MÁQUINAS ALGORÍTMICAS COMO OPCIÓN DIDÁCTICA DE SISTEMAS DIGITALES COMPLEJOS

T. POLLÁN, B. MARTÍN y J. PONCE DE LEÓN
Escuela Universitaria de Ingeniería Técnica Industrial de Zaragoza.
Departamento de Ingeniería Electrónica y Comunicaciones. Universidad de Zaragoza. España

Las máquinas algorítmicas son sistemas digitales con separación entre la parte de control y la parte operativa y expresión de dicho control mediante un grafo de estados. Se proponen las máquinas algorítmicas como ejemplos útiles de diseño digital de complejidad viable, es decir, son sistemas de relativa complejidad pero abordables como ejercicios o trabajos de asignatura (más aún, si su descripción se hace a través de un lenguaje circuital, como puede ser el VHDL).

1. Introducción

Es frecuente que, en congresos o reuniones sobre metodología docente, los profesores de electrónica digital nos recordemos la necesidad de que la enseñanza de esta materia no quede limitada al conocimiento de los bloques o "piezas" de diseño y al manejo de pequeños diagramas de bloques o a la descripción de sistemas muy simples. Entendemos que debemos aproximar al alumno al diseño de sistemas complejos, que son los sistemas digitales de autentico interés y utilidad hoy día.

Ciertamente, la disponibilidad de lenguajes de descripción circuital (en particular, VHDL y Verilog) ha permitido que, una vez conocidas las "piezas" de diseño digital (funciones booleanas, bloques combinacionales, grafos de estado, registros, contadores, …) podamos abordar el diseño y simulación de sistemas de diferentes grados de complejidad; además, disponemos de amplios dispositivos programables (CPLDs y FPGAs), relativamente baratos, donde implementar tales diseños.

Para ello, necesitamos enunciados de sistemas digitales que sean, a la vez, razonablemente complejos y adecuados como ejercicios de asignatura. Dentro de esta categoría, con características didácticas muy positivas, estimamos que resultan útiles y apropiados sistemas que pueden plantearse como máquinas algorítmicas, con separación entre parte de control y parte operativa y descripción del control en forma de algoritmo, representable en un grafo de estados [1].

2. Separación entre control y operaciones

La división en parte operativa y parte de control permite tratar por separado ambas partes y separar, también, cada uno de los recursos de cálculo de la parte operativa. De forma que al diseñar los recursos de cálculo no es preciso considerar el orden con que las operaciones se ejecutan, ni el número de veces que se repite cada operación, ni los necesarios sincronismos entre operaciones y en la transferencia de datos. Asimismo, al diseñar la parte de control se evita el considerar la configuración en detalle de los bloques que efectúan las operaciones.

Un esquema apropiado para este tipo de diseño puede ser el siguiente:

1. Idear el método de operación: secuencia de operaciones a realizar y forma de efectuarlas.
2. Describir dicho esquema de calculo mediante un algoritmo.
3. Identificar los recursos operativos necesarios (la parte operativa).
4. Expresar el control en un grafo de estados (o un «ordinograma»).
5. Abordar la descripción circuital por separado del grafo de estados de control y de los diversos operadores y registros que conforman la parte operativa. En la parte de control se describirá, de un lado, la evolución de los estados y, de otro, la «activación de las salidas»; si los recursos de cálculo son simples pueden incluirse directamente en este segundo apartado, dentro de un listado («case») de estados y acciones asociadas a los mismos.

3. Ejemplo de una máquina algorítmica: división de números binarios

Como primer ejemplo de máquina algorítmica, realizable como ejercicio de asignatura, presentamos la división de un número binario de 64 dígitos por otro de 16 dígitos, con el siguiente esquema de cálculo: recorrer el dividendo bit a bit (desde el más significativo), restando el divisor cuando se pueda; en cada desplazamiento, añadir un **1** al resultado si se efectúa la resta y un **0** cuando ésta no es posible.

Este esquema se deduce directamente del empleado para hacer «a mano» la división en binario:

- se toman los primeros dígitos del dividendo (los más significativos), de forma que correspondan a un número mayor o igual que el divisor, se les resta el divisor y se pone un **1** en el resultado;
- a partir de aquí, se toma el resto resultante y se le añade (se «baja») un nuevo dígito del dividendo y, si es posible, se le resta el divisor y se añade un **1** al resultado y, caso de que la resta no sea factible, se añade un **0** al resultado;
- y se repite, sucesivamente, el anterior proceso de «bajar» un nuevo dígito del dividendo y restar, si es posible, hasta completar el recorrido del mismo.

Se utilizan dos registros auxiliares: un contador **C** para expresar el número de veces que se ha realizado el ciclo y un registro **S**, «resto parcial» o «dividendo actual», que recoge el número binario que, en cada momento, se compara con el divisor; dicho registro **S** se conecta con el **P**, formando un registro de desplazamiento (hacia la izquierda) **S-P**.

