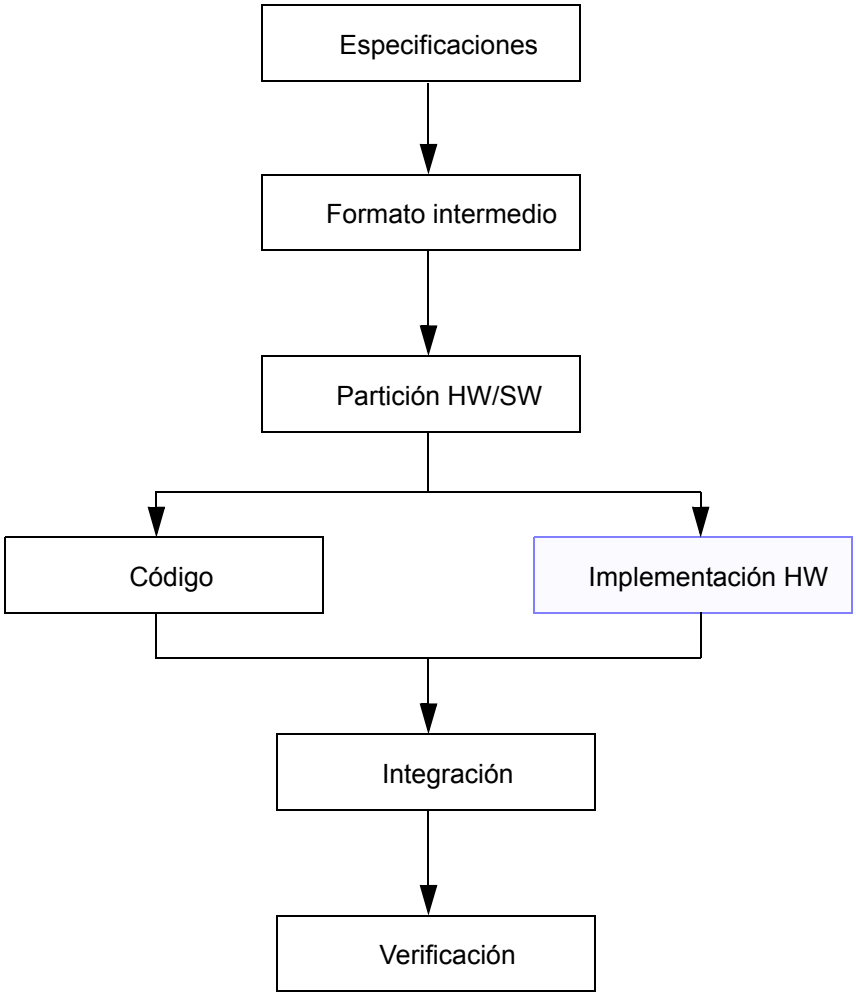
Tema III

Síntesis Hardware

Una vez que se ha realizado la partición HW/SW y conocemos las operaciones que se van a implementar por hardware y software, debemos abordar el proceso de estas implementaciones. Empezaremos por la implementación hardware, explorando las técnicas de síntesis de alto nivel.



3.1. Introducción

Para llevar a cabo esta tarea no podemos perder de vista la máxima de acortar el tiempo de diseño lo más posible, debido a la vista comercial de estos sistemas. Para ello, la síntesis hardware se realiza desde el más alto nivel, le cual se puede ver en la figura 3.1. Esto es debido a una serie de factores entre los que se pueden destacar los siguientes:

- Complejidad del diseño VLSI; si vamos a un diseño de más bajo nivel, el tiempo de diseño aumenta ya que el número de elementos con los que hay tratar crece considerablemente (un procesador o cientos de miles de transistores).
- Madurez de la tecnología VLSI; la tecnología está lo suficientemente desarrollada como para tener una buenas herramientas de diseño bien establecidas que dan lugar a unos diseño *buenos*.
- Mejor exploración del espacio de diseño; al subir el nivel de abstracción estamos haciendo el proceso más independiente de la tecnología (que si tuviésemos un proceso de más bajo nivel), por lo que podemos ampliar el espacio de diseño para utilizar los componentes más adecuados a nuestro problema. En el caso de tratar con procesadores, memorias y buses, no estamos restringiendo la arquitectura de los registros ni de las ALUs, y mucho menos las familias lógicas en las que se implementarán las puertas lógicas y flip-flops.

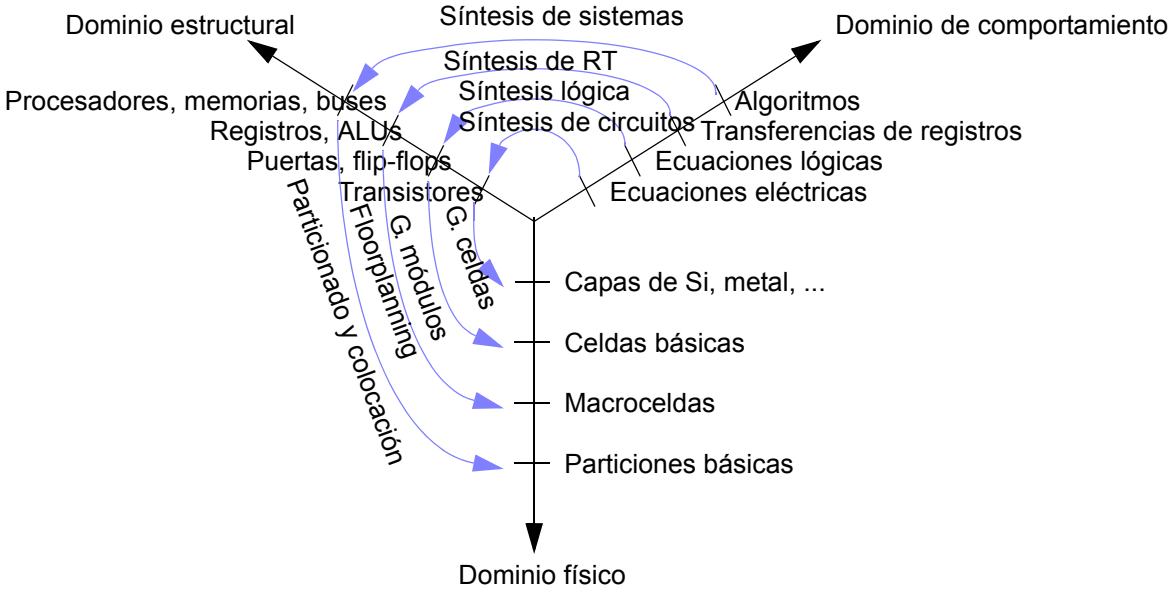


Figura 3.1.- Niveles de abstracción de un sistema electrónico.

Centrándonos en los sistemas empotrados, esta tarea de síntesis se debe desarrollar para aquellas aplicaciones que se deben implementar en un coprocesador (sistema hardware), para mejorar las prestaciones del sistema global. Por lo tanto, en primer lugar vamos a definir lo que se entiende por el proceso de síntesis.

El proceso de síntesis consiste en pasar de un dominio de abstracción a otro dominio; generalmente desde el dominio de comportamiento al dominio de estructura

Una vez que se haya realizado el diseño del sistema hardware, se debe garantizar una serie de restricciones de sus características, como pueden ser la velocidad del sistema, el consumo de potencia. No obstante, no se debería llegar a realizar la síntesis completa para determinar si se cumplen dichas restricciones, por lo que debemos utilizar una serie de métodos para estimar dichos parámetros en los pasos intermedios de la síntesis.

3.2. Procesos de diseño

Al disponer de cuatro grandes niveles de abstracción en cada dominio de aplicación, podemos tener cuatro síntesis de circuitos, según el nivel de abstracción en el que nos movamos. No obstante, en el campo de los sistemas digitales el nivel más bajo de abstracción no se utiliza a modo de síntesis (paso del dominio de comportamiento al de estructura). En el resto, los pasos a seguir serían los siguientes:

- Nivel lógico combinacional
  - Compilación
  - Minimización
  - Mapeado tecnológico
  - Dimensionamiento de transistores

En el proceso de compilación, se pasa de una descripción informal a una descripción interpretable por los algoritmos de síntesis (en este nivel a una tabla de combinaciones). Una vez que tengamos esta nueva descripción se realiza una proceso de minimización utilizando alguno de los algoritmos existentes (como pueden ser el método de Karnaugh o de McCluskey). Cuando se ha realizado la minimización se elige la tecnología a utilizar, tras lo cual se realiza una nueva optimización con el dimensionado de los transistores a valores óptimos.

