14 DISEÑO SECUENCIAL SÍNCRONO

Los biestables síncronos son las celdas básicas del sincronismo, gracias a las cuales se consigue configurar sistemas síncronos. Las variables de estado se almacenan en biestables síncronos y las funciones de evolución del estado corresponderán a las de activación de dichos biestables.

Hoy día se utilizan habitualmente biestables básicos de tipo D; el diseño con biestables D requiere la obtención funcional del nuevo valor de la variable de estado, a través de la tabla de evolución del estado ($D_i = q_i^+$).

Ahora bien, en casos de tabla funcional muy compleja (de gran número de variables) resulta útil separar las condiciones de marcado y de borrado: desde el propio grafo de estados es posible obtener directamente las funciones de marcado S y borrado R y el diseño con biestables tipo D se reduce a agrupar ambas funciones. Ello equivale a razonar en términos de biestables tipo JK, que son el equivalente síncrono (ampliado) del biestable RS, y la traslación del diseño a biestables D se hace en forma análoga a como se conforman biestables JK a partir de biestables D.

Un diversificado conjunto de ejemplos de diseño ilustran la forma de aplicar estos procedimientos; entre ellos se incluyen muchos casos típicos, como son: sumadores serie, detección de secuencias, contadores, dados electrónicos, generación de ondas sincronizadas entre sí, detectores de entrada y salida de personas, activación/desactivación mediante detección de secuencias, semáforo con demanda de paso, …

Al igual que la configuración ROM permite construir las funciones booleanas sin obtener su expresión algebraica, directamente desde su tabla funcional, la evolución de las variables de estado puede construirse mediante un registro (que contenga el estado actual) y un codificador ROM (que proporcione el estado siguiente): esta forma de configurar grafos de estado se denomina «microprogramada» ya que el codificador contendrá el «programa» de estados sucesivos, en función del estado actual y del vector de entrada.

Tomás Pollán Santamaría. Tecnología Electrónica. E.U.I.T.I.Z. Universidad de Zaragoza. tpollan@posta.unizar.es

14.1. Diseño de sistemas secuenciales con biestables síncronos

Cualquier sistema secuencial de una cierta complejidad (es decir, con excepción únicamente de los sistemas digitales muy simples o muy directos) debe diseñarse en forma síncrona, con una señal de reloj **CK**, común a todo el sistema, que organiza el tiempo en unidades y señala con precisión los momentos de cambio de estado: flancos activos del reloj.

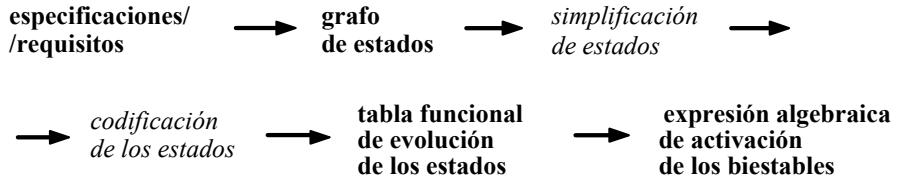
El sincronismo facilita en gran medida el diseño de un sistema complejo, al dividir el tiempo en unidades sucesivas y, con ello, hacer posible el razonamiento con unidades de tiempo discretas y numeradas.

Pero, sobre todo, el sincronismo proporciona seguridad de funcionamiento: evita fallos en las transiciones de estado al dedicar el intervalo correspondiente a cada unidad de tiempo al cálculo del nuevo estado y ejecutar simultáneamente el nuevo estado, ya completamente preparado, al finalizar la unidad de tiempo (en el siguiente flanco activo del reloj). De forma que los «espurios» o valores transitorios de algunas variables que puedan formarse durante la transición de un estado al siguiente no llegan a afectar al circuito porque el nuevo estado no es «admitido» hasta el siguiente flanco activo del reloj, cuando ya ha sido calculado por completo a lo largo de la unidad de tiempo.

El sincronismo de una variable, su modificación coincidente con los flancos activos de la señal de reloj, se consigue mediante una configuración «amo/esclavo» (*master/slave)*, es decir, empleando un biestable síncrono para dicha variable. Por ello, los sistemas secuenciales síncronos se diseñan utilizando biestables síncronos: habitualmente biestables síncronos tipo **D**.

El diseño de un sistema secuencial síncrono, en lo que a sus variables de estado se refiere, consiste en expresar las funciones de evolución del estado en términos de activación de las entradas de sus biestables.

Para ello han de recorrerse los sucesivos niveles de descripción del sistema secuencial:



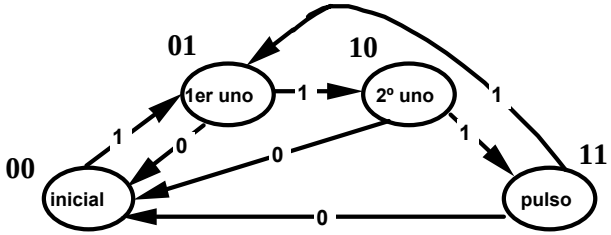
14.1.1. Diseño con biestables D

El biestable tipo D es el biestable síncrono básico y el más utilizado; para diseñar un sistema secuencial con biestables D conviene construir la tabla funcional de evolución del estado y, en ella, la propia columna $\mathbf{q_i}$ constituye la función de activación de su entrada:

$$\mathbf{D_i(t) = q_i(t+1) = q_i^+ = f_i(\ X\ ,\ Q\ )}$$

La síntesis de las funciones de activación de las salidas constituye un proceso meramente combinacional: $\mathbf{Y = f(\ X\ ,\ Q\ )}$.

Consideremos un ejemplo clásico, muy simple, el detector de una secuencia de bits consistente en tres «unos» seguidos: *un sistema con una sola entrada **A**, por la cual recibe dígitos binarios en unidades de tiempo sucesivas, debe responder con un pulso (de duración igual a una unidad de tiempo) cuando detecta la llegada de tres unos seguidos **111**; suponemos que el sistema no admite solapes (una vez recibidos tres unos, para volver a activar la salida es preciso que reciba otros tres unos diferentes).*



Este grafo puede ser codificado con dos variables de estado; las funciones de activación de las entradas de los biestables **D**, obtenidas a partir de la tabla de evolución del estado serán:

estado anterior q_2	q_1	**a**	q_2^+	q_1^+ *nuevo estado*	
0	**0**	**0**	**0**	**0**	
		1	**0**	**1**	
0	**1**	**0**	**0**	**0**	
		1	**1**	**0**	
1	**0**	**0**	**0**	**0**	$D_1 = q_1^+ = a.(q_2 + \overline{q_1})$
		1	**1**	**1**	
1	**1**	**0**	**0**	**0**	$D_2 = q_2^+ = a.(q_2 \oplus q_1)$
		1	**0**	**1**	

y la función de activación de la salida corresponde al estado **11**: $y = q_2.q_1$.

14.1.2. Diseño desde el grafo de estados

Razones de tiempo y de complejidad obligan en ocasiones a realizar el diseño directamente desde el grafo de estados (por la dificultad de construir la tabla funcional de evolución de los estados). En tal caso, una vez realizada la simplificación y la codificación de los estados, se pasa a considerar cada variable de estado $\mathbf{q_i}$, dividiendo para cada una de ellas el grafo de estados en dos partes:

- conjunto de estados en que la variable $\mathbf{q_i}$ tiene valor **1**;
- conjunto de estados en que la variable $\mathbf{q_i}$ tiene valor **0**.

Para cada una de las transiciones que pasan de la parte en que $\mathbf{q_i}$ vale **0** a la parte en que $\mathbf{q_i}$ vale **1** se obtiene un término de marcado de dicha variable $\mathbf{q_i}$, dado por el producto booleano del término mínimo correspondiente al estado anterior por el término de entrada que provoca dicha transición.

