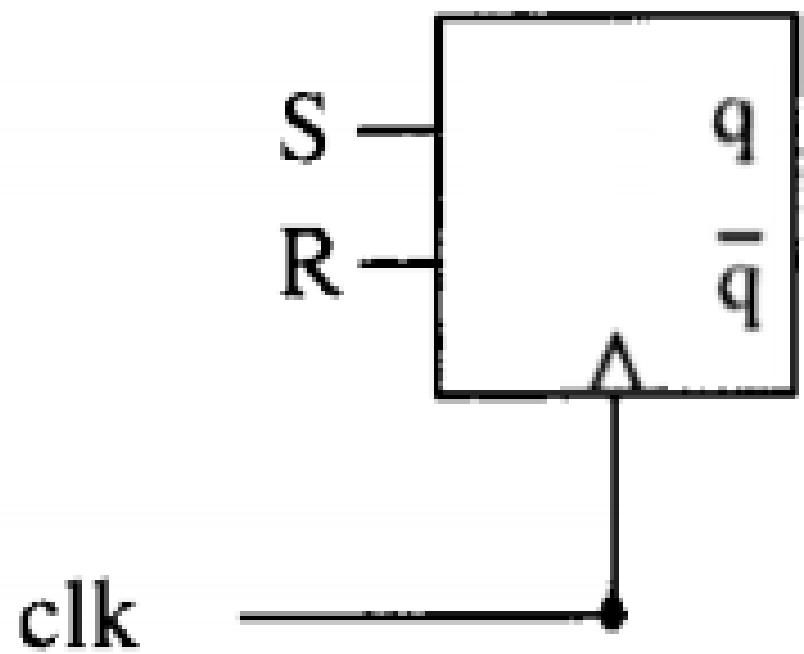
# Diagrama de tiempos de un biestable Master - Slave



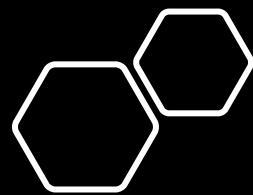
Un flip-flop maestro/esclavo se obtiene conectando dos flip-flops SR en cascada.



<http://tinyurl.com/ydz82945>

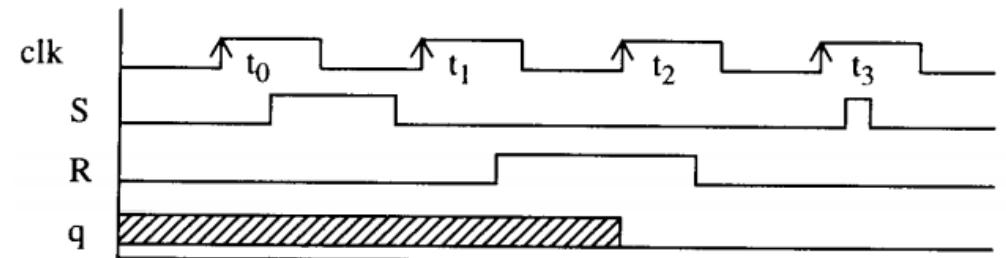


Biestables disparados  
por flanco (Edg-Trggr)

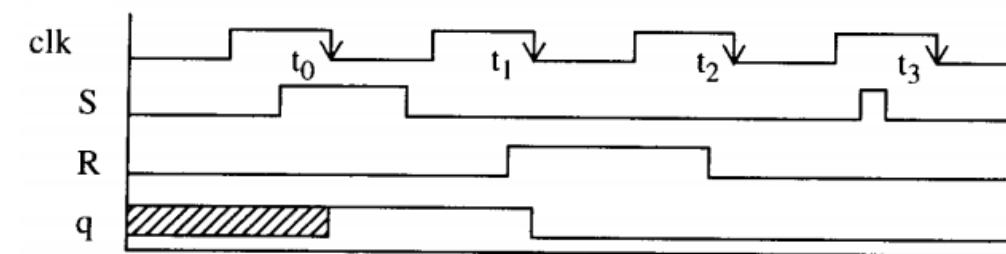


# Diagrama de tiempos de FF

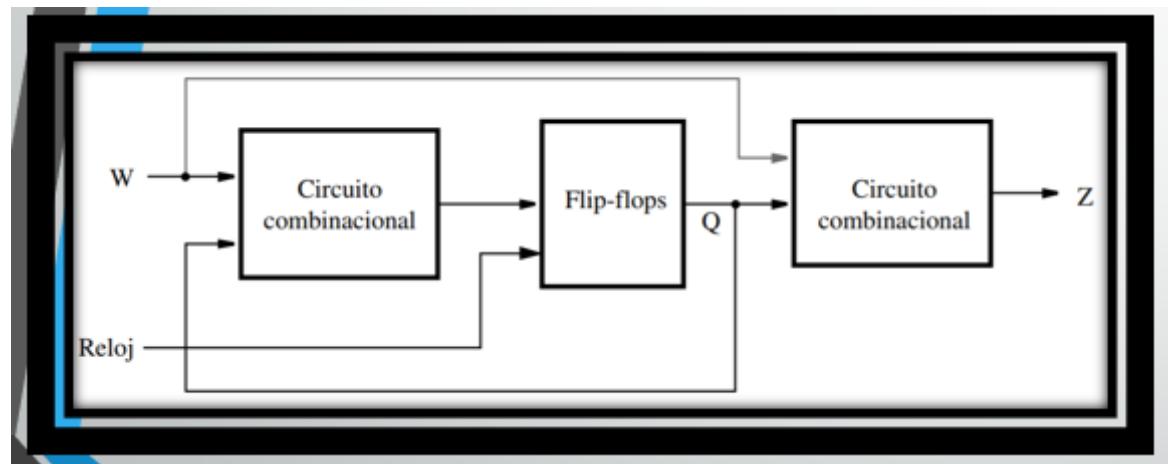
Por Flanco Positivo



Por Flanco Negativo

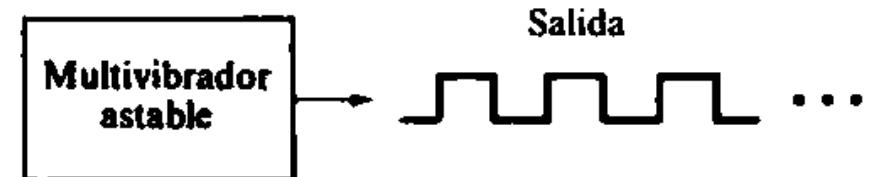


# Circuito secuencial

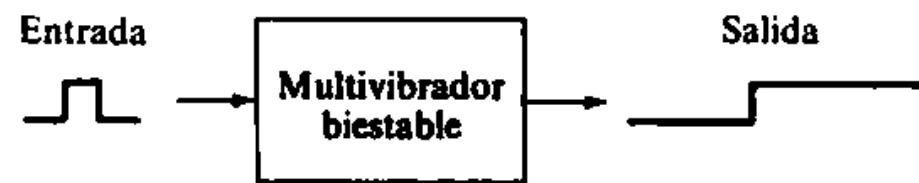


Circuitos de  
disparo para FF.

