24 APROXIMACIÓN ESTRUCTURAL AL DISEÑO DE SISTEMAS COMPLEJOS

El estudio de la electrónica digital suele recorrer un itinerario acumulativo de funciones booleanas, conjuntos de funciones, bloques combinacionales, biestables y registros, contadores y sus derivados, memorias, …, en una aproximación «batton-up», de lo sencillo hacia arriba; de esta forma llegamos a conocer los elementos disponibles, los recursos constructivos, es decir, las «piezas del diseño». Pero el diseño real supone algo más que conocer las «piezas» disponibles, es un ejercicio de síntesis, un proceso en dirección contraria «top-down», del todo hacia abajo.

Resulta difícil enfrentarse a «la complejidad», pero es algo que los diseñadores de sistemas digitales venimos haciendo en el «día a día» y conviene transferir una pautas o «formas de actuar» que orienten y ayuden a quienes se inician en las tareas de diseño.

Si nos acercásemos al templo de Apolo en Delfos, con confianza en el oráculo, y nuestra pregunta fuese «¿Cómo abordar el diseño de sistemas complejos?», es seguro que la pitonisa, en la forma oscura y ambigua que acostumbra, respondería con dos palabras: «Divide y vencerás». Si, además, la pitonisa tuviera conciencia de la complejidad de nuestros sistemas actuales, seguramente añadiría: «Divide, sí, pero con estructura».

Dividir en partes, pero sin perder de vista la globalidad. Es decir, dividir dotando al conjunto de una estructura adecuada, conociendo «la relación, orden y enlace de las partes para formar el todo» (que es la definición apropiada de "estructura").

En esta perspectiva, al tratar de los sistemas secuenciales, se introduce el sincronismo como división estructurada de algo tan complejo como es el tiempo: el sincronismo separa el tiempo en unidades sucesivas, como forma de manejarlo con seguridad, eficacia y sencillez.

El presente capítulo, además de plantear, en forma conceptual y genérica, la estrategia de «dividir con estructura», presenta en detalle dos procedimientos sumamente útiles de efectuar tal división: la arquitectura de buses como estructura eficaz para sistemas con mucha transferencia de información y la separación parte de control/parte operativa para esquemas de cálculo complejos. Se incluyen varios diseños de máquinas algorítmicas (parte de control) relativamente complejas (división, raíz cuadrada, conversión BCD-binario) así como dos ejemplos «no aritméticos», referidos a control de cronómetros.

Además, se comenta, en forma breve, el mecanismo de segmentación (pipe-line) como forma de aumentar la velocidad de trabajo, a través de la división en partes de los tiempos de propagación de los bloques combinacionales (cuando tales tiempos son altos).

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

24.1. Hacer manejable la complejidad

¿Cómo abordar el diseño de sistemas digitales complejos? Divide y vencerás, es una buena respuesta; pero, como complemento a la división, conviene añadir la exigencia de «estructura» (dividir teniendo clara la relación de cada parte con el conjunto).

¿Cómo abordar un diseño que nuestra mente no puede abarcar en conjunto?[1] «Fraccionar sin perder la globalidad», podría ser una respuesta concisa: la idea es dividir el sistema en partes, conservando la visión global del mismo; aún más, dando particular importancia a la estructura, a la relación de cada parte con el conjunto.

Estructura: relación, orden y enlace de las partes para formar un todo (un sistema).

La relación de cada parte con el conjunto presenta una triple referencia:

- funcionalidad de cada parte en relación con el sistema, con las demás partes,
- comunicación física (líneas de conexión) de las partes,
- y ordenación temporal de las señales entre ellas.

Por partes (divide y vencerás) y, a la vez, con estructura (con perspectiva organizada del conjunto): una metodología que permita poner en relación el diseño de cada parte con los requisitos globales que se pretenden, combinando adecuadamente los aspectos de partición, jerarquía y coordinación.

Esto es lo que hacemos cuando formulamos la arquitectura de un sistema en términos de «diagrama de bloques»; cada bloque constituye una parte diferenciada del sistema y la conexión entre ellos sitúa cada parte en la perspectiva de conjunto: confiere estructura a la división.

Tal conexión presenta el triple aspecto citado: conexión funcional (funcionalidad de cada bloque respecto a los demás), conexión física (comunicación entre los bloques) y conexión temporal (ordenación de las señales entre ellos).

Un diagrama de bloques representa una de las posibles soluciones (*arquitecturas*) que realizan el sistema, es decir, que cumplen las especificaciones o requisitos del proyecto; supone una *división estructural* en la que cabe destacar ambos aspectos:

- una *división en partes* (bloques) que permitirá el diseño independiente de cada parte,
- y una *estructura* que expresa la función de cada parte respecto al conjunto (al sistema) y, asimismo, la comunicación entre las partes.

El aspecto estructural, de relación entre partes, exige que se preste especial atención a la conexión entre ellas: tanto la coordinación funcional entre las diversas partes como la comunicación (adecuadamente sincronizada) entre las mismas.

---

[1] *Every engineering construction is an organized system. If it contains only a few components, the designer has no difficulty in bearing them all in mind at once, but if there are many components, the designer almost instinctively creates a hierarchical design with subassemblies and sub-subassemblies as necessary so that he can shape the design without exceeding his span of understanding. By such means in every field of engineering the design task is broken down into manageable sections.*

***G. G. Scarrott** From computing slave to knowledgeable servant: the evolution of computers.*

El nombre de *arquitectura* resulta sumamente apropiado, ya que destaca la idea de una estructura que conecta, funcional y físicamente, partes diversas; conocida la situación de cada parte en la estructura, es posible un tratamiento individualizado de la misma.

El diseño es un trayecto desde las especificaciones o requisitos que definen al sistema hasta las funciones o bloques booleanos que determinan su configuración circuital. En esta perspectiva, la división en partes se ejecuta, por lo general, en dos etapas sucesivas (a través de dos «escalones» diferenciados que descienden del todo a las partes): una primera división en módulos, referidos a las especificaciones o prestaciones del sistema y su posterior acomodación a bloques, en relación con los recursos (subsitemas digitales).

En la primera aproximación se divide el sistema en módulos con significado funcional respecto al propio sistema, respecto a las especificaciones o funciones que se pretende que el sistema realice; en un segundo paso se configuran o reajustan dichos módulos en bloques o subsistemas digitales conocidos (y, si es preciso, se introducen bloques «específicos»), definidos todos ellos por su comportamiento booleano.

En todo este proceso de división, conservando la globalidad a través de la estructura, resulta importante dedicar tiempo a:

- dejar claras las especificaciones del sistema,
- llegar a comprenderlas en profundidad,
- conocer sus razones, sus efectos y las relaciones entre ellas,
- idear un esquema de funcionamiento detallado y coherente,

todo ello para llegar a establecer la arquitectura correspondiente, representada en un diagrama de bloques.

De todo lo cual debe dejarse constancia por escrito, porque la escritura es la «herramienta» de las ideas, la única forma de dejarlas claras, de transmitirlas y de contrastarlas y de recuperarlas andando el tiempo.

Nunca insistiremos demasiado en la necesidad de «escribir», de documentar el diseño, de describir cada detalle, cada variable, cada registro,… (tanto su función propia como su relación con el conjunto); en el ahorro de tiempo que, en definitiva, supone el texto escrito tanto para clarificar ideas, como para transmitirlas, no solamente a otras personas, sino también a uno mismo con el transcurso del tiempo.

La escritura nos impone la disciplina de aclarar y concretar las ideas, de «darles forma» expresándolas en modo explícito y de recogerlas en un soporte material. A cambio, la escritura es un eficaz vehículo de comunicación para transmitir las ideas y permitir su revisión y debate y para almacenarlas y permitir recuperarlas después.

Precisamente, el lenguaje de descripción circuital VHDL nació como herramienta de documentación; como respuesta a la necesidad de describir, en forma clara, precisa y exenta de ambigüedades, el comportamiento detallado de los circuitos digitales complejos. La propia potencialidad de la «escritura estructurada» hizo que la descripción VHDL pasase a ser, también, una excelente herramienta de diseño.

En la perspectiva de «dividir en partes», cabe destacar que el sincronismo no es sino una partición aplicada al tiempo que, al cuantificarlo en unidades, facilita el diseño y le confiere seguridad funcional; es una forma de «modularizar» un aspecto particularmente complejo por su carácter de «continuo» e impreciso cual es el tiempo.

En buena medida el sincronismo en los sistemas digitales equivale a un «buen manejo del tiempo»: al dividir el tiempo en unidades sucesivas facilita la planificación temporal en el diseño de los sistemas complejos (permite referir su actividad funcional a unidades de tiempo discretas y numerables), simplifica los cálculos relativos a los tiempos de propagación y tiempos funcionales de los biestables (el análisis de tiempos) y, a la vez, proporciona una gran seguridad de funcionamiento.

[En tal sentido, se invita, a quienes no lo hayan hecho, a estudiar el capítulo 15 (segundo volumen) dedicado al "significado, requisitos y utilidad del sincronismo". Por no aumentar el número de páginas, dicho capítulo no ha sido repetido en este libro; pero hubiera sido muy apropiado hacerlo.]

En los próximos apartados consideraremos en detalle otros dos casos genéricos de división estructural que presentan particular interés y, además, son muy frecuentes:

- Sistemas con mucha transferencia de información, en los que resulta sumamente útil la división procesador/memoria y, dentro de ésta, la numeración de los diversos registros que permite el tratamiento individualizado de grupos de registros (y, en su caso, de periféricos conectados a tales registros).
- Sistemas con esquema de cálculo complejo, para los cuales es apropiada la división parte operativa/parte de control, que, a su vez, permite tratar por separado cada uno de los recursos de cálculo.

En ambos casos, la división conceptual en dos partes diferenciadas (procesador/memoria, operaciones/control) se refleja sobre una de las partes en una partición de múltiples elementos cuyo diseño se puede abordar por separado: los diversos elementos de la memoria (diversos bloques de memoria o diferentes periféricos) y los varios recursos de cálculo (operadores y registros activos) pueden individualizarse en cuanto a su definición y diseño.

Ni qué decir tiene que muchos sistemas digitales presentan, a la vez, las dos situaciones anteriormente descritas (mucha transferencia de información y esquema de actuación complejo) y, en consecuencia, les son de aplicación las dos particiones antes indicadas.

Por otra parte, las citadas no son las únicas formas de dividir un diseño digital en partes, sino que la estrategia de división es una metodología de tipo conceptual que puede ser aplicada en forma diversa y resulta de particular interés plantearse cúal es el tipo de división estructural que resulta más adecuado a cada caso.

El esquema de «fraccionar sin perder la globalidad» puede (y debe) ser aplicado sucesivamente en forma jerárquica (iterativamente), es decir, a partir de un primer esquema de bloques, los bloques complejos pueden ser divididos en «subbloques», dando lugar a un segundo nivel de «esquemas de bloques» y así sucesivamente hasta «hacer manejable» la complejidad.

En los casos citados en la página anterior, la división se refleja sobre una de las partes en forma múltiple:

- en los sistemas con mucha transferencia de información, la memoria se subdivide naturalmente en bloques y en periféricos (y, respecto al procesador, suele interesar la división en parte de control y parte operativa);
- en los sistemas con esquema de cálculo complejo, la parte operativa se descompone en los diversos operadores y registros necesarios (y la máquina de estado relativa a la «parte de control» puede segmentarse, caso de resultar compleja, en varias máquinas de estado comunicadas entre sí).

Conceptualmente, la idea de «particionar» se opone a la práctica «chapucera» de incorporar «parches» y hacer añadidos «al paso»; por ejemplo, incluyendo al socaire del diseño nuevas condiciones mediante biestables adicionales. Toda especificación o variable de diseño debe estar integrada en el conjunto y quedar reflejada en el diagrama de bloques y, en su caso, en el correspondiente grafo de estado; no es buena técnica desarrollar máquinas de estado dejando biestables de control fuera de las mismas.

Por ello, de vez en cuando, en el desarrollo de prototipos, conviene volver atrás y reescribirlo todo, replanteando situaciones y estados en esquemas de conjunto, que se reflejarán en los correspondientes grafos de estado.

Junto con las anteriores reflexiones conceptuales y formulaciones metodológicas, no puede dejarse de lado la importancia de la experiencia: *Sabe más el diablo por viejo que por diablo*. La experiencia es un requisito insustituible para el «saber hacer» (*know-how*): difícilmente será posible abordar con eficacia el diseño de sistemas complejos sin el bagaje de una experiencia conseguida a través de la práctica, con la dedicación personal y el proceso temporal que ello supone.

24.2. Sistemas con mucha transferencia de información

En el caso de elevada transferencia de información (bien porque el sistema necesite almacenar gran cantidad de datos, bien porque su comunicación con el exterior sea muy intensa o diversa) resulta muy eficaz la división en dos partes, procesador/«memoria», conectadas por buses y la identificación de los registros mediante un número. La sencilla idea de numerar los registros y seleccionarlos a través de su número determina una división estructural múltiple, que permite tratar cada registro o grupo de registros por separado.

