CENTRO DE INVESTIGACION Y ESTUDIOS AVANZADOS DEL I.P.N

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION COMPUTACION

HERRAMIENTA DE SINTESIS DE ALTO NIVEL PARA SISTEMAS DIGITALES

Tesis que para obtener el grado de Maestro en Ciencias con especialidad en Ingeniería Eléctrica opción Computación, presenta el Ing. Jesús Hernández Tapia.

Director: Arturo Díaz Pérez
Codirector: Adriano de Luca Pennachia

RESUMEN

La síntesis de alto nivel de sistemas digitales es un proceso en el cual se traduce la descripción funcional de un sistema en un nivel de abstracción alto, a una representación a nivel de transferencia entre registros. Esta última representación utiliza unidades funcionales, de almacenamiento y de interconexión para llevar a cabo las funciones especificadas. Esta tesis presenta una herramienta de software que automáticamente genera diseños a nivel estructural a partir de una descripción funcional. El dominio de aplicación de esta herramienta esta restringido al diseño de procesadores de expresiones aritméticas y lógicas. Tales diseños se generan en el lenguaje estándar VHDL para facilitar su posterior simulación e implementación.

Indice:

Capítulo 4: Planificación

Capítulo 5: Mapeo

Capítulo 6: Generación de código VHDL

Capítulo 7: Pruebas de HLSynth

CONCLUSIONES

REFERENCIAS

Capítulo 1

Introducción

La alta competitividad existente en la industria electrónica requiere generar circuitos cada vez más complejos en tiempos de diseño y fabricación cada vez menores. La demanda de diseños de circuitos de muy alta velocidad, sin defectos, y de circuitos integrados de aplicación específica en pequeños volúmenes, crece cada vez más. El diseño automático de sistemas digitales es un medio por el que se pueden lograr estos objetivos.

El diseño automático se entiende como la creación automática de un diseño físico a partir de alguna especificación funcional escrita en un alto nivel de abstracción. La especificación funcional consiste en describir que es lo que el sistema debe hacer sin preocuparse aún por la forma en que lo hará. La automatización reduce considerablemente el ciclo de diseño, permitiendo al diseñador experimentar con varios diseños para obtener, por ejemplo, el diseño óptimo en espacio o velocidad para una aplicación determinada. Idealmente, una vez que la especificación inicial se ha verificado y simulado, un circuito sintetizado a partir de ella no requiere verificación ni simulación, es decir, es correcto por construcción. También debe mencionarse que las especificaciones funcionales de alto nivel, son, generalmente, más cortas que las especificaciones estructurales de mas bajo nivel. La especificación estructural incluye más detalles acerca de la implantación del diseño. Por ello las descripciones funcionales son más fáciles de escribir, comprender y modificar. Por todo lo anterior, el diseño automático facilita considerablemente el diseño de sistemas complejos.

La construcción de sistemas digitales es un proceso de diseño que transforma descripciones de nivel alto a descripciones de nivel cada vez más bajo. Se han desarrollado varias herramientas de síntesis. Sin embargo, la mayoría de ellas automatizan los niveles bajos del proceso de diseño. La dificultad de desarrollar herramientas de síntesis en los niveles altos se debe a la gran variedad de sistemas y métodos para especificarlo. Esto hace que un sintetizador de alto nivel para aplicaciones generales sea prácticamente imposible. Más aún, los diseños producidos por un sintetizador, aunque correctos, en muchas ocasiones no satisfacen las restricciones de tiempo y espacio, por ejemplo, que aparecen en la construcción de un sistema digital, Sin embargo, es posible construir un sintetizador de alto nivel para aplicaciones específicas que proporcione resultados aceptables.

El objetivo de esta tesis es:

- Construir un sintetizador de alto nivel para generar automáticamente procesadores de expresiones aritméticas y lógicas.

El sintetizador toma como entrada una expresión aritmética o lógica y genera como resultado un diseño de un procesador, al nivel de transferencia entre registros, que ejecuta la expresión. Para este proceso es necesario hacer una planificación de las operaciones y hacer un asignamiento de las unidades funcionales que el problema de entrada requiere.

Tomando en cuenta posibles restricciones de tiempo y espacio es necesario aplicar soluciones diversas a los problemas de planificación y asignación (mapeo).

La salida del sintetizador es una máquina de estados finitos que consiste en una unidad de control, una ruta de datos y las unidades de interconexión, todo esto es descrito en el lenguaje VHDL, un lenguaje de descripción de hardware orientado a simulación de hardware digital.

A partir de una descripción en VHDL, se pueden utilizar sintetizadores que permiten transformar un diseño en el nivel de transferencia entre registros, a componentes discretos, dispositivos programables o circuitos integrados. Para el caso particular de la síntesis de alto nivel, las etapas del proceso son: Compilación, Planificación, Mapeo, Generación de la salida.

El resto de la tesis está organizado como sigue:

El capítulo 2 da una visión general del proceso de síntesis de alto nivel. Se explica a detalle cada uno de las etapas del proceso de síntesis haciendo énfasis en la forma en que se desarrollan en un sintetizador. Esto nos sirve para explicar las características deseables para el sintetizador desarrollado en esta tesis.

El capítulo 3 cubre la primera etapa del proceso de síntesis: la compilación. Se introducen los conceptos básicos de la compilación, se describen las diferentes formas intermedias de representación que se utilizan en los sintetizadores y se elige una para la herramienta desarrollada. La representación elegida es un grafo de flujo de datos que servirá como entrada a la segunda etapa de la síntesis.

El capítulo 4 discute la segunda etapa de planificación de la síntesis de alto nivel. Se formaliza el problema de la planificación, se identifican los problemas que se tiene al planificar las operaciones a realizar y como transformar el grafo de entrada a una máquina de estados finitos con ruta de datos. Finalmente se describen a detalle los tres algoritmos de planificación implementados en la herramienta construida.

La etapa de mapeo se presenta en el capítulo 5. Se describe el proceso de asignar unidades funcionales, de interconexión y almacenamiento a la ruta de datos obtenida en la etapa de planificación, con la finalidad de obtener un diseño estructural a nivel de transferencia entre registros. También se explica el algoritmo de mapeo que utiliza la herramienta construida.

El capítulo 6 describe el generador de código VHDL. Esta última etapa del proceso de síntesis consiste en generar automáticamente una salida en la que se especifique el

diseño generado. Esta salida es una descripción en VHDL, y se genera a partir de la estructura obtenida en la etapa de mapeo. La descripción en VHDL se hace al nivel estructural y especifica transferencias entre rcomponentes y/o registros los cuales pueden ser especificados por el diseñador o, por omisión, ser tomados de una biblioteca por el sintetizador.

El capítulo 6 muestra algunos diseños generados por nuestro sintetizador aplicando los diferentes algoritmos que éste provee con fines comparativos. Finalmente, se presentan las conclusiones de este trabajo.

Capítulo 2

Síntesis de Alto Nivel de Sistemas Digitales

2.1. Introducción.

La construcción de un sistema digital parte de la especificación de los requerimientos y las funciones del sistema. Con esta especificación, el diseñador aplica todas las técnicas de diseño de sistemas digitales que sean necesarias para obtener un circuito digital que cumpla con la especificación inicial. Este diseño puede implementarse en un circuito integrado, utilizando dispositivos programables (FPGAs, CPLDs, etc) o incluso con base en componentes discretos seleccionándolos de una biblioteca (Figura 2.1).

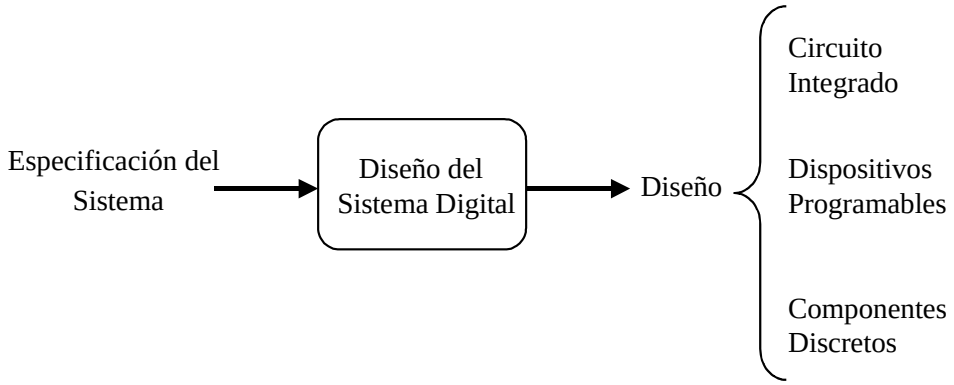

Figura 2.1: La construcción de sistemas digitales

En el proceso de diseño de sistemas digitales, existen cuatro etapas típicas que representan el nivel de abstracción en que se describe el sistema:

- Nivel de Sistema (Especificación del sistema)
- Nivel de microarquitectura
- Nivel lógico
- Nivel de circuito

El nivel de abstracción más alto es llamado nivel de sistema. En este nivel, un sistema es descrito como un conjunto de procesadores, memorias, etc., junto con la estructura general del sistema así como el flujo de la información. El siguiente nivel de abstracción hacia abajo es el nivel de transferencia de registros o microarquitectura. El sistema a este nivel es visto como un conjunto de unidades funcionales, de almacenamiento y de interconexión que determinan la estructura del sistema digital. El funcionamiento es descrito como una serie de transferencias de datos y transformaciones de los mismos que se efectúan entre las unidades de almacenamiento.

El nivel de abstracción inmediato inferior es llamado Nivel Lógico. En este nivel, el sistema es descrito por compuertas lógicas y flip-flops interconectados. El último nivel de abstracción es el nivel de circuito, que tiene mucho que ver con la electrónica de los dispositivos a un nivel muy bajo, es decir, transistores y conexiones.

Para cada etapa o nivel de abstracción existen diversas formas de representación del diseño. Estas formas corresponden a los siguientes dominios: Dominio Funcional, Dominio Estructural y Dominio Físico. Estos dominios pueden considerarse como el nivel de detalle con el cual se describe el hardware.

En el dominio funcional, el sistema se describe mediante la relación entrada-salida de sus señales. Se especifica qué es lo que el sistema debe realizar, sin pensar en cómo estará construido. Se concibe el sistema como una caja negra, para la que se definen sus entradas y salidas así como los resultados que debe proporcionar en función del tiempo. En el dominio estructural se contemplan los componentes cuya interconexión presenta una estructura y, finalmente, una vez que se cuenta con la estructura, debe pensarse en su fabricación y en la manera en que realmente estará construida, es decir, los elementos de más bajo nivel necesarios para ello. La representación que se obtiene a este nivel ignora tanto como sea posible, qué es lo que el sistema debe hacer, y busca representar la estructura del sistema en espacio o en silicio.

En la tabla 2.1, se presentan todos los dominios de representación de hardware aplicados a cada nivel de abstracción. El proceso de diseño de hardware consiste en comenzar del nivel de abstracción más alto e ir bajando hasta la implementación física del sistema.

Nivel de Abstracción	**Representación funcional**	**Representación Estructural**	**Representación Física**
Sistema	Diagramas de Flujo Algoritmos	Procesadores Controladores Memorias Buses	Gabinetes Tarjetas Módulos multi chip Chips
Microarquitectura	Transferencia de registros	ALUs Multiplicadores Multiplexores Registros Memorias	Chips Floorplans Módulos de floorplans
Lógico	Ecuaciones booleanas Secuenciadores	Compuertas Flip-Flops	Módulos Celdas
Circuito	Funciones de transferencia	Transistores Conexiones	Layouts Segmentos alambrados Contactos

Tabla 2.1. Niveles de abstracción en los dominios de representación del hardware.

Existen ciertas herramientas de apoyo al proceso de diseño, éstas son:

- Editores de esquemáticos: Permiten de manera gráfica instanciar e interconectar los componentes del sistema

- Simuladores: Permiten evaluar el comportamiento del sistema diseñado en función del tiempo, y verificar la correctitud del mismo.

- Sintetizadores: Transforman automáticamente un diseño en un nivel alto de abstracción a un nivel bajo.

Sería ideal tener un sintetizador para todas las etapas de diseño, sin embargo esto es algo difícil de lograr. Los sintetizadores han sido aplicados con éxito en las etapas bajas del proceso de diseño, ya que ahí existen métodos formalmente especificados que aseguran la correctitud del diseño generado. Sin embargo, en el nivel alto, aún no existen métodos generales que nos aseguren una correcta transformación a un nivel inferior. Los sintetizadores pueden clasificarse en dos tipos (Figura 2.2):

- Sintetizadores de alto nivel: Transforman un diseño a nivel de sistema en un diseño a nivel de microarquitectura.

- Sintetizadores de bajo nivel: Trabajan cualquiera de las tres etapas de más bajo nivel.

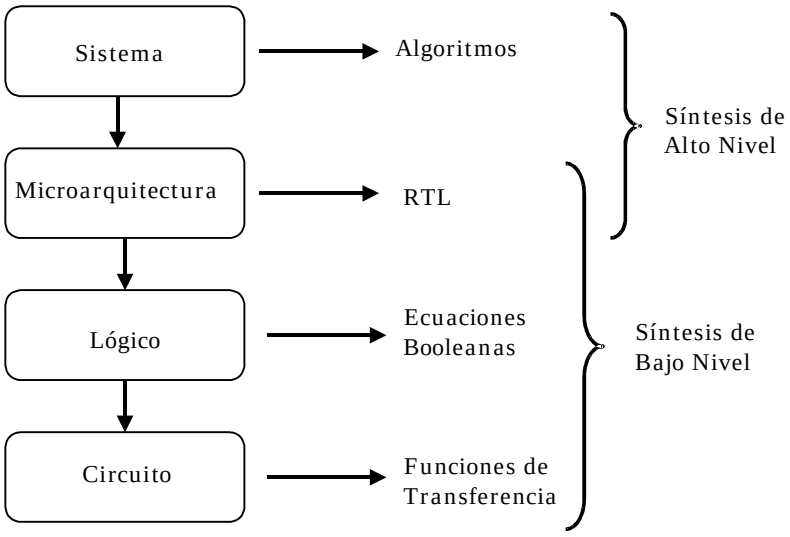

Figura 2.2. Etapas del proceso de diseño de sistemas digitales, y clasificación de los sintetizadores.

2.2. Síntesis de alto nivel.

Se conoce como síntesis de alto nivel al proceso que inicia a partir de una especificación inicial de tipo funcional (por ejemplo un algoritmo en lenguaje C o Ada o en un lenguaje de descripción de hardware) y produce una arquitectura capaz de ejecutar la especificación inicial [1] (Figura 2.3). La arquitectura es generalmente producida como una especificación al nivel de microarquitectura o de transferencia entre registros. En este nivel, el diseño se basa en el modelo de máquina de estados finitos con ruta de datos (FSMD: Finite State Machine with Datapath), que consta de

un conjunto de componentes conectados llamado “ruta de datos” (data-path) y un controlador que secuencializa y controla el funcionamiento de tales componentes [6].

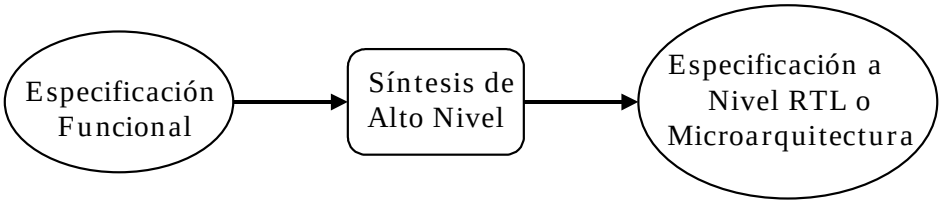

Figura 2.3: Entrada y salida de la síntesis de alto nivel

En la ruta de datos se efectúan todas las transformaciones de los datos para lograr el resultado final. Esta estructura se conforma de unidades de almacenamiento (registros, memorias, etc), unidades funcionales (ALUs, comparadores, etc.) y de interconexión (buses, multiplexores, etc).

En el modelo FSMD el control puede identificarse como una máquina de estados finitos que cuenta con un conjunto de estados y de transiciones entre ellos así como un conjunto de acciones asociados con cada transición.

En el proceso de síntesis de alto nivel, se llevan a cabo las siguientes tareas principales [6], ilustradas en la Figura 2.4:

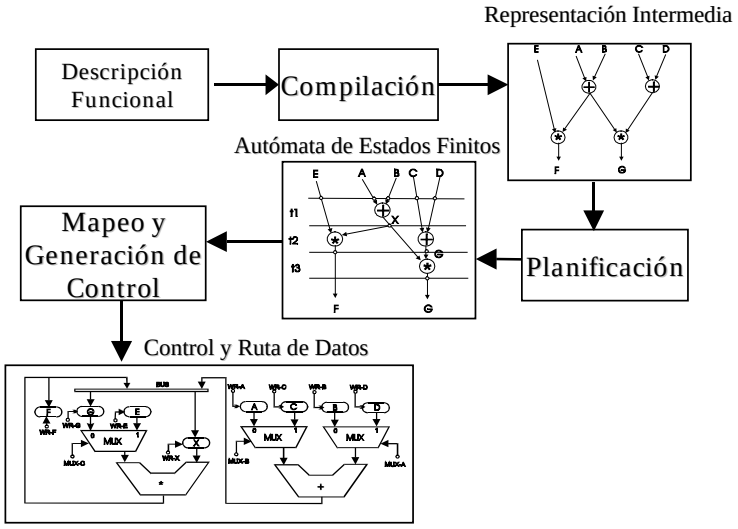

Figura 2.4. Tareas principales del proceso de síntesis de alto nivel.

- Compilación: Traduce la descripción del diseño a una representación intermedia a la que es más sencillo manipular en las tareas siguientes.

- Planificación: Divide la representación intermedia en pasos de tiempo o estados, generando un modelo de máquina de estados finitos.

- Mapeo: Particiona la representación intermedia con respecto al espacio. Es decir, asigna unidades funcionales a operaciones, unidades de interconexión a variables, etc.

- Generación de control: Consiste en la obtención del controlador que secuencializa el diseño y controla las unidades funcionales y de almacenamiento en la ruta de datos.

2.3. Compilación

La compilación consiste en la traducción de la descripción del diseño a una representación intermedia que es más adecuada para efectuar el proceso de síntesis.

Para que el proceso de síntesis sea automático es necesario haber definido previamente el dominio de aplicación y haber seleccionado alguna representación intermedia. Entonces se define o selecciona un lenguaje de definición que modele al sistema, es decir, que proporcione la descripción funcional del mismo.

Una herramienta de síntesis de alto nivel actúa como un compilador que mapea una especificación de alto nivel dentro de una arquitectura. Si la descripción funcional cambia, entonces también lo hará la arquitectura. Además, en este proceso deben considerarse restricciones y objetivos que deben satisfacerse en cuanto a espacio y velocidad.

Los grafos de control y de flujo de datos se utilizan como representaciones intermedias ya que representan de manera más práctica el control y las dependencias en el flujo de datos de una descripción dada. Esta representación intermedia es la salida que se obtiene en la fase de compilación del proceso de síntesis y a partir de la cual se realizarán todas las transformaciones necesarias en cada etapa de dicho proceso.

2.4. Planificación

La salida de la etapa de compilación, una forma de representación intermedia como grafos de flujo de datos y grafos de control de flujo, por ejemplo, proporcionan al proceso de planificación todas las operaciones que debe realizar el sistema.

La planificación asigna todas las operaciones a estratos de tiempo, de tal forma que cada operación es ejecutada en un estrato. Cada estrato de tiempo corresponde a un estado en una máquina de estados finitos del modelo FSMD.

2.4.1 Ejemplo de planificación

Con el fin de lograr una mayor comprensión de los primeros pasos del proceso de síntesis, se mostrará un ejemplo simple en el que es posible visualizar los resultados después de aplicar la compilación y la planificación.

La descripción funcional que se ha propuesto se encuentra descrita en un lenguaje de alto nivel, y dicha descripción se hace en un nivel alto de abstracción:

$$F = E * (A+B)$$
$$G = (A+B) * (C+D)$$

La representación intermedia que se obtiene después de la compilación es el grafo dirigido de flujo de datos mostrado en la Figura 2.5a. Después de aplicar alguno de los métodos de planificación en el que se hace una optimización en cuanto a los recursos utilizados, se obtiene la planificación de la Figura 2.5b [11]. En la Figura se puede observar que la planificación de las operaciones se logra en tres estados o pasos de control y ello debido a que se trató de optimizar el espacio aún cuando se sacrifica la velocidad.

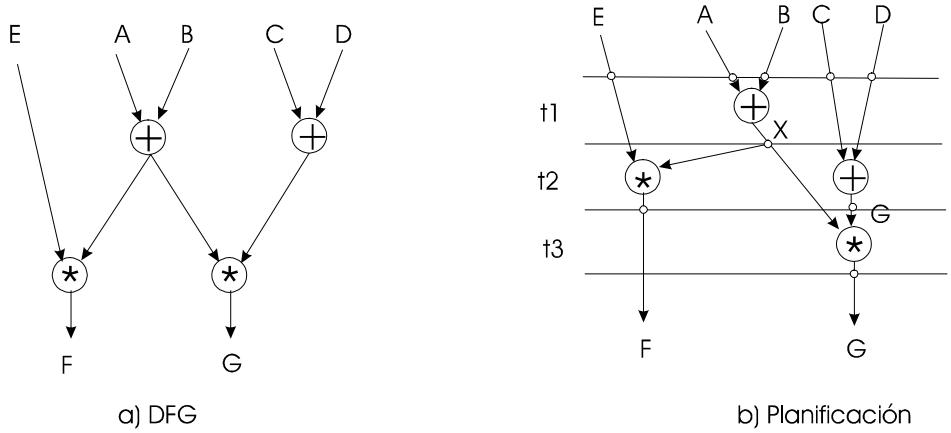

Figura 2.5: Ejemplo de planificación

Se logró utilizar únicamente dos unidades funcionales, una de ellas es un sumador y la otra es un multiplicador. La razón es que las dos operaciones de suma que aparecen en el DFG sin planificar (Fig. 2.5a), pueden ser ejecutadas en estados diferentes ya que no existe ninguna dependencia de datos entre ellas, esto ayuda a que únicamente se utilice una unidad funcional que se encargue de la adición. Lo mismo ocurre con las operaciones de multiplicación, y debido a que ambas multiplicaciones esperan el resultado de la suma A+B, esta suma es la que se realizará primero en la planificación para suministrar su resultado al multiplicador de la operación E*(A+B) en el estado t2, mientras simultáneamente se ejecuta C+D que es requerida hasta t3.

A continuación se discutirán algunos de los algoritmos existentes para el proceso de planificación, clasificados en cuatro grupos principales [6] (fig. 2.6).

2.4.2. Algoritmos básicos

Los algoritmos básicos de planificación son las formas más simples de realizar la planificación. Estos asumen que el número de unidades funcionales ha sido previamente especificado, y las operaciones en el grafo se ordenan respetando la

dependencia de datos [9]. La planificación resultante puede no ser la óptima, pero si la más rápida de obtener.

