UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

TESIS DOCTORAL

MODELADO Y GENERACIÓN FLEXIBLE DE NÚCLEOS DE PROCESADOR SPARC PARA APLICACIONES ESPECÍFICAS

TOMÁS BAUTISTA DELGADO

Las Palmas de Gran Canaria, 1999

55/1998-99

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
UNIDAD DE TERCER CICLO Y POSTGRADO

Reunido el día de la fecha, el Tribunal nombrado por el Excmo. Sr. Rector Magfco. de esta Universidad, el/a aspirante expuso esta TESIS DOCTORAL.
Terminada la lectura y contestadas por el/a Doctorando/a las objeciones formuladas por los señores miembros del Tribunal, éste calificó dicho trabajo con la nota de SOBRESALIENTE 'CUM LAUDE' POR UNANIMIDAD

Las Palmas de Gran Canaria, a 2 de julio de 1999.

El/a Presidente/a: Dr. D. Roberto Sarmiento Rodríguez,

El/a Secretario/a: Dr.D. Juan Antonio Montiel Nelson,

El/a Vocal: Dr.D. Mateo Valero Cortés,

El/a Vocal: Dr.D. Leopoldo García Franquelo,

El/a Vocal: Dr.D. Adrianus Cornelius Verschueren,

El Doctorando: D. Tomás Bautista Delgado,

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
DOCTORADO EN INGENIERÍA DE TELECOMUNICACIÓN
DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA
PROGRAMA DE INGENIERÍA ELECTRÓNICA

Modelado y Generación Flexible de Núcleos de Procesador SPARC para Aplicaciones Específicas

Tesis Doctoral presentada por:
TOMÁS BAUTISTA DELGADO

Dirigida por el Doctor:
D. ANTONIO NÚÑEZ ORDÓÑEZ

El Director,

El Doctorando,

Tomás Bautista

Las Palmas de Gran Canaria, julio de 1999.

A mi madre, mi padre
Eli y Sofi.

Sumario

En esta tesis se ha establecido una metodología basada en VHDL para estudiar de forma cómoda y versátil la incidencia de diferentes estrategias microarquitecturales y de diseño en las prestaciones finales de un procesador generado como un núcleo para sistemas integrados en un chip. Esta metodología se apoya en el empleo eficiente de un entorno gráfico de modelado. Esta eficiencia se orienta principalmente a facilitar el control de los diferentes detalles de la microarquitectura, tanto en la concepción y el modelado como en la fase de síntesis lógica y física. Con frecuencia los entornos de diseño han concentrado su capacidad de análisis en los niveles arquitecturales, lo que se ha demostrado insuficiente en núcleos destinados a sistemas empotrados.

Empleando esta metodología se ha desarrollado un modelo de un núcleo de una Unidad de Enteros de un Procesador SPARC, de sus extensiones VIS, y de la Unidad de Coma Flotante. Partiendo de estos modelos se han desarrollado distintas versiones con idéntica Arquitectura de Juego de Instrucciones pero con pequeñas variaciones tanto microarquitecturales como de diseño. De esta forma, y gracias al flujo de diseño establecido, ha resultado sencillo evaluar el impacto de las cuestiones de diseño que intervienen en estas versiones. Esta evaluación se realiza en forma de análisis de sensibilidad.

Al realizarse la medida de los parámetros característicos de las síntesis físicas resultantes se ha puesto de relieve la importancia que tiene el haber establecido los mecanismos para determinar la influencia que presenta cada decisión de diseño. Esta relevancia se ha manifestado en la gran dispersión de prestaciones y en la naturaleza heterogénea de las características que han resultado en los núcleos generados. Se aumenta así la visibilidad del espacio de diseño a explorar, su dispersión y sus causas. El análisis cuantitativo ha permitido establecer la relación entre el diferente comportamiento de las propiedades estudiadas y las distintas estrategias de diseño empleadas.

Las experiencias demuestran la necesidad de integrar de forma más interactiva las etapas de diseño microarquitectural, lógico y físico con las etapas arquitecturales de más alto nivel. Las ayudas al flujo de diseño aportadas y las experiencias de análisis de sensibilidad permiten diseñar con óptimas prestaciones núcleos procesadores para sistemas empotrados. El caso de núcleos SPARC ha sido el referente tomado y sobre el que se ha hecho un completo análisis con más de cien implementaciones generadas. El campo de aplicaciones de SPARC con extensiones VIS, tomado como referencia en este trabajo, ha sido el de procesado de vídeo comprimido, según estándares de bajo y medio régimen binario.

Agradecimientos

El hombre es él y sus circunstancias.
José Ortega y Gasset.

Esta página presidida por el epígrafe "Agradecimientos" no pretende ser más que un reflejo de la gratitud que diariamente se siente ante el apoyo de tanta gente. Sin ese apoyo no hubiera sido posible ni la mitad de lo que en este documento se expone. Sin embargo, la limitación del espacio y tiempo hace que esta intención de expresar mi gratitud no llegue a conseguirse en su plenitud, que se corra el riesgo de no poder exponer cuanto se desea o cometer alguna omisión de forma involuntaria. No en vano se hace necesario recordar que en cualquier trabajo donde se emplea tanto esfuerzo existen momentos de alegría y momentos de menos alegría, y que tanto en un caso como en otro cualquier apoyo siempre es de agradecer. Desde estas pocas líneas quisiera agradecerles a todos mi más sincero agradecimiento y reconocimiento.

Agradezco a Antonio, quien se ofreció para dirigirme, su empuje firme y decidido durante todo este tiempo, así como el que me haya transmitido parte de sus conocimientos para ayudarme a llegar hasta este punto. Muchas gracias por confiar en mí para desarrollar esta tarea, de la que tanta experiencia he podido extraer.

Agradezco a Pedro su disponibilidad, apoyo y amistad, sin lo cual me hubiese costado salir de más de algún atolladero. También agradezco el apoyo de Gustavo, quien además me ha ayudado a lanzarme en este trabajo de forma incuestionable y siempre ha encontrado palabras para animar. Sería injusto si me olvidase de mencionar a Enrique y Jesús, quienes siempre han estado ahí para garantizar que las cosas funcionen.

También quisiera expresar mi reconocimiento público a los compañeros de trabajo, con los que siempre ha existido una estupenda camaradería, y entre los que, con la convicción de continuo aprendiz irremediable, incluyo a antiguos y actuales estudiantes. A los compañeros actuales y a los que han pasado por el CMA (hoy IUMA), la ETSIT

y la EUITT, como Toni, Loli, Octavio, Valentín de Armas, Roberto Esper-Chaín, Juan Antonio Montiel, Margarita, Alfonso, Roberto Sarmiento, José López, y un largísimo etcétera. Y también quiero agradecer su apoyo, no menos importante y sentido, a mis familiares y a los amigos externos a este entorno universitario.

Y sobre todas las cosas quisiera agradecer a mi madre Eli y a mi padre Agustín las tantísimas horas de dedicación para ayudarme a ser quien deseaba ser y concederme su incondicional apoyo, así como a mi hermana Eli y a mi novia Sofi. Al gozar del apoyo y estima de Vds. uno se siente realmente afortunado y privilegiado.

El trabajo que aquí intento exponer lo dedico a todos Vds. en justa, modesta y merecida consideración.

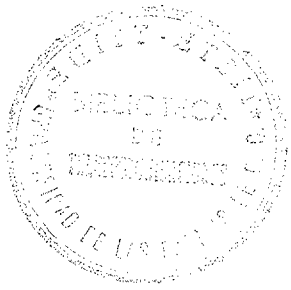
Tomás.
Julio de 1999.

Índice

Índice de Figuras

Índice de Tablas

CAPÍTULO 1



Resumen

1.1. Planteamiento general

El constante aumento de la densidad de transistores que ofrece hoy la electrónica integrada en un circuito monolítico ha abierto la investigación de nuevos paradigmas de diseño que permitan hacer un uso eficiente de esta mayor densidad.

Estos paradigmas incluyen el diseño de grandes circuitos de aplicación específica sintetizados con modernas herramientas de diseño, el codiseño que parte de la descomposición del problema en tareas que se ejecutan por circuitería específica y tareas que se ejecutan por programas sobre procesadores sintetizados *ad hoc*, el codiseño basado en programas sobre procesadores estándares y coprocesadores y extensiones *ad hoc*, o el diseño basado sobre programas ejecutados por procesadores de juego de instrucciones específicos. Simplificamos estos grandes enfoques con los términos ASIC (*Application Specific Integrated Circuit*), codiseño, SOC (*System on a Chip*) y ASIP (*Application Specific Integrated Processor*) o ASISP (*Application Specific Instruction Set Processor*).

Con todos estos enfoques pueden realizarse grandes sistemas empotrados o embebidos [Ern98, Ern97, GVNG94, LSVH96, GZ97, Pay], que con tecnología convencional se realizaban hasta el presente mediante una integración de componentes sobre placas PCB o de módulos MCM. Sin embargo, el enfoque que supone un traslado más directo de la integración PCB a la integración sobre un chip es el que hemos denominado SOC [Fos99].

El diseño de sistemas empotrados SOC se basa en la disponibilidad en *almacén* de componentes previamente diseñados, *componentes virtuales*, que puedan ser reutilizados. Estos componentes pueden ser componentes físicos, es decir, diseños físicamente trazados sobre un proceso tecnológico determinado (*hard cores* y *hard components*), o

pueden ser componentes lógicos (*soft cores* o *soft components*), es decir, modelos lógicos completamente elaborados y verificados, pendientes de su mapeado y personalización sobre el proceso tecnológico elegido como material de integración.

Ambos tipos de componentes serán propiedad intelectual o industrial de las entidades que tengan esos derechos. Su integración en un chip requiere el diseño, el acceso o la compra de esos componentes. En el esquema SOC, los componentes más importantes reciben el nombre de IPs (*Intelectual Property*). Estos IPs pueden haber sido desarrollados por el mismo integrador o por terceros. En este último caso es aún más importante la caracterización del componente IP, de forma que su integración sea fiable [PHC99, AL99, Haa99].

La integración de componentes requiere la definición de sus interfaces, de sus entradas y salidas de datos y de interconexión funcional y de alimentación. En la actualidad hay avances, aunque limitados, en la definición de estándares de interfaces virtuales para componentes virtuales.

La integración de componentes físicos es más sencilla puesto que están completamente terminados y caracterizados para la tecnología disponible, y se pueden realizar simulaciones integradas precisas. Sin embargo, este tipo de ingeniería de componentes encuentra grandes limitaciones, en la práctica, para diseñadores terceros, equivalentes a los tradicionales OEMs (*Original Equipment Manufacturers*) que ahora reaparecen en versión OCMs (*Original Chip Manufacturers*). Es más común esta ingeniería de integración de componentes físicos en el seno de empresas de sistemas con fundición propia o establecida.

Las empresas de diseño y las de sistemas sin fábricas o sin fundiciones tienen más libertad en el mapeado tecnológico y necesitan flujos de diseño e implementación más portables en tecnología, con implementaciones rápidamente caracterizables y con tiempos de síntesis cortos.

1.2. Objeto de la tesis

El paradigma SOC se está revelando como el más directo beneficiario de la creciente densidad de integración. Esto es fácilmente comprensible por ser el heredero más directo de la integración de componentes reales sobre PCB. Su potencial está también directamente relacionado con la disponibilidad de potentes componentes microprocesadores y coprocesadores virtuales, especialmente en forma de componentes físicos (*hard cores*). Los mayores esfuerzos de investigación actuales en el esquema SOC se dirigen por un lado a resolver la estandarización de los interfaces [See99, Bak98, VSI97a, BdKK+99, Fai98, RSV97, VSI] y por otro a flexibilizar la integración de componentes virtuales [VG99, Fos99], independizando más el proceso de integración respecto de los procesos tecnológicos, y estableciendo el paradigma SOC con componentes virtuales lógicos [GZ97, Pay, GDWL91].

El área clave de trabajo consta de varios problemas:

1. Desarrollar amplias librerías de componentes virtuales lógicos de tipo estándar en lenguajes de descripción de *hardware* HDL, y especialmente en VHDL y Verilog [RR99, BBS98, DJ99].

2. Desarrollar y caracterizar modelos HDL de grandes núcleos procesadores y coprocesadores, y en particular de los microprocesadores y DSPs líderes del mercado. Esto está en relación con la ventaja diferencial del esquema SOC de no requerir el desarrollo de compilador *ad hoc* para nuevos juegos de instrucciones como ocurre en los esquemas de codiseño y ASIP, y así permitir que el diseñador trabaje a alto nivel con lenguajes estándares como C o incluso a niveles funcionales superiores (MATLAB y otros) [JDKR97, LWMG96, DJ99].

3. Desarrollar técnicas de modelado HDL con integración de componentes virtuales lógicos y simulación integrada, con precisión de ciclo de reloj, previo al mapeado [KCGW98, KB98].

4. Desarrollo de técnicas y entornos de síntesis física, que permitan la integración física de componentes con esquemas de síntesis/integración o de integración/síntesis y que permitan una completa exploración y caracterización del espacio de diseño, con precisión circuital de las prestaciones en tiempo de ciclo, área y consumos de potencia. Para ello se requiere crear nuevos flujos de modelado y generación flexible de componentes y de sistemas integrados compuestos por estos componentes [OAIP98, BAMS98, PHSMR95].

El presente trabajo contribuye a la solución de los tres últimos problemas (2, 3 y 4). El problema 1 va encontrando rápidamente soluciones impulsadas por el mercado y es de menor complejidad. Ejemplos de empresas que ofrecen un catálogo amplio de núcleos IP son, entre otras, *Alta Group*, *VAutomatic, Inc.*, *Virtual Chips*, *Intronic*, *Design & Reuse*, o *Thomson CSF*. En las páginas amarillas de *Design & Reuse* aparecen 98 empresas y más de 1300 IPs. En España podemos mencionar SIDSA y algún grupo universitario [RCS99, VSI97b, BBS98, BS97, BBK+97, BKS98, BBA+98, But95]. Un interesante ejemplo de núcleos IP para GSM desarrollado por *Alta Group* utilizando el entorno de muy alto nivel *Felix*, de Cadence, se ofrece en [CM99a].

1.3. Campos de aplicación

Para fijar con precisión el marco experimental y de investigación de esta tesis, se ha seleccionado un campo de aplicaciones donde la demanda de soluciones SOC se espera que tenga un crecimiento explosivo en los próximos años: aplicaciones específicas, basadas en procesadores con capacidad multimedia, y en especial capacidad de procesar flujos MPEG en distintos formatos [TM96, MS98, vdWLG+99, KPS+99]. Además, la mayor demanda se prevé en el lado del usuario, fundamentalmente en decodificación y presentación de estos flujos [dL96, TK96b, TK96a], y también en codificación pero a bajos regímenes binarios [Bjo96, TF96]. Para este tipo de aplicaciones las soluciones basadas en tareas *software* —soluciones de programación— son dominantes [SL97, Eck95, Lee96, Pat93, PWW97, TONH96, Sun, CGK, Int, Lap97, PW97].

La misma razón del éxito del paradigma SOC para sistemas empotrados lleva a basar los diseños en procesadores estándares, y no en procesadores específicos. Esto plantea la cuestión de la disponibilidad de tales procesadores. Afortunadamente los principales diseñadores de procesadores están respondiendo a los nuevos mercados con soluciones cada vez más abiertas y audaces respecto a la disponibilidad de sus productos.

Así, la práctica totalidad de los proprietarios de IPs de procesadores buscan activamente segundos fabricantes y conceden derechos de producción como segundas fuentes. Aún más, arquitecturas y familias de productos como ARM, MIPS o SPARC han nacido con el propósito explícito de licenciar la tecnología a fabricantes de modo completamente general, lo que los ha convertido en estándares industriales. El caso extremo en esta tendencia lo representa *Sun Microsystems* con su arquitectura SPARC. SPARC es una arquitectura verdaderamente abierta, en el sentido de libre, para cuya implementación individual basta la obtención de la correspondiente certificación por *SPARC Intenational*, una organización sin ánimo de lucro, y que ha propuesto SPARC como un estándar IEEE. *Sun Microsystems* ha anunciado la disponibilidad gratuita de modelos Verilog de las arquitecturas SPARC y PicoJava, entre otros diseños. Este paso se basa en el éxito y gran futuro del paradigma SOC sobre componentes IP virtuales antes comentado.

Existen otras muchas áreas de aplicación de interés, como pueden ser los microcontroladores industriales, los microcontroloadores embarcados de la industria del automóvil, los DSP embarcados de módems y equipos de telecomunicaciones de las redes de acceso y los equipos de datos portátiles y móviles *palm-top*.

En particular, tanto las aplicaciones multimedia como la electrónica portátil serán las de mayor impacto y conducirán a un nuevo dominio de aplicaciones y un mercado nuevo en el futuro reciente [KP98]. El procesador que se necesitará para estos dispositivos de computación móviles en realidad será un híbrido entre un procesador de propósito general y un procesador digital de señales. Estos microprocesadores deben cumplir cuatro requisitos principales:

- altas prestaciones para funciones multimedia,
- eficiencia en la energía y potencia,
- pequeño tamaño y
- baja complejidad de diseño.

Las técnicas y entornos de diseño desarrollados en la tesis, el banco de experimentación y los principales resultados de esta tesis, construidos en torno a SPARC y orientados a sus aplicaciones multimedia, no dependen estrictamente ni del procesador elegido ni del área elegida de aplicación y son fácilmente generalizables a otras áreas.

1.4. Metodología

El proceso de síntesis de una implementación se resume en trasladar una arquitectura (e.gr., SPARC v. 8 IU) definida a alto nivel, hasta el nivel físico de implementación

sobre un proceso tecnológico determinado y con restricciones de área, aspecto, consumo, velocidad, interconexión con otros componentes, test o precio.

El factor clave al tratar de dar solución a los tres problemas indicados es la gran incidencia que tienen el modelado y la síntesis sobre las prestaciones del componente físico obtenido. Dicho de otra manera, el propio concepto de *componente virtual lógico* está tan abierto que en su paso a componente físico sintetizado las opciones del flujo de diseño son tan grandes que se generan resultados muy dispersos en prestaciones.

Es necesario analizar en detalle, de forma cualitativa y cuantitativa y —en un contexto de producción más amplio al de este trabajo— incluso de forma estadística y con técnicas de análisis de sensibilidad, el impacto que tienen sobre las prestaciones finales de los componentes físicos los siguientes niveles del flujo de diseño SOC basado en modelos de componentes virtuales:

1. Diseño de la microarquitectura del componente. Una arquitectura definida permite numerosas opciones al nivel microarquitectural RTL, tanto en las rutas de datos como en el flujo de control.

2. Diseño lógico de cada opción microarquitectural, que a su vez permite numerosas opciones lógicas al nivel de puertas lógicas y transistores.

3. Diseño físico de cada solución lógica. Este nivel incluye diversas opciones para el colocado, el ruteado y los interfaces primarios de la lógica del sistema.

4. La composición final del SOC, ya sea mediante composición de componentes físicos, ya sea mediante combinación de componentes lógicos, posteriormente sintetizados.

Este proceso es viable mediante el uso extensivo de las herramientas CAD disponibles en todos esos niveles, su conexión, y la investigación en optimizar su uso para el modelado y generación flexible de componentes IP para SOC de aplicaciones específicas. Así:

1. En primer lugar, se ha trabajado en técnicas de modelado en VHDL ideando métodos de descripción adecuados para realizar modificaciones y optimizaciones microarquitecturales.

2. En segundo lugar, se han encontrado métodos estructurales basados en concepciones gráficas mediante bloques de los elementos del diseño y se han idenfiticado como los más idóneos para entornos de generación flexible de núcleos procesadores.

3. Se ha hecho un uso completo de las posibilidades de definición y generación de módulos en la síntesis lógica, y de agrupación de lógica en bloques fijos.

4. Se han usado las posibilidades de las herramientas lógica y física para definir y estudiar distintos niveles de granularidad lógica y física en el diseño y su impacto en las prestaciones.

5. Se ha analizado la forma de los núcleos, y su adaptación para la interconexión con otros núcleos mediante el concepto de *plasticidad*.

6. Se han creado medios de caracterización de las implementaciones obtenidas (más de 100) y se han estructurado en tablas, gráficas y medidas estadísticas para facilitar su análisis y la toma de decisiones.

1.5. Desarrollo del modelado de núcleos IP de SPARC

El contenido fundamental de los componentes IPs utilizados ha sido las unidades de la arquitectura SPARC v. 8 correspondientes a la Unidad de Enteros, la Unidad de Coma Flotante, y las extensiones del juego de instrucciones VIS para multimedia.

Estos componentes IPs han sido desarrollados en VHDL por nosotros mismos en este trabajo. El desarrollo ha estado presidido por la finalidad de síntesis e integración SOC en un entorno de diseño y banco experimental para generación flexible de componentes y núcleos físicos.

Una parte sustancial del trabajo ha estado centrada en la obtención de un buen diseño base, de referencia, para los componentes virtuales de SPARC, en especial, y por su importancia, de la Unidad de Enteros y de las extensiones VIS.

Los resultados de esta tesis matizan el aforismo clásico entre arquitectos: 'cuanto más altos sean los niveles de concepción y flujo de diseño de un circuito en los que se introducen mejoras, mayores son los beneficios obtenidos'. Lo matizan en el sentido de que la libertad de síntesis y mapeado tecnológico aportada por los entornos de síntesis, y el gran impacto de la microarquitectura, diseño lógico y diseño físico en este proceso, hacen más necesaria una estrecha unión y entrelazamiento entre el proceso de concepción y proyectación arquitectural y el de implementación física. La relevancia de muchas decisiones microarquitecturales queda sustancialmente transformada, condicionada o filtrada por los niveles inferiores del proceso de implementación, de forma que no se pueden tomar esas decisiones sin tener en consideración estos efectos de filtrado. Este efecto va en aumento debido a la creciente importancia del ruteado y las interconexiones frente a la lógica, según decrece la geometría de los procesos.

Sin embargo, esta tesis no cuestiona la necesidad de realizar el máximo esfuerzo de optimización en cada nivel, empleando para ello los métodos propios de cada nivel, sino que refuerza la idea de que no pueden ser esfuerzos aislados. Consecuentemente estos métodos se han aplicado en cada uno de los niveles y en particular en el modelado, microarquitectura y diseño del procesador, y se han creado opciones de configuración para observar sus efectos en el diseño físico.

En este aspecto se ha combinado un flujo de síntesis natural *top-down* y una concepción *bottom-up* más cercana al diseñador y más adecuada para la aplicación de técnicas de optimización. Para ello se han creado elementos clásicos de librería a nivel RTL, y se ha hecho uso del entorno gráfico SGE de Synopsys y de técnicas de parametrización y configurabilidad de los elementos.

Las opciones microarquitecturales han sido investigadas para la ruta de datos en sus buses, multiplexores y *bypasses*, sus etapas de segmentación y su equilibrado, los tipos de registros de segmentación más adecuados, los módulos operadores y de fichero de

registros y los mecanismos de direccionamiento en las ventanas de registros. Todas estas opciones se han configurado para los experimentos.

Igualmente, y respecto al control, se ha investigado una generación de señales de control centralizada en una FSM, frente a técnicas distribuidas, y una unidad de secuenciamiento que gestiona el cauce segmentado, el flujo secuencial de instrucciones y las transferencias de control. Entre estas últimas se han investigado las opciones para gestionar el tratamiento de saltos, con diversas predicciones, y se han dispuesto configuraciones para los experimentos.

El control ha incluido opciones para la gestión del cálculo de direcciones y de los contadores de programa, y de la gestión de excepciones.

La interpretación microarquitectural resultante, según se resume en la tabla de ciclos de ejecución del juego de instrucciones, sitúa la versión base de referencia desarrollada entre las más avanzadas de la industria.

1.6. Herramientas de ayuda en el flujo de diseño

El segundo esfuerzo sustancial de la tesis ha sido el establecimiento del entorno de exploración del espacio de diseño y experimentación para la generación flexible de los núcleos SPARC.

El entorno está compuesto de:

1. Modelos VHDL de componentes y modelos VHDL de módulos.
2. Librerías de síntesis lógica.
3. Librerías de síntesis física.
4. Diagramas de bloques en SGE de Synopsys.
5. Generación de módulos en Epoch de Duet Technologies.
6. Simulación RTL *pre-layout.*
7. Síntesis lógica en Design Analyzer de Synopsys.
8. Síntesis física en Epoch.
9. Simulación *post-layout.*
10. Herramientas de toma de medidas, prestaciones y caracterización de versiones.
11. Conjunto de *scripts* —muchos de ellos en PERL 5— de automatización o ayuda a las transisciones en el flujo de diseño.
12. Criterios, recomendaciones y documentación de ejemplos para el diseñador.

En este flujo de diseño la herramienta Epoch [Cas97] tiene muchas ventajas en la generación rápida de *layouts*. No obstante, en el futuro es necesario evaluar los recientes avances introducidos por Cadence en su arquitectura *Layout Acceleration Environment* LAE, extensión de Virtuoso [MGJK99]. Y también su eventual combinación con herramientas de modelado de alto nivel de IPs como *Felix* [RSV98]. Entre estas herramientas y técnicas, no nos parecen de importancia secundaria las técnicas establecidas para la sistematización de experiencias, la toma de medidas, la denominación de subdirectorios y ficheros, la codificación y tabulación de versiones, o los *scripts* de ayuda. Todo este entorno ha permitido obtener más de 100 implementaciones físicas, plenamente caracterizadas, de los núcleos de procesador SPARC. Este elevado número de diseños aporta un alto grado de confianza a las conclusiones extraídas del análisis cuantitativo de resultados. El banco experimental así creado es general para analizar cuantitativa y cualitativamente otro tipo de núcleos IP procesadores, y su integración con otros componentes en sistemas SOC.

1.7. Experiencias y análisis de resultados

Mediante las técnicas de modelado descritas, los modelos desarrollados de componentes de SPARC, y el entorno o banco experimental establecido, se han realizado numerosas experiencias de obtención de implementaciones físicas de esos núcleos SPARC que de hecho cubren un gran espacio de diseño. Los núcleos están destinados a la construcción de SOCs de bajo costo en aplicaciones específicas.

Como se ha indicado, la clave de este trabajo es poder generar rápida y flexiblemente esos núcleos y caracterizarlos, para analizar la incidencia que tienen sobre las prestaciones las opciones arquitecturales, microarquitecturales, estructurales, lógicas, de agrupación de bloques, de colocado, de test, de conexión con otros núcleos o componentes, y finalmente las opciones tecnológicas.

El análisis de resultados de las opciones microarquitecturales puede agruparse de dos formas. En primer lugar, estudiando el impacto no teórico o simulado sino real de las opciones sobre las prestaciones medidas en los *layouts* físicos implementados. En segundo lugar, estudiando en qué medida la mejora relativa de cada opción *resiste* hasta el final del proceso de implementación, es decir, en qué medida la opción es *dominante* respecto a algún parámetro de prestaciones, y de ahí conocer a ciencia cierta la importancia mayor o menor de considerar esa opción al nivel microarquitectural.

Se han estudiado las opciones de predicción de salto, de gestión de códigos de condición, de modo de cálculo de la dirección de búsqueda siguiente, de los mecanismos posibles de *bypass*, de las estructura de módulos y operadores de la ruta de datos.

Los experimentos demuestran que las opciones de cálculo de la dirección de la siguiente instrucción es dominante en velocidad, y en menor medida las de predicción de salto. Son óptimas las opciones de *dos sumadores de direcciones* y *predecir el salto como que se tomará cuando es hacia atrás*. El resto de opciones sufren una gran dispersión en la implementación. En general cuanto más se *delegan* las decisiones sobre cada uno de los elementos que influyen en el secuenciamiento, más rápida resulta la implementación.

Un ejemplo de la problemática que se analiza es el hecho de que dos núcleos de SPARC con todo tipo de opciones idénticas, a nivel microarquitectural, lógico o físico, con idénticas opciones de las herramientas, y con la única variante microarquitectural *pequeña* de predecir los saltos como que se tomarán siempre frente a predecir que se tomarán sólo si son hacia atrás, genera *mejores* núcleos (en cuanto más veloces) en las implementaciones sobre 0.50 μm, pero *peores* (más lentos) en las implementaciones sobre 0.35 μm. Una vez más se debe resaltar la necesidad cada vez mayor de integrar los análisis arquitecturales de niveles altos con los análisis físicos de los niveles bajos, con herramientas más integradas.

Otro ejemplo es observar que la técnica más común de situar la circuitería de *bypass* al principio de la etapa de ejecución puede bajar la velocidad y aumentar la potencia, en especial en situaciones por otro lado óptimas, con un empleo *medio* de módulos.

Las experiencias sobre el grado de granularidad lógica y uso de módulos de ruta de datos demuestran que un empleo razonable de módulos en zonas concretas del diseño suele mejorar las prestaciones en velocidad y área. Sin embargo, las versiones con mayor granularidad lógica, y más módulos, no continúan la mejora de velocidad, ni de área, ni de consumo de potencia. Existen, por tanto, combinaciones *medias* óptimas que es preciso explorar. Otro dato importante es la significativa variación en prestaciones que aparecen entre las versiones, su gran dispersión.

Las experiencias sobre granularidad física y definición de grupos demuestran que este nivel de agrupación y jerarquización física, de uso frecuente para *guiar* el colocado y ruteado, raramente compensa. El área y el consumo de potencia aumentan. Es notable, sin embargo, que la velocidad no se deteriore tanto como el aumento de área hace prever. En este caso, el proceso de deterioro queda frenado por la mayor proximidad entre puertas funcionalmente relacionadas, obtenida con el guiado de la herramienta. El número de transistores tampoco empeora mucho. Este resultado podría conducir al diseñador no experimentado a efectuar modificaciones lógicas jerárquicas alegremente: su impacto a veces no es perceptible en transistores pero es muy grande en velocidad y área.

Otro efecto similar se obtiene fijando los bloques jerárquicos estableciendo el atributo `fixedblock` al valor 1. Ni el número de transistores, ni la velocidad quedan apenas modificados, pero sí el resto de parámetros. El efecto es tanto mayor cuanto más *gruesa* es la granularidad física.

Otras variantes como el número de ventanas de registros o el impacto de la inserción de cadenas de test es muy nítido y lineal, como cabe de esperar.

La integración SOC con componentes de memoria y de coma flotante produce también importantes alteraciones en las prestaciones. Un resultado de interés es el mejor compartimiento obtenido con la combinación y conexión de componentes previa a la síntesis física, frente a una composición y conexión posterior. En el primer caso los componentes demuestran las ventajas de una mayor *plasticidad*, que aprovecha mejor la inteligencia de las herramientas en las interconexiones. Sin embago, las experiencias también demuestran que las prestaciones son mejores si, además, se escogen y fijan antes de la síntesis las opciones más deseables para el núcleo del procesador, según resulte de

su caracterización previa en el espacio de diseño.

La extensión del juego de instrucciones y de la ruta de datos para soportar operaciones de procesado de señal (MAC, DSP) y multimedia (SIMD, VIS) tiene unas notables repercusiones en todos los parámetros. Potencia y número de transistores crecen quizás más drásticamente que el área. El tiempo de ciclo aumenta significativamente —entre un 20 y un 40 %—, por lo que sólo se justifican estas extensiones cuando el campo de aplicación, por la composición de las cargas de trabajo de los programas, así lo demanden, y en este caso parece claro que cuanto mayor sea la longitud del paralelismo vectorial SIMD del procesador (8 operandos frente a 4, etc.) mejor se compensa en el *throughput* la mayor latencia física provocada. Por el contrario, el costo de estas extensiones en cuanto a su control y gestión es muy pequeño. Estos resultados son coherentes con otros estudios teóricos [PM98, Asa98].

Finalmente la influencia del mapeado y síntesis en diferentes tecnologías sobre todas las opciones anteriores es también obviamente muy importante.

Los experimentos confirman en general las conocidas reglas de escalado. Al evolucionar de una tecnología de una longitud de canal determinada a otra de menor longitud, se mejoran todas las propiedades, incluso la del número de transistores. Sin embargo, es importante señalar —ya se ha hecho antes— que el peso relativo de las mejoras de cada opción en una tecnología no tiene porqué mantenerse en general en otra, incluso cambia de signo y de ser una opción 'positiva' puede pasar a ser 'negativa' para las prestaciones.

No se han realizado experiencias con escalados en tensión de alimentación en cambios de tecnología. Esta es una investigación interesante a realizar en el camino hacia tecnologías submicrónicas profundas y nanométricas. Finalmente no se ha introducido otro factor adicional como es el número de niveles de metal disponibles, de gran incidencia obviamente en densidad y área. Todas las tecnologías utilizadas tienen los mismos tres niveles de metal.

1.8. Algunas recomendaciones metodológicas para el diseño SOC con componentes IP

También como resultado de estas experiencias, y de otros diseños anteriores realizados en el grupo, puestos ahora en una nueva perspectiva, este trabajo permite extraer algunas recomendaciones para el diseño SOC con componentes IP [Per99]. Estas recomendaciones se resumen a continuación:

- El diseño SOC basado en IPs y otros componentes virtuales requiere implementar con facilidad circuitos de cientos de miles de transistores, incluso millones. Esto requiere una metodología connsistente y muy estructurada, y aplicar técnicas de optimización en diversos niveles.

- Debe disponerse de una amplia librería de componentes virtuales lógicos y físicos, caracterizados y con sus modelos de síntesis o los modelos extraídos.

- Debe integrarse la toma de decisiones en los niveles altos con la exploración del espacio de diseño y con resultados físicos caracterizados en prestaciones

- Conviene ser concisos en la escritura de código VHDL. Esta concisión implica evitar el uso excesivo de operadores y variables intermedias. En caso que esto no se evite las herramientas de síntesis lógica suelen intentar eliminar la lógica redundante pero en ocasiones no lo consiguen debido a posibles dependencias inadvertidas e indeseadas, en las variables y señales internas de las descripciones, produciendo diseños innecesariamente mayores y más lentos.

- Los núcleos IP más importantes deben explorarse en sus opciones microarquitecturales y lógicas, caracterizando su comportamiento en el diseño físico. Una vez caracterizados conviene tomar las decisiones oportunas y fijar entonces las opciones a este nivel.

- Evitar en lo posible una jerarquización excesiva: emplear la dosis *correcta*. Los bloques jerárquicos pequeños se optimizan y sintetizan enseguida, pero si hay que manejar muchos de ellos la situación puede asemejarse a la producida con las descripciones complicadas, pudiendo ser el resultado peor que utilizar menos bloques pero más grandes. Este proceso tiene un punto óptimo de compromiso. Con frecuencia conviene combinar los bloques y elementos que se han creado de forma separada y sustituirlos por nuevas descripciones planas. El tamaño óptimo de bloque y módulos depende del tipo de núcleo, su funcionalidad, de la herramienta, de la tecnología (niveles de metal de ruteado) y de la estación en la que corren las herramientas. Con frecuencia se encontrarán buenos diseños con bloque y módulos en un rango de granularidad de entre 3000 y 30000 transistores, con módulos de tipo 'medio'.

- Conviene pensar como si se estuviera diseñando el *hardware* propiamente dicho. Conviene utilizar diagramas de bloques para los principales componentes del diseño, y entradas gráficas como SGE, y dibujar los esquemáticos de interconexión. A continuación se puede crear código VHDL para cada uno de los bloques y módulos del diagrama. Conviene estructurar el código para configurar opciones en estos bloques, tanto microarquitecturales como lógicas.

- Sólo debe manejarse un reloj por bloque.

- Los registros de segmentación deben ser disparados por flanco.

