

# Tema 6: Sistemas Secuenciales



Universidad  
de Huelva

Escuela Técnica Superior de Ingeniería

Departamento de  
Ingeniería Electrónica, Sistemas Informáticos y Automática

# Tema 6: Sistemas Secuenciales

## Introducción

Los circuitos lógicos se clasifican en dos tipos:

- **Combinacionales**, aquellos cuyas salidas sólo dependen de las entradas **actuales**.
- **Secuenciales**, aquellos cuyas salidas dependen no sólo de sus entradas actuales, sino también de sus entradas **anteriores**.

Esta “información” de las entradas anteriores, debe preservarse en el circuito y se denomina **estado interno** o simplemente **estado del circuito**. Es necesario distinguir el valor presente de una señal del que poseía en un instante inmediatamente anterior, y éste del anterior,... Por ello habrá una intervención explícita del tiempo. Es decir, los sistemas secuenciales necesitan “recordar” su pasado, la “secuencia de vectores de entrada” a través de la cual se ha llegado a la situación presente; para ello han de tener “**memoria**”, que se configura con un “vector de estado”. Este vector de estado está formado por las denominadas **variables de estado**.

Un sistema secuencial posee  $2^n$  estados de entrada para  $n$  entradas ( $X_1...X_n$ ). Poseen además  $2^p$  estados de salida para  $p$  salidas ( $Y_1...Y_p$ ) y un número finito de estados internos ( $Q_1...Q_m$ ) de ahí que sean conocidos como **autómatas finitos**.

Según la relación entre las salidas y los estados internos podemos distinguir dos tipos de autómatas:

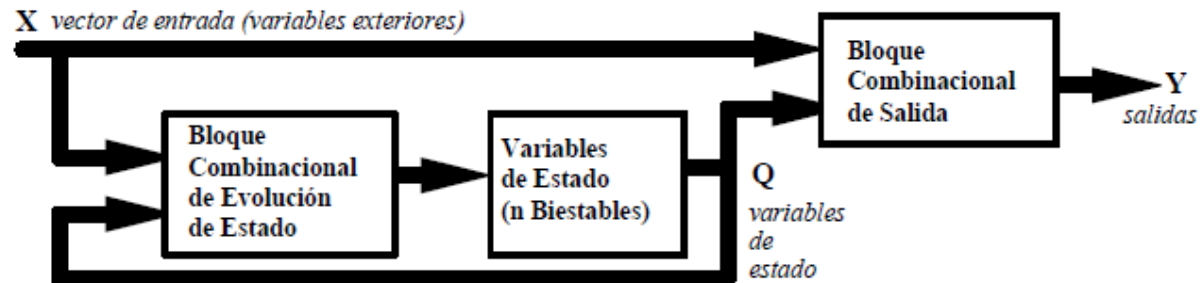
- ❑ **AUTÓMATA de MEALY**, las salidas se obtienen en función de las entradas y los estados internos.
- ❑ **AUTÓMATA de MOORE**, las salidas coinciden o dependen solo de los estados internos.

El autómata más genérico es el de Mealy. Existe un autómata de Moore equivalente a cada autómata de Mealy y viceversa.

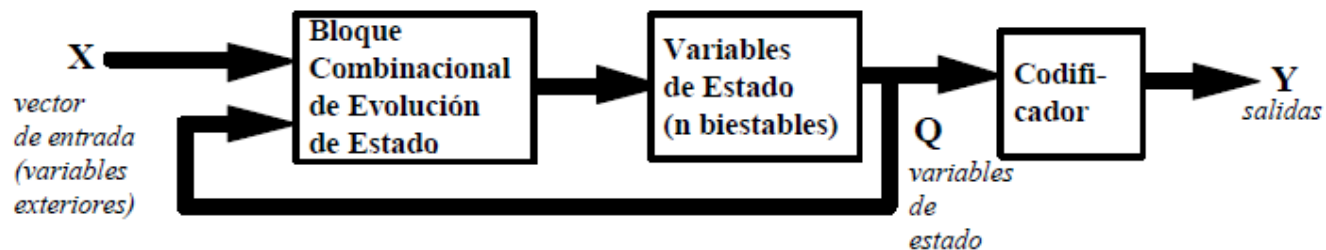
## Tema 6: Sistemas Secuenciales

### Introducción (II)

Todo sistema secuencial puede ser construido mediante un conjunto de elementos de memoria, que contendrán las variables de estado, y el resto del sistema será meramente combinacional. Por tanto, un autómata engloba tres conjuntos de variables: entradas (X), salidas (Y) y estado (Q). La relación entre el conjunto de variables de salida y el resto de conjuntos de variables (estado y entradas) determina el tipo de autómata.



Autómata de MEALY



Autómata de MOORE

# Tema 6: Sistemas Secuenciales

## Tipos de Sistemas Secuenciales

Según la forma de realizar el elemento de memoria nos podemos encontrar distintos tipos de sistemas secuenciales, principalmente dos:

- **Sistemas Secuenciales Síncronos**: su comportamiento puede definirse en instantes discretos de tiempo. Se necesita una sincronización de los elementos del sistema mediante una señal de reloj, que no es más que un tren de pulsos periódico. Las variables internas no cambian hasta que llega un pulso del reloj.
- **Sistemas Secuenciales Asíncronos**: actúan de forma continua en el tiempo, un cambio de las entradas provoca cambios en las variables internas sin esperar a la intervención de un reloj. Son sistemas más difíciles de diseñar.

El cambio de las variables internas se puede producir de dos maneras en un sistema secuencial síncrono:

- **Por niveles**: cuando permiten que las variables de entrada actúen sobre el sistema en el instante en el que la señal de reloj toma un determinado nivel lógico (0 ó 1).
- **Por flancos, o cambios de nivel**: cuando la acción de las variables de entrada sobre el sistema se produce cuando ocurre un flanco activo del reloj. Este flanco activo puede ser de subida (cambio de 0 a 1) o de bajada (cambio de 1 a 0).

El elemento de memoria básico de los circuitos secuenciales síncronos es el **biestable**. Almacena el estado **0** ó el estado **1**, y de ahí su nombre, tienen dos estados estables de funcionamiento. También se les suele conocer como **FLIP-FLOPS**.

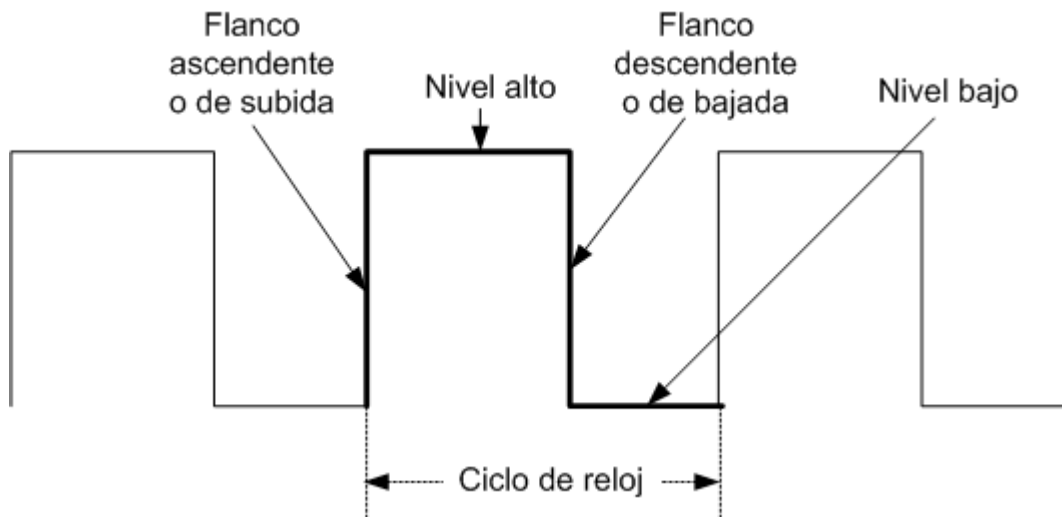
## Tema 6: Sistemas Secuenciales

### Concepto de señal de sincronismo: el reloj

La señal de reloj es la que “sincroniza” el funcionamiento de las memorias en el sistema secuencial. Estudiaremos en este tema exclusivamente los sistemas secuenciales totalmente síncronos. Esto significa que la señal de sincronismo llega al mismo tiempo a todos los biestables. El biestable totalmente síncrono sólo puede cambiar su estado interno (el bit que almacena) cuando se produce un determinado cambio en la señal de reloj.

La señal de reloj es una sucesión de unos y ceros, normalmente producidos por un oscilador de cuarzo, con un periodo de tiempo fijo. El periodo completo se denomina también **ciclo de reloj**. Su frecuencia viene determinada por el número de ciclos que se producen en un segundo (1 ciclo/segundo = 1 Hercio). El tiempo de duración de un ciclo puede obtenerse fácilmente con la fórmula:

$$t = \frac{1}{f}; \quad f = \text{frecuencia del reloj en Hercios}$$



Múltiplos de frecuencia	Nº de ciclos en un segundo
Kilohercio (KHz)	1000
Megahercio (MHz)	$10^6$
Gigahercio (GHz)	$10^9$

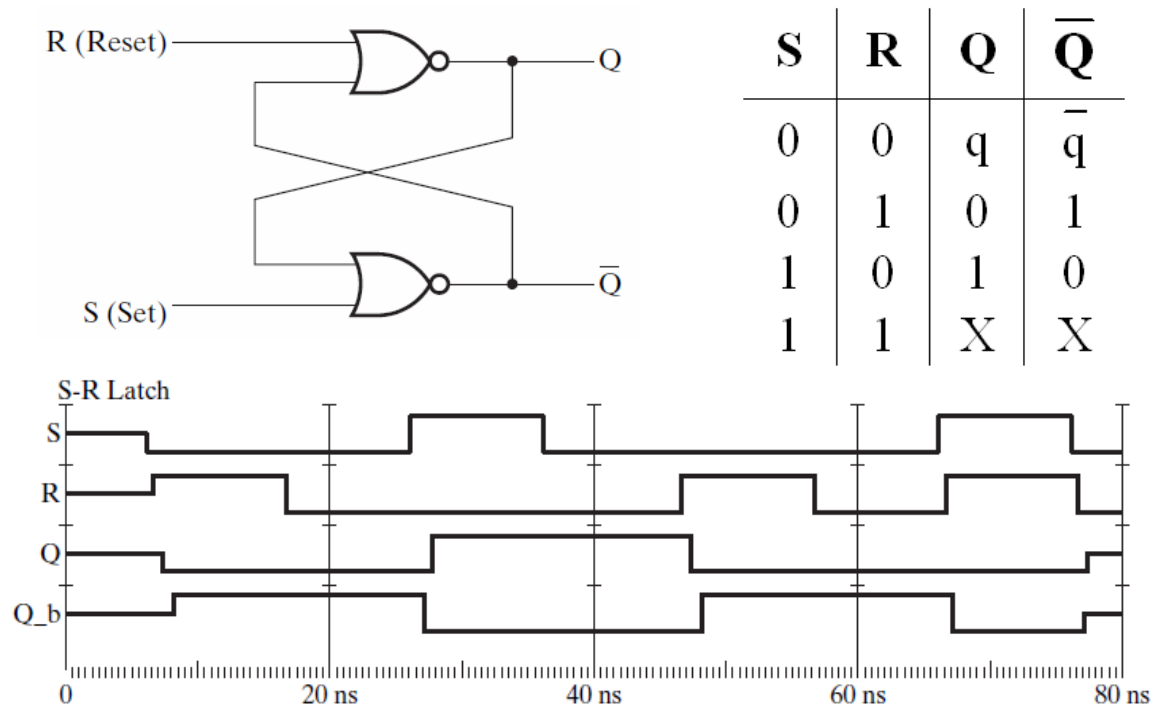
## Tema 6: Sistemas Secuenciales

### Biestables (I)

#### Biestable asíncrono SR

Un elemento de almacenamiento puede mantener un estado binario indefinidamente (mientras no se retire la alimentación del circuito) hasta que una señal de entrada cambie su estado. Los elementos de almacenamiento más básicos son los biestables asíncronos.

Un biestable asíncrono SR es un circuito construido con dos puertas NOR conectadas como se ve en la figura. Su tabla básica de funcionamiento también se muestra, y podemos ver que no está permitido que ambas entradas sean '1' a la vez, ya que la salida sería indefinida. La simulación de su funcionamiento muestra como la salida cambia de estado una vez que termina un tiempo de propagación de las señales a través de las puertas. No se espera a ninguna señal de reloj, por lo que es completamente asíncrono.

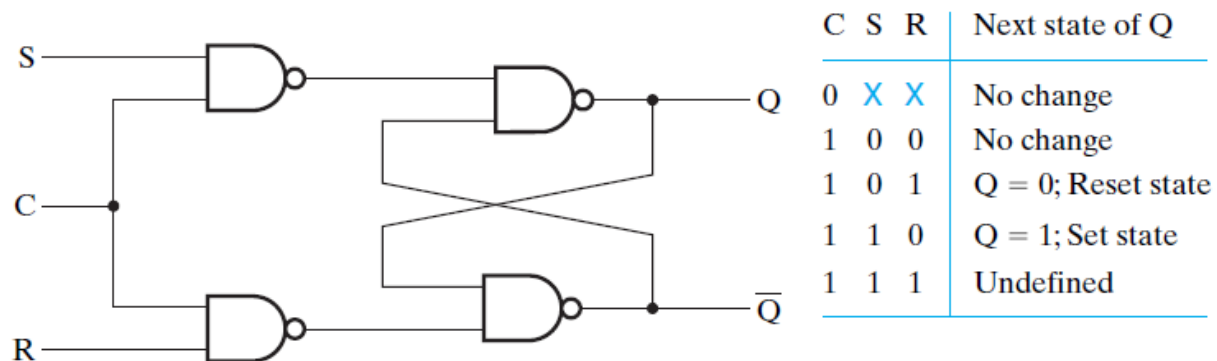


## Tema 6: Sistemas Secuenciales

### Biestables (II)

#### Latch

El funcionamiento de un biestable se puede sincronizar mediante una señal adicional (señal de reloj) que indica cuando pueden las entradas afectar al contenido del elemento de memoria. A continuación se muestra un latch SR activado por nivel y su tabla de funcionamiento.

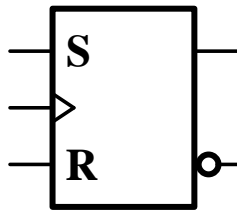


## Tema 6: Sistemas Secuenciales

### Biestables (III)

#### Flip-flop SR

Tiene dos entradas **S** (*set*) y **R** (*reset*), y tiene dos salidas complementarias **Q** ( $q_n$ ) y **Q'**, tiene además una entrada **CLK** (*reloj*) activa por flanco de subida.



Ecuación característica:  
 $Q_{n+1} = S + R'Q_n$

$q_n$ : estado presente  
 $q_{n+1}$ : estado futuro

Modo de Operación	Entradas			Salidas	
	CLK	S	R	$q_{n+1}$	$\overline{q_{n+1}}$
Mantenimiento	↑	0	0	$q_n$	$\overline{q_n}$
Reset	↑	0	1	0	1
Set	↑	1	0	1	0
Prohibido	↑	1	1	1	1
Off	↓	X	X	$q_n$	$\overline{q_n}$

$q_n$	S	R	$q_{n+1}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Indeterminado
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminado

Tabla de funcionamiento básico del biestable SR



## Tema 6: Sistemas Secuenciales

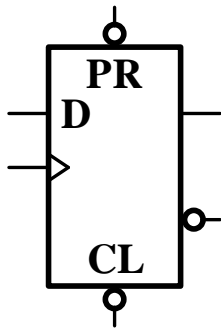
### Biestables (IV)

#### Flip-flop D

Se trata de otro tipo de biestable, esta vez con una entrada **D** (*datos*) y dos salidas de estados complementarias, **Q** y **Q'**, además de una entrada de reloj (**CLK**), activada por flanco (en el ejemplo, flanco de subida). También puede contar con dos entradas más, conocidas por **PR** (de *preset*: reiniciar) y **CLR** (de *clear*: despejar). Estas últimas pueden ser de tipo asíncrono o síncrono (en el ejemplo que se muestra, son asíncronas).

El flip-flop D que aparece en la figura, puede funcionar de dos formas:

- **Síncrona**: Usa una señal de reloj. Si la transición de la señal de reloj es de bajo a alto (o sea, de 0 a 1) se traslada el dato D a la salida, se dice que el biestable ha sido disparado por la señal de reloj. Si por el contrario la transición en el pulso de reloj es de estado alto a bajo (o sea, pasa de 1 a 0) el biestable no responde. En este caso, el último valor permanece almacenado sin cambios.
- **Asíncrona**: Las entradas **PR** y **CLR** son lo que se llaman *entradas asíncronas*, pues independientemente de cómo esté la señal de reloj, reiniciarán (pondrán un 1 en la salida) o despejarán (pondrán un 0 en la salida) el biestable. Éste es el modo de funcionamiento asíncrono.



Modo de Operación	Entradas				Salidas	
	Asíncronas		Síncronas			
	$\overline{\text{PR}}$	$\overline{\text{CLR}}$	CLK	D	Q	Q'
Set asíncrono	0	1	X	X	1	0
Reset asíncrono	1	0	X	X	0	1
Prohibido	0	0	X	X	1	1
Set	1	1	↑	1	1	0
Reset	1	1	↑	0	0	1

Ecuación característica:

$$Q_{n+1} = D$$

$q_n$	D	$q_{n+1}$
0	0	0
0	1	1
1	0	0
1	1	1

Tabla de funcionamiento básico del biestable D

→ Activo por flanco de Subida  
→ Activo por flanco de bajada

## Tema 6: Sistemas Secuenciales

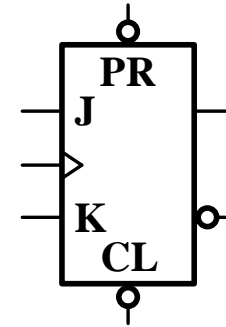
### Biestables (V)

#### Flip-flop JK

El flip-flop JK puede considerarse como el biestable universal. Dispone de tres entradas síncronas: **J** y **K**, para especificar la operación y **CLK**, para disparar el biestable. También puede disponer de dos entradas asíncronas **PR** y **CLR**.

Su ecuación característica es:

$$Q_{n+1} = JQ_n + K'Q_n$$



Modo de Operación	Entradas					Salidas	
	Asíncronas		Síncronas				
	PR	CLR	CLK	J	K	$Q_{n+1}$	$\overline{Q_{n+1}}$
Set asíncrono	0	1	X	X	X	1	0
Reset asíncrono	1	0	X	X	X	0	1
Prohibido	0	0	X	X	X	1	1
Mantenimiento	1	1	↑	0	0	$q_n$	$\overline{q_n}$
Reset	1	1	↑	0	1	0	1
Set	1	1	↑	1	0	1	0
Conmutación	1	1	↑	1	1	$\overline{q_n}$	$q_n$

$q_n$	J	K	$q_{n+1}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

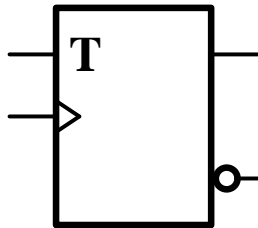
Tabla de funcionamiento básico del biestable JK

## Tema 6: Sistemas Secuenciales

### Biestables (VI)

#### Flip-flop T

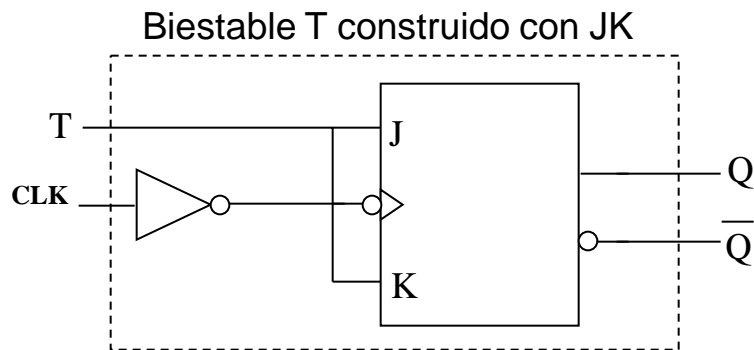
Se trata de un biestable que se comporta como un biestable JK en el que hemos unido las entradas J y K.



Su ecuación característica:

$$Q_{n+1} = TQ_n' + T'Q_n$$

T	$q_{n+1}$
0	$q_n$
1	$q_n'$



$q_n$	T	$q_{n+1}$
0	0	0
0	1	1
1	0	1
1	1	0

Tabla de funcionamiento básico del biestable T

## Tema 6: Sistemas Secuenciales

### Biestables (VII)

#### Tablas de excitación de los biestables

Estas tablas relacionan estado presente y estado siguiente del biestable frente a sus entradas y son muy utilizadas en el proceso de diseño.

$q_n$	$q_{n+1}$	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Biestable SR

$q_n$	$q_{n+1}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Biestable JK

$q_n$	$q_{n+1}$	D
0	0	0
0	1	1
1	0	0
1	1	1

Biestable D

$q_n$	$q_{n+1}$	T
0	0	0
0	1	1
1	0	1
1	1	0

Biestable T

## Tema 6: Sistemas Secuenciales

### Biestables (VIII)

#### Otros tipos de biestables: Master & Slave (Maestro-Esclavo)

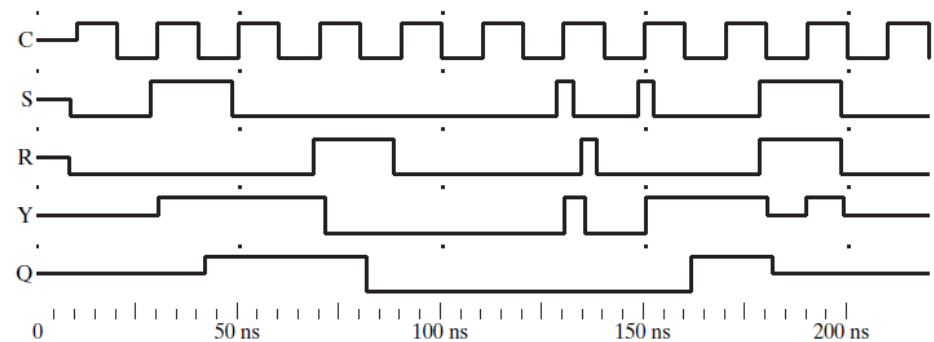
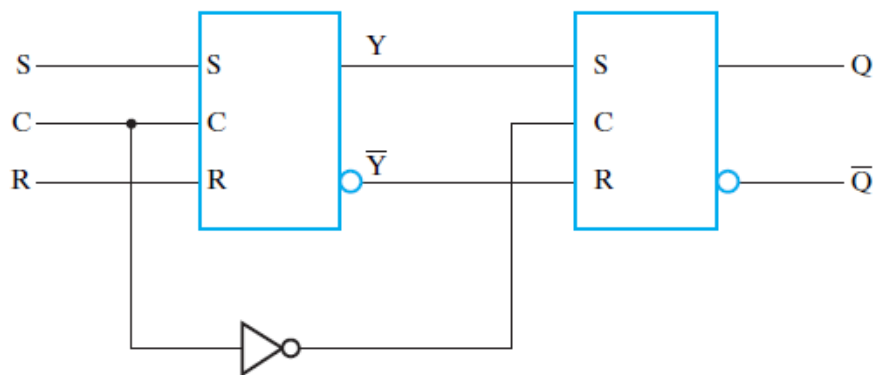
La mayor parte de los sistemas digitales complejos operan como un sistema secuencial síncrono, lo que requiere un reloj maestro que envíe señales a todas las partes del sistema para coordinar la operación del mismo.

Los biestables que hemos visto transfieren la entrada a la salida cuando se lo indica el cambio en la señal de reloj. Ya hemos visto que están disparados por flancos de subida o de bajada. Pero muchos biestables son dispositivos disparados por **pulsos**, denominándose **biestables maestro-esclavo**. Un *biestable maestro-esclavo* está formado por varias puertas y flip-flops conectados de manera que se usa el pulso completo de reloj (tiempo que el reloj está a nivel alto) para transmitir el dato de la entrada a la salida. Aquí se expone un ejemplo realizado con biestables RS.

La señal de reloj controla el *maestro*, se invierte y controla el *esclavo*. Así, cuando  $CLK=1$  (reloj alto) el maestro registra los datos presentes en las entradas S y R, permaneciendo inhibido el *esclavo*, por lo que no hay transferencia de información al mismo.

Con el reloj en nivel bajo ( $CLK=0$ ) el maestro se inhibe, no hay modificaciones en sus salidas, y éstas actúan como entradas al esclavo, transfiriéndose su estado a la salida del mismo.

Es decir, la entrada sólo se transfiere a la salida cuando ha terminado el pulso (como si fuera disparado por un flanco de bajada), pero se pueden detectar los cambios producidos en la entrada mientras que  $CLK=1$ .



## Tema 6: Sistemas Secuenciales

### Biestables (IX)

#### Descripción de los biestables en VHDL (I)

En VHDL, los procesos tienen **memoria implícita**, esto quiere decir que si una señal recibe una asignación condicional dentro de un proceso, y el conjunto de asignaciones no es completo (es decir, existe alguna condición en que la asignación a dicha señal no está especificada), el proceso asigna por defecto la conservación del valor de dicha señal. Es como si al comienzo del proceso existiera la asignación  $\text{señal} \leq \text{señal}$ , referida a cada una de las señales que reciben alguna asignación dentro del proceso.

Ejemplo:

```
process (a, b)
begin
    if a = '1' then p <= b; end if;
end process;
```

En este caso, cuando **a** adopta el valor '1', **p** adopta el valor de **b**, y cuando **a** = 0, como no se ha especificado nada dentro del proceso, **p** conserva el valor que tenía anteriormente.