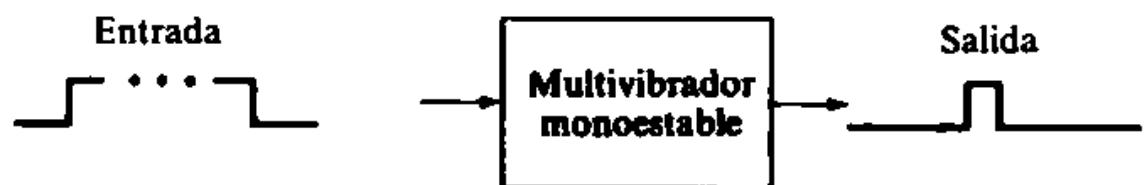
# MULTIVIBRADORES



(a) Salida de un MV astable

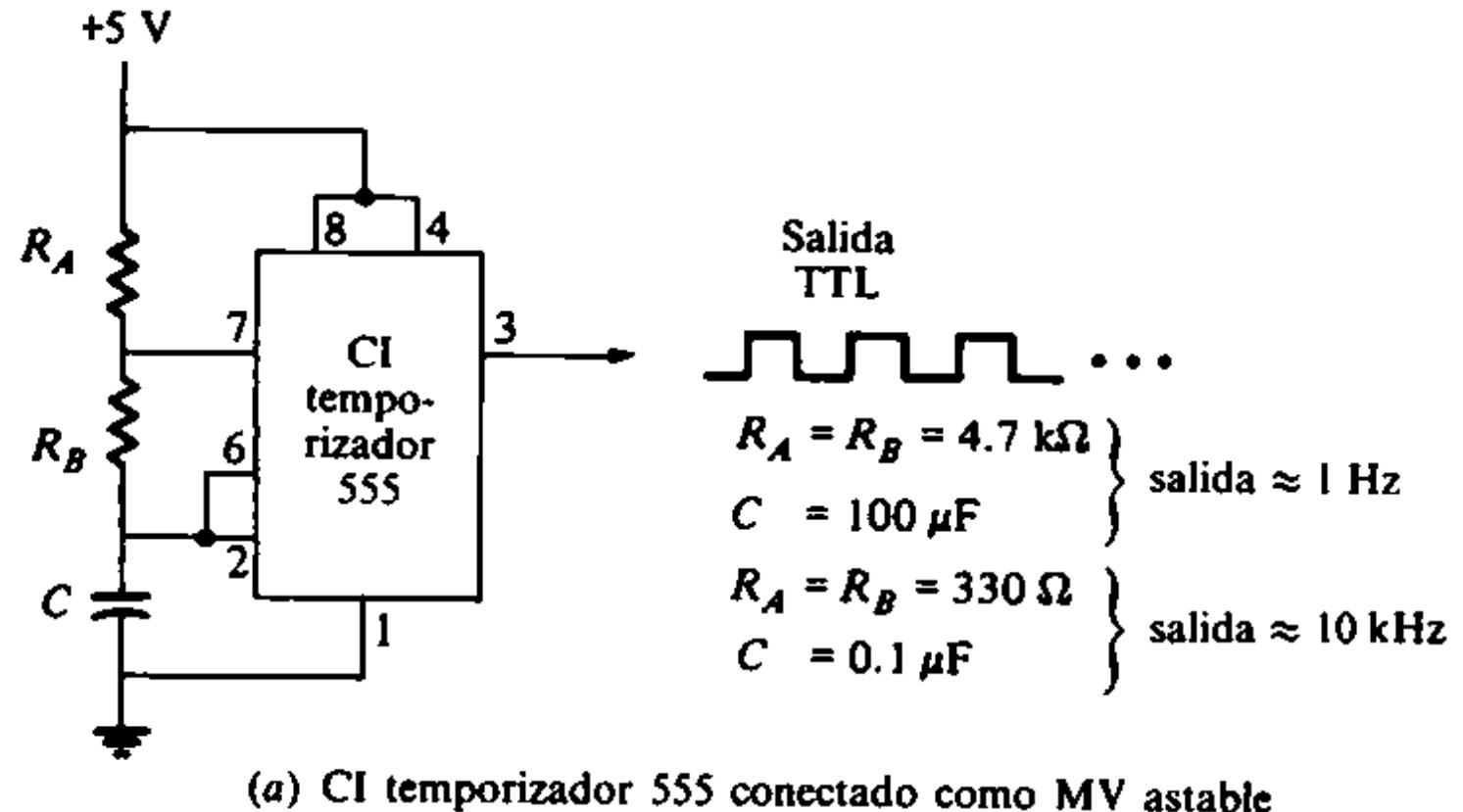


(b) Salida de un MV biestable

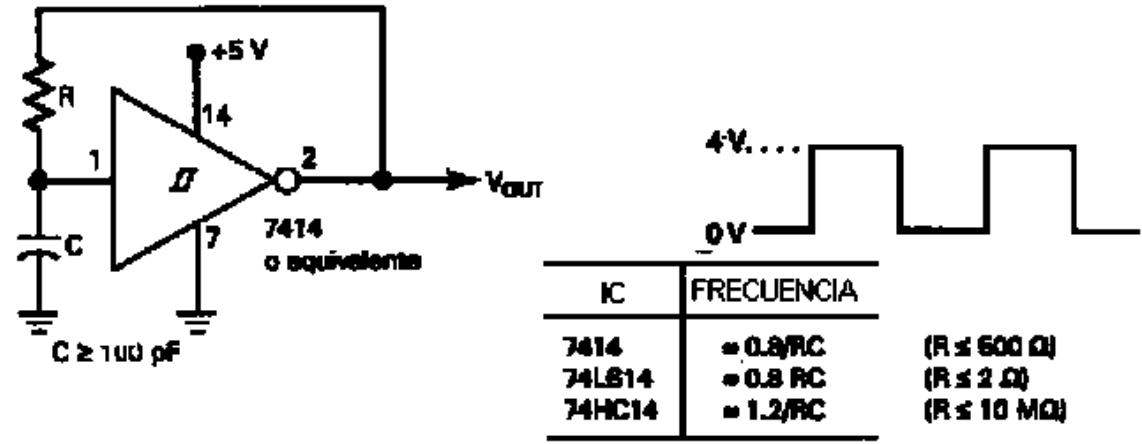


(c) Salida de un MV monoestable

## Circuito generador de pulsos



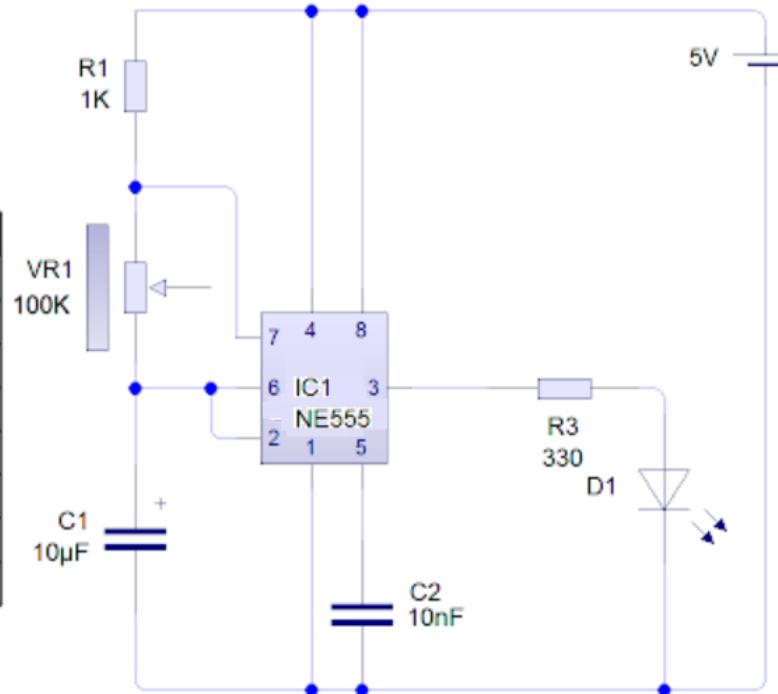
# Oscilador

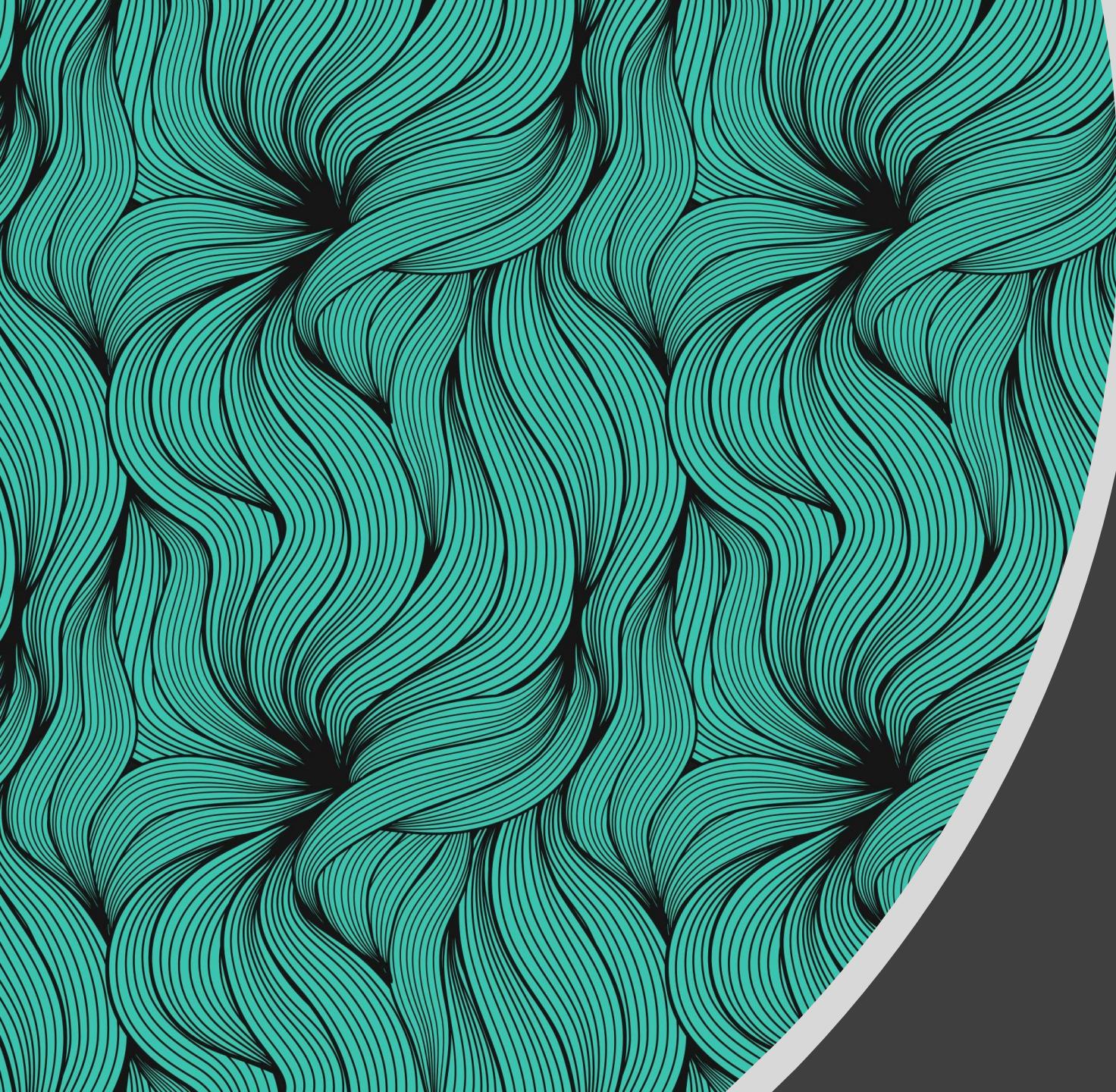


Schmitt Trigger como oscilador simple

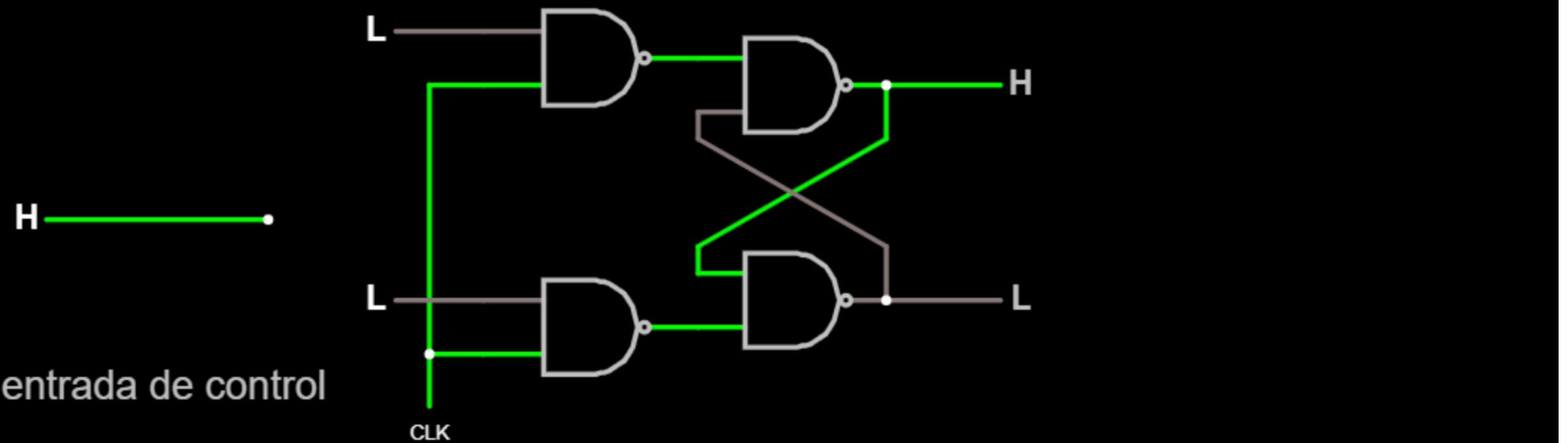
# 555 como circuito astable

Lista de Componentes	
Nombre	Cantidad
10µF Electrolytic Capacitor	1
100K Potentiometer	1
10nF Capacitor	1
1K Resistor (1/4W)	1
330 Resistor (1/4W)	1
5V Cell (Ideal)	1
LED (Red)	1
NE555 Bipolar Timer	1

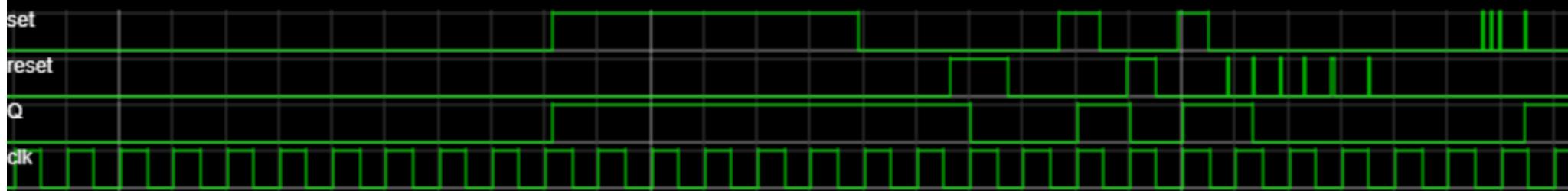


A large, abstract graphic on the left side of the slide features a dense pattern of teal-colored, wavy, line-art shapes resembling stylized flowers or petals. These shapes are set against a solid black background and are separated from the main text area by a thick white diagonal line.

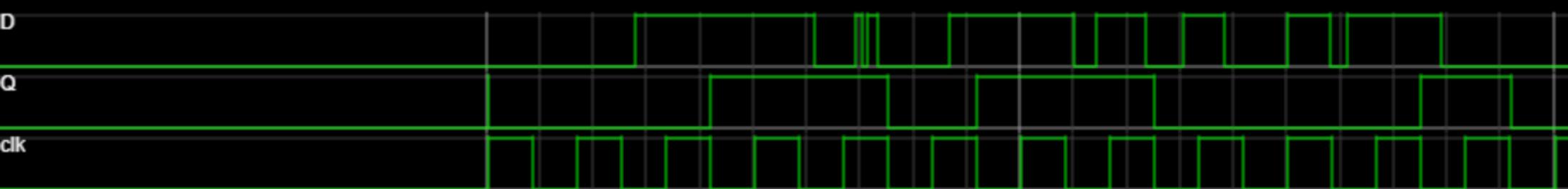
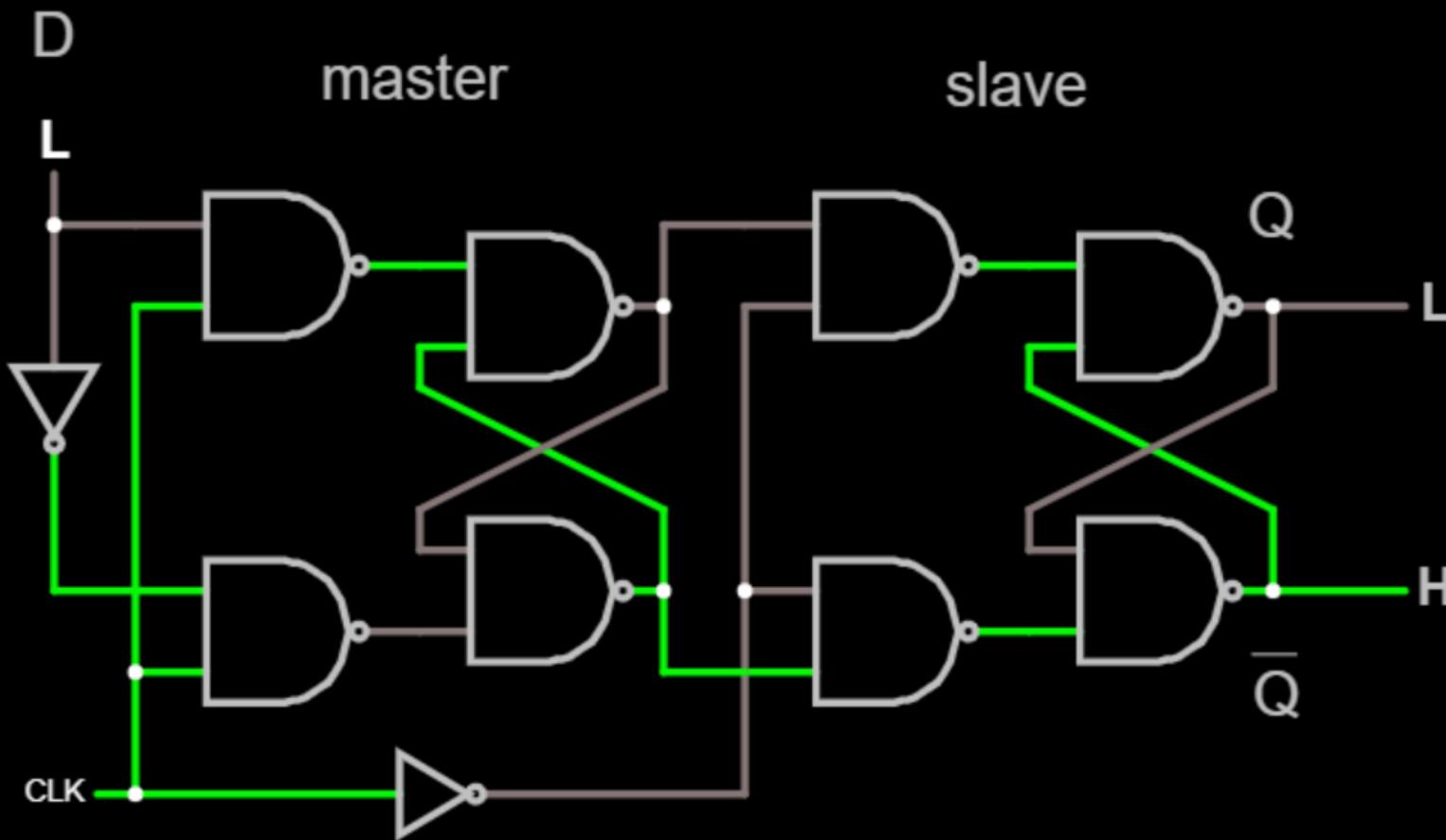
# Anexos



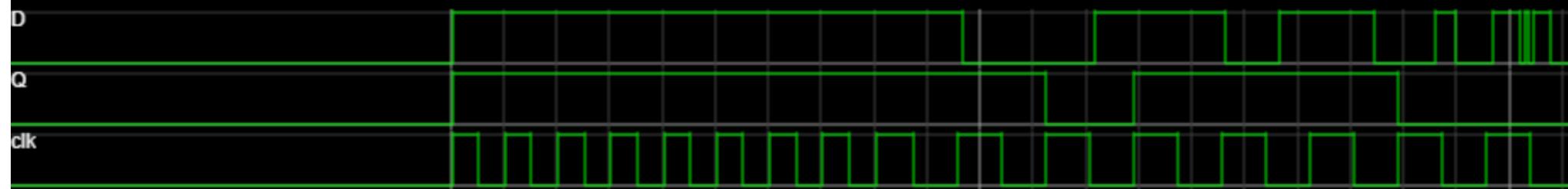
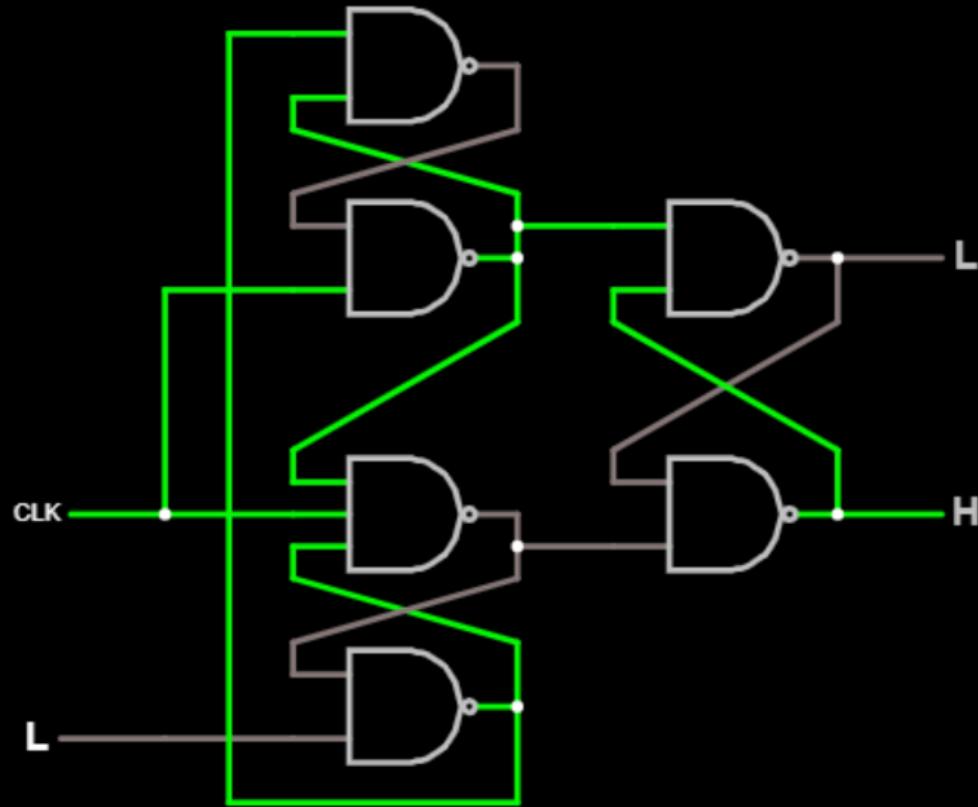
entrada de control