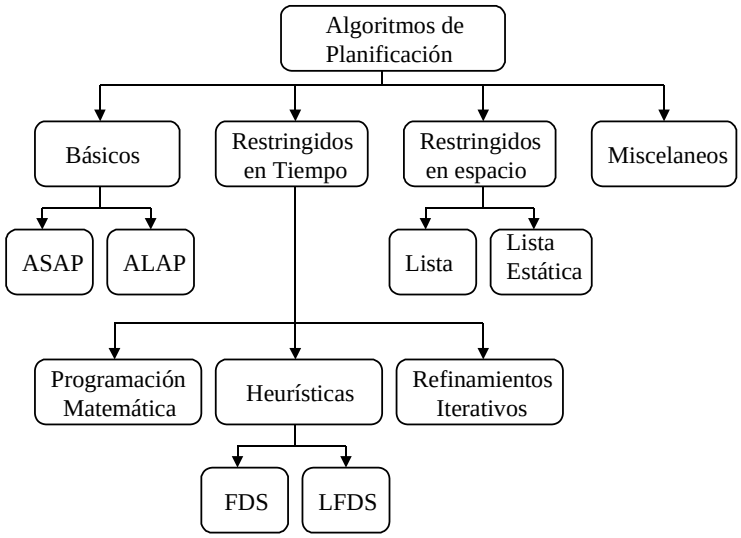

Figura 2.6: Clasificación de los algoritmos de planificación

Uno de estos algoritmos es llamado As Soon As Possible (ASAP). Este algoritmo comienza con los nodos más altos en el diagrama de flujo de datos, es decir, aquellos que no tienen padres, a ellos se les va asignando estados o estratos de tiempo en orden creciente conforme se viaja hacia abajo en el grafo. Esto asegura que un nodo sucesor que representa una instrucción, puede ejecutarse únicamente después de que lo ha hecho su padre. Este algoritmo provee la planificación más rápida posible. Fig. 2.7b.

El segundo algoritmo básico es llamado As Last As Possible (ALAP). Este método es una variante del ASAP, e involucra el posponer bajo ciertas condiciones la ejecución de algún nodo del grafo. Este algoritmo trabaja de la misma manera en que lo hace el algoritmo ASAP, excepto que este comienza en la parte más baja del grafo de flujo de datos y procede hacia arriba. Esta técnica proporciona la planificación más lenta que puede existir. Fig. 2.7c.

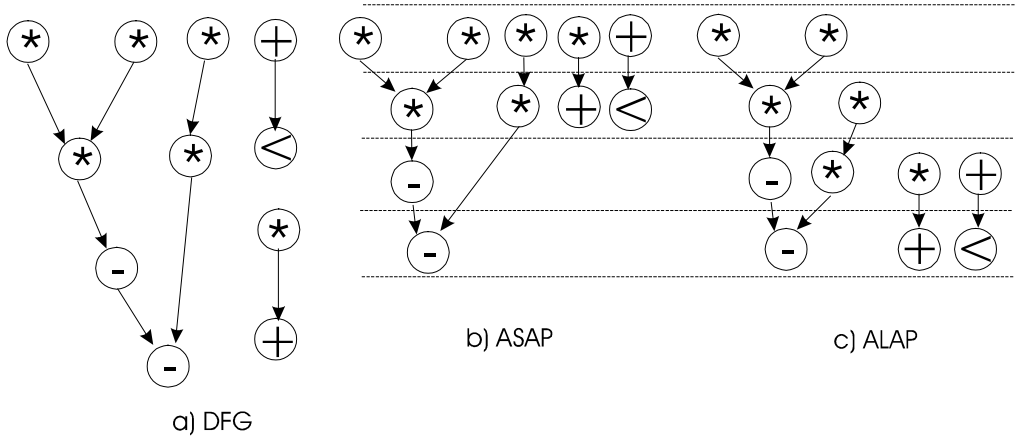

Figura 2.7: Algoritmos básicos de planificación.

2.4.3. Algoritmos de planificación con restricciones en tiempo

Este tipo de planificación es útil para sistemas digitales de tiempo real. Este es un método en el que se asume un número fijo de pasos de control y la planificación debe sujetarse a esa condición. Estos algoritmos utilizan tres diferentes técnicas:

- Programación Matemática.
- Heurísticas
- Refinamientos iterativos.

2.4.4. Algoritmos de planificación con restricciones en espacio

Esta clase de algoritmos se utilizan cuando el número de unidades funcionales que se pueden utilizar están restringidos. El objetivo de estos algoritmos es el de producir un diseño con el mejor desempeño dentro de las restricciones de espacio.

2.4.5. Algoritmos misceláneos

Existen algunas otras soluciones para el problema de planificación que no pueden clasificarse dentro de los grupos anteriores. Dichos algoritmos reciben el nombre de Algoritmos misceláneos.

Después de aplicar alguno de los algoritmos y obtener la planificación de las operaciones a ejecutar, es necesario mapearlas en un espacio físico, determinando las unidades funcionales, de almacenamiento y de interconexión que se utilizarán. Esto corresponde al siguiente paso del proceso de síntesis llamado mapeo.

2.5. Mapeo

La ruta de datos en un modelo FSMD, está compuesta por tres tipos principales de componentes o unidades de transferencia entre registros: unidades funcionales, unidades de almacenamiento y unidades de interconexión (Figura 2.8).

Las unidades funcionales ejecutan las operaciones especificadas en la descripción funcional, tales como los sumadores, ALUs etc. Las unidades de almacenamiento son las RAMs y ROMs que pueden mantener valores de variables durante la ejecución. Las unidades de interconexión, tales como los buses y multiplexores, se encargan de transportar los datos entre las unidades funcionales y de almacenamiento.

El mapeo determina la ruta de datos para una FSMD generada por la planificación, y consta de cuatro tareas dependientes entre sí: Selección de las unidades funcionales y de almacenamiento, Mapeo de unidades funcionales, Mapeo de unidades de almacenamiento y Mapeo de unidades de interconexión.

La ruta de datos se deriva de la transferencia entre registros asignada a cada paso de control. A esta tarea se le llama síntesis de la ruta de datos o mapeo de la ruta de datos [5].

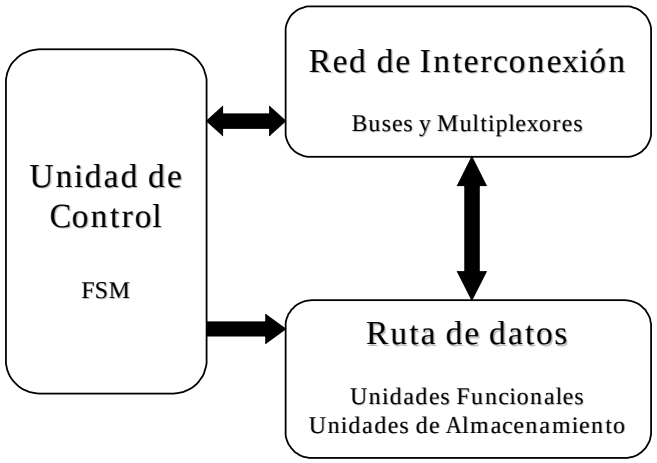

Figura 2.8: Elementos del modelo de máquina de estados finitos con ruta de datos.

2.5.1. Selección de Unidades

La selección de unidades consiste en elegir de una biblioteca de componentes, el número y tipo de unidades funcionales y de almacenamiento que se utilizarán. En esta biblioteca existen varios tipos de unidades con diferentes características como funcionalidad, tamaño, retardo y disipación de potencia. Con algunos componentes es posible implementar varias operaciones diferentes de la descripción de transferencia entre registros, como por ejemplo con las ALUs.

2.5.2. Mapeo de Unidades Funcionales

Una vez que se han seleccionado las unidades funcionales a utilizar, las operaciones de la descripción funcional se mapean a éstas. Si una operación puede ser mapeada en más de una unidad, entonces es necesario aplicar un algoritmo para determinar el mejor mapeo de tal operación.

2.5.3. Mapeo de Unidades de Almacenamiento

El mapeo de constantes, variables y estructuras de datos se hace a unidades de almacenamiento: RAMs, ROMs, registros, etc. Generalmente a las constantes se les asigna espacio de ROM, mientras que a las variables se les mapea en RAM o registros. Si el tiempo de vida de una variable, es decir, el intervalo de tiempo entre su primera asignación de valor y su última utilización, no se traslapa con el de otra variable, entonces estas variables pueden compartir un mismo registro. Si ya se tienen todas las asignaciones de variables a registros, entonces se puede formar un solo conjunto de

registros en el que, si las variables no son utilizadas simultáneamente, es posible utilizar un solo puerto. Un conjunto multipuerto de registros será necesario en el caso contrario.

2.5.4. Mapeo de Unidades de Interconexión

Las transferencias de datos (entre unidades) se mapean a unidades de interconexión. Dos transferencias de datos pueden compartir parte o la totalidad de una ruta de interconexión si no se ejecutan simultáneamente, y de hecho, el objetivo principal de este mapeo es el de maximizar la compartición de las unidades de interconexión, minimizando así el costo.

2.5.5. Métodos de Mapeo

Como se muestra en la Figura 2.9, los métodos de mapeo se pueden clasificar en dos tipos: Los métodos constructivos y los métodos de descomposición.

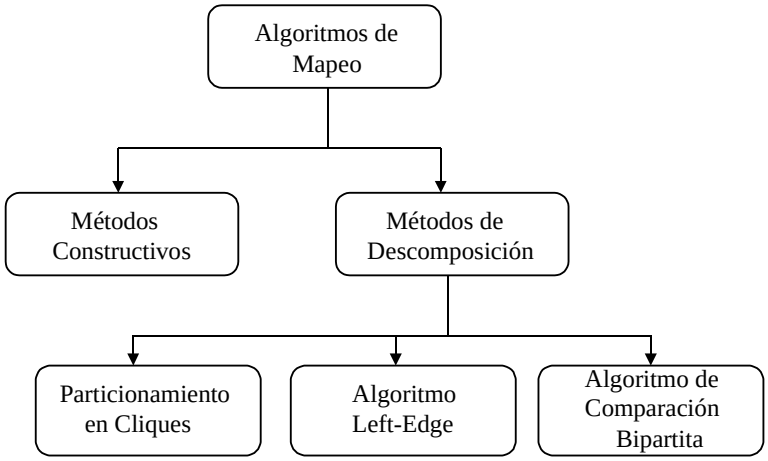

Figura 2.9: Métodos de Mapeo.

Los algoritmos constructivos comienzan con una ruta de datos vacía, y la van construyendo gradualmente agregándole unidades funcionales, de almacenamiento y de interconexión conforme van siendo necesarios.

Para cada operación, tratan de encontrar una unidad funcional en la ruta de datos parcialmente construida, que pueda ser utilizada. Esta unidad funcional debe estar disponible en el paso de control en el que la operación debe ejecutarse. Si existen dos o más unidades funcionales que cumplan con estas condiciones, entonces se utiliza la que incremente en menor grado el costo de interconexión. Por otro lado, si no hay unidades funcionales en la ruta de datos parcialmente construida que puedan utilizarse, se selecciona alguna de la biblioteca de componentes.

De la misma manera, cuando se trata de asignar una variable a una unidad de almacenamiento, se busca en la ruta de datos, un registro en el que el tiempo de vida de las variables que ya le han sido asignadas no se traslape con la nueva variable. Si se encuentra este registro, entonces puede compartirse. Y si existen varios registros que pueden ser compartidos y asignados a esta variable, entonces se elige aquel que represente un menor incremento en el costo de interconexión. Así también, si no hay ningún registro que cumpla con dichas características, deberá agregarse uno nuevo a la ruta de datos con toda la interconexión que esto implica.

Los algoritmos constructivos son los más simples, pero en la práctica sus soluciones pueden estar muy alejadas de la óptima. Por ello, y para mejorar la calidad de los diseños producidos, se han propuesto los métodos de descomposición. En estos métodos el proceso de mapeo se divide en una secuencia de tareas independientes. Cada una de estas tareas se convierte en un problema bien definido de teoría de grafos y son resueltas como tales.

Una diferencia importante entre los dos tipos de algoritmos, es que los constructivos mezclan todas las tareas del mapeo (el mapeo de unidades funcionales, el mapeo de unidades de almacenamiento y el mapeo de unidades de interconexión), mientras que en los métodos de descomposición siempre se termina una tarea antes de comenzar con la otra. De cualquier manera, debido a la interdependencia de las tareas del mapeo, no se garantiza en ningún caso que el resultado sea óptimo, aún si las tareas fueron resueltas de forma óptima.

2.5.6 Generación de Control.

La asignación de operaciones a pasos de control que genera la planificación es la traducción de una descripción funcional a un conjunto de transferencias entre registros que pueden ser descritas por una tabla de estados. La unidad de control se genera a partir de la secuencia de pasos de control

2.5.7. Ejemplo de Mapeo

En el ejemplo de la sección anterior, el resultado de la planificación proporciona las operaciones, el flujo de datos y las transformaciones de los datos.

Con esta información, la tarea de mapeo construye la arquitectura de nivel de transferencia de registros (la ruta de datos) mostrada en la Figura 2.10. La unidad de control, que es una máquina de estados finitos (FSM), se presenta en forma de tabla (tabla 2). Cada renglón representa un estado y en cada uno de ellos se encuentran las entradas, señales de control y transiciones que deben llevarse a cabo.

Es importante tomar en cuenta que se ha considerado que los registros A, B, C, D y E tenían algún valor cargado previamente, es decir, no se toma en cuenta ningún estado en el que se desarrolle la carga de algún valor a los registros, sino que se parte de la idea de que el valor existe ya en ellos.

Como resultado de la tarea de mapeo a partir de la planificación obtenida en la sección anterior, se determinó que era necesaria únicamente una unidad funcional por cada operación a realizar, es decir, solo se empleará un multiplicador y un sumador. Así, para que las unidades funcionales pudieran ser utilizadas con diferentes variables de entrada dependiendo del estado en el que se realiza la operación, fue necesario agregar a la ruta de datos algunos multiplexores.

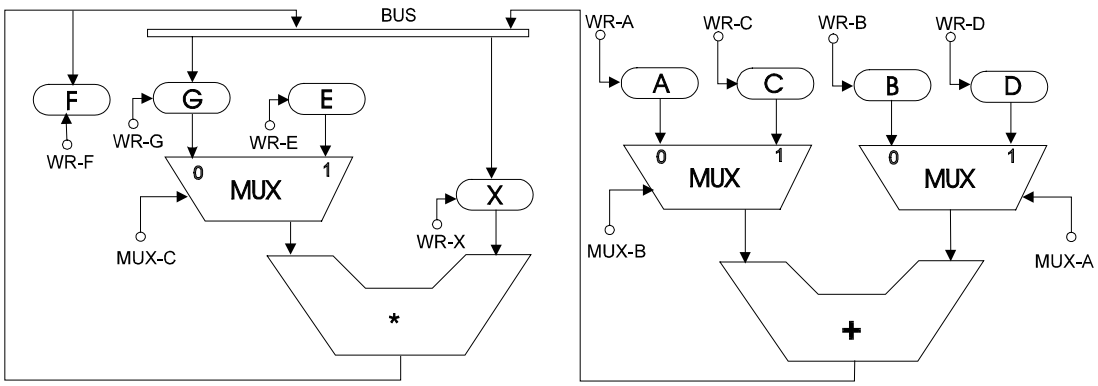

Figura 2.10: Ruta de datos del ejemplo de mapeo.

Estado	Entradas	Señales de control	Salidas	Siguiente Estado
T1	A B C D E	MUX-A = 0 MUX-B = 0 Habilita WR-X	X = (A+B)	T2
T2	E X	MUX-A = 1 MUX-B = 1 MUX-C = 1 Habilita WR-G Habilita WR-F	G = (C+D) F = E*(A+B)	T3
T3	X G	MUX-C = 0 Habilita WR-G	G = (A+B)*(C+D)	----

Tabla 2: Máquina de estados finitos del ejemplo de mapeo.

También fue necesario agregar un registro más al conjunto básico, al cual se le nombra “X”, en el que temporalmente se almacenará el resultado de la operación (A+B), ya que es útil a dos operaciones que se realizan en distintos estados o pasos de control. Este resultado no pudo ser almacenado en algún otro registro existente en la ruta de datos que pudiera compartirse con alguna otra variable debido a su tiempo de vida. No fue así con el resultado de la suma (C+D), la cual pudo compartir un registro con la variable “G”, registro designado a almacenar el valor de una de las dos variables finales del sistema (F y G).

En cuanto a los elementos de interconexión, se utilizó un bus que permite distribuir los resultados entre los registros X y G, resultados provenientes de las dos unidades funcionales existentes logrando que fuera compartido por ambas.

Pueden identificarse en la fig. 2.10, señales de control a los registros y multiplexores, señales que se habilitan o deshabilitan dependiendo de la transición de los estados.

En el estado T1, las señales MUX-A y MUX-B, entradas de selección para el multiplexor, son puestas en cero, con lo que se seleccionan las variables de la izquierda, es decir A y B, para que funjan como entradas a la unidad funcional de adición que, al ser un circuito combinacional, presenta casi inmediatamente (después de un cierto retardo) el resultado obtenido que es finalmente almacenado en el registro X cuando el control habilita la señal WR-X.

En el estado T2, ya con el resultado de la primera suma en el registro X, que ahora es el primer operando en la unidad funcional de multiplicación, se selecciona el contenido del registro E como segundo operando (MUX-C = 1) y se obtiene ahora la multiplicación E*X, con lo cual se ha resuelto el valor para la variable F; así es que se habilita la señal WR-F para almacenarlo. Mientras tanto, se han colocado las señales MUX-A = 1 y MUX-B =1 para realizar la suma C+D, la cual, por medio de la habilitación de WR-G, se almacena temporalmente en el registro G.

En el último estado, T3, se selecciona el valor de G como el segundo operando (puede observarse que el primero siempre es el valor de X) de la operación de multiplicación, por medio de la señal MUX-C = 0, y el resultado es almacenado en el registro G nuevamente por medio de la habilitación de WR-G. Al final de estos estados, se tendrá en sus respectivos registros los valores de las variables G y F.

2.6. Herramienta de síntesis de alto nivel “HLSynth”.

Habiendo descrito algunos de los puntos principales del proceso de síntesis de alto nivel, es posible exponer la idea de una herramienta de software que eficientemente realiza la síntesis de alto nivel de sistemas digitales, proporcionando uno o múltiples diseños al nivel de abstracción de transferencia entre registros.

Las etapas de síntesis que cubre nuestra herramienta, llamada HLSynth, son las descritas en este capítulo: Compilación, Planificación, y Mapeo, incluyendo dentro de este último la generación del control. Además, dado que el diseño resultante es modelado en VHDL, HLSynth incluye un módulo de generación de código en dicho lenguaje (ver Figura 2.11).

Los diseños que genera HLSynth son aplicaciones orientadas a flujo de datos, y la arquitectura del diseño es la de una máquina de estados finitos con ruta de datos.

En cuanto a la descripción funcional, debe decirse que, en principio, en HLSynth se restringe el dominio de aplicación a aquellos sistemas que sean posibles de definir con expresiones lógicas y aritméticas.

HLSynth toma como entrada una expresión aritmética o lógica que describe al sistema, y después de una primera etapa de compilación, genera una representación intermedia que utiliza grafos de flujo de datos. A esta nueva representación es a la que se aplican los algoritmos adecuados de planificación y mapeo, dependiendo de las restricciones que se tengan y objetivos que se persigan, obteniendo una ruta de datos y una unidad de control para un procesador que se comporta de una manera determinada por La descripción inicial.
El procesador, es especificado en VHDL, un lenguaje de descripción de hardware (HDL), de forma que utilizando herramientas actuales como ALTERA o las de XILINX, es posible implementarlo en FPGAs.

Los HDLs tienen dos aplicaciones principales: la documentación y el modelado de un diseño [2]. Además existen simuladores que pueden validar un diseño especificado en tales lenguajes. De entre estos lenguajes, el más empleado es VHDL.

VHDL es el lenguaje de descripción de Hardware para VHSIC (Very High Scale Integrated Circuits), y permite el modelado y simulación de circuitos en todas las etapas del proceso de diseño. VHDL permite realizar síntesis, pruebas de hardware y análisis de tiempos de un circuito. En este lenguaje, un componente del sistema es representado como una entidad que a su vez puede estar asociada con varias arquitecturas alternativas, pudiendo cada arquitectura especificar un dispositivo ya sea funcionalmente o estructuralmente en términos de los componentes de carácter más simple que lo conforman [3].

VHDL es un lenguaje concurrente que permite la descripción de un sistema a diferentes niveles, soporta un manejo concurrente de señales así como las descripciones secuenciales ordinarias. El diseño en VHDL generalmente se hace genérico, para después asignarle bibliotecas específicas. VHDL es un lenguaje fuertemente tipificado, cuyo tipo de dato básico es la señal (signal), una variable que contiene una componente de tiempo.

Dado que HLSynth genera código en VHDL, consideramos que es una herramienta de gran utilidad en el diseño de sistemas digitales por la compatibilidad que (potencialmente) tiene con otras herramientas de diseño de sistemas digitales

2.7. Resumen.

El proceso de síntesis de alto nivel transforma una descripción de un sistema en el dominio funcional (basado en algún lenguaje de alto nivel o un HDL) a una representación en el dominio de transferencia entre registros (que involucre una ruta de datos y un control). Las tareas de la síntesis de alto nivel son: Compilación, Planificación, Mapeo y Generación de control. Estas tareas guardan cierta similitud con la forma en que el ser humano realiza la síntesis. Sin embargo, la capacidad del ser humano está limitada, por lo que surge la idea de automatizar tales tareas de forma que existan herramientas para facilitarlas independientemente de la magnitud y complejidad de los sistemas.

El problema de construir una herramienta muy eficiente para el proceso de síntesis de alto nivel es muy grande si se quiere hacer muy general. Por ello suelen restringirse únicamente a algunos tipos de sistemas o formas de representación funcional. Este último es el caso nuestra herramienta, la cual está restringida a expresiones aritméticas y lógicas como medio de descripción funcional, estableciendo un conjunto muy específico de problemas a los que se puede aplicar de manera eficiente.

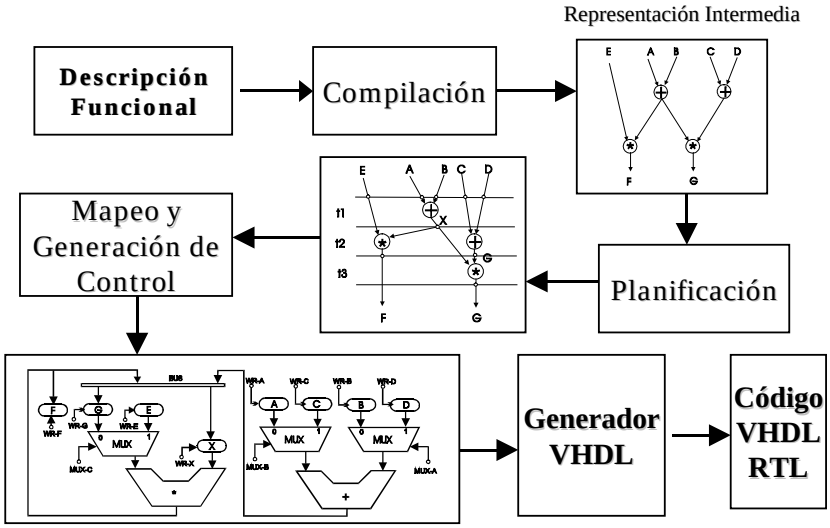

Figura 2.11: Diagrama a bloques de HLSynth.

Capítulo 3

Compilación

3.1 Introducción

La síntesis de alto nivel es el proceso de refinar una descripción funcional, hasta una descripción a nivel de transferencia entre registros (RTL), a través de un conjunto de transformaciones sucesivas [1].

La primera transformación, llamada compilación, consiste en la traducción de la descripción funcional de un diseño a una representación adecuada para realizar las transformaciones siguientes (ver Figura 3.1). La elección de la representación interna, también llamada forma intermedia, constituye una de las principales decisiones cuando se construye una herramienta de síntesis de alto nivel. Aún cuando la representación intermedia no es accesible al usuario de la herramienta, su comprensión puede ayudar a entender mejor el funcionamiento de la misma.