En los biestables, la salida actúa también como entrada (realimentación) y, habida cuenta que las salidas VHDL (*port out*) no pueden «ser leídas» desde dentro del circuito (es decir, no pueden actuar como entradas de ninguna asignación), **es necesario utilizar para la realimentación una señal interior, del mismo valor que la salida.**

## Tema 6: Sistemas Secuenciales

### Biestables (X)

#### Descripción de los biestables en VHDL (II)

```
.....  
port ( q :out std_logic;  
.....  
architecture nombre_de_la_arquitectura of nombre_de_la_entidad is  
signal q_interna: std_logic;  
begin  
    q <= q_interna;  
.....
```

La necesidad de «duplicar» una señal de salida que se realimenta dentro de un proceso no es necesaria cuando la realimentación se refiere solo a la *memoria implícita* que presenta el proceso; es decir, cuando simplemente conserva el valor de la señal sin efectuar ninguna asignación explícita en la que intervenga dicha señal como entrada. Si *q* recibe asignaciones dentro de un proceso y en ninguna de ellas interviene en la parte derecha de la asignación, no es necesario introducir *q\_interna*.

Veamos a continuación distintas formas de describir un biestable básico SR. Vemos que no es necesario emplear *q\_interna* en los dos últimos ejemplos, aprovechando además la memoria implícita del proceso.

```
q_interna <= '0' when R = '1' else '1' when S = '1' else q_interna;
```

```
-----  
process (R, S)  
begin  
    if R = '1' then q <= '0'; end if;  
    if S = '1' then q <= '1'; end if;  
end process;
```

```
-----  
process (R, S)  
begin  
    if S = '1' then q <= '1'; elsif R = '1' then q <= '0'; end if;  
end process;
```

## Tema 6: Sistemas Secuenciales

### Biestables (XI)

#### Descripción del reloj en circuitos síncronos

La descripción de la señal de reloj **CK** ha de hacerse dentro de un proceso, de las siguientes formas:

-si todo el proceso es síncrono

```
process --sin lista de sensibilidad  
begin  
  wait on CK until CK = '1'; -- flanco ascendente
```

-si hay una parte asíncrona (por ejemplo, un borrado asíncrono con R)

```
process (R, CK)  
begin  
  if R = '1' then .....  
  elsif CK'event and CK = '1' then -- flanco ascendente
```

O también: *elsif rising-edge*(CK) *then* -- flanco ascendente



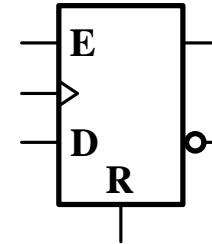
## Tema 6: Sistemas Secuenciales

### Biestables (XII)

#### Ejemplos de descripción de biestables

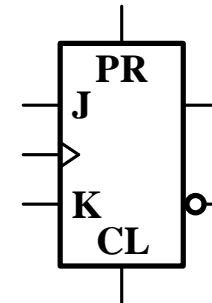
Biestable síncrono D con habilitación (E) y con borrado asíncrono (R)

```
process (R, CK, E, D)
begin
  if R = '1' then q <= '0';
  elsif CK'event and CK = '1' then
    if E = '1' then q <= D; end if;
  end if;
end process;
```



Biestable síncrono JK con marcado (S) y borrado (R) asíncronos

```
process (CL, PR, CK, J, K, q_interna)
begin
  if CL = '1' then q_interna <= '0';
  elsif PR = '1' then q_interna <= '1';
  elsif CK'event and CK = '1' then
    if J = '1' and K = '1' then q_interna <= not q_interna;
    elsif J = '1' then q_interna <= '1';
    elsif K = '1' then q_interna <= '0';
    end if;
  end if;
end process;
```



En este biestable es necesario emplear *q\_interna*, ya que necesitamos realizar una asignación a una señal que interviene también en el lado derecho de la asignación (*q\_interna <= not q\_interna*)

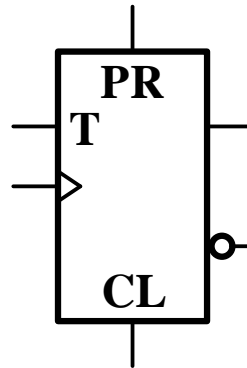
## Tema 6: Sistemas Secuenciales

### Biestables (XIII)

#### Ejemplos de descripción de biestables (II)

Ejercicio 1: Describa un módulo completo en VHDL de un biestable T con las siguientes características:

- Activo al flanco de subida en CK
- Línea de borrado asíncrono (R)
- Línea de puesta a '1' asíncrona (S)



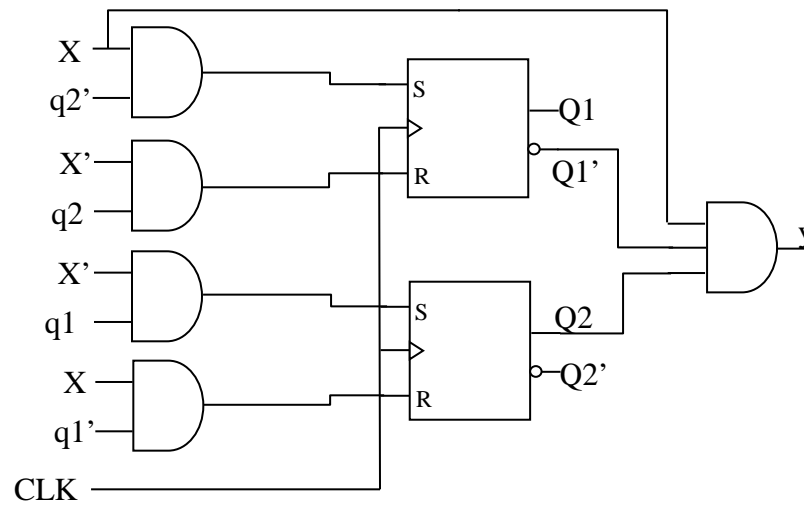
## Tema 6: Sistemas Secuenciales

### Análisis de Sistemas Secuenciales (I)

El análisis consiste en obtener una **tabla de estados** (o tabla de transición) y/o un **diagrama de flujo o diagrama de estados**, de las secuencias de tiempo de las entradas, salidas y estados internos del sistema secuencial. También es posible escribir expresiones booleanas que describan su comportamiento.

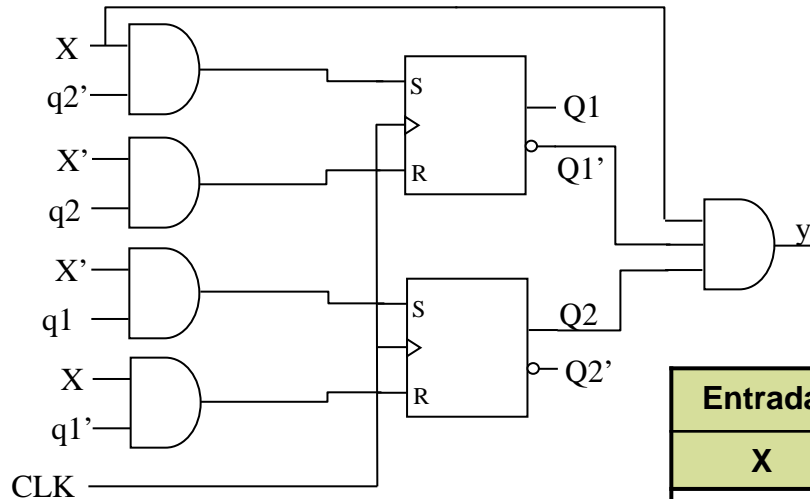
La tabla consta de 4 secciones principales: **entradas**, **estado presente**, **estado siguiente** y **salidas**. En la sección **estado presente** se indican los estados de los biestables antes de la ocurrencia del pulso de reloj bajo las condiciones de entrada indicadas. En la sección **estado siguiente** se muestra el estado de los biestables después del pulso de reloj. La sección de **salidas** muestra los valores de las variables de salida durante el **estado presente**.

Veamos el proceso de análisis mediante el siguiente ejemplo, que es un autómata de Mealy (se observa claramente ya que la única salida del circuito (Y) depende tanto de los estados internos de los biestables como de la entrada (X)).



## Tema 6: Sistemas Secuenciales

### Análisis de Sistemas Secuenciales (II)



#### Ecuaciones:

$$Q1 = S + R'q1 = Xq2' + (X'q2)'q1 = Xq2' + (X + q2')q1 = Xq2' + Xq1 + q2'q1$$

$$Q2 = S + R'q2 = X'q1 + (Xq1')'q2 = X'q1 + (X' + q1)q2 = X'q1 + X'q2 + q1q2$$

$$Y = XQ1'Q2$$

Entrada	Estado presente		Estado futuro		Salida
X	q2	q1	Q2	Q1	Y
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	0	0	1
1	1	1	1	1	0

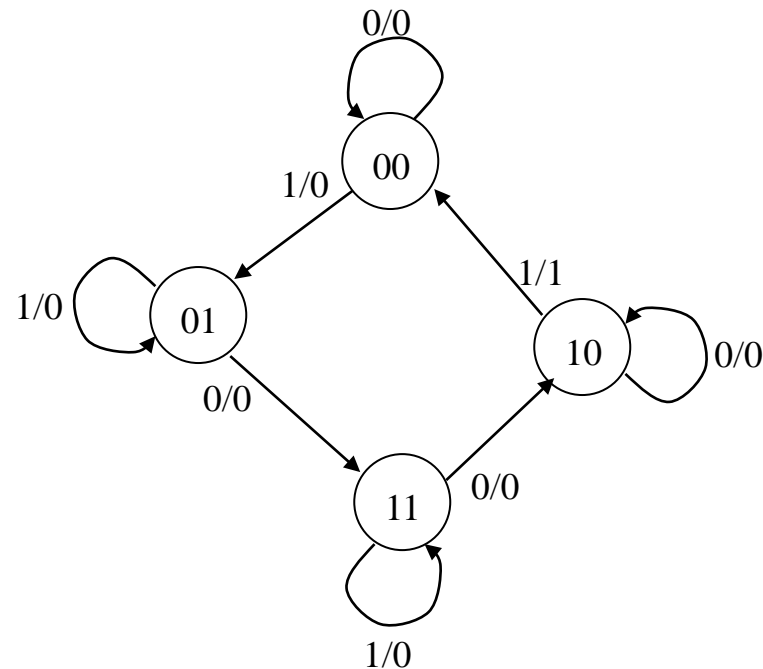
La tabla de estados se interpreta de la siguiente manera: Tomemos como ejemplo el estado presente 01: si el circuito está en este estado presente (dado por q2q1) y recibe una entrada 0 (X), al recibir el flanco de subida del reloj, pasa al estado 11 (Q2Q1) con salida 0 (Y); si estando en 01 lo que recibe es una entrada 1, entonces la tabla nos dice que pasa al estado 01, o sea, se queda en el mismo estado y la salida es 0.

Tabla de estados

## Tema 6: Sistemas Secuenciales

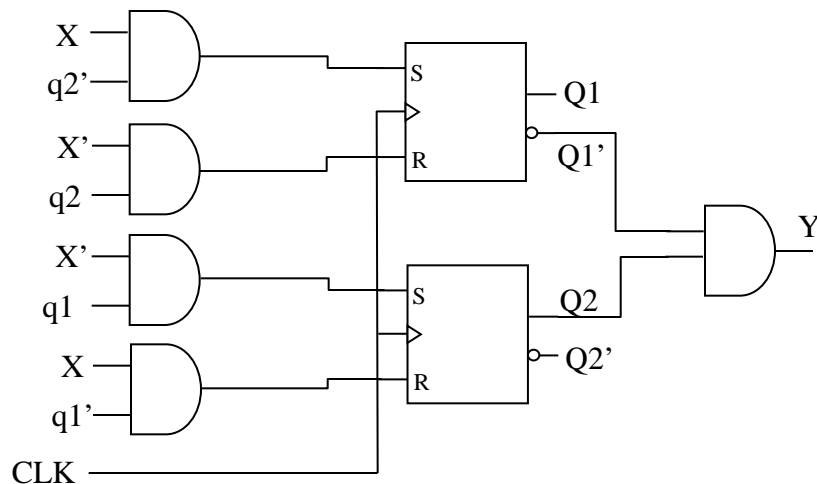
### Análisis de Sistemas Secuenciales (III)

El **diagrama de estados** se puede obtener fácilmente desde la tabla de estados. Cada estado se representa por un círculo y en su interior o bien su nombre o su valor binario. Las transiciones se representan mediante flechas que conectan los estados. Cada flecha muestra qué combinaciones de entrada la afectan y las salidas que provoca. Al ser un autómata de Mealy, las salidas se muestran en las transiciones. Los datos que se indican en cada flecha es ENTRADAS / SALIDAS.



## Tema 6: Sistemas Secuenciales

### Análisis de Sistemas Secuenciales (IV)



Modifiquemos ligeramente el circuito anterior para convertirlo en un autómata de Moore. Ahora la salida sólo depende de los estados internos del circuito. El proceso sería el mismo que antes, aunque las salidas las señalamos en una tabla aparte, junto con los estados posibles del circuito.

#### Ecuaciones:

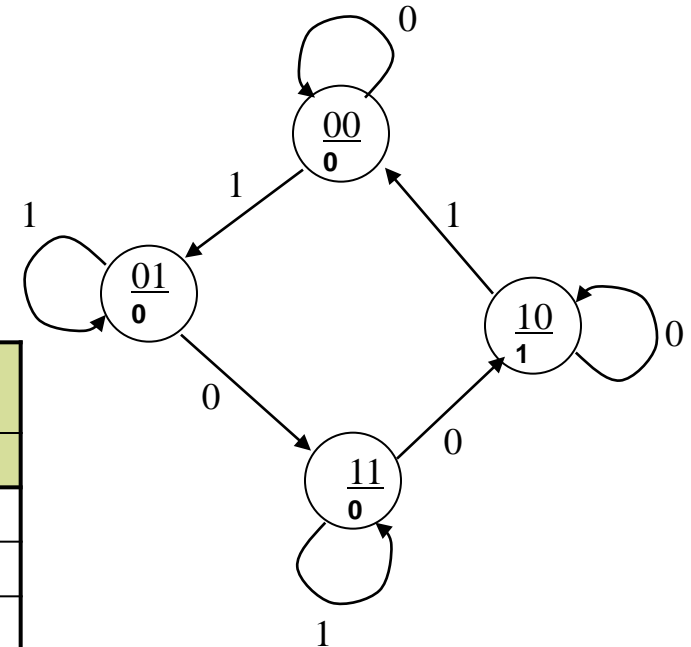
$$Q1 = S + R'q1 = Xq2' + (X'q2)'q1 = Xq2' + (X + q2')q1 = Xq2' + Xq1 + q2'q1$$

$$Q2 = S + R'q2 = X'q1 + (Xq1')'q2 = X'q1 + (X' + q1)q2 = X'q1 + X'q2 + q1q2$$

$$Y = Q1'Q2$$

Entrada	Estado presente		Estado futuro	
	X	q2, q1	Q2, Q1	
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

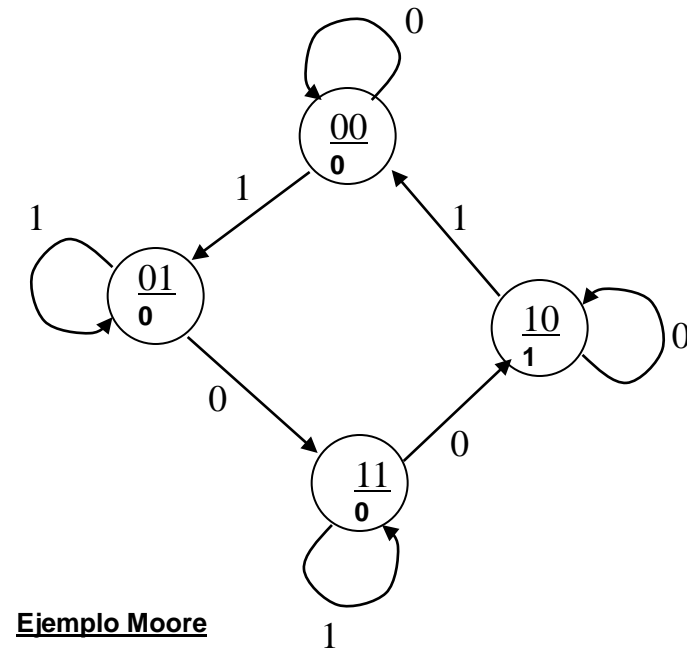
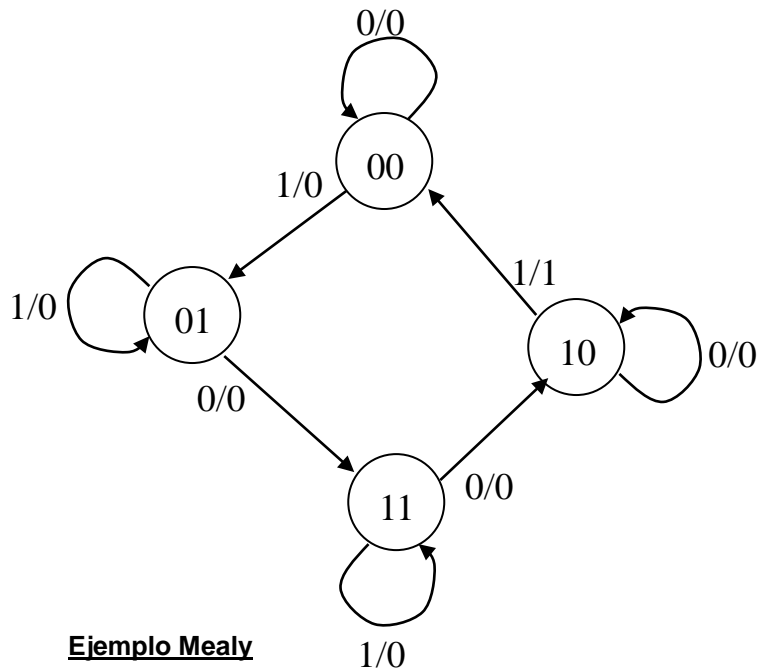
Estado		Salida
q2	q1	Y
0	0	0
0	1	0
1	0	1
1	1	0



## Tema 6: Sistemas Secuenciales

### Consideraciones sobre las salidas en los autómatas de Mealy y Moore

- Las salidas en un autómata de Mealy dependen de las entradas. Esto quiere decir que la salida responde de forma inmediata a un cambio en las entradas, sin esperar al flanco activo de reloj.
- Las salidas en un autómata de Moore no dependen de las entradas. Por tanto, cambian de forma síncrona con el reloj, al cambiar los estados internos.
- Si queremos que las salidas de un autómata de Mealy cambien de forma síncrona con el reloj, tendremos que añadirles biestables, de forma que sólo aparecerán cambios en las salidas al llegar el flanco de reloj. Si hacemos esto, la transición que nos ha hecho cambiar de estado es la que indica la salida que va a aparecer cuando llegue el flanco de reloj.



---

## Tema 6: Sistemas Secuenciales

### Diseño básico de circuitos secuenciales síncronos con biestables (I)

El proceso de diseño de circuitos secuenciales con dispositivos SSI y MSI se compone de los siguientes pasos:

1. Se parte de las especificaciones de funcionamiento, que pueden o no incluir un diagrama de estados.
2. Se obtiene el diagrama de estados desde las especificaciones iniciales. **En cualquier diagrama de estados, es imprescindible que todas las flechas que parten de un estado sean excluyentes entre sí.**
3. Se procede a la reducción del número de estados, es opcional.
4. Se determina el número de FF necesarios y se asigna un símbolo a cada uno.
5. Se asignan valores binarios a cada estado.
6. Se escoge el tipo de FF que va a utilizarse
7. Se obtiene una tabla de estados. Esta tabla incorpora también las entradas de los biestables que vayamos a utilizar.
8. Mediante la tabla de estados se derivan las entradas de los biestables a partir de la tabla de excitación de los biestables que utilicemos.
9. Mediante algún método (Karnaugh, McCluskey ...) se simplifican dichas funciones o se construyen usando dispositivos MSI y SSI combinacionales.
10. Se dibuja el diagrama lógico.

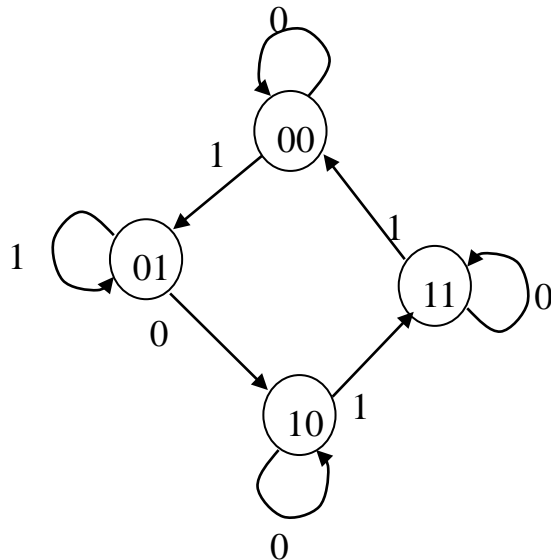


## Tema 6: Sistemas Secuenciales

### Diseño básico de circuitos secuenciales síncronos con biestables (II)

#### Ejemplo (I):

Partimos del siguiente diagrama de flujo. No existen salidas, luego suponemos que las salidas de los biestables son las salidas del circuito.



Como existen dos variables de estado interno (**q2q1**) se necesitan dos elementos de memoria, dos flip-flops. Vamos a realizar el diseño con biestables JK. No es necesario asignar valores binarios a cada estado, ya que éstos se indican en el diagrama de estados. Para cada uno de los biestables necesitamos deducir la entrada J y la entrada K. Dichas funciones se pueden realizar usando puertas básicas o dispositivos MSI.

Entrada	Estado presente		Estado futuro		Entradas Biestables			
X	q2	q1	Q2	Q1	J2	K2	J1	K1
0	0	0	0	0	0	X	0	X
0	0	1	1	0	1	X	X	1
0	1	0	1	0	X	0	0	X
0	1	1	1	1	X	0	X	0
1	0	0	0	1	0	X	1	X
1	0	1	0	1	0	X	X	0
1	1	0	1	1	X	0	1	X
1	1	1	0	0	X	1	X	1

Tabla de estados

## Tema 6: Sistemas Secuenciales

### Diseño básico de circuitos secuenciales síncronos con biestables (III)

Ejemplo (II):

Simplificando se obtiene una expresión booleana de J1, K1, J2 y K2:

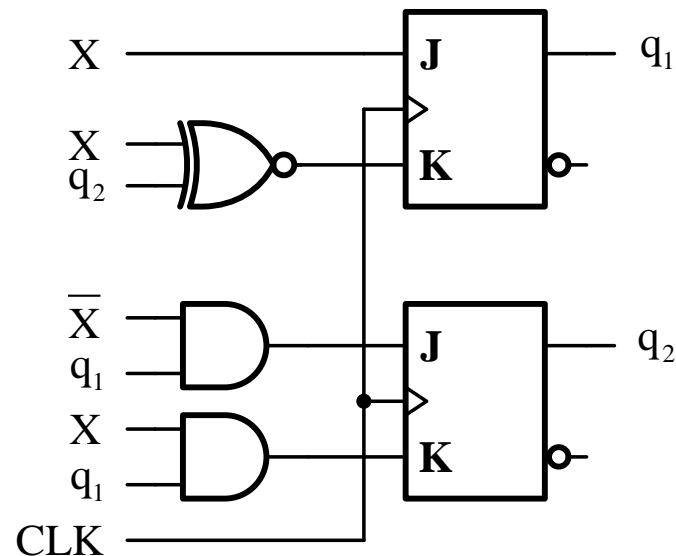
$$J2 = q1X'$$

$$K2 = q1X$$

$$J1 = X$$

$$K1 = q2'X' + q2X$$

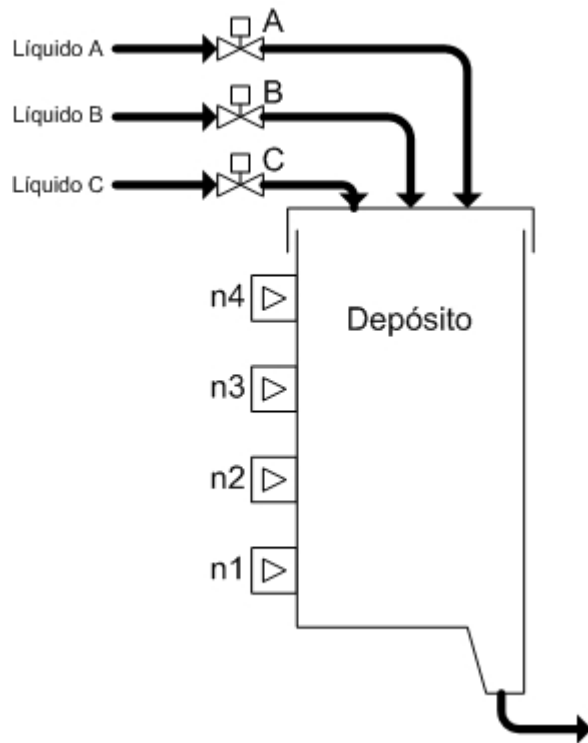
El circuito resultante se muestra a continuación.