Inicio del algoritmo
Leer **P** y **Q;** Borrar **R, S** y **C;**
Repetir
    Desplazar hacia la izquierda **S-P;**
    Si S ≥ Q Restar **S - Q** sobre **S**;
        Desplazar **R** hacia la izquierda con entrada **1**;
        Desplazar hacia la izquierda **S-P;** Incrementar **C** en una unidad**;**
    Si no Desplazar **R** hacia la izquierda con entrada **0**;
        Desplazar hacia la izquierda **S-P;** Incrementar **C** en una unidad**;**
    Fin_de_Si;
    Si C = 64 (64 ejecuciones de la repetición)
        Fin del algoritmo.
    Fin_de_Si;
Fin_de_Repetir (volver al inicio de la repetición)**;**

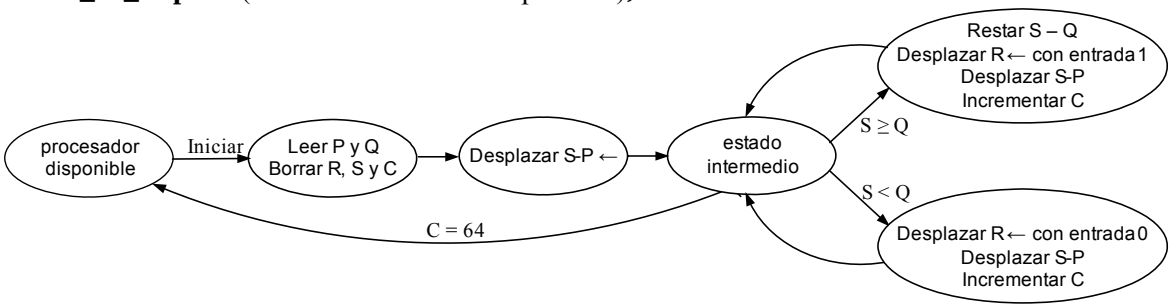

Figura 1. Grafo de estados de control de la división de un número de 64 bits por otro de 16 bits.

Cada estado de este grafo (excepto el inicial de *procesador disponible*) dura un solo ciclo de reloj y el número total de ciclos necesarios para efectuar la división es de 131 (1 + 1 + 64x2 + 1). Conviene destacar la necesidad de introducir un *estado intermedio* para que la acción *desplazar S-P* se ejecute antes de llegar al salto condicional (comparación de S con Q) que depende de dicha acción y, también, el *incremento de C* se efectúe antes de su comparación con 64.

Las acciones propias de un estado, si son de tipo síncrono (y lo serán la mayoría de ellas), no se ejecutan en el ciclo de reloj que corresponde a ese estado, sino cuando llega el flanco de reloj y el estado pasa al siguiente. Un salto condicional que dependa de acciones síncronas propias del estado anterior no se encontrará con el valor derivado de la aplicación de tales acciones, sino con el valor anterior a las mismas, dando lugar a una transición errónea; el *estado intermedio* permite la ejecución de tales acciones, actualizando los valores relativos a las condiciones antes de abordar el salto.

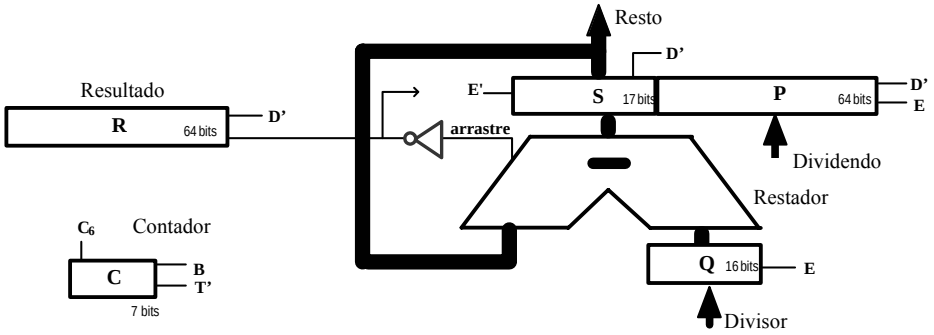

Figura 2. Parte operativa de la división de un número de 64 bits por otro de 16 bits.

El registro **S** unido al **P**, como registro de desplazamiento **S-P**, permite ir añadiendo al «resto parcial» o «dividendo actual» **S** los dígitos del número **P** uno a uno; la entrada de control **D'** se refiere a desplazamiento hacia la izquierda. La salida de arrastre del restador (acarreo **carry**) sirve para comparar **S** (el «resto parcial») con el divisor **Q**: cuando dicha salida es **0** (**S ≥ Q**) se ejecuta la resta (se habilita el resistro **S**) y se carga un **1** (por desplazamiento) en el resultado; cuando es **1** (**S < Q**) no se efectúa la resta (habilitación del resistro **S** a **0**) y en el resultado se carga un **0**.

Un ejemplo del mismo tipo es el referido a la multiplicación de dos números binarios de 64 bits. El producto de dos números binarios puede hacerse multiplicando el primer número **P** por cada dígito del segundo **Q** (comenzando por el menos significativo), lo cual da como resultado **P** cuando el bit de **Q** vale **1** y **0** si dicho bit es nulo; los productos parciales así obtenidos se suman, tras efectuar el correspondiente desplazamiento para ajustarlos a su valor relativo.

