TEMA VI: DISEÑO SECUENCIAL MSI

Al igual que sucedía con los circuitos combinacionales, existen macrofunciones secuenciales que son implementadas en un solo integrado. Por lo tanto, estos circuitos, junto con los MSI combinacionales, permiten la realización del diseño de circuitos digitales con elementos MSI.

1. Dispositivos MSI Secuenciales.

Básicamente existen dos tipos de dispositivos MSI secuenciales, registros y contadores. A continuación serán tratados en más profundidad.

1.1. Registros.

El elemento fundamental de los sistemas secuenciales (que no está presente en los sistemas puramente combinacionales) es el elemento de memoria, ya sea transparente, latch o flip-flop. El registro es la versión MSI de estos elementos, y se puede definir como:

> Un **registro** es el elemento capaz de almacenar varios bits, en general una palabra

Por lo tanto, un elemento de memoria se puede tratar como un registro de un bit. De aquí podemos concluir que un registro de n bits estará formado por n elementos de memoria en paralelo, como se muestra en la figura 6.1. De aquí en adelante utilizaremos flip-flops por ser los más empleados, no obstante podemos encontrar versiones similares utilizando latches.

Si analizamos este circuito, podemos obtener que cuando hay un cambio de estado (la señal clk sufre una transición de subida) las salidas obtienen el valor de las entradas, es decir,

$$\text{Para todo i, cuando clk sube} \Rightarrow Q_i = D_i$$

En estos elementos solamente se puede almacenar una palabra. Sería interesante el almacenamiento de varias palabras, por lo que surgieron los **registros de desplazamientos** (*shift register*). Este tipo de registro es capaz de almacenar más de una palabra, la cual suele tener un tamaño de un bit. Un ejemplo de este tipo de registro se muestra en la figura 6.2. A medida que se producen transiciones de subida en la señal clk, el dato se va desplazando por los diferentes biestables, comportamiento que le ha dado su nombre. Este bloque, con la restricción de tener únicamente como salida la señal Q3, se dice que muestra una arquitectura **FIFO** (First In First Out, es decir, el primer dato que entra es el primero que sale).

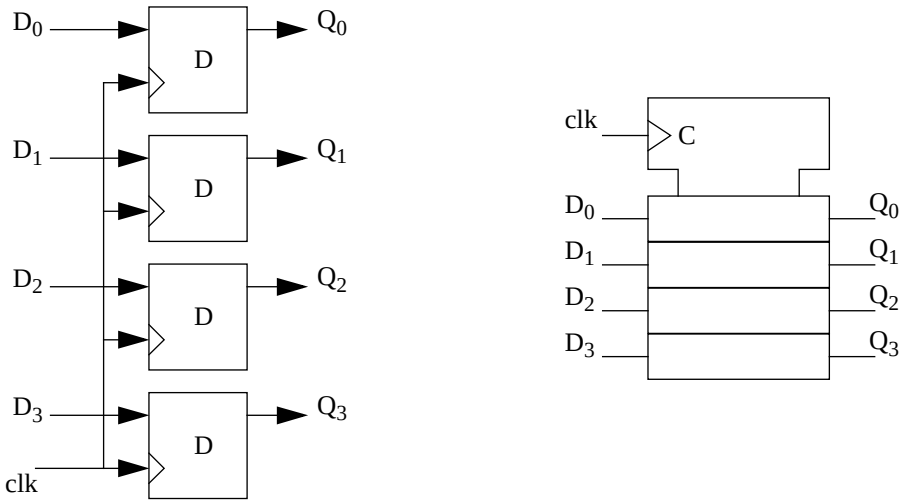


Figura 6.1.- Registro de cuatro bits, junto a su símbolo.

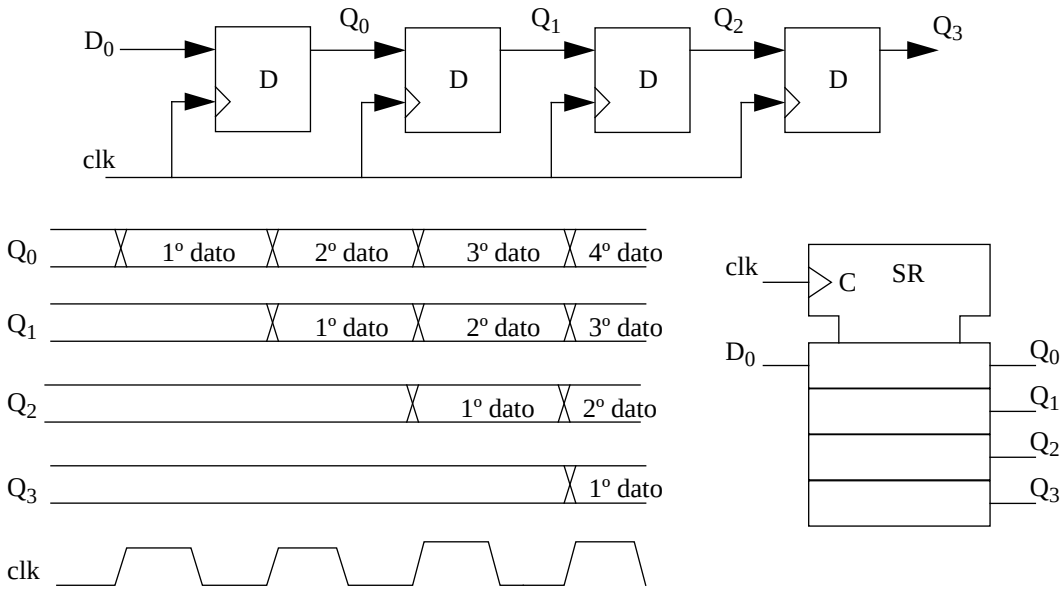


Figura 6.2.- Registro de desplazamiento de cuatro bits, junto a su comportamiento y símbolo.

Con esta solución, si deseamos inicializar el registro de desplazamiento de n bits, necesitaremos n pulsos de la señal de almacenamiento, clk. Este tiempo puede ser demasiado restrictivo, por lo que existen registros de desplazamientos con carga paralela. Estos últimos pueden ser inicializados en un solo ciclo de reloj, independientemente del número de bits que tenga el registro. Un posible esquema lógico de un registro de desplazamiento con carga paralela puede ser el mostrado en la figura 6.3. El funcionamiento de este registro es el siguiente:

- Cuando la señal de control SH/$\overline{\text{LD}}$ se encuentra a nivel bajo, hemos seleccionado la carga paralela. Por lo tanto, en el siguiente pulso de la señal de reloj se cumplirá que todo $Q_i = D_i$.

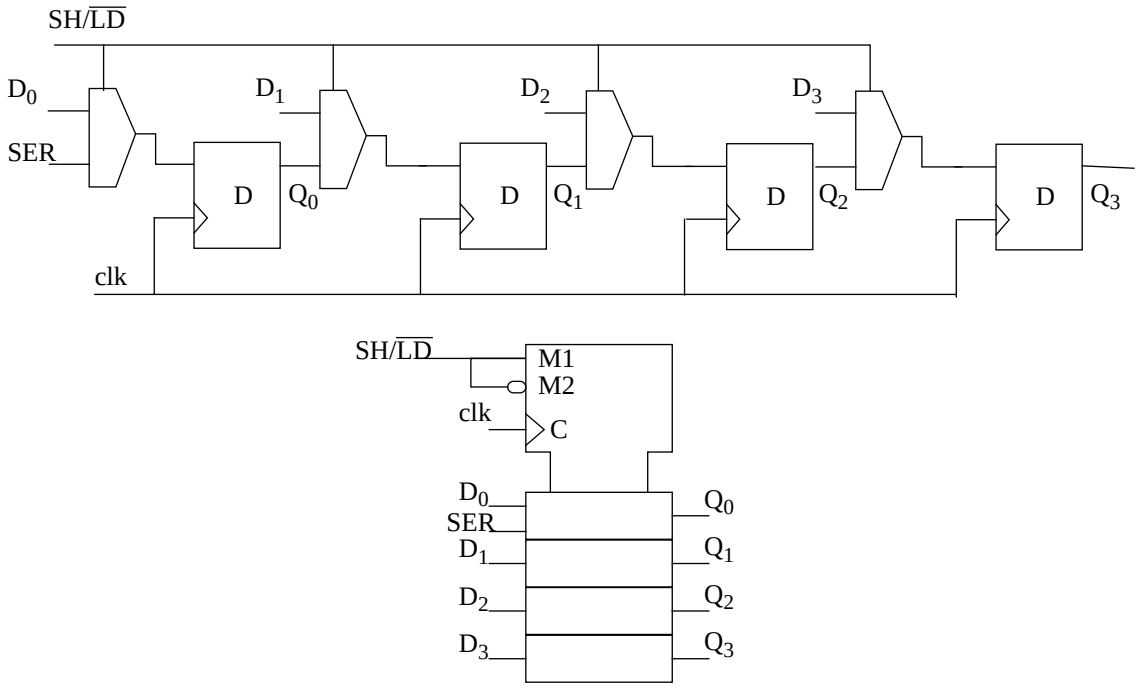


Figura 6.3.- Registro de desplazamiento con carga paralela de cuatro bits, junto a su símbolo.

- En cambio, cuando la señal de control SH/$\overline{LD}$ se encuentre a nivel alto, estaremos seleccionando el desplazamiento hacia la derecha a partir de la señal SER.

Hasta ahora sólo hemos considerado el desplazamiento en un solo sentido. No obstante también es interesante obtener un desplazamiento en los dos posibles sentidos. Debido a esta reflexión surgió el denominado **registro de desplazamiento universal**. Este registro muestra las siguientes operaciones:

- Carga paralela,
- desplazamiento hacia la izquierda,
- desplazamiento hacia la derecha y
- no operación.

En la figura 6.4 mostramos un posible esquema lógico de este último registro que engloba a todos los demás. En éste se encuentran dos señales de control, S_1 y S_0, necesarias para poder controlar las cuatro operaciones de las que dispone. En la tabla 6.1 se muestra la codificación de las señales de control que corresponde a cada una de las operaciones que realiza este tipo de registro.

Este registro es utilizado para generar una arquitectura **LIFO** (Last In First Out, el último dato que entra es el primero que sale). Esta arquitectura, también denominada pila, es como un pozo o pila en el que sólo podemos acceder al elemento superior, ya sea para colocar otro encima o para sacarlo, como mostramos en la figura 6.5. Por lo tanto, cuando queremos escribir en la memoria LIFO, debemos realizar un desplazamiento hacia la derecha; mientras que si queremos realizar una lectura, debemos realizar un desplazamiento hacia la izquierda

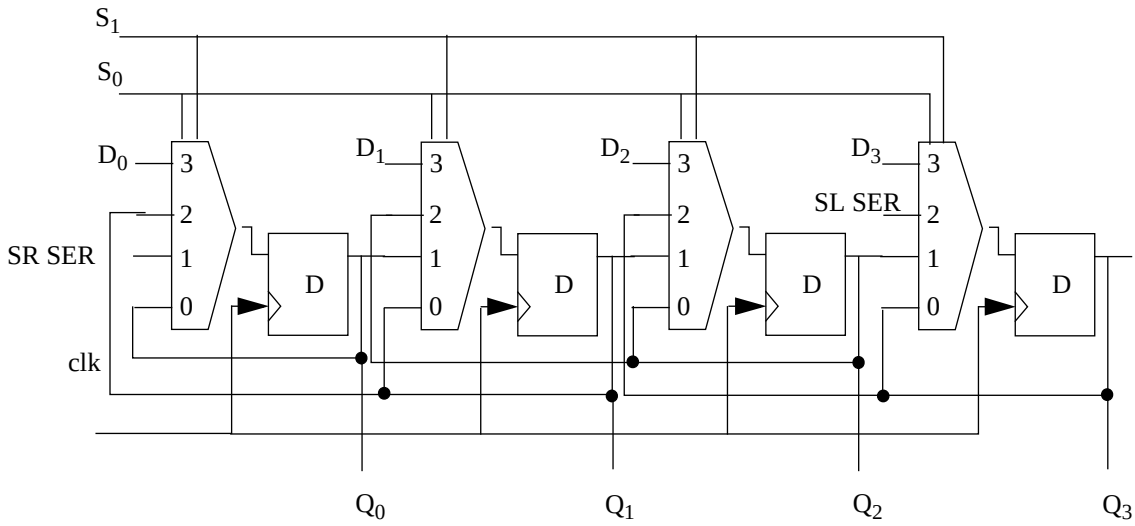


Figura 6.4.- Registro de desplazamiento universal.

S_1 S_0	Operación
00	No operación, la salida del biestable permanece constante al tenerla conectada a la entrada.
01	Desplazamiento hacia la derecha, la salida del biestable anterior está conectado a la entrada del actual.
10	Desplazamiento hacia la izquierda, la salida del biestable posterior está conectado a la entrada del actual.
11	Carga paralela, la entrada del biestable está conectado a la entrada paralela.

Tabla 6.1. Codificación de las operaciones del registro de desplazamiento universal.

para que el biestable accesible (el primero) tenga un dato válido. Un posible esquema lógico de esta arquitectura es mostrado en la figura 6.5. En esta arquitectura es necesario añadir una unidad de control para detectar posibles situaciones no deseadas, como puede ser el desbordamiento (intentar escribir más información de la que se puede almacenar) o intentar leer cuando no existe ningún dato almacenado.

1.2. Contadores.

Un contador se puede definir de la siguiente forma:

un **contador** es un circuito secuencial que realiza una o varias de las siguientes funciones:

- cuenta el número de pulsos recibidos y almacena un número que representa dicha cuenta,
- proporcionan un tren de pulsos obtenidos a partir de la entrada, pero a una frecuencia menor, y
- proporciona una secuencia de patrones binarios para aplicaciones tales como direccionamiento de memoria.