Entendemos aquí por procesador la parte operativa y de control, en contraposición con la «memoria» que almacena y transfiere la información; en tal sentido, el término es genérico y no se refiere solamente a los procesadores típicos que actúan bajo programa.

Por otra parte, téngase en cuenta que el nombre de «memoria» se utiliza aquí en su sentido amplio, para designar a la parte «no procesador» que no solamente contiene elementos de memoria que almacenan la información (archivo) sino también dispositivos de entrada y de salida que intercambian información con el exterior (comunicación).

La *memoria* de un sistema digital complejo engloba a todo tipo de registros de los que se extrae directamente o a los que se envía directamente información digital. En tal sentido, incluye los registros de trabajo donde se memorizan datos y resultados (generalmente englobados en bloques de tipo RAM), los registros de información fija que contienen tablas de valores (como pueden ser los bloques ROM) y, también, los registros de adaptación de entradas y de salidas (adaptadores de periféricos).

La *memoria* alude a dos unidades diferenciadas por su finalidad funcional:

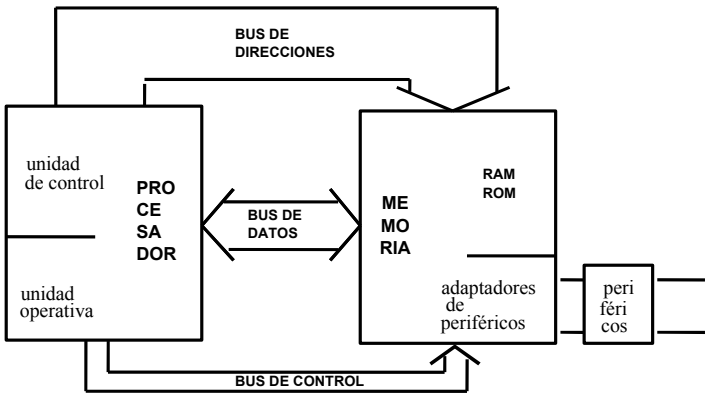
• la unidad de memoria, como lugar de almacenamiento de la información disponible,

• y la unidad de entradas/salidas, conjunto de periféricos que comunican con el exterior.

La comunicación del sistema con el exterior se realiza a través de registros adaptadores de salida que presentan la información hacia un periférico (optoacoplador, LED, visualizador, conversor D/A, etc.) o a través de adaptadores de entrada, asimilables conceptualmente a registros, que reciben la información desde el periférico (pulsador o conmutado, comparador, conversor A/D, etc.). Desde su punto de vista, el procesador se encuentra con un conjunto de registros que, a su vez, se comunican con un periférico a través del cual reciben o transmiten información respecto al exterior.

De esta forma, el sistema digital puede dividirse en dos partes conceptuales:

- el procesador que controla el proceso y efectúa las operaciones pertinentes,
- y la *memoria*, conjunto de registros que almacenan, reciben o envían la información:



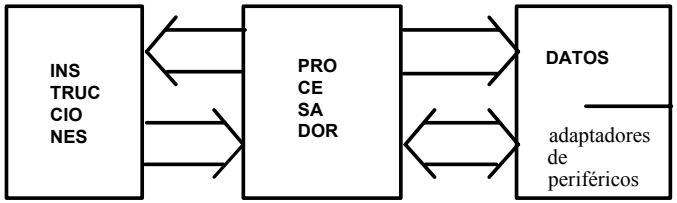
Arquitectura de buses con «memoria única»

La utilización de buses diferenciados permite organizar tanto el almacenamiento como la transferencia de información en forma simple: todos los registros se numeran correlativamente accediendo a ellos por las mismas líneas (bus de datos), un segundo conjunto de líneas seleccionan el registro sobre el que se opera (bus de direcciones) y unas pocas líneas de control determinan la operación a realizar y sincronizan la transferencia de información (bus de control).

[En los capítulos 19 y 20 (segundo volumen) se describe en detalle la arquitectura de buses y la configuración de mapas de memoria, con un amplio número de ejemplos de ubicación de bloques de memoria o adaptadores de periféricos en ellos.]

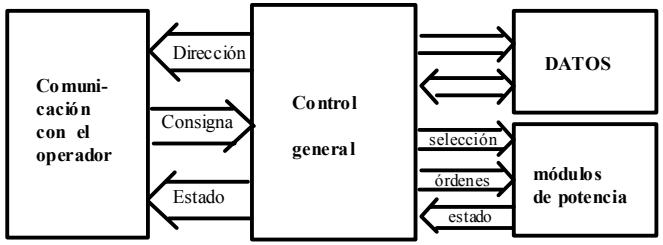
La organización mediante un mapa de memoria, gestionado a través de buses, es una estrategia de tipo genérico que puede ser aplicada de formas muy diversas (memoria única, arquitectura Harvard, módulos especializados, …): lo básico de esta metodología es el acceso a múltiples registros a través de su número. Además de la división en dos partes, procesador y «memoria», esta arquitectura permite abordar por separado el diseño de cada elemento de la «memoria» (de los diversos bloques o partes de memoria RAM y ROM y de los distintos periféricos), una vez establecida la numeración de sus registros.

A diferencia de la configuración de memoria única, la arquitectura Harvard separa las instrucciones que constituyen el programa de la parte de datos y comunicación con el exterior; de esa forma consigue mayor velocidad de procesamiento, a costa de utilizar un mayor número de líneas debido a la duplicación de los buses.



Arquitectura Harvard de procesadores que actúan bajo programa

En sistemas complejos de control de potencia, puede resultar útil la separación en varias partes, comunicadas a través de buses de reducido número de líneas (incluso algunos de una sola línea); suele interesar diferenciar entre sí los diversos módulos de potencia, la comunicación con el operador y la memoria de datos, controlados a través de un módulo central de control general.



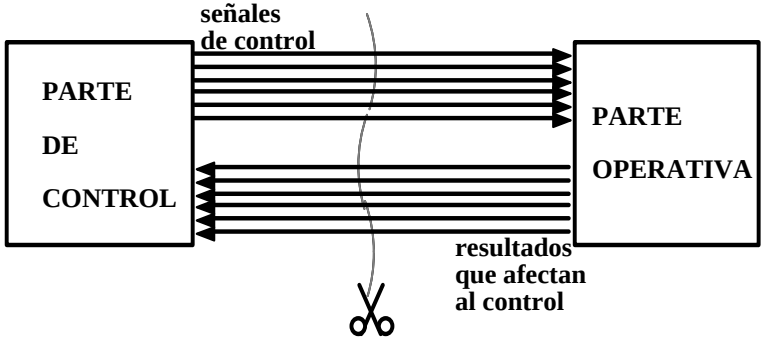
Ejemplo de arquitectura con módulos especializados para sistema de control de potencia

24.3. Sistemas con esquema de cálculo complejo

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

1. Idear el método de operación (el esquema de cálculo), es decir la secuencia genérica de operaciones a realizar y la forma de efectuarlas.
2. Describir dicho esquema mediante un algoritmo o un grafo de estados.
3. Identificar los recursos operativos necesarios (la parte operativa), a saber:
   - los recursos de cálculo (operadores) que permiten efectuar todas las operaciones incorporadas en el algoritmo,
   - los registros necesarios para ejecutar las operaciones en comunicación con los operadores,
   - el conexionado de tales elementos entre sí y su comunicación con el exterior.
4. Expresar la secuencia de control (la parte de control) en forma de máquina de estados que ejecuta el algoritmo de control, activando secuencialmente, a través de sus salidas, los recursos operativos establecidos en el punto anterior. Cada estado corresponde a un conjunto de acciones que se realizan a la vez («en paralelo»), mientras que acciones que requieren un orden correlativo dan lugar a varios estados sucesivos.
5. Abordar el diseño por separado de la parte operativa delimitada en el punto 3 (y, dentro de ella, los diversos operadores y registros) y de la parte de control definida funcionalmente en el punto 4.

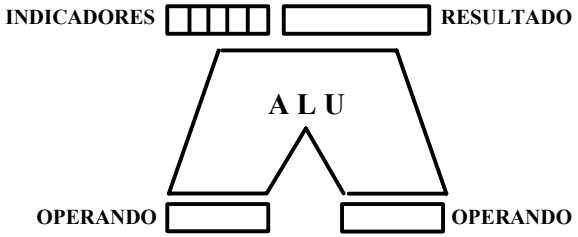
La distinción entre parte operativa y parte de control, establecida en los puntos 3 y 4, simplifica en gran manera el diseño, por cuanto que permite tratar por separado (en momentos diferentes y con una cierta independencia) las operaciones a realizar y la secuencia en que tales operaciones se realizan.



De forma que al diseñar los recursos de cálculo no es preciso considerar el orden con que tales 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.

La parte operativa será construida, en general, mediante bloques operacionales, registros (algunos de ellos con capacidad de desplazamiento), contadores y multiplexores (para estructurar las comunicaciones); también suelen requerirse algunos biestables individuales (que reciben el nombre de indicadores, *flags*) para resultados parciales (de un solo bit) necesarios en operaciones posteriores (como el «arrastre» de suma o resta) o para informar a la parte de control respecto a posibles saltos condicionados (resultado nulo de una operación, resultado negativo, «desbordamiento», comparaciones, …).

En muchas ocasiones los bloques operacionales se reducen a uno solo: una ALU que contenga las operaciones aritméticas y lógicas necesarias.



Parte de control: máquina algorítmica

La parte de control puede ser sintetizada por los métodos generales de los sistemas secuenciales, a partir de su máquina de estados.

Ahora bien, los algoritmos de control suelen corresponder a máquinas de estados de tipo Moore, relativamente particulares:

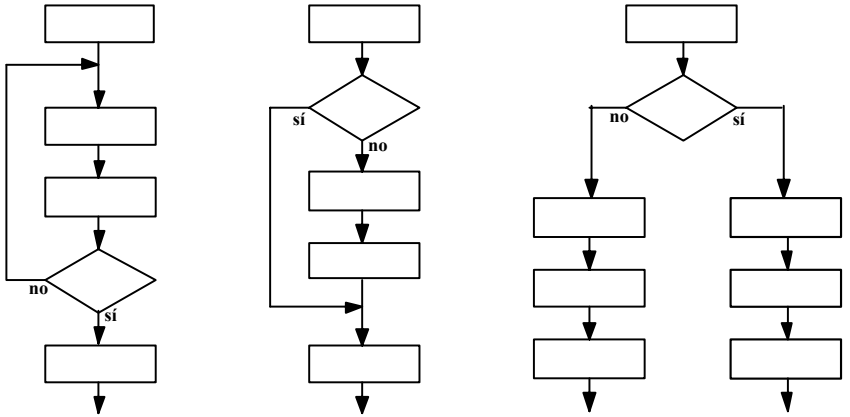
a) cuyos estados están presentes una sola unidad de tiempo (salvo los estados de «espera» a que se cumpla una condición, que también pueden ser formalizados en términos de «una sola unidad de tiempo») ,

b) y su estructura es predominantemente lineal (series de estados sucesivos), junto con «bucles», «avances» y «alternativas».

Tales máquinas de estados pueden formularse en términos de series de estados sucesivos y de saltos condicionados, los cuales pueden determinar:

- el recorrido de una serie de estados en forma cíclica, repetidos varias veces hasta que se cumple la correspondiente condición (bucle),
- una modificación en el recorrido de los siguientes estados, de forma que se produce un salto hacia adelante sin pasar por los inmediatamente siguientes (avance),
- la existencia de varias posibilidades o caminos que conducen a diferentes series de estados (alternativa).

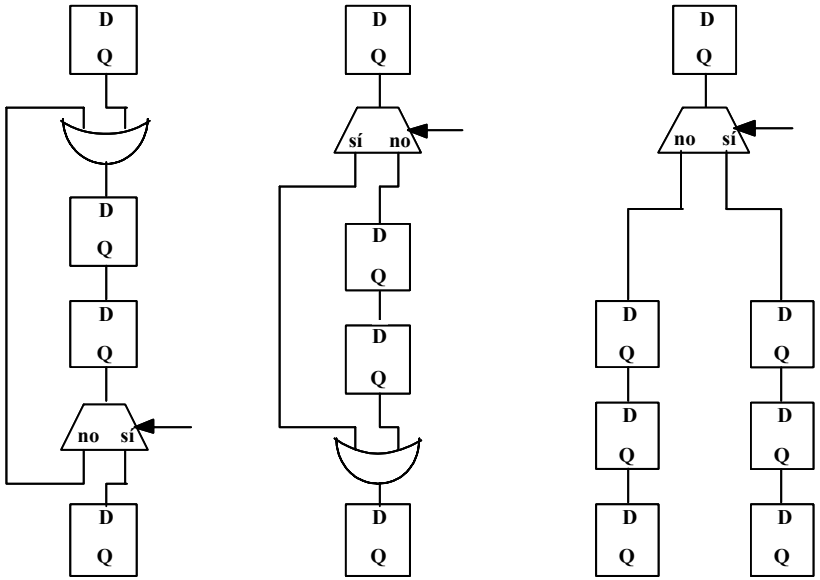
Este tipo de estructura queda muy bien reflejado en los organigramas que se utilizaban para expresar gráficamente la estructura de los programas de computador. En ellos las «cajas» rectangulares representan los estados (conjunto de acciones que se ejecutan en paralelo) y los «rombos» representan los saltos condicionales, que pueden dar lugar a bucles (caso de establecer un lazo cíclico sobre la serie de estados ya recorridos), a avances (caso de saltar en la serie de estados sin recorrer los inmediatamente siguientes) o alternativas (caso de dar paso a dos series diferentes de estados).