$t = 374.37 \text{ ms}$   
intervalo tiempo = 5  $\mu\text{s}$

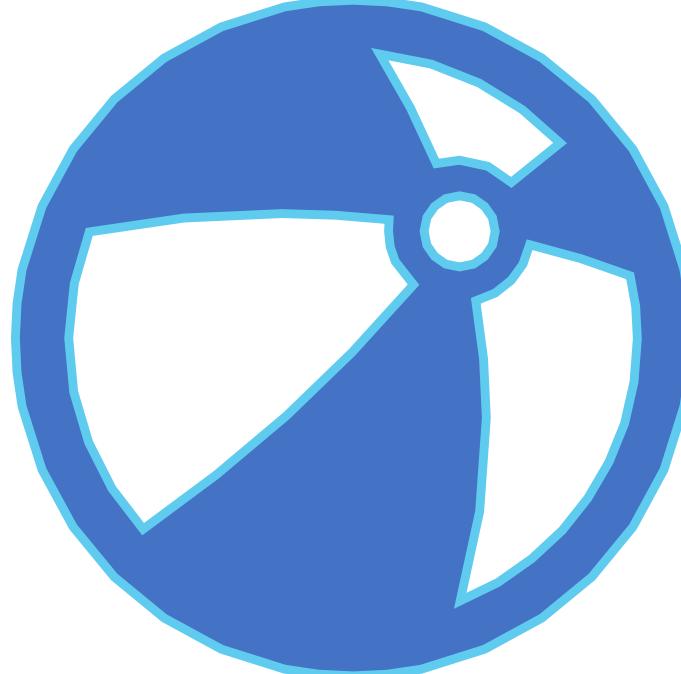


$t = 204.44 \text{ ms}$   
 interval tiempo = 5  $\mu\text{s}$

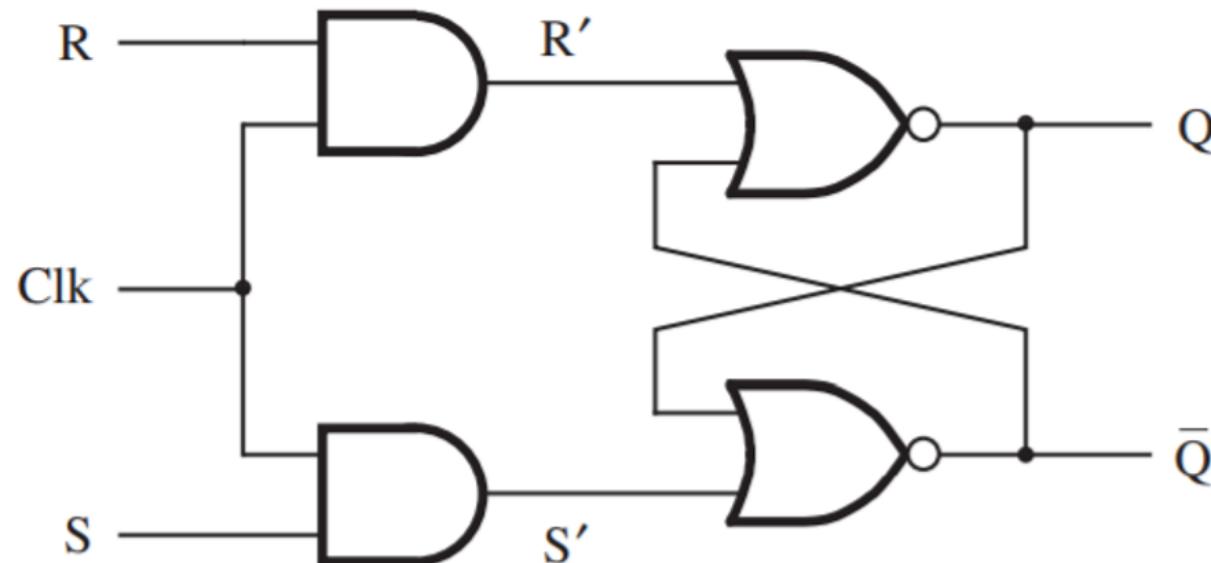




# Tipos de FLIP-FLOPS.



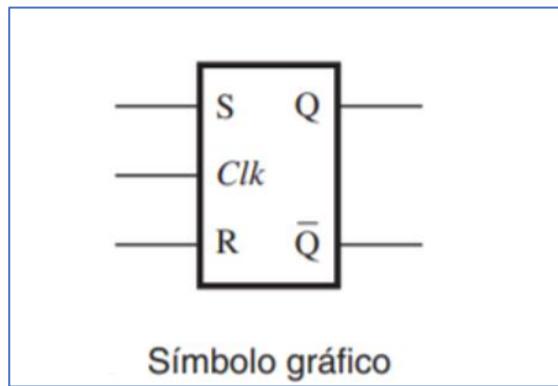
# Flip Flop SR



Circuito

Clk	S	R	$Q(t + 1)$
0	x	x	$Q(t)$ (sin cambio)
1	0	0	$Q(t)$ (sin cambio)
1	0	1	0
1	1	0	1
1	1	1	x

Tabla característica



Símbolo gráfico

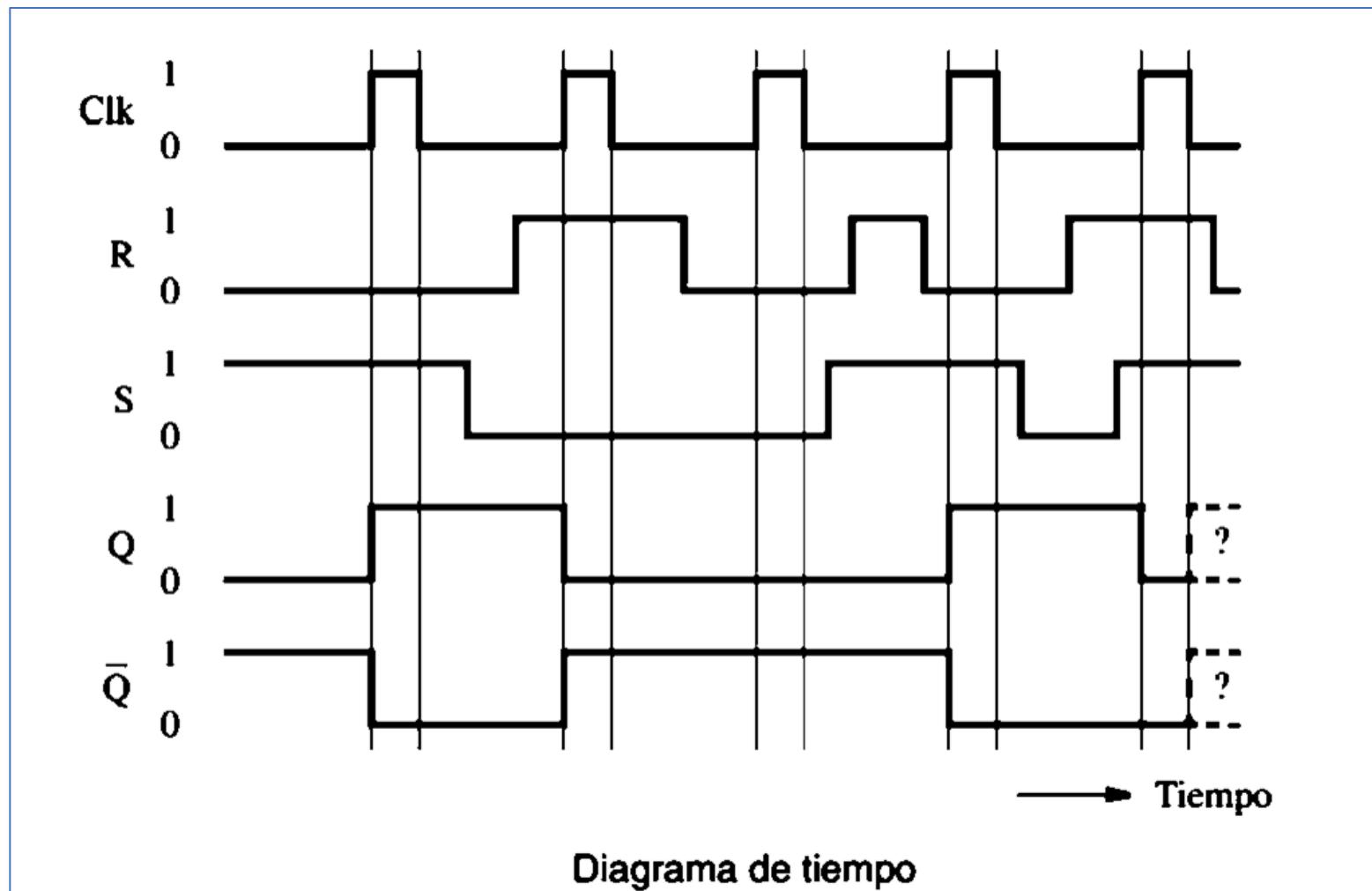


Diagrama de tiempo

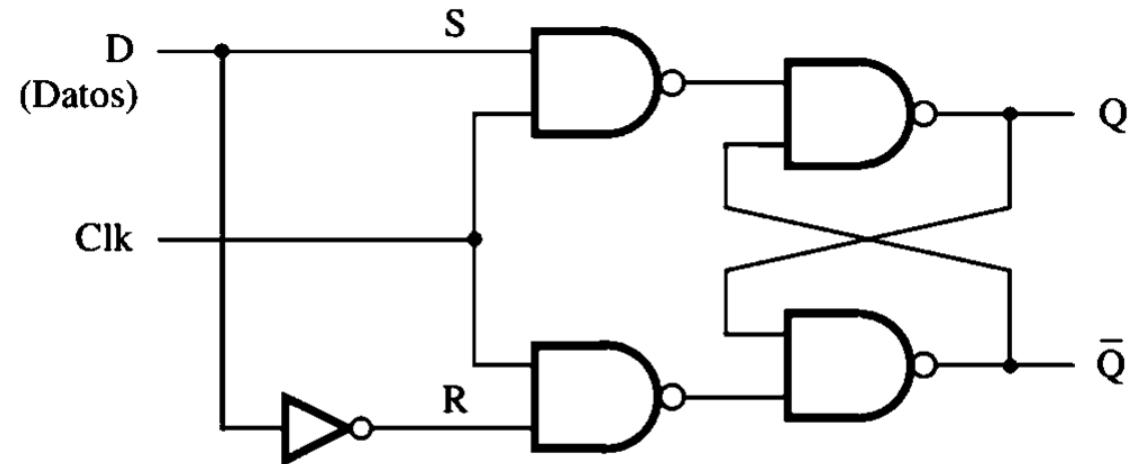
Entradas			Salidas		<i>Comentarios</i>
<i>S</i>	<i>R</i>	<i>CLK</i>	<i>Q</i>	$\bar{Q}$	
0	0	X	$Q_0$	$\bar{Q}_0$	No cambio
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	?	?	No válida

↑ = transición del reloj de nivel BAJO a nivel ALTO

X = irrelevante ("condición indiferente")

$Q_0$  = nivel de salida previo a la transición del reloj

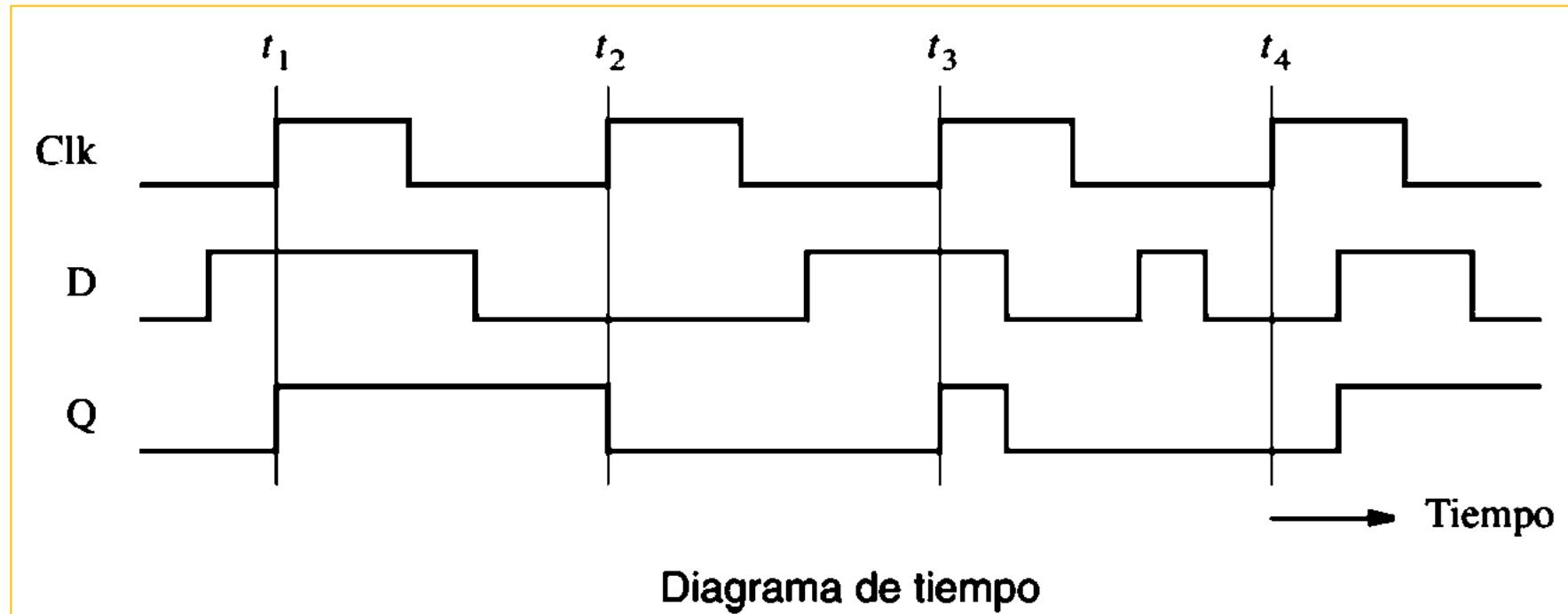
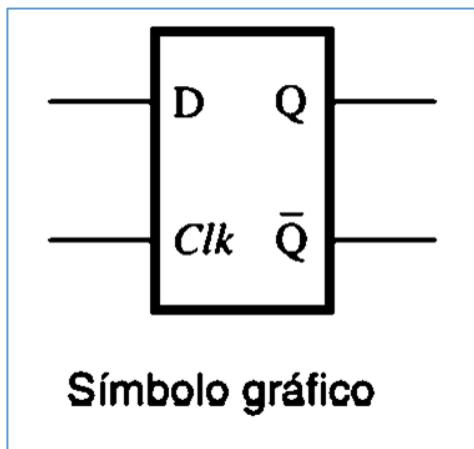
# Flip flop D



Circuito

Clk	D	$Q(t+1)$
0	x	$Q(t)$
1	0	0
1	1	1

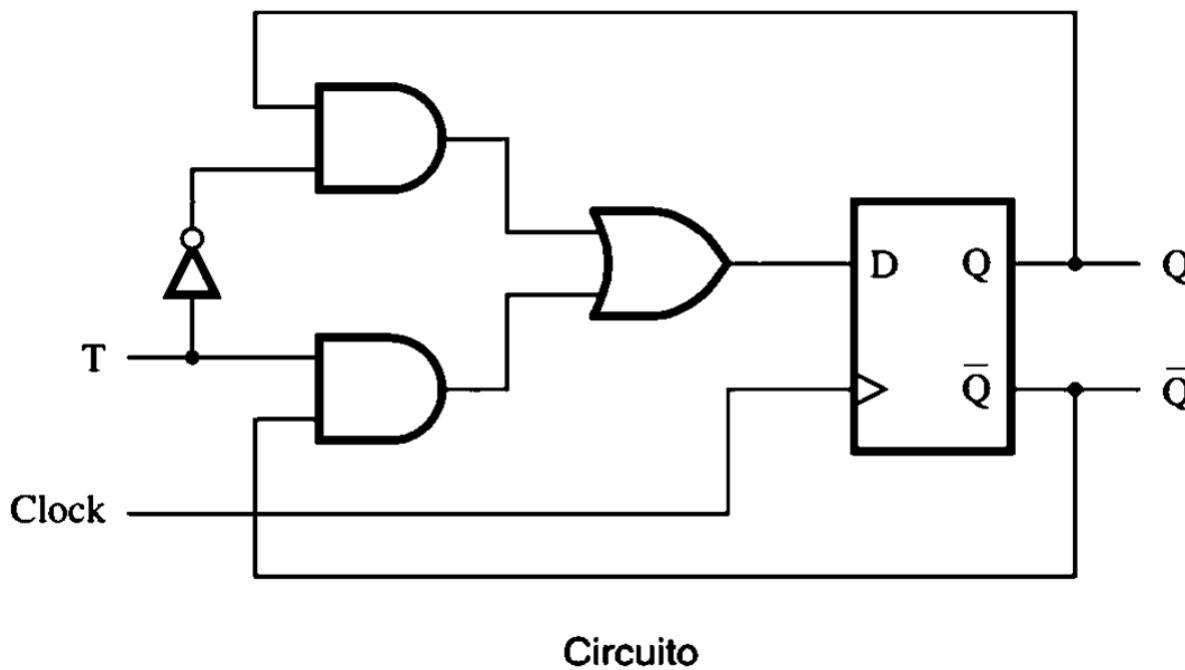
Tabla característica



Entradas		Salidas		<i>Comentarios</i>
<i>D</i>	<i>CLK</i>	<i>Q</i>	$\bar{Q}$	
1	↑	1	0	SET (almacena un 1)
0	↑	0	1	RESET (almacena un 0)

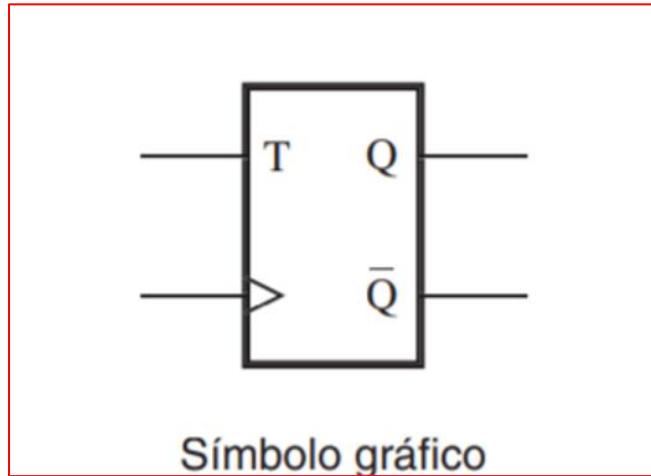
↑ = transición del reloj de nivel BAJO a nivel ALTO