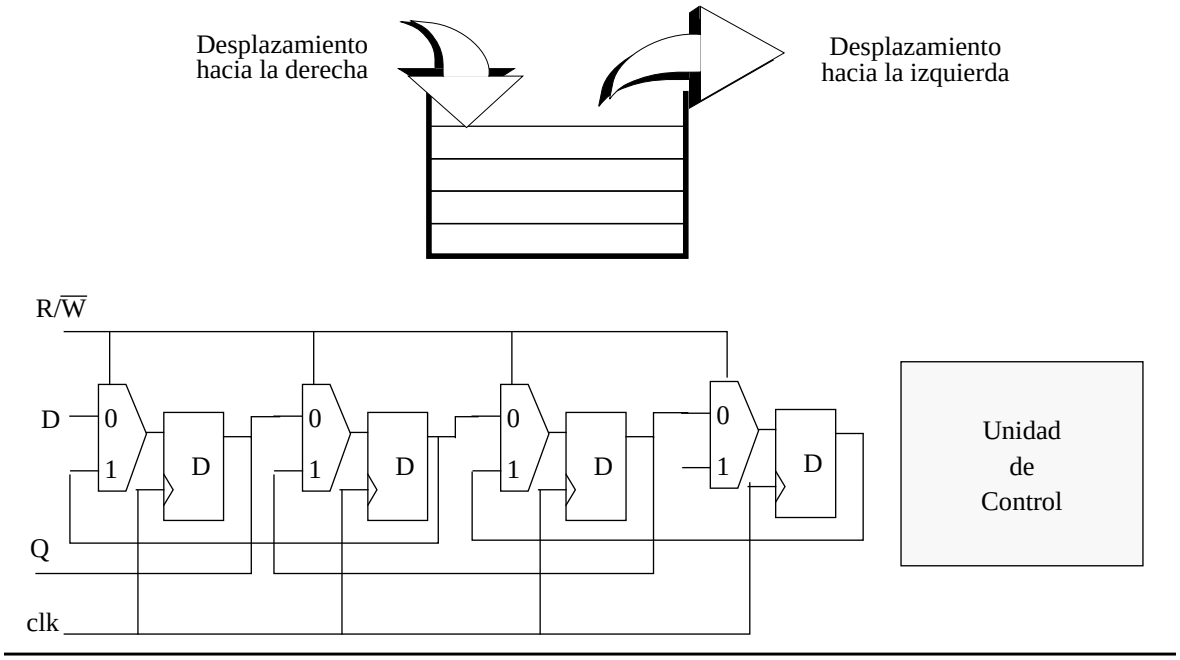
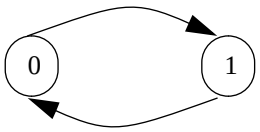


Figura 6.5.- Esquema de operación y lógico de una arquitectura LIFO.

Consideremos el diseño de un contador de un bit, según la primera definición. El diagrama de estados con la tabla de transición de dicho contador será el mostrado en la figura 6.6. Como podemos ver, el próximo estado coincide con el complemento del estado presente por lo que utilizaremos biestables tipo T, obteniendo el circuito adjunto.



Estado	Entrada
0	1
1	0

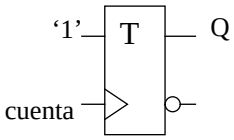


Figura 6.6.- Diseño del contador de un bit.

Si queremos ampliar el contador a un número mayor de bits podemos conectar el contador anterior en cascada. Para ello, debemos tener en cuenta que la siguiente cuenta sólo se debe producir cuando la primera haya finalizado, es decir, haya vuelto a cero. Por lo tanto, el siguiente biestable debe ser disparado por el flanco de bajada. Por homogeneidad del diseño, se disparan todos los biestables por el flanco de bajada, contado realmente los flancos negativos.

Un contador de cuatro bits se muestra en la figura 6.7, formado por una conexión en serie de varios biestables T disparados por flancos. Las ecuaciones lógicas correspondientes al comportamiento de dicho circuito serán las siguientes:

$$Q_0 = \overline{D} \oplus q_0$$

$$Q_1 = \overline{q_0} \oplus q_1$$

$$Q_2 = \overline{q_1} \oplus q_2$$

de donde se obtienen las formas de onda mostradas en la figura 6.7.

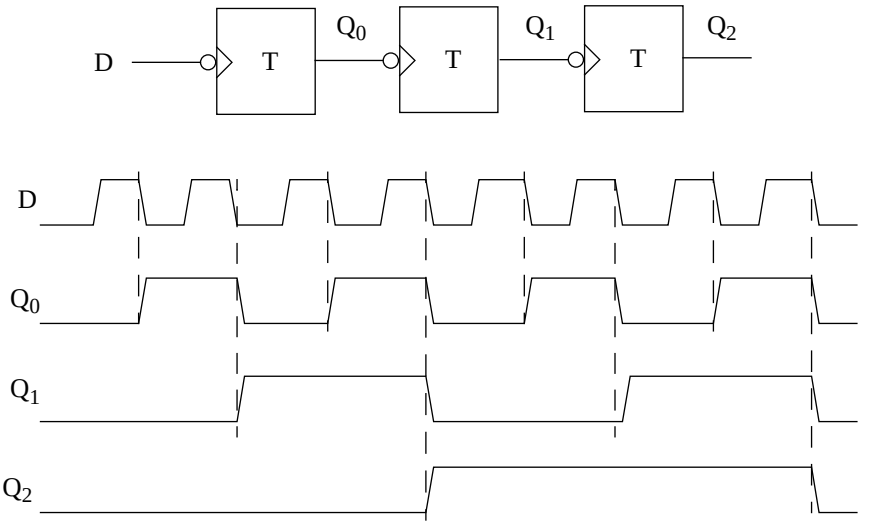


Figura 6.7.- Contador asíncrono o de rizado ascendente, junto a sus formas de onda.

Las señales de salida, $Q_2\ Q_1\ Q_0$, coinciden con la codificación binaria del número de pulsos de la señal de entrada (en su flanco de bajada), por lo que se produce la cuenta del número de pulsos, además de su almacenamiento. Si nos fijamos en las señales de salida de forma individual, podemos comprobar que $T_i = 2^i\ T_D$, o lo que es lo mismo, $F_i = 2^{-i}\ F_D$, por lo que obtenemos señales que dividen la frecuencia de la señal de entrada. Por lo tanto, este circuito cumple la definición de contador, de hecho estamos ante un contador asíncrono (ya que no existe una señal global de reloj) o de rizado ascendente. El hecho de denominarse contador de rizado es debido a que las transiciones de los diferentes biestables no son simultáneas (ya que sus señales de control son diferentes); por lo tanto, para llegar al dato estable es necesario pasar por un pequeño rizado correspondiente a las diferentes transiciones de los biestables.

Consideremos ahora una versión del anterior contador de rizado, mostrado en la figura 6.8. Las ecuaciones lógicas correspondientes al comportamiento de dicho circuito serán las siguientes:

$$Q_0 = \overline{D} \oplus q_0$$

$$Q_1 = q_0 \oplus q_1$$

$$Q_2 = q_1 \oplus q_2$$

de donde se obtienen las formas de onda mostradas en la figura 6.8

Si nos fijamos, podemos comprobar que la cuenta se realiza en orden inverso, es decir,

7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> 0 -> 7

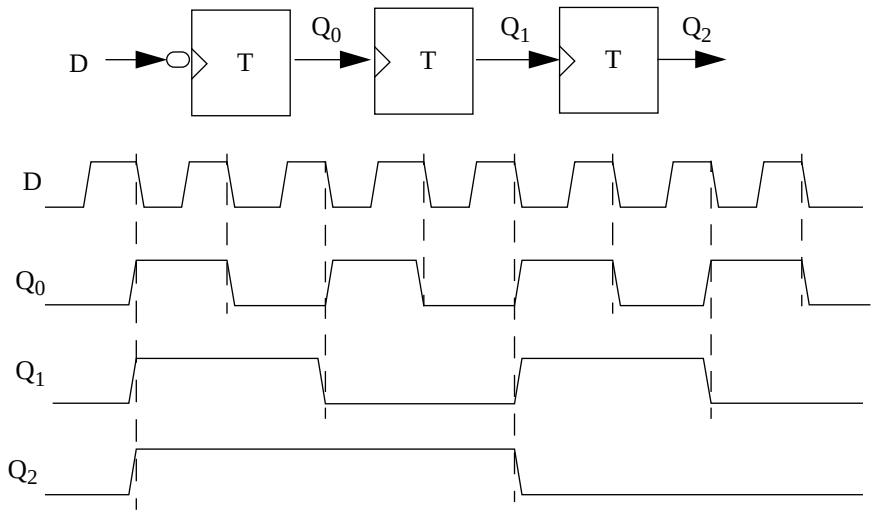


Figura 6.8.- Contador asíncrono o de rizado descendente, junto a sus formas de onda.

por lo que nos encontramos ante un contador descendente. En esta versión, se sigue manteniendo la función de dividir la frecuencia de la señal de entrada.

También podemos encontrar contadores bidireccionales, es decir, contadores ascendentes y descendentes, integrados en un solo dispositivo. Un ejemplo de estos contadores es mostrado en la figura 6.9. Cuando la señal U/$\overline{D}$ se encuentre a nivel alto, estaremos configurando la cuenta como ascendente. En cambio, Cuando la señal U/$\overline{D}$ se encuentre a nivel bajo, estaremos configurando la cuenta como descendente.

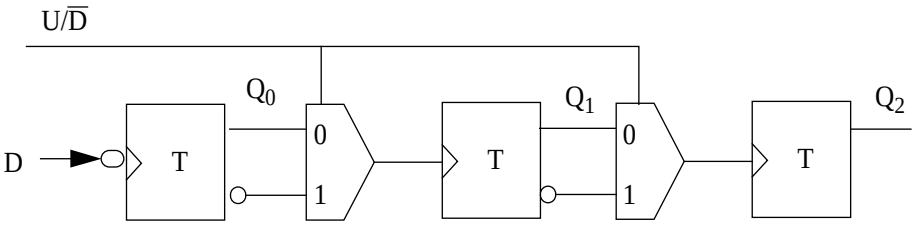


Figura 6.9.- Contador asíncrono bidireccional.

Hasta ahora solamente hemos tratado contadores asíncronos, ya que no existe ninguna señal de control global que gobierne todos los biestables.

A continuación vamos a considerar el diseño de un contador síncrono de tres bits. El proceso de diseño se muestra en la figura 6.10. De nuevo, y como el estado presente es muy parecido al próximo estado, hemos utilizado biestables tipo T. Esta versión es síncrona porque todos los biestables son controlados por la misma señal. No obstante, la operación lógica es la misma que la del contador asíncrono. Las principales diferencias, entre ambas versiones, estriban en:

- el contador síncrono es más costoso debido a la presencia de lógica combinacional que no se encuentra en el asíncrono, y

- muestra un mayor consumo de potencia ya que todos los biestables tienen que realizar una operación en cada flanco de subida de la señal de cuenta (el reloj), mientras que en el asíncrono, cada biestable sólo realiza su operación en cada flanco de subida (o bajada) de la salida anterior. De forma matemática, el número de operaciones será:
  - nN, en el caso síncrono con n pulsos de cuenta y N biestables, y
  - $\Sigma\, n2^{-i}$, desde 0 hasta N-1 en el caso asíncrono.

Estado presente	Cuenta (= clk)
000	001
001	010
010	011
011	100
100	101
101	110
110	111
111	000

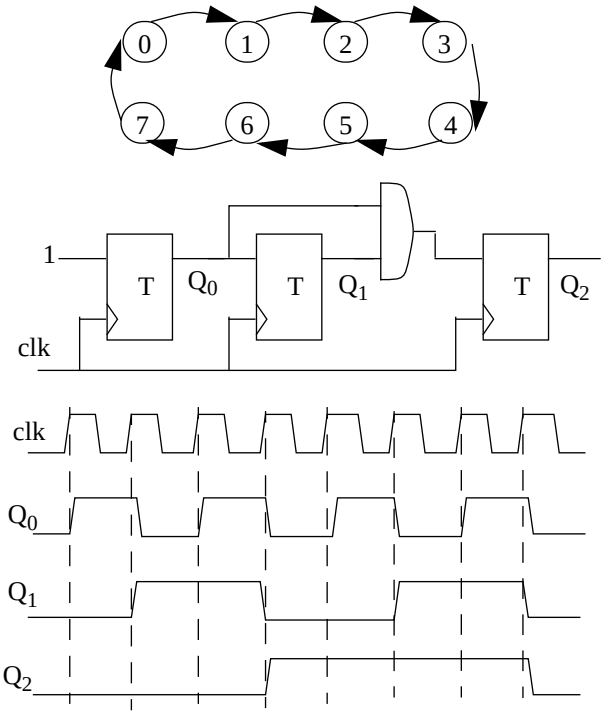


Figura 6.10.- Contador síncrono ascendente, junto a sus formas de onda.

Cuando se dice que estamos ante un **contador de módulo N**, dicho contador realiza la cuenta desde 0 hasta N-1. Hasta ahora solamente hemos considerado contadores cuyo módulo es potencia de dos. Cuando el módulo del contador no es potencia de dos, hay que generar el mintérmino N con lógica combinacional, para que cuando lleguemos a él podamos inicializar todos los biestables a través de sus terminales de reset e iniciar de nuevo la cuanta. Por ejemplo, si queremos diseñar un contador asíncrono de módulo 3, necesitaremos como mínimo dos biestables, ya que $1 < \lg_2 3 < 2$. Cuando llegue al mintérmino 3, debemos inicializar todos los biestables. Podemos comprobar que cuando llegamos a 3, Q_1Q_0 = “11“, los dos biestables son inicializados pasando inmediatamente (casi instantáneamente, el retraso de la puerta AND, que genera el mintérmino correspondiente, y la inicialización de los biestables) a Q_1Q_0 = “00“. Este caso particular se muestra en la figura 6.10.