Inicio del algoritmo
Leer **P** y **Q;** Borrar **R** y **C;**
Repetir
Si bit_0 (unidades) de **Q = 1**
Sumar **P + R'** sobre **R'**; (véase en la figura 3 el registro R')
Desplazar **Q** hacia la derecha**;**
Desplazar **R** hacia la derecha**;**
Incrementar **C** en una unidad**;**
Si no Desplazar **Q** hacia la derecha**;**
Desplazar **R** hacia la derecha**;**
Incrementar **C** en una unidad**;**
Fin_de_Si;
Si C = 64 **Fin del algoritmo.**
Fin_de_Si;
Fin_de_Repetir (volver al inicio de la repetición)**;**

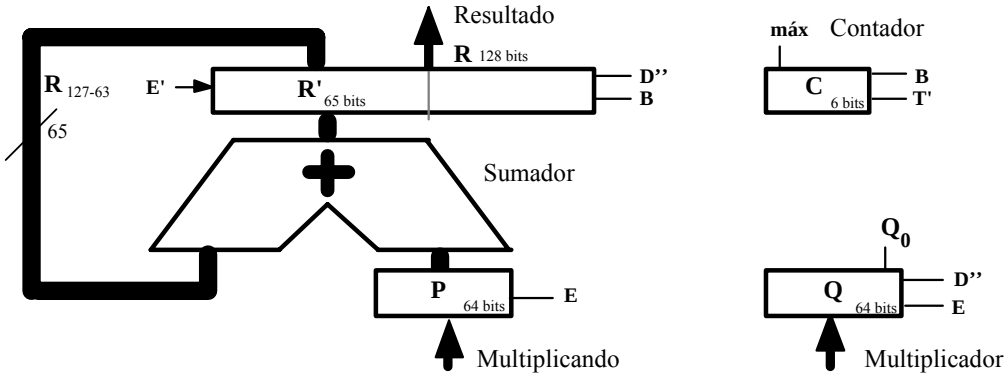

Figura 3. Recursos de cálculo para el producto de dos números de 64 bits.

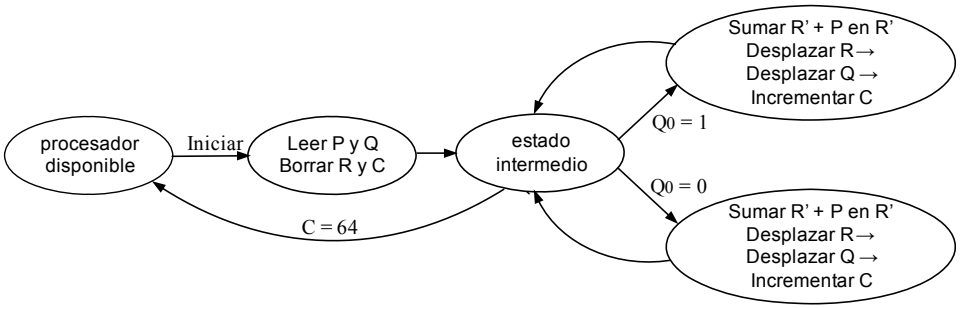

Figura 4. Grafo de estados de control del producto de dos números de 64 bits.

4. Raíz cuadrada de un número binario

Una tercera máquina algorítmica que se propone es la dedicada a efectuar la raíz cuadrada de un número binario de 64 dígitos, con el siguiente esquema de cálculo:

- recorrer el número **P** de dos en dos dígitos (comenzando por los más significativos), restando, cuando se pueda, el resultado parcial **R** con el añadido **01** al final;
- en cada desplazamiento, añadir un **1** al resultado **R** si se efectúa la resta y un **0** cuando no es posible.

Este esquema se deriva del que se emplea para hacer «a mano» la raíz cuadrada de un nº binario:

- se toman los dos primeros dígitos (los más significativos; si el número de dígitos es impar se toma solamente el primero de ellos) y, si es posible, se les resta **01** y se pone un **1** en el resultado (si no es factible, no se hace la resta y el resultado es **0**);
- a partir de aquí, se toma el resto resultante y se le añaden (se «bajan») dos nuevos dígitos del número de manera que, si resulta un resto mayor o igual que el doble del resultado parcial con un **1** añadido (que equivale a añadir, simplemente, **01** al resultado parcial), se resta y se añade un **1** al resultado y, caso de que la resta no sea factible, se añade un **0** al resultado;
- y se repite, sucesivamente, el anterior proceso de «bajar» dos nuevos dígitos y restar, si es posible, hasta completar el recorrido del número inicial.

Al igual que en el caso de la división, se utilizan dos registros auxiliares: un contador **C** para el número de veces que se ha realizado el ciclo y un registro **S**, «resto parcial», que recoge el número binario que, en cada momento se compara con el **R01** (el resultado parcial, seguido de **01**); el registro **S** se une como registro de desplazamiento al **P** (registro **S-P**), para ir añadiendo al «resto parcial» **S** los dígitos del número **P** de dos en dos.