De igual forma para cada transición que pasa de la parte en que $\mathbf{q_i}$ vale **1** a la parte en que $\mathbf{q_i}$ vale **0** se obtiene un término de borrado de la variable $\mathbf{q_i}$, dado asimismo por el producto booleano del término mínimo correspondiente al estado anterior por el término de entrada que provoca la transición.

Las funciones de marcado y borrado conducen, de por sí, a un diseño con biestables **JK**: $\mathbf{J_i}$ equivale, en tal sentido, a $\mathbf{S_i}$ y debe recoger todos los términos de marcado de $\mathbf{q_i}$ y $\mathbf{K_i}$, que equivale $\mathbf{R_i}$, debe recibir todos los términosde borrado:

$\mathbf{J_i}$ = suma booleana de los términos de marcado de $\mathbf{q_i}$;

$\mathbf{K_i}$ = suma booleana de los términos de borrado de $\mathbf{q_i}$.

En el ejemplo anterior, la obtención de las funciones de marcado y borrado directamente desde el propio grafo de estados, conduce a:

q_1: $J_1 = \overline{q_2}.\overline{q_1}.a + q_2.\overline{q_1}.a = \overline{q_1}.a$ $\qquad K_1 = \overline{q_2}.q_1 + q_2.q_1.\overline{a} = q_1.(\overline{a} + \overline{q_2})$

q_2: $J_2 = \overline{q_2}.q_1.a$ $\qquad K_2 = q_2.\overline{q_1}.\overline{a} + q_2.q_1 = q_2.(\overline{a} + q_1)$.

Resulta sencillo pasar de las funciones de marcado y borrado $\mathbf{J_iK_i}$ a biestables tipo **D**, mediante la siguiente transformación funcional:

$$D_i = \overline{q_i}.J_i + q_i.\overline{K_i}\ ;$$

las condiciones de marcado actúan cuando la variable de estado se encuentra a **0** y las de borrado cuando se encuentra a **1**. La expresión anterior coincide, obviamente, con la forma de construir un biestable **JK** a partir de uno **D** (apartado 13.1, página 65).

En el caso que nos ocupa:

$$D_1 = \overline{q_1}.J_1 + q_1.\overline{K_1} = \overline{q_1}.a + q_1.a.q_2 = a.(q_2 + \overline{q_1})$$

$$D_2 = \overline{q_2}.J_2 + q_2.\overline{K_2} = \overline{q_2}.q_1.a + q_2.a.\overline{q_1} = a.(q_2 \oplus q_1)$$

que coinciden con las obtenidas directamente de la tabla funcional.

Este diseño directo a partir del grafo o de la máquina de estados conduce a funciones booleanas que pueden quedar menos simplificadas que en el caso de construir las tablas funcionales; además, tiene el riesgo de no considerar alguna situación particular que no se encuentre expresamente reflejada en el grafo de estados y que pudiera conducir a errores de funcionamiento. A costa de asumir este riesgo y una menor simplificación, se evita la pesada tarea de construir fila a fila la tabla funcional.

Se indica, a continuación, la forma de obtener las funciones de activación de las entradas J_i K_i desde la tabla de evolución del estado: será preciso ampliar dicha tabla con las condiciones de marcado J_i y borrado K_i de cada una de las variables de estado q_i, a partir de la siguiente correspondencia entre la evolución de q_i y los valores de J_i y de K_i.

$q_i(t)$	→	$q_i(t+1)$	⇒	J_i	K_i
0		0		0	X
0		1		1	X
1		0		X	1
1		1		X	0

lo cual permite expresar la tabla de verdad para las entradas J_i y K_i de los biestables y, a partir de ella, construir las funciones booleanas que deben actuar sobre dichas entradas:

$$J = f'(X, Q) \qquad K = f''(X, Q)$$

En la práctica no tiene mucho interés el diseño con biestables **JK** utilizando la tabla de evolución del estado, ya que es un proceso más complejo que para biestables tipo **D** y, además, es más habitual la disponibilidad de los biestables **D** (entre otras razones porque son más básicos y de circuitería más reducida y porque son los biestables típicos de los dispositivos programables y de las librerías de celdas estándar para el diseño de ASICs).

En el caso del detector de una secuencia de bits consistente en tres «unos» seguidos, la tabla funcional de evolución del estado, incluyendo las columnas correspondientes a la activación de las entradas **J** y **K**, es la representada en la página siguiente.

q_2	q_1	a	q_2^+	q_1^+	J_2 K_2	J_1 K_1
0	0	0	0	0	0 X	0 X
		1	0	1	0 X	1 X
0	1	0	0	0	0 X	X 1
		1	1	0	1 X	X 1
1	0	0	0	0	X 1	0 X
		1	1	1	X 0	1 X
1	1	0	0	0	X 1	X 1
		1	0	1	X 1	X 0

q_1:

$J_1 = a$

$K_1 = \bar{a} + \overline{q_2}$

q_2:

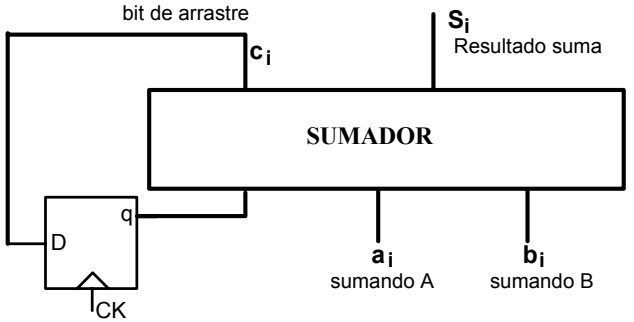
$J_2 = q_1.a$

$K_2 = \bar{a} + q_1$

14.2. Ejercicios de diseño de sistemas secuenciales síncronos

14.2.1. *Sumador secuencial: realiza la suma de dos números binarios, los cuales recibe «en serie», bit a bit, por dos líneas en unidades de tiempo sucesivas (comenzando por el dígito de menor valor significativo).*

Este sumador requiere memoria, por cuanto que el bit de arrastre o acarreo que se produce al sumar dos dígitos ha de actuar como operando (como entrada) en la suma de los siguientes dígitos; el correspondiente sistema secuencial ha de ser síncrono pues ha de adaptarse a las unidades de tiempo en las cuales recibe los sucesivos dígitos de los números a sumar.



Este sumador secuencial se reduce a una celda sumadora básica (apartado 3.1, primer volumen) a la que se añade un biestable **D** para almacenar el arrastre.

Las tablas funcionales de evolución del estado c_i (bit de acarreo) y de activación de la salida s_i (resultado de la suma) son las mismas que en la celda sumadora, sólo que aquí la entrada y la salida de acarreo corresponden a una misma variable de estado:

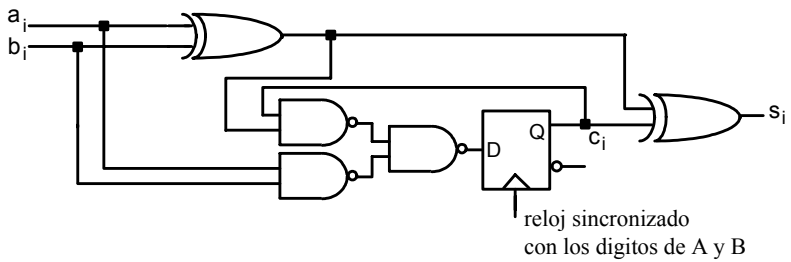
c_i	b_i	a_i	c_i^+	s_i
0	0	0	0	0
	0	1	0	1
	1	0	0	1
	1	1	1	0
1	0	0	0	1
	0	1	1	0
	1	0	1	0
	1	1	1	1

Resolviendo este diseño con un biestable **D**, para almacenar c_i, resulta:

variable de salida $\quad s_i = \bar{c}.(\bar{b}.a + b.\bar{a}) + c.(\overline{\bar{b}.\bar{a}} + b.a) = c \oplus (a \oplus b)$

variable de estado c_i: $\quad c^+ = b.a + c.\bar{b}.a + c.b.\bar{a} = b.a + (b \oplus a).c$

que dan como resultado el circuito de la figura siguiente.