# Flip Flop T



T	$Q(t + 1)$
0	$Q(t)$
1	$\bar{Q}(t)$

Tabla característica



Símbolo gráfico

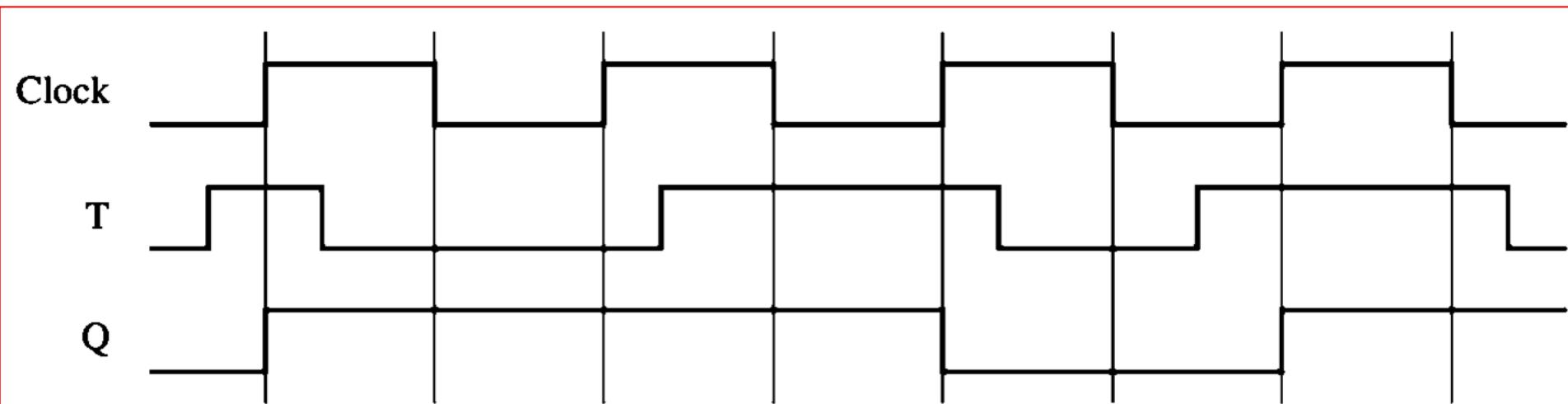
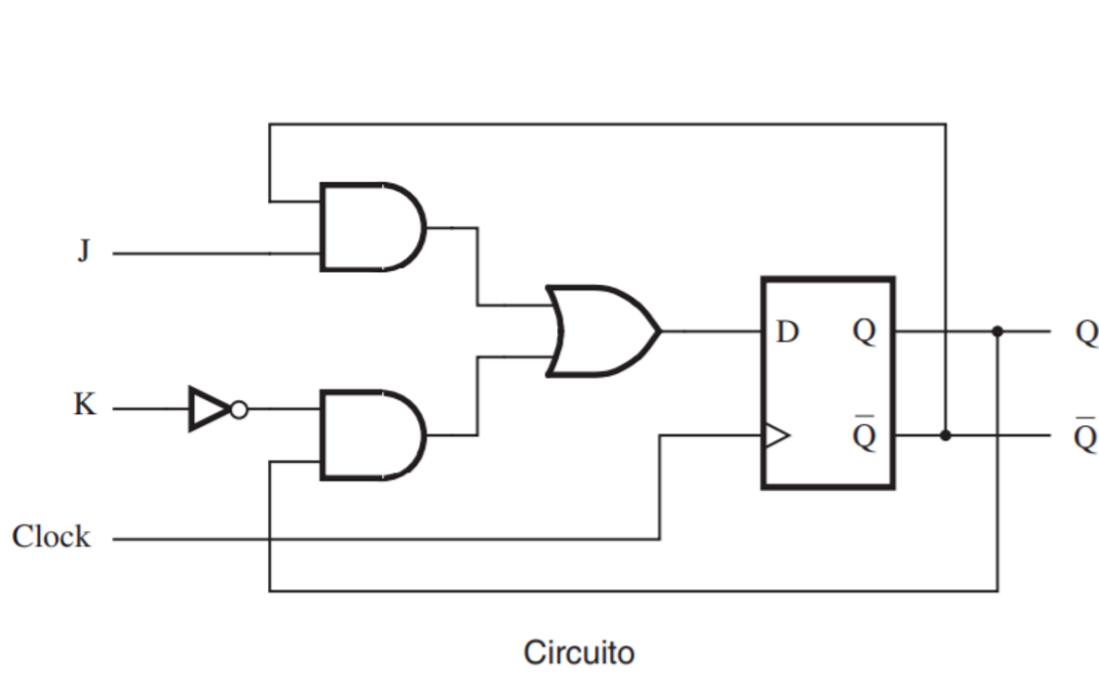


Diagrama de tiempo

# Flip Flop JK



J	K	$Q(t + 1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$\bar{Q}(t)$

Tabla característica

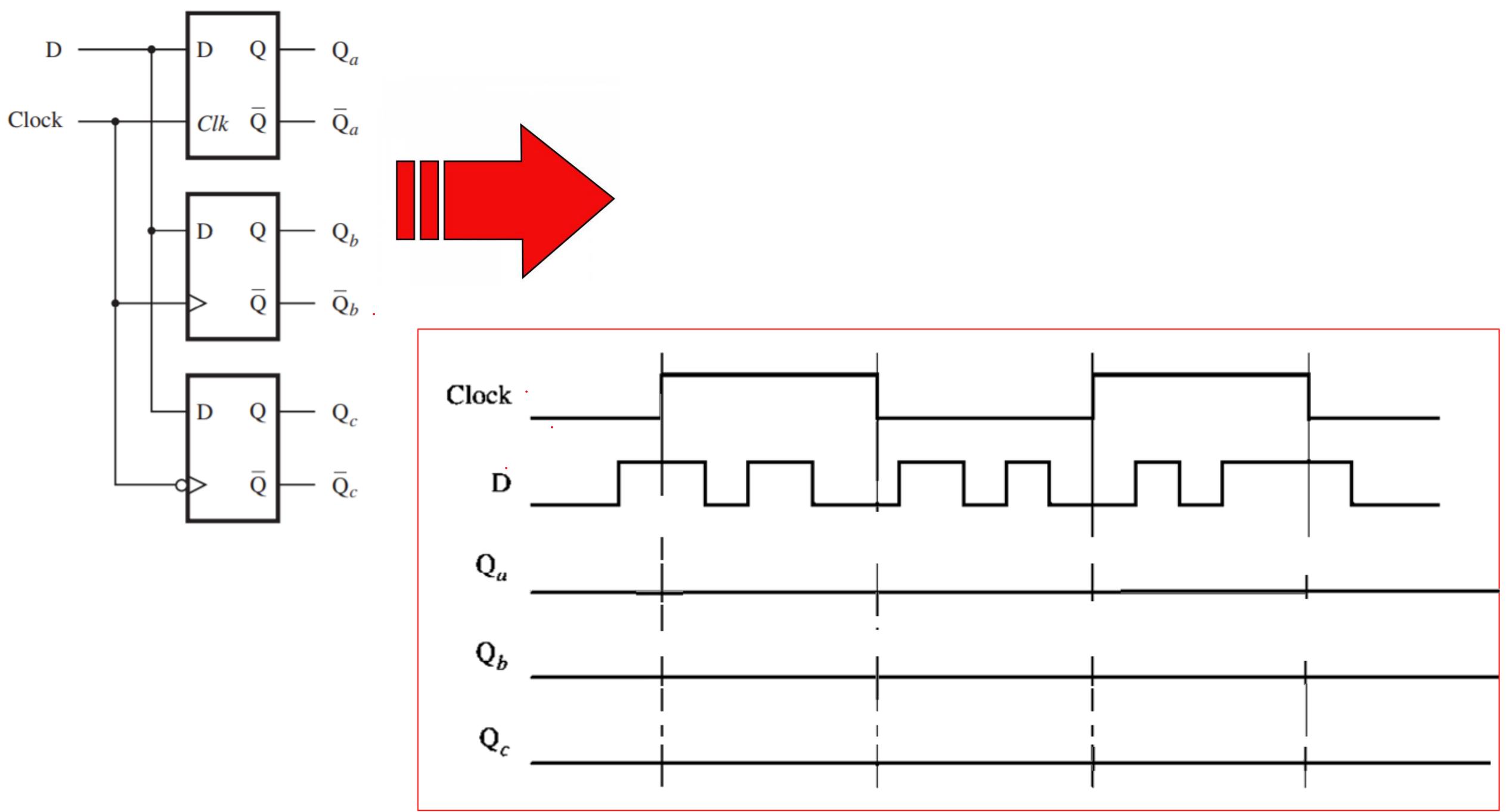
Entradas			Salidas		<i>Comentarios</i>
<i>J</i>	<i>K</i>	<i>CLK</i>	<i>Q</i>	$\bar{Q}$	
0	0	↑	$Q_0$	$\bar{Q}_0$	No cambio
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	$\bar{Q}_0$	$Q_0$	Basculación

↑ = transición del reloj de nivel BAJO a nivel ALTO

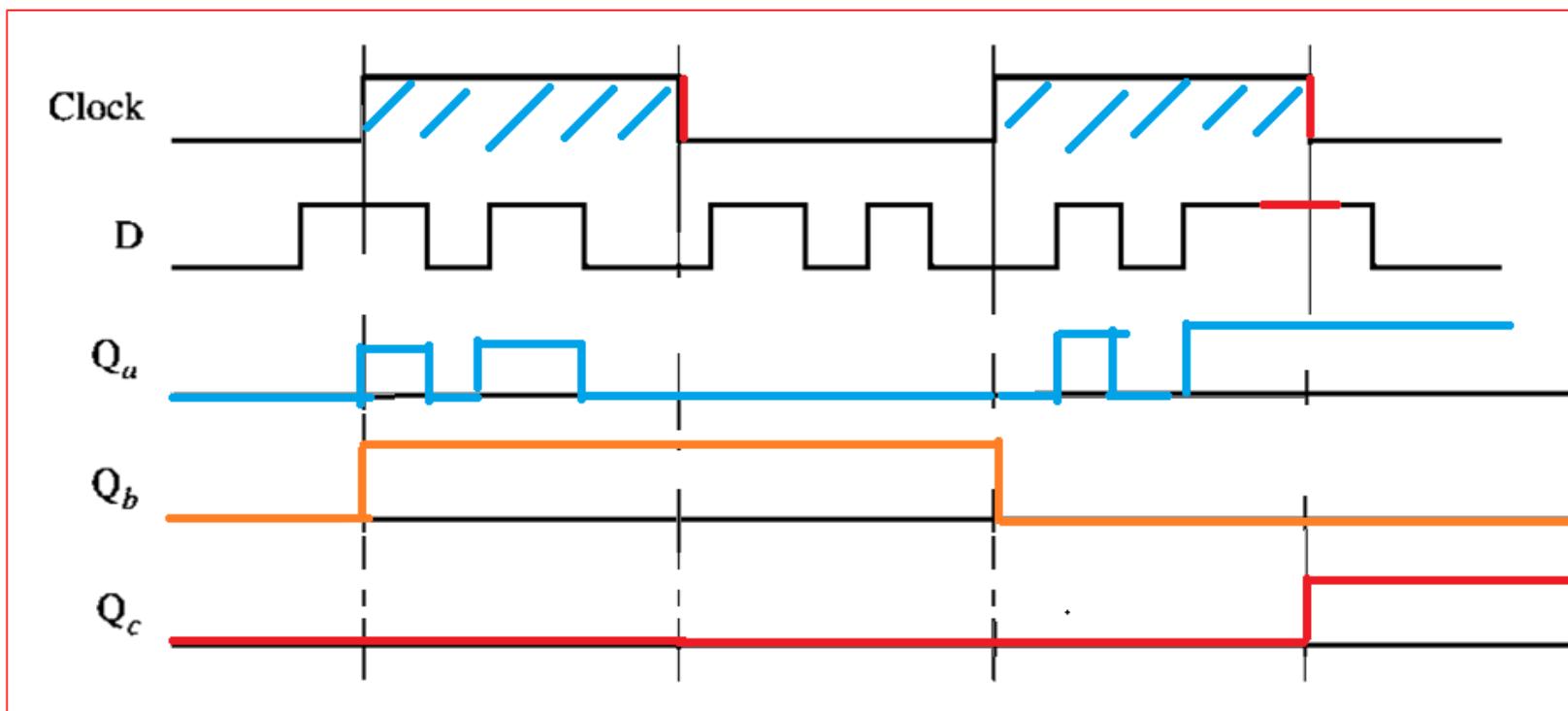
$Q_0$  = nivel de salida previo a la transición del reloj