Inicio del algoritmo
Leer **P;** Borrar **R, S** y **C;**
Desplazar hacia la izquierda **S-P** 2 bits;
Repetir
Si S ≥ R&01
Restar **S - R&01** sobre **S**;
Desplazar **R** hacia la izquierda con entrada **1**;
Desplazar hacia la izquierda **S-P** 2 bits;
Incrementar **C** en una unidad;
Si no Desplazar **R** hacia la izquierda con entrada **0**;
Desplazar hacia la izquierda **S-P** 2 bits;
Incrementar **C** en una unidad;
Fin_de_Si;
Si C = 32 Fin del algoritmo.
Fin_de_Si;
Fin_de_Repetir (volver al inicio de la repetición)**;**

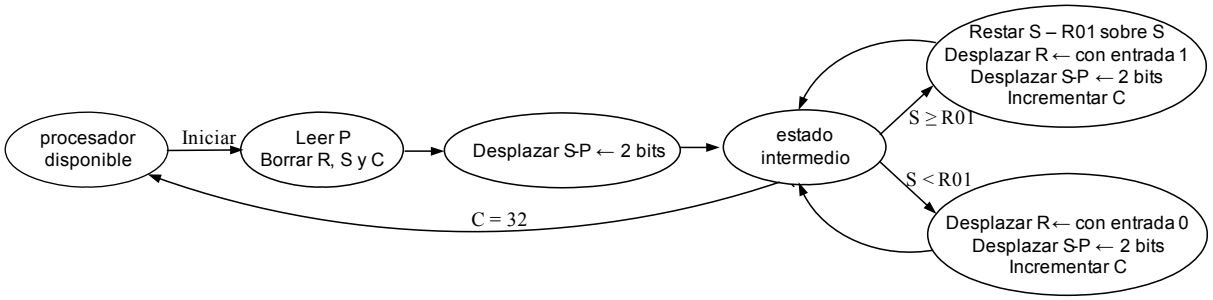

Figura 5. Grafo de estados de control de la raíz cuadrada de un número de 64 bits.

Todos los estados, menos el inicial, duran un ciclo de reloj y la ejecución completa de la raíz cuadrada requiere 67 ciclos (1 + 1 + 32x2 + 1); se añade, también, un *estado intermedio* para que la acción *desplazar S-P* se ejecute antes de llegar al salto condicional (comparación de S con R01) y, también, el *incremento de C* se efectúe antes de compararlo con 32.

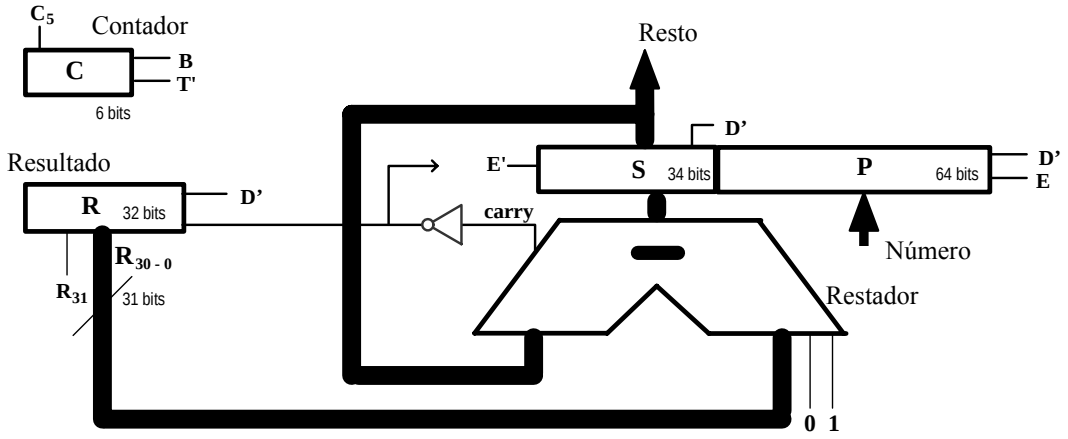

Figura 6. Recursos de cálculo para la raíz cuadrada de un número de 64 bits.

La entrada de control **D'** controla el desplazamiento (de 2 bits) hacia la izquierda y la salida de arrastre del restador sirve para comparar **S** (el «resto parcial») con **R01** y para almacenar en el resultado **R** un **1** (cuando **S ≥ Q**) o un **0** (cuando **S < Q**).

5. Conversión entre binario y BCD

También es de interés la máquina algorítmica correspondiente a la conversión de binario a BCD. El esquema de cálculo para dicha conversión puede ser el siguiente:

- recorrer el número **P** de bit en bit (comenzando por el más significativo) y sumar (en BCD) dicho bit al resultado;
- en cada desplazamiento, salvo en el último, multiplicar por dos (en BCD) el resultado.