- Nivel lógico secuencial
  - Compilación
  - Minimización de estados
  - Asignamiento de estados
  - Elección de biestables
  - Síntesis lógica combinacional

En el proceso de compilación se pasa de una descripción informal a una descripción interpretable por los algoritmos de síntesis (en este nivel a una máquina de autómatas finitos). Seguidamente se realizará un proceso de optimización que devolverá el número mínimo de estados. Una vez que tenemos el número mínimo de estados, tenemos que asignar una codificación binaria a cada uno de ellos. Finalmente elegiremos los biestables más adecuados. Tras lo cual únicamente tenemos que sintetizar el bloque combinacional con el proceso anterior.

- Nivel de transferencia de registros
  - Compilación
  - Scheduling
  - Allocation
  - Síntesis de la unidad de control
  - Síntesis del procesador

En el proceso de compilación se pasa de una descripción informal a una descripción interpretable por los algoritmos utilizados en el proceso de síntesis. Después se realizarán unos procesos de scheduling y de allocation para determinar en qué ciclo de operación se realizará cada una de las operaciones (scheduling) y cuáles serán los bloques que las implementarán (allocation). Una vez aclarado esto, se divide el sistema completo en una unidad de control y un procesador, cada parte implementado con los procesos anteriores.

- Nivel de sistemas
  - Compilación
  - Particionado
  - Síntesis de interfaces
  - Síntesis de procesado

En el proceso de compilación se pasa de una descripción informal a una descripción interpretable pos los algoritmos utilizados. Seguidamente se realiza un proceso de particionado para dividir el problema total en varios problemas menos complejos que puedan ser tratados de forma independiente. Una vez realizada esta tarea, se abordará la síntesis de las interfaces (bloques que conectan los diferentes bloques de procesado) y los bloques de procesado, que se llevarán a cabo con los procesos de síntesis anteriores.

Por lo que podemos apreciar, los procesos de síntesis no están desconectados sino que siguen un flujo top-down (desde un nivel de abstracción mayor, sistemas, a uno menor, lógica combinacional), como podemos apreciar en la figura 3.2. De hecho, el resultado de los niveles inferiores sirven como primitivas para el nivel inmediatamente superior.

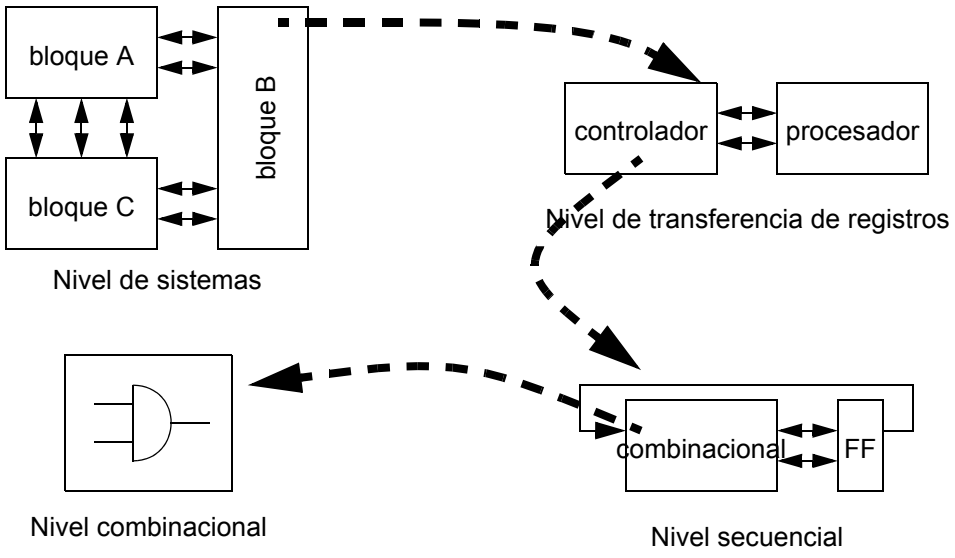


Figura 3.2.- Vistas de diferentes niveles de abstracción y la relación entre ellos.

Como ya hemos dicho, en el campo de los sistemas empotrados, la síntesis hardware se realizará desde alto nivel debido a criterios de tiempo de diseño, por lo que los procesos desde los niveles secuenciales y combinacionales suelen ser transparentes y realizados de forma automática. Así que desde ahora nos centraremos en los niveles de sistema y de transferencia de registros.

3.2.1. Compilación

La tarea de compilación se puede definir de la siguiente forma:

> La compilación es la tarea que transformará una descripción informal en otra que sea interpretable por los algoritmos utilizados en el proceso en cuestión.

Esta descripción puede ser realizada a través de lenguajes o a través de grafos. Dentro de los lenguajes de descripción de hardware, vamos a utilizar el lenguaje VHDL debido a que es uno de los primeros lenguajes que se liberaron y el número de herramientas que lo interpretan es mayor. En cambio, en la descripción basada en grafos vamos a utilizar los diagramas ASM (Algorithmic State Machine), que también suelen ser llamados CDFG (Control and Data Flow Graph).

3.2.1.1. Lenguaje VHDL

El lenguaje VHDL fue ideado en primer lugar para documentar los circuitos desarrollados por el ministerio de Defensa de USA. Después se desarrollaron herramientas para realizar tareas de simulación y síntesis utilizando este tipo de documentación (modelo).

Entre las principales figuras que están definidos en este lenguaje podemos encontrar las siguientes:

- Tipos de datos. Podemos encontrar diferentes tipos de datos, tanto conceptuales como estructurales. Entre los conceptuales podemos encontrar datos tipo bits, lógicos (el mismo que el bit pero incluyendo un número mayor de posibilidades como valores desconocidos, de alta impedancia y débiles), de tiempo, ... Por contra, entre los estructurales podemos encontrar escalares, matriciales y registros.
- Operadores. Dentro de los operadores podemos encontrar los diferentes operadores lógicos y de relación.
- Sentencias concurrentes. Estas funciones son aquellas que se ejecutan de forma paralela, unas a otras, y además de los operadores podemos encontrar sentencia de alto nivel como pueden ser bucles.
- Sentencias secuenciales. Estas funciones son aquellas que se ejecutan de forma secuencial, es decir, se ha de ejecutar la predecesora para poder ejecutar la siguiente. Estas suelen ser de alto nivel, aunque también se pueden incluir operadores. Estas sentencias se utilizan para describir algoritmos y deben estar contenidas en una sentencia concurrente especial denominada proceso.
- Definiciones de usuario. El usuario puede definir cualquier figura de las pertenecientes a las categorías anteriores, y utilizarlas de igual forma a las que vienen incluidas de forma estándar.

El modelo VHDL se divide en dos partes bien diferenciadas, las cuales existen con vista a una jerarquización del diseño. Una primera parte es la denominada entidad, en la que se describe la conexión con el exterior del sistema. Por ejemplo,

```
entity prueba is
port(  A, B: in bit_vector(7 downto 0);
```

```
        inicio: in std_logic;
        result: out bit_vector(8 downto 0)
        fin : out bit);
end prueba;
```

En este caso, tenemos un módulo denominado prueba, que dispone de cuatro puertos: tres de entrada (A, B e inicio, de los cuales A y B son dos buses de 8 bits codificados como tipo bit, y inicio que sería una señal codificada como std_logic) y dos de salida (result, que es un bus de 9 bits codificados como tipo bit, y fin que sería una señal codificada como bit).