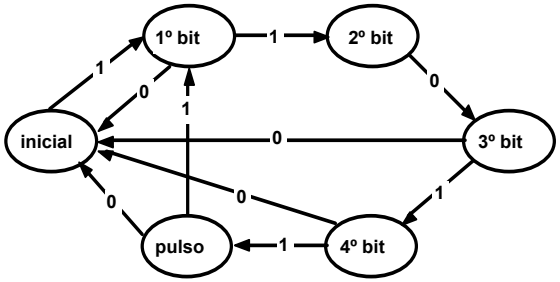
Sumador secuencial

14.2.2. *Detector de una secuencia de bits: un sistema con una sola entrada **A**, por la cual recibe dígitos binarios en unidades de tiempo sucesivas, responde con un pulso (de duración igual a una unidad de tiempo) cuando detecta la llegada de una determinada secuencia, por ejemplo, la secuencia **11011**.*

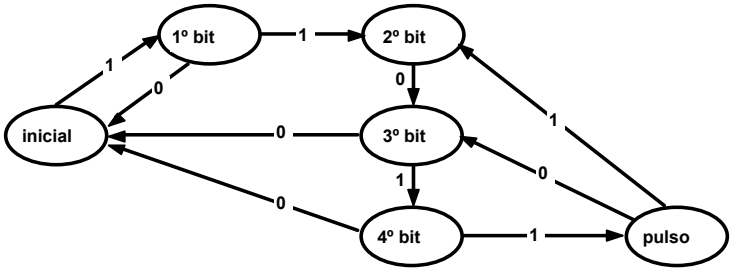
El problema de detección de una secuencia genérica (en este caso, la detección de **11011**) es diferente según se admita o no la posibilidad de solape (es decir, que la sucesión **11011011** produzca doble detección de secuencia, la primera correspondiente a los cinco primeros bits y la segunda relativa a los cinco últimos, dos de ellos solapados en ambas, o solamente produzca una detección).

Los grafos de estado correspondientes a ambas posibilidades serán los siguientes:

a) sin solape



b) con solape



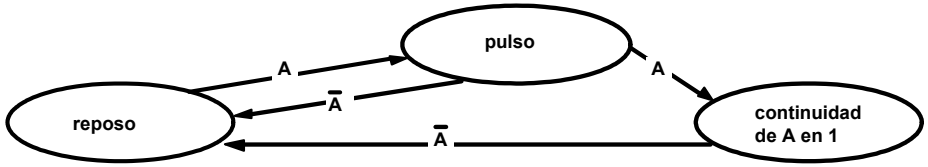
Ambos casos pueden ser codificados con tres variables de estado, su tabla de evolución del estado tendrá a cuatro variables de entrada (las de estado más la entrada **A**) y la salida se activará en el estado anotado como *pulso*.

Se deja para el lector (como ejercicio de aplicación) la codificación de los estados, la construcción de las tablas de evolución del estado y la obtención, a partir de dichas tablas, de las funciones de activación de los biestables D.

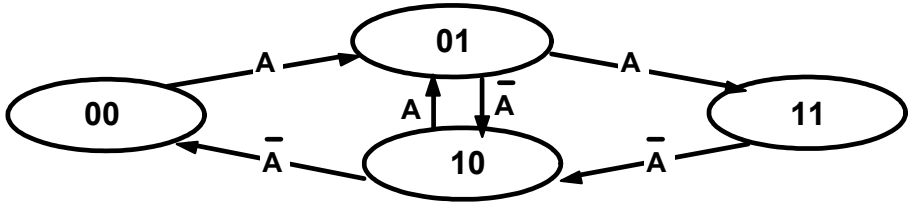
La detección con solape podría resolverse, también, con un registro de desplazamiento (y el término mínimo correspondiente a la secuencia a detectar), pero en este caso serían necesarios cinco biestables, frente a los tres que requiere el anterior grafo de estados.

14.2.3. *Detección síncrona de flancos de subida: dada una señal proveniente de un pulsador **A** (o, simplemente, dada cualquier señal digital **A**) se desea generar un pulso cuya duración sea igual a una unidad del tiempo de reloj, cada vez que se active el pulsador (cada vez que se produzca un flanco de subida en la señal **A**).*

Un posible grafo de estados de este sistema es el siguiente:



Son necesarias, al menos, dos variables de estado para codificar el grafo anterior; la introducción de un cuarto estado no complica en modo alguno dicho grafo y, en cambio, introduce una gran simetría en el mismo:



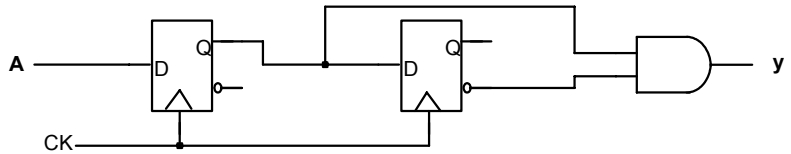
De la codificación del ciclo principal del grafo en código Gray resulta la siguiente tabla funcional:

q_2	q_1	A	q_2^+	q_1^+
0	0	0	0	0
		1	0	1
0	1	0	1	0
		1	1	1
1	0	0	0	0
		1	0	1
1	1	0	1	0
		1	1	1

que, realizada con biestables D, da lugar a un registro de desplazamiento de dos bits:

$D_1 = A$ $\qquad$ $D_2 = q_1$;

siendo la función de salida (estado **01**) la siguiente: $y = \overline{q_2}.q_1$.



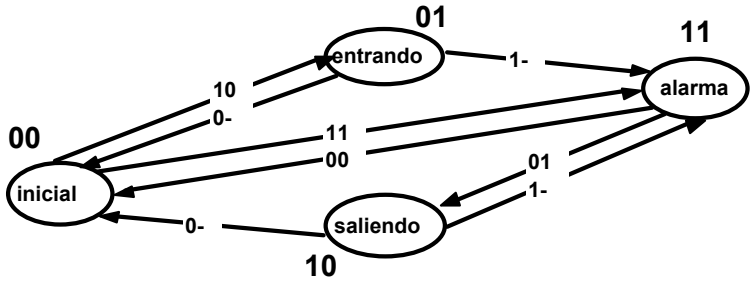
El resultado es obvio: para detectar un flanco de subida en la señal **A** basta retener el valor de dicha señal durante dos unidades de tiempo seguidas y seleccionar la secuencia **01** que corresponde a un flanco ascendente (paso de **0**, valor previo, a **1**, valor siguiente).

Este circuito es un detector de flanco de subida, un módulo muy útil para el diseño secuencial, que aparece frecuentemente en sus diagramas de bloques (ver apartado 16.4, *Contaje de pulsos diferentes del reloj*) .

El flanco de bajada se detecta con la secuencia **10**: $y = q_2.\overline{q_1}$, es decir con un circuito análogo, cambiando las salidas de los biestables que van a la puerta puerta "**y**"; para detectar ambos flancos (el de subida y el de bajada) será necesaria una puerta "**o-exclusiva**" que reciba las salidas de ambos biestables: $y = \overline{q_2}.q_1 + q_2.\overline{q_1} = q_2 \oplus q_1$.

14.2.4. *Activación gradual de alarma: un sistema de detección de temperatura proporciona cuatro niveles codificados en binario (**00**, **01**, **10**, **11**); la situación de alarma debe activarse cuando se detecta el nivel 3 **11** (temperatura muy alta), o si se detecta el nivel 2 **10** (alta) en dos ciclos seguidos de reloj y debe desaparecer cuando se detecta el nivel 0 **00** (muy baja), o si se detecta el nivel 1 **01** (baja) en dos ciclos de reloj consecutivos.*

El grafo de estados del sistema de activación de la situación de alarma puede ser el representado en la siguiente figura; su codificación, con dos variables de estado, se ha indicado al lado de cada estado.