La codificación de los estados con «un solo 1» resulta sumamente apropiada, ya que permite construir la máquina de estados reflejando directamente el organigrama sobre un conjunto de biestables, cada uno de ellos correspondiente a un estado («caja» rectangular), y un conjunto de demultiplexores de dos líneas, corresponden a los saltos condicionales («rombos»). [Véase figura de la página siguiente.]

Comentario importante: Conviene tener en cuenta que las acciones propias de un estado, si son de tipo síncrono (y lo serán la mayoría de ellas), no se ejecutan en la unidad de tiempo que corresponde a ese estado, sino cuando llega el flanco de reloj y el estado pasa al siguiente. Por ello, 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. Esto puede evitarse añadiendo un *estado intermedio* que permita la ejecución de tales acciones, actualizando los valores relativos a las condiciones antes de abordar el salto.

Por ejemplo, sea un estado en el cual se carga un determinado valor en un registro **P**, seguido de una transición condicionada a **P = 0**; si la carga es síncrona y la condición (el «rombo» que indica el salto) es inmediatamente posterior a dicho estado, la evaluación de la condición se hará sobre el anterior valor del registro **P** y no sobre el nuevo valor. Puede resolverse este problema con un estado adicional entre el citado y el salto condicional, de forma que la actualización de **P** se hará en la transición a este estado, antes de llegar a la evaluación de la condición para ejecutar o no el salto.



Esta forma de construir la parte de control, configurando el correspondiente sistema secuencial de forma que refleje directamente la estructura del algoritmo de control, recibe el nombre de *máquina algorítmica*.

*Como ejemplo de descomposición en parte operativa y de control se desarrolla a continuación el diseño (uno de los posibles diseños) de un procesador dedicado específicamente a calcular el producto de dos números binarios **P** y **Q** de 64 dígitos, cuyo resultado **R** será un número binario de hasta 128 dígitos.*

[Para facilitar su exposición, este ejemplo se refiere a un cálculo numérico conceptualmente simple y bien conocido; el próximo apartado contiene otros ejemplos de cálculo y un par de ejemplos «no aritméticos» referidos a control de cronómetros.]

1. El *esquema de cálculo* puede ser el siguiente:
   - multiplicar el multiplicando **P** por cada dígito del multiplicador **Q** (comenzando por el menos significativo); dicho producto da como resultado **P** cuando el bit de **Q** vale **1** y **0** si dicho bit es nulo,
   - efectuar el correspondiente desplazamiento sobre los productos parciales anteriores para ajustarlos a su valor relativo,
   - y sumar los productos parciales una vez desplazados.

Este esquema de actuación se corresponde con el que se emplea al hacer «a mano» una multiplicación: cada fila de dicha multiplicación es un producto parcial por uno de los dígitos del multiplicador, cada fila se desplaza un lugar respecto a la anterior para tener en cuenta su valor relativo y, una vez desplazadas, se suman todas las filas.

El esquema de cálculo propuesto necesita un sumador de 128 bits (ya que el resultado puede alcanzar tal longitud); ahora bien, es posible utilizar un sumador de solamente 64 dígitos si la suma se efectúa sobre los bits más significativos del resultado y lo que se desplaza, en cada suma parcial, es el propio resultado; es decir:

- hacer 64 veces lo que sigue
- recorrer sucesivamente los dígitos del multiplicador **Q** desde el bit menos significativo y, cuando el correspondiente bit de **Q** valga **1**, sumar el multiplicando **P** sobre el resultado **R**; efectuar esta suma sobre la parte más significativa del resultado: **R'**= {65 bits superiores de **R**} (obsérvese que las sumas parciales se efectúan sobre los 65 bits más significativos de **R**, ya que el resultado de una suma con operandos de 64 bits puede llegar a tener 65 bits)
- desplazar el resultado **R** hacia la derecha (dividir por 2) para ajustar el valor relativo de cada suma parcial respecto a la siguiente a efectuar; no hacer este desplazamiento tras la última de las sumas.

Este esquema da lugar a 64 sumas parciales posibles, correspondientes a los 64 dígitos del multiplicador (dichas sumas se producen sobre los bits 127-63 del resultado); pero sólo produce 63 desplazamientos, de forma que el bit menos significativo del producto, que se forma en la primera de las sumas parciales sobre el bit 63 del resultado, pasa finalmente al bit 0 (unidades) del mismo.

2. El *algoritmo* correspondiente al anterior esquema de cálculo puede ser el siguiente:

Se introduce un registro **C** (contador) para expresar el número de veces que se ha realizado el ciclo básico del algoritmo (sumas y desplazamientos).

Inicio del algoritmo
Leer **P** y **Q;**
Borrar **R** y **C;**
Repetir
  Si bit_0 (unidades) de **Q = 1**
    Sumar **P + R'** sobre **R';**
  Fin_de_Si;
  Si C = 111111 (63 ≡ 64 ejecuciones de la repetición;
    la primera vez C = 0)
    Fin del algoritmo.
  Fin_de_Si;
  Desplazar **Q** hacia la derecha**;**
  Desplazar **R** hacia la derecha**;**
  Incrementar **C** en una unidad**;**
Fin_de_Repetir (volver al inicio de la repetición)**;**

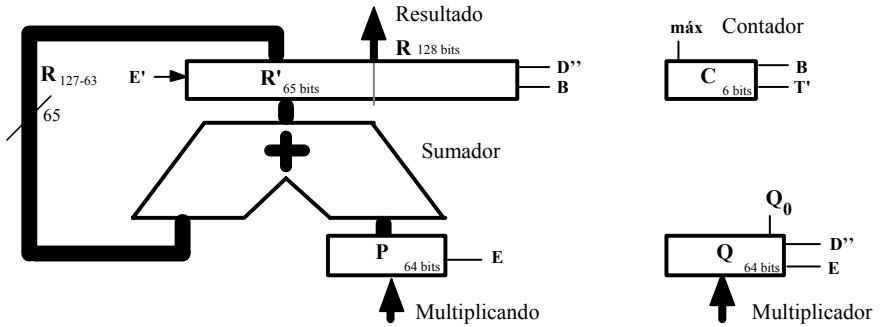
3. *Recursos de cálculo* necesarios:

Operadores:

- un sumador de 64 bits (salida en 65 bits),
- capacidad de desplazamiento hacia la derecha del resultado **R** y del multiplicador **Q**,
- un contador **C** módulo 64 con entrada de borrado y detección de máximo (**111111**).

Registros:

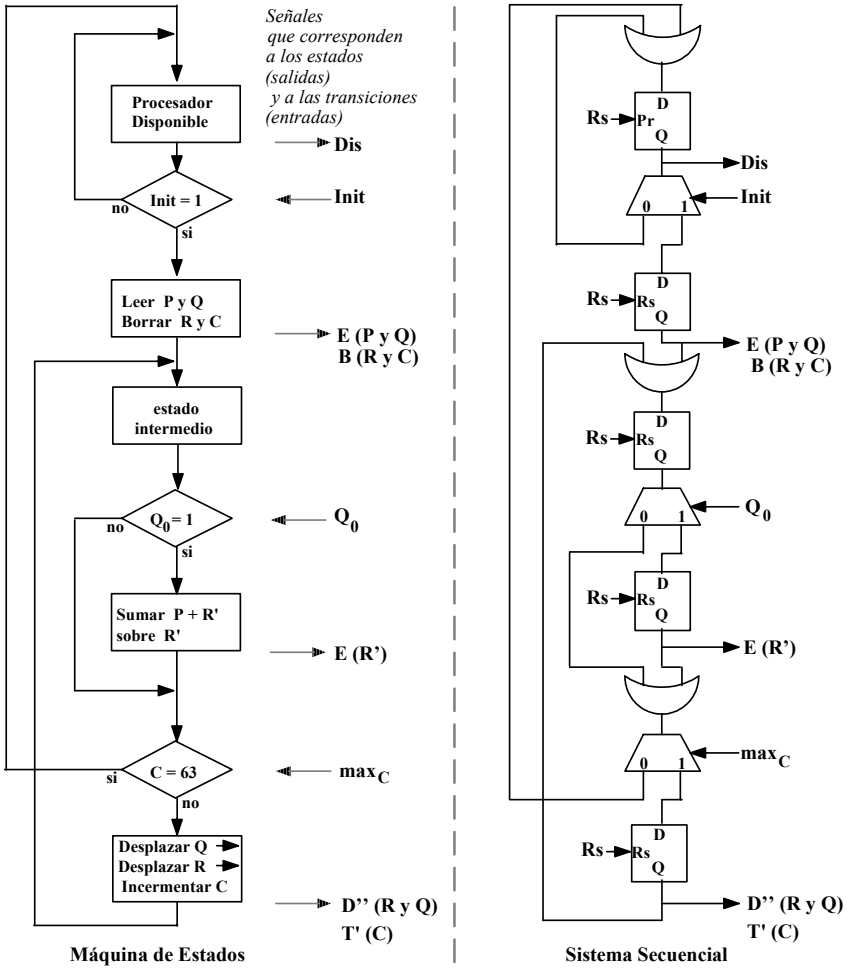
- un registro de retención de 64 bits para el multiplicando **P**,
- un registro de retención de 64 bits para el multiplicador **Q** (con desplazamiento hacia la derecha según se ha dicho en los requisitos de operadores),
- un registro de 128 bits para el resultado **R**, con entrada de borrado (y desplazamiento hacia la derecha según también se ha dicho), cuyos 65 bits más significativos **R'** han de poderse cargar en paralelo (registro de retención respecto a dichos 65 bits),
- un contador de 6 bits **C**, con entrada de borrado (y detección de máximo).



Las señales expresadas en la figura anterior, correspondientes a señales de control de los recursos de cálculo, son las siguientes:

- **E** : entrada de habilitación del correspondiente registro de retención; cuando **E = 1** el registro recoge la palabra binaria situada en sus entradas (en el caso del registro de resultados, su habilitación **E'** solamente actúa sobre los 65 bits más significativos: **R'**),
- **B** : entrada de borrado síncrono del correspondiente registro (incluido el contador); cuando **B = 1** todos los bits del registro pasan a valor **0**,
- **D''** : entrada de desplazamiento hacia la derecha (división por 2) del correspondiente registro; cuando **D'' = 1** cada biestable recoge el valor que tenía el biestable de su izquierda y el biestable más significativo pasa a valor **0**,
- **T'** : entrada de habilitación de contaje del contador; cuando **T' = 1** el contador incrementa su valor en una unidad (con el reloj del sistema secuencial),
- **máx** : salida indicadora de que el contador se encuentra en su valor máximo **111111**
- **Q0** : salida correspondiente al biestable menos significativo (unidades) del registro **Q**.

4. La *maquina de estados* que expresa la secuencia de control puede ser la representada en el organigrama de la figura siguiente (parte de la izquierda).



5. En esta misma figura (parte de la derecha) se representa el *sistema secuencial* que realiza dicha máquina de estados (en una codificación de estados con «un solo 1»).

Se ha supuesto que el proceso de cálculo se activa con una señal **Init** (inicio) y que cuando acaba se vuelve al estado inicial, de forma que la señal **Disp** (disponible) señala que se ha completado el último proceso de cálculo que ha sido activado. La señal **Rs** (*reset*) de los biestables sirve para inicializar el sistema: borra todos los biestables (**Rs**) menos el primero (**Pr**), que lo pone a **1**.

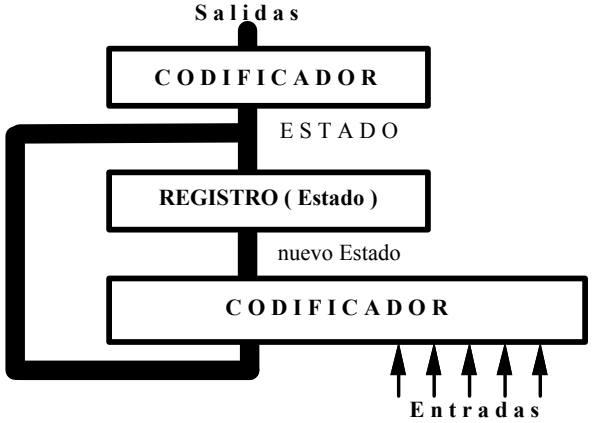
En la anterior máquina algorítmica se ha introducido un *estado intermedio* para que las acciones *cargar Q* e *incrementar C* (propias, respectivamente, del segundo estado y del último) se ejecuten antes de llegar a los saltos condicionales que dependen de ellas (**Q_0 = 1** y **C = 63**).