La segunda parte del modelo se denomina arquitectura (architecture), en la que se especifica la arquitectura (valga la redundancia) del módulo descrito. Como un módulo puede ser implementado de diferentes formas, podemos tener diferentes arquitecturas para una misma entidad. Dentro de la arquitectura tendremos una parte declarativa, donde se declara todos los elementos que se vayan a utilizar (señales internas, componentes, ...), y otra parte en la que se interrelacionan dichos elementos. Por ejemplo,

```
architecture primera of prueba is
    type estado is (S0, S1, S2, S3, S4, S5);
    signal cont: bit_vector (3 downto 0);
    signal estado_presente, proximo_estado: estado;
begin
    P1:process(estado_presente)
    begin
        case estado_presente is
            when S0 =>
                cont <= "0000";
                result <= "000000000";
                fin <= '0';
                if (inicio = '1') then proximo_estado <= S1;
                else proximo_estado <= S0;
                end if;
            when S1 =>
                if (A < B) then proximo_estado <= S2;
                elsif (A > B) then proximo_estado <= S3;
                else proximo_estado <= S4;
                end if;
            when S2 =>
                result <= B;
                cont <= cont+1;
                if (cont < "0101") then proximo_estado <= S2;
                else proximo_estado <= S5;
                end if;
            when S3 =>
                result <= A;
                cont <= cont+1;
                if (cont < "0010") then proximo_estado <= S3;
                else proximo_estado <= S5;
                end if;
            when S4 =>
                result <= A+B;
                cont <= cont+1;
                if (cont < "1000") then proximo_estado <= S4;
```

```
                else proximo_estado <= S5;
                end if;
            when S5 =>
                fin <= '1';
                proximo_estado <= S0;
        end case;
    end process;
    B1:block (clk = '1' and not clk'stable)
    begin
        estado_presente <= guarded proximo_estado;
    end block;
end;
```

(Si introducimos este código directamente en un compilador VHDL, seguramente dará errores ya que para realizar las sumas aritméticas directamente, debemos cargar la librería aritmética).

3.2.1.2. Diagramas ASM

En cambio, un diagrama ASM es una representación basada en grafos, más intuitivo para el usuario, pero de mayor dificultad de tratamiento por herramientas no adecuadas. Está orientada hacia el nivel de transferencia de registros, por lo que en primer lugar definiremos este término:

> Una transferencia de registro se puede definir como una operación sobre unos datos mientras pasan de un registro a otro.

En este tipo de representaciones, únicamente existen tres elementos diferentes:

- Cajas de estado, que son cajas en cuyo interior puede existir una transferencia de registro, las cuales se realizarán en el siguiente ciclo de operación. También suele pasar que encontremos algunas de estas cajas vacías, en cuyo caso, su única función será esperar al siguiente ciclo de operación. Estas cajas suelen estar indicadas como cajas rectangulares.
- Cajas de selección, que son cajas en cuyo interior habrá una condición, que según se cumpla o no, el flujo de señal irá por un camino o por otro. El resultado de la condición no tiene porqué ser un valor booleano (verdadero o falso), sino que puede tener más de dos posibles valores, como las salidas de decodificador. Estas cajas suele estar indicadas por un rombo.
- Cajas de condición, que son cajas muy similares a las cajas de estados. La única diferencia entre ellas es que la transferencia indicada en esta caja se realiza en el ciclo de operación en curso, y no en el siguiente. Luego, la existencia de estas cajas dará lugar a un controlador basado en una máquina de Mealy. Estas cajas suelen estar indicadas por una caja con las esquinas ovaladas.
- A estos tres elementos se le suele añadir una nueva caja identificadora en la que se declaran las diferentes señales de dato que intervienen en el sistema (tanto de entrada, salida como internas) y sus valores iniciales para que el sistema opere correctamente.

El sistema modelado en el lenguaje VHDL del apartado anterior estaría representado por el siguiente diagrama ASM.

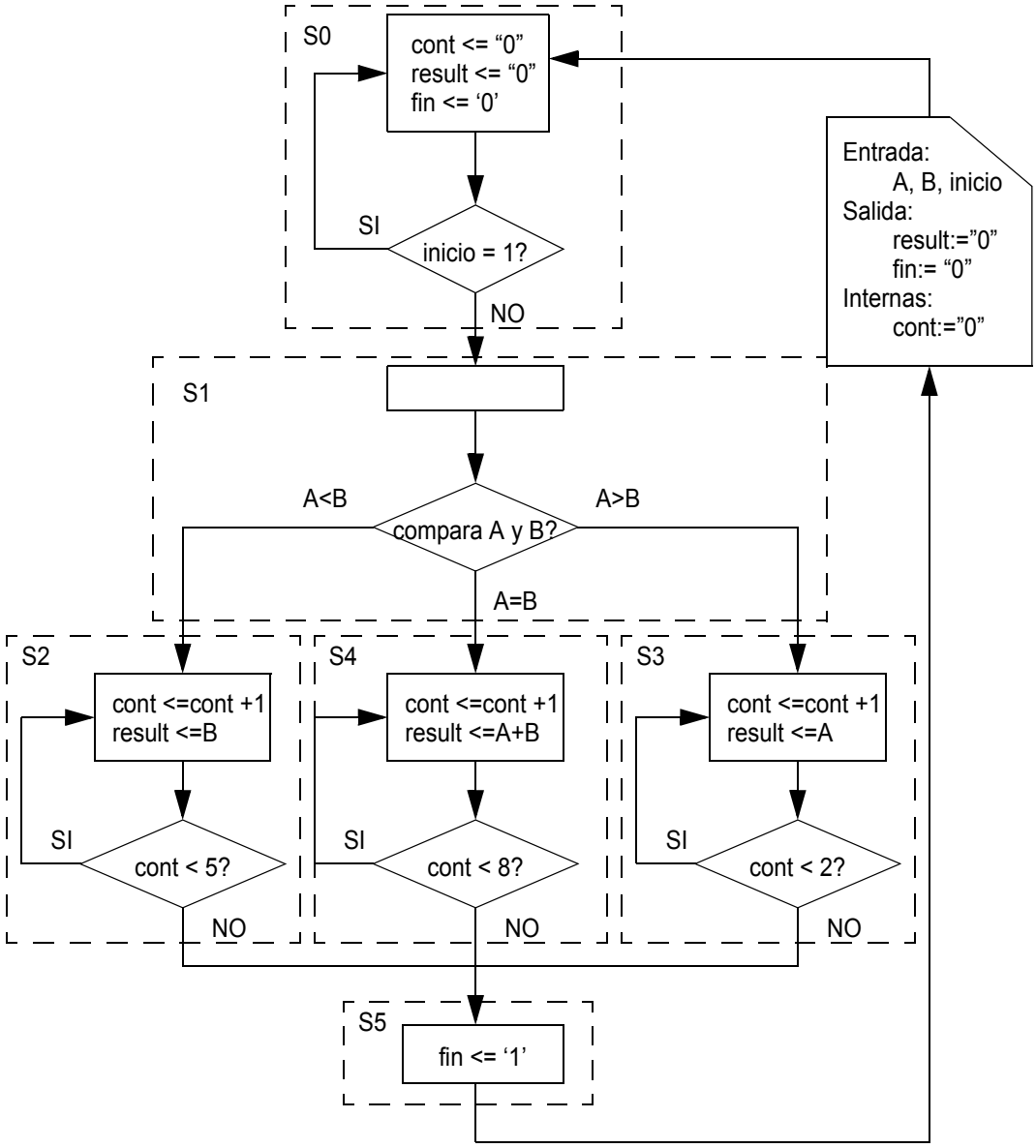


Figura 3.3.- Ejemplo de un diagrama ASM.

3.2.2. Particionado

Realmente el particionado está formado por una serie de técnicas cuya misión primordial consiste en descomponer un grafo en varios subgrafos que puedan ser tratados de forma independiente. Por lo tanto, esta tarea puede ser aplicada en las diferentes partes del flujo de diseño de alto nivel.

Al poder utilizarse en las diferentes partes del flujo de diseño, podemos hacer uso de lo aprendido en el apartado 3.3 del tema III.

3.2.3. Scheduling

La tarea de scheduling se puede definir de la siguiente forma:

Scheduling es la tarea según la cual se asigna las operaciones a instantes de tiempo (pasos de control), según restricciones y minimizando una función de coste.

Es decir, en el diagrama de la figura 3.3 habría un paso de control (ciclo de operación) por cada subgrafo (que representan los diferentes ciclos de reloj. Por lo tanto, las tareas asignadas a los ciclos de operación serían las mostradas en la figura 3.4, en las que también se han añadido los ciclos de operación en los que se deberían realizar. Obviamente esta elección no es la óptima, e incluso podríamos decir que buena, ya que existen un elevado número de ciclos sin que se realice ninguna operación, ya que deberíamos considerar que únicamente se ejecutará uno de los tres siguientes ciclos: S2, S3 y S4. No obstante, será la de partida al ser obtenida directamente del diagrama.