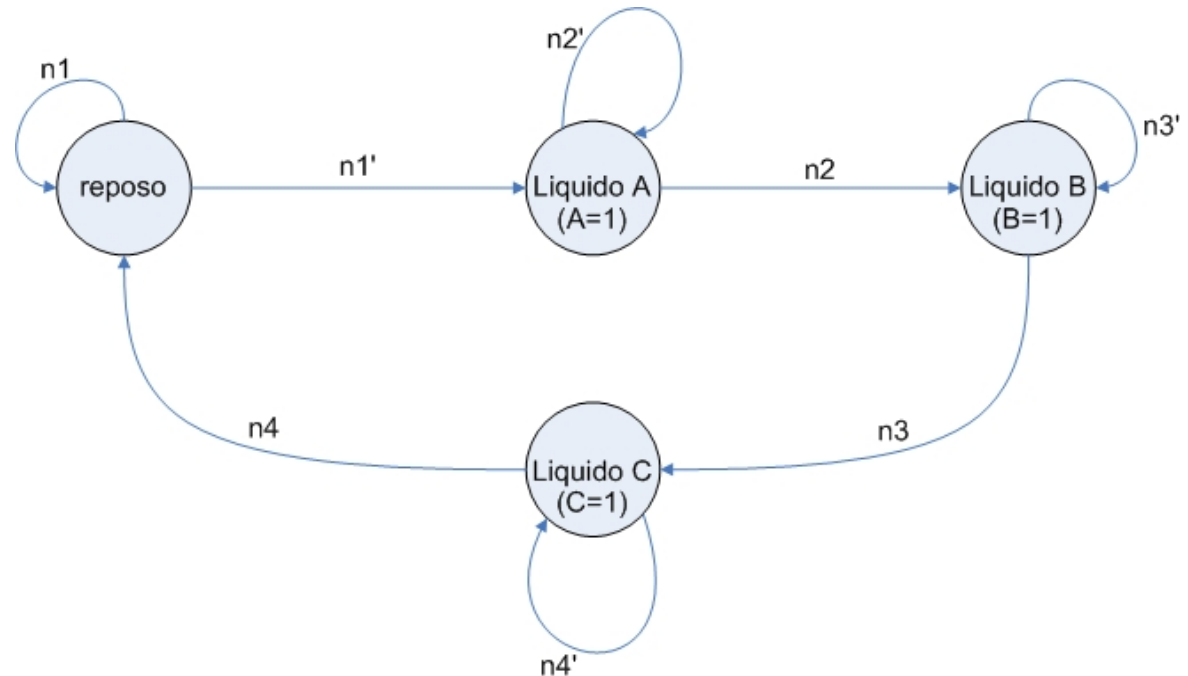
## Tema 6: Sistemas Secuenciales

### Ejemplo de diseño desarrollado con los dos tipos de autómatas (I)

Un depósito se llena con una mezcla de tres líquidos diferentes, para lo cual dispone de tres electroválvulas **A**, **B**, **C** que controlan la salida de dichos líquidos y de cuatro detectores de nivel **n1**, **n2**, **n3**, **n4**, siendo **n1** el inferior y **n4** el de llenado máximo. Solamente cuando el nivel del depósito desciende por debajo del mínimo **n1** se produce un ciclo de llenado: primero con el líquido A hasta el nivel **n2**, luego el líquido B hasta el nivel **n3** y, finalmente, el líquido C hasta completar el depósito (nivel **n4**). Se pide realizar el circuito usando biestables D y elementos combinacionales SSI y MSI.



El diagrama de estados correspondiente al *autómata de Moore* se muestra a continuación. **Obsérvese que todas las flechas que parten de un estado son excluyentes entre sí.**



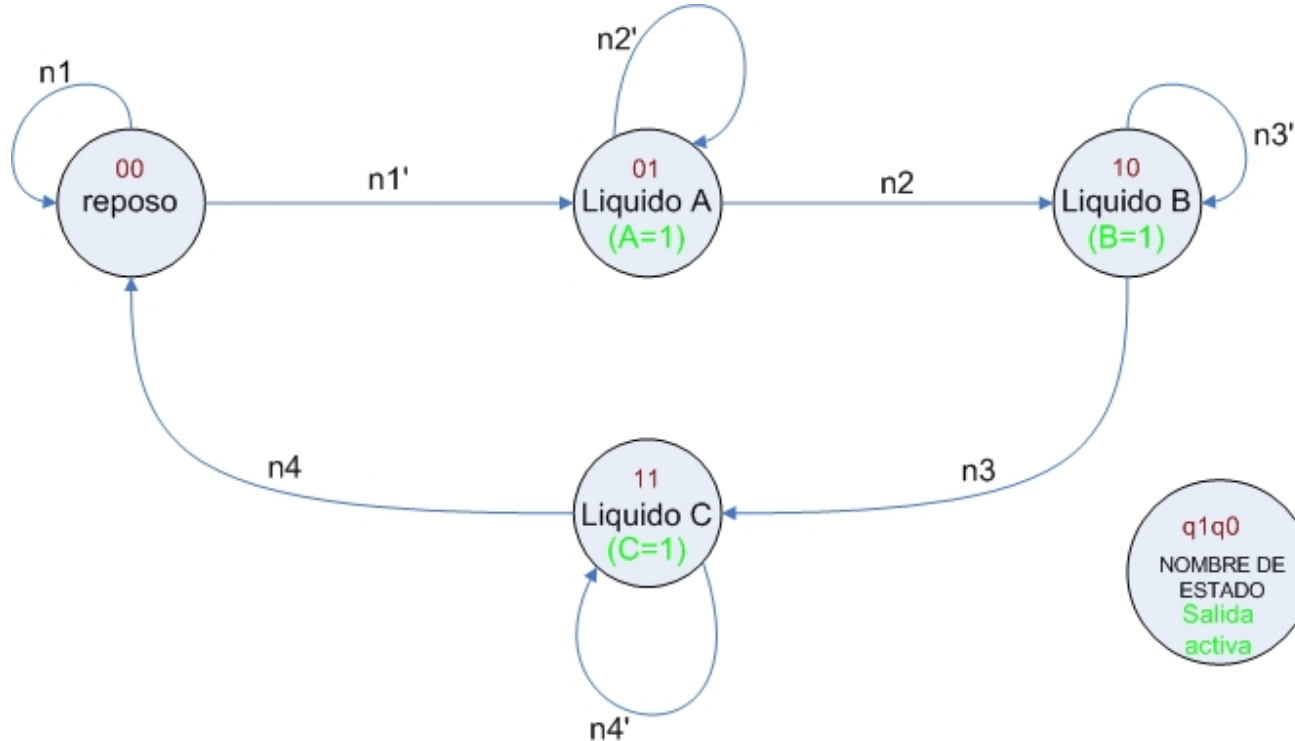
## Tema 6: Sistemas Secuenciales

### Ejemplo de diseño desarrollado con los dos tipos de autómatas (II)

#### Asignación de valores binarios a cada estado

Una vez obtenido el diagrama de estados, el siguiente paso es asignar valores a los estados. Ya se ha comentado que la asignación es libre, siendo la única condición necesaria que cada estado esté representado con una combinación binaria distinta. La asignación dará en cada caso circuitos distintos, pero equivalentes, ya que evolucionarán según el mismo diagrama de estados.

Por ejemplo, podemos asignar los valores 00, 01, 10 y 11 respectivamente a los estados. De esta forma sólo necesitamos dos variables de estado, que denominaremos  $q_1$  y  $q_0$ . Por lo tanto, necesitamos dos biestables, que denominaremos D1 y D0 respectivamente.



ESTADO	ASIGNACIÓN	
	$q_1$	$q_0$
Reposo	0	0
Liquido A	0	1
Liquido B	1	0
Liquido C	1	1



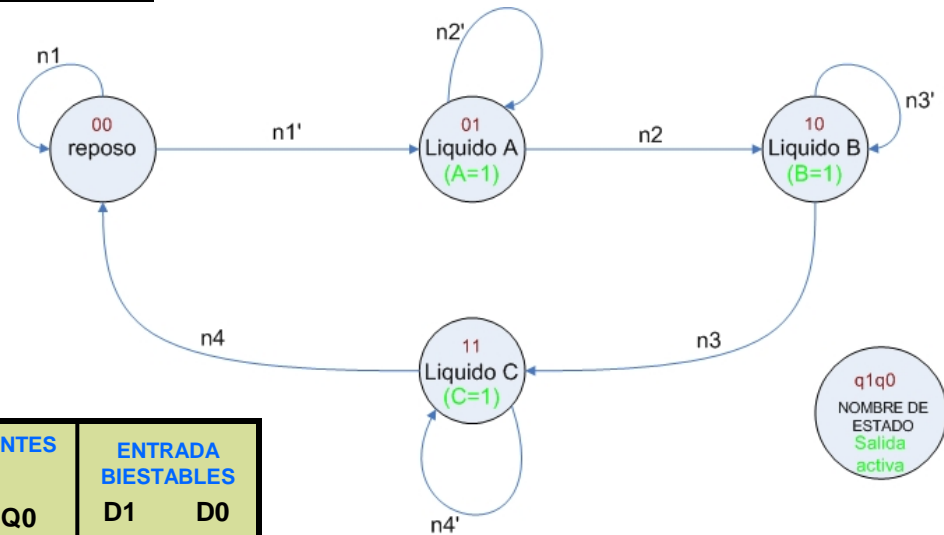
## Tema 6: Sistemas Secuenciales

### Ejemplo de diseño desarrollado con los dos tipos de autómatas (II)

Obtención de la tabla de estados a partir del diagrama de estados

ENTRADAS				E. PRESENTES		E. SIGUIENTES		ENTRADA BIESTABLES	
n1	n2	n3	n4	q1	q0	Q1	Q0	D1	D0
1	X	X	X	0	0	0	0	0	0
0	X	X	X	0	0	0	1	0	1
X	0	X	X	0	1	0	1	0	1
X	1	X	X	0	1	1	0	1	0
X	X	0	X	1	0	1	0	1	0
X	X	1	X	1	0	1	1	1	1
X	X	X	0	1	1	1	1	1	1
X	X	X	1	1	1	0	0	0	0

Tabla de Estados



## Tema 6: Sistemas Secuenciales

### Ejemplo de diseño desarrollado con los dos tipos de autómatas (III)

ESTADO	SALIDAS		
	A	B	C
Reposo (00)	0	0	0
Liquido A (01)	1	0	0
Liquido B (10)	0	1	0
Liquido C (11)	0	0	1

Tabla de salidas

Al ser un autómata de Moore, necesitamos una tabla de estados y una tabla de salidas, que obtendremos fácilmente desde el diagrama de estados.

Las expresiones de D1 y D0 que se obtienen de la tabla de estados son:

$$D1 = n2 \cdot \overline{q1} \cdot q0 + \overline{n3} \cdot q1 \cdot \overline{q0} + n3 \cdot q1 \cdot \overline{q0} + \overline{n4} \cdot q1 \cdot q0$$

$$D0 = \overline{n1} \cdot q1 \cdot q0 + \overline{n2} \cdot \overline{q1} \cdot q0 + n3 \cdot q1 \cdot \overline{q0} + \overline{n4} \cdot q1 \cdot q0$$

Viendo las salidas del sistema, podemos comprobar que un simple decodificador conectado a los biestables nos proporcionará las salidas necesarias:

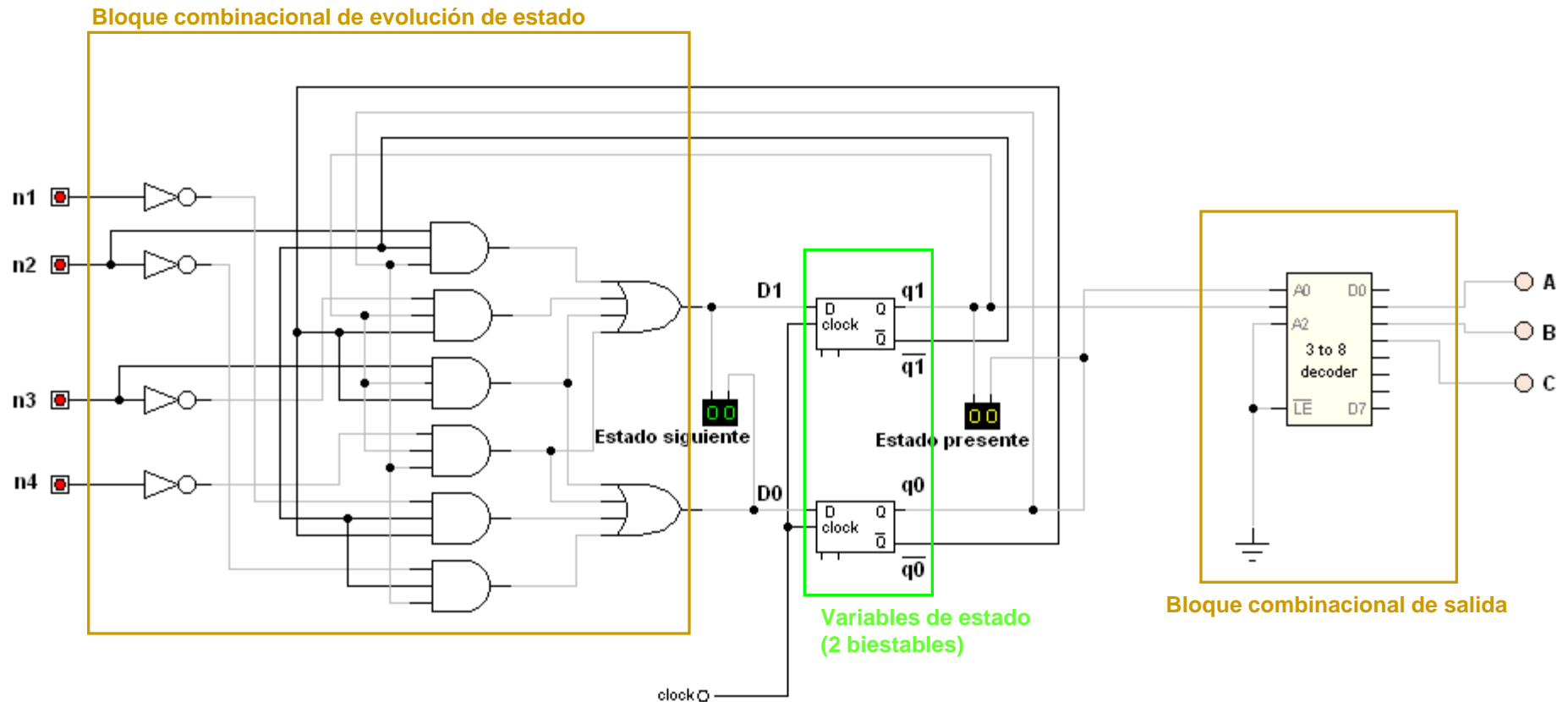
$$A = \sum_2(1); \quad B = \sum_2(2); \quad C = \sum_2(3)$$

También podríamos obtener las salidas mediante un circuito con puertas lógicas diseñado para generar las expresiones de A, B y C obtenidas. Otra opción hubiera sido codificar los estados como 000, 001, 010, 001, de esta forma emplearíamos 3 biestables pero nos ahorraríamos el decodificador, ya que las mismas variables de estado servirían como salidas.

## Tema 6: Sistemas Secuenciales

### Ejemplo de diseño desarrollado con los dos tipos de autómatas (IV)

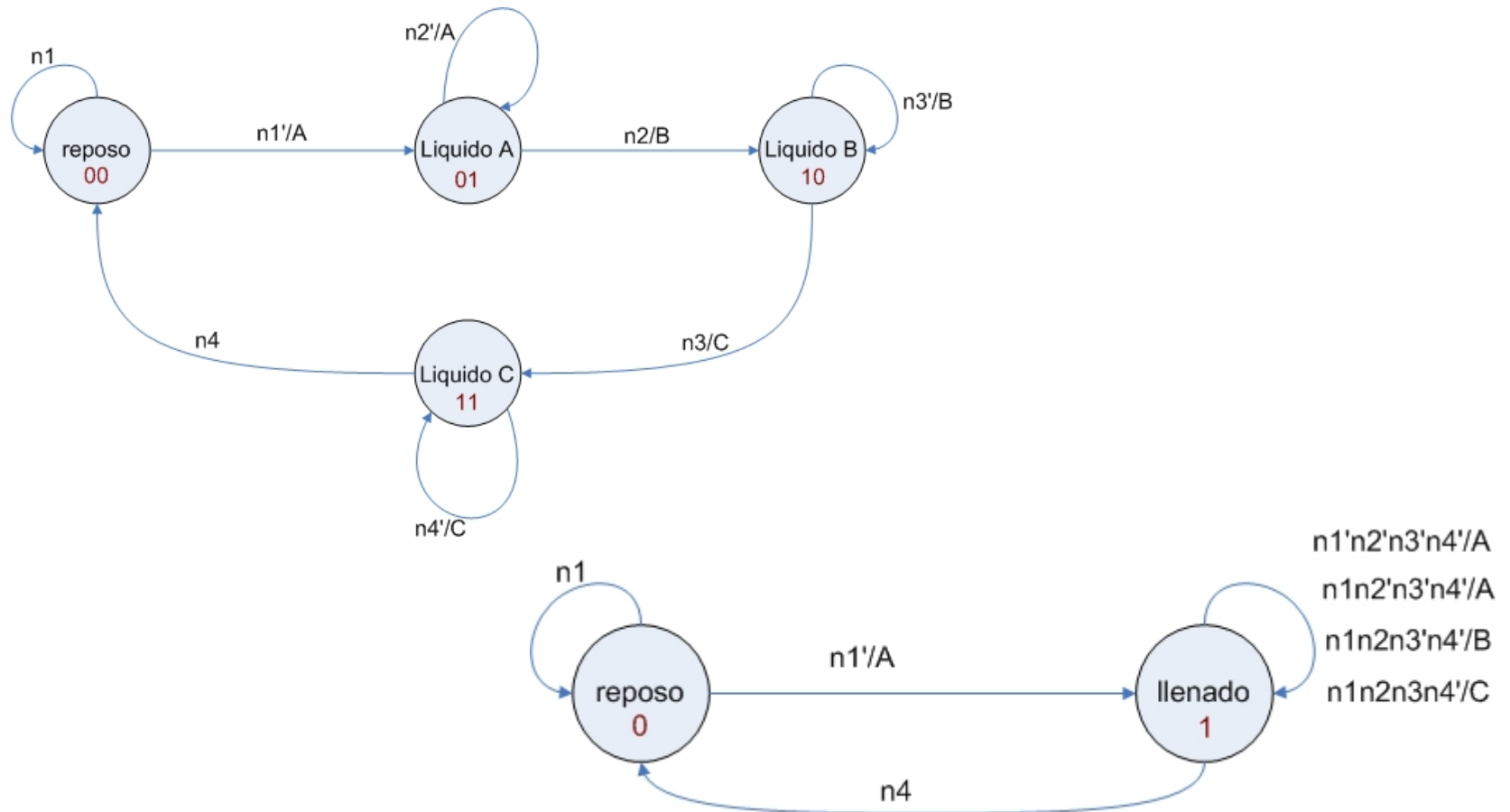
A continuación se muestra el circuito realizado en el software *Digital Works*. Las salidas se han obtenido con un decodificador de 2 a 4 realizado con un decodificador de 3 a 8 líneas. Se muestra la situación de depósito lleno ( $n1 = n2 = n3 = n4 = 1$ ) y sistema en estado de reposo (00).



## Tema 6: Sistemas Secuenciales

### Ejemplo de diseño desarrollado con los dos tipos de autómatas (V)

Veamos ahora la solución del ejemplo del depósito mediante un autómata de Mealy. En este caso, las transiciones entre estados señalan las salidas del sistema. También es posible reducir el diagrama de estados a sólo dos estados, ya que la activación de las salidas puede distinguirse entre sí por los distintos valores de las entradas.



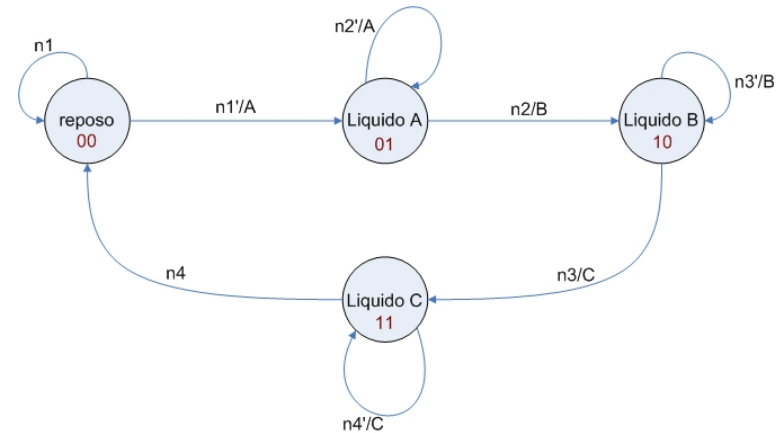


## Tema 6: Sistemas Secuenciales

### Ejemplo de diseño desarrollado con los dos tipos de autómatas (VI)

#### Tabla de estados y ecuaciones lógicas

La tabla de estados en un **autómata de Mealy** incluye también las salidas. Estas salidas dependen del estado presente y del valor actual de las entradas. Esto quiere decir que no esperan al reloj para cambiar, ya que un cambio en las entradas provoca de forma inmediata un cambio en el vector de salida.



$$D1 = n2 \cdot \overline{q1} \cdot q0 + \overline{n3} \cdot q1 \cdot \overline{q0} + n3 \cdot q1 \cdot q0 + \overline{n4} \cdot q1 \cdot q0$$

$$D0 = \overline{n1} \cdot \overline{q1} \cdot \overline{q0} + \overline{n2} \cdot \overline{q1} \cdot q0 + n3 \cdot q1 \cdot \overline{q0} + \overline{n4} \cdot q1 \cdot q0$$

$$A = \overline{n1} \cdot \overline{q1} \cdot \overline{q0} + \overline{n2} \cdot \overline{q1} \cdot q0$$

$$B = n2 \cdot \overline{q1} \cdot q0 + \overline{n3} \cdot q1 \cdot \overline{q0}$$

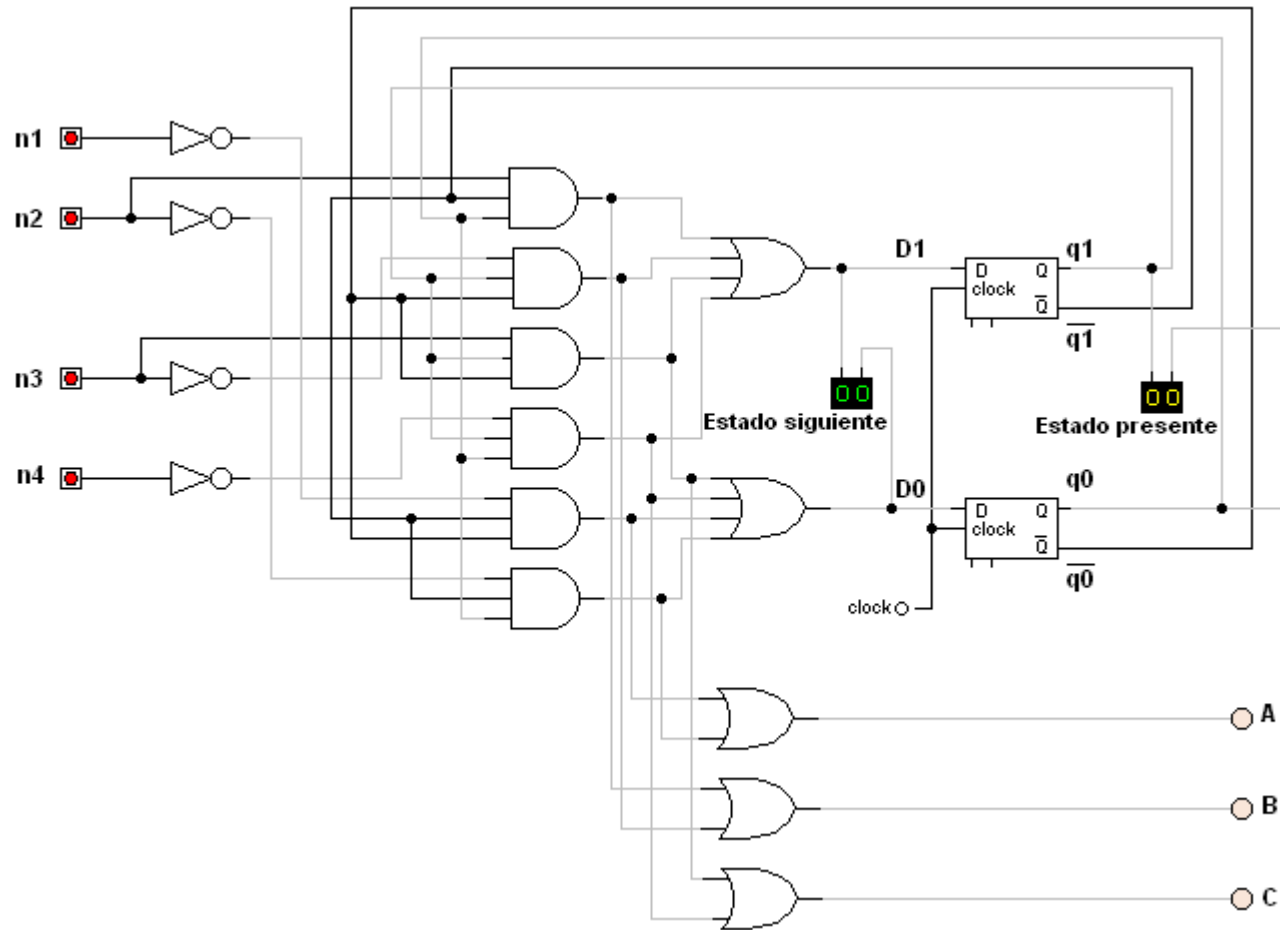
$$C = n3 \cdot q1 \cdot \overline{q0} + n4 \cdot q1 \cdot q0$$

ENTRADAS				E. PRESENTES		E. SIGUIENTES		ENTRADA BIESTABLES		SALIDAS		
n1	n2	n3	n4	q1	q0	Q1	Q0	D1	D0	A	B	C
1	X	X	X	0	0	0	0	0	0	0	0	0
0	X	X	X	0	0	0	1	0	1	1	0	0
X	0	X	X	0	1	0	1	0	1	1	0	0
X	1	X	X	0	1	1	0	1	0	0	1	0
X	X	0	X	1	0	1	0	1	0	0	1	0
X	X	1	X	1	0	1	1	1	1	0	0	1
X	X	X	0	1	1	1	1	1	1	0	0	1
X	X	X	1	1	1	0	0	0	0	0	0	0

## Tema 6: Sistemas Secuenciales

### Ejemplo de diseño desarrollado con los dos tipos de autómatas (VI)

#### Diagrama lógico del autómata de Mealy



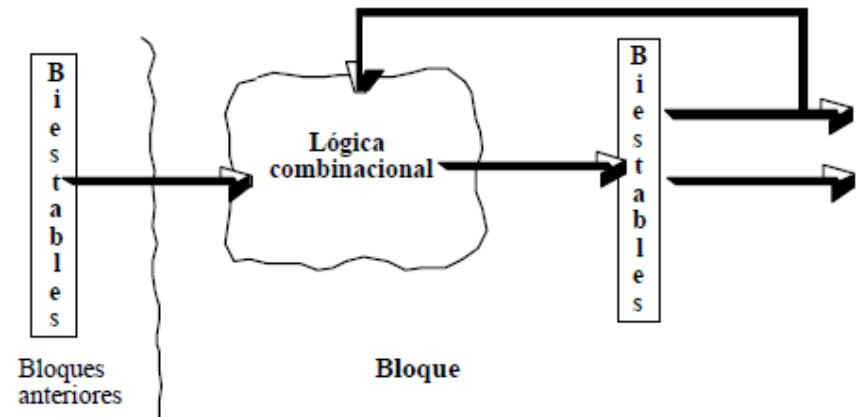
## Tema 6: Sistemas Secuenciales

### Consideraciones sobre la señal de reloj (I)

El sincronismo de las entradas es necesario para asegurar la estabilidad de sus valores a lo largo de cada unidad de tiempo y, con ello, garantizar la preparación correcta del nuevo estado; tal sincronismo se consigue conectando cada entrada a un biestable **D** síncrono, que «capture» los valores de la entrada con el flanco activo de la señal de reloj. En el caso de autómatas de Mealy, el sincronismo de las entradas evita, además, transitorios intermedios en las salidas.