Sin dicho estado intermedio, la primera vez que se evalúe $Q_0 = 1$ se encontrará con el registro **Q** a cero (pendiente de cargar el nuevo valor de Q); asimismo, cuando, desde el último estado, se pase directamente a la condición ***C = 63*** (por ser Q_0 = **0**), el valor de **C** estará pendiente del último incremento correspondiente a dicho estado. Un solo estado adicional, previo a las dos condiciones de salto citadas, facilita, en este caso, la ejecución de las dos acciones señaladas (*cargar Q* e *incrementar C*).

Configuración microprogramada de la parte de control

El circuito de la página anterior corresponde a la forma «microcableada» de construir la parte de control, con codificación de «un solo uno» que permite identificar cada estado (y el conjunto de acciones asociadas al mismo) con un biestable y trasladar directamente el grafo de estados (especialmente si se dibuja en forma de «ordinograma») a su configuración circuital.

La parte de control puede ser construida, también, en forma «microprogramada» (ver apartado 14.3, volumen segundo), recogiendo el estado sobre un registro y utilizando un codificador para calcular el nuevo estado a partir del estado anterior y del vector de entradas; no resulta oportuna la codificación con «un solo uno» sino que, para reducir el tamaño del codificador, interesa utilizar el menor número posible de variables de estado.



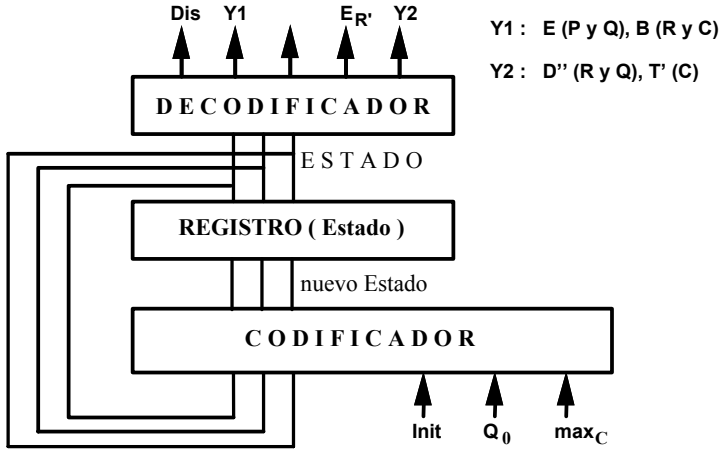
En estructura ROM, los codificadores se configuran directamente desde su tabla de conversión: su «Matriz O» coincide con la propia tabla funcional; podemos considerar que los codificadores contienen «microinstrucciones» en la siguiente forma:

- el primer codificador contiene la tabla relativa al cálculo del estado siguiente, a partir del estado anterior y el vector de entrada;
- el segundo contiene la tabla de los vectores de salida en relación con los estados.

La forma microprogramada conserva, «en forma de programa», la estructura de la parte de control, 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.

En principio, la configuración «microcableada» es más rápida (menores tiempos de propagación) y ocupa un área de integración más reducida, pero puede ser muy compleja. La forma «microprogramada» es más directa, más fácil de modificar (basta cambiar las correspondientes «microinstrucciones» en los codificadores), pero puede necesitar bloques ROM grandes.

En el caso del multiplicador de dos números binarios, el número de estados necesarios es 5, que pueden codificarse con 3 variables de estado; la parte de control requiere un codificador de 6 entradas (3 para el estado anterior y 3 para el vector de entradas) y 3 salidas para calcular el nuevo estado y un simple decodificador de 3 entradas para el vector de salida, según el diagrama de bloques siguiente:



Sistemas que actúan bajo programa

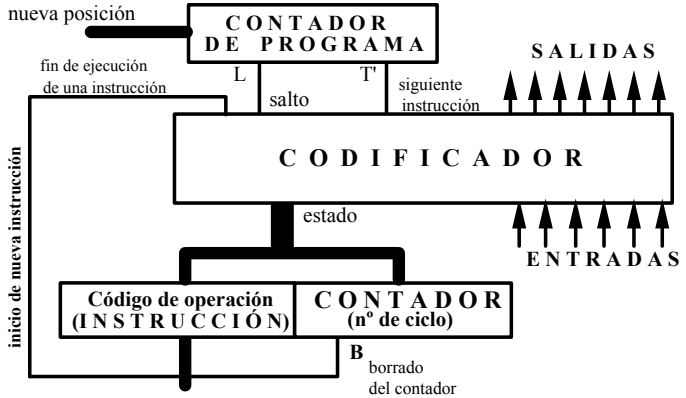
En los sistemas digitales que actúan bajo programa (procesadores programables, computadores, microprocesadores, microcontroladores,...) es clásica la separación entre la parte operativa (ALU y registros) y la parte de control; en esta segunda, el estado se encuentra conformado básicamente por el contador de programa, la instrucción que se está ejecutando y un contador de ciclos que desglosa dicha ejecución.

Dentro de una instrucción, es preciso distinguir dos partes claramente diferenciadas por su funcionalidad: el código de operación (que establece cuál es la actuación específica de la instrucción) y la dirección (o, en su caso, direcciones) del operando (o de los diversos operandos).

El código de operación de una instrucción constituye la parte más relevante del estado del procesador; junto con ella, un contador numera sucesivamente los ciclos de ejecución de la instrucción (ya que, en general, cada instrucción empleará varios ciclos de reloj para su ejecución):

estado = código de operación (de la instrucción) + **nº de ciclo** (contador)

La realización microprogramada de la máquina de estados que controla la ejecución de las instrucciones es de particular interés y el correspondiente codificador coincide con el denominado «decodificador de instrucciones».



El contador de programa señala la ubicación (en el mapa de memoria) de la siguiente instrucción a ejecutar y, en tal sentido, es parte del estado del procesador (pues es relevante en el control de la secuencia de instrucciones), pero el «estado del contador» no se realimenta hacia dentro de la parte de control.

El primer ciclo de todas las instrucciones (nº de ciclo = **0**) consiste en la búsqueda de la propia instrucción, comenzando por su código de operación. La «lectura» de una instrucción completa requiere, en ocasiones, varios ciclos ya que puede ocupar varias palabras sucesivas; tras lo cual, el contador de programa avanza una unidad, pasando a señalar la instrucción siguiente; los «saltos» se producen modificando el contenido de contador mediante «carga en paralelo» de la nueva localización.

24.4. Máquinas algorítmicas: varios ejemplos

Se detallan, a continuación, varios ejemplos de sistemas digitales relativos a cálculos numéricos: división, raíz cuadrada, conversión BCD-binario y binario-BCD, seguidos de un par de ejemplos «no aritméticos» referidos a control de cronómetros.

Nota: Todas las máquinas algorítmicas de este apartado (así como la del multiplicador de 64 bits, descrita en el apartado anterior) han sido comprobadas, a través de su descripción en VHDL y la posterior simulación de la misma (utilizando el compilador MAX+plus II de ALTERA).

24.4.1. Divisor de dos números binarios

Sean P y Q dos números binarios, de 64 y 16 dígitos, respectivamente.

Esquema de cálculo

- recorrer el dividendo **P** dígito a dígito (comenzando por el más significativo), restando el divisor **Q** cuando se pueda;
- en cada desplazamiento, añadir un **1** al resultado **R** si se efectúa la resta y un **0** cuando ésta no es posible.

Este esquema se deduce directamente del que se emplea 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.

Algoritmo

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;** Incrementar **C** en una unidad**;**

Si S ≥ Q Restar **S - Q** sobre **S**;

Desplazar **R** hacia la izquierda con entrada **1**;

Si no Desplazar **R** hacia la izquierda con entrada **0**;

Fin_de_Si;

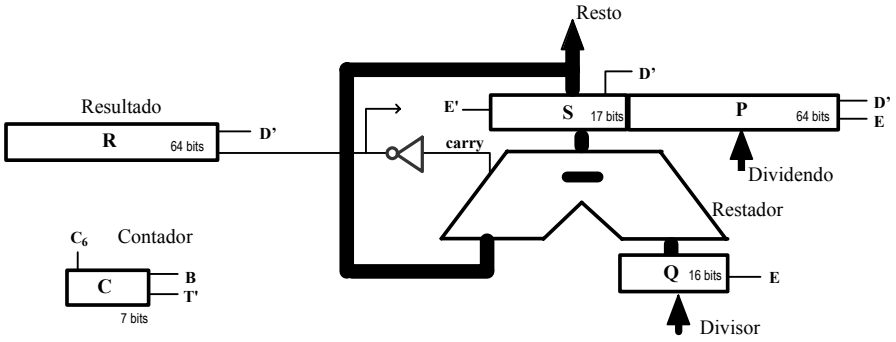
Si bit_6 (más significativo) de **C = 1** (64 ejecuciones de la repetición)

Fin del algoritmo.

Fin_de_Si;

Fin_de_Repetir (volver al inicio de la repetición)**;**

Recursos de cálculo

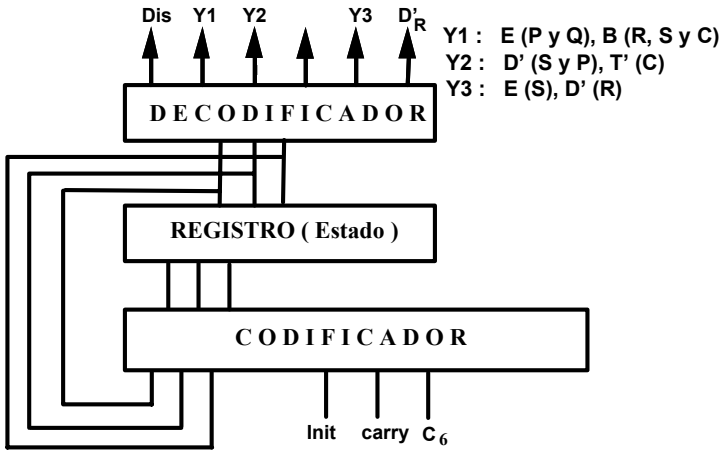


El registro **S** unido al **P**, como registro de desplazamiento **S-P**, permite ir añadiendo al «resto parcial» o «dividendo actual» **S** los dígitos del número **P** uno a uno; la entrada de control **D'** se refiere a desplazamiento hacia la izquierda (en el caso del multiplicador se utilizaba **D''** para el desplazamiento hacia la derecha).

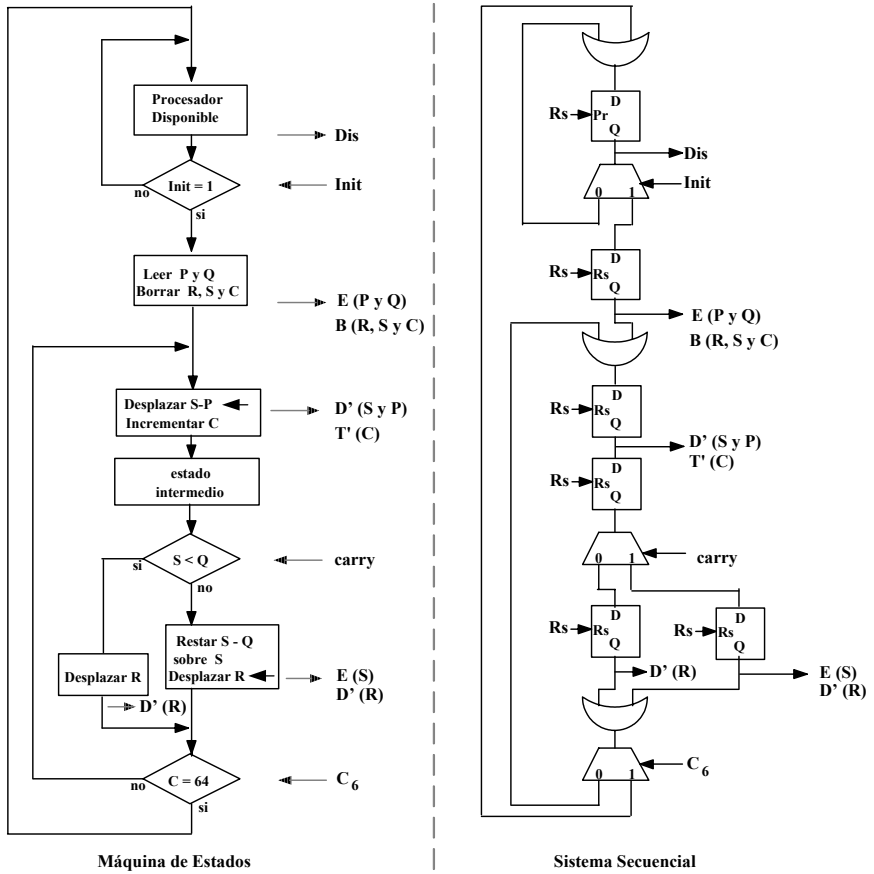
La salida de arrastre del restador (acarreo **carry**) sirve para comparar **S** (el «resto parcial») con el divisor **Q**: cuando dicha salida es **0** (**S ≥ Q**) se ejecuta la resta (se habilita el resistro **S**) y se carga un **1** (por desplazamiento) en el resultado; cuando es **1** (**S < Q**) no se efectúa la resta (habilitación del resistro **S** a **0**) y en el resultado se carga un **0**.