Este esquema se deriva de la expresión del valor relativo de los dígitos de un número binario; consideremos un número más pequeño **n'** de 16 dígitos, ponmlkjihgfedcba(2:

$$\mathbf{n'} = p.2^{15} + o.2^{14} + n.2^{13} + m.2^{12} + l.2^{11} + k.2^{10} + j.2^{9} + i.2^{8} + h.2^{7} + g.2^{6} + f.2^{5} + e.2^{4} + d.2^{3} + c.2^{2} + b.2 + a$$

$$= (((((((((((((p \cdot 2) + o) \cdot 2 + n) \cdot 2 + m) \cdot 2 + l) \cdot 2 + k) \cdot 2 + j) \cdot 2 + i) \cdot 2 + h) \cdot 2 + g) \cdot 2 + f) \cdot 2 + e) \cdot 2 + d) \cdot 2 + c) \cdot 2 + b) \cdot 2 + a$$

El producto por 2 puede hacerse sumando **R** consigo mismo: **R** + **R** = 2.**R**; las operaciones han de hacerse en BCD para que el resultado quede en tal codificación; de forma que el esquema de cálculo (para un número de 64 dígitos binarios) puede ser reordenado de la siguiente manera:

- recorrer el número **P** de bit en bit (comenzando por el más significativo) haciendo lo siguiente: (multiplicar **R** por 2) sumar en BCD el resultado consigo mismo (**R** + **R**); y sumar también el correspondiente bit del número **P** (dicho bit puede sumarse a través del arrastre inicial).

Inicio del algoritmo
Leer **P;** Borrar **R** y **C;**
Repetir Sumar (*en BCD*) **R + R + bit más significativo de P** en **R;**
Desplazar hacia la izquierda **P;**
Incrementar **C** en una unidad**;**
Si C = 63 (64 ejecuciones de la repetición; la primera vez C = 0)
Fin del algoritmo.
Fin_de_Si;
Fin_de_Repetir (volver al inicio de la repetición)**;**

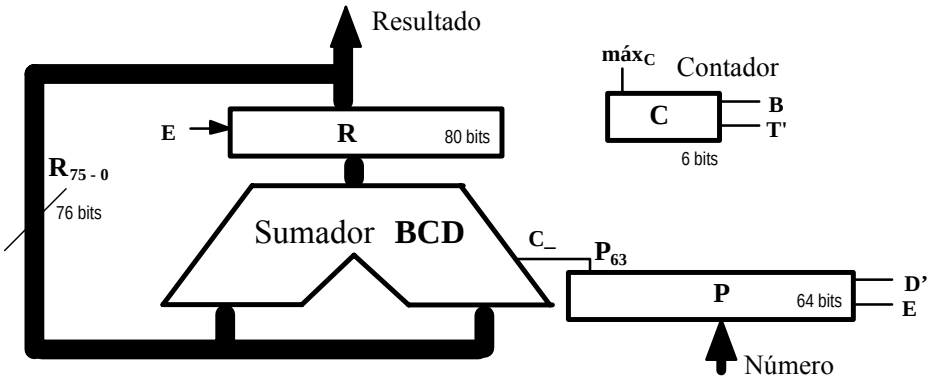

Figura 7. Recursos de cálculo para la conversión a BCD de un número de 64 bits binario.

La conversión de un número binario de 64 bits a BCD ocupa un máximo de 20 cifras BCD ($10^{20} > 2^{64} > 10^{19}$), que equivalen a 80 bits; el número BCD (en **R**) anterior a la última suma (corresponde a 63 bits) ocupa un máximo de 19 dígitos ($10^{19} > 2^{63}$), 76 bits.

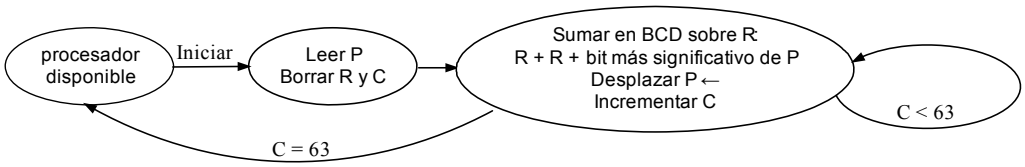

Figura 8. Grafo de control de la conversión a BCD de un número de 64 bits binario.

No se ha añadido ningún *estado intermedio* y, por ello, la condición *C = 63* se verifica cuando el ciclo se ha recorrido 64 veces, ya que la acción *incrementar C* se ejecuta en forma síncrona en el mismo flanco de reloj que el salto condicionado (la primera vez C = 0). El número de ciclos de reloj necesarios para completar la conversión es de 65 (1 +64).

La conversión recíproca de BCD a binario puede realizarse recorriendo el número **P** de cifra en cifra (es decir, de 4 en 4 bits) comenzando por la cifra BCD más significativa, sumando sus cuatro bits al resultado y multiplicando por diez el resultado en cada desplazamiento, salvo en el último. El producto por 10 puede hacerse, en binario, mediante la suma **R000 + R0** = R x 8 + R x 2.