También puede resultar conveniente sincronizar las variables de salida en los autómatas de Mealy, pues ello evita pequeños pulsos transitorios o «fisuras» (*glitches*) que pueden producirse al inicio de las unidades de tiempo, al efectuarse el cálculo de las salidas con el nuevo estado. Ahora bien, la sincronización de las salidas (a través de un biestable **D** síncrono para cada una de ellas) supone retrasarlas una unidad de tiempo de reloj, lo cual, en ocasiones, puede no interesar .

Un sistema síncrono se compone de biestables que almacenan las variables (variables de estado, de entrada y, si el sincronismo es total, también las de salida) y de partes combinacionales que conectan los biestables entre sí. Cada «parte combinacional» se encuentra emparedada» (a modo de *sándwich*) entre dos registros o conjuntos de biestables, uno de ellos correspondiente al bloque o bloques anteriores (entradas) y el otro, el propio del bloque (salidas).



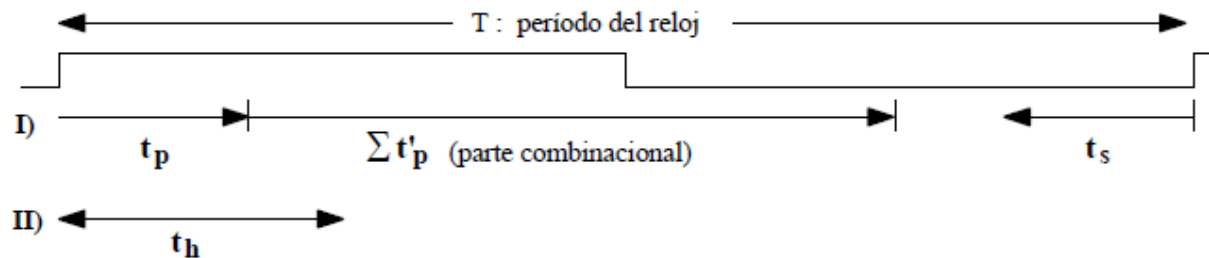
## Tema 6: Sistemas Secuenciales

### Consideraciones sobre la señal de reloj (II)

El cambio de estado de un biestable síncrono se produce coincidiendo con el flanco activo de la onda de reloj; en el caso de un biestable síncrono tipo **D**, en ese momento (flanco activo del reloj), la salida efectúa una copia del valor presente en la entrada. Obviamente existe un pequeño retraso entre el flanco activo de la señal de reloj y la consolidación del correspondiente estado en la salida: tiempo de propagación del dato  $t_p$ . Además, para asegurar el correcto funcionamiento del biestable **D**, cuando llega el flanco activo del reloj es necesario que el valor correcto del dato se encuentre presente en la entrada **D** con una cierta anticipación a dicho flanco (*setup*:  $t_s$ ) y que tal valor se mantenga durante un cierto intervalo posterior (*hold*:  $t_h$ ).

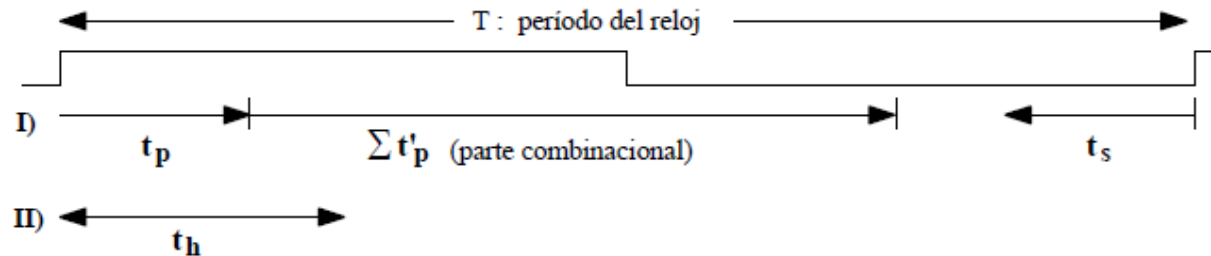
Un camino va desde un biestable **FF1** a otro biestable **FF2** atravesando solamente la parte combinacional del bloque al que corresponde el segundo de ellos **FF2**; obviamente, pueden existir caminos que conectan la salida de un biestable **FF2** con su entrada, caso de darse realimentación de dicho biestable sobre sí mismo.

El sincronismo permite «particionar» (dividir en partes) el cálculo de tiempos, aplicándolo a *caminos* definidos entre biestables, y calcular sobre ellos la velocidad máxima de reloj y las posibles violaciones de permanencia.



## Tema 6: Sistemas Secuenciales

### Consideraciones sobre la señal de reloj (III)



El tiempo de propagación de la señal por un «camino» corresponde a la suma de tiempos de propagación del primer biestable y de la parte combinacional que atraviesa el camino, más el tiempo de anticipación que requiere el segundo biestable:

$$t_p (\text{FF1}) + \sum t'_p (\text{parte combinacional}) + t_s (\text{FF2})$$

El tiempo de anticipación (*set-up*) del biestable receptor interviene en esta suma: es un tiempo adicional a los de propagación, necesario para su funcionamiento correcto. La duración de una unidad de tiempo  $T = 1/f_{CK}$  ha de ser suficiente para que todos los caminos completen la propagación de las señales a través de ellos, es decir:

$$T_{CK} > t_p (\text{FF1}) + \sum t'_p (\text{parte combinacional}) + t_s (\text{FF2})$$

desigualdad que ha de cumplirse para todos los caminos existentes en el sistema digital. Caso de no respetarse esta desigualdad se produce una violación de la anticipación necesaria para el funcionamiento correcto del biestable **FF2**: «violación de *set-up*»; es decir, el nuevo dato a la entrada de dicho biestable no ha completado su preparación con el tiempo de anticipación suficiente (previo al flanco activo del reloj) para asegurar que será capturado correctamente.

Los tiempos de propagación de diversos caminos serán diferentes, ya que lo es la «parte combinacional» que atraviesa cada camino. La frecuencia máxima de trabajo del sistema digital vendrá limitada por aquellos caminos cuyos tiempos de propagación sean mayores. Estos se denominan *caminos críticos*.

## Tema 6: Sistemas Secuenciales

### Consideraciones sobre la señal de reloj (IV)

El correcto funcionamiento de los biestables requiere, además de respetar su tiempo de anticipación, el mantenimiento del dato durante un tiempo de permanencia (*hold*)  $t_h$  posterior al flanco activo de reloj. El tiempo que tarda un dato en cambiar, en la entrada de un biestable, será el debido al retraso que sufre para recorrer el correspondiente camino, o sea, la suma de tiempos de propagación del biestable anterior y de la parte combinacional que atraviesa el camino:

$$t_p(\text{FF1}) + \sum t'_p(\text{parte combinacional})$$

Para garantizar que el dato se mantiene estable durante un intervalo no inferior al tiempo de permanencia  $t_h$  ha de cumplirse que:

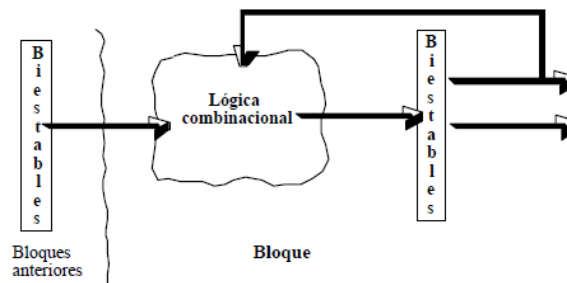
$$t_p(\text{FF1}) + \sum t'_p(\text{parte combinacional}) > t_h(\text{FF2})$$

Caso de que esta desigualdad no se respete, se produce una violación de la permanencia del dato necesaria para asegurar el correcto funcionamiento del biestable **FF2**: «violación de *hold*»; es decir, el dato no se mantiene en la entrada de dicho biestable suficiente tiempo (después del flanco del reloj) para asegurar que será capturado correctamente.

$$\begin{aligned} T_{CK} &> t_{p,m\acute{a}x}(\text{FF1}) + \sum t'_{p,m\acute{a}x}(\text{parte combinacional}) + t_{s,m\acute{a}x}(\text{FF2}) \\ t_h(\text{FF2}) &< t_{p,m\acute{i}n}(\text{FF1}) + \sum t'_{p,m\acute{i}n}(\text{parte combinacional}) \end{aligned}$$

Por tanto, la frecuencia del reloj debe cumplir la siguiente desigualdad:

$$f_{CK} < 1/t_{pc} \text{ siendo } t_{pc} = t_{p,m\acute{a}x}(\text{FF1}) + \sum t'_{p,m\acute{a}x}(\text{combinacional}) + t_{s,m\acute{a}x}(\text{FF2})$$



## Tema 6: Sistemas Secuenciales

### Descripción de diagramas de estados en VHDL (I)

#### Descripción de estados en VHDL

Para describir los estados de un autómata en VHDL se da nombre y número binario a los estados mediante la definición de un tipo (en el ejemplo que se muestra este tipo se denomina *mis\_estados*) y la enumeración de los estados y asignación de valores a ellos, a través de su declaración como constantes. Esto se señala dentro de **architecture**, antes del **begin**:

```
subtype mis_estados is std_logic_vector (1 downto 0);  
constant reposo    : mis_estados := "00";  
constant liquido_A : mis_estados := "01";  
constant liquido_B : mis_estados := "10";  
constant liquido_C : mis_estados := "11";  
signal estado      : mis_estados;
```

Otra forma que conduce exactamente a la misma declaración de estados y asignación de valores, es la siguiente:

```
type mis_estados is (reposo, liquido_A, liquido_B, liquido_C);  
attribute enum_encoding: string;  
attribute enum_encoding of mis_estados: type is "00 01 10 11";  
signal estado: mis_estados;
```

También puede hacerse una declaración de estados sin asignar valores, permitiendo que el compilador efectúe esta asignación de forma automática. Generalmente, si no elegimos otra opción del compilador, los numera en binario natural, aunque esto depende del compilador):

```
type mis_estados is (reposo, liquido_A, liquido_B, liquido_C);  
signal estado: mis_estados;
```

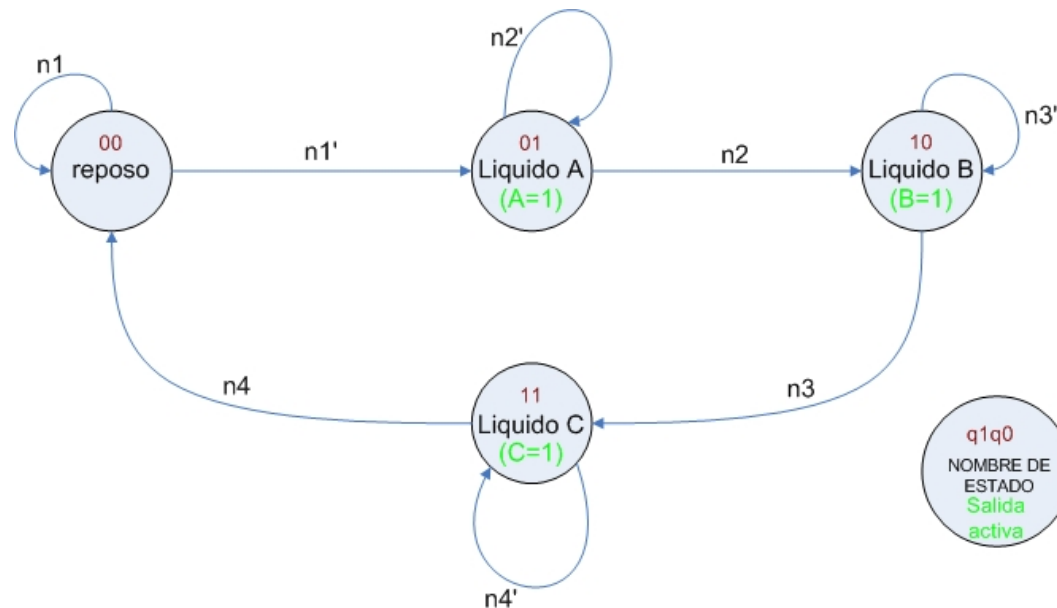
## Tema 6: Sistemas Secuenciales

### Descripción de diagramas de estados en VHDL (II)

La evolución del estado de un sistema secuencial se describe muy bien con la asignación múltiple **case** para referirse a cada uno de los estados y, dentro de ella, utilizando adecuadamente la asignación condicional **if** para las transiciones entre estados. Existen diversas posibilidades para asignar nombres y códigos binarios a los estados; asimismo, puede encomendarse al compilador la tarea de codificar los estados libremente.

### Descripción VHDL del ejemplo del depósito (versión autómeta de Moore)

Vamos a describir el circuito de control del depósito mediante VHDL. Repetimos por conveniencia el diagrama de estados desde el que partiremos. Inicialmente usaremos la versión de autómeta de Moore. Colocaremos una línea síncrona de RESET (RS) para inicializar el circuito.





## Tema 6: Sistemas Secuenciales

### Descripción de diagramas de estados en VHDL (III)

#### Ejemplo del depósito (Moore)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DEPOSITO is
    port ( CK, RS, n1, n2, n3, n4 : in std_logic;
          A, B, C : out std_logic );
end DEPOSITO;

architecture MOORE of DEPOSITO is
    subtype mis_estados is std_logic_vector(1 downto 0);
    constant reposo      : mis_estados := "00";
    constant liquido_A   : mis_estados := "01";
    constant liquido_B   : mis_estados := "10";
    constant liquido_C   : mis_estados := "11";
    signal estado : mis_estados;
begin
    -- evolución del estado. Cada proceso puede llevar un nombre identificativo
    EVOLUCION: process
    begin
        wait until CK = '1';
        if ( RS = '1' ) then estado <= reposo;
        else case estado is
            when reposo      => if (n1 = '0') then estado <= liquido_A; end if;
            when liquido_A  => if (n2 = '1') then estado <= liquido_B; end if;
            when liquido_B  => if (n3 = '1') then estado <= liquido_C; end if;
            when liquido_C  => if (n4 = '1') then estado <= reposo; end if;
            when others =>
                end case;
        end if;
    end process;
    -- funciones de activación de las salidas:
    A <= '1' when estado = liquido_A else '0';
    B <= '1' when estado = liquido_B else '0';
    C <= '1' when estado = liquido_C else '0';
end MOORE;
```

## Tema 6: Sistemas Secuenciales

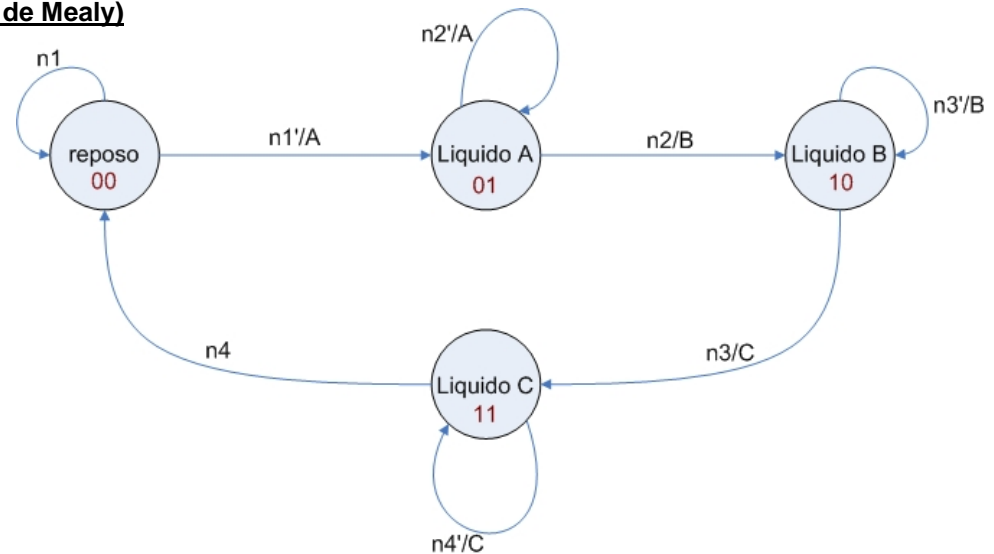
### Descripción de diagramas de estados en VHDL (IV)

**Ejemplo del depósito (Moore)** Otra forma de definir las salidas en el autómata de Moore (depósito):

```
-- activación de las salidas:
SALIDAS: process (estado)
begin
    A <= '0'; B <= '0'; C <= '0'; -- valores por defecto
    case estado is
        when reposo =>
        when liquido_A => A <= '1';
        when liquido_B => B <= '1';
        when liquido_C => C <= '1';
        when others =>
    end case;
end process;
```

### Descripción VHDL del ejemplo del depósito (versión autómata de Mealy)

Vamos ahora a describir el autómata de Mealy. Emplearemos la versión de cuatro estados. Su diagrama de estados se repite abajo. En los autómatas de Mealy, las salidas dependen tanto de los estados internos como de las entradas, y esto debe quedar reflejado en el código VHDL.



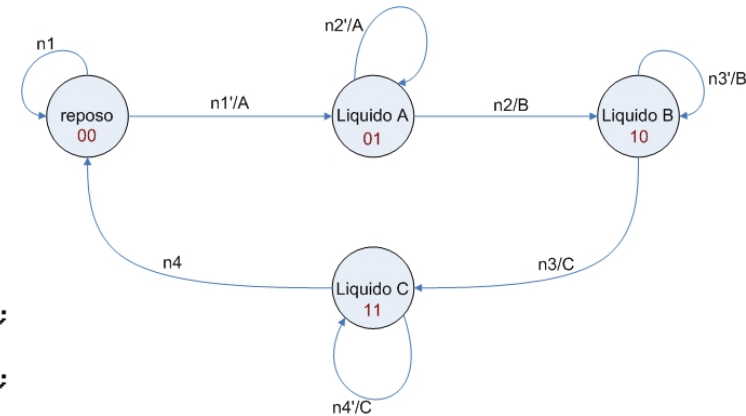
# Tema 6: Sistemas Secuenciales

## Descripción de diagramas de estados en VHDL (VII)

### Ejemplo del depósito (Mealy). Declaración de entidad y proceso de evolución de estados

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DEPOSITO is
    port ( CK, RS, n1, n2, n3, n4 : in std_logic;
          A, B, C : out std_logic );
end DEPOSITO;
architecture MEALY of DEPOSITO is
    type mis_estados is (reposo, liquido_A, liquido_B, liquido_C);
    attribute enum_encoding: string;
    attribute enum_encoding of mis_estados: type is "00 01 10 11";
    signal estado: mis_estados;
begin
    -- evolución del estado. Cada proceso puede llevar un nombre identificativo
    EVOLUCION: process
    begin
        wait until CK = '1';
        if ( RS = '1' ) then estado <= reposo;
        else case estado is
            when reposo => if (n1 = '0') then estado <= liquido_A; end if;
            when liquido_A => if (n2 = '1') then estado <= liquido_B; end if;
            when liquido_B => if (n3 = '1') then estado <= liquido_C; end if;
            when liquido_C => if (n4 = '1') then estado <= reposo; end if;
            when others =>
                end case;
        end if;
    end process;
end process;
```

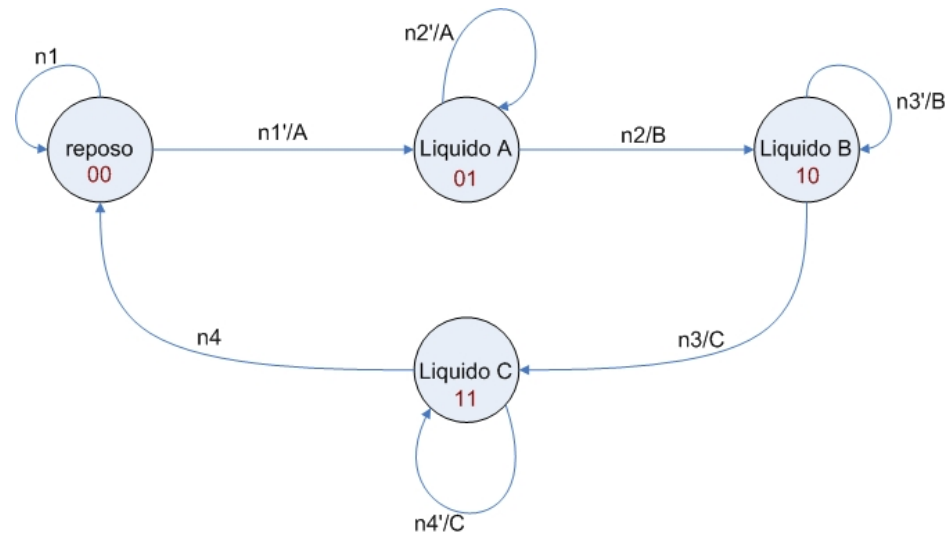


## Tema 6: Sistemas Secuenciales

### Descripción de diagramas de estados en VHDL (VIII)

#### Ejemplo del depósito (Mealy). Proceso de evolución de salidas

```
-- funciones de activación de las salidas:
SALIDAS: process (estado, n1, n2, n3, n4)
begin
    A <= '0'; B <= '0'; C <= '0'; -- por defecto
    case estado is
        when reposo => if (n1 = '0') then A <='1'; end if;
        when liquido_A => if (n2 = '0') then A <='1';
                        else B <= '1'; end if;
        when liquido_B => if (n3 = '0') then B <='1';
                        else C <= '1'; end if;
        when liquido_C => if (n4 = '0') then C <='1'; end if;
    end case; end process; end MEALY;
```



## Tema 6: Sistemas Secuenciales

### Descripción de diagramas de estados en VHDL (VI)

Como puede observarse en los ejemplos anteriores, un sistema secuencial definido por su diagrama de estados puede ser descrito mediante dos procesos: uno de ellos para la evolución del estado y el otro para la activación de las salidas; en cada uno de ellos un «case» recorrerá todos los estados:

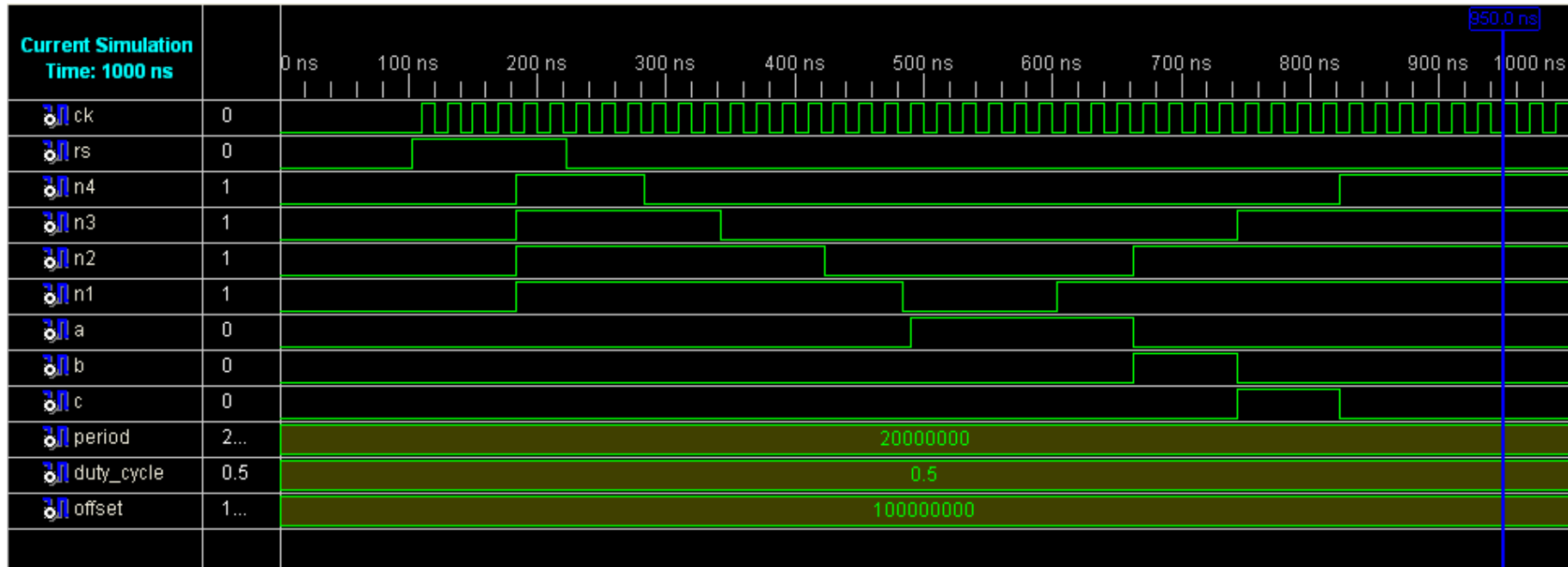
- El primer proceso (evolución del estado) asignará a cada estado las transiciones que deben producirse desde el mismo, a través de sus correspondientes condiciones (*if*) referidas a las variables de entrada.
- El segundo proceso (activación de las salidas) incluirá, para cada estado, las variables de salida que se activan en el mismo y, en el caso de autómatas de Mealy, las condiciones (*if*) de las variables de entrada que influyen en tal activación. La lista de sensibilidades en este segundo proceso será el estado actual en autómatas de Moore y el estado actual y las entradas en el autómata de Mealy.