Flip - flops sensibles al  
nivel vs elementos  
disparados por flanco





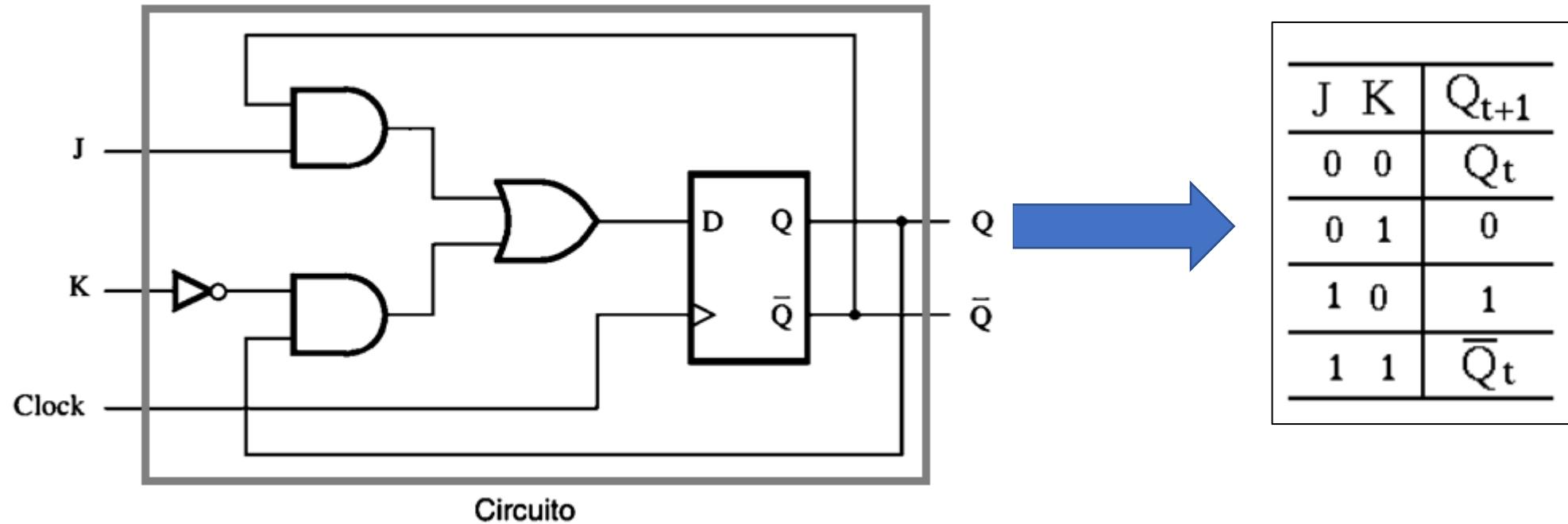
# Solución

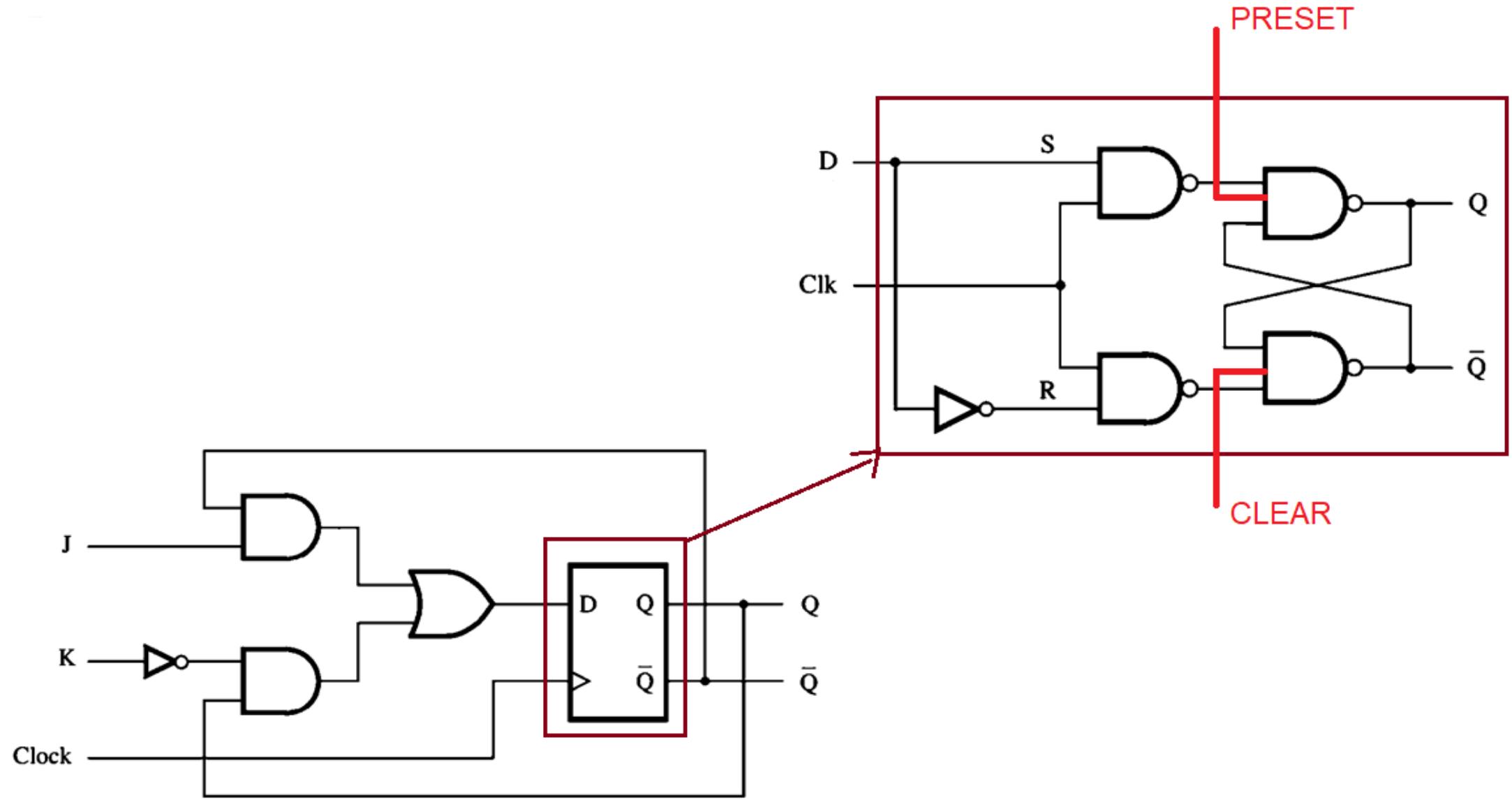




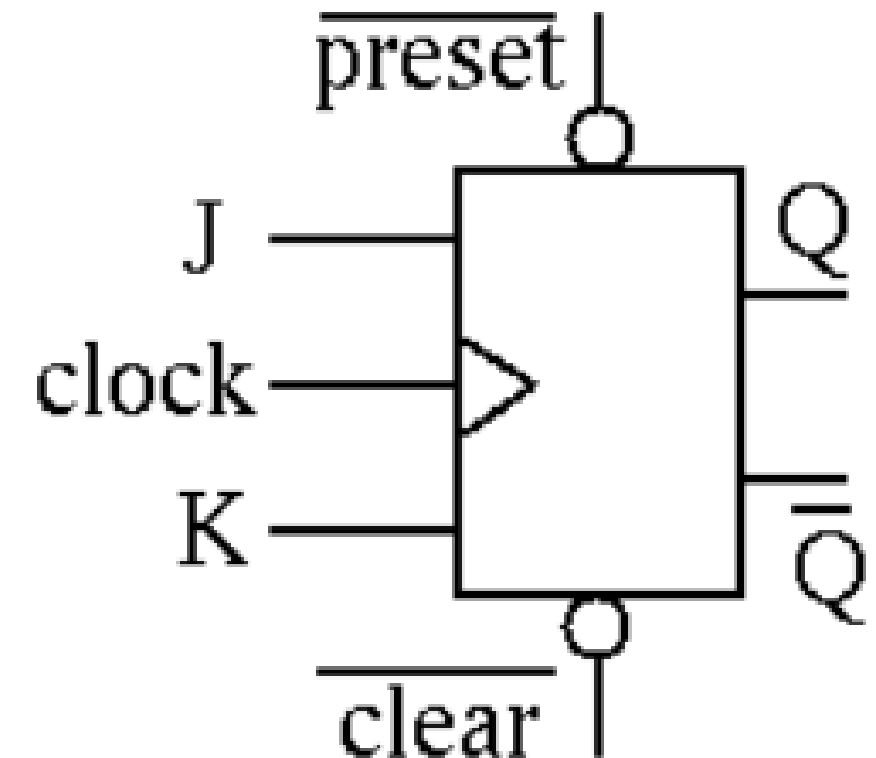
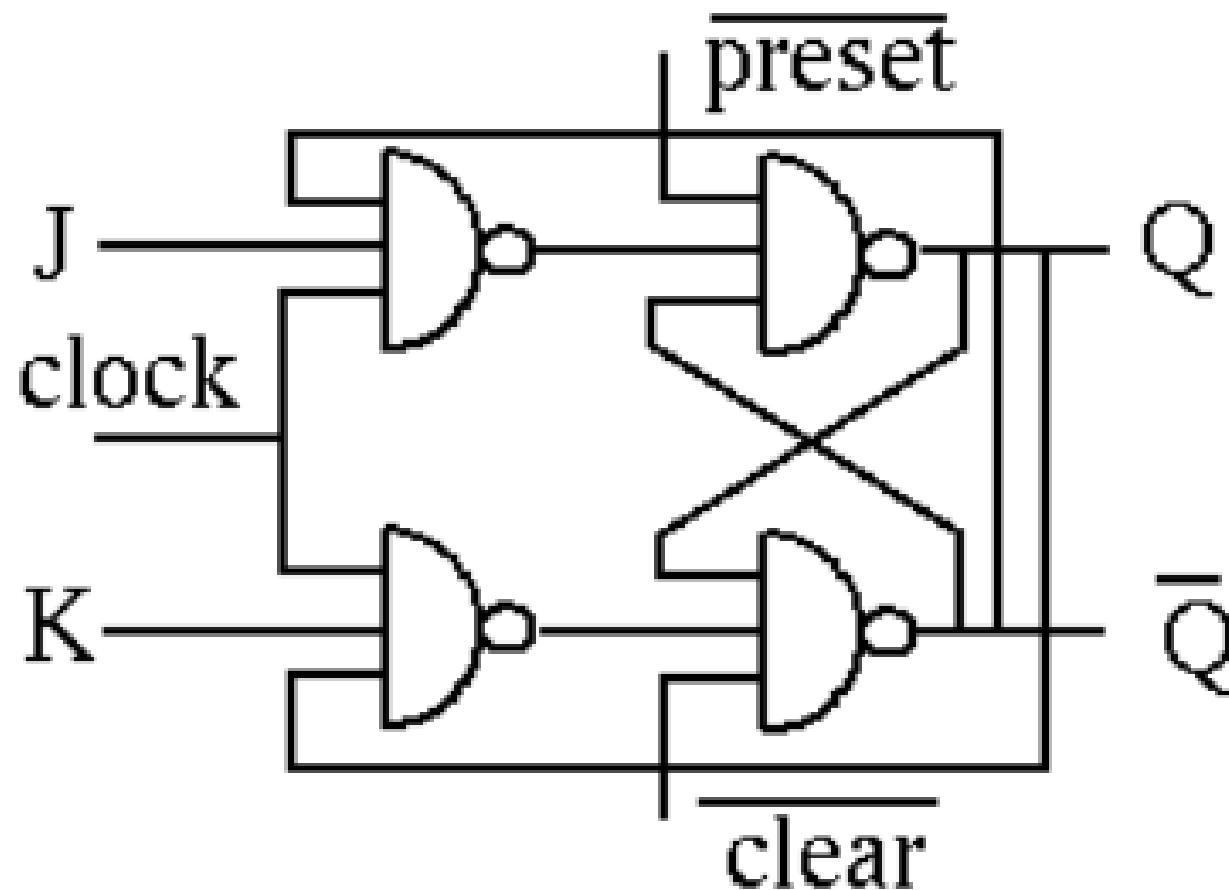
Flip Flops con entradas  
asíncronas

Flip Flop JK sin entradas de control externas utilizando un FF D como base.





## Entradas asíncronas Preset y Clear



	INPUTS			OUTPUTS			
	PRESET	CLEAR	CLOCK	J	K	Q	$\bar{Q}$
1	0	0	X	X	X	1	1
2	0	1	X	X	X	1	0
3	1	0	X	X	X	0	1
4	1	1	↓	0	0	0	$\bar{Q}$
5	1	1	↓	1	0	1	0
6	1	1	↓	0	1	0	1
7	1	1	↓	1	1	TOGGLE	
8	1	1	0	X	X	Q	$\bar{Q}$

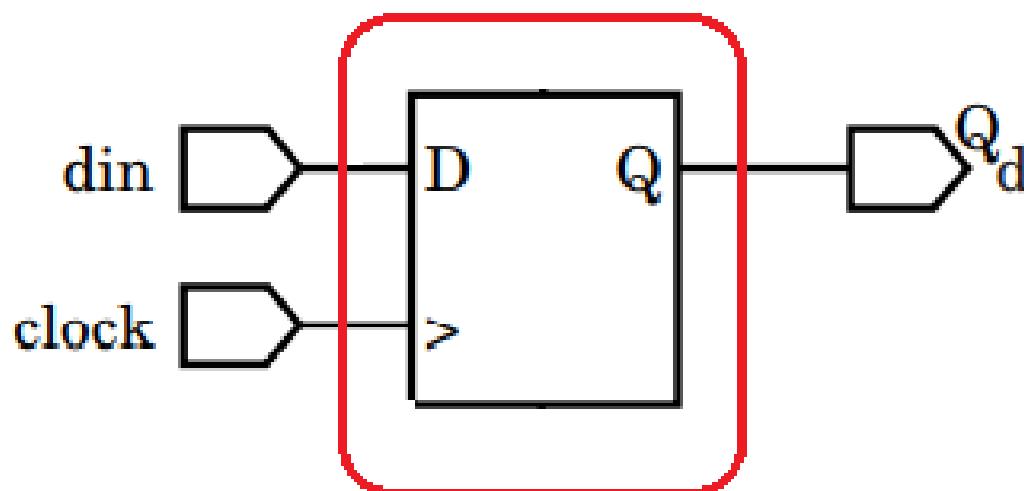
**X = IRRELEVANT**

Tabla  
característica  
con PRS y CLR



# Declaración de flip flops en hdl

# Declaración secuencial simple.

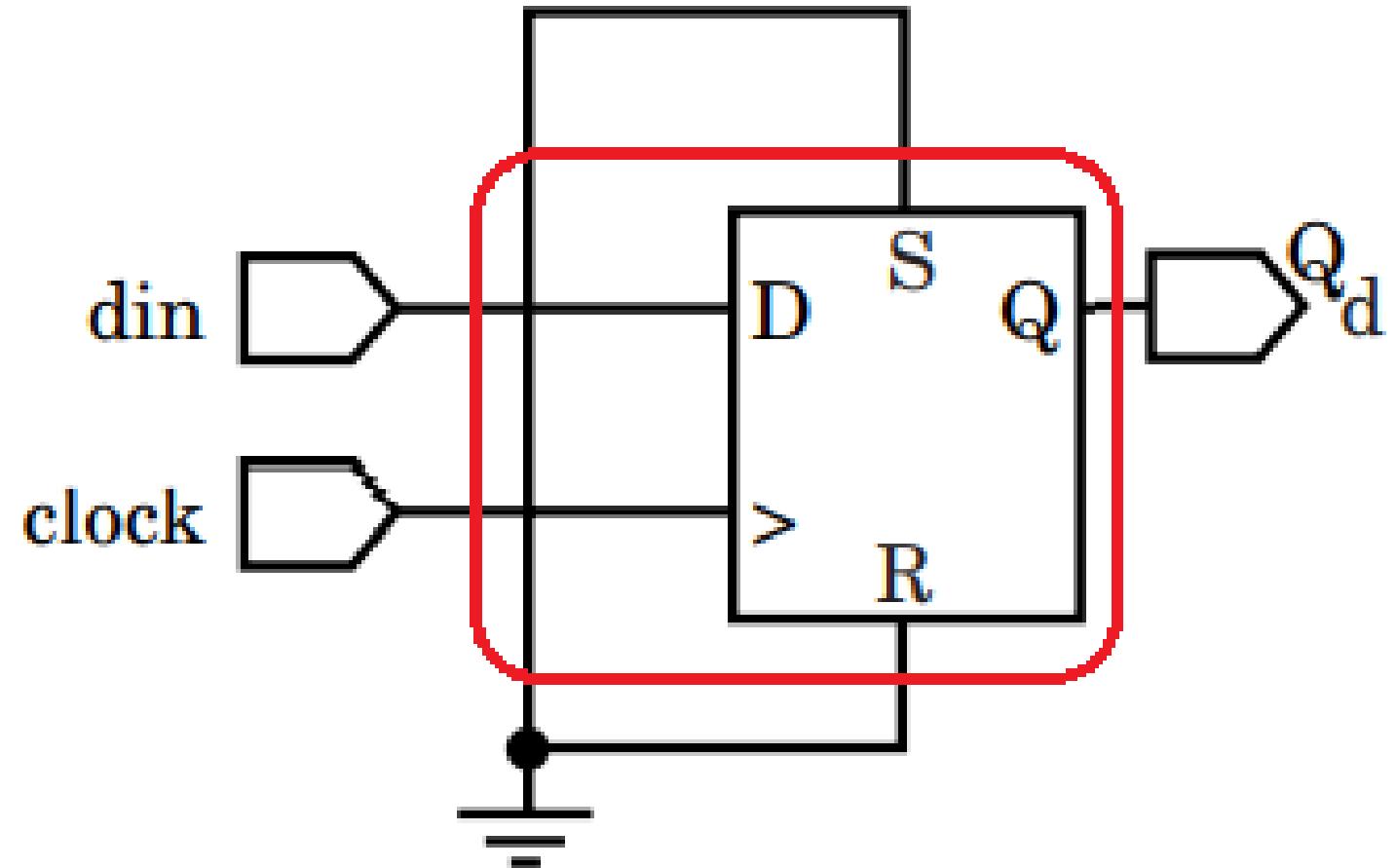


```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY dff IS
PORT( clock, din : IN std_logic;
      Qd : OUT std_logic);
END dff;

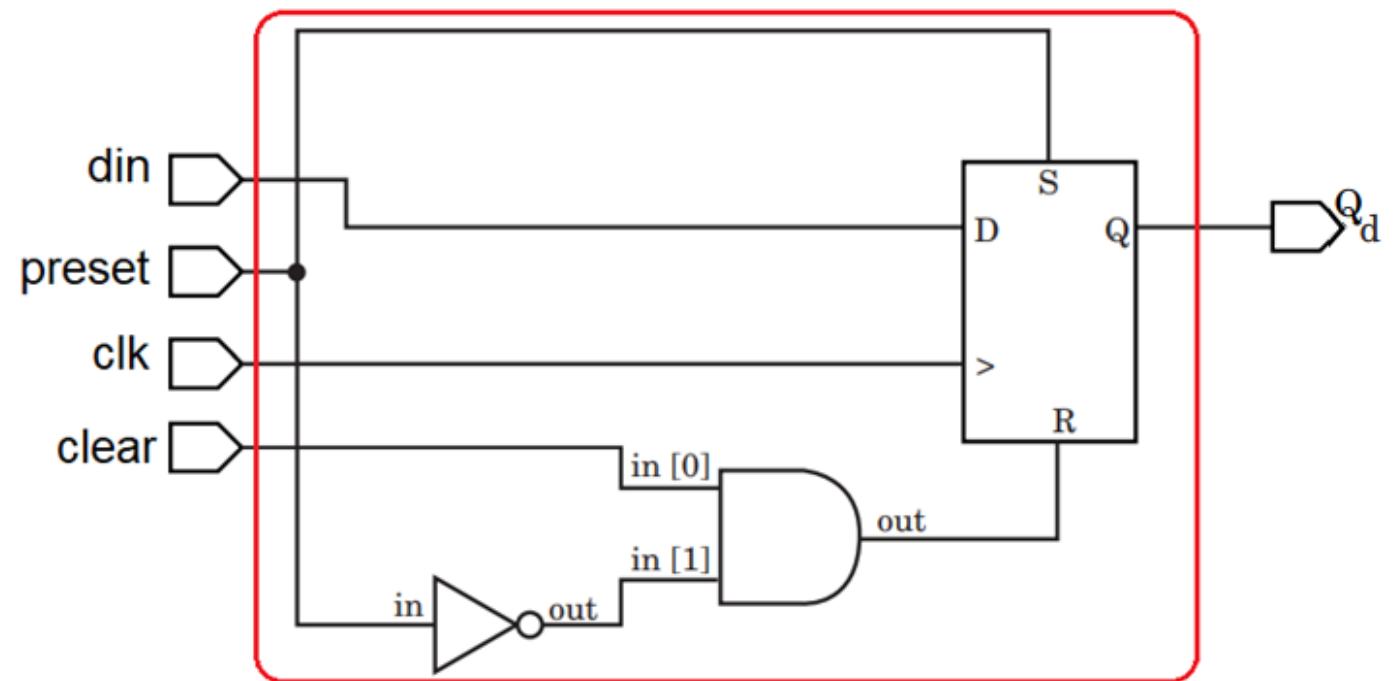
ARCHITECTURE ffd_asin OF dff IS
BEGIN
  PROCESS
    BEGIN
      WAIT UNTIL ((clock'EVENT) AND (clock = '1'));
      Qd <= din;
    END PROCESS;
END ffd asin;
```