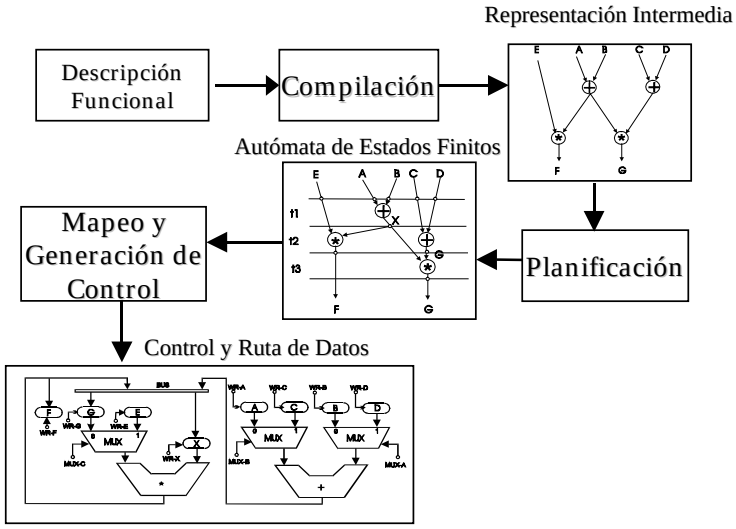

Figura 3.1: El proceso de compilación como primera etapa de la síntesis de alto nivel.

En este capítulo se trata el proceso de compilación para la síntesis de alto nivel. Se presenta primero una revisión de los tres tipos de representaciones intermedias más utilizados en herramientas de síntesis desarrolladas y después el lenguaje de descripción funcional que es utilizado por nuestra herramienta.

3.2 Descripción Funcional

La descripción funcional especifica la función que debe ser desarrollada por el sistema a diseñar (esta descripción puede ser gráfica o textual). Una herramienta de síntesis de alto nivel, actúa como un compilador que mapea una especificación de alto nivel dentro de una arquitectura. Si la descripción funcional cambia, entonces también lo hará la arquitectura. Además en este proceso se deben considerar ciertas restricciones y objetivos que deben satisfacerse, por ejemplo restricciones en espacio o en tiempo, y como objetivos el optimizar la velocidad o el espacio así como el funcionamiento global del sistema.

Aún cuando el lenguaje de descripción funcional tiene gran importancia desde el punto de vista del diseñador, ésta descripción es de menor importancia desde el punto de vista del sintetizador. Esto es debido a que por medio del proceso de compilación, dicha descripción se transforma a una representación interna sobre la cual se aplican los algoritmos de los siguientes procesos de la síntesis de alto nivel.

3.3 Representaciones de diseño

Existen dos clases principales de representaciones intermedias: orientadas a lenguaje y orientadas a la arquitectura. Las primeras, que serán revisadas a detalle más adelante, utilizan una representación basada en grafos que pueden ser de tres tipos: Grafos de Flujo de datos (DFG), Grafos de Control de flujo (CFG) y Grafos de flujo de Datos y de Control de flujo (CDFG).

Las representaciones intermedias orientadas a arquitectura se basan en un modelo de máquina de estados finitos con ruta de datos (FSMD), que es más cercana a la salida que se espera de la síntesis funcional.

Las representaciones que se utilizaron para la construcción de la herramienta de síntesis en esta tesis fueron orientadas a lenguaje. Este tipo de representaciones se presenta a continuación.

3.3.1 Grafos de Flujo de Datos (DFG)

Los grafos de flujo de datos son los más populares en la representación de un programa en la síntesis de alto nivel, entendiendo como programa al conjunto de instrucciones que secuencialmente, o en paralelo, deben ejecutarse para obtener los resultados requeridos.

Aquí los nodos representan los operadores del programa y las aristas representan valores. La función de un nodo es la de generar un nuevo valor a su salida dependiendo de sus entradas. Formalmente, un grafo de flujo de datos esta definido de la siguiente manera [1]:

Un grafo de flujo de datos DFG es un grafo G=(V,E) donde:

(i) $V = \{v_1, \ldots, v_n\}$ *es un conjunto finito cuyos elementos son nodos, y*
(ii) $E \subset V \times V$ *es una relación asimétrica de flujo de datos, cuyos elementos son aristas dirigidas que representan datos.*

Los nodos en un DFG representan operaciones. Una arista dirigida e_{ij} desde $v_i \in V$ hasta $\upsilon_j \in V$ existe, si el dato producido por la operación o_i (representada por v_i) es consumido por la operación o_j (representada por v_j). Entonces v_i es un predecesor inmediato de v_j. De igual manera, v_j es un sucesor inmediato de v_i.

En la Figura 3.2, puede observarse un DFG para el cálculo de:

$$e = (a+c)*(b-d)$$

Este grafo está compuesto de tres nodos: v_1 representa la operación "+", v_2 la operación "*" y v_3 la operación "-". Los datos producidos por v_1 y v_2 son consumidos por v_3, por lo que v_3 es un sucesor inmediato de v_1 y v_2, así mismo, v_1 y v_2 son predecesores inmediatos de v_3.

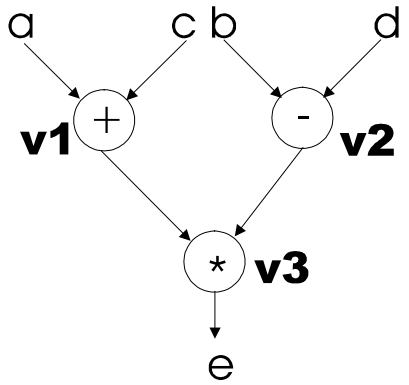

Figura 3.2: Grafo de Flujo de Datos para e = (a+c)*(b-d)

3.3.2 Grafos de Control de Flujo (CFG)

Esta es la representación más adecuada para modelar el diseño del control del sistema digital. Formalmente, esta representación se define de la siguiente manera [1]:

Un grafo de control de flujo CFG es un grafo G=(V,E) donde:

(i) $V = \{v_1, \ldots, v_n\}$ *es un conjunto finito cuyos elementos son nodos, y*
(ii) $E \subset V \times V$ *es una relación de control de flujo, cuyos elementos son aristas dirigidas que representan la secuencia.*

En la Figura 3.3, se muestra el grafo de control de flujo para el siguiente conjunto de instrucciones a realizar (cálculo del máximo común múltiplo):

```
L1: Loop
      x = xi                    (1)
      y = yi                    (2)
L2: While (x ≠ y) Loop  (3)
      If (x<y)                  (4)
            Then y = y-x        (5)
            Else x = x-y        (6)
      End If
    End Loop L2
     ou = x                     (7)
End Loop L1
```

Los números que aparecen del lado derecho de algunas instrucciones son para referir a cada operación en el grafo de la Figura 3.3; así, *v1* representa la ejecución de la instrucción 1.

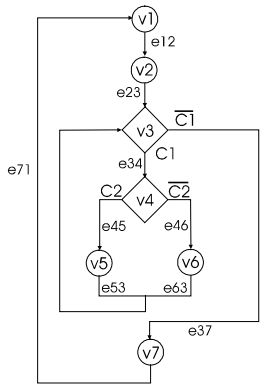

Figura 3.3: Diagrama de Control de Flujo para el problema de MCD

El grafo consta de siete nodos, V = $\{v_1,\ldots,v_7\}$ y ocho aristas E = $\{e_{12}, e_{23}, e_{34}, e_{45}, e_{53}, e_{63}, e_{37}, e_{71}\}$. Puede observarse la existencia de nodos que permiten la bifurcación del control de flujo. Además, debido a la existencia de la arista e_{71}, se infiere que el programa se ejecutará indefinidamente.

Los nodos pueden ser de dos clases en este tipo de grafos:

- Nodos de operación, tales como asignaciones, operaciones aritméticas y lógicas, y llamadas a procedimientos. Estos nodos son representados por un subconjutno Vo de V (Vo = $\{v_1, v_2, v_5, v_6, v_7\}$ en el ejemplo anterior).

- Nodos de decisión, que modelan las construcciones tales como if, Case y saltos condicionales, representados por un subconjunto Vb de V. (Vb = $\{v_3, v_4\}$ en el ejemplo anterior).

De forma que V = Vo ∪ Vb.
Los nodos de operación únicamente pueden tener un sucesor, mientras que los de decisión pueden tener más de uno.

Una arista *eij* representa la relación de precedencia entre dos nodos v_i y v_j.

Los modelos basados en estos grafos, representan bastante bien la inclusión de ciclos anidados, sentencias de sincronización (wait), sentencias de control (if, case) y excepciones (EXIT); además, proveen ciertas facilidades aunque restringidas, para el análisis de flujo de datos y transformaciones.

3.3.3 Grafos de Control de Flujo y Flujo de Datos (CDFG)

Los grafos de Control de flujo y flujo de datos extienden los DFG con nodos de control (If, case, ciclos). Este modelo es adecuado para la representación de aplicaciones orientadas a flujo de datos y también es utilizada por varias herramientas de síntesis orientadas a control de flujo.

Los CDFG utilizan bloques básicos que representan secuencias de operaciones en las cuales no existe ningún control de flujo y para cada bloque básico se hace un grafo de flujo de datos. En este esquema los bloques de control de flujo se mantienen separados de los de flujo de datos, de manera que un bloque básico puede representar un nodo de sentencia de control y un grafo de flujo de datos que a su vez contiene varios nodos de flujo de datos (Figura 3.4).

Cualquiera de las representaciones de diseño descritas puede ser utilizada como representación intermedia.

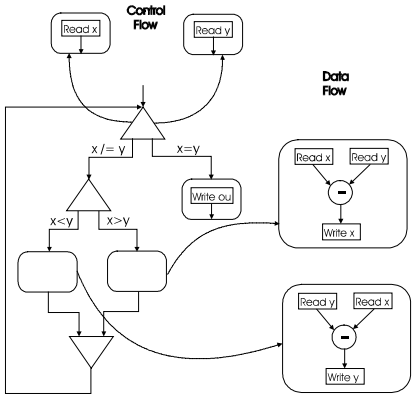

Figura 3.4: Grafo de Control de Flujo y Flujo de Datos para el cálculo del Máximo Común Múltiplo.

3.4 Compilación en HLSynth

Nuestra herramienta de síntesis requiere como entrada un modelo del sistema que se va a sintetizar. Este modelo, debe ser especificado utilizando un lenguaje de descripción funcional.

El lenguaje de descripción funcional de HLSynth permite modelar DFGs de manera sencilla. No es de muy alto nivel ya que requiere cierto trabajo previo por parte del diseñador. Esto será explicado más adelante.

El dominio de aplicación para el que fue construida esta herramienta es el modelado de procesadores de expresiones aritmético-lógicas, y por ello la forma de representación intermedia elegida es un grafo de flujo de datos (DFG), el cual es suficiente para modelar este tipo de expresiones.

El trabajo previo que el diseñador debe realizar es dibujar el DFG que modela el comportamiento del procesador que desea generar y enumerar los nodos que lo constituyen. En la Figura 3.5 se muestra un DFG con los nodos enumerados, el cual resulta de las expresiones aritméticas siguientes:

F = (A+B) * (C+D)
G = F * E

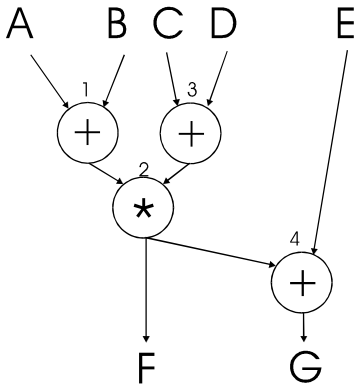

Figura 3.5: DFG con nodos enumerados.

En el DFG de la Figura 3.5 pueden identificarse las variables de entrada, de salida y la dependencia de datos a través del DFG. El lenguaje de descripción funcional de HLSynth permite definir variables de entrada e interconectarlas en el grafo, así como definir nodos de operación y variables de salida.

3.4.1 Definición de variables

Las variables que se pueden definir como entradas en la descripción funcional de un sistema, son aquellas que no se generan como resultado de alguna operación durante la evaluación de la expresión. Las variables de entrada son las que no tienen ningún nodo predecesor en el DFG. En el ejemplo, podemos ver que las variables de este tipo son: A, B, C, D y E. Se asume que estas variables contienen un valor determinado antes de comenzar a evaluar la expresión.

Una sentencia escrita en el lenguaje de descripción funcional para definir una variable de entrada tendrá la siguiente forma:

VAR *nombre_de_variable nodo_final1 , nodo_final2, … , nodo_finalN*

En este formato puede observarse que la primera palabra de la sentencia es "**VAR**", lo cual indica al compilador que se está definiendo una variable. A continuación debe incluirse el nombre de la misma. El nombre de la variable es una cadena alfanumérica que servirá de identificador para ésta. Después se incluyen separados por comas, los números de los nodos a que toman como entrada a esa variable.

Por ejemplo, en la Figura 3.5, puede observarse que la variable C es tomada como entrada por el nodo 3, entonces la definición de esa variable es:

```
VAR  C  3
```

3.4.2 Definición de nodos de operación

Cuando se define una operación, se debe escribir una sentencia con el siguiente formato:

Operación nodo_de_inicio nodo_final1, nodo_final2, … , nodo_finalN

Primero se especifica la operación a ejecutar (+, *, -, min, max, etc.) y después se indica el nodo de origen y los nodos de destino separados por comas.

Por ejemplo, la operación de multiplicación en el nodo 2 de la Figura 3.5, genera un resultado que debe transferirse al nodo 4 y a la variable de salida F, entonces esa operación queda definida como:

```
* 2 4,F
```

Aquí puede observarse una regla más del lenguaje de descripción, y es que las variables de salida no requieren definición, solo se incluyen como cadena alfanumérica en alguno de los destinos de una operación. La descripción funcional para el ejemplo de la Figura 3.5, es la siguiente:

```
VAR A 1
VAR B 1
VAR C 3
VAR D 3
VAR E 4
+ 1 2
+ 3 2
* 2 4,F
+ 4 G
```

El compilador se encargará de crear una estructura de datos que represente el grafo de la descripción. Con base en esta estructura de datos se realizarán todas las transformaciones necesarias para obtener una descripción a nivel RTL de un sistema digital que evalúe la expresión inicial.

3.5 Resumen

El proceso de compilación en HLSynth, no es muy complejo. De hecho, es casi una traducción directa de un DFG representado gráficamente a una estructura de datos que será manipulada por software. Dicha manipulación consiste en realizar todos los refinamientos necesarios sobre ella, y poder generar un sistema digital que evalúe las expresiones.

La estructura de datos que se genera en la etapa de compilación, es la entrada a la siguiente etapa en el proceso de síntesis, la “Planificación”.

Capítulo 4

Planificación

4.1 Introducción.

Un sistema típico de síntesis de alto nivel convierte la descripción funcional de entrada que define a un sistema digital deseado en una representación intermedia en el proceso de compilación y sobre esta representación intermedia, se aplica el proceso de planificación, el cual es tratado en este capítulo.

La planificación puede ser descrita como el proceso de dividir la representación intermedia en estados y pasos de control, de tal forma que esta representación pueda ser sintetizada directamente a una máquina de estados finitos con ruta de datos (FSMD). Esto es conveniente pues, al nivel de transferencia entre registros, las FSMDs son el medio más utilizado para describir un sistema digital. Es decir, la planificación provee un mapeo temporal de la representación dada [6].

En una FSMD puede identificarse el control como una máquina de estados finitos que, como todo autómata, tiene un conjunto de estados y de transiciones entre ellos, y un conjunto de acciones asociadas con cada transición. También puede identificarse la ruta de datos (datapath), que comprende la estructura formada por unidades funcionales y de almacenamiento que conforman al sistema. Cada paso de control corresponde a un estado de la máquina de estados finitos en el modelo FSMD [7].

Un paso de control corresponde a una transición en una FSM. Este puede incluir varias operaciones ejecutándose en paralelo, asumiendo, por supuesto, que existen los suficientes recursos (unidades funcionales) para ejecutarlas [1]. Un algoritmo de planificación puede tomar en cuenta restricciones de tiempo o restricciones de espacio.

Dentro de un paso de control, se requiere una unidad funcional para ejecutar cada operación asignada a ese paso. Por ello, el número total de unidades funcionales que se requieren en un paso de control corresponde directamente al número de operaciones planificadas en él. Así es que si se planifican más operaciones dentro de cada paso de control, se requerirán más unidades funcionales, con lo que se reduce el número de pasos de control en que se ejecutan el total de las operaciones. Por otro lado, si se planifican menos operaciones por paso de control, entonces se incrementará el número de pasos de control necesarios para la ejecución total.

Una planificación para un DFG determinado (producto de la compilación), será adecuada si cumple con las restricciones de ejecución. Pero también es deseable encontrar una planificación que sea la óptima (por ejemplo, aquella con la menor cantidad de unidades funcionales). Es decir, que aún dentro del proceso de planificación existen niveles de optimización basados en ciertos objetivos y metas de diseño.

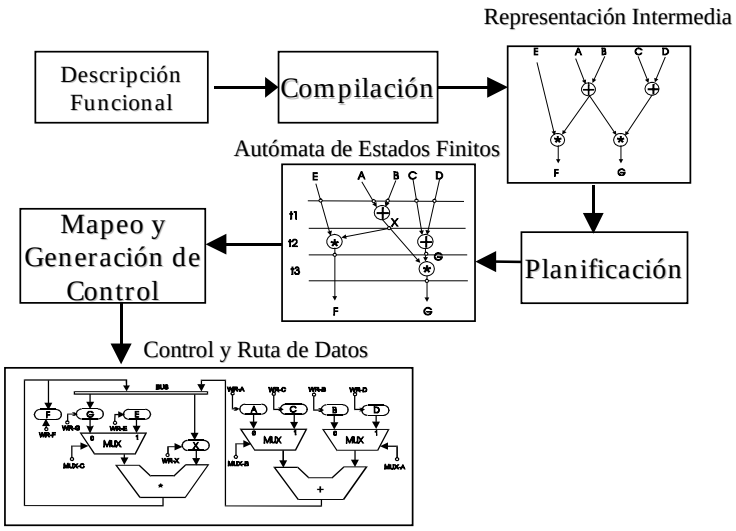

Figura 4.1: El proceso de planificación como segunda etapa en la Síntesis de Alto Nivel.

4.2 Definición del problema de planificación

El problema de planificación puede ser dividido en cuatro problemas básicos. Una definición de estos problemas es dada por Walker y Chaudhri [12]. En esta tesis se redefinirá el problema de la manera siguiente.

Retomemos la definición de un grafo de flujo de datos presentada en el capítulo 3:

Un grafo de flujo de datos DFG es un grafo G=(V,E) donde:

V = $\{v_1, \ldots, v_n\}$ *es un conjunto finito cuyos elementos son nodos, y*
$E \subset V \times V$ *es una relación asimétrica de flujo de datos, cuyos elementos son aristas dirigidas que representan datos.*

Los nodos en el DFG representan operaciones. Una arista dirigida e_{ij} desde $v_i \in \mathbf{V}$ hasta $\upsilon_j \in \mathbf{V}$ existe, si el dato producido por la operación representada por v_i es consumido por la operación representada por v_j. Entonces v_i es un predecesor inmediato de v_j. De igual manera, v_j es un sucesor inmediato de v_i.

La planificación puede describirse como un mapeo de los nodos de un DFG a un número finito de pasos de control o estados de la FSM. Un mismo paso de control puede asignarse a varios nodos, pero un nodo solo puede tener asignado un paso de control. Lo anterior lo podemos representar como:

Sea **C** = {0,1,2,…,*n*-1} un conjunto finito de pasos de control.
Sea **P**: $\mathbf{V} \rightarrow \mathbf{C}$ la planificación del DFG.

Si $P(v_i) = k$; $k \in \mathbf{C}$, entonces k es el paso de control asignado al nodo v_i.

En la planificación de un DFG, debe respetarse la dependencia de datos entre operaciones. Por ello, debemos especificar lo siguiente:

Sea v_s un nodo sucesor inmediato del nodo v_p.
Entonces **P** es una planificación válida si $P(v_s) > P(v_p) \ \forall \ v_s, v_p \in$ **V.**

Es claro que podrían existir muchas planificaciones posibles para un solo DFG. A continuación se presentan algunas condiciones que se agregan a todas esas planificaciones para que sean válidas.

4.2.1 Planificación con Restricciones en tiempo

La planificación sin restricciones contempla todos los elementos básicos del problema de planificación, pero las planificaciones pueden tener que satisfacer algunos objetivos de diseño que requieren restricciones adicionales. Por ejemplo, podría limitarse el tiempo de ejecución del diseño restringiendo el número de pasos de control de toda la planificación. Este proceso adiciona una restricción de tiempo que la planificación del diseño final debe satisfacer.

Entonces el problema de la planificación con restricciones en tiempo se define como sigue:

Sea **P**: **V** $\rightarrow$ **C** la planificación válida para un DFG G, y sea τ la restricción en el tiempo.

Si $k_{\max} = \max_{v_i \in V}\{P(v_i)\}$
Entonces **P** satisface las restricciones en tiempo si $k_{\max} \leq \tau$

4.2.1.1 Planificación óptima con restricciones en tiempo

Una planificación **P***:**V** $\rightarrow$**C** es óptima si para cualquier otra planificación **P'**: **V** $\rightarrow$ **C** :

$k^*_{\max} = \max_{v_i \in V}\{P^*(v_i)\}$

$k'_{\max} = \max_{v_i \in V}\{P'(v_i)\}$

$k^*_{\max} \leq k'_{\max} \leq \tau$

4.2.2 Planificación con Restricciones en Recursos.

Otro conjunto de restricciones que comúnmente se agrega al problema de planificación y que reflejan el objetivo de limitar el espacio en el que se implementa, son las restricciones del número de unidades funcionales de cada tipo. Agregando estas restricciones al problema de la planificación sin restricciones, puede definirse el problema de la planificación con restricciones en recursos de la siguiente manera:

Supongamos que todas las operaciones son del mismo tipo, entonces, las unidades funcionales serían todas similares.

Sea **P**: **V** $\rightarrow$ **C** una planificación válida para un DFG G, y sea ρ el número máximo de unidades funcionales.

Si r_k = card ({ $v_j \in$ **V** | P(v_j) = k }) $\forall\ k \in$ **C**
Entonces, la planificación P satisface las restricciones en recursos si $r_k \leq \rho$, $\forall\ k \in$ **C**

4.2.2.1 Planificación óptima con restricciones en recursos

Una planificación **P***:**V** $\rightarrow$**C** es óptima si para cualquier otra planificación **P'**:**V** $\rightarrow$**C**:
V^*_k = { $v_j \in$ **V** | $P^*(v_j) = k$ } $\forall\ k \in$ **C**
V'_k = { $v_j \in$ **V** | $P'(v_j) = k$ } $\forall\ k \in$ **C**
$\max_{k \in C}\{card(V_k^*)\} \leq \max_{k \in C}\{card(V'_k)\} \leq \rho$

4.2.3 Planificación con Restricciones en Tiempo y Recursos.

Los problemas de planificación con restricciones en tiempo y con restricciones en recursos pueden combinarse, conformando un nuevo problema en el que se restringe tanto el número de estados total de la planificación, como el número de unidades funcionales de cada tipo que pueden ser utilizadas simultáneamente en el diseño. El problema queda definido de la siguiente manera:

Asumiendo que todas las operaciones son del mismo tipo, entonces las unidades funcionales serían todas similares.

Sea **P**: **V** $\rightarrow$ **C** una planificación válida para un DFG G.
Sea τ la restricción el tiempo y sea ρ el número máximo de unidades funcionales.