- Conviene concentrar la optimización de la síntesis en opciones que afecten a los bloques y módulos previsiblemente críticos, y eliminar en lo posible todo exceso de lógica en esos bloques. El esfuerzo de la optimización debe concentrarse en aquellas rutas entre señales que formen parte de las rutas globales más lentas para intentar descargar estas rutas intermedias de lógica y trasladarla a otras rutas menos críticas. En ocasiones se hace necesario fijar en estos elementos la síntesis realizada para que al sintetizar los bloques de jerarquía superior en los que se hayan inmersos estos elementos críticos no se modifiquen.

- Deben separarse los bloques de temporización crítica de los que no lo son. Conviene adaptarse al flujo de optimización de las herramientas y realizar agresivamente

aquellas optimizaciones de área y consumo que cumplan la especificación de tiempo.

- Es bueno agrupar y crear un nivel jerárquico separado para las FSM de control. El control debe estar bien estructurado.

- Para componentes y núcleos de tamaño medio conviene seguir un flujo *top-down*, especialmente mientras no se establezcan estándares VSI (*Virtual Socket Interface*). Este flujo *top-down* debe basarse en bloques medios de hasta 30000 transistores, y sintetizar bloques preservando la jerarquía a estos tamaños medios. Si se requiere una reducción adicional de área, entonces se debe eliminar toda la jerarquía en bloques pequeños y agrupar en tamaños medios y grandes, o alternativamente crear descripciones planas. Si se requiere una reducción adicional de tiempo, entonces se deben optimizar los retardos en cada bloque crítico identificando aquellas señales que formen parte de los caminos críticos y concentrando en ellas el mayor esfuerzo de optimización. Puede ayudar el crear nuevas jerarquías combinando varios bloques críticos en uno solo mayor, y los no críticos de forma separada, y entonces desagrupar toda jerarquía en el interior de los bloques críticos en tiempo y los críticos en área. De esta forma se puede posteriormente optimizar esta circuitería y caracterizar los resultados.

- Para componentes, núcleos y sistemas grandes conviene combinar el flujo *top-down* con flujos *bottom-up*. Entonces se debe explorar el espacio y optimizar cada componente del núcleo por separado y después componerlos juntamente. Al ir conformando bloques más grandes éstos no se deben re-optimizar posteriormente, sino que deben incorporarse con sus propiedades fijas previamente seleccionadas. Después se puede explorar el espacio y optimizar el colocado y ruteado del conjunto de bloques.

- Se recomienda colocar un nivel de registros separando e intercomunicando los bloques y componentes mayores cuando se necesita preservar la temporización y caracterizacón de cada bloque y del conjunto.

- Por último se hace necesario generar un análisis temporal en tiempo de pre-colocado/ruteado y en tiempo de post-colocado/ruteado y tomar otras medidas de prestaciones, caracterizando el diseño y realimentar esta información para modificar las opciones microarquitecturales, de granularidad y bloques, y de colocado/ruteado cuando se necesite.

Todo ello requiere de un entorno de generación y análisis de componentes virtuales, y un banco de diseños caracterizados y caracterizables, como los desarrollados en este trabajo.

La reorganización de componentes, IPs, y diseños reutilizables, en un entorno convencional de síntesis *algorítmica* $\longrightarrow$ *RTL* $\longrightarrow$ *lógica* $\longrightarrow$ *física* no es adecuada para lograr una exploración amplia del espacio de diseño y una selección correcta de las opciones y versiones de los IPs y componentes de interés para integrar en un SOC. De hecho, si se intenta una optimización a nivel arquitectural, microarquitectural o lógico, es con frecuencia a ciegas de su verdadero efecto en el diseño físico.

Estas recomendaciones y metodología de diseño difieren de los resultados de [Per99], que se limitan a síntesis *top-down* sobre FPGA; de la metodología de [PJRL99] basada en explorar el espacio de diseño y hacer una extensa preclasificación de clases de objetos de diseño, más que en una exploración del espacio de diseño integrada en el proceso de síntesis, con toma de decisiones en los diferentes niveles, como es nuestro caso.

Igualmente difieren de [PHSMR95], que se basa en la parametrización HDL de componentes virtuales, y de [AOS94], que descansa en modelos basados en un conjunto de características. Los dos primeros trabajos comparten con nosotros el énfasis en el espacio de diseño, más que en los objetos del diseño propiamente dichos. Sin embargo, no consideran el análisis de los importantes efectos que el proceso de diseño físico tiene sobre las opciones de niveles superiores. Los componentes utilizados como ejemplos son además muy específicos o tienen una complejidad pequeña, alejada del interés principal en síntesis de sistemas en un chip, SOC.

Otros enfoques [Küç99] incluyen la síntesis de procesadores estándares, pero en este caso se centran en modificar el núcleo a medida de la aplicación explorando el espacio de diseño a nivel arquitectural, y sólo en el sentido de recortar o extender el juego de instrucciones estándar, lo que implica el empleo de compiladores *ad hoc* [Lie97]. Este enfoque suele hacer surgir desconfianza en el campo industrial, y alarga el ciclo de desarrollo de *software*.

Finalmente, nuestro enfoque de explorar el espacio de diseño de forma integrada, desde la microarquitectura hasta los resultados del *layout* física, se centra en un ámbito distinto a lo que se conoce como exploración del espacio de diseño para la arquitectura en síntesis de arquitecturas, en síntesis de sistemas y en codiseño [ANH+93, PGLM94, HD94, KCG+98, Küç98, CM99b, BCF+97, HGG+99, LvPK+95, LM98, LPCJ95, WKV+96, KDVvdW97, BSC+97]. La mayor carencia en este tipo de metodologías es justamente limitarse a toma de decisiones arquitecturales, con análisis y con precisión sólo a nivel de ciclos de reloj, raramente al nivel de puertas y transistores, y menos aún al nivel completo de *layouts* físicos. Consecuentemente enfatizan los aspectos de generación de código para las arquitecturas objetivo encontradas. Estas críticas están en línea con el punto de vista industrial brevemente expuesto en [PS99]. Este trabajo ha demostrado el importante impacto que tiene en las prestaciones los niveles inferores de diseño, también y especialmente en el ciclo de reloj, y cuánto y cómo las deciones de alto nivel quedan matizadas y modificadas por el entorno de diseño en cuanto al resultado esperable y deseado. De ahí la necesidad de integrar más todos los niveles de la concepción de sistemas en un chip.

CAPÍTULO 2

Antecedentes

2.1. Introducción

Con el avance de la tecnología, los microprocesadores han sido cada vez más complejos y más rápidos. La creciente complejidad de éstos se ha debido a las mayores densidades de integración de transistores en un chip. Inicialmente esta densidad se ha utilizado para dotar al microprocesador de instrucciones complejas, semánticamente similares a sentencias propias de lenguajes de alto nivel, lo que se justificaba como un medio de facilitar la elaboración de los programas en código máquina. Debido a las capacidades cada vez mayores de estos microprocesadores, el abanico de posibilidades de utilización se ha ido haciendo cada vez más amplio, beneficiado además por su creciente popularidad y por las facilidades aportadas en el desarrollo y depuración de programas.

Los procesadores se han ido introduciendo de forma gradual como elementos básicos en la construcción de sistemas de una cierta complejidad, de modo que el diseñador puede centrar su atención en la comunicación con los elementos capturadores de datos y actuadores, y en el desarrollo del *software*. Ha surgido así un nuevo modo de pensar y concebir los sistemas digitales de complejidad media: como sistemas digitales basados en microprocesador.

Sin embargo, a la hora de utilizar un microprocesador para una aplicación determinada resulta patente que no sirve cualquier tipo de arquitectura. Por esto se han planteado distintos tipos de arquitecturas adaptadas al tipo o dominio de aplicaciones a las que se destinan estos procesadores en algunos casos. De este modo han aparecido los primeros ASIPs (*Application Specific Integrated Processors*, como, por ejemplo, los Procesadores Digitales de Señal o DSPs) y los microcontroladores.

El desarrollo de procesadores semánticamente cada vez más complejos (Computadores de Juego de Instrucciones Complejo, o CISCs), favorecido por los arquitectos y dominante en la industria hasta 1980, ha cedido paso a la escuela de diseñadores de arquitecturas de procesador que aboga por el principio de sencillez en sus estructuras. Esta escuela ha dado origen a los llamados RISCs, o *computadores de juego de instrucciones reducido*. En el campo de los equipos destinados a aplicaciones de propósito general (ordenadores), estas arquitecturas están experimentando una utilización cada vez mayor, gracias a, fundamentalmente, la mayor velocidad de ejecución de los programas.

En el campo de los microcontroladores, existen aplicaciones en las que las necesidades de velocidad no son tan imperantes como en los procesadores de propósito general. Esto se debe fundamentalmente a los ambientes hostiles en los que estos procesadores van a trabajar y a que en aplicaciones de tiempo real de control industrial muchas veces la dinámica del proceso es lenta, por lo que tales velocidades no suponen ninguna ventaja. En estos casos puede resultar más beneficioso el uso de algunas estrategias CISC con el fin de disminuir el tamaño de los programas, y, ocasionalmente, el tamaño del conjunto del sistema. Existen microcontroladores pequeños para este tipo de situaciones donde el tamaño de los datos llega a ser de unos pocos (4 u 8) bits e incluso se prescinde completamente de mecanismos básicos para acelerar la ejecución de los programas como puede ser la segmentación.

Sin embargo, en otro tipo de aplicaciones donde la velocidad es fundamental, como es el caso de sistemas de tiempo real crítico, de tratamiento digital de señales, o de sistemas multimedia, se han realizado estudios sobre el comportamiento de procesadores RISC, de propósito general, en sistemas de este tipo. En el caso del tratamiento de señales se ha llegado a demostrar que en varias aplicaciones con RISCs se pueden lograr mejores prestaciones que con la utilización de DSPs [Smi92a, Smi92b]. De hecho se ha producido cierto grado de convergencia entre RISCs y DSPs intercambiándose mutuamente algunas características típicas de diseño. Hoy en día existen ya algunos procesadores comerciales y otros resultantes de algunos proyectos de investigación (como el RASSP, *Rapid Prototyping of Application Specific Signal Processors*), con los que se han conseguido arquitecturas de procesadores RISC con algunas adaptaciones de la ruta de datos para acelerar operaciones comunes en las tareas de procesado de señales. Recíprocamente, los DSPs han adoptado técnicas de segmentación y mayores espacios de direccionamiento.

2.2. Opciones de diseño de procesadores

Los aspectos primordiales que influyen decisivamente en la solución CISC, DSP o RISC que se adopte son la finalidad del sistema que vamos a diseñar, el coste, tanto de diseño como de mantenimiento; y el tiempo que se disponga para elaborar el sistema (conocido en inglés como *time-to-market*[1]).

A la hora de afrontar el diseño de un sistema determinado, se ha de tener presente, por encima de otras cuestiones, que el producto final debe ser competitivo. Esto quiere decir, básicamente, que las prestaciones que ofrezca el sistema final deben ser aceptables

[1]En términos muy mercantiles algunos prefieren incluso hablar del *time-to-profit*.

para el costo que supone tanto su desarrollo y fabricación como su utilización, frente a soluciones alternativas o ya existentes.

En el campo de los procesadores, y dentro del ámbito de los que se destinan a ordenadores y sistemas de cálculo, el hecho de decidirse por una arquitetura determinada debe proceder de una comparación entre las ventajas e inconvenientes de utilizar una de entre distintas soluciones.

Por lo que respecta a los sistemas en los que se utilizan procesadores de aplicación específica, la situación está cambiando radicalmente. Hasta ahora, para utilizar procesadores para implementar ciertos sistemas, se han elegido procesadores con una arquitectura adaptada a determinado tipo de aplicaciones. Una vez decidido el procesador a utilizar, se desarrollaba el sistema completo añadiendo circuitería y se programaba este procesador. Este es el caso, por ejemplo, de los DSPs y de los microcontroladores comerciales.

Sin embargo, con el gran avance que han experimentado las herramientas de CAD electrónico, empieza a ser factible el diseño de sistemas completos integrados en los que el procesador se adapte más perfectamente al problema planteado. A los procesadores que van integrados en este tipo de sistemas se les conoce como procesadores empotrados (en inglés *embedded processors*). En la elaboración de este tipo de sistemas, la arquitectura del procesador ya no es totalmente rígida, sino que queda a la voluntad del diseñador el control de algunas de sus características, y consecuentemente de sus prestaciones. La modificación de la arquitectura hace que el compilador para este procesador también deba modificarse, de modo que podamos obtener, a partir una aplicación desarrollada en un lenguaje de alto nivel, el código que se ejecutará sobre esta arquitectura. De este modo, el diseñador puede comparar distintas decisiones de diseño del procesador para ver cuál de ellas se adapta mejor a sus necesidades.

Por otro lado, la adaptación del diseño del procesador al problema implica también una pérdida de generalidad en su programación, y en la existencia o desarrollo de *software* y aplicaciones. La decisión entre soluciones *programables* y soluciones *fijas* o específicas depende del problema.

2.3. Opciones para el control

Si el sistema que se necesita implementar no precisa de una circuitería de control excesiva, con las actuales herramientas de diseño y síntesis de sistemas digitales se pueden obtener resultados muy satisfactorios. En estos sistemas suelen aparecer en la circuitería de control uno o varios autómatas, en cuyo caso una solución que incorpore un procesador empotrado puede resultar desaconsejable. Sin embargo, a medida que la complejidad de este control crece, el empleo de estas máquinas algorítmicas puede resultar costoso en el aspecto del diseño —al necesitarse un mayor número ciclos de prueba y depuración— o bien ir en detrimento de las prestaciones —al aumentar la cantidad de circuitería—.

El principal problema planteado en este tipo de situaciones es que, cuando el control del sistema supera ciertos niveles de complejidad es preciso utilizar jerarquización para

describir la lógica de control. Esta jerarquía es muy similar a la que aparece en los procesadores microprogramados, sólo que, en este caso, en vez de utilizar microsecuenciadores se emplean procesadores completos, y en lugar de microensambladores podemos recurrir a los potentes compiladores optimizantes que se han desarrollado para dar soporte a estos procesadores.

Una vez decidida qué arquitectura utilizar, se ha de tener presente que el modelo para su estudio debe ser lo suficientemente cómodo para permitir una evaluación y desarrollo "rápidos". Este es un aspecto en el que conviene insistir, pues cuanto más simple y fácil de simular y sintetizar sea el modelo utilizado del procesador, menor será el coste de diseño del sistema final.

2.4. Opciones de herramientas CAD

Hoy en día, como consecuencia de las evidentes carencias que presentan las herramientas CAD, está comenzando a aparecer un gran número de investigaciones con el propósito de automatizar y facilitar el diseño de sistemas específicos complejos. La aproximación tradicional, *bottom-up*, está siendo sustituida por un enfoque global *top-down*. En este enfoque se parte de una especificación del problema realizada a alto nivel, frecuentemente una especificación funcional o de tareas de tipo meta-algorítmico. Se han generado así entornos de síntesis de alto nivel de arquitecturas y de sistemas, cuya entrada es una especificación funcional y cuya salida es un grafo de flujos de datos y control particionado en unidades de proceso (*hardware*) planificadas por unidades de control (*software*).

Si bien gran parte de los conceptos básicos en el campo de la síntesis de alto nivel de sistemas están desarrollados desde hace tiempo, el verdadero problema del diseño y evaluación cuasi-simultáneos del *software* y el *hardware* (lo que en inglés se conoce como *hardware-software co-design*) casi nunca se ha tocado con profundidad. Hasta ahora, normalmente lo que se ha hecho es fijar de algún modo la arquitectura de los sistemas y realizar la programación del procesador que forme parte de éste, realizando pequeñas variaciones en la arquitectura del sistema cuando se evidencien ventajas sustanciales. De este modo, el verdadero objetivo consiste en hacer que la complejidad del hardware añadido sea menor que la del software eliminado, haciendo que el producto sea más eficiente.

El planteamiento actual parte de la base de que este tipo de aproximaciones muy probablemente no coincida necesariamente con las soluciones mejores en cuanto a prestaciones, si bien, hasta la llegada de nuevas herramientas, son las que ofrecen menor coste de desarrollo. Precisamente éste es el principal problema con el que se están enfrentando los diseñadores de las herramientas CAD. La necesidad de crear sistemas de estas características con un coste razonable hace que se empiecen a introducir nuevos tipos de metodologías y conceptos en las herramientas de ayuda al diseño. Pero la solución y disponibilidad de herramientas CAD no es elevada, y hasta que lleguen habrá que hacer uso de las herramientas existentes para obtener una solución con un coste razonable a este tipo de problemas.

Por otro lado, existe una multitud de situaciones en las que los actuales fabricantes de las herramientas CAD están considerando el impedir la modificación de la funcionalidad de los núcleos de procesador a los usuarios. Por esta razón, si bien les están ofreciendo modelos simulables de estos procesadores y algunas posibilidades de parametrización, de cara al usuario estos modelos quedan como auténticas cajas negras con el mismo aspecto que los microprocesadores discretos.

2.5. Sistemas empotrados y de tiempo real

A medida que se ha ido ganando experiencia en la utilización de procesadores, se ha empezado a plantear la posibilidad de utilizarlos en sistemas orientados al control de procesos, en sistemas de telecomunicación, y en sistemas audiovisuales y multimedia. El procesador se encuentra *empotrado* con otros subsistemas de proceso de entrada/salida o comunicación. Estos sistemas suelen ser muy específicos y de tamaño reducido. La aparición de sistemas de este tipo con procesadores incorporados presenta unas características especiales distintas de las de los ordenadores, y por ello ha surgido el concepto de *sistema empotrado.* Bajo *empotrado* se designa al sistema o aplicación en el que la función principal *no* es el de procesamiento de información, sino la supervisión, organización y control de los flujos de señales y datos. Este tipo de sistemas suele pertenecer al dominio de problemas que requieren procesamiento en *tiempo real.*

> Un sistema de tiempo real es cualquier actividad o sistema de procesamiento de información que debe responder ante estímulos de entrada generados externamente dentro de un período de tiempo finito y especificado.[2]"

El buen funcionamiento de un sistema de tiempo real no sólo depende del resultado de la computación sino también del instante en el que se haya disponible. Aunque un sistema sea capaz de obtener una respuesta correcta, si no la entrega dentro un determinado intervalo de tiempo este sistema no servirá para aplicaciones de tiempo real. Es esta circunstancia, el tiempo de procesamiento, junto a la diversidad de tareas que se deben realizar, la que determina en gran medida la complejidad de este tipo de sistemas.

A pesar de la gran diversidad de tipos de sistemas empotrados que han aparecido, existen algunas propiedades comunes en muchos de ellos. Esto se debe, fundamentalmente, a que el tipo de problemas y necesidades a las que se enfrentaron sus diseñadores tienen muchas similitudes:

Tamaño y complejidad: Por regla general, se desea que los programas sean pequeños. Esto facilita cambios futuros y el mantenimiento del *software.* Sin embargo, es bastante frecuente encontrarse con ocasiones en las que el problema a resolver es lo suficientemente complejo como para que los programas tiendan a ser grandes. Además, aunque el problema original dé lugar a programas pequeños, el

[2]Se cita en [BW90] como procedente de "Young S.J. (1982). *Real Time Languages: Design and Development. Chichester: Ellis Horwood.*

entorno en el que se encontrará el sistema puede obligarnos a atender nuevas necesidades y realizar ampliaciones. Esto redunda en un aumento del tamaño de estos programas y/o del *hardware*. Por esta razón es deseable que el *software* esté diseñado de modo que facilite no sólo posibles depuraciones sino también eventuales ampliaciones.

También existen situaciones en las que, al intentar establecer medios para reducir el espacio de los programas, el procesador se hace muy complejo, produciendo ocasionalmente pérdida de velocidad, cosa no deseable en aquellos casos en los que el tiempo de respuesta debe ser muy pequeño.

Manejo de números reales: Los sistemas de tiempo real están destinados a operar con los datos procedentes de señales externas. Estas señales son "capturadas" por acondicionadores de señal con los que se obtienen representaciones numéricas de estas señales.[3] Para el manejo y procesamiento con estos datos, se debe disponer de unidades funcionales que puedan operar con ellos.

Además, si pensamos en que los procesos que deben manejar estos datos suelen ser complejos, y que los datos numéricos en sí se agrupan en estructuras más o menos complejas, no resulta difícil comprender que la mayor parte de los sistemas de este tipo presenten características propias de los computadores. Si además nos encontramos con sistemas en los que las señales con las que se opera son de naturaleza analógica, se puede entender que existen muchos casos en los que se hace necesario un mínimo de precisión en los cálculos. Por esta razón, y evitando la posible necesidad de utilizar una unidad de coma flotante, los formatos de los datos que se emplean se suelen elegir de modo que los cálculos se realicen con una precisión aceptable. Esto hace que el sistema global tenga un tamaño y una complejidad bastante menor, lo cual es lo deseable en este tipo de sistemas. Incluso en algunos compiladores para procesadores de propósito general se puede forzar la obtención de código que no recurra a la unidad de coma flotante a través de opciones de compilación específicas.

Fiabilidad y seguridad: Cada vez es mayor el número de sistemas en los que se ha optado por emplear soluciones basadas en arquitecturas de procesador. El buen funcionamiento de estos sistemas suele resultar vital para el normal desarrollo de un sinfín de actividades. Por esta razón, cada vez se hacen mayores esfuerzos por diseñar estos sistemas con fiabilidades mejores.

En el caso de los sistemas que van a trabajar en ambientes hostiles, esta fiabilidad se traduce en una serie de consideraciones que en otro tipo de circunstancias no se tienen en cuenta. Por ejemplo, a veces el sistema debe ser *tolerante* a ciertos fallos. Otras veces, cuando se proyecta una interface con un posible operario, ésta puede llevar incorporada mecanismos que disminuyan el efecto de posibles errores humanos.

Todo este conjunto de nuevas condiciones de trabajo en las que operan estos sistemas han planteado nuevos retos a los diseñadores, implicando la aparición de normas nuevas, como la ISO 9001 [ISO94].

[3] En realidad, se obtienen números en alguna representación binaria de muestras de estas señales y con un error de cuantificación [OS90]

En otro tipo de aplicaciones, como en algunas de tratamiento de imágenes, la fiabilidad se puede extender a otros aspectos. Por ejemplo, a veces se puede considerar tolerable cierta pérdida en la información sobre éstas con tal de lograr niveles de compresión aceptable en comparación con el tipo de información que se pierde [SDS96].

Por otra parte, ya que con frecuencia es preciso que un sistema cuasiautónomo proporcione información a otros, existen riesgos asociados a estas vías de comunicación, que van desde la pérdida ocasional y modificación de información hasta posibles intentos de control por personas no autorizadas, de fraude electrónico o de contagio de virus. Para este tipo de problemática se está investigando continuamente tanto en nuevos modos de recuperar la información original en destino en situaciones de pérdidas puntuales como en algoritmos de cifrado para garantizar la privacidad de la información que viaja por las redes.

Control concurrente de componentes separados: Debido a su propia naturaleza, en un sistema de tiempo real típico suelen coexistir distintos elementos externos que deberán interactuar. Algunos de estos elementos estarán especialmente destinados a proporcionar algunas de las características *destacables* del sistema, tanto sea de recogida de datos por diversos canales de entrada hasta el cifrado para enviar información *segura*. En algunos casos esta interacción debería ser simultánea. Si bien a veces esto no es posible, estos sistemas se deben diseñar de modo que los tiempos empleados en estas interacciones sean lo suficientemente pequeños como para considerarlos *aceptables* o *permisibles*.

Para este tipo de situaciones tradicionalmente han existido diversos enfoques. En algunos casos este tipo de problemas se ha abordado siguiendo estrategias típicas de arquitecturas paralelas. Para estos sistemas la elaboración del *software* se ha realizado con lenguajes de programación concurrente.

En algunos de estos sistemas, aunque la velocidad es un factor determinante, los tiempos de ejecución que aporta la tecnología actual permiten que algunos de estos problemas se puedan afrontar con sistemas monoprocesador. Sin embargo, las necesidades de tiempo de respuesta frente a los estímulos externos tienen en este tipo de sistemas una influencia decisiva sobre muchas cuestiones de diseño. Por ello, muchas veces podemos encontrar procesadores que no tienen una arquitectura paralela pero que se interconectará con elementos especialmente dedicados a determinadas subtareas críticas que consumen mucho tiempo. En este escenario estos elementos tienen un tratamiento similar a los demás elementos del sistema empotrado pero en algunos casos con mayores requisitos. Con esto el procesador, en vez de dedicar recursos a realizar estas tareas, sólo necesitará recoger los datos de estas unidades especiales y gestionarlas, con el consiguiente ahorro de tiempo.

Facilidades de tiempo real: El tiempo de respuesta es crucial en cualquier sistema de tiempo real. Sin embargo, el diseñar un sistema que produzca las señales apropiadas en los márgenes de tiempo adecuados suele resultar complicado. Por ello, la mayor parte de las decisiones de diseño se eligen con objeto de obtener los datos dentro de los márgenes de tiempo establecidos. Por otra parte, en oca-

siones se diseña el sistema previendo posibles fallos en las unidades principales de ejecución. Por esta razón, muchos sistemas de tiempo real se han diseñado contemplando recursos de reserva auxiliares para ser utilizados en los casos en los que los principales fallen.

También es típico en muchos sistemas de tiempo real en control y telecomunicaciones encontrar temporizadores y contadores de pulsos. Con éstos y algunos elementos de control y/o software adicionales también se le permite al programador tener en cuenta el factor tiempo a la hora de establecer el comportamiento del sistema. Gracias a estas facilidades de control de tiempo real se logra sincronizar los procesos con la magnitud tiempo.

Interacción con los elementos externos: Los sistemas de tiempo real necesitan interactuar con el mundo exterior, o con canales y medios de comunicación. Para realizar esta interacción es necesario operar con sensores —para recoger datos del exterior— y actuadores —para transmitir información al exterior— a los que el núcleo estará interconectado, o equivalentemente en redes de telecomunicación con dispositivos de recepción y de transmisión. Normalmente el núcleo procesador se comunica con estos dispositivos a través de determinadas posiciones en memoria o con puertos de entrada y salida. Estos dispositivos suelen también generar señales de interrupción que se le envían al procesador para indicarle que se han cumplido determinadas condiciones de operación o de error. La respuesta ante estos elementos externos debe ser, por lo general, lo más rápida posible. Esto implica en la mayoría de los casos que el procesador deberá tener tiempos de respuesta pequeños ante las peticiones de las interrupciones. Esto suele conllevar decisiones de diseño tanto en el *hardware* como en el *software*.

En lo que respecta a la gestión *software* de estas interacciones, normalmente se deja que sea el sistema operativo el que las lleve a cabo, para lo cual éste utiliza *drivers*. Sin embargo, en ocasiones la gestión de algunas de estas interacciones puede resultar crítica, en cuyo caso el diseño del software se realiza de modo que sea el propio proceso principal el que se encargue de gestionarlas.

Eficiencia en la implementación: Ya que en los sistemas de tiempo real pueden coexistir diversos factores críticos, las estrategias de diseño necesitan contemplar un conjunto mayor de variables. Esto hace que la eficiencia sea más difícil de evaluar, al tiempo que resulta más importante. Por esta razón, a la hora de realizar los diseños es deseable contar con distintas alternativas, pero imperando, en cualquier caso, los criterios de eficiencia en la elección final.

Consecuentemente, el factor tiempo es la principal magnitud que se debe contemplar en las decisiones de diseño de un sistema de tiempo real. Como se habrá observado, todas y cada una de las propiedades que presentan estos sistemas son producto de la necesidad de lograr que el sistema responda correctamente dentro de los intervalos de tiempo *aceptables*, de modo que los elementos externos puedan actuar *a tiempo*. En función de la prontitud con la que se precisa la entrega de estas respuestas al exterior, algunas de estas características presentadas tendrán mayor peso que las otras. Sin embargo, esta escala de prioridades deberá también cumplir con el resto de especificaciones del sistema

de tiempo real en lo que concierne a tamaño, consumo de potencia y otras, por lo que finalmente se deberán adoptar soluciones de compromiso.

Debido a que el conjunto de posibilidades que entran en juego en el diseño es tan variado, las soluciones son diversas. A continuación exponemos las distintas alternativas que comúnmente se han seguido en el diseño de estos sistemas.

2.6. Elección de una alternativa: circuitería fija o programable

Para decantarse por una solución muchas son las variables que hay que tener en consideración. Muchas veces intervienen factores que no dependen exclusivamente del problema a solucionar en un determinado instante sino que también se puede contemplar la viabilidad de permitir futuras modificaciones, mejoras o extensiones. En tales casos la flexibilidad es algo que se impone desde las primeras etapas del diseño, en las especificaciones del sistema a desarrollar. En otras ocasiones, esta flexibilidad puede permitir una mayor simplificación de la realización física de un sistema, ya que en ocasiones esto puede suponer una mayor optimización en el uso de los recursos del sistema. Es entonces cuando aparece el problema de qué subtareas desarrollar en *hardware* y cuáles a través de *software*.

Por este motivo, en la actualidad existen numerosos estudios para el desarrollo de metodologías con las que realizar el codiseño simultáneo del *hardware* y el *software*. Con estas metodologías se puede estimar cuál será la relación entre las prestaciones y el coste para diversos sistemas orientados a la resolución de problemas específicos. Basándose en estas estimaciones se puede tener una idea de los costes de las soluciones realizadas tanto enteramente *hardware* como con soluciones con mayor peso en el *software*, y a su vez también el coste de un amplio abanico de soluciones intermedias [GdM93, Gup95].

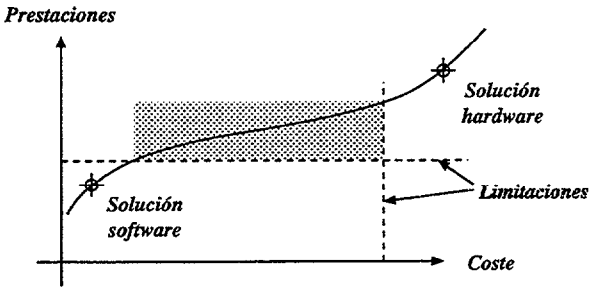


FIGURA 2.1: Enfoque propuesto para la implementación de sistemas en metodologías de cosíntesis de sistemas digitales

En estos casos, según se observa en la fig. 2.1, cuanto mayor es la cantidad de *hardware* mayores prestaciones logramos pero también el sistema se encarece. Sin embargo, cuanto más sencillo sea el *hardware*, mayor será el aporte del *software* y las prestaciones serán peores.

Éste es el espacio de diseño general en el que la totalidad de los diseñadores se mueven.

En el extremo inferior tendremos una gran parte del *hardware* destinada a almacenar el *software*. Sin embargo, a medida que el juego de instrucciones del procesador se hace más rico el tamaño del *software*, y por lo tanto los recursos que se destinan a almacenarlo, disminuirá.

Debido a esta gran diversidad de variables de que se dispone, al acometer el diseño de un sistema para una aplicación específica el número de factores que determinan la alternativa elegida debe restringirse.

Teniendo en cuenta el tipo de aplicaciones que queremos desarrollar, donde normalmente pueden coexistir múltiples tareas, la implementación de un sistema empotrado basado en un microprocesador se hace completamente necesaria. Este microprocesador debe tener unas características tales que permita lograr unas prestaciones adecuadas, al mismo tiempo que facilite el diseño del sistema final. Resulta especialmente justificada la elección de un núcleo procesador RISC para este tipo de situaciones, como veremos.

2.7. Sistemas basados en microprocesador, SBC

Los primeros sistemas empotrados surgieron como resultado de la aplicación directa de los primeros microprocesadores que presentaban una capacidad de cálculo adecuada para su empleo en tareas de control. Actualmente existen múltiples soluciones similares, donde en una placa se integran pequeños sistemas empotrados que incorporan microprocesadores (SBC, *Single Board Computer*). En estas soluciones se intenta explotar las capacidades de los procesadores de propósico general para combinarlos con elementos adicionales —necesarios para el tipo de aplicación para el que se conciben estos sistemas—, logrando sistemas de reducido tamaño y buenas prestaciones con coste reducido.

Una de las ventajas de estos sistemas es que, al diseñarse con características similares a los ordenadores tradicionales, el desarrollo de las aplicaciones para estos se puede acometer de forma sencilla.

2.8. Microcontroladores

Sin embargo, para algunas aplicaciones en las que resulta innecesario un sistema con la complejidad de un microprocesador de propósito general, basta para acometerlas pequeños procesadores y algunos elementos de memoria y de interfaz. Es este tipo de sistemas el que da lugar a una nueva visión de los *sistemas empotrados*, en los que el procesador se encuentra en un sistema autónomo junto a sus elementos de memoria, comunicación y control. En éstos se encuentra un proceso principal ejecutándose cuya construcción no se necesitará modificar una vez incorporado.

De este modo surge el concepto de *microcontrolador*, que básicamente se trata de un procesador al que se le ha añadido circuitería para facilitar las operaciones de recogida de datos y envío de información y control. Debido al tipo de aplicaciones al que se destinan,

los núcleos procesadores de éstos suelen tener datos de tamaños muy pequeños, por lo general de 4 ó de 8 bits, y en algunos casos 16 bits.

2.9. Sistemas a medida o ASICs

En ocasiones conviene tener un sistema que sea muy eficiente a la hora de desarrollar una determinada tarea. Si las labores de control necesarias son escasas y el procesado está completamente definido, la utilización de un procesador se hace innecesaria e incluso inadecuada por encarecer el coste tanto en el diseño como en su uso. Entonces la alternativa que se suele seguir es que el control lo lleve a cabo una pequeña máquina algorítmica que resulta más económica. El circuito integrado resultante es un ASIC (*Application Specific Integrated Circuit*) y más recientemente, y para el caso en que el número de estados o la capacidad de proceso de la máquina algorítmica sea grande, se ha introducido el término ASIP (*Application Specific Integrated Processor*).

Este tipo de soluciones resulta especialmente útil cuando se desea que el sistema tenga, además de una alta eficiencia, una funcionalidad que se mantendrá fija y no se prevean futuras alteraciones. En este caso cualquier tipo de flexibilidad en sus características de operación no ofrecerá beneficios y sí inconvenientes como un mayor área ocupada y consumo de potencia. Por lo tanto, en este tipo de soluciones todo el *hardware* se organiza única y exclusivamente para desarrollar un tipo de funcionalidad que lo sitúa conceptualmente en el extremo opuesto al de los ordenadores tradicionales —donde toda la especificación del procesado a realizar se realiza a través del *software*—.

Gracias a las densidades de integración y las herramientas de diseño electrónico actuales, hoy en día se puede acometer el diseño de este tipo de sistemas más o menos complejos en un espacio de tiempo relativamente corto.

2.10. Sistemas programables o ASISPs

Sin embargo, a veces se requiere cierto tipo de flexibilidad debido al tipo de aplicación. Por ejemplo, existen situaciones en las que se pueden necesitar realizar cambios en el futuro debido fundamentalmente a una evolución de los estándares o a posibles extensiones del campo de aplicaciones. También existen circunstancias en las que la aplicación necesita expresamente distintos niveles de programabilidad, como es el caso de la codificación de vídeo.

Para este tipo de circunstancias la estrategia seguida es emplear una situación intermedia, donde se busca combinar una alta eficiencia en los métodos de procesado fundamentales de los datos —utilizando *hardware* específico— y la flexibilidad que permite la programabilidad para indicar cómo estos métodos deben interrelacionarse. En estos casos las posibles modificaciones se pueden abordar desde el *software*. Es el caso de los *Application Specific Instruction-Set Processors*, o ASISPs. En la medida en que son programables los ASICs que contienen un procesador integrado, descritos antes como ASIPs, pueden considerarse también en esta categoría. Con frecuencia la denominación

ASIP/ASISP, aunque conceptualmente distinta, resulta intercambiada en la práctica.