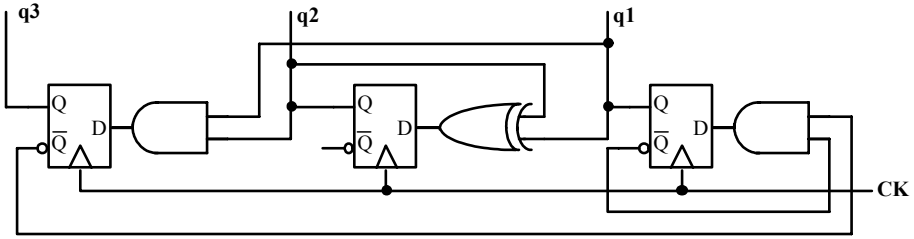
La situación de alarma corresponde a los estados **11** y **10**: alarma $= q_2$ y la tabla de evolución de estados será la siguiente:

q_2	q_1	b	a	q_2^+	q_1^+	
0	0	0	0	0	0	
		0	1	0	0	
		1	0	0	1	
		1	1	1	1	
0	1	0	0	0	0	
		0	1	0	0	
		1	0	1	1	
		1	1	1	1	
1	0	0	0	0	0	
		0	1	0	0	
		1	0	1	1	
		1	1	1	1	
1	1	0	0	0	0	$D_1 = b$
		0	1	1	0	
		1	0	1	1	$D_2 = b.(a + q_2 + q_1) + q_2.q_1.a$
		1	1	1	1	

14.2.5. *Contador síncrono módulo 5: su grafo tendrá 5 estados, comenzando en el estado 0, pasando de cada número al siguiente y del estado 4 al primer estado.*

q_3	q_2	q_1	q_3^+	q_2^+	q_1^+
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	X	X	X
1	1	0	X	X	X
1	1	1	X	X	X

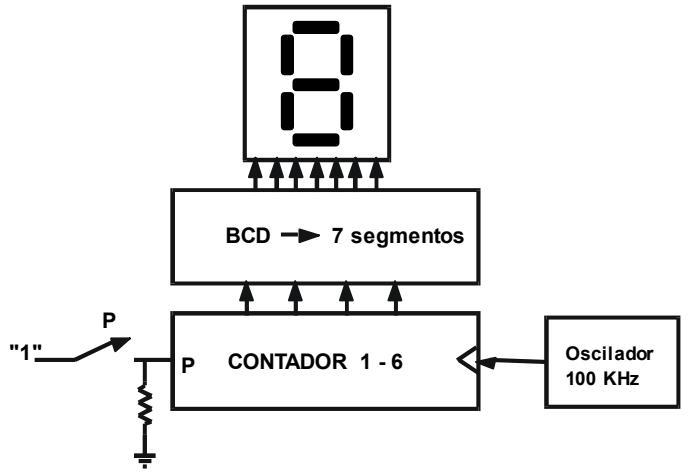
$D_1 = \overline{q_3}.\overline{q_1}$ $\quad D_2 = q_2.\overline{q_1} + \overline{q_2}.q_1 = q_2 \oplus q_1$ $\quad D_3 = q_2.q_1$



Contador síncrono módulo 5

14.2.6. *Dado electrónico: un contador especial 1 - 6, activado con una frecuencia muy alta (por ejemplo, 100 KHz) y de forma que cuente cuando se actúa sobre un pulsador **P** y deje de contar cuando se suelta dicho pulsador, dará como resultado un número entero entre 1 y 6, obtenido al azar (ya que el final del pulsado manual es realmente aleatorio respecto al contaje producido por una frecuencia muy alta).*

Una posible forma de seleccionar números al azar, entre el 1 y el 6 consiste en construir un sistema que cuente de 1 a 6 cíclicamente, con una entrada de habilitación, siendo los pulsos a contar de frecuencia relativamente alta; al pulsar la habilitación, el contador realizará el contaje muy rápidamente y el número en que se detenga al finalizar la habilitación será aleatorio.



La tabla funcional del contador habilitado por el pulsador **P** será la siguiente (en ella se ha hecho que el estado 0 pase siempre al 1 y que el estado 7 pase al 6, para evitar que el contador se sitúe en valores no pertinentes):

P	q_3	q_2	q_1	q_3^+	q_2^+	q_1^+
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	1	0	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	1	0
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	1
1	0	1	1	1	0	0
1	1	0	0	1	0	1
1	1	0	1	1	1	0
1	1	1	0	0	0	1
1	1	1	1	1	1	0

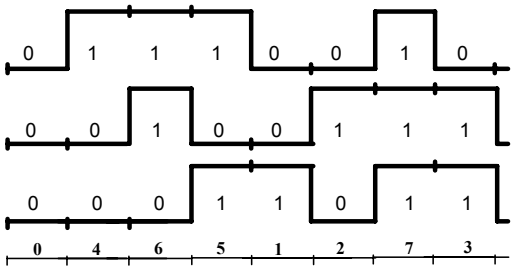
Este contador (de 1 a 6) puede programarse sobre un bloque PLS utilizando 3 módulos y una sola entrada **P** :

$D_1 = P.\overline{q_1} + \overline{q_3}.\overline{q_2}.\overline{q_1} + \overline{P}.\overline{q_3}.q_1 + \overline{P}.\overline{q_2}.q_1$

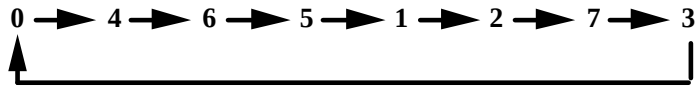
$D_2 = P.\overline{q_2}.q_1 + \overline{P}.q_2 + \overline{q_3}.q_2.\overline{q_1} + q_3.q_2.q_1$

$D_3 = P.q_2.q_1 + \overline{P}.q_3 + q_3.\overline{q_2}$

14.2.7. *Generador de múltiples ondas repetitivas, sincronizadas entre sí: a partir de una señal de reloj de frecuencia fija se desea generar las formas de onda representadas en la figura, las cuales se repetirán indefinidamente.*



La secuencia de valores booleanos correspondientes a estas ondas es : **0**, **4**, **6**, **5**, **1**, **2**, **7**, **3**,…, de manera que la generación de estas formas de onda corresponde a un contador cuya secuencia de contaje (grafo de estados) será:

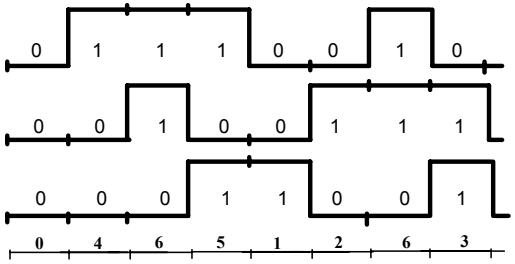


Su diseño requiere tres variables de estado (3 biestables), sin variables de entrada.

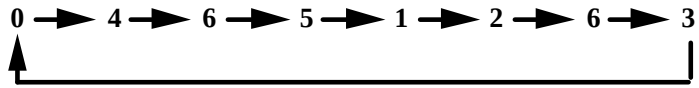
q_3	q_2	q_1	q_3^+	q_2^+	q_1^+
0	**0**	**0**	**1**	**0**	**0**
0	**0**	**1**	**0**	**1**	**0**
0	**1**	**0**	**1**	**1**	**1**
0	**1**	**1**	**0**	**0**	**0**
1	**0**	**0**	**1**	**1**	**0**
1	**0**	**1**	**0**	**0**	**1**
1	**1**	**0**	**1**	**0**	**1**
1	**1**	**1**	**0**	**1**	**1**

$D_1 = q_3.q_1 + q_2.\overline{q_1}$ $\quad D_2 = q_3 \oplus (q_2 \oplus q_1)$ $\quad D_3 = \overline{q_1}$

Si las formas de onda hubieran sido las de la figura siguiente:



la secuencia de contaje sería:



en la cual se repite el estado 6.