Configuración microprogramada de la máquina de estados

La parte de control requiere 6 estados, según la correspondiente máquina de estados representada en la figura de la página siguiente:



Máquina de estados y su configuración microcableada



Ha sido necesario introducir un *estado intermedio* antes de la condición *S > Q* para que la acción *desplazar S-P* se ejecute antes de llegar al salto condicional que depende de ella (**S < Q**). Sin dicho estado intermedio, al llegar al salto, el registro **S** no habría tomado, por desplazamiento, un nuevo dígito de **Q**.

24.4.2. Raíz cuadrada de un número binario

Sea P un número binario de 64 dígitos.

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 ésta no es posible.

Este esquema se deriva del que se emplea para hacer «a mano» la raíz cuadrada de un número 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.

Algoritmo

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**).

Al igual que en el caso anterior, 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;**

Repetir

Desplazar hacia la izquierda **S-P**; Incrementar **C** en una unidad;

Desplazar hacia la izquierda **S-P**;

Si S ≥ R&01

Restar **S - R&01** sobre **S**;

Desplazar **R** hacia la izquierda con entrada **1**;

Si no Desplazar **R** hacia la izquierda con entrada **0**;

Fin_de_Si;

Si bit$_5$ (más significativo) de **C = 1** (32 ejecuciones de la repetición)

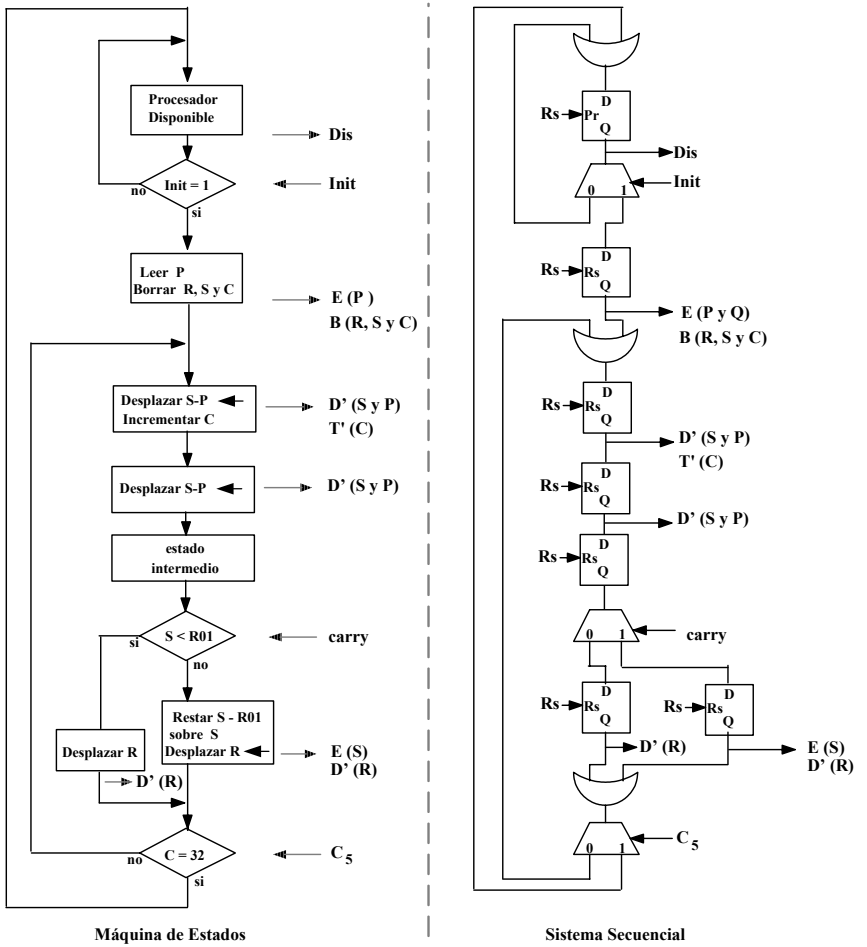
Fin del algoritmo.

Fin_de_Si;

Fin_de_Repetir (volver al inicio de la repetición)**;**

Máquina de estados y su configuración microcableada

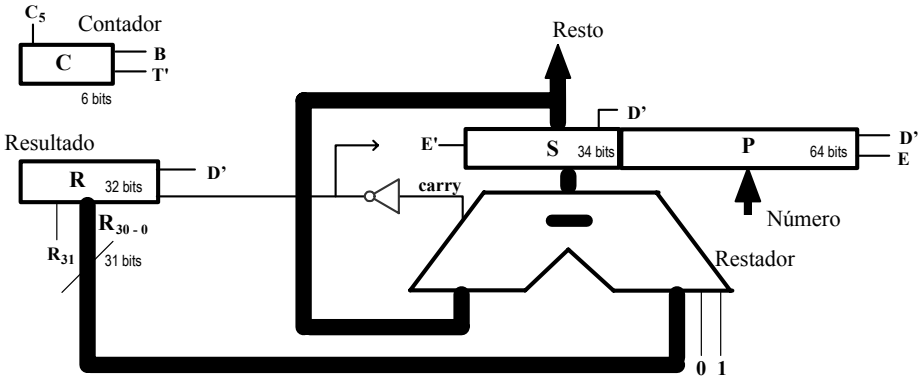
El control expresado en el algoritmo anterior requiere 6 estados (más un *estado intermedio*), según la máquina de estados representada en la figura siguiente.



Ha sido necesario introducir un *estado intermedio* antes de la condición *S < R01* para que la acción *desplazar S-P* se ejecute antes de llegar al salto condicional que depende de ella (**S < R01**); de otra forma, el registro **S** no hubiera tomado, por desplazamiento, dos nuevos dígitos de Q sino solamente uno.

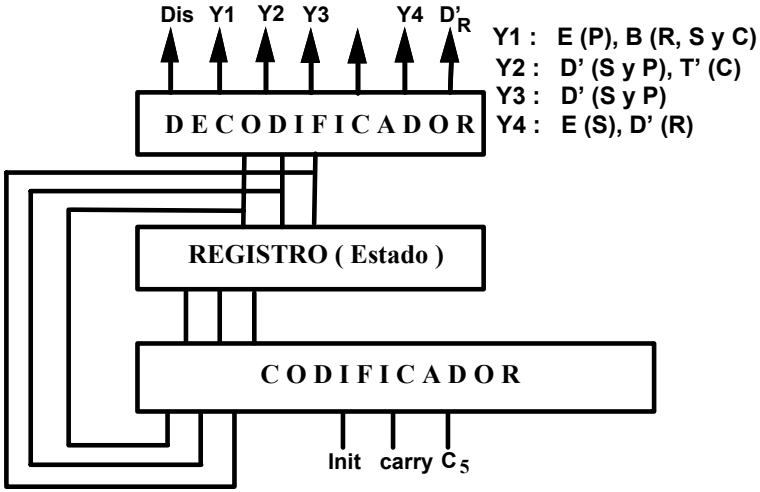
Puede ahorrarse un estado (y el consiguiente biestable) haciendo que el desplazamiento del registro **S-P** sea de dos bits en una sola vez; para ello bastará conectar la salida de cada biestable con la entrada del biestable que está dos lugares más hacia la izquierda del mismo (no del inmediatamente siguiente).

Recursos de cálculo



La entrada de control **D'** controla el desplazamiento 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**).

Configuración microprogramada de la máquina de estados



24.4.3. Conversión binario a BCD

Sea P un número binario de 64 dígitos.

Esquema de cálculo

- 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:

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

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

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.

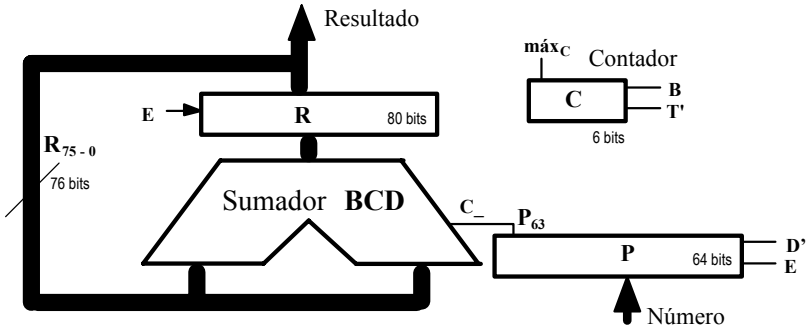
El esquema de cálculo puede ser reordenado de la siguiente manera:

- 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 (en la misma operación (**R** + **R**) a través del arrastre inicial.

Algoritmo

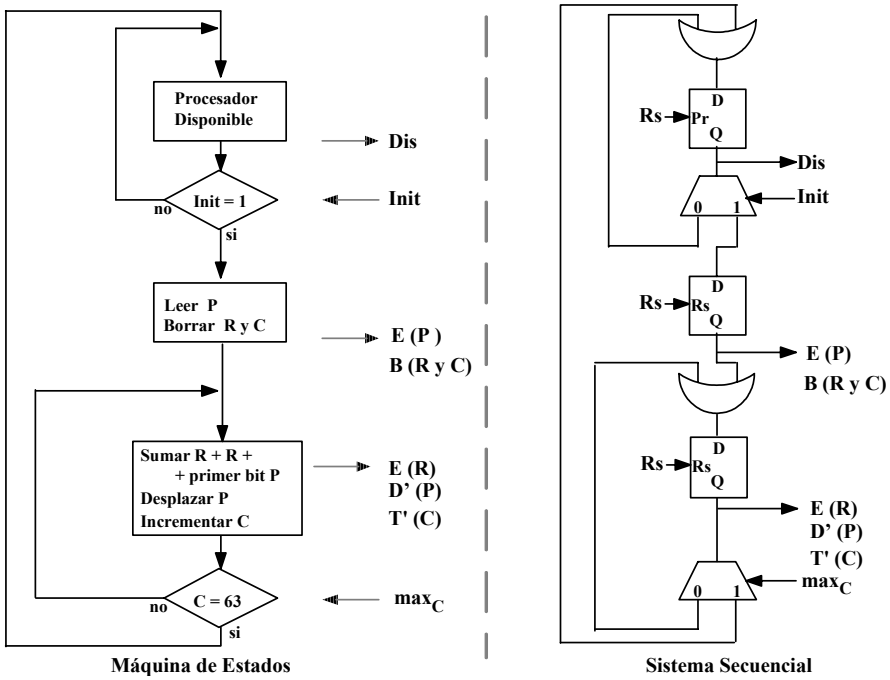
Inicio del algoritmo
Leer **P;** Borrar **R** y **C;**
Repetir Sumar (*en BCD*) **R + R + primer bit de P** en **R;**
Desplazar hacia la izquierda **P;**
Incrementar **C** en una unidad**;**
Si C = 111111 (64 ejecuciones de la repetición; la primera vez C = 0)
Fin del algoritmo. **Fin_de_Si;**
Fin_de_Repetir (volver al inicio de la repetición)**;**

Recursos de cálculo



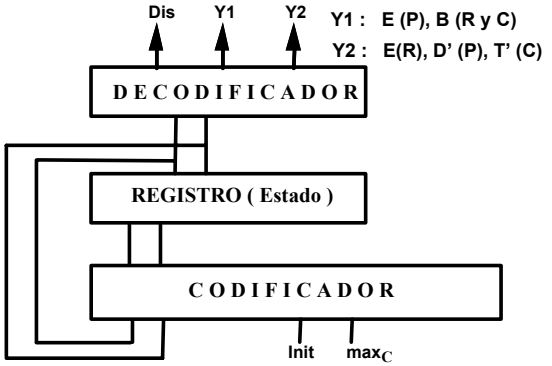
La conversión de un número binario de 64 bits a BCD ocupa un máximo de 20 dígitos BCD ($10^{20} > 2^{64}$), que equivalen a 80 bits; el número BCD (en **R**) anterior a la última suma (corresponde a 63 bits) ocupa un máximo de 19 dígitos ($10^{19} > 2^{63}$), 76 bits.

Máquina de estados y su configuración microcableada



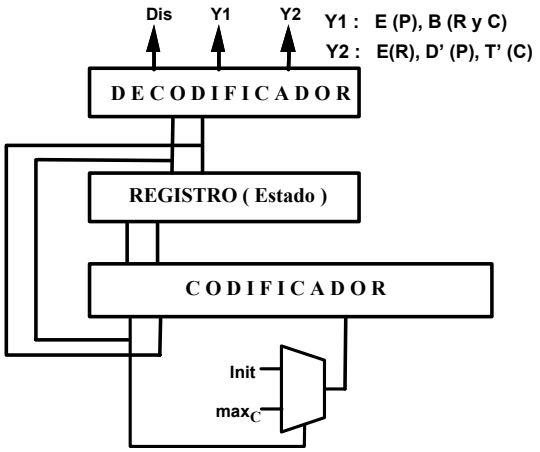
No se ha añadido ningún *estado intermedio* y, por ello, la condición *C = 63* se verifica cuando el ciclo se ha recorrido 64 veces, ya que la acción *incrementar C* se ejecuta en forma síncrona en el mismo flanco de reloj que el salto condicionado (la primera vez C = 0).