En los contadores también podemos encontrar versiones con capacidad de carga paralela, situación necesaria en los contadores descendentes cuyo módulo no es potencia de dos. Una solución a este problema consiste en modificar los biestables para darle esta nueva función. El esquema lógico de esta solución se muestra en la figura 6.12. En función de la señal $C/\overline{L}$, se producirá la cuenta, si dicha señal está a nivel alto al configurar el biestable como tipo T, o la

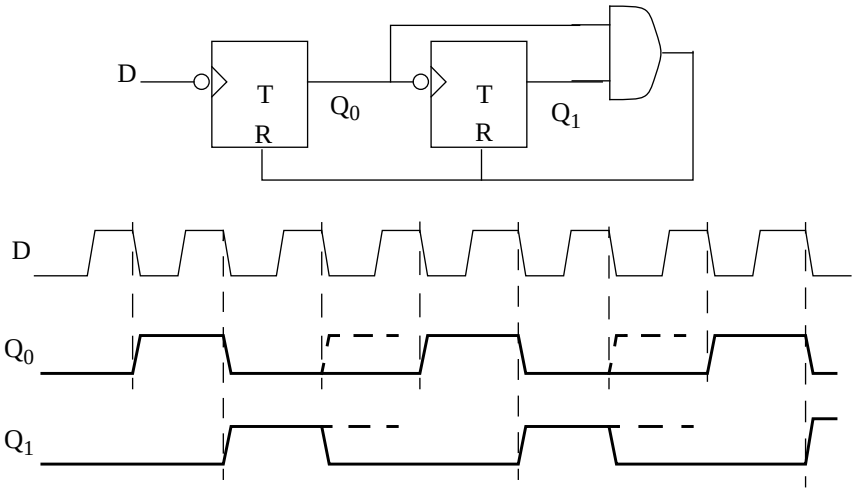


Figura 6.11.- Contador asíncrono ascendente de módulo 3, junto a sus formas de onda.

carga paralela, si dicha señal está a nivel bajo al configurar el biestable como tipo D. Otra forma será utilizando los terminales de set (puesta a 1) o reset (puesta a cero) del biestable. Con esta última forma hay que tener cuidado de que ambos terminales sean síncronos para que el cambio de estado siga a la señal de reloj, que coincide con la de cuenta.

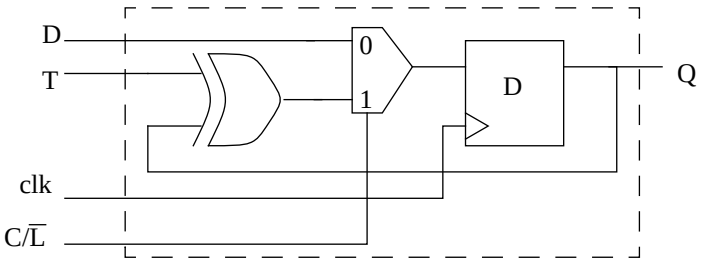


Figura 6.12.- Modificación del biestable tipo T para producir la carga paralela.

Consideremos ahora el circuito de la figura 6.13. En él podemos ver tres biestables tipo D realimentados a través de un multiplexor que permite introducir un dato externo al sistema. Este circuito genera una serie de patrones binarios, por lo que entra dentro de la categoría de contadores, y al estar basado en los registros de desplazamientos reciben el nombre de **contadores de desplazamientos**. También son conocidos como **contadores anulares o en anillo** debido a la realimentación anteriormente mencionada.

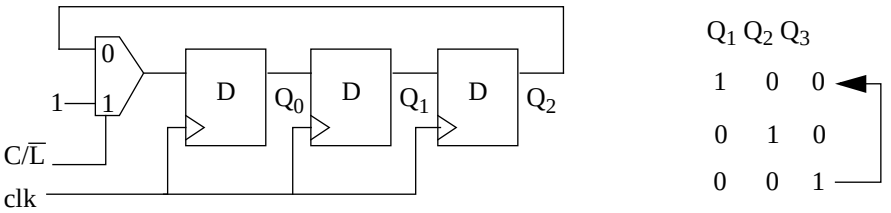


Figura 6.13.- Contador de desplazamiento sin autocorrección.

Entre las principales características de estos contadores podemos encontrar las siguientes:

- Necesitan más biestables que los estrictamente necesarios, en estos contadores necesitamos N biestables para un contador de módulo N. Esta característica provoca que la salida no sea la codificación binaria del número de la cuenta.
- Son muy sensibles ante la posibilidad de fallos, por lo que se suele introducir circuitería de autocorrección, como podemos ver en la figura 6.14. Podemos comprobar que se corrigen los fallos en un máximo de N-1 pulsos de la señal de cuenta, donde N es el módulo.

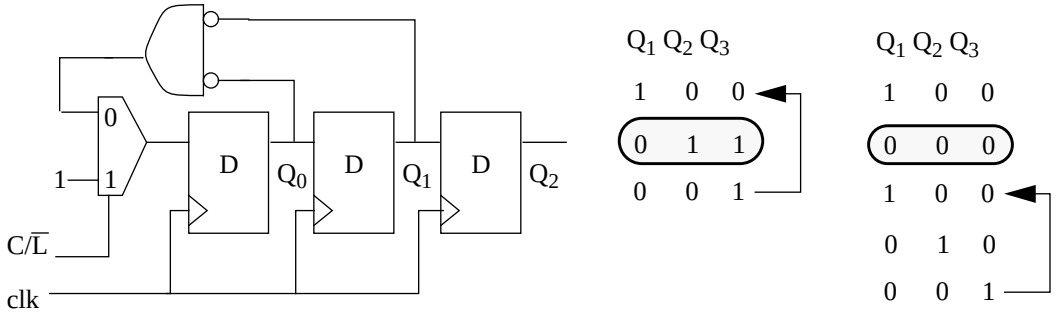


Figura 6.14.- Contador de desplazamiento con capacidad de autocorrección.

Existe un contador de desplazamiento, denominado **contador de Johnson**, que muestra un mejor aprovechamiento del número de biestables. Este contador se muestra en la figura 6.15. Debido a la realimentación de la señal complementada Q_5, podemos generar el doble de estados.

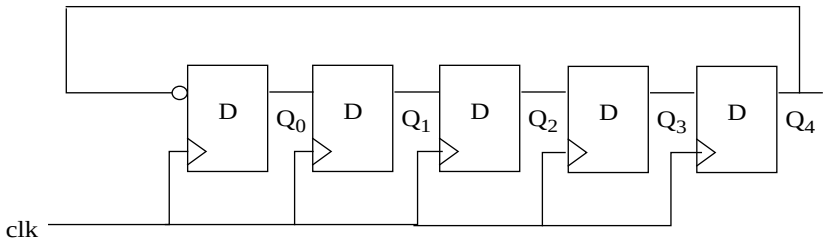


Figura 6.15.- Contador Johnson de módulo 10 sin autocorrección.

Debido a que no muestra capacidad de autocorreción, el más conocido y utilizado es el mostrado en la figura 6.16 que sí tiene autocorreción.La capacidad de contar el doble del número de biestables, tiene como penalidad de que para determinar su estado hay que generarlo con un pequeño decodificador formado por 2N puertas AND de dos entradas, como se puede ver en la tabla 6.2.

En el caso de necesitar contadores de módulos muy grandes, la técnica más empleada es la conexión de varios contadores más pequeños. La manera más intuitiva de conectar dos contadores consiste en utilizar como señal de cuenta una señal que nos indique el final de la cuenta del primer contador. Por ejemplo, en la figura 6.17 mostramos un contador de módulo 64 realizado con dos contadores de módulo 8.

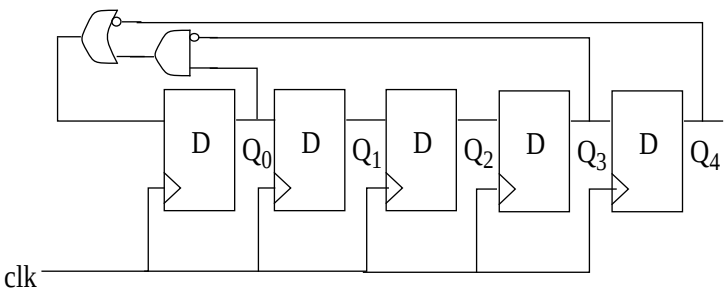


Figura 6.16.- Contador Johnson de módulo 10 con autocorrección.

Q_0	Q_1	Q_2	Q_3	Q_4	Decodificación
0	0	0	0	0	$\bar{q}_4\bar{q}_0$
1	0	0	0	0	$q_0\bar{q}_1$
1	1	0	0	0	$q_1\bar{q}_2$
1	1	1	0	0	$q_2\bar{q}_3$
1	1	1	1	0	$q_3\bar{q}_4$
1	1	1	1	1	q_4q_0
0	1	1	1	1	$\bar{q}_0q_1$
0	0	1	1	1	$\bar{q}_1q_2$
0	0	0	1	1	$\bar{q}_2q_3$
0	0	0	0	1	$\bar{q}_3q_4$

Tabla 6.2. Salidas del contador Johnson de módulo 10.

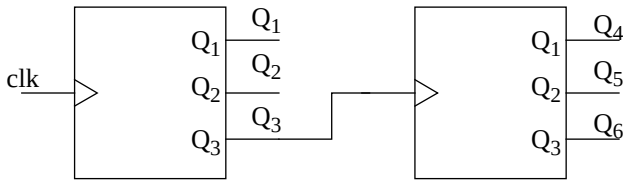


Figura 6.17.- Conexión asíncrona de dos contadores de módulo 8 para formar uno de 64.

Otra forma similar a la anterior consiste en utilizar dos señales que suelen tener todos los contadores:

- fin de cuenta, que nos indica cuando ha finalizado la cuenta y va a empezar de nuevo, y
- habilitación de cuenta, que inhibe o desinhibe la cuenta manteniendo el valor anterior almacenado cuando se inhibe la cuenta.

En este caso, la conexión sería como se muestra en la figura 6.18.

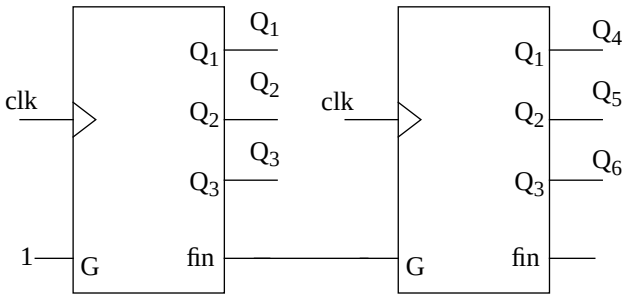


Figura 6.18.- Conexión síncrona de dos contadores de módulo 8 para formar uno de 64.

2. Diseño secuencial MSI.

En el diseño de sistemas secuenciales con dispositivos MSI, los elementos almacenadores de información podrán ser registros o contadores. La utilización de uno u otro dependerá de la adecuabilidad de los cambios de estados a ser realizados con cuenta, es decir, los próximos estados serán los siguientes en codificación a los estados presentes, como sucede:

1 -> 2 -> 3 -> 4 -> 5

las transiciones de estados serán simplemente la función de un contador.

2.1. Diseño con registros.

El diseño MSI con registros coincide con el diseño utilizando biestables tipo D, los cuales son la base de los registros.

En el caso de utilizar registros de desplazamiento universales, la generación del próximo estado supondrá la obtención de las siguientes señales:

- Señales de control, las cuales seleccionarán la operación que realizará el registro: desplazamiento hacia la izquierda, desplazamiento hacia la derecha, carga paralela y no operación.
- Entradas de carga paralela, que será el valor al que debe llegar el registro cuando se produzca una carga paralela.
- Entrada de datos de desplazamiento (izquierda y/o derecha), que será el valor que tenga que tomar el primer biestable cuando se produzca un desplazamiento.

2.2. Diseño con contadores.

En este caso, los cambios de estados son producidos mediante un contador. Por lo tanto, las señales para generar el próximo estado que hay que generar serán tres:

- Señal de cuenta, cuando el próximo estado coincide con el número siguiente al actual,

- Señal de carga paralela, cuando el próximo estado no es el siguiente número ni el mismo. Además hay que generar los datos de carga paralela para actualizar la información, y
- Señal de habilitación, cuando el próximo estado coincide con el actual, ya que no se debe realizar la cuenta ni la carga paralela.

3. Diseño RTL

Ya hemos visto que una de las formas de describir un sistema secuencial, y a partir de ahí realizar el diseño digital que muestra dicho comportamiento, es el diagrama de estados. No obstante su utilidad está limitada a un número relativamente pequeño de estados, o lo que es lo mismo, de requerimientos de memoria.

Si tomamos el ejemplo sencillo de la máquina de refresco, su diagrama (figura 6.19) muestra un considerable número de estados con la limitación impuesta de un único refresco y sin la posibilidad de devolver cambio. Al añadir la funcionalidad de diversificar el número de refrescos y devolver cambio, el diagrama de estado aumentaría de tal forma que ya no sería tratable.