Es importante recordar la propiedad de memoria implícita de los procesos en VHDL. Si no se especifica el valor que debe tomar una señal, el proceso asigna por defecto la conservación del valor de la señal.

## Tema 6: Sistemas Secuenciales

### Descripción de diagramas de estados en VHDL (VIII)

#### Ejemplo del depósito (Mealy). Simulación funcional (I)

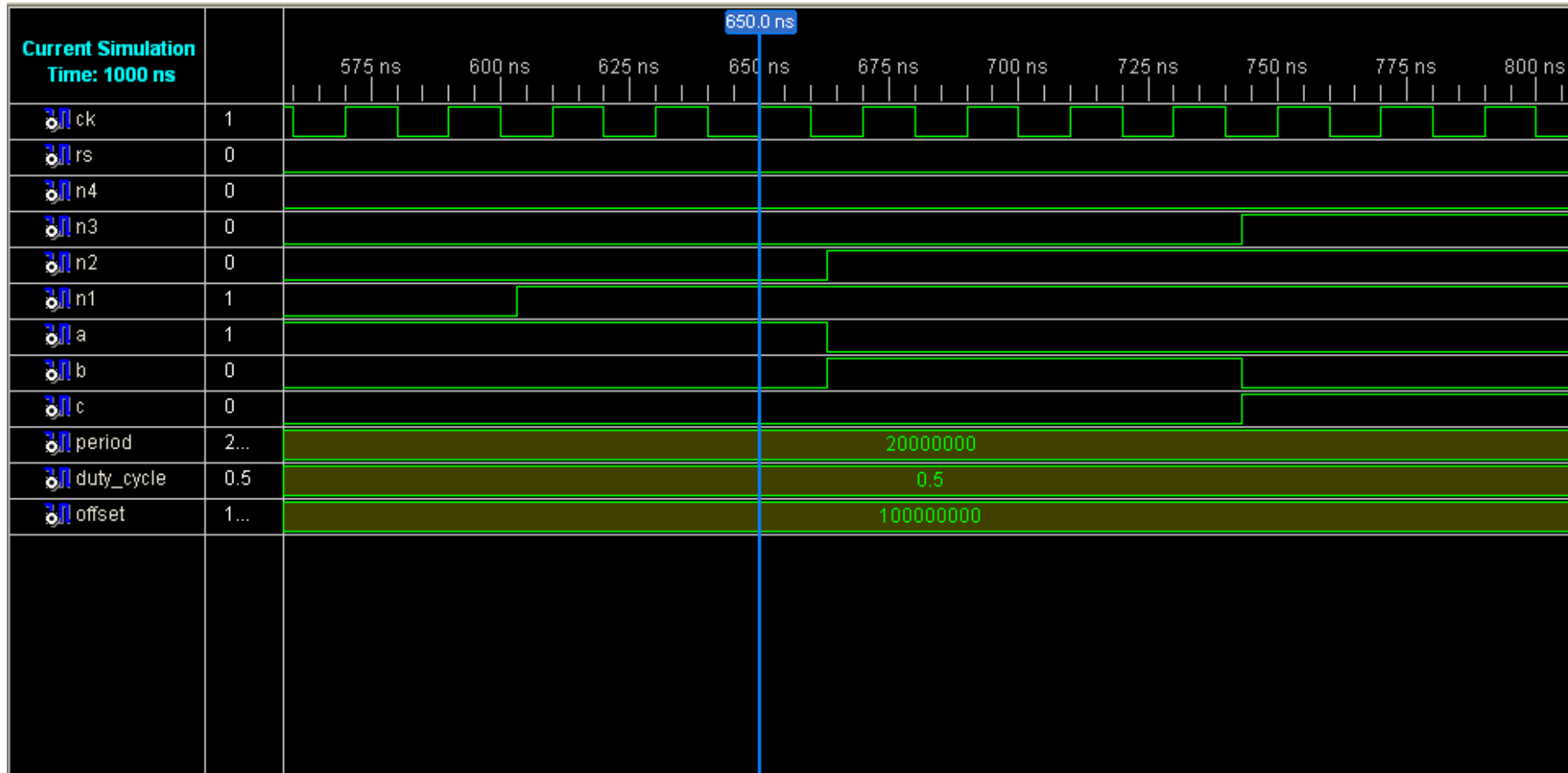


## Tema 6: Sistemas Secuenciales

### Descripción de diagramas de estados en VHDL (VIII)

#### Ejemplo del depósito (Mealy). Simulación funcional (II)

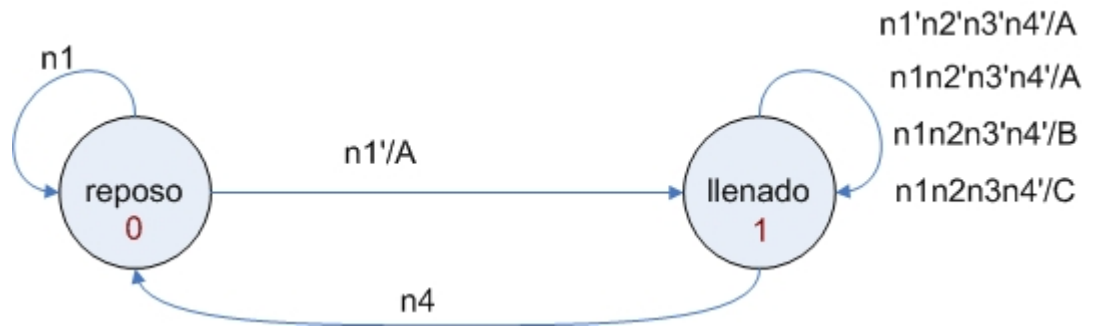
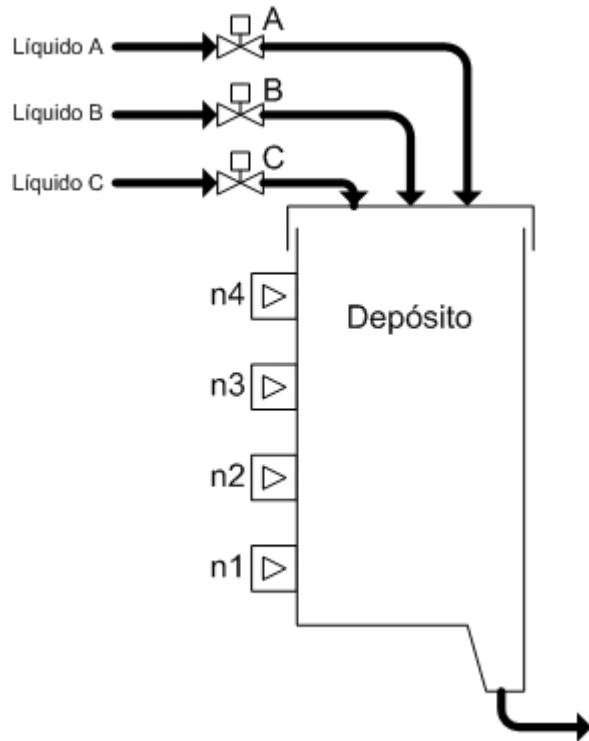
Detalle de la simulación funcional, donde se observa que las salidas no cambian con el flanco de reloj, sino en el momento en que cambien las entradas que les afectan.



## Tema 6: Sistemas Secuenciales

### Ejercicio VHDL

Describe en VHDL el autómata de Mealy que controla el depósito del ejemplo anterior, pero basándose en el diagrama de dos estados.

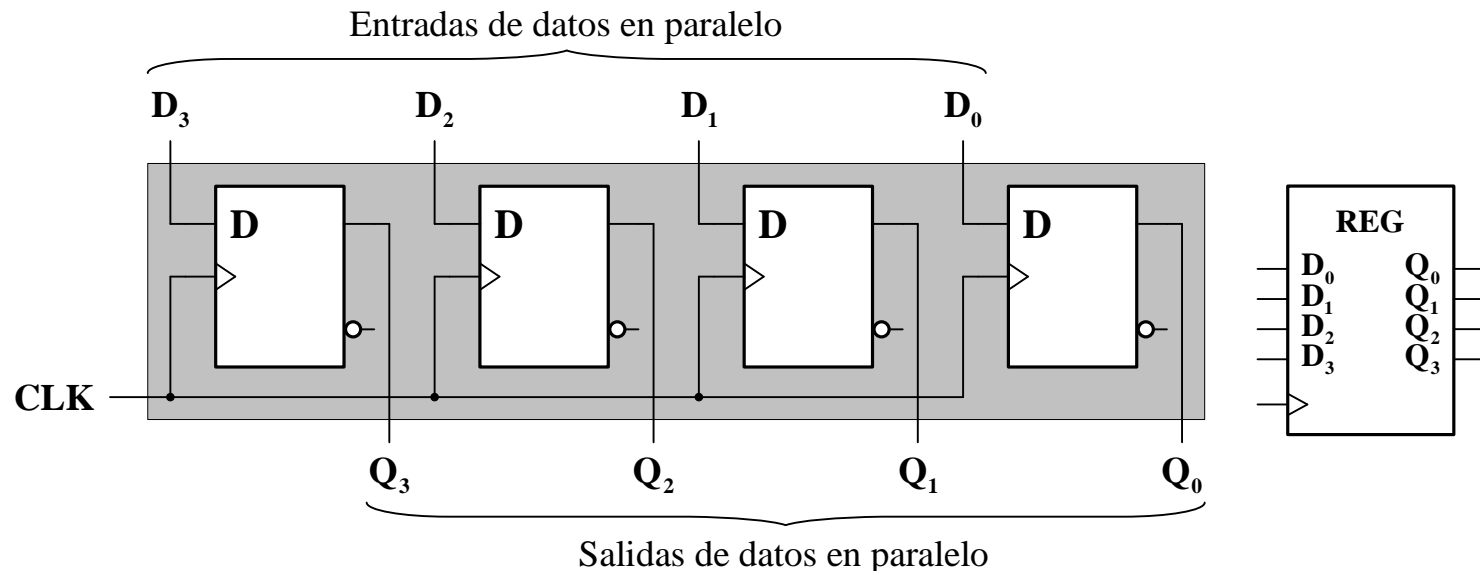




## Tema 6: Sistemas Secuenciales

### Circuitos Secuenciales de Propósito General: Registros

El biestable **D** constituye la célula básica de memoria capaz de almacenar y conservar un valor binario, bajo el control de la entrada de habilitación. Agrupando **n** de estos biestables en paralelo, con su entrada de habilitación común, se configura un registro de longitud **n**, capaz de almacenar una palabra de **n** dígitos o bits. Tal registro recibe el nombre de *memoria de retención* o «memoria cerrojo» (*latch-memory*), porque retiene la información almacenada cuando **E=1** durante todo el tiempo en que **E=0**, es decir, la conserva entre dos habilitaciones sucesivas. El registro así configurado recibe sus entradas en paralelo (todos los bits a la vez) y proporciona sus salidas en paralelo.

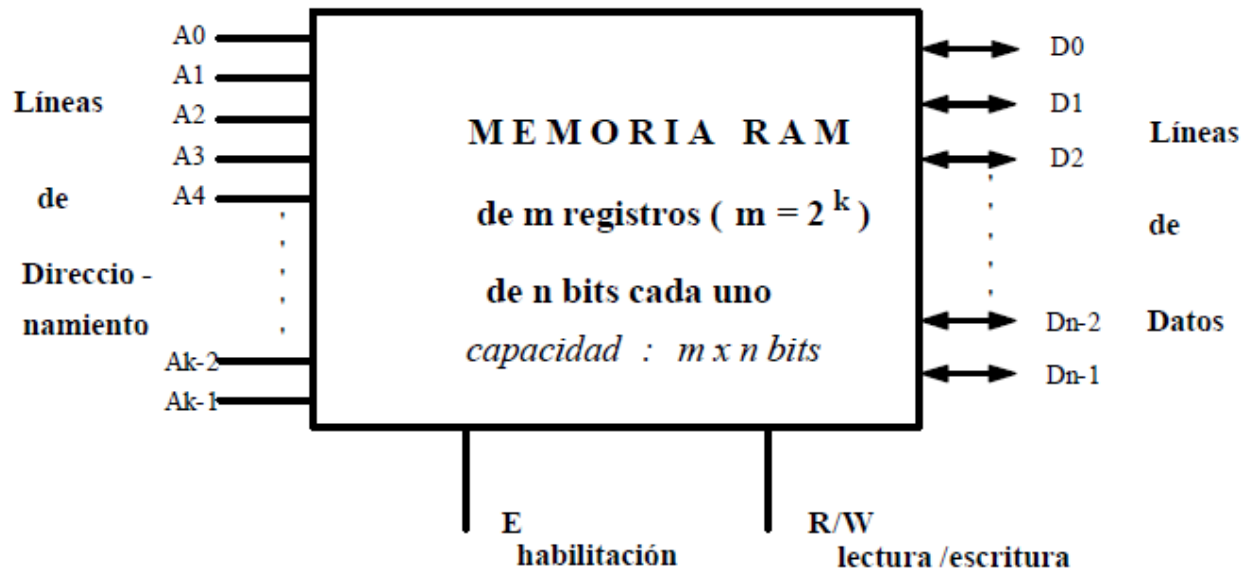


## Tema 6: Sistemas Secuenciales

### Circuitos Secuenciales de Propósito General: Memorias RAM estáticas (I)

La agrupación de  $m$  registros de  $n$  bits, controlados a través de  $k$  entradas de direccionamiento ( $m=2^k$ ), de forma que cada registro queda seleccionado por su número en binario, da lugar a un bloque de memoria **RAM**, *memoria de acceso directo* o aleatorio (*random access memory*).

Los terminales de entrada y de salida de los registros son comunes para todos ellos y, en cada momento, el vector presente en las entradas de control o direccionamiento indicará sobre cuál de los registros se actúa.



## Tema 6: Sistemas Secuenciales

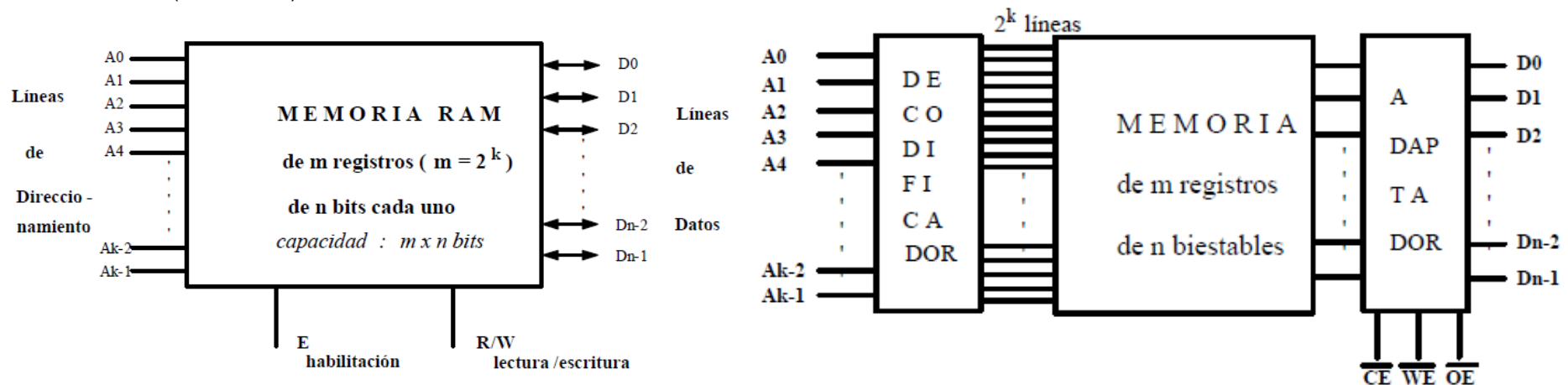
### Circuitos Secuenciales de Propósito General: Memorias RAM estáticas (II)

Además, por simplicidad de acceso, no hay distinción entre terminales de entrada y terminales de salida de los registros (es decir, la misma línea actúa como entrada y como salida), de forma que el tipo de acceso al registro seleccionado (lectura del registro o escritura del mismo) ha de ser indicado por una línea adicional **R/W** (*lectura/escritura*).

Un bloque RAM tendrá **k** líneas de direccionamiento **A<sub>i</sub>**, que actúan como entradas, **n** líneas de datos **D<sub>i</sub>**, que actúan bidireccionalmente, una entrada de selección de la operación a realizar **R/W**, que distingue entre lectura y escritura y una entrada de habilitación **E**.

Su esquema conceptual (mostrado abajo a la derecha) está conformado por un módulo central que contiene los **m** registros de **n** biestables, un decodificador de **k** líneas de entrada que selecciona los registros (**m = 2<sup>k</sup>**) y un circuito adaptador de entradas/salidas que controla la actuación de las **n** líneas de datos a tenor de las entradas de habilitación **E** y lectura/escritura **R/W**.

De esta forma, se dispone de unidades de memoria (biestables **D**) capaces de almacenar y conservar un bit, de registros (conjuntos de biestables **D**) capaces de almacenar una palabra binaria y de bloques de memoria (conjuntos de registros numerados) capaces de almacenar múltiples datos ordenados (numerados).

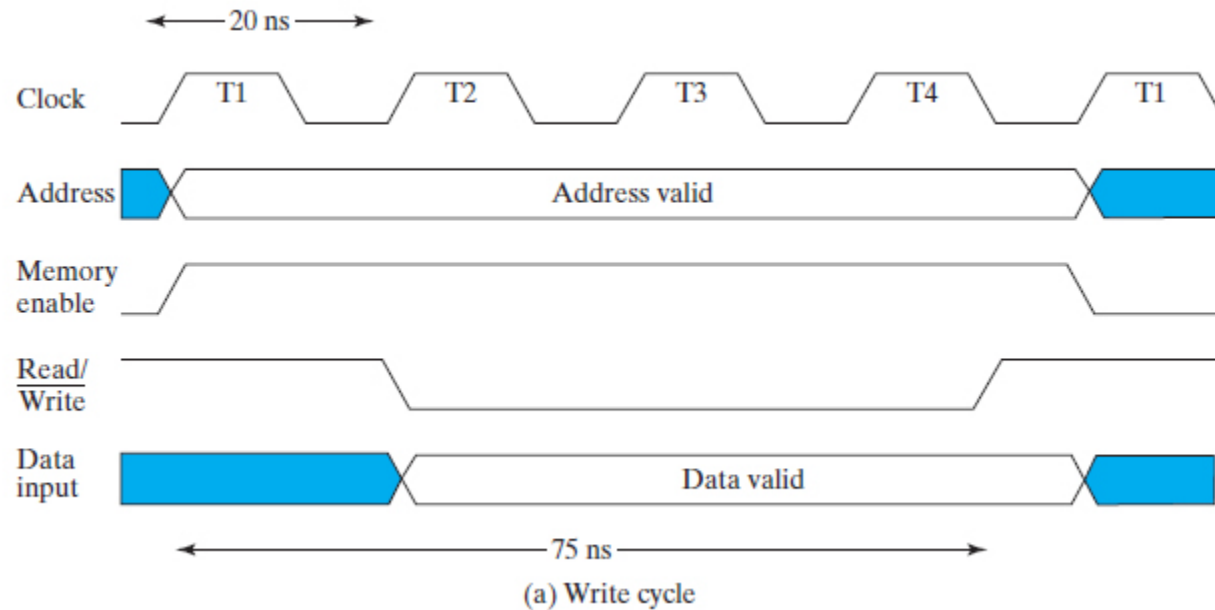


## Tema 6: Sistemas Secuenciales

### Circuitos Secuenciales de Propósito General: Memorias RAM estáticas (III)

#### Diagrama de tiempos de una RAM (I):

- El **tiempo de ciclo de escritura** es el tiempo máximo que transcurre desde que se pone la dirección hasta completar todas las operaciones internas que necesita la memoria para almacenar una palabra.

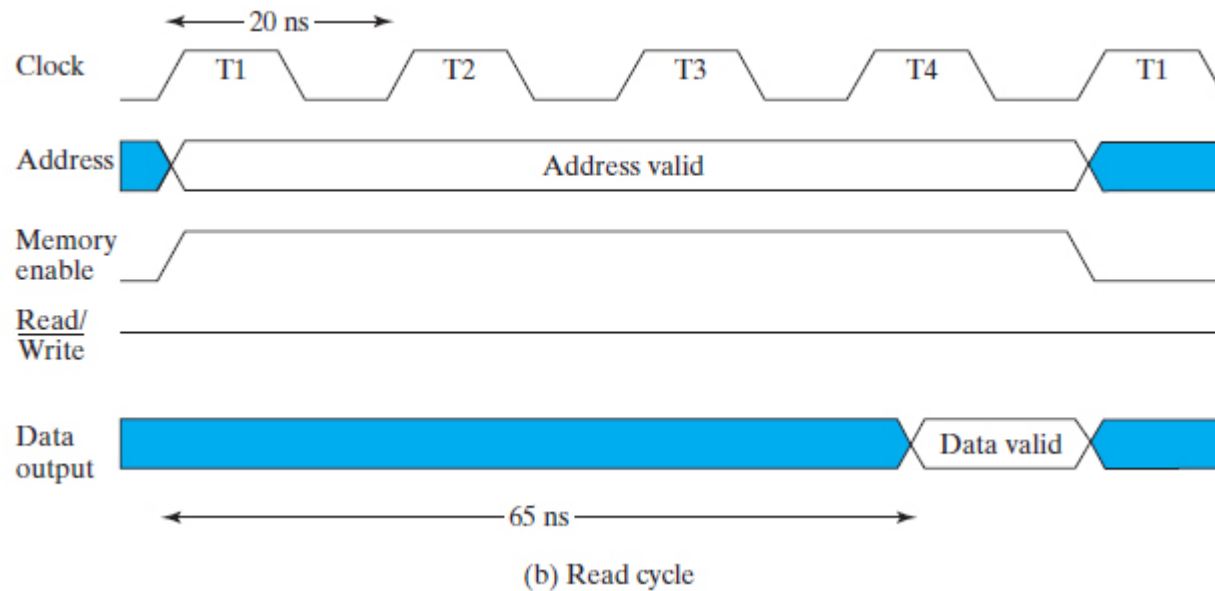


## Tema 6: Sistemas Secuenciales

### Circuitos Secuenciales de Propósito General: Memorias RAM estáticas (IV)

#### Diagrama de tiempos de una RAM (II):

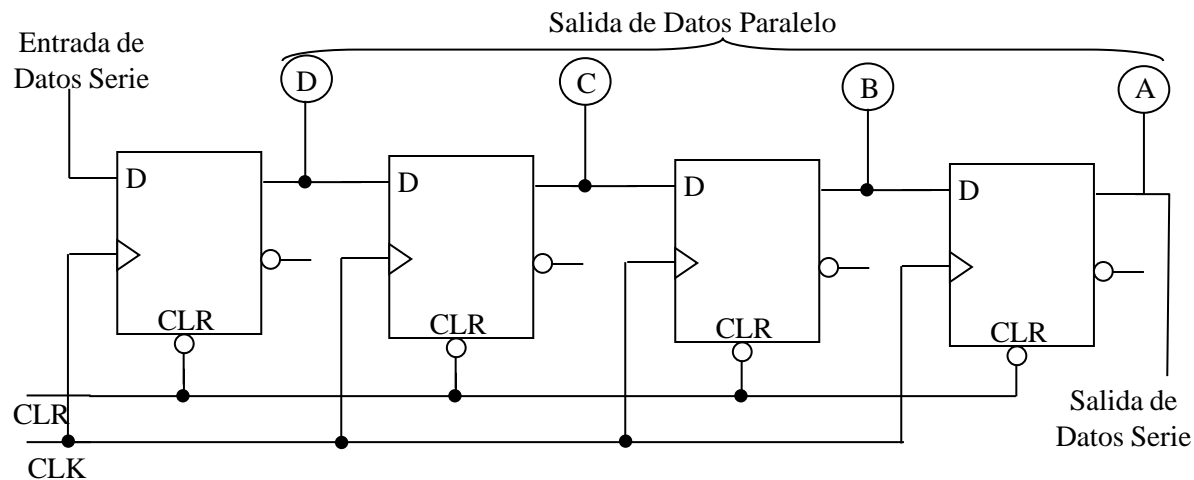
- El **tiempo de acceso de una operación de lectura** es el tiempo máximo que transcurre desde la aplicación de la dirección hasta que aparece la información en la salida de datos.



## Tema 6: Sistemas Secuenciales

### Registros de desplazamiento (I)

Un **registro de desplazamiento** 'registra' y 'desplaza' la información. La carga de la información puede ser en serie, la palabra se carga en el registro bit a bit con cada pulso de reloj, o en paralelo, la palabra se carga completa en el registro con un solo pulso de reloj. La transmisión del dato se hace también en **serie o en paralelo**.



La señal de borrado (**CLR**) coloca a los biestables en el estado 0. Cada vez que llega un flanco de subida de la señal de reloj (CLK), cada biestable 'captura' lo que hay en su entrada, o sea, lo que hay en el biestable anterior, con lo que la información se va desplazando a la derecha.