Existen numerosos entornos para síntesis de núcleos procesadores a medida de las aplicaciones. Todos ellos exploran el espacio de diseño, sobre todo en la selección del juego de instrucciones [ANH+93, PGLM94, HD94, KCG+98, Küç98, CM99b, BCF+97, HGG+99, LvPK+95, LM98, LPCJ95, WKV+96].

2.11. Los RISCs

Los RISCs (*Reduced Instruction Set Computers*) son procesadores cuyo diseño se realiza siguiendo una filosofía que hace énfasis en la simplicidad y la eficiencia. El diseño de un RISC parte de un juego de instrucciones necesario y suficiente.

Esta filosofía en la concepción de los procesadores surgió como reacción a la creciente complejidad que iban presentando los procesadores que se desarrollaban [Sta88]. La justificación para esta complejidad se fundamentaba en el intento de disminuir el salto cualitativo entre los lenguajes de alto nivel que utilizaban los programadores y el lenguaje de código máquina [PH94b]. Los procesadores aparecían con juegos de instrucciones cada vez más ricos. Sin embargo los arquitectuos encontraron que el 80% de los cálculos de un programa típico solamente necesita el 20% del juego de instrucciones de un procesador [Cat91]. Fue entonces cuando surgió la idea de no hacer evolucionar más las arquitecturas *hacia arriba*, y hacer énfasis en la concepción de arquitecturas con un juego de instrucciones reducido. Ha beneficiado especialmente esta tendencia la cantidad de investigación que se ha desarrollado en torno a los compiladores, ya que se puede demostrar que el código generado hoy por estos compiladores resulta más optimizado que el que puede desarrollar un programador en código máquina [PH94b]. Por esta razón existen autores que indican que con los RISCs la complejidad ha pasado de estar en la construcción de la arquitectura a estar en la elaboración del código.

El objetivo final de las arquitecturas tipo RISC es el de maximizar la velocidad efectiva de los programas, desarrollando para ello las funciones menos frecuentes en *software* e incluyendo en el *hardware* sólo aquellas funciones que sean frecuentes y que den lugar a ganancias netas de prestaciones[4]. Para lograr esto, los RISC presentan, sin ser éstas exclusivas de estos procesadores, las siguientes propiedades:

Ejecución en un solo ciclo: La mayor parte de las instrucciones se ejecutan en un único ciclo de máquina. Al minimizar el número de ciclos de reloj que se necesitan para ejecutar una instrucción, se obtienen ganancias significativas en prestaciones. Esto también supone ventajas importantes en la etapa de diseño.

Control cableado: Con esto se consigue la operación más rápida posible en cada ciclo. La utilización de microcódigo añade un nivel de complejidad, da lugar a rutas de control más lentas y añade otro nivel de interpretación de las instrucciones.

[4]Una idea muy similar es la que se está aplicando recientemente en algunas estrategias de codiseño: trasladar al *hardware* únicamente aquello que suponga un significativo ahorro en *software*, y que suponga un coste global menor.

Todo esto hace que el número de ciclos por instrucción a veces se incremente y que el tiempo de ciclo del procesador sea, por lo general, mayor.

Diseño de tipo carga/almacenamiento: En todas las instrucciones con cálculo hay registros involucrados. Sólo las instrucciones de carga y almacenamiento tienen acceso a la memoria.

Relativamente pocas instrucciones y modos de direccionamiento: Al reducir el número de instrucciones y modos de direccionamiento se facilita el diseño de un *hardware* de control del procesador más rápido y sencillo.

Formato de instrucciones sencillo y fijo: Todas las instrucciones tienen la misma longitud (normalmente 32 bits). La longitud uniforme de las instrucciones simplifica el *hardware* de decodificación de las instrucciones, lo que acelera las rutas de control.

Segmentación o *pipelining*: La segmentación permite comenzar la ejecución de una instrucción nueva en cada ciclo y optimizar el uso de los recursos de la máquina.

Memorias de altas prestaciones: La mayoría de las máquinas RISC tiene al menos 32 registros de propósito general, organizados en un fichero de registros, y memorias *cache* grandes. Con esto se intenta minimizar las dependencias de la velocidad con respecto a posibles memorias lentas donde se almacenan los datos y los programas.

Migración de funciones complicadas al *software*: Sólo las características que mejoren considerablemente las prestaciones se implementan en *hardware*. Para ejecutar funciones complicadas, se ha prescindido de instrucciones que realicen directamente estas funciones, ya que, por lo general, es el *hardware* asociado a estas funciones el que determina el tiempo de ciclo de la máquina. Al eliminar este *hardware* el tiempo de ciclo se hace menor y el tiempo de ejecución de cada una de las instrucciones se reduce proporcionalmente. Por ello, para realizar una de estas funciones lo que se hace es introducir una secuencia de instrucciones del RISC (que son sencillas) con las que desarrollar la función deseada. Gracias a esta reducción de la complejidad del diseño del *hardware* se aumentan las prestaciones y se mejora la eficiencia del sistema.

En muchos casos los RISCs ejecutan instrucciones retardadas, como son las de salto retardado. En estas instrucciones el salto se produce después de ejecutarse la instrucción (o instrucciones) posterior a la instrucción de salto. Esto se realiza así para no introducir ciclos de espera que harían desperdiciar los recursos de la máquina hasta que se obtenga la dirección efectiva de salto, y por ello es el salto el que se pospone.

En definitiva, en un RISC todo el diseño está orientado a sacar el máximo rendimiento de los recursos de que se dispone en la arquitectura en todo momento. Teniendo en cuenta la fórmula clásica con la que se puede analizar el tiempo que se emplea en la ejecución de un programa [PH94b]:

$$\frac{\text{Tiempo}}{\text{Programa}} = \frac{\text{Tiempo}}{\text{Ciclo}} \times \frac{\text{Ciclo}}{\text{Instrucciones}} \times \frac{\text{Instrucciones}}{\text{Programa}}$$

vemos que existen varios términos sobre los que podemos incidir a la hora de intentar mejorar el tiempo de ejecución de las tareas. En concreto, en el término $\frac{\text{Instrucciones}}{\text{Programa}}$ la mejora de este factor corresponde al compilador. En cuanto al número medio de ciclos por instrucción, éste es un término relativamente crítico que se ve afectado por la filosofía seguida en el diseño de la arquitectura.

El término relevante para esta tesis, pues depende de la implementación, que realmente puede verse fácilmente modificado, y con el que se debe tener especial cuidado a la hora de realizar modificaciones en el diseño, es el término $\frac{\text{Tiempo}}{\text{Ciclo}}$. En el caso de querer introducir alguna modificación en la microarquitectura, deberá ponerse especial cuidado en que este término no se vea afectado por la introducción de esa modificación. En caso de verse afectado, sólo se debería mantener dicha modificación si, y sólo si, el rendimiento total del sistema mejora.

Este último punto es de especial interés cuando se utiliza un microprocesador, tanto cuando se está diseñando para un propósito general como si se va a utilizar en un sistema empotrado como es el objeto de esta tesis.

2.12. DSPs y Procesadores de Señales de Aplicación Específica, ASSP

Existen circunstancias en las que, debido a las características de la aplicación, la naturaleza de las tareas que van a coexistir e interrelacionarse son tan diversas que no cabe otra solución que decantarse por un procesador. Sin embargo, para una mayoría de las tareas podría convenir que el procesador tuviera cierto tipo de estructuras que facilite la ejecución de determinados métodos de procesado. En este sentido el procesador está orientado a un *dominio* específico de aplicación [BCF+97].

En este tipo de procesadores la arquitectura puede tener ciertas estructuras que permitan una ejecución acelerada de determinados cálculos necesarios en ciertos tipos de aplicación. Esta alternativa es la que ha dado origen, entre otros, a los procesadores para el tratamiento digital de señales, o DSPs. A modo de ejemplo, en los DSPs normalmente se dispone de un multiplicador para acelerar las multiplicaciones y, en algunos casos, *hardware* específico para dar soporte a la indexación de datos consecutivos. La disposición consecutiva de datos es una organización típicamente utilizada para almacenar tablas de datos (que normalmente proceden de señales que evolucionan continuamente con el tiempo). De este modo, la ejecución de estas subtareas se realizan en menor tiempo, y además, en determinandos casos, se prescinde de destinar circuitería específica externa para realizarlas, con lo que se evita este eventual aumento del coste del sistema, y se logra un mejor rendimiento global del sistema.

Un ejemplo de desarrollo de núcleos DSP por el autor de este trabajo es [Bau92, BNCS92]. En el grupo se viene trabajando con la familiad TMS320 de *Texas Instruments*. La información puede consultarse en [DSP].

Un problema importante en este ámbito es la decisión sobre la longitud de palabra del procesador. Existen herramientas [fro] y métodos [SK95, SVR+97, CRS+98] para

obtener y refinar implementaciones en punto fijo desde análisis de alto nivel en MATLAB o C++ en coma flotante.

Una amplia documentación sobre núcleos DSP disponibles para integración en sistemas en un chip puede consultarse en [Bie95]. Es recomendable la consulta de guías de vendedores de núcleos DSP, por ejemplo, en [ISD].

Ejemplos de incorporación de núcleos DSP modificados para arquitecturas de procesadores de señal de aplicación específica (ASSP) se dan en [SM95, Mad95]. El programa RASSP de ARPA mantiene información de sus resultados en [RAS].

2.13. Procesadores específicos de vídeo

La carga computacional del procesamiento multimedia está dominada por las tareas de procesado de vídeo [Pir98, Ack93, Ack94, Ack95, Núñ95]. Estas tareas requieren realizar operaciones complejas sobre grandes volúmenes de datos a elevadas velocidades de muestreo. Aparecen requisitos de tiempo real para satisfacer las exigencias de la percepción visual humana. Además, se necesita manejar flujos (cada vez con más frecuencia denominados *streams* y *stream processors* los procesadores que operan con ellos) de diferentes tipos de datos.

La viabilidad de muchas aplicaciones multimedia depende de esquemas de compresión que faciliten la transmisión y almacenamiento de datos multimedia. Entre los estándares de compresión existentes pueden mencionarse los H.261, H.263, H.263++ y H.26L del ITU-T, que cubren aplicaciones de comunicaciones como videotelefonía; el MPEG-1 de ISO, que se emplea en almacenamiento y reproducción CD-ROM; y el estándar más genérico MPEG-2, que se dirige a aplicaciones como difusión de TV o como vídeo bajo demanda. El estándar MPEG-4 de ISO, actualmente en desarrollo, utiliza una codificación más eficiente, y ofrece mayor funcionalidad como la integración de fuentes sintéticas y naturales o la interacción del usuario basada (o guiada) en los contenidos.

La literatura científica sobre arquitecturas de procesadores específicos de vídeo es muy amplia. En el breve resumen que hacemos seguimos la clasificación de [Fer98] y de [SPM98] para los aspectos VLIW más recientes.

Al ser los algoritmos multimedia notablemente sofisticados, el éxito comercial de las aplicaciones descansa en su eficiente ejecución en procesadores estándares de altas prestaciones o en su implementación VLSI. Los procesadores estándares actuales (RISCs, superescalares, etc.) no pueden procesar aplicaciones multimedia de tipo general sin ser adaptados. Estos procesadores no están orientados al procesado de señal, al procesado de *streams*, y además son demasiado caros y con demasiado consumo para el mercado de aplicaciones aisladas multimedia, el campo de mayor crecimiento en los próximos años. A su vez, los DSPs convencionales, aunque están orientados y optimizados para el procesado de audio y voz, no llegan a las prestaciones que requiere el vídeo. En consecuencia, se están desarrollando arquitecturas adaptadas a vídeo derivadas de procesadores y DSPs estándares, y arquitecturas específicas, en ambos casos manteniendo objetivos de bajo coste. Existe un proceso de convergencia entre ellas. Las primeras

suelen orientarse a aplicaciones de bajo régimen binario, o bien de decodificación en tiempo real, y los trataremos en la siguiente sección. Las segundas suelen afrontar el reto de codificar MPEG-2 en tiempo real, así como los perfiles y niveles superiores del estándar. La integración monolítica de un decodificador MPEG MP@ML data de 1994 [Tho94]. Hoy en día empresas como *Thomson*, *LSI Logic*, *C-Cube*, *IIT* o *IBM* disponen de estos decodificadores, cuyo mercado principal son las cajas de usuario (*set-top box*) de los equipos conectados al aparato de TV. Se han comenzado a anunciar codificadores MPEG-2 MP@ML, como los DV^X5110 y DV^X6210 [CCu97].

Las arquitecturas específicas se clasifican en fijas y programables. Las fijas suelen estar formadas por un elemento de control y varias unidades específicas dedicadas a diferentes partes del algoritmo, y adoptan una arquitectura tipo ASIP o ASSP. Las específicas programables, denominadas VDSP o VP, están formadas por una o varias rutas de datos programables con estructuras menos específicas. Presentan mayor flexibilidad y utilidad conforme evolucionan los estándares. Los estándares varían en resolución de imagen, tasa binaria a la salida, sintaxis de la trama, márgenes de variación de diferentes parámetros, algoritmos de interpretación, carga computacional, etc., pero muchas estructuras de cálculo son comunes.

El VRP (*Video RISC Processor*) CL4000 [Bur93] es un ASIP escalable. Con dos VRPs se puede codificar MPEG-1 en formato SIF. Con 10 VRPs se codifica MPEG-2 MP@ML [BK95]. Un sólo VRP2 CL4100 codifica MPEG-1 en tiempo real.

El VideoFlow [Lee94] codifica con dos chips H.261 en CIF o MPEG en SIF. Los Enc-C y Enc-M de NTT [Kon96, Ike96] codifican MPEG-2 SP@ML. Todas estas arquitecturas son ASIP.

Entre las arquitecturas específicas *heterogéneas* VDSP se encuentran los VC (*Vision Controller*), VP (*Video Processor*) [Bai92] y VCP [IIT93] de IIT, el AVP1300E para H.261 ó los AVP1400E y AVP4310E [Ack93] para mayores niveles, de *Lucent Technologies*, los VDSP [Aon92] y VDSP2 [Ara94, Toy94, Aki94] de *Matsushita*, para MPEG o el *chip-set* VISC de *LSI Logic* [LSI96] para MPEG-2.

Entre las arquitecturas específicas programables VDSP, es encuentra el VSP3 [Ino93, Eno93] de *NEC*, que codifica H.261 en CIF, o el HiPAR-USP [RK96] que puede codificar MPEG-2 MP@ML sin estimación de movimiento. Estas arquitecturas son *homogéneas*; en ellas los coprocesadores trabajan sobre una parte de la imagen.

La tendencia arquitectural a acoplar cada vez más fuertemente los coprocesadores o unidades funcionales con la misma unidad de control ha llevado a las arquitecturas monoprocesador de palabra larga (LIW) y muy larga (VLIW) [NC89, FDF98]. Estos procesadores son idóneos para aplicaciones multimedia, específicamente para procesado de imagen y gráficos, y, en menor medida, vídeo [DWWO96, MT97]

Los tres represantes más señalados son, quizás, la arquitectura VelociTI, de *Texas Instruments* [Ses98, TMS], la arquitectura Mpact2 de *Chromatic Research* [Pur98] y la arquitectura Trimedia de *Philips Semiconductors* [RS98]. Trimedia ha sido especialmente concebido para la decodificación de MPEG-2 en tiempo real, y probablemente ofrece las mejores prestaciones.

2.14. Procesadores estándares adaptados con soporte de vídeo

El tiempo empleado en ejecutar aplicaciones de procesado de vídeo en los procesadores convencionales superan los límites razonables. Por esta razón han comenzado a emplearse otros enfoques en la construcción de sistemas multimedia.

Los diseñadores de arquitecturas han observado que el tamaño de los tipos de datos que se proporcionan en los procesadores resulta excesivo para alojar la estructura y el rango dinámico de la información de tipo multimedia con la que se necesita operar. Al almacenar una información de 8 bits en las palabras disponibles de 32 ó 64 bits se desperdicia espacio. Sin embargo, en una palabra de información de 32 bits se pueden ubicar simultáneamente 4 datos de 8 bits en el mismo espacio. En los procesadores que trabajan con datos de 64 bits en este espacio se pueden alojar 8 datos de 8 bits ó 4 de 16. De esta forma, en un número menor de unidades de almacenamiento se puede almacenar mayor información. Si se tratan estos formatos de datos como válidos y se modifica la ruta de datos para realizar operaciones sobre cada uno de los distintos campos de forma concurrente, entonces se puede acelerar la ejecución de las aplicaciones de forma significativa. Estas nuevas operaciones emplean las mismas técnicas SIMD que se emplean en algunos multiprocesadores, donde una misma instrucción se ejecuta sobre datos diversos. La aceleración lograda permite la codificación y decodificación de vídeo en aplicaciones de bajo régimen binario, y puede lograr igualmente decodificación MPEG de moderada definición en tiempo real.

Inspirados en estas ideas y objetivos, *Hewlett-Packard* diseñó el PA-7100LC, que incorpora un pequeño juego de instrucciones multimedia denominado MAX-1 [Lee95]. Tras esto *Sun Microsystems* introdujo este tipo de extensiones especiales para aplicaciones multimedia en sus microprocesadores UltraSPARC. A esta extensión del juego de instrucciones se le denominó *Visual Instruction Set*, o VIS [KMT+95, TONH96].

Posteriormente *Hewlett-Packard* desarrolló el MAX-2 [Lee96], que consistía en el MAX-1 con instrucciones añadidas para alineamiento de datos y mayor paralelismo en las subpalabras. Asimismo *Intel* incorporó el juego de instrucciones MMX en sus procesadores *Pentium* [PW96]. Estas extensiones eran muy similares a las anteriores, incorporando además instrucciones de multiplicación de subpalabras en paralelo.

Otros fabricantes que han empleado técnicas similares han sido *Silicon Graphics*, que introdujo MDMX [MIP97], y *Digital*, que ha anunciado un pequeño juego de instrucciones MVI para el Procesador *Alpha 21264* y especialmente ideado para acelerar el algoritmo *MPEG-2*.

Para el presente trabajo se han tomado estas extensiones como referencia, y especialmente el VIS empleado en los procesadores UltraSPARC. En estos procesadores este juego de instrucciones es desarrollado en la FPU. Sin embargo, en el campo de aplicación contemplado que es el entornos multimedia de bajo coste —donde no se hace necesario emplear una FPU [BMCN96]—, y también siguiendo el criterio de *Hewlett-Packard*, se ha optado por realizar las operaciones para multimedia en la Unidad de Enteros.

La práctica totalidad de procesadores estándares nuevos se están desarrollando con ciertas capacidades de procesado de vídeo en tiempo real. Al haber superado los 300 MHz de reloj, muchos procesadores estándares pueden procesar vídeo de bajas prestaciones. A estos procesadores con extensiones en su juego de instrucciones a veces se les denomina como NSP (*Native Signal Processors*) [LBSL97, Lap95].

El presente trabajo de tesis se ha centrado en este tipo de procesadores, y en particular sobre la arquitectura SPARC con sus extensiones VIS, para la realización de sistemas empotrados para multimedia, a baja tasa binaria, y a más alta tasa (MPEG-2) con múltiple escalaridad [HYTU96, VIS].

CAPÍTULO 3

La arquitectura SPARC y la extensión VIS para multimedia

3.1. Introducción

SPARC (*Scalable Processor Architecture*) es una arquitectura a nivel de juego de instrucciones pensada para ser ejecutado por RISCs. Por lo tanto, SPARC define una arquitectura y no una implementación particular. Al contrario, posibilita que se puedan desarrollar múltiples implementaciones con diferentes relaciones precio-prestaciones. Es en este aspecto donde los creadores de esta arquitectura incidieron al calificarla como *escalable*.

Esta arquitectura fue definida por *Sun Microsystems, Inc.* entre 1984 y 1987. Sin embargo, y para adaptarse a nuevas aplicaciones que surgen en la evolución de la aplicación de los procesadores, esta arquitectura está sujeta a nuevos cambios que dan lugar a la definición de distintas versiones. La última, la versión 9, apareció en 1994 [WG94].

Las influencias fundamentales de las primeras versiones de esta arquitectura son los diseños de RISC-I y RISC-II [Kat85] y SOAR [Pen85]. Basándose en las características de estas arquitecturas, se diseñó la SPARC, donde aparecieron numerosas novedades, entre las que podemos encontrar algunas para dar apoyo a sistemas multiprocesadores, y para realizar operaciones con coma flotante y con coprocesadores acoplados.

Además, la definición de esta arquitectura se hizo completamente abierta. De este modo que se animaba a otras compañías a hacer distintas implementaciones con esta arquitectura. Esta filosofía rompía con la existente hasta entonces, que hacía que las compañías fueran las propietarias de las arquitectura y que aquellos que quisieran hacer desarrollos sobre ellas tuvieran que pagar derechos.

SPARC define las instrucciones, la estructura de registros y los tipos de datos para una Unidad de Enteros (IU) y una Unidad de Coma Flotante (FPU). Además, asigna códigos de operación para un coprocesador opcional (CP), del que se define un interface pero no su estructura interna. En cada uno de estos elementos se consideran registros de propósito general y registros de estado.

En el caso de la versión 8, las instrucciones de carga y almacenamiento trabajan con un espacio de direccionamiento de 2^{32} bytes, lo que supone que se puede acceder a 256 espacios de direccionamiento distintos de 4 Gbytes cada uno. Para la versión 9, las direcciones son de 64 bits. El tamaño del bus de datos depende de la implementación.

Las instrucciones que se definen en la arquitectura SPARC son sencillas, todas ellas de 32 bits y organizadas en 3 formatos las de la versión 8 y en 4 las de la versión 9. La gran mayoría de ellas se ejecutan en un solo ciclo. Las instrucciones para la Unidad de Enteros, la Unidad de Coma Flotante y el Coprocesador se pueden ejecutar en paralelo, y la arquitectura está diseñada para poder ejecutar procesos de modo concurrente y atender excepciones (o *traps*).

3.2. El fichero de registros

Del diseño del RISC-II [Kat85] se heredó, entre otras cosas, las ventanas de registros superpuestas. Con este diseño *especial* del fichero de registros se logran disminuir los accesos a la memoria de datos, ya que disminuye el número de operaciones de carga y almacenamiento en los cambios de contexto y en las llamadas y retornos de procedimientos.

Los registros de propósito general de una IU están organizados en grupos o *ventanas* de 32 registros de 32 bits en la versión 8, o de 64 en la 9. En un determinado instante, un proceso puede acceder a cualquiera de los 32 registros de una determinada ventana, de los que 8 son accesibles desde las demás ventanas (por lo que se denominan *globales*). La ventana a la que se puede acceder, llamada *activa*, viene determinada por el contenido del campo CWP (*Current Window Pointer*, o puntero de la ventana vigente) del PSR (*Processor Status Register*, o registro de estado del procesador). Sin embargo, las ventanas tienen ciertas superposiciones (fig. 3.1). En conjunto, los registros accesibles por un determinado proceso se subdividen en:

- 8 registros de entrada (del r_{24} al r_{31}), los cuales son accesibles desde una ventana anterior,
- 8 registros locales (del r_{16} al r_{23}), exclusivos para la ventana vigente,
- 8 registros de salida (del r_8 al r_{15}), accesibles para la ventana siguiente, y
- 8 registros globales (del r_0 al r_7), visibles desde todas las ventanas.

Al igual que en la arquitectura RISC-II [Kat85], el procedimiento llamante pasa los parámetros al procedimiento llamado a través de sus registros de salida (del r_8 al r_{15}),

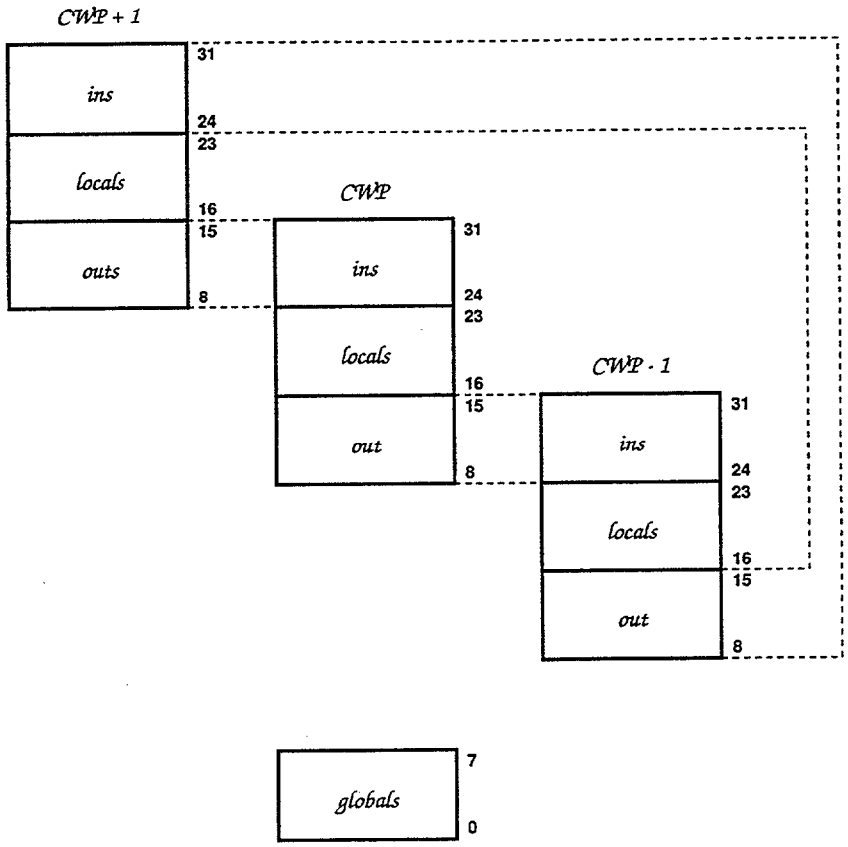


FIGURA 3.1: Tres ventanas de registros solapadas y registros globales (©Sun Microsystems, Inc., 1987).

que son los registros de entrada del procedimiento llamado. De este modo se evita tener que pasar el contenido de estos parámetros a través de la memoria de datos externa, lo que conlleva un ahorro de tiempo. Esta característica suele ser especialmente interesante en entornos donde pueden coexistir diversos procesos, como es el caso de muchos sistemas empotrados.

El número de ventanas existentes en diferentes implementaciones de la arquitectura SPARC oscila entre 2 y 32. Esto supone un número de registros de propósito general en la Unidad de Enteros de 48 a 548, respectivamente.[1] A este conjunto de registros organizados en ventanas se le denomina *fichero de registros enventanado.*

3.3. Otros elementos de la arquitectura

Además de los registros *enventanados* de propósito general, existen otros que son de propósito especial. Entre estos podemos contabilizar los siguientes:

- Registro de Estado del Procesador, PSR.
- Registro de Máscaras de Ventana No Válida, WIM.

[1]En aspectos como éste, en los que el número de un determinado tipo de elementos se deja a la elección del fabricante es donde interviene el concepto de "escalabilidad."

- Registro de Base de las Excepciones, TBR.
- Registro de Multiplicación Y.
- Registro de Estado Auxiliar, ASR.

En concreto, un determinado bit del PSR permite que distinguir dos modos de operación del *software* sobre un procesador SPARC: en modo usuario y en modo supervisor. SPARC no determina que en todas las posibles implementaciones se deba ejecutar el mismo *software* de supervisor. Este se podrá adaptar a las necesidades del sistema.

3.4. Interface con el exterior

También se permite en la definición de SPARC un amplio abanico de sub-arquitecturas de entrada-salida, unidades de gestión de memoria (MMUs) y sistemas de *caché*. SPARC presupone que estos elementos están especialmente adaptados a las necesidades específicas de la aplicación del sistema. La arquitectura SPARC no establece que sólo se pueda usar un tipo de MMU para todas las implementaciones, sino que deja a los diseñadores las decisiones sobre el tipo de MMU que se adapte mejor a las necesidades, o sobre la no utilización de éstas.

Sin embargo, si bien la especificación de la arquitectura establece distintas posibilidades en la organización de la memoria, existen algunas organizaciones que se deben encontrar indefectiblemente en todas las implementaciones de la SPARC.

La especificación de la arquitectura también establece un método para atender a las interrupciones tanto externas como internas —que se deben al *software*—. De este modo se atienden tanto las peticiones de atención por parte de elementos exteriores como el manejo de situaciones extrañas en la ejecución de los programas. Las peticiones que realizan los elementos externos se conocen como *interrupciones* y las debidas al *software* se conocen como *excepciones* o *traps*.[2]

3.5. Características

Como principales características de la arquitectura SPARC versión 8 podemos citar las siguientes:

- Espacio de direcciones lineal de 32 bits.
- Pocos formatos de instrucción y sencillos. Todas las instrucciones son de 32 bits. Existen formatos de instrucción básicos y presentan una posición uniforme de los campos de código de operación y direccionamiento de registros.
- Pocos modos de direccionamiento. La dirección a memoria se da o bien por `registro+registro` o bien por `registro+inmediato`.

[2] A las excepciones también se les conoce como *interrupciones software*.

- Direccionamiento de registros triádico. La mayoría de las instrucciones trabajan con dos operandos de registros (o un registro y una constante) y depositan el dato en un tercer registro.

- Un fichero *enventanado* de registros de enteros. En cualquier instante un proceso puede acceder a 8 registros globales de enteros y a una ventana de 24 registros que se hayan en este *fichero de registros.*

- Un fichero de registros de números de coma flotante separado. Se puede configurar por software para almecenar números de precisión simple (32 bits), de precisión doble (64 bits), de precisión cuádruple (128 bits), o una mezcla de éstos.

- Transferencia de control retardada. El procesador siempre coge la instrucción posterior a una de salto retardado. Su ejecución depende del valor de un bit de la palabra de instrucción de salto. Este tipo de saltos surge como medio de evitar penalizaciones en la segmentación debido a los saltos.

- Gestores de *traps* rápidos. Los *traps* se gestionan con una tabla de vectores y se les asigna una ventana de registros sin usar en el fichero de registros.

- Instrucciones con etiquetas o *tags.* Suponen que los dos bits más significativos (MSBs) de los operandos corresponden a etiquetas suyas.

- Instrucciones de sincronización en sistemas multiprocesadores.

- Unidad de Coma Flotante (FPU) y Coprocesador (CP). La arquitectura define un conjunto de instrucciones para una FPU y otro para un CP.

Además, en la versión 9 se introdujeron novedades como:

- Saltos predichos. Se permite que el programador o el compilador cree instrucciones de salto condicional con una indicación sobre la probabilidad de si el salto se tomará o no.

- Instrucciones de eliminación de salto. Se pueden utilizar varias instrucciones para eliminar los saltos.

- Pila *hardware* para las excepciones. De este modo se permiten las excepciones anidadas y se realiza el manejo de las condiciones de error y fallo más simple, rápido y seguro.

3.6. SPARC para sistemas empotrados

SPARC es una alternativa muy atractiva para utilizarla como referencia en el diseño de núcleos para sistemas empotrados [CvC91, OGN99]. Algunas de las razones más destacables para utilizar SPARC en este tipo de sistemas son las siguientes:

- SPARC es una arquitectura abierta y escalable, lo que deja al diseñador con un alto grado de libertad. Existe un amplio abanico de alternativas de diseño dependiendo de las prestaciones que se desean.

- SPARC presupone la existencia en su arquitectura de un fichero de registros enventanados y solapados. Esta característica es muy atractiva en aplicaciones donde coexistirán tareas de naturaleza diversa y se necesitan cambios de contexto rápidos. Esto puede permitir en los sistemas empotrados mayor margen de maniobra en el diseño al cumplir con las imposiciones de tiempo real.

- Con SPARC se puede conseguir una atención bastante rápida a las interrupciones que soliciten los elementos externos del procesador. Algo que facilita esto es, entre otras cosas, el fichero de registros enventanado.

- SPARC posee extensiones específicas para sistemas empotrados en las especificaciones de la arquitectura.

- Existe bastantes herramientas de desarrollo de *software* para SPARC que puede ayudar en el diseño, compilación eficiente y depuración de los programas que se ejecutarán en estos núcleos.

- Los procesadores con arquitectura SPARC son muy utilizados en WANs. Esto es especialmente significativo, ya que aparte de indicar claramente que SPARC se puede emplear con éxito en situaciones en las que el procesador debe manejar tareas de diversa índole, la capacidad de comportarse con esta eficiencia de redes de ordenadores tan grandes lo hace especialmente atractiva para aplicaciones donde se necesita comunicaciones de altas prestaciones.

- La introducción de sistemas operativos para sistemas empotrados, como el Windows CE, abre el área a nuevas aplicaciones multimedia para dispositivos portátiles basados en procesadores no-Intel. Existe un soporte amplio de otros sistemas operativos, incluyendo algunos de tiempo real, para plataformas SPARC.

3.7. Implementaciones de referencia de SPARC como componentes

En la tabla 3.1 se pueden observar las características de algunas implementaciones realizadas por otros fabricantes. Se trata de versiones *full-custom* para producción en volumen como componentes estándares, salvo las señaladas con un '*'.

A efectos comparativos, en la presente tesis la versión sintetizada que ha ofrecido la mejor velocidad ha sido de 93 MHz con tecnología CMOS de 0.35 μm. Hay que indicar que esta velocidad es la estimada por la herramienta tras el colocado, ruteado y trazado físico completo del núcleo. La estimación corresponde a un peor caso esperable en la producción de los componentes.

TABLA 3.1: Características de implementaciones de Unidades de Enteros de SPARC

Empresa	Modelo	Velocidad	Tecnología
Cypress	CY7C601	25, 33, 40	0.8 μm CMOS
Weitek	8701	33, 40	0.8 μm CMOS
Fujitsu	MB86930*	40	0.8 μm CMOS
Fujitsu/ Sun Microsystems	MB86904 (microSPARC-II)	60, 70, 85	0.5 μm CMOS
Fujitsu	MB86907 (TurboSPARC)	160, 170	ND
Bridgepoint/Ross	RT620/RT625	80 – 200	ND
LSI Logic	L64801	20, 25	ND
LSI Logic	L64811	40	ND
SIDSA	IDeAS*	33	0.7 μm CMOS
Temic Semiconductors	TSC691E	14, 25	0.8 μm CMOS
Texas Instruments	TMS390S10 (microSPARC)	50	0.8 μm CMOS
Texas Instruments	TMS390Z50 (SuperSPARC II)	60 – 85	0.8 μm BiCMOS
Texas Instruments/ Sun Microsystems	UltraSPARC	140 – 600	0.07 μm CMOS

3.8. VIS: Visual Instruction Set

El VIS [TONH96, Ric96, Sun97, Sun] de UltraSPARC es un conjunto de instrucciones optimizadas para operar con tipos de datos que se emplean en algoritmos de multimedia: valores en punto fijo de 8, 16 ó 32 bits. En el campo de los gráficos y las imágenes en movimiento, la información de los *pixels* consta normalmente de cuatro enteros sin signo de 8 bits cada uno, ocupando palabras de 32 bits. Por ejemplo, la información de color normalmente se representa con valores de 8 bits para la componente de rojo, la de verde y la de azul, y un valor de 8 bits para un coeficiente adicional de transparencia. Los datos de punto fijo consisten en cuatro componentes de punto fijo de 16 bits o dos de 32 bits y en ambos casos ocupan palabras de 64 bits. Los algoritmos suelen usar estos tipos de datos para resultados intermedios en los procesamientos donde se necesitan una mayor precisión o un mayor rango dinámico. Algunas instrucciones también emplean tipos de datos fijos con ocho componentes de 8 bits. UltraSPARC almacena estos datos en el fichero de registro de la FPU.

