UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMATICA

Departamento de Arquitectura de Computadores y Automática

SÍNTESIS DE ALTO NIVEL GUIADA POR GESTIÓN DE PATRONES Y DESCOPOSICIÓN DE OPERACIONES

MEMORIA PARA OPTAR AL GRADO DE DOCTOR
PRESENTADA POR

Pedro Luis García Repetto

Bajo la dirección de los doctores

Mª del Carmen Molina Prego
Rafael Sautua Ruiz

MADRID, 2013

SÍNTESIS DE ALTO NIVEL GUIADA POR GESTIÓN DE PATRONES Y DESCOMPOSICIÓN DE OPERACIONES

Pedro Luis García Repetto

Tesis Doctoral

Dpto. de Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Síntesis de alto nivel guiada por gestión de patrones y descomposición de operaciones

Memoria presentada por D. Pedro Luis García Repetto para optar al grado de doctor por la Universidad Complutense de Madrid, realizada bajo la dirección de Dª. Mª del Carmen Molina Prego y D. Rafael Sautua Ruiz.

Madrid, 4 de enero de 2013

AGRADECIMIENTOS

En primer lugar deseo agradecer la labor de dirección de este trabajo por parte de los profesores María del Carmen Molina y Rafael Sautua, geniales transmisores del arte de la investigación. Ella y él con su paciencia, ánimo y buen hacer han mantenido mi interés y mi dedicación a esta ingente tarea en que se ha convertido esta bonita historia de investigación que hemos llevado a cabo durante los últimos años. Historia que ha sido mi entretenimiento en las horas libres que me dejaban mis responsabilidades en el Ministerio de Hacienda como auditor informático, en la Facultad de Informática como profesor asociado y en mi familia, a la que he sustraído muchas horas de nuestro tiempo común.

Quisiera además mostrar mi agradecimiento a todos los profesores y miembros del personal de administración y servicios de la Facultad de Informática de la Universidad Complutense de Madrid que, de un modo u otro, me han ayudado durante todo este proceso de creación investigadora.

También quiero manifestar mi agradecimiento a todos aquellos profesionales, tanto del mundo académico, de las administraciones públicas y del mundo de la empresa privada, que me han ofrecido su apoyo y sugerencias para encaminar esta tesis doctoral.

Finalmente, a todos aquellos que habéis estado conmigo desde siempre, con vuestra amistad.

A mi madre en el recuerdo y a mi padre en el recuerdo de su silencio

A Carmen y Jozsef por su paciencia y comprensión

CONTENIDO

FIGURAS

TABLAS

ALGORITMOS

CAPÍTULO

1

SÍNTESIS DE ALTO NIVEL: ORIGEN Y EVOLUCIÓN

En 1965 Gordon E. Moore, cofundador de Intel, vaticinó en un artículo publicado en la revista “Electronics Magazine” que cada 18 meses se duplicaría el número de transistores que podría admitir un circuito integrado. Más tarde, en 1975, redujo esa perspectiva afirmando que la complejidad de los circuitos integrados se duplicaría cada 24 meses.

Años más tarde Gordon Moore y Stephen Hawking debatieron conjuntamente sobre las limitaciones de la ley de Moore. Según Hawking esta ley estaría limitada por los propios límites de la microelectrónica. Estos límites son fundamentalmente la naturaleza atómica de la materia y la velocidad de la luz.

Debido a los límites anteriores, el propio creador de la ley ha afirmado que en 10 ó 15 años a partir de 2007 su ley dejaría de cumplirse. No obstante y hasta el momento, los fabricantes de circuitos integrados continúan dando la razón a Gordon E. Moore presentando nuevos circuitos que validan la actualidad de la afirmación que hizo en 1965.

En principio, la Ley de Moore se refiere a la evolución de la integración de transistores en una sola capa. Si se superponen varias capas de transistores se podría crecer en tres dimensiones en cuanto a número de transistores a integrar. En este modelo tridimensional aparecerían otros problemas a resolver como son el aumento en el consumo eléctrico o la disipación de calor. Problemas que ya son importantes en una capa de dos dimensiones pasarían a ser acuciantes en este modelo de tres dimensiones.

Siguiendo con la evolución y el aumento en el nivel de integración, recientemente Intel ha presentado un prototipo de futuro chip denominado "Single-Chip Cloud Computer" de 48 núcleos, cuya tecnología podría hacer escalar su número hasta los 100 núcleos por procesador.

Parece lógico pensar que se alcanzará un límite físico en los materiales utilizados hoy en día que hará que se cumplan las predicciones sobre el límite mencionado anteriormente por el propio creador de la ley de Moore. Mientras tanto, ya no se habla de circuitos integrados con tecnología VLSI[1], sino que se ha pasado a manejar los conceptos de ULSI[2] y GLSI[3].

---

[1] VLSI: Very Large Scale Integration. Entre 10.000 y 100.000 transistores.

[2] ULSI: Ultra Large Scale Integration. Entre 100.000 y 1.000.000 transistores.

[3] GLSI: Giga Large Scale Integration. Más de 1.000.000 de transistores.

1.1 Automatización del diseño de circuitos

Para aprovechar esos aumentos exponenciales en las capacidades de integración por un lado, y de proceso por otro, ha sido necesario el desarrollo en paralelo de técnicas de diseño de circuitos integrados.

Hasta la generación anterior a la VLSI, es decir la LSI[4], se utilizaban técnicas de diseño basadas en prototipos, con las que podían diseñarse módulos sencillos. Tras verificar el correcto funcionamiento de estos módulos de forma individual se integraban unos con otros. Estas técnicas, capaces de implementar módulos complejos a partir de diseños sencillos, se denominaban técnicas de diseño "bottom-up" (de abajo a arriba).

Cuando el número de transistores comenzó a ser inmanejable para las técnicas de diseño tradicionales, comenzaron a aparecer los primeros lenguajes de descripción hardware orientados a la síntesis. Esta síntesis era a nivel RT o de transferencia entre registros. La síntesis RT, o síntesis lógica, tenía entre otros objetivos automatizar las tareas de diseño que requerían más tiempo, con el fin de obtener un circuito descrito a nivel de puertas lógicas.

En la generación de integración VLSI, el índice de integración es tan elevado que las técnicas anteriores dejaron de ser válidas. Es entonces cuando surgen nuevas técnicas y herramientas gráficas de diseño. Estas nuevas herramientas obtienen un esquema estructural a partir de una especificación conductual del circuito y de una serie de restricciones. Estas técnicas aparecieron a finales de la década de los 80 y constituyen las primeras aproximaciones al proceso de síntesis de alto nivel[5]. Estas primeras aproximaciones generan implementaciones muy influenciadas por el propio algoritmo de síntesis que incorporan las herramientas. Esta influencia provoca que los diseñadores estén obligados a tener en cuenta la naturaleza de dicho algoritmo de síntesis al describir las especificaciones conductuales.

[4] LSI: Large Scale Integration. Entre 1.000 y 10.000 transistores.
[5] Síntesis de alto nivel: SAN en adelante.

Aún hoy en día, las herramientas comerciales y la mayoría de los algoritmos de SAN producen diseños muy influenciados por el estilo descriptivo usado en la especificación de partida. En general, el número, tipo, representación y anchura de los recursos funcionales, de almacenamiento y de encaminamiento de datos de un diseño está estrechamente relacionado con el número, tipo, representación y anchura de las operaciones y de los datos de la especificación conductual.

En los últimos años, se han propuesto varias alternativas en la línea de desvincular el circuito resultante de la SAN del estilo descriptivo usado por el diseñador en la especificación de la conducta, con el objetivo de alcanzar diseños de mayor calidad que los sintetizados con los algoritmos convencionales. Por un lado, estas propuestas han consistido en nuevos algoritmos de SAN capaces de obtener diseños más eficientes en términos de área, tiempo de ejecución o consumo de potencia. Y por otro lado, se han presentado algoritmos de optimización de especificaciones conductuales que transforman la especificación de partida en otra con más posibilidades de alcanzar circuitos de mayor calidad. En ambos casos, se aplica un conjunto de transformaciones sobre las operaciones de la especificación conductual que conlleva la descomposición de algunas de las operaciones más complejas en varias operaciones más sencillas con ciertas dependencias de datos entre ellas.

Con el objetivo de contribuir a la independencia de las implementaciones generadas del estilo descriptivo utilizado en la especificación, en la presente tesis se propone una nueva metodología de diseño basada en la combinación de las técnicas de descomposición de operaciones e identificación y gestión de patrones. Esta metodología permite reducir el área en unidades funcionales, de almacenamiento y de interconexionado en los circuitos resultantes de la SAN, disminuyendo significativamente el área final de los diseños.

La metodología propuesta se implementa en un algoritmo de SAN que mediante refinamientos sucesivos descubre los patrones operacionales más

comunes en la especificación y decide la ejecución de las operaciones involucradas en el patrón en el mismo hardware. De esta forma y ejecutando todas las operaciones de un patrón en un mismo ciclo de forma encadenada, además del ahorro en unidades funcionales y de almacenamiento, se evitan transferencias de datos innecesarias entre los distintos componentes de la ruta de datos. Esto redunda no sólo en un ahorro en el área ocupada por el cableado de interconexiones, sino también en una reducción en el número de otros elementos de enrutado tales como por ejemplo los multiplexores.

La técnica de síntesis propuesta parte de una restricción de tiempo impuesta por el diseñador, a través de la definición de la duración del ciclo de reloj o la latencia del circuito (número de ciclos). Por ello, el objetivo principal es optimizar y reducir el área del esquema estructural resultante del proceso de síntesis.

A lo largo de este primer capítulo se realiza un breve recorrido por la historia de la SAN y se presentan los aspectos más importantes de dicha materia. En el apartado 1.2 se describen los pilares básicos o principales conceptos que dan forma a la teoría de la SAN. En el apartado 1.3 se presentan los principales hitos que han marcado la evolución de la SAN mediante una división de su historia en generaciones. Se parte de una generación cero o protohistoria en la que comienzan a pincelarse las primeras ideas de una nueva etapa en el diseño de circuitos integrados. La aparición de esta nueva etapa vino forzada, tal como se ha explicado anteriormente, por el imparable aumento en el índice de integración de los circuitos. Dicha historia continúa hacia el futuro con las tendencias que en el mundo de la SAN se vislumbran hoy en día. Por último, en el apartado 1.4, se esbozan los principales objetivos perseguidos en este trabajo de investigación. Así mismo, se presenta la estructura y contenido del resto de la memoria.

1.2 Principales conceptos de SAN

La SAN es aquel proceso mediante el cual a partir de una descripción conductual se genera una descripción estructural. Para que esta definición tenga sentido es necesario definir previamente los siguientes conceptos:

1) *Proceso: según el diccionario de la lengua española de la Real Academia Española[6], proceso se define en su tercera acepción como el "Conjunto de las fases sucesivas de un fenómeno natural o de una operación artificial".*

2) *Descripción conductual: descripción algorítmica de un sistema digital.*

3) *Descripción estructural: es el conjunto formado por una ruta de datos, descrita por un conjunto de módulos o circuitos y las conexiones que los relacionan, y un controlador que dirige las acciones de la ruta de datos según interacciones recibidas del exterior o de la propia ruta de datos.*

Tras estas definiciones se puede volver a definir la SAN de una forma más detallada: la SAN es la secuencia de fases sucesivas de una operación artificial que genera una ruta de datos y un controlador a partir de una descripción algorítmica de un sistema digital.

La ruta de datos generada tras el proceso de SAN, está formada por los siguientes elementos:

1) *Unidades funcionales encargadas de ejecutar las operaciones.*

2) *Unidades de almacenamiento receptoras de los resultados de las unidades funcionales y fuentes de los operandos de entrada de las unidades funcionales.*

3) *Unidades de interconexionado encargadas de establecer las uniones entre las unidades funcionales, las unidades de almacenamiento y el controlador.*

[6] Real Academia Española: www.rae.es.

Según la definición detallada anterior, la SAN parte de un objeto en el dominio conductual y obtiene un resultado en el dominio estructural. Según la definición de proceso, la SAN está formada por una serie de fases intermedias. Cada una de estas fases se puede considerar a su vez como un proceso en si mismo o subproceso del proceso general de SAN. Cada uno de estos subprocesos tendrá un objeto de entrada y generará un resultado. Más adelante se detallarán cada uno de estos subprocesos.

Continuando con el desarrollo genérico de generación de un circuito y como proceso posterior a la SAN, se tiene la síntesis lógica o RT. Esta síntesis toma como entrada el producto generado por la SAN en el dominio estructural y obtiene un sistema en el dominio físico, caracterizado por el número, ubicación y relación de las puertas lógicas que lo forman.

Los distintos procesos de SAN generan circuitos cuya calidad está en relación a tres dimensiones: el área del circuito generado, el tiempo de ejecución y el consumo de energía. Los algoritmos de SAN exploran los espacios de diseño y tratan de optimizar alguna de las dimensiones anteriores.

Tradicionalmente se trabajaba sólo con las dos primeras dimensiones: el área y el tiempo de ejecución. En el nivel RT el área viene determinada por el número de puertas lógicas necesarias para implementar el circuito sintetizado. En este espacio bidimensional de trabajo se diseñan dos grandes tipos de algoritmos de SAN:

1) *Algoritmos de SAN con restricción de área. El diseñador fija un área máxima a utilizar y el algoritmo trata de encontrar la mejor solución en el nivel estructural optimizando el tiempo de ejecución.*

2) *Algoritmos de SAN con restricción de tiempo. El diseñador fija un tiempo máximo de ejecución y el algoritmo trata de encontrar la mejor solución en el nivel estructural optimizando el área necesaria.*

El algoritmo de síntesis propuesto en la presente memoria se engloba en los algoritmos de SAN con restricción de tiempo. Por ello, el objetivo principal del algoritmo será minimizar u optimizar el área necesaria para implementar las

descripciones conductuales partiendo de una restricción de tiempo definida por el diseñador.

1.2.1 Fases del proceso de SAN

El proceso de SAN se divide en cuatro fases o subprocesos: compilación, planificación de operaciones, selección y asignación de recursos físicos y síntesis del controlador. Algunos algoritmos de SAN dividen la tercera fase en dos fases distintas; una fase de selección de recursos físicos y otra de asignación de los recursos físicos a cada operación o dato.

Las fases o subprocesos de la SAN no tienen por qué seguir el orden indicado en el párrafo anterior. A lo largo de la historia de la SAN ha habido múltiples propuestas en la secuencia o paralelización de dichos subprocesos. Algunas propuestas de SAN incluyen una primera fase de optimización de la descripción conductual con el fin de obtener resultados de mayor calidad. Las transformaciones propuestas son de carácter general, por lo que los resultados obtenidos dependen del algoritmo de SAN utilizado para sintetizar las conductas. La especialización de las transformaciones en función del tipo de algoritmo de SAN utilizado produce en la mayoría de los casos mejores resultados. Sin embargo, esta especialización requiere continuas renovaciones de los algoritmos de optimización para adaptarlos a las nuevas tendencias de la SAN. Por este motivo, la implementación de nuevos algoritmos de SAN que incorporan algunas de estas optimizaciones durante el proceso de diseño constituye una alternativa viable a la fase de optimización de especificaciones conductuales. La técnica propuesta en la presente memoria aplica la descomposición de operaciones en la primera fase del proceso de síntesis y en fases sucesivas con el fin de optimizar el área de los diseños sintetizados.

Otras propuestas incluyen una nueva fase posterior a la de compilación, consistente en realizar una partición o división de la representación intermedia con el fin de reducir el tamaño del problema. Esta fase adicional no se incorpora en la técnica propuesta ya que la síntesis de la especificación completa permite identificar un mayor número de operaciones y fragmentos

de operación que pueden compartir los mismos recursos funcionales, de almacenamiento y de encaminamiento, lo que finalmente redunda en una reutilización mayor de los recursos de las rutas de datos. A continuación se describen las fases más importantes de la SAN.

1.2.1.1 Compilación

El subproceso de compilación tiene como entrada una descripción conductual del sistema a sintetizar. Esta descripción suele expresarse mediante un lenguaje de alto nivel. Tal como se verá en el análisis posterior de la historia de la SAN, hoy en día predomina el uso del lenguaje C o similares como lenguaje de descripción de conductas. El proceso de compilación genera como salida una representación interna fácilmente procesable en la siguientes fases de la síntesis.

Es habitual el uso de grafos de flujo de datos (GFD) para representar las dependencias de datos entre las operaciones. También se utilizan diagramas de flujo de control (GFC) para plasmar la secuencia de las operaciones, los saltos condicionales y los bucles de la descripción conductual. Una tercera alternativa es el uso de diagramas de flujo de datos y control (GFDC). Estos últimos representan el flujo de control de los GFC y el flujo de datos de los GFD.

En la fase de compilación algunas herramientas o técnicas aplican una serie de optimizaciones o mejoras a la descripción conductual del sistema a sintetizar. Estas optimizaciones han sido heredadas normalmente de la teoría de los compiladores. Entre las optimizaciones más clásicas se encuentran el desenrollado de bucles [AhSU86], la eliminación de variables temporales, la localización de sub-expresiones comunes y código muerto o la propagación de constantes.

Otras técnicas posteriores de mejora u optimización propias de la SAN son la segmentación [PaPa88], explotación de la exclusión mutua [GDGN03] [KoWo99] [PeMH02] o la retemporización [PoRA94].

1.2.1.2 Planificación

La principal función de este subproceso es determinar en qué ciclo de reloj comienza la ejecución de cada una de las operaciones que forman parte de la descripción conductual. Habitualmente el controlador proporciona a la ruta de datos las señales de control correspondientes en cada ciclo de reloj. Sin embargo, es posible encontrar diseños en los que las señales de control son las mismas durante varios ciclos de reloj. A ese conjunto de ciclos se les denomina paso de control. En el trabajo presentado en esta tesis se ha identificado siempre un paso de control con un único ciclo de reloj, por lo que en la presente memoria se utilizarán ambos términos indistintamente.

En la fase de planificación se plantean dos tipos de problemas a resolver según el parámetro de calidad definido. Estos problemas son:

1) *Planificación con restricción de tiempo.* Una vez fijada la restricción de tiempo por parte del diseñador, expresada en un número máximo de ciclos de reloj, se trata de minimizar el área de los circuitos generados al asignar las operaciones a ciclos de reloj.

2) *Planificación con restricción de área.* El proceso de síntesis tiene a su disposición un conjunto limitado de unidades funcionales, de almacenamiento y de interconexionado, o de recursos físicos en general, y trata de minimizar el número de ciclos o pasos de control necesarios para la ejecución de la conducta especificada.

Para resolver ambos problemas se han propuesto a lo largo de la historia de la SAN múltiples soluciones; unas basadas en métodos exactos y otras basadas en métodos heurísticos.

Si la solución aplicada está basada en métodos exactos, se realiza una búsqueda exhaustiva en el espacio de diseño que provoca un crecimiento exponencial en el tiempo de diseño al aumentar el número de operaciones y variables de la especificación. Por ello, en la práctica este tipo de soluciones exactas sólo son viables en el caso de especificaciones conductuales sencillas.

Estos métodos suelen basarse en técnicas de programación lineal entera [GeEl93] [KuSu01] [LaMD94].

Al no ser válidos los métodos exactos para especificaciones con un elevado número de operaciones, se propusieron métodos de planificación basados en algoritmos heurísticos. Estos algoritmos no exploran el espacio de diseño completo, sino que orientan la búsqueda explorando únicamente las soluciones incompletas más prometedoras, por lo que consiguen tiempos de ejecución asumibles a costa de no alcanzar el diseño óptimo.

Los algoritmos de planificación heurísticos se clasifican en tres grandes grupos según el método utilizado para seleccionar la operación que se planifica en cada momento:

1) *Algoritmos transformacionales.* Estos algoritmos tienen como entrada el circuito planificado por otros algoritmos de planificación. A dicha planificación inicial se aplican diversos métodos de optimización que planifican de nuevo algunas de las operaciones [CoCL01] [PaKy91] [SeKP02] [WTLS03] [ZhGa99].

2) *Algoritmos globales.* Para asignar una operación a un ciclo de reloj tienen en cuenta todas las operaciones de la especificación y todos los ciclos de reloj disponibles [FaRS94] [GuKa02] [SHEE00]. El algoritmo de planificación global más conocido y referenciado en la historia de la SAN es el algoritmo de planificación dirigido por fuerzas [PaKn89]. EL objetivo de este algoritmo de planificación es reducir la cantidad de recursos físicos necesarios para ejecutar las operaciones de la especificación, una vez fijado un número máximo de pasos de control disponibles. El algoritmo pretende alcanzar una distribución homogénea de las operaciones entre todos los ciclos utilizando el concepto de *fuerza*. Este concepto representa la bondad de cada posible asignación de una operación a cada paso de control.

3) *Algoritmos constructivos.* Estos algoritmos no trabajan de manera global con todas las operaciones y pasos de control sino que asignan cada

operación a un paso de control siguiendo un orden preestablecido. Este orden limita las operaciones y los pasos de control a tener en cuenta en cada momento. Los algoritmos constructivos más sencillos y más conocidos son el ASAP (As Soon As Possible) y el ALAP (As Late As Possible). La planificación ASAP asigna cada operación al primer ciclo posible una vez que sus dependencias de datos ya han sido satisfechas. Por el contrario, la planificación ALAP asigna la operación al último ciclo en que puede ejecutarse. Estos dos tipos de planificaciones no tienen en cuenta la prioridad de las operaciones y por lo tanto las operaciones del camino crítico se pueden retrasar. Otros tipos de algoritmos constructivos son los basados en listas [KuSu01] [NeTh86] [NiMa03] [SlDr02]. Estos algoritmos utilizan criterios globales para asignar las operaciones que cumplen ciertas restricciones a ciclos. A diferencia de los anteriores, no trabajan con cada operación de forma individual sino que manejan listas de operaciones candidatas a ser planificadas (tienen resueltas sus dependencias de datos). Las operaciones se encuentran ordenadas en la lista según su prioridad. Una propuesta de esta naturaleza fue hecha por [PaGa87] introduciendo el concepto de movilidad de cada operación. La movilidad se definió como la diferencia entre el paso de control asignado por una planificación ALAP y el paso de control asignado por una planificación ASAP. Otras propuestas tienen en cuenta otro tipo de condicionantes como son las operaciones predecesoras y sucesoras de cada operación, el árbol al que pertenece cada operación o el número total de sucesoras que tiene una operación [SlDr02].

1.2.1.3 Selección y asignación de recursos físicos

La selección de recursos físicos consiste en determinar qué unidades funcionales, de almacenamiento y de encaminamiento de datos van a ser necesarias para la ejecución de las operaciones. Todos los recursos mencionados se seleccionan de una biblioteca de diseño. En la asignación se asocia a cada operación un recurso físico concreto de los seleccionados, a cada variable un recurso de almacenamiento (registro o memoria) y a cada conexión un recurso de interconexión (multiplexor o bus).

Algunos algoritmos de selección y asignación de especificaciones simples están basados en programación lineal entera [GeEl93] [KuSu01] [LaMD94] [RMJD94]. En la síntesis de especificaciones complejas se han propuesto algoritmos de selección y asignación heurísticos. Estos últimos se pueden clasificar en alguno de los siguientes grupos:

1) *Algoritmos basados en descomposiciones.* Estos algoritmos dividen el proceso de selección y asignación en varias fases. Muchos de los algoritmos de descomposición utilizan técnicas de grafos. Para cada tipo de recurso físico se ejecuta un proceso de SAN distinto. Los algoritmos de esta naturaleza más referenciados han sido los que aplican métodos de particionamiento en cliques [CoCL01] [KuSu01] [NiMa03] [TsSi86] y los basados en el algoritmo del lado izquierdo [ChCF03] [KuPa87].

2) *Algoritmos transformacionales.* Al igual que los algoritmos de planificación del mismo nombre, estos parten de una selección y asignación ya realizada por otros algoritmos y tratan de optimizarla mediante la reasignación de recursos [SeKP02] [SHSS03] [TsHs90].

1.2.1.4 Síntesis del controlador

Tras terminar la síntesis de la ruta de datos se lleva a cabo la síntesis del controlador generándose un autómata finito. Este autómata se encarga de generar las señales de control necesarias para que en cada uno de los pasos de control funcionen correctamente todos los componentes de la ruta de datos.

1.2.2 Secuencia de ejecución de las fases de la SAN

La primera fase del proceso de SAN es la fase de compilación, y la última es la fase de síntesis del controlador. Las otras fases: planificación, selección y asignación son interdependientes ya que las decisiones que se toman en cada una de ellas influyen en las demás, restringiendo el número de diseños alcanzables. Por ello, y según el orden que se siga en la ejecución de estas fases, se obtendrán implementaciones de distinta calidad. En esta línea, las

propuestas han sido muy diversas, desde ejecutar las distintas fases de forma secuencial en un determinado orden, hasta ejecutarlas en paralelo según combinaciones diversas. Algunas propuestas han tenido en cuenta la influencia de algunas decisiones de diseño en el resto del proceso de SAN dando lugar a implementaciones de mayor calidad [CLou90] [LaMD94] [MoHF96] [NiMa03] [SaZa90].

1.3 Historia de la SAN

Se puede considerar que el concepto de SAN comienza a modelarse a principios de la década de los setenta con los trabajos de investigadores como Mario Barbacci. Estas primeras investigaciones se pueden considerar como la etapa prehistórica de la SAN o generación cero y abarca la década de los setenta.

La primera generación se desarrolló durante la década de los ochenta y primeros años de la década de los noventa. En ella se investigan y definen los conceptos que constituyen los pilares básicos sobre los que se asientan las futuras líneas de investigación en esta materia.

La segunda generación abarca desde mediados de la década de los noventa hasta principios de la década del 2000. Esta generación se caracterizó por la aparición de las primeras herramientas comerciales de SAN, surgidas a parir de los resultados obtenidos en las investigaciones de la generación anterior.

La tercera generación comprende desde principios de la década del 2000 hasta la fecha actual. En esta generación las herramientas de síntesis han tendido a generalizar la adopción del lenguaje C, o adaptaciones del mismo, como lenguaje de descripción de conductas en las herramientas de síntesis.

La cuarta generación está surgiendo de la actual tercera y en ella aparecerán herramientas enfocadas a potenciar las dos dimensiones principales de la SAN: el dominio de control y el dominio de la ruta de datos.

El desarrollo y la evolución de la SAN han sido siempre muy seguidos por las casas comerciales debido a que facilita el diseño de nuevos ASIC[7], simplificando su proceso de generación y ahorrando costes en todas las fases del diseño y la fabricación. Según aumentaba la capacidad de los circuitos integrados cumpliendo continuamente las predicciones de la ley de Moore, los costes del diseño también aumentaban. Debido a ello, se hizo necesario que los diseñadores de circuitos integrados fueran ascendiendo en el nivel de abstracción de sus diseños para poder aprovechar el aumento en la capacidad de los circuitos integrados.

En todas las generaciones se ha perseguido un mayor nivel de abstracción en el diseño, captura e implementación de los circuitos. Esto ha contribuido a una evolución constante en las herramientas de diseño en el mundo de la SAN.

A continuación se detallan las principales características de cada una de las generaciones.

1.3.1 Generación cero

En los años setenta, el grupo de investigación de la Universidad Carnegie Mellon liderado por Mario Barbacci consideraba que, en teoría, se podría traducir el juego de instrucciones de un procesador en hardware. Esto es, era factible pasar de descripciones conductuales de alto nivel a diseños hardware, siendo las descripciones conductuales superiores en nivel de abstracción a las descripciones estructurales.

En estas primeras investigaciones, los puntos de interés se situaban en las especificaciones del diseño y en la síntesis en los dos niveles: RT y algorítmico. Es decir, comienza a deslindarse el futuro campo de la síntesis algorítmica y de la síntesis a nivel RT. Como ejemplo de ello se puede indicar la publicación de A. Parker et al [Park79] sobre el diseño automatizado de la ruta datos.

[7] ASIC: application-specific integrated circuit.

Se puede considerar que la generación cero constituyó un punto de ruptura en las técnicas de diseño de circuitos integrados. Los investigadores de esta época atisbaron con claridad la necesidad de ascender a nuevos niveles de abstracción, con el fin de afrontar los nuevos retos en el crecimiento de la capacidad de los circuitos integrados.

1.3.2 Generación uno

Durante esta generación se realizó un gran esfuerzo en todos los frentes de la SAN. Se presentaron trabajos cuyos contenidos pasaron a ser pilares fundamentales en la teoría de la síntesis. Entre estas investigaciones cabe reseñar las siguientes:

1) *Algoritmo de planificación por fuerzas propuesto por el grupo de investigación liderado por P.G. Paulin, and J.P. Knight [PaKn89].*

2) *Trabajos de investigación de D. Gajski [Gajs92], G. De. Micheli [DeMi94] y R. Camposano y W. Wolf [CaWo91] que dieron lugar a varias publicaciones sobre teoría de la SAN.*

3) *Definición de teorías de planificación básicas como ASAP y ALAP.* A partir de ellas se ampliaron las alternativas de planificación con otras métricas como las basadas en la urgencia de la ejecución de las operaciones o la movilidad de las operaciones.

4) *Representación y gestión del paralelismo en la representación simbólica.* En principio se explotó la representación de las relaciones basadas en la dependencia de datos utilizando los GFD. Más adelante se añadió a estos GFD el concepto de flujo de control desarrollándose los GFDC.

Las primeras herramientas comerciales de SAN salieron a la luz a finales de la década de los ochenta y principios de los noventa. No tuvieron mucho éxito debido a varios factores entre los que cabe destacar:

1) Durante su desarrollo todavía se estaba asumiendo la implantación comercial y el uso productivo de las herramientas de síntesis RTL. Esto provocó que los diseñadores no pudieran afrontar el nuevo reto de ascender en el nivel de abstracción del diseño de circuitos integrados con el fin de usar las nuevas herramientas de SAN, cuando realmente estaban todavía asimilando las herramientas de síntesis lógica.

2) Los lenguajes que incorporaban las nuevas herramientas de SAN resultaban poco intuitivos y ofrecían unas capacidades de descripción de conductas muy limitadas, lo que provocó un rechazo inicial. Así por ejemplo, los sucesivos proyectos de la familia Cathedral I y II, dirigidos por Hugo de Man y otros, utilizaron el lenguaje Silage. Como esta familia de proyectos estaba dirigida al diseño concreto de sistemas de procesamiento de señales, era de difícil aplicación a otros entornos del diseño de ASIC.

Tal como se verá más adelante, esta situación provocó que en la siguiente generación las herramientas de SAN adoptaran lenguajes más familiares y de uso general. En cualquier caso, los resultados generados por estas herramientas no gozaban de gran calidad.

Inicialmente, las herramientas de SAN se especializaron en obtener diseños relacionados con el procesamiento digital de señales, por lo que resultaban inapropiadas para la gran mayoría de los ASIC, que requerían un mayor énfasis en el flujo de control que en el procesamiento de datos y señales.

1.3.3 Generación dos

Tal como se comentó anteriormente, a principios de la década de los noventa se comienzan a publicar como compendios los vastos resultados de las investigaciones realizadas en el campo de la SAN durante la década de los ochenta. Muchas de estas publicaciones continúan siendo trabajo de referencia en la actualidad.

Durante esta generación emergieron herramientas comerciales de SAN que comenzaron a tener cierto éxito y a ser adoptadas por los diseñadores de circuitos integrados. Entre ellas destacan las herramientas de SAN de las compañías Synopsys, Cadence y Mentor Graphics. Esta generación de herramientas comerciales atrajo el interés de los diseñadores aunque supuso un fracaso comercial y no resultaron útiles a los diseñadores por su ineficiencia. El manejo de las mismas resultó poco intuitivo, los resultados eran difíciles de validar y los tiempos de simulación eran tan largos como los de la síntesis RTL.

Estas herramientas adoptaron lenguajes de descripción hardware[8], tales como Verilog o VHDL, por lo que entraron en conflicto con otras herramientas de síntesis lógica. Esta situación provocó confusión en los diseñadores al no ver claramente definida la frontera entre la SAN y la síntesis lógica.

Sin embargo, el gran problema de estas herramientas eran los resultados tan pobres que se obtenían al utilizarse para sintetizar especificaciones dominadas por el flujo de control. Únicamente producían resultados aceptables en el caso de circuitos dominados por el procesamiento de datos o señales.

1.3.4 Generación tres

Durante la década del 2000 comenzaron a aparecer modelos que se basaban en la agrupación de modelos estructurales y componentes. Estos modelos se sintetizaban de forma independiente y permitían su reutilización entre diferentes soluciones. Se definieron distintos niveles de abstracción teniendo como referencia la influencia del concepto temporal en cada uno de ellos. De mayor a menor abstracción se trabajan con los siguientes modelos de sistemas:

1) *Nivel 1. Es el nivel más alto de abstracción, en el que no se maneja el concepto de tiempo y donde lo que se define es un modelo funcional del sistema.*

[8] HDL: Hardware Description Language.

2) *Nivel 2. Este nivel está formado por el modelo de la arquitectura de las componentes en el que se maneja el tiempo pero sólo de manera funcional.*

3) *Nivel 3. Se comienzan a gestionar las comunicaciones entre los elementos de forma aproximada.*

4) *Nivel 4. Se incluye el concepto de ciclo de reloj que marca la pauta de comunicación entre los distintos elementos del sistema.*

5) *Nivel 5. En este nivel el concepto de ciclo de reloj no sólo influye en la comunicación entre componentes del sistema sino también en la ejecución de operaciones.*

La mayoría de las herramientas comerciales de síntesis que se presentan durante esta generación han generalizado la adopción del lenguaje C, o derivados del C, para presentar las descripciones conductuales o algoritmos a sintetizar. Esta decisión ha provocado que los diseñadores encuentren familiares estas herramientas. Grandes compañías de diseño y fabricación de circuitos integrados han adoptado en sus procesos de fabricación herramientas de SAN.

Las herramientas SAN han adoptado muchas de las técnicas aplicadas en la optimización de los compiladores para mejorar la calidad de los circuitos sintetizados. Estas optimizaciones se aplican sobre todo en las fases iniciales del proceso de SAN.

En resumen, en esta tercera generación se han desarrollado algoritmos y herramientas eficientes de SAN tanto en la síntesis de especificaciones formadas por el flujo de datos como de control.

1.3.5 Futuro de la SAN

El futuro de la SAN quizás llegue a plasmarse en un entorno de trabajo en el que varias herramientas de diseño interactúen para explorar distintas implementaciones alternativas, incluyendo diferentes arquitecturas objetivo:

ASIC o FPGA; diferentes tipos de implementaciones: puramente hardware, puramente software o mixtas en las que coexistan ambos tipos de diseños; implementaciones basadas en puertas o en procesadores; etc. En este entorno de diseño, las premisas a definir, las restricciones a utilizar y la representación del circuito a sintetizar serán únicas para el conjunto de herramientas de diseño integradas. Actualmente parece muy difícil alcanzar este tipo de entornos de trabajo debido a la poca capacidad de integración que ofrecen las herramientas de SAN actuales, ni siquiera de un modo semiautomático.

Como lenguaje de descripción HW o de diseño en las herramientas de SAN se consolidará la tendencia marcada en la generación actual, la tercera, de uso generalizado del C o variantes de éste.

1.4 Objetivos y organización de esta memoria

El objetivo principal del trabajo de investigación descrito en la presente memoria ha sido la propuesta e implementación de un sistema de SAN capaz de minimizar el área de los circuitos sintetizados, aumentando la reutilización de los elementos funcionales, de almacenamiento y de encaminamiento de las rutas de datos, sin degradar el rendimiento. Estos elementos aparecen comúnmente sobredimensionados en las implementaciones convencionales, debido a que los algoritmos y herramientas comerciales de SAN procesan las operaciones y datos como elementos indivisibles. La metodología de diseño propuesta permite la descomposición de operaciones y operandos con el fin de aumentar la reutilización de los recursos HW de la ruta de datos, mediante la identificación y repetición de las mismas cadenas de operaciones a lo largo de la ejecución de la conducta especificada.

En el capítulo dos se presentan las principales ideas relativas a los algoritmos de síntesis con restricción de tiempo y los principales objetivos a conseguir en este tipo de escenarios. Así mismo, se esbozan las técnicas más utilizadas para aumentar la calidad de los circuitos sintetizados, y en particular

se introducen aquellas que constituyen los pilares básicos sobre los que se apoya la metodología de diseño propuesta en este trabajo: la descomposición de operaciones y la gestión de patrones. Su aplicación conjunta permite obtener reducciones significativas del área requerida para la implementación de los esquemas estructurales generados por las herramientas de SAN.

El capítulo tres constituye el núcleo principal del trabajo de investigación expuesto. A través del mismo, se analiza en detalle la metodología de diseño propuesta y la viabilidad de su aplicación al proceso de SAN.

En el capítulo cuatro se presentan las fases de planificación, selección y asignación de HW del algoritmo de SAN propuesto. La fase de planificación se basa en el algoritmo de planificación dirigida por fuerzas que ha sido adaptado al manejo de patrones. Y la selección y asignación de recursos HW tiene por objeto que las ocurrencias de un patrón compartan los mismos recursos con el fin de incrementar su reutilización y disminuir el número de elementos de encaminamiento.

En el capítulo cinco se presenta un ejemplo sencillo de la ejecución del algoritmo propuesto. Así mismo, se presentan los principales resultados experimentales obtenidos en comparación con otros algoritmos convencionales de SAN.

Por último, en el capítulo seis se resumen las principales conclusiones y aportaciones del presente trabajo. Así mismo, se esbozan las líneas abiertas de mejora que pueden ser tratadas en futuras investigaciones.

CAPÍTULO

2

INFRAUTILIZACIÓN DE HARDWARE EN LA SÍNTESIS DE ALTO NIVEL

Uno de los principales inconvenientes de las herramientas de SAN y de los algoritmos relacionados con ella es que suelen generar rutas de datos en las que aparece un cierto desaprovechamiento de área debido a la excesiva incorporación de recursos físicos. Éstos generalmente no son necesarios durante toda la ejecución del circuito, y en ocasiones se encuentran

sobredimensionados. Estas características se traducen en una cierta cantidad de unidades funcionales, de almacenamiento y de interconexión por encima de las necesidades reales, y en un aprovechamiento no óptimo de las mencionadas unidades.

Recientemente se han propuesto un conjunto de técnicas basadas en la descomposición de operaciones en conjuntos de operaciones de menor complejidad, que aplicadas a la SAN han conseguido reducir el área, tiempo de ejecución o consumo de energía de los circuitos sintetizados [Moli05] [Saut07] [MRBM09]. En particular, las técnicas propuestas para la reducción del área tienen por objetivo equilibrar el número de operaciones del mismo tipo, representación de datos y anchura ejecutadas por ciclo. De esta manera se incrementa la reutilización potencial de los recursos funcionales, disminuyéndose el desaprovechamiento de HW, medido este desaprovechamiento como el número de ciclos en los que las unidades funcionales no ejecutan operaciones, o bien ejecutan operaciones de menor anchura.

En este capítulo se describen las técnicas convencionales aplicadas a la reducción del desaprovechamiento del área en la SAN, mostrando sus posibles puntos de mejora. Al mismo tiempo y como consecuencia de ello, se irán introduciendo los mecanismos novedosos que se proponen en esta memoria tendentes a aumentar el aprovechamiento de área en un escenario de SAN con restricción de tiempo.

En el apartado 2.1 se realiza una introducción a las causas que propician el desaprovechamiento de HW en las implementaciones sintetizadas por los algoritmos convencionales de SAN. En el apartado 2.2 se profundiza en los algoritmos de SAN con restricción de tiempo y se presentan las principales técnicas utilizadas para mitigar el desaprovechamiento de HW (encadenamiento de operaciones, unidades funcionales multiciclo, unidades funcionales segmentadas y descomposición de operaciones). En el apartado 2.3 se analizan las principales aportaciones de las técnicas de identificación y gestión de patrones a la SAN. Y finalmente, en el apartado 2.4 se lleva a cabo

una presentación general de la metodología de SAN propuesta en esta memoria y que se desarrolla en detalle en el siguiente capítulo. Dicha metodología se basa en la aplicación combinada de las técnicas de descomposición de operaciones y de identificación y gestión de patrones, y permite optimizar el área de las implementaciones generadas por las herramientas de SAN.

2.1 Desaprovechamiento de HW en los circuitos resultantes de la SAN

En la mayoría de los circuitos el aprovechamiento de los recursos físicos es inferior al 100%. Esto produce que se genere un desaprovechamiento o infrautilización de las unidades funcionales, unidades de almacenamiento y unidades de interconexión, debido a que en ciertos momentos del funcionamiento del sistema algunas unidades no están realizando un trabajo útil. Esta situación está provocada por la falta de eficiencia de los algoritmos de SAN en las fases de planificación y asignación de recursos. También es cierto que el aprovechamiento completo, o del 100%, es imposible en la práctica, aunque en la mayoría de los casos pueden introducirse mejoras en los algoritmos que reduzcan la infrautilización de los recursos.

Tomando como referencia las unidades funcionales, el desaprovechamiento de área aparece en las siguientes situaciones:

1) *Unidades funcionales ociosas en algún ciclo de reloj.* Esta situación ocurre cuando existe al menos una unidad funcional en la ruta de datos, a la que no se le ha asignado una operación de la especificación conductual en algún paso de control. En esos pasos de control concretos se dice que la unidad funcional está ociosa porque no está ejecutando operaciones de la especificación. Además de existir cierto desaprovechamiento de área, se está produciendo un consumo innecesario de energía ya que la unidad funcional está activa y en la mayoría de los casos está procesando una operación no útil, es decir que no se encuentra en la especificación dada.

2) *Unidades funcionales que procesan operandos de anchura menor que su capacidad.* Esta situación ocurre cuando la anchura de los operandos es menor que la de la unidad funcional. En estos casos se descartan los bits más significativos procesados por la unidad funcional.

3) *Unidades funcionales cuyo tiempo de ejecución es menor que el tiempo de ciclo.* Si la unidad funcional termina de procesar los operandos de entrada para generar el operando de salida antes de que termine el ciclo de reloj, permanecerá un tiempo sin procesar información nueva. Durante ese tiempo se puede considerar que la unidad funcional está ociosa y por lo tanto se está produciendo un desaprovechamiento de área.

El aprovechamiento completo del área sólo sería posible desde un punto de vista teórico si el número de operaciones de cada tipo diferente fuera múltiplo de la latencia (número de ciclos en que se ejecuta la especificación) y pudieran además planificarse en ciclos distintos, lo que no siempre es posible debido a las dependencias de datos entre las operaciones. En esta situación se podría obtener un equilibrio en el número de operaciones de cada tipo diferente ejecutadas por ciclo. No obstante, este aprovechamiento de los recursos funcionales no implica un tiempo de ejecución óptimo. En la mayoría de los casos los tiempos de ejecución de las operaciones varían notablemente en función del tipo y el tamaño de la operación y del recurso funcional seleccionado para ejecutarla. Igualmente, las técnicas de encadenamiento de operaciones y operadores multiciclo influyen en la desigualdad de los retardos de los distintos caminos de datos de los circuitos.

En la práctica, el aprovechamiento total tanto del área como del tiempo de ejecución de los circuitos está lejos de poder alcanzarse, debido al estilo descriptivo utilizado al especificar las conductas y a la rigidez de las herramientas de SAN disponibles actualmente.

A continuación se ilustra mediante un ejemplo el desaprovechamiento de área genérico tras aplicar un proceso convencional de SAN a una especificación conductual.

EJEMPLO 2.1

La figura 2.1 muestra el GFD de una especificación conductual, una posible planificación en 4 ciclos de reloj y la ruta de datos sintetizada por un algoritmo convencional. La ruta de datos está compuesta por dos unidades funcionales: un sumador de 16 bits y un multiplicador de 16x16 bits, tres registros: dos de 16 bits y uno de 32 bits, y cinco multiplexores: dos multiplexores de 2 entradas y 16 bits, dos de 3 entradas y 16 bits, y uno de 2 entradas y 32 bits.

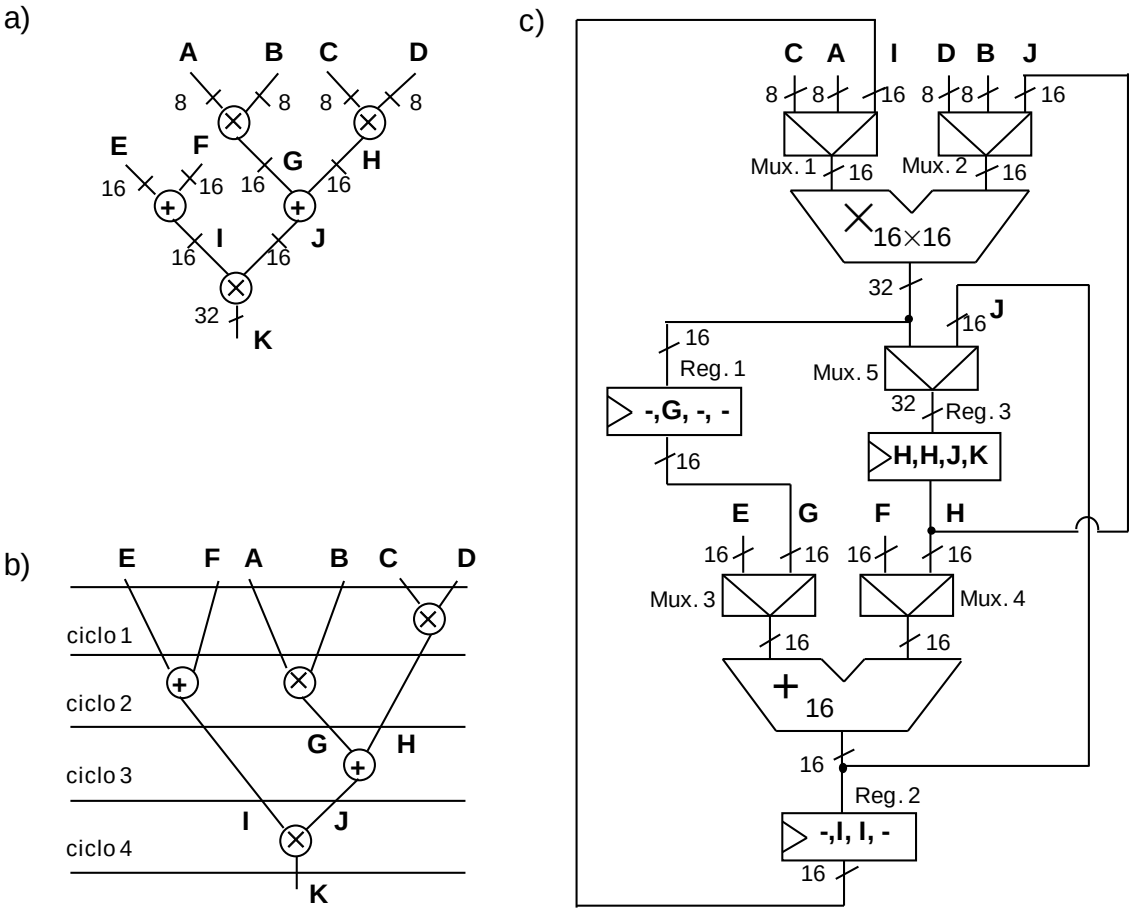

Figura 2.1 *SAN de una especificación conductual aplicando un algoritmo convencional.*

a) GFD.
b) Planificación.
c) Ruta de datos.

Nótese que con el objeto de simplificar la figura de la ruta de datos se ha evitado la representación de los truncamientos y extensión de operandos necesarios para la reutilización de los componentes de la ruta de datos. No obstante, resulta evidente a partir de las anchuras del cableado y de los recursos HW representados.

La tabla 2.1 muestra el desaprovechamiento de HW presente en esta implementación. Las celdas sombreadas muestran los recursos no utilizados en algún ciclo de reloj. Las parcialmente sombreadas muestran la porción desaprovecha de los recursos utilizados para ejecutar operaciones de menor anchura. Nótese que la parte sombreada de la tabla supone un desaprovechamiento total superior al 50%.

	Ciclo 1	**Ciclo 2**	**Ciclo 3**	**Ciclo 4**
Multiplicador 16×16 bits	C x D	A × B		I × J
Sumador - 16 bits		E + F	G + H	
Registro 1 - 16 bits		G		
Registro 2 - 16 bits		I	I	
Registro 3 - 32 bits	H	H	J	K
Mux. 1 (3 a 1) - 16 bits	C	A		I
Mux. 2 (3 a 1) - 16 bits	D	B		J
Mux. 3 (2 a 1) - 16 bits		E	G	
Mux. 4 (2 a 1) - 16 bits		F	H	
Mux. 5 (2 a 1) - 32 bits	H	H	J	K

Tabla 2.1 *Desaprovechamiento de los recursos de la ruta de datos de la figura 2.1*

Por otro lado, la duración del ciclo de reloj debe ser equivalente a la latencia del multiplicador de 16x16 bits. Sin embargo, en los ciclos 1, 2 y 3 se computan operaciones de menor anchura, y por tanto, las operaciones están terminadas antes de finalizar el ciclo. En particular, el tiempo desaprovechado (slack time) en estos ciclos supone aproximadamente la mitad del ciclo.

Otras posibles planificaciones y asignaciones de HW darían lugar a circuitos con desaprovechamientos semejantes a los del ejemplo. Únicamente la utilización de técnicas de descomposición de operaciones, como las que se presentarán más adelante, producirían reducciones significativas en el desaprovechamiento de HW y por consiguiente reducciones de área.

2.2 Desaprovechamiento de HW durante la planificación de operaciones

La fase de planificación de operaciones resulta clave en el desaprovechamiento de HW resultante de la SAN. En esta sección se estudian algunas de las técnicas utilizadas para mitigar sus posibles efectos adversos en el aprovechamiento de los recursos HW de las rutas de datos.

Los algoritmos convencionales de planificación utilizados en la SAN asumen los siguientes enunciados:

1) La latencia del circuito (número de ciclos de reloj en que se ejecuta la especificación) es equivalente al número de operaciones existentes en el camino crítico de la especificación (camino del GFD con el mayor número de operaciones). Nótese que en este enunciado no se han tenido en cuenta las técnicas de encadenamiento de operaciones.
2) El tiempo de ciclo (duración del ciclo de reloj) es equivalente a la suma de:
    a. El retardo de la operación más lenta de la especificación.
    b. El tiempo necesario para garantizar el acceso a los operandos de entrada antes de ejecutar la operación.
    c. El tiempo necesario para garantizar el almacenamiento de los resultados.

Con estas asunciones, resulta imposible obtener planificaciones sin desaprovechamiento de HW, ya que no todas las operaciones tienen el mismo retardo, ni todos los caminos de datos tienen la longitud del camino crítico. Todo esto sin considerar las dependencias de datos entre las operaciones, ni la existencia de diferentes tipos de operación en la especificación, lo que complica aún más la posibilidad de obtener planificaciones con el mismo número de operaciones de cada tipo ejecutadas por ciclo.

De este modo, los algoritmos de selección y asignación de HW trabajan habitualmente con especificaciones conductuales planificadas en las que no

existe un equilibrio completo en cuanto al número de operaciones de cada tipo ejecutadas por ciclo. Esto provoca que se generen rutas de datos en las que existen unidades funcionales ociosas en algún ciclo de reloj de la planificación.

Esta misma situación se puede aplicar a las unidades de almacenamiento. Y como derivación de ambas situaciones, si hay unidades funcionales ociosas y unidades de almacenamiento ociosas en algún paso de control, hay también unidades de encaminamiento ociosas por las que no se transmite información útil. Además, estas unidades funcionales, de almacenamiento y de encaminamiento ociosas consumen energía no destinada a procesar información útil. Por ello, además de existir un desaprovechamiento de área se produce un consumo innecesario de energía.

Durante los ciclos en que esas unidades están ociosas, continúan procesando, almacenando o transmitiendo información innecesaria para la correcta ejecución de la especificación dada. El consumo dinámico de energía en estas situaciones se podría paliar estabilizando las entradas de los recursos ociosos lo que supone un aumento del área del circuito. Por otro lado, el consumo estático es inevitable.