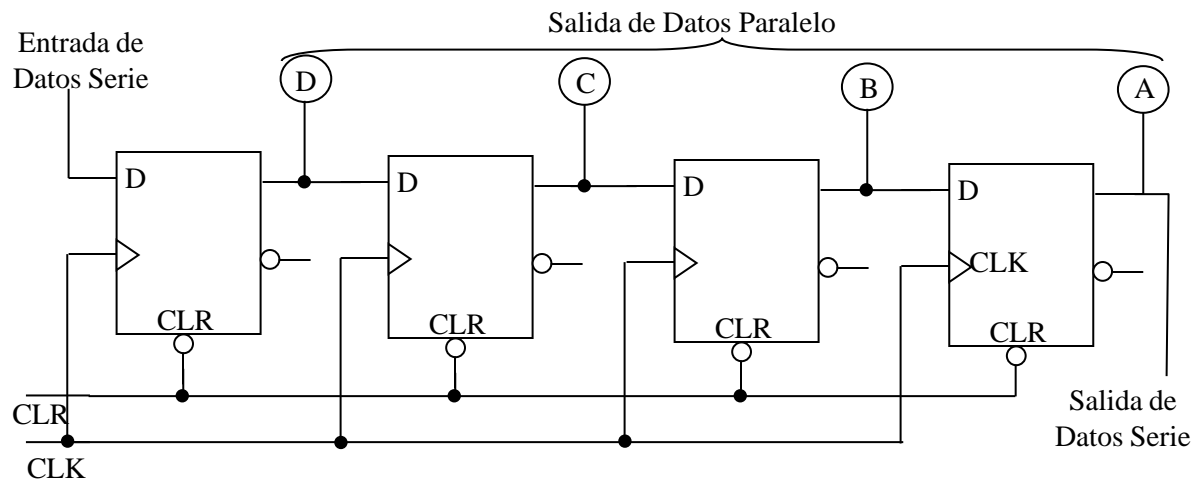
## Tema 6: Sistemas Secuenciales

### Registros de desplazamiento (II)

#### Ejemplo (I):

Vamos a registrar y desplazar la palabra de 4 bits **0101**. La secuencia de funcionamiento para registrar (memorizar en el registro) esta palabra de 4 bits se muestra en la siguiente tabla.

Valores de reloj y línea CLR	Salidas
	DCBA
CLR = 0	0 0 0 0
CLR = 1 / 1 <sup>er</sup> Pulso de CLK	1 0 0 0
CLR = 1 / 2 <sup>o</sup> Pulso de CLK	0 1 0 0
CLR = 1 / 3 <sup>er</sup> Pulso de CLK	1 0 1 0
CLR = 1 / 4 <sup>o</sup> Pulso de CLK	0 1 0 1



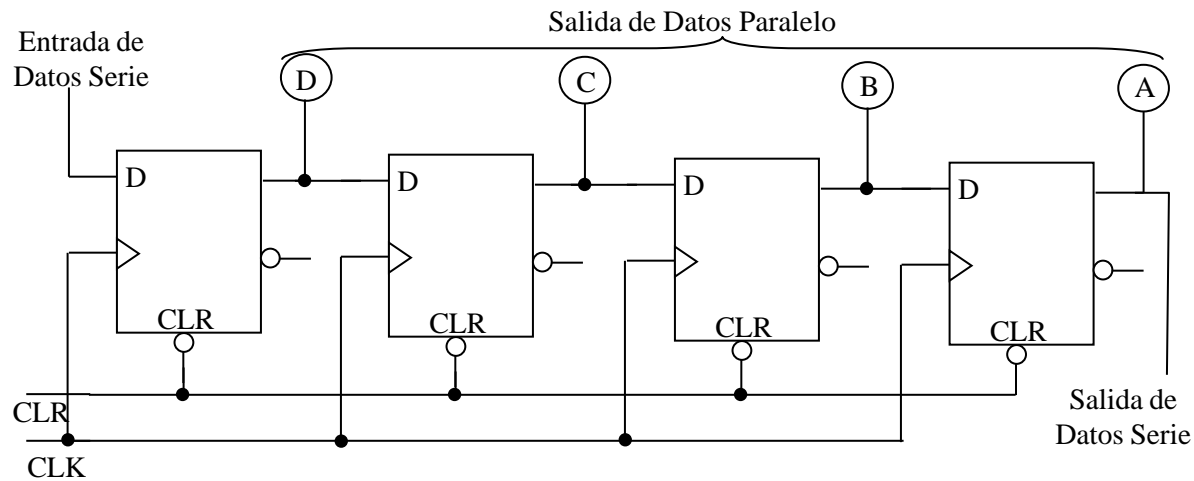
## Tema 6: Sistemas Secuenciales

### Registros de desplazamiento (III)

#### Ejemplo (II):

La palabra **0101** ha sido cargada en el registro y está disponible en las salidas paralelo. Para que estuviera en la salida serie:

Valores de reloj y línea CLR	Salida
	A
CLR = 1 / 4º Pulso de CLK	1
CLR = 1 / 5º Pulso de CLK	0
CLR = 1 / 6º Pulso de CLK	1
CLR = 1 / 7º Pulso de CLK	0

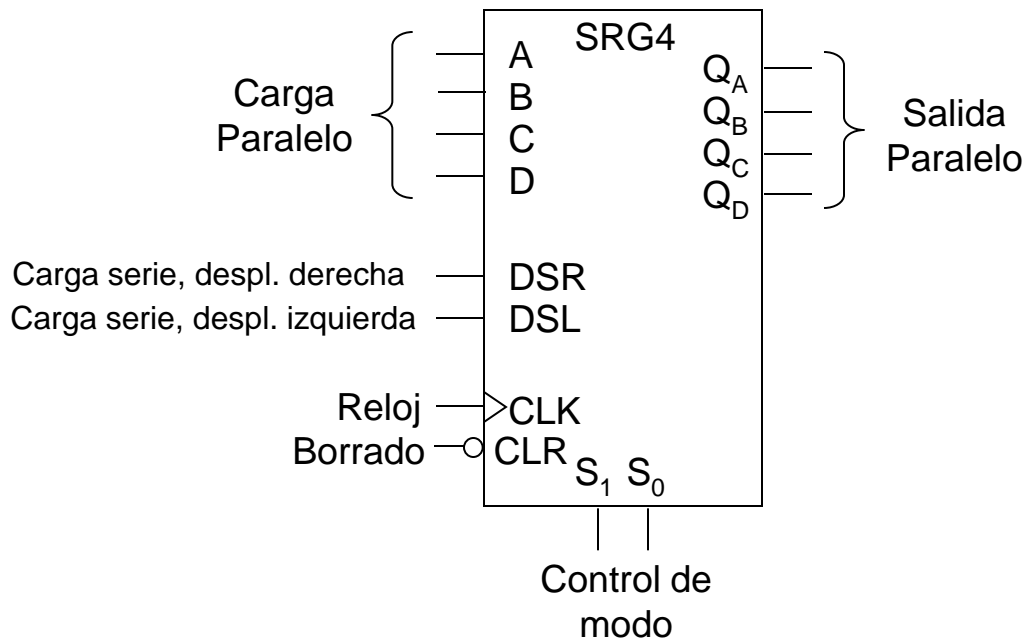




## Tema 6: Sistemas Secuenciales

### Registros universales

Se trata de un circuito integrado que dispone de un registro de desplazamiento, con carga serie, carga paralela, desplazamiento a izquierda y a derecha (por tanto también salida serie) seleccionables mediante unas señales de control.



Modo de operación	S <sub>1</sub>	S <sub>0</sub>
Mantenimiento	0	0
Despl. Izquierda	0	1
Despl. Derecha	1	0
Carga paralelo	1	1

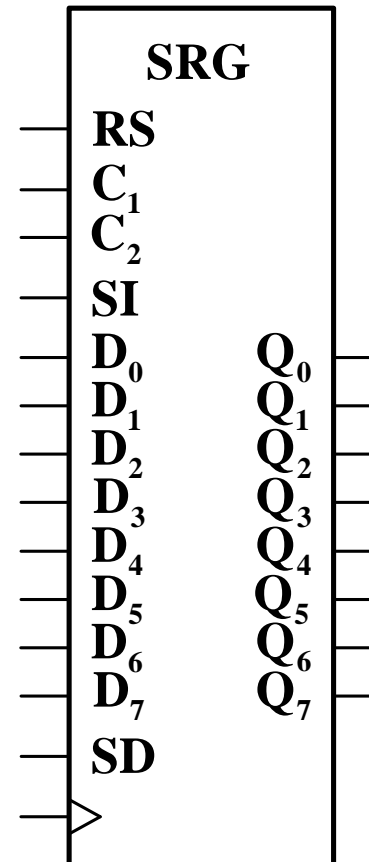
## Tema 6: Sistemas Secuenciales

### Descripción de Registros en VHDL (I)

Ejemplo: Registro de desplazamiento bidireccional de 8 bit con carga paralela síncrona

Se trata de diseñar un registro de desplazamiento con las cuatro posibilidades funcionales siguientes, controladas por dos entradas de selección (**C2** y **C1**):

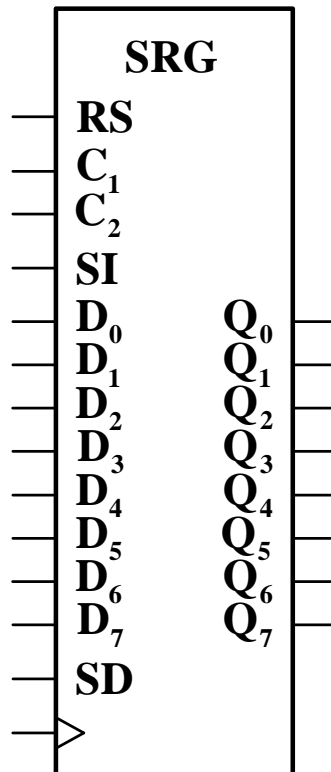
- **00**: retención del valor anterior;
- **01**: desplazamiento hacia la izquierda (entrada **SI**);
- **10**: desplazamiento hacia la derecha (entrada **SD**);
- **11**: carga paralelo (entradas **D**).



## Tema 6: Sistemas Secuenciales

### Descripción de Registros en VHDL (II)

Módulo VHDL: Obsérvese que no es necesario añadir  $Q\_interior \leq Q\_interior$ ; en *when others*, ya que un proceso tiene *memoria implícita* (conserva los valores).



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

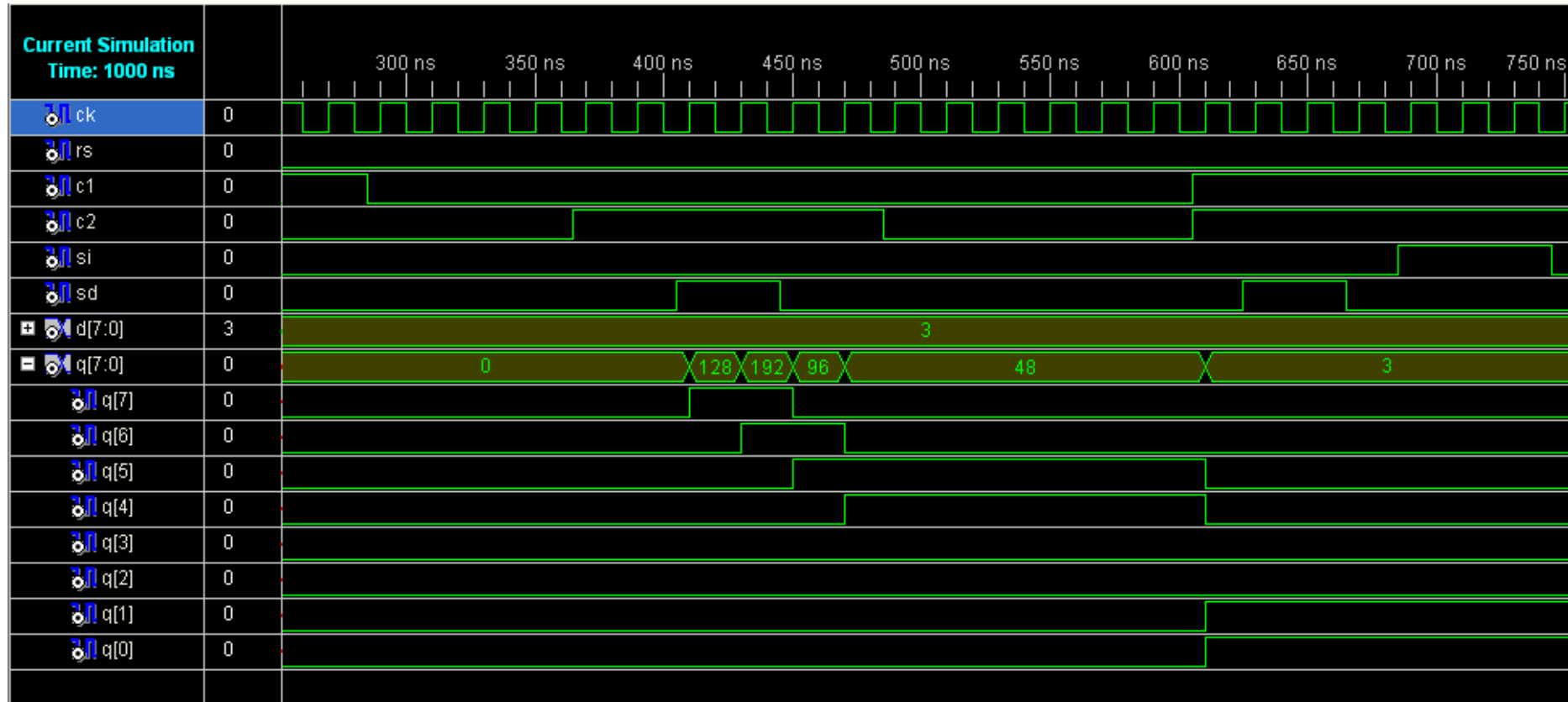
entity registro_8bits is
    port ( RS, CK, C1, C2, SI, SD : in std_logic;
          D : in std_logic_vector(7 downto 0);
          Q : out std_logic_vector(7 downto 0) );
end registro_8bits;