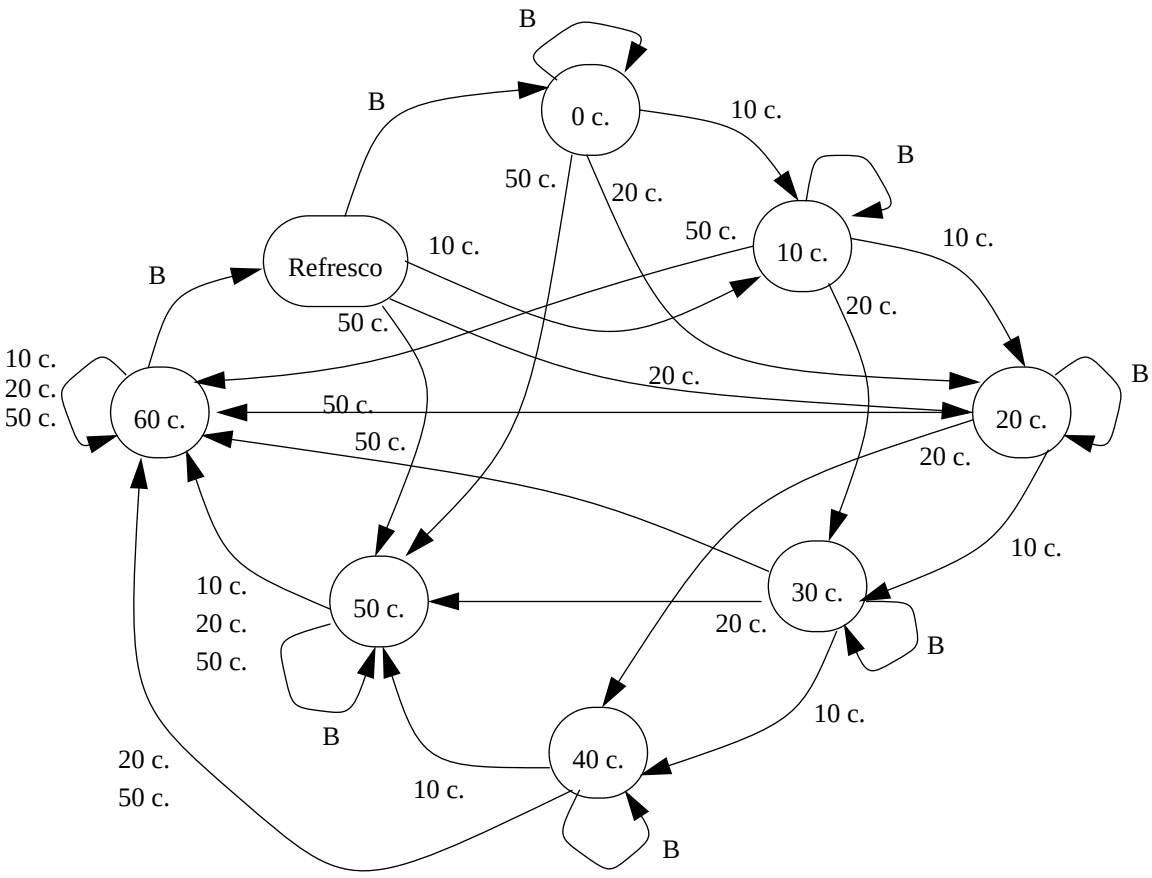


Figura 6.19.- Diagrama de estado correspondiente a la máquina de refresco.

No obstante, podemos encontrar una descripción verbal muy simple:

> Se insertan monedas para acumular el importe del refresco
> Mientras no se haya insertado dicho importe, no se hace nada
> Cuando haya la cantidad suficiente, devuelve el refresco y la diferencia entre la cantidad acumulada y el precio del refresco.

Por lo tanto, debe existir otro medio de descripción para estos problemas con una descripción sencilla, cuya diagrama de estado sea demasiado complejo para manejarlo. Si recapacitamos sobre la descripción verbal anterior, podemos observar que existen operaciones, como la suma de las diferentes monedas insertadas y la diferencia entre la cantidad almacenada y el precio del refresco seleccionado, que no intervienen en la secuenciación de las operaciones operaciones.

Una forma útil de describir estos sistemas más complejos será la descripción de la secuenciación y la manipulación de los datos de manera bien diferenciada, de tal forma que permita un tratamiento separado, así podremos realizar el autómata de control independientemente de los bloques de procesado utilizados. Esta descripción es un algoritmo, cuya definición podría ser la siguiente:

> Un **algoritmo** es el conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

Esta nueva forma de descripción se encuentra en un nivel de abstracción diferente ya que utiliza como componentes los dispositivos MSI (tanto combinacionales como secuenciales) y no se limita a puertas lógicas y biestables. Este nuevo nivel se denomina transferencia de registros o más comúnmente RT (Register Transference). En este caso, la funcionalidad es descrita mediante una secuencia de transferencias de registros, la cual se puede definir de la siguiente forma:

> Una **transferencia de registros** es una transformación realizada sobre un dato mientras es transferido de un registro a otro.

Esta transferencia necesita la existencia de un secuenciamiento para garantizar la operación correcta. Por lo tanto, un sistema descrito en el nivel RTL se dividirá en dos grandes bloques, como se muestra en la figura 6.20:

- Procesador. Es el bloque encargado de realizar las transformaciones (procesados) de los datos.
- Controlador. Es el bloque encargado de gobernar la secuencia de las operaciones, para la operación correcta.

A pesar de posibilitar el diseño de ambos bloques por separado, no podemos olvidar que los dos forman un mismo circuito por lo que estarán conectados por las señales de estado, que determinarán la situación en la que se encuentra la operación, y las señales de control, que habilitarán la operación necesaria en cada instante.

3.1. Procesador

Como ya se ha comentado, el procesador (que algunos autores también denominan *ruta de datos*) es el bloque encargado de realizar las diferentes operaciones con los datos. Para ello necesitaremos cuatro tipos de señales:

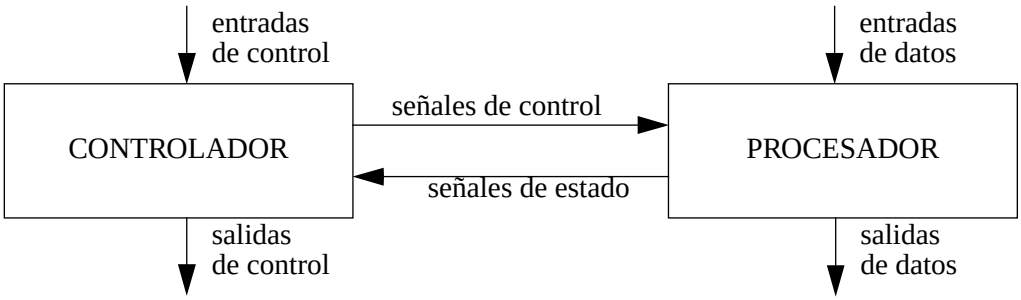


Figura 6.20.- Estructura interna de un sistema visto desde el nivel RTL.

- entradas de datos
- salidas de datos
- señales de estado, las cuales indicarán la situación o estado de la operación entre los datos que se está realizando en cada momento
- señales de control, las cuales activarán la operación que se debe realizar en cada momento.

En este bloque podemos encontrar unidades de almacenamiento (para datos y condiciones), unidades de procesado (para las operaciones de los datos) y unidades de camino de datos (para la dirección correcta de los datos a lo largo del algoritmo). Según estas unidades, podemos clasificar los sistemas RTL en:

- Sistemas con recursos no compartidos. Cada unidad funcional realiza única y exclusivamente una operación.
- Sistemas con recursos compartidos. Cada unidad funcional puede realizar varias operaciones, con la única limitación de que deben ejecutarse en diferentes ciclos de operación. Esta situación se lleva al extremo en los sistemas **unimódulos**, en los que existe un único módulo que lleva a cabo todas las operaciones.

3.2. Controlador

El controlador es el bloque encargado de gobernar la secuencia correcta de operaciones del sistema. Por lo tanto, podremos tener cuatro tipos de señales:

- entradas de control
- salidas de control
- señales de estado, las cuales indicarán la situación o estado de la operación entre los datos que se está realizando en cada momento
- señales de control, las cuales activarán la operación que se debe realizar en cada momento.

Las entradas y salidas de control por excelencia son las señales de inicio y fin de operación, las cuales pueden ser señales específicas, o depender de las entradas de datos (como sucede en la máquina de refrescos en la que la inicialización comienza con la inserción de la primera

moneda). Por último, la señales de comunicación con el procesador deben estar presentes en ambos bloques, es decir, las señales de estado y de control.

Con respecto al gobierno de la secuencia de operaciones, se llevará a cabo con un autómata finito, como situación más habitual. Este gobierno o control puede ser realizado como:

- Control centralizado. Este tipo se da cuando existe un único controlador que gobierna todas las operaciones del sistema.
- Control descentralizado. Este tipo se da cuando cada unidad funcional tiene su propio controlador. La coordinación entre todos ellos se realiza a través del cableado.
- Control semicentralizado. Este tipo es similar al control descentralizado. La diferencia entre ambos radica en que la coordinación es realizada por un controlador central que gobierna al resto de controladores.

3.3. Diagramas ASM (Algorithmic State Machine)

Un diagrama ASM es un medio de representación gráfico/textual de un algoritmo, tal que permite describir ciclo a ciclo el funcionamiento con temporización síncrona y control centralizado. En el caso de optar por un control descentralizado o semicentralizado, se debería realizar el particionado del sistema según la descentralización requerida con los subsiguientes diagramas ASM. Un ejemplo de diagrama algorítmico se muestra en la figura 6.21.

El algoritmo descrito por el diagrama anterior realiza las siguientes operaciones:

- Mientras que no se seleccione ningún refresco, se deben ir acumulando las monedas.
- Cuando se seleccione un refresco, se obtiene la diferencia entre la cantidad de dinero almacenada y el precio del refresco seleccionado.
- Si dicha diferencia es negativa, se deben seguir acumulando monedas.
- Si la diferencia no es negativa, se expulsará el refresco, devolverá dicha diferencia e inicializará la cantidad almacenada a cero.

En el diagrama anterior podemos ver dos componentes fundamentales de los diagramas ASM: cajas de estado y de selección; a los que habría que añadir un tercer componente: cajas de condición o condicionales. Los tres componentes, cuya apariencia se muestra en la figura 6.22, tienen las siguientes funciones:

- Cajas de estado. Este componente especifica las transferencias que se deben realizar en un nuevo ciclo de operación. Esta transferencia puede ser un simple almacenamiento o un procesado con almacenamiento.
- Caja de selección. Este componente especifica una determinada condición que determinará la siguiente acción a realizar. Por lo general, esta condición será el resultado de una comparación, aunque puede ser una selección entre más de dos valores.
- Cajas de condición. Este componente especifica la transferencia que se realizará en función de una determinada condición, por lo que siempre estarán después de una caja de selección. La diferencia con las cajas de estado radica en que las cajas de condición serán evaluadas en el mismo ciclo de operación, mientras que en las cajas de estado la asignación se realizará en el siguiente ciclo.

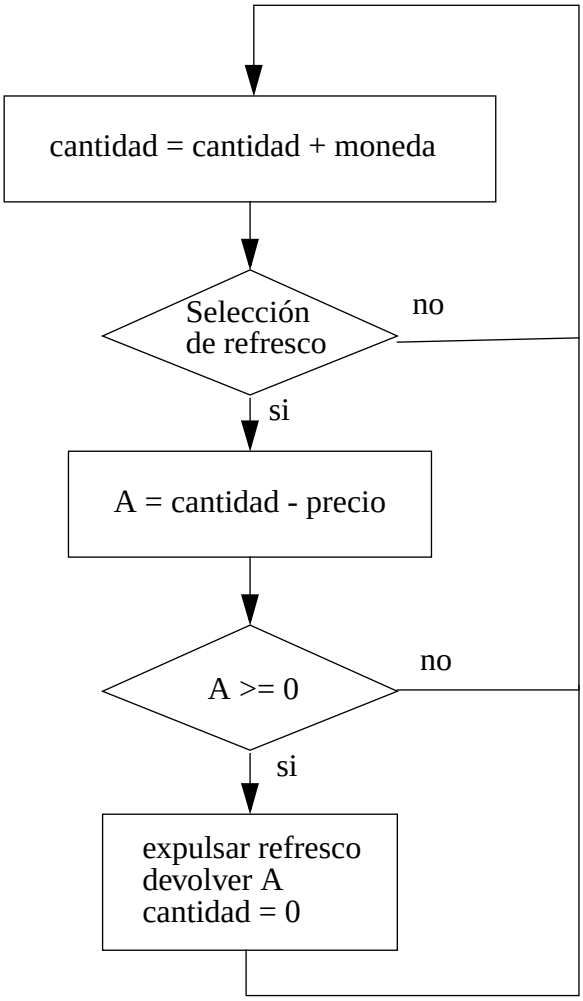


Figura 6.21.- Ejemplo de diagrama algorítmico.

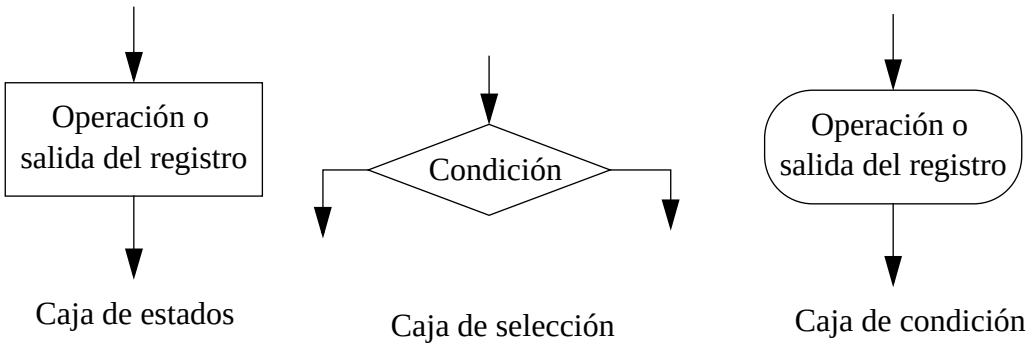


Figura 6.22.- Componentes de los diagramas ASM.

- Bloques ASM. Todos los elementos anteriores se agruparán según el ciclo de operación donde sean realizados. Estas agrupaciones son los denominados bloques ASM.

Todos estos componentes se agruparán en diferentes bloques ASM, los cuales especifican

todas las acciones realizadas en un mismo ciclo de operación, por lo que irán de una caja de estado a otra. Estos bloques se identifican con los estados de los autómatas finitos. Las propiedades que muestran dichos bloques son las siguientes:

- En su interior tiene que haber una única caja de estado, y puede que múltiples cajas de selección y/o condicionales. Esta necesidad es debida a que cada caja de estado realizará su operación en un nuevo ciclo, es decir, en ciclos diferentes. Esta caja de estados puede estar vacía, como veremos posteriormente.
- Hay un solo punto de entrada (la caja de estado) y múltiples puntos de salida.
- Cada señal sólo puede ser asignada una vez por bloque, con el fin de evitar conflictos. Esto no quiere decir que no puedan aparecer más de una vez, ya que podemos encontrar dos asignaciones tras una caja de selección puesto que se realizará únicamente uno de los dos.