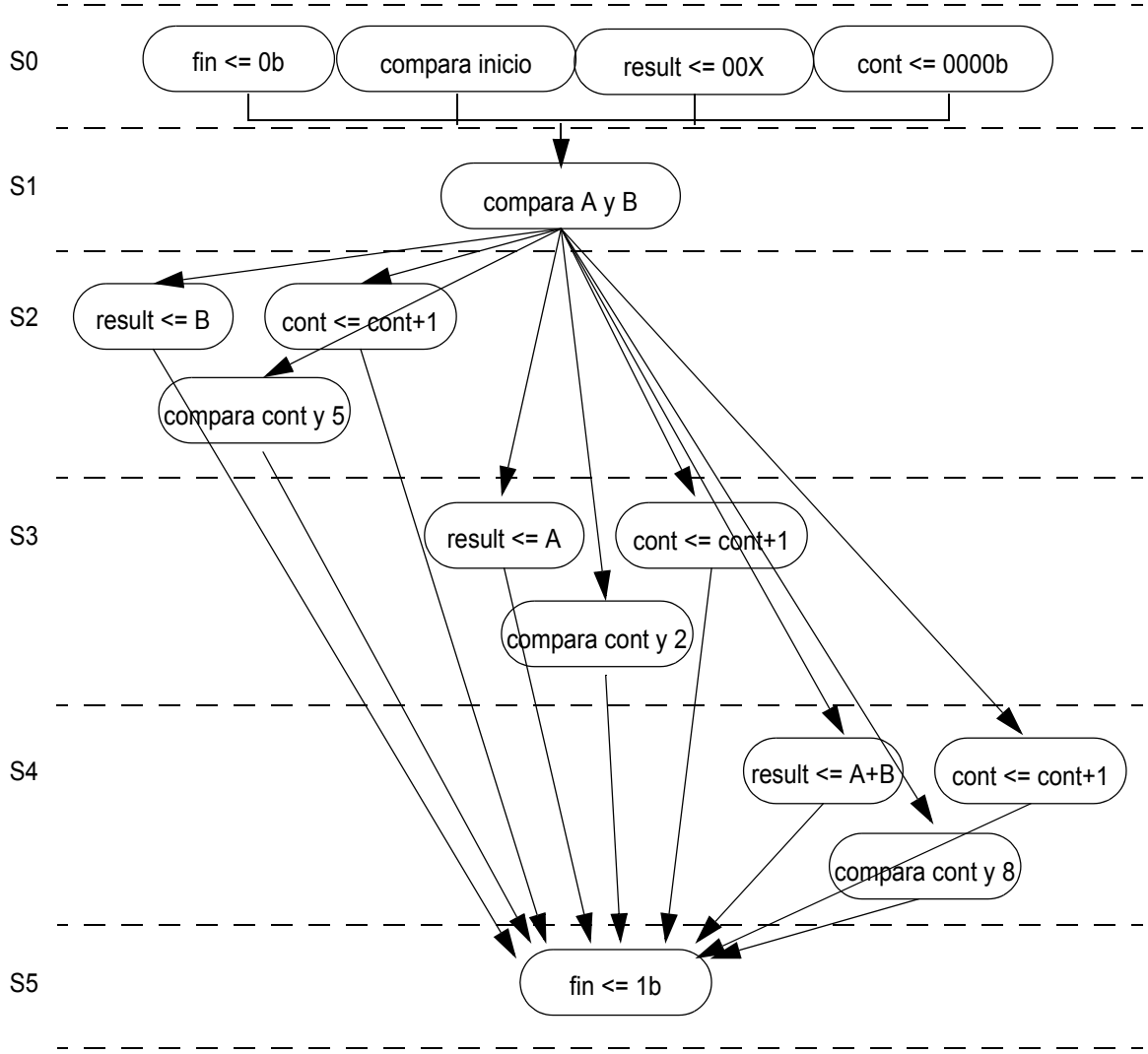


Figura 3.4.- Posible scheduling para el diagrama de la figura 3.3.

El siguiente paso será la obtención de un scheduling *bueno* sino óptimo. Para ello se parte de dos posibilidades, denominadas ASAP y ALAP. El algoritmo ASAP (As Soon As Posible) se basa en realizar las operaciones tan pronto como sea posible. En cambio, el algoritmo ALAP (As Late As Posible) se basa en realizar las operaciones tan tarde como sea posible. La figura 3.4 se puede ver como un algoritmo ASAP, ya que aunque las operaciones de los estados S2, S3 y S4 pueden realizarse en el mismo ciclo, no se colocan de esa forma ya que no se pueden realizar los tres pares de operaciones en el mismo instante al ser estados excluyentes (estarán en uno o en otro pero no en los dos a la vez).

En la figura 3.5 se muestra una versión ALAP del mismo sistema. Podemos observar que coincide casi en su totalidad con el caso ASAP, excepto en que la puesta a 0 del contador se realiza en el estado S1. Las otras dos puestas a 0 (fin y resultado) no pueden bajar ya que son las señales de salida que deben ser inicializadas forzosamente al comienzo de la operación.

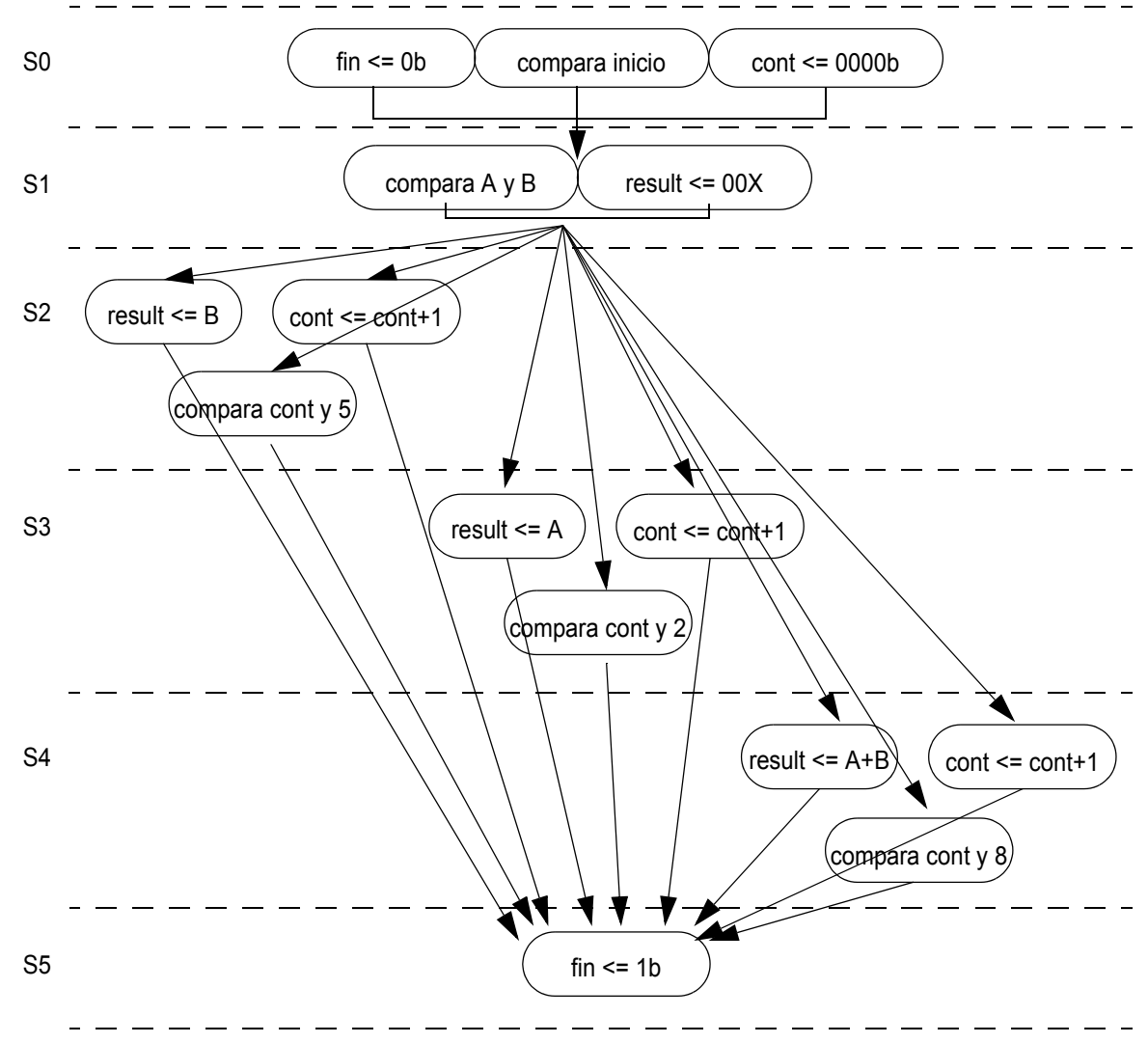


Figura 3.5.- Scheduling ALAP para el diagrama de la figura 3.3.