**Set/preset y  
reset  
asíncronos**



```
--B-i-b-l-i-o-t-e-c-a-s--  
  
PORT( clock, reset, din : IN std_logic;  
      Qd : OUT std_logic);  
END dff_asincrono;  
  
ARCHITECTURE ffd_asin OF dff_asincrono IS  
BEGIN  
  PROCESS(reset, clock)  
    BEGIN  
      IF (reset = '1') THEN  
        Qd <= '0';  
      ELSEIF (clock'EVENT) AND (clock = '1') THEN  
        Qd <= din;  
      END IF;  
    END PROCESS;  
END ffd_asin;
```

## Declaración de un FF en HDL con entradas Preset y Clear



```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY FFD_pc IS
    PORT( preset, clear, clk, din : IN std_logic;
          Qd : OUT std_logic);
END FFD_pc;

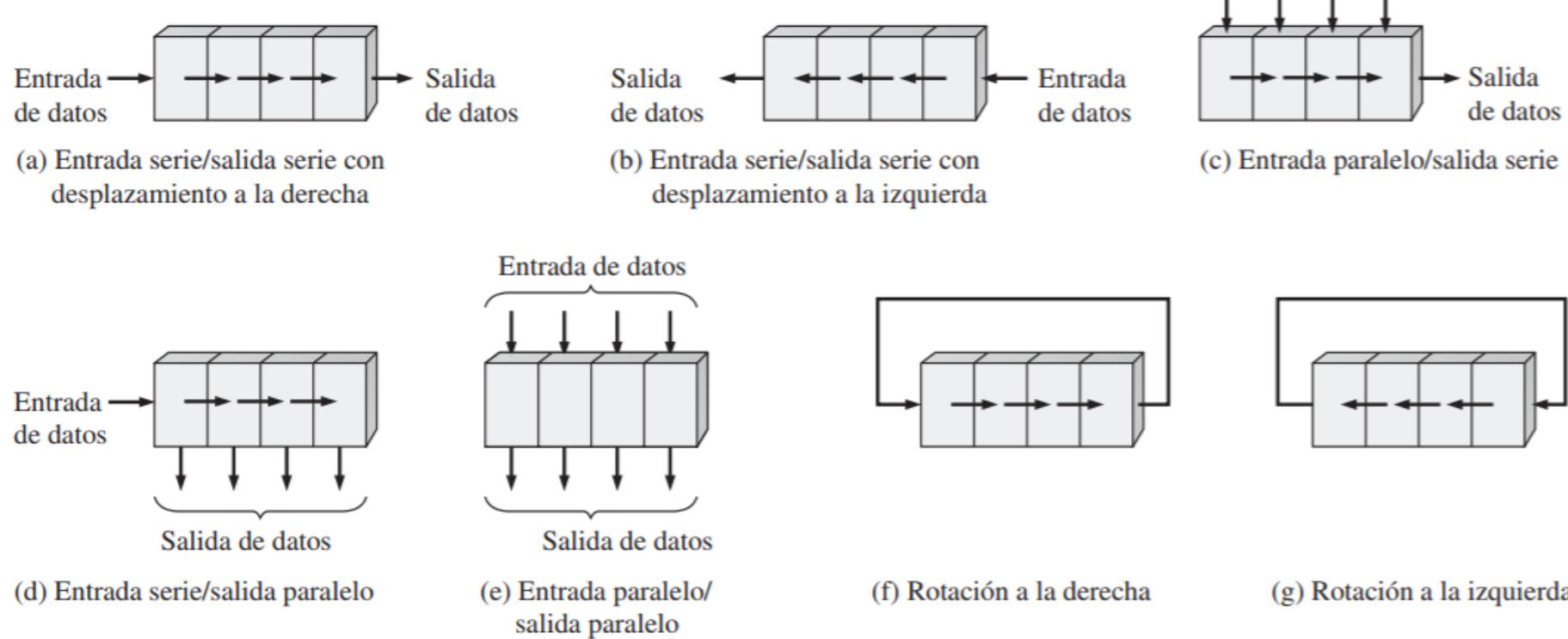
ARCHITECTURE sincrono OF FFD_pc IS
BEGIN
    PROCESS(preset, clear, clk)
        BEGIN
            IF (preset = '1') THEN
                Qd <= '1';
            ELSEIF (clear = '1') THEN
                Qd <= '0';
            ELSEIF (clk'EVENT) AND (clk = '1') THEN
                Qd <= din;
            END IF;
    END PROCESS;
END sincrono;
```

# Registros

# *Registro*

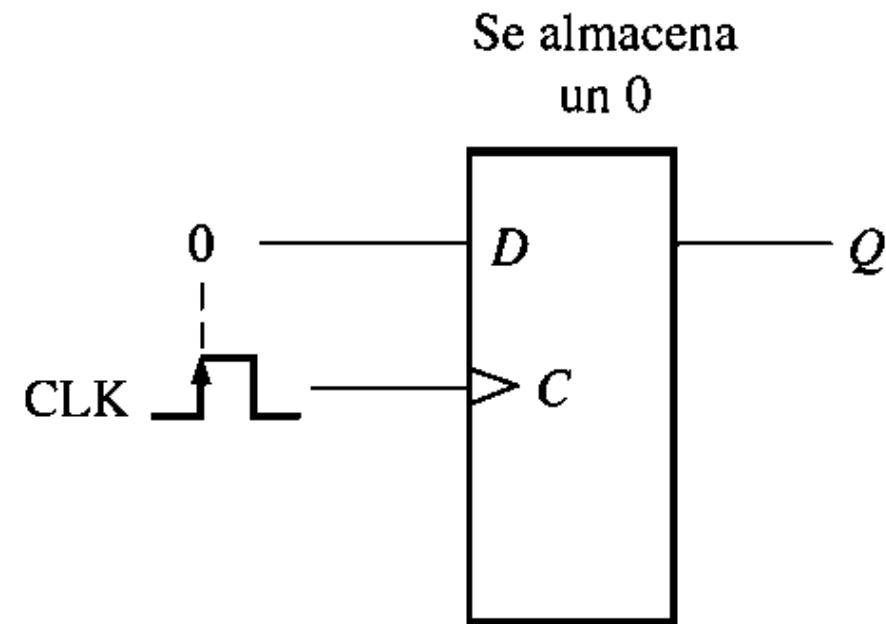
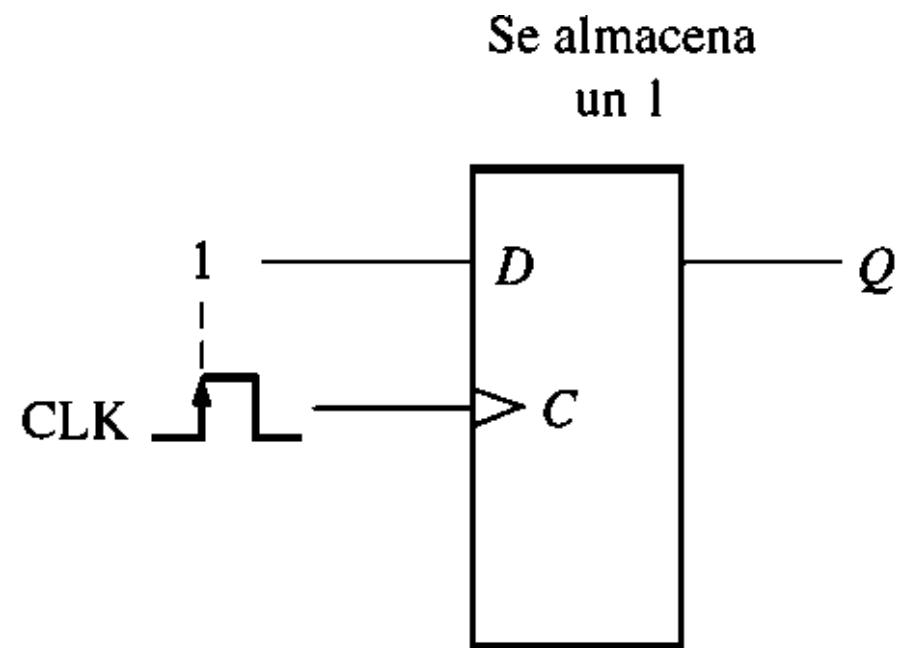
- Los registros de desplazamiento son circuitos lógicos secuenciales, y se construyen con flip-flops, se utilizan como memorias temporales y para desplazar datos a la izquierda o a la derecha; también se utilizan para convertir datos serie en paralelo o viceversa

- Un registro puede tener compuertas combinacionales que realizan ciertas tareas de procesamiento de datos.
- Los flip-flop contienen la información binaria y las compuertas determinan cómo esa información se transfiere al registro.
- La capacidad de almacenamiento de un registro le convierte en un tipo importante de dispositivo de memoria
- **\*\*Un contador es básicamente un registro que pasa por una sucesión predeterminada de estados. Las compuertas del contador están conectadas de tal manera que producen la sucesión prescrita de estados binarios\*\*.**

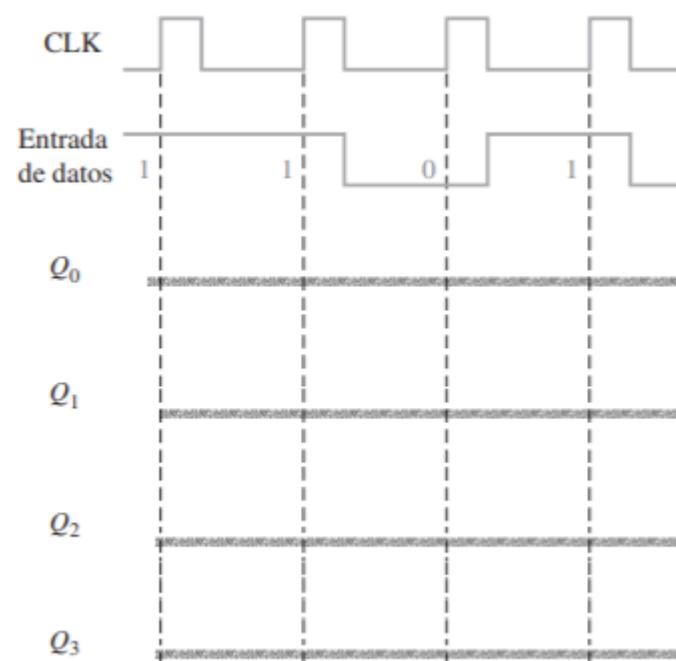
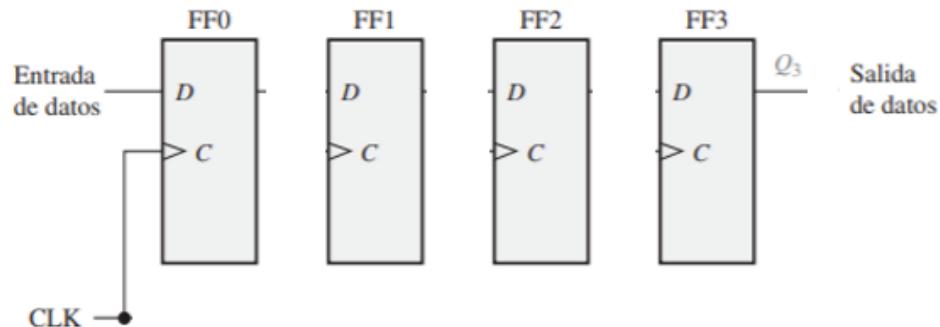
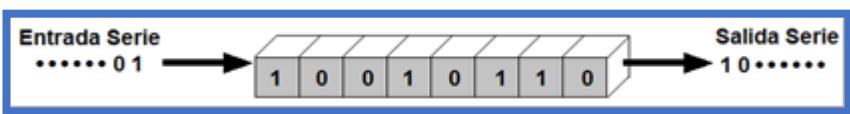


# Tipos de registros de 4 bits.

# Flip – flop como elemento de almacenamiento

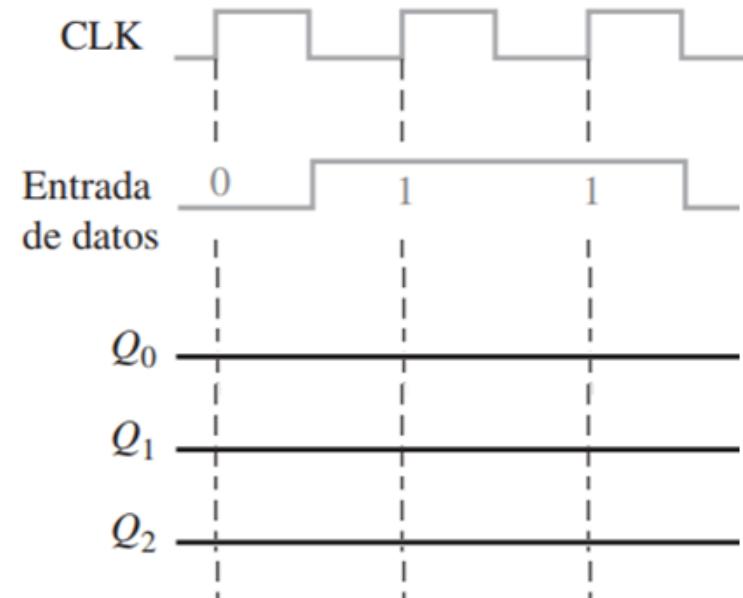
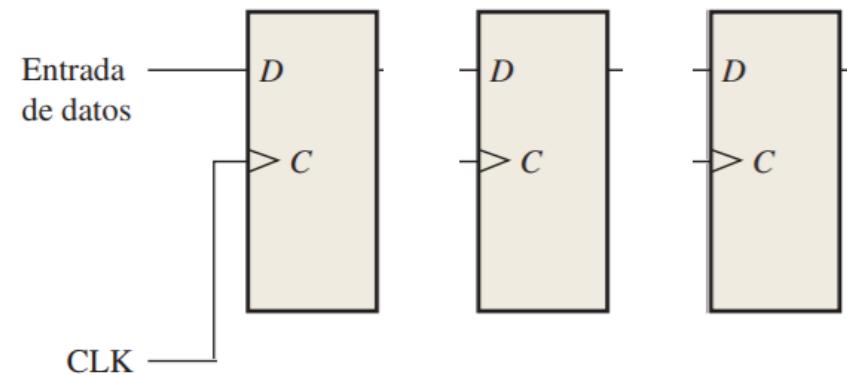
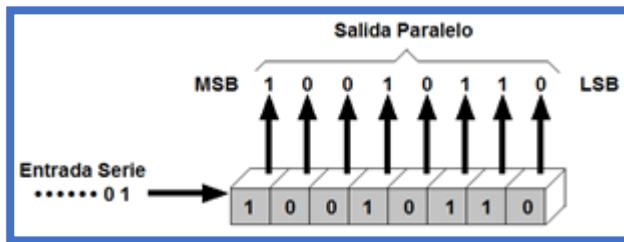