Una vez que conocemos los diferentes componentes de un diagrama ASM y las propiedades que cumplen, ya estamos en disposición de generar uno de ellos. No obstante, de forma previa, debemos tener en cuenta una serie de pautas, entre las que podemos destacar:

- Los estados y transiciones del controlador se especifican gráficamente.
- Las transferencias se especifican mediante objetos y operadores.
- Las transferencias realizadas en un mismo ciclo no pueden tener conflictos en el uso de recursos, es decir, una misma unidad no puede estar realizando dos operaciones diferentes en el mismo ciclo de operación.
- El periodo de reloj viene determinado por el camino combinacional con mayor retraso.
- El diagrama está formado por uno o más bloques ASM y una tarjeta declarativa, la cual especifica:
  - Nombre, anchura y codificación de señales y puertos
  - Estado inicial del sistema y valor inicial de las señales y puertos
  - Dirección de los puertos
- No tiene puntos de entrada ni de salida
- Toda salida de un bloque ASM debe estar conectado a la entrada de otro.

Veamos como ejemplo de diagrama ASM, el algoritmo de multiplicación utilizando sumas sucesivas con el número mínimo de ciclos. La operación comenzará con un pulso de la señal *inicio* y se indicará la finalización cuando se active la señal *acabado*. Los operandos se introducirán por los puertos *A* y *B* y el resultado se leerá por el puerto *total*. El diagrama se muestra en la figura 6.23.

En primer lugar vamos a identificar el tipo de señales según el esquema genérico de un sistema RTL mostrado en la figura 6.20. Podemos identificar las siguientes señales:

- Entradas de control, son señales de entrada sobre las que no se realiza ningún procesado sino que intervienen para un cambio de estado. En este caso tenemos la señal *inicio*.
- Salidas de control, son señales de salida que identificarán un determinado estado, el

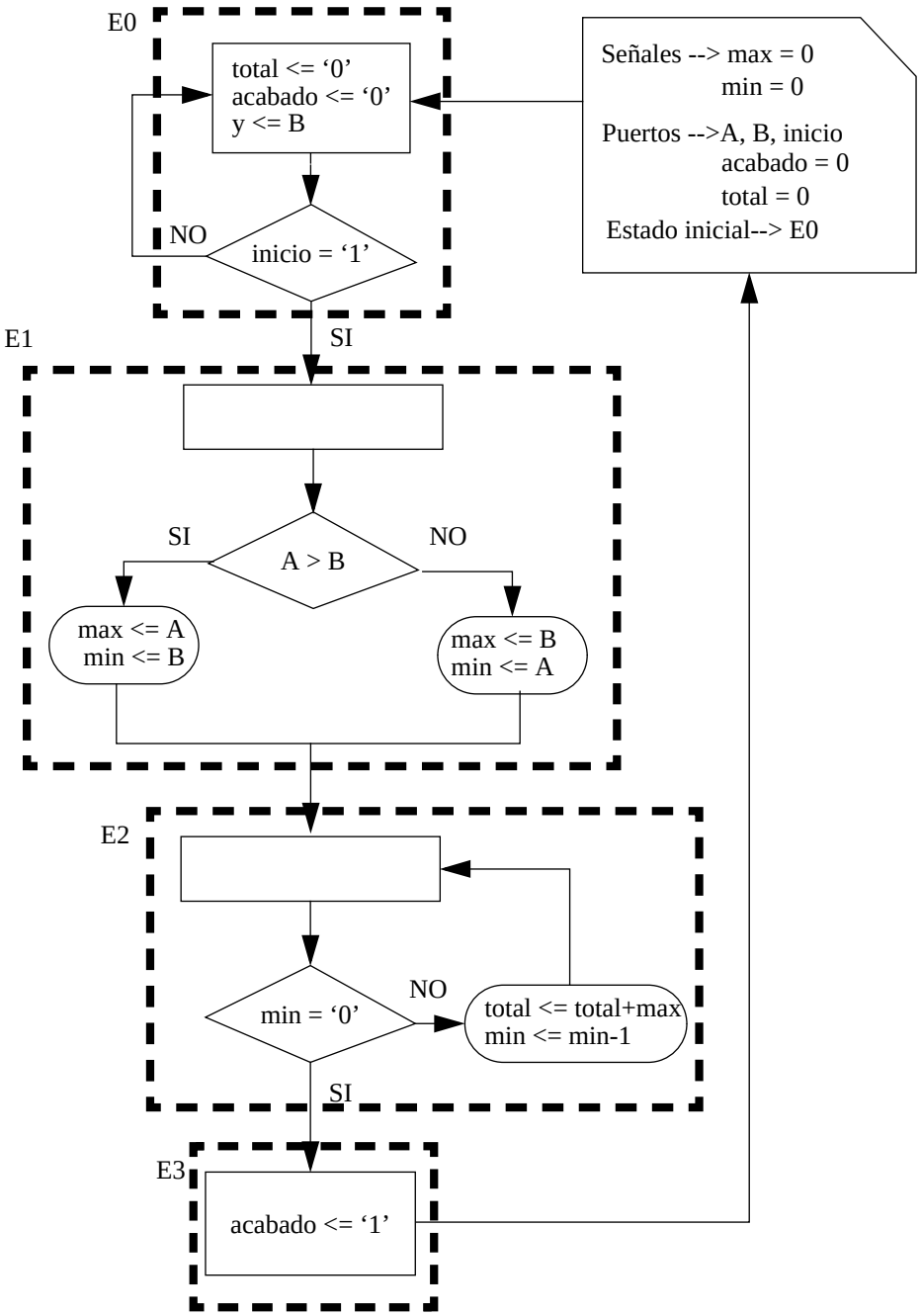


Figura 6.23.- Diagrama ASM correspondiente al algoritmo de multiplicación de sumas sucesivas, con el número mínimo de ciclos.

más comúnmente utilizado es el estado de finalización de la operación. En este caso tenemos la señal *acabado*.

- Entradas de datos, son señales de entrada sobre las que se realizará el procesado oportuno. En este caso tenemos las señales *A* y *B*.
- Salidas de datos, son señales de salida en las que estarán almacenadas el resultado final de la operación cuando lo indiquen las salidas de control. En este caso tenemos la señal *total*.
- Señales de estado, son señales internas del procesador que determinarán los posibles

cambios de estados para el siguiente ciclo de operación. Estas señales serán salidas de alguna operación que se encuentren en las cajas de selección. En este caso tenemos las salida de la comparación entre los dos operandos, y la salida de la comparación entre el operando menor (almacenado en *min*) y el '0'.

- Señales de control, son señales internas del controlador para gobernar la operaciones de los diferentes bloques del procesador. Por lo tanto, hasta que no se decidan qué componentes tendrá el procesador, no podremos definir este tipo de señales.

Seguidamente vamos a estudiar el significado de cada estado y la razón de su inclusión en el diagrama:

- El estado E0 es el estado de inicio donde se chequeará la señal *inicio* para dar comienzo a la operación
- El estado E1 determinará cuál de los operandos es el mayor para almacenarlo en el registro correspondiente. Esta acción es necesaria para que el número de ciclos (que será función del número de sumas sucesivas a realizar) sea mínimo; ya que dicha condición sólo se cumplirá cuando se sume el operando mayor.
- El estado E2 es el encargado de realizar la multiplicación, es decir, las sumas sucesivas. El número de estas sumas será igual al operando menor.
- Por último, el estado E3 activará la señal *acabado*, indicando la finalización de la operación y que el resultado se encuentra en *total*.

Por último vamos a comentar algunas características de los diagramas ASM tomando como base el diagrama anterior. En dicho diagrama podemos identificar los tres componentes que podemos encontrar en cualquier diagrama: cajas de estado, de selección y de condición. Así mismo podemos ver la tarjeta declarativa (recuadro con la esquina truncada) y los diferentes bloques ASM (o estados del controlador, identificados por los recuadros punteados). Otro dato a destacar es la existencia de cajas de estado vacías (como se mencionó anteriormente); las cuales pueden venir determinadas porque no se deba realizar ninguna transferencia o algunas señales deban ser asignadas a valores diferentes en función de una condición. En nuestro ejemplo vemos dos cajas vacías, en los estados E1 y E2. En el caso del estado E1, la caja de estado debe estar vacía ya que las señales internas del procesador de los operandos máximo y mínimo pueden estar conectadas a puertos diferentes; mientras que en el caso del estado E2, la caja debe estar vacía ya que se deberá asignar a *total* un nuevo valor (si es necesario realizar más sumas) o no tendrá ninguna asignación (si ya se han hecho todas las sumas). La existencia de cajas de condición implica que la señal *total* dependerá directamente del valor de una señal de estado, y no de un bloque ASM; por lo tanto, esta situación implicará que el autómata correspondiente pertenecerá a la categoría de Mealy.

3.4. Flujo de diseño RTL

Hasta ahora, se ha visto cómo podemos describir un determinado algoritmo utilizando un diagrama ASM, pero no podemos perder de vista que el fin último es llegar al sistema lógico que implementa dicho comportamiento.

El flujo de diseño RTL (secuencia de actuaciones para obtener el sistema lógico) difiere del flujo basado en los diagramas de estados en que se implementan por separado dos subsistemas: procesador y controlador. Este nuevo flujo se muestra en la figura 6.24.

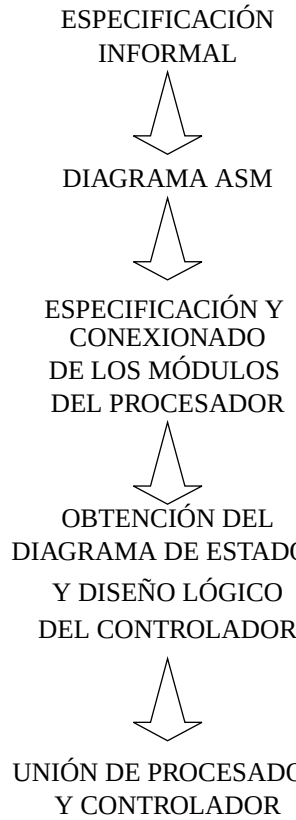


Figura 6.24.- Flujo de diseño RTL.

Como en la mayoría de los flujos de diseño, el punto de partida es la especificación del diseño a realizar. En estas especificaciones se deben indicar la funcionalidad del diseño, un esquema de puertos, una descripción del algoritmo (lo más detallada posible) y una especificación del protocolo de comunicaciones.

Todas la información suministrada en las especificaciones es descrita en forma de diagrama ASM para un tratamiento más adecuado. De esta forma, la determinación de los módulos del procesador y el comportamiento del controlador (su diagrama de estados) se facilita en gran medida.

Por lo general, las operaciones del diagrama ASM son componentes lógicos (ya sean puertas básicas o dispositivos MSI) que están directamente accesibles, como pueden ser comparadores, sumadores/restadores, contadores o registros de desplazamiento. Así, la composición del procesador es directa. En el caso de que exista una operación que no tenga asociada un componente, por ejemplo una multiplicación, habrá que implementar un componente para dicha operación, pero de forma aislada del diseño global facilitando así la tarea a través del particionado del problema. Como el diagrama también nos indica la secuencia que deben seguir las operaciones, también nos da idea del conexionado de dichos módulos, aunque no sea completamente directo.

Con respecto al controlador, los bloques ASM identificados en el diagrama determinarán

el diagrama de estados del autómata que hará la función de controlador. Así mismo, el diagrama ASM también identificará las señales de estado provenientes del procesador y las señales de control externas (que determinarán los cambios de estado). A pesar de que el controlador apenas tenga dependencia del procesador, es necesario identificar los módulos del procesador previamente al controlador para conocer las salidas de control que el controlador debe generar. Una vez que tenemos en cuenta todos estos detalles, el diseño lógico del controlador se limita al diseño de un autómata finito como se vio en el tema 4.

Por último, se lleva a cabo la conexión entre los diferentes componentes del controlador y procesador. Tras esta tarea tendremos un circuito digital que implementa toda la funcionalidad del diagrama ASM.

3.4.1. Optimización del procesador

No podemos perder de vista que una de las metas fundamentales del diseño es la optimización de los principales parámetros, entre los que se encuentran el coste y el área. Por lo tanto, el número de módulos del procesador deberá tender al mínimo con la consecuente utilización de sistemas con módulos compartidos. No obstante, la mejora en el número de componentes funcionales puede llevar asociada un empeoramiento en otros parámetros como puede ser la velocidad (al aumentar el retraso de la lógica combinacional). Para ello, debemos abordar dos situaciones:

- Reuso de unidades funcionales, de tal forma que se reduzca el número y coste de las unidades requeridas para ejecutar todas las operaciones del diagrama ASM. Esta situación es posible siempre y cuando existan dos operaciones de la misma naturaleza en ciclos de operación diferentes; así sólo deberemos seleccionar las entradas oportunas en cada ciclo.
- Reuso de unidades de almacenamiento, de tal forma que se reduzca el número de unidades para almacenar todos los resultados, parciales y finales, de un diagrama ASM. Esta situación es posible siempre y cuando existan dos resultados que no convivan en los mismos ciclos; así sólo deberemos seleccionar las entradas oportunas en cada ciclo.

Veamos un ejemplo de esta reutilización. En la figura 6.25 mostramos una parte de un diagrama ASM. En esta porción, cada caja de estados forma un bloque ASM, es decir, un estado del controlador. Por lo que podríamos solapar todas las operaciones comunes. En este caso tendríamos como operaciones: sumas (en los estados E5 y E6), resta (en el estado E7) y el valor máximo (en el estado E8). Por lo tanto, con un solo módulo, un sumador/restador, podríamos realizar tres operaciones diferentes ya que se realizan en ciclos diferentes, mientras que el valor máximo se debería implementar con un módulo diferente.

Para determinar las entradas correctas en cada instante, se seleccionan con multiplexores (u otros elementos con una funcionalidad equivalente, como pueden ser buses o dispositivos triestado), cuyas señales de selección vendrán determinadas por el estado en el que estén, es decir, por el controlador. Por este motivo es preciso definir el procesador antes que el controlador. Para el caso de las salidas no es preciso un demultiplexor, sino habría que manipular las habilitaciones de los registros; pero se ha elegido ese esquema por claridad en el dibujo.