Una vez que hemos realizado uno de los dos scheduling anteriores, se deben redefinir para llegar a una solución óptima. Esta solución óptima será alcanzada tratando dos objetivos fundamentales: restricciones temporales y restricciones de recursos.

En las restricciones temporales se fija el número de ciclos de operación. Una vez definido estos ciclos de operación, las diferentes operaciones se deben redistribuir para poder minimizar el número de unidades funcionales. Las unidades funcionales vamos a dividirlas en unidades de procesado y unidades de almacenamiento. En el caso de las unidades de procesado (su valor no va a depender del instante temporal en el que se ejecute), aquellas operaciones del mismo tipo que se deban ejecutar en ciclos diferentes se pueden implementar con la misma unidad de procesamiento. En el caso de las unidades de almacenamiento (su valor va a depender del instante temporal en el que se ejecuten), aquellas señales cuyo valor no es necesario mantenerlo almacenado en algún ciclo en común pueden ser almacenadas en la misma unidad.

En las restricciones de recursos se fija el número de unidades funcionales de cada tipo. Una vez definidos estos número, se permitirá la duplicidad de operaciones según el número anterior. Se puede hacer uso de esta duplicidad para eliminar ciclos de operación al poder realizarse varias operaciones del mismo tipo en el mismo ciclo.

En el ejemplo anterior, y con las restricciones de utilizar el número mínimo de recursos y de ciclos temporales, un scheduling bueno (sino óptimo porque se ha realizado a mano sería el mostrado en la figura 3.6. En este scheduling, además de las relaciones de precedencia en función de los resultados de comparación, también se han incluido las relaciones de dependencia de dato (que no se incluyeron en los algoritmos anteriores ya que no influían en operaciones internas). También se han unido los tres bloques en los que la operación de un bloque excluía la operación de los otros dos, y con la operación común a los tres (la cuenta) extraída de los tres.

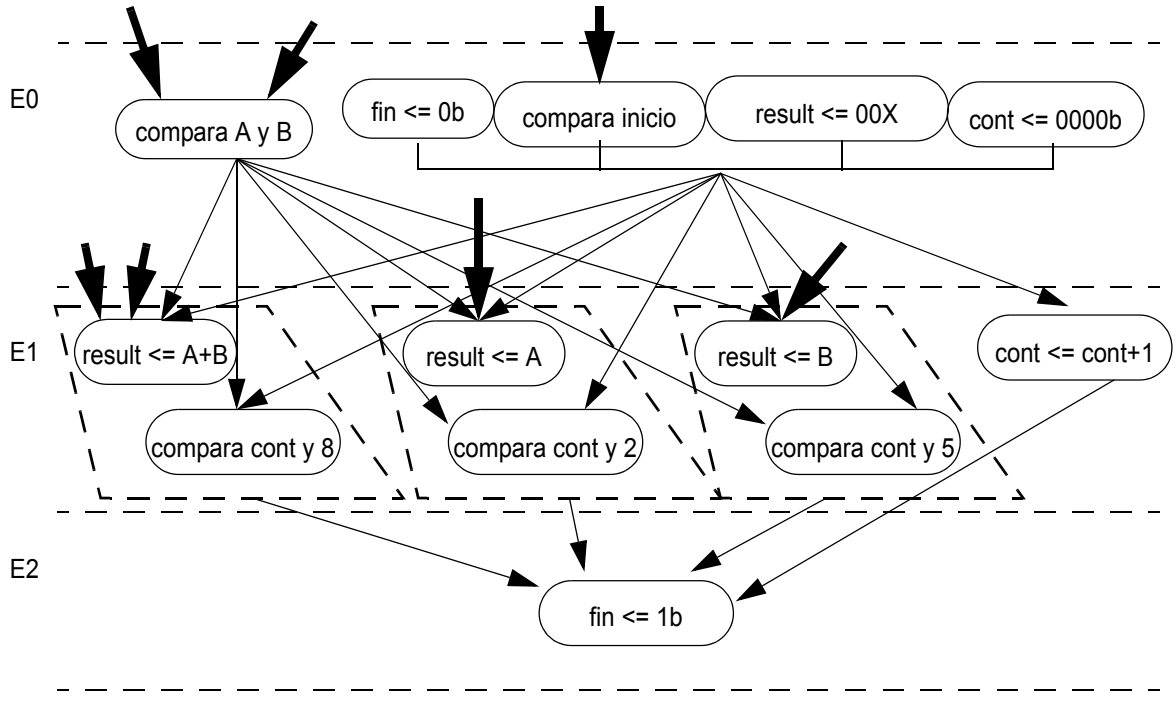


Figura 3.6.- Resultado del scheduling realizado a mano.

Esta elección de scheduling ha cambiado el diagrama ASM de cuatro posibles ciclos de operación (contando sólo el bloque que se debería ejecutar) a tres. Por lo tanto, el diagrama quedaría de la forma mostrada en la figura 3.7.

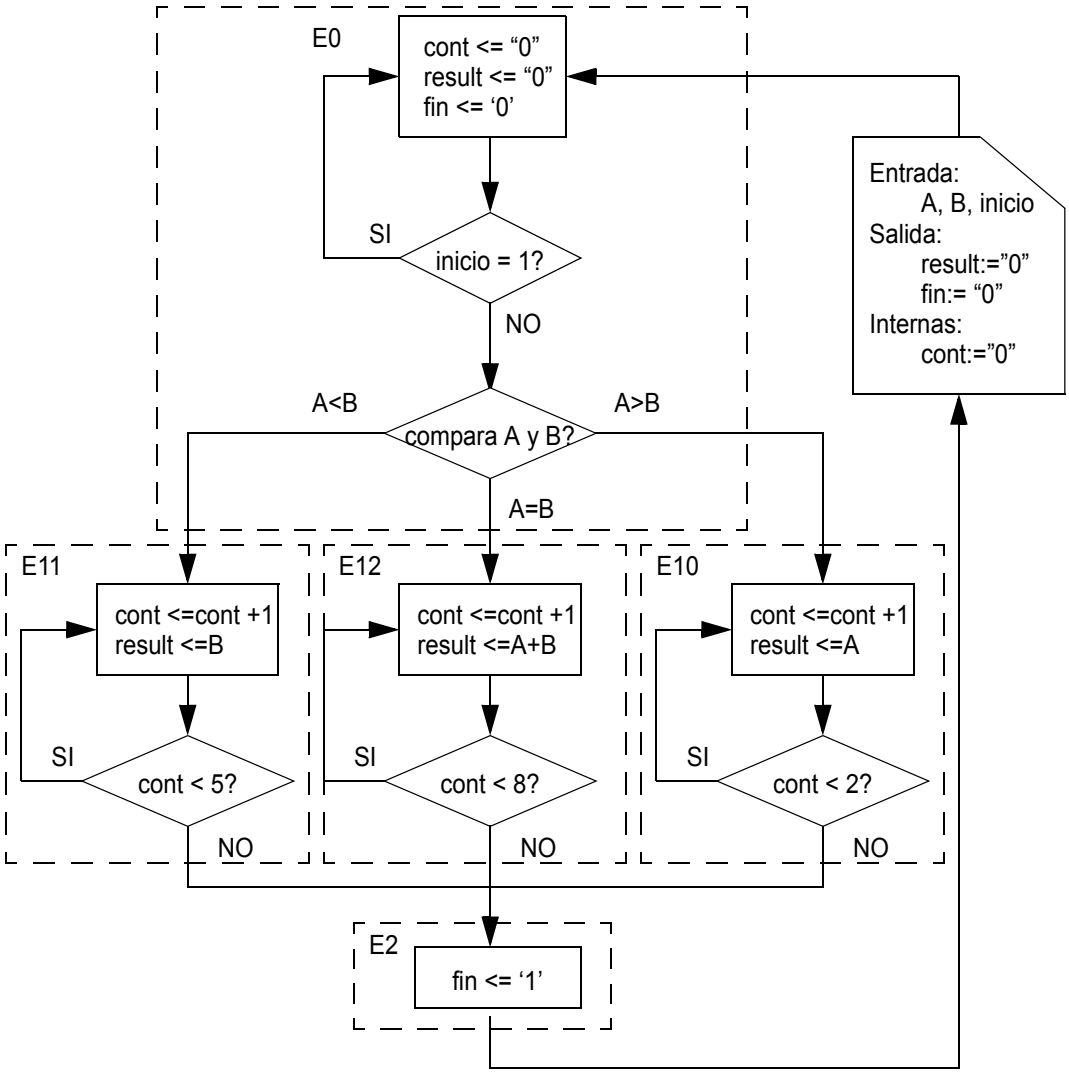


Figura 3.7.- Diagrama ASM tras el proceso de scheduling.