Con el fin de reducir el desaprovechamiento de área se han utilizado diversas técnicas tales como: encadenamiento de operaciones, unidades funcionales multiciclo, unidades funcionales segmentadas y técnicas de descomposición de operaciones. A continuación se estudian en detalle cada una de ellas.

2.2.1 Encadenamiento de operaciones

Dadas dos operaciones con dependencias de datos de tipo “lectura después de escritura” (una de las operaciones genera un resultado que es operando de entrada de la otra), una planificación convencional tiende a ejecutar dichas operaciones en ciclos distintos. Como consecuencia, aparece la necesidad de asignar unidades de almacenamiento para guardar la

información generada por la primera hasta que sea procesada por la segunda. A esa necesidad de unidades de almacenamiento hay que añadir el área dedicada a las rutas de interconexión entre las distintas unidades (funcionales y de almacenamiento).

El encadenamiento de operaciones consiste en la ejecución de dos o más operaciones con dependencias de datos en el mismo ciclo de reloj. De esta manera pueden obtenerse las siguientes ventajas:

1) Al encadenar operaciones del camino crítico se reduce la latencia del circuito.
2) Al planificar en cada ciclo operaciones y cadenas de operaciones con tiempos de ejecución similares, se reducen los tiempos muertos (slack time).
3) El tiempo de ejecución de varias operaciones aritméticas encadenadas es menor que la suma de los tiempos de ejecución de las mismas operaciones, dado que la ejecución de operaciones encadenadas conlleva el cálculo en paralelo de algunos bits del resultado de ambas operaciones [PaCh01][MaLD97].
4) Al encadenar varias operaciones en un ciclo, estas operaciones no pueden utilizar las mismas unidades funcionales Sin embargo, no es necesario almacenar el resultado producido por la primera ya que se utiliza como operando de entrada de la segunda en el mismo ciclo.

A la técnica tradicional de encadenamiento de operaciones, se han añadido algunas variaciones, como el encadenamiento de operaciones multiciclo, denominado multiciclo no entero, o el encadenamiento de fragmentos de operación [PaCH01][MRGM09].

Otros campos del diseño en los que se han aplicado recientemente optimizaciones basadas en el encadenamiento de operaciones son la planificación de conductas condicionales [KoWo02], el emplazamiento físico [OKBS05] y el rendimiento de los microprocesadores [GKKR02].

Como se estudiará más adelante, las mejoras en el aprovechamiento de área obtenidas al aplicar la técnica de encadenamiento de operaciones pueden verse incrementadas con las siguientes propuestas:

1) Incremento del número de operaciones aplicando la técnica de descomposición de operaciones.

2) Gestión de las ocurrencias de conjuntos de operaciones encadenadas aplicando técnicas de gestión de patrones.

2.2.2 Unidades funcionales multiciclo

El uso de unidades funcionales multiciclo incorpora la posibilidad de ejecutar una operación durante varios ciclos consecutivos de reloj. Al utilizar este tipo de unidades funcionales se puede reducir la duración del ciclo de reloj por debajo del retardo de las operaciones más lentas de la descripción conductual.

Las unidades funcionales multiciclo también permiten reducir los tiempos muertos que aparecen al final de cada ciclo. Esta situación ocurre, tal como se explicó anteriormente, cuando una operación termina antes del fin del ciclo y por tanto, la unidad funcional sobre la que se implementa permanece ociosa durante ese periodo de tiempo.

En las unidades funcionales multiciclo no se admite una nueva operación al principio de cada ciclo. Por ello, y a diferencia de las unidades funcionales segmentadas, se debe esperar a que termine la operación actual para comenzar con la siguiente.

Al igual que con una unidad funcional monociclo, también en las multiciclo puede aparecer un tiempo muerto al final del último ciclo. Por ello se han propuesto técnicas para intentar relajar ese tiempo muerto. Entre ellas destaca la técnica denominada multiciclo no entero [PaCh01] que propone el encadenamiento de unidades multiciclo con otras unidades multiciclo o

monociclo para calcular operaciones con dependencias de datos, minimizando el tiempo muerto al final del último ciclo.

Como mejora de esta técnica y con el fin de optimizar otros aspectos del diseño algorítmico se han propuesto algunas variantes aplicadas a la planificación de conductas condicionales [KoWo02] o a la reducción de los retardos debidos a la propagación de los datos [JKSC01].

2.2.2.1 Desaprovechamiento de área en las unidades funcionales multiciclo

Los algoritmos de SAN tratan de distribuir uniformemente las operaciones multiciclo teniendo en cuenta que la ejecución de estas operaciones abarca varios ciclos de reloj, durante los cuales no se puede compartir la unidad funcional multiciclo correspondiente. Al igual que sucede con las operaciones monociclo, cuanto más homogénea sea la distribución de las operaciones multiciclo en ciclos de reloj, menor será el desaprovechamiento de los recursos funcionales de la ruta de datos. En el caso de las unidades funcionales multiciclo existe además un desaprovechamiento interno de las mismas debido a que su HW no se usa completamente en ningún ciclo de reloj. En cada uno de los ciclos que abarca la ejecución de una operación multiciclo se utiliza únicamente una parte del HW de la unidad funcional para realizar algunos de los cómputos de la operación.

En la metodología de SAN propuesta, se plantea la descomposición de las operaciones multiciclo con el fin de:

1) Eliminar las unidades funcionales de mayor área y contribuir a la reducción del desaprovechamiento de HW inherente al uso de este tipo de unidades.

2) Aumentar el número de operaciones del mismo tipo, representación y anchura de datos, de tal modo que puedan compartir los mismos recursos funcionales. Este objetivo se persigue en el trabajo propuesto integrando en el proceso de SAN las técnicas de identificación y explotación de patrones de operaciones.

A continuación se presenta un ejemplo que ilustra el desaprovechamiento de HW que aparece al utilizar unidades funcionales multiciclo, y algunas posibles mejoras para reducir dicho desaprovechamiento de área.

EJEMPLO 2.2

La figura 2.2 muestra el HW de un multiplicador de 5x4 bits que requiere 2 ciclos de reloj para completar una operación. En cada uno de los ciclos sólo se utiliza una parte del HW para calcular la operación asignada. Las celdas coloreadas en la figura corresponden con el HW del multiplicador utilizado en cada ciclo. La estructura de la celda básica utilizada en el diseño del multiplicador se muestra en la figura 2.3.

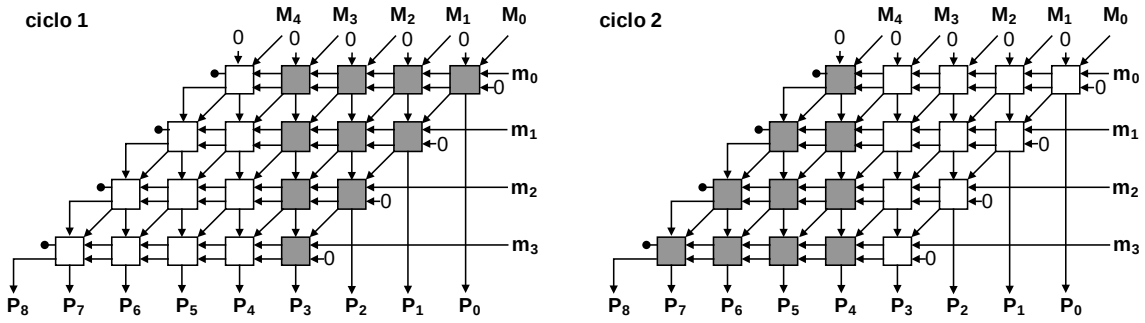

Figura 2.2 *Ejecución de una multiplicación de 5x4 bits en un multiplicador multiciclo de dos ciclos.*

El desaprovechamiento de HW del multiplicador multiciclo del ejemplo puede reducirse mediante la reutilización de algunas celdas del multiplicador en ambos ciclos, sin que esto suponga incremento alguno en la latencia de la operación. La figura 2.4 muestra una posible implementación del multiplicador con la mitad de celdas. Para simplificar la figura anterior se ha cambiado la dirección de los acarreos en la ejecución de la multiplicación durante el ciclo 2, tal y como se muestra en la estructura de la celdas básicas de la figura 2.5. Nótese que salvo este pequeño cambio en la representación, las estructuras de ambas celdas son idénticas.

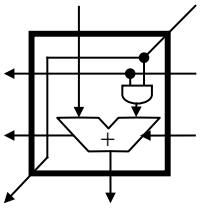

Figura 2.3 *Estructura de la celda básica utilizada en el diseño de la figura 2.2.*

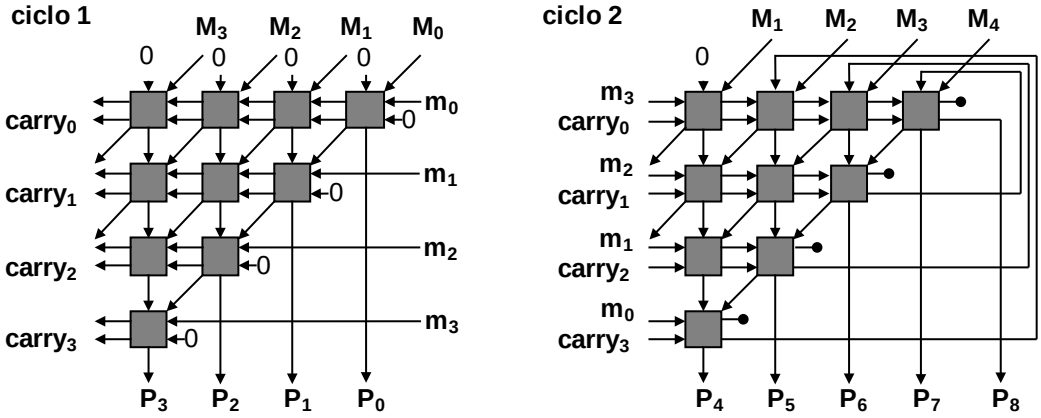

Figura 2.4 *Ejecución de una multiplicación de 5x4 bits en un multiplicador multiciclo de dos ciclos compartiendo las celdas básicas.*

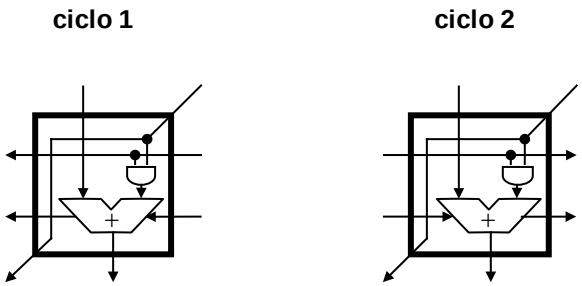

Figura 2.5 *Estructura de las celdas básicas utilizadas en el diseño de la figura 2.4.*

2.2.3 Unidades funcionales segmentadas

El desaprovechamiento interno que se produce en todos los ciclos de ejecución de una unidad multiciclo puede mitigarse si se permite la ejecución simultánea de varias operaciones en la misma unidad funcional. Para que esto suceda es necesario dividir el HW de la unidad funcional en varias etapas, cada una de las cuales realiza ciertos cálculos de la operación en cuestión, obteniéndose el resultado correcto al final de la última etapa. Los resultados intermedios calculados en cada etapa deben almacenarse en registros para ser utilizados como operandos de entrada en las siguientes etapas. El número máximo de operaciones que pueden estar ejecutándose en paralelo en este tipo de unidades funcionales coincide con el número de etapas. La duración de la etapa más larga, o si asumimos que todas las etapas tienen igual duración, la duración de una etapa cualquiera indica cada cuanto tiempo (medido en número de ciclos de reloj) puede iniciarse la ejecución de una nueva operación. Si la unidad se encuentra ejecutando el máximo número de operaciones producirá también un nuevo resultado cada vez que transcurra el tiempo equivalente a una etapa. Las unidades funcionales así diseñadas se denominan unidades funcionales segmentadas.

Tanto las unidades funcionales segmentadas como las multiciclo se han utilizado habitualmente para aumentar el rendimiento de los circuitos. En el caso de las unidades multiciclo se plantea el problema del desaprovechamiento de HW interno en todos los ciclos, y en el caso de las segmentadas el desaprovechamiento sucede cuando no es posible iniciar una nueva operación. Este desaprovechamiento afecta a toda la unidad ya que cada operación debe recorrer todas las etapas, y si en algún momento no se lanza una nueva operación, la siguiente siempre irá precedida por una etapa en que no se calculan nuevos datos. Además siempre se produce cierto desaprovechamiento durante el llenado y el vaciado de la unidad segmentada.

2.2.4 Sustitución de unidades funcionales multiciclo

Las unidades funcionales multiciclo y las multiciclo segmentadas pueden sustituirse por varias unidades funcionales monociclo independientes, capaces de ejecutar las operaciones atómicas calculadas en cada ciclo o etapa, obteniéndose así rutas de datos de menor área.

Las unidades funcionales monociclo se pueden utilizar para ejecutar operaciones multiciclo si el resultado producido por una unidad funcional es utilizado en el siguiente ciclo para proseguir con el cálculo de la operación especificada. De este modo las unidades funcionales monociclo no están dedicadas a la ejecución de las operaciones multiciclo sino que pueden reutilizarse para ejecutar otras operaciones presentes en la descripción conductual. Así mismo, pueden reutilizarse para ejecutar distintos fragmentos de una operación multiciclo.

La sustitución de unidades multiciclo por unidades monociclo requiere el uso de registros para almacenar los resultados intermedios calculados en cada ciclo. Esto supone un coste extra, en comparación con las unidades funcionales multiciclo, que se compensa ampliamente con la reducción del área de los recursos funcionales. En el caso de las unidades funcionales segmentadas, los registros ya están presentes en la propia unidad funcional, por lo que no supone un coste extra de área. Por el contrario, esta técnica permite reutilizar estos recursos de almacenamiento para almacenar otros datos de la especificación.

La utilización de unidades funcionales multiciclo o segmentadas para ejecutar operaciones monociclo no siempre es viable, ya que conlleva en muchos casos retrasos innecesarios que no siempre son asumibles, como por ejemplo las operaciones del camino crítico.

A continuación se presenta un ejemplo que ilustra esta situación continuando con el razonamiento del ejemplo anterior.

EJEMPLO 2.3

La figura 2.6 muestra un multiplicador de 5x4 bits segmentado en dos etapas capaz de iniciar una nueva multiplicación en cada ciclo de reloj. La celda básica de este multiplicador es idéntica a la mostrada en la figura 2.3. La implementación equivalente con unidades funcionales monociclo se muestra en la figura 2.7. Esta implementación también es capaz de iniciar una nueva multiplicación de 5x4 bits en cada ciclo de reloj. Ambas implementaciones ocupan la misma área, pero la segunda de ellas es capaz de ejecutar operaciones monociclo en un solo ciclo de reloj, evitando en muchos casos la necesidad de HW extra. Por ejemplo, una multiplicación de 3x2 bits podría ejecutarse sobre una de las unidades funcionales monociclo, mientras la otra calcula una operación diferente. La figura 2.7 muestra el desaprovechamiento de HW al ejecutar una multiplicación de 3x2 bits. Las celdas coloreadas son las no utilizadas durante la ejecución de la operación. Obviamente, la multiplicación de 3x2 bits podría también ejecutarse en el multiplicador segmentado, pero en ese caso tardaría dos ciclos en obtenerse el resultado de la misma. Además, el desaprovechamiento de HW del multiplicador sería mucho mayor. En la figura 2.6 se muestran coloreadas las celdas desaprovechadas en dicha ejecución. Nótese que todas las celdas de la segunda etapa del multiplicador segmentado están desaprovechadas, ya que la operación se completa en el primer ciclo.

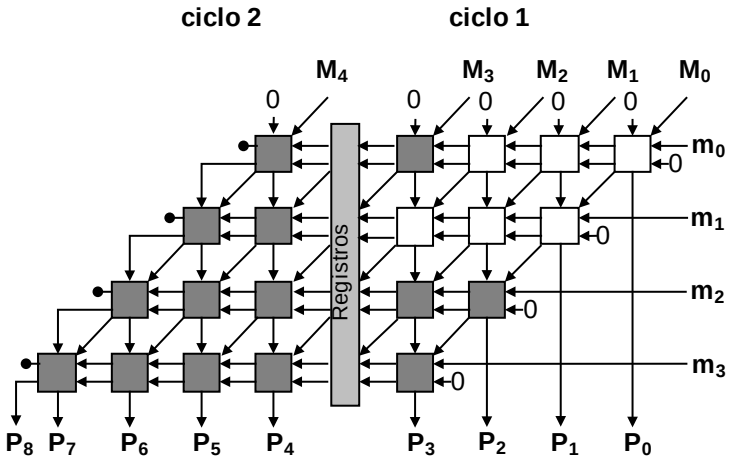

Figura 2.6 *Implementación de un multiplicador de 5x4 bits segmentado de dos etapas.*

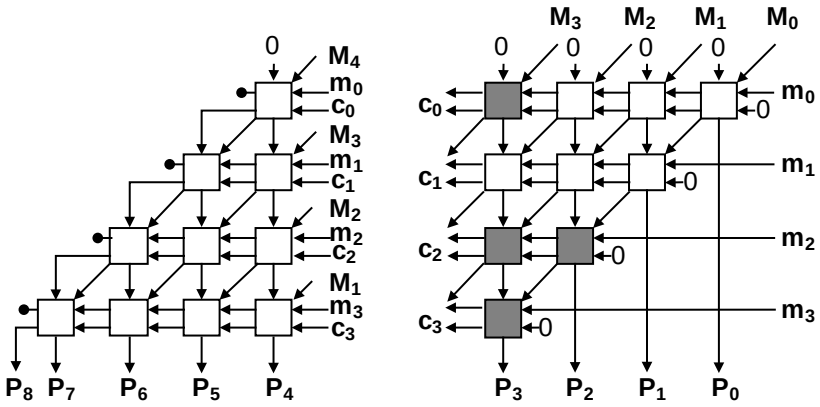

Figura 2.7 *Implementación del multiplicador de 5x4 bits segmentado utilizando unidades funcionales monociclo.*

2.2.5 Técnicas de descomposición de operaciones

Desde un punto de vista convencional de la SAN, las operaciones se caracterizan por:

1) Ser consideradas como elementos atómicos que no son susceptibles de poder dividirse o descomponerse en elementos más pequeños.

2) Ejecutarse en un determinado ciclo de reloj o en un conjunto de ciclos consecutivos.

3) Ejecutarse en una unidad funcional del mismo tipo que la operación y de anchura igual o mayor que la anchura de la operación.

La descomposición de operaciones rompe los esquemas de la SAN convencional permitiendo la división de una operación en varias más sencillas con dependencias de datos entre sí. Estas técnicas se utilizaron previamente en la síntesis lógica o RT de algoritmos de procesamiento de señales digitales (DSP), y más adelante en la SAN para optimizar el área, el tiempo de ejecución o el consumo de energía. A continuación se resumen algunos de los logros alcanzados mediante la aplicación de esta técnica:

1) *Reducción del área de los circuitos.* Dentro de este ámbito algunos trabajos han estudiado la viabilidad de las técnicas de descomposición de operaciones, así como de las técnicas de unificación de operaciones (ejecución en el mismo ciclo de varias operaciones del mismo tipo en el mismo recurso funcional) para reducir el área de los circuitos [CoCL00]. En estos trabajos la descomposición se ha aplicado de manera limitada, ya que sólo permiten la división de una operación en otras del mismo tipo. Aparte de estos estudios, la descomposición de operaciones se ha integrado con éxito en algunos algoritmos de SAN, aplicándose tanto a la fase de planificación de operaciones como a la de selección y asignación de recursos HW [MoMH03a] [MoMH03b] [MRMH06]. Las propuestas realizadas ejecutan en primer lugar la planificación y a continuación la fase de selección y asignación de recursos. Durante la planificación, las operaciones se descomponen con el fin de equilibrar el coste computacional (número de cálculos elementales útiles) de las operaciones planificadas en cada ciclo. Cuanto mayor haya sido este equilibrio mayores serán las reducciones de área que se obtendrán finalmente. Durante la fase de selección y asignación de HW, algunas de las operaciones resultantes de la fase anterior pueden descomponerse con el fin de aumentar la reutilización de los recursos de la ruta de datos. El principal inconveniente de estos algoritmos se encuentra en la selección del tipo de descomposición a realizar en cada momento. Dicha selección se realiza de manera local sin tener en cuenta el resto de operaciones de la especificación, lo que lleva a que en muchos casos se produzca una fragmentación excesiva de los recursos de la ruta de datos, lo que afecta también al controlador que resultará más complejo de lo estrictamente necesario.

2) *Reducción del tiempo de ejecución.* Las técnicas de diseño desarrolladas dentro de este ámbito se han centrado en la minimización del tiempo de ciclo, mediante la ejecución encadenada a nivel de bit de varios fragmentos de operaciones en el mismo ciclo [Saut07]. Tanto el número de fragmentos que se obtienen de cada operación, como el tamaño de los

mismos viene determinado por el mínimo tiempo de ciclo calculado para una especificación dada y una latencia fija. En dicho cálculo se tienen en cuenta los cómputos que pueden realizarse en paralelo al encadenar varias operaciones aritméticas en el mismo ciclo de reloj. Análogamente, fijado el tiempo de ciclo podría obtenerse la latencia mínima necesaria para ejecutar una especificación conductual. De este modo, se alcanzan diseños donde el tiempo de ciclo o la latencia no dependen directamente ni del número ni del retardo de las operaciones de la especificación. La aplicación de técnicas de descomposición de operaciones a la reducción del tiempo de ejecución, permite tanto la ejecución de una operación en varios ciclos de reloj no necesariamente consecutivos, como la ejecución de un fragmento de operación antes incluso de que se haya completado la ejecución de sus operaciones predecesoras. En el caso particular de las multiplicaciones, permite además la ejecución parcialmente desordenada de algunos fragmentos.

3) *Reducción del consumo de energía.* Las propuestas dentro de este ámbito se han centrado en la reducción tanto del consumo estático como del consumo dinámico de los circuitos. La descomposición de operaciones ha contribuido a reducir el consumo estático disminuyendo el área de los circuitos. Para ello, se produce un aumento de la reutilización de los recursos de la ruta de datos mediante la ejecución distribuida de las operaciones complejas en varias unidades funcionales [ErKP99] [ChJC00]. La contribución de la descomposición de operaciones a la reducción del consumo dinámico ha sido doble. Por un lado, se han diseñado circuitos en los que no se utilizan recursos de mayor anchura que las operaciones o datos, con el fin de minimizar el número de conmutaciones innecesarias en los recursos HW de la ruta de datos. De nuevo esto ha sido posible gracias a la descomposición de las operaciones complejas, lo que ha permitido eliminar las unidades funcionales de mayor tamaño y consumo de la ruta de datos. Con el mismo objetivo, los operandos de entrada de las unidades funcionales se estabilizan con los valores anteriores en los ciclos en los que no tienen operaciones asignadas. Por otro lado, la caracterización de los

valores de entrada de los circuitos también ha permitido la reducción del número total de conmutaciones. En el caso de la descomposición de operaciones, ésta ha contribuido a encontrar fragmentos de operación que tienen un comportamiento similar en sus valores de entrada, y por tanto, ejecutados en la misma unidad funcional reducen significativamente el consumo dinámico del circuito [MRBM09].

La técnica de descomposición de operaciones puede aplicarse a las unidades funcionales multiciclo y multiciclo segmentadas para reducir su desaprovechamiento de HW, y contribuir al aumento de la reutilización de HW entre las operaciones de la especificación conductual. En particular, la descomposición de operaciones multiciclo en operaciones más sencillas que puedan ejecutarse en unidades funcionales monociclo reduce tanto el número de unidades funcionales ociosas como el desaprovechamiento de HW debido a la ejecución de operaciones sobre recursos de mayor anchura, y contribuye notablemente a la reducción del área de los circuitos.

Esta técnica de diseño puede reducir significativamente la cantidad de área de las implementaciones si se aplica a las unidades funcionales más complejas, tales como las multiciclo, segmentadas o en coma flotante. En esta tesis, la técnica de descomposición de operaciones se aplica en primera instancia a las operaciones multiciclo con el objeto de que se ejecuten en unidades funcionales monociclo. En este contexto, la reducción de área de un multiplicador multiciclo que requiere dos ciclos para calcular una operación alcanza ¼ del área total del mismo, 1/3 si requiere tres ciclos y ½ si requiere 4 ciclos, tal como se detalla en [MRMH07]. Nótese que el ahorro real de área es generalmente mayor debido a la ejecución de operaciones monociclo en las unidades funcionales seleccionadas para ejecutar las operaciones multiciclo. Esta técnica puede extenderse a otros tipos de operaciones y, lo que resulta más interesante, puede modelarse para cada tipo diferente de especificación y restricciones distintas de tiempo con el objeto de maximizar la reducción de área.

En el siguiente ejemplo se ilustra cómo se puede aumentar el aprovechamiento de área mediante la aplicación de las técnicas de descomposición de operaciones.

EJEMPLO 2.4

En los ejemplos anteriores se ha descendido al nivel de celdas básicas para mostrar el desaprovechamiento de HW y el posible ahorro de área mediante la reutilización de dichas celdas. La técnica de descomposición de operaciones nos permite obtener las mismas ganancias en área de los ejemplos anteriores sin la necesidad de descender al nivel de celdas básicas.

La figura 2.8 muestra una posible optimización del área de un multiplicador multiciclo de 128x64 bits que requiere dos ciclos de reloj para completar una operación. Dicha optimización se basa en la reutilización de un multiplicador monociclo de 64x64 bits durante los dos ciclos de reloj necesarios para ejecutar una multiplicación de 128x64 bits. Esta implementación requiere un 40% menos de área que el multiplicador multiciclo, y mayores ahorros de área podrían alcanzarse si las nuevas unidades funcionales (multiplicador de 64x64 bits y sumador de 128 bits) se utilizaran para ejecutar otras operaciones de la especificación, contribuyendo de este modo a la reducción del área total de la ruta de datos.

Reutilizando el multiplicador de 64x64 bits se consigue reducir el área del circuito. El tiempo de ejecución de la nueva implementación es similar al de un multiplicador segmentado de dos etapas de 128x64 bits. Sin embargo, la reutilización potencial de HW es mayor en la implementación basada en unidades funcionales monociclo que en las implementaciones multiciclo (incluidas las unidades segmentadas), dado que son capaces de ejecutar tanto operaciones multiciclo como monociclo sin retrasar el resultado de las operaciones monociclo.

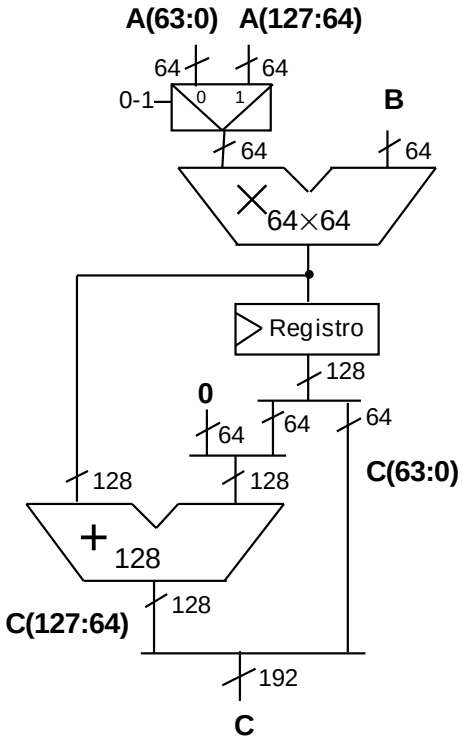

Figura 2.8 *Implementación optimizada de un multiplicador multiciclo de dos ciclos y anchura 128x64 bits.*

2.3 Técnicas basadas en la gestión de patrones

Según el Diccionario de la RAE, *patrón* en su acepción novena significa "*Modelo que sirve de muestra para sacar otra cosa igual*". El concepto de patrón ha sido utilizado extensamente en multitud de campos productivos y de investigación. En la década de los setenta aparecen las primeras referencias a la existencia de patrones en el desarrollo arquitectónico y diseño de ciudades [CAIS77].

Con posterioridad, el concepto de patrón comienza a ser usado en temas tan dispares como el análisis de cadenas de ADN, previsiones meteorológicas, reconocimiento de imágenes, estudios topográficos y geofísicos a partir de imágenes por satélite.

En el campo de las tecnologías, los patrones se han utilizado ampliamente en el desarrollo e implementación de compiladores desde principios de la década de los setenta. En el diseño de circuitos, los ingenieros de IBM fueron los primeros en aplicar técnicas basadas en la gestión de patrones durante el proceso de síntesis lógica.

2.3.1 Técnicas basadas en la gestión de patrones en la SAN

Dentro de la SAN, la extracción regular de patrones también se ha utilizado ampliamente. La síntesis basada en patrones abarca dos grandes tareas:

1. Reconocimiento de patrones: proceso mediante el cual se extrae información y características de determinados objetos concretos o abstractos con el objetivo de clasificarlos en un determinado conjunto.
2. Identificación de patrones: proceso mediante el cual se chequea la presencia de un determinado patrón en el objeto bajo estudio o análisis.

Los trabajos iniciales en la SAN que hacen uso de patrones tratan de planificar operaciones y asignar recursos HW según una librería de patrones proporcionada por el diseñador [BrRo97] [CKGP96] [LKMM95]. Estudios más recientes se han centrado en el reconocimiento de patrones estructurales basándose en técnicas gráficas de identificación [HuWP03] [InWM00] [CoJi08] [TeKa04].

Así mismo, se han aplicado técnicas de identificación exacta de patrones [HuWP03] [InWM00] y técnicas flexibles de identificación. En estas últimas, al relajar las condiciones de identificación de ocurrencias de un patrón se persigue aumentar el número de las mismas [CoJi08] [TKVI04]. Se consideran similares las cadenas de operaciones con variaciones en las anchuras de los datos, variaciones estructurales o variaciones en los puertos. Así por ejemplo, dentro del diseño con FPGA, se han utilizado técnicas flexibles de reconocimiento de patrones con el fin de reducir el coste de interconexionado [CoJi08]. Sin embargo, estas técnicas flexibles de identificación de patrones en

las que se consideran algunas transformaciones estructurales, no pueden obtener los beneficios alcanzables mediante la descomposición de operaciones.

Las técnicas basadas en la descomposición de operaciones permiten la fragmentación de operaciones en varias operaciones independientes (salvo por las dependencias de datos entre ellas) de diferentes tipos, representaciones y formatos. Estas nuevas operaciones pueden pertenecer a diferentes patrones y, en consecuencia, pueden planificarse en diferentes ciclos y ejecutarse sobre diferentes unidades funcionales. La combinación de las técnicas de identificación y gestión de patrones y la descomposición de operaciones resulta aún más versátil que las técnicas flexibles de identificación de patrones, ya que permiten identificar patrones que no existían originalmente en la especificación conductual, logrando implementaciones de mayor calidad.

2.4 Metodología de SAN propuesta para la reducción del desaprovechamiento de HW

El proceso de SAN que se propone en esta memoria incorpora las tres fases convencionales de un proceso típico de SAN: planificación, selección de HW y asignación. Desde un punto de vista abstracto se puede describir por los siguientes elementos:

1) *Elementos de entrada:* al igual que los procesos convencionales de SAN, la entrada se compone de:

    a. *Descripción conductual:* implementación algorítmica de un sistema digital.

    b. *Restricciones:* tal como se ha descrito en el capítulo primero las restricciones pueden ser de tiempo o de área. En el trabajo expuesto en esta memoria, se parte de una restricción de tiempo cuantificada

y definida por el diseñador (duración del ciclo de reloj o latencia fijos).

2) *Proceso de SAN:* conjunto de subprocesos especializados en cada una de las fases de la SAN convencional.

3) *Elementos de salida:* de igual manera que en los procesos clásicos de síntesis, se obtiene una ruta de datos y un controlador. La ruta de datos está compuesta por:

   a. *Unidades funcionales,* escogidas de la biblioteca de diseño durante la fase de selección. Las características y número de dichas unidades guarda cierta relación con los resultados obtenidos tras aplicar las técnicas de descomposición de operaciones y formación de patrones.

   b. *Registros,* necesarios para almacenar las salidas intermedias en la ejecución de la descripción conductual a lo largo de los ciclos de reloj.

   c. *Multiplexores,* necesarios para poder encaminar los datos provenientes de diferentes fuentes hacia los recursos funcionales y de almacenamiento de la ruta de datos.

   d. *Lógica pegamento,* conjunto de puertas lógicas necesarias para asegurar el correcto funcionamiento de la ruta de datos.

El controlador obtenido del proceso de síntesis es una máquina de estados finitos encargada de generar las señales de control necesarias para gobernar en cada paso de control todos los elementos de la ruta de datos.

Las rutas de datos obtenidas al aplicar la metodología propuesta en esta memoria están formadas por un conjunto de recursos cuyas características en cuanto al formato, anchura, tipo y número de los mismos, no tiene por qué corresponder con el número y formato de las operaciones y operandos de la descripción conductual. La no correspondencia de estas características

(número, tipo, formato y anchura) entre la descripción conductual y la ruta de datos generada, es debida a las transformaciones que realiza la técnica propuesta al aplicar la descomposición de operaciones y la teoría de patrones de una forma conjunta y recursiva en un mismo proceso de síntesis.

Del mismo modo, el número de operaciones útiles ejecutadas en la ruta de datos resulta mayor que en el caso de los algoritmos convencionales de SAN, debido igualmente a la descomposición de operaciones.

2.4.1 Técnicas de diseño aplicadas al proceso de SAN propuesto

Las técnicas de descomposición de operaciones se han aplicado con éxito para reducir el área de los circuitos. No obstante, su explotación en combinación con otras técnicas de diseño puede obtener mejores resultados.

En particular, la regularidad en las cadenas de operaciones que se repiten a lo largo de la especificación puede utilizarse para guiar las descomposiciones produciendo en muchos casos circuitos de gran calidad. De este modo, las operaciones se descomponen para extraer cierto patrón (conjunto de operaciones encadenadas) común que se repite en la mayoría de los ciclos. Los recursos funcionales necesarios para ejecutar dicho patrón son compartidos por un mayor número de operaciones, y se reducen los requerimientos de almacenamiento y encaminamiento de datos entre las operaciones del patrón, contribuyendo así a la reducción del área de la ruta de datos.

En esta tesis se propone un algoritmo de SAN capaz de realizar la planificación, selección y asignación de HW a partir de especificaciones conductuales. Dicho algoritmo descompone algunas operaciones en varias más sencillas con el fin de planificar en cada ciclo un número similar de operaciones con idéntico patrón. De esta manera, el área de las rutas de datos sintetizadas es bastante menor que las obtenidas por los algoritmos convencionales. Únicamente se consideran las descomposiciones que cumplen las restricciones de tiempo impuestas por el diseñador. Además, muchas de ellas también contribuyen a la reducción del ciclo de reloj,

produciendo así importantes mejoras en el rendimiento de los circuitos sintetizados.

La aplicación conjunta de ambas técnicas, descomposición y gestión de patrones, implica que se incrementen tanto la probabilidad de definir nuevos patrones, como la probabilidad de aumentar el número de ocurrencias de cada patrón. Además, los circuitos sintetizados siguiendo la metodología propuesta presentan las siguientes características:

1) Reducción de los requerimientos de unidades funcionales debido a un mayor reuso de las mismas, lo que reduce el desaprovechamiento de HW.

2) Reducción de los requerimientos de almacenamiento debido al encadenamiento de las operaciones de un patrón en un solo ciclo de reloj.

3) Reducción de los requerimientos de encaminamiento de datos entre los elementos HW que ejecutan las operaciones pertenecientes a patrones.

4) División de operaciones complejas en operaciones más sencillas.

5) Definición de operaciones abstractas asociadas a patrones.

2.4.2 Fases de la metodología de SAN propuesta

El proceso de síntesis propuesto puede dividirse en las siguientes fases:

1) *Descomposición de operaciones multiciclo.* Inicialmente se analizan las operaciones presentes en la descripción conductual. Este análisis consiste en localizar aquellas operaciones que debido a las restricciones de tiempo impuestas y al conjunto de unidades funcionales disponibles en la biblioteca de diseño, requieren más de un ciclo para ejecutarse. Estas operaciones se descomponen inicialmente en un conjunto de operaciones que pueden ejecutarse en un único ciclo de reloj.

2) *Identificación de patrones.* Este paso consiste en la identificación de patrones formados por un número determinado de operaciones de la

especificación. Dichos patrones deben poder ejecutarse en un único ciclo de reloj cuya duración ha sido fijada previamente por el diseñador, y superar cierto parámetro de calidad. Con el fin de incrementar tanto el número de patrones identificados como el número de ocurrencias de cada patrón, durante esta fase se exploran diversas descomposiciones de las operaciones de la especificación y se aplican las más prometedoras.

3) *Planificación de patrones.* Se selecciona el patrón de mayor calidad de entre los identificados en el paso previo. La métrica utilizada para medir la calidad de un patrón tiene en cuenta el número de ocurrencias del mismo, el número, tipo y anchura de las operaciones que lo forman y las movilidades de sus ocurrencias. Una vez seleccionado el patrón de mayor calidad, se utiliza una variante del algoritmo clásico de planificación dirigida por fuerzas para planificar las ocurrencias del mismo.

4) *Selección y asignación de HW.* Se selecciona de la biblioteca de diseño el conjunto de unidades funcionales necesarias para ejecutar las ocurrencias del patrón planificadas. A continuación se asignan dichos recursos a las operaciones del patrón.

5) *Gestión de las operaciones residuales.* Tras repetir iterativamente las fases 2), 3) y 4) se alcanza una situación en la que no es posible definir ningún patrón que cumpla la restricción de calidad impuesta. En este punto, las operaciones sin planificar son operaciones residuales que se planifican y asignan intentando reutilizar los recursos HW de la ruta de datos.

6) *Optimización de la ruta de datos.* Con el fin de optimizar el circuito obtenido, e*n esta* últim*a fase* se intenta reducir el "slack time", o tiempo ocioso de las unidades funcionales al final del ciclo de reloj, sustituyendo algunas de las unidades funcionales por otras más lentas y de menor área.

2.5 Conclusiones

En este capítulo se han analizado algunas de las principales técnicas propuestas para optimizar los circuitos resultantes de la SAN: encadenamiento de operaciones, unidades funcionales multiciclo, unidades funcionales segmentadas y descomposición de operaciones, poniéndose de manifiesto sus inconvenientes y posibles optimizaciones.

Igualmente se ha introducido la metodología de diseño propuesta en esta tesis, cuya base es la aplicación conjunta de las técnicas de descomposición de operaciones y gestión de patrones, y se ha mostrado la viabilidad de su aplicación al proceso de SAN. En el siguiente capítulo se analiza en detalle la metodología de diseño propuesta.

CAPÍTULO

3

DESCOMPOSICIÓN DE OPERACIONES Y GESTIÓN DE PATRONES

El objetivo de este capítulo es presentar una nueva metodología de diseño basada en el uso conjunto de las técnicas de descomposición de operaciones y gestión de patrones, y su aplicación a la descripción conductual objeto de síntesis. Se analizará así mismo cómo las dos técnicas, descomposición de operaciones y gestión de patrones, se realimentan de forma recíproca con el

fin de introducir transformaciones en la descripción conductual que conlleven una mejora en el uso y gestión del área necesaria, cumpliendo las restricciones de tiempo fijadas por el diseñador. En el capítulo 4 se procederá a presentar los algoritmos de planificación, selección y asignación de HW que incorporan la metodología de diseño propuesta en el presente capítulo.

Las técnicas de gestión de patrones se aplican con el objetivo de agrupar conjuntos de operaciones que se repiten en la especificación conductual según un determinado patrón de ejecución. Estas operaciones se tratan de manera especial durante el posterior proceso de SAN con el objeto de obtener el máximo beneficio en términos de área del circuito resultante. Con este fin, previamente se aplican las técnicas de descomposición de operaciones a las operaciones multiciclo. Esta descomposición tiene un doble objetivo: eliminar las operaciones que provocan cierto desaprovechamiento de HW debido a que deben ejecutarse en unidades funcionales multiciclo, y generar nuevas operaciones a partir de las anteriores que permitan una mejor reutilización de los recursos funcionales. Durante el proceso de SAN la técnica de descomposición de operaciones se aplica con el fin de aumentar el número de operaciones que pueden compartir los mismos recursos HW y disminuir así el área de los circuitos resultantes. En este caso, la descomposición de operaciones está guiada por las técnicas de identificación y gestión de patrones.

3.1 Optimización de la especificación conductual

Como paso previo al proceso de síntesis basado en la gestión de patrones, se lleva a cabo una optimización de la especificación conductual de partida. Esta optimización se divide en dos fases. Primero, tiene lugar una fase de homogeneización de las representaciones de los datos y extracción de los núcleos de ejecución comunes a las operaciones de la especificación. En segundo lugar, se lleva a cabo una fase de descomposición de las operaciones multiciclo. El objetivo principal de la homogeneización de los datos y la extracción del núcleo común de ejecución es aumentar el número

de operaciones que pueden ejecutarse en las mismas unidades funcionales. El objetivo de la descomposición de las operaciones multiciclo en la segunda fase consiste en reducir el desaprovechamiento de HW inherente a este tipo de recursos. Ambas fases generan habitualmente un aumento en el número de operaciones de la especificación. Este aumento en operaciones conlleva un enriquecimiento en los patrones potenciales a descubrir posteriormente.

El algoritmo utilizado para homogeneizar los datos de entrada y extraer los núcleos aditivos y multiplicativos de las operaciones de la especificación conductual es el presentado en [Moli05].

La descomposición de las operaciones multiciclo tiene lugar en varios pasos que se describen en las siguientes secciones.

3.1.1 Identificación de operaciones multiciclo

La identificación de las operaciones de la especificación conductual cuya ejecución requiere más de un ciclo de reloj se realiza teniendo en cuenta el conjunto de unidades funcionales de la biblioteca de diseño, y el valor del tiempo de ciclo definido por el diseñador. En particular, es necesario acceder al tiempo de ejecución de cada uno de los recursos funcionales de la biblioteca de diseño.

Definición 3.1: Se define la función *Tej(ope, uf)* siendo *ope* una operación de la descripción conductual y *uf* una unidad funcional de la biblioteca de diseño capaz de ejecutar la operación *ope*, como aquella función que devuelve el tiempo de ejecución de la operación *ope* en la unidad funcional *uf*.

Nótese que el resultado de esta función puede ser distinto para las diferentes unidades funcionales presentes en la biblioteca de diseño que sean capaces de ejecutar la operación en cuestión. Por consiguiente, el tiempo de ejecución final que tendrá una operación dependerá de la unidad funcional seleccionada para ejecutarla.

Esta fase de optimización es previa al proceso de síntesis propuesto por lo que no se ha realizado todavía la asignación de unidades funcionales a operaciones. Por ello, es necesario que se llegue a un compromiso sobre el tiempo de ejecución de una operación para que se le pueda aplicar la característica de ser multiciclo o no.

Durante esta fase, y para determinar si una operación es multiciclo, se ha decidido usar el tiempo de ejecución de la unidad funcional más rápida de entre todas las capaces de ejecutar dicha operación disponibles en la biblioteca de diseño. De esta forma, se podrá cumplir la restricción de tiempo de ciclo definida por el diseñador.

Definición 3.2: Sea *ope* una operación de la descripción conductual, *Librería* el conjunto de unidades funcionales de la biblioteca de diseño y *TiempoCiclo* la duración del ciclo de reloj fijada por el diseñador. Se define que la operación *ope* tiene la característica de ser multiciclo si su tiempo de ejecución en cualquier unidad funcional de la biblioteca de diseño capaz de ejecutarla es mayor que la duración del ciclo del reloj.

$$ope \text{ es multiciclo} \Leftrightarrow \forall uf \in Librería,\ Tej(ope, uf) > TiempoCiclo$$

Definición 3.3: Sea *ope* una operación de la descripción conductual y *Librería* el conjunto de unidades funcionales de la biblioteca de diseño. Se define el mínimo tiempo de ejecución de la operación *ope*, *MinTej(ope)*, como el tiempo de ejecución de la unidad funcional más rápida disponible en la biblioteca de diseño para implementar dicha operación.

$$MinTej(ope, Library) = \min_{uf \in Librería} \text{Tej}(ope, uf)$$

Definición 3.4: Se define *MC* como el conjunto de operaciones multiciclo localizadas en la fase de descubrimiento o identificación de operaciones de este tipo, aplicando para ello la definición 3.2.

El conjunto *SPEC* lo constituyen todas las operaciones presentes en la descripción conductual tras la fase de homogeneización de datos y extracción de núcleos comunes de ejecución. Este conjunto es uno de los elementos de entrada al proceso de síntesis. En la fase de identificación de

operaciones multiciclo se irán analizando de una forma secuencial todas las operaciones del conjunto *SPEC*.

A continuación se presenta la implementación algorítmica de la fase de identificación de operaciones multiciclo:

	Algoritmo 3.1 – Identificación de operaciones multiciclo	
1	Inicializar(SPEC);	Conjunto de operaciones iniciales
2	MC=Ø;	Conjunto de operaciones multiciclo
3	PAT=Ø;	Conjunto de patrones
4	**Para** ope € SPEC **hacer**	Descubrimiento de operaciones multiciclo
5	**Si** MinTej(ope, Librería) > TiempoCiclo **entonces**	Mínimo tiempo ejecución de la operación > Tiempo de Ciclo
6	MC = MC U {ope};	Se añade *ope* al conjunto de operación multiciclo
7	**FinSi**;	
8	**FinHacer**;	

Identificación de operaciones multiciclo

3.1.2 Descomposición de operaciones multiciclo

A las operaciones multiciclo identificadas se les aplica la técnica de descomposición. Es necesario recordar que el objetivo principal de esta fase es aumentar el número de operaciones con el propósito de que en la posterior formación de patrones el número de estos y las ocurrencias de cada uno de ellos pueda ser mayor. De esta forma, se aumentará la reutilización de los recursos de la ruta de datos: unidades funcionales, recursos de almacenamiento y de encaminamiento de datos.

Aunque la técnica de descomposición de operaciones también se puede aplicar a las operaciones monociclo, en esta propuesta de SAN se ha optado por descomponer inicialmente sólo las operaciones multiciclo, con el fin de no fragmentar excesivamente la especificación original. Algunas de las operaciones monociclo y de los nuevos fragmentos de operaciones también se descompondrán posteriormente durante el proceso de síntesis con el fin de tratar de aumentar el número de ocurrencias de un determinado patrón[1].

El objetivo concreto de las técnicas de descomposición es obtener fragmentos de operación del mismo tipo, representación y anchura que

[1] Patrón en análisis en un tiempo concreto de ejecución del algoritmo de SAN.

puedan finalmente reutilizar los mismos recursos de la ruta de datos. Al mismo tiempo que aumenta esta reutilización, disminuye el desaprovechamiento de los recursos HW.

Definición 3.5: Dada una operación multiciclo *ope* y la restricción de entrada referida a la longitud del ciclo de reloj *TiempoCiclo* definido por el diseñador, se define *NumFragmentos* como la cantidad de operaciones nuevas que se generan al descomponer una operación multiciclo *ope*.

En principio, el número de fragmentos que se obtienen al descomponer una operación multiciclo es igual al tiempo de ejecución de dicha operación dividido entre la longitud del ciclo de reloj.

$$NumFragmentos(ope) = \left\lceil \frac{MinTej(ope, Library)}{\text{TiempoCiclo}} \right\rceil$$

Tal como se indicó anteriormente, la función *MinTej(ope,Librería)* se corresponde con el tiempo de ejecución de la operación multiciclo *ope* en la unidad funcional más rápida de entre las disponibles en la librería de componentes.

Básicamente, las operaciones multiciclo se subdividen en varios fragmentos dividiendo sus matrices de cálculo verticalmente. Las sumas multiciclo se dividen en sumas de anchuras menores que la original. En el caso de las multiplicaciones multiciclo se obtienen multiplicaciones de anchuras menores, adiciones para sumar los resultados de las nuevas multiplicaciones y ciertas operaciones lógicas para recomponer completamente el resultado final de la multiplicación original multiciclo. Con el fin de reducir el número de operaciones de la especificación, algunas de esas operaciones lógicas y algunas sumas se han agrupado para formar nuevas operaciones denominadas *TUp, TDown* y *Rectangle*, para las que se han añadido a la biblioteca de diseño nuevas unidades funcionales capaces de ejecutarlas. De esta manera, las operaciones de los tipos *TUp, TDown* y *Rectangle* se sintetizan de manera similar al resto de operaciones. Todas estas descomposiciones se describen en detalle en [MRGH09].

Para implementar esta fase de descomposición de operaciones es necesario proceder a la definición de la función encargada de esta tarea.

Definición 3.6: Se define *FragmentosMonociclo* como la función que aplica las técnicas de descomposición a una operación multiciclo dada, generando un conjunto de nuevas operaciones monociclo. Según la naturaleza de la operación multiciclo en cuestión, se obtendrán operaciones diferentes.

La implementación algorítmica de esta fase se corresponde con:

	Algoritmo 3.2 - Descomposición de operaciones multiciclo	
1	**Para** ope en MC **hacer**	Se tratan secuencialmente todas las operaciones multiciclo
2	SPEC = SPEC – {ope}	La operación multiciclo a tratar se quita del conjunto de operaciones de la descripción conductual
3	SPEC = SPEC U FragmentosMonociclo(ope)	Se obtienen los fragmentos o nuevas operaciones procedentes de la operación multiciclo
4	**FinHacer**;	

Descomposición de operaciones multiciclo

Tal como se muestra en la implementación algorítmica, durante la fase de descomposición de operaciones multiciclo se incrementa el número de operaciones que van a estar disponibles durante la fase siguiente del proceso de síntesis.

En resumen, en esta fase se ha transformado el grafo de flujo de datos o GFD original en un nuevo GFD en el que cada operación multiciclo ha sido sustituida por un grupo de operaciones monociclo. Este nuevo grupo de operaciones realiza la misma función que la operación multiciclo original.

3.2 Gestión de patrones

Las técnicas convencionales que trabajan con patrones utilizan, para la formación de los mismos, las operaciones presentes en la descripción conductual original. Ello provoca que los resultados no estén optimizados ya que no se aplican técnicas previas que ayuden a alcanzar una mayor calidad

de los patrones a definir, y por tanto, una mayor calidad del circuito resultante de la SAN. Esta mayor calidad de los patrones se puede conseguir aplicando procedimientos de optimización de la descripción conductual tendentes a aumentar el número de operaciones disponibles previas a la formación de los patrones. De esta forma, el conjunto de operaciones que potencialmente pueden pasar a formar parte de un patrón aumenta.

El aumento en el número de operaciones redunda habitualmente en un mayor número de ocurrencias de los patrones presentes en la descripción conductual. Dicho aumento puede alcanzarse mediante la aplicación de dos técnicas convencionales en la teoría de SAN:

1) *Descomposición de operaciones:* esta técnica permite descomponer las operaciones para obtener fragmentos de operación u operaciones más sencillas que sustituyen a la operación original. Mediante una descomposición guiada de las operaciones de la especificación puede aumentarse el número de operaciones que pueden compartir los mismos recursos funcionales, reduciendo así el área de los circuitos obtenidos.

2) *Encadenamiento de operaciones:* esta técnica permite ejecutar de forma secuencial un conjunto de operaciones con dependencia de datos sin utilizar recursos de almacenamiento intermedios. Si se aplica después de la técnica indicada en el párrafo anterior, se pueden encadenar tanto operaciones originales como nuevas operaciones o fragmentos resultantes de la descomposición de operaciones. Al tratarse de operaciones con tiempos de ejecución menores es posible encadenar un mayor número de ellas en el mismo ciclo lo que aumentará la reutilización de HW y disminuirá las necesidades de recursos de almacenamiento. La condición que se debe dar para encadenar operaciones es que el tiempo de ejecución de la cadena sea inferior o igual a la duración del ciclo de reloj. Adicionalmente, la reutilización de los mismos recursos para encadenar operaciones según un mismo patrón disminuye los requerimientos de elementos de interconexionado.