architecture SINCRONA of registro_8bits is
    signal Q_interior : std_logic_vector(7 downto 0);
    signal control : std_logic_vector(2 downto 1);
begin
    Q <= Q_interior;
    control <= C2 & C1;
    REGISTRO: process
    begin
        wait until CK = '1';
        if ( RS = '1' ) then Q_interior <= (others => '0');
        else case control is
            when "01" => Q_interior <= Q_interior(6 downto 0) & SI;
            when "10" => Q_interior <= SD & Q_interior(7 downto 1);
            when "11" => Q_interior <= D;
            when others =>
                end case;
        end if;
    end process; end SINCRONA;
```

## Tema 6: Sistemas Secuenciales

### Descripción de Registros en VHDL (III)

#### Simulación funcional del registro del ejemplo



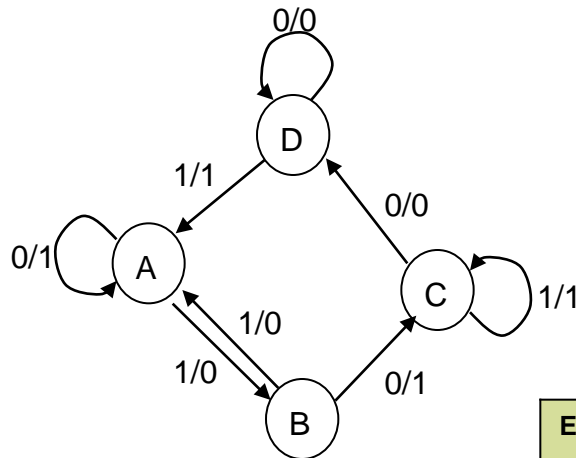
## Tema 6: Sistemas Secuenciales

### Diseño de sistemas secuenciales con registros

Existen diferentes variantes a la hora de diseñar un sistema secuencial con registros. De hecho, el método ya estudiado cuando empleamos biestables D, es equivalente a sustituir los  $n$  biestables D que tendría el sistema secuencial por un registro de  $n$  bits.

En ocasiones es interesante elegir una codificación para los estados distinta al sistema binario natural. Vamos a ver una de las más utilizadas, que se denomina **codificación mediante un código de uno entre n (un flip-flop por estado)**. Esta codificación permite que sólo un biestable esté activo en cada estado. Suele ser útil para el diseño de autómatas de Moore donde en cada estado se active una sola salida, pues permite asignar directamente cada salida a una variable de estado. Tiene la desventaja de que necesitamos más biestables (registros más grandes) para codificar los estados.

Ejemplo:



Ecuaciones:

$$Q3_{n+1} = AX' + DX + BX$$

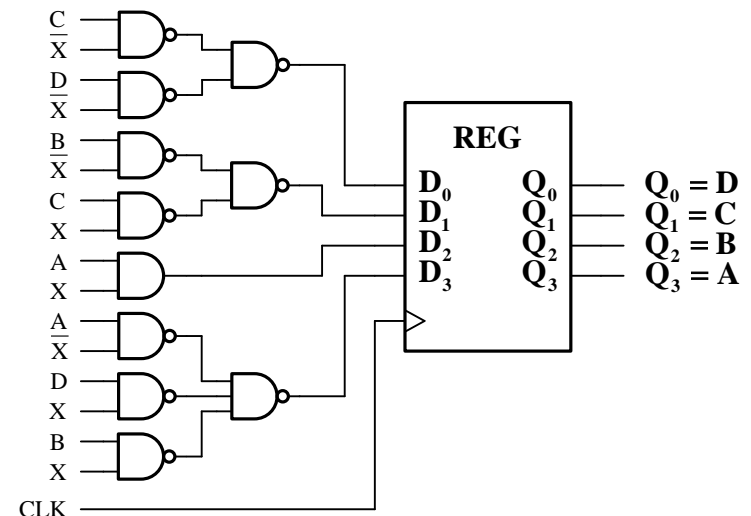
$$Q2_{n+1} = AX$$

$$Q1_{n+1} = BX' + CX$$

$$Q0_{n+1} = CX' + DX'$$

*Como ejercicio, calcular el valor de la salida del circuito, e incluirla en el esquema lógico.*

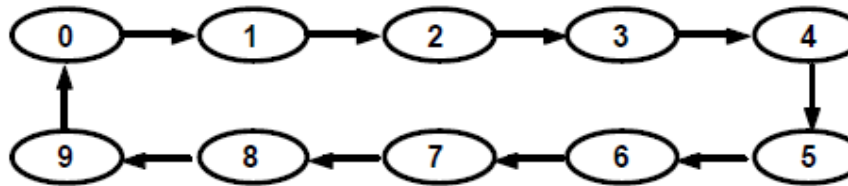
Estados	Q3	Q2	Q1	Q0
A	1	0	0	0
B	0	1	0	0
C	0	0	1	0
D	0	0	0	1



## Tema 6: Sistemas Secuenciales

### Contadores (I)

Un contador es un sistema secuencial conceptualmente muy simple: con cada pulso que recibe pasa de un estado al siguiente (cuenta el número de pulsos). Un contador módulo n presenta n estados (de 0 a n-1) y su evolución es circular: cuando llega al último estado, continúa desde el primero de ellos. Su diagrama de estados es un anillo, con una sola transición para cada estado (que le lleva al estado siguiente  $i+1$ ). En la figura puede verse el diagrama de estados de un contador de décadas (módulo 10).



A pesar de la sencillez de este comportamiento funcional, los contadores son extraordinariamente útiles en el diseño de sistemas digitales de medida y de control. Entre sus usos podemos señalar contar pulsos y dividir frecuencias. La propia cuenta directa de unidades, además de la información sobre número de objetos, personas o sucesos, permiten controlar, por ejemplo, el número de objetos a insertar en un envase, el número máximo de personas presentes en un recinto...

La división de frecuencias, consecuencia directa de la cuenta de sus pulsos, ofrece la posibilidad de disminuir (dividir por un factor) la frecuencia de las señales, aumentando así la duración de sus períodos.

Un aspecto de interés es la conexión de contadores para conformar otros más grandes, así como la posibilidad de respetar la codificación BCD para mantener la estructura de nuestros números decimales (base 10). La gran utilidad de los contadores se traduce en la disponibilidad de una amplia variedad de los mismos, entre los que se cuentan los contadores bidireccionales (cuentan hacia delante o hacia atrás, según el valor de una línea de entrada de control, contadores que permiten cargar en paralelo un valor inicial a partir del cual empiezan a contar, etc.

## Tema 6: Sistemas Secuenciales

### Contadores (II)

Un contador es un sistema secuencial, y si necesitáramos diseñar uno, podríamos hacerlo siguiendo el procedimiento de diseño básico ya explicado e este tema.

#### Ejercicio

Vamos a diseñar un contador síncrono binario que cuente de 0 a 7 y sea reversible. Emplearemos biestables JK y circuitos lógicos SSI y MSI.

Se muestra la tabla de estados que se obtiene directamente desde las especificaciones iniciales del problema.

Ent	Estado presente			Estado futuro			Entrada biestables					
A/D	qc	qb	qa	QC	QB	QA	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0	0	0	0	1	0	X	0	X	1	X
0	0	0	1	0	1	0	0	X	1	X	X	1
0	0	1	0	0	1	1	0	X	X	0	1	X
0	0	1	1	1	0	0	1	X	X	1	X	1
0	1	0	0	1	0	1	X	0	0	X	1	X
0	1	0	1	1	1	0	X	0	1	X	X	1
0	1	1	0	1	1	1	X	0	X	0	1	X
0	1	1	1	0	0	0	X	1	X	1	X	1
1	0	0	0	1	1	1	1	X	1	X	1	X
1	0	0	1	0	0	0	0	X	0	X	X	1
1	0	1	0	0	0	1	0	X	X	1	1	X
1	0	1	1	0	1	0	0	X	X	1	X	1
1	1	0	0	0	1	1	X	1	1	X	1	X
1	1	0	1	1	0	0	X	0	0	X	X	1
1	1	1	0	1	0	1	X	0	X	1	1	X
1	1	1	1	1	1	0	X	0	X	0	X	1

## Tema 6: Sistemas Secuenciales

### Contadores (III)

#### Ejercicio (cont.):

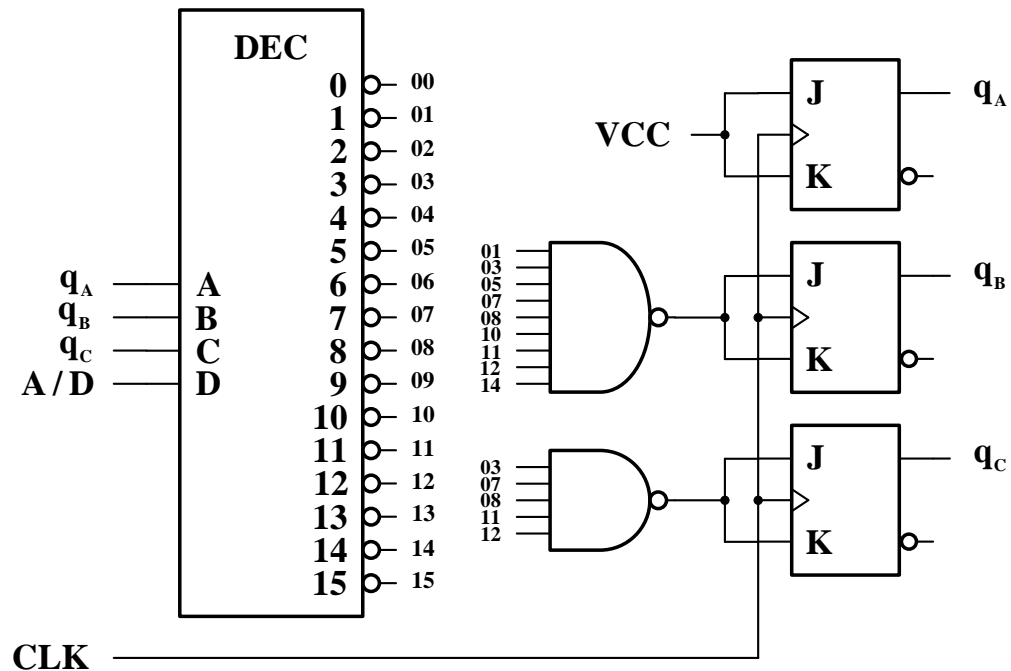
De la tabla se obtienen las funciones que excitarán cada biestable:

$$J_a = K_a = 1$$

$$J_b = K_b = \Sigma m(1,3,5,7,8,10,11,12,14)$$

$$J_c = K_c = \Sigma m(3,7,8,11,12)$$

Las funciones se realizan mediante un decodificador de 4 a 16 líneas y el circuito queda como:



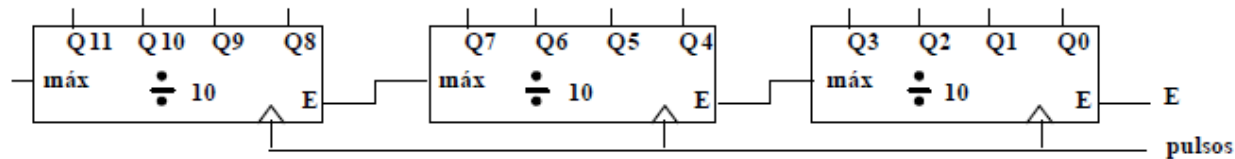


## Tema 6: Sistemas Secuenciales

### Conexión de contadores síncronos

Para conectar entre sí contadores síncronos, a fin de configurar un contador más amplio, es necesario añadir a cada contador una entrada de habilitación de conteo **E** y una salida que se active cuando el contador alcanza el estado máximo **max**; de esa forma conectando la salida **max** de un contador con la entrada **E** del siguiente, el segundo de los contadores se incrementará en una unidad cuando el primero de ellos haya alcanzado el valor máximo de su conteo.

Ejemplo: El contador de la figura es un contador módulo 1000 (10 x 10 x 10) que cuenta en BCD, ya que cada uno de los contadores que lo componen es módulo 10 (contador década); sus salidas pueden representarse sobre visualizadores de 7 segmentos, a través de sendos conversores de BCD en 7 segmentos.



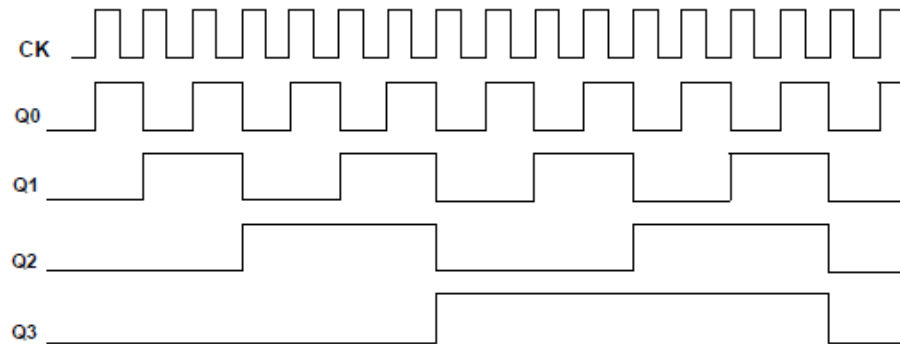
### Borrado de contadores

Generalmente los contadores disponen de una entrada de borrado cuya activación lleva a sus biestables al estado **0**. El borrado puede ser asíncrono, a través de la correspondiente entrada **CLR (Clear)** o **Reset** de borrado de cada uno de los biestables, o síncrono, con una entrada **B** que actúa sobre la función booleana de las entradas **D** de los biestables; el borrado asíncrono se produce inmediatamente después de activar la entrada de borrado **CLR**, mientras que el síncrono **B** se ejecuta en el flanco activo de reloj.

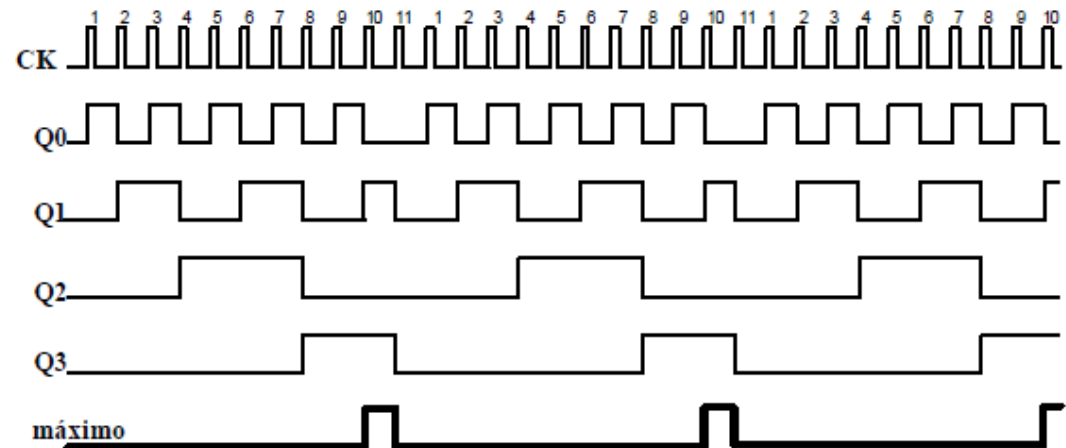
## Tema 6: Sistemas Secuenciales

### División de frecuencias y ampliación de períodos con contadores

La cuenta de pulsos se encuentra asociado directamente a la división de la frecuencia de los mismos: los biestables de un contador «completo» (módulo  $n$  potencia entera de 2) proporcionan en sus salidas ondas digitales cuyas frecuencias son, respectivamente, la mitad ( $1/2$ ), la cuarta parte ( $1/4$ ), la octava parte ( $1/8$ ),... ( $1/2^i$ )..., de la frecuencia de los pulsos de entrada. La división de la frecuencia por 2 implica la duplicación de su período  $T = 1/F$ :  $2T, 4T, 8T, 16T, \dots 2^i T, \dots$



Un contador módulo  $n$  configura un divisor de escala por  $n$  que genera un pulso por cada  $n$  pulsos recibidos en su entrada; este pulso puede obtenerse en la salida del último biestable (el más significativo) del contador; la cual proporciona un pulso por cada «vuelta» del contador (por cada  $n$  pulsos). Resulta preferible tomar el pulso de la salida máximo **max** del contador pues, en tal caso, la duración del pulso es de una unidad de tiempo del reloj (onda de temporización), que permite habilitar cualquier cambio o transición en forma síncrona y solamente durante un pulso de reloj.

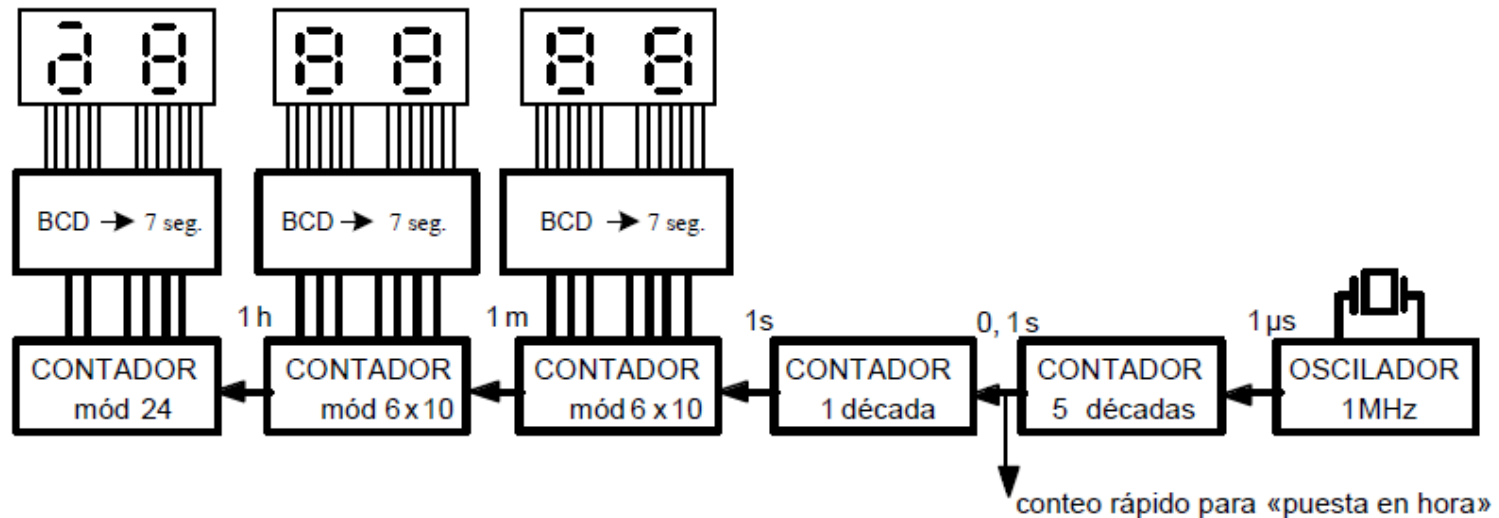


Ondas en las salidas de un contador módulo 11

## Tema 6: Sistemas Secuenciales

### Ejemplo de uso de contadores para dividir frecuencias: medida de tiempos

A partir de un generador de pulsos de frecuencia fija y muy precisa, cuyo período sea mucho menor que los intervalos temporales a medir, la medida de tiempos se reducirá a contar el número de pulsos en cada intervalo; dicha medida quedará expresada en unidades equivalentes al período de los pulsos. La figura representa un ejemplo de uso de contadores para crear un reloj digital a partir de un oscilador de cuarzo de 1 Mhz.



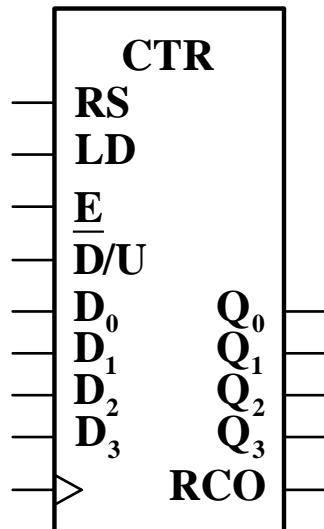
Un contador adicional módulo 7 permite indicar los días de la semana y un nuevo contador hasta 31 señalará el día del mes; el cual necesitará un contador módulo 12 para obtener el número del mes y la correspondiente lógica de ajuste para los meses de 30 días (y los 28/29 días de febrero). La puesta en hora de este reloj puede realizarse llevando directamente la señal rápida de 0,1 segundos, mediante pulsadores apropiados, a la señal de reloj del contador de minutos y del de horas, hasta que en cada uno de ellos se contabilice el número deseado.

Es sencillo dotar a este reloj de alarma o despertador mediante un contador duplicado de horas y minutos en el que se fija, por contaje directo con la señal de 0,1 s, la hora y el minuto en el que debe sonar la alarma; un comparador entre ambos contadores (horas y minutos) activa, con su salida de igualdad, un pequeño zumbador (en cuyo caso la alarma sonará durante 1 minuto).

## Tema 6: Sistemas Secuenciales

### Descripción de contadores en VHDL (I)

*Ejemplo: Contador módulo 10, bidireccional, con habilitación, borrado y carga paralela síncronos*



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

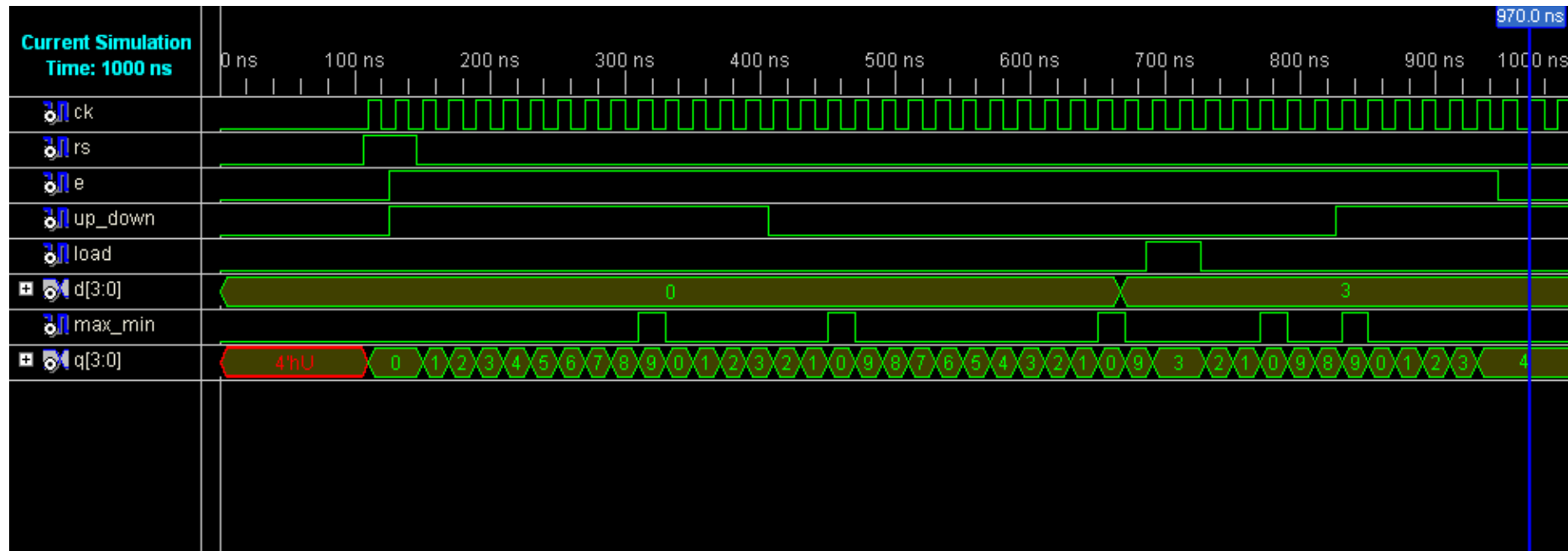
entity DECADA is
    port ( RS, CK, E, UP_DOWN, LOAD : in std_logic;
          D : in std_logic_vector(3 downto 0);
          MAX_MIN : out std_logic;
          Q : out std_logic_vector(3 downto 0) );
end DECADA;

architecture CONTADOR of DECADA is
    signal Q_interior : std_logic_vector(3 downto 0);
begin
    -- Definimos salidas del contador
    Q <= Q_interior;
    MAX_MIN <= '1' when ((UP_DOWN = '1') and (Q_interior = "1001") and E = '1')
    or ((UP_DOWN = '0') and (Q_interior = "0000") and E = '1') else '0';
    SINCRONO: process
        begin
            wait until CK = '1'; -- contador totalmente síncrono
            if ( RS = '1' ) then Q_interior <= "0000";
            elsif ( LOAD = '1' ) then Q_interior <= D;
            elsif ( E = '1' ) then
                if ( UP_DOWN = '1' ) then
                    if Q_interior = "1001" then Q_interior <= "0000";
                    else Q_interior <= Q_interior + 1; end if;
                else if Q_interior = "0000" then Q_interior <= "1001";
                else Q_interior <= Q_interior - 1; end if;
            end if;
        end if;
    end process;
end CONTADOR;
```

## Tema 6: Sistemas Secuenciales

### Descripción de contadores en VHDL (II)

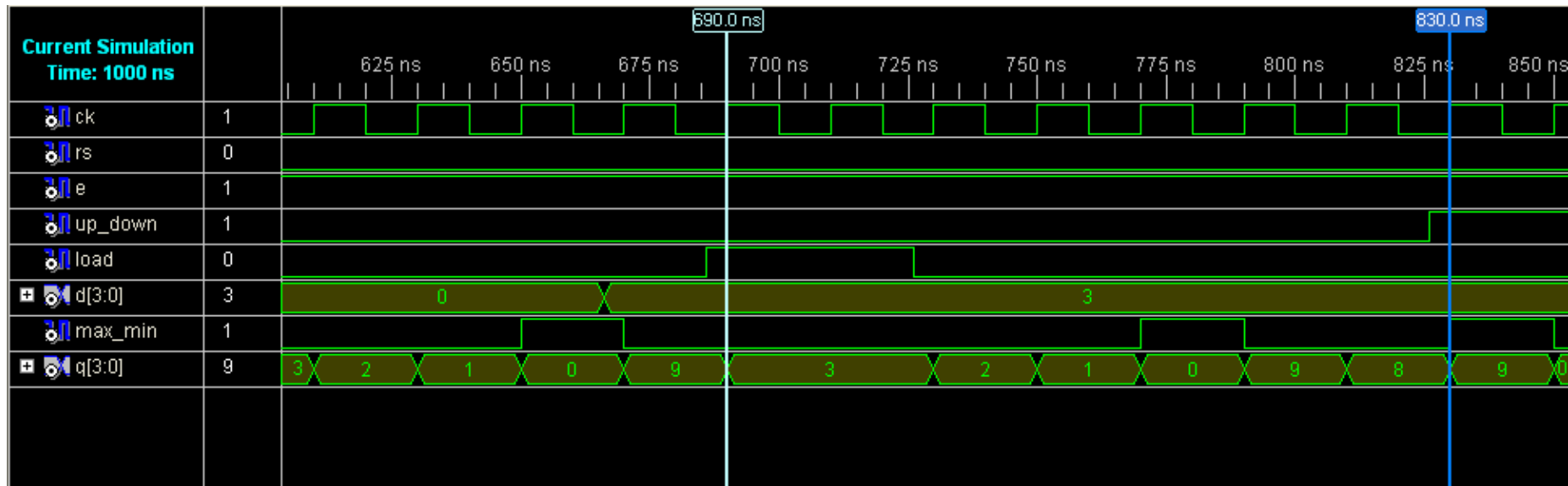
Simulación funcional del contador del ejemplo



## Tema 6: Sistemas Secuenciales

### Descripción de contadores en VHDL (III)

Detalle de la simulación funcional, donde se observa que la carga es síncrona. El contador carga el '3' (0011) una vez que la línea *LOAD* pasa a valer '1' y llega el primer flanco activo de reloj (instante 690 ns en la figura).

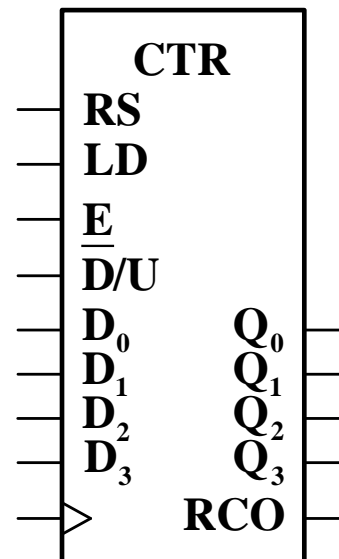


## Tema 6: Sistemas Secuenciales

### Descripción de contadores en VHDL (IV)

Ejercicio: Escribir un módulo VHDL de un contador con las siguientes características:

- contador módulo 30, bidireccional, con habilitación
- Borrado y carga paralela asíncronos



## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (I)

#### Triestado: alta impedancia

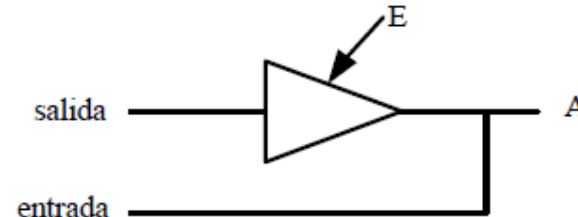
La forma de configurar el estado de alta impedancia en señales triestado es la siguiente:

```
salidas : out std_logic_vector (7 downto 0);  
....  
salidas <= "ZZZZZZZZ" when E = '0' else y;  
-- para no especificar el número de valores,  
salidas <= (others => 'Z') when E = '0' else y;  
-- dentro de un proceso  
if E = '0' then salidas <= (others => 'Z')  
else salidas <= y; end if;
```

#### Terminal bidireccional

Para configurar un terminal bidireccional se efectuarán dos asignaciones relativas a ese terminal; en una de ellas se le hará asignación como salida y se le indicará alta impedancia (Z) cuando actúe como entrada; en la segunda asignación se «leerá» dicho terminal como entrada, asignando su valor a otra señal.

```
A : inout std_logic;  
....  
signal entrada, E : std_logic;  
signal salida : std_logic;  
begin  
A <= salida when E = '1' else 'Z';  
entrada <= A;
```



Cuando el terminal bidireccional **A** actúa como entrada ( $E = '0'$ ), recibe el valor de fuera; en tal situación su asignación como salida debe hacerse a **Z** (alta impedancia).



## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (II)

**Ejemplo de uso de terminales bidireccionales:** Diseñar en VHDL un contador de 4 bits cuyas salidas actúan, también, como entradas paralelo. El contador debe ser completamente síncrono y su funcionamiento se basa en las líneas de control **LOAD** y **OE** (*Output Enable*), según la tabla que se muestra.

LOAD	OE	ACCIÓN DEL CONTADOR
1	1	Borrado
1	0	Carga paralela
0	1	Salidas muestran la secuencia de cuenta
0	0	Mantiene el valor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

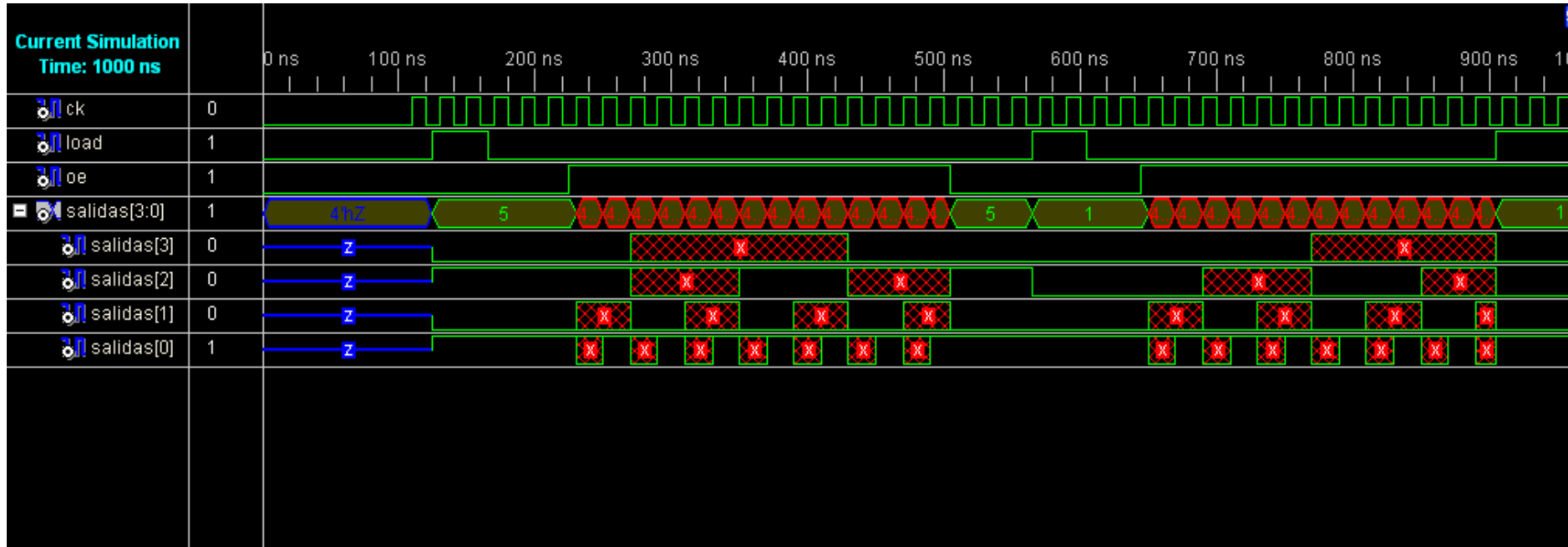
entity contador_inout is
    port( salidas : inout std_logic_vector(3 downto 0);
          CK, oe, load : in std_logic );
end contador_inout;

architecture bidir of contador_inout is
    signal qq : std_logic_vector(3 downto 0);
begin
    -- actuación como salidas
    salidas <= qq when (oe = '1') and (load = '0') else (others => 'Z');
    process
    begin
        wait until CK = '1';
        if load = '1' and oe = '1' then qq <= "0000"; -- borrado
        elsif load = '1' then qq <= salidas; -- carga paralelo
        elsif oe = '1' then qq <= qq + 1; -- contaje
        end if;
    end process;
end bidir;
```

## Tema 6: Sistemas Secuenciales

## Otros recursos VHDL (III)

### Simulación funcional del contador del ejemplo



## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (IV)

#### Funciones

Una función consiste en un conjunto de asignaciones, cuya aplicación a las entradas de la función sirve para devolver un valor. Se describe al inicio de la arquitectura, antes del *begin* de ésta y puede ser llamada, dentro de la arquitectura, cuantas veces sea necesaria; también puede estar descrita en un paquete de una librería.

Dentro de una función no puede ir una instrucción de espera (*wait*), ni una actuación por flancos (*rel*). Su descripción general es de la forma:

```
function nombre_de_la_función (entradas: tipo)
  return tipo_de_la_salida is
  begin
    asignaciones (como en un proceso)
  return .....; end;
```

*llamada a la función:*

```
señal <= nombre_de_la_función (entradas);
```

## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (V)

**Ejemplo de uso de funciones:** Diseñar un sistema que reciba 4 números de 8 bits cada uno y que proporcione a la salida el mayor de los 4.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mayor_de_4_numeros is
    port (A, B, C, D: in std_logic_vector (7 downto 0);
          salida: out std_logic_vector (7 downto 0) );
end mayor_de_4_numeros;

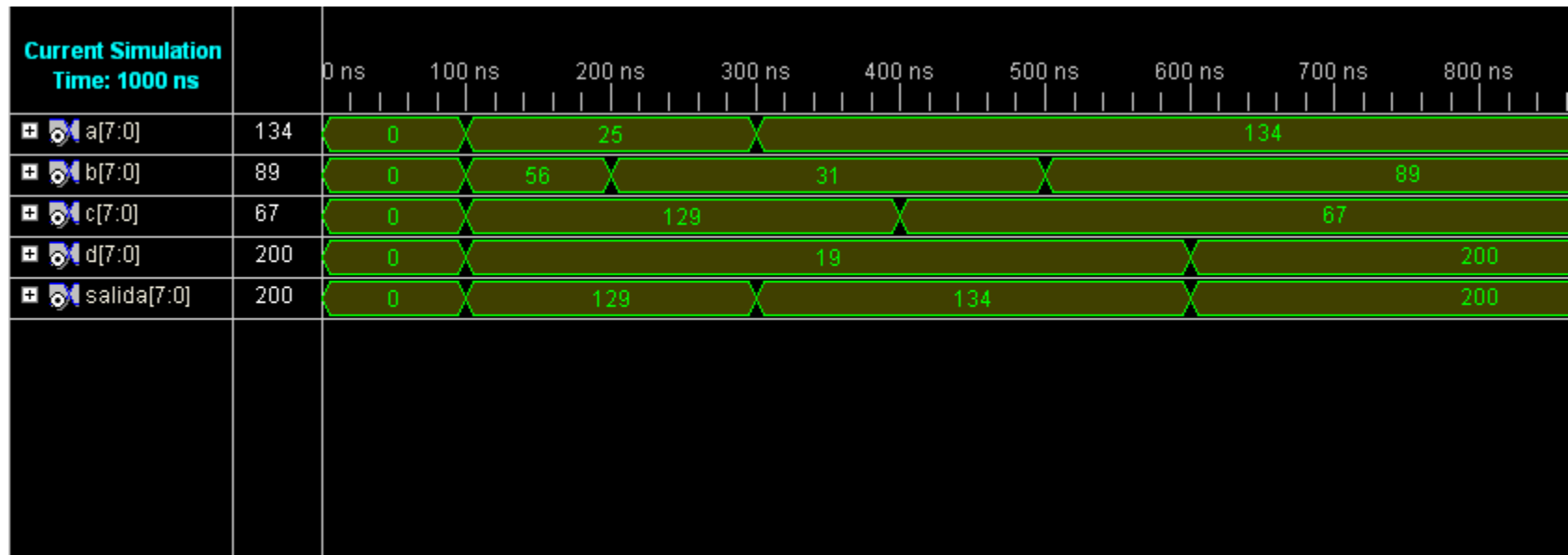
architecture Behavioral of mayor_de_4_numeros is
    signal x, y : std_logic_vector (7 downto 0);
    -- función que calcula el mayor de dos números y lo devuelve
    function mayor_de_2 (n1, n2: std_logic_vector (7 downto 0))
    return std_logic_vector is
    variable numero_mayor: std_logic_vector (7 downto 0);
    begin
        if (n1 > n2) then numero_mayor:= n1; else numero_mayor:= n2; end if;
    return numero_mayor; end;

    begin
        -- llamadas a funciones
        x <= mayor_de_2 (A, B);
        y <= mayor_de_2 (x, C);
        salida <= mayor_de_2 (y, D);
    end Behavioral;
```

## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (VI)

#### Detalle de la simulación funcional del ejemplo de uso de funciones



## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (VII)

#### Procedimientos

Un procedimiento es un subcircuito o módulo circuital que se describe antes del *begin* de la arquitectura para utilizarlo luego múltiples veces a lo largo de ella; también puede estar descrito en un paquete dentro de una librería como módulo reutilizable.