Si $k_{\max} = \max_{v_i \in V}\{P(v_i)\}$ y,
Si r_k = card ({ $v_j \in$ **V** | P(v_j) = k }) $\forall\ k \in$ **C**

Entonces, la planificación **P** satisface las restricciones en tiempo y recursos si: $k_{\max} \leq \tau \wedge r_k \leq \rho$

4.2.3.1 Planificación óptima con restricciones en tiempo y recursos

Una planificación **P***:**V** $\rightarrow$**C,** es óptima si para cualquier otra planificación **P'**:**V** $\rightarrow$**C** se cumple lo siguiente:

$$k^*_{\max} = \max_{v_i \in V}\{P^*(v_i)\}$$

$V^*_k = \{\ v_j \in \mathbf{V} \mid P^*(v_j) = k\ \}\ \forall\ k \in \mathbf{C}.$

$$k'_{\max} = \max_{v_i \in V}\{P'(v_i)\}$$

$V'_k = \{\ v_j \in \mathbf{V} \mid P'(v_j) = k\ \}\ \forall\ k \in \mathbf{C}.$

$$k^*_{\max} \le k'_{\max} \le \tau \ \wedge\ \max_{k \in C}\{card(V_k^*)\} \le \max_{k \in C}\{card(V'_k)\} \le \rho$$

4.3 Algoritmos de Planificación.

Los algoritmos de planificación pueden clasificarse como se muestra en la Figura 4.2. En este documento se describen tres de los algoritmos de planificación más comunes en la síntesis de alto nivel, estos son:

- As Soon As Possible
- As Last As Possible
- Force Directed Scheduling

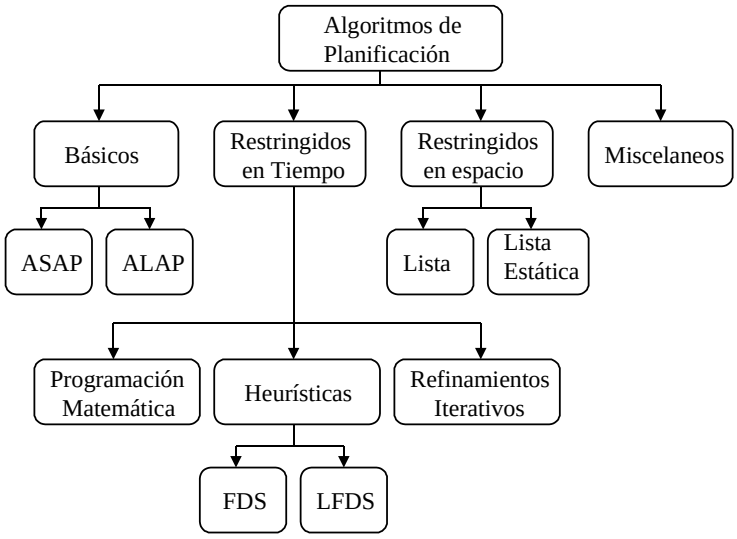

Figura 4.2: Clasificación de los algoritmos de planificación.

Estos tres algoritmos son constructivos y heurísticos, es decir, iterativamente seleccionan y planifican una operación a la vez en el paso de control apropiado. Estos algoritmos realizan una serie de decisiones locales, seleccionando en cada punto el par mejor operación – mejor paso de control. La diferencia entre ellos es la forma en que eligen la siguiente operación a planificar y en como determinan en que paso de control planificarán cada operación.

La solución que proporciona este tipo de algoritmos puede no ser la óptima global, pero se obtiene rápidamente y, generalmente, es lo suficientemente adecuada para una aplicación práctica.

4.3.1 Algoritmo As Soon As Possible (ASAP)

El algoritmo de planificación más simple es el ASAP, y es utilizado en sistemas de síntesis de alto nivel tales como el Bridge System y CMU-DA System [13].

Este algoritmo comienza con los nodos más altos en el DFG, es decir, aquellos que no tienen padres. A estos nodos se les va asignando estados o estratos de tiempo en orden creciente conforme se viaja hacia abajo en el DFG. Esto asegura que un nodo sucesor que representa una instrucción, puede ejecutarse únicamente después de que lo ha hecho su padre. Este algoritmo provee la planificación más rápida posible [7].

El pseudocódigo para este algoritmo se presenta en la Figura 4.3

```
For each node υi ∈ V do
     If Pred υi = ∅ then
          Ei  = 1;
          V = V – {υi};
     Else
          Ei = 0;
     Endif
Endfor

  While V ≠∅ do
     For each node υi ∈ V do
          If ALL_NODES_SCHED (Pred υi, E) then
               Ei = MAX (Pred υi, E) +1;
               V = V – { υi};
          Endif
     Endfor
  Endwhile
```

Figura 4.3: Pseudocódigo de algoritmo ASAP

El algoritmo ASAP, asigna una etiqueta E_i a cada nodo v_i del grafo, que indica el paso de control en que se planificará. Como ya se mencionó, se buscará planificar a cada operación lo más pronto posible, utilizando el primer paso de control disponible para ello, S_{Ei}, respetando las restricciones de precedencia.

En el pseudocódigo, $Pred_{vi}$ denota todos los nodos en el grafo que son predecesores inmediatos del nodo v_i. La función ALL_NODES_SCHED ($Pred_{vi}$, E) regresa un valor verdadero si todos los nodos del conjunto $Pred_{vi}$ han sido planificados, es decir, tienen una etiqueta diferente de cero.
La función MAX ($Pred_{vi}$, E) regresa el índice del nodo perteneciente al conjunto $Pred_{vi}$, con el máximo valor de E.

El primer ciclo (for), inicializa la etiqueta de todos los nodos en el grafo, asignando los nodos que no tienen ningún predecesor al primer paso de control, y a todos los demás nodos, el paso de control cero, es decir, aún no se planifican.

Después de la inicialización comienzan las iteraciones (ciclo while), en las que se evalúa para cada nodo la existencia de predecesores sin planificar. Es posible planificar un nodo si todos sus predecesores han sido planificados (si el paso de control al que han sido asignados es distinto de cero, es posible).

Para determinar el paso de control al que será asignado un nodo, es necesario conocer la etiqueta E del predecesor planificado mas tarde, es decir, se debe encontrar el mayor valor de E de todos los nodos en el conjunto $Pred_{vi}$. Una vez que se cuenta con el paso de control en que se planificó el predecesor más tardío, basta con incrementarlo en uno y ese será el paso de control que corresponde al nodo siendo planificado. El resultado de una planificación de este tipo puede observarse en la Figura 4.4b.

4.3.2 Algoritmo As Late As Possible (ALAP)

El algoritmo ALAP es una variante del ASAP, e involucra el posponer bajo ciertas condiciones la ejecución de algún nodo del DFG. Este algoritmo trabaja en la misma manera que lo hace el algoritmo ASAP, excepto que este comienza en la parte más baja del DFG y procede hacia arriba. Esta técnica proporciona la planificación más lenta que puede existir. El algoritmo ALAP es utilizado en sistemas tales como IIT Delhi's Synthesis System, además de ser parte importante en el cálculo de la movilidad de operaciones junto con el ASAP en otros sistemas, todos ellos descritos en [13].

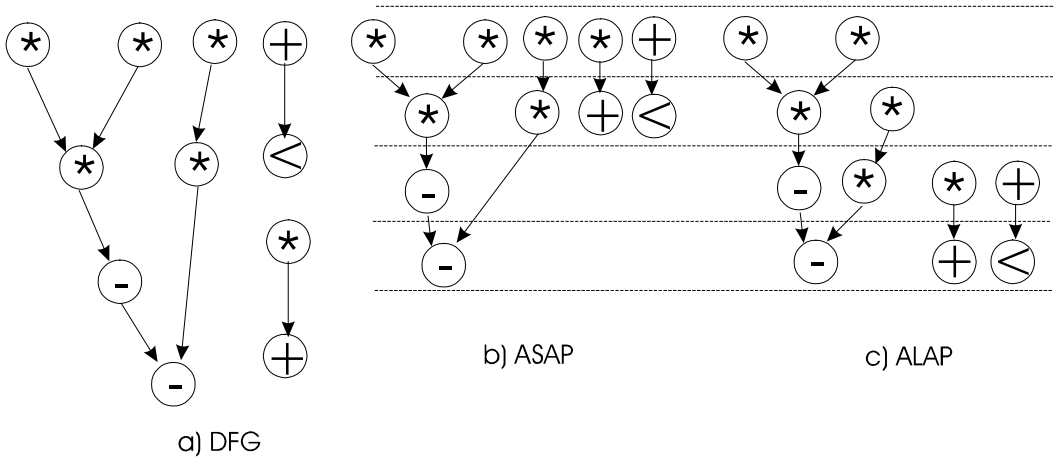

Figura 4.4: a)DFG a planificar, b)Planificación ASAP, c)Planificación ALAP.

El pseudocódigo de este algoritmo de planificación se sugiere en [5] de la manera en que se presenta en la Figura 4.5:

```
For each node υi ∈ V do
     If Succ υi = ∅ then
          Li  = T;
          V = V – {υi};
     Else
          Li = 0;
     Endif
Endfor

While V ≠ ∅ do
     For each node υi ∈ V do
          If ALL_NODES_SCHED (Succ υi, L) then
          Ei = MIN (Succ υi, L) -1;
          V = V – {υi};
          Endif
     Endfor
Endwhile
```

Figura 4.5: Pseudocódigo del algoritmo ALAP

El paso de control que asigna este algoritmo para cada nodo, es el último en el cual posiblemente pueda ser planificado. El algoritmo consiste en que, dada una restricción de máximo **T** pasos de control, se determina el último posible paso de control en el cual una operación debe iniciar su ejecución. Este algoritmo asigna una etiqueta L_i a cada nodo v_i del DFG, que corresponderá al último paso de control S_{Li} en que puede planificar la operación o_i.

La función ALL_NODES_SCHED (*Succ* $_{vi}$, *L*) tiene una pequeña variante con respecto a la función del mismo nombre en el algoritmo ASAP, y es que ésta regresa un valor verdadero si todos los nodos denotados por el conjunto $Succ_{vi}$ de todos los nodos sucesores del nodo vi, han sido ya planificados.

El primer ciclo (ciclo “for”) inicializa las etiquetas para todos los nodos en el grafo, asignando el último paso de control T, a todos aquellos que no tienen sucesores, es decir, son los nodos terminales del grafo. A los demás nodos se les etiqueta con el paso de control cero, para indicar que aún no han sido planificados.

En las iteraciones siguientes se verifica, para cada nodo, que sus sucesores ya hayan sido planificados, y de no ser así, se pospone la planificación de este nodo. Si efectivamente ya fueron planificados todos los nodos sucesores de v_i, entonces v_i será asignado a un paso de control inmediato anterior al del sucesor asignado a un paso de control mas temprano. Este valor se obtiene con la función ***MIN (Succ*** $_{vi}$***, L)***, que regresa el índice del paso de control del nodo con el mínimo valor de **L** del conjunto de nodos sucesores. Si a este valor se le hace un decremento en uno, se obtiene el paso de

control al que deberá ser asignado el nodo siendo planificado. Un ejemplo de este tipo de planificación es mostrado en la Figura 4.4c, correspondiente a la planificación del grafo en la Figura 4.4a.

4.3.3 Algoritmo Force Directed Scheduling (FDS)

El algoritmo Force Directed Scheduling [10][11] fue originalmente desarrollado como parte del sistema HAL [13] de la universidad de Carleton. Este algoritmo es de tipo constructivo y resuelve el problema de planificación con restricciones en tiempo por medio de la distribución uniforme de las operaciones de cada tipo en una planificación con pasos de control restringidos.

El balanceo de las operaciones de esta manera resulta en una alta utilización de unidades funcionales, minimizando el número de ellas para cada tipo de operación. Este algoritmo reduce no solo el número de unidades funcionales, también lo hace con el número de registros y buses que serán requeridos posteriormente en la etapa de mapeo dentro de la síntesis de alto nivel.

La estrategia del algoritmo es colocar operaciones similares en diferentes pasos de control, balanceando la concurrencia de las operaciones asignadas a las unidades funcionales sin incrementar el tiempo total de ejecución. Este balanceo se realiza en tres pasos: Determinar el rango de movilidad de cada operación, calcular la distribución de las operaciones en cada paso de control, y calcular la "*fuerza*" (descrita más adelante) asociada con cada asignación en la planificación.

Por ser este algoritmo uno de los más utilizados dentro de las herramientas actuales de síntesis de alto nivel, será explicado con mayor detalle y en forma de tutorial a partir del grafo de la Figura 4.4a, que se asumirá como la representación intermedia generada después de la etapa de compilación.

4.3.3.1 Rango de Movilidad

El rango de movilidad de una operación está dado por el número de pasos de control en que puede ser planificado, desde el paso de control más temprano hasta el más tardío considerando las restricciones de dependencia de datos.

Para calcular el rango de movilidad es necesario aplicar a la representación inicial los dos algoritmos básicos vistos hasta ahora (ASAP y ALAP). La aplicación del algoritmo ASAP nos provee, para cada operación, el mínimo paso de control S_{Ei} en el cual es posible planificarla, mientras que el algoritmo ALAP calcula el último paso de control S_{Li} hasta donde es posible posponerlo. Entonces, el rango de movilidad está compuesto por todos los pasos de control intermedios entre S_{Ei} y S_{Li} incluyendo estos dos.

Para el ejemplo de la Figura 4.4, el resultado de aplicar los algoritmos ASAP y ALAP se presenta en las Figuras 4.4b y 4.4c respectivamente. Para este ejemplo obtenemos el diagrama de los rangos de movilidad mostrado en la Figura 4.6.

Los rangos de movilidad resultantes se presentan como rectángulos cuyo ancho esta determinado por la probabilidad (suponiendo una distribución uniforme) de que la operación sea planificada en alguno de los pasos de control dentro del rango de movilidad.

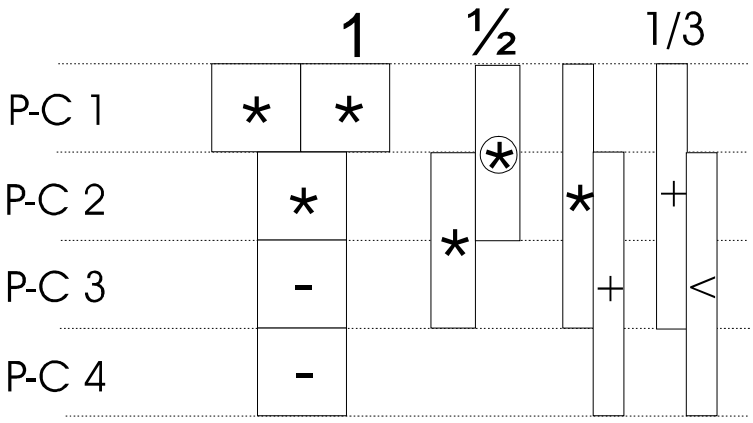

Figura 4.6: Gráfica de rangos de movilidad

Entonces, la probabilidad $P_{i,j}$ de planificar una operación o_i en un paso de control S_j, es:

$$P_{i,j} = 1/(ALAP_i - ASAP_i+1)$$

En la Figura 4.6, podemos ver que algunas operaciones que se encuentran en un cuadrado con probabilidad 1. Estas operaciones son las que forman parte de la ruta crítica. La ruta crítica es el subgrafo en el que todas las operaciones tienen una probabilidad de uno en el paso de control en que se asignan. Esto significa que el paso de control asignado por los algoritmos ASAP y ALAP es el mismo, de manera que inherentemente quedan planificados en ese paso de control.

Los rectángulos con ancho ½ muestran las operaciones que pueden ser planificadas en dos posibles pasos de control. Las de ancho 1/3 pueden ser planificadas en tres posibles pasos de control, etc. Pero debe considerarse que, en cualquiera de estos últimos casos, el asignar una operación a un paso de control específico puede modificar el rango de movilidad de otras operaciones debido a las restricciones de dependencia.

4.3.3.2 Cálculo de distribución de operaciones

Para cada tipo de operación, debe calcularse la distribución de las operaciones del mismo tipo en todos los pasos de control. Este cálculo consiste en sumar, para cada paso de control, las probabilidades que tiene cada operación del mismo tipo de ser planificada en el paso de control respectivo.

El resultado de los cálculos de la distribución indica la concurrencia de operaciones similares y puede ser representada en forma de gráfica de barras. La distribución es calculada para cada operación *Opn* en cada paso de control *i* de la siguiente manera:

$$DG(i) = \sum_{Opn\_Type} prob(Opn,i) \qquad (1)$$

La suma se realiza sobre todas las operaciones del mismo tipo en el mismo paso de control. Para el ejemplo que se sigue, los valores DG(i) para cada operación involucrada en el grafo se presentan en la tabla 4.1.

	*	-	+	<
DG(1)	2.833	0	0.333	0
DG(2)	2.333	0	0.666	0.333
DG(3)	0.833	1	0.666	0.333
DG(4)	0	1	0.333	0.333

Tabla 4.1: Distribución de todas las operaciones en cada paso de control.

4.3.3.3 Cálculo de la Fuerza

Cada operación tiene una fuerza asociada con cada paso de control de su rango de movilidad. La fuerza es una cantidad que refleja el efecto que produce la asignación de una operación a un paso de control específico sobre la concurrencia global de las operaciones del mismo tipo. La cantidad es positiva si la asignación causa un incremento en la concurrencia sobre esa operación, y es negativa si la disminuye.

La fuerza se calcula para cada operación, planificándola en cada uno de los pasos de control dentro de su rango de movilidad y, para cada asignación tentativa reduciendo el rango de movilidad a únicamente aquel paso de control i al que se ha asignado. Esto causa un incremento en 1-DG(i) en la distribución de dicha operación en el paso de control elegido. Así mismo, se origina una reducción de la distribución en DG(i) en todos los pasos de control contemplados en el rango de movilidad de dicha operación. Entonces, la fuerza se calcula con la ecuación 2 [11]:

$$Force(i) = (DG(i) + x(i)/3) * x(i) \qquad (2)$$

Donde x(i) es el cambio en la distribución DG(i) para el paso de control i. La fuerza total será la suma de todas las fuerzas originadas por la asignación tentativa de la operación en el paso de control i en los pasos de control contemplados en su rango de movilidad. Para nuestro ejemplo, se planificará la operación encerrada en un círculo en la Figura 4.6, obteniendo lo siguiente:

Cálculo de la fuerza para la planificación tentativa de la operación de multiplicación en el paso de control 1:

$$Total\_Force(1) = Force(1) + Force(2)$$
$$Total\_Force(1) = ((DG(1) * x(1) / 3) * x(1)) + ((DG(2) + x(2) / 3) * x(2))$$
$$Total\_Force(1) = ((2.833 + 0.5 / 3) * 0.5) + ((2.333 - 0.5 / 3) * (-0.5))$$
$$Total\_Force(1) = 0.4166$$

Cálculo de la fuerza para la planificación tentativa de la misma operación en el paso de control 2:

$$Total\_Force(2) = Force(1) + Force(2)$$
$$Total\_Force(2) = ((DG(1) * x(1) / 3) * x(1)) + ((DG(2) + x(2) / 3) * x(2))$$
$$Total\_Force(2) = ((2.833 - 0.5 / 3) * (-0.5)) + ((2.333 + 0.5 / 3) * 0.5)$$
$$Total\_Force(2) = -0.0833$$

El resultado mayor de las dos probabilidades anteriores indica el paso de control en el que la concurrencia sobre el tipo de operación a planificar es mayor, por lo que el planificar la operación en ese paso de control generaría un efecto adverso sobre la distribución global. Así es que se selecciona el paso de control en el que la fuerza total es menor, balanceando dicha concurrencia.

En nuestro ejemplo, la operación de multiplicación que está siendo planificada, será asignada al paso de control 2. Al planificar esta operación en el paso de control 2, automáticamente se planifica la operación siguiente (multiplicación), ya que a pesar de tener un rango de movilidad de dos pasos de control, existe dependencia de datos con la operación anterior, así es que su rango de movilidad se reduce a un paso de control.

El pseudocódigo para este algoritmo se muestra en la Figura 4.7. El algoritmo se repetirá hasta que todas las operaciones hayan sido planificadas. El primer paso consiste en determinar los rangos de movilidad, utilizando las planificaciones ALAP y ASAP para determinar el paso de control mínimo y el máximo en el que se puede planificar cada operación.

Con el rango de movilidad, se calcula la distribución de cada operación para cada paso de control y con base en esto. Se calculan las fuerzas totales generadas al planificar cada operación en cada paso de control, eligiendo para su planificación aquel paso de control en que las fuerzas totales son menores.

Una vez que se planifica una operación a un paso de control, se ajustan los rangos de movilidad de otras operaciones, ya que pueden cambiar al planificar una operación.

Al terminar de aplicar este algoritmo al ejemplo, se obtiene la planificación mostrada en la Figura 4.8.

Repetir hasta (todas las operaciones han sido planificadas);

1:Evaluar movilidad
Planificación ALAP
Planificación ASAP
2:Recalcular la distribución.
3:Calcular las fuerza totales para cada paso de control
4:Planificar las operaciones con la menor fuerza
5:Ajustar la movilidad

Fin de repetición.

Figura 4.7: Pseudocódigo del algoritmo FDS

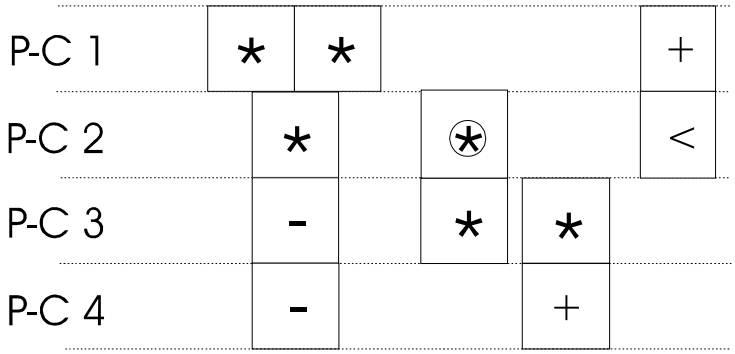

Figura 4.8: Planificación con FDS.

	*	-	+	<
DG(1)	2	0	1	0
DG(2)	2	0	0	1
DG(3)	2	1	0	0
DG(4)	0	1	1	0

Tabla 4.2: Distribución final de las operaciones para el ejemplo.

4.4 Planificación en HLSynth

En nuestra herramienta de síntesis el proceso de planificación se ha implementado de la siguiente manera:

Se parte de la representación intermedia generada en la etapa anterior (compilación). Esta representación intermedia es un DFG que representa todas las operaciones que el sistema digital debe realizar, incluyendo la dependencia de datos entre operaciones. Los procesos siguientes a la compilación trabajan con la representación intermedia, la cual es más fácil de manipular (La descripción funcional del sistema es vista solo por la compilación).

Al DFG obtenido se le aplican los algoritmos de planificación para dividirlo en estratos de tiempo, o pasos de control, en los que se realizarán cada una de las operaciones. La planificación respeta la dependencia de datos y algunas otras restricciones, y además aprovecha el paralelismo con aquellas operaciones que no tengan relación alguna de dependencia.

Los algoritmos de planificación que han sido implementados en HLSynth, son los descritos en este capítulo (Figura 4.9), es decir:

- Algoritmo ASAP
- Algoritmo ALAP
- Algoritmo FDS

Los dos primeros algoritmos fueron implementados en base al pseudocódigo presentado en este capítulo, y pueden ser utilizados como métodos de planificación, aunque fueron implementados como apoyo al tercer algoritmo, FDS, que los utiliza para el cálculo del rango de movilidad.

El algoritmo FDS, implementado también basándose en el pseudocódigo presentado en este capítulo, produce una planificación adecuada para cualquier sistema digital, tratando de hacer un balanceo de las unidades funcionales del mismo tipo, y de las unidades de interconexión y almacenamiento. El cálculo de los rangos de movilidad, se hace con ayuda de los algoritmo ASAP y ALAP para obtener el paso de control mas temprano y el mas tardío concernientes a una operación. El cálculo de la distribución de las operaciones y el cálculo de la fuerza son hechas utilizando las ecuaciones también descritas en este documento.

Como ya se mencionó, debido al carácter constructivo de estos algoritmos, el resultado obtenido muy posiblemente no sea el óptimo, pero genera una planificación adecuada en poco tiempo para cualquier sistema digital representable en DFGs.