La aplicación de las técnicas de descomposición y encadenamiento a las operaciones de la descripción conductual no se realiza de forma arbitraria, sino que tiene lugar de manera guiada siendo el objetivo principal maximizar la reutilización de los recursos HW de la ruta de datos. En la metodología de diseño propuesta, el grado en que se aplican dichas técnicas está relacionado con ciertos parámetros de calidad cuyos valores son fijados por el propio diseñador.

3.2.1 Escenario previo a la gestión de patrones

Tras haber dividido las operaciones complejas en fragmentos u operaciones monociclo, la descripción conductual original se ha transformado. Se ha pasado de un conjunto de operaciones formadas por operaciones multiciclo y monociclo a un nuevo conjunto formado únicamente por operaciones monociclo. Obviamente, el número de operaciones totales de este conjunto es superior al número de operaciones del conjunto inicial.

El objetivo principal de los algoritmos de gestión de patrones es agrupar o asociar un subconjunto de operaciones, bien sean operaciones de la descripción conductual original o nuevas operaciones o fragmentos generados en la fase anterior, en patrones para ser usados en las siguientes fases.

Antes de comenzar la descripción detallada del proceso de descubrimiento de patrones es necesario proceder a definir formalmente el concepto de SPEC introducido en apartados anteriores.

Definición 3.7: Se define *SPEC* como el conjunto de las operaciones de la descripción conductual en su estado inicial y en cualquiera de los estados por los que atraviesa dicha descripción en la aplicación de las diversas técnicas de optimización dentro del proceso de síntesis descrito.

Según se vayan aplicando las técnicas de descomposición el número de operaciones de este conjunto irá variando.

A continuación, se introduce el concepto de patrón como el objeto principal a ser manejado por el proceso de SAN propuesto.

Definición 3.8: Un patrón es un conjunto de operaciones originales, o nuevas operaciones obtenidas al descomponer operaciones complejas, o una combinación de ambas, que se ejecutan de forma encadenada en un sólo ciclo.

Los fragmentos que forman parte de un patrón pueden tener su origen en operaciones distintas. Así mismo, los distintos fragmentos cuyo origen es una operación concreta pueden pasar a formar parte de patrones diferentes. Al aumentar el número de operaciones, aumenta a su vez el espacio de exploración que tendrá nuestro algoritmo como objeto de búsqueda de patrones.

3.2.2 Operaciones cabecera de patrón

Los patrones se forman a partir de una operación que tiene un número de ocurrencias significativo en la especificación conductual (operación cabecera del patrón), y se van añadiendo otras operaciones dependientes de las que ya forman parte del patrón de manera que el esquema resultante tenga un número de repeticiones significativo dentro de la especificación. Por tanto, la fase de formación de patrones comienza por la selección de aquellas operaciones que puedan pasar a ser operaciones cabecera de patrón. Una vez formados los posibles patrones a partir de las operaciones cabecera, se escogerán de entre ellos aquellos que superen cierto parámetro de calidad. Los patrones con una calidad suficiente, serán procesados por el algoritmo.

Se puede dar el caso de que habiendo operaciones cabecera, a partir de ellas no se genere patrón alguno que cumpla las condiciones de calidad mínima. En este caso, sería necesaria la descomposición o fragmentación de algunas operaciones de la especificación conductual con el fin de aumentar el número de ocurrencias de algunas operaciones ya existentes y, por lo tanto, de un determinado patrón, lo que se describirá en apartados posteriores.

La construcción o definición de un patrón se realiza a partir de una operación que ha adquirido la condición de operación cabecera de patrón. En principio, todas las operaciones del nuevo GFD obtenido tras aplicar el proceso de descomposición a las operaciones multiciclo, podrían adquirir la condición de operación cabecera de patrón. Esto podría derivar en un coste computacional de cálculo que crecería de forma exponencial y llegar a hacer inviable la propuesta de síntesis presente.

Por dicho motivo, se hace necesario limitar las operaciones que pueden adquirir la condición de operación cabecera de patrón. Para ello se ha definido una heurística basada en un parámetro o propiedad de calidad denominado *PH* (*Pattern Head*[2]) cuyo valor es fijado por el diseñador.

Definición 3.9: Una operación se considera cabecera de patrón si el número de ocurrencias que tiene en el GFD es igual o superior al parámetro de calidad *PH* definido por el diseñador.

En la tabla 3.1 se muestran varios escenarios diferentes caracterizados cada uno de ellos por dos dimensiones: número de ocurrencias de las operaciones y valor de la propiedad de calidad *PH* definida por el diseñador.

Para un valor exigente de *PH*, por ejemplo 7, sólo las operaciones que tienen un alto número de ocurrencias podrían llegar a adquirir la naturaleza de operaciones cabecera. Esto se traduce en que los futuros patrones que se definan a partir de estas operaciones tendrán probablemente un número de ocurrencias elevado y, por lo tanto, podrán reutilizar las mismas unidades funcionales.

Con un valor exigente de *PH*, además de alcanzar el objetivo anterior, también se reduce el tiempo de cálculo computacional del algoritmo al tener que procesar menos operaciones cabecera. Hay que recordar que toda operación que se defina como operación cabecera se constituye como futuro punto de exploración de posibles patrones.

[2] *Pattern Head: Cabecera de patron.*

	Ocurrencias	PH=3 Cabecera patrón	PH=4 Cabecera patrón	PH=6 Cabecera patrón	PH=7 Cabecera patrón
Operación 1	5	SI	SI	NO	NO
Operación 2	4	SI	SI	NO	NO
Operación 3	6	SI	SI	SI	NO
Operación 4	7	SI	SI	SI	SI
Operación 5	3	SI	NO	NO	NO
Operación 6	1	NO	NO	NO	NO
Nº de operaciones cabecera patrón		**5**	**4**	**2**	**1**
Tiempo de cálculo computacional		Alto	Medio	Bajo-Medio	Bajo

Tabla 3.1 *Influencia del parámetro PH*

Con esta heurística basada en el parámetro *PH*, mínimo número de ocurrencias de una operación para poder ser considerada como operación cabecera de patrón, además de evitar que el tiempo de cálculo computacional crezca exponencialmente, se consigue aumentar la calidad de los futuros patrones a definir. Esto es debido a que se seleccionan operaciones que aparecen frecuentemente en el GFD. La frecuencia de la operación cabecera tiene una influencia directa en la frecuencia del futuro patrón a definir a partir de ella.

A partir de los resultados experimentales se ha podido comprobar que es recomendable que el valor que se asigne a *PH* sea próximo a la latencia del circuito. De esta forma se conseguirá una reutilización alta de los recursos HW de la ruta de datos.

3.2.3 Gestión de la calidad de los patrones

Por cada operación cabecera, se inicia un proceso de exploración de búsqueda de patrones a partir de la misma. Este proceso se describirá en posteriores apartados. Si se tuvieran en cuenta todos los patrones potencialmente posibles a partir de cada operación cabecera, el tiempo necesario para su proceso, es decir, el tiempo de cálculo computacional crecería exponencialmente.

Por ello, es necesario proceder a la definición del concepto de calidad mínima que debe reunir un patrón para ser considerado un patrón candidato.

Definición 3.10: Un patrón es considerado como *candidato* para ser procesado si el número de ocurrencias que tiene en el GFD es igual o superior al valor del parámetro de calidad *CP* (*Candidate Pattern*[3]) definido por el diseñador.

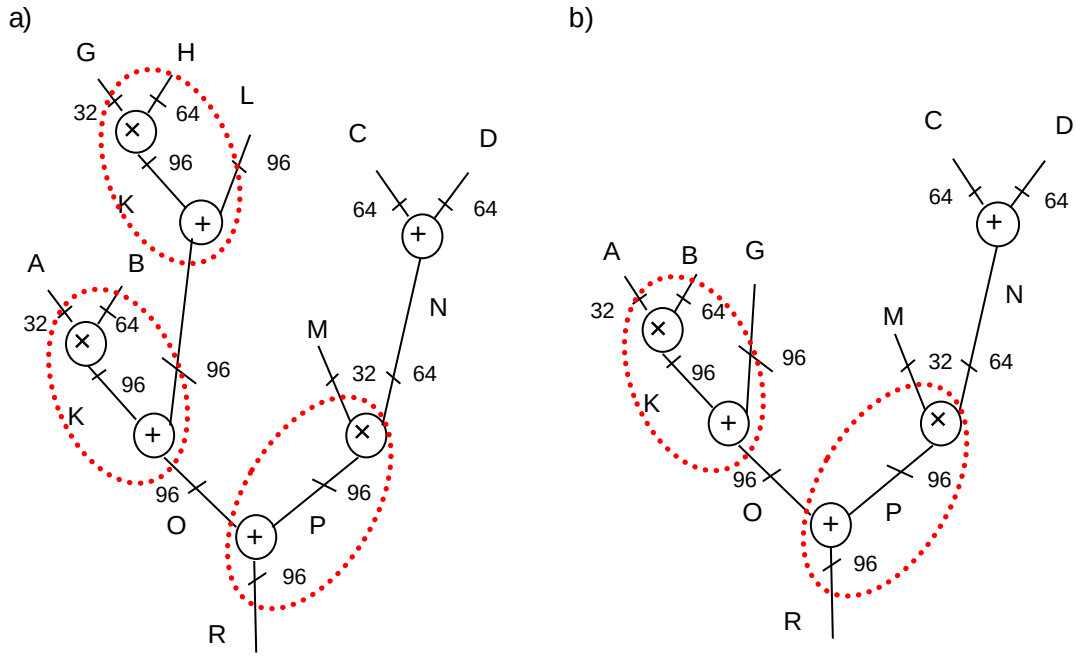

Figura 3.1 *Influencia del parámetro de calidad CP = 3 en la selección de patrones.*

En la figura 3.1 se presentan dos ejemplos sobre la influencia del parámetro de calidad *CP* en la selección de patrones. Para un valor de *CP* igual a 3, el patrón de la figura 3.1 a) será considerado patrón candidato pues su número de ocurrencias es igual al valor del parámetro, mientras que el de la figura 3.1 b) no lo será por tener un número de ocurrencias menor que *CP*.

Los patrones que no superen la condición definida no serán considerados como patrones candidatos y por lo tanto no serán procesados como tales.

En consecuencia, la calidad de un patrón está condicionada por los parámetros indicados, *PH* y *CP*. Variando el valor de estos parámetros se

[3] *Candidate Pattern: Patrón candidato.*

obtendrán diferentes resultados del proceso de síntesis, con rutas de datos y tiempos de ejecución distintos.

En la tabla 3.2 se puede comprobar el efecto que tienen distintos valores del parámetro *CP* en el número de patrones a procesar y en el tiempo de cálculo del algoritmo.

Para un valor poco exigente del parámetro *CP*, habría un alto número de patrones seleccionados lo que provocaría dos situaciones negativas:

1) *Tiempo de cálculo elevado*.

2) *Aprovechamiento bajo de las unidades funcionales:* En general, las unidades funcionales que se asignan a cada patrón serán reutilizadas pocas veces al tener dichos patrones un número reducido de ocurrencias.

	Ocurrencias	CP=3	CP=5	CP=6	CP=7
		Patrón selec.	Patrón selec.	Patrón selec.	Patrón selec.
Patrón 1	9	SI	SI	SI	SI
Patrón 2	3	SI	NO	NO	NO
Patrón 3	6	SI	SI	SI	NO
Patrón 4	8	SI	SI	SI	SI
Patrón 5	5	SI	SI	NO	NO
Patrón 6	4	SI	NO	NO	NO
Patrones seleccionados		**6**	**4**	**3**	**2**
Tiempo de cálculo computacional		Muy Alto	Alto	Medio	Bajo

Tabla 3.2 *Influencia del parámetro CP*

Al igual que con el parámetro *PH*, es recomendable que el valor del parámetro *CP* se aproxime a la latencia definida por el diseñador para que la reutilización de las unidades funcionales sea alta. Además, el valor de *CP* debería ser menor o igual que *PH* para garantizar que todas las operaciones cabecera puedan ser usadas para formar patrones candidatos. Esto es debido a que un patrón no puede tener más ocurrencias que el número de ocurrencias de la operación cabecera a partir de la cual se generó. Nótese que según esto, la propia operación cabecera es un patrón candidato formado por una operación. También, cualquier subconjunto de operaciones encadenadas de un patrón candidato es a su vez un patrón candidato.

La combinación de estos dos parámetros, *PH* y *CP*, definidos por el diseñador, representan el esfuerzo computacional que realiza el algoritmo para ejecutar el proceso de descubrimiento de patrones en el espacio de exploración. Estos dos parámetros tienen una influencia directa en los resultados que se obtienen de la síntesis. Al definir valores pequeños, se obtendrán muchos patrones con pocas ocurrencias, lo que provoca en la mayoría de los casos una reutilización baja de los recursos HW. Por el contrario, si los parámetros son altos y siempre que se cumplan las condiciones indicadas anteriormente, se obtendrán pocos patrones y el desaprovechamiento de HW asociado a los mismos será mucho menor que en el caso anterior.

Sin embargo, no existen unos valores de *PH* y *CP* ideales válidos para todos los diseños. El número y naturaleza de las operaciones presentes en la descripción conductual, así como el conjunto de las relaciones de dependencias entre ellas, constituyen el principal condicionante en la forma en que el algoritmo de SAN procesará el GFD. Por todo ello, sería recomendable que para cada diseño se llevase a cabo un proceso de ajuste en ambos parámetros hasta llegar a una solución acorde con los objetivos definidos por el diseñador. Unos valores exigentes en los parámetros de calidad *CP* y *PH*, no siempre conllevan una optimización alta de los recursos HW.

3.2.4 Cálculo de patrones

Los patrones se forman a partir de operaciones que cumplen la propiedad de calidad *PH* (su número de ocurrencias es igual o mayor que el valor de *PH*) y son por tanto operaciones cabecera de patrón. El algoritmo propuesto utiliza cada una de las operaciones cabecera para generar un conjunto de patrones candidatos. Dichos patrones deben poder ejecutarse en un único ciclo de reloj, utilizando las unidades funcionales disponibles en la biblioteca de diseño, y cumplir la propiedad de calidad *CP*, es decir, su número de ocurrencias debe ser igual o superior al valor del parámetro *CP*. El proceso de cálculo de patrones puede dividirse en las siguientes fases:

1) *Cálculo de las operaciones cabecera de patrón.*

2) *Generación de patrones.*

3) *Análisis del tiempo de ejecución del patrón.*

4) *Comprobación de la condición de calidad CP.*

A continuación se detallan cada una de las fases anteriores, según el orden en que se ejecutan las mismas.

3.2.4.1 Cálculo de las operaciones cabecera de patrón

El primer paso para la generación de patrones consiste en la identificación de las operaciones cabecera de patrón. Esta identificación se lleva a cabo contando el número de ocurrencias de cada tipo de operación distinto presente en la especificación conductual. Aquellas operaciones cuyo número de ocurrencias sea mayor o igual al valor del parámetro *PH* se convierten en operaciones cabecera de patrón, a partir de las cuales se generarán posteriormente los patrones.

A continuación se presenta la implementación algorítmica de la selección de las operaciones del GFD que cumplen el parámetro de calidad *PH*, y el cálculo de los patrones candidatos a partir de las mismas.

	Algoritmo 3.3 – Cálculo de patrones candidatos	
1	PAT=Ø;	Conjunto inicial de patrones a procesar
2	**Para** ope en SPEC	Se tratan secuencialmente las operaciones del GFD de SPEC
3	**Si** Ocurrencias(ope) >= PH **entonces**	Gestión de calidad *PH* según ocurrencias de la operación
4	PAT = PAT U **CalcularPatron**(ope)	Si cumple calidad *PH* se calculan los patrones que genera *ope*
5	**FinSi**;	
6	**FinHacer**;	

Cálculo de patrones candidatos

En el cuerpo algorítmico anterior, el cálculo de los patrones se realiza mediante la función *CalcularPatron*. Esta función devuelve todos los patrones candidatos que se pueden formar a partir de una operación cabecera dada. El proceso de cálculo de los mismos se describe en las siguientes secciones.

3.2.4.2 Generación de patrones

A partir de cada operación que cumple la propiedad de calidad *PH* se pueden generar varios patrones. El primer patrón potencial es el formado únicamente por la propia operación cabecera. El resto de patrones se forman añadiendo operaciones que tengan una dependencia de datos descendente con las operaciones del patrón validado previamente (operaciones sucesoras). Así, tras validar el patrón formado por la operación cabecera, el siguiente patrón potencial estará formado por la operación cabecera y una operación sucesora.

Se podría haber planteado un proceso de exploración inverso a partir de las operaciones cabecera. En los dos casos el conjunto de operaciones cabecera inicial sería el mismo. Pero en este segundo caso, la exploración sería ascendente en el árbol de dependencias de operaciones. Las dos opciones son válidas y a priori no hay ventajas o inconvenientes que lleven a optar por una u otra. Sin embargo, los resultados del proceso de síntesis serán diferentes debido a que tanto el conjunto de patrones como sus ocurrencias son distintos. También las descomposiciones de operaciones que tengan lugar durante la síntesis podrían ser diferentes, y por lo tanto, también el conjunto de operaciones de la especificación conductual con la que trabaja el algoritmo.

El proceso de cálculo de patrones a partir de una operación cabecera es un proceso repetitivo en el que se añade un nuevo patrón en cada paso, siempre que dicho nuevo patrón pueda ejecutarse en un único ciclo de reloj y cumpla la condición de calidad *CP*.

3.2.4.3 Análisis del tiempo de ejecución del patrón

Antes de proceder a comprobar si el nuevo patrón cumple el parámetro de calidad *CP* definido por el diseñador, es necesario comprobar que todas las operaciones que forman parte del mismo pueden ejecutarse en un ciclo de reloj, es decir, el tiempo de cálculo del patrón es menor o igual que el tiempo de ciclo definido por el diseñador. Para el cálculo del tiempo de ejecución del patrón se tiene en cuenta el tiempo de cálculo de las unidades funcionales más rápidas de las disponibles en la biblioteca de diseño. Sin embargo, esta asunción no compromete la posterior fase de selección de HW, ya que podrán

seleccionarse otras unidades funcionales más lentas siempre y cuando todas las operaciones del patrón puedan calcularse en el ciclo en que ha sido planificado.

Nótese que las operaciones cabecera son siempre patrones válidos ya que existe alguna unidad funcional en la biblioteca de diseño capaz de ejecutarlas en un tiempo menor o igual que el tiempo de ciclo definido. En el caso contrario, hubieran sido procesadas como operaciones multiciclo y descompuestas en varias operaciones monociclo, en las primeras etapas del algoritmo.

Si el patrón en proceso de evaluación sobrepasa el tiempo de ciclo, la búsqueda de patrones a partir de la operación cabecera finaliza al no poder añadirse más operaciones al patrón sin sobrepasar la duración del ciclo de reloj.

Una vez comprobado que el patrón puede ejecutarse en un único ciclo de reloj, se procede a comprobar si éste cumple la propiedad de calidad *CP*.

3.2.4.4 Comprobación del parámetro de calidad *CP*

Un patrón potencial se convierte en un patrón candidato si cumple la propiedad de calidad *CP*, es decir su número de ocurrencias es igual o mayor que el valor de *CP*.

El patrón básico formado únicamente por una operación cabecera también debe satisfacer esta condición para ser considerado un patrón candidato. Por tanto, las operaciones cabecera deben cumplir tanto la propiedad *PH* como la *CP* para ser patrones candidatos.

En la figura 3.2 se muestra de una forma gráfica un patrón que no cumple la propiedad de calidad *CP*. En este caso el valor de la propiedad *CP* se ha fijado en cuatro, es decir, es necesario que un patrón tenga al menos cuatro ocurrencias para poder ser considerado patrón candidato. El patrón que se muestra en la figura 3.2 a) tiene solamente tres ocurrencias por lo que no cumple la propiedad *CP* y no podrá formar parte del conjunto de patrones candidatos. En el caso de que se siguiese con la exploración y se probara a evaluar el patrón formado por el patrón anterior más la operación S=Q+R, tal

como se indica en la figura 3.2 b), tampoco se alcanzaría a cumplir la propiedad *CP* ya que para este caso el número de ocurrencias del nuevo patrón es de solamente una.

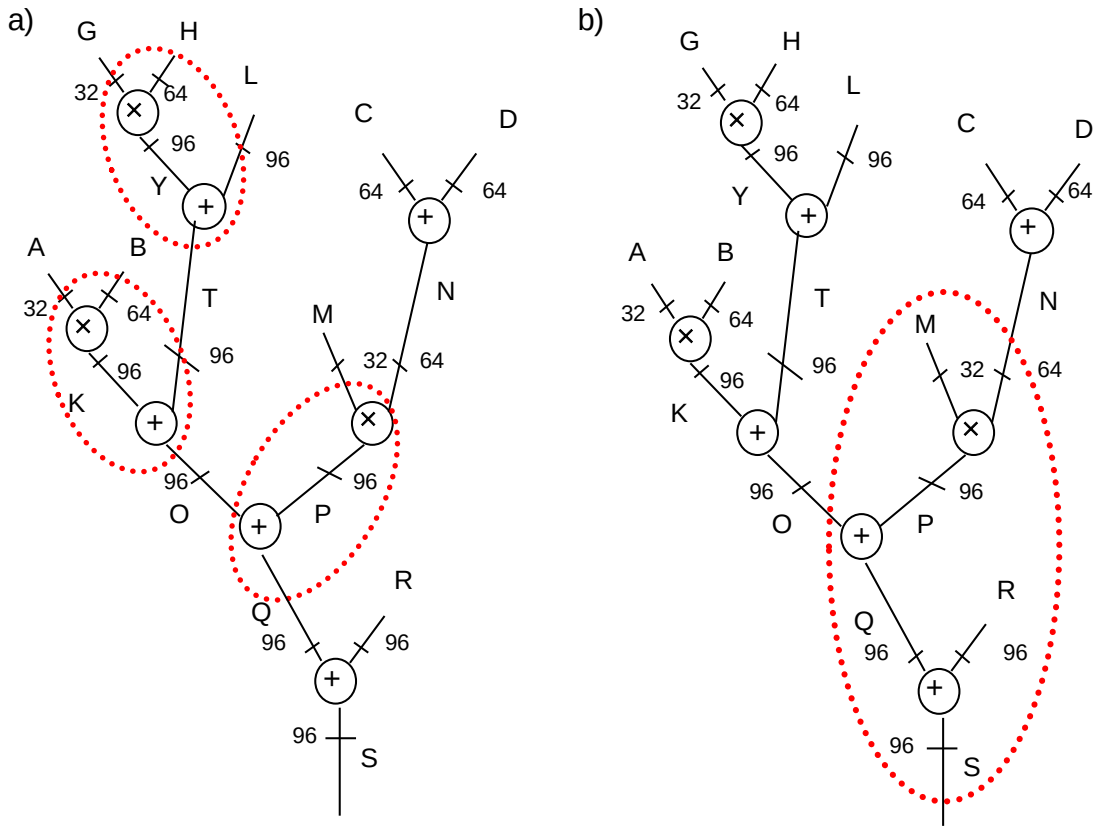

Figura 3.2 *Incumplimiento de propiedad de calidad CP = 4.*
a) Patrón con tres ocurrencias. No cumple CP.
b) Patrón formado por patrón a) y operación sucesora tampoco cumple CP.

De este último razonamiento se puede concluir que en el caso de que un patrón potencial no cumpla la propiedad de calidad, tampoco la cumplirán los patrones que pudieran crearse al añadir operaciones sucesoras al mismo.

Si el patrón potencial no cumple la condición *CP* se lleva a cabo un proceso de descomposición o fragmentación de operaciones con el fin de aumentar el número de ocurrencias del mismo hasta alcanzar el valor del parámetro. Este proceso está formado por dos pasos:

1) Simulación de las posibles descomposiciones de las operaciones sucesoras de aquellas ocurrencias del patrón reconocido previamente, y a las que no

ha sido posible añadir una nueva operación sucesora con el objeto de adaptarse al esquema del nuevo patrón (patrón maestro). Con estas simulaciones se pretende comprobar si los fragmentos de operación que se generan pueden aumentar el número de ocurrencias del patrón con el fin de que cumpla la propiedad de calidad *CP*.

2) Ejecución de las descomposiciones de operaciones simuladas previamente, en el caso de que el patrón potencial cumpla así la propiedad de calidad *CP*.

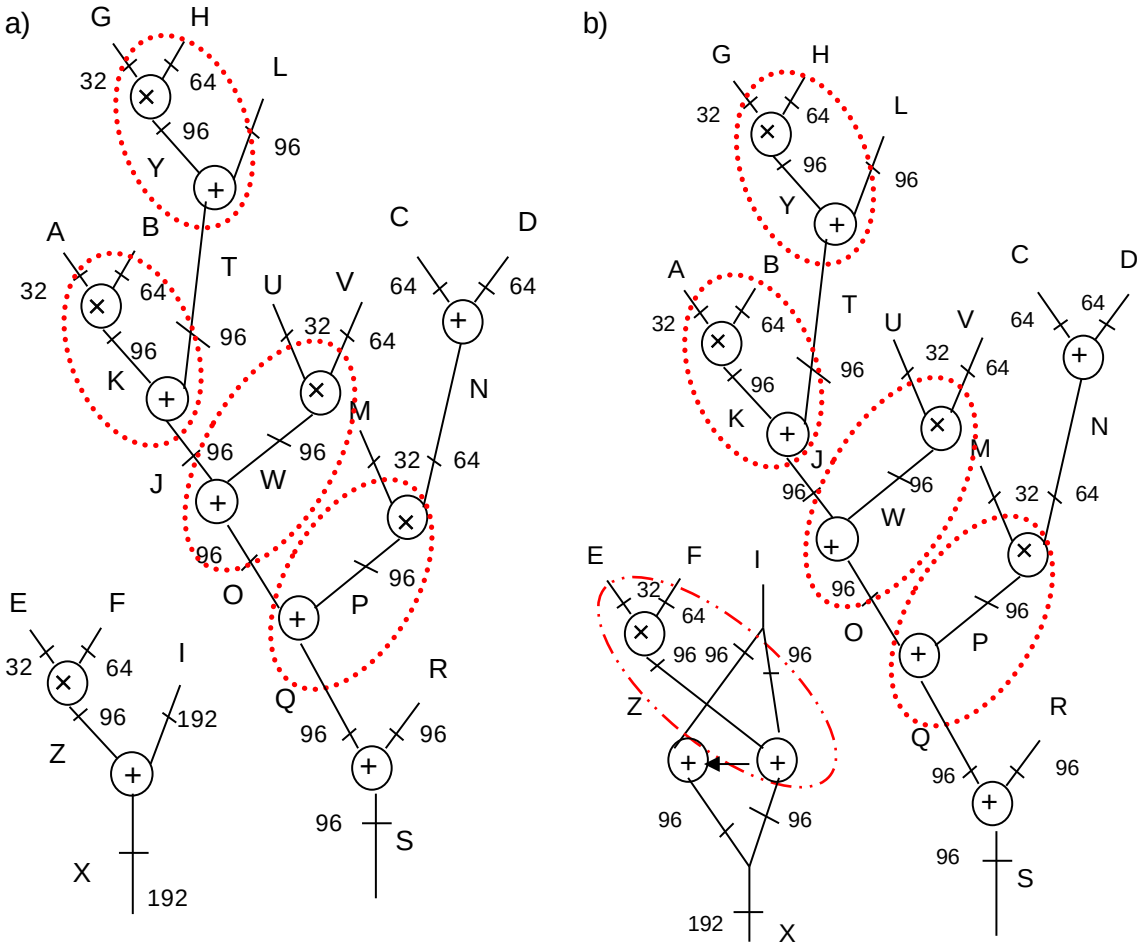

Figura 3.3 *Descomposición de operación para cumplir CP=5.*

a) Patrón en estudio con cuatro ocurrencias. Posible descomposición de la operación X:192 = Z:96 + I:192 en dos sumas.

b) Descomposición de la operación X en dos sumas de anchura 96 bits. Con la descomposición aparece una nueva instancia del patrón en estudio y alcanza las cinco ocurrencias cumpliendo la propiedad de calidad CP.

La figura 3.3 muestra gráficamente la descomposición de una operación sucesora para obtener una nueva ocurrencia de un determinado patrón.

Una vez identificado un nuevo patrón que cumple la propiedad de calidad *CP*, el algoritmo evalúa el patrón que se obtiene al añadir a éste una nueva operación sucesora. En el ejemplo de la figura 3.2, si cambiamos el parámetro de calidad *CP* al valor 3, tras identificar el patrón de la figura a) se pasaría a examinar el patrón de la figura b). Este último está formado por el patrón recién reconocido y la operación sucesora S:96 = Q:96 + R:96.

A continuación se presenta el pseudocódigo del algoritmo de cálculo de patrones a partir de una operación cabecera.

	Algoritmo 3.4 - Cálculo patrones a partir de una operación cabecera	
1	**Funcion CalcularPatron**(ope)	ope: operación cabecera base de la función
2	PATope=Ø;	Conjunto de patrones con ope como operación cabecera
3	patron=ope;	El primer patrón es la propia operación cabecera ope
4	**Mientras** (TiempoEjecucion(patron)<=TiempoCiclo) **and** Ocurrencias(patron) >= CP **hacer**	Si el patrón puede ejecutarse en un ciclo y cumple calidad CP, intentar construir siguiente patrón
5	PATope=PATope U patron;	Se añade al conjunto de patrones de ope por cumplir CP
6	suc=ImmSuc(patron)	Se calcula la operación sucesora del patrón
7	**Si** Ocurrencias(patron+suc) < CP **entonces**	Si patrón+sucesor no cumple CP, se simula la fragmentación
8	NumOcurrencias=0;	Se inicializa el número de ocurrencias del patrón potencial
9	DOP = Vacio;	Conjunto de operaciones a ser fragmentadas
10	**Para** cadena **en Instancias(**patron**) hacer**	Para cada instancia, cadena, del patrón actual
11	**Si EsPosibleFragmentar(Sucesor(**cadena**),**suc**)**	Si de la operación sucesora se puede obtener suc
12	**entonces**	
13	NumOcurrencias=NumOcurrencias + 1	Se incrementan las ocurrencias del patrón patron+suc
14	DOP=DOP U **Sucesor**(cadena)	Operaciones sucesoras de las instancias del patrón a fragmentar
15	**FinSi**;	
16	**FinHacer**;	
17	**Si** NumOcurrencias **>= CP entones**	Numero ocurrencias del patrón potencial cumple calidad CP
18	**Para** ope **en** DOP **hacer**	Para cada operación a fragmentar
19	**Fragmentar(**ope,suc,SPEC**);**	Fragmenta operación ope de DOP para obtener operación suc
20	**FinHacer**;	
21	patron=patron+suc;	Patrón potencial pasa a ser nuevo patrón por fragmentación
22	**EnCasoContrario**	
23	patron=Ø	Patrón potencial no pasa a ser nuevo patrón por no cumplir CP
24	**FinSi**;	
25	**EnCasoContrario**	Si patron+suc cumple calidad CP, pasa a ser nuevo patrón
26	patron=patron+suc;	Patron + suc pasa a ser nuevo patrón por cumplir CP
27	**FinSi**;	
28	**FinHacer**;	
29	**Devolver** PATope**;**	Devuelve todos los patrones con operación cabecera ope
30	**FinFuncion;**	

Cálculo de patrones a partir de una operación cabecera

3.2.5 Verificación del cálculo de patrones

Una vez ejecutada la fase de cálculo de patrones a partir de las operaciones cabecera identificadas al principio del proceso, se comprueba si realmente se han generado patrones candidatos a partir de las mismas. Tal como se adelantó en la introducción de esta sección, puede darse el caso de que no se identifiquen patrones que satisfagan la propiedad de calidad *CP* para operaciones cabecera que sí satisfacen la propiedad de calidad *PH*. Esto sucede en algunos casos en que *CP* > *PH*, ya que si *CP* ≤ *PH* al menos las operaciones cabecera serían patrones candidatos. Nótese que también puede darse el caso de que no se hayan identificado operaciones cabecera, es decir, operaciones que cumplan la propiedad de calidad *PH*.

En cualquiera de las dos situaciones anteriores no se han localizado patrones candidatos, a partir de los cuales optimizar el uso de los recursos físicos necesarios para implementar la ruta de datos. Para solventar este escenario, se procede a la identificación de nuevas operaciones que puedan adquirir la condición de operaciones cabecera. Posteriormente, y a partir de estas nuevas operaciones cabecera, se ejecutará de nuevo el proceso de generación de patrones.

A continuación, se presenta el pseudocódigo del algoritmo de verificación con el que se comprueba si se han generado patrones a partir de las operaciones cabecera identificadas inicialmente. En caso negativo, se procede a la invocación de la función que identifica nuevas operaciones cabecera, y al cálculo de patrones para cada una de las nuevas operaciones cabecera.

	Algoritmo 3.5 – Verificación de la generación de patrones	
22	**Si** PAT = Ø **entonces**	Si no se han generado patrones a partir de las cabeceras
23	OpeInicio= **IdentificarNuevasOperacionesInicio(**SPEC);	Identificar nuevas cabeceras en la descripción conductual
24	**Para** ope en OpeInicio **hacer**	Para cada nueva operación cabecera identificada
25	PAT = PAT U **CalcularPatron**(ope);	Calcular nuevos patrones y añadirlos al conjunto de patrones
26	**FinHacer**;	
27	**FinSi**;	

Verificación de la generación de patrones

3.2.5.1 Identificación de nuevas operaciones cabecera

En el caso de no haberse identificado ningún patrón candidato a partir de las operaciones cabecera, o no haberse identificado ninguna operación cabecera, debido a las bajas ocurrencias de operaciones del mismo tipo en la especificación conductual, se hace necesario aumentar el número de ocurrencias de algunas operaciones con el fin de que puedan convertirse en nuevas operaciones cabecera. Para ello se aplica nuevamente la técnica de descomposición o fragmentación de operaciones.

Esta fase consiste básicamente en analizar cada operación de la descripción conductual actual que no cumple las propiedades de calidad *PH* y *CP*, y comprobar si descomponiendo alguna otra operación se obtiene la primera. En caso afirmativo se incrementará el número de ocurrencias de la misma con el objetivo de satisfacer ambas propiedades. Nótese que en el caso de que el valor de *CP* sea inferior al valor de *PH* (caso común) será suficiente comprobar que la operación satisface el parámetro *PH*. En el caso contrario deberá comprobarse la propiedad *CP*.

En esta sección se analiza en detalle el proceso de cálculo de nuevas operaciones cabecera, que puede dividirse en las siguientes fases:

1) Ordenación de las operaciones de la descripción conductual según su frecuencia de aparición de mayor a menor número de ocurrencias.

2) Análisis de las operaciones de la especificación conductual según el orden establecido en el paso anterior. Para cada una de ellas se realizan las siguientes acciones:

    a. Si la operación no cumple la propiedad de calidad *PH* o *CP*, es decir, no ha podido ser considerada como operación cabecera o como patrón candidato, comienza una nueva fase de exploración. Esta fase consiste en analizar todas las operaciones de la descripción, exceptuando aquellas del mismo tipo, representación y anchura que la operación en cuestión. En dicho análisis se comprueba si la operación contiene a la operación en caso de

estudio, es decir, si al descomponer o fragmentar la operación analizada se obtendría una operación similar a la operación en estudio, en cuanto al tipo, representación y anchura de los datos.

Una vez analizadas todas las operaciones, se verifica si la operación en estudio puede satisfacer las propiedades de calidad *PH* y *CP,* en caso de que se llevaran a cabo las descomposiciones ensayadas durante el análisis. Para ello se comprueba si el número de ocurrencias originales más las nuevas ocurrencias simuladas superan o igualan el valor de las propiedades de calidad *PH* y *CP* definidas por el diseñador.

En caso afirmativo se procede a descomponer o fragmentar todas las operaciones que contienen a la operación en estudio. Para cada operación que contiene a la operación en caso de estudio, la descripción conductual sufre las siguientes modificaciones:

i. Se elimina de la especificación la operación que contiene a la operación en caso de estudio.

ii. Se añade una nueva ocurrencia de la operación en estudio que inicialmente estaba contenida en la operación fragmentada.

iii. Se añaden el resto de las operaciones obtenidas de la fragmentación de la operación.

A continuación se presenta el pseudocódigo de la sección del algoritmo que implementa la función de identificación de nuevas operaciones cabecera.

	Algoritmo 3.6 – Identificación de nuevas operaciones cabecera	
1	**Funcion IdentificarNuevasOperacionesInicio(UOP)**	UOP contiene las operaciones a explorar
2	OrdenarFrecuencia(UOP);	Ordenar operaciones de UOP orden ascendente de frecuencia
3	Inicio=Ø;	Se inicializa el nuevo conjunto de operaciones cabecera
4	Calidad = Max(PH, CP)	Se selecciona el parámetro más exigente
5	**Para** ope1 en UOP **hacer**	Procesar todas las operaciones de más frecuentes a menos
6	NumDescompOpe1=0;	Número de operaciones que al ser fragmentadas dan ope1
7	OpeConOpe1=Ø;	Conjunto de operaciones que al ser fragmentadas dan ope1
8	**Si** Ocurrencias(ope1) < Calidad **entonces**	Si la operación Ope1 no cumple la propiedad de calidad
9	**Para** ope2 en {UOP - ope1} **hacer**	Comprobar posibles fragmentaciones en el resto de operaciones
10	**Si** EsPosibleDescomponer(ope2,ope1) **entonces**	Si ope2 contiene a ope1
11	NumDescompOpe1=NumDescompOpe1 + 1;	Se incrementa el número de operaciones que dan ope1
12	DOP = DOP U ope2;	Incrementa e conjunto de operaciones que dan ope1
13	**FinSi**;	
14	**FinHacer**;	
15	**Si** (Ocurrencias(ope1)+NumDescompOpe1) ≥ Calidad **entonces**	Si la operación Ope1 cumple la propiedad de calidad
16	**Para** ope2 en DOP **hacer**	Para cada ope2 que contiene a ope1
17	Descomponer(ope2,ope1,UOP);	Descomponer ope2 en ope1 y otros fragmentos a añadir a UOP
18	**FinHacer**;	
19	Inicio = Inicio U ope1	Se añade ope1al conjunto de nuevas operaciones cabecera.
20	**FinSi**;	
21	**FinSi**;	
22	**FinHacer**;	
23	**Devolver** Inicio;	
24	**FinFunción;**	

Identificación de nuevas operaciones cabecera

3.2.6 Descomposición de operaciones

En las fases de descubrimiento de nuevas operaciones cabecera y en la identificación de patrones se procede a la búsqueda y extracción de una operación concreta contenida en otras operaciones de la especificación. Con ello se pretende aumentar el número de ocurrencias de la operación buscada, bien para convertirse en operación cabecera de patrón o bien para aumentar el número de ocurrencias de un determinado patrón.

Las descomposiciones o fragmentaciones de operaciones que se ensayan en la búsqueda de nuevas ocurrencias de una determinada operación son:

1) *Descomposición de sumas* en nuevas sumas y lógica pegamento.

2) *Descomposición de multiplicaciones* en operaciones de los tipos *TUp*, *TDown*, *Rectangle*, nuevas multiplicaciones de menor anchura, sumas y lógica pegamento.

3) *Descomposición de las operaciones TUp, TDown y Rectangle* en nuevas operaciones *TUp* y *TDown*, multiplicaciones, sumas y lógica pegamento.

La simulación de las descomposiciones de operaciones y la posterior fragmentación de las mismas se realiza consultando una base de datos con todas las posibles descomposiciones. En cada momento se escoge la transformación más sencilla capaz de obtener una operación concreta a partir de otra operación de la especificación. La descomposición más sencilla es siempre la que produce un menor número de fragmentos de operación. La base de datos de posibles fragmentaciones se puede ampliar con nuevas técnicas de fragmentación en el futuro. Nótese que estas ampliaciones de la base de datos no afectan al algoritmo de síntesis propuesto.

Según la descomposición seleccionada en cada caso, se obtendrán, además de una nueva ocurrencia de la operación buscada, una serie de nuevas operaciones formadas por los fragmentos restantes de la operación descompuesta. Dichos fragmentos pasan a ser nuevas operaciones de la especificación conductual. El número, tipo y anchura de estos fragmentos o nuevas operaciones dependerá de la fragmentación realizada en cada caso.

A lo largo de la presente memoria, las referencias a fragmentos hacen alusión a operaciones no existentes en la especificación conductual original. Es decir, son nuevas operaciones que se han generado al aplicar las técnicas de descomposición o fragmentación a operaciones de la especificación original o a fragmentos de operaciones. Durante el proceso de síntesis tanto las operaciones originales como los fragmentos son considerados a todos los efectos como operaciones, y por lo tanto forman igualmente parte de la especificación conductual en cada momento concreto.

3.2.7 Conclusiones del proceso de descubrimiento de patrones

Los pasos anteriores constituyen el núcleo principal del algoritmo de detección de patrones presentado en esta memoria. Esta propuesta está basada en el algoritmo diseñado por Boyer y Moore [BoMo77] mediante el que se cuentan las ocurrencias de un determinado patrón en un espacio de exploración. En la estructura de dicho algoritmo modelo se han introducido, tal como se ha explicado anteriormente, dos modificaciones principales para implementar el algoritmo propuesto de detección de patrones:

1) *Simulación de la descomposición o fragmentación de operaciones.* Se comprueba si al introducir cambios o extensiones[4] en el espacio de búsqueda se puede aumentar el número de patrones a examinar o el número de ocurrencias de estos.

2) *Espacio de búsqueda de soluciones dinámico.* Al cumplirse la condición indicada en el punto anterior el espacio de búsqueda se torna dinámico y la probabilidad de encontrar nuevos patrones aumenta. Este hecho provoca a su vez que se optimice el uso de los recursos físicos de la ruta de datos que se seleccionarán en fases posteriores del proceso de síntesis. Principalmente se optimiza el uso de las unidades funcionales, al aumentar la reutilización de las mismas, y las conexiones entre los recursos funcionales que ejecutan cada patrón.

La modificación del espacio de exploración durante el proceso de búsqueda de patrones, podría llegar a generar una excesiva fragmentación de operaciones que, en principio, puede parecer negativa. Sin embargo, cuantos más fragmentos o nuevas operaciones haya en el GFD, más probabilidad hay de formar nuevos patrones y de reutilizar las unidades funcionales, tal como se detalla en secciones posteriores de la presente memoria. No obstante, las descomposiciones se limitan mediante las comprobaciones previas que se realizan para verificar su éxito. Este éxito viene dado por la aparición de nuevas ocurrencias de la operación buscada que la

[4] Mediante la aplicación de la técnica de fragmentación o descomposición.

convierten en una operación cabecera de patrón, o por la generación de un nuevo patrón candidato a partir de otro ya existente.

3.3 Conclusiones

A lo largo de este capítulo se ha presentado una nueva metodología de diseño basada en el uso conjunto de las técnicas de descomposición de operaciones y gestión de patrones. El objetivo de la técnica propuesta es aumentar el número de patrones existentes en la especificación conductual, así como el número de ocurrencias de los mismos. La SAN basada en patrones es capaz de obtener grandes beneficios de la utilización de esta técnica. En general, aumenta significativamente la reutilización de los recursos de la ruta de datos, reduciéndose así el desaprovechamiento de HW presente en la mayoría de las implementaciones convencionales. Y en particular, se reduce el área de los recursos funcionales, debido al mayor reuso de los mismos, el área de los recursos de almacenamiento, ya que la técnica propuesta favorece el encadenamiento de operaciones en un mismo ciclo, y el área debida al enrutado entre las unidades funcionales que ejecutan cada patrón.

En el capítulo cuatro se presenta un algoritmo de SAN que incorpora la técnica de diseño propuesta en el presente capítulo. El algoritmo realiza las fases de planificación de operaciones y selección y asignación de recursos HW tomando como unidad básica el patrón. De esta forma planifica de manera conjunta todas las operaciones de cada patrón, y selecciona y asigna el mismo HW a todas las instancias planificadas del mismo. Tal como se expondrá en el capítulo quinto, las rutas de datos sintetizadas por este algoritmo presentan reducciones significativas de área en comparación con los algoritmos convencionales de SAN.

CAPÍTULO

4

ALGORITMO DE SÍNTESIS DE ALTO NIVEL BASADO EN LA METODOLOGÍA PROPUESTA

El objetivo de este capítulo es la propuesta de un algoritmo de SAN que incorpora las técnicas de diseño basadas en la gestión de patrones y la descomposición de operaciones, con el objeto de aumentar la reutilización de los recursos HW de la ruta de datos, y por consiguiente, reducir el área de los circuitos sintetizados.

La principal diferencia entre el algoritmo de SAN propuesto y los algoritmos convencionales de síntesis radica en que el algoritmo propuesto utiliza como unidad básica el patrón, y toma decisiones de diseño (planificación de operaciones, selección y asignación de HW) de manera conjunta para todas las operaciones que forman cada ocurrencia del mismo. La reutilización de HW se consigue entre las operaciones de distintas ocurrencias de un mismo patrón que se planifican en ciclos distintos y utilizan los mismos recursos funcionales. Además del ahorro en área que se produce por el motivo anterior, también se reducen los recursos de almacenamiento debido al encadenamiento de las operaciones que conforman las ocurrencias de los patrones dentro de un mismo ciclo. Así mismo, se reducen los recursos de encaminamiento de datos entre las unidades funcionales que ejecutan cada patrón.

El algoritmo de SAN propuesto en esta tesis, parte de una especificación conductual y de unas restricciones de tiempo fijadas por el diseñador. La restricción de tiempo se refiere a la duración del ciclo de reloj y adicionalmente también a la latencia del circuito. En el caso de fijarse únicamente el tiempo de ciclo, el algoritmo calcula la latencia mínima a partir del tiempo de ejecución de la implementación monociclo, según se describe más adelante.

Una vez fijados el valor de la duración del ciclo de reloj y la latencia del circuito, el algoritmo obtiene un conjunto de patrones candidatos formados a partir de las operaciones cabecera, siguiendo la metodología propuesta en el capítulo anterior. El proceso de síntesis consiste básicamente en un bucle, y en cada iteración del mismo se ejecutan las siguientes fases:

1) *Gestión de la calidad de los patrones definidos:* se cuantifica la calidad de cada uno de los patrones con el objeto de seleccionar aquel de mayor calidad. La métrica utilizada para medir la calidad de los patrones tiene en cuenta la reutilización potencial de HW que conlleva la planificación y selección y asignación de HW de cada patrón.

2) *Selección de recursos HW*: se seleccionan las unidades funcionales necesarias para ejecutar las operaciones que forman parte del patrón seleccionado.

3) *Planificación de las operaciones del patrón:* se planifican en ciclos distintos algunas de las ocurrencias del patrón seleccionado. Cuantas más ocurrencias del patrón se planifiquen mayor será la reutilización de los recursos asignados en la fase previa. El hecho de planificar una ocurrencia de un patrón en un ciclo determinado, implica a su vez la planificación de todas las operaciones que lo forman en ese mismo ciclo de manera que dichas operaciones se ejecutarán de forma encadenada.

4) *Asignación de unidades funcionales:* una vez planificadas las ocurrencias del patrón, se le asigna a cada una de ellas un conjunto de unidades funcionales para ejecutar las operaciones del patrón. Dicho conjunto será compartido por todas las ocurrencias del patrón planificadas en la fase previa.

5) *Gestión de patrones pendientes:* por último, se realizará un ajuste del conjunto de patrones pendientes de ser procesados. Durante este proceso se eliminarán algunos de los patrones pendientes, por no tener un número mínimo de ocurrencias[1], y también pueden añadirse nuevos patrones, previa descomposición de algunas operaciones. En el conjunto de patrones pendientes de ser procesados puede estar todavía el patrón seleccionado en las fases previas, en el caso de haber quedado un número significativo de ocurrencias del mismo sin planificar, y por tanto seguir cumpliendo las propiedades de calidad impuestas.

[1] No cumplir las propiedades de calidad definidas.

4.1 Latencia del circuito

Dado que el proceso de síntesis propuesto es un proceso con restricción de tiempo, el diseñador debe fijar a priori la latencia y la longitud del ciclo de reloj del circuito.

En teoría de SAN se define la *latencia* de un circuito como el número máximo de ciclos de reloj en los que se puede ejecutar una especificación conductual dada.

En el caso de que el diseñador no proporcione el valor de la latencia, el algoritmo calculará el mismo como el cociente entre el tiempo de ejecución de la implementación monociclo de la conducta especificada[2], y la longitud del ciclo de reloj definida por el diseñador.

$$Latencia = \left\lceil \frac{\text{EjecuciónMonociclo}(SPEC)}{TiempoCiclo} \right\rceil$$

Para el cálculo del tiempo de ejecución de la especificación en una implementación monociclo, se utilizan las unidades funcionales más rápidas de entre las disponibles en la librería de diseño capaces de implementar cada una de las operaciones. Debido a ello, la latencia calculada según la definición anterior se corresponderá con el mínimo número de ciclos necesarios para ejecutar la especificación objeto de síntesis para un valor fijo de la longitud del ciclo de reloj.

4.2 Gestión de la calidad de los patrones

En esta fase del algoritmo se tiene un conjunto de patrones que han superado la propiedad de calidad *CP*, definida en el capítulo anterior como el mínimo número de ocurrencias que debe tener un patrón de la especificación conductual para adquirir la condición de patrón candidato.

[2] SPEC: Especificación conductual.

El objetivo de esta primera fase es seleccionar el patrón de mayor calidad de entre todos los que superan y cumplen la propiedad de calidad *CP*. El algoritmo de SAN con restricción de tiempo propuesto en este capítulo tiene como objetivo principal reducir el área del circuito resultante, respetando las restricciones de tiempo impuestas por el diseñador. El patrón de mayor calidad será, por tanto, aquel que potencialmente provoca una mayor reducción del área necesaria (o un mayor aprovechamiento de los recursos HW seleccionados).

Por consiguiente, la calidad de un patrón se define en función de los siguientes factores:

1) *El número de ocurrencias del patrón.*

2) *La movilidad de cada una de las ocurrencias del patrón.*

3) *El rango de movilidad de las ocurrencias del patrón (número de ciclos en que puede planificarse cada una de las ocurrencias del patrón).*

4) *Características de las operaciones del patrón.*

Definición 4.1: la calidad de un patrón es un valor cuantitativo que se corresponde con el producto del área del patrón (utilizando las unidades funcionales más rápidas de la biblioteca de diseño), el número de ocurrencias del mismo, el número de ciclos en que es posible planificar cada una de estas y la suma de las movilidades de sus ocurrencias.

Según la definición anterior, el algoritmo de síntesis propuesto utiliza para el cálculo de la calidad de un patrón la siguiente fórmula:

$$\text{Calidad}(Patron) = \text{Area}(Patron) \times n \times \left\| \bigcup_{i=1}^{n} \text{Rango}(P_i) \right\| \times \sum_{i=1}^{n} \text{Movilidad}(P_i) \quad \text{siendo:}$$

1) *Patron: Conjunto de operaciones encadenadas que constituyen un patrón y han superado la propiedad de calidad CP.*

2) *Area(Patron): Suma de las áreas de las unidades funcionales más rápidas presentes en la biblioteca de diseño capaces de ejecutar las operaciones de Patron. La decisión de tener en cuenta el área de las unidades funcionales más rápidas se ha tomado por coherencia con el cálculo del tiempo de ejecución del patrón, en el que se contabilizaba el tiempo de ejecución de estas mismas unidades. Nótese que el valor de Area(Patron) aumenta con el número de operaciones del patrón y la complejidad de las mismas.*

3) *n: Número de ocurrencias reales de Patron. Se contabilizan únicamente las ocurrencias que pueden planificarse, es decir aquellas cuyo rango de movilidad es distinto del conjunto vacío, según se explica más adelante.*

4) *Pi: Ocurrencia de Patron.*

5) *Rango(Pi): Conjunto de ciclos de reloj en los que se pueden ejecutar todas las operaciones de la ocurrencia Pi del patrón.*

6) *Movilidad(Pi): Número de ciclos en los que se pueden ejecutar todas las operaciones de la ocurrencia Pi del patrón.*

En las siguientes secciones se explica cómo influye cada uno de estos factores en la calidad del patrón candidato, y por tanto cómo repercute su planificación y la selección y asignación de HW de las operaciones del mismo en la implementación resultante.