Con respecto a la reutilización de las unidades de almacenamiento, este hecho sólo es

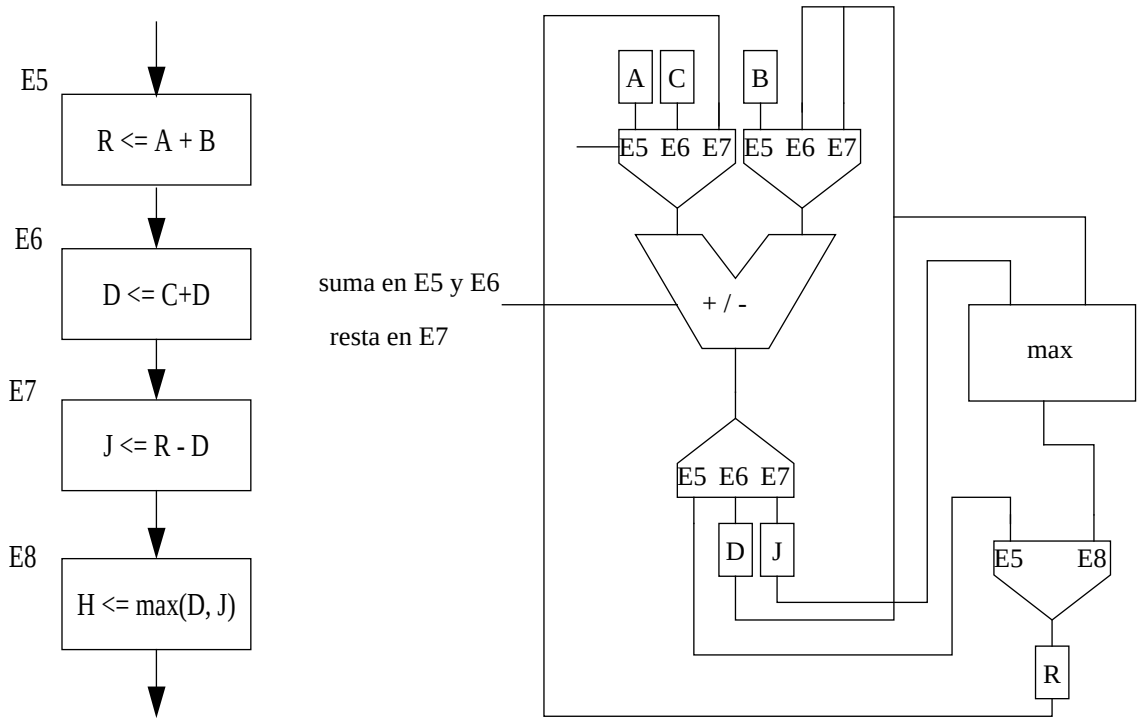


Figura 6.25.- Porción de un diagrama ASM, junto a una posible implementación.

posible si no existe solapamiento temporal de los datos. Por lo tanto, si suponemos que los datos no se van a utilizados en el resto del diagrama (excepto las entradas *A*, *B* y *C*), podemos observar que la vida de los siguientes datos coincide con la mostrada en la tabla 6.2. La señal *R* debe tener un tiempo de vida correspondiente a los estados E5, E6 y E7 (aunque no se necesite en E6, se debe mantener almacenada ya que es necesaria en E7), la señal *D* lo tendrá correspondiente a E6, E7 y E8, la señal *J* lo tendrá correspondiente a E7 y E8, y la señal *H* lo tendrá correspondiente a E8. Luego es necesario un total de tres registros para almacenar *D*, *J* y otro más que será utilizado por *R* y *H*, ya que son las únicas señales que no están solapadas en el tiempo. Para introducir el dato correcto en el registro se actúa igual que con los módulos del procesador, es decir, utilizando multiplexores cuya selección será controlada por el controlador.

	E5	E6	E7	E8
R	X	X	X	
D		X	X	X
J			X	X
H				X

Tabla 6.2. Ciclos de operación donde son válidos cada señales del diagrama de la figura 6.25.

3.5. Ejemplo de diseño RTL

Para terminar de afianzar los contenidos expuestos vamos a realizar el proceso completo con un mismo problema.

3.5.1. Especificaciones del diseño

El sistema sobre el que se realizará el diseño se muestra en la figura 6.26, de tal forma que se controlará el nivel de un tanque de líquido. En ella podemos distinguir el esquema del tanque y sus operaciones. Por lo tanto, tenemos dos señales de control, C_1 y C_0, que determinarán la operación que hay que realizar y el detector de nivel del tanque, *N*, que nos indicará la capacidad de llenado en cualquier instante de tiempo. Así mismo, tendremos una entrada más, denominada *R*, para un llenado preciso, y dos señales de control adicionales, *inicio* y *parada*, para el comienzo de la operación y una parada de emergencia.

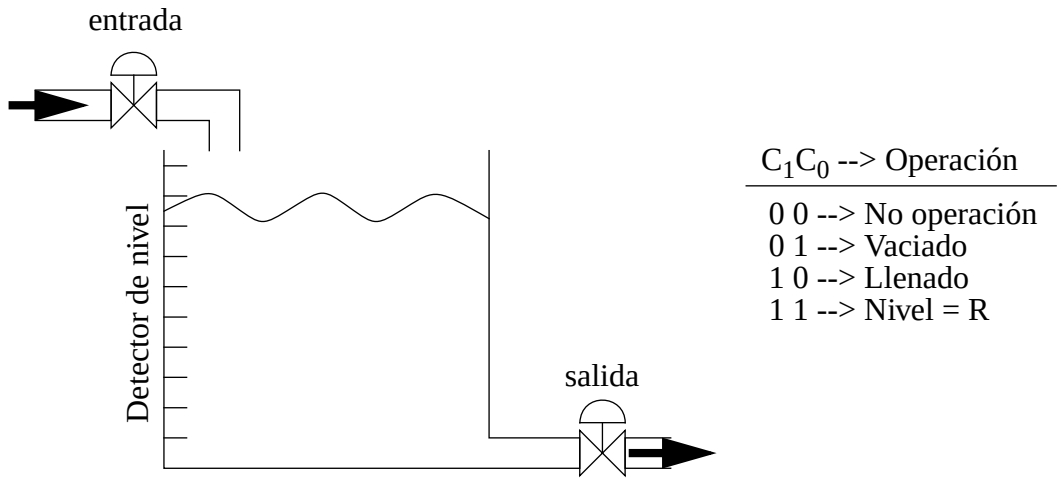


Figura 6.26.- Esquema de un controlador de nivel de un tanque de líquido.

La operación deberá empezar por un pulso de la señal de *inicio*. Tras lo cual, comenzará la operación determinada por las señales de control de operación. Esta operación se deberá mantener hasta que cambien las señales de control de operación, o bien si se pulsa la señal *parada*. En el caso de que se pulse la señal *parada*, el sistema deberá ir a un estado inicial de espera, del cual saldrá con un pulso de la señal *inicio*. En el caso de que hayan cambiado las señales de control de operación, el sistema, después de realizar la operación previa, deberá realizar la nueva operación.

Consideraciones:

- Tanto el detector de nivel como el nivel de llenado preciso deberán estar codificados en binario natural.
- Las señales de *entrada* y *salida* son tales que tomarán el valor ‘1’ cuando dejen pasar el líquido (llave abierta) y ‘0’ cuando no dejen pasar el líquido (llave cerrada).

3.5.2. Diagrama ASM

El sistema de control descrito con las especificaciones anteriores puede ser descrito con el diagrama ASM de la figura 6.27. En este diagrama podemos distinguir seis bloques ASM, que vamos a analizar a continuación.

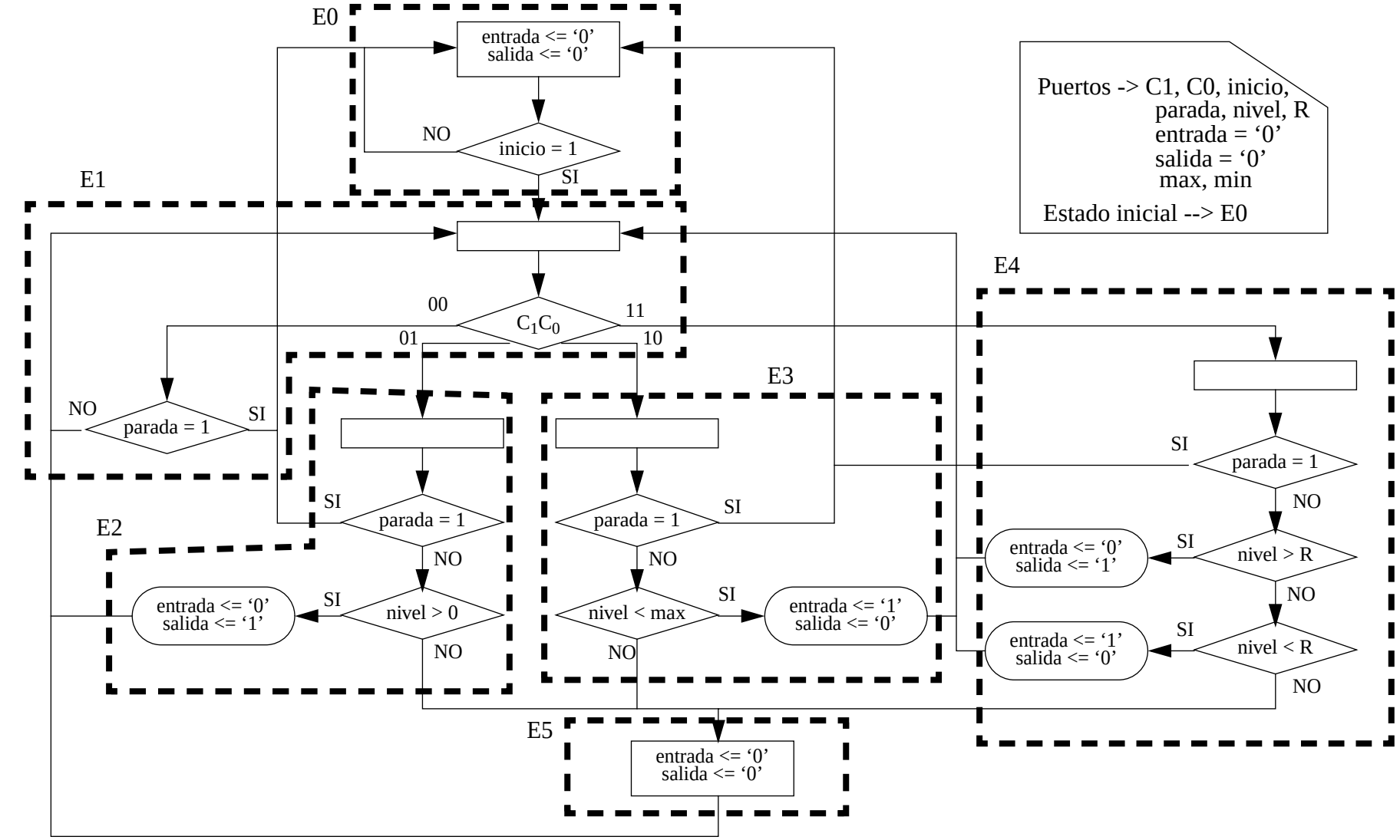


Figura 6.27.- Diagrama ASM del diseño ejemplo a resolver.

El estado E0 será el estado de inicio, de tal forma que mientras no exista un pulso en la señal *inicio*, el sistema permanecerá en dicho estado abriendo ambas llaves de paso para mantener el mismo nivel de líquido.

El estado E1 decidirá qué operación se deberá realizar en función de los valores de las señales de control de operación, C_1 y C_0. El sistema permanecerá en este mismo estado mientras la codificación de control de operación es “00”.

El estado E2 es el estado de operación de vaciado. Sólo se podrá sacar al sistema de este estado en dos situaciones: activar la señal parada, en cuyo caso deberá ir al estado inicial; o cambiar el código de las señales de control de operación, en cuyo caso deberá ir al estado correspondiente a la nueva operación.

El estado E3 es el estado de operación de llenado. Sólo se podrá sacar al sistema de este estado en dos situaciones: activar la señal parada, en cuyo caso deberá ir al estado inicial; o cambiar el código de las señales de control de operación, en cuyo caso deberá ir al estado correspondiente a la nueva operación.

El estado E4 es el estado de operación de llenado preciso, por lo que el nivel del tanque debe tender a un nivel fijado por el puerto *R*. Sólo se podrá sacar al sistema de este estado en dos situaciones: activar la señal parada, en cuyo caso deberá ir al estado inicial; o cambiar el código de las señales de control de operación, en cuyo caso deberá ir al estado correspondiente a la nueva operación.

Finalmente, el estado E5 es el estado al que se llega desde cualquier estado de operación (E2, E3 o E4) si ya se ha cumplido la operación correspondiente. De esta manera, no se podrá alterar el nivel del tanque.

3.5.3. Módulos y conexionado del procesador

El sistema que queremos diseñar se trata de un controlador, por lo que su mayor parte estará formado por el controlador. Luego el procesador estará formado por los bloques necesarios para las diferentes tomas de decisión, y no para el procesado de las señales.

En nuestro caso particular, el procesador estará formado por comparadores, necesarios para tomar las decisiones oportunas. En el diagrama observamos tres comparaciones:

- Nivel del tanque con el nivel ‘0’, en el caso de la operación de vaciado.
- Nivel del tanque con el nivel máximo, en el caso de la operación de llenado.
- Nivel del tanque con el nivel de llenado preciso, en el caso de la operación de llenado preciso.

Luego, una posible opción sería utilizar tres comparadores con sus correspondientes entradas. No obstante, esta elección no sería óptima en cuestión de recursos ya que ninguna comparación coincide en el tiempo con las otras. Así que consideraremos un único comparador con las entradas multiplexadas según la operación en curso. El esquema del procesador sería el mostrado en la figura 6.28.