Las instrucciones incorporan las operaciones fundamentales sobre estos datos que se encuentran en la mayoría de los algoritmos de gráficos y multimedia. Las instrucciones de VIS se pueden clasificar en:

- de conversión,
- aritmético-lógicas,
- de manipulación de direcciones,
- de acceso a memoria, y
- de estimación de movimiento.

3.8.1. Aspectos sobre el fichero de registros

Los datos de multimedia con los que operan las instrucciones VIS residen en el fichero de registros de la FPU. Los operandos de las instrucciones en el fichero de registros de la FPU se pueden referir a registros de precisión simple o de precisión doble. Los operandos en el fichero de registros de la FPU se pueden referir a registros de precisión simple o doble. Muchas instrucciones tienen formas que emplean como operandos datos de precisión simple de 32 bits o datos de precisión doble de 64 bits. La arquitectura SPARC versión 9 [WG94] define 32 registros de precisión simple y 32 registros de precisión doble. Los 32 registros de precisión simple se corresponden con los primeros 16 registros de precisión doble. Los operandos de precisión simple no pueden acceder los últimos 16 registros de precisión doble.

A menudo esta disposición le permite al procesador acceder y actualizar las mitades superior e inferior de 32 bits de un registro de 64 bits de doble precisión de forma independiente haciendo uso de los registros de precisión simple correspondientes. Debido a que los algoritmos realizados con VIS suelen emplear esta característica, el juego de instrucciones VIS contiene operaciones que acceden y actualizan de forma eficiente las mitades superior e inferior de un registro de 64 bits.

3.8.2. Instrucciones de conversión

Las instrucciones siguientes realizan conversiones de los datos entre los distintos formatos: `fexpand`, `fpack16`, `fpack32`, `fpackfix` y `fpmerge`.

La instrucción `fexpand` convierte valores de 8 bits en valores de 16, mientras las instrucciones `fpack` convierten datos de 16 o 32 bits en tipos de datos de menor precisión. Todas estas instrucciones realizan las operaciones apropiadas de escalado, cortado y truncado.

La instrucción `fpmerge` intercala los cuatro valores de 8 bits que proceden de dos operandos de 32 bits para construir un resultado de 64 bits, que consiste en 8 componentes de 8 bits cada uno. Esta instrucción es útil para conversiones entre los formatos de banda entrelazada y banda secuencial. También es útil al rotar o trasponer matrices de dos dimensiones.

3.8.3. Instrucciones lógicas y aritméticas

Las instrucciones de suma y resta fraccionada —`fpadd16`, `fpadd32`, `fpsub16` y `fpsub32`— realizan sumas o restas en componentes de 16 ó 32 bits. Por ejemplo, la suma fraccionada de 16 bits (`fpadd16`) toma dos registros y suma cada una de las cuatro componentes de 16 bits por separado.

Las instrucciones de multiplicación fraccionada multiplican componentes de 8 y 16 bits. Todas las instrucciones de multiplicación se basan en cuatro subunidades multiplicadoras de 8×16 bits. VIS emplea una serie segmentada de tres instrucciones para emular multiplicaciones de 16×16 bits.

La instrucción `fmul8x16` multiplica cada uno de los cuatro componentes de 8 bits de un *pixel* con los componentes de 16 bits correspondientes de una palabra de 64 bits. De cada producto la instrucción alamcena los 16 bits más significativos en los componentes de 16 bits correspondientes en el registro de destino.

La instrucción `fmul8x16au` es como la anterior, excepto que emplea el componente superior de 16 bits de un registro de 32 para multiplicárselo a cada uno de los cuatro componentes de los *pixels* de 8 bits. La instrucción `fmul8x16au` es similar a la `fmul8x16au`, pero en su lugar emplea la componente de 16 bits inferior.

Juntas, `fmul8sux16` y `fmul8ulx16` se pueden combinar con `fpadd16` para realizar cuatro multiplicaciones de 16×16 bits de componentes con signo, produciendo cuatro valores de 16 bits. Estas tres instrucciones operan de forma similar al método que se emplea sobre el papel para realizar multiplicaciones.

VIS también incluye 16 operaciones lógicas para realizar cualquiera de las operaciones lógicas a nivel de bits entre dos registros.

3.8.4. Instrucciones de manipulación de direcciones

`Alignaddr` opera en el fichero de registros de la IU. Esta instrucción suma sus dos operandos fuente y almacena el resultado habiéndole fijado previamente los tres bits inferiores a 0; los tres bits inferiores de esta suma se almacenan en un campo de un registro de estado de gráficos (GSR). Esta instrucción normalmente se emplea de forma conjunta con la instrucción `falignadata`, que concatena dos registros de doble precisión de la FPU y extrae 8 bytes comenzando desde el desplazamiento que se indica en el registro de estado de gráficos.

Las instrucciones de comparación de *pixels* `fcmp` comparan cuatro pares de 16 bits o dos de 32. La comparación produce una máscara de 4 ó 2 bits, que se almacena en un registro de destino de la IU y que se puede emplear posteriormente para controlar almacenamientos parciales. Las instrucciones de comparación realizan cuatro comparaciones básicas: 'igual', 'distinto', 'menor que o igual' y 'mayor que'.

Las diferentes versiones de instrucciones de borde también generan máscaras apropiadas para la instrucción de almacenamiento parcial. Esto elimina la necesidad de código especial para manipular los límites no alineados en los algoritmos de *rendering* basados en inspección de líneas. Las diferentes versiones generan máscaras para componentes de 8, 16 ó 32 bits, con ordenación de la memoria tipo *big-endian* o *little-endian.*

3.8.5. Instrucciones de aceso a memoria

Las instrucciones de almacenamiento parcial emplan la máscara especificada en un registro de la IU para seleccionar qué componente escribir en la dirección especificada. Los lugares no seleccionados por la máscara permanecen sin modificarse. Los almacenamientos parciales operan en combinación con las instrucciones de comparación de *pixels* o con las de manejo de bordes para eliminar código ineficiente y lleno de saltos. La

codificación de las instrucciones proporciona una combinación completamente ortogonal para los modos *little-endian* y *big-endian* y para componentes de distintos tamaños.

Las instrucciones de carga y almacenamiento de palabras cortas de la FPU transfieren 1 ó 2 bytes de la memoria a o desde el fichero de registros de la FPU. La dirección debe estar alineada con el tamaño de la transferencia. Los datos de una carga tiene una extensión de ceros en el registro de destino. Los almacenamientos acceden al byte o a los dos bytes menos significativos. Estas instrucciones operan en combinación con `faligndata` para ensamblar 64 byts desde componentes no contiguos en memoria.

Las instrucciones de carga y almacenamiento de bloques transfieren bloques de 64 bytes entre la memoria y un grupo de ocho registros consecutivos de doble precisión en la FPU.

Muchos algoritmos de multimedia tienden a leer un bloque de datos, realizan unas pocas operaciones y escriben los resultados. Ya que los conjuntos de datos suelen ser mucho mayores que las cachés, el algoritmo nunca reutiliza datos antes de reemplazarlos. Al proporcionar un mecanismo de alto ancho de banda para transferir los datos entre el procesador y la memoria o el *frame buffer* sin trastornar las cachés, el VIS mejora de forma considerable las prestaciones de los algoritmos multimedia intensivos en memoria.

3.8.6. Instrucción de distancia entre pixels

La utilidad principal de la instrucción `pdist` es en la estimación de movimiento, la computación principal en algoritmos como el MPEG-2 [ISO90] y H.261 [ITU90]. La estimación de movimiento realiza una búsqueda de una trama de referencia más parecida a un bloque de 16×16 *pixels* en la trama de destino. El procesador determina los datos más similares encontrando el bloque con la menor diferencia absoluta entre la trama de destino y la de referencia. La instrucción `pdist` calcula la diferencia absoluta entre las componentes de 8 bits correspondientes en un par de registros de precisión doble y acumula los valores de error. La estimación de movimiento es sólo una de las tareas a realizar en el emparejado de patrones. No obstante, `pdist` también puede acelerar con frecuencia otros algoritmos de emparejado de patrones.

3.8.7. El VIS en la Unidad de Enteros v. 8

Al tratarse ésta de una arquitectura de 32 bits, se ha hecho necesario realizar algunas variaciones al VIS que se encuentra en los UltraSPARC (que son de 64 bits). Las instrucciones que han resultado de esta adaptación se pueden encontrar en [Qui99]. En la tabla 5.15 y en la fig. 5.35 se muestran los resultados de las implementaciones de la Unidad de Enteros desarrollada con este juego de instrucciones integrado. Para la versión `av753` se ha empleado el módulo de VIS desarrollado en [Qui99] y en la versión `aw753` el módulo VIS tiene la misma funcionalidad pero presenta algunas mejoras microarquitecturales que permiten eliminar este elemento de la ruta crítica.

CAPÍTULO 4

Modelado, microarquitectura y diseño del procesador

4.1. Introducción

Para el diseño del núcleo procesador de la Unidad de Enteros se ha partido de las especificaciones que fija el manual de la arquitectura [SPA91]. En la exposición que se realizará a continuación se necesitará hacer énfasis en los aspectos arquitecturales que han resultado ser los más singulares que se han encontrado en el diseño del procesador. Se ha tomado como marco de referencia en cuanto a conceptos, definiciones y términos en arquitectura de procesadores los establecidos en textos clásicos [HP96, PH94a].

Si bien las decisiones arquitecturales son las consideraciones fundamentales en la construcción del diseño, la complejidad de este diseño ha impuesto una metodología de trabajo donde el modelado se ha erigido como una herramienta fundamental para la toma de decisiones. A través del modelado se fijan las primeras tentativas de diseño pero también se detectan posibles errores de concepción y se evalúan alternativas. La estrategia de modelado que se infiere de la metodología seguida permite hacer un seguimiento sencillo del funcionamiento final. El modelado resulta importante, tanto para facilitar la toma de decisiones de diseño como para realizarlas en tiempos cortos, y permite realizar una fácil configuración, concebir la integración con otros bloques para el diseño de sistemas mayores y ponderar las decisiones realizadas integrando los modelos en sistemas de evaluación basados en software.

4.2. Modelado del núcleo del procesador

4.2.1. Necesidad de niveles de abstracción altos

Las densidades de integración alcanzadas hoy en día permiten superar el millón de transistores en un solo *chip*. Sin embargo, al principio la capacidad de integración era muy pequeña. Los entornos de diseño que inicialmante aparecieron estaban básicamente orientados a facilitar la captura y modificación de los planos físicos de los circuitos integrados. Posteriormente se percibió la necesidad de herramientas adicionales que ayudaran al diseñador en sus labores de comprobación antes de que enviase sus planos al fabricante. De este modo surgieron los primeros verificadores de reglas de diseño y simuladores y, más tarde, los capturadores de esquemáticos. A medida que las densidades de integración crecían, la complejidad de los Circuitos Integrados (CIs) aumentaba tanto en la naturaleza de sus componentes como en tamaño. La capacidad de los entornos iniciales de captura y simulación resultaba insuficiente para dar el apoyo necesario en el diseño de estos grandes microsistemas.

Debido a estos problemas comenzaron a aparecer los conceptos de *niveles de abstracción*, los primeros lenguajes de descripción de *hardware* y los simuladores de alto nivel. Desde un principio, el nivel de transferencia de registros, o RTL, resultó ser el más adecuado para describir el diseño de los circuitos digitales de cierta complejidad. En este nivel las primitivas con las que se podía trabajar eran los registros, matrices de registros, operadores, buses, multiplexores y otros [Har87]. Trabajando en este nivel de abstracción se presentaban indudables ventajas frente a trabajar en niveles inferiores como los de puertas o transistores. El trabajar en un nivel de abstracción superior reducía considerablemente la carga notacional empleada y los elementos que el diseñador contemplaba en cualquier decisión de diseño. Además, con el uso de nuevas metodologías basadas en este RTL y con las herramientas que se apoyasen en él los tiempos de diseño y depuración se reducían. Hoy día existe una indudable tendencia a seguir ascendiendo en los niveles de abstracción a medida que los niveles de integración superan el millón de transistores y los sistemas que se diseñan pueden integrar subsistemas de gran complejidad.

No obstante, desde el tradicional punto de vista de los diseñadores de estos sistemas, casi siempre ha permanecido latente la concepción basada en bloques de procesamiento de datos de mayor o menor complejidad y datos que se intercambian entre estos bloques. Este enfoque en la concepción de los sistemas casi siempre se ha empleado de una forma más o menos sistemática. Por esta razón, a la hora de implementar procesadores, la concepción realizada en RTL era tomada como punto de partida, pero el diseñador necesitaba forzosamente seguir pasos intermedios de forma *manual* hasta la obtención de los planos del diseño físico, necesarios para la fabricación. Actualmente existen herramientas con las que automatizar, en lo posible, este proceso.

Para servir como base a la concepción y manejo de estas herramientas de automatización del diseño electrónico (EDA) surgió el llamado diagrama en Y [MLD92], mostrado en la fig. 4.1. Como se puede observar, sobre este diagrama se localizan los distintos dominios en los que se puede describir cualquier sistema electrónico: comportamental,

estructural y físico o geométrico. A su vez, dentro de cada uno de los dominios existen distintos niveles de abstracción. En los más altos se concibe el sistema como un conjunto de subsistemas grandes y en los más bajos el énfasis se centra en los elementos más simples que lo constituyen.

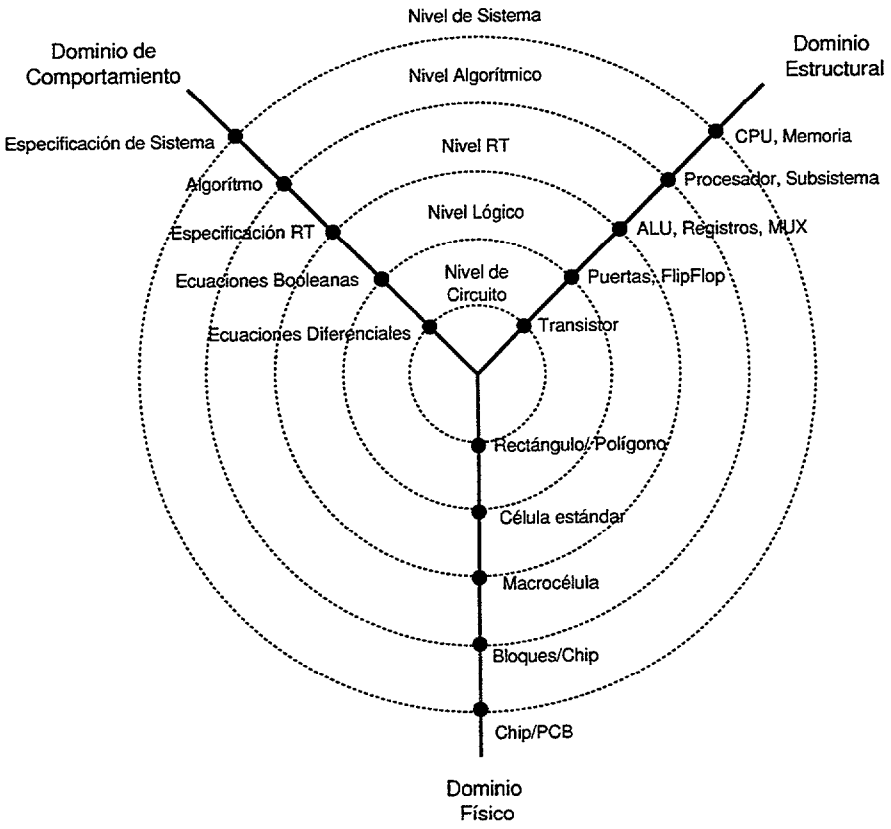


FIGURA 4.1: Dominios y niveles de descripción

Por tanto, cuanto más cerca estemos del centro de la 'Y' más próximos nos encontraremos de las fases finales del diseño. Cuanto más lejos estemos de éste, mayor abstracción estaremos utilizando en la concepción del sistema y más alejados nos encontraremos de la implementación física. Concebir un sistema como un conjunto de CPUs, elementos de memoria y periféricos no tendrá la misma complejidad que concebir el mismo sistema en términos de puertas, por ejemplo. Sin embargo, estas dos *vistas* del mismo sistema deben ser equivalentes entre sí.

4.2.2. Herramientas

Con densidades de integración inferiores a VLSI, las herramientas de los entornos de diseño trabajaban en niveles de descripción próximos a los inferiores. A medida que los sistemas que se diseñan para integrarlos en CIs han alcanzado mayores complejidades, los niveles de abstracción desde los que se necesita partir son mayores. Para permitir una migración asistida del diseño hacia niveles inferiores, y verificaciones sobre estas migraciones, han ido apareciendo herramientas que realizan diversas transiciones entre los distintos dominios y niveles, tal como se muestra en la fig. 4.2.

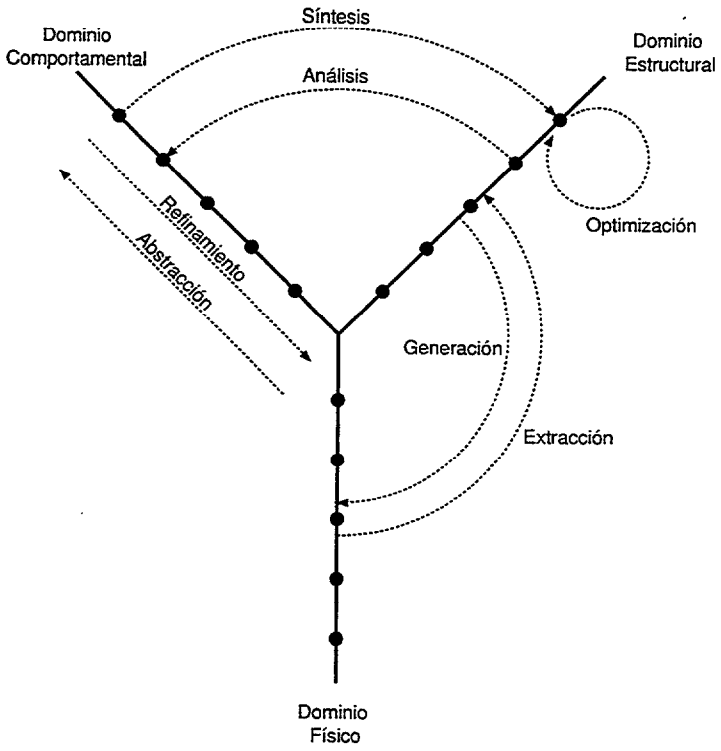


FIGURA 4.2: Transiciones posibles entre los dominios y niveles de descripción

En los niveles superiores de abstracción los aspectos relacionados con la implementación final apenas son tenidos en cuenta más que como funciones de coste y estimaciones. Sin embargo, la experiencia indica que esta afirmación a veces encuentra excepciones. Esto es debido a que para las herramientas actuales, si bien es cierto que consiguen un alto nivel de abstracción, existen determinadas restricciones en el modo de realizar el diseño impuestas por las características de los niveles inferiores. No se deben considerar estas restricciones como limitaciones a la capacidad de abstracción, pues en realidad se deben a cualidades intrínsecas del *hardware* que pueden reflejarse en varios niveles a la vez.

4.2.3. Lenguajes de descripción

Normalmente, el nivel de abstracción más aconsejable para concebir la arquitectura de un procesador suele ser el de transferencia de registros. Para permitir que una herramienta de diseño pueda recoger la visión del diseñador en este nivel, éste último debe realizar una descripción de su idea. Esta descripción se realiza apoyándose en un lenguaje de descripción del *hardware* (HDL). De este modo, las herramientas CAD a emplear podrán tener conocimiento de lo que el diseñador desea realizar y le podrán ayudar a desarrollar los procesos con los que fluir a través del diagrama en 'Y' según se mostró en la fig. 4.2.

A lo largo de los últimos años han aparecido diversos HDLs. Sin embargo, sólo algunos han terminado utilizándose con mayor profusión, y por ello han adquirido mayor soporte entre las herramientas. En el campo que nos ocupa, los dos más difundidos son

VHDL y Verilog.

VHDL (*Very-High-Speed-ICs Hardware Description Language*) es un lenguaje de descripción del *hardware* cuyo desarrollo se inició gracias a un programa del Departamento de Defensa de los EE.UU. de América. Posteriormente fue adoptado como norma del IEEE en 1987 y se revisó por última vez en 1993. Este lenguaje fue concebido para describir cualquier tipo de *hardware*, y está orientado a documentar cualquier sistema electrónico.

Por otra parte, Verilog está orientado a describir sistemas de carácter puramente digital. Se ha desarrollado en el seno de CADENCE y se fue haciendo un gran hueco entre los diseñadores de procesadores, especialmente en EE.UU., que terminaron acogiéndolo como un estándar *de facto*. Actualmente está recogido como norma del IEEE.

Para el desarrollo de este modelado se ha elegido VHDL como medio de descripción, si bien podía elegirse Verilog. Las razones de decidirse por uno o por otro lenguaje no son fáciles de establecer. De hecho, en muchos casos esta elección puede fundamentarse en criterios puramente subjetivos. Por ejemplo, existen diseñadores que tradicionalmente han trabajado con VHDL y cuando desean utilizar Verilog echan en falta determinadas capacidades de VHDL. Sin embargo, otros encuentran Verilog más fácilmente utilizable y muy cómodo, y se adaptan muy rápidamente a él. Esto podría deberse a que VHDL fue concebido no para el diseño sino para el intercambio de información sobre sistemas electrónicos de cualquier tipo, por lo que en este sentido Verilog podría tener ciertas ventajas.

Por estas razones, entre otras consecuencias sucede que los simuladores de Verilog suelen ser más rápidos que los de VHDL por construcción. Pero no es menos cierto que al usar Verilog perdemos determinadas facilidades. A modo de ejemplo, Verilog no considera el concepto de librería de VHDL, con lo que, de usar Verilog, se pueden encontrar mayores dificultades a la hora de incorporar nuestros diseños en otros mayores.

Como consecuencia de lo anteriormente expuesto, y gracias a la abstracción que permite estos lenguajes y las herramientas asociadas, los beneficios que se consiguen se pueden resumir en los siguientes puntos:

- Reutilización de funciones lógicas complejas.
- Parametrización. Se puede permitir la configuración de algunos elementos del diseño.
- Posible utilización de uno o varios niveles de abstracción en la descripción de cualquier componente del diseño.
- Independencia tecnológica. Los cambios de tecnología no afectan al comportamiento funcional del diseño.
- Gestión, evaluación y depuración más sencilla tanto de los sistemas como de los elementos de construcción en diseños complejos.
- Menores tiempos de desarrollo de los productos, con lo que se reduce drásticamente el tiempo de introducción en el mercado.

- Aumento importante de productividad.

4.2.4. Descripción del proceso de modelado de núcleos procesadores

El proceso de modelado que se ha seguido en la elaboración de núcleos procesadores de SPARC v. 8 ha ido cristalizándose simultáneamente con la elaboración del modelo global. De hecho, las líneas maestras de modelado reflejan esta experiencia. No obstante, el proceso de modelado seguido es general y puede aplicarse con resultados igualmente válidos en cualquier núcleo de tipo RISC.

Algunos de los puntos que se han establecido en este proceso de modelado son herencia tanto de un enfoque tipo *bottom-up* en el diseño de sistemas como de una visión tradicional en la concepción y comprensión de las arquitecturas de procesador. Por lo general, un diagrama de bloques de un procesador suele ser más clarificador que una descripción completa en uno o varios ficheros de texto, y ofrece mecanismos mejores para estudiar detalles específicos sobre el funcionamiento. Es más, para desarrollar o entender una arquitectura frecuentemente se toma como referencia un diagrama de bloques sobre el que identificar y estudiar todas las interrelaciones y operaciones posibles con los datos [BMCN97].

Con la intención de incorporar las ventajas que ofrece este tipo de enfoques, y tomando como puntos de apoyo las herramientas y facilidades disponibles, se ha determinado que la metodología a utilizar debía fundamentarse en el uso extensivo de un entorno gráfico. En este caso se ha elegido SGE de Synopsys. A partir de éste, el procesador se puede definir a través de un diagrama de bloques de tipo RTL. Para llegar a completar esta definición el procesador se divide en unidades menores, y a su vez éstas en otras aún menores. Esta división jerárquica tiene un límite cuando se alcance un nivel donde los elementos de construcción resulten fáciles de concebir, comprender, gestionar, depurar y modificar si resultara necesario. De esta forma se controla de modo preciso cada elemento en detalle sin perder sus interrelaciones con el resto de los elementos del núcleo. Por esta razón, el primer paso para elaborar esta metodología ha sido el desarrollo de una librería de elementos de construcción clásicos en diagramas RTL para utilizarlos dentro de SGE. Esta librería se puede extender creando elementos parametrizables RT y descripciones VHDL a nivel de flujo de datos dentro de este entorno. Específicamente para el diseño que se ha considerado, los elementos realizados que se contienen en esta librería son memorias, ALUs, sumadores, desplazadores, ficheros de registros de propósito general, registros de propósito especial, registros de segmentación y unidades de I/O.

Las descripciones de estos elementos RTL se han desarrollado intentando incorporar el mayor número posible de parámetros, para permitir que puedan reutilizarse en cualquier lugar del diseño.[1] Esto puede resultar muy atractivo para permitir determinadas opciones de parametrización del núcleo.

[1]Sobre este particular debe mencionarse que los anchos configurables se permiten en SGE, pero ocasionan problemas en algunas de las versiones de Synopsys. Si bien VHDL permite hacer algunas cosas —empleando genéricos—, en ocasiones puede que el entorno gráfico imponga ciertas limitaciones.

Los diagramas de bloques se pueden así construir interconectando copias de algunos de estos elementos con otros. En algunos casos resulta necesario introducir otros elementos que no se encuentran en la librería de elementos generales, por lo que en estos casos la potencialidad del HDL puede resultar muy beneficiosa. Estos elementos necesitan diseñarse específicamente para cubrir las necesidades de la aplicación del núcleo y son particulares de la arquitectura. Ejemplos de estas sub-unidades pueden ser el mecanismo para el manejo de las ventanas en SPARC o la circuitería para la determinación de prioridades en las interrupciones del procesador.

Con este diagrama de bloques se define la microarquitectura del núcleo entero. Los elementos específicos de la arquitectura se pueden construir o bien utilizando un nivel inferior en la jerarquía de esquemáticos o bien utilizando una descripción completa en VHDL. Las últimas descripciones en VHDL de la jerarquía deberán ser lo más simples posible, de modo que se permita una fácil depuración y eventuales modificaciones de estos elementos. En algunas ocasiones, la partición de algunos de los elementos de construcción debe realizarse buscando precisamente esta fácil gestión y depuración posterior del modelo del conjunto. Ésta ha sido, por ejemplo, la alternativa elegida en el diseño de los modelos de la lógica de control de la Unidad de Enteros. En ella se ha ganado no sólo claridad en el nivel de diagrama de bloques (frente a lo que hubiera supuesto un gran bloque monolítico basado en una única descripción en VHDL), sino que también se permite un mejor seguimiento del comportamiento del procesador y una mayor facilidad en la configuración del juego de instrucciones a interpretar. Este enfoque puede resultar igualmente válido para cualquier otro núcleo de tipo RISC.

Si, dependiendo de la aplicación a la que se desea destinar el procesador, se necesitan realizar modificaciones en el núcleo, éstas se pueden realizar en su modelo gráfico. De este modo se puede obtener en poco tiempo una descripción RT completa del nuevo núcleo modificado. Entonces debería llevarse a cabo un nuevo proceso de depuración a partir de esta nueva descripción y cualquier decisión arquitectural puede volverse a evaluar de un modo sencillo. Merece una mención especial el caso en el que se desea modificar el juego de instrucciones, ya que para evaluar los recursos que ocasionalmente dejarían de utilizarse se precisa identificarlos con una etapa posterior de procesamiento.

Ya que los elementos de construcción se han desarrollado siguiendo el subconjunto de VHDL para síntesis (recomendación IEEE 1076.3), el modelo resultante es completamente sintetizable. Entonces, además de las posibles modificaciones arquitecturales que dependan de la futura aplicación del procesador, se puede elegir la estrategia de síntesis que se considere más apropiada.

Configurabilidad de los modelos

Gracias a las técnicas y elementos empleados en el modelado del diseño, existen posibilidades de configurar distintos aspectos de éste.

En el caso de la arquitectura SPARC, ésta es una arquitectura que ofrece ciertos grados de libertad a la hora de realizar una implementación determinada. Entre estas características se pueden mencionar los tamaños de los ficheros de registros generales y auxiliares, la implantación de algunas instrucciones, el poder hacer privilegiado el acceso

a determinados recursos, el comportamiento de algunos registros auxiliares (que podrían presentar una funcionalidad especial, como, e.gr., temporizadores), el número de etapas de segmentación o el número de ciclos empleados en la ejecución de las instrucciones.

Debido a las propiedades del entorno de trabajo establecido, existen elementos del diseño que pueden ser susceptibles de configurarse de un modo sencillo, pero otros no. Esto se debe a dos razones fundamentales. Por un lado, las herramientas con las que se ha trabajado no han permitido que en ocasiones algunas características se pudieran parametrizar. En otras ocasiones se ha considerado que si se hubiese podido introducir cierta configurabilidad en determinadas características del núcleo los modelos resultantes hubieran sido extremadamente complicados y difíciles de comprender y gestionar. Un ejemplo es el número de etapas de segmentación del procesador. Si se hubiese logrado establecer un mecanismo por el cual el diseñador pudiera elegir el número de etapas a través de un parámetro tendría que poderse indicar qué funciones deben realizarse en qué etapas. Todo esto conduciría a un modelo lleno de funciones y expresiones condicionales, con las consiguientes complicaciones no sólo en su diseño sino también en su seguimiento, en la depuración y modificación, tanto del modelo original como de las versiones modificadas por el usuario.

En otras ocasiones, existen características muy sencillas de parametrizar, ya que obedecen a propiedades con una incidencia en elementos muy concretos y localizados de la arquitectura. Estas propiedades deberán indicarse a través de constantes, de forma que sean fácilmente reconocibles en la síntesis lógica como propiedades que no se pueden alterar. Entre éstas se pueden mencionar:

- Número de ventanas.
- Número de Registros Auxiliares.
- Identificador de versión e implementación.
- Indicador de ciclo adicional para los almacenamientos[2].

También existen otras opciones de parametrización, pero éstas se deben a cuestiones puntuales de funcionamiento. Por lo tanto, a éstas sólo se podrá acceder modificando explícitamente el modelo del elemento en cuestión. En ocasiones esto se ha realizado así porque la solución alternativa conducía a resultados no óptimos. El establecimiento de las prioridades de los distintos tipos de excepciones es un ejemplo de esto. Los elementos asociados a estas propiedades son fácilmente reconocibles en el modelo del procesador, ya que estos elementos toman sus nombres de la función concreta que tienen en el diseño. Por ejemplo, al elemento que indica si existe petición de excepción y cuál gestionar en cada caso se le ha llamado `trapsadvisor`.

[2]Este ciclo adicional se precisa en los sistemas con memorias caché.

4.3. Microarquitectura y diseño del procesador

Siguiendo la organización típica que se suele emplear en el estudio de las arquitecturas de procesadores tradicionales, la primera partición que se ha empleado para el diseño del procesador ha sido identificar dos secciones diferenciadas: una sección de procesamiento de datos o *ruta de datos* y una sección de control [BN99a].

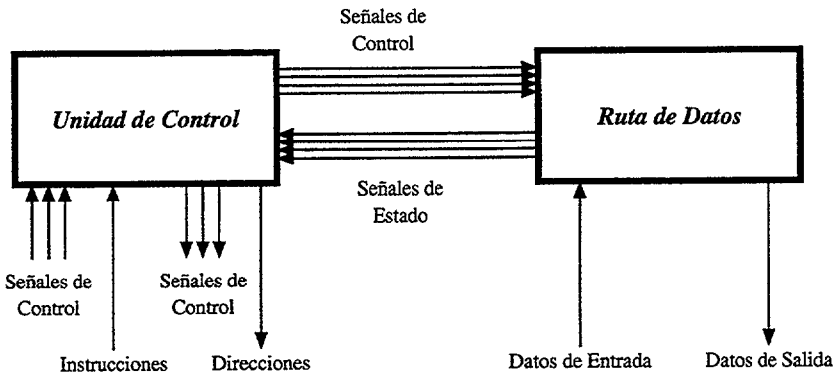


FIGURA 4.3: Estructura del procesador

En la ruta de datos se realizan las operaciones sobre los datos, los cuales procederán o bien de otros recursos de la ruta de datos o bien de una memoria o de un elemento de I/O exterior. En el caso que nos ocupa, cual es el de un RISC (véase apartado 2.11, pág. 26) estos datos se introducen en el procesador a través de instrucciones '*load*' que buscan los datos en las memorias externas y los introducen en los registros internos, para posteriormente operar con ellos, y en su caso almacenan los resultados en memoria con instrucciones '*store*'.

Por otra parte, la unidad de control es la encargada de tomar las decisiones sobre cuáles son las operaciones a realizar y en qué orden. Para ello toma las instrucciones de una memoria externa, y señales tanto del exterior (entre otras, las interrupciones) como de la ruta de datos. Estas últimas nos indicarán algunas condiciones y propiedades de los resultados de operaciones previas. Con todos estos condicionantes se decide en cada momento cuál es la operación que debe realizarse a continuación y cuáles son los elementos que deben interactuar para llevarla a cabo.

4.3.1. La Ruta de Datos

La ruta de datos es la sección donde se realizan las operaciones típicas de suma, resta, comparaciones, funciones lógicas, etc. que sirven de base al procesamiento de los datos. Para realizarse estas operaciones deben residir en esta zona del procesador recursos capaces de efectuar dichas operaciones. Estos elementos suelen ser puramente combinacionales, y entre éstos suelen encontrarse una *Unidad Aritmético-lógica* o ALU, *desplazadores*, *multiplicadores* y otros. Sin embargo, tanto el origen de los datos como el destino de los resultados de estas operaciones deben ser elementos de memoria. En el caso de los RISCs, estos elementos de memoria son registros internos del procesador,

con lo que se evita el estar realizando accesos a memoria continuamente, con las penalizaciones de tiempo que conllevaría. A través de estas unidades de almacenamiento de información se garantiza que las instrucciones posteriores de un programa tengan conocimiento del resultado de las operaciones anteriores. Para poner en comunicación los distintos elementos de la ruta de datos se utilizan multiplexores y *buses*.

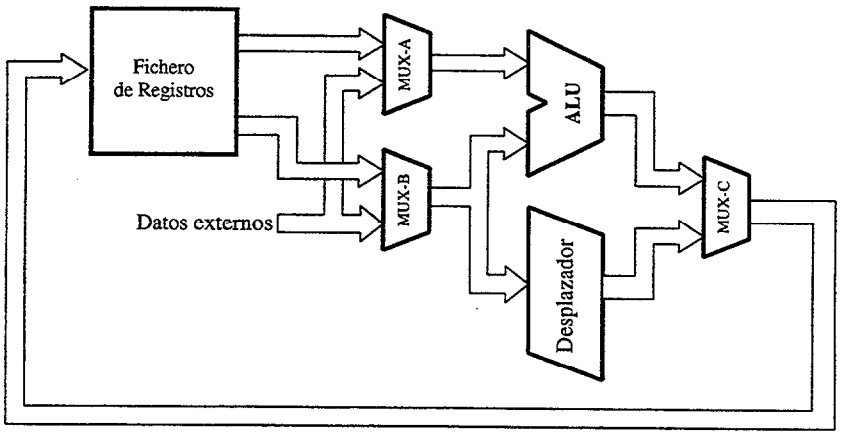


FIGURA 4.4: Una ruta de datos de procesador sin segmentación

La selección de los recursos se realiza tomando como referencia el juego de instrucciones y todas las posibles operaciones que deben desarrollarse en el núcleo del procesador. Su distribución e interconexiones debe determinarse a partir de un estudio de los elementos por los que deben fluir los datos. A partir de este estudio se debe esbozar una primera aproximación de la arquitectura de la ruta de datos y que se debe tomar como referencia. Un ejemplo de una ruta de datos clásica de este estilo, basada en dos buses y tres direcciones de operandos, se muestra en la figura 4.4.