El código 6 se asigna a dos estados distintos (que presentan transiciones diferentes); será preciso diferenciar ambos estados, lo cual puede conseguirse con una variable adicional $\mathbf{q_4}$, cuyo valor sea indiferente **X** en todos los estados salvo en los dos que es preciso distinguir:

q_4	q_3	q_2	q_1	q_4^+	q_3^+	q_2^+	q_1^+
X	**0**	**0**	**0**	**X**	**1**	**0**	**0**
X	**0**	**0**	**1**	**X**	**0**	**1**	**0**
X	**0**	**1**	**0**	**1**	**1**	**1**	**0**
X	**0**	**1**	**1**	**X**	**0**	**0**	**0**
X	**1**	**0**	**0**	**0**	**1**	**1**	**0**
X	**1**	**0**	**1**	**X**	**0**	**0**	**1**
0	**1**	**1**	**0**	**X**	**1**	**0**	**1**
1	**1**	**1**	**0**	**X**	**0**	**1**	**1**
X	**1**	**1**	**1**	**X**	**X**	**X**	**X**

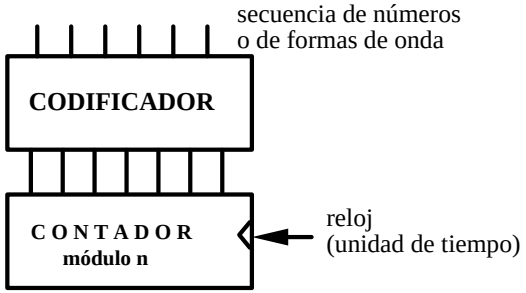
$D_1 = q_3 . (q_2 + q_1)$ $\quad D_2 = q_3 \oplus (q_2 \oplus q_1) + q_4.q_3.q_2$

$D_3 = (\overline{q_4} + \overline{q_3} + \overline{q_2}) . \overline{q_1}$ $\quad D_4 = q_2$

En los dos casos anteriores las correspondientes formas de onda (representadas en las figuras) aparecen en las salidas de los biestables; en el segundo caso, aparecen en las salidas de los tres últimos biestables, mientras que el biestable más significativo sirve simplemente para distinguir dos estados en los que el valor de los otros tres biestables es idéntico.

Esta forma de diseño supone que las formas de onda o, lo que es lo mismo, la secuencia de números binarios a obtener se produce directamente sobre las salidas de los biestables, es decir, las salidas coinciden con las variables de estado.

Otra manera, igualmente válida, de resolver el diseño de una secuencia numérica cualquiera, de longitud **n**, que se repite indefinidamente (**n** pasos de reloj), consiste en utilizar un contador módulo **n**, seguido de un codificador (conversor de código) que transforme los valores numéricos sucesivos del contador (**0**, **1**, **2**, **3**, …, **n-1**) en los códigos correlativos de la secuencia deseada.

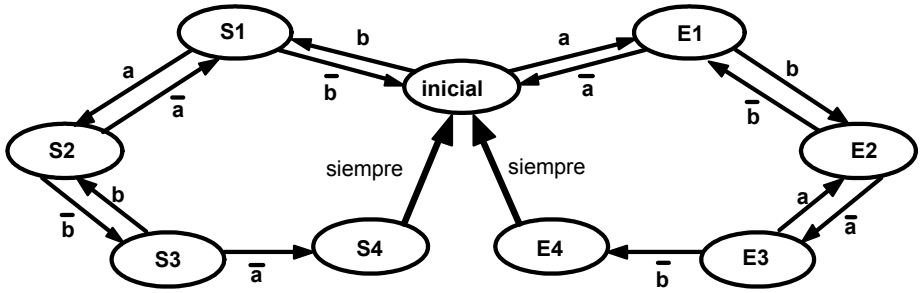


Supongamos que programamos sobre un bloque **PLS** el generador de ondas repetitivas diseñado en la página anterior: se necesitan 4 módulos para las 4 variables de estado **q_4 q_3 q_2 q_1** (de las cuales, las tres últimas coinciden con las variables de salida); pero, si la programación corresponde a la configuración anterior contador-codificador, el número de módulos necesarios es de 6 (3 para el contador de 8 estados y 3 para el codificador que proporciona las salidas).

14.2.8. *Detector de número de personas presentes: supuesto que exista un solo acceso que es, a la vez, entrada y salida y que las personas entran y salen de una en una y con una mínima distancia entre ellas, un par de células fotoeléctricas próximas **a** y **b** permite detectar la salida o entrada de personas y el sentido en que cruzan dicho acceso.*

Siendo **a** la célula fotoeléctrica exterior, las secuencias normales de las situaciones de entrada y de salida son, respectivamente:

- entrada: $\bar{a}.\bar{b}$ $a.\bar{b}$ $a.b$ $\bar{a}.b$ $\bar{a}.\bar{b}$
- salida: $\bar{a}.\bar{b}$ $\bar{a}.b$ $a.b$ $a.\bar{b}$ $\bar{a}.\bar{b}$



La figura anterior representa un primer grafo de la evolución de los estados de este sistema secuencial; los estados **E4** y **S4** se presentan durante una sola unidad de tiempo y corresponden a las salidas: incrementar y decrementar en una persona, respectivamente.

Supuesto que el contaje de las personas presentes se efectúe con un contador bidireccional, al estado **E4** le corresponde contar una unidad y al estado **S4** le corresponde descontar una unidad:

- habilitación del contador (estados **E4** y **S4**): $E = q_2.\overline{q_1}$
- contaje ascendente (estado **E4**): "control ascendente/descendente" = $\overline{q_3}$.

Habida cuenta de que el número de estados es alto, conviene considerar la posibilidad de que algunos de ellos sean simplificables entre sí:

1.1 estados agrupables: los estados inicial, E1, E2 y E3

1.2 variables que los diferencian: **a** y **b** : **00**, **10**, **11**, **01** respectivamente

1.3 posibilidad de diferenciar correctamente las transiciones desde el nuevo estado:

La transición con $\bar{b}$ debe producirse desde el estado **E3** hacia **E4** : $\bar{b}.\bar{a}.b$ (= **0**)**;** no es posible distinguir correctamente esta transición (una de las variables que diferencian al estado **E3** es la que produce la transición desde dicho estado) y, por tanto, no es posible efectuar esta agrupación de estados.

Lo mismo sucede en la transición con $\bar{a}$ desde el estado **E1** hacia el inicial.

Consideremos otras posibilidades de agrupación de estados:

2.1 estados agrupables: los estados **E1** y **E2**

2.2 variable que los diferencia: **b** : **0**, **1**, respectivamente

2.3 posibilidad de diferenciar correctamente las transiciones desde el nuevo estado:

Una transición con $\bar{a}$ debe producirse desde el estado **E1** hacia el inicial : $\bar{a}.\bar{b}$

Asimismo, debe producirse otra transición con $\bar{a}$ desde el estado **E2** hacia **E3** : $\bar{a}.b$

El nuevo estado agrupado tendrá una transición de salida $\bar{a}.\bar{b}$ hacia el estado inicial y otra transición $\bar{a}.b$ hacia el estado **E3**.