Configuración microprogramada de la máquina de estados



Habida cuenta de que, en la configuración «microcableada» el tamaño de los codificadores aumenta fuertemente con el número de sus variables de entrada (se duplica cada nueva entrada), puede resultar conveniente utilizar técnicas de multiplexado de las entradas: suele suceder que no todas las entradas actúan en todos los estados, de forma que el número efectivo de ellas puede reducirse por multiplexado.

En este caso, la entrada *Init* actúa solamente con el primer estado y la entrada *maxC* lo hace con el último estado; es posible multiplexarlas mediante la variable de estado más significativa y el codificador se reduce a la mitad de tamaño.



24.4.4. Conversión BCD a binario

Sea P un número BCD de 16 dígitos (64 bits).

Esquema de cálculo

- recorrer el número **P** de dígito en dígito (es decir, de 4 en 4 bits) comenzando por el dígito BCD más significativo y sumar (en binario) dichos cuatro bits al resultado;
- en cada desplazamiento, salvo en el último, multiplicar por diez el resultado.

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

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

El producto por 10 puede hacerse de la siguiente forma: **R** . 8 + **R** . 2 (se suma **R** desplazado 3 dígitos **R000** –que equivale, en binario, a multiplicar por 8– con **R** desplazado 1 dígito, **R0** –que equivale a multiplicar por 2–).

El esquema de cálculo anterior puede ser reordenado en la forma siguiente:

- recorrer el número **P** de cuatro en cuatro bits (comenzando por los más significativos) haciendo lo siguiente: (multiplicar **R** por 10) sumar el resultado **R** desplazado tres bits (resultado con 3 ceros añadidos al final: **R000**) con el mismo resultado desplazado un bit (resultado con 1 cero añadido detrás: **R0**); y sumar también dichos cuatro dígitos del número **P**.

Algoritmo

Se utiliza el contador **C** para contar el número de veces que se ha realizado el ciclo.

Inicio del algoritmo

Leer **P;** Borrar **R** y **C;**

Repetir

Sumar **R&000 + R&0 + 4 primeros bits de P en R;**

Si C = 1111 (15 ≡ 16 ejecuciones de la repetición; la primera vez C = 0)

Fin del algoritmo.

Fin_de_Si;

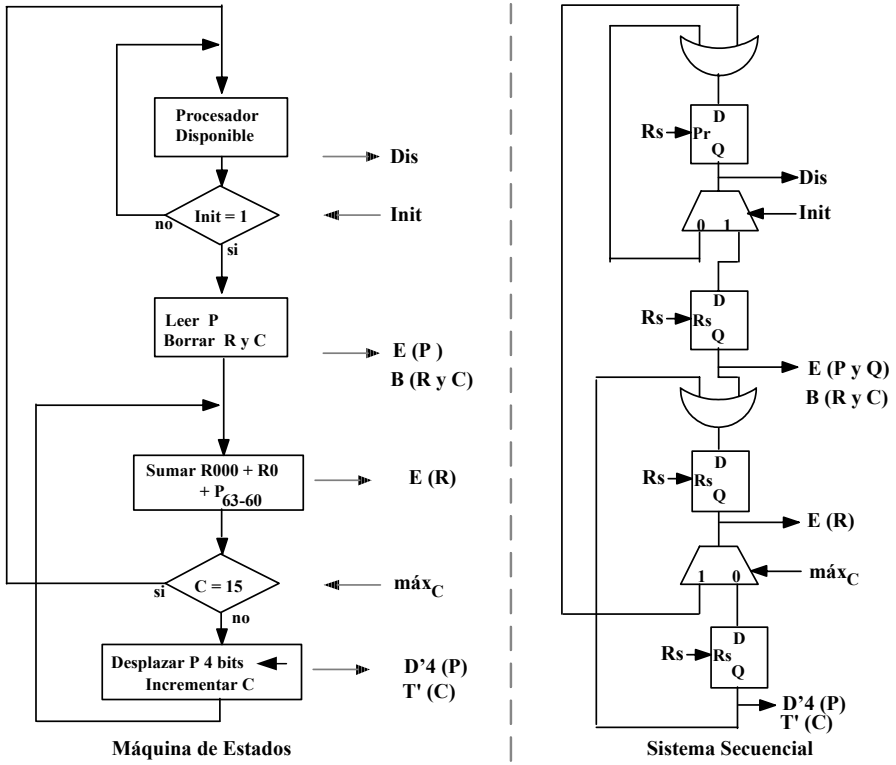
Desplazar hacia la izquierda **P** 4 dígitos;

Incrementar **C** en una unidad**;**

Fin_de_Repetir (volver al inicio de la repetición)**;**

Máquina de estados y su configuración microcableada

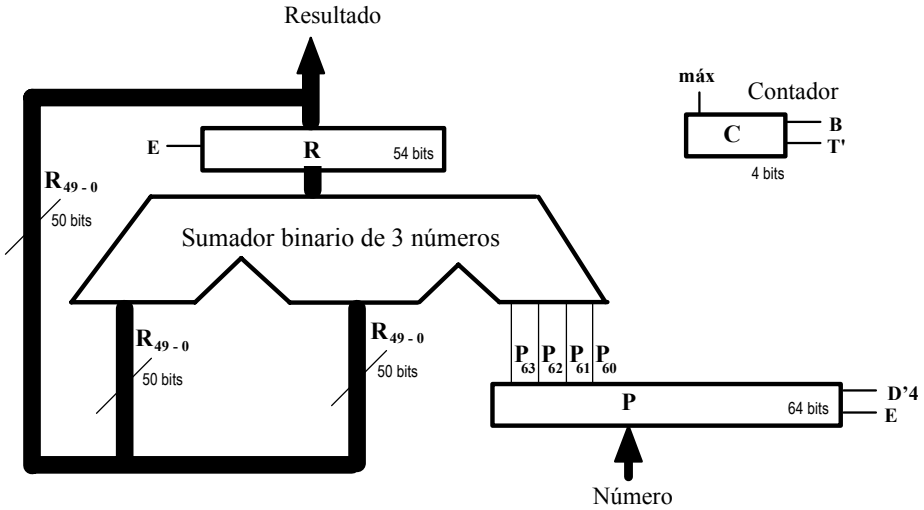
El control necesita solamente cuatro estados, conforme a la siguiente máquina algorímica:



El desplazamiento de **P** cuatro dígitos hacia la izquierda puede efectuarse en un solo estado, a través de la conexión serie adecuada de los biestables del registro **P**; si fuera un registro de desplazamiento normal, requeriría cuatro estados para hacerlo bit a bit. De esta forma, el cálculo es más rápido, la parte de control requiere menor número de estados y el circuito secuencial menor número de biestables.

No ha sido necesario introducir ningún *estado intermedio* pues la condición *C = 15* no se encuentra inmediatamente detrás de la acción *incrementar C*, sino que hay un estado «en medio» que permite ejecutar el incremento.

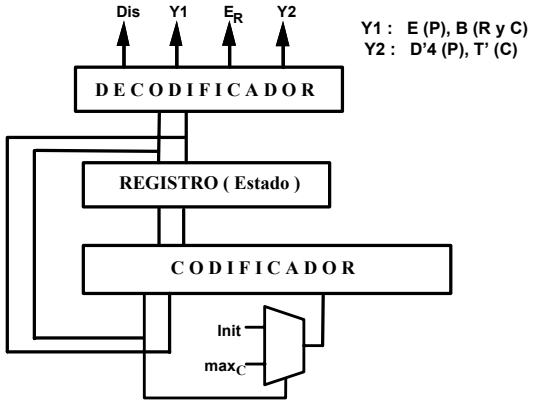
Recursos de cálculo



La conversión de un número de 16 dígitos BCD a binario ocupa un máximo de 54 bits ($2^{54} > 10^{16}$); el número binario (en **R**) anterior a la última suma (el correspondiente a 15 dígitos BCD) ocupa un máximo de 50 bits ($2^{50} > 10^{15}$).

La entrada **D'4** controla el desplazamiento cuatro bits hacia la izquierda: para ello es necesario que las conexiones serie del registro de desplazamiento sean de la salida de cada biestable con la entrada del situado cuatro lugares más a la izquierda del mismo.

Configuración microprogramada de la máquina de estados



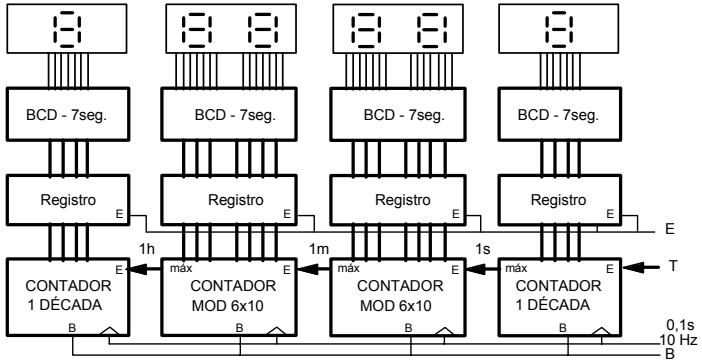
Las cinco máquinas algorítmicas descritas anteriormente se refieren a operaciones aritméticas. A fin de dejar constancia de que este método de división *parte operativa – parte de control* es de tipo general (y no solamente para cálculos matemáticos) se incluyen, a continuación, dos ejemplos de máquinas algorítmicas relativas a control de cronómetros.

24.4.5. Control de un cronómetro mediante pulsadores

*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 poner a cero el cronómetro; 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**.*

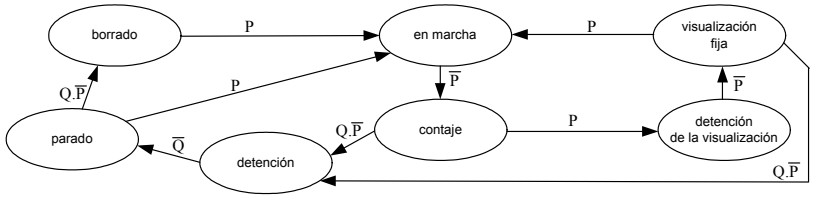
Parte operativa

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).

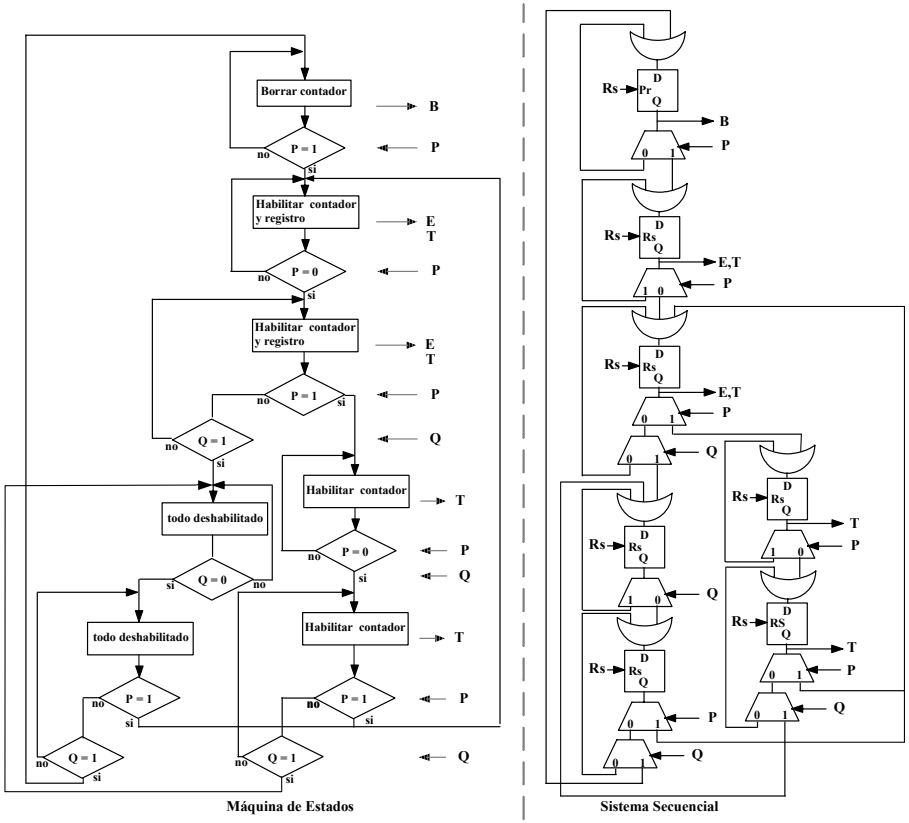


Parte de control

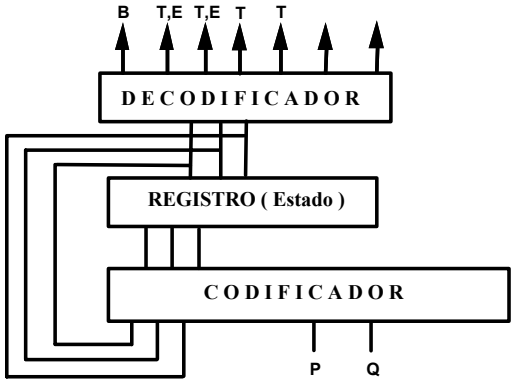
El control de este sistema responde al siguiente grafo de estados (se ha dado prioridad al pulsador **P** respecto a **Q**):



Máquina de estados y su configuración microcableada



Configuración microprogramada de la máquina de estados



24.4.5. Control de un cronómetro para medir tiempos de 8 nadadores

*Interesa 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).*

Parte operativa

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.