Este esquema corresponde a la expresión del valor relativo de las cifras de un número BCD; sea **n** de 16 dígitos BCD, $PONMLKJIHGFEDCBA_{(10}$, que serán 64 bits:

$$\mathbf{n} = P.10^{15} + O.10^{14} + N.10^{13} + M.10^{12} + L.10^{11} + K.10^{10} + J.10^{9} + I.10^{8} + H.10^{7} +$$
$$+ G.10^{6} + F.10^{5} + E.10^{4} + D.10^{3} + C.10^{2} + B.10 + A$$
$$= \mathbf{(}\,\mathbf{(}\,\mathbf{(}\,\mathbf{(}\,\mathbf{(}\,\mathbf{(}\,\mathbf{(}\,\mathbf{(}\,\mathbf{(}\,\mathbf{(}\,\mathbf{(}\,\mathbf{(}\,P\,.\,10\,\mathbf{)} + O\,\mathbf{)}\,.\,10 + N\,\mathbf{)}\,.\,10 + M\,\mathbf{)}\,.\,10 + L\,\mathbf{)}\,.\,10 + K\,\mathbf{)}\,.\,10 + J\,\mathbf{)}\,.\,10 +$$
$$+ I\,\mathbf{)}\,.\,10 + H\,\mathbf{)}\,.\,10 + G\,\mathbf{)}\,.\,10 + F\,\mathbf{)}\,.\,10 + E\,\mathbf{)}\,.\,10 + D\,\mathbf{)}\,.\,10 + C\,\mathbf{)}\,.\,10 + B\,\mathbf{)}\,.\,10 + A$$

Inicio del algoritmo
Leer **P;** Borrar **R** y **C;**
Repetir
Sumar **R&000 + R&0 + 4 bits más significativos de P en R;**
Desplazar hacia la izquierda **P** 4 dígitos; Incrementar **C** en una unidad**;**
Si C = 15 Fin del algoritmo.
Fin_de_Si;
Fin_de_Repetir (volver al inicio de la repetición)**;**

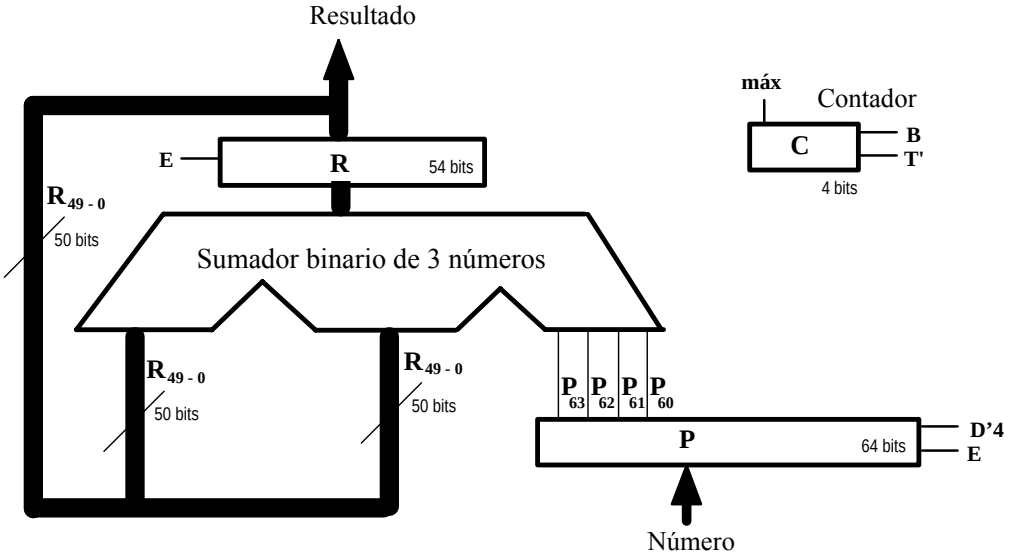

Figura 9. Recursos de cálculo para la conversión a binario de un número de 16 cifras BCD.

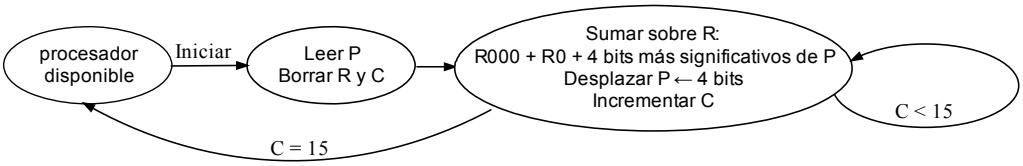

Figura 10.

Grafo de control de la conversión a binario de un número de 16 cifras BCD.

La conversión de 16 cifras BCD a binario ocupa un máximo de 54 bits ($2^{54} > 10^{16} < 2^{53}$); el número binario (en **R**) anterior a la última suma (correspondiente a 15 cifras BCD) ocupa un máximo de 50 bits ($2^{50} > 10^{15}$). Para completar el cálculo se necesitan 17 ciclos de reloj.

6. Control de cronómetros mediante pulsadores

Las cinco máquinas algorítmicas descritas anteriormente se refieren a operaciones aritméticas. Ahora bien, las máquinas algorítmicas no se limitan a sistemas dedicados a cálculos aritméticos; al contrario, son un procedimiento general para abordar sistemas digitales complejos y para ponerlo de manifiesto se describen dos ejemplos no numéricos referidos a dos cronómetros, uno de ellos de tipo habitual controlado mediante dos pulsadores y el otro dedicado a medir tiempos de 8 nadadores.