Posteriormente, se deben sopesar las ventajas e inconvenientes entre las situaciones hipotéticas sin segmentación y con el mayor número de etapas de segmentación posible. En estas etapas de segmentación no son los recursos físicamente los que se distribuyen sino que lo son las operaciones a realizar con ellos. Por ejemplo, en el caso de un fichero de registros de propósito general del diseño realizado —que tiene cuatro etapas de segmentación—, la lectura de los datos de este fichero se realiza en la segunda etapa de segmentación, y la escritura de los datos en este fichero se realiza una vez completadas las operaciones, esto es, en la cuarta etapa. Sin embargo, no se debe establecer en ningún caso que es el fichero de registros el que se encuentra en la segunda o en la cuarta etapa de segmentación, sino, por el contrario, su lectura y su escritura.

El número final de etapas de segmentación se determina a partir de una evaluación de las operaciones con los datos, los elementos que intervienen, el equilibrado de estas etapas de segmentación y las operaciones necesarias en situaciones particulares (como es el caso de las transferencias de control). Para esta evaluación puede resultar necesario algún conocimiento sobre la tecnología a utilizar, así como de la memoria y otros bloques. Por lo general, en los RISCs simples tipo SPARC y MIPS, el número de etapas de segmentación suele ser de cuatro o cinco, dato éste que tomaremos como referencia.

Una vez decidido el número de etapas de segmentación se hace visible el paralelismo a nivel de instrucciones (ILP) que se expone en el cauce. Pueden contemplarse entonces

los mecanismos que se deben proporcionar para facilitar la resolución de situaciones de conflicto o riesgo por dependencia de datos y por el flujo de control. En algunas arquitecturas, esto se realiza desde el compilador introduciendo instrucciones de relleno. En otras arquitecturas —como es el caso de SPARC— no se contemplan tales rellenos, por lo que las implementaciones de dicha arquitectura que tengan segmentada la ruta de datos deben contener mecanismos que solucionen estas situaciones. Un mecanismo para esto es el de adelanto de datos (*by-pass* o *forwarding*). Otro mecanismo es, por ejemplo, introducir ciclos adicionales para que las siguientes instrucciones operen con datos válidos. La complejidad de la solución que se deba emplear para resolver estas situaciones de conflicto puede ocasionalmente replantear la segmentación inicialmente realizada.

Decisiones de segmentación

La segmentación es muy importante en los procesadores que se van a emplear en aplicaciones donde se requiere velocidad. Gracias a ella, un conjunto de recursos de procesamiento en cascada se puede transformar en una cadena donde realizar ese procesamiento por pasos, conectados pero separados, de modo que puedan coexistir dos o más pasos distintos de solicitudes de procesamiento distintas. De esta forma, se solapan tareas. La verdadera ventaja reside en que al estarse atendiendo varias solicitudes de forma simultánea (aunque sea ocupándose de fases distintas de cada una de ellas, en cada instante) el tiempo de ejecución global resulta menor.

Supóngase que en la sencilla ruta de datos de la figura 4.4 —que se había tomado como ejemplo— se van a realizar consecutivamente dos operaciones como las siguientes:

- una de suma (a través de la ALU) con un operando externo y otro en el fichero de registros, depositando luego el resultado en el mismo fichero de registros:

  `add %r1, 0xFF, %r2`

- otra de desplazamiento, tomando un elemento del fichero de registros, pasándolo luego por el desplazador y por último se lleva el resultado al fichero de registros:

  `sll %r3, 3, %r1`

En estas circunstancias esta ruta de datos se ocuparía en dos ciclos de la siguiente forma:

- Para la instrucción `add %r1, 0xFF, %r2` se tomaría un dato del fichero de registros, se pasaría por `MUX-A`, simultáneamente se tomaría un dato externo (`0xFF`) a través de `MUX-B` y se sumarían en la ALU. El resultado se haría llegar a través de `MUX-C` hasta el fichero de registros, donde se almacenaría.

- Para la instrucción `sll %r3, 3, %r1`, se accedería a un dato del fichero de registros, se llevaría a través de `MUX-B` para llegar al desplazador y el resultado se enviaría al fichero de registros gracias a `MUX-C`.

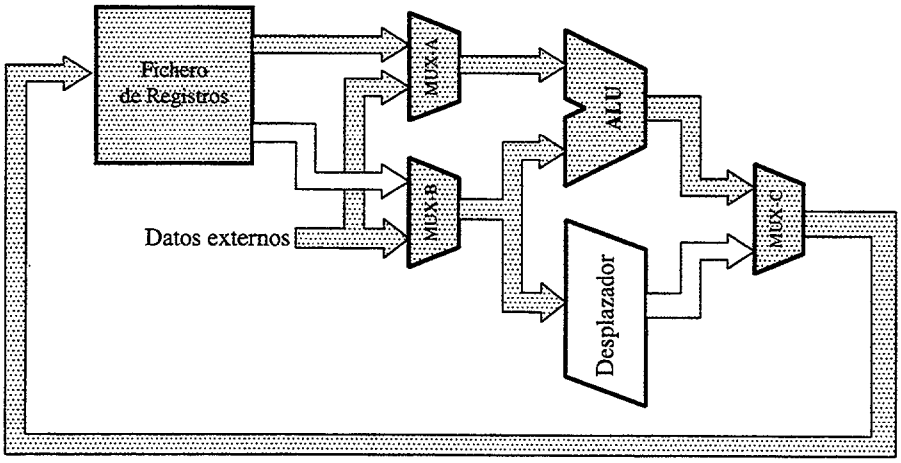


FIGURA 4.5: Elementos activos en la ejecución de `add %r1, 0xFF, %r2` en la ruta de datos sin segmentación de ejemplo

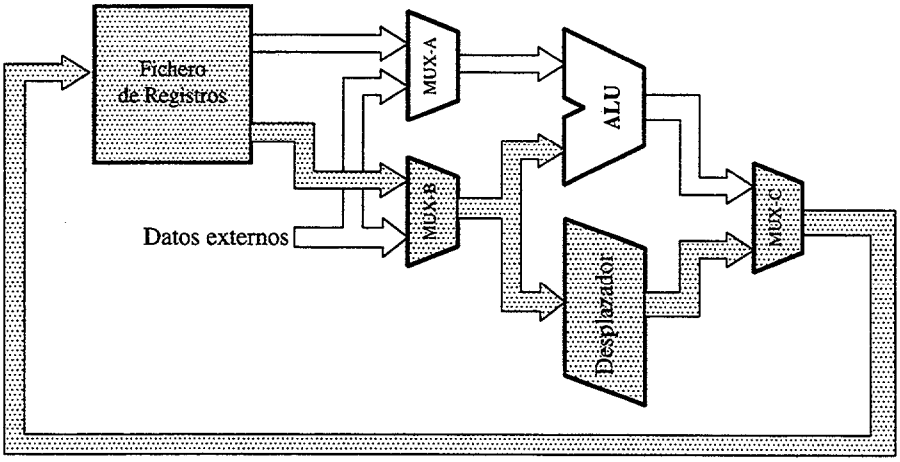


FIGURA 4.6: Elementos activos en la ejecución de `sll %r3, 3, %r1` en la ruta de datos sin segmentación de ejemplo

En estas circunstancias, el tiempo de ciclo, que vendrá limitado por la operación más lenta, será

$$\begin{aligned} T_{\text{ciclo}_{\text{sin segmentación}}} = \text{máx}(&t_{\text{Señales de control}} + t_{\text{Lectura del fichero}} + t_{\texttt{MUX-A}} + t_{\text{ALU}} + \\ &t_{\texttt{MUX-C}} + t_{\text{Escritura en el fichero}}, \\ &t_{\text{Señales de control}} + t_{\text{Lectura del fichero}} + t_{\texttt{MUX-B}} + t_{\text{Desplazador}} + \\ &t_{\texttt{MUX-C}} + t_{\text{Escritura en el fichero}}) + \\ &t_{\text{Margen de seguridad}} \end{aligned} \tag{4.1}$$

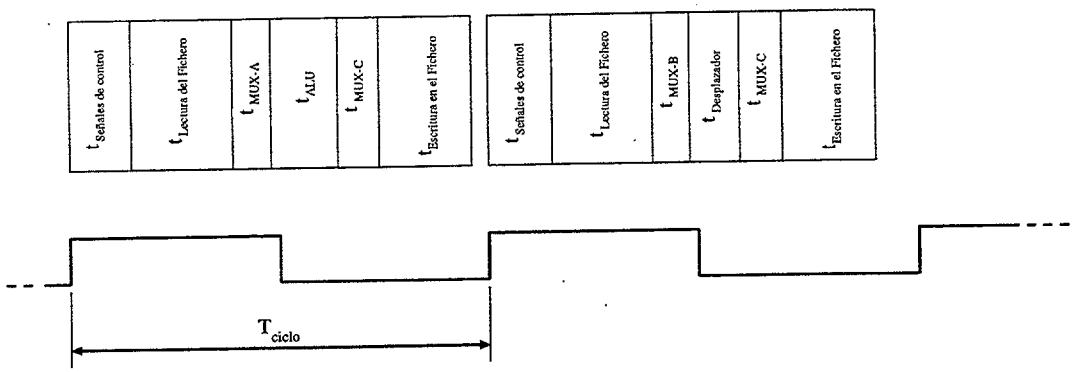


FIGURA 4.7: Temporización en la ruta de datos sin segmentación

En este caso, para la ejecución de las dos instrucciones que se han utilizado como ejemplo el tiempo empleado para ejecutarlas es de $2 \times T_{\text{ciclo}_{\text{sin segmentación}}}$.

Supóngase ahora que se decide segmentar la arquitectura introduciendo unos registros como se indica en la figura 4.8.

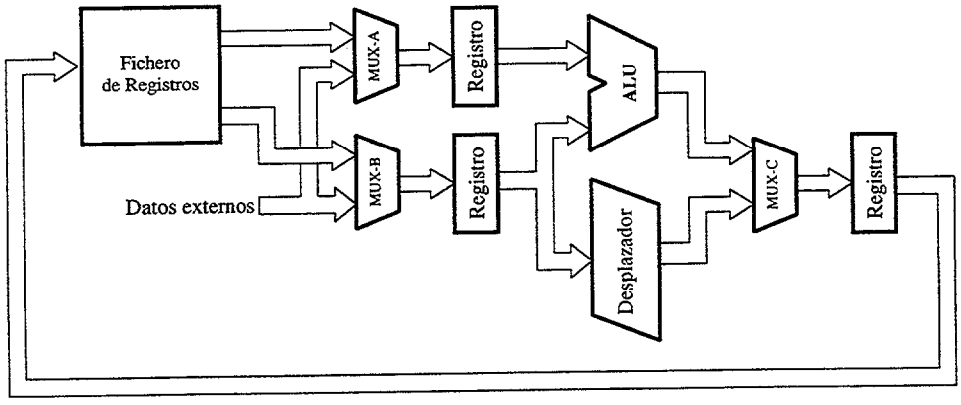


FIGURA 4.8: Una ruta de datos de procesador segmentada

Con la arquitectura segmentada los datos recorren diversas etapas —en este caso tres— en las cuales se realizan los distintos pasos en los que se subdividen las operaciones necesarias para la ejecución de las distintas instrucciones. De este modo, en un determinado instante tendremos distintas subsecciones de la ruta de datos dedicadas a la ejecución de instrucciones distintas. El efecto resultante de esta ejecución "seccionada" (o bien, "por pasos") de las instrucciones es que cada instrucción individualmente

emplea mayor tiempo en su ejecución, pero en la ejecución de la secuencia de varias instrucciones se emplea globalmente menos tiempo. Esto se debe a la ejecución simultánea de pasos distintos de instrucciones distintas. En la figura 4.9 se puede observar la temporización de las dos instrucciones que se ha tomado como ejemplo en esta arquitectura segmentada.

Si t_{Total_k} es el tiempo empleado para ejecutar un determinado programa en una ruta de datos con k niveles de segmentación para la ejecución de una secuencia de N instrucciones, y T_k es el tiempo de ciclo para dicha ruta de datos, entonces para una ruta de datos sin segmentar se obtiene

$$t_{\text{Total}_0} = N \times T_0$$

y en el caso de la ruta de datos con un nivel de segmentación

$$t_{\text{Total}_1} = (N+1)T_1$$

En general se tiene

$$t_{\text{Total}_k} = (N+k)T_k$$

donde T_k es del orden de $\frac{T_0}{k+1}$ en un reparto equilibrado de tareas en las etapas del cauce.

Esta afirmación sólo es válida cuando las instrucciones que se consideran emplean únicamente un ciclo en la ruta sin segmentar, sin necesidad de ciclos adicionales para completar la ejecución.

Por otro lado, téngase en cuenta que

$$T_0 > T_1 > T_2 > \cdots > T_{k-1} > T_k > T_{k+1}$$

ya que es en el procesador sin segmentar donde los datos necesitan viajar a través de un mayor número de elementos hasta que comienza la ejecución de una instrucción nueva. Al introducir un nivel de segmentación adicional normalmente se persigue que el tiempo empleado en la ejecución del paso más lento que puede coexistir sea más rápido que en la situación original. De no ser así, la inclusión de este nivel de segmentación adicional resulta no deseable.

Con estas premisas, el beneficio de introducir una etapa de segmentación adicional se producirá cuando se cumpla que

$$\begin{aligned} t_{\text{Total}_k} &> t_{\text{Total}_{k+1}} \\ (N+k)T_k &> (N+k+1)T_{k+1} \\ T_{k+1} &< \frac{N+k}{N+k+1}T_k \end{aligned}$$

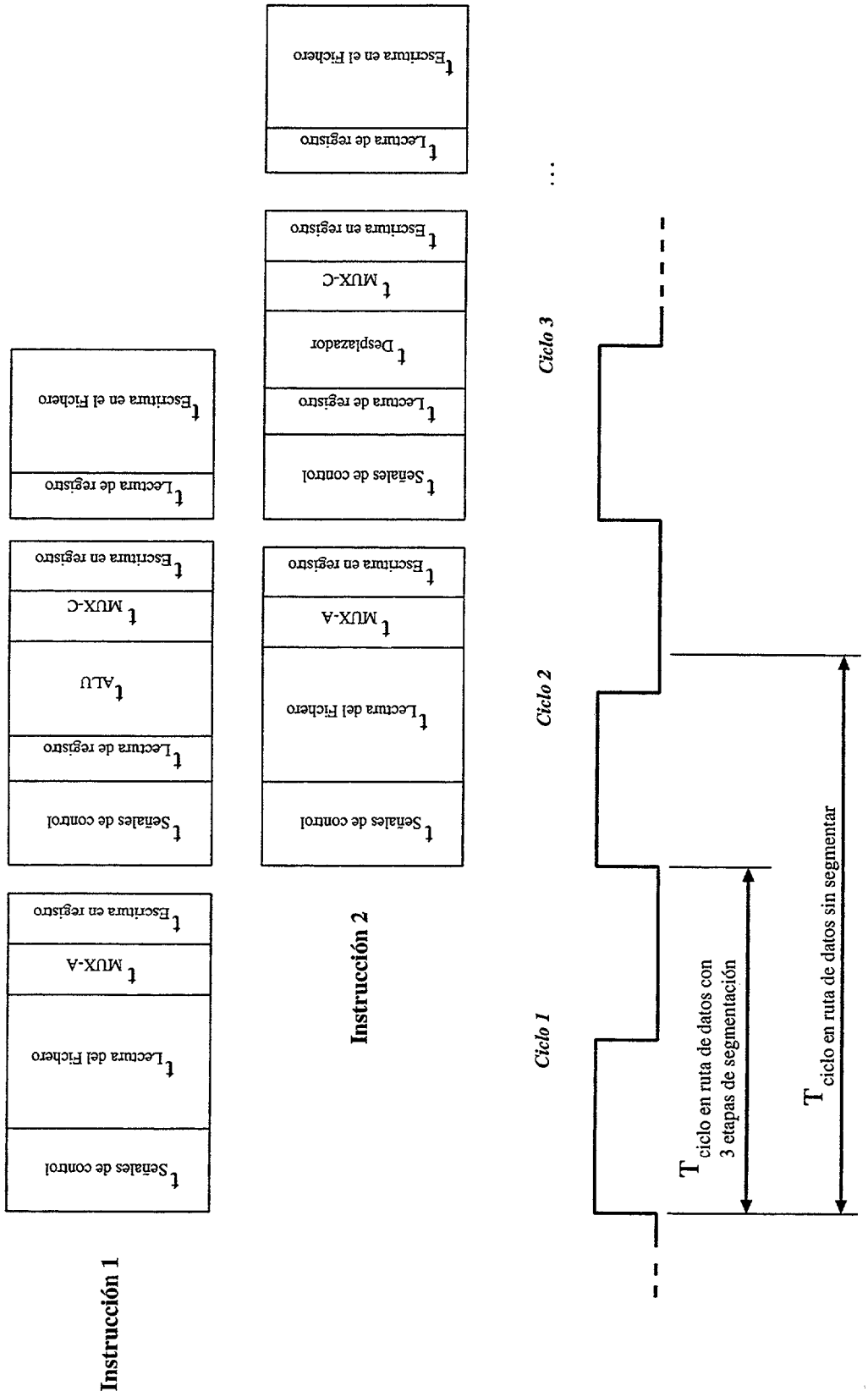


FIGURA 4.9: Temporización en la ruta de datos segmentada

Sin embargo, suelen encontrarse situaciones en las que existen instrucciones que precisan ciclos adicionales. En estos casos, el número de ciclos a emplear en una ruta con k niveles de segmentación será mayor que $N + k$. Además, a veces también ocurre que, debido a dependencias en los datos que se encuentran en la ruta, en la transición desde ciertas instrucciones a las siguientes la introducción de un nivel de segmentación pudiera llevar a añadir algunos ciclos adicionales para la ejecución. Entonces podemos concluir que, de forma general, se tiene un beneficio real en la introducción de un nivel de segmentación adicional sólo cuando

$$T_{k+1} < \frac{N_k}{N_{k+1}} T_k \tag{4.2}$$

siendo N_k el número de ciclos necesarios para la ejecución de la secuencia de instrucciones en un procesador con k etapas de segmentación. En el caso de que ninguna instrucción precise ciclos adicionales será $N_k = N + k$, y en general se tendrá $N_k \geq N + k$. Al introducir un nivel de segmentación adicional, normalmente se tiene que $N_{k+1} = N_k + 1$, excepto en los casos con ciclos adicionales. Por esto, de modo general se tiene de nuevo que $N_{k+1} \geq N_k + 1$. El coste adicional de al menos $k + 1$ ciclos para cargar y gestionar sin conflictos el cauce de instrucciones *sólo* es despreciable si la secuencia (N) o bloque básico es grande y si la frecuencia de ciclos adicionales es pequeña.

Por estas razones la elección de los lugares donde introducir los elementos de segmentación y su número no es arbitraria. De hecho la introducción de un nivel de segmentación adicional incluso puede no conllevar una reducción del tiempo de ciclo, por no estar afectando a la ruta crítica, y en algunos casos podría incluso aumentarlo. Es más, no siempre se podrán elegir tantos niveles de segmentación como se desee, ya que existe una limitación impuesta por el diseño y número de recursos que normalmente intervienen en la ejecución de cada una de las instrucciones.

Por otro lado, es probable que cada pocas instrucciones se ejecute alguna bifurcación o salto que haga perder eficacia a un solapamiento elevado de instrucciones. Otro tanto ocurre, aunque más esporádicamente, con las excepciones e interrupciones.

Esto ocasiona que la elección del número de niveles de segmentación sea motivo de un cuidadoso estudio basado en probar la arquitectura con varios programas tipo que representen la carga de trabajo que se va a ejecutar, incluyendo aplicaciones, sistema operativo y núcleo, y elegir aquel valor que minimice los tiempos de ejecución.

En la figura 4.10 se puede observar una comparación entre las dos situaciones en lo concerniente a la ejecución de 5 instrucciones que no precisan ciclos adicionales, tanto en una ruta sin segmentar como en una ruta con cuatro niveles de segmentación.

Como se puede observar, $T_0 > T_3$, e incluso, para la ejecución de una única instrucción, en la ruta de datos sin segmentar se emplea T_0 y en la ruta de datos segmentada se emplea $4 \times T_3$, siendo el tiempo empleado para ejecutar la *Instrucción 1* (su latencia) mayor en la ruta de datos segmentada que en la no segmentada:

$$T_0 < 4 \times T_3$$

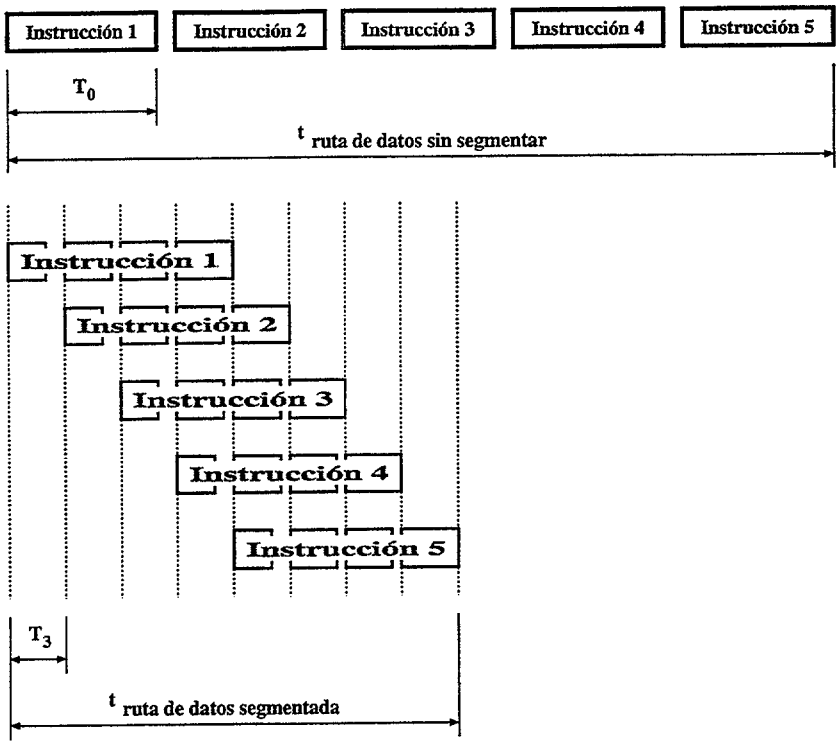


FIGURA 4.10: Comparación en los tiempos de ejecución de cinco instrucciones

Pero, como puede comprobarse igualmente en la figura, existe una ganancia apreciable en el tiempo globalmente empleado al ejecutar varias instrucciones.

Puede apreciarse que además en la ruta segmentada también se completa una instrucción por ciclo (*throughput*). La fig. 4.11 presenta la ruta de datos finalmente diseñada.

Tipo de Registros de segmentación

Los registros de segmentación son probablemente uno de los ejemplos más significativos del cuidado especial que necesitan ciertos elementos de la arquitectura. Su diseño tiene un impacto considerable en las prestaciones que se pueden conseguir.

En el caso del diseño de la IU de SPARC v8 se han elegido para estos registros de segmentación *flip-flops* tipo E. Esta opción es la que se ha estimado recomendable cuando se desea realizar un diseño seguro de un núcleo RISC.

Con una distribución cuidada de estos registros especiales dentro del modelo gráfico RTL de la Ruta de Datos, el diseñador puede tener, en todo momento, una visión rápida de cómo se ha organizado la segmentación de la arquitectura. En el caso del modelado de la IU de SPARC, estos registros se han dispuesto en determinadas posiciones verticales, de modo que se pueda determinar rápidamente si una determinada operación sobre una unidad se desarrolla en la primera, segunda o n-ésima etapa de ejecución de cada instrucción (véase la fig. 4.12).

Para estos registros es necesario permitir o impedir su carga dependiendo de que la máquina esté en operación normal o detenida. Se debe contemplar la posible modificación de la implementación de estos registros para cumplir especificidades concretas que

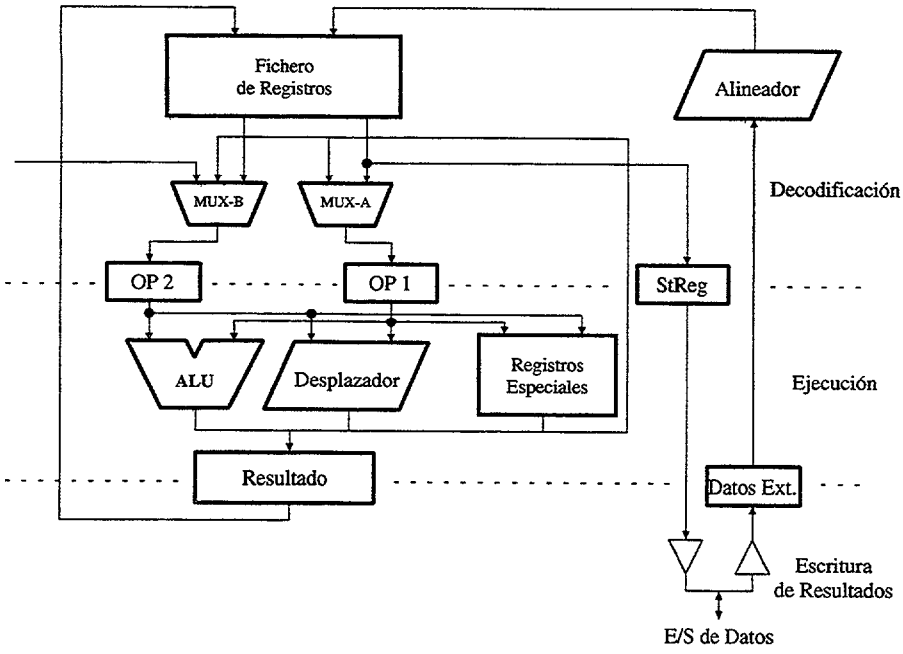


FIGURA 4.11: Arquitectura de la Ruta de Datos de la IU

podrían requerirse a estos elementos. Desde el punto de vista que se está contemplando en este diseño, la señal de reloj debe alcanzar a estos elementos de almacenamiento en estado puro. Es decir, no debe existir circuitería en medio de las líneas de distribución de esta señal tan crítica que es el reloj, ya que podría producir *skews* (desfase de reloj) indeseados. Utilizando los *flip-flops* tipo E, se puede controlar de modo aceptable la distribución de esta señal de reloj empleando las herramientas de CAD disponibles, de modo que se evita el mal funcionamiento. Esta situación es todavía más crítica en diseños de baja potencia [RP96] al aparecer nuevos condicionantes. Los *flip-flops* tipo E cumplen bien con estas necesidades.

Por otro lado, con los *flip-flops* tipo E se evita establecer restricciones especiales para el ciclo de trabajo de la señal de reloj del circuito, a la vez que se mantiene un diseño claro y fácilmente gestionable. Hay que tener en cuenta que cuanto más restricciones se impongan para el mecanismo de reloj, mayores problemas pueden surgir cuando se utilicen estos núcleos en entornos hostiles (para los que se diseñan algunos sistemas empotrados).

Si aún así en algunas circunstancias se desea elegir un diseño distinto para la segmentación (por ejemplo, utilizando múltiples fases de reloj), la metodología que se ha establecido es igualmente válida mientras se mantenga la coherencia entre el modo en el que fluyen los datos y el modo en el que se transportan las señales de control.

Después de estudiarse diversas alternativas, y compararse con algunas empleadas en otra implementaciones, para la unidad de enteros de la SPARC versión 8 se ha decidido que la segmentación más adecuada para esta implementación es de 4 etapas. En estas etapas de segmentación, y por lo que concierne a la ruta de datos, se realizan, entre otras, las siguientes operaciones:

BI, Búsqueda de Instrucción: En esta etapa se efectúa la búsqueda de instrucciones de la

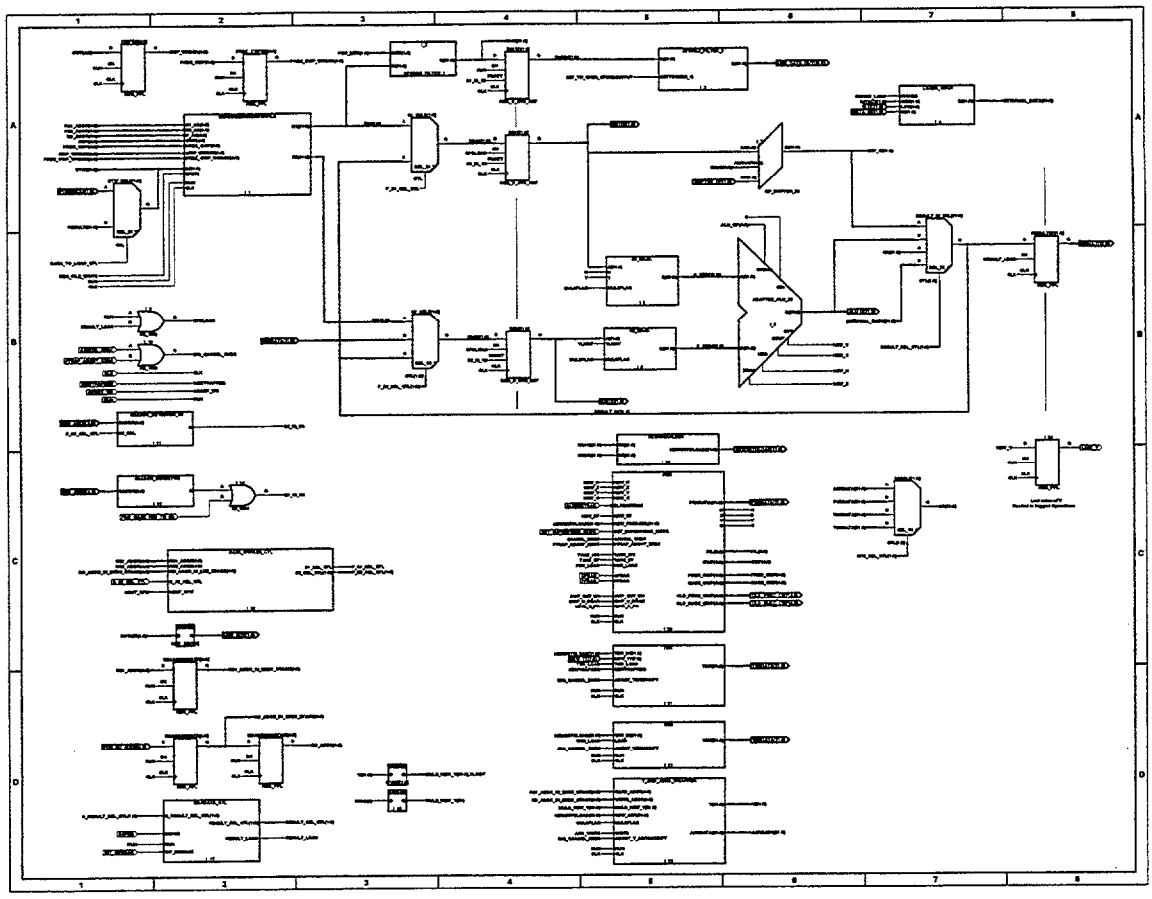


FIGURA 4.12: La arquitectura de la ruta de datos de la IU de SPARC

Memoria de Programa. Esta etapa no afecta a elementos de la Ruta de Datos.

DI, Decodificación de la Instrucción: En esta etapa se realiza la decodificación de la instrucción y la búsqueda de los operandos fuente del Fichero de Registros.

EI, Ejecución de la Instrucción: En esta etapa se realizan las operaciones aritméticas o lógicas necesarias que sirven de cálculo en la ejecución de las instrucciones. Además se escriben los registros especiales del PSR, TBR, WIM y ASRs.

ER, Escritura de Registros: En esta etapa se escriben los resultados de la etapa anterior en el Fichero de Registros.

Decisiones sobre recursos a medida

En algunas situaciones, resulta más ventajosa la utilización de generadores de módulos para la realización de determinados elementos de relativa complejidad, frente a la síntesis en células estándares que realizan las actuales herramientas. Las razones para tomar esta alternativa suelen basarse en criterios como el área ocupada, menores retrasos y homogeneidad en el comportamiento de los subcomponentes. Por lo general, la alternativa de emplear estos módulos se ve favorecida por las características óptimas de estas estructuras al estar meditadas concienzudamente en su diseño. Este es el caso de elementos como los ficheros de registros, las ALUs o los desplazadores.

De este modo se puede hacer un énfasis especial en determinadas rutas críticas de modo que se cumpla con las necesidades de tiempo, así como atajar el problema de cumplir con las imposiciones de tiempo y área desde otro frente distinto del de la síntesis automática.

Aunque se pierda cierta generalidad en los modelos, las ganancias debidas a este tipo de alternativas casi siempre resultan valiosas. Sin embargo, estas alternativas no deberían utilizarse frecuentemente ya que conduciría a modelos muy dependientes de las herramientas utilizadas, y, por lo tanto, no deseables.

Cuando se desea emplear este tipo de elementos en el diseño, los modelos realizados en HDL presentan elementos propios de las herramientas utilizadas, por lo que los modelos pierden generalidad en beneficio de mejores prestaciones. Es más, no resulta extraño tener que introducir elementos especiales con los que adaptar adecuadamente estos módulos con el resto de los elementos que forman parte del diseño.

Circunscribiéndose al diseño realizado sobre la IU de SPARC, merece una especial atención los ficheros de registros asíncronos. Estos recursos son extremadamente exigentes con la temporización de las órdenes de lectura y escritura. Por ello, para emplearlos con garantías dentro del núcleo se precisan determinados elementos de interface con los que crear estas señales con los requisitos necesarios. Eligiendo y diseñando cuidadosamente estos elementos adicionales[3] la visión de estos ficheros de registros asíncronos se puede cambiar a la de elementos completamente síncronos. Con esto el beneficio que se obtiene es doble, ya que la gestión de las operaciones de lectura y escritura pueden resultar más cómodas en lo que respecta al elemento de generación de señales de control del núcleo procesador, y también puede facilitar la tarea a un posible usuario de los modelos que desee introducir cambios en el comportamiento del núcleo.

Mecanismo de direccionamiento de registros enventanados

Éste es uno de los más claros ejemplos de cómo influye una estrategia de diseño y lo poco que puede intervenir una buena herramienta para paliar la situación ante el modelo realizado.

Como se ha indicado anteriormente, la búsqueda de los operandos fuente que se precisan para ejecutar una instrucción se realiza en la etapa de Decodificación de Instrucción. Ya que es al comienzo de esta etapa cuando el procesador *conoce* la instrucción a ejecutar, se dispone *únicamente* del tiempo de un ciclo para realizar esta búsqueda de operandos. Por lo tanto, un direccionamiento rápido de los registros fuente que intervienen es esencial.