## Registros de entrada serie - salida serie



simulacion

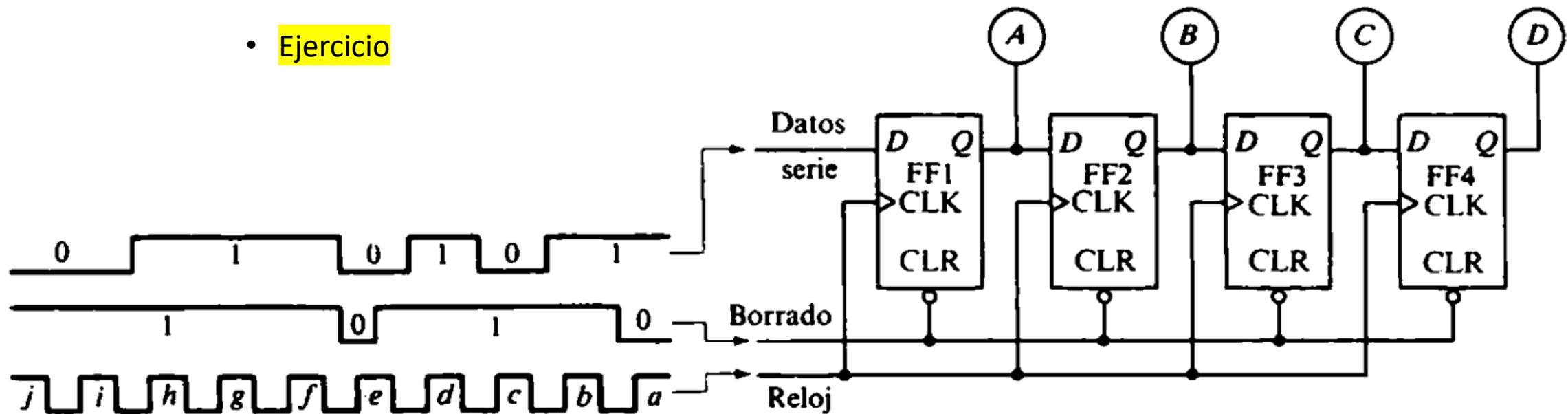
## Registros de entrada serie - salida paralelo



<http://tinyurl.com/yysxyk2h>

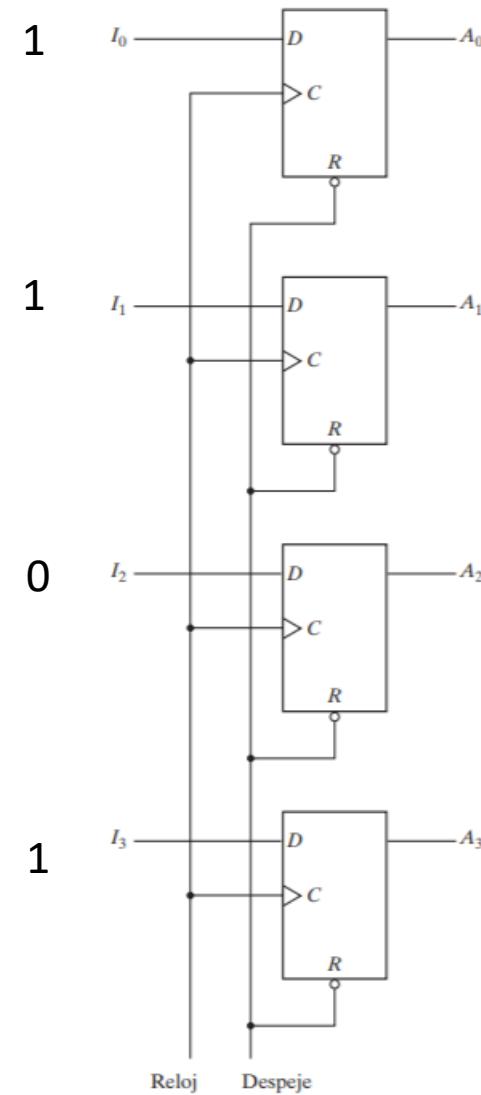


- Ejercicio



- Ejercicio. Realizar el diagrama de tiempos del siguiente registro operando con flancos de bajada para la señal de RELOJ.

<http://tinyurl.com/y6lh4chc>



# Funcionamiento de los Flip flops/latch.



Respuesta al nivel positivo



Respuesta al borde positivo



Respuesta al borde negativo

# REGISTROS CON VHDL

# Registros en VHDL.

## Variables vs señales

```
-- Uso incorrecto de las señales
ARCHITECTURE ejempl1 OF entidad IS
SIGNAL a,b,c,x,y: integer;
BEGIN
  p1: PROCESS(a,b,c)
  BEGIN
    c<=a; -- Se ignora
    x<=c+2;
    c<=b; -- Se mantiene
    y<=c+2;
  END PROCESS p1;
END ejempl1;
```

```
-- Uso correcto de las variables
ARCHITECTURE ejempl1 OF entidad IS
SIGNAL a,b,x,y: integer;
BEGIN
  p1: PROCESS(a,b)
  VARIABLE c: integer;
  BEGIN
    c:=a; -- Inmediato
    x<=c+2;
    c:=b; -- Inmediato
    y<=c+2;
  END PROCESS p1;
END ejempl1;
```

Latch con una  
entrada *d*, salida *q*  
y señal de  
habilitación *ena*

```
PROCESS(ena,d)
BEGIN
  IF ena='1' THEN q<=d;
  END IF;
END PROCESS;
```

- NOTA: En VHDL no es necesario poner dos biestables en serie para definir un registro; al utilizar una descripción comportamental, lo que se hace es definir su función, es decir, que la salida capture la entrada cuando se produzca uno de los flancos de reloj y que no haga nada en caso contrario..

## Declaración de un flip-flop en VHDL

```
PROCESS(clk,reset)
BEGIN
    IF reset<='1' THEN q<='0';
    ELSIF clk='1' AND clk'event THEN q<=d;
    END IF;
END PROCESS;
```

# Descripción de Lógica Secuencial

- Una de las propiedades más importantes de un *process* es la capacidad de su estructura para almacenar los valores de las señales que se asignan en su interior. Debido a esta característica se utilizarán los process para generar HW secuencial
- NOTA. Que un *process* genere HW secuencial no implica que las distintas instrucciones internas a un process se ejecuten secuencialmente

# Hardware Secuencial

```
if (CLK'event and CLK='1') then ...
```

Creación de un biestable D con reset

```
entity Biestable_rD is
    port(d, clk, reset: in std_logic; q: out std_logic);
end Biestable_rD;

architecture ARCH_ASYNC of Biestable_rD is
begin
    process (clk, reset, d)
    begin
        if (reset = '1') then q <= '0';
        elsif clk = '1' and clk'event then q <= d;
        end if;
    end process;
end ARCH_ASYNC;
```

```

architecture ARCH_SYN of Biestable_rD is
begin
    process (clk, reset, d)
    begin
        if clk = '1' and clk'event then q <= d;
            if (reset = '1') then q <= '0';
            end if;
        end if;
    end process;
end ARCH_SYN;

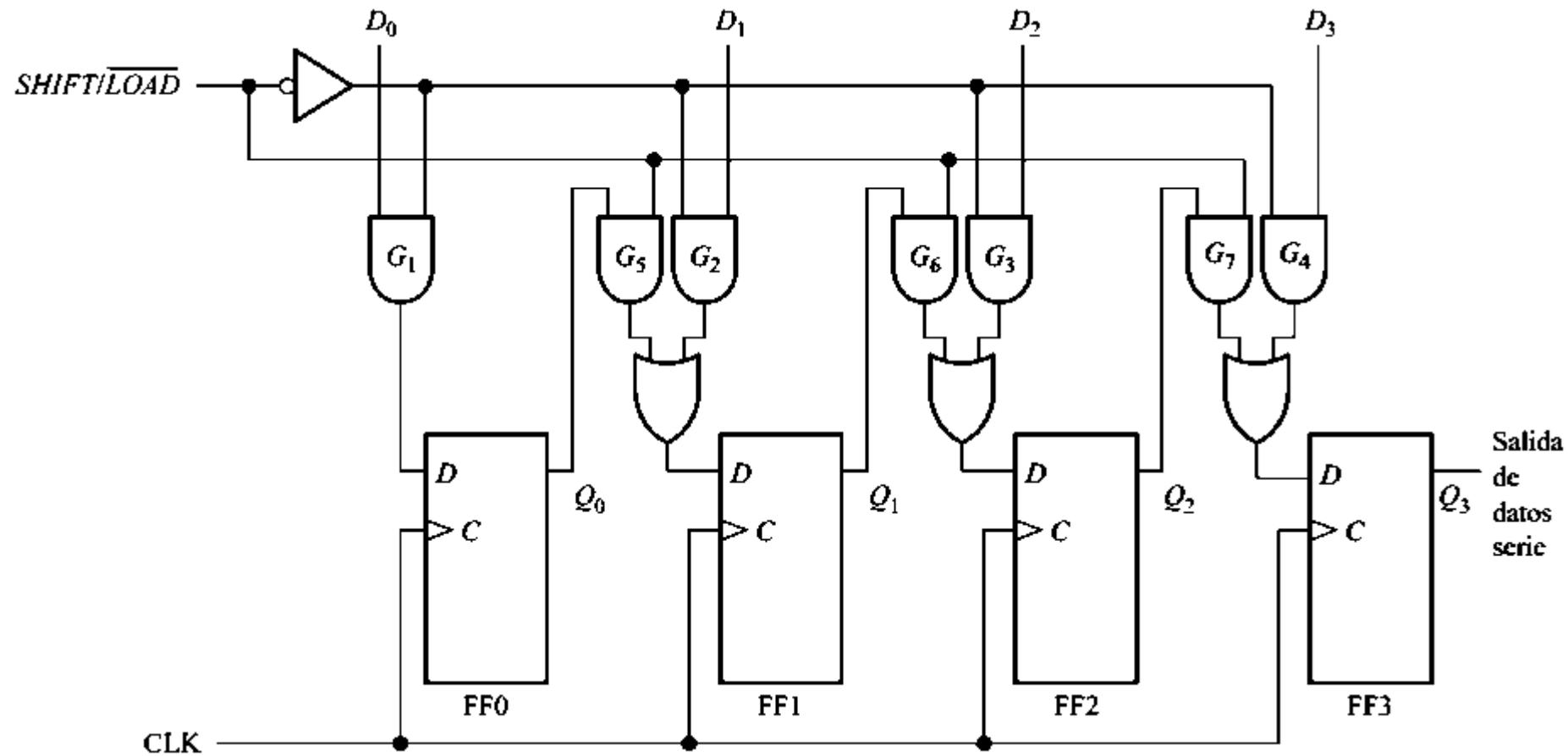
```

Para conseguir crear el HW secuencial esperado hay que cumplir una serie de reglas:

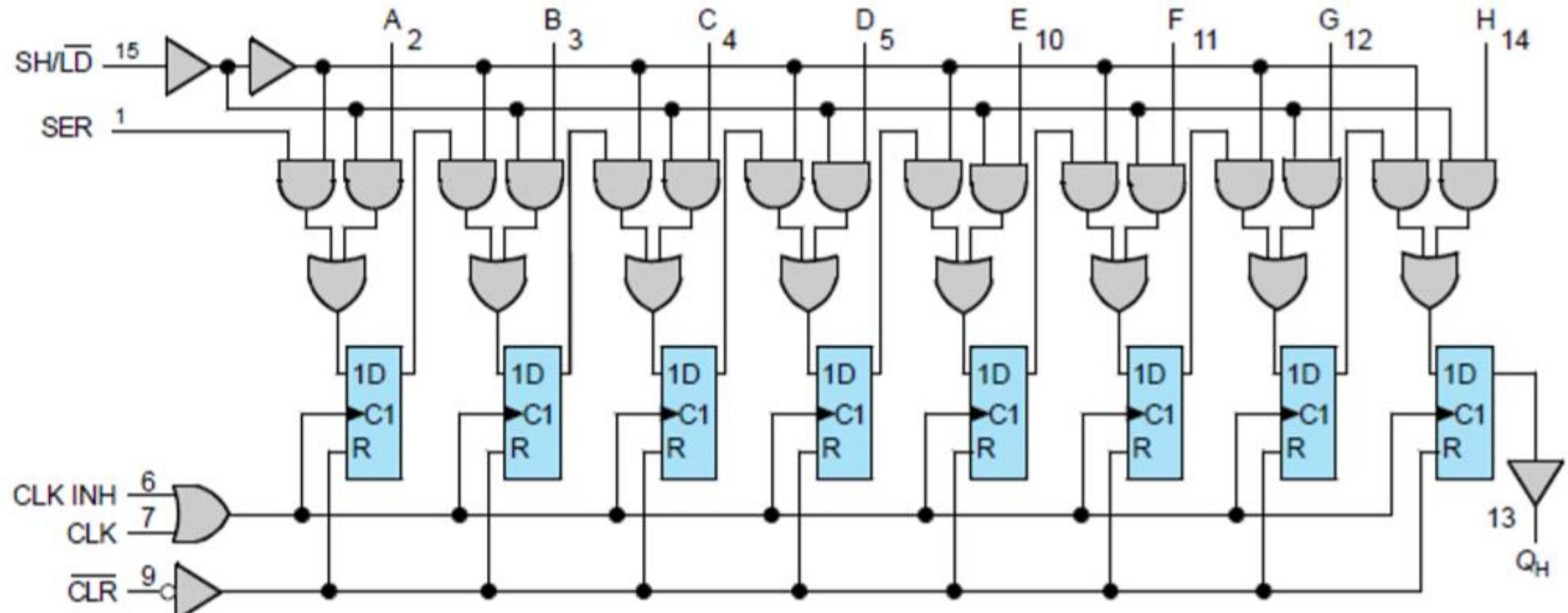
- Una sentencia ***if*** que tenga por condición una especificación de flanco no puede tener rama ***else***, en caso contrario la rama ***else*** debería realizarse en todo momento menos en el preciso instante en el que el reloj cambia.

- En sentencias ***if-then-elsif*** la especificación de flanco sólo podrá ser la condición del último ***elsif*** (que no podrá tener rama ***else***).
- Una sentencia ***if*** que tenga por condición una especificación de flanco puede tener internamente sentencias ***if-else*** encadenadas.
- En un ***process*** solo puede existir una única especificación de flanco, en caso contrario se estaría especificando HW secuencial sensible a varios relojes.

# Registros de entrada paralela- salida serie

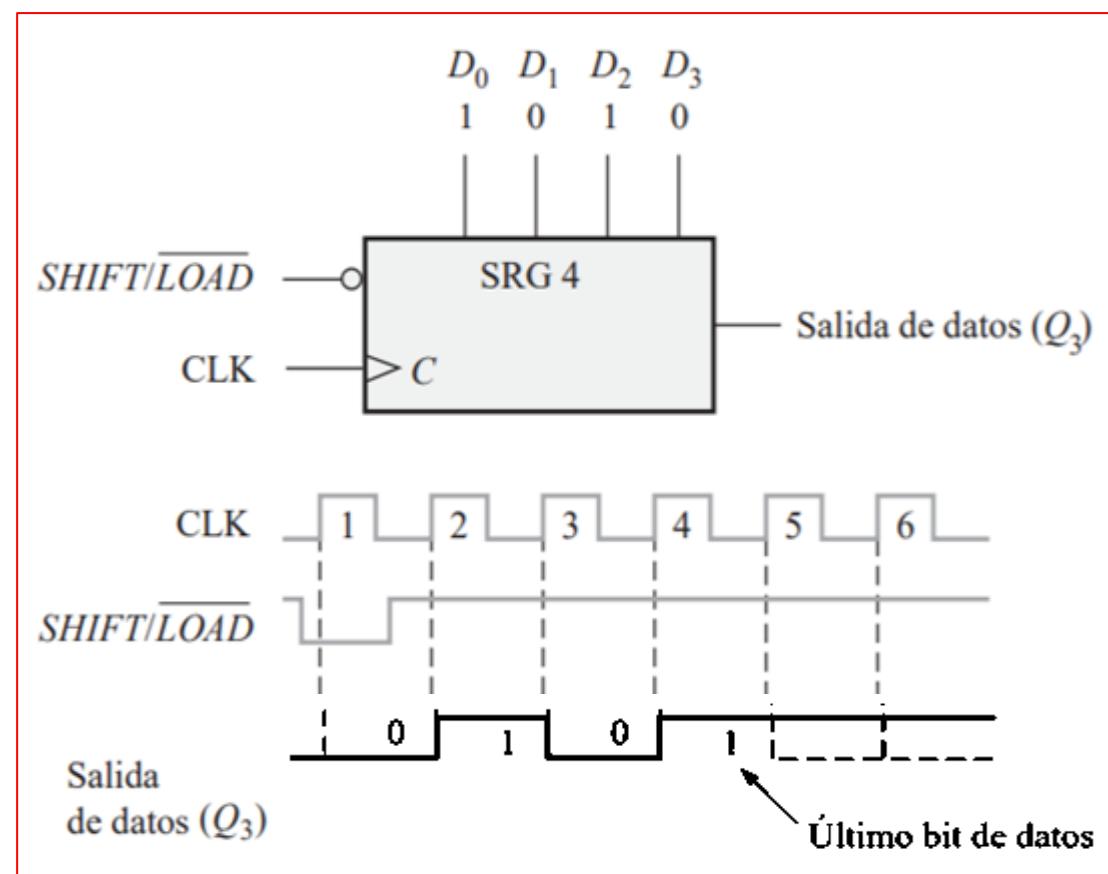


# Serie TTL 74166 registro de desplazamiento de 8 bits



La carga de datos en paralelo del Registro puede ser del tipo Síncrona o asíncrona