4.5. Resumen

El proceso de planificación de la síntesis de alto nivel, es un paso determinante para la obtención de diseños estructurales que cumplan con los requerimientos de su descripción funcional, y que además satisfagan, de la mejor manera, las restricciones

que para su implantación pudieran existir, tales como tiempo restringido, espacio restringido o ambos. En este capítulo se abordó la definición formal del problema de planificación para cada uno de sus casos:

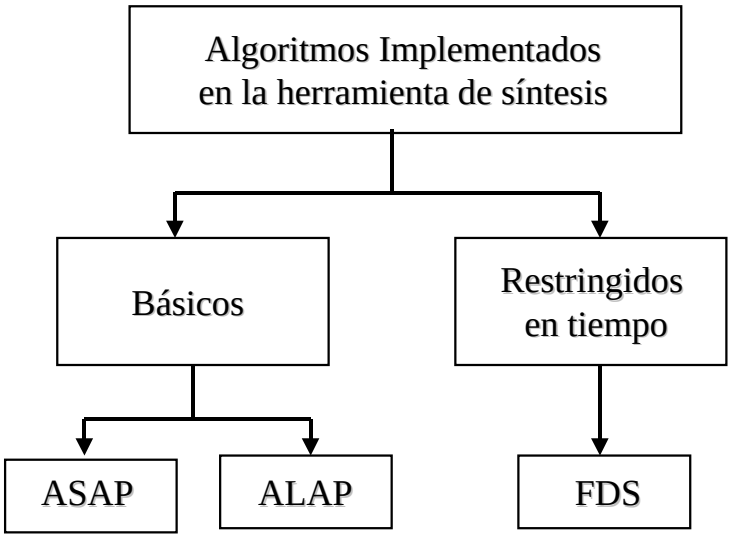

Figura 4.9: Algoritmos de planificación implementados en HLSynth.

- Planificación sin restricciones
- Planificación con restricciones de tiempo
- Planificación con restricciones de espacio
- Planificación con restricciones de espacio y de tiempo

También se trataron las soluciones actuales al problema de la planificación, explicando brevemente el funcionamiento de los algoritmos básicos ASAP y ALAP, para después describir a detalle un algoritmo que es ampliamente utilizado en los sistemas de síntesis funcional que han sido desarrollados [13]. Este algoritmo, llamado “Force Directed Scheduling”, es el algoritmo en el que se basa la planificación de HLSynth.

Capítulo 5:

Mapeo

5.1. Introducción

En este capítulo se aborda la etapa de mapeo del proceso de síntesis de alto nivel (Figura 5.1). La entrada a esta etapa es el grafo de flujo de datos planificado, producido en la etapa de planificación, y la salida será un diseño, al nivel de transferencia entre registros, que contemple una unidad de control y una ruta de datos, capaz de realizar la función del sistema.

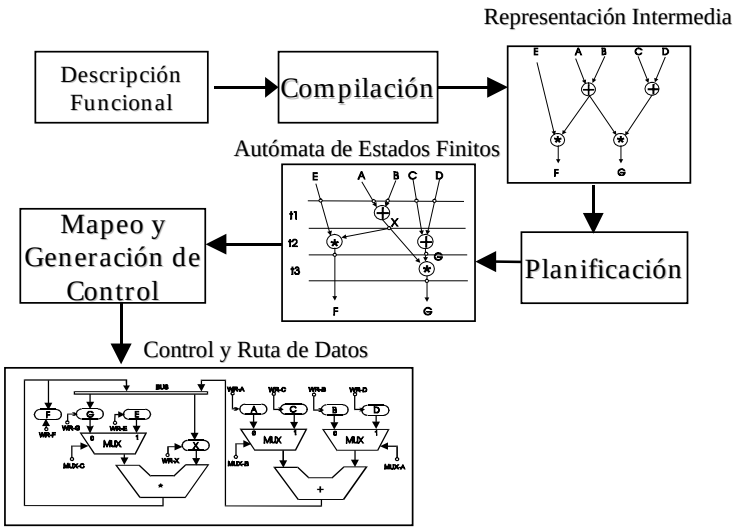

Figura 5.1: El proceso de Mapeo como tercera etapa del proceso de síntesis de alto nivel.

El mapeo de la ruta de datos involucra el asignar operaciones a operadores, asignar valores a registros y proveer las interconexiones entre operadores y registros utilizando buses y multiplexores. La decisión de utilizar unidades funcionales complejas (por ejemplo las ALUs) en vez de operadores simples, también se toma en esta etapa.

Uno de los objetivos del proceso de mapeo es minimizar algunas funciones objetivo, tales como [9]:

- Longitud total de la interconexión
- Costo total de operadores, registros, buses y multiplexores
- Retardos en la ruta crítica

Las técnicas que existen para desarrollar el mapeo de la ruta de datos pueden ser clasificadas en dos tipos: iterativas y constructivas. Estas técnicas serán descritas con detalle en las secciones posteriores. al realizar un mapeo de ruta de datos, es necesario saber cual será la arquitectura objetivo a la que se desea llegar.

Se inicia este capítulo con la descripción del problema de mapeo, para después exponer los diferentes modelos de ruta de datos que pueden adoptarse como arquitectura objetivo. También se describe cada una de las subtareas que deben llevarse a cabo durante el proceso de mapeo y se discute el orden en que estas pueden ser ejecutadas considerando la interdependencia que existe entre ellas. Estos elementos permitirán describir como se implementó esta etapa del proceso de síntesis en nuestra herramienta de síntesis.

5.2. Descripción del problema de mapeo

Durante el refinamiento de una especificación dada para llegar a un diseño estructural en base a componentes de tipo estándar, un diseñador debe, primero, seleccionar un estilo de diseño y entonces definir una arquitectura objetivo. El término “estilo de diseño” se refiere a las principales características cualitativas de diseño, tales como definir si el diseño contemplará interrupciones con prioridades o sin priorizar, cache de instrucciones, de datos o ambas, una ruta de datos orientada a buses o a multiplexores, entrada/salida serial síncrona, asíncrona o ambas, etc.

Una “arquitectura objetivo” define un diseño de manera mas precisa en términos de unidades particulares, sus parámetros, y las conexiones entre las unidades. Por ejemplo, una arquitectura de un procesador podría incluir el número de registros que éste contempla, el número de buses en la ruta de datos, el número de etapas en el pipeline, etc. Todo proceso de síntesis asume una cierta arquitectura objetivo.

Una arquitectura objetivo presenta el modelo de máquina de estados finitos con ruta de datos (FSMD) que se trató en el capítulo anterior. A partir de este modelo, es posible derivar la unidad de control y la ruta de datos. La unidad de control se deriva de la secuencia de pasos de control y de las condiciones utilizadas para determinar el siguiente paso en la secuencia.

La ruta de datos se puede derivar de la transferencia entre registros asignada a cada paso de control; a esta tarea se le llama, síntesis de la ruta de datos. Una ruta de datos en el modelo FSMD es una lista de componentes de nivel de transferencia entre registros compuesta por tres tipos de unidades: Unidades funcionales, unidades de almacenamiento y unidades de interconexión.

En resumen, podemos decir que para este modelo, la ruta de datos es el lugar en donde las operaciones se llevan a cabo, mientras que el controlador o unidad de control es el lugar en donde se toman las decisiones. La Figura 5.2, muestra las principales características de ambas partes y su interconexión.

En la Figura 5.2 puede observarse que la unidad de control es una máquina de estados finitos (FSM) cuyas entradas son las señales de control externas y algunas banderas que indican el estado de la ejecución en la ruta de datos.

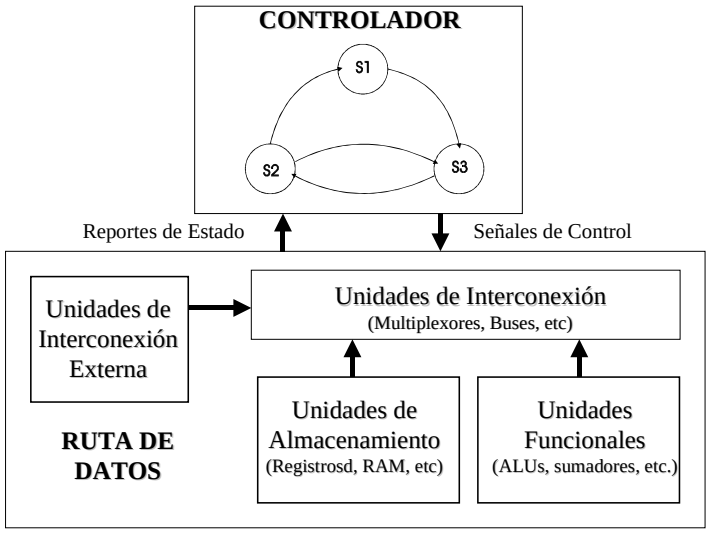

Figura 5.2: Modelo de arquitectura general.

En la ruta de datos podemos identificar los tres tipos de unidades que se mencionaron arriba:

1. **Unidades Funcionales**. Estas unidades, también llamadas operadores, ejecutan las operaciones especificadas en la descripción funcional. Unidades de este tipo pueden ser tan simples como un sumador o multiplicador, o más complejas como los co-procesadores.

2. **Unidades de almacenamiento.** Las unidades de almacenamiento mantienen los valores de las variables y constantes generadas y consumidas durante la ejecución de la función que se pretende desarrollar. Ejemplos de estas unidades son los registros, RAMs, ROMs, etc.

3. **Unidades de interconexión.** Las unidades de interconexión construyen la red de comunicación para transferencia de datos entre las unidades de almacenamiento, las unidades funcionales y los puertos externos. Ejemplos de este tipo de unidades son los buses, multiplexores, etc.

La arquitectura conformada por estas unidades es síncrona, es decir, un cambio del estado interno, seguido por ciertas acciones dentro del controlador y la ruta de datos, ocurren bajo eventos de reloj.

5.3. Modelos de ruta de datos.

Conceptualmente, una ruta de datos puede ser definida por dos aspectos. Primero, la estructura interna, incluyendo todos los tipos de unidades permitidas, y segundo, la organización funcional, es decir, los tipos de transferencias que se permiten, como se ejecutan esas transferencias y como se desarrolla la comunicación bidireccional con el controlador. Cada una de estas partes será considerada de manera general en las siguientes secciones.

5.3.1 Representación de la ruta de datos.

La organización conceptual básica de una arquitectura de la ruta de datos se muestra en la Figura 5.3. El intercambio de datos entre unidades se hace a través de unidades de interconexión, siguiendo rutas establecidas por las señales de control. Estas transferencias obedecen a un modelo que define las dependencias entre la fuente y el destino de una transferencia, y las reglas de generación de señales de control que la permiten.

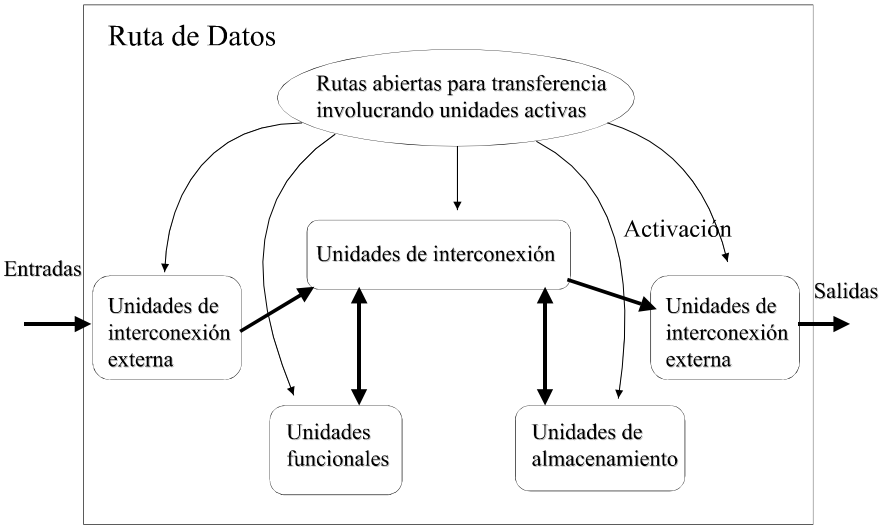

Figura 5.3: Organización conceptual de una arquitectura de ruta de datos

Cada unidad dentro de la ruta de datos está construida de la manera general ilustrada en la Figura 5.4. De manera general, cada unidad manipula datos y los transforma, los transmite o almacena. Por ello, cada tipo de unidad funcional diferirá de las otras principalmente por su funcionalidad, pero su conexión con las unidades que la rodean, seguirá esquemas similares. Una unidad funcional incluye una entrada y una salida de datos. También provee una entrada para las señales de control, como por ejemplo las de habilitación o de selección. Así mismo puede tener señales de salida que indiquen su estado de ejecución. Algo importante es que, pensando en que la arquitectura objetivo es síncrona, la unidad funcional debe contener una entrada de sincronización con los otros componentes de la ruta de datos. Esta sincronización puede lograrse por medio de una señal de reloj por ejemplo.

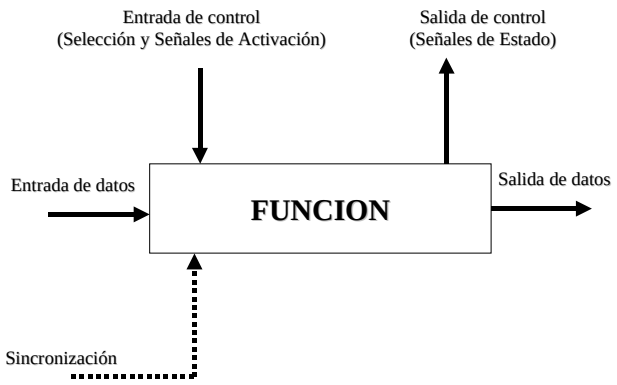

Figura 5.4: Modelo general de una unidad de ruta de datos

5.3.1.1 Unidades Funcionales

Una unidad funcional es una unidad de la ruta de datos dedicada a ejecutar las operaciones de la descripción funcional. El término de "unidad funcional" utilizado aquí es muy general e involucra todos los tipos de unidades que realizan transformaciones sobre los datos. Una unidad funcional puede ejecutar varias operaciones de diferentes tipos:

- Operaciones lógicas: AND, OR, XOR, NOT, etc.
- Operaciones aritméticas: +, -, incremento, comparación.
- Operaciones de desplazamiento

Cada operación se caracteriza por un cierto tiempo de ejecución, expresado en términos del número de ciclos de reloj que le toma producir el resultado. El tiempo de ejecución condiciona decisiones importantes en cuanto a la planificación y a la elección del periodo del ciclo de reloj que es óptimo para el diseño.

5.3.1.2 Unidades de Almacenamiento.

Las unidades de almacenamiento tales como registros, RAMs y ROMs, son los componentes que almacenan los valores de variables o constantes.

Las unidades de almacenamiento, de acuerdo al tipo de dato que almacenan, se dividen en tres categorías:

- **Registros para constantes**: Son utilizados para generar una sola constante. Es útil si el número de constantes a utilizar es pequeño. En caso contrario, se recomienda el uso de una ROM.

- **Registros para variables**: Es un registro clásico que permite almacenar nuevos valores en él, así como mantener este valor en su puerto de salida.

- **Registros de estado**: Es un registro parecido al de variables, pero tiene una salida que presenta el estado de ejecución de una unidad. Esta clase de registro se utiliza cuando el controlador hace uso de valores generados por la ruta de datos para evaluar las condiciones.

3.1.3 Unidades de Interconexión:

Las unidades de interconexión son utilizadas para controlar las transferencias de datos entre otras unidades a través de redes y buses. La Figura 5.5 muestra la forma general de una unidad de interconexión.

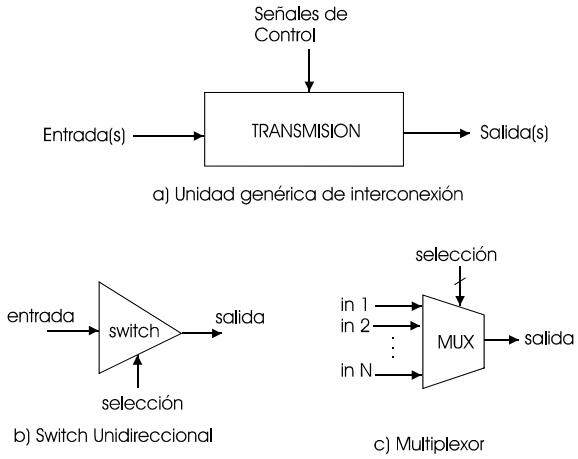

Figura 5.5: Estructura básica de las unidades de interconexión.

Existen dos tipos principales de unidades de interconexión:

- **Conmutadores uni o bi – direccionales**. Los conmutadores transfieren datos linealmente en uno o los dos sentidos respectivamente. Actúan como un buffer de tres estados (1,0, circuito abierto), cuya salida puede ser conectada a un bus.

- **Multiplexores**: Los multiplexores controlan la transferencia de datos desde multiples entradas y hacia una única salida. El multiplexor permite seleccionar una de varias entradas.

Las transferencias de datos entre los componentes, hacen uso de una red de comunicación que está compuesta, a su vez, de unidades de interconexión y el medio de comunicación (cables).

5.4. El proceso de mapeo

El Mapeo del DFG (producido por la planificación) a una ruta de datos consiste en dos tareas principales:

- Selección de unidades.
- Asignación de unidades.

La selección de unidades determina el número y tipo de componentes de nivel de transferencia entre registros (RT) que serán utilizados en el diseño. La asignación de unidades comprende el mapeo de las variables y operaciones del DFG dentro de las unidades funcionales, de almacenamiento y de interconexión.

Por cada operación en el DFG es necesaria una unidad funcional que sea capaz de ejecutar la operación. Por cada variable que se utiliza a lo largo de varios pasos de control en el DFG, se necesita una unidad de almacenamiento que mantenga el valor de las variables durante el ciclo de vida de éstas. Así mismo, para cada transferencia de datos especificada por el DFG, es necesario un conjunto de unidades de interconexión que efectúen esa transferencia.

Los métodos para resolver el problema de mapeo, se clasifican en tres tipos:

- **Métodos constructivos**: Que construyen progresivamente un diseño, mientras se explora el DFG.

- **Métodos de descomposición**: Estos métodos descomponen el problema de mapeo en sus partes constitutivas (descritas mas adelante), y resuelven cada una de ellas de manera separada.

- **Métodos iterativos**: Estos tratan de combinar e intercalar la solución de los sub-problemas del mapeo.

5.5. Tareas del proceso de mapeo.

El proceso de mapeo se conforma de cuatro tareas interdependientes:
- Selección de unidades
- Mapeo de unidades funcionales
- Mapeo de unidades de almacenamiento
- Mapeo de unidades de interconexión

5.5.1 Selección de unidades:

Un modelo de diseño simple asume que se cuenta únicamente con un tipo particular de unidad funcional por cada operación a realizar. Sin embargo, una biblioteca real de componentes contiene múltiples tipos de unidades funcionales, cada una con diferentes características (tales como funcionalidad, tamaño, retardo y disipación de potencia).

Cada unidad funcional puede implementar una o varias operaciones diferentes en la descripción al nivel de transferencia entre registros.

La tarea de selección de unidades elige el número y tipo de unidades funcionales y de almacenamiento a partir de una biblioteca de componentes.

5.5.2 Asignación de unidades funcionales

Una vez seleccionadas las unidades funcionales, sobre éstas se mapean las operaciones especificadas en el DFG. Cuando una operación puede ser mapeada a más de una unidad funcional, se aplica un algoritmo de asignación de unidad funcional que determina el mapeo exacto de las operaciones a las unidades funcionales.

5.5.3 Asignación de unidades de almacenamiento

La asignación de unidades de almacenamiento mapea los portadores de datos (tales como constantes, variables y estructuras de datos como arreglos) a unidades de almacenamiento (RAMs, ROMs, registros, etc.).

Las constantes, tales como coeficientes para algún algoritmo, generalmente son almacenadas en memorias de solo lectura (ROM). Las variables se almacenan en registros o memorias RAM. El tiempo de vida de una variable es el intervalo de tiempo entre su primera asignación de valor y su último uso y cuando el tiempo de vida de dos o mas variables no se traslapa, tales variables pueden compartir el mismo registro o espacio de memoria.

Después de que las variables han sido asignadas a los registros, éstos pueden unirse en conjuntos de registros con un solo puerto de acceso si los registros en el conjunto no son accesados de manera simultánea. De la misma manera, el conjunto de registros puede tener múltiples puertos siempre y cuando el número de registros accesados en cada paso de control no exceda el número de puertos.

5.5.4 Asignación de unidades de interconexión

Cada transferencia de datos necesita una ruta de interconexión desde su fuente hasta su destino. El objetivo de la asignación de unidades de interconexión es maximizar la utilización de estas unidades para minimizar el costo de interconexión, esto es, dos o más transferencias de datos pueden compartir todo o parte de la ruta de interconexión si no ocurren simultáneamente.

Todas las tareas de mapeo son interdependientes, esto es, los requerimientos de interconexión son mas claros después de que las tareas de asignación de unidades funcionales y de almacenamiento ya han sido desarrolladas. Por otro lado, la asignación de unidades funcionales puede tomar mejores decisiones si la tarea de asignación de unidades de almacenamiento se ha desarrollado previamente y

viceversa. Además, aquella tarea que se adopte como la inicial, no podrá aprovechar la información importante que le pudiera proporcionar alguna de las posteriores.

5.6. Métodos de mapeo

5.6.1 Método constructivo

Un método constructivo comienza con una ruta de datos vacía y va construyéndola de manera gradual adicionando unidades funcionales, de almacenamiento e interconexión conforme va siendo necesario.

Para cada operación en el DFG, un método constructivo trata de encontrar una unidad funcional en la ruta de datos (parcialmente diseñada) que sea capaz de ejecutar tal operación y que no esté en uso durante el paso de control en el cual la operación está planificada. En el caso de que existan dos o más unidades funcionales que cumplan estos requisitos, se elige aquella para la que el costo de conexión resultante sea menor. Si no se encuentra ninguna unidad funcional, entonces se deberá agregar una unidad funcional más a la ruta de datos, tomándola de la biblioteca de componentes disponibles, que pudo haber sido creada por la tarea de selección de unidades del proceso de mapeo.

De la misma manera, es posible asignar una variable a un registro disponible solo si su tiempo de vida no se traslapa con el de aquellas variables que previamente hayan sido asignadas a ese mismo registro. Si ninguno de los registros en la ruta de datos cumple esta condición, se agrega un nuevo registro. Así mismo, cuando existen múltiples alternativas para la asignación de una variable a un registro, se selecciona aquel que incremente menos el costo de la ruta de datos.

En este método, el mapeo de las unidades de interconexión se realiza inmediatamente después de que se han determinado el destino y la fuente de una transferencia de datos.

5.6.2 Métodos de descomposición

El proceso intuitivo adoptado por el método constructivo lo hace simple de desarrollar. Sin embargo, las soluciones que encuentra pueden estar lejos de ser óptimas. Con la finalidad de mejorar la calidad de los resultados, se han propuesto los métodos de descomposición, en donde el proceso de mapeo se divide en una secuencia de tareas independientes; cada tarea se transforma en un problema bien definido de teoría de grafos y entonces se soluciona con alguna técnica especializada y probada para ello.

Mientras que un algoritmo constructivo mezcla las tareas de asignación de unidades funcionales, de almacenamiento y de interconexión, los métodos de descomposición completarán una tarea antes de comenzar la otra. Debido a la interdependencia que existe entre las tareas, no se puede garantizar una solución óptima al problema de mapeo, aún cuando todas las tareas hayan sido resueltas de manera óptima

individualmente. Ejemplos de estos métodos de descomposición son el de particionamiento en cliques, el de la arista izquierda y el de comparación bipartita [5].

5.6.3 Métodos de refinamiento iterativo

Dada una ruta de datos sintetizada por algún método que bien puede ser constructivo o de descomposición, su calidad puede ser mejorada utilizando un "re-mapeo". El principal interés en el método de refinamiento iterativo son las modificaciones que deben aplicarse a la ruta de datos, la selección de una modificación durante una iteración y el criterio de terminación del proceso de refinamiento.