3.1 estados agrupables: los estados **S1** y **S2**

3.2 variable que los diferencia: **a** : **0**, **1**, respectivamente

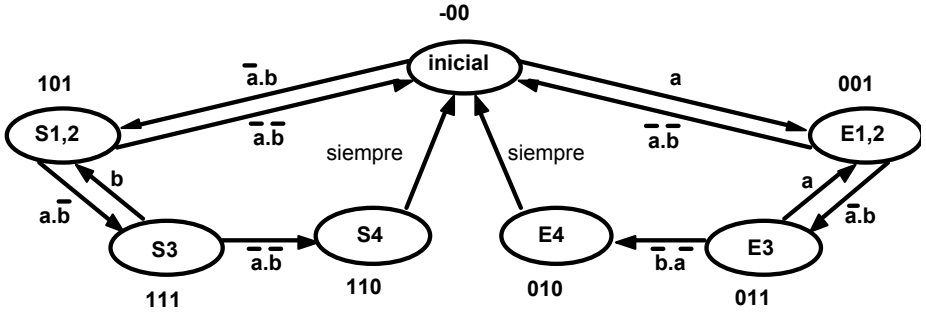
3.3 posibilidad de diferenciar correctamente las transiciones desde el nuevo estado:

Una transición con $\bar{b}$ debe producirse desde el estado **S1** hacia el inicial : $\bar{b}.\bar{a}$

También debe producirse otra transición con $\bar{b}$ desde el estado **S2** hacia **S3** : $\bar{b}.a$

El nuevo estado agrupado tendrá una transición de salida $\bar{b}.\bar{a}$ hacia el estado inicial y otra transición $\bar{b}.a$ hacia el estado **S3**.

La agrupación de los estados **E1** y **E2** en uno solo **E1,2** y la simétrica de **S1** y **S2** en **S1,2** reduce el grafo a siete estados, codificables con tres variables $q_3\,q_2\,q_1$.



Respetando la simetría del grafo, el estado inicial se ha codificado con **-00** (es indiferente para dicho estado el valor q_3) y, para evitar marcados duplicados, se ha condicionado con $\bar{a}$ la transición que produce b desde dicho estado; lo mismo se ha hecho (evitar la posibilidad de marcados duplicados) con las transiciones que salen de los estados **E3** y **S3** ($\bar{b}.\bar{a}$).

Extrayendo las condiciones de marcado y de borrado del grafo de estados:

q_1: $J_1 = \overline{q_2}.\overline{q_1}.(b + a)$ $\qquad K_1 = q_1.\bar{b}.\bar{a}$

q_2: $J_2 = \overline{q_2}.q_1.(\overline{q_3}.b.\bar{a} + q_3.\bar{b}.a)$ $\qquad K_2 = q_2.\overline{q_1} + q_2.(q_3.b + \overline{q_3}.a)$

q_3: $J_3 = \overline{q_2}.\overline{q_1}.b.\bar{a}$ $\qquad K_3 = \overline{q_2}.\overline{q_1}.a$

Para construir este sistema secuencial biestables **D**, aplicaremos la transformación: $D = \bar{q}.J + q.\bar{K}$

$$D_1 = (\overline{q_2} + q1) \,.\, (b + a)$$

$$D_2 = \overline{q_2}.q_1.(q_3.\bar{b}.a + \overline{q_3}.b.\bar{a}) + q_2.q_1.(q_3.\bar{b} + \overline{q_3}.\bar{a} + \bar{b}.\bar{a})$$

$$D_3 = \overline{q_3}.\overline{q_2}.\overline{q_1}.b.\bar{a} + q_3.(q_2 + q_1 + \bar{a})$$

En este caso la tabla de evolución de estados presenta 5 variables (es decir, es amplia pero manejable); construyendo dicha tabla se obtienen las siguientes funciones (bastante más simplificadas que las anteriores):

$$D_1 = (\overline{q_2} + q1) \,.\, (b + a)$$

$$D_2 = q_1.(q_3.\bar{b}.a + \overline{q_3}.b.\bar{a}) + q_2.q_1.\bar{b}.\bar{a}$$

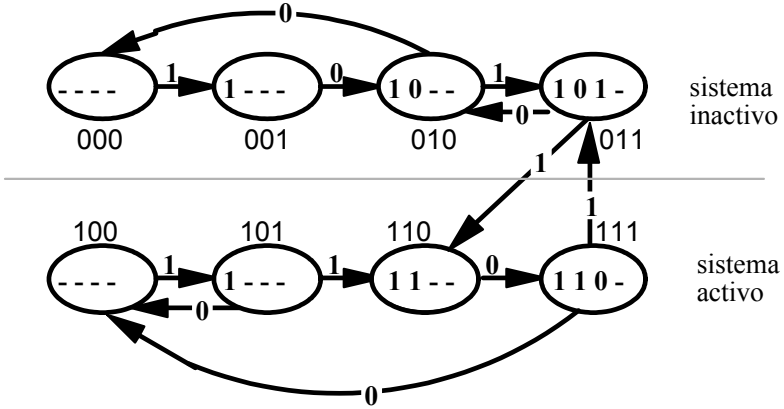
$$D_3 = \overline{q_1}.\bar{a} + q_3.q_1$$

14.2.9. *Activación y desconexión de un sistema por detección de secuencia: un sistema eléctrico es controlado a través de una línea **a** por la que recibe valores booleanos sucesivos, sincronizados con un reloj de 1 MHz; el sistema se activa al recibir la secuencia **1011** y se desactiva cuando detecta **1101** (se admite el solapamiento entre secuencias. Diseñar el circuito de activación/desactivación, de forma que pueda ser programado sobre un circuito **PLS** y la salida (activo = 1, desactivo = 0) se corresponda con una de las variables de estado.*

La cuestión no consiste en detectar dos secuencias **1011** y **1101**, sino en diferenciar dos situaciones activo/inactivo: en la situación de sistema inactivo interesa detectar la secuencia **1011** que lo activa y cuando el sistema se encuentra activo importa detectar la otra secuencia **1101** que lo lleva a inactivo.

Habida cuenta que se admite el solape entre las secuencias a detectar, cuando el sistema se activa se encuentran ya detectados los dos primeros dígitos de la secuencia que lo desactiva **1011** ≡ **11--**; del mismo modo, al pasar a inactivo se encuentran detectados los tres primeros dígitos de la secuencia que lo puede activar **1101** ≡ **101-**.

Un posible grafo de estados es el siguiente, en cuya codificación se ha reservado la variable q_3 para representar la situación de sistema activo $q_3 = 1$ o inactivo $q_3 = 0$:



La función de salida, **y** = sistema activo, corresponde a la variable de estado q_3 **y = q_3** y la tabla de evolución de los estados es la representada en la página siguiente.

q_3	q_2	q_1	a	q_3^+	q_2^+	q_1^+
0	0	0	0	0	0	0
			1	0	0	1
0	0	1	0	0	1	0
			1	0	0	1
0	1	0	0	0	0	0
			1	0	1	1
0	1	1	0	0	1	0
			1	1	1	0
1	0	0	0	1	0	0
			1	1	0	1
1	0	1	0	1	0	0
			1	1	1	0
1	1	0	0	1	1	1
			1	1	1	0
1	1	1	0	1	0	0
			1	0	1	1

A partir de esta tabla las funciones de activación de los biestables $D_i = q_i^+$ serán las siguientes:

$$D_1 = q_3.q_2.q_1.a + q_3.q_2.\overline{q_1}.\overline{a} + \overline{q_3}.\overline{q_2}.a + \overline{q_3}.\overline{q_1}.a + \overline{q_2}.\overline{q_1}.a$$

$$D_2 = q_3.q_2.\overline{q_1} + q_3.q_1.a + \overline{q_3}.q_1.\overline{a} + q_2.a$$

$$D_3 = q_3.\overline{q_2} + q_3.\overline{q_1} + q_3.\overline{a} + \overline{q_3}.q_2.q_1.a$$

Un diseño alternativo consiste en configurar un registro de desplazamiento de 4 biestables **q_4 q_3 q_1 q_1** para recibir la secuencia, detectándola sobre las salidas de dicho registro y utilizar un biestable adicional **q_5** que se marque con la secuencia **1011** y se borre con la otra secuencia **1101**:

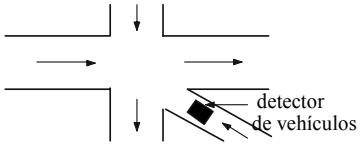
$$D_1 = a \qquad D_2 = q_1 \qquad D_3 = q_2 \qquad D_4 = q_3$$

$$D_5 = \text{marcado} + q_5.\overline{\text{borrado}} = q_4.\overline{q_3}.q_2.q_1 + q_5.\overline{(q_4.q_3.\overline{q_2}.q_1)} =$$

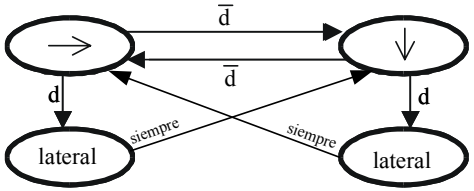
$$= q_4.\overline{q_3}.q_2.q_1 + q_5.\overline{q_4} + q_5.\overline{q_3} + q_5.q_2 + q_5.\overline{q_1}$$

Este diseño requiere 5 módulos **PLS** (uno de ellos con, al menos, 5 términos producto), mientras que el diseño anterior (como sistema secuencial directamente a partir del grafo de estados) solamente necesita 3 módulos **PLS**.