*Se desea controlar un cronómetro mediante dos pulsadores **P** y **Q** tal que el contaje de tiempo se inicie al activar **P** y se detenga al pulsar **Q** y un segundo pulso de **Q** sirva para ponerlo a cero; en cambio, una vez detenido el contaje, si se pulsa **P** se reanuda el mismo. Si mientras el cronómetro está activo se pulsa **P**, el contaje prosigue pero el visualizador queda detenido en el último valor previo a dicho pulso; se vuelve a la visualización normal del contaje pulsando nuevamente **P**.*

Dado que es un sistema relativamente simple, resulta adecuado comenzar identificando los recursos operativos necesarios. El cronómetro requiere un contador a partir de una frecuencia de 10 Hz (período de 0,1 s), con entradas de habilitación **T** y de borrado **B**, un registro para «retener» el contaje, con entrada de habilitación **E**, y un visualizador apropiado. La siguiente figura muestra el diagrama de bloques para un cronómetro con capacidad hasta 10 horas (9 h 59 m 59 s 9 décimas); por sencillez, se representa la visualización de las 6 cifras BCD en forma estática, pero lo habitual será que se realice por barrido dinámico sucesivo de las 6 cifras.

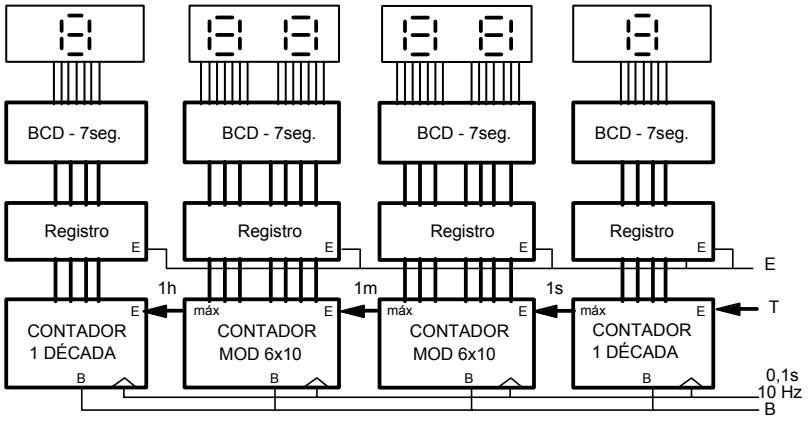

Figura 11. Diagrama de bloques de un cronómetro.

El control de este sistema responde al grafo de estados de la figura 12 (se ha dado prioridad al pulsador **P** respecto a **Q**):

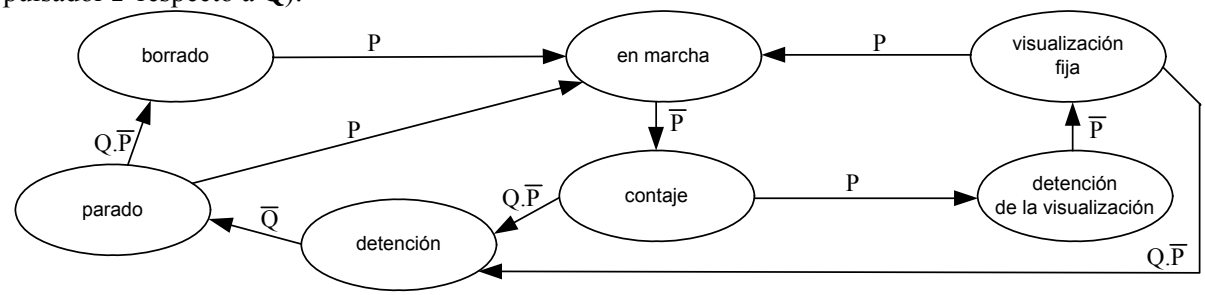

Figura 12. Grafo de estados para el control del cronómetro con dos pulsadores P y Q.

Otro ejemplo de interés: *un cronometro manual para carreras de natación de ocho nadadores, con capacidad de medida hasta 100 minutos y precisión de una décima de segundo; un pulsador **R** servirá para su puesta a cero y otro **C** para todo el control de medidas y visualización: para iniciar el contaje se pulsa **C** y, luego, se pulsará sucesivamente ocho veces más para registrar el tiempo de cada uno de los nadadores. Una vez finalizada la medida de los ocho tiempos, aparecerá en el visualizador el tiempo del primer nadador y cada vez que se pulse **C** el visualizador pasará a representar el tiempo del siguiente (obviamente, del octavo vuelve a pasar al primero de ellos).*

Al igual que en el caso anterior, el cronómetro requiere un contador con un reloj de 10 Hz (período de 0,1 s) y entradas de habilitación **T** y de borrado **B** y un visualizador; también se necesitará un registro para almacenar el tiempo de cada nadador. Los ocho registros han de tener sus entradas conectadas a las salidas del contador y sus salidas deben poder conectarse, en forma multiplexada, al visualizador; para facilitar el diseño (y el dibujo del mismo) el multiplexado se realiza dotando a los registros de salida tri-estado, de forma que todos ellos estarán conectados al visualizador y, en cada momento, se activan las salidas del registro que corresponda. Cada registro tendrá una entrada de habilitación de escritura $\mathbf{E_i}$ (**i** variando de 0 a 7), otra de habilitación de lectura $\mathbf{OE_i}$ y una entrada común de borrado **B**.