La forma más directa de lograr esto es un simple cambio de asignación. En general estos algoritmos realizan una serie de modificaciones a la ruta de datos con la finalidad de lograr un decremento en el costo de la ruta de datos. Por ejemplo, algunos algoritmos evalúan todos los posibles intercambios de asignación de operaciones planificadas en el mismo paso de control, midiendo la ganancia en el costo de la ruta de datos debido al cambio en las interconexiones. Entonces se elige el intercambio que resulte en la mayor ganancia y la ruta de datos se actualiza reflejando las operaciones intercambiadas. Este proceso se repite hasta que ningún intercambio de operaciones produzca una ganancia positiva.

5.7. Mapeo en HLSynth

En nuestra herramienta de síntesis El método de diseño de ruta de datos es del tipo constructivo, es decir, inicia a partir de una ruta de datos vacía a la que se le irán agregando componentes de nivel RT. La entrada es el grafo estratificado que se genera durante la planificación. En este grafo, es posible identificar las dos partes principales en una arquitectura objetivo: El controlador y la ruta de datos.

En el grafo se cuenta con toda la información acerca de las operaciones que van a ser ejecutadas, y cuantas del mismo tipo se realizarán de manera simultanea en cada paso de control. Esto ayuda a decidir cuantas unidades funcionales del mismo tipo se ocuparán como mínimo. Además, la estratificación en tiempo que presenta este grafo ayuda a construir el controlador como una máquina de estados finitos y permite observar cuales señales de control se deben habilitar en cada paso de control. Es decir, la arquitectura objetivo elegida, es una máquina de estados finitos con ruta de datos (FSMD).

La arquitectura objetivo que se eligió es simple. Está basada en multiplexores. No existen buses, y así toda la información debe hacerse pasar por los multiplexores si se pretende compartir entre varios componentes una misma entrada. Las transferencias directas entre operaciones no son permitidas; es decir, no se permite el encadenamiento de unidades funcionales. Cualquier transferencia de datos entre operadores se lleva a cabo utilizando un registro intermedio, así es que tal transferencia deberá hacerse en al menos un ciclo de reloj. La arquitectura orientada a

multiplexores permite hacer una interconexión simple. Más aún, el acceso simultáneo entre componentes resulta ser poco complejo.

Una parte importante en el algoritmo implementado se refiere a la decisión de utilizar una unidad funcional existente o agregar una nueva. El criterio de decisión es el costo de interconexión. Primero, se excluye de la lista de posibles unidades funcionales a aquellas ocupadas en el paso de control en el que la nueva operación a asignar debía ser desarrollada. Después de esto, se revisan aquellas unidades funcionales en las que es menor el costo de interconexión generado por agregar nuevas variables de entrada a dicho operador y que tanta complejidad agrega esta asignación al controlador. En la asignación de unidades funcionales, pueden darse varios casos:

1er caso. *No hay unidades funcionales disponibles que puedan realizar la operación en el paso de control actual.*

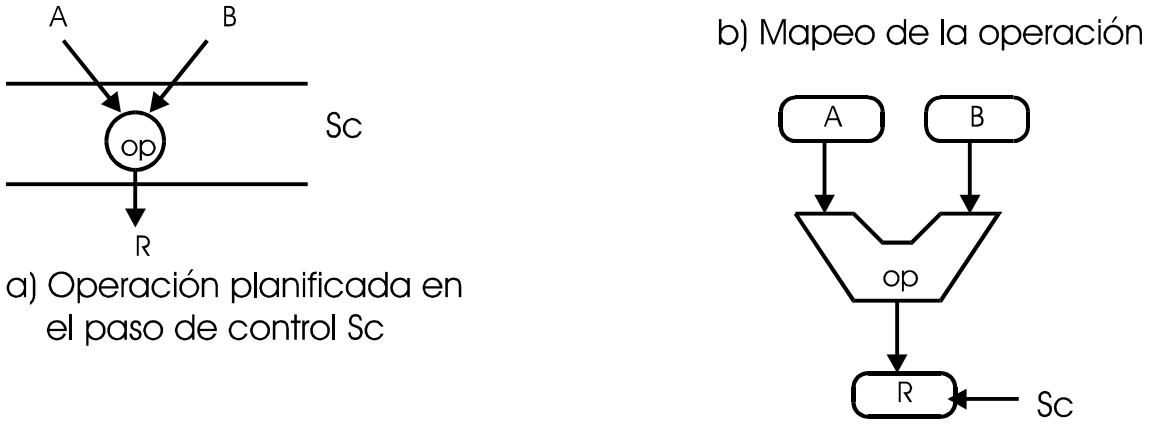

Figura 5.6 1er caso de la asignación de unidades funcionales

En la Figura 5.6 se muestran el primer caso que se presenta al mapear unidades funcionales. Supóngase que la operación mostrada en la Figura 5.6a está planificada en el paso de control *Sc*, y que la ruta de datos parcialmente construida (RDPC) está vacía, o bien, no hay unidades funcionales en la RDPC que puedan ejecutar la operación requerida en el paso de control *Sc*. Entonces, deben incluirse, tanto los registros que almacenan las variables *A* y *B*, la unidad funcional que realiza la operación *op* y el registro *R* en donde se almacenará la salida. También debe agregarse una señal de carga para el registro R en el paso de control *Sc*. En este caso, se incrementa en gran medida el costo de la ruta de datos, ya que deben agregarse tanto las unidades funcionales como la interconexión necesaria.

2º. Caso: *Existen unidades funcionales que pueden realizar la operación y que además presentan en sus entradas a uno de los operandos de la nueva operación.*

En la RDPC de la Figura 5.7, podemos observar que una de las unidades funcionales disponibles en el paso de control *Sc* presenta en una de sus entradas a la variable *A*. Por lo tanto debemos interconectar esta unidad funcional con la variable *B*. Como dos variables (*B* y *C*) deben compartir la misma entrada de la unidad funcional, debe

agregarse un multiplexor y las señales de control adecuadas. Debido a que nuestra FSM nos entrega una señal S_i por estado (Sc = '1' cuando el estado actual es Sc), podemos utilizar esta última para controlar los dispositivos. En este caso particular, cuando Sc='1' la salida del multiplexor es *B*. Además, puede observarse que el registro R se cargaba solo en el paso de control *Sp*, mas ahora también lo hace en *Sc*.

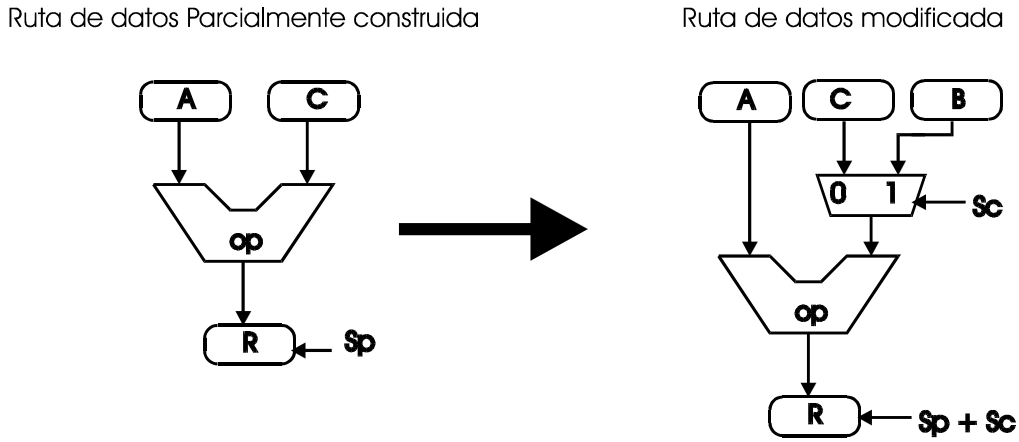

Figura 5.7: 2º caso de la asignación de unidades funcionales (Sp y Sc son el paso de control previo y nuevo respectivamente).

3er caso: *Existen unidades funcionales que pueden realizar la operación en el paso de control actual, pero todas ellas tiene operandos diferentes a los de la nueva operación.*

La forma de atacar este caso es válida si todas las unidades funcionales disponibles tiene el mismo número de operaciones asignadas en la RDPC.

Cuando se presenta este caso, se puede seleccionar cualquiera de las unidades funcionales, pero debe realizarse la interconexión de las dos variables a las entradas de las unidades. Esto provoca que se tengan que utilizar dos multiplexores para compartir las entradas de la unidad funcional, y agregar señales de control ala RDPC. En la Figura 5.8 se muestra la solución a este caso.

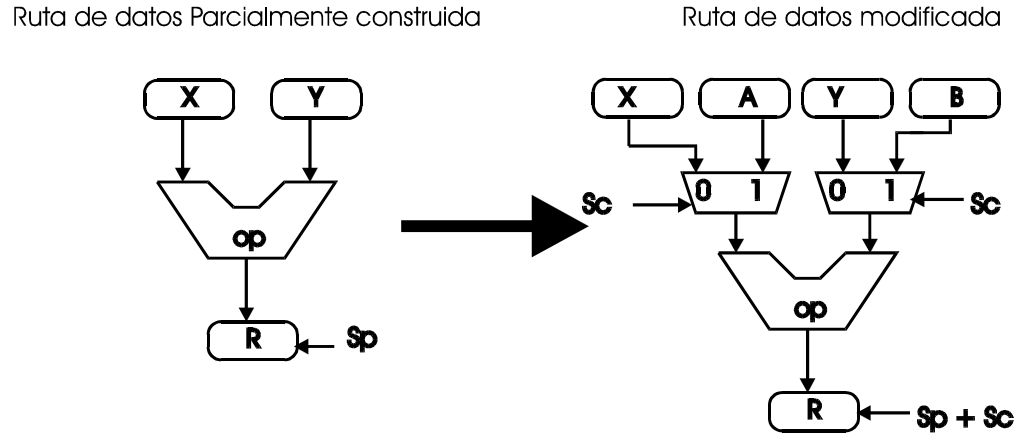

Figura 5.8: 3er caso de asignación de unidades funcionales (Sp y Sc son el paso de control previo y nuevo respectivamente).

4º caso: *Existen unidades funcionales que pueden realizar la operación en el paso de control actual, pero todas ellas tiene operandos diferentes a los de la nueva operación y además tienen diferente número de operaciones asignadas a cada una de ellas.*

Para seleccionar una unidad funcional en este caso, debe tomarse en cuenta el costo de interconexión y el efecto de ésta en la complejidad del controlador. Generalmente, se elige aquella unidad funcional que tenga el menor número de operaciones asignadas y se realiza la interconexión. Estop no ahorra multiplexores, ya que como en el caso anterior, deberemos utilizar dos, pero si distribuye la carga entre las diferentes unidades funcionales además de que no incrementa la complejidad del controlador. Este caso se muestra en la Figura 5.9.

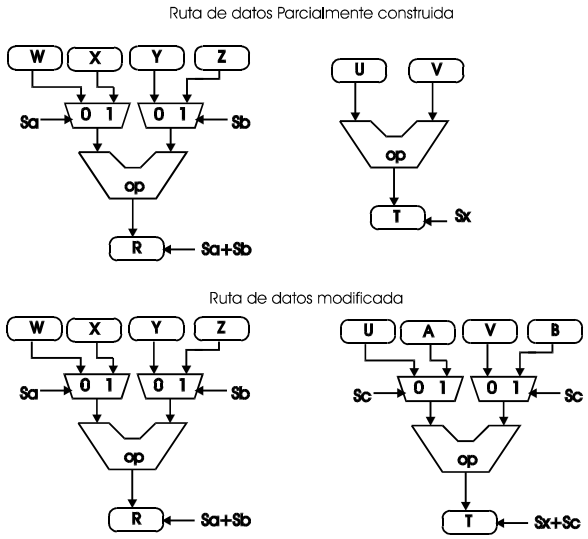

Figura 5.9: 4º caso de asignación de unidades funcionales

5.7.1 Generación de la unidad de control.

La unidad de control es una máquina de estados finitos en la que, por la naturaleza de los problemas que se resuelven con HLSynth, la transición entre estados es secuencial y en orden creciente como si fuera un contador. Por ello, utiliza un bit de control por cada estado de la máquina de estados finitos. Con este bit, indicamos en que paso de control se encuentra y habilitamos todas las señales de selección de los multiplexores que actúen en ese paso de control y todas las señales de carga de los registros que deban almacenar valores en ese paso de control. Esto permite que el modelo del controlador pueda ser extendido sin problema a cualquier número grande de estados de manera automática.

En HLSynth toda la información sobre la ruta de datos y unidad de control se mantiene en memoria. Por ello, la siguiente etapa de este proceso de síntesis consiste en generar una salida con base en la cual sea posible implementar el diseño obtenido. El proceso de síntesis como tal, ha terminado aquí, generando un diseño a nivel

estructural o RT de un sistema digital que de manera correcta desarrolla la función especificada en su descripción funcional inicial.

5.7.2 Ejemplo de Mapeo en HLSynth.

A continuación se presentan dos ejemplos de diseño que nuestra herramienta de síntesis construye a partir del grafo de flujo de datos estratificado que es generado por el proceso de planificación. La información ha sido traducida gráficamente para poder visualizar el sistema diseñado y las señales de control.

En los dos ejemplos se se sintetiza la descripción funcional:

$$F = E * (A+B)$$
$$G = (A+B) * (C+D)$$

El primer ejemplo (Figura 5.10) realiza la planificación por medio del algoritmo Force Directed Schedulling [8], obteniendo el grafo de flujo de datos estratificado mostrado en la Figura 5.10a. Después de aplicar el algoritmo de mapeo, la herramienta genera el diseño mostrado en la Figura 5.10b. En ella puede apreciarse la arquitectura objetivo orientada a multiplexores, puede observarse la ausencia de buses, y las señales de control son etiquetadas como t0, t1, t2, etc, correspondiendo el subíndice al paso de control el que se habilitan, su función puede ser la de seleccionar alguna entrada del multiplexor o bien habilitar una carga en un registro. Puede también observarse que no hay transferencias directas de datos entre unidades funcionales y que se ha agregado un registro llamado X, que aun cuando no está en la especificación inicial, es necesario por que no es posible compartir esta variable temporal con otras en un mismo registro.

Con el objeto de comparar como cambia el diseño con base a la elección del algoritmo de planificación aún cuando se tenga la misma especificación inicial, es que se incluye un segundo ejemplo (Figura 5.11). En esta ocasión, se ha elegido planificar el DFG producido por el compilador utilizando el algoritmo ASAP. Puede observarse que el nuevo diseño tiene diferencias importantes con respecto al anterior, Para empezar, logra ejecutar las funciones de la especificación en un ciclo de reloj menos que con el algoritmo FDS, además de que utiliza dos multiplexores menos. Sin embargo, ocupa una unidad funcional más que el ejemplo anterior.

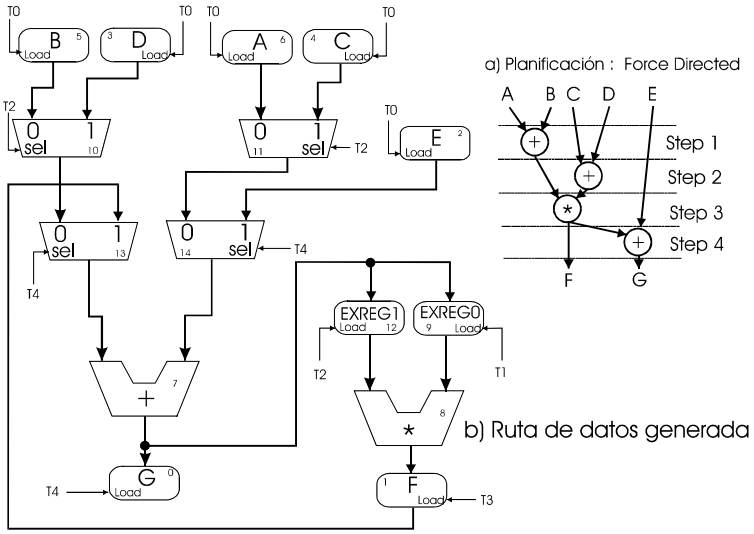

Figura 5.10: Ejemplo de mapeo, utilizando algoritmo constructivo y planificación con FDS.

La elección de cual diseño es mejor depende de los objetivos que se desean cumplir, ya sea en términos de tiempo o espacio. Ambos diseños cumplen con la especificación inicial, entonces, es ahora el diseñador quien decide con base a su experiencia, cual es el diseño que se adoptará.

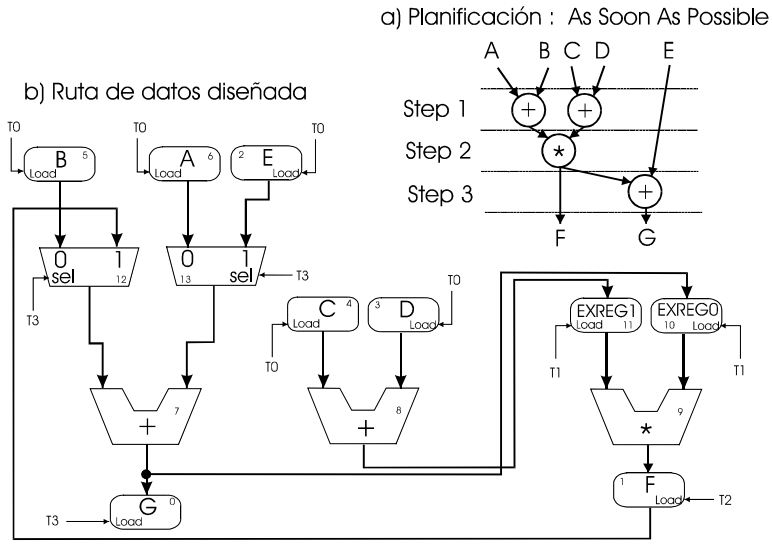

Figura 5.11: Ejemplo de mapeo cuando se ha planificado el DFG utilizando el algoritmo ASAP

5.8. Resumen

En este capítulo se describió el problema del mapeo de la ruta de datos, que esta conformado por cuatro subtareas básicas: Selección de unidades, asignación de unidades funcionales, asignación de unidades de almacenamiento y asignación de

unidades de interconexión. También se discutió la interdependencia que existe entre estas subtareas y el orden en el que pueden ser desarrolladas dependiendo del método de mapeo que utilicemos.

Se mostraron las características de algunas arquitecturas de ruta de datos y su impacto en la complejidad de su interconexión. Esto con la finalidad de resaltar la importancia de tener en cuenta una arquitectura objetivo a la que se espera llegar como resultado del mapeo, estableciendo ciertas condiciones que deben ser cumplidas para el más adecuado desempeño del sistema.

Se describieron los métodos de mapeo que pueden ser aplicados. Una conclusión importante a la que se llega es que, independientemente de la forma en que se apliquen las subtareas del mapeo, no es posible asegurar que el resultado que nos genere tal proceso será el óptimo, aún si cada una de las subtareas fueron desarrolladas de manera óptima.

En el caso de los métodos constructivos, se parte de una ruta de datos vacía, y conforme se va viajando en el grafo planificado en pasos de control, se van agregando nuevas unidades de almacenamiento o funcionales, según sea necesario para ir cumpliendo con la especificación inicial. Así mismo, el algoritmo decide si se utiliza alguno de los componentes ya existentes y disponibles en la ruta de datos para ejecutar una operación o almacenar una variable, con base a ciertos criterios que permitan mejorar la ruta de datos sin incrementar demasiado su costo.

Los métodos de descomposición se centran en particionar el problema de mapeo, en sus subtareas y resolverlas como si fueran un problema de teoría de grafos empleando técnicas especializadas para ello. En este tipo de algoritmos las subtareas son ejecutadas de manera secuencial, es decir, no se mezclan y una subtarea solo puede ser ejecutada después de que alguna otra ya haya sido terminada.

Finalmente, se explicó cuales fueron las técnicas de mapeo utilizadas en la herramienta de síntesis construida como objetivo de esta tesis, y se muestran algunos ejemplos de los diseños que se han generado con ella.

Capítulo 6

Generación de Código VHDL

6.1. Introducción

El proceso de síntesis de alto nivel, como se ha visto hasta ahora, genera un diseño al nivel estructural del sistema digital que realiza las funciones especificadas inicialmente mediante la descripción funcional.

Este diseño se realiza al nivel de transferencia entre registros, y con base en componentes digitales de ese nivel de abstracción. Cuando la herramienta genera un diseño, la salida puede ser creada de manera tan simple como por ejemplo un reporte en un archivo de texto en el cual se especifiquen los componentes que constituyen el diseño, o tan compleja como, por ejemplo, visualizar el diseño de manera gráfica mediante algún archivo que pueda ser visto y editado en alguna herramienta CAD para diseño de circuitos digitales. Existen formatos estándar para la especificación de diseños de sistemas digitales, tales como EDIF y VHDL, que permiten portar el diseño a diferentes aplicaciones de CAD para simulación y/o implantación.

Nuestra herramienta proporciona al usuario la especificación de un diseño utilizando un lenguaje de descripción de hardware, concretamente VHDL. Existe una gran cantidad de herramientas para simulación y síntesis que trabajan con VHDL: MentorGraphics, Synopsis, Vielogic, Cadence, etc. A partir de un diseño en VHDL, es posible construir un circuito para FPGAs, CPLDs o circuitos VLSI. La especificación que se hace contempla la ruta de datos y el controlador, todo ello al nivel estructural (Figura 6.1).

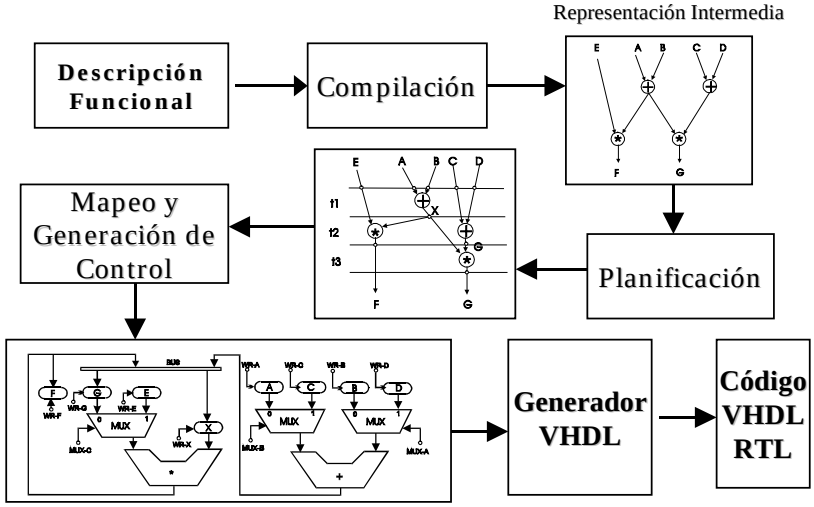

Figura 6.1: Generación de código VHDL. Ultima etapa del proceso de síntesis de alto nivel.

En este capítulo se describirán algunas características de los lenguajes de descripción de hardware, se resaltarán las ventajas de VHDL y se describirá también el generador de código VHDL desarrollado para HLSynth, nuestra herramineta de síntesis.

6.2. Lenguajes de descripción de hardware.

Las metodologías de diseño basadas en lenguajes de descripción de hardware (HDLs) han ido creciendo en aceptación y se van haciendo cada vez más populares. Los HDLs son utilizados principalmente para describir la estructura o el comportamiento de un sistema digital con finalidades de simularlo o sintetizarlo.

Las metodologías basadas en HDLs tienen ciertas ventajas sobre las metodologías tradicionales de diseño. Primero, mejoran la productividad. Esto permite que un diseñador pueda crear un nuevo diseño en menos tiempo y además permite que personas que no tengan muchos conocimientos en diseño de circuitos puedan crear diseños nuevos. Segundo, los diseños son portables a diferentes tecnologías, es decir, las descripciones en algún HDL son independientes de la tecnología.