4.2.1 Número de ocurrencias del patrón

Uno de los factores que influyen en la calidad del patrón es el número de ocurrencias que tiene el patrón en la especificación conductual en un determinado momento del proceso de síntesis. Este factor influye en la calidad del patrón porque todas las ocurrencias del patrón planificadas se ejecutarán

en las mismas unidades funcionales. Por ello, un mayor número de ocurrencias del patrón provocará potencialmente:

1) *Una mayor reutilización de las unidades funcionales asignadas a dicho patrón.*

2) *Una mayor reducción del área necesaria para implementar la ruta de datos al compartir las ocurrencias del patrón las mismas unidades funcionales.*

En general, se puede afirmar que cuanto mayor sea el número de ocurrencias de un patrón, mayor será la calidad del mismo. Sin embargo, la planificación de algunas ocurrencias puede resultar imposible debido a que su rango de movilidad es el conjunto vacío. Esto sucede cuando no existe ningún ciclo de reloj en el que sea posible la ejecución de la ocurrencia del patrón sin sobrepasar el tiempo de ciclo. Aunque la propia definición de patrón candidato garantiza que su tiempo de ejecución es menor o igual que el tiempo de ciclo, pueden existir operaciones sucesoras o predecesoras ya planificadas en ciertos ciclos en los que resulta imposible la ejecución del patrón sin sobrepasar la duración del ciclo. Cuando no existe ningún ciclo en el que pueda planificarse una determinada ocurrencia, ésta deja de contar como una ocurrencia válida del patrón. Las ocurrencias que se contabilizan para medir la calidad del patrón son aquellas que pueden planificarse en algún ciclo de reloj sin sobrepasar el tiempo de ciclo. A estas ocurrencias las denominamos ocurrencias reales, y su número puede ser diferente a lo largo del proceso de síntesis.

4.2.2 Movilidad de las ocurrencias del patrón

En la teoría tradicional de la SAN el concepto de movilidad está relacionado con las operaciones y los ciclos de reloj en que éstas pueden ejecutarse. La movilidad de las operaciones se define en relación a dos posibles estrategias diferentes de planificar una operación:

1) *Planificación ASAP(As Soon As Possible):* la estrategia que sigue esta propuesta es ejecutar las operaciones tan pronto como sea posible. Esto es, la operación se asocia al primer ciclo de reloj después de que todas sus operaciones predecesoras hayan sido planificadas.

2) *Planificación ALAP (As Late As Possible):* la estrategia que sigue esta propuesta es ejecutar las operaciones tan tarde como sea posible. Esto es, dado un tiempo límite definido, la operación se asocia al último ciclo de reloj antes de que su resultado sea utilizado por otras operaciones.

Cada una de las dos estrategias anteriores define un ciclo de reloj en el cual comienza la ejecución de la operación. En función de esos dos ciclos se define el rango de movilidad de una operación como el conjunto de ciclos situados entre el ciclo asignado por la estrategia ASAP y el asignado por la estrategia ALAP. Y se define la movilidad de dicha operación como el número de ciclos del conjunto anterior.

Por definición del proceso de síntesis propuesto, todas las operaciones de un patrón se ejecutan de forma encadenada en un solo ciclo de reloj. Aplicando esta idea y los conceptos de estrategia ASAP, estrategia ALAP, rango de movilidad y movilidad de una ocurrencia de un patrón, se pueden expresar las definiciones que aparecen a continuación.

Definición 4.2: El rango de movilidad de una ocurrencia de un patrón es el conjunto de los ciclos en los cuales se pueden ejecutar de manera encadenada todas las operaciones de dicha ocurrencia.

Nótese que una ocurrencia de un patrón sólo podrá planificarse en los ciclos incluidos en su rango de movilidad. Sin embargo, si una determinada ocurrencia no resulta planificada, posteriormente alguna de sus operaciones podría planificarse en alguno de los ciclos no incluidos en el rango de movilidad de la ocurrencia. Esto sucede cuando la operación forma parte de una ocurrencia de un patrón distinto, o bien en la fase final del algoritmo de SAN propuesto, en que no es posible identificar nuevos patrones y se pasa a planificar y asignar las operaciones restantes.

Definición 4.3: La movilidad de una ocurrencia de un patrón es el número de ciclos incluidos en su rango de movilidad.

El concepto de rango de movilidad y el concepto de movilidad está asignado a cada una de las ocurrencias de un patrón y no al patrón en sí mismo. Cada ocurrencia de un patrón puede tener un rango de movilidad y una movilidad distintos que dependerán de las operaciones predecesoras y sucesoras de dicha ocurrencia.

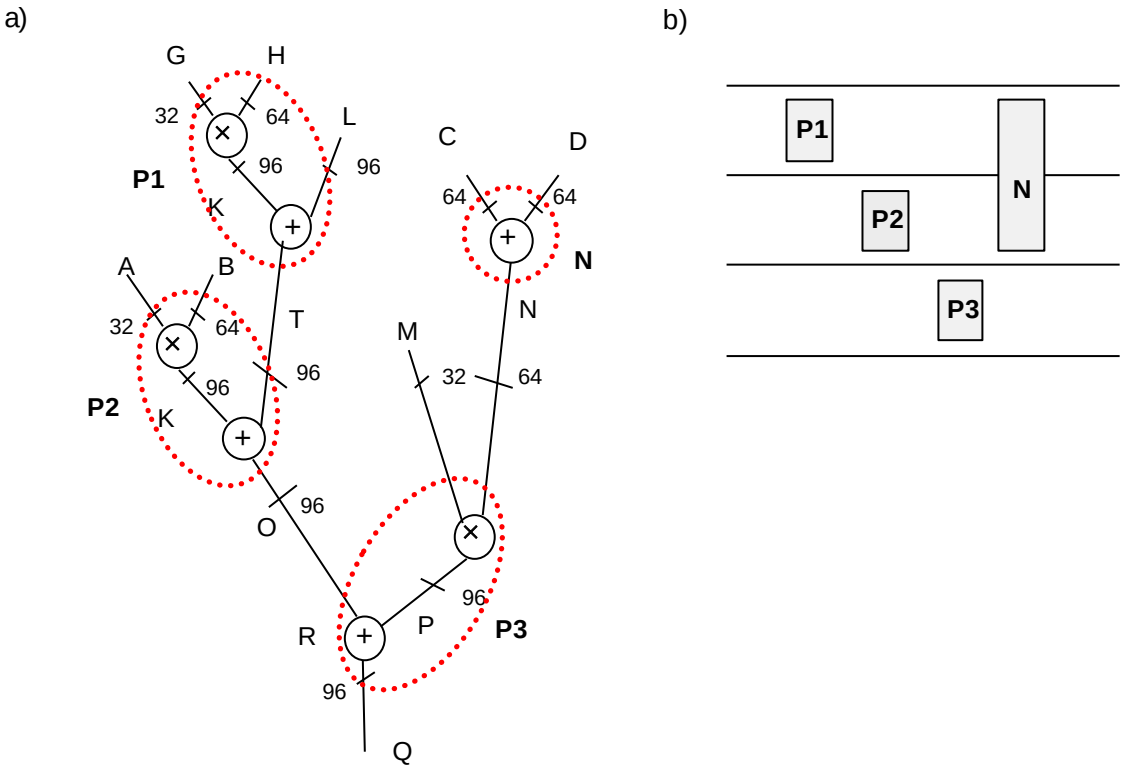

Figura 4.1 *a) GFD b) Movilidades de las ocurrencias del patrón P y de la operación N:*

1. *Ocurrencia P_1: Rango = [1,1] y movilidad = 1.*
2. *Ocurrencia P_2: Rango = [2,2] y movilidad = 1.*
3. *Ocurrencia P_3: Rango = [3,3] y movilidad = 1.*
    a. *Operación P: Rango = [2,3] y movilidad = 2.*
    b. *Operación Q: Rango = [3,3] y movilidad = 1.*
4. *Operación N: Rango = [1,2] y movilidad = 2.*

En la figura 4.1 se puede apreciar la diferencia entre el concepto de movilidad de una ocurrencia de un patrón y el concepto de movilidad de una operación. Nótese que la operación multiplicación que calcula el valor de la

variable *P* tiene una movilidad de dos ciclos de reloj, en concreto su rango de movilidad incluye los ciclos 2 y 3. La movilidad de la operación suma que calcula el valor de la variable *Q* es un ciclo de reloj y su rango de movilidad incluye únicamente el ciclo 3. Ambas operaciones, la multiplicación que produce el resultado *P* y la suma que obtiene el valor de *R*, conforman la ocurrencia *P3* del patrón del ejemplo. Por simplicidad, de aquí en adelante nos referiremos a las operaciones de una especificación utilizando los nombres de sus operandos de salida.

La movilidad de la ocurrencia *P3* del patrón es un ciclo de reloj y su rango de movilidad incluye únicamente el ciclo 3. Por lo tanto, la movilidad de la operación *P* es distinta si tenemos en cuenta únicamente la operación, o el patrón del que forma parte. En general, la movilidad de un patrón es menor que las movilidades individuales de las operaciones que lo forman.

La reutilización máxima o mayor aprovechamiento de las unidades funcionales asignadas a un patrón ocurrirá cuando las movilidades de todas las ocurrencias permitan su planificación en ciclos distintos. Por tanto, a mayores movilidades de las ocurrencias, mayor es la probabilidad de que éstas se puedan planificar en ciclos distintos.

4.2.3 Rango de movilidad de las ocurrencias

Se introduce en la fórmula del cálculo de la calidad de un patrón el rango de movilidad de las ocurrencias del mismo ya que está directamente relacionado con la reutilización de los recursos HW por parte de las ocurrencias del patrón. Si bien es cierto que cuanto mayor sean las movilidades de las ocurrencias, mayor será potencialmente la reutilización de HW, al no tenerse en cuenta los rangos de las movilidades podría darse el caso de que los rangos incluyeran conjuntos de ciclos similares. En dicho caso, las ocurrencias del patrón que pudieran planificarse en ciclos diferentes serían pocas, y la reutilización de HW muy limitada.

La unión de los rangos de las movilidades de las ocurrencias es un factor importante a la hora de medir la calidad de un patrón, siendo mayor su calidad cuanto mayor sea el número de ciclos distintos en los que pueden planificarse sus ocurrencias.

La cardinalidad de la unión de los rangos de las movilidades junto con la movilidad de las ocurrencias de un patrón proporcionan una información muy valiosa sobre la posible distribución de las ocurrencias del patrón en los ciclos de reloj. Por tanto, son un indicativo importante de la calidad de los patrones.

La figura 4.2 muestra dos GFD muy similares (GFD-1 y GFD-2). Las únicas diferencias se refieren al cálculo de la variable *P* y a la existencia de una operación más, *W*, en el segundo GFD. Ambos GFD contienen tres ocurrencias de un mismo patrón formado por una multiplicación de 32x64 bits y una suma encadenada de 96 bits. Sintetizando las especificaciones conductuales correspondientes con la siguiente restricción de unidades funcionales: un sumador de 96 bits, un sumador de 64 bits y un multiplicador de 32x64 bits, las ocurrencias del patrón presentan movilidades y rangos de movilidad distintos en ambos GFD, lo que es debido a las diferencias señaladas anteriormente. En particular, las movilidades de las ocurrencias son:

- En el GFD-1 las ocurrencias *P1* y *P2* tienen movilidad dos y la ocurrencia *P3* tiene movilidad uno. Véase figura 4.2 a) y b).
- En el GFD-2 las ocurrencias *P1*, *P2* y *P3* tienen movilidad dos. Véase figura 4.2 c) y d).

La suma de las movilidades de las tres ocurrencias es mayor en el GFD-2 (6 ciclos) que en el GFD-1 (5 ciclos).

En los dos GFD del ejemplo, la unión de los rangos de movilidad de las tres ocurrencias es el conjunto formado por los ciclos 1, 2, 3 y 4. Y por tanto, su cardinalidad es 4 en ambos casos.

Finalmente, la calidad del patrón *P* es mayor en el GFD-2 debido a que la suma de las movilidades de sus ocurrencias es mayor, siendo el número de ocurrencias del patrón, su área y la cardinalidad de la unión de los rangos de movilidad igual en ambos casos.

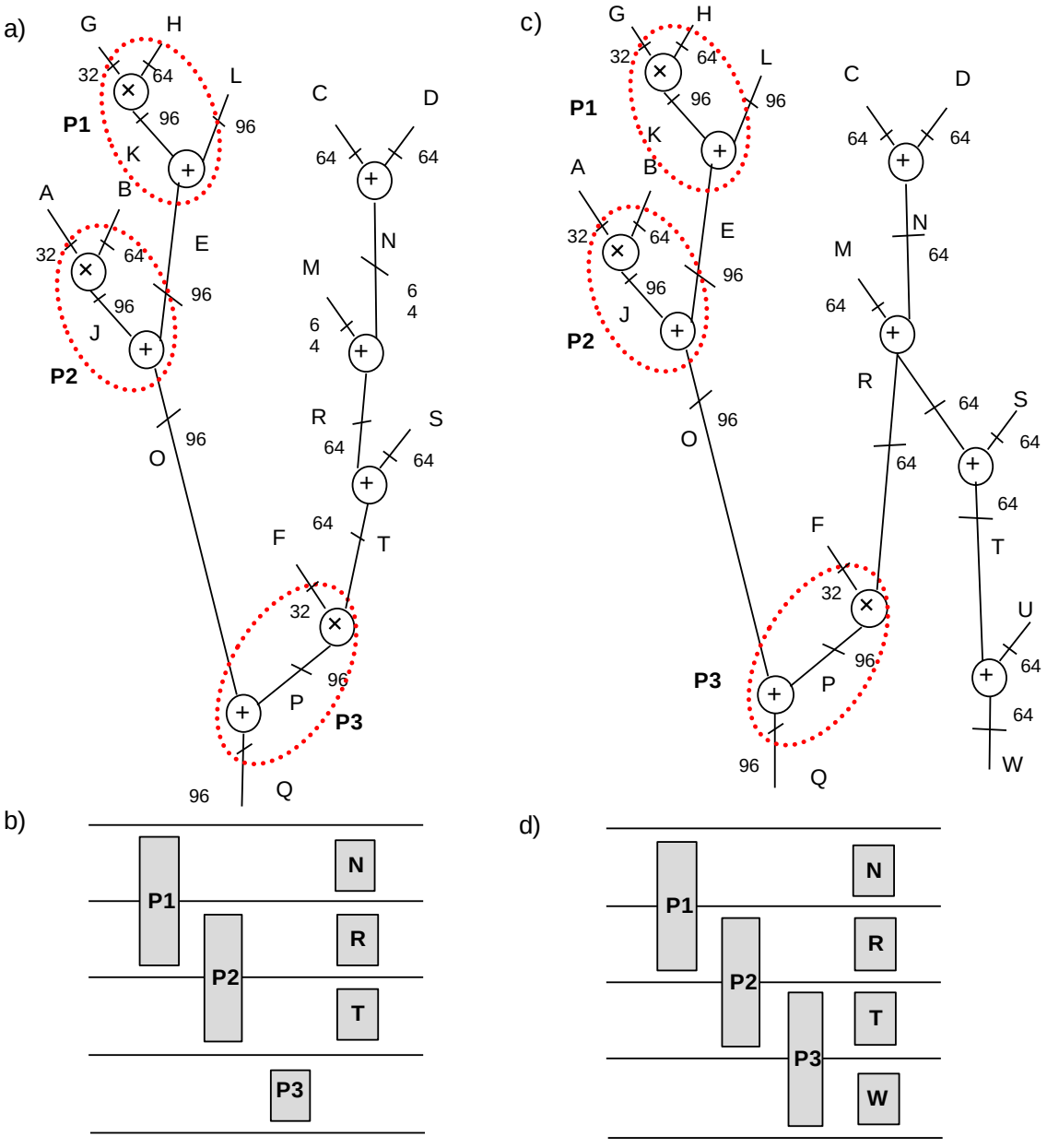

Figura 4.2 *Calidad de un mismo patrón en dos GFD distintos.*

a) GFD-1 con patrón P y ocurrencias P1, P2 y P3.

b) En GFD-1 ocurrencia P3: Rango = [3,3] y movilidad = 1.

c) GFD-2 con patrón P y ocurrencias P1, P2 y P3.

d) Ocurrencia P3: Rango = [2,3] y movilidad = 2.

4.2.4 Características de las operaciones del patrón

Las operaciones o elementos nucleares que forman parte de un patrón pueden ser de tres tipos:

1) *Operaciones de la especificación original.*

2) *Operaciones resultantes del proceso de descomposición de una operación multiciclo.*

3) *Operaciones resultantes del proceso de descomposición de una operación monociclo.*

La mayoría de las especificaciones conductuales presentan dependencias de datos entre sus operaciones. Estas dependencias de datos ejercen una influencia decisiva en la calidad de los circuitos resultantes de la SAN. Tomado un GFD con dos operaciones con dependencias de datos, para que una operación utilice como operando de entrada la salida de la otra operación, es necesario utilizar un recurso de almacenamiento para guardar el operando durante el tiempo que transcurre entre el ciclo en que se ejecuta la primera y el ciclo donde empieza a ejecutarse la segunda. Incluso si los ciclos fueran contiguos, sería necesario almacenar ese operando durante un ciclo. Cuantas más operaciones con dependencias de datos tenga el GFD más recursos de almacenamiento serán necesarios para almacenar los resultados intermedios que utilizan algunas de las operaciones como operandos de entrada.

Como hemos indicado anteriormente todas las operaciones de un patrón se ejecutan de forma encadenada en un solo ciclo de reloj. Por consiguiente, cuantas más operaciones o fragmentos de operación formen parte del patrón, mayor será la reducción de área en cuanto a recursos de almacenamiento, ya que no es necesario salvar los resultados intermedios que se calculan en el mismo ciclo.

Adicionalmente, se reduce el coste de elementos de interconexión ya que el HW asignado a un determinado patrón se utiliza únicamente para ejecutar ocurrencias del mismo, y por tanto no requiere de elementos de encaminamiento de datos, como por ejemplo multiplexores, entre las diferentes unidades funcionales que lo implementan.

Por otro lado, no sólo el número de operaciones influye en la calidad del patrón, sino que también la complejidad de las operaciones tiene una influencia directa en su calidad. Cuanto más complejas sean las operaciones del patrón (en función de su tipo y anchura de datos) más costosos serán los recursos funcionales necesarios para ejecutarlas. Por este motivo, resulta evidente que los patrones formados por operaciones complejas ofrecen un mayor beneficio en cuanto a la reutilización de unidades funcionales que los formados por otras operaciones más sencillas, garantizando así la reutilización de los módulos más costosos.

Los dos factores mencionados anteriormente como claves para medir la calidad de un patrón, número de operaciones y complejidad de las mismas, quedan recogidos en el área del patrón, definido previamente como la suma de las áreas de las unidades funcionales más rápidas presentes en la biblioteca de diseño capaces de ejecutar las operaciones del patrón.

Se ha tenido en cuenta el área de las unidades funcionales más rápidas por coherencia con el cálculo del tiempo de ejecución del patrón. Sin embargo, esto no implica que sean éstas las unidades funcionales que finalmente conformen la ruta de datos. Al finalizar el proceso de SAN se lleva a cabo una optimización del circuito resultante, cuya finalidad es sustituir algunas de las unidades funcionales más costosas de la ruta de datos por otras más lentas y de menor área, respetando siempre las restricciones de tiempo impuestas por el diseñador.

4.2.5 Ejemplo del cálculo de la calidad de un patrón

En la figura 4.3 se muestran dos GFD distintos en los que se ha identificado el mismo patrón formado por una multiplicación de 32x64 bits y una suma encadenada de 96 bits. A continuación vamos a proceder al cálculo de la calidad del patrón en ambos GFD.

En el GFD-1 el patrón identificado tiene 4 ocurrencias, cada una de ellas con una movilidad igual a 1. En este caso la suma de las movilidades de las ocurrencias es 4, y la cardinalidad de la unión de los rangos de movilidad es 2. Por tanto la calidad del patrón es:

$$\text{Calidad}(Patron) = \text{Area}(Patron) \times 4 \times 2 \times 4 = 32 \times \text{Area}(Patron)$$

En el GFD-2 el patrón en cuestión tiene 3 ocurrencias, pero cada una de ellas tiene una movilidad de 2 ciclos. La suma de las movilidades de las ocurrencias es 6 y la cardinalidad de la unión de los rangos de movilidad 4. La calidad del patrón en este caso es:

$$\text{Calidad}(Patron) = \text{Area}(Patron) \times 3 \times 4 \times 6 = 72 \times \text{Area}(Patron)$$

Dado que en ambos GFD el área del patrón es igual, ya que se trata de patrones formados por las mismas operaciones, el patrón de mayor calidad es el perteneciente al GFD-2.

4.2.6 Implementación algorítmica de la gestión de la calidad de los patrones

Una vez definido el conjunto de patrones candidatos *PAT*, comienza el procesamiento de los mismos hasta que el conjunto quede vacío. Esta situación se dará cuando se hayan procesado todos los patrones, tanto los existentes al comienzo de esta fase como los nuevos que se puedan definir a lo largo del proceso de SAN.

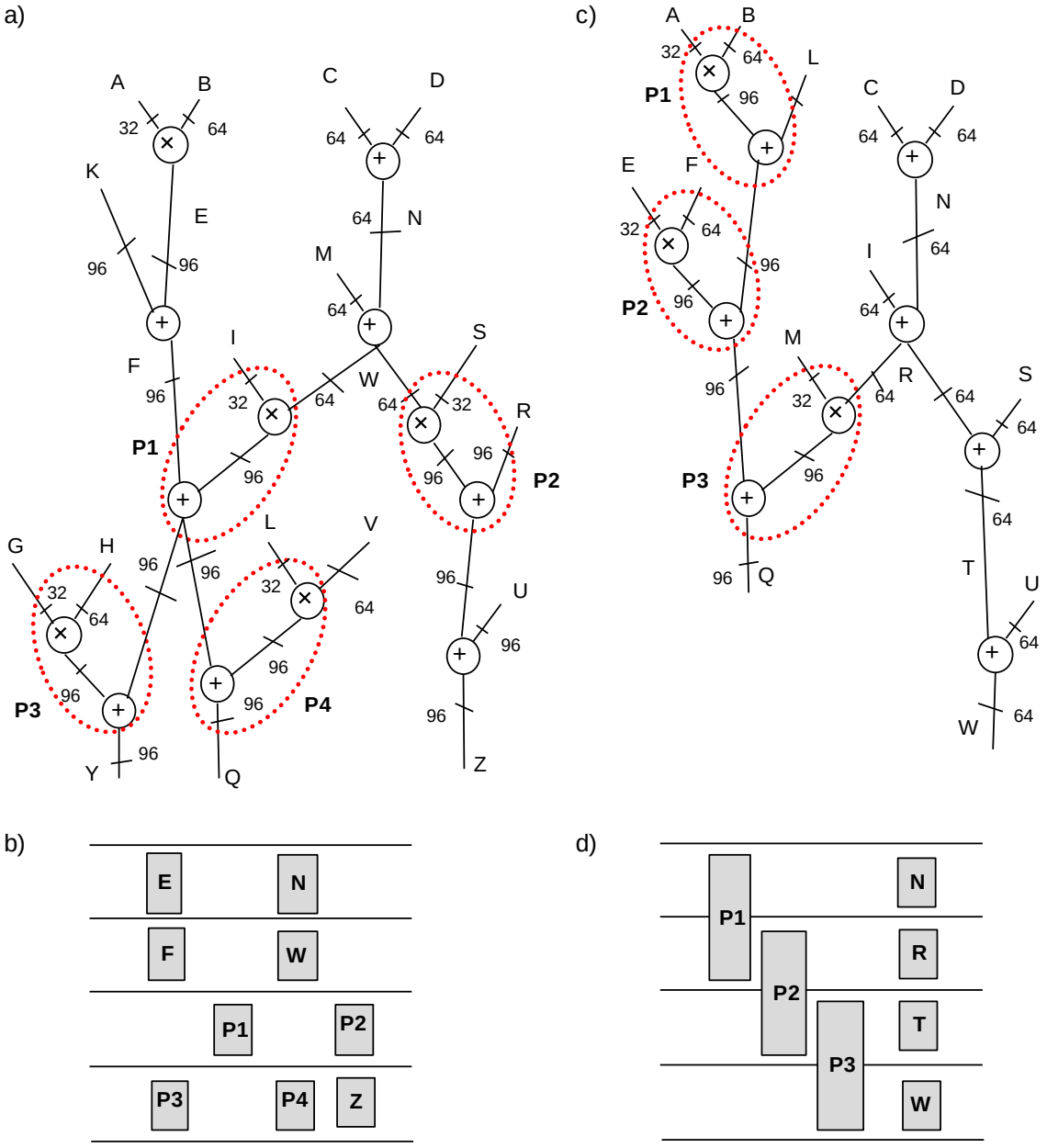

Figura 4.3 *Factores que influyen en la calidad de los patrones.*

a) GFD-1 con ocurrencias del patrón P1, P2, P3 y P4.

b) Ocurrencias en GFD-1.

1. *Ocurrencia P1 y P2: rango = [3,3] y movilidad = 1.*
2. *Ocurrencia P3 y P4: rango = [4,4] y movilidad = 1.*

c) GFD-2 con ocurrencias del patrón P1, P2, P3 y P4.

d) Ocurrencias en GFD-2.

1. *Ocurrencia P1: rango = [1,2] y movilidad = 2.*
2. *Ocurrencia P2: rango = [2,3] y movilidad = 2.*
3. *Ocurrencia P3: rango = [3,4] y movilidad = 2.*

A continuación se presenta el pseudocódigo del algoritmo de gestión de la calidad de los patrones.

	Algoritmo 4.1 – Gestión de la calidad de los patrones	
1	**Mientras** PAT ≠ Ø **hacer**	Mientras haya patrones activos se procesan
2	PatronMasCalidad = Ø	Se inicializa el patrón de más calidad como conjunto vacío. Calidad(Ø) = ∞
3	**Para** Patron **en** PAT **hacer**	Se tratan de forma secuencial todos los patrones activos
4	**Si** Calidad(Patron) > Calidad(PatronMasCalidad) **entonces**	Si la calidad del patrón actual es de mayor calidad
5	PatronMasCalidad=Patron;	Se redefine el patrón de más calidad al actual
6	**FinSi**;	
7	**FinHacer**;	
8	**FinMientras**	

Gestión de la calidad de los patrones

4.3 Selección de las unidades funcionales

Una vez determinado el patrón de mayor calidad se procede a seleccionar las unidades funcionales que ejecutarán las operaciones que forman parte del patrón. Para una operación concreta puede haber varias unidades funcionales en la biblioteca de diseño utilizada que puedan implementar la operación. De entre todas ellas, el algoritmo selecciona la que ejecute la operación en el menor tiempo posible, esto es, la más rápida. De esta forma se contribuirá a satisfacer las restricciones de tiempo definidas por el diseñador. Como se ha comentado anteriormente, algunas de estas unidades funcionales serán sustituidas al final del proceso de SAN por otras de menor área.

4.3.1 Implementación algorítmica de la selección de unidades funcionales

La función de selección de unidades funcionales para las operaciones de un patrón se realiza de manera trivial, recorriendo todas las operaciones del patrón y seleccionando, para cada operación, la unidad funcional más rápida capaz de ejecutarla.

A continuación se presenta el pseudocódigo del algoritmo de selección de unidades funcionales para las operaciones de un patrón.

	Algoritmo 4.2 – Selección de UF para operaciones del patrón	
1	**Funcion SeleccionUF(Patron)**	Selección de UF para las operaciones del patrón de más calidad
2	UFPatron=Ø;	Conjunto de UF asignadas a las operaciones del patrón
3	**Para** ope en Patron **Hacer**	Se procesan todas las operaciones del patrón
4	UFop = SeleccionUFmasRapida(ope);	Selección de la UF más rápida de la librería para ejecutar ope
5	UFPatron=UFPatron U {UFop}	Se añade la UF al conjunto de UF del patrón
6	**FinHacer**;	
7	**Devolver** UFPatron;	La función devuelve el conjunto de UF
8	**FinFunción;**	

Selección de UF para las operaciones del patrón

4.4 Planificación de operaciones

Al llegar a esta fase del proceso de síntesis se tienen seleccionadas las unidades funcionales necesarias para ejecutar las operaciones del patrón. Se procederá a seleccionar los ciclos de reloj en los que se ejecutarán algunas de las ocurrencias del patrón. Las ocurrencias así planificadas quedan asignadas a las unidades funcionales seleccionadas previamente.

4.4.1 Características de la planificación de las ocurrencias de un patrón

Tal como se adelantó en las secciones anteriores, cada ocurrencia del patrón se planifica como una unidad y por lo tanto sus operaciones se ejecutan encadenadas en un solo ciclo de reloj. En consecuencia, el algoritmo de planificación propuesto difiere de los algoritmos convencionales en que trata las ocurrencias de un patrón como una única entidad, independientemente del número y tipo de las operaciones que lo forman.

Dado que el patrón seleccionado superó la propiedad de calidad *CP*, existe al menos un número de ocurrencias del patrón igual a *CP* por planificar. El objetivo del algoritmo propuesto es intentar planificar todas las ocurrencias del patrón en ciclos distintos. De esta forma, la reutilización de las unidades funcionales seleccionadas en la fase anterior será máxima.

Sin embargo, no siempre resulta posible planificar todas las ocurrencias del patrón en ciclos distintos, bien porque el número de ocurrencias es superior

al valor de la latencia, o bien debido a que los rangos de las movilidades de algunas ocurrencias del patrón se solapan entre sí. En estos casos, la reutilización de los mismos recursos funcionales para ejecutar todas las ocurrencias del patrón resulta imposible.

Por otro lado, la instanciación de más recursos funcionales para ejecutar las ocurrencias del patrón que pudieran planificarse en el mismo ciclo puede provocar un desaprovechamiento significativo de HW. Por este motivo, el algoritmo propuesto sólo planifica como máximo una ocurrencia del patrón seleccionado por ciclo. Y dado que podrían quedar algunas ocurrencias del patrón sin planificar, éste continuaría siendo un patrón candidato[3], siempre que siga cumpliendo el parámetro de calidad *CP*. Posteriormente, podría volver a ser seleccionado como patrón candidato de mayor calidad y volverían a planificarse algunas o todas las ocurrencias restantes del mismo. En este caso, podrían coincidir en el mismo ciclo con otras ocurrencias del mismo patrón planificadas en iteraciones anteriores del algoritmo.

A continuación se analiza parte del proceso de síntesis del GFD de la figura 4.4, con el fin de mostrar cómo procede el algoritmo en el caso de existir varias ocurrencias de un patrón cuya planificación en ciclos de reloj diferentes resulta imposible.

En el GFD de la figura 4.4 se identifica un patrón formado por dos operaciones: una multiplicación de 32x64 bits y una suma encadenada de 96 bits. Dicho patrón tiene tres ocurrencias y supera la propiedad de calidad *CP* definida por el diseñador, al haberse fijado su valor en 3 (*CP*=3). Tras ser seleccionado como el patrón de mayor calidad, el algoritmo selecciona de la biblioteca de diseño el multiplicador y el sumador más rápidos capaces de ejecutar las operaciones del patrón. A continuación tiene lugar la planificación de sus ocurrencias.

[3] Este patrón candidato estaría formado por las ocurrencias no planificadas.

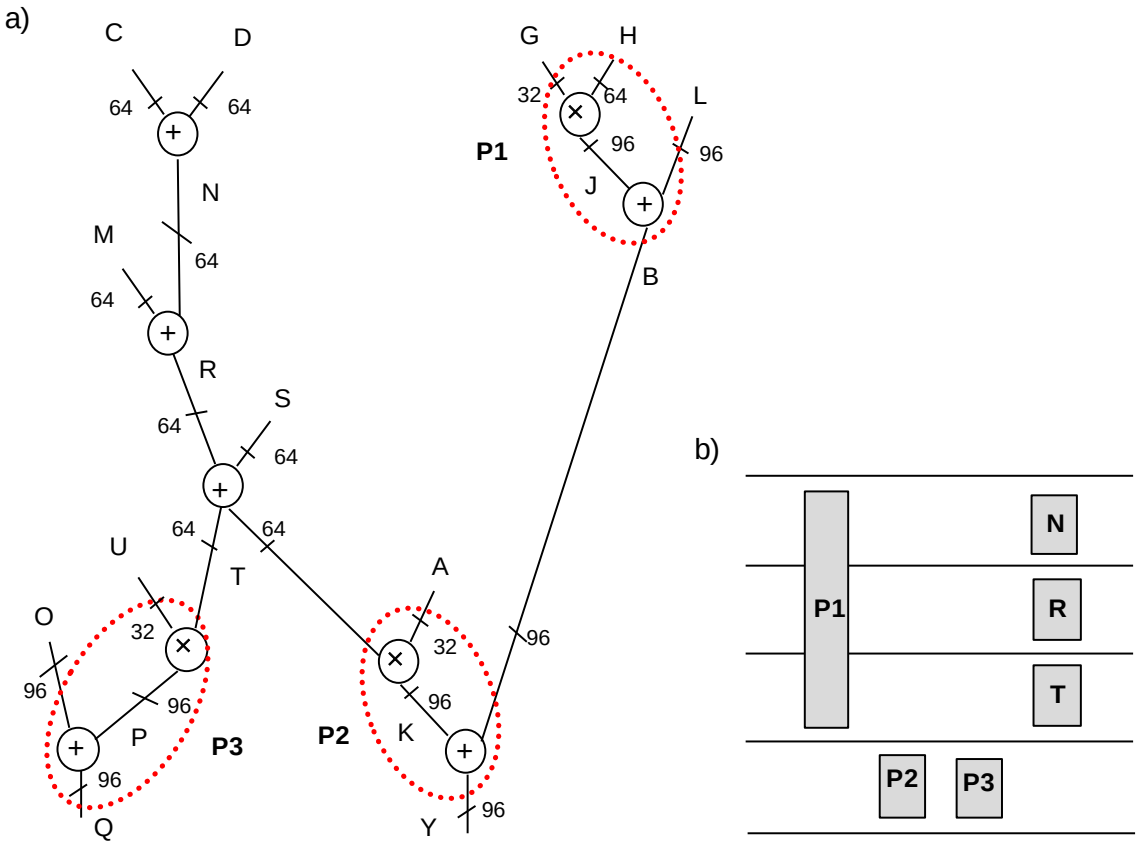

Figura 4.4 *Coincidencia de dos ocurrencias del mismo patrón en un mismo ciclo de reloj.*

a) GFD con ocurrencias del patrón P1, P2 y P3.

b) Ocurrencias P2 y P3 en el mismo ciclo:

1. *Ocurrencia P2: Rango = [4,4] y movilidad = 1.*
2. *Ocurrencia P3: Rango = [4,4] y movilidad = 1.*

Los rangos de movilidad de las ocurrencias *P2* y *P3* coinciden, y en ambos casos, el rango de movilidad es el conjunto formado únicamente por el ciclo 4. Por tanto, sólo será posible planificar una de ellas en el ciclo 4. Además, la ocurrencia *P1* también resultará planificada en alguno de los ciclos de su rango de movilidad (ciclos del 1 al 3).

En este caso, el número de ocurrencias planificadas en diferentes ciclos es dos, valor que es inferior al valor del parámetro de calidad *CP*. Nótese que el parámetro de calidad *CP* no garantiza la planificación de un número concreto de ocurrencias, ni tampoco una reutilización mínima de los recursos HW seleccionados para ejecutar las operaciones del patrón. El parámetro de

calidad *CP* representa un requisito mínimo que deben cumplir los patrones candidatos, descartando aquellos que potencialmente llevan a una reutilización menor de los recursos HW necesarios para ejecutar sus operaciones.

De este ejemplo se puede concluir que, desde el punto de vista de la reutilización de los recursos HW, las ocurrencias reales del patrón seleccionado se han visto reducidas en una unidad, de tres a dos. Nótese que este hecho ya se tuvo en cuenta al calcular la calidad del patrón, en concreto, al contabilizarse el número de ciclos en que podrían ejecutarse las ocurrencias del patrón (unión de los rangos de movilidad de las ocurrencias) y las movilidades de las mismas. El hecho de que este patrón haya sido seleccionado como el de mayor calidad, indica que muy probablemente la planificación de las ocurrencias del resto de los patrones produciría un desaprovechamiento de HW mayor.

En la siguiente sección se analiza en detalle el algoritmo de planificación utilizado en el proceso de síntesis propuesto.

4.4.2 Algoritmo de planificación

El algoritmo planteado está basado en el algoritmo de planificación dirigida por fuerzas FDS (Force Direct Scheduling) [PaKn89]. A la adaptación del FDS que se plantea en la presente memoria, se la ha denominado PFDS (Pattern Force Direct Scheduling).

Definición 4.4: Se define el algoritmo de planificación PFDS (Pattern Force Direct Scheduling) como la adaptación del algoritmo FDS en el que se planifican de manera conjunta las operaciones de las ocurrencias de un patrón.

Para adaptar este algoritmo clásico de planificación al proceso de síntesis propuesto, se ha introducido la siguiente ampliación: todas las operaciones que forman parte de una ocurrencia de un patrón son

consideradas como una unidad indivisible por el algoritmo y son planificadas de forma conjunta en el mismo ciclo de reloj.

A continuación se presentan algunas definiciones adicionales necesarias para desarrollar el algoritmo de planificación.

Definición 4.5: Para cada ocurrencia *Pi* de un patrón *Pat*, se define:

1) *PSi como el ciclo en el que se planifica la ocurrencia Pi en caso de aplicar una técnica de planificación ASAP.*

2) *PLi como el ciclo en el que se planifica la ocurrencia Pi en caso de aplicar una técnica de planificación ALAP.*

3) *MV(Pi) como la función que calcula la movilidad de la ocurrencia Pi según se definió en la sección 4.2.2.*

Cada una de las operaciones que forman parte de un patrón tiene su propia movilidad que no tiene por qué depender de la movilidad de las otras operaciones del patrón y, por tanto, tampoco tiene por qué coincidir con ellas. El patrón tiene también su propia movilidad que incluye un conjunto de ciclos igual o menor que el de las movilidades de las operaciones que lo forman.

Definición 4.6: Para cada ocurrencia *Pi* de un patrón se define *Pred(Pi)* como el conjunto de operaciones predecesoras inmediatas de la ocurrencia *Pi*. El conjunto *Pred(Pi)* se calcula como la unión de los conjuntos de las operaciones predecesoras inmediatas de cada una de las operaciones que forman la ocurrencia, exceptuando las operaciones que forman parte del patrón.

En la figura 4.5 se presenta un GFD al que se le aplica un algoritmo de SAN basado en patrones en el caso a) y un algoritmo convencional de SAN en el b). En ambos casos se muestran coloreadas las operaciones predecesoras de la ocurrencia del patrón *P3* en el GFD de la figura a) y de la operación *P3+* (operación suma de la ocurrencia *P3* en el GFD anterior) en la figura b).

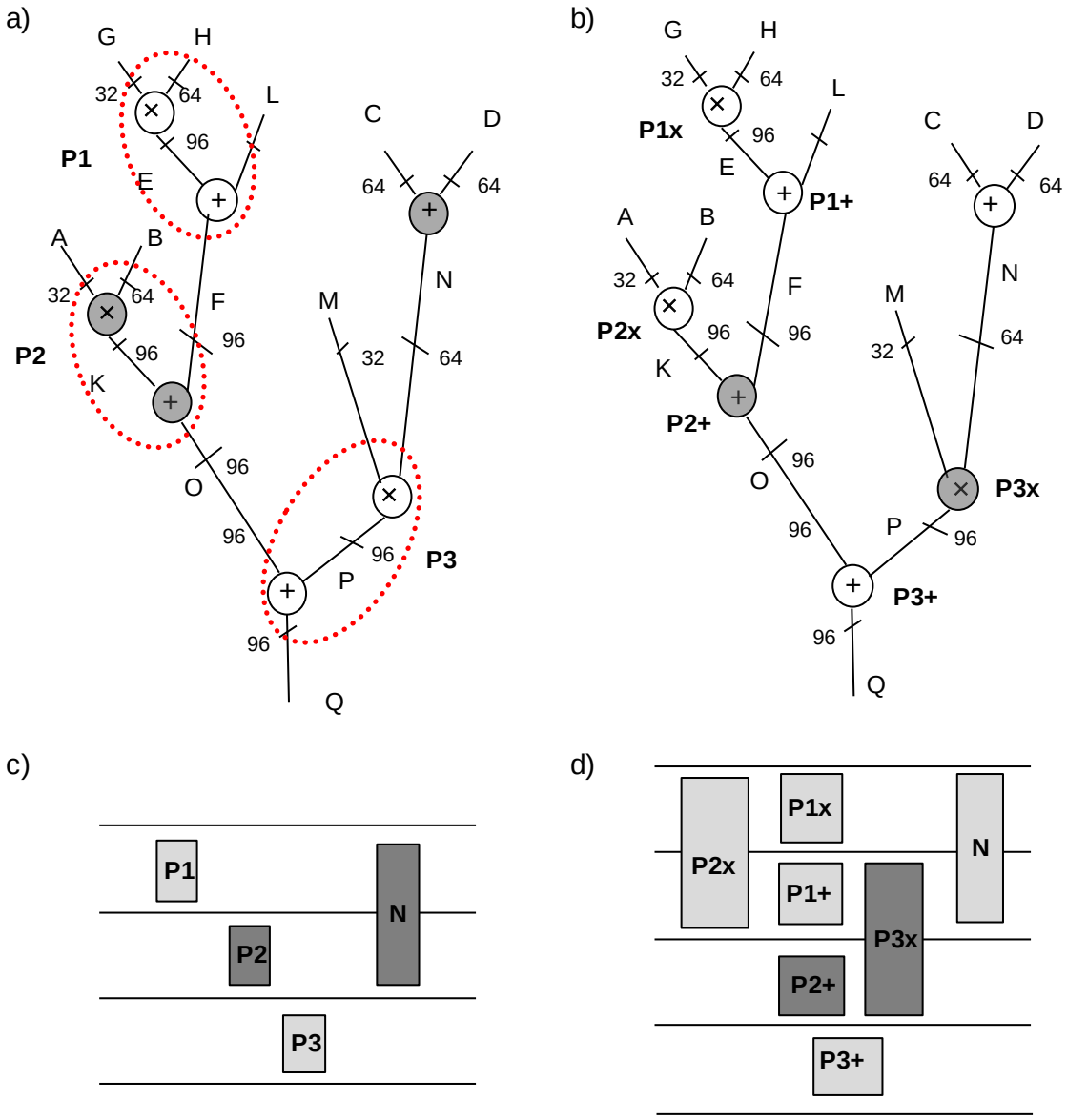

Figura 4.5 *Concepto de predecesoras en SAN con y sin patrones para un mismo GFD.*

a) SAN con patrones: Predecesoras de P3: Operación N y ocurrencia P2.

b) SAN sin patrones: Predecesoras de:

1) P3x: Operación N.

2) P3+: Operación P3x y operación P2+.

c) Rango y movilidad de las ocurrencias del patrón de la figura a).

d) Rango y movilidad de las operaciones de las ocurrencias de la figura b).

Aplicando el algoritmo de SAN basado en patrones al GFD de la figura 4.5 a) se tiene que:

- *Pred(P3) = {N , P2}*

Y aplicando un algoritmo convencional de SAN al GFD de la figura 4.5 b) se tiene:

- *Pred(P3x) = {N}*
- *Pred(P3+) = {P, O}*

Definición 4.7: Siendo *Ciclos* el conjunto de ciclos de la planificación y C_j el ciclo que ocupa el orden *j* de la planificación, se define PS_{ij} como un indicador binario que indica si la ocurrencia *Pi* del patrón *Pat* está asignada al ciclo *j*. Si PS_{ij}=1, indica que la ocurrencia del patrón está planificada en el ciclo C_j y si PS_{ij}=0 indica que no ha sido planificada en el ciclo C_j.

$\forall\ C_j \in Ciclos$ y $\forall\ P_i \in$ Pat, se cumple:

1) *PS_{ij} = 1 si la ocurrencia $P_i \in$ Pat está asignada al ciclo C_j.*

2) *PS_{ij} = 0 si la ocurrencia $P_i \in$ Pat no está asignada al ciclo C_j.*

4.4.2.1 Formulación del problema de planificación

Una planificación con restricción de tiempo, como la planteada en este capítulo, incorpora una serie de limitaciones en la formulación del problema. Dichas restricciones aplicadas al método de planificación propuesto son:

1) *Cada ocurrencia P_i de un patrón Pat se planifica en un solo ciclo C_j comprendido entre sus planificaciones ASAP(PS_i) y ALAP(PL_i) de tal forma que $PS_i \leq j \leq PL_i$*

2) *Cada ocurrencia P_i de un patrón está planificada en un único ciclo de reloj, y por tanto se cumple:*

$$\sum_{j \in Ciclos} PS_{ij} = 1$$

3) *Las ocurrencias de un patrón pueden planificarse en el mismo ciclo que algunas de sus operaciones predecesoras o sucesoras, siempre que la ejecución encadenada de las operaciones no supere el tiempo de ciclo establecido.*

Una vez expresadas las restricciones anteriores, se puede establecer el objetivo principal del algoritmo de planificación de ocurrencias como la planificación del mayor número de ocurrencias posibles de un patrón en distintos ciclos de reloj. Para alcanzar este objetivo es necesario definir dos nuevos conceptos: grafo de distribución de ocurrencias de un patrón a ciclos de reloj y probabilidad de asignación de ocurrencias a ciclos.

El valor de la *fuerza* asociada a la asignación de una ocurrencia de un patrón a un ciclo de reloj depende en gran medida del grafo de distribución de probabilidades. Dichas probabilidades se utilizan para calcular el valor de la *fuerza* y, finalmente, son decisivas para determinar la ocurrencia y el ciclo de reloj seleccionados en cada momento. A continuación se introducen las definiciones indicadas anteriormente y las fórmulas necesarias para calcular el valor de la *fuerza* asociada a la asignación de una ocurrencia a un ciclo.

4.4.2.2 Grafo de distribución de ocurrencias a ciclos

El grafo de distribución de un patrón *Pat* proporciona las posibles ubicaciones de cada ocurrencia P_i del patrón. Los ciclos en que resulta posible planificar una ocurrencia del patrón son aquellos comprendidos en el rango de movilidad de dicha ocurrencia, siempre y cuando sea posible la ejecución de todas sus operaciones sin sobrepasar el tiempo de ciclo. Se eliminan por tanto del rango de movilidad aquellos ciclos en los que existen operaciones predecesoras o sucesoras del patrón ya planificadas y cuya ejecución encadenada con las operaciones del patrón supera el tiempo de ciclo establecido previamente. Este grafo se diseña utilizando los conceptos PS_i y PL_i definidos previamente.

En la figura 4.6 a) se presenta un GFD en que se identifican dos patrones, *R* y *V*, formados por dos operaciones cada uno, y sus distintas ocurrencias. En la figura 4.6 b) se muestra el grafo de distribución de las ocurrencias de ambos patrones. Y en las figuras 4.6 c) y d) se muestran las planificaciones de las ocurrencias de los patrones *V* y *R*, respectivamente.

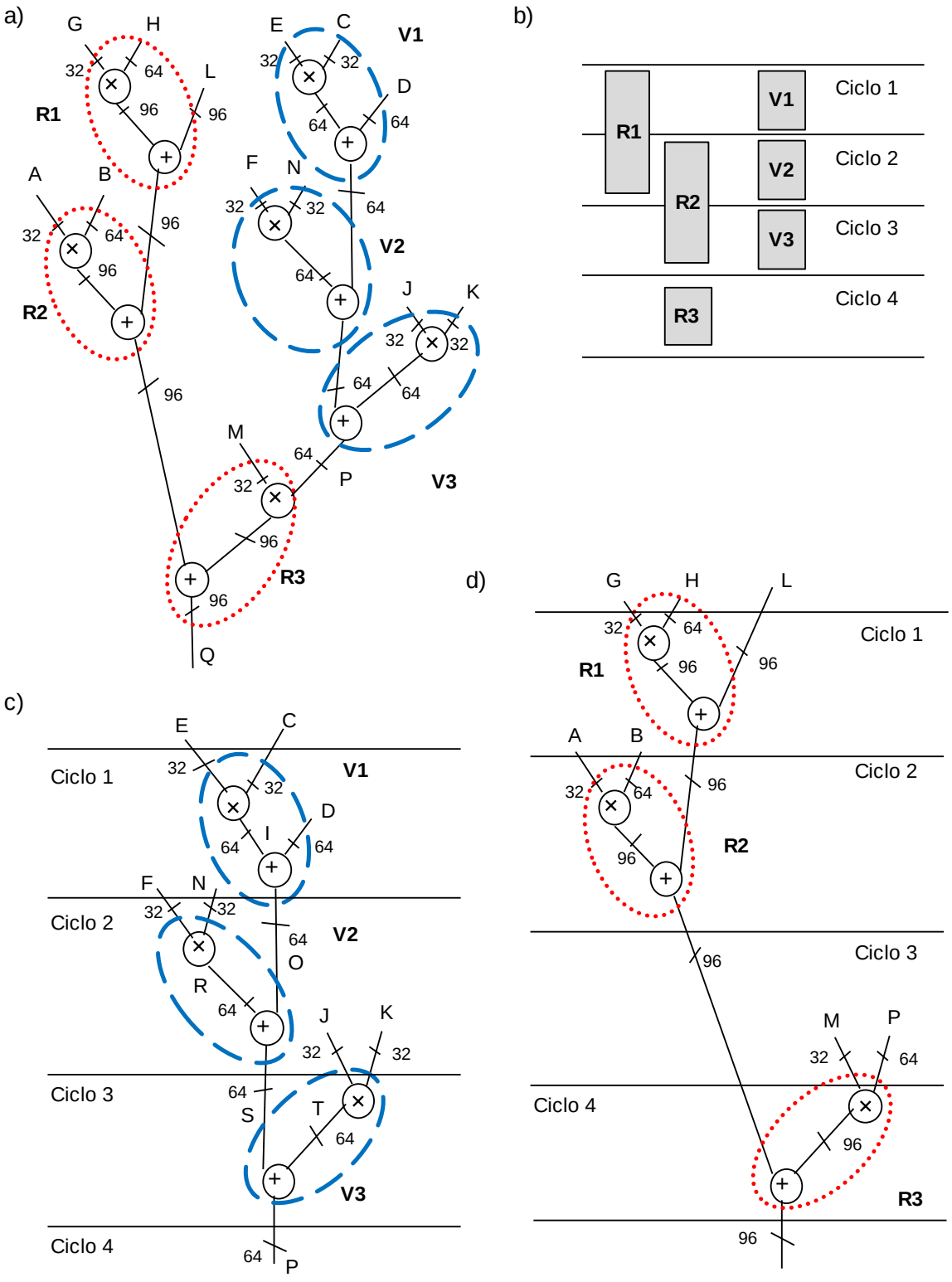

Figura 4.6 *Grafo de distribución de las ocurrencias de un patrón.*

a) GFD con patrones R y V y sus ocurrencias.

b) Grafo de distribución de las ocurrencias de los patrones R y V.

c) y d) Planificación de las ocurrencias de los patrones V y R, respectivamente.

4.4.2.3 Probabilidad de la asignación de ocurrencias a ciclos

La probabilidad de la asignación de una ocurrencia de un patrón a un ciclo de reloj mide la probabilidad de planificar dicha ocurrencia en un determinado ciclo, teniendo en cuenta la movilidad de la ocurrencia. Se considera que la probabilidad de planificar una ocurrencia en un ciclo de reloj es la misma para todos los ciclos de su rango de movilidad.