Tal como se mostró en la fig. 3.1 de la pág. 35, la arquitectura SPARC permite que en cada instante un proceso sólo tenga accesible una *ventana* de 24 registros organizados del siguiente modo:

- 8 registros de entrada (del r_{24} al r_{31}), los cuales son accesibles desde una ventana

[3] En este caso particular estos elementos fueron varios multiplexores, *flip-flops*, algunas puertas y un monoestable con estructura Muller-C.

anterior,

- 8 registros locales (del r_{16} al r_{23}), exclusivos para la ventana vigente,
- 8 registros de salida (del r_8 al r_{15}), accesibles para la ventana siguiente, y
- 8 registros globales (del r_0 al r_7), visibles desde todas las ventanas.

La ventana que se encuentra activa en cada momento vendrá determinada por el contenido del Puntero de Ventana Activa, CWP, que a su vez forma parte de la Registro de Estado del Procesador, PSR. Cuando se quiera acceder a un registro determinado de una ventana se tendrá que conocer, por lo tanto, el indicador del registro r_i y el valor del CWP.

Sin embargo, todos estos registros deben ir ubicados en ficheros de registros donde el acceso a estos se realiza de forma completamente lineal. Por esto, la indicación de un registro r_i determinado dentro de una ventana específica debe traducirse en una o varias direcciones adecuadas para este o estos ficheros de registros. Para conseguir el efecto deseado son diversas las alternativas que se pueden plantear.

Si se escogiese un único fichero de registros para almacenar todos los r_i, la dirección del registro r_i de este fichero debería obtenerse de la forma siguiente:

si $(i < 8)$ **entonces**
  $dir \leftarrow i$
si no
  si $(i \geq 24)$ **y** $(\text{CWP} = \text{NWindows} - 1)$ **entonces**
    $dir \leftarrow (i \bmod 8) + 8$
  si no
    $dir \leftarrow i + 16 \times \text{NWindows}$
  fin si
fin si

Sin embargo, esta alternativa es costosa en recursos, y puede ocasionar un aumento considerable del tiempo de ciclo, al poderse encontrar i interviniendo en un cálculo aritmético (una suma). Por esta razón se ha ideado una organización alternativa que permita un cálculo más rápido. En esta ocasión se ha determinado que una organización que responde bien a este criterio se basa en dos ficheros de registros diferenciados, donde en uno se ubican los registros globales del r_0 al r_7 y en otro los registros no globales del r_8 al r_{31} de las distintas ventanas. En esta ocasión las direcciones de los dos ficheros de registros se calcularían de la siguiente forma:

$dir_{\text{globales}} \leftarrow i \bmod 8$
si $(i < 16)$ **entonces**
  $dir_{\text{no globales}} \leftarrow 16 \times \text{prec(CWP)} + i \bmod 8$
si no
  $dir_{\text{no globales}} \leftarrow 16 \times \text{CWP} + i \bmod 8$

fin si
si $(i < 8)$ **entonces**
  Coger global
si no
  Coger no global
fin si

Como se podrá observar,

$$\begin{aligned} i \bmod 8 &\equiv i<2:0> \\ i<16 &\equiv i<5:4>=0 \\ i<8 &\equiv i<5:3>=0 \\ 16 \times \mathrm{prec(CWP)} + i \bmod 8 &\equiv \mathrm{prec(CWP)} \square i<2:0> \\ 16 \times \mathrm{CWP} + i \bmod 8 &\equiv \mathrm{CWP} \square i<2:0> \end{aligned}$$

por lo que en esta alternativa la única penalización podría encontrarse es con *prec(CWP)*. Sin embargo, es sencilla de evitar si al modificar en algún instante CWP se calcula y almacena al mismo tiempo el valor de *prec(CWP)* en un registro diferenciado.

4.3.2. La Unidad de Control

La Unidad de Control es, probablemente, la sección más delicada de implementar, ya que es más abundante en lógica no regular y para llevarla a cabo se deben tener en cuenta numerosas consideraciones. Al poseer tanta lógica no regular, uno de los retos que se presentan en su modelado es el establecer una organización adecuada que permita una fácil depuración y una gestión de modificaciones flexible.

Los procesadores suelen tener su Unidad de Control dividida en dos subsecciones. En la primera se recogen las instrucciones en el procesador, y una vez recogidas se examina en ellas diferentes campos para averiguar las operaciones que se necesitan realizar para ejecutar dicha instrucción y de dónde se deben recoger los datos con los que operar. Conjuntamente se encuentra la Unidad de Secuenciamiento. Esta subsección suele tener en los RISCs ciertos elementos en los que algunas interrelaciones responden a estructuras bastante definidas. Su realización se puede acometer de un modo similar al de la Ruta de Datos, ya que aquí se puede concebir el flujo de los datos desde un punto de vista RTL como en la sección de procesamiento de datos (véase la fig. 4.13).

En la segunda subsección se generan las verdaderas señales de control de todo el procesador y las necesarias para cualquier elemento externo con el que se desee que el procesador se comunique.

Las señales de control son de los elementos más críticos que existen en el diseño. El método que se elija para su generación puede influir en las prestaciones del núcleo en su conjunto.

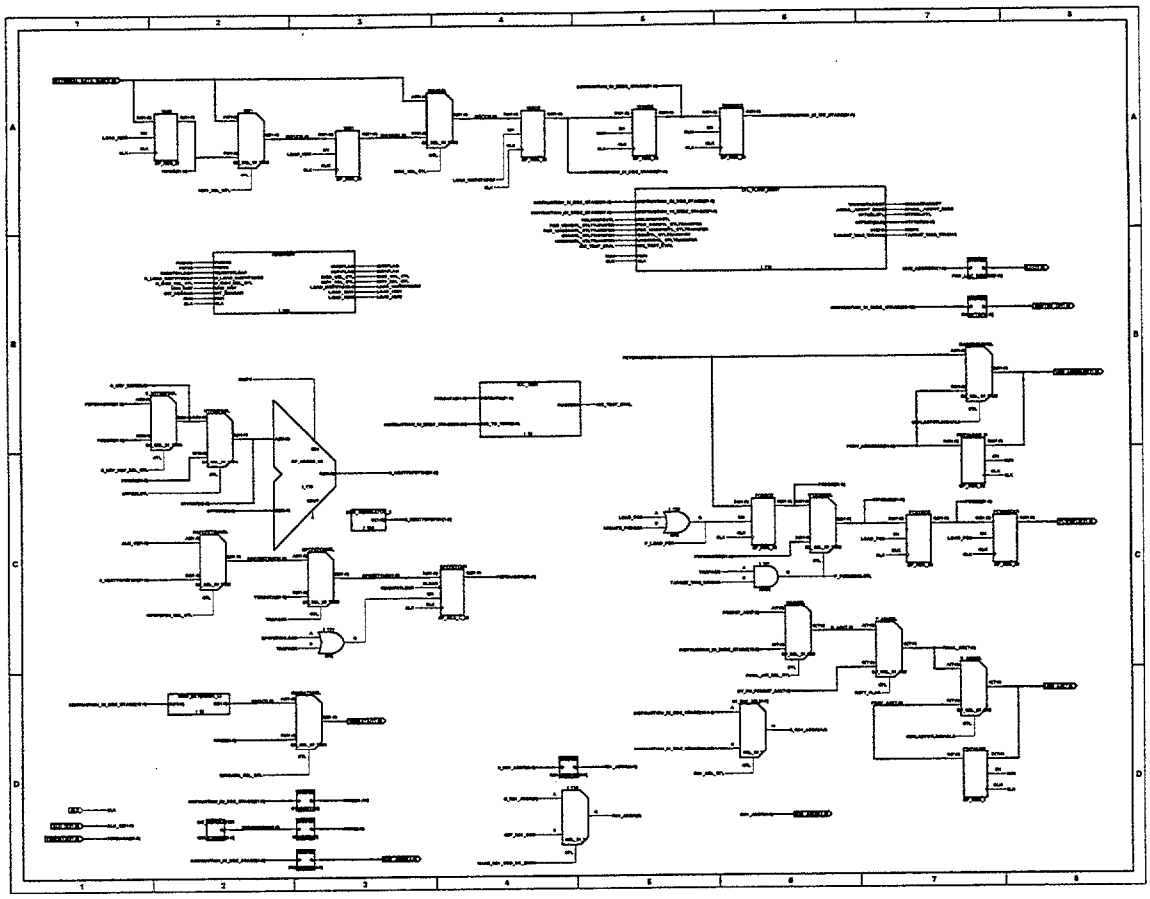


FIGURA 4.13: La subunidad de secuenciamiento y de instrucciones de la IU de SPARC

Generación de las señales de control centralizadas

Para la generación de estas señales de órdenes la unidad de control deberá llevar a cabo diversas tareas. Entre éstas encontramos las siguientes:

- Evaluar las señales y elementos de memoria, tanto internos del procesador como externos, y
- decidir en base a estas evaluaciones.

Con estas dos tareas aparentemente sencillas la Unidad de Control deberá ser capaz de *guiar* a toda la máquina. Lo realmente complejo en este tipo de situaciones es, desde el punto de vista del diseñador, cuantificar el número de elementos que se deben evaluar en cada momento (instrucción a ejecutar, palabra de estado, estado interno del procesador, niveles de interrupción, señales de inicialización, parada momentánea, etc.) y establecer cómo y cuándo deben afectar al funcionamiento de la máquina.

Para realizar estas tareas es necesario que la Unidad de Control siga una secuencia de subtareas o *pasos*. Esta secuencia se consigue gracias a la operación de una o varias máquinas de estados finitos (FSM). En el caso de los procesadores RISC, y con vistas a interactuar con una ruta de datos segmentada, dos son las alternativas que se nos plantea en cuanto al número de FSMs a elegir:

Una FSM centralizada, donde se tomarán todas las decisiones para la dirección de la máquina y generación de señales (fig. 4.15).

Varias FSM distribuidas, de tal modo que las decisiones se realizarán en distintas FSMs, destinando cada una de ellas a controlar las distintas etapas en la que se organiza la ruta de datos. Además, deberá existir coherencia en esta distribución, evitando que dos FSMs distintas colisionen en el intento de control de determinados recursos; y debería existir al menos una que coordine el conjunto (fig. 4.14).

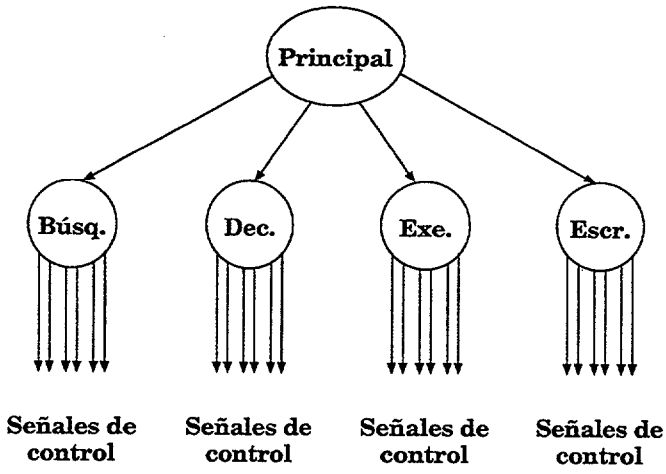


FIGURA 4.14: Máquinas de estado para control distribuido

Existen diferencias fundamentales entre las dos alternativas. En la primera, la obtención de las señales de control se hace desde una sola máquina de estados que debe contemplar todas las situaciones posibles en las que se pueda encontrar el procesador. Por esto el tamaño de esta FSM es la mayor posible, con lo que la obtención de las señales de control es la más lenta. En la distribuida, sin embargo, las FSMs suelen ser más pequeñas que la FSM centralizada, por lo que la obtención de las señales de control es relativamente más rápida.

Sin embargo, este último aspecto no debe llevar a engaño, ya que lo que realmente interesa es no cuánto tardan las señales de control en generarse sino cuánto tardan en llegar a los elementos del procesador cuando se las necesita, o sea, desde que los datos están disponibles en cada uno de los pasos de ejecución. Como se podrá observar en la fig. 4.15, en este caso las señales de control que gobiernan los elementos que interactúan en las etapas excepto la primera proceden de registros, por lo que el retraso con respecto al comienzo del ciclo es menor que el obtenido de las FSMs distribuidas. En estas circunstancias las señales de control más críticas corresponden a las que gobiernan las operaciones que se desarrollan en el primer paso de búsqueda de instrucción.

Por lo general, y siempre que sea posible, en un procesador RISC resulta más óptima la utilización de una FSM centralizada. Además, desde el punto de vista del modelado, al eligir esta opción el diseño y la depuración del *hardware* y sus modelos resultan más sencillos. Por otro lado, si el procesador incluye otras unidades como una FPU u otro coprocesador, la gestión y sincronización de todas las unidades se ve facilitada. Una

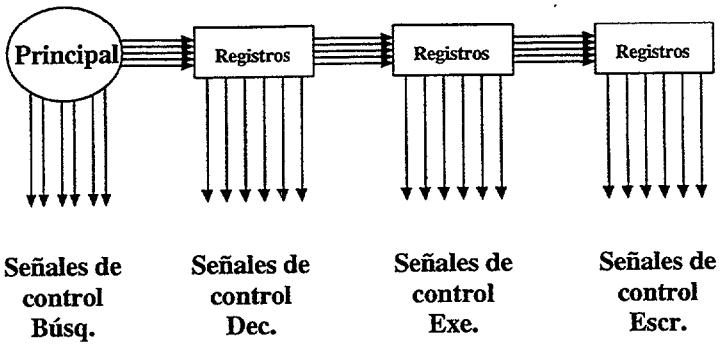


FIGURA 4.15: Máquina de estado y elementos adicionales necesarios para control centralizado

extensión de esta filosofía conduce a la gestión de un procesador superescalar con ayuda de un marcador central (*scoreboard*).

Las señales de control necesarias en una determinada etapa de segmentación podrían generarse o bien dentro del mismo ciclo en el que la circuitería de dicha etapa las necesita o bien se podrían haber obtenido en etapas anteriores. En el primer caso, los valores de estas señales deben empezar a calcularse desde el comienzo del ciclo correspondiente y podrá comenzar a operar una vez que se estabilice la circuitería que gobierna. En el segundo caso, los valores de estas señales que se recogerán en cierta etapa prodecerán del contenido de registros, por lo que desde el comienzo del ciclo permanecerán estables. De cara a obtener las mejores prestaciones, la segunda alternativa siempre es más favorable. De este modo la lógica de decodificación de las instrucciones se concentrará en las primeras etapas del flujo de ejecución de las instrucciones, evitando el aumento de los tiempos empleados en etapas posteriores debido a retrasos asociados a esta decodificación.

Por ejemplo, en una arquitectura segmentada donde se desarrollan subtareas diferentes que conciernen a distintas instrucciones, la longitud del ciclo de reloj de la máquina viene parcialmente determinada por los retardos producidos tanto en los recursos de una determinada etapa que operan con los datos como en la obtención de las señales de control para estos recursos. Mientras se consiga generar las señales de control necesarias para estas etapas críticas en etapas anteriores y propagarlas con registros, se evitará que el tiempo de ciclo aumente e incluso se puede conseguir que este tiempo sea independiente de la complejidad del juego de instrucciones —siempre que el tiempo de ciclo no se vea condicionado por la decodificación, como es recomendable— (véase la fig. 4.16).

Sin embargo, existen situaciones en las que no existe posibilidad alguna de evitar que se realicen ciertas evaluaciones a partir de las que generar señales de control específicas dentro del mismo ciclo donde se utilizarán. En cualquier caso, siempre se deberá minimizar el tiempo total para hacer estas señales disponibles lo antes posible dentro de cada etapa de procesado.

Como consecuencia de estas consideraciones de diseño, la estrategia que se recomienda seguir es la organización con una FSM única y centralizada de la fig. 4.15. De este modo el retraso para las señales de control es, en la mayoría de los casos, el debido a los registros de segmentación. La única excepción a esto son los casos particulares en los que las señales de control no se pueden generar en etapas previas, como es el caso de

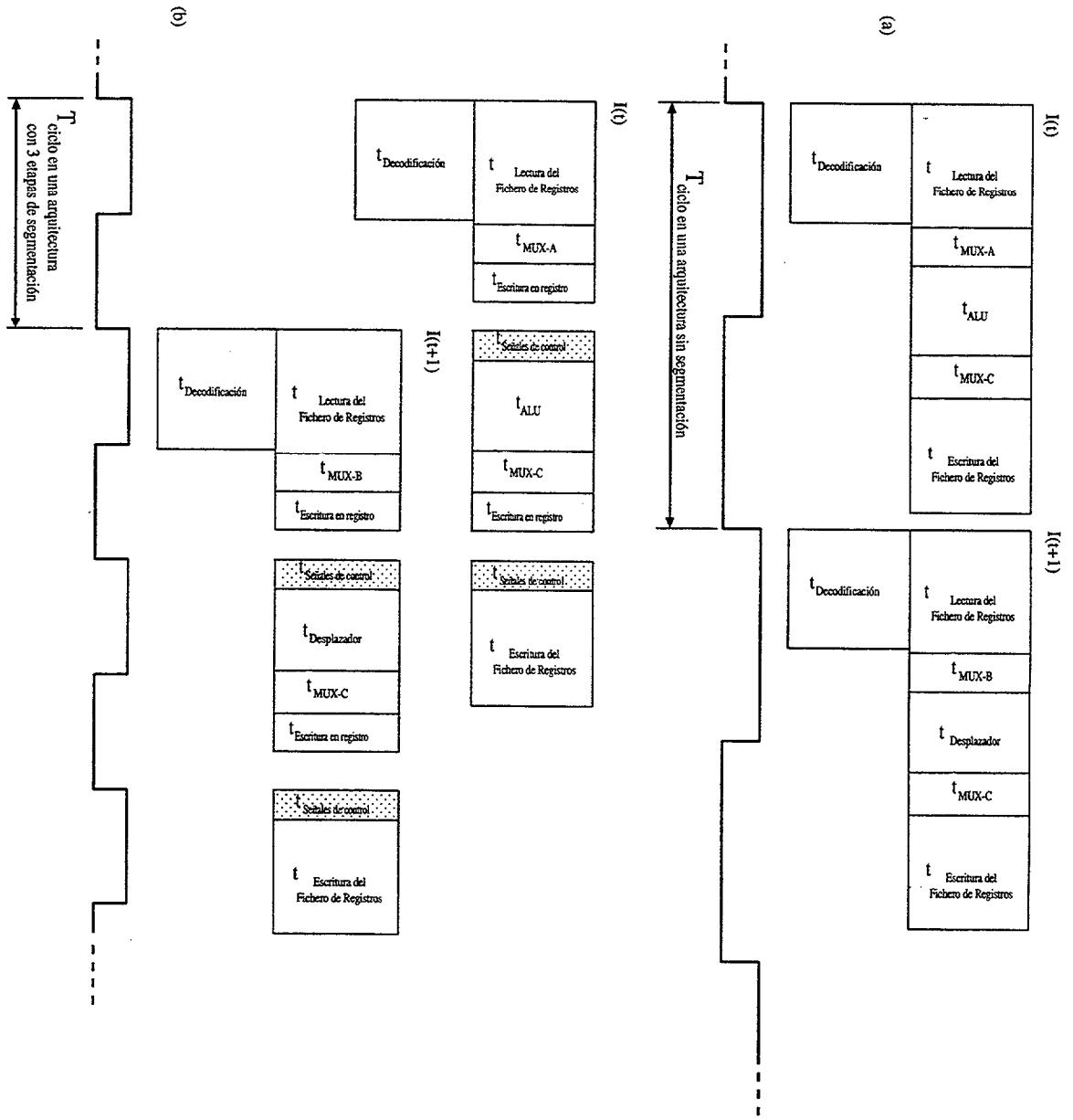


FIGURA 4.16: Influencia de las señales de control en la determinación del tiempo de ciclo en una ruta de datos segmentada (b) frente a una no segmentada (a)

algunos elementos que forzosamente deben operar en el mismo ciclo en que se realiza la decodificación.

Esta decisión tiene una única incidencia significativa sobre el diseño de la Ruta de Datos. Esta es que para mantener la coherencia en el diseño global se deben introducir registros especiales que transporten concurrentemente las señales de control. De este modo, las señales de control pueden alcanzar los elementos a los que afectan para que operen en el momento preciso. Además, también podrán existir elementos especiales con los que bloquear la propagación de estas señales de control, como, por ejemplo, para abortar la ejecución completa de una instrucción.

Unidad de Secuenciamiento

Para la gestión de las instrucciones se dispone de una sección especial de la Unidad de Control que se ha denominado *Unidad de Secuenciamiento*. La finalidad de esta sección es la de llevar control del orden en que se recogen y se van ejecutando las distintas instrucciones de los programas.

Por regla general, las instrucciones se van recogiendo en secuencia de la memoria de programa y se van ejecutando una tras otra, tal como se expone en la fig. 4.17.

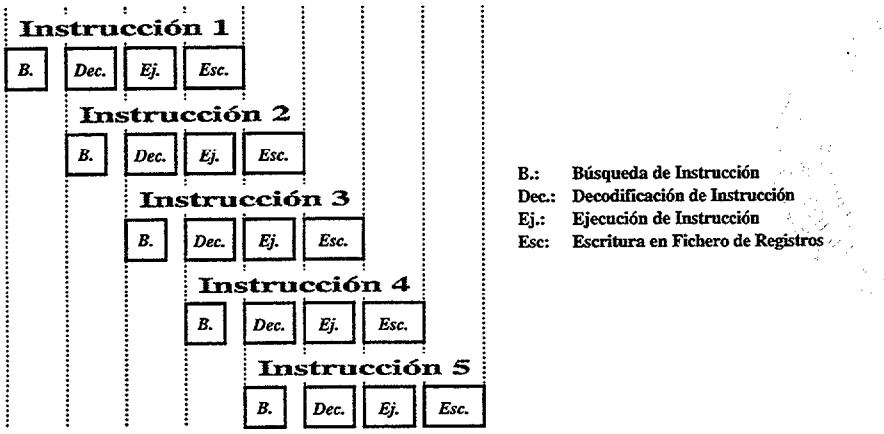


FIGURA 4.17: Ejecución normal de instrucciones

Sin embargo, existen situaciones especiales en las que esta cadencia debe romperse intencionadamente. Esto se produce en diversas situaciones como las siguientes:

- Al ejecutar instrucciones que necesitan ciclos adicionales[4]
- Al ejecutar instrucciones de transferencia de control

[4]Quizás sería más propio hablar en este caso de 'pasos adicionales'. En el caso de otros autores se habla de *instrucciones auxiliares*. Por regla general, estas instrucciones tienen correspondencia con instrucciones que se realizarían de forma análoga en varios ciclos en una arquitectura no segmentada, a diferencia del resto, que sólo precisarían uno. Por esta razón a veces se habla de instrucciones *monociclo* —cuando en realidad ocupan tantos ciclos como etapas de segmentación— e instrucciones *multiciclo* —que emplean más ciclos—.

- En algunas situaciones de excepción como en la inicialización y en la gestión de interrupciones

Todas estas situaciones requieren una respuesta especial por parte de la unidad de secuenciamiento que debe ser diferente de la situación normal. En cualquier caso, y sea cual sea el comportamiento ante estas situaciones, una vez rebasada cualquiera de estas situaciones especiales la máquina debe seguir recogiendo y ejecutando las instrucciones en secuencia como en los casos normales.

Si observamos el secuenciamiento que se expone en la fig. 4.18, donde aparece una instrucción con ciclos adicionales, lo primero que se puede observar es que el orden en que se realiza la búsqueda de las instrucciones no siempre es el mismo que aquel en que se decodifican y se lanzan a ejecución. Una vez decodificada una instrucción, los pasos siguientes que siempre corresponden son el de su ejecución y el de escritura de resultados en el fichero de registros. De esto se desprende que, en este caso, el control de la búsqueda de instrucciones debe realizarse de modo diferente al control de la decodificación de las instrucciones y de los ciclos adicionales.

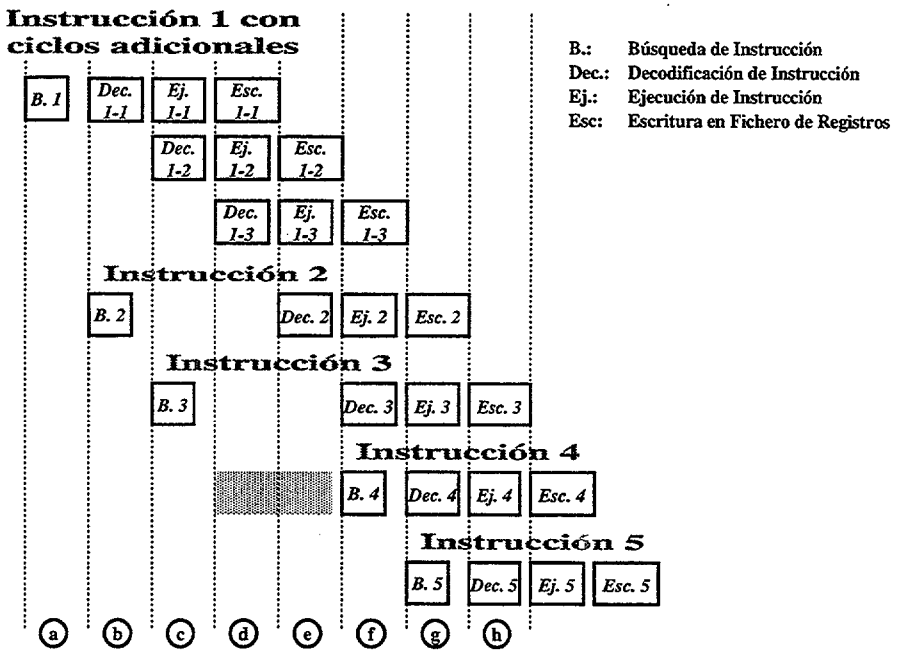


FIGURA 4.18: Ejecución de una instrucción multiciclo

Siguiendo el ejemplo de la fig. 4.18, se puede ver que al decodificar la instrucción 1 en ⓑ se concluye que se necesitan ciclos adicionales. Entonces, la instrucción 2 que se está buscando en este intervalo deberá almacenarse en un sitio temporal para decodificarla posteriormente en ⓔ, una vez que se hayan lanzado todos los pasos necesarior para ejecutar la instrucción multiciclo 1. De igual forma, al recoger la instrucción 3 en ⓒ también deberá guardarse en un sitio temporal pero se deberá recuperar después de que se recupere la instrucción 2. En el diseño que se ha realizado esto se ha efectuado con una cola FIFO donde guardar estas instrucciones. Para no complicar la arquitectura en este caso se ha decidido que la cola de instrucciones debía ser de profundidad 2. Por esta razón en el intervalo ⓓ no se realiza búsqueda de instrucción alguna. De hecho,

en algunas instrucciones, como en las de carga y almacenamiento, estos intervalos se aprovechan para intercambiar datos con el sistema de memoria. En el caso de algunas instrucciones puntuales no queda más remedio que desperdiciar estos ciclos.

Una vez se haya lanzado el último paso de la instrucción con ciclos adicionales en ⓓ en el siguiente ciclo ⓔ se realiza la decodificación de la instrucción en secuencia, que en este caso se encuentra almacenada en la FIFO. En este ejemplo, como la instrucción 2 es *monociclo* la siguiente decodificación en ⓕ es la de la instrucción 3.

En el ciclo ⓕ se realiza la búsqueda de la instrucción 4, tanto por si se necesitase su decodificación en el intervalo siguiente (como es el caso en este ejemplo), como por si se tuviese que almacenar en la FIFO (en el caso de que la instrucción 3 hubiese necesitado pasos adicionales).

Otro caso muy especial: Transferencias de control

Para gestionar las instrucciones de transferencia de control los casos más sencillos son aquellos en los que la transferencia es incondicional. Para las transferencias de control condicionales (*saltos condicionales*) la situación se presenta complicada ante la organización segmentada de la arquitectura.

El flujo de instrucciones hacia el procesador normalmente viene constituido por instrucciones que se encuentran almacenadas en secuencia dentro de la memoria de programa. Sin embargo, cuando se desea realizar una transferencia de control a una instrucción que rompe dicha secuencia, entonces resulta deseable realizar este salto sin perjudicar las prestaciones. Por esta razón en la arquitectura SPARC las transferencias de control se realizan de modo retardado.

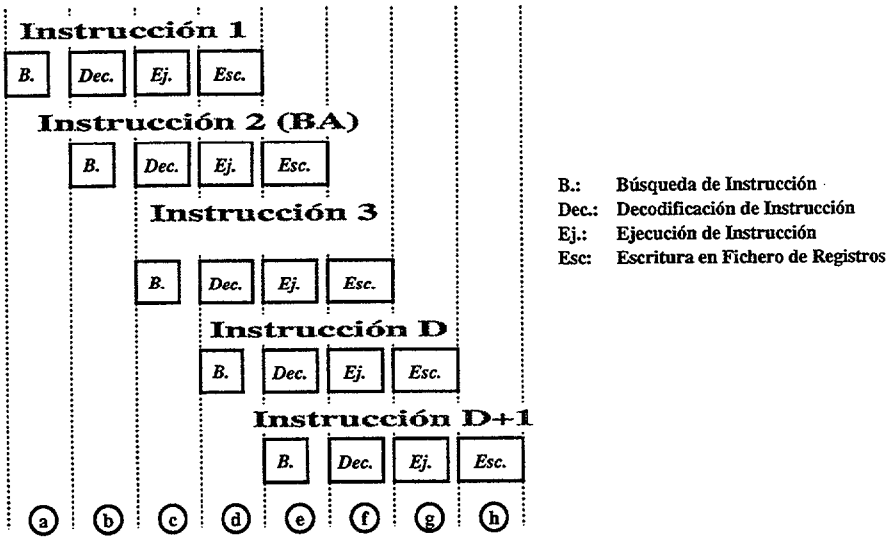


FIGURA 4.19: Temporización de una instrucción de transferencia de control incondicional en una arquitectura segmentada

Saltos incondicionales En la fig. 4.19 se puede ver un ejemplo de una instrucción de transferencia incondicional de control. Mientras esta instrucción se está decodificando

en ⓒ se está realizando la búsqueda de la instrucción en la ranura de retardo (*delay slot*) y se va calculando la dirección de salto. Esta dirección se pondrá en ⓓ para entonces realizar la búsqueda de la instrucción de destino del salto, al tiempo que se lanzará a ejecución la instrucción de la ranura de tiempo. Como se puede deducir, en ⓒ no se pone la dirección de salto que se calcula en este ciclo porque esto haría aumentar considerablemente el tiempo de ciclo, cosa que no debe hacerse.

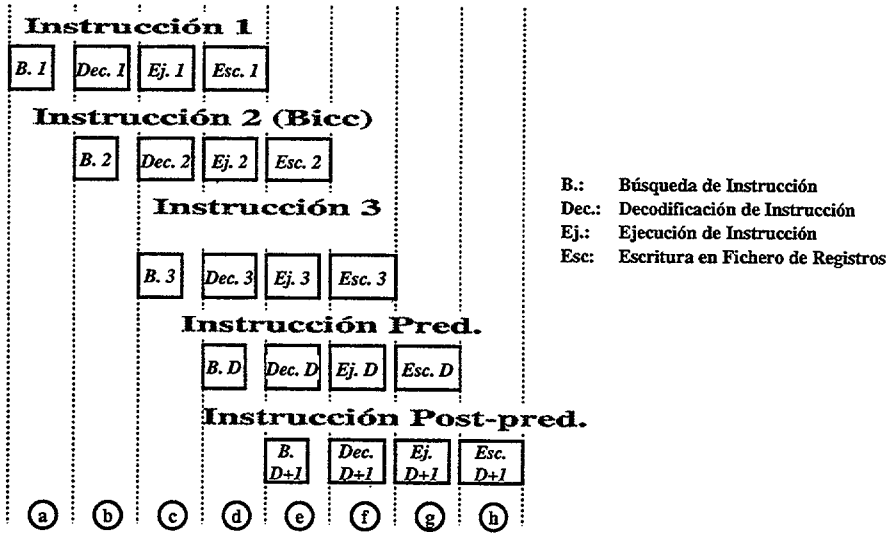


FIGURA 4.20: Temporización de una instrucción de transferencia de control condicional en una arquitectura segmentada

Saltos condicionales. Predicción de salto En el caso de las instrucciones de salto condicional la situación se torna un poco más complicada. Según se puede ver en la fig. 4.20, la instrucción inmediatamente anterior a la de salto puede estar modificando los códigos de condición en ⓒ, que es precisamente cuando la instrucción de salto condicional debe ir evaluando la dirección de salto. Entonces, en estas circunstancias, en ⓒ lo que se realiza es una predicción y se calcula la dirección correspondiente. De este modo, en ⓓ se realiza la búsqueda de la instrucción que se asocia a la predicción. Precisamente en este intervalo ⓓ la instrucción 1 ya habrá terminado de modificar los códigos de condición y entonces se estará en condiciones de averiguar si la predicción realizada ha sido buena o no. En función de esta evaluación se está en condiciones de determinar cuál debe ser la siguiente instrucción a buscar en ⓔ.

Ante este escenario, en el caso de los saltos condicionales nos podemos encontrar ante 4 situaciones distintas:

- Que la predicción haya dicho que se salte y que la evaluación posterior establezca que la decisión fue buena.
- Que la predicción haya dicho que se salte y que la evaluación posterior establezca que la decisión fue mala.
- Que la predicción haya dicho que no se salte y que la evaluación posterior establezca que la decisión fue buena.

- Que la predicción haya dicho que no se salte y que la evaluación posterior establezca que la decisión fue mala.

Sentido de la predicción y anulación En aquellas situaciones en las que la decisión fue mala se debe anular la ejecución de la instrucción que se buscó en la predicción. Además, debido a la especificación de la arquitectura, ocasionalmente se puede requerir la anulación de la instrucción en la ranura de tiempo. Como se puede observar en la fig. 4.20, para cuando se determine en ⓓ si la decisión fue buena o no aún habrá tiempo de cancelar la ejecución de la instrucción en la ranura de tiempo en sus pasos de ejecución en ⓔ y de escritura en ⓕ, y/o de la instrucción que resultó de la predicción, según convenga.

En el presente caso se obtendrá una arquitectura más optimizada o menos según el índice de éxito que se tenga en las predicciones, y siempre que el establecimiento de esta política no afecte al tiempo de ciclo. Para ello se ha debido hacer un análisis de los saltos condicionales que normalmente se encuentran en los programas, de su frecuencia y sentido, tal como se ve en la fig. 4.21 y 4.22.

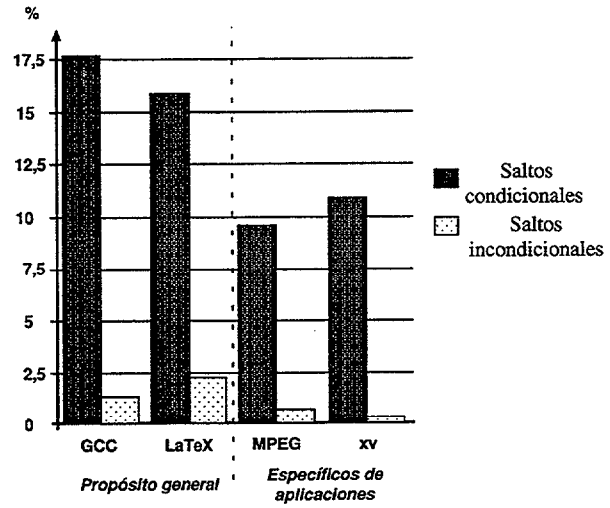


FIGURA 4.21: Proporción de instrucciones de salto encontradas en la ejecución de diversos programas de prueba para una arquitectura SPARC

Saltos sin predicción Además, también debe tenerse en cuenta que bajo determinadas circunstancias esta predicción se puede obviar y evaluar directamente los códigos de condición. Esta circunstancia se produce cuando la instrucción anterior al salto condicional no modifica la palabra de estado del procesador. Todos estas técnicas para realizar la predicción podrán acometerse con garantías mientras su realización no penalice las prestaciones del diseño.