3.2.4. Allocation

La tarea que nos ocupa se puede definir de la siguiente forma:

La allocation es la asignación de operaciones a módulos hardware, según el scheduling y restricciones, de tal forma que se minimice una función de coste.

Por lo tanto, esta tarea está íntimamente relacionada con la anterior, hasta tal punto que sería lógico incluir las dos tareas en un bucle hasta encontrar la mejor solución.

A la hora de llevar a cabo esta asignación hay que tener en cuenta una serie de características arquitecturales, entre las cuales podemos destacar:

- Características de los módulos, como pueden ser la codificación triestado
- Topología de las interconexiones:

- Bus, las interconexiones se realizan a través de buses, por lo que sería deseable que los módulos tuviesen la conexión triestado para poder realizar la conexión directa de los diferentes módulos al bus.
- Multiplexor, las interconexiones a las entradas de los diferentes módulos se realizan a través de multiplexores, por lo que hay que generar las señales de control de los multiplexores.
- Conexión punto a punto, las conexiones van directas entre dos unidades funcionales (únicamente puede existir una unidad de interconexión entre dos unidades funcionales).
- Transferencia entre registros, las conexiones entre dos unidades funcionales siempre deben pasar por un registro (además de las unidades de interconexión necesarias).

Una vez que hemos decidido las características arquitecturales del sistemas que va a ser desarrollado, debemos pasar a elegir los diferentes módulos. En la selección de los módulos se debe elegir una librería de componentes, con unas determinadas características para que se adecúen a las restricciones del diseño. Particularizando para cada tipo de módulo podemos distinguir:

- Unidades funcionales, se debe elegir qué operación o conjunto de operaciones se van a realizar en cada unidad funcional. Si tenemos varias operaciones del mismo tipo que se ejecuten en ciclos diferentes, todas ellas pueden ser implementados en la misma unidad.
- Unidades de almacenamiento, se debe decidir dónde se van a mantener los datos y variables, ya sean en memorias o registros. Con el fin de minimizar su número, las variables cuyo tiempo de vida (ciclos en los que su valor se debe mantener almacenado) no se solape pueden compartir la misma unidad.
- Unidades de interconexión, tiene el mismo tratamiento que las unidades funcionales.

Siguiendo con el mismo ejemplo de la figura 3.3, y considerando una librería de celdas estándar, se necesitarían los siguientes elementos: comparadores, contadores, sumadores y registros; y por ser una librería estándar utilizaremos una conexión a través de multiplexores. Por lo tanto, se necesitará los siguientes elementos:

- 1 comparador,
- 1 contador,
- 1 sumador,
- 1 registro,

conectados de la forma que se muestra en la figura 3.8.

Con esto acaba la síntesis del camino de datos. Lo último que nos queda será la síntesis de la unidad de control, que sigue el proceso de síntesis de un sistema secuencial partiendo de un diagrama de estados. El diagrama de estado se obtiene directamente del diagrama ASM final, considerando cada estado como aquella porción de diagrama existente entre dos cajas de estados. Por lo tanto, para el ejemplo seguido, el diagrama de estado se muestra en la figura 3.9. Las salidas de este controlador deben ser tales que se cumplan la tarea de scheduling, tal como se eligió.

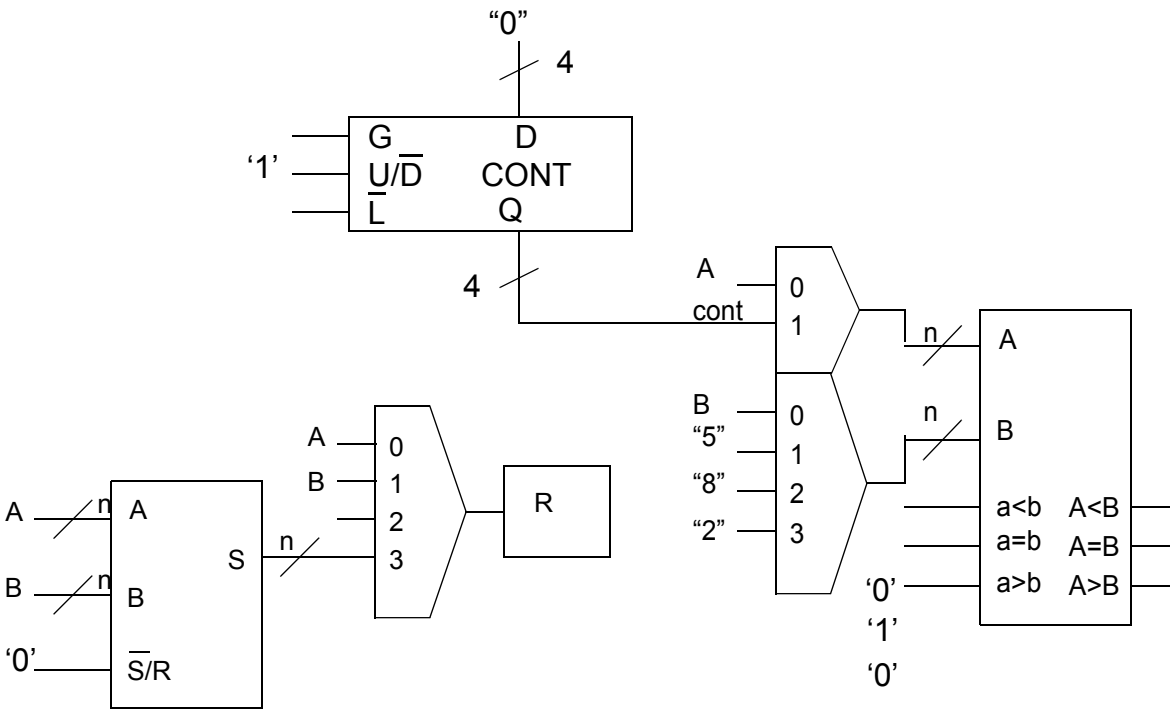


Figura 3.8.- Bloques resultantes del proceso de allocation.

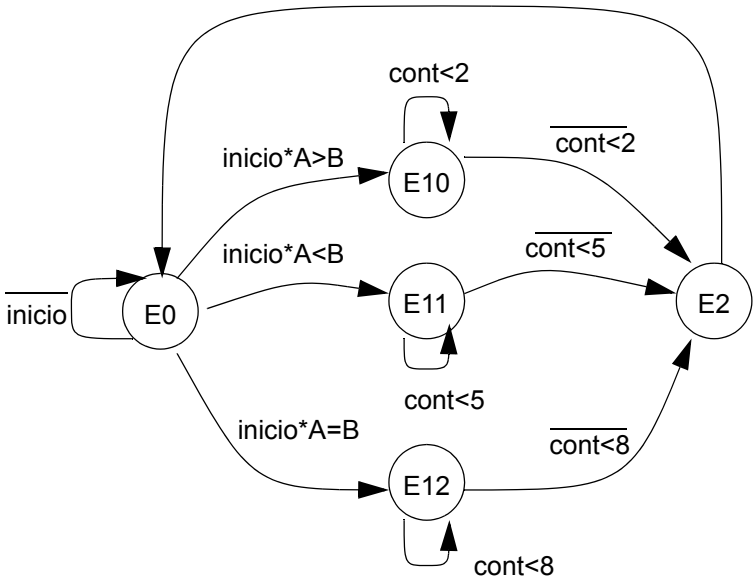


Figura 3.9.- Diagrama de estados correspondiente al controlador del ejemplo.