Definición 4.8: Siendo *Ciclos* el conjunto de ciclos de la planificación, C_j el ciclo que ocupa el orden *j* de la planificación y P_i una ocurrencia del patrón *Pat*, se define $PR_j(P_i)$ como la probabilidad de que P_i sea planificada en el ciclo C_j. El valor asociado a la probabilidad de planificar la ocurrencia P_i en el ciclo C_j es el siguiente:

1) $\mathrm{PRj}(Pi) = \dfrac{1}{\mathrm{Movilidad}(Pi)}$ *si* $PS_i \le C_j \le PL_i$

2) $\mathrm{PR_j}(P_i) = 0$ *en caso contrario.*

El primer caso corresponde a la probabilidad de que la ocurrencia P_i sea planificada en uno de los ciclos incluidos en su rango de movilidad. Nótese que la probabilidad de asignar una ocurrencia a cada ciclo de su rango de movilidad es idéntica.

En la planificación del GFD de la figura 4.6 b) tenemos la siguiente situación:

a) Tres ocurrencias del patrón *V* (*V1*, *V2*, *V3*) ya planificadas.

b) Tres ocurrencias del patrón *R* (*R1*, *R2*, *R3*) pendientes de ser planificadas.

En este caso, las probabilidades de planificar cada una de las ocurrencias del patrón *R* en cada uno de los cuatro ciclos de reloj se calculan según la fórmula anterior, y sus valores quedan reflejadas en la tabla 4.1 que se presenta a continuación.

Ocurrencia	PS	PL	Mov.	Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4
R1	1	2	2	1 / ([1,2]) = 50%	1 / ([1,2]) = 50%	0%	0%
R2	2	3	2	0%	1 / ([2,3]) = 50%	1 / ([2,3]) = 50%	0%
R3	4	4	1	0%	0%	0%	1 / ([4,4]) = 100%

Tabla 4.1 *Probabilidad de la planificación de las ocurrencias del patrón R en cada ciclo*

Definición 4.9: Siendo *Pat* un patrón presente en la especificación conductual y C_j un ciclo de la planificación, se define *Ocur*(*Pat*,C_j) (*Ocurrencias del patrón Pat planificables en el ciclo* C_j) como el conjunto de ocurrencias del patrón *Pat* cuyo rango de movilidad hace posible su planificación en el ciclo C_j.

Definición 4 .10: Siendo *Pat* un patrón presente en la especificación conductual y C_j un ciclo de la planificación, se define *EC*(*Pat*, C_j) (*Estimación del Coste de las ocurrencias del patrón Pat e*n el ciclo C_j) como la suma de la probabilidad de las ocurrencias del patrón *Pat* en el ciclo C_j.

$$\mathrm{EC}(Pat, Cj) = \sum_{Pi \in \mathrm{Ocur}(Pat, Cj)} \mathrm{PRj}(Pi)$$

4.4.2.4 Fuerza asociada a la asignación de una ocurrencia a un ciclo

Una vez presentadas las definiciones anteriores ya resulta posible introducir el concepto de *fuerza* asociada a un par (ocurrencia, ciclo) y que mide la bondad de la planificación de las operaciones de dicha ocurrencia en dicho ciclo, teniendo en cuenta que la selección del ciclo en que se ejecuta la ocurrencia puede afectar a la movilidad de sus antecesoras y sucesoras, lo que a su vez puede modificar el conjunto de ocurrencias planificables del patrón en cada ciclo.

Por tanto, la *fuerza* asociada a la asignación de una ocurrencia P_i de un patrón *Pat* a un ciclo C_j, coincide con la suma de la *fuerza* de la propia asignación y las fuerzas de todas las ocurrencias *afectadas* (antecesoras y sucesoras). La fórmula completa correspondiente al cálculo de la *fuerza*

asociada a la planificación de la ocurrencia P_i de un patrón *Pat* en el ciclo C_j es la siguiente:

$$F(Pi,Cj) = FP(Pi,Cj) + \sum_{Pi' \in (Suc(Pi) \cup Ant(Pi))} FA(Mov_{Pi'}^{nuevo}) \text{, donde}$$

$$FP(Pi,Cj) = EC(Pat,Cj) - \sum_{Ck \in Rango(Pi)} \frac{EC(Pat,Ck)}{Movilidad(Pi)} \text{ y}$$

$$FA(Mov_{Pi}^{nuevo}) = \sum_{Ck \in Mov_{Pi}^{nuevo}} \frac{EC(Pat,Ck)}{Mov_{Pi}^{nuevo}} - \sum_{Ck \in Mov_{Pi}^{inicial}} \frac{EC(Pat,Ck)}{Mov_{Pi}^{inicial}}$$

siendo Mov_{Pi}^{nuevo} y $Mov_{Pi}^{inicial}$ respectivamente los rangos de la movilidad nueva e inicial de las operaciones afectadas por la asignación en cuestión.

Cuanto menor sea el valor de la *fuerza* asociada a la asignación de una ocurrencia a un ciclo, mejor resulta dicha asignación en términos de que aumenta la probabilidad de alcanzar una planificación equilibrada. De esta forma se planifican primero las ocurrencias con menor movilidad en los ciclos cuya estimación de costes, EC(*Pat*,*Cj*), sea menor.

4.4.2.5 Secuencia de ejecución del algoritmo de planificación PFDS

El algoritmo de planificación de las ocurrencias de un patrón consiste en un bucle, y en cada iteración del mismo se planifica una ocurrencia del patrón en un ciclo de su rango de movilidad.

En cada iteración del algoritmo se calcula para cada ocurrencia sin planificar el valor de la *fuerza* asociada a la planificación de dicha ocurrencia en cada uno de los ciclos de su rango de movilidad. A continuación se selecciona el par formado por la ocurrencia y el ciclo que ha obtenido el menor valor de la *fuerza*. Dicho par es el que resulta más conveniente para planificar en cada momento con el fin de alcanzar una distribución uniforme de ocurrencias a ciclos. Si en el ciclo seleccionado se hubiera planificado previamente una ocurrencia de este patrón, se selecciona el par (ocurrencia, ciclo) que tiene el siguiente mejor valor de la *fuerza*, de modo que en cada

iteración se planifica una única ocurrencia y el número máximo de ocurrencias planificadas en cada ciclo es uno. Nótese que para reducir la complejidad del algoritmo podría calcularse únicamente la *fuerza* asociada a los ciclos en los que aún no se ha planificado ninguna ocurrencia del patrón, y la selección de la ocurrencia que se planifica y el ciclo al que se asigna sería la misma.

El algoritmo termina cuando se han planificado todas las ocurrencias del patrón, o no es posible planificar ninguna otra debido a que los rangos de movilidad de las ocurrencias sin planificar sólo incluyen ciclos en los que previamente han sido planificadas otras ocurrencias del mismo patrón.

A continuación se presenta el pseudocódigo de este algoritmo.

	Algoritmo 4.3 - Planificación de ocurrencias de patrón PFDS	
1	**Funcion Planificar(Pat)**	Función de planificación de ocurrencias de un patrón
2	PatSinPlanificar = Ocurrencias(Pat);	Se inicializa el conjunto de ocurrencias sin planificar con todas las ocurrencias
3	CiclosSinPlanificar = {1..latencia}	Se inicializa el conjunto de ciclos en que no se han planificado ocurrencias con todos los ciclos
4	fin= **false;**	Se inicializa la variable fin
5	**Mientras** fin = **false Hacer**	
6	CalcularFuerzas(PatSinPlanificar);	Se calcula la *fuerza* de las ocurrencias sin planificar en los ciclos de su movilidad
7	SeleccionarMenorFuerza(Pi,Cj,CiclosSinPlanificar);	Se selecciona el par (ocurrencia, ciclo) con menor *fuerza*
8	Planificar(Pi,Cj);	Se planifica la operación en el ciclo seleccionado
9	PatSinPlanificar = PatSinPlanificar – {Pi}	Se actualiza el conjunto de ocurrencias sin planificar
10	CiclosSinPlanificar = CiclosSinPlanificar – {Cj}	Se actualiza el conjunto de ciclos en que no se han planificado ocurrencias
11	fin = **true;**	
12	**Mientras** fin=**true Hacer**	Se comprueba si es posible planificar alguna ocurrencia más
13	**Para (**Pi ∈ PatSinPlanificar) ∧ (Cj ∈ CiclosSinPlanificar) **Hacer**	
14	**Si** Cj ∈ Rango(Pi) **entonces** fin= **false;**	
15	**FinSi;**	
16	**FinPara;**	
17	**FinMientras;**	
18	**FinMientras**;	
19	**FinFunción**;	

Planificación de las ocurrencias de un patrón

4.5 Asignación de unidades funcionales

La función de asignación de unidades funcionales a las operaciones de las ocurrencias del patrón planificado resulta trivial al haberse planificado un máximo de una ocurrencia del patrón por ciclo. El conjunto de unidades funcionales utilizadas es el que se seleccionó previamente cuando se escogió este patrón como el de mayor calidad de entre los que cumplían los parámetros de calidad definidos por el diseñador.

Al ejecutarse todas las operaciones que forman una ocurrencia de un patrón de forma encadenada en un solo ciclo, no es necesario realizar selección y asignación de unidades de almacenamiento intermedias para guardar los resultados internos producidos y consumidos por las operaciones del patrón, a menos que alguno de ellos sea requerido por alguna operación planificada en un ciclo posterior.

En la tabla 4.2 quedan reflejadas las unidades funcionales asignadas a cada una de las dos operaciones de las tres ocurrencias del patrón V (V1, V2, V3) de la figura 4.6. Al haber sido planificadas las tres ocurrencias en tres ciclos distintos pueden utilizar las mismas unidades funcionales. Es decir, las tres multiplicaciones, una de cada ocurrencia, se ejecutan en el mismo multiplicador y las tres adiciones se calculan también en el mismo sumador.

	Ciclo 1	**Ciclo 2**	**Ciclo 3**	**Ciclo 4**
Multiplicador 32x32 bits	I=ExC	R=FxN	T=JxK	
Sumador 64 bits	O=I+D	S=R+O	P=S+T	

Tabla 4.2 *Asignación de UF a las operaciones de las ocurrencias de V (figura 4.6)*

Debido al encadenamiento de operaciones, el multiplicar queda ocioso al final de cada uno de los ciclos y el sumador durante los primeros instantes de cada ciclo. Además, tanto el multiplicador como el sumador no realizan operaciones de la especificación durante el ciclo 4. Esta situación de ociosidad se representa en la tabla mediante el fondo gris de las celdas correspondientes.

4.5.1 Algoritmo de asignación

El algoritmo de asignación de unidades funcionales parte de un conjunto de ocurrencias planificadas del patrón y de un conjunto de unidades funcionales sobre las que se ejecutarán las operaciones del patrón. Recuérdese que dicho conjunto de unidades funcionales fue seleccionado con anterioridad a la fase de planificación de las ocurrencias de un patrón.

El algoritmo utilizado en esta fase recorre los ciclos de la planificación de forma secuencial. En cada ciclo se comprueba si el patrón tiene planificada una ocurrencia, en cuyo caso, se asignan las unidades funcionales seleccionadas para el patrón a cada una de las operaciones que forman la ocurrencia del mismo.

Tras recorrer todos los ciclos se comprueba si existen ocurrencias no planificadas del patrón, en cuyo caso deberá comprobarse si el patrón sigue siendo un patrón candidato, teniendo en cuenta únicamente las ocurrencias no planificadas. Si por el contrario se han planificado todas las ocurrencias del patrón, éste se elimina del conjunto de patrones candidatos.

A continuación se presenta el pseudocódigo de la función de asignación de las ocurrencias de un patrón.

	Algoritmo 4.4 - Asignación de UF a ocurrencias de patrón	
1	**Funcion** Asignar(Patron,UFPatron)	Función de asignación de UF a las ocurrencias de un patrón
2	**Para** Cj en Ciclos **Hacer**	Se recorren todos los ciclos de reloj
3	**Si** Ocurrencia(Patron, Cj) = **true Entonces**	Se comprueba si se ha planificado una ocurrencia en el ciclo *Cj*
4	Pi=Ocurrencia(Patron, Cj);	Se obtiene la ocurrencia del patrón planificada en el ciclo *Cj*
5	AsignarUFaOcuPatron(Pi , Cj , UFPatron);	Se asignan las UF a las operaciones de la ocurrencia *Pi* en ciclo *Cj*
6	Patron = Patron - {Pi};	Se elimina la ocurrencia *Pi* del patrón
7	**FinSi;**	
8	**FinHacer;**	
9	**Si** Patron = Ø **Entonces**	Si se han asignado UF a todas las ocurrencias del patrón
10	PAT = PAT – {Patron};	Se elimina el patrón del conjunto de patrones pendientes de procesar
11	**FinSi;**	
12	**FinFunción;**	

Asignación de UF a las ocurrencias de un patrón

4.6 Ajuste de patrones

Una vez terminadas las fases de planificación, selección y asignación para el patrón de mayor calidad, se llevan a cabo una serie de operaciones de ajuste. Tal como se ha comentado previamente, puede darse la situación de que no todas las ocurrencias del patrón seleccionado como el de mayor calidad hayan sido planificadas[4]. En ese caso, el patrón aún puede pertenecer al conjunto de patrones candidatos, aunque su número de ocurrencias es menor que el que tenía originalmente.

La planificación de algunas ocurrencias del patrón seleccionado no sólo ha reducido el número de ocurrencias sin planificar del mismo, sino que también puede haber provocado la reducción de las ocurrencias de otros patrones candidatos. Esta situación puede producirse si algunas de las operaciones de las ocurrencias planificadas a su vez formaban parte de las ocurrencias de otros patrones. Por lo tanto, llegado este punto se hace necesario revisar todo el conjunto de patrones candidatos y comprobar si el número de ocurrencias de los mismos es suficiente para seguir cumpliendo la propiedad de calidad *CP*. Los patrones que hayan dejado de cumplir esta propiedad se eliminarán del conjunto de patrones candidatos.

A continuación se presenta el pseudocódigo que realiza la comprobación descrita.

	Algoritmo 4.5 - Primer ajuste del conjunto de patrones	
1	**Para** Patron **en** PAT **Hacer**	Se procesan de forma secuencial los patrones de PAT
2	**Si** Ocurrencias(Patron) < CP **Entonces**	Si el patrón no cumple la propiedad de calidad *CP*
3	PAT = PAT - {Patron};	Se elimina del conjunto de patrones a ser procesados
4	**FinSi**;	
5	**FinHacer**;	Fin condición llamada a función de planificación y asignación

Primer ajuste del conjunto de patrones

La ejecución de este primer ajuste puede provocar que el conjunto de patrones candidatos quede vacío. En este caso, el método propuesto procederá a la identificación de nuevos patrones a partir de las operaciones

[4] Esta situación se puede dar si el número de ocurrencias del patrón es mayor que la latencia, o en el caso en que no sea posible planificar las ocurrencias en ciclos distintos.

disponibles en ese momento. Las operaciones disponibles son aquellas a las que aún no se les ha asignado ninguna unidad funcional para su ejecución. Nótese que su movilidad puede ser cualquiera, incluso pueden tener movilidad uno, es decir, pueden encontrarse planificadas de facto debido a sus dependencias de datos con otras operaciones de la especificación.

Siguiendo el mismo método utilizado al principio del algoritmo, primero se intentarán definir nuevas operaciones cabecera, teniendo en cuenta para ello las operaciones disponibles en la especificación conductual. Tal como se definió en apartados anteriores, para que una operación pueda ser considerada como operación cabecera debe cumplir el parámetro de calidad *PH*, es decir, tener un número de ocurrencias igual o superior al valor definido por el diseñador para dicha propiedad. Una vez definidas nuevas operaciones cabecera se procederá a identificar nuevos patrones a partir de cada una de ellas.

A continuación se presenta el pseudocódigo correspondiente al segundo ajuste del conjunto de patrones, consistente en la generación de nuevos patrones.

	Algoritmo 4.6 - Segundo ajuste del conjunto de patrones	
1	**Si** PAT = Ø **entonces**	Si después del primer ajuste, no hay patrones a ser procesados
2	OpeCabecera= **IdentificarNuevasOperacionesInicio(**SPEC**)**;	Identificar nuevas operaciones cabecera en la descripción conductual
3	**Para** ope en OpeCabecera **Hacer**	Para cada nueva operación cabecera identificada
4	PAT = PAT U **CalcularPatron**(ope);	Calcular nuevos patrones y añadirlos al conjunto de patrones
5	**FinHacer**;	
6	**FinSi**;	

Segundo ajuste del conjunto de patrones

Tras la ejecución de esta fase, si el conjunto de patrones candidatos contiene algún patrón comienza una nueva iteración del algoritmo. En esta iteración se seleccionará el patrón de mayor calidad y se planificará. Así mismo, se seleccionarán las unidades funcionales necesarias, y se asignarán a las operaciones de algunas de sus ocurrencias.

Si por el contrario, el conjunto de patrones candidatos se encuentra vacío, se procederá a realizar la planificación y la selección y asignación de HW de las operaciones restantes. A estas operaciones que no han sido procesadas por el algoritmo como parte de ningún patrón en las fases previas se las denomina operaciones residuales. A ninguna de ellas se les ha asignado aún ningún recurso funcional para ejecutarlas. Algunas se encuentran sin planificar, y otras sin embargo tienen una movilidad igual a uno debido a sus dependencias de datos con otras operaciones ya planificadas de la especificación.

4.7 SAN de las operaciones residuales

Una vez finalizada la SAN de los patrones identificados en la especificación conductual, se alcanza una situación en la que sólo quedan pendientes de ser procesadas un conjunto de operaciones residuales, a partir de las cuales resulta imposible generar nuevos patrones que cumplan los parámetros de calidad definidos por el diseñador. Con el objeto de finalizar el proceso de SAN de la especificación conductual y obtener una implementación del circuito sintetizado, es necesario llevar a cabo las etapas de planificación y selección y asignación de recursos HW de las operaciones residuales.

En esta fase, las etapas del proceso de síntesis se ejecutan en el siguiente orden: planificación, selección y asignación de HW. Durante la planificación de las operaciones residuales se tienen en cuenta todas las operaciones del mismo tipo presentes en la especificación, tanto las que pertenecen al conjunto de operaciones residuales como las que se han planificado previamente como parte de alguna ocurrencia de algún patrón. De este modo el objetivo que se persigue es obtener una distribución uniforme de las operaciones de cada tipo diferente en ciclos de reloj, lo que permite reutilizar las unidades funcionales que ya forman parte de la ruta de datos y que se encuentran ociosas en algún ciclo.

Tras la planificación de las operaciones residuales se procede a realizar la selección y asignación de HW. Para ello, en primera instancia se intenta

reutilizar los recursos funcionales presentes en la ruta de datos, y cuando sea necesario se procede a añadir nuevo HW.

A continuación se presenta el pseudocódigo que realiza las llamadas sucesivas a cada una de las etapas de la SAN de las operaciones residuales.

	Algoritmo 4.7 - SAN de las operaciones residuales	
1	PlanficarOpRes(OpRes);	Función de planificación operaciones residuales
2	SeleccionAsignacionUFOpRes(OpRes);	Función de selección y asignación de UF para operaciones residuales

SAN de las operaciones residuales

4.7.1 Planificación de las operaciones residuales

La planificación de las operaciones residuales se realiza utilizando el algoritmo clásico de planificación basada en fuerzas FDS [PaKn89]. Este algoritmo persigue alcanzar distribuciones uniformes de las operaciones de cada tipo en los diferentes ciclos de reloj, planificando un número similar de operaciones de cada tipo por ciclo. Todo ello con el objeto de aumentar la reutilización de los recursos funcionales de la ruta de datos, disminuyendo así el área de la implementación resultante.

Con el objetivo de obtener la mejor distribución de operaciones a ciclos, las operaciones de las ocurrencias de los patrones son consideradas como operaciones simples ya planificadas, y por tanto con movilidad uno. Esta consideración favorece que las operaciones sin planificar compartan las mismas unidades funcionales que las operaciones de los patrones. Para el resto de operaciones (operaciones residuales) el algoritmo trabaja con la movilidad de cada una de ellas para decidir el ciclo en que finalmente quedan planificadas.

El algoritmo utilizado consiste en un bucle que se ejecuta mientras queden operaciones sin planificar en la especificación conductual. En cada iteración del mismo se planifica una única operación. La operación y el ciclo seleccionados se escogen teniendo en cuenta una estimación del coste de la implementación resultante. Dicha estimación se calcula para cada posible

asignación de operaciones a ciclos, y se denomina *fuerza*. El valor de la fuerza calculado en esta fase del algoritmo es diferente al utilizado en la planificación de las ocurrencias de un patrón. A continuación se presenta cómo se realiza el cálculo del nuevo valor de la *fuerza* usando la misma nomenclatura que la utilizada anteriormente.

En primer lugar, es necesario introducir algunas definiciones necesarias para calcular la *fuerza* asociada a una operación y a un ciclo de reloj.

Definición 4.11: Siendo *Ciclos* el conjunto de ciclos de la planificación, *Cj* el ciclo que ocupa el orden *j* de la planificación y *Oi* una operación de la especificación conductual, se define *PRj(Oi)* como la probabilidad de que *Oi* sea planificada en el ciclo *Cj*. Los valores que puede tener asignada esta probabilidad según el posible ciclo *Cj* donde se planifique la operación son:

1) $PRj(Oi) = \frac{1}{Movilidad(Oi)}$ *si PSi ≤ Cj ≤ PLi, siendo PSi y PLi las planificaciones ASAP y ALAP de Oi, respectivamente.*

2) *PRj(Oi) = 0 en cualquier otro caso.*

El primer caso corresponde a la probabilidad de que la operación *Oi* sea planificada en un ciclo de su rango de movilidad. Nótese que la probabilidad de asignar una operación a cada ciclo de su rango de movilidad es idéntica.

Definición 4.12: Siendo *Top* un determinado tipo de operación presente en la especificación conductual y *Cj* un ciclo de la planificación, se define *Opes(Top,Cj)* (*operaciones del tipo Top planificables en el ciclo Cj*) como el conjunto de operaciones de la especificación del tipo *Top* cuyo rango de movilidad hace posible su planificación en el ciclo *Cj*.

Definición 4.13: Siendo *Top* un determinado tipo de operación presente en la especificación conductual y *Cj* un ciclo de la planificación, se define *EC(Top, Cj)* (*estimación del coste de las operaciones del tipo Top en* el ciclo *Cj*) como la suma de las probabilidades de las operaciones del tipo *Top* en el ciclo *Cj*.

$$\mathrm{EC}(Top,Cj) = \sum_{Oi \in\ Opes(Top,Cj)} PRj(Oi)$$

La *fuerza* asociada a la planificación de la operación *Oi* del tipo *Top* en el ciclo *Cj* se calcula utilizando la siguiente fórmula:

$$\mathrm{F}(Oi,Cj) = \mathrm{FP}(Oi,Cj) + \sum_{Oi' \in\ (\mathrm{Suc}(Oi)\ \cup\ \mathrm{Ant}(Oi))} \mathrm{FA}(\mathrm{Mov}_{Oi'}^{nuevo}) \text{, donde}$$

$$\mathrm{FP}(Oi,Cj) = \mathrm{EC}(Top,Cj) - \sum_{Ck \in\ \mathrm{Rango(Oi)}} \frac{\mathrm{EC}(Top,Ck)}{Movilidad(Oi)} \text{ y}$$

$$\mathrm{FA}(\mathrm{Mov}_{Oi}^{nuevo}) = \sum_{Ck \in\ \mathrm{Mov}_{Oi}^{nuevo}} \frac{\mathrm{EC}(Top,Ck)}{\mathrm{Mov}_{Oi}^{nuevo}} - \sum_{Ck\ \in\ \mathrm{Mov}_{Oi}^{inicial}} \frac{\mathrm{EC}(Top,Ck)}{\mathrm{Mov}_{Oi}^{inicial}}$$

siendo $\mathrm{Mov}_{Oi}^{nuevo}$ y $\mathrm{Mov}_{Oi}^{inicial}$ respectivamente los rangos de la movilidad nueva e inicial de las operaciones afectadas por la asignación en cuestión.

Como en el caso del algoritmo utilizado en la planificación de las ocurrencias de un patrón, la operación y el ciclo seleccionados serán aquellos que tengan el menor valor de la función *fuerza*. Por tanto, se planifican en primer lugar las operaciones con menor movilidad en los ciclos cuya estimación del costes, *EC*(*Top*,*Cj*), sea menor.

En este caso se omite el pseudocódigo del algoritmo por ser similar al del algoritmo clásico de planificación dirigida por fuerzas.

4.7.2 Selección y asignación de unidades funcionales

El algoritmo utilizado para realizar la selección y asignación de HW de las operaciones residuales tiene como objetivo seleccionar la unidad funcional en que se ejecuta cada una de las operaciones residuales, minimizando el incremento de área del circuito resultante. Para ello, se intentará en primera instancia utilizar las unidades funcionales presentes en la ruta de datos y que se encuentren ociosas en algún ciclo de reloj.

Algunas operaciones serán asignadas directamente a las unidades funcionales de la ruta de datos. En los casos en que el tamaño de la unidad funcional ociosa sea menor que el de la operación residual, se plantea la fragmentación de operaciones. De este modo, la operación se divide en varios fragmentos y estos se asignan a las unidades funcionales ociosas en un ciclo concreto. En algunos casos será necesario añadir nuevos recursos funcionales a la ruta de datos para ejecutar alguno de estos fragmentos o alguna de las operaciones residuales.

El algoritmo utilizado para realizar la selección y asignación de HW de las operaciones residuales recorre todos los ciclos de la planificación asignando HW a estas operaciones. Los ciclos se recorren siguiendo el orden cronológico de los mismos, y en cada ciclo se procesan las operaciones sin asignar en orden decreciente de complejidad, primero las multiplicaciones y a continuación las sumas. Las operaciones de cada tipo se procesan en orden decreciente de anchura. Este orden garantiza que las unidades funcionales ociosas se utilizan para ejecutar las operaciones residuales más complejas, de modo que si finalmente fuera necesario añadir nuevas unidades funcionales a la ruta de datos, éstas serían más sencillas.

Para cada una de las operaciones consideradas se utiliza el siguiente criterio para seleccionar la unidad funcional, o el conjunto de unidades funcionales, que la ejecutan:

1) Si hay una unidad funcional en la ruta de datos capaz de ejecutar la operación que se está procesando y se encuentra ociosa en el ciclo en cuestión, se seleccionará y asignará dicha unidad funcional a la operación actual.

2) Si no ha sido posible seleccionar una unidad funcional ociosa para ejecutar la operación, se comprueba si la operación puede descomponerse para que sus fragmentos puedan ejecutarse en las unidades funcionales ociosas. En este caso el algoritmo ensaya las fragmentaciones presentadas en el capítulo previo con el objetivo de maximizar la parte de la operación que se ejecuta sobre las unidades funcionales ociosas.

3) Si la operación no ha sido completamente asignada se seleccionarán las unidades funcionales de la biblioteca de diseño necesarias para completar su ejecución. Las unidades seleccionadas serán las más rápidas entre las disponibles en la biblioteca, a fin de garantizar que se cumplen las restricciones de tiempo impuestas al diseño (latencia del circuito y duración del ciclo de reloj). Posteriormente, se optimizará el circuito resultante sustituyendo algunas de estas unidades funcionales por otras de menor área sin dejar de cumplir las citadas restricciones.

4.7.2.1 Disponibilidad de unidades funcionales compatibles

En este caso se dispone de unidades funcionales ociosas en los ciclos en que se encuentran planificadas las operaciones residuales, y éstas son compatibles en tipo y anchura con las operaciones en cuestión. Por lo tanto el algoritmo de SAN reutilizará dichas unidades funcionales para ejecutar las operaciones residuales.

En la figura 4.7 se presentan 3 ocurrencias del patrón P (P1, P2, P3) ya planificadas y que reutilizan en cada ciclo correspondiente dos unidades funcionales, un multiplicador y un sumador. En esta situación el algoritmo de SAN ha planificado las ocurrencias del patrón, y ha seleccionado y asignado las unidades funcionales necesarias para ejecutar las operaciones de cada ocurrencia. A continuación, el algoritmo de SAN pasa a tratar las operaciones residuales, en este caso una sola operación aditiva de 96 bits de anchura que ha sido planificada en el ciclo 3.

Veamos en primer lugar la utilización de las dos unidades funcionales de la ruta de datos en cada ciclo. Esta información está representada en la tabla 4.3. Según se desprende de esta tabla, el sumador está disponible en el ciclo 3.

	Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4
Multiplicador 32×64 bits	K=GxH	U=AxB		P=MxW(63:0)
Sumador 96 bits	T=K+L	O=U+T		Q=P+O

Tabla 4.3 *Asignación de UF a las ocurrencias del patrón de la figura 4.7*

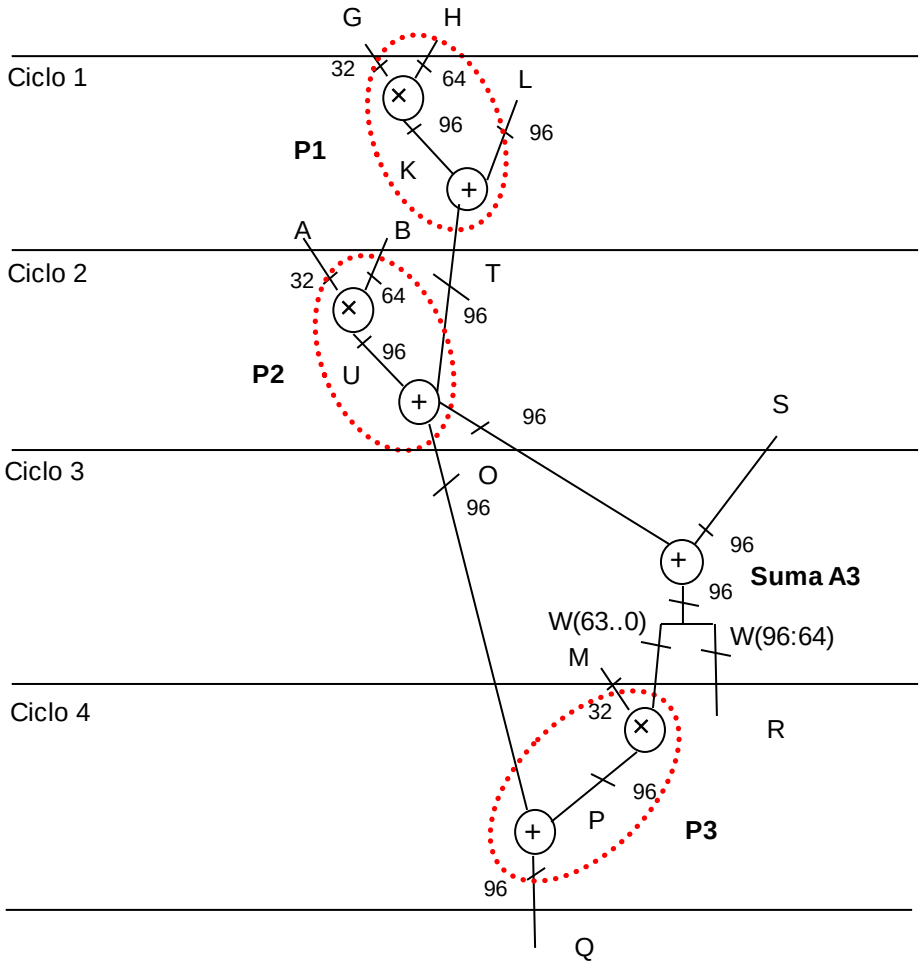

Figura 4.7 *Asignación de UF ociosas a operaciones residuales.*

Dado que el sumador de 96 bits está ocioso en el ciclo 3 y la operación A3 (W=O+S) es compatible en naturaleza y anchura con dicha unidad funcional, el algoritmo de SAN asignará dicho sumador a esta operación aditiva en el ciclo indicado. En la tabla 4.4 se muestra el uso de las unidades funcionales tras realizar la asignación de HW a las operaciones residuales.

	Ciclo 1	**Ciclo 2**	**Ciclo 3**	**Ciclo 4**
Multiplicador 32×64 bits	K=GxH	U=AxB		P=MxW(63:0)
Sumador 96 bits	T=K+L	O=U+T	W=O+S	Q=P+O

Tabla 4.4 *Asignación de UF ociosas a las operaciones residuales de la figura 4.7*

4.7.2.2 Disponibilidad de unidades funcionales no compatibles

La segunda situación que puede darse es que existan unidades funcionales ociosas en los ciclos en que resultan planificadas las operaciones residuales, siendo éstas del mismo tipo y diferente anchura. En la figura 4.8 se presentan 3 ocurrencias del patrón P (P1, P2, P3) ya planificadas y que reutilizan en cada ciclo correspondiente dos unidades funcionales: un multiplicador y un sumador.

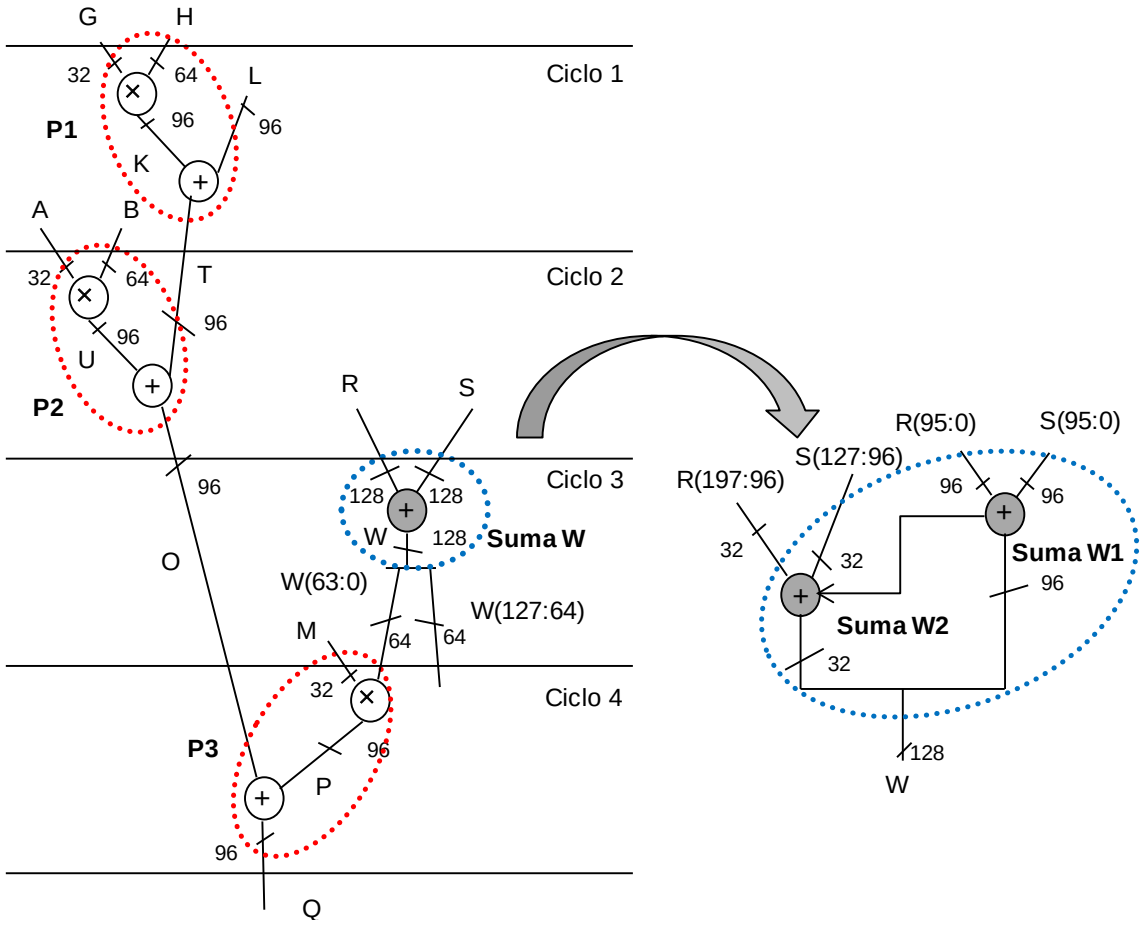

Figura 4.8 *Descomposición de operaciones residuales para aprovechar UF ociosas.*

La situación de las unidades funcionales en cada ciclo y en cada parte de cada ciclo es la misma que la mostrada en la tabla 4.3. La diferencia en este caso es que hay sumadores disponibles no compatibles en anchura con las operaciones aditivas residuales. Las unidades funcionales disponibles tienen una anchura de 96 bits y la operación aditiva residual tiene una anchura de 128 bits. Llegado a este punto, el algoritmo de SAN simulará la fragmentación

de la operación aditiva de 128 bits, mostrada en la figura 4.8, para comprobar si se obtienen fragmentos de anchura compatible con la longitud de la unidad funcional aditiva ociosa, es decir, un fragmento de 96 bits.

Una vez que el algoritmo de SAN ha simulado la fragmentación y comprobado que sí se generan operaciones aditivas compatibles en naturaleza y anchura con el sumador disponible, procederá a llevar a cabo realmente la fragmentación. Tras la fragmentación de esta operación aditiva de 128 bits, se obtienen dos fragmentos o nuevas operaciones aditivas de anchuras 32 y 96 bits. Para la primera el algoritmo de SAN procederá a seleccionar un sumador de 32 bits de la biblioteca correspondiente. Para ejecutar el segundo fragmento, utilizará el sumador de 96 bits que se encuentra ocioso en el ciclo 3 donde se ha planificado la operación.

Con la fragmentación de la operación residual aditiva de 128 bits en dos sumas de 32 y 96 bits, sólo es necesario añadir a la ruta de datos un nuevo sumador de 32 bits, en lugar de uno de 128 bits, con el consiguiente ahorro en área.

En la tabla 4.5 queda representada la utilización de cada una de las unidades funcionales.

	Ciclo 1	**Ciclo 2**	**Ciclo 3**	**Ciclo 4**
Multiplicador 32×64 bits	K=G+H	U=AxB		$P=MxW_{63:0}$
Sumador 96 bits	T=K+L	O=U+T	W(95:0)	Q=P+O
Sumador 32 bits			W(127:96)	

Tabla 4.5 *Asignación de UF a los fragmentos de la operación residual de la figura 4.8*

4.7.2.3 Implementación algorítmica

La implementación algorítmica de la función de selección y asignación de HW para las operaciones residuales trata de forma secuencial todos los ciclos de planificación según el orden cronológico de los mismos. En cada ciclo se tratan todas las operaciones residuales que hayan sido planificadas en él, en orden decreciente de complejidad, realizándose para cada una de ellas las siguientes acciones:

a) Se comprueba si existe una unidad funcional ociosa en dicho ciclo que sea compatible en tipo y anchura con la operación residual. En caso afirmativo se asigna la operación actual a la unidad funcional ociosa.

b) Si el paso anterior no tuvo éxito, se ensayan las fragmentaciones de la operación residual con el objeto de obtener fragmentos del mismo tipo y anchura de las unidades funcionales ociosas en el ciclo en cuestión.

c) Si existe una posible fragmentación que produce operaciones del tipo y anchura de los recursos funcionales ociosos se lleva a cabo la descomposición de la operación residual, y se asignan los fragmentos resultantes a las unidades ociosas.

d) Si la operación, o alguno de sus fragmentos, se encuentra aún sin asignar, entonces se selecciona de la biblioteca de diseño el conjunto de unidades funcionales necesario para su ejecución. Éstas serán las más rápidas de entre todas las disponibles en la biblioteca.

A continuación se presenta el pseudocódigo que implementa la función de selección y asignación de unidades funcionales para la ejecución de las operaciones residuales.

	Algoritmo 4.8 - Selección y asignación de UF para la ejecución de las operaciones residuales	
1	**Funcion** SeleccionAsignaciónUFOpRes(OpRes);	Función de selección y asignación UF a operaciones residuales
2	**Para** Ci **en** Ciclos **Hacer**	En todos los ciclos
3	**Para** Oi **en (**Ci ∧ OpRes) **Hacer**	Para todas las operaciones planificadas en el ciclo
4	UFOciosa = HayUFOciosa(Ci,Oi)	Comprueba si hay UF ociosa válida en el ciclo en que está planificada Oi
5	**Si** UFOciosa = Ø **Entonces**	
6	**Si EsPosibleFrag(**Oi**,**Ociosas**) Entonces**	Comprueba si es posible fragmentar Oi para obtener fragmentos del tipo y anchura de las UF ociosas
7	**Fragmentar(**Oi,UFOciosa,FAsig,FNAsig**);**	Realiza la descomposición de Oi en fragmentos
8	**FinSi;**	
9	**FinSi;**	
10	**Si (**UFOciosa = Ø) ∨ (FragNoAsig ≠ Ø) **Entonces**	Si no hay UF ociosa para Oi o para todos sus fragmentos se seleccionan nuevas UF de la librería
11	NuevasUF=**SeleUF(**Biblio,Oi,FNAsig**);**	
12	**FinSi;**	
13	Asignar(Oi,FAsig,FNAsig,UFOciosa,NuevasUF);	Realiza las asignaciones de UF a Oi o a sus fragmentos
14	**FinHacer**;	
15	**FinHacer**;	
16	**FinFunción;**	

Selección y asignación de UF a las operaciones residuales

4.8 Selección y asignación de recursos de almacenamiento y encaminamiento de datos

Una vez finalizadas las fases de planificación de operaciones y selección y asignación de recursos funcionales, tiene lugar la selección y asignación de recursos de almacenamiento y de encaminamiento de datos, con lo que se completa la ruta de datos del circuito. Para ello se utiliza un algoritmo convencional de síntesis que permite el almacenamiento y encaminamiento de datos en registros y multiplexores de mayor anchura. Con esto se pretende reducir el área del circuito dedicada a los recursos de almacenamiento y encaminamiento de datos.

Debido a las descomposiciones de operaciones que han podido llevarse a cabo durante las fases anteriores, el conjunto de datos que deben almacenarse y transmitirse en el circuito resultante puede ser muy diferente del conjunto de variables presentes en la especificación conductual de partida. En particular, el número de datos será mayor y sus anchuras menores que los datos presentes en la especificación original. La fragmentación de operaciones realizada durante las fases de planificación de operaciones y asignación de unidades funcionales, implica una fragmentación de los datos de estas operaciones, lo que a su vez conlleva una mayor reutilización de los recursos de almacenamiento y encaminamiento de datos, en comparación con los circuitos sintetizados con otros algoritmos que no incorporan las técnicas de fragmentación de operaciones.

El algoritmo heurístico utilizado para realizar la selección y asignación de recursos de almacenamiento procesa las variables en orden decreciente de anchura. Para cada variable procesada se comprueba si puede almacenarse en alguno de los registros disponibles en la ruta de datos. Para ello recorre los registros en orden decreciente de anchura, y selecciona el primero cuyas variables asignadas tengan vidas no solapadas con la vida del dato en cuestión. Si tras recorrer todos los registros de la ruta de datos ninguno cumple el requisito anterior, entonces se añade un nuevo registro al circuito del

tamaño del dato en cuestión. A continuación se realiza la asignación del dato al registro seleccionado de la ruta de datos o al que se acaba de incluir en la misma. Al finalizar este bucle la ruta de datos contiene un conjunto de registros válido para almacenar todas las variables del circuito, y cada uno de los datos ha sido asignado a alguno de dichos registros.

A continuación, se añaden a la ruta de datos los multiplexores necesarios para transferir los operandos de cada operación de la especificación definitiva a la unidad funcional en que se ejecuta, y para transferir el resultado de cada operación al conjunto de registros seleccionados para su almacenamiento. Así, es necesario añadir un multiplexor a la entrada de cada uno de los recursos funcionales y de almacenamiento que reciben datos de diversas fuentes. La anchura de cada uno de estos multiplexores coincide con la anchura del recurso al que se encuentra conectado, y el número de entradas con el número de fuentes diferentes de las que recibe datos.

Nótese que de manera análoga a cómo se ha aplicado la descomposición de operaciones para aumentar el aprovechamiento de los recursos funcionales de la ruta de datos, podrían descomponerse los datos en otros de menor anchura y almacenar/transmitir los fragmentos en diferentes recursos de almacenamiento/transmisión de datos. Esta medida permitiría la reutilización de los recursos de almacenamiento y de encaminamiento de datos presentes en la ruta de datos sin necesidad de añadir nuevos elementos HW a la misma. Una posible implementación de esta técnica se presenta en [Moli05] y su incorporación al algoritmo de SAN propuesto es evidente.

A continuación se presenta el pseudocódigo del algoritmo de selección y asignación de recursos de almacenamiento y encaminamiento de datos utilizado.

	Algoritmo 4.9 – Selección y asignación de registros y multiplexores	
1	RegsRD = Ø**;**	Se inicializa el conjunto de registros de la ruta de datos
2	**Para** Var **en** VarParaAlmac **Hacer**	Se recorren todas las variables en orden decreciente de anchura
3	Valido = false;	
4	**Para** Reg **en** RegsRD **Hacer**	Se recorren los registros en orden decreciente de anchura
5	**Si not**(Valido) **and** VidaNoSolapa(Var,Reg) **Entonces**	Se comprueba si la variable Var y las variables asignadas al registro tienen vidas solapadas
6	RegValido = Reg;	Se selecciona el registro de la ruta de datos
7	Valido = True;	
8	**FinSi**;	
9	**FinHacer**;	
10	**Si not**(Valido) **Entonces**	
11	Reg = SelecRegistro(Var,Biblio);	Se selecciona un nuevo registro de la biblioteca de diseño
12	RegsRD = RegsRD ∪ {Reg};	Se añade el nuevo registro a la ruta de datos
13	**FinSi**;	
14	Asignar(Var,Reg);	Se asigna la variable al registro seleccionado o al nuevo
15	**FinHacer**;	
16	AñadirMuxes;	Se añaden los multiplexores necesarios a la ruta de datos

Selección y asignación de recursos de almacenamiento y encaminamiento

4.9 Optimización del área de los circuitos resultantes

Como resultado de las etapas anteriores se ha obtenido un circuito completo a nivel de transferencia entre registros que realiza todas las operaciones especificadas en la descripción conductual. El circuito resultante cumple las restricciones de tiempo impuestas al diseño (latencia del circuito y duración del ciclo de reloj) gracias a que durante todo el proceso se han seleccionado siempre las unidades funcionales más rápidas de la biblioteca de diseño. Nótese que estas unidades funcionales son probablemente las de mayor área de la biblioteca utilizada. Llegado este momento, lo que se pretende es reducir el área de la implementación sintetizada mediante la sustitución de algunas de las unidades funcionales de la ruta de datos por otras más lentas y de menor área, sin que ello afecte al rendimiento del circuito.

La fase de optimización consiste en un bucle que recorre todas las unidades funcionales de la ruta de datos, en orden decreciente de complejidad, probando si es posible sustituir dicha unidad por otra de menor área entre las disponibles en la biblioteca de diseño. Las unidades funcionales son sustituidas en orden decreciente de complejidad, ya que cuanto más complejas son, mayor área ocupan y su sustitución por otras de menor coste

implica potencialmente una mayor reducción del área total del circuito resultante.

Las unidades funcionales de la biblioteca de diseño se prueban en orden creciente de complejidad hasta encontrar una que satisface las restricciones de tiempo impuestas al diseño. La unidad funcional seleccionada es la de menor área de las disponibles en la biblioteca de diseño que no viola las restricciones de tiempo. Si esta unidad funcional no es la utilizada en la ruta de datos se procede a la sustitución de la misma. El ahorro de área obtenido viene determinado por la diferencia de área de ambas unidades funcionales.

La comprobación de si una unidad funcional puede sustituirse por otra de menor área, consiste en verificar que en cada ciclo de reloj en que tiene asignada una operación pueden ejecutarse todas las operaciones planificadas en él sin sobrepasar el tiempo de ciclo fijado por el diseñador. Por tanto, sólo será posible dicha sustitución si en cada ciclo de reloj la unidad funcional en cuestión permanece ociosa durante parte del ciclo.

A continuación se presenta el pseudocódigo del algoritmo de optimización de área de los circuitos sintetizados por nuestro algoritmo.

	Algoritmo 4.10 – Optimización de área del circuito sintetizado	
1	**Para** Uf **en** RutaDatos **Hacer**	Se recorren todas las UF en orden decreciente de complejidad
2	NewUf=SelecUFMenorArea(Uf,Biblio);	Se selecciona la UF de menor área del mismo tipo
3	**Mientras** ViolaRest(NewUf,Uf,latency,ciclo)=**true Hacer**	Se comprueba si la nueva UF viola las restricciones de tiempo
4	NewUf=SelecUFSig(NewUf,Biblio);	Si no es válida se selecciona la siguiente UF de menor área
5	**FinHacer**;	
6	**Si** NewUf ≠Uf **Entonces**	Comprueba si la UF válida es distinta a la de la ruta de datos
7	Sustituir(Uf,NewUf);	En caso afirmativo se sustituye por una de menor área
8	**FinSi**;	
9	**FinHacer**;	
10	**FinAlgoritmo**	Fin algoritmo

Optimización de área de la ruta de datos sintetizada

4.10 Complejidad computacional del algoritmo

En esta sección se estudia la complejidad de cada una de las etapas del algoritmo de SAN propuesto en este capítulo:

1) *Cálculo de las operaciones cabecera en la especificación conductual: complejidad cuadrática.*

2) *Identificación de patrones en la especificación conductual: complejidad cuadrática.*

3) *Planificación de las ocurrencias de los patrones y de las operaciones residuales: complejidad cúbica heredada del algoritmo original FDS.*

4) *Selección y asignación de recursos funcionales, de almacenamiento y encaminamiento de datos: complejidad cuadrática.*

5) *Optimización de las unidades funcionales de la ruta de datos: complejidad lineal.*

A continuación se describen los órdenes de complejidad de cada una de las fases del algoritmo. Nótese que aunque en algunos casos se procede a fragmentar algunas de las operaciones de la especificación conductual, aumentando así el número de operaciones de la especificación, este hecho no se ha tenido en cuenta en el cálculo del coste computacional del algoritmo. La razón es que dicho efecto se anula por el hecho de considerar agrupadas en patrones las operaciones de la especificación, que se planifican y asignan como si de una única operación se tratara.

4.10.1 Cálculo de las operaciones cabecera en la especificación conductual

En el cálculo de las operaciones cabecera, se recorren todas las operaciones de la especificación y para cada una de ellas se obtiene el número de operaciones de su mismo tipo y anchura existentes en la especificación, para lo que se recorre nuevamente el conjunto de operaciones de la especificación. Por lo tanto, siendo n el número de operaciones de la especificación el coste computacional de este algoritmo será $n\times(n\text{-}1)$, es decir complejidad cuadrática, $O(n^2)$.

En caso de no obtenerse ninguna operación cabecera debido a que ninguna de las operaciones de la especificación tiene un número de ocurrencias mayor o igual al valor del parámetro de calidad *PH*, se ensaya la posible descomposición de operaciones de la especificación de manera que se obtengan más operaciones de un determinado tipo y anchura. En este caso la complejidad anterior queda multiplicada por una constante *k* equivalente al número de descomposiciones que puede llevar a cabo el algoritmo. Por tanto, en el caso peor la complejidad del algoritmo sería $k.n^2$. Sin embargo, en la mayoría de los casos si el valor del parámetro *PH* está bien elegido el coste computacional del algoritmo tiende a n^2, siendo posible identificar operaciones cabecera sin proceder a la descomposición de las operaciones de la especificación.

4.10.2 Identificación de patrones en la especificación conductual

El algoritmo de identificación de patrones recorre las operaciones cabecera, y para cada una de ellas añade tantas operaciones sucesoras como sea posible sin violar el parámetro de calidad *CP*. El caso peor corresponde a aquel en que todas las operaciones de la especificación son operaciones cabecera, y cada patrón está formado por todas las operaciones sucesoras de la operación cabecera. En este caso, siendo *n* el número de operaciones de la especificación el coste computacional del algoritmo sería $n\times(n\text{-}1)$, es decir complejidad cuadrática, $O(n^2)$.

En realidad el caso descrito anteriormente corresponde a una situación que no puede darse, ya que las dependencias de datos entre operaciones impiden que todas las operaciones de la especificación sean operaciones cabecera a partir de las cuales se formen patrones con todas las demás operaciones de la especificación. Si el valor de los parámetros *PH* y *CP* está definido adecuadamente, el coste computacional de este algoritmo estará muy por debajo del coste cuadrático y se aproximará a $O(n)$.