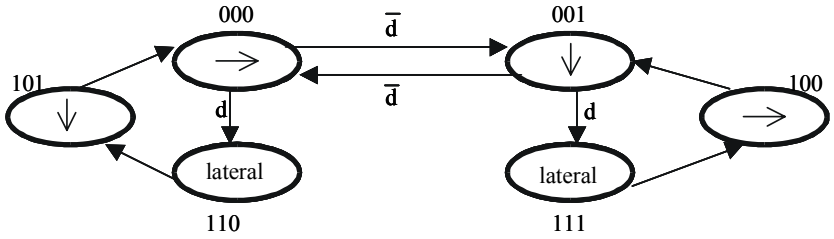
14.2.10 *Un sencillo cruce de dos calles perpendiculares y unidireccionales dispone de semáforos (con sólo rojo y verde) que, cada 20" dan paso, alternativamente, a cada una de las dos direcciones; sobre dicho cruce incide una pequeña calle lateral (ver figura) que dispone de un sensor de vehículos **d**, de forma que el paso de dicha calle está normalmente interrumpido, pero, cuando se detecta un vehículo en ella, se espera a finalizar el intervalo de 20" y se da paso a dicha dirección lateral (interrumpiendo las otras dos), también durante 20". Diseñar el circuito de control de los semáforos de las tres calles.*



Como los tiempos de los semáforos son todos ellos de 20 segundos, dicha temporización se resuelve directamente mediante un reloj de 20" de período (frecuencia 0,05 Hz). Un posible grafo de estados es el siguiente, siendo **d** la entrada conectada al sensor de vehículos:



Este grafo asigna prioridad a la calle lateral, de forma que cuando hay mucha demanda de paso por la misma, se produce un verde en la calle lateral después de cada paso en una de las otras direcciones. Parece más adecuado asegurar el paso de las otras dos direcciones entre cada dos verdes de la calle lateral, para lo cual sirve el grafo de estados siguiente (todas las transiciones se producen con los flancos activos del reloj, es decir, cada 20" y las transiciones no anotadas se producen siempre).



semáforo calle horizontal: verde = $\overline{q_2}.\overline{q_1}$

semáforo calle vertical: verde = $\overline{q_2}.q_1$

semáforo calle lateral: verde = $q_3.q_2$

La tabla de evolución de los estados es la siguiente.

q_3	q_2	q_1	d	q_3^+	q_2^+	q_1^+
0	0	0	0	0	0	1
			1	1	1	0
0	0	1	0	0	0	0
			1	1	1	1
0	1	0	0	X	X	X
			1	X	X	X
0	1	1	0	X	X	X
			1	X	X	X
1	0	0	0	0	0	1
			1	0	0	1
1	0	1	0	0	0	0
			1	0	0	0
1	1	0	0	1	0	1
			1	1	0	1
1	1	1	0	1	0	0
			1	1	0	0

A partir de esta tabla las funciones de activación de los biestables serán las siguientes:

$$D_1 = q_3.\overline{q_1} + \overline{q_3}.q_1.d + \overline{q_1}.\overline{d}$$

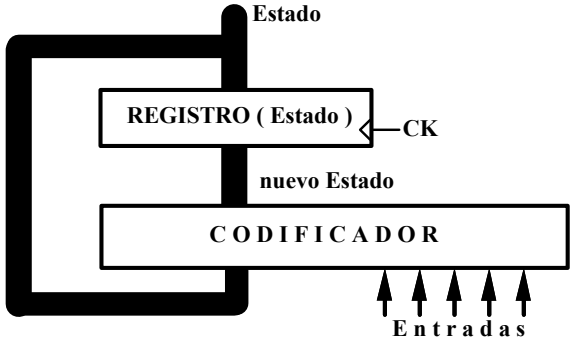
$$D_2 = \overline{q_3}.\overline{q_2}.d$$

$$D_3 = q_2 + \overline{q_3}.d$$

14.3. Configuración microprogramada de grafos de estado

El diseño de un grafo o máquina de estados puede ser abordado en dos formas conceptualmente muy diferentes:

- según el método aplicado en los apartados anteriores, que trata cada variable de estado por separado, reflejando las transiciones en condiciones de activación de sus biestables individuales –forma microcableada–;
- recogiendo el estado sobre un registro global y codificando los cambios de estado a partir del estado anterior y del vector de entradas, mediante un codificador que calcula el nuevo estado –forma microprogramada–.



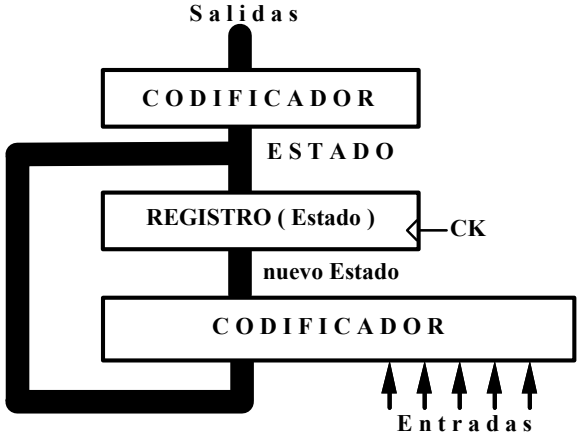
La configuración ROM de los codificadores permite construirlos directamente desde su tabla funcional, sin necesidad de obtener la expresión algebraica de sus funciones: bastará trasladar la tabla de evolución de los estados a la «Matriz O» del codificador; podemos considerar que este codificador contiene «microinstrucciones» relativas al cálculo del estado siguiente, «numeradas» por el estado anterior y el vector de entrada.

La forma microprogramada conserva en forma explícita la estructura del grafo de estados, pues el codificador expresa directamente la correspondencia entre la situación actual (estado y vector de entrada) y el nuevo estado; por ello, resulta fácil efectuar modificaciones de esta máquina de estados, cambiando las correspondientes «microinstrucciones» (la programación) del codificador.