La opción elegida será útil a nuestro sistema en el caso de que los requerimientos de

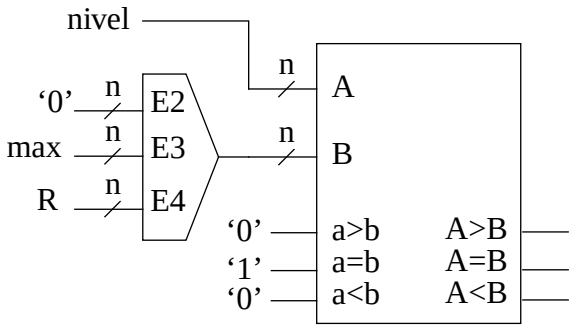


Figura 6.28.- Esquema del procesador del ejemplo de diseño.

velocidad nos permita mantener el retraso correspondiente a la conexión anterior, es decir, un multiplexor con un comparador. En caso contrario deberemos volver a este punto y continuar con otra opción más recomendada para las especificaciones.

3.5.4. Controlador

Para realizar el diseño del controlador, debemos obtener en primer lugar el diagrama de estado correspondiente. Para ello, basándonos en el diagrama ASM, utilizamos los diferentes bloques ASM como estados del controlador. Las entradas del controlador serán las entradas de control (*inicio*, *parada*, C_1 y C_0), y las señales de estado del procesador (las salidas del comparador); en cambio las salidas del controlador serán las salidas de control (*entrada* y *salida*) y las salidas de control del procesador (las señales de selección para el multiplexor que determinará las entradas del comparador). El diagrama de estados es mostrado en la figura 6.29.

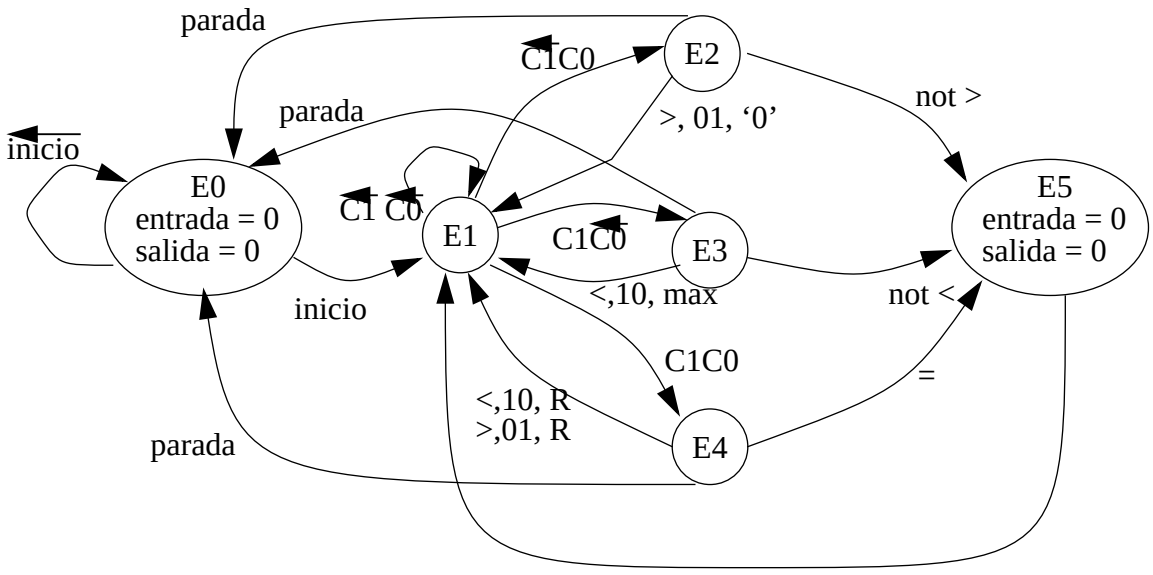


Figura 6.29.- Diagrama de estados del controlador del diseño ejemplo.

Vamos a indicar algunos comentarios breves sobre el diagrama.

- Las etiquetas de las transiciones corresponderán al formato:
  entradas, llave_de_entrada llave_de_salida, señales de control
- Para simplificar el diagrama sólo se considerarán en las transiciones aquellas señales que son relevantes.
- En el caso de que una salida sea función de un determinado estado, y no de una condición de entradas, se indicará dentro del recuadro del estado. Por ejemplo, siempre que estemos en el estado E0, ambas llaves estarán abiertas para no alterar el nivel.
- En el caso de la transición del estado E4 al E5, se tendría que cumplir que las comparaciones no fuesen ni mayor ni menor, por lo que al disponer del terminal igual, se ha optado por la utilización de este terminal. De esta forma eliminamos la puerta AND necesaria para garantizar la condición.
- Cuando se alcanza el estado E1, las salidas de control no deben cambiar ya que puede provenir de cualquier código de operación. Esta situación se puede conseguir de dos formas diferentes, basadas en el almacenamiento de estas señales en un biestable: activar o desactivar el terminal de habilitación correspondiente; o realimentar su valor actual. Como las salidas de control son un solo bit y sólo es necesario un biestable, vamos a elegir la segunda opción por si el biestable utilizado no tiene entrada de habilitación.

Una vez que disponemos del diagrama de estados, pasamos a la implementación del controlador. Para ello, seguimos los pasos del flujo de diseño mostrado en el tema 4: minimización del diagrama de estados, asignación de estados, tabla de transición, tabla de excitación e implementación lógica.

- Minimización del diagrama de estados.
  Este paso no se llevará a cabo para que la coincidencia con el diagrama ASM de partida, y por tanto del procesador, sea total.
- Asignación de estados.
  En este tipo de diseños, en los que hay un elevado número de estados, una codificación muy utilizada es la conocida como *one-hot*. Esta codificación se basa en utilizar una señal por cada estado. De esta forma, podemos obtener una expresión relativamente sencilla a partir del diagrama. Esta solución no es problemática ya que al tratarse de un circuito síncrono (y generalmente) con flip-flops, los problemas de las carreras no tendrán efecto. Siguiendo esta codificación, la señal de estado del estado inicial sería:

$$E0 = \overline{inicio} \cdot e0 + parada(e2+e3+e4)$$

- Tabla o ecuaciones de transición.
  Al partir directamente desde el diagrama de estado, daremos directamente las ecuaciones de transición. Estas ecuaciones no serán necesariamente mínimas ya que no serán aprovechadas las condiciones de inespecificación. No obstante, al considerar la relación tiempo de diseño / optimización, observamos que esta solución es aceptable. Luego, las ecuaciones de transición obtenidas se muestran a continuación:

$$E0 = \overline{inicio} \cdot e0 + parada(e2+e3+e4)$$

$$E1 = inicio \cdot e0 + \overline{C1} \cdot \overline{C0} \cdot e1 + \overline{parada}(mayor \cdot e2 + menor \cdot e3 + \overline{igual} \cdot e4) + e5$$

$$E2 = \overline{C1} \cdot C0 \cdot e1$$

$$E3 = C1 \cdot \overline{C0} \cdot e1$$

$$E4 = C1 \cdot C0 \cdot e1$$

$$E5 = \overline{\text{mayor}} \cdot e2 + \overline{\text{menor}} \cdot e3 + \text{igual} \cdot e4$$

$$\text{entrada} = e3 \cdot \text{menor} + e4 \cdot \text{menor} + e1 \cdot \text{entrada}$$

$$\text{salida} = e2 \cdot \text{mayor} + e4 \cdot \text{mayor} + e1 \cdot \text{salida}$$

$$\text{sel\_0} = e2$$

$$\text{sel\_max} = e3$$

$$\text{sel\_R} = e4$$

- Tabla o ecuaciones de excitación.
  En este caso, la mayoría de controladores son implementados utilizando biestables tipo D, en los que las tablas de excitación coinciden con las de transición.
- Implementación lógica.
  Una vez que se ha separado el comportamiento secuencial del combinacional, pasamos a obtener el circuito lógico según los métodos de la lógica combinacional vistos en la asignatura del cuatrimestre anterior. Como ya tenemos expresadas las funciones como fórmulas lógicas, pasamos a obtener directamente el circuito lógico. No obstante, previo a este paso debemos obtener la codificación correcta de las selecciones del multiplexor. Todo ello se muestra en la figura 6.30.

3.5.5. Conexionado global

Por último, se debe unir el procesador y controlador en un único circuito utilizando las señales de estado y de control del circuito correspondiente. Por lo tanto, el circuito global se muestra en la figura 6.31, donde las señales externas se han marcado en negrita para diferenciarlas de las conexiones internas (las cuales no se han puesto para clarificar el esquema).

Como se comentó al comienzo, la parte de mayor envergadura se encuentra en el controlador ya que esta es la funcionalidad principal del circuito. No obstante, este ejemplo nos sirve para ilustrar la diferencia entre el procesador (que por lo general contendrá dispositivos MSI, tales como sumadores) y el controlador (que, al ser un autómata finito, por lo general contendrá puertas y biestables).

3.5.6. Consideraciones finales

Por último, el siguiente paso consiste en verificar que el circuito obtenido cumple con las especificaciones de partida.

Las especificaciones de partida se pueden diferenciar en dos grandes grupos: de comportamiento y de caracterización. En el caso de las especificaciones de comportamiento, se debe garantizar la funcionalidad del diseño requerido. En cambio, en las especificaciones de caracterización se debe garantizar las características requeridas, como pueden ser el consumo de potencia o la velocidad de operación.

S1 S0	Selección
0 0	‘0’
0 1	max
1 0	R
1 1	--

S1 = sel_R = e4
S0 = sel_max = e3

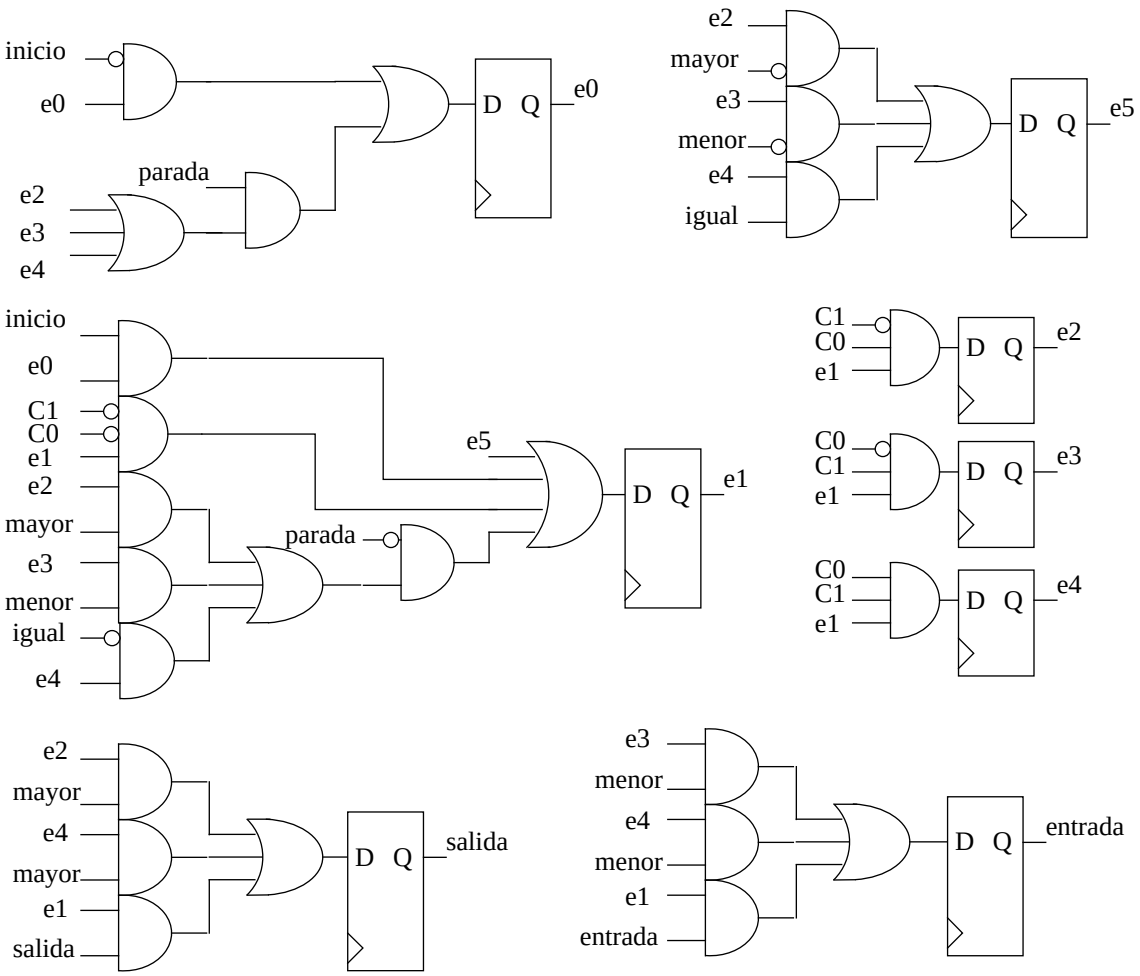


Figura 6.30.- Esquema lógico del controlador del diseño de ejemplo

En nuestro caso particular, vamos a comprobar las especificaciones de comportamiento a través de unas formas de onda tal que recorran todo el controlador. En la figura 6.32 mostramos las formas de onda mencionadas anteriormente. Podemos apreciar las siguientes señales:

- Señal de inicio
- Señal de parada
- Señal C1
- Señal C0
- Señal de nivel del tanque