En el caso en que no sea posible identificar nuevos patrones, y deba procederse a la fragmentación de algunas operaciones para aumentar el número de ocurrencias de los patrones, el coste computacional anterior se multiplicaría por una constante *k* cuyo valor corresponde al número de posibles descomposiciones consideradas por el algoritmo propuesto. En definitiva, la complejidad del algoritmo en el caso peor sería del orden $O(k.n^2)$.

4.10.3 Planificación de las ocurrencias de los patrones y de las operaciones residuales

El algoritmo utilizado para la planificación es una variante del algoritmo clásico de planificación dirigida por fuerzas, cuyo coste computacional es cúbico. En el caso de la planificación de las ocurrencias de los patrones el coste es habitualmente inferior debido a que en cada iteración se planifican varias operaciones pertenecientes a un mismo patrón, y adicionalmente el espacio de diseño está reducido a las ocurrencias del patrón. En el caso peor en que todas las operaciones son ocurrencias de un mismo patrón el coste es el del algoritmo clásico, es decir $O(\lambda \cdot n^3)$, siendo λ la latencia del circuito y *n* el número de operaciones de la especificación conductual.

Igualmente, el coste del algoritmo de planificación de operaciones residuales es cúbico, al tratarse básicamente del algoritmo clásico. En ambos casos, la complejidad puede reducirse a $O(n^2)$ aplicando las dos técnicas propuestas en [PaKn89] para reducir la complejidad del algoritmo clásico de $O(\lambda \cdot n^3)$ a $O(n^2)$. A continuación se describen estas técnicas:

1) El factor λ de la complejidad puede eliminarse reduciendo la movilidad de las operaciones que pueden planificarse en un número de ciclos superior a una cierta constante *N*. Las movilidades de estas operaciones se reducen eliminando los ciclos que producen los valores más altos de la *fuerza*, ya que en ellos la probabilidad de planificar la operación es menor. Con esta reducción de las movilidades, para cada operación se calcularían un máximo de *N fuerzas* distintas, y por ello la complejidad del algoritmo propuesto se reduciría así a $O(n^3)$.

2) El orden de complejidad del algoritmo puede también reducirse empleando un método distinto para calcular la *fuerza* de la propia asignación y la de sus antecesoras y sucesoras. Con este método las *fuerzas* de la propia asignación, sus antecesoras y sucesoras se calculan en tres fases separadas. En una primera fase se computan y almacenan las *fuerzas* de cada una de las operaciones. En la segunda fase, se recorre el GFD de abajo arriba y se actualiza el valor almacenado de la *fuerza* sumando el valor de las *fuerzas* de las sucesoras inmediatas. De esta forma se calcula en tiempo lineal la suma de las *fuerzas* de todas las sucesoras de una operación. En la tercera y última fase, se realiza el mismo proceso recorriendo el grafo de arriba abajo para sumar las *fuerzas* de las antecesoras. Con este método la complejidad del algoritmo propuesto se reduce a $O(\lambda \cdot n^2)$.

Finalmente la aplicación conjunta de estas dos técnicas consigue reducir la complejidad del algoritmo a $O(n^2)$.

4.10.4 Selección y asignación de recursos funcionales, de almacenamiento y encaminamiento de datos

El algoritmo de selección y asignación de recursos funcionales para las operaciones de un patrón, selecciona un conjunto de recursos de la biblioteca de diseño y recorre las ocurrencias planificadas del patrón asignando sus operaciones a dichas unidades funcionales. El caso peor corresponde a patrones formados por una operación y donde cada operación de la especificación es una ocurrencia del mismo planificada. En este caso el coste computacional es $O(n)$.

La selección y asignación de recursos funcionales para las operaciones residuales, recorre dichas operaciones comprobando para cada una de ellas si es posible seleccionar un recurso funcional de la ruta de datos para ejecutarla, y en caso contrario selecciona un nuevo recurso de la biblioteca de diseño. Así, el caso peor corresponde a aquel en que todas las operaciones de la especificación sean operaciones residuales y todas deban ejecutarse en

recursos distintos. En este caso, para un número *n* de operaciones en la especificación conductual, la complejidad del algoritmo es $n\times(n-1)$, es decir complejidad cuadrática, $O(n^2)$.

El algoritmo de selección y asignación de recursos de almacenamiento recorre todas las variables que deben almacenarse y para cada una de ellas el conjunto de recursos de almacenamiento hasta encontrar uno disponible, y en caso contrario añade un nuevo recurso a la ruta de datos. Así, el caso peor corresponde a aquel en que las *n* variables de la especificación deban almacenarse en recursos distintos. En este caso, la complejidad del algoritmo es $n\times(n-1)$, es decir complejidad cuadrática, $O(n^2)$.

La selección y asignación de recursos de encaminamiento de datos se realiza recorriendo el circuito sintetizado hasta el momento y añadiendo los multiplexores necesarios a las entradas de las unidades funcionales y de almacenamiento. Siendo *n* el número total de recursos funcionales y de almacenamiento, la complejidad de esta fase sería lineal, $O(n)$.

En definitiva, podemos concluir que el coste computacional del algoritmo de selección y asignación de HW es cuadrático.

4.10.5 Optimización de las unidades funcionales de la ruta de datos

El algoritmo de optimización propuesto recorre las unidades funcionales de la ruta de datos y ensaya su sustitución por los recursos funcionales disponibles en la biblioteca de diseño. En este caso, siendo *n* el número de unidades funcionales de la ruta de datos y *k* el número de recursos de cada tipo diferente presentes en la biblioteca de diseño, la complejidad del algoritmo es $n\times k$. No obstante, en la mayoría de los casos no resulta necesario recorrer todos los recursos presentes en la biblioteca y el coste computacional se aproxima a $O(n)$.

4.11 Conclusiones

En el presente capítulo se ha propuesto un algoritmo completo de SAN que incorpora la metodología de diseño basada en la gestión de patrones y descomposición de operaciones, objeto de esta tesis.

El algoritmo propuesto parte de una especificación conductual, que transforma progresivamente a lo largo del proceso de SAN con el objetivo de identificar nuevos patrones de operaciones y ocurrencias de los mismos. El patrón se vuelve la unidad básica de planificación y selección y asignación de unidades funcionales. La principal ventaja de esta metodología de diseño radica en la reutilización de las mismas unidades funcionales entre las distintas ocurrencias del patrón, lo que redunda en una reducción del área ocupada por los recursos funcionales. Sin embargo, esta ventaja no es única, como se ha explicado en el capítulo la metodología de diseño propuesta provoca adicionalmente una reducción de los recursos de almacenamiento y de encaminamiento de datos.

Parte del éxito de la metodología de SAN propuesta se encuentra en la selección en cada momento del patrón de mayor calidad, es decir en la métrica utilizada para medir la calidad de los patrones. Esta métrica tiene en cuenta las principales características de las ocurrencias de los patrones. Al cuantificar diferentes aspectos o características de los patrones, se obtiene una medida eficaz de su calidad que permite comparar unos patrones con otros, y seleccionar en cada momento aquel cuya planificación produzca los mayores ahorros de área en el circuito resultante.

CAPÍTULO

5

CASO PRÁCTICO Y RESULTADOS EXPERIMENTALES

En la primera parte de este capítulo se va a llevar a cabo una aplicación práctica del algoritmo de síntesis propuesto. De una forma gráfica se aplicarán cada una de las fases descritas en los capítulos anteriores a una descripción conductual modelo. Paso a paso se comprobará cómo se modifica y procesa dicha descripción aplicando las técnicas de síntesis propuestas con el objetivo de reducir el área necesaria para implementar la descripción conductual dada.

Posteriormente, se presentan los principales resultados experimentales de las pruebas realizadas aplicando el algoritmo de SAN propuesto. Estos resultados se comparan con los obtenidos por otros algoritmos convencionales de SAN, y se analizan las mejoras conseguidas con el método presentado.

5.1 Caso práctico

La descripción conductual que se va a sintetizar a modo de ejemplo está formada por varias operaciones de diferentes tipos y anchuras. En la tabla 5.1 se presentan las operaciones que constituyen esta descripción conductual, y en la figura 5.1 se presenta el GFD correspondiente a la misma.

E:128	=	**A:64**	x	**B:64**
I:192	=	E:128	x	**F:64**
Q:192	=	I:192	+	J:192
K:96	=	**G:32**	x	**H:64**
N:64	=	**C:64**	+	**D:64**
O:96	=	K:96	+	L96
P:96	=	**M:32**	x	N:64
R:96	=	O:96	+	P:96

Tabla 5.1 *Operaciones, operandos y anchuras de la descripción conductual a sintetizar mediante la técnica propuesta*

5.1.1 Planificación convencional de la descripción conductual

Como primera aproximación de planificación se realiza una planificación para una latencia igual a cinco ciclos de reloj. Partiendo del tiempo de ejecución mínimo de la conducta especificada, correspondiente a la implementación monociclo de la misma, es posible estimar un tiempo de ciclo mínimo a partir del cual realizar la síntesis del circuito. Dada esta restricción de tiempo, el objetivo del proceso de síntesis es reducir el área de la implementación resultante, sin sobrepasar el tiempo máximo fijado por el valor de la latencia y del tiempo de ciclo.

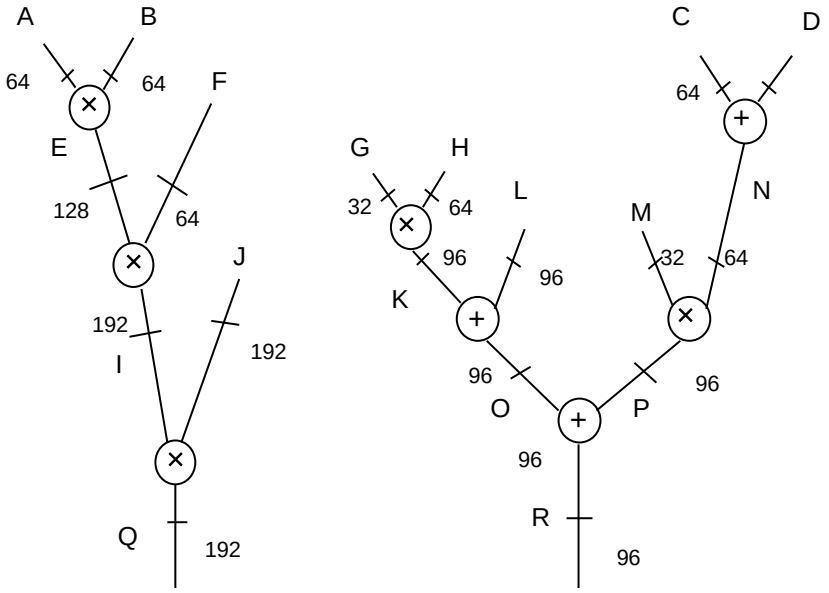

Figura 5.1 *GFD de la descripción conductual.*

El tiempo de ciclo calculado en nuestro caso es inferior al tiempo de ejecución de algunas operaciones de la especificación, por lo que resulta indispensable su ejecución en unidades funcionales multiciclo. En la figura 5.2 se presenta una posible planificación utilizando unidades funcionales multiciclo para ejecutar las multiplicaciones y sumas más lentas de la especificación.

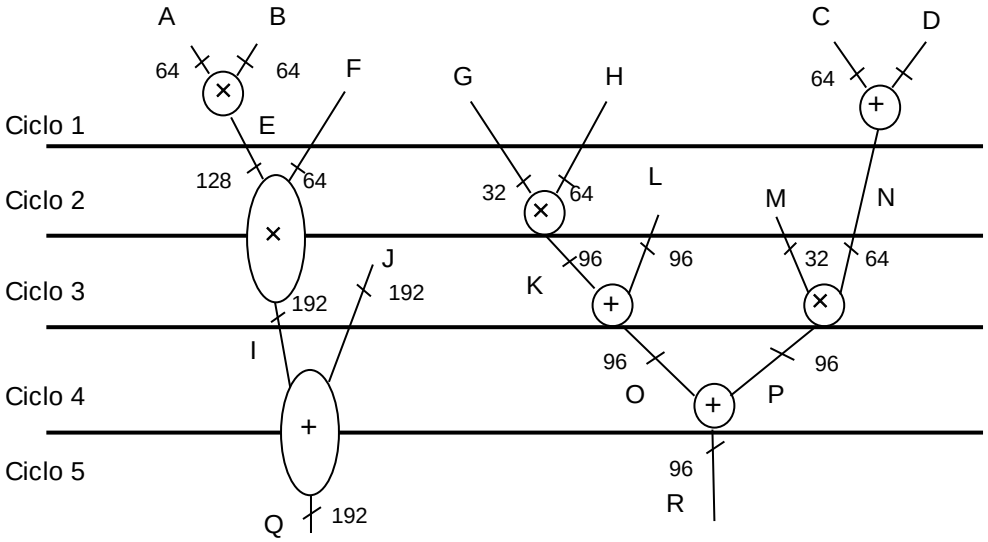

Figura 5.2 *Planificación convencional con unidades funcionales multiciclo*.

5.1.2 Descomposición de las operaciones multiciclo

En las primeras fases del algoritmo de síntesis propuesto se procede a identificar las operaciones multiciclo comparando su tiempo de ejecución en las unidades funcionales más rápidas de la biblioteca de diseño, con la longitud del ciclo de reloj. Aplicando el algoritmo al GFD propuesto es posible identificar dos operaciones multiciclo. En concreto, el conjunto *MC* de unidades multiciclo pasaría a estar formado por las siguientes operaciones:

1) *Multiplicación de dos operandos de anchuras 128 y 64 bits. Se corresponde con el cálculo de la variable I de 192 bits de la descripción conductual dada.*

2) *Suma de dos operandos de anchuras 192 bits. Se corresponde con el cálculo de la variable Q de 192 bits de la descripción conductual dada.*

Una vez identificadas las operaciones multiciclo se pasa a aplicar las técnicas de descomposición de operaciones utilizando la función *FragmentosMonociclo*, descrita en los capítulos anteriores.

5.1.2.1 Descomposición de la multiplicación

A la multiplicación multiciclo I:192 = E:128 x F:64, se le aplica la técnica de descomposición recogida en la figura 5.3. Esta técnica divide la multiplicación en cuatro nuevas operaciones o fragmentos, que se añaden al conjunto de operaciones de la especificación sustituyendo a la multiplicación original. Las cuatro nuevas operaciones se corresponden son:

1) *Dos multiplicaciones de anchura 32 x 64 bits.*

2) *Una operación TDown de 64 bits.*

3) *Una operación TUp de 64 bits.*

Estas cuatro nuevas operaciones se completan con dos sumas de 64 bits siendo la función de cada una de ellas la siguiente:

1) *Una primera adición para sumar el resultado de la operación TUp con el resultado de la primera multiplicación.*

2) *Una segunda adición para sumar el resultado de la segunda multiplicación con el resultado de la operación TDown y el acarreo resultante de la suma anterior.*

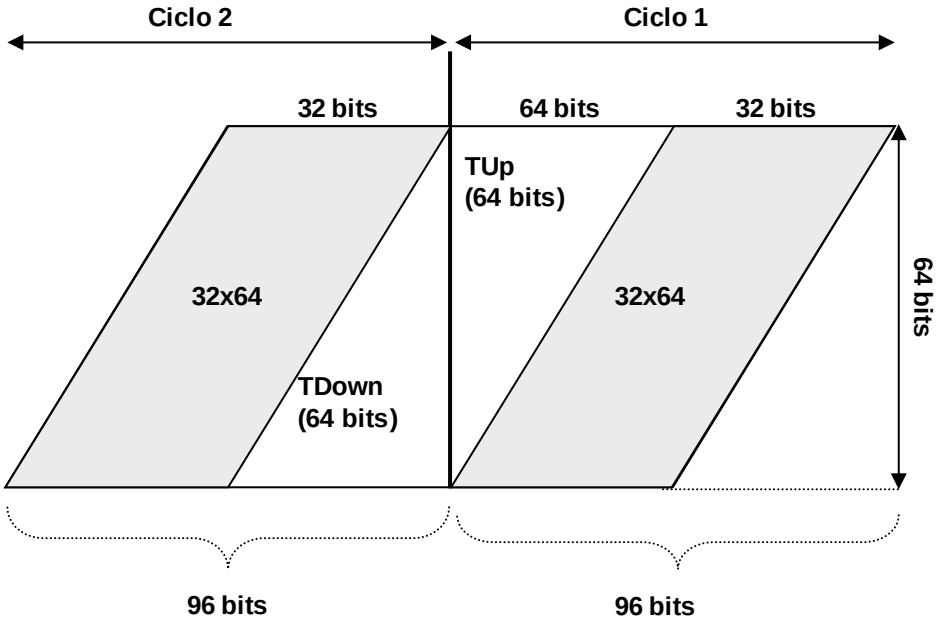

Figura 5.3 *Descomposición de una multiplicación multiciclo.*

5.1.2.2 Descomposición de la suma

A la suma multiciclo Q:192 = I:192 + J:192, se le aplica la técnica de descomposición recogida en la figura 5.4. Esta técnica divide la adición en dos nuevas operaciones o fragmentos del tipo suma. Estas dos nuevas operaciones se añaden al conjunto de operaciones de la especificación sustituyendo a la suma multiciclo que calcula la variable *Q* de 192 bits. Las dos nuevas operaciones se corresponden con dos sumas de anchura 96 bits en las que el acarreo de salida de la primera suma actúa como acarreo de entrada en la segunda.

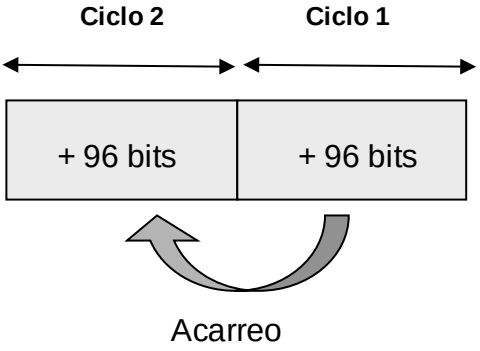

Figura 5.4 *Descomposición de una operación aditiva multiciclo.*

5.1.2.3 Especificación conductual tras el proceso inicial de descomposición

Tras la descomposición de las dos operaciones multiciclo, la multiplicación I:192 = E:128 x F:64 y la suma Q:192 = I:192 + J:192, la especificación conductual se ha transformado en otra con mayor número de operaciones, siendo posible la ejecución de todas ellas en un único ciclo de reloj. El GFD correspondiente a la nueva especificación se muestra en la figura 5.5.

La nueva especificación conductual es equivalente a la original, habiéndose sustituido las operaciones multiciclo por varias operaciones encadenadas monociclo. El resto de operaciones originales se mantienen.

5.1.3 Cálculo de patrones

Una vez que se ha llevado a cabo la primera fase de descomposición de operaciones multiciclo, el algoritmo de síntesis propuesto comienza la fase de cálculo de patrones. Para ello, realiza un análisis de las operaciones que forman parte del GFD actual, que se encuentra formado por:

1) *Operaciones monociclo presentes en la especificación original.*

2) *Nuevas operaciones o fragmentos resultantes de la descomposición de las operaciones multiciclo presentes en la especificación original.*

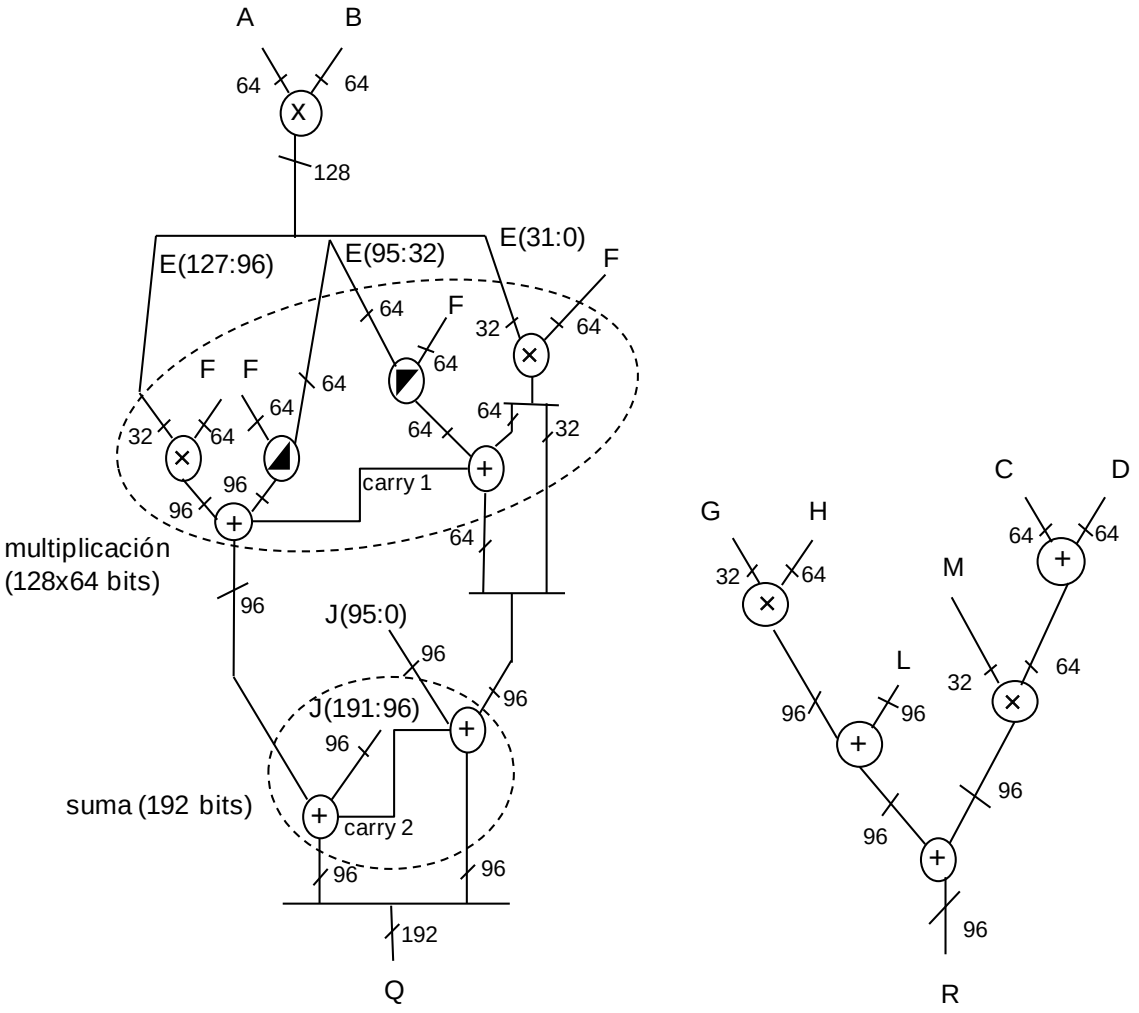

Figura 5.5 *GFD tras la descomposición de las operaciones multiciclo.*

5.1.3.1 Propiedad de calidad *PH*

Se analizan las operaciones del GFD actual seleccionándose aquellas que cumplan la propiedad de calidad *PH*, es decir, que su número de ocurrencias en la especificación actual sea igual o superior al valor del parámetro *PH* fijado por el diseñador. Cuanto más exigente sea el valor de *PH*, más ahorro de área se producirá en el circuito resultante del proceso de síntesis. Pero si su valor es excesivamente exigente, se puede llegar a una situación en que el proceso de síntesis no descubra ningún patrón que cumpla las propiedades de

calidad definidas, y por tanto no puedan obtenerse resultados de síntesis buenos.

En nuestro ejemplo, asignamos a *PH* el valor de la latencia definida por el diseñador como restricción en el proceso de síntesis. Por tanto, para que una operación pueda convertirse en operación cabecera de patrón deberá tener un número de ocurrencias igual o superior a *PH*, es decir, igual o superior a la latencia definida por el diseñador (cinco ciclos en el ejemplo propuesto).

En el GFD mostrado en la figura 5.5 puede comprobarse que no hay ninguna operación que satisfaga la propiedad *PH* ya que ninguna operación tiene un número de ocurrencias igual o superior a cinco. Es necesario recordar que al hablar de ocurrencias de una operación, dichas ocurrencias deben coincidir en:

1) *Tipo de las operaciones.*

2) *Anchuras de sus operandos tanto de entrada como de salida.*

En la figura 5.5, la multiplicación de anchura 32x64 bits tiene un número de ocurrencias igual a cuatro. Estas ocurrencias aparecen resaltadas mediante circunferencias punteadas en la figura 5.6. Las cuatro ocurrencias de la operación tienen orígenes distintos:

1) *Dos ocurrencias estaban en el GFD original. Son las resaltadas en la figura 5.6 por una circunferencia doble.*

2) *Dos ocurrencias han aparecido en el GFD actual como resultado del proceso de descomposición aplicado a las operaciones multiciclo. En concreto se han generado en el proceso de descomposición de la operación multiciclo original I=ExF indicado en el apartado 5.1.2.1.*

En este primer intento de identificar patrones no se ha podido localizar ningún patrón candidato debido a que ninguna operación del GFD actual cumple la propiedad de calidad *PH*. Es decir, no se han podido definir operaciones cabecera, y por ello es necesario pasar a identificar nuevas

operaciones que puedan llegar a ser operaciones cabecera aplicando nuevamente técnicas de descomposición de operaciones, en este caso a las operaciones monociclo de la especificación.

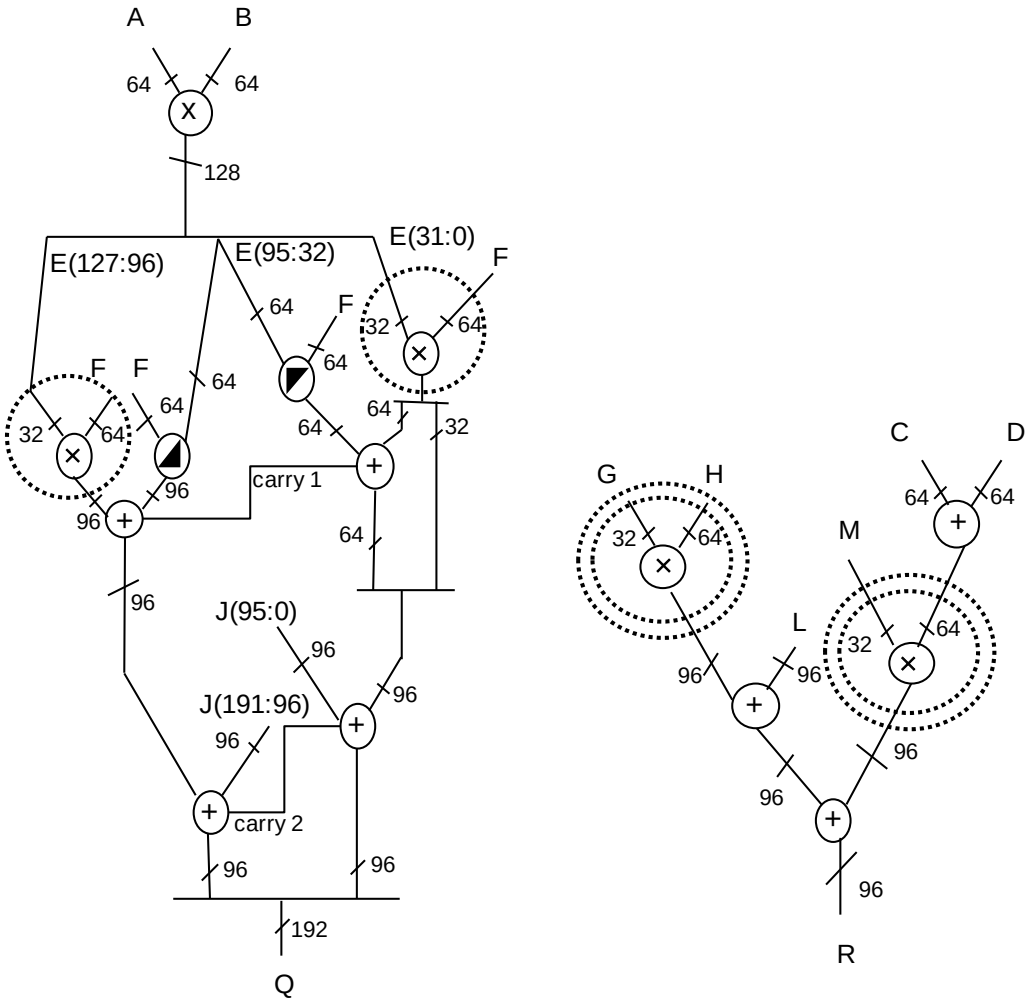

Figura 5.6 *La multiplicación de 32x64 bits no satisface PH=5.*

5.1.3.2 Identificación de nuevas operaciones cabecera

Para que aumente el número de las ocurrencias de las operaciones actuales, es necesario proceder a descomponer algunas de las operaciones de la especificación actual. Antes de descomponer las operaciones, el algoritmo ensaya si las posibles descomposiciones aumentan las ocurrencias de otras operaciones hasta el punto de convertirse en operaciones cabecera de

patrón. Si el resultado es positivo, se procede a ejecutar las descomposiciones ensayadas.

La figura 5.6 muestra el GFD actual, en el que se ha resaltado una multiplicación de la especificación con cuatro ocurrencias. En el intento de aumentar el número de ocurrencias de esta operación, el algoritmo explora la descomposición de la multiplicación E:128 = A:64 x B:64, comprobando que se puede descomponer aplicando las técnicas disponibles y que de esa descomposición se obtienen dos nuevas operaciones multiplicativas de la misma anchura que la resaltada en la figura.

Si se llevara a efecto la descomposición indicada, el número de ocurrencias de la operación multiplicativa objeto de estudio pasaría de cuatro a seis ocurrencias. Con este incremento, esta operación pasaría a cumplir la propiedad de calidad *PH*. Tras esta exploración y tras comprobar el éxito de la descomposición, el algoritmo procede a ejecutar la descomposición de forma real. La descomposición realizada se muestra en la figura 5.7.

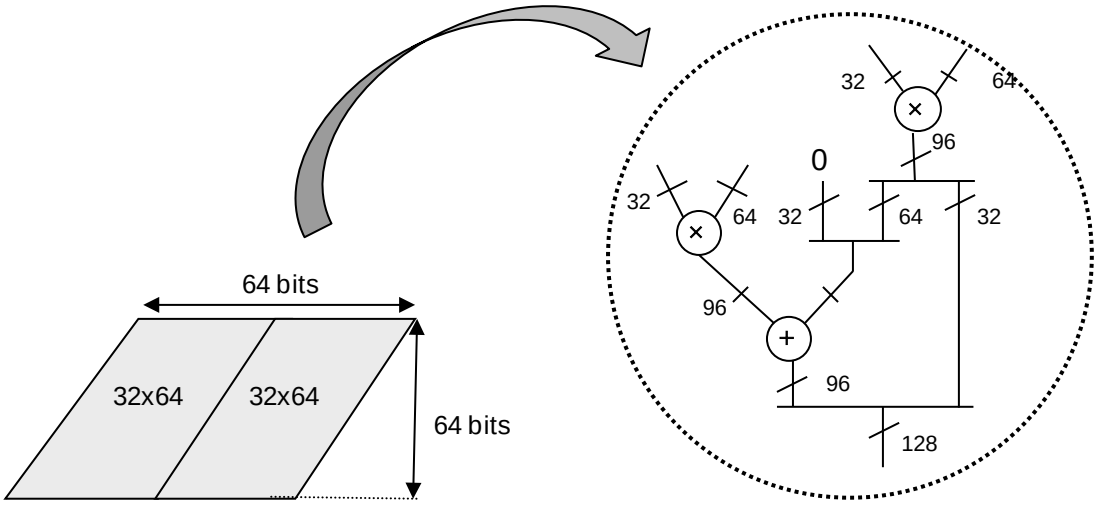

Figura 5.7 *Descomposición de una multiplicación monociclo.*

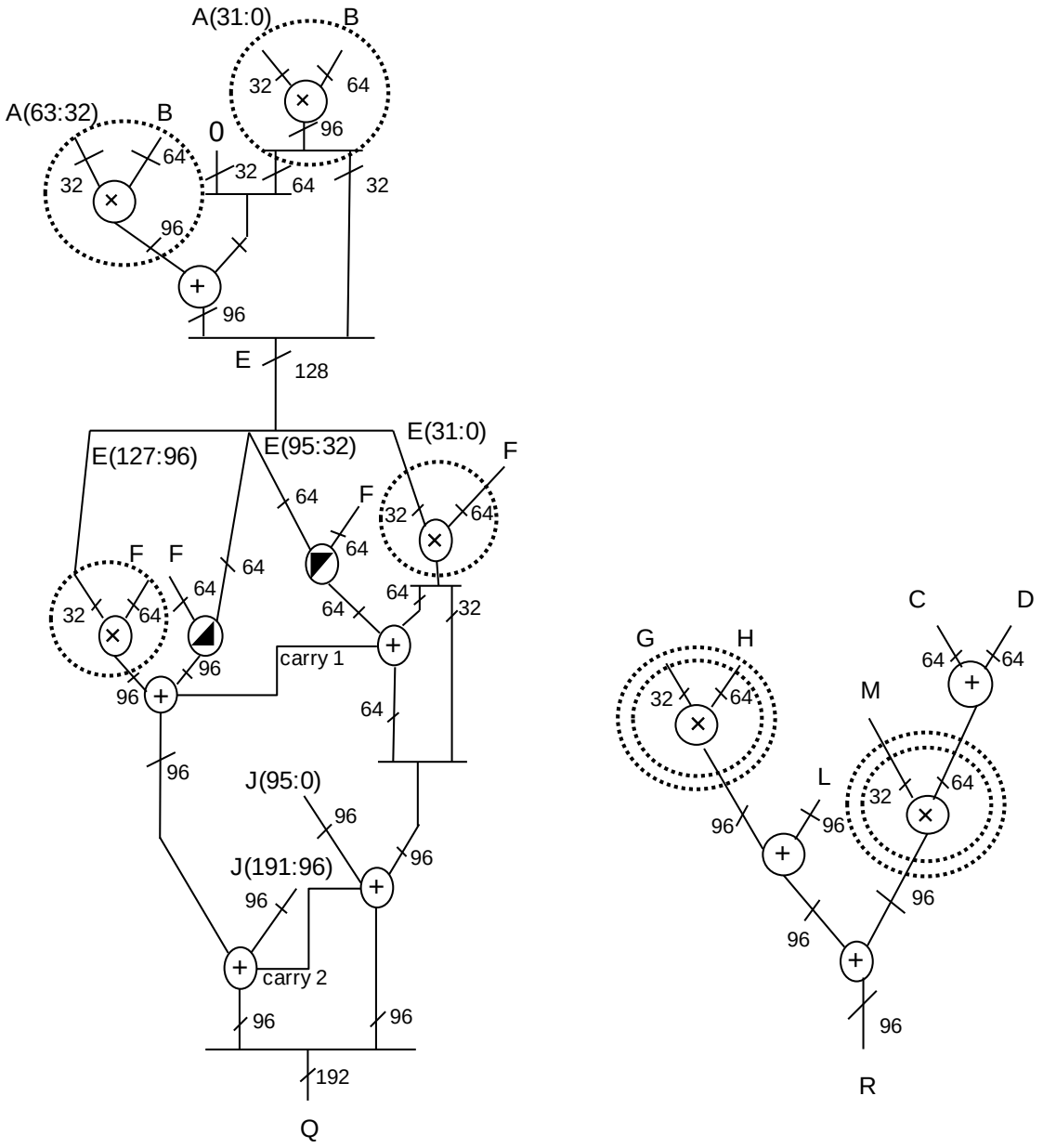

Figura 5.8 *GFD previo a la identificación de patrones.*

5.1.3.3 Identificación de patrones

A continuación tiene lugar la fase de identificación de patrones a partir de las operaciones cabecera (operaciones que cumplen la propiedad de calidad *PH*). La figura 5.8 ilustra el GFD actualizado tras la descomposición de la multiplicación que ha dado lugar a la identificación de una operación

cabecera de patrón. En dicha figura se diferencian las ocurrencias de la operación cabecera que ya figuraban en el GFD inicial (circunferencia doble) y las ocurrencias que se han generado en los procesos de descomposición ejecutados (circunferencia simple).

A partir de la operación cabecera comienza el proceso de identificación de patrones. El primer patrón que intenta validar es la propia operación cabecera. Para ello se compara el número de ocurrencias de dicha operación con el valor del parámetro *CP*. Tras las diversas pruebas realizadas en la fase experimental se ha comprobado que los resultados de mayor calidad se obtienen para valores de *CP* iguales a un 70% del valor de *PH*. Aplicando esto a nuestro caso particular, para que un patrón cumpliera la propiedad de calidad *CP* debería tener un número de ocurrencias igual o superior a cuatro.

El patrón formado por la propia operación cabecera tiene un número de ocurrencias igual a seis por lo que se convierte en un patrón candidato.

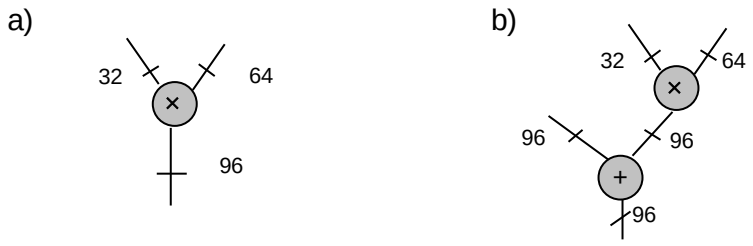

Figura 5.9 *Patrones identificados a partir de la operación cabecera.*
a) Patrón formado por la operación cabecera solamente.
b) Patrón formado por la operación cabecera y una operación sucesora.

Continuando la exploración del GFD a partir del patrón anterior (formado sólo por la operación cabecera), se identifica un nuevo patrón reflejado en la figura 5.9. Este patrón está formado por la operación multiplicativa anterior definida como operación cabecera y su operación aditiva sucesora. El nuevo patrón tiene un número de ocurrencias en el GFD de cinco por lo que también cumple la propiedad de calidad *CP* (valor mínimo cuatro). En la figura 5.10 se

presenta el GFD actual y remarcadas las ocurrencias del nuevo patrón. Las cinco ocurrencias de este nuevo patrón tienen los siguientes orígenes distintos:

1) *Las dos ocurrencias remarcadas por circunferencias de trazo tipo raya-punto se encontraban en el GFD original.*

2) *Las dos ocurrencias remarcadas por circunferencias de trazo tipo punto, aparecieron como resultado del proceso de descomposición de las operaciones multiciclo detectadas en las primeras fases de ejecución del algoritmo de síntesis.*

3) *La ocurrencia remarcada por una circunferencia de trazo continuo se obtuvo del proceso de descomposición de una operación monociclo realizado con el fin de encontrar operaciones cabecera de patrón.*

De los dos patrones identificados, el algoritmo selecciona el de mayor calidad. Para ello se aplica la definición de calidad propuesta en el capítulo 4. Mediante la aplicación de dicha fórmula, se obtiene que el patrón de más calidad es el formado por dos operaciones. Las ocurrencias de este patrón son las remarcadas mediante elipses en la figura 5.10.

5.1.4 Selección de unidades funcionales

Las unidades funcionales seleccionadas para ejecutar las operaciones de un patrón serán reutilizadas por las distintas ocurrencias de dicho patrón en el GFD, siempre y cuando puedan planificarse en ciclos distintos como ocurre en nuestro ejemplo. El nivel de ahorro alcanzado depende de la planificación realizada, que se detalla más adelante.

Para realizar la selección de las unidades funcionales que ejecutarán las operaciones del patrón se dispone de una biblioteca de diseño con recursos funcionales capaces de calcular todas las operaciones de la especificación. En el caso de existir varias unidades funcionales que puedan implementar una determinada operación, se escoge siempre la unidad más rápida. Con el objeto de simplificar nuestro ejemplo, supondremos que sólo se encuentra

disponible en la biblioteca de diseño una unidad funcional de cada tipo y anchura diferentes.

Como resultado de esta fase se obtiene un conjunto de unidades funcionales capaces de ejecutar las operaciones del patrón.

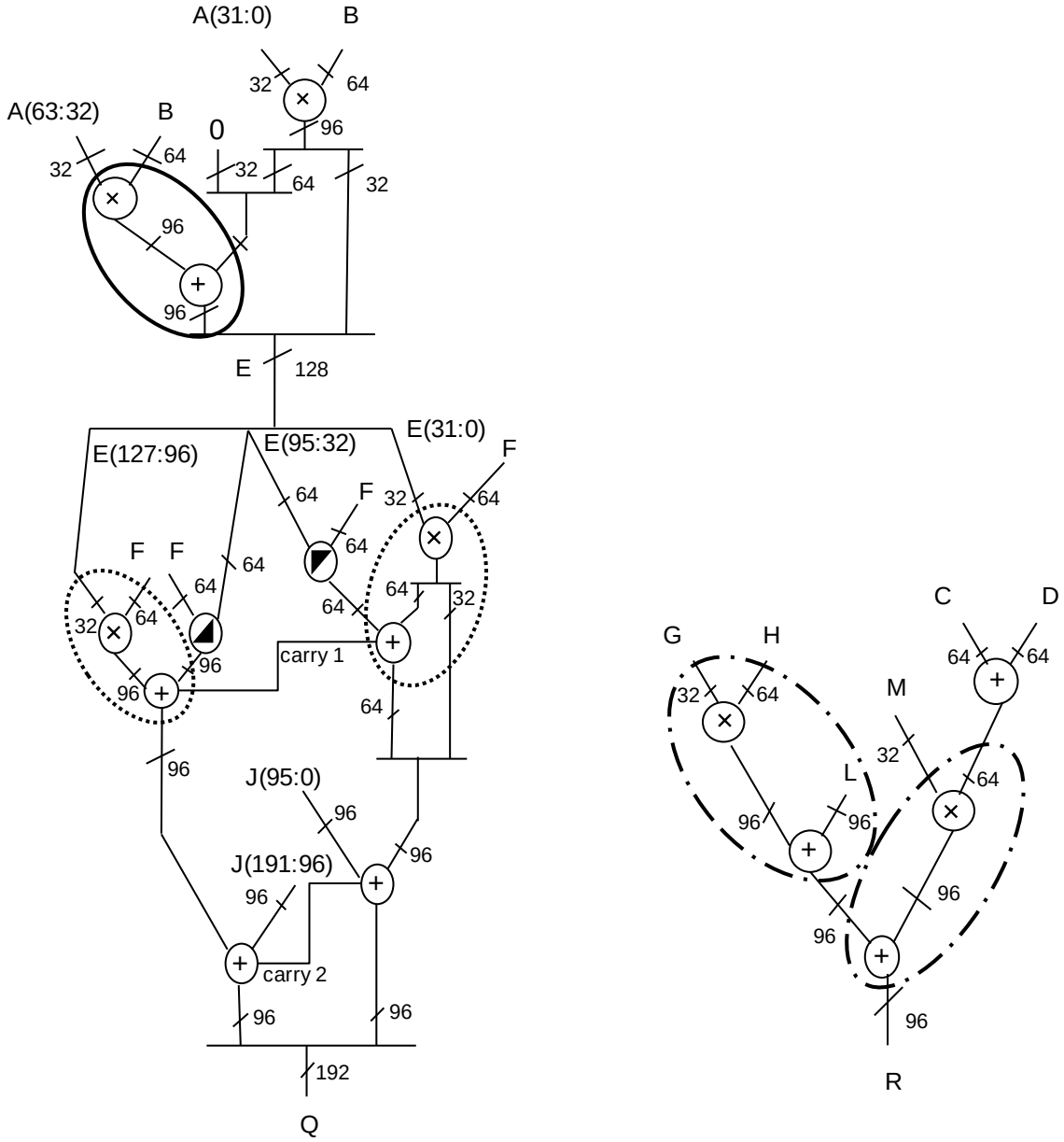

Figura 5.10 *GFD, patrón identificado y ocurrencias del patrón.*

5.1.5 Planificación obtenida por el algoritmo de síntesis propuesto

Una vez seleccionadas las unidades funcionales correspondientes, comienza la fase de planificación de las ocurrencias del patrón. Tal como se explicó en el capítulo anterior para planificar las ocurrencias del patrón se utiliza un algoritmo de fuerzas adaptado a la gestión de patrones, y que se ha denominado PFDS (Pattern Force Direct Scheduling).

En nuestro ejemplo las movilidades de las ocurrencias del patrón hacen posible planificar cada ocurrencia en un ciclo distinto, y por tanto se planifican todas ellas, no quedando ocurrencias del patrón sin planificar. Una vez planificadas las ocurrencias no quedan patrones candidatos y se procede a identificar nuevas operaciones cabecera. En este pequeño ejemplo, esta identificación no tiene éxito y las operaciones restantes pasan a considerarse operaciones residuales.

A continuación tiene lugar la planificación de las operaciones residuales. En la figura 5.11 se presenta la planificación final, incluidas las operaciones residuales que no pertenecen a los patrones planificadas. Estas operaciones residuales se corresponden tanto con operaciones del GFD original, como con operaciones resultantes de las descomposiciones de otras operaciones.

Tal como se comprueba en la figura 5.11, la ejecución de las operaciones que han sustituido a las operaciones multiciclo originales se realiza en los mismos ciclos que las operaciones multiciclo en la planificación convencional mostrada en la figura 5.2. Sin embargo, en este segundo caso el área necesaria es mucho menor que en el circuito convencional.

En particular, la planificación de las nuevas operaciones o fragmentos resultantes de descomponer la multiplicación multiciclo es la siguiente:

1) *Ciclo 2*. Se ejecutan en paralelo la operación TUp de operandos E(95:32) y F(63:0), y la multiplicación de E(31:0) y F(63:0). Se suman los resultados parciales de las dos operaciones anteriores (los 64 bits de más peso de la multiplicación y el resultado de la operación TUp). El acarreo de esta suma se utilizará más adelante. La concatenación de los 32 bits menos

significativos del resultado de la multiplicación y los 64 bits del resultado de la suma producen los 96 bits menos significativos del resultado de la multiplicación original, es decir, I(95:0).

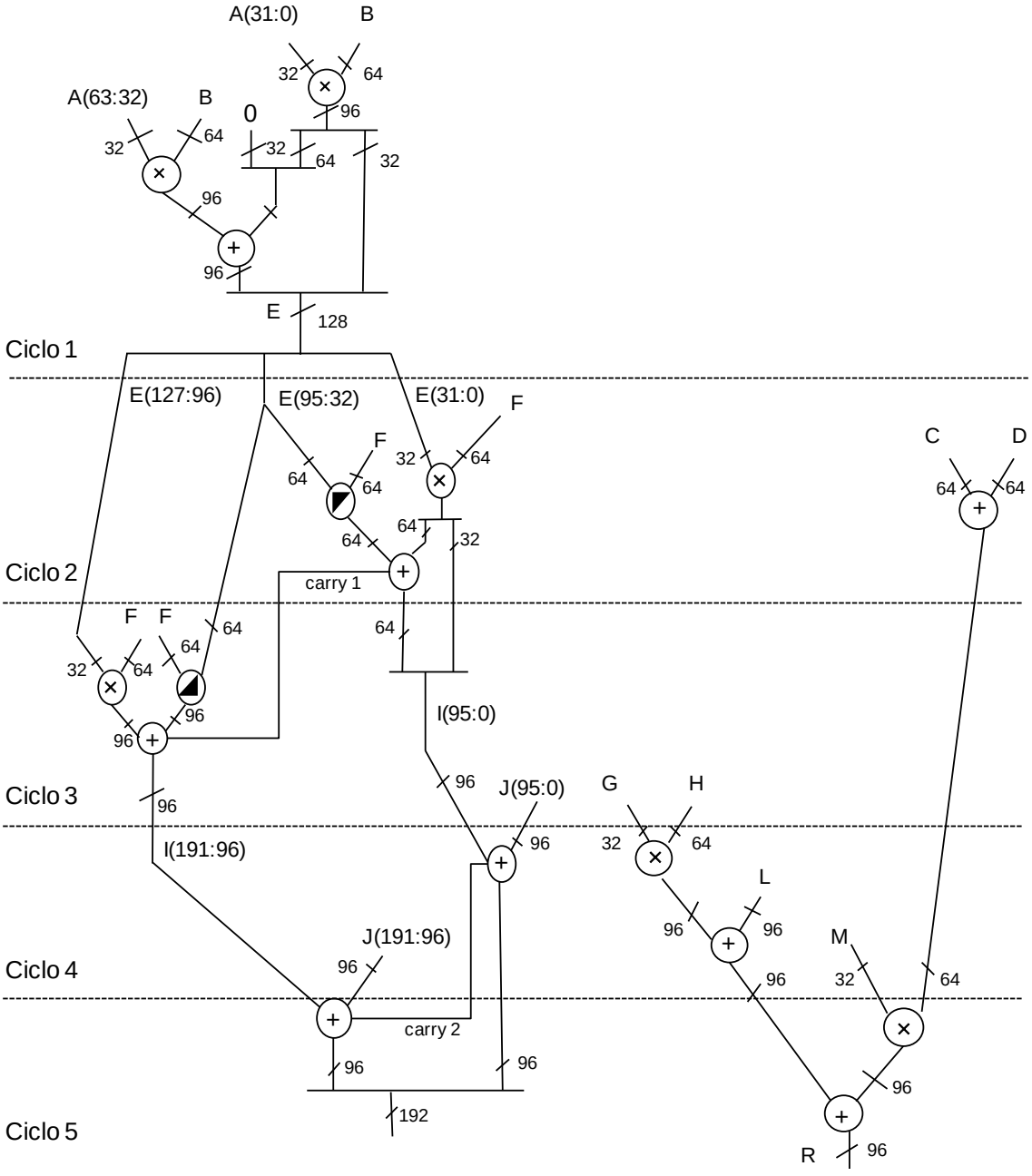

Figura 5.11 *Planificación final resultante del algoritmo de síntesis propuesto.*

2) *Ciclo 3*. Se ejecutan en paralelo la multiplicación de E(127:96) y F(63:0), y la operación TDown de E(95:32) y F(63:0). Se suman los resultados parciales de 96 bits de ambas operaciones y el posible acarreo de la suma del ciclo anterior. El resultado obtenido se corresponde con los 96 bits más significativos de la multiplicación original, es decir, I(191:96).

La planificación de las nuevas operaciones aditivas o fragmentos resultantes de descomponer la suma multiciclo, según el esquema de la figura 5.4 se corresponde con:

1) *Ciclo 4. S*e ejecuta la suma de los 96 bits menos significativos de los dos sumandos *I* y *J*.

2) *Ciclo 5.* Se ejecuta la suma de los 96 bits más significativos de los dos sumandos *I* y *J*, y el acarreo producido en ciclo anterior.

5.1.6 Unidades de almacenamiento y encaminamiento de datos

La implementación propuesta también supone un ahorro de área en cuanto a los elementos de almacenamiento y de encaminamiento de datos. El ahorro en los elementos de almacenamiento de datos viene dado por varios factores. En primer lugar, se producen ahorros en unidades de almacenamiento al ejecutar todas las operaciones del patrón de manera encadenada en un solo ciclo. En estos casos no son necesarias unidades de almacenamiento ya que la salida de una operación alimenta directamente la entrada de la siguiente operación del patrón. Este es el caso de las dos ocurrencias del patrón procesado que figuraba en el GFD original. En la situación original eran necesarias unidades de almacenamiento para guardar los valores intermedios. En la figura 5.12 se presenta esta situación.

En segundo lugar, al descomponer las operaciones la mayoría de los fragmentos resultantes no necesitan los operandos completos, por lo que habitualmente sólo es necesario almacenar parte de ellos, reduciendo así los requerimientos de elementos de almacenamiento.

En cuanto a los recursos de encaminamiento de datos, al estar las unidades funcionales que ejecutan las operaciones del patrón encadenadas no hay necesidad de elementos de transmisión de datos entre ellas.