Permite no tener que repetir la descripción de un mismo subcircuito cuando este aparece varias veces dentro de una arquitectura. Su descripción general es de la forma:

```
procedure nombre_del_procedimiento (parámetros) is  
begin  
    asignaciones (como en un proceso)  
end procedure;
```

llamada al procedimiento:

```
nombre_del_procedimiento (parámetros);
```

En buena medida la descripción (y también la utilidad) de un procedimiento se asemeja a la de un proceso (*process*); es una especie de proceso reutilizable.

*En los diseños en los que una señal de un procedimiento aparezca en ambos lados de una asignación (por ejemplo  $q \leq q + 1$ ), debemos declarar esa señal como tipo **inout** y no como **out**.*

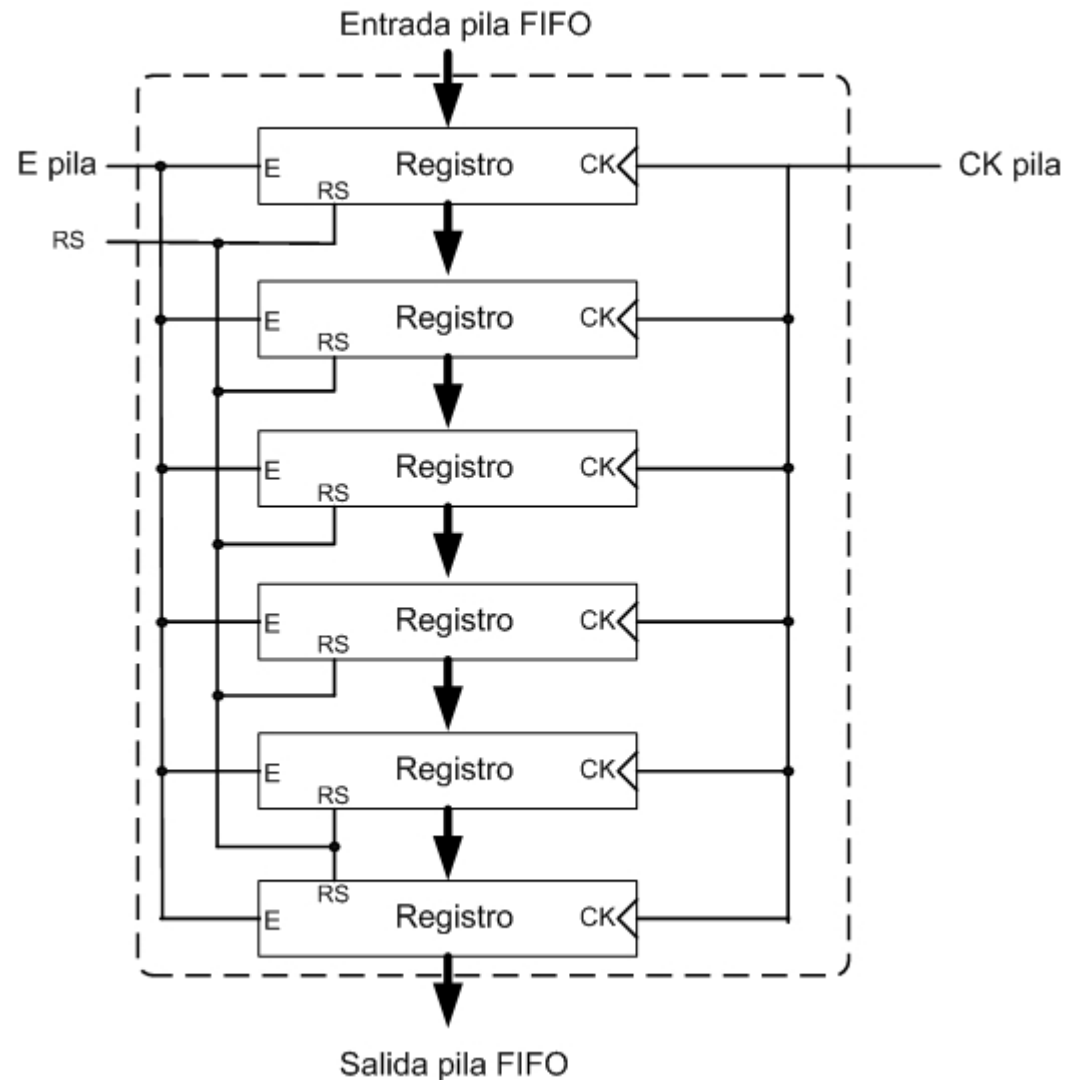
## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (VIII)

#### Ejemplo de uso de procedimientos (I):

*Descripción en VHDL de un registro en forma de procedimiento y construcción de una pila FIFO (First In First Out: Primero en Entrar, Primero en Salir) de 6 registros de 8 bits cada uno.*

Un pila FIFO es un conjunto de registros ordenados de tal manera que cuando entra un nuevo dato por los terminales de entrada, este dato “empuja” a los demás a la salida, de manera que el primer dato que entró es el primero que sale de la pila. La pila que diseñaremos tiene una línea de habilitación E y una entrada de reloj (CK), además de las líneas de entrada y de salida de datos. Dispone también de una línea de puesta a cero asíncrona (RS).



## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (IX)

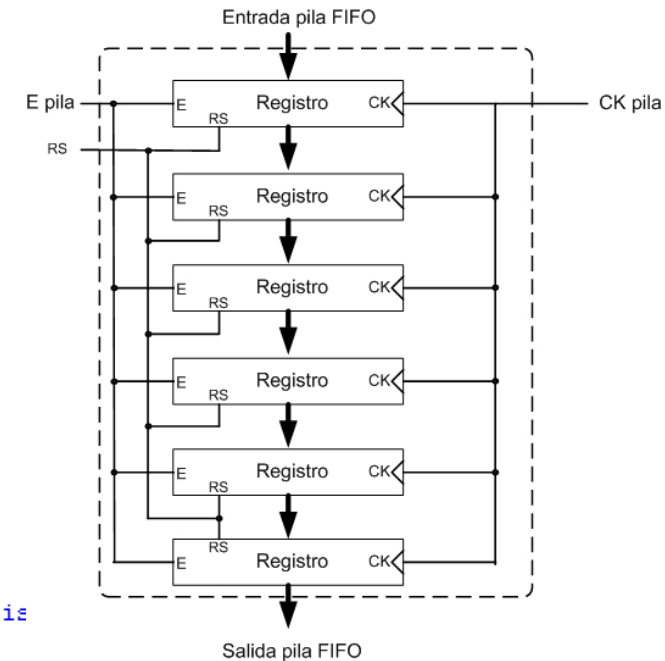
**Ejemplo de uso de procedimientos (II):** Pila FIFO de 6 registros en VHDL.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FIFO is
    port( RS, CK, E : in std_logic;
          entrada : in std_logic_vector(7 downto 0);
          salida : out std_logic_vector(7 downto 0) );
end FIFO;

architecture PROCEDIMIENTO of FIFO is
    procedure registro ( signal reset , clock, enable : in std_logic;
                        signal d : in std_logic_vector(7 downto 0);
                        signal q : out std_logic_vector(7 downto 0) ) is
    begin
        if reset = '1' then q <= "00000000";
        elsif clock'event and clock = '1' then if enable = '1' then q <= D; end if;
        end if;
    end procedure;
    type coleccion_6_reg is array(1 to 6) of std_logic_vector(7 downto 0);
    signal mis_registros : coleccion_6_reg;

    begin salida <= mis_registros(6);
    -- parámetros del registro: (reset, clock, enable, entrada, salida)
    registro (RS, CK, E, entrada, mis_registros(1));
    registro (RS, CK, E, mis_registros(1), mis_registros(2));
    registro (RS, CK, E, mis_registros(2), mis_registros(3));
    registro (RS, CK, E, mis_registros(3), mis_registros(4));
    registro (RS, CK, E, mis_registros(4), mis_registros(5));
    registro (RS, CK, E, mis_registros(5), mis_registros(6));
end PROCEDIMIENTO;
```

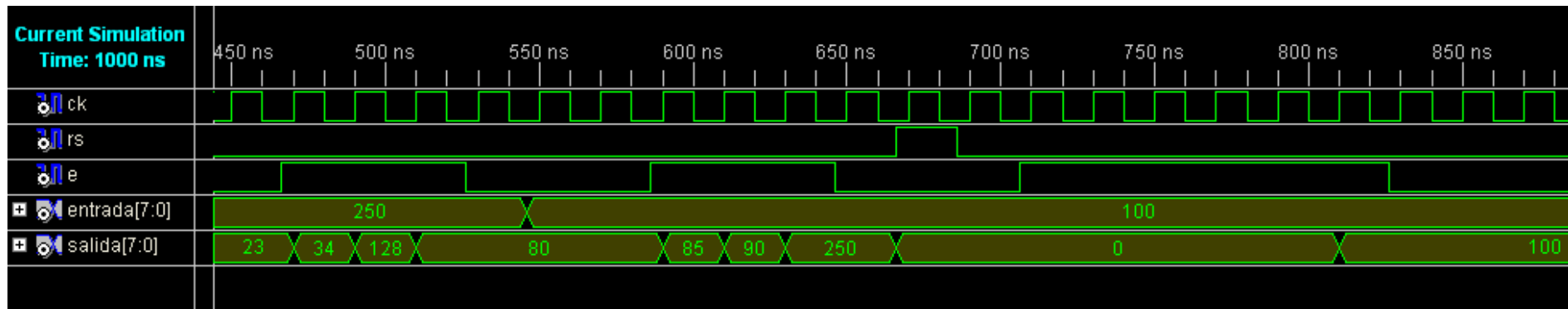
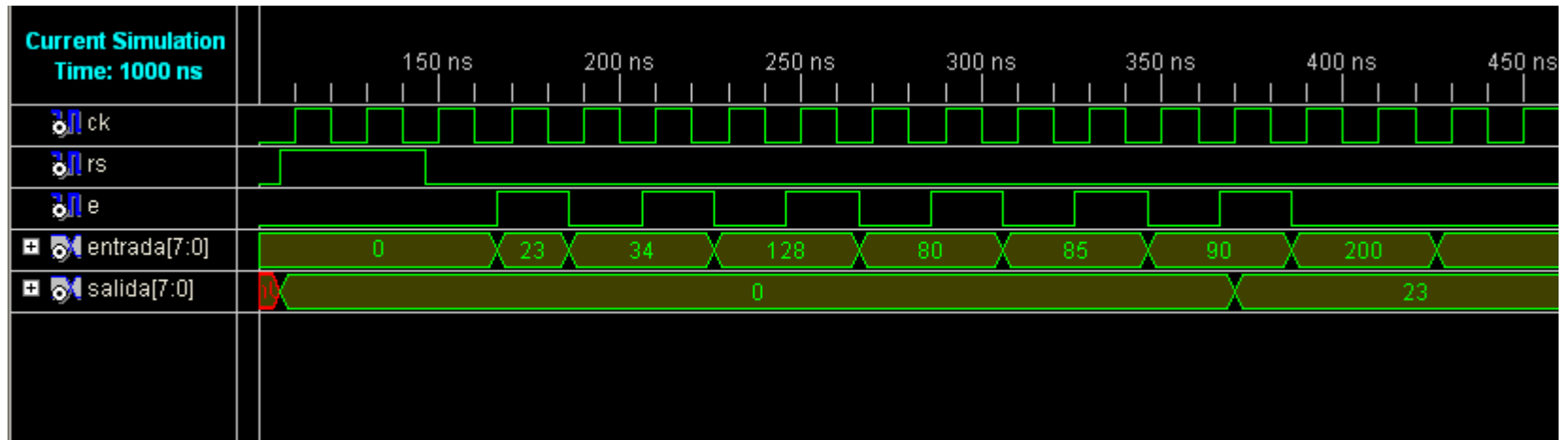




## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (X)

#### Simulación funcional del la pila FIFO



## Tema 6: Sistemas Secuenciales

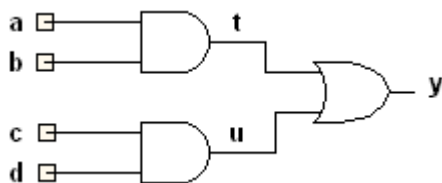
### Otros recursos VHDL (XI)

#### Descripción estructural: conexión de

#### módulos

Hasta aquí, la descripción VHDL de sistemas digitales se ha desarrollado en forma «funcional», pero este lenguaje admite también una descripción «estructural», detallando las conexiones entre celdas o módulos circuitales.

Ejemplo: descripción estructural de una sencilla combinación de puertas en forma de suma de productos.



Obsérvese que es necesario declarar la librería y los paquetes que se utilizan en cada entidad: la declaración de librerías y paquetes se «satura» (se gasta) cada vez que dicha librería se utiliza

```
library ieee; use ieee.std_logic_1164.all;
entity puerta_and is
    port(in1, in2 : in std_logic; salida : out std_logic); end puerta_and;
architecture uno of puerta_and is
    begin salida <= in1 and in2;
end uno;

library ieee; use ieee.std_logic_1164.all;
entity puerta_or is
    port(in1, in2 : in std_logic; salida : out std_logic); end puerta_or;
architecture dos of puerta_or is
    begin salida <= in1 or in2;
end dos;

library ieee; use ieee.std_logic_1164.all;
entity suma_de_productos is
    port(a, b, c, d : in std_logic; y : out std_logic);
end suma_de_productos;
architecture andor of suma_de_productos is
    component puerta_and
        port(in1, in2 : in std_logic; salida : out std_logic);
    end component;
    component puerta_or
        port(in1, in2 : in std_logic; salida : out std_logic);
    end component;
    signal t, u: std_logic;
    begin
        u1: puerta_and port map(in1=>a, in2=>b, salida=>t);
        u2: puerta_and port map(in1=>c, in2=>d, salida=>u);
        u3: puerta_or port map(in1=>t, in2=>u, salida=>y);
    end andor;
```

## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (XII)

#### **Declaración de parámetro y constantes generales**

También pueden definirse parámetros o valores constantes en la declaración de entidad, antes de los puertos, con la declaración de *generic* (genéricos), en la forma siguiente:

```
entity nombre_de_entidad is  
generic (  
    nombre_valor_generico : tipo := valor;  
    .... );  
port (.....
```

Ejemplo:

```
generic( m : integer := 8 );
```

#### **Bucle para generar varios módulos: generate**

Cuando se trata de conectar varios nodulos del mismo tipo, la «instrucción» **generate** actúa en forma de bucle según un índice que recorre un intervalo de valores:

```
etiqueta: for i in ... to ... generate  
    inserción de módulos  
end generate;
```

## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (XIII)

**Ejemplo de uso de la instrucción *generate* y de declaraciones de genéricos:** Diseñar una pila FIFO como la del ejemplo anterior empleando la instrucción *generate*. Este diseño consta de dos ficheros, el primero de ellos define un registro de tamaño ***num\_bits*** y el segundo define una pila, basada en los registros definidos en el primer fichero, de tamaño ***tam\_pila***. Simplemente cambiando el valor de estas dos constantes es fácil generar una pila de cualquier tamaño sin modificar el código fuente.

#### **Fichero de declaración del registro**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity registro is
    generic (num_bits : integer := 8);
    port( reset, clock, enable : in std_logic; d : in std_logic_vector((num_bits-1) downto 0);
          q : out std_logic_vector((num_bits-1) downto 0) );
end registro;
architecture funcional of registro is begin
    process(reset, clock, enable, d)
    begin
        if reset='1' then q <= (others => '0');
        elsif clock'event and clock = '1' then
            if enable = '1' then q <= d; end if;
        end if;
    end process; end funcional;
```

## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (XIV)

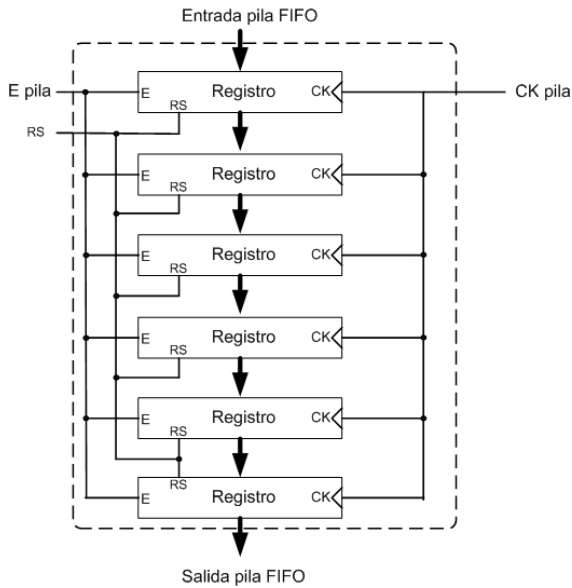
**Fichero de declaración de la pila que emplea registros como el definido en la diapositiva anterior**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FIFO is
    generic (num_bits : integer := 8);
    port( RS, CK, E : in std_logic;
          entrada : in std_logic_vector((num_bits-1) downto 0);
          salida : out std_logic_vector((num_bits-1) downto 0) );
end FIFO;

architecture pila of FIFO is
    component registro is
        port( reset, clock, enable : in std_logic; d : in std_logic_vector((num_bits-1) downto 0);
              q : out std_logic_vector((num_bits-1) downto 0) );
    end component;
    constant tam_pila: integer := 6;
    type coleccion is array(1 to tam_pila) of std_logic_vector((num_bits-1) downto 0);
    -- mis_vectores es una colección de señales, numeradas de 1 a (tam_pila) y que
    -- son vectores de (num_bits) bits
    signal mis_vectores: coleccion;

begin
    salida <= mis_vectores(tam_pila);
    u1: registro port map(reset=>RS, clock=>CK, enable=>E, d=>entrada, q=>mis_vectores(1));
    gen1 : for i in 2 to tam_pila generate
        uresto: registro port map(reset=>RS, clock=>CK, enable=>E, d=>mis_vectores(i-1), q=>mis_vectores(i));
    end generate;
end pila;
```



## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (XV)

#### Librerías y paquetes (I)

- ❑ Una **librería** es una «carpeta» (un directorio) en la que se almacenan diseños (con su entidad y su arquitectura, cada uno de ellos) que pueden ser utilizados como componentes en otros diseños. También se almacenan en las librerías **paquetes**, *que* contienen tipos, subtipos, constantes, componentes, funciones, procedimientos... como módulos disponibles para su uso en otros diseños.
- ❑ Librerías y paquetes sirven para organizar y ordenar los diseños y para aprovechar el trabajo generando módulos reutilizables.
- ❑ La librería directa de trabajo, en la que se desarrolla el diseño actual, se denomina **work** y no es necesario declararla.
- ❑ Ejemplo de librerías y paquetes son los declarados habitualmente al inicio del texto:

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

- ❑ Esta declaración indica el uso de la librería **ieee**; dentro de ella se usan los paquetes **std\_logic\_1164** y **std\_logic\_unsigned**; y dentro de dichos paquetes pueden utilizarse todos los elementos contenidos en ellos. Si se necesitase solo un componente del paquete, bastaría poner el nombre del elemento (en lugar de *all*).
- ❑ El «paquete» **std\_logic\_unsigned** describe operaciones aritméticas entre vectores y entre un vector y un entero. En el caso de que sean de longitud inferior a la señal que recibe el resultado de la operación, añade «ceros» delante hasta completar tal longitud.
- ❑ El «paquete» **std\_logic\_signed** describe las mismas operaciones pero completa dicha longitud duplicando el bit de signo, el primero de ellos; es decir, añade «unos» si el bit más significativo es 1 (lo cual equivale a considerar a los números codificados en complemento a 2).

## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (XVI)

#### Librerías y paquetes (II)

- ❑ La librería básica necesaria para las señales de los tipos introducidos en la normalización de IEEE (*std\_logic*) se encuentra disponible en los compiladores; también el diseñador puede crear sus propias librerías y paquetes.
- ❑ Un **paquete** es un fichero, dentro de una librería, en el que se declaran tipos y subtipos, se declaran y se asigna valor a constantes, se describen componentes, funciones y procedimientos...
- ❑ Un paquete se describe por «duplicado» mediante una **parte declarativa** (que, simplemente, enumera los contenidos del paquete con cierto detalle de su tipo y sus «parámetros») y otra **parte descriptiva** (que describe, en la forma habitual, cada uno de los módulos contenidos en el paquete a través de las asignaciones correspondientes).
- ❑ La declaración de un paquete y la descripción del mismo se realizan, una detrás de otra, en un mismo fichero de texto, con el nombre del paquete y la extensión *.vhd*; en caso de que los componentes del paquete utilicen otros paquetes es necesario declarar estos y sus librerías:

## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (XVII)

#### Declaración y descripción de un paquete:

Declaración previa de librerías y paquetes que se utilizan

**package** nombre\_del\_paquete **is**

-- declaración de tipos, constantes, componentes, funciones, procedimientos...

**type** nombre **is** ... ..;

**subtype** nombre **is** ... ..;

**constant** nombre : tipo;

**component** nombre **port**( ... .. ) **end component**;

**function** nombre (entradas) **return** tipo;

**procedure** nombre (parámetros)

**end** nombre\_del\_paquete;

**package body** nombre\_del\_paquete **is**

-- valores de las constantes y descripción de componentes, funciones... (los tipos o subtipos no figuran en la descripción del paquete porque ya han sido definidos, por completo, en su declaración)

**constant** nombre: tipo := valor;

**entity** y **architecture** de cada componente conforme a su descripción VHDL normal

-- descripción completa de cada función

**function** nombre (entradas) **return** tipo **is**

**begin**

asignaciones

**return** expresión;

**end**;

descripción completa de cada procedimiento

**procedure** nombre (parámetros)

**begin**

asignaciones

**end procedure**;

**end** nombre\_del\_paquete;



## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (XVIII)

#### Ejemplo de uso de librerías y paquetes (I)

Definimos de nuevo la pila FIFO (**FIFO.vhd**), pero usando ahora una librería (**libreria1**) que contiene un paquete (**paquete1**) que contiene la descripción del registro.

```
library IEEE, libreria1;
use IEEE.STD_LOGIC_1164.ALL;
use libreria1.paquete1.all;

entity FIFO is
    generic (num_bits : integer := 8);
    port( RS, CK, E : in std_logic;
          entrada : in std_logic_vector((num_bits-1) downto 0);
          salida : out std_logic_vector((num_bits-1) downto 0) );
end FIFO;

architecture pila of FIFO is
    component registro is
        port( reset, clock, enable : in std_logic; d : in std_logic_vector((num_bits-1) downto 0);
              q : out std_logic_vector((num_bits-1) downto 0) );
    end component;
    constant tam_pila: integer := 6;
    type coleccion is array(1 to tam_pila) of std_logic_vector((num_bits-1) downto 0);
    -- mis_vectores es una colección de señales, numeradas de 1 a (tam_pila) y que
    -- son vectores de (num_bits) bits
    signal mis_vectores: coleccion;

    begin
        salida <= mis_vectores(tam_pila);
        u1: registro port map(reset=>RS, clock=>CK, enable=>E, d=>entrada, q=>mis_vectores(1));
        gen1 : for i in 2 to tam_pila generate
            u_resto: registro port map(reset=>RS, clock=>CK, enable=>E, d=>mis_vectores(i-1), q=>mis_vectores(i));
        end generate;
    end pila;
```

## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (XIX)

#### Ejemplo de uso de librerías y paquetes (II)

Definición del paquete (**paquete1.vhd**) que contiene la descripción del registro.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package paquete1 is

    component registro is
        generic ( num_bits : integer := 8 );
        port( reset, clock, enable : in std_logic; d : in std_logic_vector((num_bits-1) downto 0);
              q : out std_logic_vector((num_bits-1) downto 0) );
    end component;

end paquete1;

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity registro is
    generic (num_bits : integer := 8);
    port( reset, clock, enable : in std_logic; d : in std_logic_vector((num_bits-1) downto 0);
          q : out std_logic_vector((num_bits-1) downto 0) );
end registro;
architecture funcional of registro is begin
    process(reset, clock, enable, d)
    begin
        if reset='1' then q <= (others => '0');
        elsif clock'event and clock = '1' then
            if enable = '1' then q <= d; end if;
        end if;
    end process; end funcional;
```

## Tema 6: Sistemas Secuenciales

### Otros recursos VHDL (XX)

#### Ejemplo de uso de librerías y paquetes (III)

A continuación puede verse como relaciona los distintos ficheros del diseño el software ISE WebPACK de Xilinx y el esquemático RTL sintetizado.