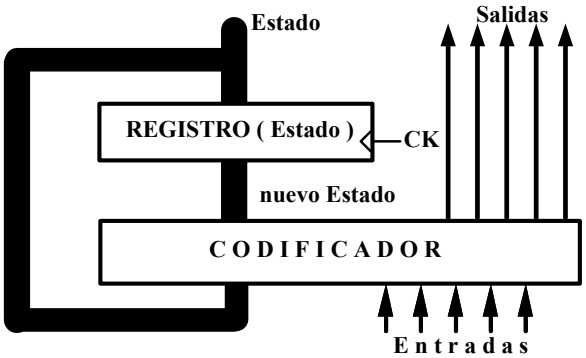
La «programación» del codificador (es decir, su Matriz O) corresponde a la tabla de evolución de estados; en tal sentido, consideremos la configuración «microprogramada» de los ejercicios detallados en el apartado anterior de este mismo capítulo:

- *14.2.4*: su tabla de evolución de estados se encuentra en la página 90, requiere 2 variables de estado y 2 de entrada: el codificador tendrá 4 entradas y 2 salidas;
- *14.2.5*: (pág. 91) 3 variables de estado: codificador de 3 entradas y 3 salidas;
- *14.2.6*: (pág. 92) 3 variables estado y 1 entrada: codificador de 4 entradas y 3 salidas;
- *14.2.7*: (pág. 93) 3 variables de estado: codificador de 3 entradas y 3 salidas;
- *14.2.8*: (pág. 97) 3 variables estado y 2 entradas: codificador de 5 entradas y 2 salidas;
- *14.2.9*: (pág. 98) 3 variables estado y 1 entrada: codificador de 4 entradas y 3 salidas.

Respecto al vector de salida, en el caso de autómata de Moore un segundo codificador puede calcular las salidas a partir del estado (en muchas ocasiones las salidas serán variables de estado y no es necesario el codificador); podemos considerar que este codificador contiene las «microinstrucciones» referidas a las salidas que corresponden a cada estado:



En los autómatas de Mealy, como las salidas dependen del estado y de las entradas, el mismo codificador puede calcular el nuevo estado y el vector de salidas:



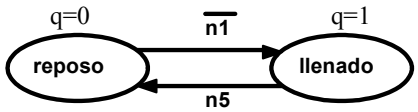
En estas configuraciones «microprogramadas» el tamaño de los codificadores aumenta fuertemente con el número de sus variables de entrada: se duplica cada nueva entrada (dependencia según 2^m); de ahí el interés en reducir el número de variables de estado (codificaciones de «un solo uno» resultan altamente desaconsejables) y, cuando ello sea posible, utilizar técnicas de multiplexado de las variables del vector de entrada (suele suceder que no todas las entradas actúan en todos los estados, de forma que el número efectivo de entradas puede reducirse por multiplexado).

14.4. Los grafos de estado en VHDL

Como ejemplo de descripción VHDL de grafos de estado se incluye la correspondiente al control de llenado de un depósito con una mezcla de cuatro líquidos diferentes (apartado 11.4, páginas 29-31); se detalla su diseño síncrono en las dos formas de autómata de Mealy y autómata de Moore.

Un depósito se llena con una mezcla de cuatro líquidos, para lo cual dispone de cuatro electroválvulas ***A, B, C, D,*** *y de cinco detectores de nivel* n_1, n_2, n_3, n_4 *y* n_5*; cuando el nivel del depósito desciende por debajo del mínimo* n_1 *se produce un ciclo de llenado: primero A hasta el nivel* n_2*, luego B hasta* n_3*, C hasta* n_4 *y, finalmente, D* n_5*.*

Autómata de Mealy

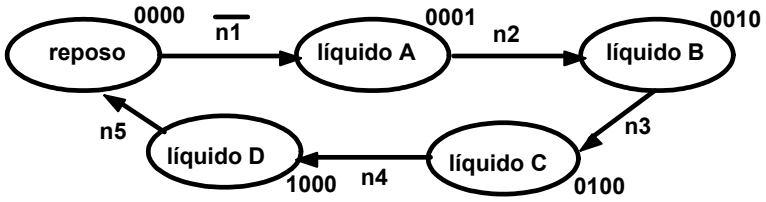



```
constant reposo      : bit := '0';
constant llenado     : bit := '1';

signal estado        : bit ;

process
begin
wait until CK = '1';
if ( Reset = '1' ) then                          estado <= reposo;
  else     case estado is
                  when reposo =>
                        if (n1 = '0') then       estado <= llenado;      end if;
                  when llenado =>
                        if (n5 = '1') then       estado <= reposo;       end if;
           end case;
end if;
end process;
```

(téngase en cuenta que *proceso* conserva el estado cuando no se indica lo contrario: por ello no es necesario añadir *else estado <= estado;*)

```
--funciones de activación de las salidas:
A <= '1' when (estado = llenado) and (n2='0') else '0';
B <= '1' when (estado = llenado) and (n2='1') and (n3 ='0') else '0';
C <= '1' when (estado = llenado) and (n3='1') and (n4 ='0') else '0';
D <= '1' when (estado = llenado) and (n4='1') else '0';
```

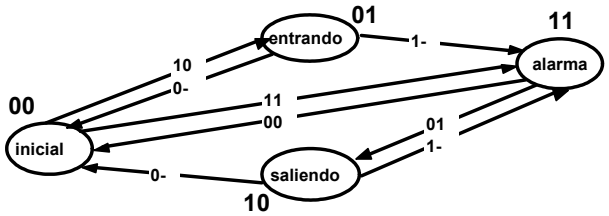
Autómata de Moore




```
constant reposo          : bit_vector_(4 downto 1):="0000";
constant líquido_A       : bit_vector_(4 downto 1):="0001";
constant líquido_B       : bit_vector_(4 downto 1):="0010";
constant líquido_C       : bit_vector_(4 downto 1):="0100";
constant líquido_C       : bit_vector_(4 downto 1):="1000";

signal estado            : bit_vector_(4 downto 1);

process
begin
wait until CK = '1';
if ( Reset = '1' ) then                                  estado <= reposo;
  else case estado is
          when reposo =>       if (n1 = '0')   then   estado <= líquido_A;   end if;
          when líquido_A =>    if (n2 = '1')   then   estado <= líquido_B;   end if;
          when líquido_B =>    if (n3 = '1')   then   estado <= líquido_C;   end if;
          when líquido_C =>    if (n4 = '1')   then   estado <= líquido_D;   end if;
          when líquido_D =>    if (n5 = '1')   then   estado <= reposo;      end if;
          when others =>
       end case;
end if;
end process;

--funciones de activación de las salidas:
A <= '1' when estado = líquido_A else '0';
B <= '1' when estado = líquido_B else '0';
C <= '1' when estado = líquido_C else '0';
D <= '1' when estado = líquido_D else '0';
```


El ejemplo anterior solamente presenta una transición desde cada estado y en cada una de ellas actúa solamente una variable de entrada; el siguiente grafo, referido a la activación gradual de alarma (apartado 14.2.4, página 88), incluye mayor número de transiciones entre estados y dos variables de entrada involucradas en ellas.

*Un sistema de detección de temperatura con cuatro niveles (**00**, **01**, **10**, **11**); la alarma debe activarse cuando se detecta **11** (temperatura muy alta), o si se detecta el nivel **10** (alta) en dos ciclos seguidos de reloj y debe desaparecer cuando se detecta **00** (muy baja), o si se detecta el nivel **01** (baja) en dos ciclos de reloj.*




```
constant inicial         : bit_vector_(2 downto 1):="00";
constant entrando        : bit_vector_(2 downto 1):="01";
constant saliendo        : bit_vector_(2 downto 1):="10";
constant alarma          : bit_vector_(2 downto 1):="11";

signal estado            : bit_vector_(2 downto 1);

process
begin
wait until CK = '1';
if ( Reset = '1' ) then                                          estado <= reposo;
  else case estado is
          when inicial =>     if (entrada = "10")          then   estado <= entrando;
                                 elsif (entrada = "11")    then   estado <= alarma;  end if;
          when entrando =>    if (entrada(2) = '1')        then   estado <= alarma;
                                 elsif (entrada(2) = '0')  then   estado <= inicial;  end if;
          when saliendo =>    if (entrada(2) = '1')        then   estado <= alarma;
                                 elsif (entrada(2) = '0')  then   estado <= inicial;  end if;
          when alarma =>      if (entrada = "00")          then   estado <= inicial;
                                 elsif (entrada = "01")    then   estado <= saliendo; end if;
       end case;
end process;

--función de activación de la salida:
y <= '1' when (estado = alarma) or (estado = saliendo) else '0';
```