En general, cuanto mayor sea el número de patrones y ocurrencias de los mismos, mayor será el ahorro de área no sólo de unidades funcionales, sino también de unidades de almacenamiento y de encaminamiento de datos obtenido por el algoritmo de síntesis propuesto en comparación con las implementaciones convencionales.

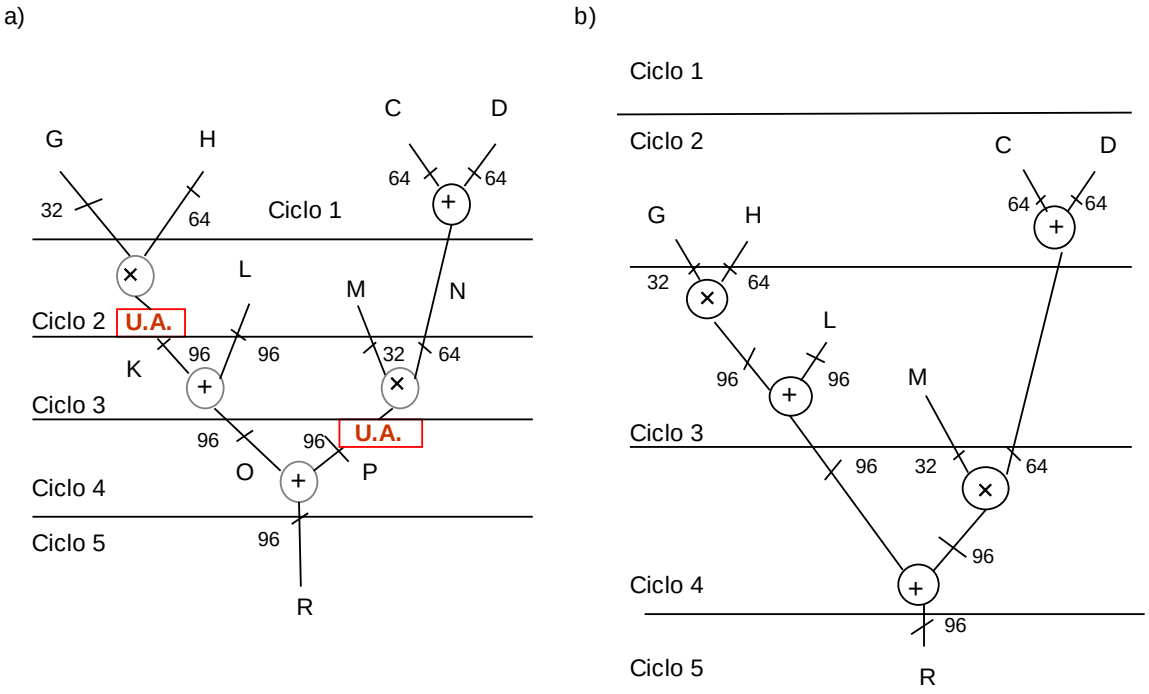

Figura 5.12 *Ahorro en unidades de almacenamiento.*
a) Planificación convencional: unidades de almacenamiento entre las operaciones x, +.
b) Planificación del algoritmo de síntesis propuesto: no se requieren unidades de almacenamiento entre las operaciones x, + de las ocurrencias del patrón.

5.1.7 Optimización del área del circuito resultante

Con el objeto de simplificar el ejemplo propuesto se ha supuesto una biblioteca de diseño con un único tipo de recurso funcional para cada operación diferente, por lo que esta fase del algoritmo carece de sentido en nuestro caso.

5.1.8 Conclusiones

El algoritmo propuesto parte de una especificación conductual y la va transformando mediante sucesivas descomposiciones de operaciones con el objeto de reducir el área de la ruta de datos resultante. El mayor número de operaciones de la especificación con la que trabaja el algoritmo propuesto incrementa la probabilidad de identificar operaciones cabecera y patrones dentro de la especificación. También aumenta el número de ocurrencias de los patrones detectados, y por tanto la reutilización de las unidades funcionales que ejecutan las operaciones del patrón, con la consiguiente reducción de elementos de almacenamiento e interconexión de datos entre las unidades funcionales asignadas al patrón. En general, cuantos más patrones se descubran en el GFD y cuantas más ocurrencias tengan dichos patrones, mayor será el ahorro total de área.

Debido a las descomposiciones de operaciones, las implementaciones generadas por el algoritmo de síntesis propuesto presentan, en general, un mayor número de unidades funcionales que las implementaciones convencionales. No obstante, estas unidades funcionales ocupan un área significativamente menor que en el caso de los circuitos sintetizados por algoritmos convencionales. En nuestro ejemplo, el beneficio obtenido alcanza el 40% de ahorro en el área total del circuito. Esta reducción de área es debida a la reutilización de las unidades funcionales entre las ocurrencias del patrón, y a una reducción en las necesidades de unidades de almacenamiento y de interconexión de datos entre las unidades funcionales que ejecutan las operaciones del patrón.

El ahorro en área no ha repercutido en el tiempo de ejecución de la conducta especificada, que es similar en ambos escenarios: implementación convencional e implementación obtenida por el algoritmo de síntesis propuesto. Las áreas y los tiempos de ejecución de ambos circuitos se han medido después de la síntesis lógica utilizando la herramienta comercial *Synopsys Design Compiler*. En los resultados comparativos están incluidas la ruta de datos completa y la unidad de control.

5.2 Resultados experimentales

5.2.1 Introducción

Para medir la calidad de la metodología de SAN propuesta, se ha llevado a cabo la síntesis de varios circuitos. Con el fin de comparar los resultados, a cada especificación conductual se le ha aplicado un conjunto de técnicas de síntesis, incluida la propuesta en esta memoria. Los procesos de SAN llevados a cabo son:

1) *SAN de descripciones conductuales utilizando la técnica propuesta basada en descomposición de operaciones y gestión de patrones.*

2) *SAN de descripciones conductuales utilizando algoritmos de planificación y selección y asignación de HW a nivel de bit, especializados en la reducción del área propuestos por M. Molina et al [MoMH03a].*

3) *SAN de descripciones conductuales aplicando el algoritmo tradicional FDS en la fase de planificación y un algoritmo de selección y asignación de unidades funcionales basado en patrones propuesto por J. Cong et al [ChCF03].*

Las rutas de datos obtenidas por las tres técnicas de SAN utilizadas han sido tratadas de la siguiente manera:

1) *Síntesis lógica de los circuitos obtenidos utilizando la herramienta comercial Synopsys Design Compiler (DC) y la librería de componentes VTVTLIB25*[1] *basada en componentes de tecnología 0,25 µm TSMC*[2]*.*

2) *A partir de los informes posteriores a la síntesis, generados por DC para cada una de las implementaciones, se han obtenido los tiempos de ejecución y las áreas utilizadas en cada caso. Estas áreas incluyen todos los*

[1] VTVTLIB25: Librería de unidades funcionales mantenida por Virginia Tech VLSI for Telecommunication.
[2] TSMC: Taiwan Semiconductor Manufacturing Company Limited.

elementos de la ruta de datos (unidades funcionales, unidades de almacenamiento, multiplexores y lógica pegamento) y la unidad de control. Los tiempos de ejecución se han medido en nanosegundos y las áreas en número de puertas equivalentes (inversores).

Aprovechando la flexibilidad que el algoritmo propuesto proporciona al diseñador, y con el fin de maximizar el reuso de los recursos HW de las rutas de datos, en los experimentos realizados se han tomado las siguientes decisiones iniciales:

1) *El parámetro de calidad PH se ha fijado a un valor igual a la latencia del circuito. De esta forma, para que una operación pueda convertirse en cabecera de patrón debe aparecer en la descripción conductual, inicial o intermedia, un número de veces igual o superior al número de ciclos fijado por el diseñador.*

2) *El parámetro de calidad CP se ha fijado al valor obtenido de multiplicar 0,7 por la latencia. Esta decisión implica que para que un patrón pueda ser considerado patrón candidato, debe tener un número de ocurrencias en la descripción conductual, inicial o intermedia, igual o mayor al setenta por cien del número de ciclos fijado por el diseñador.*

Estas decisiones no han sido arbitrarias sino que están basadas en la ejecución de múltiples pruebas variando ambos parámetros de calidad. Tras la realización de dichas pruebas preliminares y el análisis de los resultados obtenidos, se llegó a la conclusión de que en la mayoría de los casos con los valores indicados para los parámetros de calidad *PH* y *CP* se obtenían resultados muy satisfactorios en cuanto al área de los circuitos sintetizados. Esta reducción del área no implica aumentos significativos de los tiempos de ejecución. Por el contrario, en todos los experimentos realizados se han obtenido tiempos de ejecución muy similares a los obtenidos por los otros algoritmos de SAN utilizados.

A continuación se presentan los distintos escenarios experimentales llevados a cabo aplicando las tres técnicas de síntesis indicadas:

1) *Influencia de la longitud del ciclo de reloj en el área necesaria para implementar la especificación conductual.*

2) *Bondad de la técnica de identificación y gestión de patrones propuesta frente a otras técnicas basadas en patrones.*

3) *Uso de unidades funcionales multiciclo frente a unidades funcionales monociclo.*

5.2.2 Longitud del ciclo de reloj vs área

Las técnicas de SAN con restricción de tiempo producen resultados muy influenciados por los valores seleccionados como límite de tiempo. Tanto las restricciones de tiempo que fijan anchuras de ciclo pequeñas como las que fijan anchuras de ciclo grandes, influyen negativamente en el área de los circuitos resultantes.

A continuación se van a analizar las consecuencias negativas en cada unos de los dos casos.

5.2.2.1 Valores bajos del ciclo de reloj

Si el diseñador opta por aplicar las técnicas de SAN con restricción de tiempo definiendo una duración del ciclo de reloj pequeña, está optando en consecuencia por utilizar una latencia alta. En esta situación el número de operaciones que no podrán resolverse en un solo ciclo con las unidades funcionales disponibles en la librería de diseño aumentará. Por lo tanto, el número de operaciones multiciclo será elevado.

En términos teóricos se podría llegar a afirmar que el número de operaciones multiciclo en una descripción conductual es inversamente proporcional a la duración del ciclo de reloj. Esto nos podría llevar a afirmar también que el número de operaciones multiciclo es proporcional a la latencia definida.

Aunque las ventajas del uso de las unidades funcionales multiciclo han sido ampliamente desarrolladas en la historia de la SAN, también existen

ciertas desventajas. La principal es el bajo índice de reutilización que tienen las unidades funcionales multiciclo, debido a que en ningún caso se utiliza todo el HW de la unidad funcional.

5.2.2.2 Valores altos del ciclo de reloj

El caso contrario al descrito en el apartado anterior corresponde a la situación en que el diseñador opta por definir una latencia baja, es decir, un número reducido de ciclos. En este escenario, y como consecuencia de la decisión indicada, la duración del ciclo de reloj es elevada, lo que permite aplicar la técnica clásica de encadenamiento de operaciones con dependencias de datos en el mismo ciclo. Esta situación conlleva los siguientes aspectos:

1) *Al ejecutar varias operaciones con dependencias de datos en el mismo ciclo, no son necesarias unidades de almacenamiento para albergar los resultados intermedios. Estos resultados son los generados como datos de salida por algunas operaciones y son a su vez datos de entrada de otras operaciones planificadas en el mismo ciclo. En la figura 5.12 ya se ilustró el ahorro en unidades de almacenamiento debido al encadenamiento de operaciones durante la planificación de una descripción conductual sencilla.*

2) *Las unidades funcionales encadenadas en el mismo ciclo no pueden reusarse para ejecutar otras operaciones en dicho ciclo, aún cuando han terminado de calcular la operación que tenían asignada, en dicho caso permanecen ociosas hasta el final del ciclo. Podemos afirmar que dadas dos unidades funcionales encadenadas en un determinado ciclo, cada una de ellas está siendo desaprovechada durante alguna parte del ciclo de reloj.*

5.2.2.3 Duración del ciclo de reloj

La elección de la longitud adecuada del ciclo de reloj para una determinada descripción conductual depende de varios factores:

1) *El tamaño de la descripción conductual a sintetizar definido principalmente por el número de operaciones y la complejidad de las mismas.*

2) *Las dependencias de datos entre las distintas operaciones de la descripción conductual.*

3) *El estilo descriptivo utilizado al especificar la descripción conductual. Este estilo está condicionado por una serie de factores relacionados con las operaciones que forman parte de dicha especificación, como son:*

   a. *Número total de operaciones.*

   b. *Tipo de las operaciones.*

   c. *Formatos de representación de los datos de las operaciones.*

   d. *Anchura de las operaciones.*

La influencia que ejerce cada uno de los factores anteriores queda mitigada o suavizada por la técnica de síntesis propuesta en esta memoria. Por ello, y al contrario de lo que sucede con los algoritmos convencionales de SAN, la calidad de los circuitos sintetizados por el algoritmo propuesto presenta un alto grado de independencia de los factores expuestos anteriormente.

5.2.2.4 Descripción del escenario experimental

Con el fin de analizar la influencia de las restricciones de tiempo impuestas por el diseñador (latencia o duración del ciclo de reloj) se ha definido un escenario experimental en el que se ha sintetizado la descripción conductual del filtro FIR de orden 16, denominado "beamformer" de 16 elementos en el repositorio de benchmarks [Dutt92]. Los bucles internos de la descripción conductual se han desenrollado obteniéndose una especificación formada por 272 operaciones de los tipos multiplicación y suma.

Se han realizado tres procesos de SAN independientes con las tres metodologías de diseño descritas anteriormente. Cada proceso de síntesis se ha repetido para distintos valores de la longitud del ciclo de reloj. La latencia

se ha restringido, en todos los casos, a su valor mínimo calculado teniendo en cuenta la propuesta descrita en el primer apartado del capítulo cuarto sobre la latencia del circuito.

En algunos casos, los procesos de SAN basados en la planificación a nivel de bit y FDS no han podido generar planificaciones que cumplan las restricciones de tiempo impuestas al diseño: longitud del ciclo de reloj y latencia. Con el fin de obtener circuitos que satisfagan las restricciones de tiempo impuestas, se ha optimizado la especificación conductual aplicando las transformaciones aritméticas propuestas por Verma et al [VeLe04]. Estas transformaciones tienen por objetivo la aceleración de las operaciones situadas en el camino crítico del GFD, maximizando las oportunidades de usar árboles de sumadores CSA (Carry Save Adders) en circuitos combinacionales.

La técnica de optimización de especificaciones utilizada sustituye inicialmente las multiplicaciones por operaciones lógicas y sumas. A continuación, reorganiza las sumas para obtener sumas con múltiples entradas. Y finalmente, estas nuevas sumas son sustituidas por un árbol compresor de sumas seguido de una última y única suma de propagación de acarreos.

Una vez optimizada la especificación mediante la técnica indicada, se sintetiza nuevamente utilizando la planificación a nivel de bit y el algoritmo FDS. En este caso, ambos algoritmos obtienen planificaciones válidas, es decir, planificaciones que cumplen las restricciones de tiempo impuestas (longitud del ciclo de reloj y latencia).

Mediante la técnica de síntesis propuesta se han obtenido planificaciones válidas en todos los casos. Por este motivo, se ha trabajado directamente con la especificación original, en lugar de usar la especificación optimizada. Además, las implementaciones propuestas tienen un tamaño significativamente menor que en el caso de los circuitos sintetizados por las otras técnicas. En el siguiente apartado se detallan los resultados obtenidos en cada caso.

5.2.2.5 Resultados experimentales

Las pruebas realizadas se han repetido para distintos valores de la longitud del ciclo de reloj. La longitud del ciclo ha variado desde 10 nanosegundos hasta 80, y se ha experimentado con los valores múltiplos de 10 en el rango indicado.

Para cada valor del tiempo de ciclo, se ha sintetizado la descripción conductual con las técnicas de SAN indicadas: metodología de diseño propuesta en esta memoria, algoritmos de SAN a nivel de bit y el algoritmo de planificación FDS combinado con la selección y asignación de HW basado en patrones.

En la figura 5.13 se presentan los resultados de área obtenidos por cada una de las tres técnicas. Por otro lado, en la figura 5.14 se presentan las latencias de los circuitos sintetizados. El área se expresa en miles de puertas y la latencia en número de ciclos de la planificación.

En las pruebas realizadas, todas las técnicas de SAN utilizadas obtienen planificaciones válidas para valores del tiempo de ciclo iguales o superiores a 60 ns. En estos casos (60 ns, 70 ns y 80 ns) las tres técnicas de síntesis obtienen implementaciones con el mismo valor de la latencia, tal como se muestra en la figura 5.14.

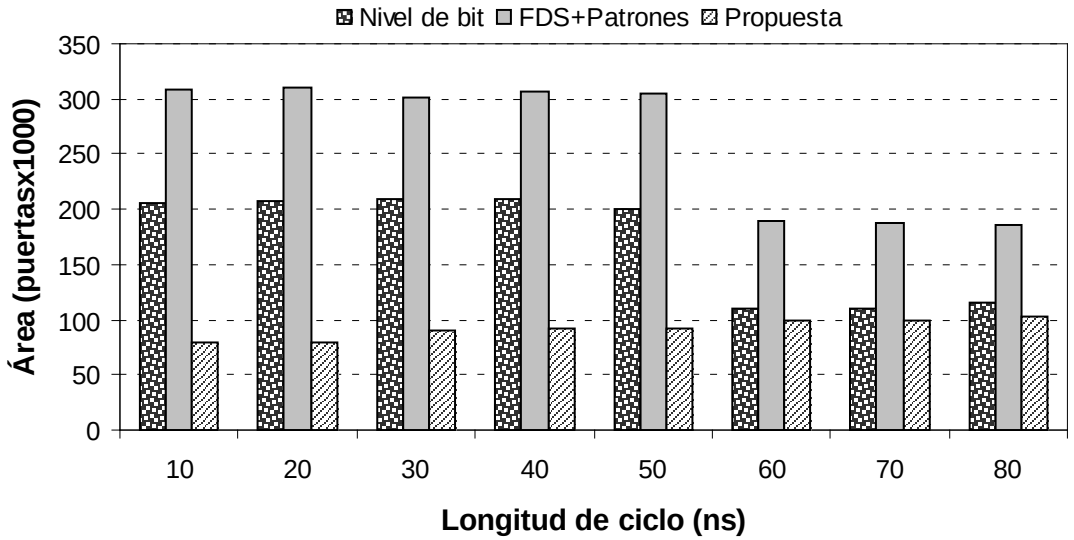

Figura 5.13 *Área del circuito sintetizado por cada una de las tres técnicas para diferentes longitudes del ciclo de reloj.*

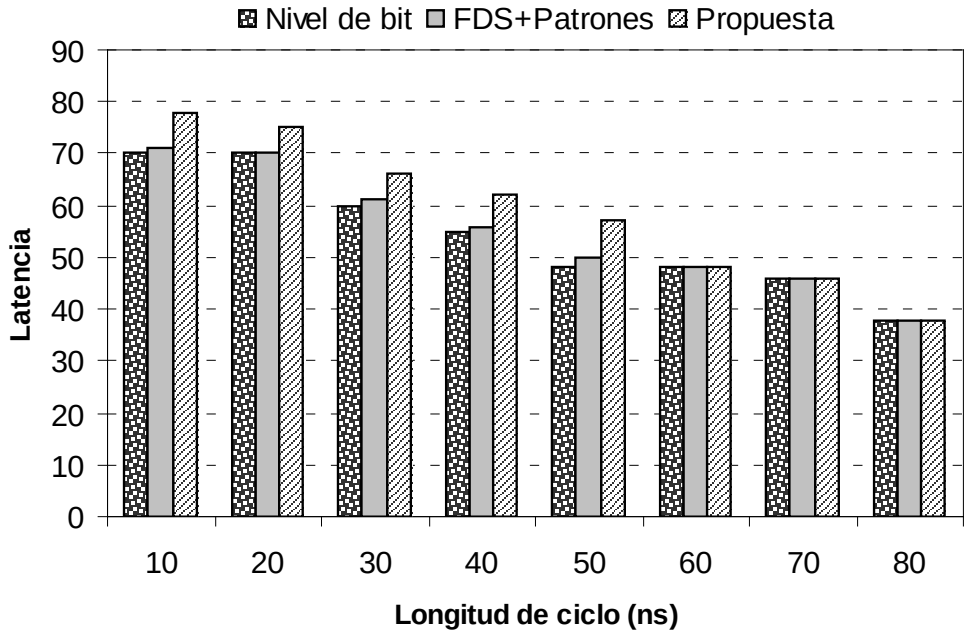

Figura 5.14 *Latencia del circuito sintetizado por cada una de las tres técnicas para diferentes longitudes del ciclo de reloj.*

En los casos anteriores, y según se observa en la figura 5.13, el área de las implementaciones obtenidas por la metodología de diseño propuesta es de media un 44% y un 10% inferior que los circuitos sintetizados mediante el algoritmo FDS combinado con la asignación basada en patrones, y la SAN a nivel de bit, respectivamente. Por tanto, la técnica propuesta obtiene mejoras sustanciales en el área necesaria para la implementación, sin incrementar la latencia del circuito con respecto a las otras dos técnicas de síntesis.

Para valores de la duración del ciclo de reloj inferiores a 60 ns, el algoritmo FDS combinado con la asignación basada en patrones y la SAN a nivel de bit no producen planificaciones válidas. En estos casos se ha aplicado la técnica de optimización de especificaciones conductuales explicada anteriormente. A pesar de dichas optimizaciones, las necesidades de área en ambos casos son significativamente mayores que en los circuitos sintetizados mediante la técnica propuesta, tal y como se muestra en la figura 5.13. En particular, las implementaciones sintetizadas por el algoritmo FDS combinado

con la asignación basada en patrones requieren un área tres veces superior a la ocupada por los circuitos sintetizados mediante la técnica de diseño propuesta. Y las implementaciones obtenidas por el algoritmo de SAN a nivel de bit requieren el doble de área que los circuitos sintetizados por nuestro algoritmo. En estos casos, el algoritmo propuesto introduce una penalización en la latencia del circuito que en el peor de los casos incrementa el número de ciclos en un 20% con respecto a las otras dos técnicas.

El área de las implementaciones sintetizadas mediante la técnica propuesta se mantiene entre 80 miles de puertas y 103 miles de puertas para el intervalo de longitud de ciclo entre 10 y 80 ns, mientras en el caso de los circuitos sintetizados por los otros algoritmos el área llega casi a duplicarse. Este hecho demuestra que el valor de la longitud del ciclo no afecta igualmente a los resultados obtenidos por las tres técnicas de SAN. El algoritmo propuesto es el que muestra una menor dependencia de la longitud del ciclo definida por el diseñador, tal como se adelantó anteriormente.

Continuando el análisis de los resultados plasmados en la figura 5.13, los mejores resultados de área se obtienen para las longitudes de ciclo menores, es decir, 10 y 20 ns. Esto se debe a que al reducir la anchura del ciclo aumenta el número de operaciones multiciclo, y el algoritmo de síntesis propuesto descompone estas operaciones obteniendo una especificación con un mayor número de operaciones más sencillas. En general, el aumento en el número de operaciones de la especificación produce un incremento en el número de patrones que pueden identificarse en la especificación y en el número de ocurrencias de los mismos.

En resumen, la técnica de síntesis propuesta en la presente memoria obtiene mejores resultados en comparación con las otras dos técnicas de SAN, siendo estos mejores cuanto menor sea la longitud de ciclo definida por el diseñador. Además, la reducción de área lograda tiene una repercusión moderada en la latencia de los circuitos sintetizados.

5.2.3 Identificación de patrones

Para comprobar la eficacia del algoritmo de identificación de patrones propuesto en la presente memoria, se ha comparado con la propuesta realizada por J. Cong et al [CoJi08]. Con el fin de aumentar el número de patrones reconocidos y sus ocurrencias, los autores aceptan distintas variaciones en el reconocimiento de patrones tales como:

1) *Cambios en las anchuras de los operandos.*

2) *Modificaciones estructurales de los diagramas de datos y de las dependencias entre operaciones.*

3) *Cambios en el número y anchura de los datos de entrada y salida de las operaciones debido a las variaciones anteriores.*

Las pruebas comparativas entre ambas propuestas se han llevado a cabo con diversos benchmarks clásicos de SAN propuestos por Dutt et al [DuPa95] y Verma et al [Vele04]. En concreto se han sintetizado los siguientes benchmarks:

1) *Beamformer o filtro de orden 16 (beamformer en la tabla de resultados).*

2) *Diferential heat release computation (diffheat en la tabla de resultados).*

3) *Transformador rápido de Fourier (fourier en la tabla de resultados).*

4) *Filtro de onda elíptica de quinto orden (elliptic en la tabla de resultados).*

5) *Resolutor de ecuaciones diferenciales (diffeq).*

6) *Filtro IIR de orden cuarto (irr4).*

7) *Filtro FIR de segundo orden (fir2).*

Los siete benchmarks anteriores se han sintetizado con diferentes restricciones de tiempo. Se ha experimentado con longitudes de ciclo desde 45 ns hasta 120 ns, y con restricciones de la latencia desde 6 ciclos hasta 50

ciclos de reloj. De esta forma se han definido distintos escenarios caracterizados por tres variables:

1) Benchmark aplicado.

2) Longitud del ciclo definida por el diseñador.

3) Latencia definida por el diseñador.

Los resultados obtenidos en cada uno de los escenarios (benchmark, longitud del ciclo y latencia) por cada una de las dos técnicas se muestran en la tabla 5.2. Estos resultados están agrupados en:

1) Tiempo de ejecución de la descripción conductual medida en ns (tiempo de ciclo x latencia).

2) Número de patrones localizados por cada técnica y número total de ocurrencias de los mismos.

3) Área del circuito sintetizado medida en número de puertas equivalentes.

El número de patrones identificados por la técnica propuesta no se corresponde únicamente con los descubiertos durante la primera fase del algoritmo, sino que se han tenido en cuenta todos los patrones identificados a lo largo de todo el proceso de SAN.

En la tabla de resultados, se puede comprobar que el número de patrones y el número de ocurrencias identificados mediante la técnica propuesta es, de media, el doble que los identificados mediante la técnica propuesta por J. Cong. Esto se debe a la aplicación de la técnica de descomposición de operaciones, tanto a la descomposición inicial de operaciones multiciclo, como a la descomposición de operaciones monociclo realizada para identificar nuevos patrones que cumplan las propiedades de calidad definidas.

Asimismo, se puede observar que los mejores resultados, en cuanto al área de los circuitos sintetizados, se obtienen con longitudes de ciclo

pequeñas. En concreto, el mayor ahorro de área, 53%, se obtiene para la menor longitud de ciclo, 45 ns. Esto es debido a que cuanto menor sea la longitud del ciclo de reloj mayor es el número de operaciones multiciclo, e inicialmente se descomponen un mayor número de operaciones, lo que aumenta el número de patrones y de sus ocurrencias. Por otro lado, las implementaciones propuestas por los algoritmos convencionales presentan un elevado número de unidades funcionales multiciclo, que debido a su estructura interna tienen un reuso limitado (desaprovechamiento interno de HW) lo que finalmente redunda en circuitos de mayor área que los sintetizados por el algoritmo propuesto en esta memoria.

Benchmark	**Restricciones tiempo**		**Patrones (num / occurrencias)**		**Área (puertas)**		
	Ciclo	**Latencia**	**FDS+Pat.**	**Propuesta**	**FDS+Pat.**	**Propuesta**	**Ahorro**
beamformer	85	45	(14 / 560)	(25 / 993)	176.286	107.046	40%
beamformer	120	35	(17 / 481)	(36 / 1078)	284.019	155.538	46%
diffheat	80	50	(15 / 562)	(27 / 958)	140.867	85.722	40%
diffheat	110	40	(19 / 552)	(34 / 987)	168.972	86.651	49%
fourier	85	20	(14 / 226)	(26 / 428)	128.754	73.995	43%
fourier	100	14	(19 / 205)	(34 / 394)	139.753	75.644	46%
elliptic	45	16	(12 / 151)	(23 / 283)	67.944	32.084	53%
elliptic	65	12	(17 / 160)	(29 / 274)	70.652	45.926	35%
diffeq	110	6	(13 / 71)	(24 / 128)	48.762	28.036	43%
diffeq	120	4	(15 / 54)	(32 / 112)	55.063	29.678	47%
iir4	85	8	(7 / 48)	(18 / 122)	12.096	8.502	30%
iir4	100	6	(9 / 55)	(22 / 136)	14.771	8.634	42%
fir2	85	8	(6 / 42)	(15 / 96)	9.754	6.671	32%
fir2	100	6	(8 / 38)	(19 / 104)	10.987	6.993	37%

Tabla 5.2 *Resultados de la síntesis de varios benchmarks clásicos*

Teniendo en cuenta los resultados obtenidos en todos los escenarios reflejados en la tabla, el ahorro medio de área logrado con la técnica propuesta alcanza el 42% con respecto a la técnica de identificación de patrones propuesta por J. Cong y el algoritmo FDS.

5.2.4 Unidades multiciclo vs monociclo

Mediante la descomposición de operaciones se transforma inicialmente cada operación multiciclo de la especificación en un conjunto de operaciones monociclo encadenadas, obteniéndose una nueva especificación conductual formada únicamente por operaciones monociclo.

A diferencia de la técnica propuesta, otros algoritmos de SAN obtienen implementaciones que contienen unidades funcionales multiciclo para ejecutar las operaciones multiciclo presentes en las descripciones conductuales.

A continuación se va a analizar el efecto que produce en el área de las implementaciones sintetizadas el uso de unidades funcionales multiciclo. Para ello se ha sintetizado el filtro de onda elíptica de quinto orden utilizando los siguientes algoritmos de SAN: un algoritmo de SAN a nivel de bit especializado en la reducción del área, el algoritmo FDS combinado con un algoritmo de asignación basado en patrones y la metodología de diseño propuesta en la presente memoria.

En la figura 5.15 se muestran los resultados obtenidos por los tres algoritmos. Los resultados de área se han desglosado en los distintos tipos de recursos de la ruta de datos: unidades funcionales, unidades de almacenamiento, elementos de enrutamiento de datos y lógica pegamento. Igualmente se muestra en la tabla el área de la unidad de control.

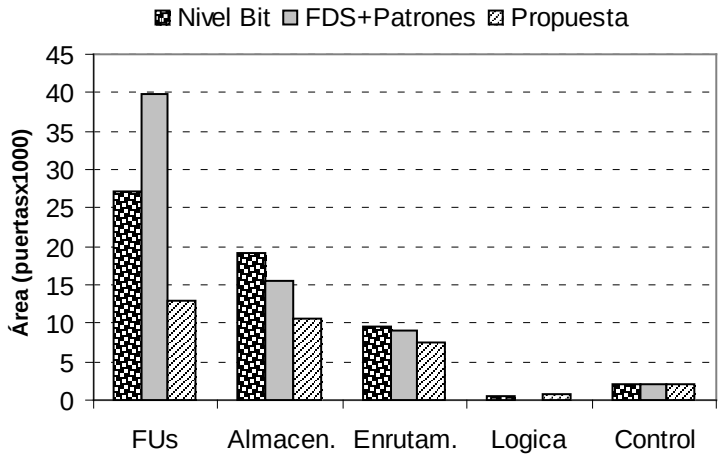

Figura 5.15 *Resultados de área para el filtro de onda elíptica.*

En todos los circuitos sintetizados el área ocupada por las unidades funcionales corresponde aproximadamente al 60% del área total. Por ello, la reducción del área ocupada por las unidades funcionales resulta primordial para lograr importantes decrementos en el área total de los circuitos. Como se muestra en los resultados de la figura 5.15, el algoritmo propuesto produce implementaciones cuyas unidades funcionales ocupan un área bastante menor que en el caso de los circuitos sintetizados con las otras técnicas. Esto implica que el área total de los circuitos sea también menor, dado que el área ocupada por el resto de los recursos HW también se reduce.

El algoritmo FDS combinado con el algoritmo de selección y asignación de HW basado en patrones ha sintetizado una ruta de datos con unidades funcionales multiciclo, cuya área corresponde al 40% del área total de las unidades funcionales. Sin embargo, el ahorro de área logrado por nuestro algoritmo en este caso es superior al 40% debido a una mayor reutilización de las unidades funcionales monociclo.

El algoritmo a nivel de bit también descompone las operaciones multiciclo en varias operaciones monociclo con el fin de equilibrar el coste computacional de las operaciones ejecutadas en cada ciclo. Por tanto, en las implementaciones propuestas por este algoritmo sólo existen unidades funcionales monociclo, igual que en los circuitos sintetizados mediante la

técnica de diseño propuesta. No obstante, la técnica propuesta obtiene mejores resultados en cuanto al reuso de los recursos funcionales, lo que produce que el área ocupada por las unidades funcionales sea menor.

La utilización de las técnicas de descomposición de operaciones no implica un aumento en el coste de los recursos de almacenamiento, como a priori podría parecer. Resulta evidente que la ejecución en varios ciclos de una operación descompuesta en varias operaciones más sencillas, requiere unidades de almacenamiento para guardar los resultados y acarreos intermedios calculados en los primeros ciclos. Sin embargo, una vez que se ha calculado un fragmento de operación, en muchos casos, los operandos completos de entrada no son necesarios para calcular el resto de los fragmentos. Normalmente, sólo son necesarios algunos bits de los operandos de entrada. De esta manera, las unidades de almacenamiento utilizadas para almacenar los operandos de entrada se pueden usar también (cuando ya no sea necesario almacenar los operandos completos) para almacenar los resultados intermedios calculados por los fragmentos de la operación. Esta reutilización de las unidades de almacenamiento sólo se puede llevar cabo si para la selección y asignación se utiliza un algoritmo a nivel de bit como el propuesto por Molina et al [MRMH06].

La técnica propuesta reduce además el área ocupada por las unidades de almacenamiento y de transmisión de datos, debido a que favorece el encadenamiento de operaciones según un determinado patrón. Al encadenar operaciones en un mismo ciclo se hace innecesario el uso de recursos de almacenamiento para almacenar los resultados intermedios calculados. Y la reutilización de las mismas unidades funcionales para ejecutar las distintas ocurrencias de un patrón, implica que entre dichas unidades funcionales no sean necesarios elementos de encaminamiento de datos, tales como por ejemplo multiplexores.

Las unidades de control sintetizadas con las tres técnicas utilizadas presentan áreas muy similares, como se puede comprobar en la figura 5.15.

El ahorro total de área obtenido con la técnica propuesta alcanza el 42% con respecto al algoritmo de SAN a nivel de bit, y el 49% con respecto a la SAN utilizando el algoritmo FDS y el algoritmo de asignación basado en patrones.

El número de operaciones de la especificación conductual al final del proceso de SAN ha crecido un 26% con respecto a la descripción conductual original, debido a las descomposiciones de operaciones realizadas. Sin embargo, este aumento apenas ha repercutido en el tiempo de ejecución del algoritmo, ya que la fragmentación o descomposición de operaciones se realiza de una forma progresiva, a lo largo de todo el proceso de SAN. Finalmente, el tiempo de ejecución de nuestro algoritmo es ligeramente mayor (16% mayor de media) que el tiempo de ejecución del algoritmo FDS y la asignación basada en patrones, como se muestra en la Tabla 5.3.

5.2.5 Síntesis de un circuito real: ADPCM

Además de los benchmarks clásicos referidos en los aparatados anteriores, también se ha aplicado la técnica de síntesis propuesta a algunos módulos del algoritmo de decodificación ADPCM[3] descrito en la recomendación G.721 del CCITT[4]. Los módulos que se han sintetizado son los siguientes:

1) Inverse Adaptative Quantizer (IAQ).

2) Quantizer Scale Factor (QSF).

3) Output PCM Format Conversion (OPFC).

4) Synchronous Coding Adjustment (SCA).

5) Tone and Transition Detector (TTD).

[3] Adaptive Differential Pulse Code Modulation.

[4] Comité Consultatif International Téléphonique et Télégraphique. CCITT renombrado en 1993 como ITU-T, International Telecommunication Union.

Los módulos OPFC y SCA se han sintetizado juntos debido a su proximidad en el grafo del decodificador. Por otra parte, los módulos IAQ, QSF y TTD se han sintetizado de forma independiente.

Benchmark	Restricciones de tiempo		Tiempo de ejecución (s)		
	Long. Ciclo	Latencia	FDS+Pat.	Propuesta	Aumento
beamformer	85 ns	45 ciclos	48 s	55 s	13%
beamformer	120 ns	35 ciclos	45 s	52 s	13%
diffheat	80 ns	50 ciclos	49 s	54 s	9%
diffheat	110 ns	40 ciclos	42 s	49 s	14%
fourier	85 ns	20 ciclos	20 s	28 s	29%
fourier	100 ns	14 ciclos	18 s	26 s	31%
elliptic	45 ns	16 ciclos	38 s	44 s	14%
elliptic	65 ns	12 ciclos	32 s	40 s	20%
diffeq	110 ns	6 ciclos	26 s	31 s	16%
diffeq	120 ns	4 ciclos	24 s	29 s	17%
iir4	85 ns	8 ciclos	27 s	30 s	10%
iir4	100 ns	6 ciclos	15 s	19 s	21%
fir2	85 ns	8 ciclos	20 s	21 s	5%
fir2	100 ns	6 ciclos	18 s	20 s	10%

Tabla 5.3 *Resultados de tiempo de ejecución obtenidos tras la síntesis de varios benchmarks clásicos*

En la tabla 5.4 se muestran las áreas y latencias de las implementaciones obtenidas para distintos valores de la duración del ciclo de reloj. El algoritmo FDS y la asignación basada en patrones obtienen resultados muy similares de área en todos los escenarios. Para valores altos de la duración del ciclo de reloj, se generan rutas de datos con un número muy bajo de unidades funcionales multiciclo. Por tanto, se minimiza el desaprovechamiento de área

asociado a este tipo de recursos. Y en el caso de valores bajos de la duración del ciclo de reloj, se incrementa la latencia, el reuso de unidades funcionales monociclo y los requerimientos de unidades multiciclo. Debido al aumento de unidades funcionales multiciclo, los circuitos sintetizados por el algoritmo FDS y la asignación basada en patrones presentan, para ciclos de reloj pequeños, ligeras reducciones de área en comparación con las implementaciones con ciclos más largos.

	Longitud de ciclo = 10ns					
	Área (puertas)			**Latencia**		
Módulo	**FDS+Patrón**	**Bit-level**	**Propuesta**	**FDS+Patrón**	**Bit-level**	**Propuesta**
IAQ	792	597	468	20	19	18
QSF	1124	1124	894	26	25	22
TTD	1086	1086	772	21	18	18
OPFC +	769	769	582	29	27	26
	Longitud de ciclo = 20ns					
	Area (puertas)			**Latencia**		
	FDS+Patrón	**Bit-level**	**Propuesta**	**FDS+Patrón**	**Bit-level**	**Propuesta**
IAQ	786	684	542	12	11	11
QSF	2382	1342	1986	19	17	15
TTD	1762	1217	934	14	13	12
OPFC + SCA	1312	986	607	22	22	18
	Longitud de ciclo = 30ns					
	Area (puertas)			**Latencia**		
	FDS+Patrón	**Bit-level**	**Propuesta**	**FDS+Patrón**	**Bit-level**	**Propuesta**
IAQ	820	798	670	8	8	7
QSF	2486	2084	1398	16	14	12
TTD	1866	1873	1276	11	10	8
OPFC + SCA	1332	1226	812	18	17	14

Tabla 5.4 *Resultados de la síntesis de algunos módulos del decodificador ADPCM*

La metodología de diseño propuesta en esta tesis y los algoritmos a nivel de bit se benefician de la descomposición de las operaciones multiciclo en los escenarios en que la longitud de ciclo es corta, obteniendo circuitos

significativamente menores que los obtenidos por el algoritmo FDS y la asignación basada en patrones.

Los circuitos obtenidos con la metodología de diseño propuesta presentan áreas menores que los sintetizados con los algoritmos de SAN a nivel de bit, debido a una reutilización más eficiente de los recursos funcionales. La técnica propuesta contribuye a introducir ahorros adicionales en unidades de almacenamiento y de encaminamiento de datos.

Tal como se muestra en los resultados presentados en la tabla 5.4, el área media ahorrada utilizando la técnica propuesta alcanza el 46% en comparación con el algoritmo FDS y la asignación de HW basada en patrones, y el 20% comparada con los algoritmos de SAN a nivel de bit.

Además, el número de ciclos de reloj es menor en el caso de los circuitos sintetizados utilizando la metodología de diseño propuesta, debido al cálculo de la latencia mínima llevado a cabo por nuestro algoritmo.

5.2.6 Características adicionales

Adicionalmente, se han llevado a cabo una serie de pruebas con varios benchmarks que manejan longitudes de palabra entre 16 y 64 bits. Los valores de la longitud del ciclo de reloj utilizados para realizar la síntesis se corresponden con los valores típicos inferidos por un algoritmo convencional. El algoritmo propuesto ha obtenido resultados válidos en todos los experimentos, ciñéndose a las restricciones de tiempo definidas y generando siempre diseños de áreas reducidas.

5.2.6.1 Algunas limitaciones

Aunque la cantidad de área ahorrada en comparación con las aproximaciones anteriores aumenta al reducir la duración del ciclo de reloj, se encuentran ciertas limitaciones en la aplicación del algoritmo. La descomposición de las operaciones multiciclo resulta ineficiente cuando las longitudes del ciclo de reloj son demasiado pequeñas. En esta situación se produce una fragmentación excesiva que provoca un aumento considerable

en el número de unidades funcionales, de almacenamiento, y de encaminamiento de datos, no obteniéndose beneficio alguno de la descomposición de las operaciones.

5.2.6.2 Consumo de energía

El consumo de energía tiene dos componentes principales: consumo estático y consumo dinámico. La técnica propuesta contribuye a reducir el consumo estático ya que tiende a reducir el área de los circuitos sintetizados. El consumo dinámico depende de los valores de los datos de entrada y de cómo se propagan a lo largo del GFD. Este consumo es más elevado cuantos más cambios se produzcan en los operandos de entrada de las unidades funcionales, por lo que, en general, la reutilización de las unidades funcionales es directamente proporcional al consumo dinámico.

Sin ser el objeto de esta memoria, se supone que la técnica propuesta provocará consumos menores de energía estática y mayores de energía dinámica, con respecto a otras técnicas que proponen rutas de datos de mayor área y, por tanto, con una menor reutilización de los recursos HW.

No obstante, el consumo de energía dinámica podría disminuirse incorporando a nuestro algoritmo algunas técnicas de bajo consumo. Por un lado, podría realizarse una caracterización de los datos de entrada de manera que durante la planificación y selección y asignación de HW, tanto de patrones como de operaciones residuales, se tuvieran en cuenta los valores típicos de cada operación, con el fin de reducir el número de conmutaciones que tienen lugar en cada recurso funcional. Por otro lado, también podría dedicarse parte del ahorro de área logrado por la metodología de diseño propuesta para sustituir las unidades funcionales de mayor consumo por unidades especiales de bajo consumo y mayor área.

5.2.7 Conclusiones

Los experimentos llevados a cabo para medir la calidad de la técnica de diseño propuesta, y el algoritmo de SAN que la incorpora, han demostrado que es posible obtener reducciones de área significativas en comparación

con los algoritmos convencionales de SAN. En particular, los resultados obtenidos se han comparado con los algoritmos a nivel de bit que incorporan técnicas de descomposición de operaciones, y con algoritmos basados en la identificación y gestión de patrones.

Los ahorros de área obtenidos no han supuesto un incremento en el tiempo de ejecución de los circuitos resultantes, que en la mayoría de los casos se han mantenido muy similares. Igualmente, el tiempo de ejecución de nuestro algoritmo es ligeramente superior al de las otras propuestas consideradas.

CAPÍTULO

6

CONCLUSIONES, PRINCIPALES APORTACIONES Y LÍNEAS ABIERTAS

En la presente memoria se ha pretendido plantear una metodología de diseño que contribuya a resolver el problema del desaprovechamiento de HW presente en la mayoría de los circuitos generados por los algoritmos convencionales y herramientas comerciales de SAN. En los circuitos sintetizados por los algoritmos o herramientas indicados, aparecen muchos recursos HW, tanto unidades funcionales como de almacenamiento y de encaminamiento de datos que se encuentran ociosos o desaprovechados en algunos de los

ciclos de la ejecución de la conducta o, en parte de ellos. Este desaprovechamiento aparece en uno o varios ciclos de la planificación y/o durante parte de algunos ciclos. El problema del desaprovechamiento aparece tanto en el caso de las especificaciones homogéneas (todos los datos de la especificación conductual tienen la misma anchura) como en las especificaciones heterogéneas (los datos de la especificación presentan anchuras diferentes). Los algoritmos convencionales pretenden distribuir homogéneamente las operaciones del mismo tipo y anchura entre los ciclos de la planificación. Este objetivo resulta habitualmente difícil de alcanzar si se tienen en cuenta los diferentes tipos de operaciones presentes en la especificación, las restricciones de tiempo y área impuestas a los diseños y las dependencias de datos entre las operaciones. Adicionalmente, la fuerte dependencia que presentan los algoritmos convencionales de SAN con respecto al estilo descriptivo utilizado en las especificaciones conductuales, dificulta aún más la búsqueda de soluciones óptimas al problema de la SAN.

El objetivo principal de esta investigación se definió alrededor de la reducción del desaprovechamiento de los recursos HW de la ruta datos, intentando mitigar la dependencia del estilo descriptivo ya mencionada. Con el objeto de ampliar el espacio de búsqueda de soluciones en la síntesis de especificaciones conductuales, se ha propuesto una metodología de diseño basada en la aplicación combinada de las técnicas de descomposición de operaciones y gestión de patrones. Estas dos técnicas no se aplican de forma independiente sino que cooperan entre sí para alcanzar un objetivo común: la reducción del área de los circuitos sintetizados.

Mediante la consecución de este objetivo, el trabajo de investigación expuesto en esta memoria ha pretendido poner a disposición de la comunidad de diseñadores una metodología de diseño basada en la combinación de ambas técnicas, y un algoritmo de SAN que implementa dicha metodología de manera eficiente.

6.1 Principales aportaciones

En líneas generales, las principales aportaciones de esta investigación al estado del arte de la SAN son:

1) Análisis de la influencia que el estilo descriptivo usado en las especificaciones conductuales ejerce sobre la calidad de los circuitos sintetizados por los algoritmos convencionales.

2) Nueva metodología de diseño capaz de reducir el área de los circuitos sintetizados, en comparación con los algoritmos convencionales de SAN. Esta reducción se ha logrado mediante:

    a. *Descomposición de operaciones*. Incluyendo la descomposición tanto de las operaciones complejas multiciclo como de las operaciones más sencillas monociclo.

    b. *Gestión de patrones*. Identificación de patrones y ocurrencias de los mismos mediante el ensayo de las posibles descomposiciones de operaciones de la especificación.

3) Algoritmo de SAN que implementa la metodología de diseño propuesta en esta tesis.

    a. *Planificación de operaciones.* Implementación de un algoritmo de planificación de patrones que incorpora las técnicas propuestas con el objetivo de reducir el área de los circuitos. Las principales aportaciones incluidas en esta fase del algoritmo son:

        i. Asignación de las ocurrencias de un mismo patrón a ciclos distintos, con el objeto de favorecer el reuso de las unidades funcionales asignadas al mismo.

        ii. Asignación de las operaciones residuales a ciclos donde existan unidades funcionales ociosas compatibles en tipo y anchura.

b. *Selección y asignación de HW.* Implementación de un algoritmo de selección y asignación de recursos HW basado en la descomposición de operaciones y gestión de patrones cuyo objetivo es también reducir el área de los circuitos resultantes. Las principales aportaciones incluidas en esta fase del algoritmo son:

   i. Reutilización de las mismas unidades funcionales para ejecutar todas las ocurrencias del patrón planificadas en ciclos distintos.

   iii. Reutilización de las unidades funcionales asignadas a los patrones para ejecutar otras operaciones residuales en los ciclos en que se encuentran ociosas.

   iv. Reducción de las unidades de almacenamiento debido al encadenamiento de las operaciones que forman los patrones.

   v. Reducción de las unidades de interconexión de datos entre las unidades funcionales encadenadas que ejecutan las operaciones de los patrones.

4) El trabajo experimental realizado demuestra que las técnicas de fragmentación de operaciones y de gestión de patrones implementadas en el algoritmo de SAN propuesto, no implican un aumento en el tiempo de ejecución de los circuitos sintetizados. Inicialmente se descomponen todas las operaciones multiciclo de la especificación, generando así una nueva especificación conductual formada únicamente por operaciones monociclo. Durante el proceso de síntesis se llevan a cabo más descomposiciones de operaciones, en este caso de operaciones monociclo, con el objeto de identificar nuevos patrones y aumentar el número de ocurrencias de los mismos. Todas estas descomposiciones favorecen la eliminación de las operaciones complejas de la especificación, y su sustitución por operaciones sencillas que no sólo

pueden ejecutarse en un único ciclo de reloj, sino que en la mayoría de los casos, pueden hacerlo de manera encadenada con otras operaciones de la especificación.

El tiempo de ejecución del algoritmo propuesto es similar al presentado por los algoritmos convencionales y herramientas comerciales de SAN, por lo que su uso resulta viable en el diseño de circuitos. El mayor número de operaciones en la especificación con la que trabaja nuestro algoritmo se ve compensado por el hecho de que se utiliza el patrón como unidad mínima de planificación y selección y asignación de HW.

Gran parte de estas aportaciones han quedado recogidas en diversas publicaciones en conferencias y revistas de reconocido prestigio en el ámbito internacional, incluyendo IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), Integration the VLSI Journal, Design Automation & Test in Europe Conference (DATE), Euromicro Conference on Digital System Design (DSD) y Conference on Design of Circuits and Integrated Systems (DCIS). La relación completa de publicaciones puede consultarse en la bibliografía.

6.2 Líneas abiertas

A continuación se describen algunas líneas de investigación abiertas en el marco de este trabajo que podrían aportar mejoras a la metodología de diseño propuesta, y por tanto que pueden considerarse como posibles líneas de continuación del mismo.

6.2.1 Nuevas técnicas de descomposición de operaciones

Con el fin de enriquecer los resultados de la técnica de descomposición de operaciones, se podrían plantear nuevas formas de fragmentar las operaciones complejas de la especificación, para obtener no sólo operaciones de tipos conocidos, sino también operaciones nuevas que

permitan identificar un mayor número de patrones y de ocurrencias de los mismos.

Este aumento en el número de las posibles descomposiciones de operaciones supone una ampliación del espacio de búsqueda de soluciones, con la identificación de nuevos patrones y ocurrencias de los mismos, que finalmente puede permitir mayores reúsos de los recursos HW de la ruta de datos y circuitos de menor área.

La definición de nuevas descomposiciones de operaciones que produzcan nuevos tipos de operaciones, deberá ir acompañada de un diseño adecuado de las unidades funcionales capaces de ejecutar dichas operaciones, que deberán incluirse en la biblioteca de diseño correspondiente de manera previa a la síntesis.

6.2.2 Relación entre los parámetros de calidad y la latencia

Como se ha expresado en varias ocasiones a lo largo de esta memoria, el valor de los parámetros de calidad, *PH* y *CP,* y su relación con el valor de la latencia de la planificación influyen y condicionan el resultado obtenido por el algoritmo de SAN al sintetizar la descripción conductual propuesta.

El parámetro *PH* influye en qué operaciones se seleccionan en cada momento para adquirir la cualidad de ser operaciones cabecera de patrón. Y el parámetro *CP* afecta a la selección de los patrones, formados a partir de las operaciones cabecera, que pueden ser considerados patrones candidatos, teniendo en cuenta para ello el número de ocurrencias de los mismos.

Dependiendo de los valores de ambos parámetros de calidad y del valor fijado para la latencia, el algoritmo de SAN propuesto puede obtener resultados de diferente calidad. Esta graduación de la calidad está en relación con el tiempo de cómputo necesario para la ejecución del algoritmo propuesto. Por ello, y basándose en los resultados experimentales obtenidos en las pruebas realizadas, se llegó a la conclusión de que el valor del parámetro *PH* debería ser como mínimo igual a la latencia del circuito. Además, el valor

del parámetro *CP* debería ser menor o igual que *PH* para garantizar que todas las operaciones cabecera puedan usarse para formar patrones candidatos.