3.3. Estimación de prestaciones

Hasta ahora, hemos presentado el flujo de un sistema hardware desde alto nivel, pero en todas las tareas de diseño hay que mantener unas funciones de coste por debajo de un valor mínimo. Esta situación también se debe cumplir en el sistema final, no obstante no sería deseable comprobar la función de coste cuando ya se ha realizado el ciclo de diseño completo (por

motivos obvios). Además también puede ser que la función de coste elegida no sea la adecuada y se deban cambiar las especificaciones de diseño para el correcto comportamiento.

Las principales prestaciones que se suelen evaluar son el tiempo de ejecución de cada operación, que nos va a indicar la velocidad del sistema; el área ocupada, que nos va a indicar el tamaño del sistema donde puede ir ubicada la parte hardware; y el consumo de potencia, que nos va a indicar el tiempo de vida de un sistema sin cable. Todos estos parámetros están muy relacionados con la arquitectura destino; por lo tanto, vamos a suponer una arquitectura típica en la que se encuentra separada la unidad de control y de procesado (o camino de dato), como se muestra en la figura 3.10. En esta figura se ha particularizado la unidad de control por su modelo de sistema secuencial (lógica combinacional realimentada a través de registros), y la unidad de procesado (por una conexión mediante multiplexores, para los registros y para la unidad funcional).

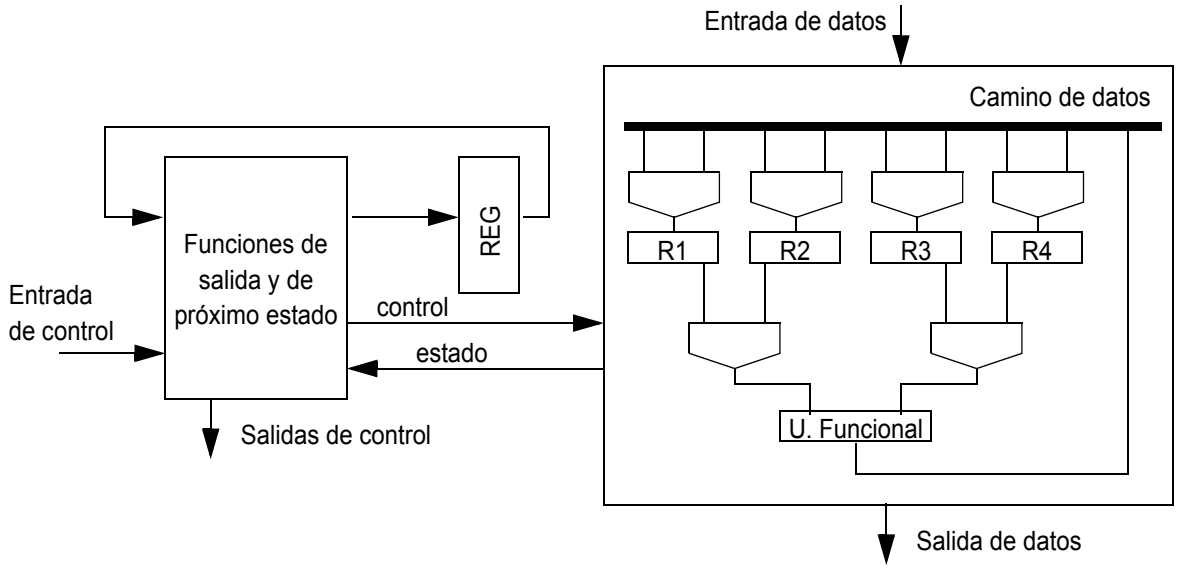


Figura 3.10.- Arquitectura de un controlador más camino de dato

3.3.1. Tiempo de ejecución

Según la arquitectura anterior, el tiempo de ejecución de una operación vendrá determinado por el retraso de la unidad de control y del camino de datos (datapath). En el camino de dato, el retraso global se debe a las siguientes acciones:

- Multiplexores de entrada a los registros
- Lectura de los operandos de entrada
- Multiplexores de entrada a la unidad funcional
- Retraso de la unidad funcional
- Tiempo de setup del registro donde se almacenará el resultado (para un almacenamiento correcto).

En cambio, el retraso global de la unidad de control se debe a las siguientes acciones:

- Retraso de la lógica de próximo estado
- Almacenamiento en el registro de estado
- Tiempo de setup del registro de estado
- Retraso de la lógica de control que actúa sobre el camino de dato.

Por lo tanto, el retraso global del sistema, que coincidirá con el periodo mínimo de la señal de reloj será el mostrado en la ecuación (4.3). No obstante, no podemos perder de vista que esta estimación se realizará desde alto nivel, por lo que las contribuciones de las conexiones (tanto por carga como por longitud) no serán consideradas.

$$t_p(DP) = t_{p(\text{MUX_R})} + t_{p(\text{Lectura})} + t_{p(\text{MUX_F})} + t_{p(\text{Unidad Funcional})} + t_{p(\text{Setup})} \quad (4.1)$$

$$t_p(CU) = t_{p(\text{Próximo estado})} + t_{p(\text{Setup})} + t_{p(\text{Lectura})} + t_{p(\text{Lógica de control})} \quad (4.2)$$

$$t_{clk} = t_p(DP) + t_p(UC) \quad (4.3)$$

El algoritmo de estimación utilizado deberá simular el proceso de síntesis completo, incluyendo todos los niveles posibles de la jerarquía utilizada. Para ello se deben obtener dos parámetros:

- El tiempo de ejecución de cada comportamiento elemental (nivel de la jerarquía más bajo), es decir, un microestado que se entiende como el conjunto de operaciones que se realizan en un ciclo de reloj. Para obtener el primer parámetro, debemos calcular el número de microestados, el cual coincidirá con el número de pasos diferentes obtenidos del proceso de scheduling.
- El tiempo de ejecución del siguiente comportamiento en la jerarquía, utilizando cálculo de probabilidades de transición entre comportamientos.

A la hora de obtener el tiempo de ejecución de los niveles superiores de jerarquía, hay que obtener los macronodos del sistema que muestran la siguiente definición

> un macronodo es el conjunto de sentencias que se pueden ejecutar en el mismo ciclo de operación

La diferencia que existe entre los macronodos y los microestados está en que los macronodos consideran todas las operaciones de un ciclo de operación, aunque se realicen de forma concurrente y únicamente haya que considerar la más lenta (mayor retraso). Los macronodos se logran identificar después de realizar un tarea de scheduling, que al tratarse de una estimación se utilizará la más rápida, por ejemplo el algoritmo ASAP (figura 3.4). Siguiendo con el ejemplo anterior, los macronodos serán los siguientes:

- Macronodo 1 (correspondiente al estado S0) -> {contador, registro}
- Macronodo 2 (correspondiente al estado S1) -> {comparador}
- Macronodo 3 (correspondiente al estado S2, S3, S4) -> {contador, comparador, registro, {sumador}}
- Macronodo 4 (correspondiente al estado S5) -> {biestable}

Una vez que tenemos todos los macronodos, debemos obtener el tiempo de ejecución de cada uno de ellos, que coincidirá con el retraso del operador más lento. Este retraso vendrá determinado por la ecuación (4.4), en la que figuran la actividad del operador (número de veces que se hace uso de él), el número de bloques que se disponen de dicho operador, el número de etapas de pipeline necesarias para implementar dicho operador (en el caso de que no se implemente en pipeline, este valor será 1) y el retraso de dicho operador.

$$\text{TE_macronodo(j)} = MAX_{operadores}([\text{usos_operador} \cdot numbloques + pipeline - 1]retraso) \qquad (4.4)$$

Suponiendo que hay un único bloque por cada operador, sin pipeline y todos con el mismo retraso (que al normalizarse tomará el valor 1); los tiempos de ejecución de cada macronodo serán los siguientes:

- TE_S0 = MAX [(1cont/1 +1 -1)·1, (1reg/1 +1 -1)·1]
- TE_S1 = (1comp/1 +1 -1)·1
- TE_S2 = MAX [(1cont/1 +1 -1)·1, (1reg/1 +1 -1)·1, (1comp/1 +1 -1)·1]
- TE_S3 = MAX [(1cont/1 +1 -1)·1, (1reg/1 +1 -1)·1, (1comp/1 +1 -1)·1]
- TE_S4 = MAX [(1cont/1 +1 -1)·1, (1reg/1 +1 -1)·1, (1comp/1 +1 -1)·1, (1comp/1 +1 -1)·1]
- TE_S5 = (1bies/1 +1 -1)·1

Por último debemos agrupar todos los cálculos anteriores. No obstante, vamos a considerar dos casos diferentes. Cuando no existen ramificaciones, el tiempo de ejecución total será la suma del correspondiente a todos los macronodos, es decir, todos tienen la misma probabilidad y frecuencia. Cuando existen ramificaciones, hay que asignarle una frecuencia y probabilidad a cada uno de los macronodos para poder realizar la suma ponderada. La frecuencia se puede obtener del flujo de datos, mientras que la frecuencia hay que suponerla (por ejemplo dando la misma probabilidad a cada rama). Lo normal es crear un diagrama de flujo de control donde se indican los macronodos (o estados), en el que se indican la frecuencia y probabilidad de cada uno.