- ▶ En el impulso de reloj 1, los datos paralelo ( $D_0D_1D_2D_3 = 1010$ ) se cargan en el registro, poniendo la salida Q3 a 0. En el impulso de reloj 2, el 1 de Q2 se desplaza a Q3; en el impulso de reloj 3, el 0 se desplaza a Q3; en el impulso de reloj 4, el último bit de datos (1) se desplaza a Q3 y en el impulso de reloj 5 todos los bits se han desplazado y salido del registro, y sólo quedan 1s en el mismo (suponiendo que la entrada D0 permanece a 1)



Registro de desplazamiento de 4 bits con entrada paralelo salida serie

## Registro de desplazamiento

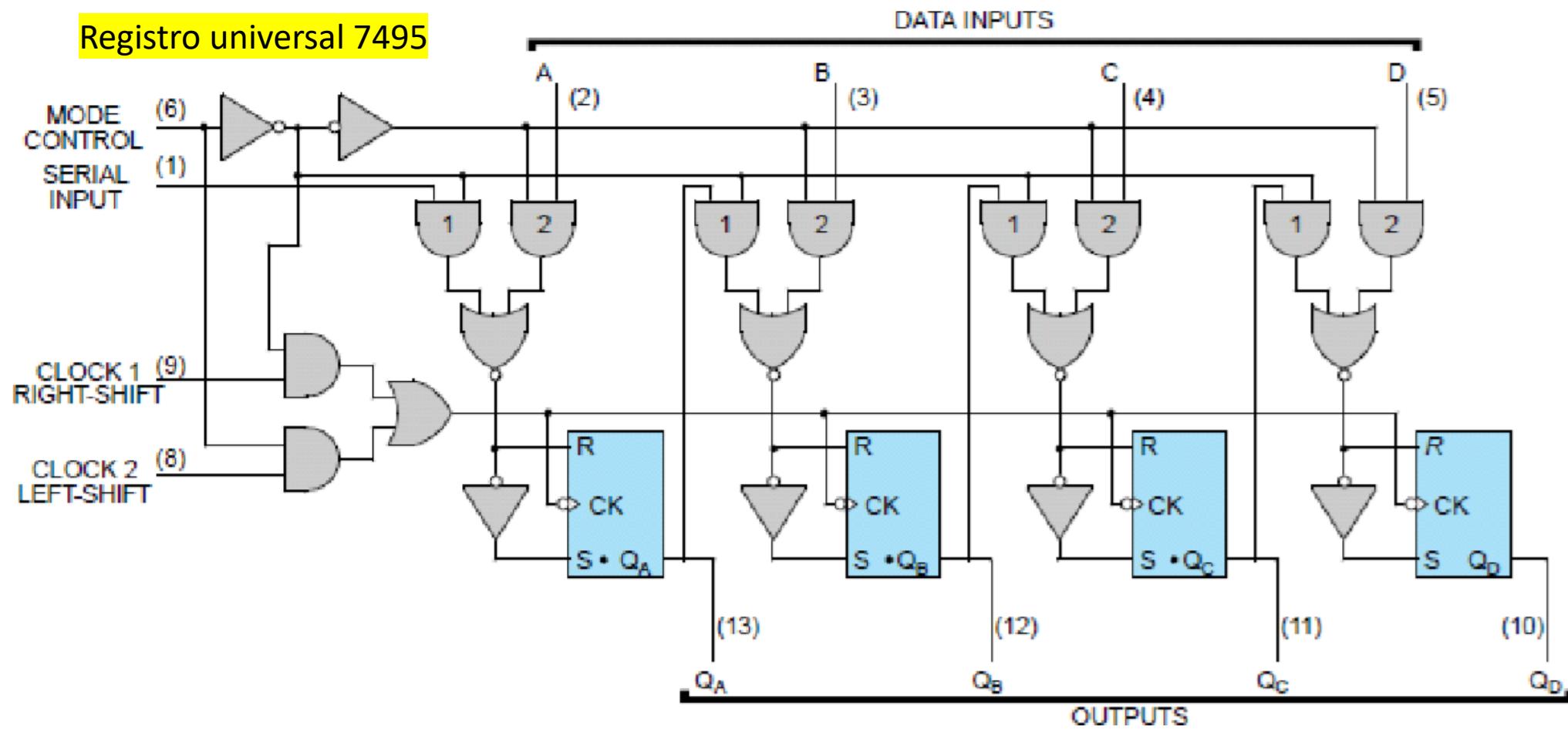
- Consiste en una cadena de flip-flops en cascada, con la salida de un flip-flop conectada a la entrada del siguiente flip-flop. Todos los flip-flops reciben pulsos de reloj comunes, que activan el desplazamiento de una etapa a la siguiente.

## Registro bidireccional

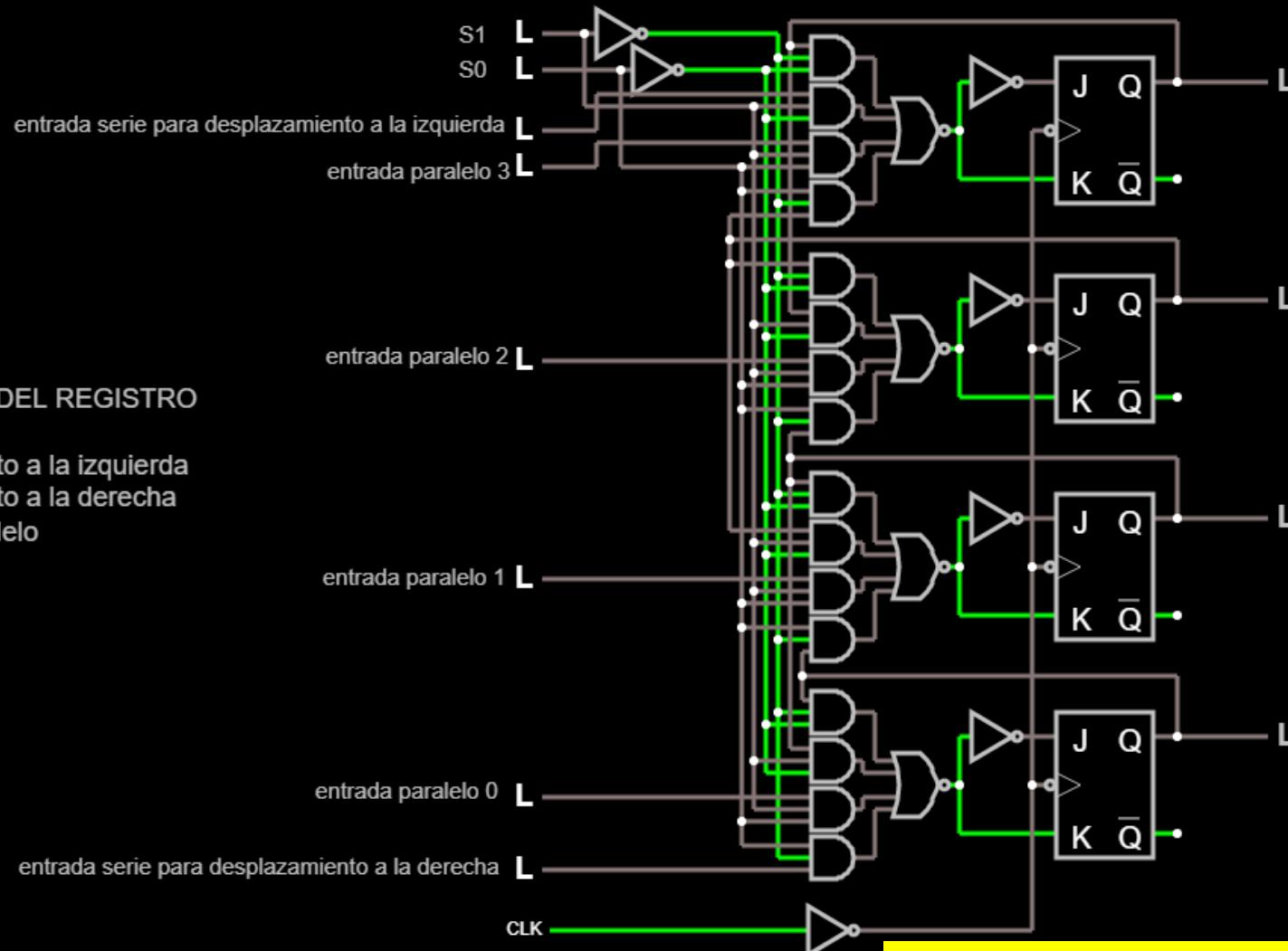
- Este tipo de registro tiene la opción de elegir la dirección en que se transmiten los datos. Estos registros tienen una señal de control que permite seleccionar el sentido de desplazamiento de los datos.

Un registro que sólo puede desplazar en una dirección es un registro de desplazamiento unidireccional. Uno que puede hacerlo en ambas direcciones es un registro de desplazamiento bidireccional. Si el registro tiene ambos desplazamientos y capacidad de carga paralela, se denomina **registro de desplazamiento universal**.

Registro universal 7495

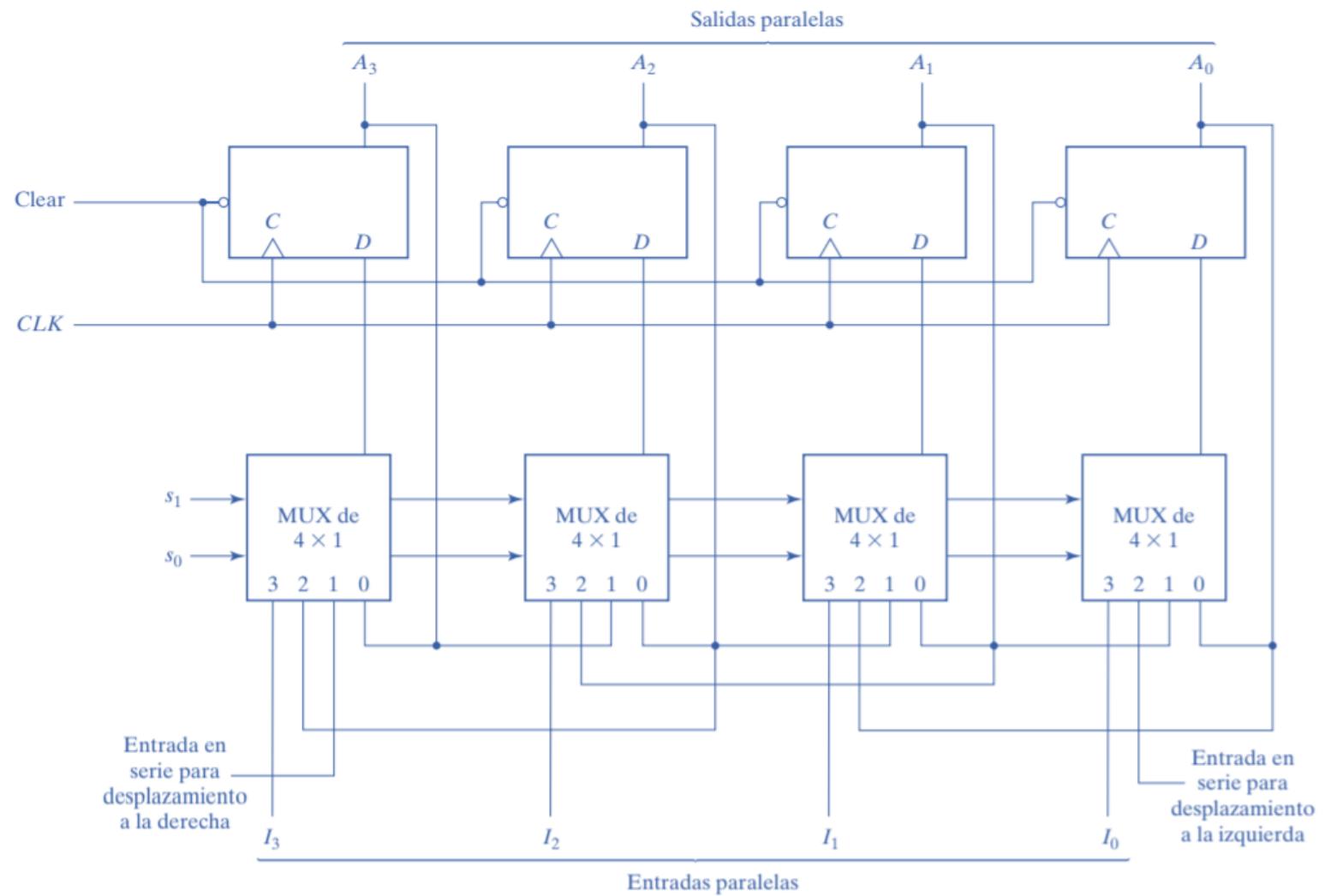


S1 S0   OPERACIÓN DEL REGISTRO	
L	L   sin cambio
L	H   desplazamiento a la izquierda
H	L   desplazamiento a la derecha
H	H   carga en paralelo



<https://tinyurl.com/yfh2k3yq>

# Registro de desplazamiento universal de 4 bits

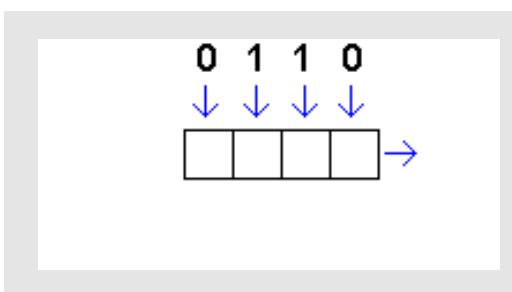
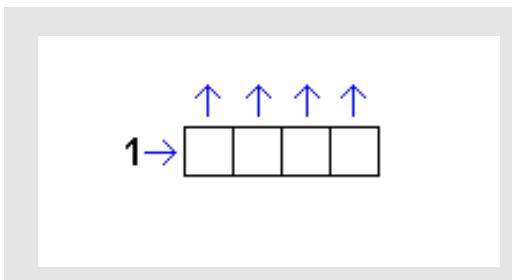
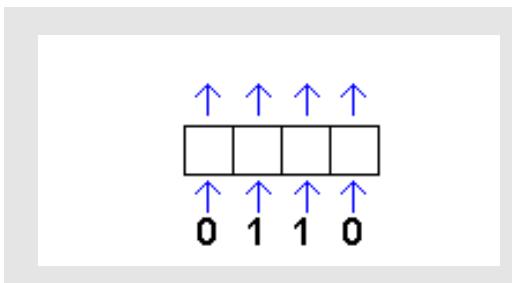
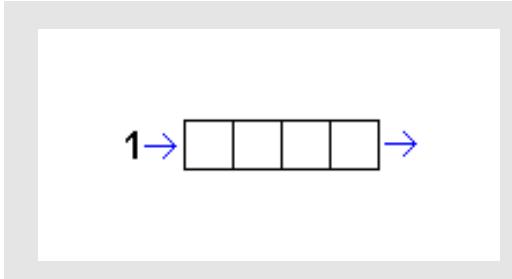


# Tabla de operación de un registro universal de 4 bits

## Control de modo

$s_1$	$s_0$	Operación del <i>registro</i>
0	0	Sin cambio
0	1	Desplazamiento a la derecha
1	0	Desplazamiento a la izquierda
1	1	Carga en paralelo

# RESUMEN DE REGISTROS.



- *Serie*: los bits se transfieren uno a continuación del otro por una misma línea.
- *Paralelo*: se intercambian todos los bits al mismo tiempo, utilizando un número de líneas de transferencia igual al número de bits.

- El uso de los registros de desplazamiento dentro de la electrónica secuencial es de gran utilidad, ya que pueden utilizarse para diferentes aplicaciones como:

***Multiplicadores***: realizan la multiplicación mediante sumas y desplazamientos.

***Circuitos de Retardo***: pueden utilizar (para retardar un bit) un número entero de ciclos de reloj. Consiste simplemente en un conjunto de biestables en cascada (tantos como ciclos de reloj deseemos retardar los bits).