En principio, el grafo de estados de este sistema tendrá, al menos, 17 estados, debido a la necesidad de diferenciar los registros, en las dos situaciones: escritura (ocho estados) y lectura (otros ocho); para evitar tan amplio número de estados se puede utilizar un contador módulo 8 (**cont**), seguido de un decodificador, que seleccione el registro al que corresponde actuar. De igual forma, resulta conveniente utilizar un detector de los flancos de subida del pulsador **C**, para evitar tener que detectar cada pulsado a través de una secuencia (C = 0, C = 1).

En tal caso, el grafo de estados se reduce a diferenciar las situaciones de reposo, escritura y lectura; la escritura corresponde al registro de tiempos de los ocho nadadores y la lectura a la posterior visualización cíclica de los mismos.

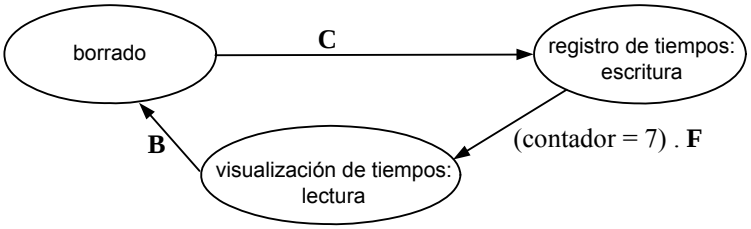

Figura 13. Grafo de estados para el control del cronómetro para 8 nadadores.

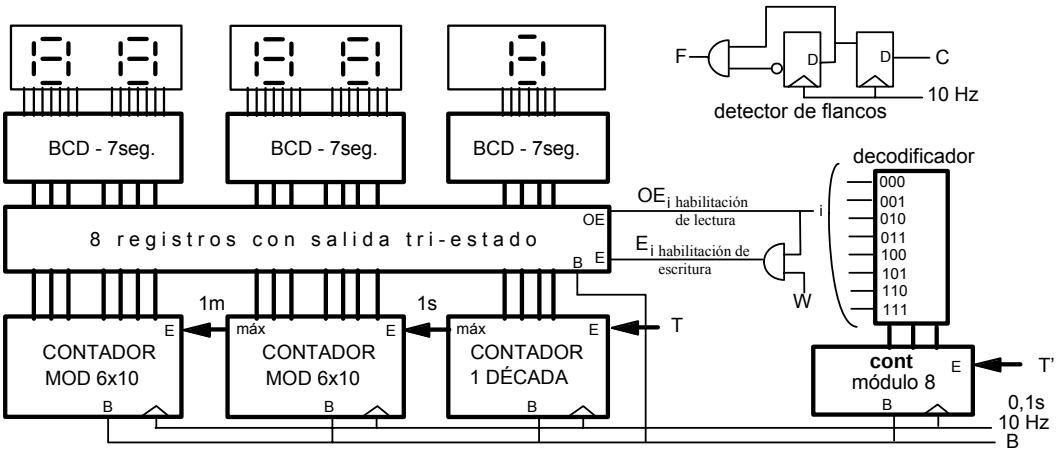

Figura 14. Diagrama de bloques de un cronómetro para 8 nadadores.

El contador auxiliar (módulo 8), que recorre los ocho registros, debe «contar» tanto en el estado de escritura como en el de lectura cuando se detecta un flanco del pulsador **C**; para ello su habilitación se producirá en ambas situaciones (escritura y lectura) cuando se detecta flanco **F** de la señal **C**: $T' = (\text{escritura} + \text{lectura}) . F$.

7. Conclusión y futuras ampliaciones

La finalidad de esta comunicación es llamar la atención de los profesores de electrónica digital sobre la utilidad y viabilidad de las máquinas algorítmicas como ejercicios de diseño de sistemas digitales relativamente complejos.

Otros detalles de estas máquinas algorítmicas y su descripción VHDL podrán ser consultados en la referencia [1] y en la página web: http://www.unizar.es/euitiz/digital.htm.

Finalmente, este mismo método conceptual permite configurar procesadores y autómatas programables, de forma que también pueden abordarse, como ejercicios de diseño, procesadores con un número de instrucciones limitado, que el alumno puede implementar en FPGAs y verificar su funcionamiento.

Referencias

[1] T. Pollán *Electrónica Digital III. Microelectrónica* . Prensas Universitarias de Zaragoza. Colección Textos docentes nº 104. Universidad de Zaragoza 2004. En particular el capítulo 24 (sobe todo, 24.3 y 24.4) y el apéndice A6 (apartados 7, 8 y 9).
[2] Página web: http://www.unizar.es/euitiz/areas/aretecel/docencia/digitel/digitelib.htm , capítulo 24
[3] Página web: http://www.unizar.es/euitiz/areas/aretecel/docencia/digitel/digitelib.htm , apéndice A6