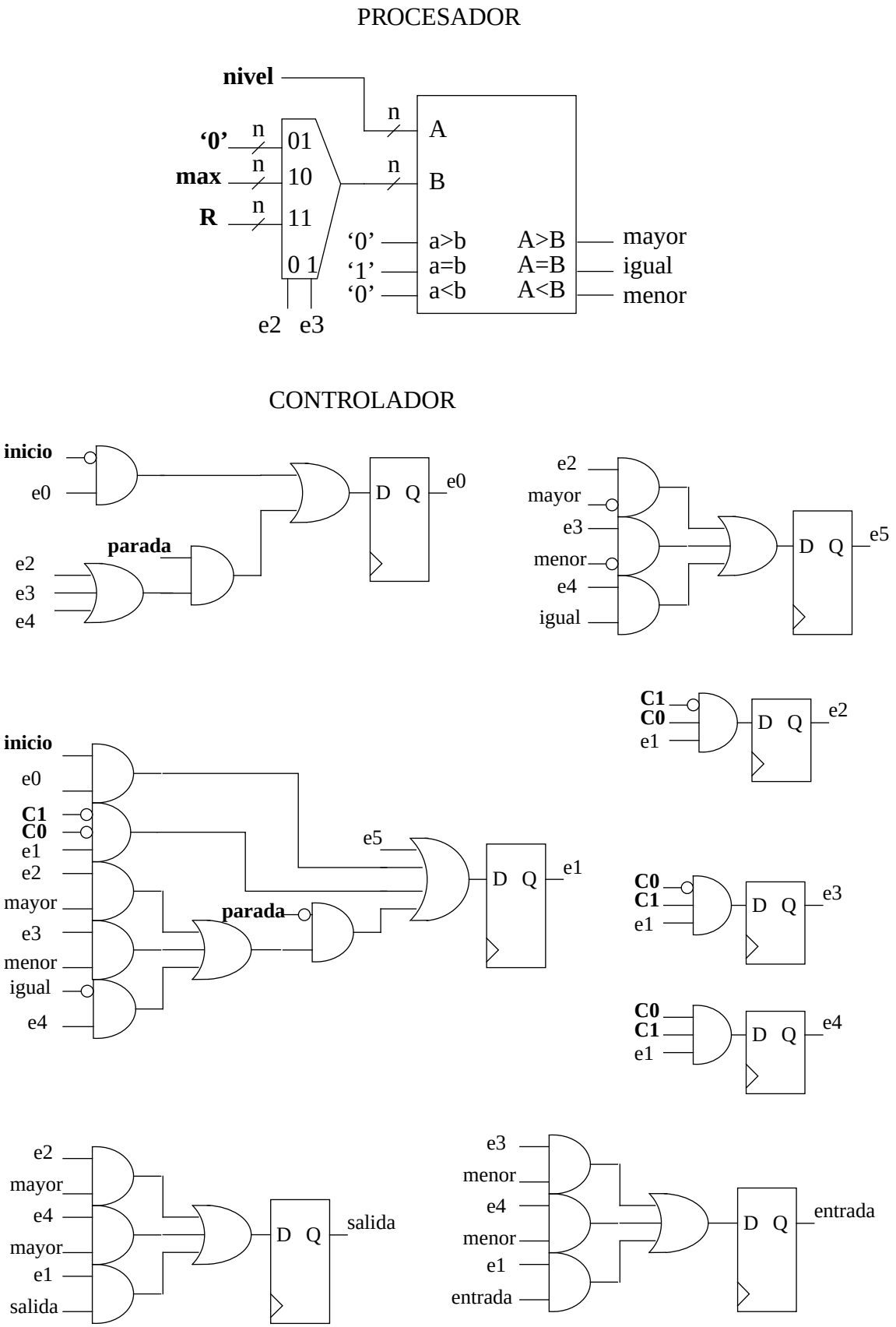


Figura 6.31.- Esquema lógico del controlador del tanque de líquido.

- Estado en el que se encuentra el controlador
- Entrada del comparador (ent_comp)
- Resultado de la comparación (comparación)
- Señal de la llave de entrada
- Señal de la llave de salida
- Señal de reloj del sistema

En el caso de las comparaciones sólo se ha identificado los valores en aquellos estados en los que se utiliza su valor; en el resto de los estados, al no tener ninguna relevancia, no se han indicado ni el valor de la entrada ni el del resultado de la comparación.

Podemos apreciar que se han incluido la mayoría de las situaciones en las que se puede encontrar el sistema: inicio, parada y cualquiera de las operaciones, finalizadas o sin finalizar. Según el diagrama podemos ver que la operación de parada sólo se realizará cuando esté chequeando las señales de control de operación, y no cuando esté realizando un paso de la operación, luego puede que se debiera dejar activa esta señal hasta que se llegue al estado E0. En el caso de que quisiéramos activar la parada desde cualquier estado deberíamos modificar el diagrama ASM, tal que en cada bloque ASM se incluya la caja de selección de parada.

Con respecto a las especificaciones de caracterización, no se ha indicado ninguna. A modo de ejemplo vamos a analizar una especificación de retraso. Para garantizar la operación correcta se debe verificar que el tiempo necesario para el cambio de valor en el detector de nivel sea superior al retraso del sistema para que no exista efecto rebote. En el caso de no verificarse esta especificación, deberíamos ir a implementaciones con retraso menor, pero a costa de incrementar los recursos; o la utilización de una llave con un caudal menor (por lo que el cambio de los sentidos de las comparaciones requerirán un tiempo mayor).

4. Temporizadores.

Dentro de la mayoría de los circuitos secuenciales, hacemos uso de señales de reloj, o en su defecto, de señales periódicas. Estas señales han sido tratadas hasta el momento como señales externas a nuestro sistema, y por tanto, no hemos considerado su generación. No obstante, su generación es función de sistemas secuenciales (que no entrarían dentro de la definición dada en esta asignatura) denominados genéricamente temporizadores. Una definición más formal de éstos puede ser la siguiente:

> Un **temporizador** es aquel sistema capaz de volver a un estado inicial después de un determinado espacio temporal, según se cumpla una determinada situación particular.

Consideremos el comportamiento del circuito mostrado en la figura 6.33. Cuando la señal A se encuentra a nivel bajo, la salida está a nivel alto. En cambio, cuando la señal A se encuentra a nivel alto, la salida cambia de estado cada vez que transcurre el retraso del inversor y de la puerta AND. Por lo tanto, encontramos una situación en la que se vuelve al estado anterior después de que transcurra un determinado tiempo (valor analógico, por lo que el circuito cae fuera de la definición de circuito secuencial) aunque no se produzca ningún cambio en las señales de entrada. Cuando estos circuitos carecen de señales de entrada, reciben el nombre de

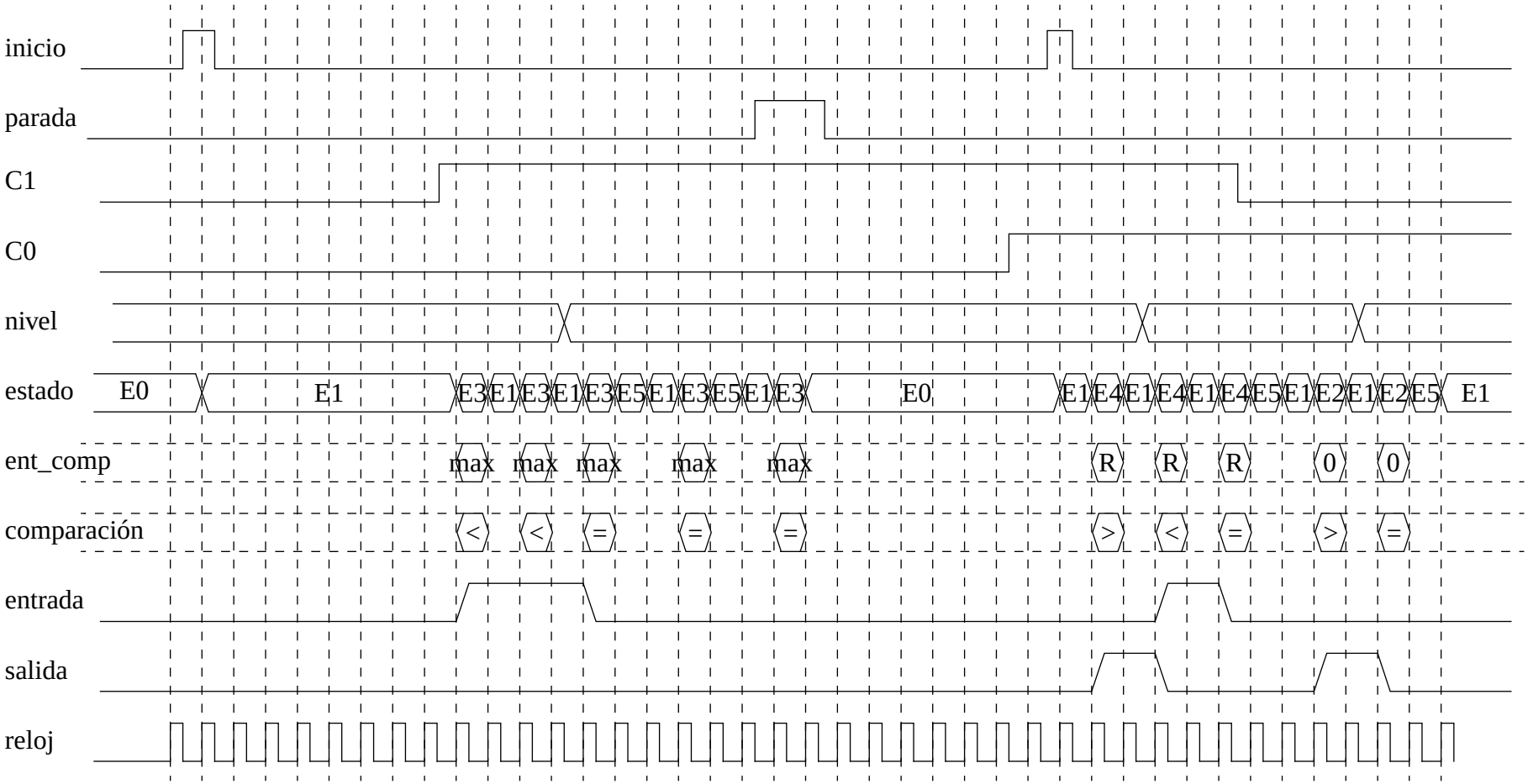


Figura 6.32.- Comportamiento funcional del circuito obtenido a través del diagrama ASM.

generadores de impulsos ya que no realizan ninguna otra función lógica.

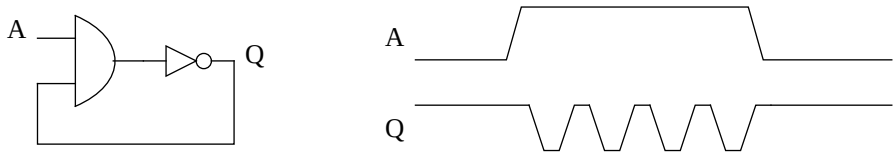


Figura 6.33.- Temporizador formado por una puerta AND y un inversor.

Los temporizadores pueden clasificar en dos grandes grupos: temporizadores no realimentados y temporizadores realimentados. En los temporizadores no realimentados, la dependencia temporal viene determinada por un elemento de retraso, como se puede apreciar en la figura 6.34. Cuando la señal de entrada A toma el valor lógico alto, existe un espacio temporal, determinado por Δ, en el que las dos entradas de la puerta AND tienen el valor lógico alto, apareciendo en la salida un pulso de nivel bajo cuya duración es aproximadamente el tiempo introducido por el elemento de retraso. Básicamente podemos decir que este tipo de circuitos aprovechan las situaciones típicas de los azares de la lógica combinacional. Los elementos de retraso pueden ser formados de muy diversa forma, como pueden ser redes analógicas RC o un número par de inversores conectados en cascada.

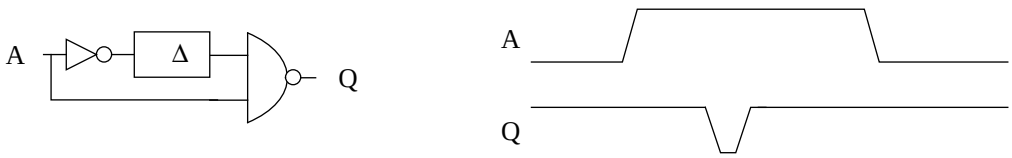


Figura 6.34.- Temporizador no realimentado.

En el caso de temporizadores realimentados, estamos ante circuitos secuenciales asíncronos que operan en el modo fundamental, por lo que el almacenamiento de la información se realiza por realimentación directa. En este caso, el tiempo de los pulsos estará condicionado por elementos de retraso explícitos y/o los retrasos de la lógica del circuito en cuestión. Un ejemplo de este tipo de temporizadores es el mostrado en la figura 6.33. Si nos centramos en los generadores de impulsos, la característica fundamental es que carecen de estados estables, por lo que también reciben el nombre de **circuitos astables**. La idea consiste en lograr un circuito cuya función lógica sea de la forma:

$$Q = \overline{q}$$

el cual no tendrá ningún estado estable. La forma más directa de obtener dicha función lógica será la conexión en cascada de un número impar de inversores, como se muestra en la figura 6.35. El problema de este generador de impulsos consiste en el poco control existente en el periodo de la señal obtenida. Este periodo será $n \cdot t_{inv}$, donde n es el número de inversores conectados en cascada y t_{inv} es el retraso de un inversor. El retraso de cada inversor va a depender de multitud de factores, entre los que se encuentra la tecnología de fabricación. Además, el retraso es diferente para cada transición, por lo que la señal tenderá a ser asimétrica.

Figura 6.35.- Generador de impulsos construido mediante inversores.

Para tratar de minimizar este problema en la mayor medida de lo posible, generalmente se introducen elementos de retraso cuyo valor sea muy superior al de los inversores. Con esta medida se logran dos objetivos:

- Minimizar la dependencia con la tecnología, ya que al ser despreciables los retrasos de los inversores, también lo serán sus variaciones.
- Tener un mayor control de los periodos de los impulsos, ya que el retraso es común a todas las transiciones, y muestra un rango de variación menor que el de un inversor.

Por lo tanto, en la FIGURA se muestra un posible esquema de un generador de impulsos con elementos de retraso.

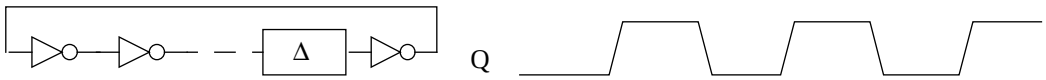


Figura 6.36.- Generador de impulsos construido mediante inversores y elementos de retraso.