Siguiendo el ejemplo anterior, en la figura 3.11 se muestra el grafo de flujo de control con las correspondientes frecuencias y probabilidades. En este grafo no se han considerado las realimentaciones al mismo estado ya que han sido sustituidas por su frecuencia. También cabe destacar que tenemos tres tipos diferentes de ramificaciones. La primera ramificación es la autoalimentación del macronodo E0; en esta ramificación no se puede determinar la frecuencia por los datos, por lo que se ha elegido una frecuencia igual a 1, ya que es el macronodo inicial. El segundo tipo de ramificación dirige el flujo de datos por tres caminos diferentes, por lo que la frecuencia elegida no se alterará, sino que la probabilidad de cada uno de ellos será 1/3 al no tener ningún argumento para que uno de los caminos se de un mayor número de veces que otro. Por último, el tercer tipo de ramificación es un bucle cuya ejecución depende de un contador; por lo tanto, este contador me está indicando la frecuencia de dicho macronodo. Así, la estimación del tiempo de ejecución será la suma ponderada (con frecuencia y probabilidad) de los tiempos de ejecución de cada uno de los macronodos.

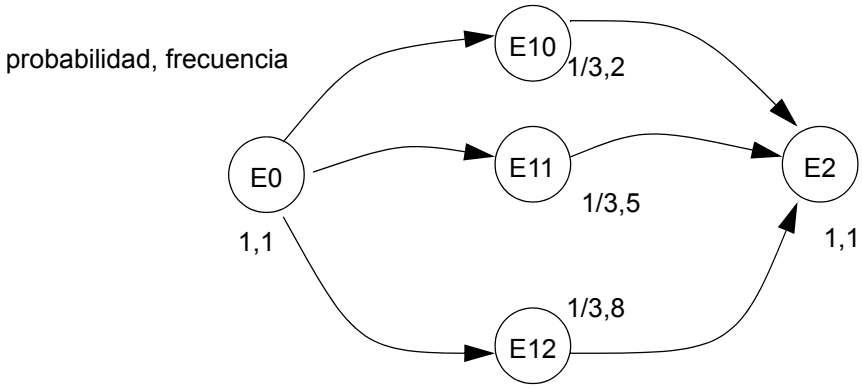


Figura 3.11.- Diagrama de flujo de control para la estimación del tiempo de ejecución.

3.3.2. Área ocupada

Al igual que sucedía en la estimación del tiempo de ejecución, vamos a considerar la arquitectura de la figura 3.10, y por lo tanto, el área total se dividirá en el camino de datos y la unidad de control, y eventualmente unidades de memoria necesarias para almacenar algún tipo de información inicial.

Así mismo, como sucedía en la estimación del tiempo de ejecución con el proceso de scheduling, en la estimación de área se debe simular un proceso de allocation para lo cual se suele utilizar una librería de celdas y una tecnología de partida. Este proceso de allocation nos va a dar una estimación de los elementos del camino de datos, así como los multiplexores de interconexión; tanto en número de ellos como en tamaño (considerando el número de entradas necesarias).

Con respecto a la unidad de control, se utilizarán estimadores que implementarán la unidad como una memoria ROM y biestables. Luego, el parámetro relevante de la unidad de control será el número de estados y de entradas.

Por último, con respecto a las unidades de memoria, se utilizarán tantas unidades como especificaciones se deban almacenar, como pueden ser datos de entrada.

En el caso del ejemplo de la figura 3.3, se necesitarían las siguientes unidades:

- Unidades funcionales del camino de datos:
  - Sumador para el estado S4
  - Contador para los estados S0, S2, S3 y S4
  - Comparador para los estados S1, S2, S3 y S4
- Unidades de almacenamiento:
  - Registro para almacenar el resultado de la operación (result) para los estados S0, S2, S3 y S4

- Multiplexores de interconexión:
  - Multiplexor 4:1 de entrada al registro de almacenamiento (ya que puede haber tres entradas diferentes, correspondientes a los estados S2, S3 y S4)
  - Multiplexor 4:1 de entrada al comparador (ya que puede haber cuatro entradas diferentes correspondientes a los estados S1, S2, S3 y S4)
  - Para el contador no hace falta multiplexores porque siempre tiene que hacer la misma operación comenzando desde el mismo valor, desde cualquiera de los estados.
- Unidad de control:
  - Memoria ROM de 4+5 señales de entrada. Cuatro señales de entrada (inicio y las tres salidas del comparador y cinco estados.
  - Cinco biestables, para cinco estados con la codificación *one-hot*.

3.3.3. Consumo de potencia

Antes de entrar en materia vamos a repasar lo que se entiende por consumo de potencia y su importancia en los sistemas empotrados.

> El consumo de potencia se puede definir como la velocidad a la que la energía se transforma de una forma en otra.

Más concretamente, las formas de energía que suelen intervenir en los sistemas electrónicos son la energía eléctrica y el calor. Esta transformación fue el primer motivo para aumentar la prioridad del consumo de potencia como parámetro restrictivo, ya que el sistema sufre un aumento de coste debido a la necesidad de refrigeración, que puede ser a través de determinados encapsulados (como los cerámicos) o de distribuciones especiales de las líneas de alimentación en el interior del circuito. No obstante, e íntimamente relacionado con la transformación de energía, existe una reducción considerable del tiempo de vida de las baterías que es uno de los parámetros fundamentales en los sistemas móviles. También relacionado con la tensión de alimentación, y por tanto, con el consumo de potencia, existe la influencia del ruido digital (de conmutación), el cual es producido ‘por la conmutación de las señales digitales que provocan una no idealidad en la fuente de alimentación introduciendo fluctuaciones (que pueden disminuir las prestaciones del sistemas, e incluso a un mal funcionamiento del mismo).

El consumo de potencia de un circuito se puede expresar según la ecuación (4.5). En dicha ecuación podemos distinguir las cuatro contribuciones principales al consumo de potencia: el primer sumando se corresponde con el consumo de potencia dinámica debida a la conmutación de los transistores que carga y descarga las capacidades; el segundo sumando se corresponde con el consumo de potencia debido al cortocircuito existente entre polarización y tierra durante la conmutación de los transistores; el tercer sumando se corresponde con el consumo debido a las corrientes de fuga a través del sustrato; y el cuarto sumando se corresponde con el consumo de potencia estática típica de algunas familias lógicas como la NMOS. De estas cuatro contribuciones, la componente más importante es la primera (excepto en las familias que permiten un camino resistivo entre polarización y tierra, en cuyo caso pasa a ser la cuarta).

$$P = C_L V_{DD}^2 f + (t_r + t_f) V_{DD} I_P f + V_{DD} I_{fugas} + V_{DD} I_{\text{estática}} \qquad (4.5)$$

Por lo tanto, la estimación del consumo de potencia se reduce a estimar los parámetros del primer sumando: capacidad de carga, frecuencia y tensión de polarización. El último parámetro suele estar fijado por las especificaciones de diseño, así que únicamente hay que estimar la capacidad de carga y la frecuencia de conmutación de cada nodo, que se puede sustituir por la frecuencia global del sistema por la actividad de conmutación del sistema. Este hecho se va a cuantificar en el nivel de arquitectura, por lo tanto hay que seguir tres pasos:

- Realizar una simulación RTL (de la estimación de área) para obtener la actividad de cada unidad funcional.
- Elegir un modelo de energía, que puede ser analítico o sensibles a transiciones de los diferentes elementos del sistema.
- Cuantificar este modelo con la actividad obtenida en el primer punto y los parámetros tecnológicos utilizados.

No obstante, para las tecnologías más actuales, las diferencias entre las diferentes contribuciones de la ecuación (4.5). Luego, al despreciar alguno de los sumandos estamos aumentando el error en la estimación; además del error de la estimación de la contribución más influyente.