Los HDLs tienen dos aplicaciones principales: la documentación y el modelado de un diseño [5]. Existen simuladores que pueden validar un diseño especificado en lenguajes y herramientas de síntesis que pueden generar automáticamente un diseño a partir de una especificación en un HDL. De entre estos lenguajes, el más empleado es VHDL, que es reconocido por la IEEE como un estándar (el estándar 1076).

6.2.1 VHSIC Hardware Description Languaje (VHDL)

VHDL es un lenguaje fuertemente tipificado con un amplio conjunto de construcciones computacionales. VHDL permite el modelado y simulación de circuitos en todas las etapas del proceso de diseño, además de realizar síntesis, pruebas al hardware y análisis de tiempos de un circuito [3].

VHDL es un lenguaje concurrente que permite una descripción de un sistema a diferentes niveles, soporta un manejo concurrente de señales así como las descripciones secuenciales ordinarias. El diseño en VHDL generalmente se hace genérico, para después asignarle bibliotecas específicas. El tipo de dato básico de VHDL es la señal (signal), una variable que contiene una componente de tiempo.

VHDL permite la descripción de un diseño digital a diferentes niveles de abstracción tales como nivel algorítmico, de transfrenecia entre registros, y a nivel de compuertas lógicas. Es posible abstraerse en un diseño u ocultar los detalles de implementación del mismo. En una metodología de diseño de tipo “top-down”, el diseñador comienza representando el diseño en un nivel de abstracción alto, y después lo va convirtiendo a niveles de abstracción más detallados.

En VHDL, un componente de un circuito es representado como una entidad que puede estar asociada con varias arquitecturas alternativas. Típicamente, una arquitectura

puede ser especificada por una descripción abstracta de tipo funcional de un dispositivo, o proveer una definición estructural concreta en términos de componentes simples (Figura 6.2). La equivalencia de las arquitecturas puede ser confirmada mediante simulaciones comparativas. Las arquitecturas generalmente pueden ser de los tres tipos siguientes:

- **Arquitectura de tipo funcional**: Esta especifica que es lo que hace un sistema, en una manera parecida a la de un programa de cómputo, pero sin proveer detalles sobre la implementación del diseño.

- **Arquitectura de tipo de flujo de datos**: Esta arquitectura especifica un sistema como una representación concurrente del control de flujo y movimiento de datos. Aquí se modela el flujo de la información o el comportamiento del flujo de datos como funciones de lógica combinacional, tales como sumadores, comparadores, decodificadores y compuertas lógicas primitivas.

- **Arquitectura de tipo estructural**: La implementación estructural se define utilizando declaración e instancias de componentes. La descripción estructural incluye una lista de componentes activos de manera concurrente y sus respectivas interconexiones.

A continuación se describirán las técnicas de modelado de circuitos digitales a nivel estructural y de transferencia entre registros empleando VHDL.

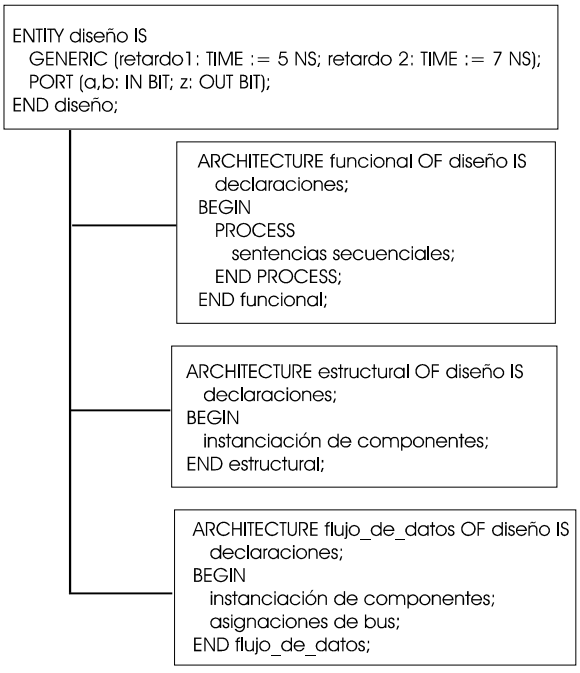

Figura 6.2: Código típico de VHDL con arquitecturas alternativas

6.3. Modelado de circuitos digitales

Un sistema digital generalmente es representado como una colección jerárquica de componentes. Cada componente tiene un conjunto de puertos que lo comunican con otros componentes. Por ello, la unidad básica de una descripción estructural es una instancia de un componente.

Debido a que el sintetizador de alto nivel diseña la ruta de datos al nivel de transferencia entre registros, entonces los componentes que utilizaremos para construirla serán:

- Multiplexores como unidades de interconexión
- Registros como unidades de almacenamiento
- Unidades funcionales para ejecutar las operaciones.

Por ello, se muestra a continuación la descripción en VHDL de cada uno de estos elementos.

6.3.1 Modelado del Multiplexor.

Este multiplexor tiene dos entradas, una salida y un bit de selección que permite determinar cual de los dos datos en las entradas debe pasar a la salida. La descripción en VHDL de este componente es la siguiente:

```
entity MULTIPLEXER is
     port (InA, InB  : in integer;
          Output     : out integer;
          Sel        : in BIT);
end MULTIPLEXER;


architecture BHV of MULTIPLEXER is
begin
     process (InA, InB, Sel)
     begin
          if (Sel = '0') then
               Output <= InA;
          else
               Output <= InB;
          end if;
     end process;
end BHV;
```

6.3.2 Modelado de un registro:

Esta es la unidad de almacenamiento que se utiliza en la ruta de datos que se construye. Por medio de la activación del bit de entrada Load, se almacena en el

registro el valor que se encontraba a su entrada, y mientras Load esté deshabilitado, no se permite ningún cambio en la información almacenada.

```
entity REG is
     port  (Din      : in integer;
          Dout       : out integer;
          Load       : in BIT);
end REG;

architecture BHV of REG is
begin
     process (Load)
     begin
          if (Load'event and Load = '0') then
               Dout <= Din;
          end if;
     end process;
end BHV;
```

6.3.3 Modelado de unidades funcionales

Por la naturaleza de los sistemas que sintetiza nuestra herramienta de síntesis, se describieron las siguientes unidades funcionales. Pero como se explicará en secciones posteriores, la herramienta es flexible y permite al usuario definir sus propias unidades funcionales y que éstas sean consideradas dentro del diseño. En las siguientes subsecciones mostraremos las descripciones de las unidades funcionales implementadas en el sintetizador.

6.3.3.1 Sumador y Restador

La descripción funcional de esta unidad nos muestra un sumador que no puede ser conectado en cascada con otros debido a la ausencia del acarreo. Este componente realiza la suma de dos enteros presentando a la salida el resultado de esa operación. El componente es asíncrono.

```
entity ADDER is
     port (InA, InB  : in integer;
          SUM        : out integer);
end ADDER;

architecture BHV of ADDER is
begin
     process (InA, InB)
     begin
          SUM <= InA + InB;
     end process;
end BHV;
```

El restador sigue el mismo principio que el sumador. Su descripción se muestra a continuación:

```
entity SUBS is
      port (InA, InB  : in integer;
            Result          : out integer);
end SUBS;

architecture BHV of SUBS is
begin
      process (InA, InB)
      begin
            Result <= InA - InB;
      end process;
end BHV;
```

6.3.3.2 Comparadores

Los comparadores son unidades funcionales para determinar el máximo y mínimo valor de dos números enteros en sus entradas. Estas unidades pueden conectarse secuencialmente si se requiere dicho cálculo para más de dos valores.
La descripción de la función “mínimo” es:

```
entity MIN is
      port (AData, BData    : in integer;
            Dout            : out integer);
end MIN;

architecture BHV of MIN is
begin
      process (AData, BData)
      begin
            if (AData >= BData) then
                  Dout <= BData;
            else
                  Dout <= AData;
            end if;
      end process;
end BHV;
```

La descripción de la función “máximo” es:

```
entity MAX is
      port (     AData, BData    : in integer;
            Dout            : out integer);
end MAX;
```

```
architecture BHV of MAX is
begin
      process (AData, BData)
      begin
            if (AData <= BData) then
                  Dout <= BData;
            else
                  Dout <= AData;
            end if;
      end process;
end BHV;
```

Como puede observarse, es sencillo el crear nuevas unidades funcionales que sean necesarias para la implementación del diseño obtenido. La herramienta de software que se utilizó en este trabajo para poder compilar el código VHDL generado y simular el comportamiento del sistema diseñado, fue ALTERA.

Para poder compilar el código VHDL del diseño generado por la herramienta de síntesis, la descripción de cada uno de los componentes debe estar en la biblioteca Work.

6.4. Generación de código VHDL.

En esta sección describiremos el proceso de generación de código VHDL que realiza HLSynth a partir de las estructuras de datos en memoria con las que la computadora representa internamente el diseño. El proceso es el mostrado en la Figura 6.3. En esta figura puede identificarse una FSM que funciona como entrada a este proceso. Esta FSM está planificada.

Para generar el código VHDL que modele a este sistema deben considerarse todas las operaciones que realiza y el generador debe tener una correspondencia entre las operaciones y los componentes de una biblioteca de código que modela esas operaciones.

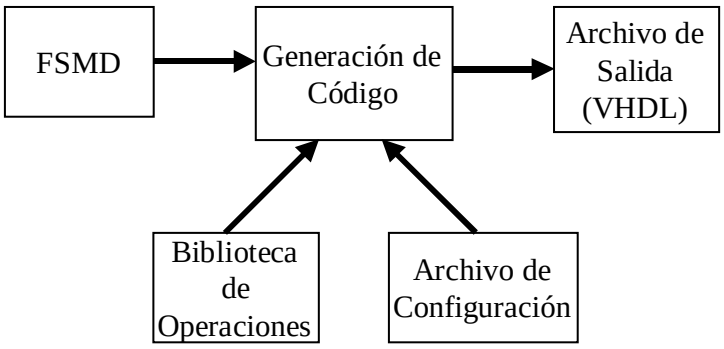

Figura 6.3: Proceso de generación de código VHDL

La bilbioteca en HLSynth está constituida por todos los modelos en VHDL de la sección anterior, y es llamada WORK. La correspondencia de operaciones–elementos de biblioteca está dada por un archivo de configuración de operaciones que provee el usuario. Esta es una tabla que indica para cada operación, que modelo en VHDL debe utilizar para que pueda realizarse tal operación.

Con estos elementos el generador puede construir un código VHDL en el que interconecte los elementos de biblioteca necesarios para realizar la función requerida en la descripción funcional.

Los pasos del proceso de generación de código pueden resumirse de la siguiente manera (Figura 6.4):

1: Solicitar archivo de configuración

2: Generación de ruta de datos
2.1 Generar entidad
2.1.1 Búsqueda de los registros de entrada y salida para utilizarlos como signals en la entidad.
2.1.2 Incluir un signal de entrada para sincronización
2.1.3 Construir la entidad

2.2 Generar arquitectura
2.2.1 Identificar y crear los signal intermedios para interconexión de dispositivos
2.2.2 Identificar y crear los signal para la interconexión con la unidad de control
2.2.3 Generación de la representación estructural utilizando los componentes de la biblioteca WORK con base en el archivo de configuración
2.2.3.1 Identificar e instanciar componentes de la biblioteca WORK (unidades funcionales, registros, multiplexores)
2.2.3.2 Interconexión de registros
2.2.3.3 Interconexión de multiplexores
2.2.3.4 Interconexión de unidades funcionales
2.2.3.5 Agregar al sistema, buffers de salida

3: Generación de unidad de control
3.1 Generar entidad
3.1.1 Crear un signal de salida por cada paso de control en la FSM
3.1.2 Incluir un signal de entrada para sincronización
3.1.3 Construir entidad
3.2 Generar arquitectura

3.2.1 Describir funcionalmente un contador cuyo módulo es el total de pasos de control de la FSM
3.2.2 Describir funcvionalmente un decodificador que habilite un signal de salida por cada paso de control

Figura 6.4 Algoritmo de generación de código VHDL

Para explicar este procedimiento, se tomará como base uno de los ejemplo descritos en el capítulo anterior. La salida del mapeo se presenta en la Figura 6.5, y a partir de ahí se generará el código VHDL que describa a ese diseño.

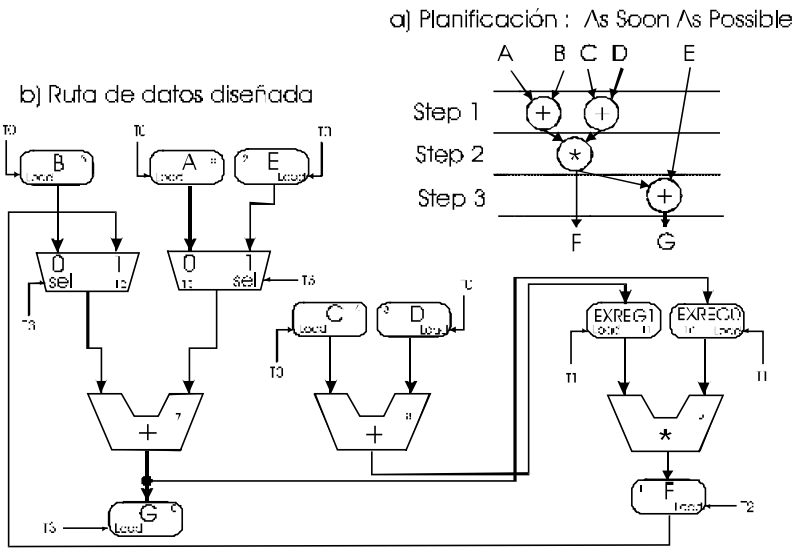

Figura 6.5: Ejemplo de Mapeo a partir del cual se generará su código VHDL

6.4.1 Archivo de configuración

El archivo de configuración es una tabla en la que se especifica una correspondencia entre la operación a realizar y la unidad funcional (que debe existir en WORK) a ser utilizada para desarrollarla. Esta tabla puede ser editada por el diseñador e incluir ahí sus nuevas unidades funcionales indicando para que tipo de operación serán utilizadas. Cuando el generador de código comienza a ejecutarse, lo primero que solicitará al diseñador es el archivo de configuración de operaciones. Este archivo tiene la forma siguiente:

```
+     Adder
*     Multiplier
-     Subs
MAX   Max
MIN   Min
<     Min
>     Max
```

Del lado izquierdo se especifica la operación y del derecho la unidad funcional a utilizar.

6.4.2 Generación de ruta de datos

Cuando la computadora termina con el proceso de síntesis, mantiene en memoria un arreglo de objetos, en donde cada objeto puede ser identificado como unidad funcional, unidad de almacenamiento o de interconexión. A partir de esta estructura, podemos obtener toda la información necesaria para construir la entidad y arquitectura de la ruta de datos.

6.4.2.1 Generación de la entidad de la ruta de datos

Primero se buscan todos los registros de entrada del sistema (aquellos que no tiene predecesores) para ser declarados como puertos de entrada en la lista de sensitividad de la entidad (entity) que se generará para la ruta de datos. En el caso del ejemplo, se contemplarán los registros de entrada A, B, C, D y E.

Así mismo, se buscan también todos los registros de salida, es decir aquellos que no tienen sucesores y que mantienen las variables finales después del procesamiento interno de la información. Estos registros, también son declarados como puertos de salida en la lista de sensitividad. En el ejemplo, los registros de salida son G y F, a los cuales se les agrega la terminación "out" para indicar que son de salida.

Agregando únicamente una señal de entrada más correspondiente al reloj, podemos generar totalmente la entidad para esta ruta de datos. Para el ejemplo que se está siguiendo, se obtiene el siguiente fragmento de código:

```
LIBRARY WORK;
Use WORK.ALL;

entity simple is
port ( CLK           : in BIT;
       E, D, C, B, A :  in integer;
       Gout, Fout    :  out integer);
end simple;
```

Como puede observarse, también se ha agregado la biblioteca en la que tenemos la descripción de los componentes básicos "WORK".

6.4.2.2 Generación de la arquitectura de la ruta de datos

Después de haber generado la entidad, debemos generar la parte operativa, es decir, la arquitectura para la ruta de datos. Debido a que estamos construyendo el diseño a nivel estructural, debemos recopilar información acerca de los signals intermedios del diseño. Se considerará un signal intermedio a aquel punto que une a una unidad de cualquier tipo con otra de cualquier tipo. Por ejemplo, si a la salida de un sumador se coloca un registro, entonces ahí existirá un signal intermedio, y es el punto de unión entre la unidad funcional y el registro. Mediante la exploración de todas las unidades dentro de la ruta de datos, el generador de código obtiene una lista de todos los signals intermedios y les asigna un nombre que comienza con "N" y es seguido de un número de identificación consecutivo (0,1,2…).

En esta parte, también es necesario crear otros signals para la interface con la unidad de control. Se van a crear tantos signals como estados tenga la máquina de estados finitos de la unidad de control., y serán llamados T0, T1,…Para el ejemplo, se genera un fragmento de código de este tipo:

```
architecture STRUCTURE of simple is

   signal N0, N1, N7, N8, N9, N10, N11, N12, N13     :  integer;
   signal T0, T1, T2, T3                             : BIT;
```

El último paso en la generación del código de la arquitectura de la ruta de datos, es realizar las conexiones en base a los componentes descritos funcionalmente en la biblioteca WORK. Considerando para cada componente sus signals intermedios, es posible interconectarlo con ellos utilizando la función “portmap” de VHDL y generar la estructura final asociando para cada operación el operador correspondiente de acuerdo a la tabla que se introduce al programa como archivo de configuración de operaciones.

La arquitectura generada para la ruta de datos del ejemplo, es la siguiente:

```
architecture BHV of simple is
begin
          U1:     Work.Reg          port map ( N7, N0, T3 );
          U2:     Work.Reg          port map ( N9, N1, T2 );
          U3:     Work.Adder        port map ( N12, N13, N7 );
          U4:     Work.Adder        port map ( D, C, N8 );
          U5:     Work.Multiplier   port map ( N11, N10, N9 );
          U6:     Work.Reg          port map ( N7, N10, T1 );
          U7:     Work.Reg          port map ( N8, N11, T1 );
          U8:     Work.Multiplexer  port map ( B, N1, N12, T3 );
          U9:     Work.Multiplexer  port map ( A, E, N13, T3 );
          U10:    Control           port map ( CLK, T0, T1, T2, T3 );
          U11:    Work.Buf          port map ( N0, Gout );
          U12:    Work.Buf          port map ( N1, Fout );
end STRUCTURE;
```

6.4.3 Generación de la unidad de control

6.4.3.1 Generación de la entidad de la unidad de control

A continuación debe especificarse la entidad correspondiente al controlador. Sabemos que el controlador será una máquina de estados finitos, así es que por su naturaleza síncrona, requerirá como entrada una señal de reloj, y como salida tendremos un bit por estado, como se había convenido en la arquitectura objetivo en el capítulo anterior. De esta forma, para el ejemplo, se generaría lo siguiente:

```
entity CONTROL is
      port (CLK              : in BIT;
            T0, T1, T2, T3   : out BIT );
end CONTROL;
```

6.4.4 Generación de la arquitectura para la unidad de control

El controlador, es una máquina de estados finitos, y su implementación es por medio de un contador que lleva la secuencia de los estados, y un decodificador que se encarga de habilitar un bit de su salida para indicar el estado presente de la máquina de estados finitos. Ese bit que se habilita en el decodificador es el que se utiliza como señal de control habilitando unidades funcionales, almacenando valores en los registros y seleccionando rutas de flujo de datos mediante los multiplexores.

La implementación funcional se hace con un contador y después se utilizan construcciones del lenguaje VHDL para seleccionar el bit (que ya se ha definido como puerto de salida en la entidad) que deberá ser habilitado de acuerdo al valor del contador. La arquitectura generada apara el ejemplo, es la siguiente:

```
architecture BHV of CONTROL is
begin
            process (CLK)
            variable state : integer range 0 to 3;
            begin
            if (CLK'event and CLK = '1') then
                  State := State + 1;
                  case State is
               when 0 =>
                  T0 <= '1'; T1 <= '0'; T2 <= '0'; T3 <= '0';
               when 1 =>
                  T0 <= '0'; T1 <= '1'; T2 <= '0'; T3 <= '0';
               when 2 =>
                  T0 <= '0'; T1 <= '0'; T2 <= '1'; T3 <= '0';
               when 3 =>
                  T0 <= '0'; T1 <= '0'; T2 <= '0'; T3 <= '1';
                  State := 0;
            end case;
      end if;
      end process;
end BHV;
```

La compilación y simulación de este código, comprueba que el diseño generado realmente desarrolla de manera correcta las funciones especificadas en la descripción funcional.

5. Resumen

La etapa de generación de código VHDL, es la última en el proceso de síntesis de alto nivel que sigue la herramienta de software desarrollada para ello. Aquí se obtiene une representación del diseño generado, utilizando una descripción estructural en VHDL al nivel de transferencia entre registros.

La descripción estructural que se hace del sistema, tiene como unidades básicas, los componentes a nivel RTL, como son registros, multiplexores, unidades funcionales,

etc. y como unidades básicas, estos componentes requieren una descripción aparte, tal que pueda ser incluida en la representación estructural si entrar en detalles de implementación.

Estas descripciones son incluidas en la biblioteca “WORK” del lenguaje de descripción de hardware y son accesadas cuando se les requiere. En este capítulo se revisaron las descripciones para esos componentes básicos y queda abierta la posibilidad para el diseñador, de crear nuevas unidades funcionales que puedan ser incluidas en futuros diseños.

Se describió también a detalle, cuál es el procedimiento que sigue la herramienta de síntesis para generar código VHDL a partir de las estructuras de datos que la computadora tiene como representación del diseño y se mostró un ejemplo de tal generación de código.

La ruta de datos se genera ensamblando componentes por medio de la función “portmap” de VHDL, mientras que el controlador se implementa como un contador con decodificador, permitiendo esto el habilitar un bit por cada estado en el que se encuentre el autómata.

Las pruebas que se han hecho a este generador de código han sido exitosas en todos los casos, siempre la traducción ha sido correcta, ya que el sistema es correcto por construcción.

Capítulo 7

Pruebas de HLSynth

7.1 Introducción

En este capítulo se muestran algunos diseños obtenidos por la herramienta a partir de una descripción funcional. Se desarrollará un ejemplo de síntesis y se generarán diferentes diseños que cumplen con objetivos distintos.

Este capítulo está dividido en secciones que muestran los resultados obtenidos para cada una de las etapas del proceso de síntesis. Se inicia en la sección 7.2, con la compilación, para después seguir con los métodos de planificación y mapeo en las secciones 7.3 y 7.4 respectivamente. Por último se cubre la generación de código VHDL.

El ejemplo que se presentará es el de generar un procesador que evalue las expresiones siguientes:

$$F = (A+B) * (C+D)$$
$$G = F * E$$

La expresiones de arriba son dos ecuaciones en las que existe una fuerte dependencia de datos. Los diseños será optimizados para recursos y para tiempo.

7.2 Compilación

Nuestra herramienta de síntesis requiere como entrada un modelo del sistema que se va a sintetizar. Este modelo, es especificado utilizando un lenguaje de descripción funcional. El diseñador debe dibuja un DFG y enumera los nodos que lo constituyen. El grafo dibujado y enumerado del ejemplo, queda como se muestra en la Figura 7.1.

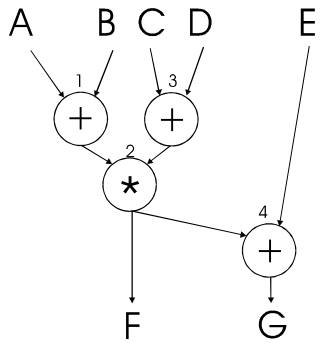

Figura 7.1: DFG con nodos enumerados.