Para clarificar estas cuestiones de diseño, retomemos el ejemplo de la figura 4.20. Como se podrá observar, si la *Instrucción 1* no modifica la palabra de estado del procesador en su etapa de ejecución, entonces una instrucción de salto condicional en secuencia puede evaluar la condición de salto sin problemas en su decodificación. De este modo, puede poner la dirección de la instrucción siguiente en el ciclo ⓓ sin riesgo alguno. Sin

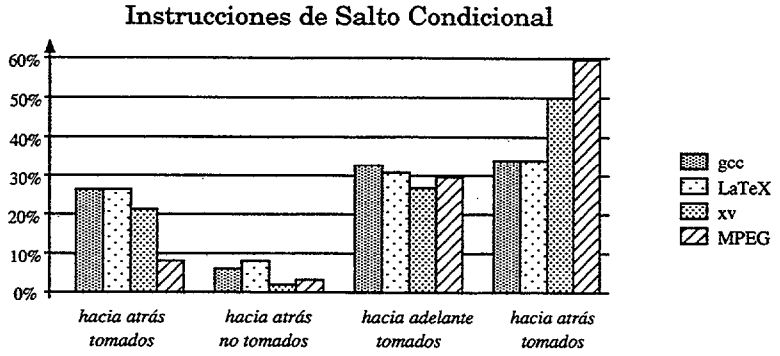


FIGURA 4.22: Resultados de la ejecución de diversos programas de prueba para el estudio de los saltos en una arquitectura SPARC

embargo, si la *Instrucción 1* modifica la palabra de estado en su etapa de ejecución, entonces para la búsqueda de la *Instrucción 3* se necesitará la dirección de una predicción. Cuando se haya rebasado la etapa de ejecución de la instrucción de salto ya se conocerá si la predicción fue buena o no, se podrá anular tanto la instrucción en la ranura de retardo como la del resultado de la predicción y se podrá progresar según la secuencia adecuada.

Otras implementaciones En otras implementaciones lo que se ha establecido como predicción es suponer que siempre se va a saltar. Sin embargo, del examen de algunas trazas de la ejecución de algunos programas de prueba (véase figura 4.22, pág. 74) se puede deducir que la mayor parte de las ocurrencias de saltos hacia atrás se toman, y en el caso de los saltos hacia adelante la mayor parte no se toman. Persiguiendo favorecer la situación más frecuente, y tomándose este hecho como referencia, se puede realizar una predicción más con mayor probabilidad de éxito y, aún así, sencilla.

En el caso de otras implementaciones es al final del intervalo © en la decodificación del salto cuando se termina de evaluar las condiciones para colocar la dirección correcta en @ [Tem96]. Sin embargo, esta otra alternativa presenta un inconveniente como se puede demostrar a continuación.

La mejora en el tiempo de ejecución total se puede estimar de un modo sencillo al tener en cuenta que con esta última predicción los casos menos favorables (los saltos hacia atrás que no se toman y los hacia adelante que sí), que son los que se predicen erróneamente, suponen en torno al 7 y 6,2% para aplicaciones generales como el gcc y LaTeX, y en torno al 3,1% para aplicaciones de presentación de imágenes, del conjunto de instrucciones ejecutadas.

En estas circunstancias, supóngase que el tiempo de ciclo normal estuviese en 10 ns, y que pasaría a unos 12 ns si se quiere que la evaluación de la condición se realice dentro de la etapa de ejecución de la instrucción de transferencia de control. En este caso, para un segmento de código que emplease 100 ciclos de la versión en 12 ns, corresponderían $100 + (0,07 \times 100) = 103$ ciclos en la de 10 ns[5], y por lo tanto

[5]Se ha escogido la suposición de que el 7% son saltos por ser el caso con mayores inconvenientes.

$$T_{\text{100 ciclos versión de 12ns}} = 1200\text{ns}$$

$$T_{\text{107 ciclos versión de 10ns}} = 1070\text{ns}$$

$$T_{\text{100 ciclos versión de 12ns}} > T_{\text{107 ciclos versión de 10ns}}$$

por lo que claramente con la segunda opción se gana una reducción del tiempo de ejecución total en torno al 11%. Además, como se puede observar esta ganancia se obtendría considerando el peor de los casos, cual es que siempre antes de una instrucción de salto condicional se encuentre una instrucción que pueda modificar la palabra de estado del procesador.

La gestión del secuenciamiento

Para llevar a cabo una operación correcta ante este tipo de instrucciones se han empleado unos registros especiales llamados *contadores de programa.* Estos registros suelen aparecer en la práctica totalidad de arquitecturas de procesador. En el caso concreto de las implementaciones de la SPARC IU que se presenta se han empleado cuatro de estos contadores, uno para cada etapa de segmentación.

La gestión de estos contadores debe hacerse teniendo en cuenta la temporización de las distintas situaciones que pueden presentarse en la ejecución de las instrucciones. Ejemplos de estas situaciones se han presentado anteriormente en las figs. 4.17, 4.18, 4.19 y 4.20.

Ante estos problemas que se salen de la situación *normal* se hace relativamente sencillo estudiar los distintos casos y adoptar soluciones adecuadas. Sin embargo, igualmente pueden presentarse situaciones algo más complejas cuando varias de estas instrucciones *conflictivas* se presentan en secuencia. En cualquier caso, las soluciones que se han tomado son lo suficientemente sencillas como para evitar penalización alguna en las prestaciones.

La gestión del contador de programa con el que se realiza las búsquedas se puede resumir tal como se indica en la tabla 4.1.

En la arquitectura que se ha diseñado el mayor número de instrucciones que pueden aparecer en la cola es de 2. Además, al realizarse la decodificación de las instrucciones se obtiene tanto el número de paso que posteriormente se debe realizar como saber si en el ciclo siguiente se debe recoger un dato o una instrucción nueva. En el caso concreto de las instrucciones multiciclo este *número de paso* valdrá en la primera decodificación de la instrucción algo distinto de cero y se recogerá en un registro especial. De este modo, en cada ciclo la decodificación se realiza teniendo en cuenta el estado de la máquina de control y el de otros elementos, como el contenido del *registro de instrucción* y el del registro donde se almacena el paso de instrucción que debe seguir en secuencia.

TABLA 4.1: Descripción de la gestión del PC para búsqueda (QPCFetch)

Etapa de segmentación	Instrucción monociclo
BI	BI/DI.RI ← (Si CI *vacía* {Mem[QPC]} si no {CI.Fondo});

Etapa de segmentación	Instrucción multiciclo
BI	BI/DI.RI ← (Si *Último Ciclo* {CI.Fondo}); (Si *Instrucción a recoger en Bus* CI.Cima ← Mem[QPC]

BI Etapa de **Búsqueda de Instrucción**
DI Etapa de **Decodificación de Instrucción**
RI **Registro de Instrucción**
CI **Cola de Instrucciones**

Esta estrategia se ha preferido a la empleada por otros diseñadores que se basan en la generación de instrucciones auxiliares. La razón de esto estriba en los siguientes aspectos:

- Si se introdujesen instrucciones auxiliares harían falta al menos un elemento con el que indicar que se tratan de instrucciones internas generadas por el procesador. Esto se debe a que, si el código que se emplea para esta instrucción auxiliar no coincide con ninguna de las indicadas en el manual de la arquitectura, entonces debería producirse una excepción en caso que se quisiera poner el código de una instrucción auxiliar desde el exterior. Y si esto no se hace así, supondría que las posibilidades de ampliación del juego de instrucciones se verían obstaculizadas por el número de instrucciones auxiliares que existiesen en el procesador. Esto supone al menos un biestable adicional cuyo contenido hay que tener en cuenta en la decodificación.

- Aunque con la alternativa elegida se emplea un biestable más que con la opción de otros diseñadores, no es menos cierto que con la opción anterior hacen falta elementos con los que generar de forma automática el código de la instrucción auxiliar y elementos con los que indicar que posteriormente se debe recoger la instrucción producida internamente. Estos otros elementos podrían ser de mayor coste que el que se produce con un biestable adicional.

- Al emplear el registro de control de pasos los modelos quedan más sencillos, tanto desde el punto de vista de gestión, depurado y modificación por el diseñador o el usuario.

Los Contadores de Programa

En los modelos realizados, el número de Contadores de Programa empleados son tres. Estos tres contadores almacenan, por lo general, las direcciones de las instrucciones de

las que se realiza su búsqueda, decodificación y ejecución en cada momento. Desde el punto de vista de la organización de la arquitectura, estas fases se corresponden con etapas de la segmentación fácilmente localizables en la ruta de datos (fig. 4.11) y en la unidad de control. Con estos contadores se permite en cada momento realizar los cálculos necesarios para conocer la dirección de la siguiente instrucción a procesar.

En caso de interrupción, el número de contadores de programa a guardar es de dos, que son los correspondientes a la instrucción ejecutándose y la que debería seguir a ésta. En la especificación de la arquitectura [SPA91] estos contadores son identificados como PC y nPC. Guardar estos dos contadores resulta suficiente para recuperar la ejecución normal tras completar la gestión de una interrupción. Esto es porque una vez terminada la rutina de servicio de una interrupción determinada

- cuando se debe volver a ejecutar la instrucción interrumpida, se debe recoger nuevamente el valor de PC guardado, y
- cuando se desea proseguir con la instrucción posterior a la interrumpida, se debe recoger el valor de nPC. Este valor no tiene necesariamente que coincidir con PC + 4, como sucedería al producirse una interrupción en la ranura de tiempo (*delay slot*) tras una instrucción de bifurcación.

Cancelación de instrucciones y cálculo del PC siguiente

Existen diversas situaciones bajo las que se debe evitar que la ejecución de una instrucción se complete. Estas situaciones son las siguientes:

1. Por tratarse de una instrucción que se debe anular en la ranura de tiempo de un salto.
2. Por tratarse de una instrucción que se ha recogido en una predicción errónea.
3. Por tratarse de una instrucción situada tras una instrucción en la que se ha producido una excepción o interrupción.
4. Por tratarse de una instrucción que ha producido una excepción al intentar realizar una operación no permitida.

Estas cuatro situaciones tienen como característica común que se detectan, en caso de producirse, en la etapa de ejecución. Sin embargo, por la naturaleza del tipo de evaluación a realizar en cada caso, la influencia de cada una de ellas debe estudiarse cuidadosamente.

Como regla general, y que se aplica como norma dentro de la Ruta de Datos, la cancelación de la ejecución de una instrucción se realiza impidiendo que cualquiera de los registros que representan el estado de las operaciones en el procesador se modifique. Esto básicamente se suele hacer forzando a las señales de control de escritura de estos registros al valor inactivo.

Sin embargo, en la Unidad de Control la cancelación de una instrucción debe ser objeto de una atención especial. La cancelación de una instrucción de salto no tiene en todos los elementos de esta Unidad el mismo tipo de influencia que en los de la Ruta de Datos, y de hecho la gestión del PC resulta especialmente delicada ante estas situaciones. En el caso de las dos primeras circunstancias de cancelación, la evaluación de estas situaciones se debe realizar de forma lo suficientemente rápida como para no tener incidencia sobre el tiempo de ciclo siempre que se elija una estructura adecuada para el cálculo del PC siguiente en saltos o en secuencia. En la tercera circunstancia las demandas de tiempo no son tan apremientes como en las demás. Para la última situación su tratamiento se realiza de forma diferente a las demás influyendo directamente en la evolución de la máquina central de estados finitos que gobierna el procesador.

En la figura 4.23 puede verse una estructura que se ha tomado como base de este cálculo, con algunas variantes que respondían a los mecanismos de secuenciamiento concebidos. El valor de algunas de sus señales de control depende única y exclusivamente del estado de la FSM central y de la instrucción que se decodifica. Son éstas las circunstancias que, en parte, deben determinar cómo calcular el siguiente valor del PC. Sin embargo, como también deben tomarse en consideración otros elementos para decidir cómo realizar este cálculo, existen otras señales de control que toman sus valores en función de la evaluación de distintas situaciones. Por ejemplo, en este diseño las señales `IncPC` y `S2_NTF_Ref_Sel_Ctl`, cuyos valores sólo están disponibles una vez se haya tomado la decisión si saltar o no. Como se puede observar en la figura, la estabilidad en los valores de estas señales es determinante para empezar a realizar la operación adecuada, por lo que pueden aparecer rutas críticas ligadas al control.

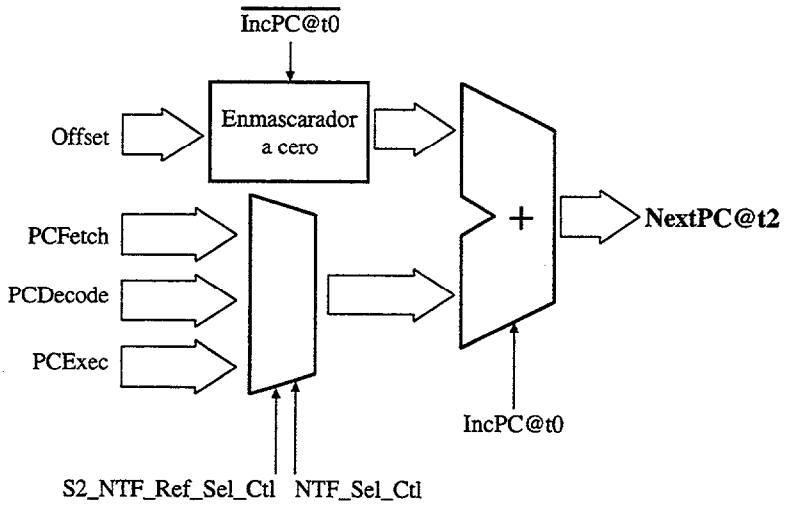


FIGURA 4.23: Estructura lenta para el cálculo del PC siguiente

En estas circunstancias, para evitar esta dependencia tan acusada, esta estructura puede sustituirse por otras estructuras en las que la cantidad de circuitería necesaria crece. Si se desea mantener la funcionalidad dentro de cada ciclo, una estructura válida podría basarse en el cálculo previo de las sumas que se pueden producir y la selección adecuada de alguna de ellas con multiplexores (véase la fig. 4.24). Para evolucionar a situaciones intermedias según las necesidades de tiempo, en ocasiones se puede permitir que sea la herramienta de síntesis quien realice la elección más adecuada.

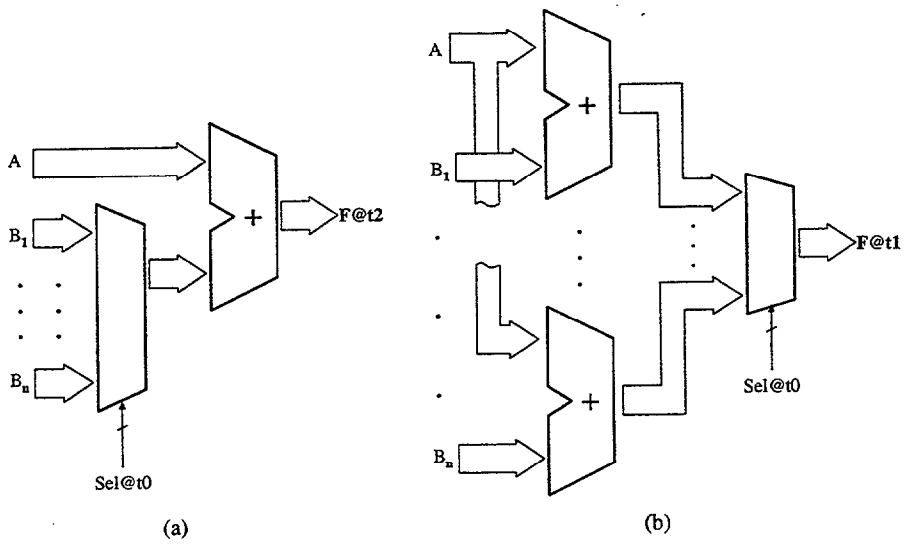


FIGURA 4.24: Estructuras de funcionalidad equivalente y distinta temporización

Sin embargo, también se ha optado por realizar versiones con una unidad de secuenciamiento rápida, de estructura intermedia, con sólo 2 sumadores, que se muestra en la fig. 4.25. Esto se ha realizado en previsión de que, al realizarse la retemporización en la estructura de la fig. 4.23 el coste fuera mayor que esta alternativa. Las estructuras lenta y rápida se presentan con mayor detalle de diseño en las figs. B.1 y B.2 del apéndice B. El funcionamiento de esta estructura es tal que, en cada instante, se calculan simultáneamente tanto la dirección PC + 4 como la que correspondería en caso de salto. Según sea el valor de las señales `S2_NTF_Ref_Sel_Ctl` e `IncPC`, que controlan los multiplexores, en esta ocasión el retraso que se consigue es menor al tratarse de señales que no son determinantes para establecer las entradas de los dos sumadores.

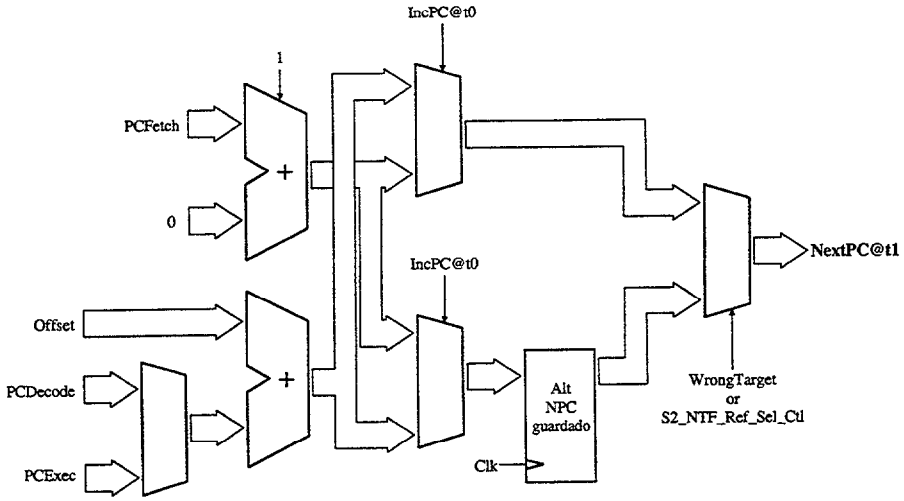


FIGURA 4.25: Estructura rápida para el cálculo del PC siguiente

Por lo que respecta a la última de las condiciones de cancelación de una instrucción, que es la producida por una excepción, se ha debido tener un especial cuidado en los elementos a los que debe afectar. Esto se debe a que la determinación de qué tipo de excepción debe atenderse en un determinado instante es una tarea mucho más costosa

en tiempo que las evaluaciones realizadas en saltos condicionales. Para las excepciones las condiciones a evaluar son muchas más y además el lanzamiento de una excepción determinada debe realizarse atendiendo a una jerarquía de prioridades definida por la especificación de SPARC [SPA91]. En este caso, se ha intentado que las señales que indican una petición de excepción viajen a través del menor número de niveles de lógica para no penalizar el tiempo de ciclo.

Situaciones de excepción

Las verificación de las excepciones e interrupciones se realiza en la etapa de ejecución de las instrucciones. El lanzamiento de una determinada excepción puede permitir o no completar la ejecución de la instrucción que se interrumpe. Una vez se detecta una petición de interrupción se contrasta su prioridad con el resto de interrupciones que se puedan haber solicitado en el mismo ciclo. Una vez evaluadas todas las interrupciones solicitadas se procede a gestionar aquella de mayor prioridad.

La gestión de una excepción determinada supone alterar la estrategia de secuenciamiento establecida para buscar las instrucciones de un programa. El comienzo de esta gestión se inicia desde la máquina central de estados del procesador, que evoluciona temporalmente hacia estados especiales. En estos estados especiales las instrucciones que quedan pendientes de completarse son anuladas y se inicia la búsqueda de las instrucciones correspondientes a la rutina de servicio. Una vez se haya realizado esta inicialización el procesador vuelve al estado normal de ejecución pero operando en modo supervisor y ejecutando la rutina de servicio. Es esta rutina la que decide cuándo proseguir con la ejecución de las instrucciones del programa interrumpido.

Otra situación de excepción que merece mencionarse es la debida a fallos de búsqueda en la caché (*cache misses*). El gobierno de estas situaciones debe realizarse de forma externa al procesador. Por lo general, el tratamiento de estas circunstancias la realiza una Unidad de Gestión de Memoria (MMU).

Las instrucciones TADDccTV y TSUBccTV

SPARC define dos instrucciones que causan una excepción cuando la operación que se realiza sobre dos operandos produce desbordamiento. La señal de desbordamiento procede de la ALU, se debe verificar y la excepción que podría ocasionar debe contrastarse con otras excepciones que se pudieran solicitar en el mismo ciclo. Esto se necesita para cumplir con las prioridades de las excepciones definidas en la arquitectura. Como en este diseño la posible generación de una excepción se realiza precisamente en la etapa de ejecución, lo más directo sería realizar la operación de suma o resta en esta etapa y sobre la marcha evaluar la posibilidad de generar o no la excepción correspondiente. Esto haría que el tiempo de ciclo aumentara respecto a la situación normal del resto de las instrucciones en las que no se necesita un tiempo de ciclo de tal duración. Esto sería así porque al retraso asociado al flujo de datos a través de la ALU se le añadiría el asociado a la verificación de excepciones y al de la gestión de secuenciamiento.

Para evitar una ruta tan larga se ha hecho ocupar a estas instrucciones un ciclo extra,

de forma que en el primer paso de ejecución se realiza la operación necesaria en la ALU y la señal de desbordamiento se almacena en un registro especial. En el segundo paso de ejecución se evalúa la condición para permitir la excepción asociada o no y se permite (o impide, en caso de tener que producirse la excepción) la escritura del resultado.

De este modo se evita que sean estas instrucciones las que fijen el tiempo de ciclo. Es más, se da la circunstancia de que en las trazas de diversos programas analizados (donde no se emplea aritmética marcada) esta instrucción no se produce nunca[6], con lo que todavía tendría menor sentido que fuera una de estas instrucciones la que estableciese los condicionantes para el tiempo de ciclo.

Instrucciones auxiliares

Algunas implementaciones de SPARC emplean instrucciones auxiliares internas [Cat91]. Estas se emplean en caso que las instrucciones necesiten pasos de ejecución adicionales para completarse. Cuando una de estas instrucciones se decodifica, se genera una instrucción interna nueva y se alimenta en el procesador para ejecutarla a continuación.

Esto presenta dos problemas:

- Se debe generar un bit adicional y después verificarse, ya que las instrucciones internas necesitan una gestión diferente. Por ejemplo, si una instrucción interna tiene un código que no coincide con ninguna externa, y una instrucción externa entra en el procesador con ese mismo código, esta situación anómala debe detectarse. El único modo de lograrlo es con una señal adicional.

- Se necesita circuitería extra para generar las instrucciones internas nuevas.

Una solución alternativa para este problema es tener bits adicionales para identificar cuántos pasos de ejecución adicionales se necesitan. Los registros adicionales para almacenar esta información se deben incluir. Con esta alternativa, cuando una instrucción necesita pasos de ejecucion adicionales ésta se retiene en el registro de instrucción y las señales de control de su decodificación dependen del paso de ejecución a realizarse. En cada uno de estos pasos de decodificación se emplea la información de registros de cuenta de pasos, y se proporciona un valor nuevo para estos contadores para conocer el paso que debe ejecutarse posteriormente.

Con esta alternativa, que ha sido la adoptada, la circuitería para construir las instrucciones nuevas internas se ha eliminado y sólo se necesita un bit adicional. Esto es así porque el máximo número de pasos que se deben realizar en secuencia es de tres. Esta decisión también tiene la ventaja de que los modelos ganan en claridad.

[6] Cuando menos podemos tener la certeza que la aparición de esta instrucción en los programas es muy baja.

Indices CPI de la microarquitectura

Para aplicaciones multimedia se observa una mezcla instrucciones con un 16.2% de carga, 8.1% de almacenamiento, 3.1% de bifurcación con predicción errónea y 72.6% de otras instrucciones con un único paso de ejecución (que incluye las bifurcaciones con predicción correcta). Basándose en la tabla 4.2, la contribución de las cargas, almacenamientos, saltos, instrucciones anuladas, cargas con ciclo adicional por *interlock* (suponiendo que un 50% de las cargas provoquen *interlock* con la siguiente instrucción) y saltos largos, al CPI —excluyendo los fallos de búsqueda en caché— resulta aproximadamente en 1.436 para una de las microarquitecturas desarrolladas[7].

TABLA 4.2: Ciclos de Ejecución de las Instrucciones

Tipo de Instrucción	Nº de ciclos	Nº de ciclos de instrucción adicionales
Carga (palabra/media-palabra/byte)[1]	2	1
Carga (doble-palabra)[1]	3	2
Almacenamiento (palabra/media-palabra/byte)	3	2
Almacenamiento (doble)	4	3
Carga-almacenamiento atómico	3	2
"*Jump and Rett*"	2	1
Bifurcación (con predicción errónea)	2	0
Bifurcación (con predicción correcta)	1	0
Instrucción tipo *tagged* con comprobación de excepción	2	1
Resto de instrucciones	1	0

[1]Cuando se necesita *interlock* se incluye un ciclo de instrucción adicional.

Para dar otra referencia conveniente, para un entorno de propósito general, las decisiones microarquitecturales de la versión que se ha mencionado producen un CPI de 1.4545 para la ejecución del compilador GCC.

4.4. Otras consideraciones sobre el modelado del procesador

4.4.1. Modelado de la subsección de generación de señales de control

Por las razones anteriormente expuestas, la estrategia arquitectural propuesta para la generación de señales de control es dividir esta subunidad de la Unidad de Control en una FSM pura, de la que sólo es visible el estado del procesador, y otro bloque sin elementos de memoria del que se extraen todas las señales de control, además de un bloque diferenciado para gestión de las interrupciones. Dentro de los procesadores RISCs estos bloques tienen propiedades especiales que hacen que su modelado resulte posible y fácil en VHDL.

[7]Concretamente la que mejores prestaciones proporciona en muchos casos tras la realización física, con las opciones 2.b, 4.b y 5.a que se presentarán en el apartado 5.2.

En los RISCs, por ejemplo, la FSM que se menciona suele tener muy pocos estados en los que evolucionar. Para el desarrollo de SPARC esta FSM ha resultado de sólo 7 estados (véase la fig. 4.26).

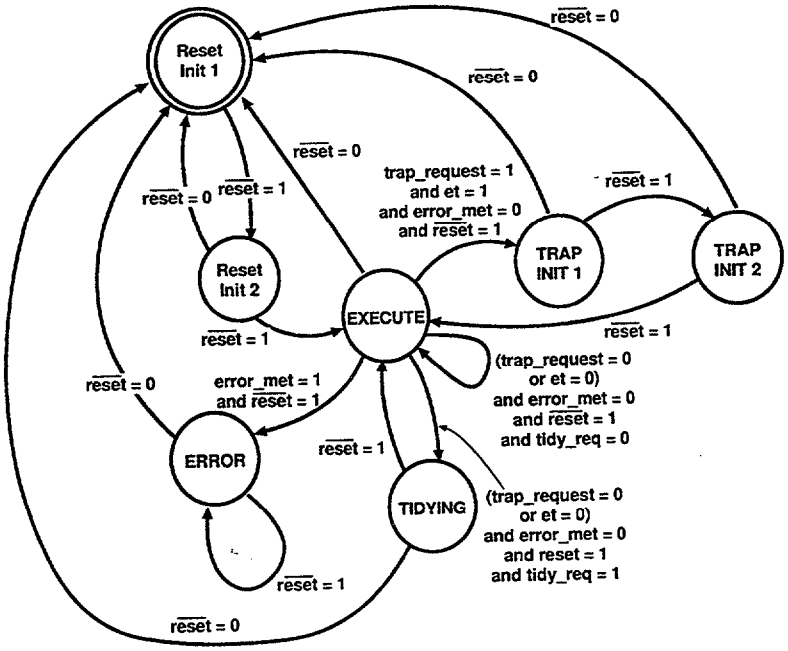


FIGURA 4.26: Diagrama de estados de la FSM de control de la IU de SPARC v8

Además de las prestaciones, esta organización presenta ventajas adicionales tanto en el diseño como en la depuración y empleo de los modelos. Entre éstas cabe mencionar que, en lo que respecta al empleo, cuando se desea incorporar una instrucción en el juego de instrucciones para su decodificación se necesita actuar sobre un único elemento del diseño. El código VHDL tendrá correspondencia con el código y formato de instrucción nuevo y con la generación de las señales de control adecuadas para que operen los elementos que intervienen en su ejecución en la forma necesaria. Esta estrategia facilita tanto el diseño y la depuración como la realización de cualquier modificación que se desee hacer en el juego de instrucciones, o en la operación de una instrucción en particular. Esto se debe a la localización sencilla e inmediata en un sólo elemento de los valores de las señales que gobiernan los componentes de la arquitectura. De otro modo se tendría que estar investigando y modificando distintos elementos simultánea y coordinadamente. Al mismo tiempo, esta organización puede presentar ventajas cuando se diseñan estos modelos con fines comerciales, ya que el modelo del resto de la arquitectura se puede hacer invisible, y permitirle al usuario final que sólo pueda retocar la sección correspondiente al juego de instrucciones para adaptarlo a las necesidades de su aplicación cuando así lo requiera. Esto es bastante atractivo sobre todo cuando se desean protejer derechos de propiedad intelectual, IPR, en la comercialización de estos núcleos como núcleos IP.

El elemento de generación de señales de control se puede describir con un proceso en el que todas sus variables[8] y todas sus entradas aparecen en la lista de sensibilidades. De este modo se garantiza que la herramienta de síntesis no introducirá elementos de

[8]En este diseño no existen para evitar problemas

almacenamiento. Dentro de este proceso primeramente deberán aparecer los valores predeterminados de estas señales y posteriormente algunas de estas señales cambiarán de valor dependiendo del estado de la FSM central y/o del estado de diferentes elementos de la máquina tales como peticiones de interrupción, contenido del registro de instrucción y otras evaluaciones especiales. Este elemento es fácilmente configurable y su depuración se puede realizar de modo sencillo, lo que facilita la labor tanto al diseñador como al usuario. Dentro de este proceso se incluye la lógica de decodificación a través de una estructura tipo `CASE`. El resultado de la síntesis sobre este elemento siempre debe ser lógica puramente combinacional.

Siguiendo esta organización, el diseñador del núcleo tendrá siempre control completo sobre los resultados de la síntesis. Este es un aspecto fundamental en relación tanto con la depuración del diseño como con las prestaciones de la implementación final.

4.4.2. Consideraciones adicionales con carácter general

Si bien existen recomendaciones concernientes al modelado en VHDL de diseños digitales en general (por ejemplo, [Cha97, Sin94]), debido a la naturaleza de este tipo de diseños, y con las pautas anteriormente expuestas, se estiman convenientes estas recomendaciones adicionales:

- Utilización de una única señal de reloj y sólo uno de sus flancos para todos los *flip-flops* disparados por flanco, y utilización de un solo nivel para los elementos de memoria activos por nivel mientras sea posible. Esto producirá un diseño completamente síncrono con un esquema de temporización sencillo. No se recomienda la utilización de distintas señales de reloj, ya que no sólo puede ocasionar problemas arquitecturales o demandar esquemas de temporización rígidos —con los inconvenientes que ocasionaría a la hora de comprender el diseño, o realizar extensiones o modificaciones—, sino que además resulta menos inmune a posibles problemas de ruido cuando se utilizan en entornos hostiles. Además, se produciría mayor longitud de interconexiones, aumentando el riesgo de problemas por retrasos en la señal de reloj o *skew*.

- No se deben utilizar elementos de memoria internos en los elementos fundamentales excepto los explícitos como memorias, ficheros de registros, registros de propósito general y similares. Es recomendable denotar explícitamente los elementos de memoria como tales y hacerlos visibles en el diagrama. De este modo el diseñador obtiene un mayor control sobre el proceso final de síntesis, y facilita el tratamiento de cualquier incongruencia que pueda haber realizado en el modelado de bloques del núcleo.

- En concordancia con el punto anterior, no se deben utilizar variables de almacenamiento en el proceso de generación de señales de control, y en caso de utilizarlas deben aparecer en la lista de sensibilidad. Asimismo todas las señales de entrada de este bloque deben aparecer en esta lista de sensibilidad.

- Es altamente recomendable la inclusión de algunas estimaciones simples de retardos en el modelado de las subunidades especialmente para los elementos de almacenamiento activos por nivel. De este modo se puede tener una mejor visión de cómo se van a comportar los módulos generados con otras herramientas en el contexto de nuestro diseño. Estos retardos se ignoran en el proceso de síntesis pero son recomendables para propósitos de depuración.

- Utilización de tipos de paquetes estándares mientras sea posible, y abstenerse de utilizar tipos definidos *ad hoc*. Esto permite que los modelos sean fáciles de comprender, gestionar, depurar y modificar por otros usuarios. Una excepción específica hay que hacerla con los estados de la FSM central, donde la definición de un tipo enumerado para ellos facilita su depuración y comprensión.

- Los nombres de los elementos del diseño no deben ser excesivamente largos, si bien se recomienda que sean claros para cualquier futuro usuario de estos modelos. Esta recomendación es crítica sobre todo cuando se produce el intercambio de la información del diseño con otras herramientas que imponen restricciones en cuando al tamaño de estos nombres. Un caso bastante peculiar es el derivado de la utilización de genéricos, ya que tras el proceso de síntesis en Synopsys aparecen los elementos con nombres a los que se les ha anexado los genéricos a su nombre original, construyendo un nombre más largo. Por ejemplo, un elemento llamado `temp_barrier` con el genérico `cycles` puesto a 16 y el genérico `resetvalue` puesto a 0 se convierte en un elemento llamado `temp_barrier_cycles16_resetvalue0`.

4.5. Flujo de diseño

La metodología que se ha presentado anteriormente se ha adoptado para las primeras etapas del proceso de desarrollo, a partir del cual se puede obtener el núcleo. En la fig. 4.27 se presenta el flujo de diseño completo.

Como se puede observar, la entrada de usuario inicial es el desarrollo de una librería completa de modelos en VHDL de los componentes básicos que formarán parte de los niveles jerárquicos más bajos del diseño. Estos elementos de librería no se cambiarán. Para obtener altas prestaciones, algunos módulos de la arquitectura considerados como críticos en tiempo y/o área se podrán obtener de un generador de módulos. En el presente trabajo se han utilizado los de la herramienta Epoch de *Cascade Design Automation* para este particular. Una vez que se han obtenido los modelos de todos los módulos básicos, éstos se interconectan dentro de una herramienta de esquemáticos. De esta forma el diseñador siempre puede tener una visión intuitiva de la organización y distribución de las operaciones a través de las representaciones gráficas que suponen estos esquemáticos. En el presente desarrollo se han empleado las del SGE de *Synopsys*. Es en este punto donde se deben establecer las decisiones arquitecturales. Esta etapa es la más crítica, ya que las decisiones que se realizan en esta fase influyen seriamente en las prestaciones del sistema entero.