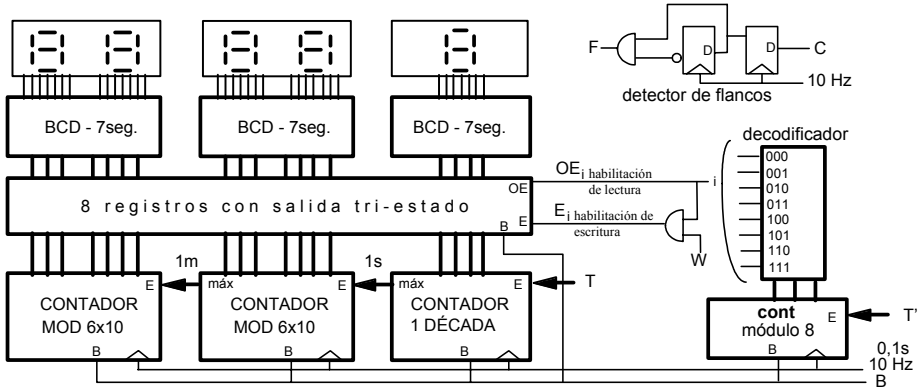
Los ocho registros han de tener sus entradas conectadas a las salidas del contador y sus salidas deben poder conectarse, en forma multiplexada, al visualizador. Para facilitar el diseño (y el dibujo del mismo) el multiplexado se realiza dotando a los registros de salida tri-estado, de forma que todos ellos estarán conectados al visualizador y, en cada momento, se activan las salidas del registro que corresponda. Cada uno de los registros 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, en cada momento, el registro al que le corresponde almacenar la salida del contador de tiempo (situación de escritura) o representar su valor en el visualizador (lectura).

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 $C - \overline{C}$.

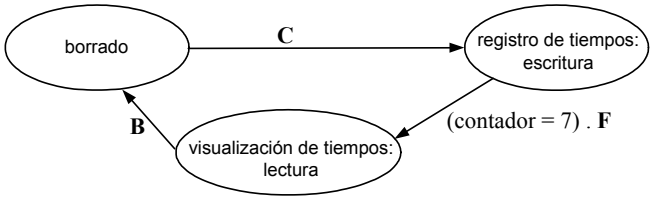
El contador auxiliar módulo 8 tendrá una entrada de habilitación para el contaje de registros **T'** y la entrada común de borrado **B**; su habilitación **T'** se producirá en ambas situaciones (escritura y lectura) pero solamente cuando se detecta flanco **F** de la señal **C**: $T' = (\text{escritura} + \text{lectura}) . F$.

De manera que la parte operativa ampliada con el contador-selector de registro y con el detector de flancos quedará en la forma representada en la figura siguiente:



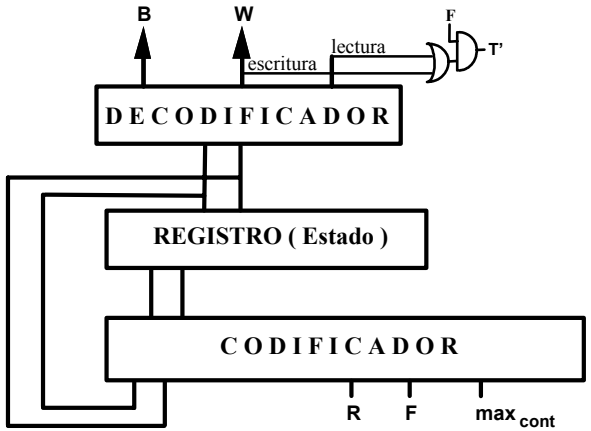
Parte de control

Un posible grafo de estados para representar el control de este sistema es el siguiente:

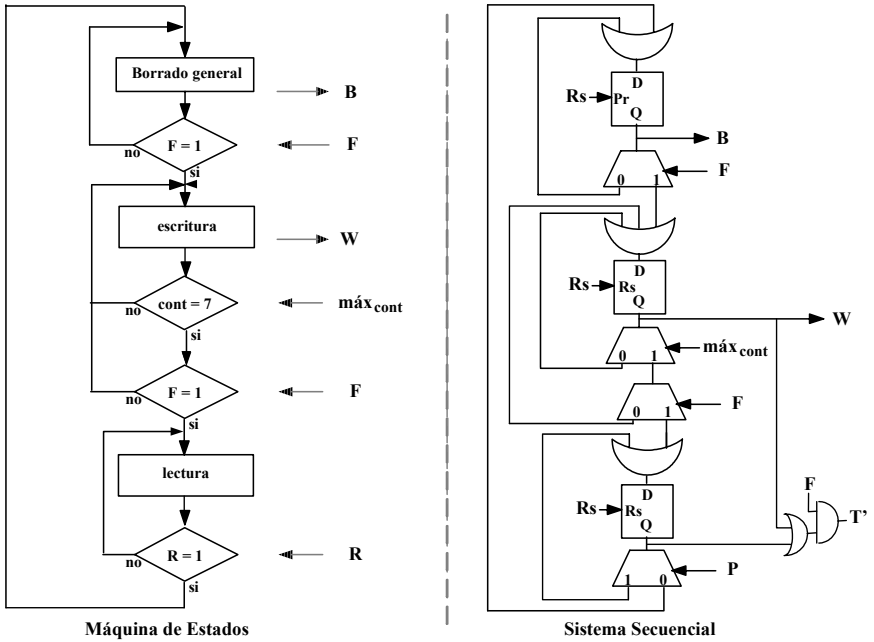


La escritura corresponde al registro de tiempos de los ocho nadadores y la lectura a la posterior visualización cíclica de los mismos. El contador auxiliar, que recorre los ocho registros, debe «contar» tanto en el estado de lectura como en el de escritura cuando se detecta un flanco del pulsador **C**.

Configuración microprogramada de la máquina de estados

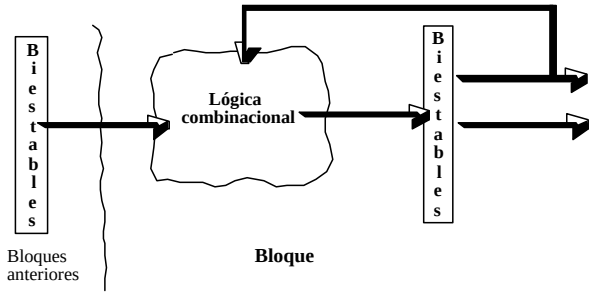


Máquina de estados y su configuración microcableada



24.5. Dividir en partes los tiempos de retraso: segmentación

Los bloques o módulos constitutivos que conforman un sistema síncrono están configurados por una parte combinacional situada entre dos registros (o conjuntos de biestables), según la figura siguiente (véase el apartado 15.3, segundo volumen, páginas 111-114):



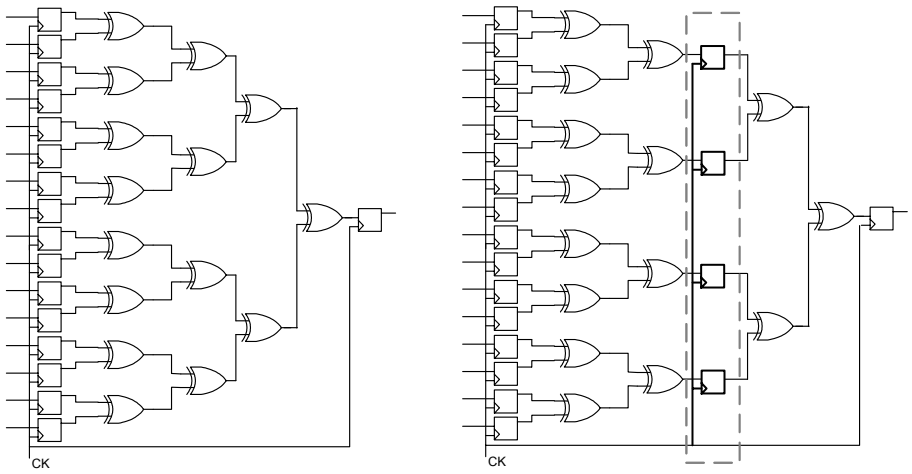
La velocidad de trabajo de uno de estos módulos queda limitada por el tiempo de propagación de los diversos «caminos» entre biestables, según la expresión:

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

donde $\mathbf{t_p}$ es el tiempo de propagación del primer biestable y $\sum \mathbf{t'_p}$ la suma de tiempos de propagación en la parte combinacional que atraviesa el camino, y $\mathbf{t_s}$ el tiempo de anticipación que requiere el segundo biestable. Esta desigualdad impone un límite a la frecuencia máxima de reloj en que el módulo puede funcionar correctamente (y, con ello, limita la frecuencia de reloj del sistema digital).

Si la parte combinacional es amplia (en el sentido de muchas puertas sucesivas), la suma de retrasos en ella $\sum \mathbf{t'_p}$ es apreciable y requiere un período de reloj largo, limitando fuertemente la velocidad de trabajo del sistema digital. Un recurso de diseño que permite aumentar dicha velocidad consiste en dividir la parte combinacional en segmentos (en *rodajas*) insertando registros intermedios; esta técnica, conocida con la denominación de *pipiline* o segmentación, es aplicable solamente a bloques sin realimentación.

Consideremos un primer ejemplo de aplicación de la segmentación a un bloque muy simple: se trata de calcular la paridad de números de 16 dígitos binarios, a través de una sucesión de puertas "**o-exclusiva**", según las figuras siguientes:



Supongamos que los tiempos de anticipación y de propagación de los biestables son iguales entre sí e iguales, asimismo, al tiempo de propagación de cada una de las puertas "**o-exclusiva**". En la figura de la izquierda, el período de reloj deberá ser mayor que

$$\mathbf{t_p} + 4.\mathbf{t_p} + \mathbf{t_s} = 6.\mathbf{t_p}\,; \quad \mathbf{f_{CK}} < 1/(6.\mathbf{t_p});$$

en la figura de la derecha, se ha insertado un registro en medio de la parte combinacional, de forma que el período de reloj deberá ser, en este caso, mayor que

$$\mathbf{t_p} + 2.\mathbf{t_p} + \mathbf{t_s} = 4.\mathbf{t_p}\,; \quad \mathbf{f_{CK}} < 1/(4.\mathbf{t_p});$$

se ha conseguido una ganancia en la frecuencia de reloj según un factor 1,5 (4 = 6/1,5).

Ha de tenerse en cuenta que, para obtener el resultado (la paridad del número de 16 bits), el segundo circuito requiere un ciclo más de reloj. La frecuencia de reloj puede ser mayor pero el cálculo de la paridad del primer número que llega al registro inicial tarda dos ciclos (en lugar de uno solo); en cambio, los resultados de paridad de los números siguientes (supuesto que cada ciclo el registro inicial reciba un nuevo número) se obtienen en ciclos de reloj sucesivos. Para una serie de 100 números binarios seguidos son precisos 101 ciclos de reloj: dos para el primer número y uno más para cada uno de los siguientes.

Este aumento de un ciclo de reloj, debido al registro que se ha añadido al dividir la parte combinacional en dos segmentos (dos «rodajas»), ha de ser tenido en cuenta en el diseño del resto del sistema digital que utilice el cálculo de la paridad.

Pueden incluirse tres registros intermedios (en lugar de uno solo), de forma que cada segmento sea de profundidad igual a una puerta, es decir, esté formado por un grupo de puertas "**o-exclusiva**" en paralelo y, entonces, el período de reloj deberá ser mayor que

$$\mathbf{t_p} + 1.\mathbf{t_p} + \mathbf{t_s} = 3.\mathbf{t_p}\,; \quad \mathbf{f_{CK}} < 1/(3.\mathbf{t_p});$$

lo cual supone una ganancia en la frecuencia de reloj según un factor 2 (3 = 6/2), en relación con el circuito inicial (sin ningún registro intermedio). En este caso, se añaden 2 nuevos ciclos de reloj para obtener el resultado: la paridad del primer número binario requiere 4 ciclos de reloj y, luego, cada ciclo de reloj suministra una nueva paridad; para una serie de 100 números seguidos son precisos 103 ciclos de reloj.

La técnica de segmentación se aplica a bloques síncronos cuya parte combinacional presenta altos tiempos de propagación y tal aplicación es efectiva cuando dichos bloques actúan repetitivamente con vectores de entrada sucesivos; en tal caso, la reducción del tiempo global es muy significativa. En el ejemplo considerado, para el cálculo de paridad de series de 100 números seguidos, resultan los siguientes tiempos mínimos:

- sin segmentación: $T_{CK} > 6.t_p$ tiempo total: $100.T_{CK} \sim 600.t_p$
- con un registro intermedio: $T_{CK} > 4.t_p$ tiempo total: $101.T_{CK} \sim 404.t_p$
- con segmentación total: $T_{CK} > 3.t_p$ tiempo total: $103.T_{CK} \sim 309.t_p$.