Utilizando el lenguaje de descripción funcional que fue desarrollado en esta tesis, deberemos ingresar al sintetizador un archivo que llamaremos "*simple.grf*", con el contenido que se muestra en la Figura 7.2.

```
VAR A 1
VAR B 1
VAR C 3
VAR D 3
VAR E 4
+ 1 2
+ 3 2
* 2 4,F
+ 4 G
```

Figura 7.2: Contenido del archivo *simple.grf*.

La interpretación de este archivo puede encontrarse en el capítulo 3, sección 3.4 de este documento, titulada "Compilación en la herramienta de síntesis".

Una vez creada la especificación y después de haber sido ingresada al sintetizador, iniciará la etapa de compilación. Esta etapa genera un DFG que funciona como representación intermedia, y que será utilizada en las etapas siguientes.

6.3 Planificación

Después de la compilación, iniciará el proceso de asignar pasos de control a cada una de las operaciones en el grafo. En esta parte, HLSynth solicitará al usuario que le indique que algoritmo de planificación debe utilizar. Las opciones que se presentan son tres:

1) As Soon As Possible
2) As Last As Possible
3) Force Directed Schedulling

Iniciaremos este ejemplo seleccionando un algoritmo que nos de la planificación que represente la ejecución del grafo más rápida posible. Entonces utilizaremos el algoritmo ASAP.

Después de seleccionar el algoritmo, se inicia el proceso y al final se genera un reporte que indica la forma en que se planificó el DFG. Este reporte se encuentra en el archivo "*sched.grf*". el contenido de este reporte se muestra en la Figura 7.3.

En este archivo se indica cual es el algoritmo de planificación utilizado, cuantos pasos de control se generaron y cuales operaciones se ejecutan en cada paso de control. La interpretación de este archivo es la siguiente:

Para cada paso de control, existe una línea de texto indicando cual de todos es. Y a continuación se encuentran en forma de lista las operaciones planificadas en ese paso de control. Para cada operación existe una línea de texto.

```
PLANIFICACION POR EL METODO:     As Soon As Possible

PLANIFICACION EN 3  PASOS DE CONTROL.

Paso de control 1:
      +     1     2,
      +     3     2,
Paso de control 2:
      *     2     -1,4,
Paso de control 3:
      +     4     -1,
```

En cada línea se encuentra, primero, cual es el tipo de operación planificada, después el número de nodo que se le asignó como identificador en la descripción funcional, seguido por los nodos a los que debe transferir el resultado de la operación. Cuando en alguno de los nodos de destino se encuentra un –1, entonces el resultado debe transferirse a un registro de salida. Gráficamente podemos ver la planificación en la Figura 7.4

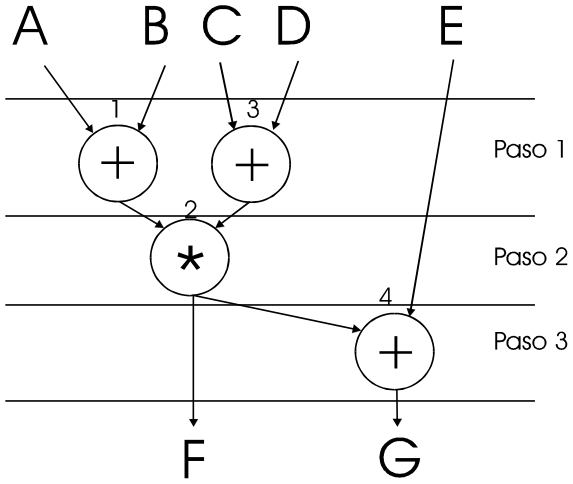

Figura 7.4 Planificación del DFG

7.4 Mapeo

Cuando ya se ha planificado el DFG, se procede a asignar unidades funcionales, de interconexión y de almacenamiento al DFG para construir la ruta de datos. Esto se realiza durante el mapeo. El algoritmo de mapeo que utiliza esta herramienta es de tipo constructivo. Al finalizar esta etapa, se habrá generado una estructura de datos compleja, que representa la ruta de datos. También se genera un reporte en el que se indican todas las unidades funcionales, registros e interconexiones entre ellas. En este reporte se asigna un número a cada unidad utilizada, con la finalidad de que pueda construirse un diagrama a nivel RTL de la ruta de datos construida. Además se incluyen para cada dispositivo las señales de control que serán utilizadas para sincronizar su funcionamiento. Este reporte se muestra en la Figura 7.5 a dos columnas.

```
Device #0   ->    G
       Load Control :     t3,
       Input Devices:
                    Device #7
       Output Devices:

Device #1   ->    F

       Load Control :     t2,
       Input Devices:
                    Device #9
       Output Devices:
                    Device #12

Device #2   ->    E
       Load Control :
       Input Devices:
       Output Devices:
                    Device #13


Device #3   ->    D
       Load Control :
       Input Devices:
       Output Devices:
                    Device #8

Device #4   ->    C
       Load Control :
       Input Devices:
       Output Devices:
                    Device #8


Device #5   ->    B
       Load Control :
       Input Devices:
       Output Devices:
                    Device #12


Device #6   ->    A
       Load Control :
       Input Devices:
       Output Devices:
                    Device #13


Device #7   ->    FU_+_DEV7
       Unit busy in :     t3,     t1,
       Input Devices:
                    Device #12
                    Device #13
       Output Devices:
                                  Device #10
                                  Device #0


Device #8   ->    FU_+_DEV8
       Unit busy in :     t1,
       Input Devices:
                                  Device #3
                                  Device #4
       Output Devices:
                                  Device #11

Device #9   ->    FU_*_DEV9
       Unit busy in :     t2,
       Input Devices:
                                  Device #11
                                  Device #10
       Output Devices:
                                  Device #1


Device #10   ->    EXREG_0
       Load Control :     t1,
       Input Devices:
                                  Device #7
       Output Devices:
                                  Device #9

Device #11   ->    EXREG_1
       Load Control :     t1,
       Input Devices:
                                  Device #8
       Output Devices:
                                  Device #9


Device #12   ->    MUX_DEV12
       Select Control :     t3,
       Input Devices:
                                  Device #5
                                  Device #1
       Output Devices:
                                  Device #7


Device #13   ->    MUX_DEV13
       Select Control :     t3,
       Input Devices:
                                  Device #6
                                  Device #2
       Output Devices:
                                  Device #7
```

Figura 7.5 Reporte de mapeo.

Si traducimos el reporte anterior a un diagrama esquemático del circuito sintetizado, obtendremos la ruta de datos mostrada en la Figura 7.6.

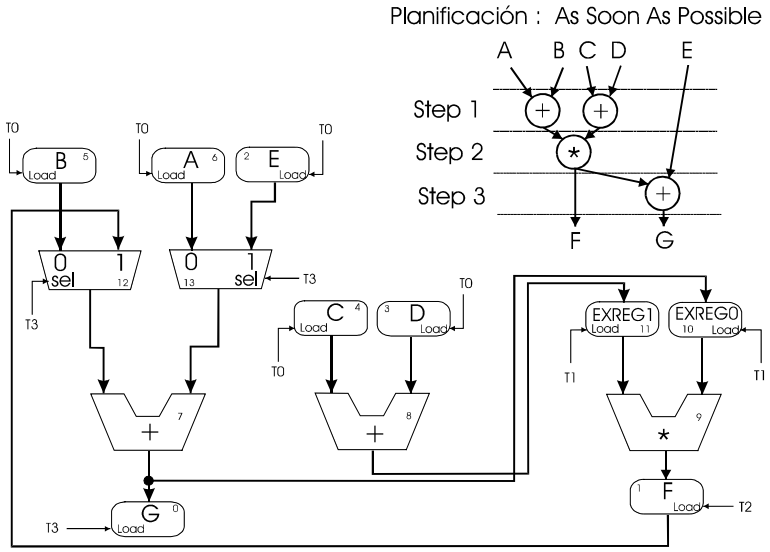

Figura 7.6 Ruta de datos para el diseño sintetizado.

7.5 Generación de código VHDL

Todo lo anterior debe ser representado en una forma que sea sencilla de implementar como sistema digital. HLsynth genera una salida en código VHDL que modela al sistema y permite simularlo o implementarlo físicamente.

Esta salida se genera en la etapa de generación de código. En esta etapa se requiere el archivo de configuración de operaciones y una biblioteca de dispositivos. Esta biblioteca deberá contener todos los tipos de dispositivos utilizados durante el mapeo y deben estar modelados en VHDL e incluidos en la biblioteca WORK.

El archivo de configuración de operaciones (ver capítulo 6, sección 6.4) es el mostrado en la Figura 7.7.

```
+ Adder
* Multiplier
- Subs
MAX Max
MIN Min
< Min
> Max
```

Figura 7.7 Archivo de configuración de operaciones

El código VHDL generado por HLSynth incluye tanto la ruta de datos como la unidad de control. Este código se presenta en la Figura 7.8 y es el final de la síntesis de alto nivel.

```
LIBRARY WORK;
Use WORK.ALL;

entity simple is
      port ( CLK    : in BIT;
             E, D, C, B, A    :  in integer;
             Gout, Fout    :  out integer);
end simple;

entity CONTROL is
      port ( CLK    : in BIT;
             T0, T1, T2, T3, T4    : out BIT );
end CONTROL;


architecture STRUCTURE of simple is

   signal N0, N1, N7, N8, N9, N10, N11, N12, N13, N14    :  integer;
   signal T0, T1, T2, T3, T4     : BIT;

   begin
          U1:Work.Reg                  port map ( N7, N0, T4 );
          U2:Work.Reg                  port map ( N8, N1, T3 );
          U3:Work.Adder                port map ( N13, N14, N7 );
          U4:Work.Multiplier           port map ( N12, N9, N8 );
          U5:Work.Reg                  port map ( N7, N9, T1 );
          U6:Work.Multiplexer          port map ( B, D, N10, T2 );
          U7:Work.Multiplexer          port map ( A, C, N11, T2 );
          U8:Work.Reg                  port map ( N7, N12, T2 );
          U9:Work.Multiplexer          port map ( N10, N1, N13, T4 );
          U10:      Work.Multiplexer    port map ( N11, E, N14, T4 );
          U11:      Control             port map ( CLK, T0, T1, T2, T3, T4 );
          U12:      Work.Buf            port map ( N0, Gout );
          U13:      Work.Buf            port map ( N1, Fout );

end STRUCTURE;

architecture BHV of CONTROL is
     begin
       process (CLK)
       variable state : integer range 0 to 4;
       begin
             if (CLK'event and CLK = '1') then
                   State := State + 1;
                   case State is
                when 0 =>
                   T0 <= '1'; T1 <= '0'; T2 <= '0'; T3 <= '0'; T4 <= '0';
                when 1 =>
                   T0 <= '0'; T1 <= '1'; T2 <= '0'; T3 <= '0'; T4 <= '0';
                when 2 =>
                   T0 <= '0'; T1 <= '0'; T2 <= '1'; T3 <= '0'; T4 <= '0';
                when 3 =>
                   T0 <= '0'; T1 <= '0'; T2 <= '0'; T3 <= '1'; T4 <= '0';
                when 4 =>
                   T0 <= '0'; T1 <= '0'; T2 <= '0'; T3 <= '0'; T4 <= '1';
                   State := 0;
             end case;
       end if;
       end process;
end BHV;
```

Figura 7.8 Código generado para el sistema sintetizado.

7.6 Otras pruebas.

A partir de la misma especificación inicial anterior, es posible obtener diferentes diseños, dependiendo de las restricciones que le agreguemos al sintetizador o de algún cambio en el algoritmo de planificación.

En este segundo ejemplo, se utilizó el algoritmo “force directed scheduling” para optimizar la utilización de los recursos y agregamos una restricción de tiempo. Esta restricción consiste en que el diseño generado debe terminar la evaluación de las expresiones en no mas de cuatro pasos de control.

El resultado obtenido fue la ruta de datos mostrada en la Figura 7.9. Como podemos observar, el diseño no excede de los cuatro pasos de control además de que optimiza los recursos utilizando, a diferencia que el diseño del ejemplo anterior, solo una unidad funcional de cada tipo. A cambio de esto, deben agregarse mas recursos de interconexión, pero la decisión de cual de los dos diseños utilizar depende de la experiencia del diseñador.

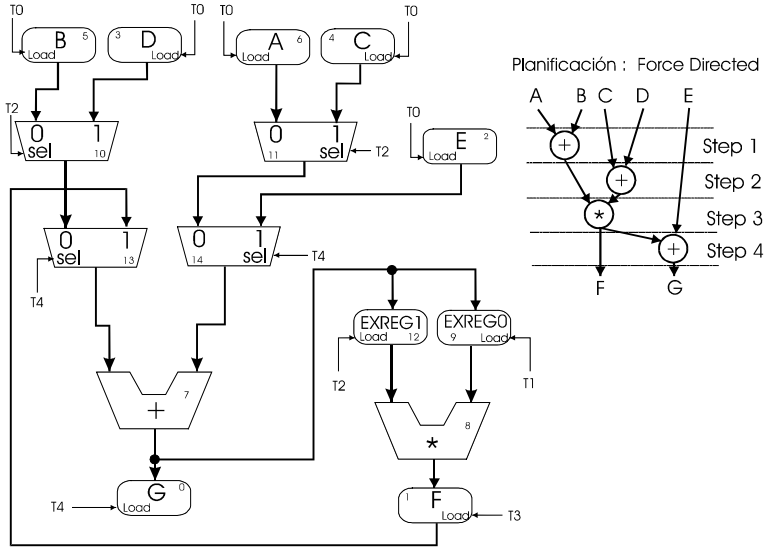

Figura 7.9 Diseño obtenido utilizando planificación FDS.

El código VHDL para este diseño, se muestra a continuación (Figura 7.10).

```
LIBRARY WORK;
Use WORK.ALL;

entity simple is
      port ( CLK    : in BIT;
             E, D, C, B, A    :  in integer;
             Gout, Fout    :  out integer);
end simple;

entity CONTROL is
      port ( CLK    : in BIT;
             T0, T1, T2, T3, T4    : out BIT );
end CONTROL;
```

```
architecture STRUCTURE of simple is

   signal N0, N1, N7, N8, N9, N10, N11, N12, N13, N14    :  integer;
   signal T0, T1, T2, T3, T4     : BIT;

   begin
          U1:Work.Reg            port map ( N7, N0, T4 );
          U2:Work.Reg            port map ( N8, N1, T3 );
          U3:Work.Adder          port map ( N13, N14, N7 );
          U4:Work.Multiplier     port map ( N12, N9, N8 );
          U5:Work.Reg            port map ( N7, N9, T1 );
          U6:Work.Multiplexer    port map ( B, D, N10, T2 );
          U7:Work.Multiplexer    port map ( A, C, N11, T2 );
          U8:Work.Reg            port map ( N7, N12, T2 );
          U9:Work.Multiplexer    port map ( N10, N1, N13, T4 );
          U10:      Work.Multiplexer    port map ( N11, E, N14, T4 );
          U11:      Control             port map ( CLK, T0, T1, T2, T3, T4 );
          U12:      Work.Buf            port map ( N0, Gout );
          U13:      Work.Buf            port map ( N1, Fout );

end STRUCTURE;

architecture BHV of CONTROL is
     begin
       process (CLK)
       variable state : integer range 0 to 4;
       begin
             if (CLK'event and CLK = '1') then
                   State := State + 1;
                   case State is
                when 0 =>
                   T0 <= '1'; T1 <= '0'; T2 <= '0'; T3 <= '0'; T4 <= '0';
                when 1 =>
                   T0 <= '0'; T1 <= '1'; T2 <= '0'; T3 <= '0'; T4 <= '0';
                when 2 =>
                   T0 <= '0'; T1 <= '0'; T2 <= '1'; T3 <= '0'; T4 <= '0';
                when 3 =>
                   T0 <= '0'; T1 <= '0'; T2 <= '0'; T3 <= '1'; T4 <= '0';
                when 4 =>
                   T0 <= '0'; T1 <= '0'; T2 <= '0'; T3 <= '0'; T4 <= '1';
                   State := 0;
             end case;
       end if;
       end process;
end BHV;
```

Figura 7.10: Código VHDL del diseño obtenido.

7.7 Resumen

Los diseños sintetizados fueron probados con éxito utilizando software de simulación. Específicamente se utilizó MAX PLUS II de ALTERA. Con estas simulaciones se comprobó que la funcionalidad de los diseños obtenidos era, efectivamente, la especificada en la descripción funcional inicial. Los diseños cumplían tanto los requerimientos de tiempo y de recursos. Se realizaron múltiples pruebas, algunas con sistemas muy grandes, sintetizándose satisfactoriamente.

CONCLUSIONES

El objetivo de esta tesis fué construir una herramienta de síntesis de alto nivel para generar automáticamente procesadores de expresiones aritméticas y lógicas.

Se abordo el problema desde la investigación del proceso de síntesis hasta la simulación de los procesadores sintetizados. A continuación se resumen las características de nuestra herramienta, en cada una de sus etapas.

Dominio de aplicación: Esta herramienta es útil en el diseño de sistemas digitales que permitan evaluar conjuntos de expresiones aritméticas y lógicas. Y genera un modelo de un sistema digital en un lenguaje de descripción de hardware, actualmente VHDL.

Descripción funcional: Debido al dominio de aplicación elegido para esta herramienta, se desarrolló un lenguaje de descripción funcional simple pero adecuado. En este lenguaje es posible describir grafos de flujo de datos en los que se incluyen operaciones y operandos. De manera inherente se representa la dependencia de datos y precedencia de operadores.

Compilador: El compilador construido toma la descripción del DFG, y lo transforma en una representación intermedia, basada en listas ligadas, sobre la cual actuarán las siguientes etapas de la síntesis de alto nivel.

Planificador: La etapa de planificación permite seleccionar tres algoritmos para estratificar el DFG en pasos de control. Uno de ellos es el algoritmo ASAP, que proporciona la planificación más rápida posible para el DFG. El segundo algoritmo es ALAP que, a diferencia de ASAP, proporciona la planificación más lenta posible para el DFG. El tercer algoritmo es FDS, que proporciona una planificación en la que se optimiza el uso de unidades funcionales mediante la reutilización, y además permite agregar restricciones de tiempo al diseño.

Mapeo: En esta etapa se realiza un mapeo de tipo constructivo. Es decir, se parte de una ruta de datos vacía y, en base a una arquitectura objetivo, se genera el control y la estructura de la ruta de datos. La arquitectura objetivo es una máquina de estados finitos con ruta de datos, que queda totalmente definida al final de esta etapa.

Generación de código VHDL: El proceso de síntesis ha generado hasta aquí un diseño que cumple con las especificaciones de la descripción funcional. Pero es necesario proporcionar una salida a partir de la cual se pueda construir el sistema digital diseñado. Esta herramienta de síntesis proporciona dicha salida en un lenguaje estándar para descripción de hardware; este lenguaje es VHDL. El modelo que proporciona el sintetizador puede ser simulado, o bien, ser implementado utilizando componentes discretos o FPGAs.

Los objetivos de esta tesis se cumplieron de manera satisfactoria. Los diseños generados no fueron construidos, pero si fueron simulados en MAX PLUS II de ALTERA. Funcionalmente, el diseño y su especificación inicial son equivalentes.

Bajo una misma especificación inicial, es posible generar varios diseños, ya que si se agregan restricciones de algún tipo o se cambian los algoritmos de planificación, se generará un diseño distinto, pero funcionalmente equivalente.

Existen varias extensiones a esta tesis que pueden desarrollarse como trabajos posteriores. En primer lugar, esta herramienta de síntesis no considera restricciones en espacio. Esto puede desarrollarse mediante un archivo de restricciones, en el cual se especifique, para cada tipo de operación, cuantas unidades funcionales como máximo deben utilizarse. Estas restricciones serán consideradas dentro de la planificación ya que en esta etapa deberá resolverse la concurrencia de operaciones para realizarlas con las unidades funcionales disponibles. Esto involucra menor paralelismo y mayor serialización de operaciones, lo cual resulta en un mayor número de pasos de control.

También existe trabajo a desarrollar en cuanto a la descripción funcional. En HLSynth, la creación de la descripción funcional requiere un trabajo previo por parte del diseñador, quien debe construir el DFG, numerarlo y después codificarlo. Es importante proporcionar un lenguaje de más alto nivel en el que se puedan escribir las expresiones aritméticas y lógicas de forma directa. Debe definirse su gramática y después crear un compilador que a partir de esa representación genere la misma representación intermedia que se utiliza en el sintetizador.

Si se desarrolla un lenguaje de alto nivel para la descripción funcional, será posible ampliar el dominio de aplicación de la herramienta, haciéndola más general.

Algo importante que debe considerarse en futuros trabajos es el cambiar la arquitectura objetivo de la herramienta. HLSynth utiliza multiplexores como unidades de interconexión, pero un diseño más eficiente utilizaría buses para interconectar las unidades funcionales y de almacenamiento. El empleo de buses en la arquitectura objetivo eficienta la red de interconexión, pero implica una mayor complejidad en la unidad de control, además de que los dispositivos de la ruta de datos deben soportar el estado de “alta impedancia” para compartir el bus y evitar cortos circuitos.

REFERENCIAS:

[1] Ahmed Amine Jerraya, Hong Ding, Polen Kission, Maher Rahmouni.
"Behavioral Synthesis and Component Reuse With VHDL".
Kluwer Academic Publishers, USA 1997.

[2] James R. Armstrong, F.Gail Gray.
"Structured Logic Design with VHDL".
Prentice Hall, 1993.

[3] Arturo Diaz-Pérez, Guillermo Morales-Luna y Michael J. Quinn.
"Generating VHDL Behavioral Models From Space-Time Descriptions".
Reporte Interno, CINVESTAV-IPN, Octubre 1997.

[4] Arturo Diaz-Perez, Michael J. Quinn y Guillermo Morales Luna.
"The Uniformization of Recurrence Equations".
II Congreso de Ingeniería Eléctrica, CINVESTAV-IPN. Agosto 7, 1996.

[5] Daniel Gajski, Nikil Dutt, Allen Wu, Steve Lin.
"High-Level Synthesis, Introduction to Chip and System Design".
Kluwer Academic Publishers, 3ª. Edition, USA 1992.

[6] Siram Govindarajan.
"Scheduling Algorithms for High-Level Synthesis".
Term Paper, ECE834. Dept. Of ECECS, University of Cincinnati Cincinnati, OH 451-0030
March 17, 1995.

[7] Jesús Hernández Tapia, Arturo Díaz-Pérez y Adriano de Luca.
"Síntesis de Alto Nivel para Sistemas Digitales".
4ª. Conferencia de Ingeniería Eléctrica. CINVESTAV-IPN.
Septiembre de 1998, México D.F.

[8] Jesús Hernández Tapia, Arturo Díaz-Pérez y Adriano de luca.
"Algoritmos de Planificación para Síntesis Funcional".
5ª. Conferencia de ingeniería eléctrica. CINVESTAV-IPN.
Septiembre de 1999, México D.F.

[9] Michael C. McFarland, Alice C. Parker y Raul Camposano.
"The High-Level Synthesis of Digital Systems".
Proceedings of the IEEE, Vol. 78, No.2, February 1990, pp. 301 - 318.

[10] P. Paulin y J. Knight.
"Algorithms for High-Level Synthesis".
IEEE Design and Test of Computers, December 1989. pp 18 – 31

[11] Pierre G. Paulin y John P. Knight.
"Force-Directed Scheduling for the Behavioral Synthesis of ASIC´s".
IEEE Transactions on Computer -Aided Design. Vol. 8. No. 6. June 1989.

[12] Robert A. Walker y Samit Chaudhuri.
"Introduction to the Scheduling Problem".
IEEE Design and Test of Computers, Summer 1995. pp 60 –69

[13] Robert A. Walker, Raul Camposano.
"A Survey of High-Level Synthesis Systems".
Kluwer Academic Publishers, USA 1991.