Una mejora sustancial en el algoritmo de SAN propuesto vendría dada por la implementación de un algoritmo previo de cálculo y ajuste de los parámetros de calidad *PH* y *CP*, en relación con la latencia fijada por el diseñador. De esta manera se podrían explorar distintas combinaciones de los parámetros que influyen en la calidad de los resultados, y escoger de forma dinámica los valores más convenientes para la descripción conductual a sintetizar en cada momento. Esta exploración podría incorporar, entre otras técnicas, una simulación de la fragmentación de las operaciones y el análisis de los fragmentos obtenidos en relación con los patrones ya definidos, o los nuevos que se pudieran definir. Para esta simulación se podría utilizar como punto de partida el algoritmo de cálculo de patrones a partir de una operación cabecera, en el que se lleva a cabo una simulación de la fragmentación de las operaciones cuando el número de ocurrencias de un patrón es menor que *CP*.

El algoritmo de ajuste de los parámetros de calidad *PH* y *CP* en relación con la latencia, debería ser un algoritmo heurístico que realizara una búsqueda guiada de soluciones evitando así la comprobación exhaustiva de todas las posibles descomposiciones y de todos los valores de los parámetros, lo que incrementaría notablemente el tiempo de ejecución del algoritmo de SAN propuesto, haciéndolo inviable.

6.2.3 Unidades funcionales segmentadas

Resulta interesante el análisis y la comparación entre la utilización de unidades funcionales segmentas y la aplicación de la técnicas basadas en gestión de patrones. La principal ventaja de nuestra técnica frente a las unidades funcionales segmentadas, es la reutilización de los recursos funcionales de la ruta de datos. Las unidades funcionales sencillas ofrecen más posibilidades de reutilización que las complejas, ya que pueden ejecutar operaciones sencillas en un tiempo equivalente al retardo de la unidad funcional en cuestión, y

pueden también ejecutar fragmentos de operaciones complejas sin producir retardos en la ejecución de las mismas. En el caso de las unidades funcionales segmentadas, la ejecución de operaciones sencillas introduce retardos innecesarios, que en muchos diseños resultan inaceptables.

Dado que las unidades funcionales multiciclo segmentadas no conllevan el desaprovechamiento interno de HW de las unidades multiciclo, podría resultar interesante la obtención de circuitos en los que se combinaran la utilización de estos recursos funcionales con las unidades funcionales monociclo. Para que no existiera un desaprovechamiento importante del HW de las unidades segmentadas, la especificación conductual debería incluir un número significativo de operaciones complejas que pudieran ejecutarse en la citada unidad funcional.

En las especificaciones que contienen un número elevado de operaciones complejas del mismo tipo, la utilización de unidades funcionales segmentadas para ejecutar estas operaciones, reduce el número de descomposiciones iniciales de las operaciones de la especificación, obteniéndose rutas de datos de área similar, aunque más compactas, y en un tiempo de ejecución menor. Por tanto, el estudio de estas implementaciones puede producir ventajas adicionales a la metodología de diseño propuesta.

6.2.4 Optimización de implementaciones de nivel RT

En cuanto a la optimización de las implementaciones de nivel RT, la metodología de diseño propuesta en esta tesis podría aplicarse a este tipo de circuitos con el objetivo de eliminar algunas de las unidades funcionales presentes en los mismos. En primer lugar deberían identificarse las unidades funcionales con cierto desaprovechamiento de HW, tanto unidades multiciclo como monociclo. Las unidades multiciclo se sustituirían por un conjunto equivalente de unidades funcionales monociclo que podrían reutilizarse para ejecutar otras operaciones de la especificación en los ciclos en que se encuentran ociosas. Esta reasignación de HW tendría por objetivo la eliminación de algunas unidades funcionales de la ruta de datos.

Las operaciones asignadas a las unidades funcionales monociclo con desaprovechamiento de HW, deberían descomponerse en operaciones más sencillas que pudieran ejecutarse en las unidades funcionales ociosas de la ruta de datos, y eliminar la unidad funcional desaprovechada. En algunos casos no será posible eliminar dicha unidad funcional, pero podrá sustituirse por otras unidades cuya área total sea menor que la del recurso sustituido. Para ello habrá sido necesaria la descomposición de alguna de las operaciones asignadas a la misma, y la asignación de algunos de los nuevos fragmentos a unidades funcionales ociosas de la ruta de datos.

6.2.5 Optimización de la energía consumida

Las nuevas metodologías de diseño y los algoritmos presentados en esta memoria han tenido como objetivo reducir el área de las implementaciones sintetizadas a partir de las especificaciones conductuales. Previamente se habían propuesto técnicas de diseño, basadas también en la descomposición de operaciones, para reducir el tiempo de ejecución de los circuitos, mediante la minimización de la duración del ciclo de reloj para una latencia dada [Saut07].

Sin embargo, además de las dimensiones de área y tiempo de ejecución, existe otro parámetro fundamental para medir la calidad de los circuitos, la energía consumida, que cada día adquiere mayor importancia en el ámbito del diseño automático de circuitos. Ambas propuestas, metodologías de diseño y algoritmos, podrían adaptarse para disminuir la energía consumida en la ejecución de una determinada conducta.

La optimización de la energía consumida en la ejecución de una conducta puede lograrse mediante la reducción del consumo estático del circuito, del consumo dinámico, o de ambos factores. La disminución del consumo estático se consigue reduciendo el número y el tamaño de los recursos HW de la ruta de datos, y por tanto su disminución se encuentra en cierto modo implícita en la reducción del área de los diseños.

En cuanto al consumo dinámico, su decremento se consigue minimizando el número de conmutaciones a nivel de bit que se producen en los recursos HW. Para ello es necesario planificar en ciclos consecutivos aquellas operaciones que, por los valores que toman sus operandos de entrada, minimizan el número de conmutaciones del circuito si se asignan al mismo recurso funcional. La mayoría de los algoritmos propuestos para reducir el consumo de energía explotan además ciertas propiedades aritméticas de las operaciones con el objeto de minimizar el número de conmutaciones. La calidad de los circuitos sintetizados por estos algoritmos depende del estilo descriptivo utilizado en la especificación conductual.

Las técnicas de descomposición de operaciones y de gestión de patrones pueden también adaptarse para reducir el consumo dinámico de los recursos funcionales de las rutas de datos, con cierta independencia del estilo empleado en la descripción de la conducta. En particular, podrían alcanzarse diseños donde las propiedades de las operaciones se apliquen únicamente a ciertos fragmentos de una operación, en lugar de a todos los fragmentos (aplicación parcial de las propiedades de las operaciones), y en los que la reutilización de unidades funcionales entre las operaciones y los fragmentos de la especificación no tuviera límites en cuanto a la procedencia de los mismos.

THESIS SUMMARY

Conventional high-level synthesis algorithms treat specification operations as atomic elements that are executed in one or several consecutive cycles and over one functional unit. However, in most specifications there exist different types, representations, and widths of operations that, handled at different decomposition levels, may produce better designs. In this way, most arithmetic operations can be decomposed into smaller operations applying several arithmetical properties. Different decompositions can be performed in order to improve the performance, or reduce the area or power consumption. In this thesis we propose a pattern-based design methodology able to treat every operation at its most appropriate decomposition level. It produces reduced datapaths while meeting specified time constraints.

1 Introduction

The high-level synthesis (HLS) process derives register-transfer level (RTL) circuits from their behavioral descriptions. Resource-constrained algorithms try to reduce the time employed to execute the behavior over a limited set of hardware resources, and time-constrained algorithms try to reduce the datapath area while meeting the imposed time restrictions.

Time-constrained circuits usually have multicycle functional units (FU) that execute the operations whose execution times are longer than the clock cycle duration. However, in most cases some slack time appears at the end of the last execution cycle of the operator. Non integer multicycle technique [PaCh01] has been proposed to reduce this slack. It allows the execution of

some data-dependent operations at the end of the last cycle. The main disadvantage of this technique becomes the need of extra HW resources to execute the chained operations, since they cannot share the same FU in the same clock cycle.

In order to minimize the datapath area, while preserving the time constraints, most conventional algorithms try to balance the number of operations (whose execution times are smaller than the cycle length) executed per cycle. Operations with different types are considered separately. This way, the number of idle FUs is minimized. Most existing algorithms also allow the execution of operations over wider FUs, extending the operands when necessary and discarding some result bits. This consideration is also taken into account to balance the distribution of operations among cycles.

A similar distribution of operations to cycles is applied for multicycle operations. In this case, HLS algorithms try to balance the number of multicycle operations executed per cycle, assuming that the execution time of a multicycle FU comprises several clock cycles where the FU cannot be shared to execute a different operation. However, multicycle FUs are not used completely in any of the cycles required to execute one operation.

Circuit performance can be improved using pipelined multicycle FUs. However, the use of independent monocycle FUs, able to execute the set of atomic operations calculated in every cycle, usually results better in terms of HW reuse. Monocycle FUs can be used to execute multicycle operations (similarly to pipelined multicycle FUs), if the result produced by one FU is provided in the next cycle to another one, and so on until the completion of the operation. Furthermore, monocycle FUs can also be used to execute other monocycle operations present in the specification, whose calculation in the pipelined multicycle FU would lead to unnecessary delays, as the operation result can be calculated faster over a monocycle FU. Additionally, the execution of monocycle operations over pipelined multicycle FUs also produces some HW waste, similar to that occurred in multicycle FUs.

The potential reuse is larger in monocycle FUs than in multicyle ones (included pipelined FUs), as they can be used to execute either multicycle or monocyle operations without delaying the results of monocycle operations to several cycles. In comparison to multicycle and multicycle pipelined FUs, the decomposition of multicycle operations to be executed over monocycle FUs reduces the number of idle FUs as well as the HW waste due to the execution of operations over wider FUs, and thus it contributes to the reduction of the circuit area.

This design technique can save significant area if applied to complex FUs, as multicycle, pipelined, or floating point ones. In this research work we focus on reducing datapath area through the decomposition of multicycle operations to be executed over monocycle FUs. In this context, the area reduction for one 2-cycle multiplier can reach 1/4 of its total area, 1/3 for one 3-cycle multiplier, and 1/2 for a 4-cycle multiplier. Note that the real area saving is usually larger due to the execution of monocycle operations over the FUs allocated to execute multicycle operations. This technique can be extended to other types of operations and, what is more interesting, can be fine tuned for every different specification and time constraint to maximize the area reduction.

Bigger area savings may be achieved if the decompositions are applied in combination with other design techniques. In particular, regularity exploitation may produce high quality circuits if it is used to guide the decompositions. This way, operations are decomposed to extract their common operation pattern that is repeated in most cycles. The HW needed to execute the selected pattern is shared by a bigger number of operations, decreasing the amount of HW required to execute the behavioral specification. Additionally, operation patterns may include fragments of different operations. This chaining of operations may also contribute to reduce the datapath area since the storage needs are usually smaller [MeMo04].

In this thesis we propose a design methodology and a HLS algorithm able to perform the scheduling, allocation, and binding of behavioral specifications.

It selectively decomposes some operations into several smaller ones in order to schedule in every cycle a similar number of suboperations with identical pattern. This way, the area of the synthesized datapaths is quite smaller than from conventional algorithms. Only decompositions that meet the time constraints imposed to the design are considered. Furthermore, some of them also contribute to reduce the cycle length, producing significant improvements in performance.

2 Related work

Decomposition and pattern-matching techniques in HLS have been applied separately to reduce the circuit area. This way, the implementations reachable in every case depend on the description style used in the specification, i.e. on the ability and expertise of the designer. However, the cooperative application of both techniques will lead to higher quality designs.

2.1 Decomposition of operations

Although state-of-the-art algorithms using decomposition techniques produce relatively good results, they cannot exploit all their potential, since similar decompositions are applied to all the specification operations. Instead, a particular study to select the best decomposition in each case would produce more efficient results, independently of the parameter to be optimized.

In order to reduce the circuit area, the work presented in [CoCL00] analyses the viability of both distributing the execution of one operation over several FUs, and of using large FUs to calculate several operations in the same cycle, provided that input operands of different operations are separated with zeros to isolate the results. Only the first methodology applies the decomposition technique, but limited to the fragmentation of some specification operations into several smaller ones of the same types, what usually avoids achieving optimal resource sharing. In [MRMH06] the authors propose a variant of the force-directed scheduling algorithm that fragments

long operations into several smaller ones that are executed in different cycles. It balances the computational cost of the operations executed per cycle in order to increase the HW reuse, and therefore to reduce the area of the implementations. A bit-level allocation algorithm is presented in [MoMH03a]. It augments HW reuse thanks to the execution of one operation over several narrower FUs. Only decompositions of additions and multiplications are considered, which are broken down into several smaller operations with the same type as the original. However, the main limitation of these algorithms [MRMH06] [MoMH03a] lies in the selection of the appropriate decomposition to be performed in each case. Although decompositions try to contribute to reach global objectives, e.g. the balance of the computational cost of the operations executed per cycle, or the allocation of the minimum set of FUs needed to execute a given scheduled data-flow graph (DFG), they are performed locally, i.e. without consideration of the type, representation and width of the remaining specification operations. This prevents binding algorithms from reusing big FUs and obliges them to excessively fragment specification operations in order to achieve an optimum reuse of the HW components, thus leading to complex designs. The fragmentation not only affects the FUs, but also the storage and routing units, as well as the controller.

The algorithm presented in [MRMH07] optimizes the area of multicycle FUs within the RTL circuits obtained from behavioral synthesis. It substitutes multicycle FUs for several monocycle chained ones, which are reused in some cycles of the execution of the multicycle unit. However, the optimization of the RTL circuit is quite limited since all the design decisions have already been taken (scheduling, allocation, and binding).

Circuit performance has also been improved through decomposing long operations into several faster ones that are executed in different cycles. The algorithms proposed to achieve this goal [MRMH06] have intensively fragmented the specification operations into many atomic ones. They complicate the posterior HW allocation and binding and produce in most cases datapaths with bigger area due to the increment in the routing resources and wires.

Decomposition techniques have also been applied to reduce the power consumption [ErKP99] [ChJC00]. The research work presented in [ErKP99] aims at reducing the capacitance and switching activity on the datapath arithmetic units. Multicycle operations are fragmented into several monocycle operations to be later scheduled in consecutive cycles. In [ChJC00] the authors allow the execution of one operation over several linked FUs in order to reduce the dynamic power consumption of FUs. This goal is achieved when the FUs that execute narrower operations in any cycle are substituted for several smaller FUs, such that some of them remain idle in some cycles. In both cases, the static consumption remains nearly constant, as well as the datapath area.

The technique proposed in this thesis overcomes the above limitations, allowing the decomposition of operations into several new ones of different types at early stages of the design process. The decompositions are not performed locally (considering only the operation to be fragmented). Instead, they try to increase the regularity of the decomposed DFG, as it is known that the regularity extraction contributes to the circuit area reduction.

2.2 Pattern matching

Regularity extraction has been successfully exploited in the behavioral synthesis domain. Pattern-based synthesis involves two major tasks: pattern recognition and pattern matching. Pattern recognition extracts the set of frequent patterns in behavioral specifications, and pattern matching checks the presence of a given pattern. Early works in pattern-based HLS tried to schedule and bind operations subject to a pattern library supplied by the designer [BrRo97] [CKGP96] [LKMM95]. More recent research has been focused on structural pattern recognition mainly based on graph matching [HuWP03] [InWM00] [CoJi08] [TKVI04]. Exact matching [HuWP03] [InWM00] has been extended to generalized matching [CoJi08] [TKVI04], where the exact matching requirements are relaxed to count more pattern occurrences within the specifications.

General pattern recognition and synthesis frameworks to reduce interconnect costs in FPGA designs are presented in [CoJi08]. The similarity of structures is captured by a mismatch-tolerant metric that handles several program variations as bit-width, structural, and port variations. The algorithm in [TKVI04] also addresses the problem of generalized matching by applying constraint satisfaction techniques in order to achieve reduced datapaths. Even these generalized matching algorithms, where some structural transformations are considered, cannot reach the benefits obtained from decomposing some specification operations.

Decomposition technique permits the fragmentation of operations into several independent ones (except for the data dependences among them) with different types, representations, and formats. These new operations may belong to different patterns, and in consequence can be scheduled in different cycles and executed over different FUs. Decomposition technique is more versatile than generalized matching, and used in combination with pattern-matching based HLS algorithms can produce higher quality designs.

3 Design methodology and proposed algorithm

This research work proposes a design methodology based on the application of both decomposition of operations and pattern matching techniques. In order to analyze its viability, an algorithm based on our design methodology has been implemented. The proposed algorithm takes as input both a behavioral specification and some time constraints, given by the cycle length and optionally the latency (number of clock cycles). The output is a complete datapath, formed by FUs, registers, multiplexers, and some glue logic, and a controller. The design methodology applied during the synthesis process produces datapaths where the number, type, and width of the resources are, in general, independent of the circuit specification, i.e. of the operations and data objects used.

The algorithm comprises four phases. In the first one a set of candidate patterns is selected. This selection is performed taking into account the cycle length constraint and the number of occurrences among the specification operations. In the second phase the quality of every selected pattern is measured, and the best pattern is chosen to allocate the FUs required to execute it. Then, the scheduling and binding of the operations matching the selected pattern is performed, followed by a rearrangement of the candidate patterns. This second phase is executed in a loop until all the patterns have been handled and it is not possible to create new ones. In the third phase, the remaining operations are scheduled and bound, trying to allocate the smallest number of new resources through sharing the HW used to execute the patterns. During the fourth and last phase slack times are exploited to replace large FUs with smaller ones. The following subsections explain in deeper detail the four phases.

3.1 Selection of operation patterns

The candidate patterns chosen are obtained from the decomposition of operations whose execution times are longer than the specified cycle length, as well as from the chaining of several operations and fragments that can be executed in a single cycle. Not all the possible decompositions and chaining are considered, as only frequent patterns (with a significant number of occurrences) are included in the selection. This way, the potential reuse of HW resources needed to execute the pattern is used to guide the synthesis process. This phase is divided into the four stages described below.

1) Identification of multicycle operations

The first stage to select the set of patterns becomes the decomposition of operations whose execution times are longer than the cycle duration. In order to decide if one operation can be calculated in a single cycle, the delay of the fastest FU available in the target library is compared to the cycle length.

2) Decomposition of multicycle operations

The operations selected in the first stage of this phase are fragmented into several smaller ones during the second stage. These decompositions try to produce several fragments with identical type, representation, and width. This way, the reuse of the datapath resources is potentially incremented. The number of fragments obtained from the decomposition of one operation matches the division of the operation delay into the cycle length. In order to calculate the operation delay we have avoided any formula that may produce a value distant from the execution times of the resources available in the design library. Instead, we have assumed the operation delay matches the execution time of the fastest FU available in the library. This consideration does not necessarily require the allocation of fast FUs, which are usually the biggest ones. On the contrary, slack times will be computed to allocate slower and smaller FUs [BGTS03] whenever possible.

In order to produce new operations with fast execution times, multicycle operations are split into several smaller fragments dividing vertically their calculus matrices. From the fragmentation of additions, the algorithm always obtains new additions, and from the fragmentation of multiplications it obtains multiplications, additions, and logical operations. In order to reduce the number of new operations, two new operation types have been defined. These new operations types, called *TUp* and *TDown*, group some logical operations and chained additions included within a triangular portion of the calculus matrix of one multiplication. *TUp*/*TDown* performs the calculus included inside an isosceles triangle with a right angle on its top/bottom. In the rest of this thesis report we will refer to *TUp* and *TDown* operations as triangles. They can be synthesized adding new modules to the target library. The design of the new modules can be easily performed and optimized using components of the target library. In [1RMMH06] a possible implementation of these operations based on adders and AND gates is presented.

3) Selection of frequent patterns

The third stage of this phase consists in the association of some operations and operation fragments in order to compose the patterns used in the following phases. To obtain the DFG considered at this stage, the slowest operations of the original DFG are substituted for their corresponding operation fragments using the decompositions described above. The candidate patterns are formed by several chained operations and fragments that can be executed in a single cycle. Note that one pattern may include fragments of different operations, but none of them is completely included in the pattern, and also different fragments of one operation may belong to different patterns. These features are the consequence of the decompositions performed previously, and imply an expansion of the design space explored by the algorithm with new and smaller implementations unreachable with previous approaches in HLS.

In order to group operations and fragments to form the patterns, the algorithm selects, each time, one operation (the pattern head) and some successors whose accumulative execution time is smaller than the cycle length. The pattern execution time is calculated taking into account the bit-level chaining (BLC) of arithmetic operations [MaLD97], which allow the execution in parallel of some bits of several chained operations with rippling effect.

The calculation of the candidate patterns may have exponential computational cost if all specification operations are considered as the head of several patterns. Instead, we have defined a parameter-based heuristic to limit the number of patterns considered. In our heuristic only the most promising operations (those with a significant number of repetitions in the specification) are selected to become a pattern head, and candidate patterns are created by adding a different number of successors each time. Furthermore, all the candidate patterns formed from the selected pattern heads are not considered in the following phase either. Only those with a significant number of occurrences among specification operations are taken into account.

The *PatternHead* (*PH*) parameter is used to decide if one operation can become the head of a pattern. Its value corresponds to the minimum number of occurrences in the specification needed to become one pattern head. It seems obvious that its selection must be done taking into account the circuit latency, and it should not be much smaller than the latency, what would result in a small number of occurrences and poor FUs reuse. The *CandidatePattern* (*CP*) parameter is used to select the subset of patterns from those beginning with the pattern heads. Its value corresponds to the minimum number of occurrences in the specification needed to become one candidate pattern. Again, its value must have a strong relation with the circuit latency to avoid a poor reuse of the HW resources. Note that *CP* must be less than or equal to *PH* to guarantee that every head operation can be used to form a set of candidate patterns. This relation assures that every head operation becomes a pattern itself. The values of these parameters represent the computational effort, selected by the designer in every case, to search within the solution space. They have a direct influence on the quality of the synthesized implementations. Smaller parameter values usually produce larger implementations, and bigger values may produce excessive fragmentation. However, the appropriate values may be different for every design. Even they may be different if the circuit is synthesized for different time constraints. Relatively good parameter values can be selected taking just into account the circuit latency. Note that the number of operations and their mobility may change as a consequence of decompositions, and therefore their influence on parameter values may be also different at every stage of the synthesis process. An appropriate *PH* value corresponds to the number of clock cycles, since the existence of such number of operations of the same type potentially contributes to achieve a balanced distribution of operations to cycles. And a proper *CP* value can be obtained if *PH* is relaxed. Several experiments have concluded that the *CP* value obtained from the multiplication of *PH* by a number in the range [0.7–0.9] usually reports high quality designs. Nevertheless, these parameter values can be fine tuned by the designer after short experimentation in order to further improve the quality of designs.

If no operation meets *PH* due to a small number of occurrences in the specification, then the algorithm tries to increase the occurrences of the most frequent ones via the decomposition of some operations in order to become a head pattern. This is performed in several steps. First, operations are sorted out from the most frequent to the least, such that they are considered in this order afterwards. Then, for every operation, the algorithm checks to see if it is possible to decompose some operations to increase the occurrences of the selected one to become a head pattern. Decompositions are only performed if they result in the creation of a new head pattern. Note that the decomposition of the most frequent operations produce, in addition to the required fragment, a set of new operations whose presence in the specification may affect the number of occurrences of other operations. This fact usually results in more head operations without appeal for new decompositions. The set of decompositions applied to identify head operations are:

- *Additive operations* (subtraction, comparison, minimum, maximum, absolute value...) may be transformed into additions, and some glue logic.
- *Multiplicative operations* (mac, factorial, power...) may be transformed into multiplications, additions, and some glue logic.
- *Multiplications* can be transformed into triangles, new multiplications, additions, and logic operations.
- *Triangles* can be transformed into multiplications, triangles, additions, and logic operations.

The number, type, and width of fragments depend on the decomposition performed, which is guided by the features of the selected operation to become a pattern head. All the possible decompositions are stored in a database, and checked to identify head operations and candidate patterns. Other decompositions can be easily added to the database without changes in the algorithm.

Once head operations have been identified, a set of candidate patterns is created from every head chaining successor operations. Note that for every

candidate pattern formed by n operations, there exist *n*-1 patterns with the same pattern head and a subset of chained operations. Our approach uses a generalized pattern-matching algorithm based on Boyer-Moore to count the occurrences of recognized patterns. Bit-width and some structural variations are considered (the set of decompositions described previously to extract common operative kernels among different operations). Variations in the structure result in some fragmentations when they contribute to the creation of new candidate patterns satisfying *CP*, likewise the decompositions done to identify the heads.

4) Calculation of the circuit latency

The circuit latency, if not provided by the designer, matches the division of the execution time of the monocycle implementation (taking into account the BLC) into the cycle length. Again, if the target library includes several resources able to implement one operation, then the fastest one is considered in this calculus. Note that this calculus produces the minimum number of cycles needed to execute the behavior for a fixed value of the cycle length.

3.2 Pattern-based scheduling and binding

Some candidate patterns are selected to allocate the set of FUs needed to execute them, and some operations matching the selected patterns are scheduled and bound to the new HW resources. This phase is repeated until all the candidates have been considered. Four different stages are executed each time.

1) Measurement of the quality of patterns

The aim of this stage is the selection of the best pattern in order to allocate the necessary HW to execute it. The best pattern produces the biggest reuse of the allocated HW among the specification operations, and in consequence usually contributes to achieve smaller datapath areas. Three main factors are considered to measure the quality of a pattern: the number of pattern occurrences, their mobility, and the number of operations and fragments that form the pattern.

The number of pattern occurrences and their mobility influence the reuse of the HW needed to execute the pattern. The potential reuse of the HW resources grows with the number of occurrences whenever the mobility of the involved operations allows the execution of the same pattern in different cycles. Note that the mobility of a set of chained operations matching a particular pattern comprises the intersection of their operations mobility, i.e. the set of operations can only be scheduled in the cycles included in the mobility of all of them.

The number of operations and fragments that form the pattern influences the storage requirements, since these needs usually decrease as the number of chained operations executed in one cycle grows.

2) Allocation of HW resources

Once the quality of all the candidate patterns is calculated, the algorithm selects the one with the biggest quality value and allocates a set of HW resources from the components library able to execute the operations within the pattern. If several FUs are available to execute one operation, then the algorithm selects the fastest one to meet the time constraints imposed by the designer. This decision preserves the mobility of unscheduled operations that could be reduced if slow FUs were allocated, avoiding slow circuits and increments in the storage requirements (due to the impossibility of chaining several data-dependent operations within the same cycle).

However, slow FUs could be used to reduce the circuit area without affecting performance in many cases. The substitution of large and fast FUs for smaller and slower ones takes place during the fourth phase of the algorithm.

3) Scheduling and Binding of Operations

The algorithm schedules a set of operations chains matching the pattern selected previously, trying to cover as many cycles as possible in order to produce a better use of the HW allocated in the previous phase. The scheduling algorithm used in this phase is a variant of the force-directed scheduling (FDS) [PaKn89], where the sets of operations matching a pattern have been considered as a unique operation. This way, several chained

operations are scheduled in the same cycle each time. And in order to meet the resource constraint imposed in the previous stage (allocated HW), a maximum of one chain of operations (matching the selected pattern) is scheduled in each cycle. The binding is performed directly as the newly scheduled operations are assigned to the HW resources allocated in the previous stage.

4) Adjustment of operation patterns

Every candidate pattern is checked to meet *CP* among the unbound operations. The unsuccessful checks result in the removal of the corresponding patterns from the candidates. Note that one already handled pattern can still belong to the set of candidate patterns, since only a maximum of one chain of operations (matching the selected pattern) per cycle is scheduled each time. The pattern will be handled again whenever it obtains the best result of the gain function.

The *CP* parameter check may also result in the removal of all the pattern candidates. In this case, the algorithm tries to identify a set of new candidates applying the decomposition of several unbound operations. As previously stated, it identifies first head operations satisfying *PH*, and afterwards it adds successors to form the candidates. Note that in order to identify head operations and candidate patterns, only unbound operations are considered. If no head patterns are identified, even considering the decomposition of operations, then the third phase is executed to schedule and bind the remaining operations.

3.3 Scheduling and binding of remaining operations

Once the set of candidate patterns is empty, the algorithm schedules and binds the remaining operations trying to allocate the smallest number of new resources. First, it tries to use the previously allocated FUs to execute the patterns in the cycles where they remain idle. FDS algorithm is again used to schedule the remaining operations. This time the pattern operations are considered one by one, in order to maximize the reuse of the FUs already

allocated. The possible decompositions of unscheduled operations are also considered to increase the FUs sharing. In order to reuse the storage area, the binding tries to reuse the same registers to store the input and output operands of every FU.

3.4 Area optimization of datapath

The synthesized datapath is optimized to reduce the circuit area without affecting its performance. Delay budgeting is used to replace large FUs by components in the target library with simpler structures and smaller area. The design techniques proposed in [BGTS03] to solve the budgeting problem have been applied to our datapaths to improve their areas.

4 Experimental results

In order to measure the quality of the proposed algorithm we have synthesized several circuits and compared our results to those obtained from previous approaches. The experimental framework comprises:

1) HLS of behavioral descriptions using the proposed algorithm based on multi-level decompositions.
2) HLS of behaviors using bit-level scheduling and allocation algorithms specially suited for area minimization [MRMH06] [MoMH03a].
3) HLS using the FDS [PaKn89] approach and a pattern-based allocation algorithm derived from [CoJi08].
4) Logic synthesis of every RTL circuit produced by the aforementioned algorithms. The commercial tool Synopsys Design Compiler (DC) and the component library VTVTLIB25 (based on 0.25 µm TSMC technology) by Virginia Tech. have been used to synthesize the RTL descriptions. The implementations of the new triangle operators have also been added to the library.

5) Extraction of the areas and execution times of the implementations from the post-synthesis reports generated by DC. The reported areas include the complete datapath (FUs, storage and routing units, and glue logic) and the controller, and are measured in number of equivalent gates (inverters). The execution time is measured in nanoseconds.

In all the experiments we have fixed *PH* to the circuit latency value, and *CP* to 0.7 times the circuit latency. Therefore, only operations repeated in the specification at least as many times as the circuit latency have become head operations, and patterns with a frequency smaller than 0.7 times the circuit latency have been discarded. Different tests have demonstrated that our algorithm produces high quality circuits with the actual values of both parameters, and the execution times are similar to those of the algorithms selected to compare our results with.

4.1 Cycle length versus area

Time constraints typically produce a direct influence on the area of the circuits obtained from the HLS. Both small and large clock cycles have a bad influence on the circuit area. On one hand, small clock cycles increase the number of multicycle operations and the disadvantages (in terms of HW sharing) associated to multicycle FUs arise. On the other hand, large cycles allow the execution of behaviors in a small number of cycles by chaining several data-dependent operations in each cycle. The set of operations executed in the same cycle cannot share any HW resource, such that many FUs are needed to execute the behavior. The appropriate cycle length and latency depend on different factors: the specification size, the data dependences among operations, and the description style used in the specification (number, type, representation, and width of the specification operations). However, the proposed technique mitigates some of these influences and synthesizes high-quality circuits with independence of the time constraints imposed by the designer.

In order to illustrate this dependency, we have synthesized the 16-element 16-beam beamformer with 16th order FIR filter as described in [DuPa95]. The internal loops have been unrolled to obtain a specification formed by 272 multiplications and additions. The synthesis process has been performed for several different values of the cycle length. The latency has also been constrained in all the cases to the minimum value, as explained before. However, bit-level and FDS scheduling algorithms have not always succeeded to reach a schedule meeting the cycle length and latency constraints. In these cases, arithmetic transformations based on carry-save representation [Vele04] are applied to accelerate critical paths in the DFG. In essence, these transformations try to maximize the opportunities to use compressor trees in combinatorial circuits. First multiplications are substituted for logic operations and additions. Then, additions are rearranged to maximize multi-input additions, which are finally replaced by a compressor tree followed by a single carry-propagate final addition. The use of optimized specifications allows bit-level and FDS scheduling algorithms to produce valid schedules (meeting time constraints). In most cases, the circuit latency is also reduced in several cycles at the cost of large increments in the circuit area. The proposed algorithm has been able to achieve valid schedules in all the cases, as well as smaller implementations.

For cycle length constraints above 50 ns, all the algorithms succeed to reach valid schedules. In particular, the three algorithms produce implementations with the same latency value. In these cases, the implementations given by our algorithm are on average 44% and 10% smaller than those synthesized with FDS and pattern-guided allocation, and bit-level algorithms, respectively. FDS and bit-level algorithms are not able to reach valid schedules for cycle length constraints under 60 ns. As explained above, the optimized specification has been used to perform the synthesis. Both algorithms achieve valid schedules from the transformed specification, with latencies several cycles smaller than the given constraint (up to 20% smaller than the latency values of our implementations). However, the circuit areas are around 70% larger in comparison to the implementations synthesized for longer cycle

lengths. For these designs, the amount of area we saved averages 73% in comparison to FDS and pattern-guided allocation, and 60% compared to bit-level algorithms. The experimental results are shown in figures 5.13 and 5.14 on chapter 5.

4.2 Pattern identification

The effectiveness of our pattern identification approach has been measured in comparison to the pattern-matching techniques proposed in [CoJi08], which handles several program variations as bit-width, structural, and port variations. In order to be fair with the comparison, we have only considered the patterns identified whose number of occurrences meets *CP* (fixed to 0.7 times the circuit latency). The number of patterns counted on our algorithm corresponds to the total number of patterns identified during the complete HLS process (not only the ones identified during the first phase). Both the number of patterns and the number of occurrences have been doubled on average, thanks to the decomposition of operations. This increment in the number of patterns and occurrences has allowed reaching quite smaller designs in all the cases. The area saved is around 42%, on average. The experimental results are shown in table 5.2 on chapter 5.

4.3 Multicycle versus monocycle FUs

The decomposition of operations initially performed transforms multicycle operations into several chained monocycle ones. Due to this decomposition, the number of operations augments, and the synthesized datapath is only formed by monocycle FUs. Instead, the implementations obtained by other approaches always contain multicycle FUs to execute multicycle operations. The area saved by our approach is due to both the reuse of FUs to execute different computations of multicycle operations, and the extraction of new candidate patterns from the decomposition of operations.

In order to illustrate this fact, we have studied the datapath synthesized from a description of a fifth order elliptical wave filter. FUs area has been

reduced affecting significantly the total circuit area, as FUs occupy around 60% of the implementations. FDS and pattern-based allocation algorithms have synthesized a datapath with multicycle FUs that occupy around 40% of the total FUs area. However, the area reduction achieved is over 40%, what means an additional reuse of monocycle FUs. Bit-level algorithms have also decomposed multicycle operations into several monocycle ones, in order to balance the computational cost of the operations executed per cycle. In this case, our pattern-based synthesis algorithm has achieved a better reuse of functional resources. The experimental results are shown in figure 5.15 on chapter 5.

The decomposition of operations into several smaller ones to be executed in different cycles requires some storage units to save the partial results and carries calculated in the first cycles. However, once any operation fragment is calculated, the complete input operands are not longer necessary in many cases (the unexecuted fragments usually require only some bits of the input operands). In these cases the units that store the input operands can be used to save the partial fragment results. This reuse can only be achieved if bit-level algorithms are used to perform the allocation and binding of storage resources [MRMH06]. The proposed method saves additional storage and routing resources because pattern-guided synthesis favors the chaining of operations, thus avoiding the storage of intermediate values. The controller area is nearly equal in the three implementations.

The total amount of area saved has reached 42% in comparison to the bit-level algorithm and 49% compared to FDS and pattern-based allocation. The amount of specification operations has grown with our approach around 26%. However, the decompositions to create candidate patterns have been performed progressively, except for the initial decomposition of multicycle operations. For this reason the execution times of our algorithm are only slightly higher than those reported by previous approaches. The experimental results are shown in table 5.3 on chapter 5.

4.4 Real application example

In addition to the classical benchmarks, we have synthesized some modules of the ADPCM decoding algorithm described in CCITT Recommendation G.721. The set of synthesized modules are Inverse Adaptative Quantizer (IAQ), Quantizer Scale Factor (QSF), Output PCM Format Conversion (OPFC), Synchronous Coding Adjustment (SCA), and Tone and Transition Detector (TTD). OPFC and SCA modules have been synthesized together due to their proximity in the decoder DFG, and IAQ, QSF, and TTD independently.

FDS and pattern-based allocation take advantage of large cycle lengths, producing datapaths with a small number of multicycle FUs, and consequently the HW waste associated to multicycle units is mitigated. The selection of smaller cycle lengths produces an increment in the latency, the reuse of monocycle FUs, and the multicycle FUs requirements. Due to the increase in the number of multicycle FUs, circuits synthesized with FDS and pattern-based allocation present, for small cycle lengths, slight area reductions in comparison to implementations with larger cycles. The experimental results are shown in table 5.4 on chapter 5.

Both bit-level algorithms and our approach can benefit from reduced cycle lengths fragmenting specification operations in order to increase the HW reuse. In consequence, circuits synthesized with both approaches are quite smaller than those produced with FDS and pattern-based allocation. In general, the amount of area saved augments with the cycle length reduction. However, our circuits are always smaller than those synthesized with bit-level algorithms thanks to our pattern-guided decomposition of operations, which contributes to save additional storage and routing area. The amount of area saved averages around 46% in comparison to FDS and pattern-guided allocation, and 20% compared to bit-level algorithms. Additionally, the number of clock cycles is smaller in our designs due to the minimal latency given by our algorithm.

5 Conclusion

In this thesis, a design methodology guided by the decomposition of specification operations in order to improve the circuit area on a pattern-based mode has been proposed. The decompositions are performed progressively in order to increase the number of identified patterns and their occurrences. In this way, internal execution of complex operations and real data dependences among operation fragments are considered to extract the candidate patterns. This data is hidden when the pattern identification is performed at the operation level, even if program variations are handled. The decompositions are performed during the scheduling and binding in order to take advantage of frequent patterns at early stages of the design process. The experimental results show great area reductions in comparison to previous approaches, as well as significant improvements in the circuit performance, due to the minimum latency value calculated by our method. To the best of our knowledge, this is the first approach that applies the decomposition of operations to the time-constrained HLS, and to pattern-matching techniques with the aim of reducing the circuit area.

BIBLIOGRAFÍA

[AhSU86] A. V. Aho, R. SEIT, J. D. Ullman. *Compilers: Principles, Techniques and Tools.* Addison-Wesley, Reading, MA, 1986.

[BGTS03] E. Bozorgzadeh, S. Ghiasi, A. Takahashi, M. Sarrafzadeh. *Optimal Integer Delay Budgeting on Directed Acyclic Graphs*. Proc. Design Automation Conference, DAC 2003.

[BoMo77] R. S. Boyer, R. S. Moore. *A Fast String Searcher Algorithm*. Communications of the ACM. 1977

[BrRo97] O. Bringmann and W. Rosenstiel. *Resource sharing in hierarchical synthesis*. Proc. International Conference on Computrer-Aided Design, ICCAD 1997.

[CAIS77] C. Alexander, S. Ishikawa, M. Silverstein. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, 1977.

[CaWo91] R. Camposano and W. Wolf, eds. *High-Level VLSI Synthesis.* Springer, 1991.

[ChCF03] D. Chen, J. Cong, Y. Fan. *Low-Power High-Level Synthesis for FPGA Architectures*. Proc. International Symposium on Low Power Electronics and Design, ISLPED 2003.

[ChJC00] J. Choi, J. Jeon, K. Choi. *Power Minimization of Functional Units by Partially Guarded Computation*. Proc. International Symposium on Low Power Electronics and Design, ISLPED 2000.

[CKGP96] M. R. Corazao, M. A. Khalaf, L. M. Guerra, M. Potkonjak, and J. M. Rabaey. *Performance optimization using template mapping for datapath-intensive high-level synthesis*. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, August 1996.

[Clou90] R. Cloutier. *Force Directed Scheduling Allocation and Mapping.* Proc. Design Automation Conference, DAC 1990.

[CoCL00] G.A. Constantinides, P.Y.K. Cheung, W.Luk. *Multiple-wordlength Resource Binding*. Field-Programmable Logic: The Roadmap to reconfigurable systems, LNCS, 2000.

[CoCL01] G.A. Constantinides, P.Y.K. Cheung, W.Luk. *Heuristic Datapath Allocation for Multiple Wordlength Systems.* Proc. Design Automation Test in Europe, DATE 2001.

[CoJi08] J. Cong, and W. Jiang. *Pattern-Based Behavior Synthesis for FPGA Resource Reduction*. Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2008.

[DeMi94] G. DeMicheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, New York, 1994.

[DuPa95] N. Dutt, P. R. Panda. *1995 High-Level Synthesis Design Repository*. University of California, Irvine.

[Dutt92] N. Dutt. *High-level Synthesis Workshop Benchmarks.* Univ. California, Irvine, CA, Technical Report 1992.

[ErKP99] M. Ercegovac, D. Kirovski, M. Potkonjak. *Low-power Behavioural Synthesis Optimization using Multiple Precision Arithmetic*. Proc. Design Automation Conference, DAC 1999.

[FaRS94] Y. Fann, M. Rim, R. Sain. *Global Scheduling for High-Level Synthesis Applications.* Proc. Design Automation Conference, DAC 1994.

[Gajs92] D. Gajski et al. High Level Synthesis: An Introduction to Chip and System Design, Kluwer (now Springer), 1992.

[GDGN03] S. Gupta, N. D. Dutt, R. K. Gupta, A. Nicolau. *Dynamic Conditional Branch Balancing during the High-Level Synthesis of Control-Intensive Designs.* Proc. Design, Automation and Test in Europe, DATE 2003.

[GeEl93] C. H. Gebotys, L. I. Elmasry. *Global Optimization Approach for Architecture Synthesis.* IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 12, Septiembre 1993.

[GKKR02] S. Gupta, T. Kam, M. Kishinevsky, S. Rotem, N. Savoiu, N. Dutt, R. Gupta, A. Nicolau. *Coordinated Transformations for High-Level Synthesis of High Performance Microprocessor Blocks*. Proc. Design Automation Conference, DAC 2002.

[GMRB08a] P.L. Garcia-Repetto, M.C. Molina, R. Ruiz-Sautua, G. Botella, Exploiting Internal Operation Patterns during the High-Level Synthesis of Time-Constrained Circuits. Proc. Euromicro Conference on Digital System Design, DSD 2008.

[GMRB08b] P.L. Garcia-Repetto, M.C. Molina, R. Ruiz-Sautua, G. Botella, Internal Operation Pattern Awareness in High-Level Synthesis. Proc. Conference on Design of Circuits and Integrated Systems, DCIS 2008.

[GuKa02] S. Gupta, S. Katkoori. *Force-Directed Scheduling for Dynamic Power Optimization*. Proc. International Symposium on VLSI, ISVLSI 2002.

[HuWP03] J. Huan, W. Wang, and J. Prins. *Efficient mining of frequent subgraphs in the presence of isomorphism*. Proc. IEEE International Conference on Data Mining, ICDM 2003.

[InWM00] A. Inokuchi, T. Washio, and H. Motoda. *An apriori-based algorithm for mining frequent substructures from graph data*. Proc. European Conference on Principles and Practice of Knowledge Discovery in Databases, PKDD 2000.

[JKSC01] J. Jeon, D. Kim, D. Shin, K. Choi. *High-level Synthesis under Multi-cycle Interconnect Delay*. Proc. Asia-Pacific Design Automation Conference, ASP-DAC 2001.

[KoWo02] A. Kountouris, C. Wolinski. *Efficient Scheduling of Conditional Behaviors for High-level Synthesis*. ACM Transactions on Design Automation of Electronic Systems, July 2002.

[KoWo99] A. Kountouris, C. Wolinski. *Hierarchical Conditional Dependency Graph for Mutual Exclusiveness Identification.* Proc. International Conference on VLSI Design, VLSI 1999.

[KuPa87] F. J. Kurdahi, A. C. Parker. *REAL: a Program for Register Allocation.* Proc. Design Automation Conference, DAC 1987.

[KuSu01] K. Kum, W. Sung. *Combined Word-Length Optimization and High-Level Synthesis of Digital Signal Processing Systems*. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Agosto 2001.

[LaMD94] B. Landwehr, P. Marwedel, R. Dömer. *OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming*. Proc. European Design Automation Conference, EDAC 1994.

[LKMM95] T. Ly, D. Knapp, R. Miller, and D. MacMillen. *Scheduling using behavioral templates*. Proc. Design Automation Conference, DAC 1995.

[MaLD97] P. Marwedel, B. Landwehr, R. Dömer. *Built-in Chaining: Introducing Complex Components into Architectural Synthesis*. Proc. Asia-Pacific Design Automation Conference, ASP-DAC 1997.

[MeMo04] M. Meribout and M. Motomura. *A Combined Approach to High-Level Synthesis for Dynamically Reconfigurable Systems*. IEEE Transactions On Computers, December 2004.

[MoHF96] R. Moreno, R. Hermida, M. Fernández. *Register Estimation in Unscheduled Data Flow Graphs.* ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 1, nº 3, Julio 1996.

[Moli05] M. C. Molina. *Reducción del desaprovechamiento de hardware en la síntesis de alto nivel de especificaciones heterogéneas.* Tesis doctoral. Universidad Complutense de Madrid, 2005.

[MoMH03a] M. C. Molina, J. M. Mendías, R. Hermida. *Allocation of Multiple-Precision Behaviours for Maximal Bit-Level Reuse of Hardware Resources.* Journal of Systems Architecture, vol. 49, nº 12, Diciembre 2003.

[MoMH03b] M. C. Molina, J. M. Mendías, R. Hermida. *Behavioural Specifications Allocation to Minimize Bit-Level Waste of Functional Units.* IEE Proceedings: Computers and Digital Techniques, vol. 150, nº 5, Septiembre 2003.

[MRBM09] M.C. Molina, R. Ruiz-Sautua, A.A Barrio, J.M. Mendias. *Subword Switching Activity Minimization to Optimize Dynamic Power Consumption*. IEEE Design & Test of Computers, vol. 26 (4) (2009), pp. 68-77. Jul-Ago 2009.

[MRGH09] M.C. Molina, R. Ruiz-Sautua, P.L. Garcia-Repetto, R. Hermida, Frequent-Pattern-Guided Multilevel Decomposition of Behavioral Specifications. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2009.

[MRGM09] M.C. Molina, R. Ruiz-Sautua, P.L. Garcia-Repetto, J.M. Mendias. *Performance-driven scheduling of behavioural specifications*. INTEGRATION, the VLSI journal. 2009.

[MRMH06] M.C. Molina, R. Ruiz-Sautua, J.M. Mendías, R. Hermida. *Bitwise Scheduling to Balance the Computational Cost of Behavioural Specifications*. IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, vol. 25 (1) (2006), pp. 31-46. Enero 2006.

[MRMH07] M. C. Molina, R. Ruiz-Sautua, J. M. Mendías, R. Hermida. *Area Optimization of Multi-Cycle Operators in High-Level Synthesis*. Proc. Design, Automation and Test in Europe, DATE, 2007.

[NeTh86] J. Nestor, D. Thomas. *Behavioural Synthesis with Interfaces.* Proc. International Conference on Computer Aided Design, ICCAD 1986.

[NiMa03] S. F. Nielsen, J. Madsen. *Power Constrained High-Level Synthesis of Battery Powered Digital Systems.* Proc. Design, Automation and Test in Europe, DATE 2003.

[OKBS05] S. Ogrenci, R. Kastner, E. Bozorgzadeh, M. Sarrafzadeh. *A Scheduling Algorithm for Optimization and Early Planning in High-level Synthesis*. ACM Transactions on Design Automation of Electronic Systems, January 2005.

[PaCh01] S. Park, K. Choi. *Performance-Driven High-Level Synthesis with Bit-Level Chaining and Clock Selection.* IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Febrero 2001.

[PaGa87] B. M. Pangrle, D. D. Gajski. *Design Tools for Intelligent Silicon Compilation.* IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 6 nº 6, 1987.

[PaKn89] P.G. Paulin, J.P. Knight. *Force-Directed Scheduling for the Behavioral Synthesis of ASICS*. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 8, nº6, Junio 1989.

[PaKy91] I. C. Park, C. M. Kyung. *Fast and Near Optimal Scheduling in Automatic Data Path Synthesis.* Proc. Design Automation Conference, DAC 1991.

[PaPa88] N. Park, A. Parker. *Sehwa: A Software Package for Synthesis of Pipelined Data Path from Behavioral Specification.* IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 7, nº 3, 1988.

[Park79] A. Parker et al. *The CMU Design Automation System- An example of Automated Data Path Design*. Proc. Design Automation Conference, DAC 1979.

[PeMH02] O. Peñalba, J. M. Mendías, R. Hermida. *A Global Approach to Improve Conditional Hardware Reuse in High-Level Synthesis*. Journal of Systems Architecture, vol. 47, nº 12, Junio 2002.

[PoRa94] M. Potkonjak, J. Rabaey. *Optimizing Resource Utilization using Transformation.* IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 13, nº 3, 1994.

[RMJD94] M. Rim, A. Mujumdar, R. Jain, R. DeLeone. *Optimal and Heuristic Algorithms for Solving the Binding Problem.* IEEE Transactions on VLSI Systems, vol. 2, nº 2, 1994.

[RMMH06] R. Ruiz-Sautua, M.C. Molina, J.M. Mendías, R. Hermida. *Pre-synthesis Optimization of Multiplications to Improve Circuit Performance*. Proc. Design, Automation and Test in Europe Conference, DATE 2006.

[Saut07] R. Ruiz-Sautua. Síntesis arquitectónica a nivel de bit guiada por tiempos de llegada. Tesis doctoral. Universidad Complutense de Madrid, 2007.

[SaZa90] A. Safir, B. Zavidovique. *Towards a Global Solution to High Level Synthesis Problems.* Proc. European Design Automation Conference, EDAC 1990.

[SeKP02] J. Seo, T. Kim, P. R. Panda. *An Integrated Algorithm for Memory Allocation and Assignment in High-Level Synthesis.* Proc. Design Automation Conference, DAC 2002.

[SHEE00] L. C. V. dos Santos, M. J. M. Heijligers, C. A. J. van Eijk, J. T. J. van Eijndhoven, J. A. G. Jess. *A Code-Motion Pruning Technique for Global Scheduling.* ACM Transactions on Design Automation of Electronics Systems, vol. 5, nº 1, Enero 2000.

[SHSS03] A. Stammermann, D. Helms, M. Schulte, A. Schulz, W. Nebel. *Binding, Allocation and Floorplaning in Low Power High-Level Synthesis.* Proc. International Conference on Computer Aided Design, ICCAD 2003.

[SlDr02] A. M. Sllame, V. Drabeck. *An Efficient List-Based Scheduling Algorithm for High-Level Synthesis.* Proc. Euromicro Symposium on Digital System Design, DSD 2002.

[TKVI04] Z. Terem, G. Kamhi, M. Y. Vardi, and A. Irron. *Pattern Search in Hierarchical High-Level Designs*. Proc. IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2004.

[TsHs90] F. S. Tsay, Y. C. Hsu. *Data Path Construction and Refinement*. Proc. International Conference on Computer Aided Design, ICCAD 1990.

[TsSi86] C. J. Tseng, D. P. Siewiorek. *Automatic Synthesis of Data Path on Digital Systems.* IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 5, nº 3, 1986.

[Vele04] A. K. Verma and P. Ienne. *Improved Use of Carry-Save Representation for the Synthesis of Complex Arithmetic Circuits*. Proc. International Conference on Computer-Aided Design, ICCAD 2004.

[WTLS03] W. Wang, T. K. Tan, J. Luo, L. Shang, K. S. Vallerio, L. Zhang, A. Raghunathan, N. K. Jha. *A Comprehensive High-Level Synthesis System for Control-Flow Intensive Behaviors.* Proc. Great Lakes Symposium on VLSI, GLSVLSI 2003.

[ZhGa99] J. Zhu, D. D. Gajski. *Soft Scheduling in High Level Synthesis.* Proc. Design Automation Conference, DAC 1999.