Un segundo ejemplo, algo más complejo, servirá para poner de manifiesto la potencia y, también, los requisitos de la segmentación.

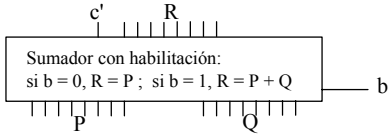
Un multiplicador de números de ocho dígitos, **A** x **B**, puede construirse mediante una máquina algorítmica, en la forma descrita en el apartado anterior (páginas 107 a 112), pero esta configuración secuencial resulta relativamente lenta: requiere 26 ciclos de reloj.

Como referencia cuantitativa supongamos que los tiempos de anticipación y de propagación de los biestables son iguales entre sí y que el tiempo de propagación de un sumador de ocho dígitos es unas tres veces mayor (el sumador estará formado por ocho celdas sumadoras sucesivas, cuyo arrastre ha de propagarse a través de ellas); en tal caso el tiempo de ciclo de reloj deberá ser mayor que

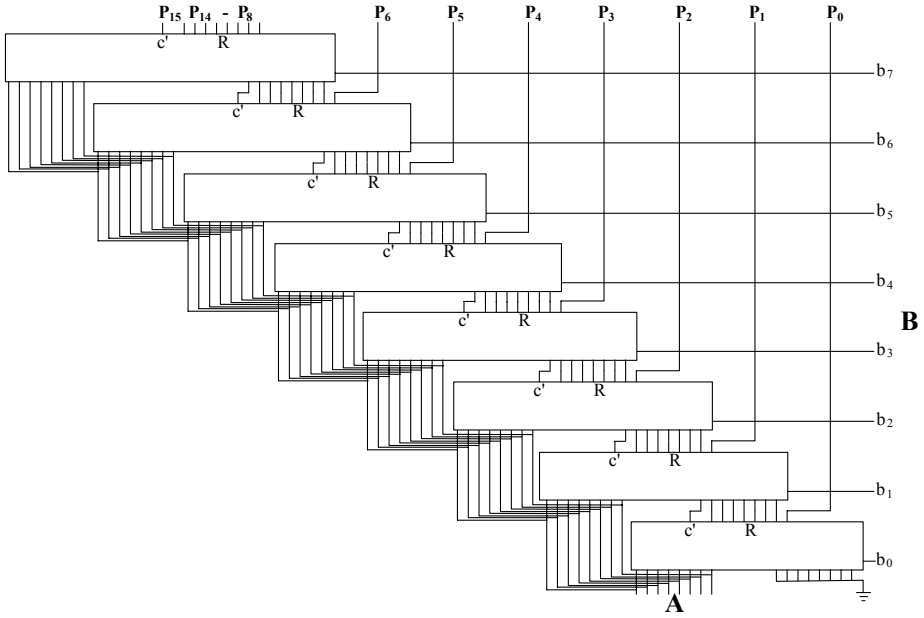
$$\mathbf{t_p} + 3.\mathbf{t_p} + \mathbf{t_s} = 5.\mathbf{t_p}\,; \quad \mathbf{f_{CK}} < 1/(5.\mathbf{t_p});$$

y el tiempo de cálculo de la multiplicación será (en el peor de los casos): $26.\mathbf{t_{CK}} \sim 130.\mathbf{t_p}$.

Otra posibilidad es utilizar ocho sumadores (con entrada de habilitación respecto a la suma) para efectuar la multiplicación en la forma siguiente:



Sumador con habilitación



Multiplicador de A x B (de 8 dígitos)

El tiempo de propagación de este bloque será del orden de 8.4.$\mathbf{t_p}$ y el período de reloj (de este multiplicador, situado entre sendos registros) deberá ser mayor que

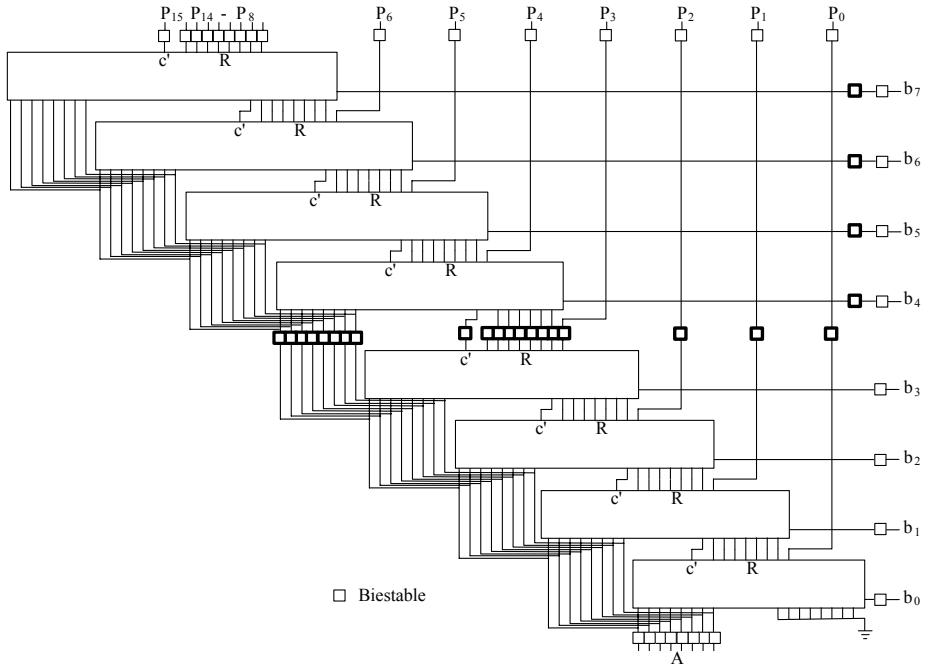
$$\mathbf{t_p} + 8.3.\mathbf{t_p} + \mathbf{t_s} = 26.\mathbf{t_p}\ ; \quad \mathbf{f_{CK}} < 1/(26.\mathbf{t_p})$$

con un tiempo de cálculo (1 ciclo: 26.$\mathbf{t_p}$) cinco veces más pequeño que el correspondiente al anterior multiplicador (130.$\mathbf{t_p}$) configurado como máquina algorítmica.

En el caso de segmentar en dos «rodajas», es decir, de insertar un registro en mitad del multiplicador (ver figura siguiente), el período de reloj deberá ser mayor que

$$\mathbf{t_p} + 4.3.\mathbf{t_p} + \mathbf{t_s} = 14.\mathbf{t_p}\ ; \quad \mathbf{f_{CK}} < 1/(14.\mathbf{t_p});$$

una frecuencia de reloj cercana al doble de la anterior.



Multiplicador de 8 dígitos segmentado en dos partes

En la figura anterior, además de los biestables propios de la segmentación, se han incluido los registros de entrada de los dos números **A** y **B** y de salida **P**.

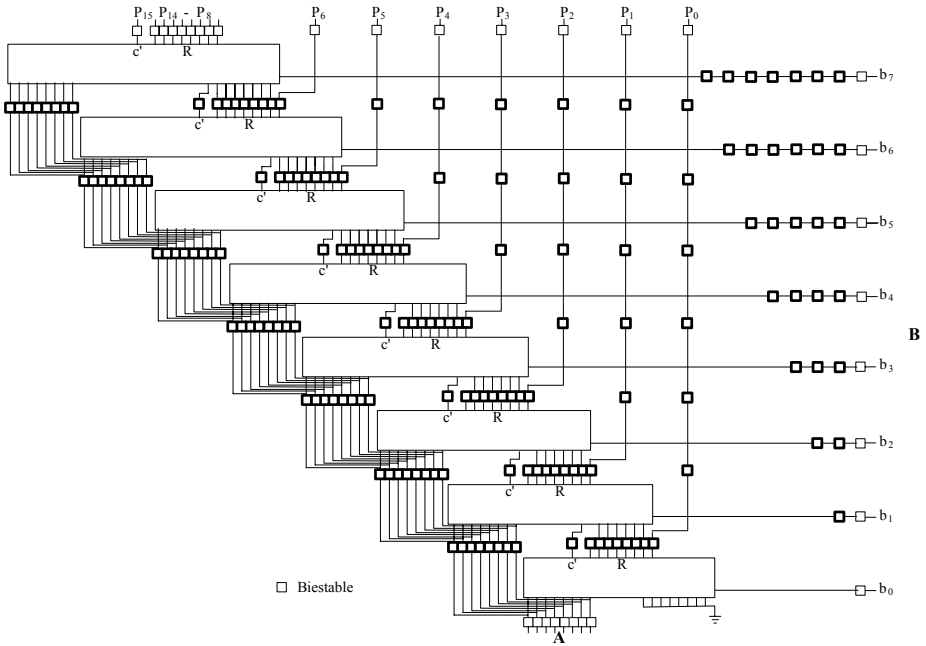
Obsérvese que es necesario añadir biestables, tanto en las salidas del cuarto sumador, como en las salidas $\mathbf{P_3 - P_0}$ de los anteriores para retrasarlas un ciclo, acompasándolas a las de los sumadores siguientes; también, hay que añadirlos en el número **A** y en los dígitos $\mathbf{b_7 - b_4}$, en cuanto a entradas a dichos sumadores. En total, han sido precisos 24 biestables para segmentar en dos partes el multiplicador.

De nuevo cabe resaltar que, para obtener el resultado de la multiplicación en este circuito segmentado, se requiere un ciclo más de reloj; la frecuencia puede duplicarse respecto al circuito anterior pero el cálculo del producto de los dos primeros números tarda dos ciclos; en cambio, los productos de las parejas de números siguientes (supuesto que en cada ciclo los registros iniciales reciban nuevos operandos) se obtienen en ciclos de reloj sucesivos. Para una serie de 100 números binarios seguidos se necesitan 101 ciclos de reloj: dos para el primer producto y uno más para cada uno de los siguientes.

Una segmentación total (es decir, cada sumador un segmento o «rodaja») requiere 168 biestables de segmentación (ver figura siguiente) con un período de reloj mayor que

$$\mathbf{t_p} + 1.3.\mathbf{t'_p} + \mathbf{t_s} = 5.\mathbf{t_p}\ ; \quad \mathbf{f_{CK}} < 1/(5.\mathbf{t_p});$$

frecuencia unas tres veces más rápida que la del circuito anterior y cino veces mayor que la del circuito sin segmentación.



Multiplicador de 8 dígitos segmentado en 8 partes

En este caso de segmentación completa, se necesitan ocho ciclos de reloj para obtener el resultado: el producto de la primera pareja de números binarios requiere tales ocho ciclos y, luego, en cada ciclo de reloj se obtiene un nuevo producto; para una serie de 100 números seguidos son precisos 107 ciclos de reloj.

Comparando las cuatro posibilidades descritas de realizar el producto, para series de parejas de 100 números sucesivos:

- máquina algorítmica: $T_{CK} > 5.t_p$ 26 ciclos de reloj para cada producto
  tiempo total: $100.26.T_{CK} > 13000.t_p$
- sin segmentación: $T_{CK} > 26.t_p$ tiempo total: $100.T_{CK} > 2600.t_p$
- con un registro intermedio: $T_{CK} > 14.t_p$ tiempo total: $101.14.T_{CK} > 1414.t_p$
- con segmentación total: $T_{CK} > 5.t_p$ tiempo total: $107.5.T_{CK} > 535.t_p$.

Un bloque síncrono (una parte combinacional situada entre dos registros o conjuntos de biestables), *sin realimentación* (sus biestables de salida no se realimentan sobre el propio bloque), puede segmentarse, es decir, dividirse en varios bloques síncronos sucesivos, de menor profundidad; lo cual permite aumentar la velocidad de trabajo del sistema, su frecuencia de reloj, a costa de insertar un amplio número de biestables.

Ciertamente el mayor número de biestables incrementa en forma importante el área de silicio, en el caso de ASICs, o el número de celdas necesarias, caso de FPGAs. Pudiera parecer que también aumenta el consumo del circuito, al existir más biestables conmutando en cada flanco de reloj; sin embargo, a igualdad de frecuencia, suele suceder que la segmentación disminuya el consumo, pues la división en segmentos de la parte combinacional limita la propagación de los espurios producidos en la conmutación, que son responsables de buena parte del consumo cuando hay muchas puertas seguidas.

La segmentación (*pipeline*) es un recurso de división en partes utilizado para conseguir mayores velocidades en procesamientos que actúan sobre series de datos sucesivos; en el diseño del sistema global, ha de tenerse en cuenta que cada división (cada nuevo segmento) introduce un registro y un ciclo adicional de reloj.

Lo que hace la segmentación es aplicar una concurrencia de tareas, poniendo a operar en paralelo a los diversos segmentos, sin tener que esperar, cada uno de ellos, a que los anteriores hayan finalizado su cálculo.

Esta concurrencia se aplica de formas diversas; por ejemplo, en los procesadores que actúan bajo programa la ejecución de instrucciones, mediante una segmentación del tipo siguiente, permite el tratamiento simultáneo de cinco instrucciones de operación sobre dos operandos:

búsqueda de instrucción	lectura de operando A	lectura de operando B	ejecución de operación	escritura de resultado
instrucción n + 4	instrucción n + 3	instrucción n + 2	instrucción n + 1	instrucción n.