A partir del SGE se puede obtener los primeros modelos en VHDL del procesador. Tras unas primeras simulaciones con ellos, los resultados deben evaluarse meditadamen-

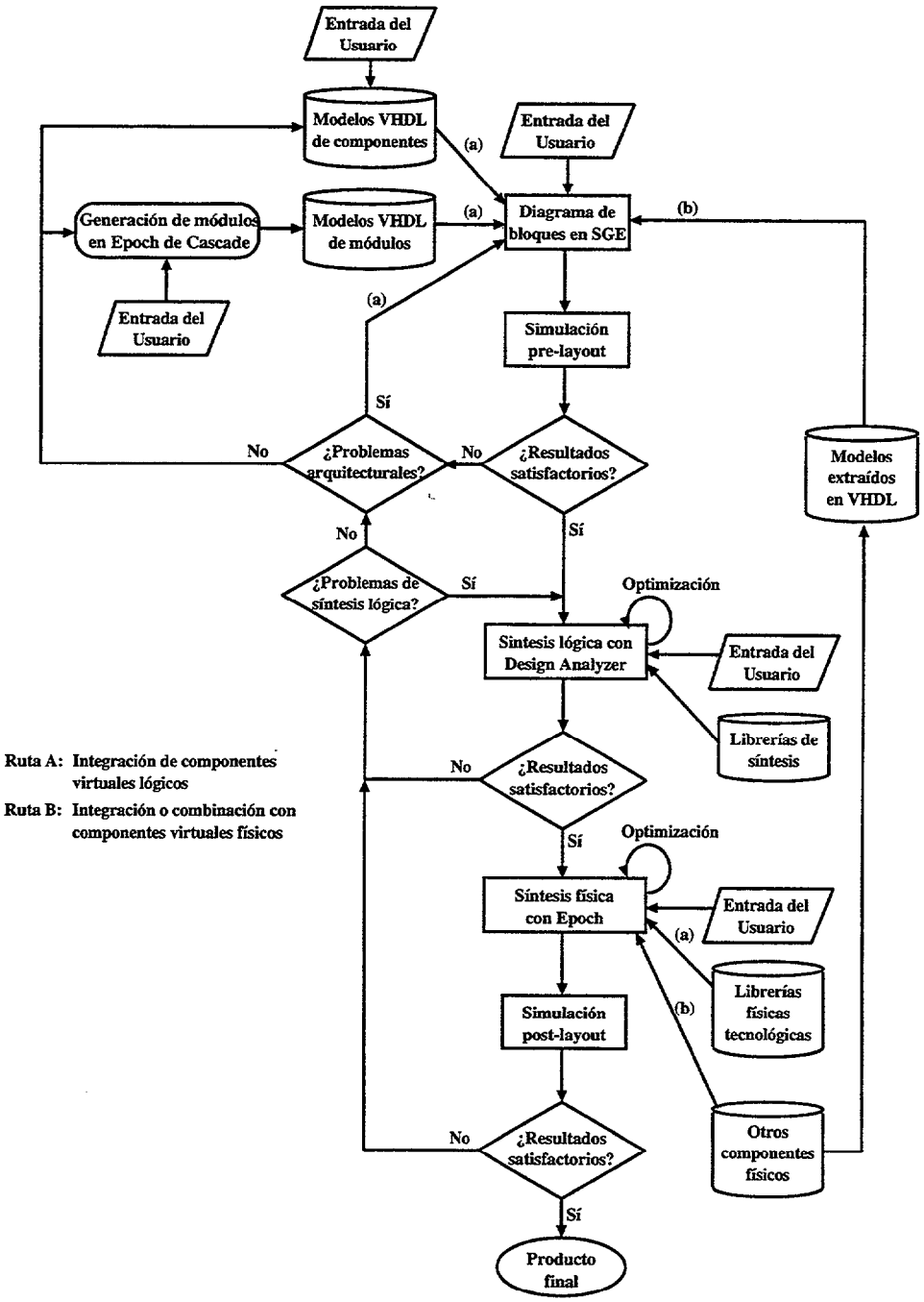


FIGURA 4.27: Flujo de diseño

te. Si no son satisfactorios, se debe verificar si los problemas son de tipo arquitectural o si son debidos a un mal modelado de alguno de los módulos. En el primer caso, los cambios se pueden realizar sobre el esquemático empleando el entorno gráfico. Si no, entonces el problema debe localizarse en el modelo de los componentes de la librería afectados y corregirse.

Cuando los resultados de estas simulaciones funcionales cumplan las especificaciones, el siguiente paso a seguir será la síntesis lógica con las herramientas de Synopsys, empleando las librerías de síntesis que Cascade proporciona. Tras realizar algunos ajustes sobre restricciones dentro de la herramienta, se hace necesario evaluar los resultados de esta síntesis lógica. Si no son del agrado del diseñador ahora los cambios hay que realizarlos o bien en la etapa de descripción o bien nuevamente en la etapa de síntesis lógica.

Una vez completada la síntesis lógica, el próximo paso es la síntesis física con la herramienta Epoch de *Cascade Design Automation*, utilizando librerías físicas. En caso de no obtener los resultados deseados los problemas pueden ser producto tanto de una evaluación errónea en la etapa de síntesis como de comportamientos imprevistos en los elementos físicos. En ambos casos se hace necesario volver a alguno de los pasos anteriores.

El paso final es una simulación sobre el *layout* final, donde la funcionalidad global debe coincidir con la de los primeros modelos que se obtuvieron en este flujo de diseño. Si los resultados no son satisfactorios en cuanto a las prestaciones podemos volver a realizar cualquiera de los apartados anteriores tras los cambios adecuados y según convenga.

La validación del diseño se realiza primeramente haciendo ejecutar sobre los modelos del núcleo pequeños segmentos de código escritos a mano. Posteriormente se simulan programas apropiados construidos con compiladores y ensambladores disponibles. Los programas resultantes en código máquina se ejecutan como programas de prueba o *benchmarks* sobre los modelos desarrollados para así poderse evaluar los resultados para posibles optimizaciones del comportamiento de la arquitectura. De los datos de las trazas dinámicas se pueden obtener resultados de este procesado. Esto es crucial para decidir donde realizar mayor énfasis cuando se replantean aspectos clave de la arquitectura.

4.5.1. Ayudas a las transiciones en el flujo de diseño

En las evoluciones desde unos elementos de este flujo de diseño a otros normalmente se precisa la intervención del usuario. En ocasiones esta intervención conlleva la modificación de ficheros que resultan del procesamiento del diseño en un determinado paso para que puedan ser interpretados por pasos posteriores. En otros casos la intervención del usuario puede ocasionar la preparación del entorno para que el paso siguiente se pueda realizar sin problemas.

Para facilitar muchas de estas situaciones, y fundamentalmente para las más monótonas, se ha elaborado un conjunto de *scripts* (muchos de ellos en Perl 5 [WCS96]) que facilitan este tipo de transiciones. No obstante, para completar los pasos con éxito, se han establecido unos criterios que resulta conveniente respetar:

- Los ficheros que llamen a módulos de Epoch deben cumplir lo siguiente:
  - su nombre comenzará con el prefijo `hm_`,
  - dentro de la entidad deben contener el atributo `fixedblock` puesto a 0 ó a 1. Lo normal debe ser que este atributo esté puesto a 0, ya que así el dimensionamiento del módulo será afrontado por la propia herramienta a la hora de incorporarlo al diseño que lo contiene.

  Se recomienda colocarlos en el directorio específico de elementos para Epoch, llamado `for-epoch`.

- También es posible la utilización de módulos algo más complejos dentro de nuestro diseño. En tal caso, para éstos se necesitará un paso de síntesis diferenciado. Éstos deben cumplir lo siguiente:
  - su nombre comenzará con el prefijo `sb_`,
  - dentro de la entidad *del resultado de la síntesis* podrán contener el atributo `fixedblock` puesto a 1. De esta forma se puede controlar su ubicación en caso necesario. No obstante, Epoch recomienda que esto no sea así para este tipo de diseños. Suele tener más sentido controlar la ubicación de módulos más grandes y bloques circuitales completos, como veremos en el capítulo 5.

A continuación iremos describiendo los *scripts* elaborados en función del orden de utilización dentro del flujo de diseño.

El script `prnet`

Una de las situaciones más paradójicas que se nos ha presentado ha sido al intentar realizar la síntesis lógica de un diseño realizado con el SGE.

Cuando se introduce un símbolo de un diseño previamente realizado en un esquemático, la descripción en VHDL es completamente estructural y se corresponde fielmente con el esquemático tanto en interconexionado como en elementos constituyentes en forma de *netlist*.

Sin embargo, a la hora de intentar preparar la síntesis realizando la compilación del diseño con

```
% vhdlan -spc_elab descripcion.vhd
```

en numerosas ocasiones nos producía un error como el siguiente:

```
SPC_Error: Cannot determine type of the aggregate
        in routine DESCRIPCION line 80
            in file '/home/usuario/disenos/descripcion.vhd'
        (This error can occur if an aggregate and a generic appear in the
        same component instantiation.) (HDL-206)
```

Esto se debía a que en el fichero VHDL de la *netlist* aparecían genéricos actuales de la siguiente forma:

```
a(18 downto 0) => b(20 downto 2)
```

Entonces, ante esta situación, el fichero VHDL de la *netlist* se tenía que modificar con un post-procesado, de forma que las líneas como la anterior se cambiaran por

```
a => b(20 downto 2)
```

Para facilitar esta tarea y no tenerla que realizar editando uno a uno los ficheros correspondientes se elaboró el *script* de Perl `prnet`[9].

Scripts de síntesis lógica

La síntesis lógica con `dc_shell` se realiza en dos pasos, y se ayuda de unos *scripts* de síntesis. De forma general, estos *scripts* están organizados de la forma siguiente:

`iuv8_core_NN-pass-01.scr`: Es el que establece cómo realizar el primer paso de la síntesis lógica. Al culminar este paso se escriben unos ficheros con la extensión `.wscr` en los que se indica distintas características que poseen algunos elementos del diseño —tales como carga existente en sus salidas y restricciones de tiempo y área— que serán empleados en un segundo paso de síntesis y permitirán un diseño más optimizado.

`iuv8_core_NN-pass-02.scr`: En él se indica cómo realizar el segundo y definitivo paso de la síntesis lógica. Una vez concluido este paso se tendrá un mapeado del diseño en forma de descripción estructural.

`iuv8_core_NN-setup.scr`: Establece algunas variables necesarias para realizar la síntesis.

`iuv8_core_XX-addons.scr`: Sirve para realizar algunas operaciones muy específicas — como poner la propiedad `ungroup`— sobre algunos elementos del diseño.

`iuv8_core_XX.con`: En este *script* se definen las restricciones deseadas para todos y cada uno de los elementos de la arquitectura y para el diseño de mayor jerarquía. Este paso es leído en el primer paso y su información se transfiere al segundo a través de los ficheros con la extensión `.wscr`.

`ungroup_dwares.scr`: Con él se desagrupan los elementos mapeados con *designwares* para permitir una mayor optimización lógica donde sea posible.

[9]No obstante, este paso debe controlarse con cuidado, ya que en determinadas ocasiones esta modificación no debe realizarse.

Los *scripts* con la partícula intermedia `NN` serán sustituidos por otros con el número que designe a la variante que se sintetizará.

Para producir la síntesis lógica con `dc_shell` ésta se producirá dentro de un directorio preparado para ello. De esta forma todos los procesos del diseño quedan perfectamente organizados y aislados unos de otros en lo posible. Para establecer de forma inicial el entorno de síntesis con los *scripts* correspondientes se ha elaborado un *script* de Perl *prsyndir* que prepara el directorio correspondiente y realiza las copias de los *scripts* de síntesis y los modifica para adaptarlos a la síntesis de la variante correspondiente. De esta forma, si se va a realizar la síntesis de la variante '04' de la Unidad de Enteros, entonces se creará un directorio llamado *syn-iu04* que contendrá los ficheros de establecimiento del entorno de Synopsys apropiado —`.synopsys_dc.setup`, `.synopsys_sge2vhdl.setup` y `.synopsys_vss.setup`— las copias modificadas `iuv8_core_04-pass-01.scr`, `iuv8_core_04-pass-02.scr` y `iuv8_core_04-setup.scr` y las no modificadas `iuv8_core_XX-addons.scr`, `iuv8_core_XX.con` y `ungroup_dwares.scr`.

Una vez preparado el directorio de síntesis tan sólo hace falta lanzar los dos pasos de síntesis lógica, con una pequeña modificación adicional de los ficheros escritos en la primera fase. Esta modificación permite que las propiedades que se describen para determinadas señales que han sido cambiadas de nivel activo[10] en algún paso de optimización sean recogidas después por las señales originales correspondientes en la segunda fase.

Al completarse la síntesis lógica, si no existen errores, deberá aparecer un fichero con el nombre `syn-iuv8_core_04.vhd`, que corresponderá con la descripción estructural mapeada en la tecnología elegida del diseño original. Este fichero será el que sirva de referencia para completar la síntesis física con Epoch.

Scripts de síntesis física

Para realizar la síntesis física con Epoch se han elaborado varios *scripts* que facilitan la tarea. En esta ocasión el punto de partida debe ser el resultado de la síntesis lógica descrito anteriormente, y que se trata de una descripción estructural de una *netlist*. Examinando esta descripción se puede comprobar que existen elementos que la entrada de VHDL de Epoch no es capaz de entender. Para estos elementos el primero de los *scripts*, `s2e`, realiza un preprocesado del fichero VHDL original para que no se produzcan problemas por esta razón. Además, se examina la existencia de módulos de Epoch y se les añade determinados atributos según lo que el usuario decida. De igual forma, existe otro *script* denominado `extrfb` con el que se extrae el nombre de los módulos de Epoch contenidos en el diseño[11] y se prepara un *script* adicional para sintetizarlos adecuadamente. Posteriormente se lanza la síntesis física de los módulos de Epoch que se van a emplear y después se lanza la del diseño completo.

[10] Los nombres de estas señales se construyen añadiéndoles a las originales la partícula `_SYNBAR`.

[11] Como ya se ha indicado, corresponde con elementos que tienen en sus nombres el prefijo `hm_` y `sb_`.

CAPÍTULO 5

Espacio de diseño y resultados

5.1. Introducción

Una vez elaborados los modelos de la Unidad de Enteros de SPARC, y creado el entorno de trabajo para poder desarrollar el estudio y la realización de diversas implementaciones de ésta, se hace necesario la consecución de datos que permitan evaluar el espacio de diseño resultante. De esta forma, al disponer de referencias previas que permitan estimar el resultado final, el usuario del modelo podrá alcanzar en menor tiempo el producto que desea. Conociendo de antemano este espacio de diseño en el que se puede desplazar, un usuario podrá discurrir dentro de éste según las necesidades de sus aplicaciones. Apoyándose en estos datos, y teniendo presente el tipo de aplicaciones al que se quiere destinar la Unidad de Enteros, el usuario del modelo debe tener elegidas las características y requisitos mínimos que deben cumplirse, realizando para ello la elección que se considere más apropiada para los valores de los parámetros a los que tendrá acceso.

5.2. Parámetros del espacio de diseño

La arquitectura SPARC es una arquitectura tan abierta que permite que el diseñador de una implementación de ésta pueda tener una serie de características muy distintas de las de otras implementaciones de la misma arquitectura. No obstante, existe un conjunto mínimo de propiedades que deben cumplirse para que una arquitectura se pueda calificar como arquitectura SPARC. Los elementos que quedan fuera de este conjunto mínimo son, entre otros, los que permiten al diseñador el poder ajustar o moldear la arquitectura a sus necesidades.

Algunas de estas características son exclusivamente arquitecturales (algunas mencionadas en el apartado 4.2.4, en la pág. 49) y otras surgen de las posibilidades que permiten tanto la tecnología empleada como las herramientas.

Se puede demostrar que algunos aspectos de la arquitectura, si bien no afectan a las características de SPARC, pueden influir decisivamente en las prestaciones de la implementación final. Ejemplo de esto es el caso de algunos elementos de la arquitectura, de forma que las prestaciones de las implementaciones resultantes pueden depender de su composición y organización en zonas específicas aún siendo el comportamiento final idéntico. Esta circunstancia se ha podido constatar también en la Sección de Control, donde, en función de la estrategia elegida para el cálculo de la dirección de memoria siguiente, la ruta crítica puede o no encontrarse en esta sección.

Para muchos de estos elementos pueden incluso aparecer problemas no sólo por la forma en que se haya realizado el modelado sino por la estrategia y restricciones que se hayan establecido para su síntesis. Esta última situación suele ser más palpable en aquellos casos en los que voluntaria o accidentalmente existan elementos para los que no se hayan establecido restricciones o se hayan establecido restricciones equivocadas. En el entorno de trabajo que se ha elegido, esto puede ocurrir al eliminar la jerarquía sobre elementos que proporcionan valores a elementos sin referencia (elementos de los que la herramienta no conoce su composición interna) o que los toman de ellos. En estas ocasiones los resultados de la síntesis resultan, en la mayoría de los casos, no deseados.

Hay otros ponderables que igualmente se pueden variar y estudiar su influencia en las prestaciones. Entre éstos cabe mencionar:

- Tecnología empleada, cuyo impacto es obviamente muy alto según geometrías mínimas, niveles de metal, etc.
- Granularidad:

  Granularidad lógica Número de elementos realizados como macrobloques y estructuras tipo *bit-slice* frente a número de elementos realizados con células estándares.

  Granularidad física Posible guiado parcial en la síntesis física a través de opciones puntuales.

- Estructura de la implementación de determinados elementos de la arquitectura.
- Opciones de las herramientas de síntesis lógica y física.

Realizando variaciones sobre cada uno de estos elementos individualmente o combinándolas con variaciones sobre otros se puede estudiar la incidencia de cada uno de estos aspectos sobre las características globales y particulares del diseño.

El entorno construido permite que, para el mismo juego de instrucciones, se puedan evaluar y comparar distintas alternativas de diseño. Las alternativas arquitecturales se introducen en los modelos operando sobre el elemento de propósito específico de la arquitectura o reorganizando algunos de éstos y sus interconexiones. Tomando en

cuenta esto, se ha desarrollado un conjunto de versiones con los parámetros distintivos siguientes[1]:

1. Proporción de módulos frente a células estándares.
2. Predicción de salto:
   (a) siempre predecir que se toma,
   (b) predecir que se toma sólo cuando sea hacia atrás.
3. Comprobación de posible modificación de los códigos de condición por la instrucción anterior en los saltos.
4. Cálculo de la dirección de la búsqueda siguiente
   (a) con una única operación de suma en un ciclo,
   (b) con dos sumas. De esta forma, las dos alternativas se calculan al mismo tiempo, se elige la adecuada dependiendo de las decisiones de secuenciado y la otra se guarda en un registro. Si posteriormente se detecta que la decisión de secuenciamiento fue errónea (como en las predicciones erróneas) entonces se recupera la guardada anteriormente.
5. Mecanismo de *bypass* puesto
   (a) desde el final de la etapa de ejecución de la instrucción previa hasta la etapa de decodificación, o
   (b) desde el comienzo de la etapa de escritura en fichero de registros de la instrucción anterior hasta la etapa de ejecución.

Estas variantes se repiten para cada tecnología a considerar.

5.3. Toma de medidas

En el entorno que se ha construido la recogida de datos resulta esencial para conocer los beneficios de unas alternativas de diseño frente a otras. Cuanto más sencilla sea esta recogida de información más cómodo resulta el análisis comparativo y más útil es la utilización del entorno para obtener diseños optimizados.

Debido a la naturaleza de las herramientas en las que se fundamenta el entorno, esta toma de medidas no se ha podido automatizar, no se ha podido *instrumentar*. Este es un tema de investigación abierto. Sin embargo, enumerar los distintos pasos para realizarla puede ayudar a todo usuario de este entorno organizar sus propias investigaciones.

Estos han sido los mecanismos empleados para efectuar las medidas que se presentarán posteriormente:

[1]Estas referencias se emplearán como indicadores de las distintas alternativas arquitecturales que se indican en otros apartados, como en la tabla 5.1.

Área, ancho, alto: Dentro del entorno de Epoch, una vez abierto el diseño con

Physical Design → Open

se debe ver este diseño con la opción

Physical Design → View

Una vez se abra la ventana correspondiente, ésta contiene una caja que representa la vista en capas de mayor jerarquía del diseño. Para conocer su área se debe seleccionar con el ratón y entonces llamar al menú con

Miscellaneous → Inquire

En la ventana de información que se abre aparece, entre otros datos, el relativo al área, ancho y alto.

Potencia: Para evaluar la potencia la herramienta realiza una estimación basada en eventos a través del circuito. Esta estimación se hace necesaria para dimensionar adecuadamente las rutas de alimentación del circuito. Como la potencia en los circuitos CMOS depende fundamentalmente de la frecuencia de trabajo, se deberá introducir, previo al lanzamiento de la síntesis física, una velocidad de trabajo a la que se estima que deberá operar el circuito. Esto se realiza en la ventana principal con

Physical Design → Parameters → Power → Power Parameters...

que a su vez abrirá otra ventana. En esta ventana la información de reloj se introduce en **Default Clock Frequency** con la opción **Project-wide** elegida. En el presente trabajo se ha escogido 100 MHz para todas las versiones.

La potencia que la herramienta estima se encuentra dentro de la misma ventana donde se recoge la información sobre el área. Como esta potencia es la calculada para la frecuencia predeterminada, para la potencia relativa, entendiéndose como la relación 'potencia/velocidad', deberá dividirse esta cifra entre la frecuencia predeterminada.

Velocidad: La velocidad del circuito se evalúa empleando la herramienta de simulación Tactic. Esta herramienta realiza un estudio de todas las rutas que se producen en el circuito que tienen como punto de partida una entrada del circuito o la salida de un registro y tienen como punto de llegada una salida del circuito o la entrada de otro registro.

Desde la ventana principal de Epoch se llama al Tactic con

Simulation → Tactic...

que abrirá la ventana del Tactic. Dentro de ésta se debe, para realizar la simulación —entendiéndose en este entorno como 'simulación' a una evaluación de las transiciones que se producen en la propagación de las señales por las distintas rutas del circuito—, realizar

Setup → Build Network...

y elegir el nombre del diseño a simular. Una vez que se complete la simulación se deberá localizar la ruta más lenta dentro del circuito, que normalmente aparece entre las primeras que se presentan.

Un punto muy importante en esta localización reside en contemplar que existen rutas que, aunque se muestren en la simulación, desde el punto de vista de la lógica combinacional empleada no se pueden producir nunca. Estas son las denominadas *rutas falsas* y no deben tomarse en consideración en las medidas.

Escogiendo la ruta crítica, la velocidad de reloj se determina con la expresión

$$\text{Velocidad} = \frac{1}{\text{Retardo}_{\text{Señal de Reloj}} - \text{Tiempo}_{\text{Elemento final}}}$$

donde el $\text{Retardo}_{\text{Señal de Reloj}}$ es el retardo con el que aparece la señal de reloj de gobierno del circuito al llegar al elemento que da origen a esta ruta a través del árbol de reloj y el $\text{Tiempo}_{\text{Elemento final}}$ el instante en el que los datos llegan al último elemento de la ruta.

Transistores: Epoch no ofrece ningún medio que indique de forma directa el número de transistores de un diseño. En esta ocasión la alternativa elegida ha sido realizar un post-procesado al fichero de SPICE que describe la *netlist* del diseño. Este fichero se obtiene llamando desde la ventana principal a

Simulation → Simulation Output → SPICE...

Esta descripción en SPICE que resulta está estructurada de forma jerárquica donde el elemento más alto de esta jerarquía es el que representa al diseño entero. Para el post-procesado del fichero de SPICE resultante se ha elaborado un *script* especial que identifica esta jerarquía de la descripción y el número de transistores de cada uno de los elementos que la componen. Identificando el elemento más alto en esta jerarquía se puede conocer el número de transistores del diseño.

Area útil y área vacía: Para medir estas áreas tampoco ofrece Epoch ningún medio por el que de forma directa se pueda conocer el espacio que realmente se está empleando en elementos del circuito y el área que queda sin ocupar dentro del dado. La forma de estimar las áreas vacías sólo se puede llevar a cabo midiendo sobre los *layouts* los espacios vacíos y sumando sus áreas, y para medir las áreas útiles restar las áreas vacías a las áreas totales.

5.4. Resultados

Dentro del abanico posible de versiones que se desprende de combinar las alternativas indicadas en el apartado 5.2, se ha establecido un conjunto de variantes limitado. La primera tecnología que se ha elegido como referencia principal para las pruebas para las pruebas ha sido un proceso CMOS de 3 niveles de metal, 1 nivel de polisilicio y 0.35

μm. Normalmente, la síntesis física de este subconjunto de versiones se ha realizado de forma automática, sin particionar. Sin embargo, en ocasiones se ha realizado una partición manual a través de la agrupación de los elementos de las implementaciones para intentar mejorar las características temporales. Son estos aspectos muy importantes que en caso de llevarse a cabo de forma cuidada pueden producir mejores resultados. Además, siempre que se deseen restricciones más específicas de la síntesis física éstas se pueden acometer a través de facilidades particulares de las herramientas. Al establecer las opciones indicadas para realizar las diferentes síntesis físicas, se pueden evaluar mejor los beneficios de las decisiones microarquitecturales y de diseño.

Por otra parte, es igualmente interesante el análisis de estas variantes al coexistir con otros elementos y grandes bloques circuitales en un sistema empotrado objeto del diseño. Esta circunstancia subraya, una vez más, la conveniencia de una estrategia de generación flexible de diseños como la que hemos establecido en este trabajo.

TABLA 5.1: Índices para las versiones de la IU atendiendo a las variantes microarquitecturales

	Sin módulos	Con *pequeña* cantidad de módulos	Con *media* cantidad de módulos	Con *gran* cantidad de módulos
Opciones 2.a, 4.a, 5.a	---01	---02	---03	---04
Opciones 2.a, 3, 4.a, 5.a	---11	---12	---13	---14
Opciones 2.a, 4.b, 5.a	---21	---22	---23	---24
Opciones 2.a, 3, 4.b, 5.a	---31	---32	---33	---34
Opciones 2.a, 4.b, 5.b	---41	---42	---43	---44
Opciones 2.b, 4.b, 5.a	---51	---52	---53	---54
Opciones 2.b, 3, 4.b, 5.a	---61	---62	---63	---64

El conjunto de variantes que se ha dispuesto para los estudios del núcleo procesador aislado es el que se indica en la tabla 5.1. Todas ellas conservan el mismo ISA y el mismo interfaz, de forma que la distinción por filas entre estas versiones se debe únicamente a cuestiones particulares de sus microarquitecturas. Estas particularidades distintivas son las recogidas en el apartado anterior 5.2.

Así por ejemplo, tomando como referencia las opciones indicadas en el apartado 5.2, la opción '2.a, 4.a, 5.a' en la tabla 5.1 se refiere a una versión que predice los saltos como tomados, realiza una suma en cada ciclo y realiza el *bypass* desde el final de la etapa de ejecución.

Por '*pequeña* cantidad de módulos' se ha querido indicar que los únicos elementos

que se han construido con módulos han sido los ficheros de registros. Las versiones construidas con '*media* cantidad de módulos' también tienen la ALU, desplazador y sumadores del PC realizadas empleando módulos. Por último, las que tienen '*gran* cantidad de módulos' son aquellas donde también se han construido como módulos hasta los registros y multiplexores.

5.4.1. Resultados globales

Para evaluar las ventajas e inconvenientes de las diferentes versiones que se han generado se han realizado medidas una vez completada la síntesis física automática. A partir de esta información, organizándola adecuadamente, se pueden deducir muchas consideraciones cualitativas y cuantitativas [BN99b].

Debe indicarse de forma especial que, a pesar de la naturaleza aleatoria de algunos algoritmos empleados en los procesos en consideración, como por ejemplo el ruteado físico, los resultados no varían significativamente al repetir las experiencias. Por lo tanto cabe tomar estos resultados como estables dentro de unos márgenes acotados muy reducidos.

Por otra parte, al haberse efectuado estas experiencias de forma automatizada estos resultados deben considerarse con las reservas adecuadas. En general, en cada uno de los casos las herramientas de síntesis lógica y física permiten intervenir de forma que se pueda guiar la obtención de implementaciones con las variantes arquitecturales que corresponden. Por esto los resultados arrojados quedan sujetos a posibles oscilaciones relativamente acotadas y mejoras individuales, y por ello deben tenerse en cuenta sobre todo como indicadores de eventuales tendencias. Algunas variaciones adicionales sobre estos datos de referencia se pueden obtener gracias a:

- El empleo de otras estrategias en la síntesis física, guiadas por el diseñador.
- El dimensionamiento manual post-síntesis para acelerar determinadas rutas.

Los datos que aparecen en las siguientes tablas corresponden a diferentes series de experiencias sobre la generación flexible de núcleos SPARC para sistemas empotrados, realizables con el entorno (o *banco de trabajo*) creado para estas experiencias, y descritos en los capítulos 2 y 4. Los diferentes *layouts* obtenidos se denominan *versiones* y están identificados por un código de dos letras y tres dígitos. Estas versiones resultan de las diferentes combinaciones de 'opciones microarquitecturales', 'variantes de modelado', 'variantes de síntesis física' (colocado y ruteado) y 'variantes externas' (con otros bloques circuitales).

En este apartado presentamos de forma resumida todo el conjunto de experiencias y sus resultados. Las restantes secciones del capítulo analizan en detalle cada una de las experiencias y las variantes resultantes para extraer conclusiones y guías de utilidad para el diseño de sistemas empotrados basados en núcleos SPARC. (La metodología es completamente generalizable a otros núcleos y sistemas).

Los *layouts* más significativos se dan en el apéndice A. El objeto de este apéndice es no sólo dar los *layouts* completos para observación o medidas adicionales, sino dar su forma y relación de aspecto, parámetros importantes para la utilidad de los diseños generados, y de difícil o imposible representación mediante tablas.

Opciones microarquitecturales

La primera serie de experimentos ha sido estudiar el efecto de diferentes opciones microarquitecturales sobre las prestaciones del núcleo. A estas alternativas se las ha llamado 'opciones', están identificadas por cada una de las filas de la tabla 5.1 y tienen como identificador el segundo dígito del código. Hay, por lo tanto, las siguientes siete opciones:

Opción 0: Código ---0-

- Predice los saltos como tomados.
- Para el cálculo de la siguiente dirección realiza una suma en cada ciclo.
- Realiza el *bypass* desde el final de la etapa de ejecución.

Opción 1: Código ---1-

- Predice los saltos como tomados.
- Comprueba la posible modificación de los códigos de condición por la instrucción anterior en los saltos.
- Para el cálculo de la siguiente dirección realiza una suma en cada ciclo.
- Realiza el *bypass* desde el final de la etapa de ejecución.

Opción 2: Código ---2-

- Predice los saltos como tomados.
- Para el cálculo de la siguiente dirección realiza dos sumas en cada ciclo.
- Realiza el *bypass* desde el final de la etapa de ejecución.

Opción 3: Código ---3-

- Predice los saltos como tomados.
- Comprueba la posible modificación de los códigos de condición por la instrucción anterior en los saltos.
- Para el cálculo de la siguiente dirección realiza dos sumas en cada ciclo.
- Realiza el *bypass* desde el final de la etapa de ejecución.

Opción 4: Código ---4-

- Predice los saltos como tomados.
- Para el cálculo de la siguiente dirección realiza dos sumas en cada ciclo.

- Realiza el *bypass* desde el comienzo de la etapa de escritura en fichero de registros.

Opción 5: Código ---5-

- Predice los saltos como tomados sólo cuando son hacia atrás.
- Para el cálculo de la siguiente dirección realiza dos sumas en cada ciclo.
- Realiza el *bypass* desde el final de la etapa de ejecución.

Opción 6: Código ---6-

- Predice los saltos como tomados sólo cuando son hacia atrás.
- Comprueba la posible modificación de los códigos de condición por la instrucción anterior en los saltos.
- Para el cálculo de la siguiente dirección realiza dos sumas en cada ciclo.
- Realiza el *bypass* desde el final de la etapa de ejecución.

La tabla 5.2 muestra, por ejemplo, las medidas de algunas versiones correspondientes a variar las opciones manteniendo fijas las restantes variables experimentales de diseño. La comparación por filas muestra la incidencia de las opciones microarquitecturales en el diseño.

Variantes de granularidad lógica y física

La segunda serie de experimentos ha consistido en variar —para cada opción microarquitectural— los parámetros de modelado, síntesis y diseño físico que afectan a la granularidad del núcleo.

Se entiende por granularidad del núcleo la relación entre lógica agrupada en módulos de diverso tamaño ('grano') y lógica suelta, normalmente mapeada en células estándares (la granularidad más fina disponible). Hablamos así de versiones de grano grueso (más modulares) y de grano fino.

Cuando adicionalmente se establecen agrupaciones jerárquicas de módulos en el diseño físico, nos referimos a ellas como mayor granularidad física.

Para facilitar el estudio de las versiones generadas con las variantes de granularidad se han agrupado los parámetros de granularidad de la siguiente forma:

Granularidad lógica:

Grano fino: Código ----1

Grano pequeño: Código ----2

Grano medio: Código ----3

Grano grande: Código ----4

Granularidad física:

Propiedad `fixedblock` a 0, sin grupos: Código a----

Propiedad `fixedblock` a 0, con grupos predefinidos: Código b----

Propiedad `fixedblock` a 1, sin grupos: Código c----

El efecto de la granularidad puede verse comparando entre sí en las tablas 5.2 a 5.10.

Variantes del fichero de registros

La tercera serie de experimentos ha consistido en variar el número de ventanas. El número de verntanas en el fichero de registros depende de las aplicaciones. La arquitectura especifica un mínimo de 2 ventanas. En el estudio de generación flexible de núcleos SPARC realizado, éste es un parámetro que afecta fundamentalmente al área y a la forma o relación de aspecto del núcleo. El área se incluye en las tablas. Como se ha indicado antes, la relación de aspecto debe observarse directamente en los *layouts* de los núcleos (apéndice A).

Para facilitar y simplificar el estudio de este parámetro de diseño, cuyo comportamiento es muy lineal, se dan solamente las versiones correspondientes a

Siete ventanas de registros: Código --7--

Cuatro ventanas de registros: Código --4--

Dos ventanas de registros: Código --2--

Su efecto puede observarse, por ejemplo, comparando las tablas 5.2, 5.3, 5.4 (con 7 ventanas) y 5.11 (con 4 y 2 ventanas).

Variantes con test

La cuarta serie de experimentos ha estudiado el impacto de la introducción de la circuitería de test y de interfaz del circuito.

Existen diversas técnicas de test aplicadas a la generación de núcleos IP, y de núcleos procesadores más específicamente. Los núcleos IP pueden incluirse en diseño de sistemas como núcleos '*software*', es decir, como modelos VHDL sintetizables por herramientas CAD en diversas tecnologías, cuyos resultados variarán tanto por las herramientas como por la tecnología empleadas. La inserción de la circuitería de test debe hacerse una vez establecido el entorno de síntesis, las propiedades del núcleo, y la tecnología objetivo de la síntesis. La técnica de test más utilizada es la de *scan-path*. Esta técnica puede continuarse con *boundary-scan* para el test completo del dado, con los diversos núcleos y bloques IP que hayan sido integrados.

Por otro lado, los núcleos IP pueden incluirse en el diseño de sistemas como núcleos '*hardware*', es decir, como *layouts* concretos sobre una tecnología definida. Estos núcleos pueden estar, en este caso, completamente pre-caracterizados, y pueden o no incluir cadenas *scan-path* de test.

Por último, la definición del nivel de test del núcleo depende de la definición del interfaz de cada núcleo y bloque integrado. Este interfaz es directo e inmediato para la integración de bloques periféricos elementales como RAM, cachés, unidades de coma flotante (FPU), coprocesadores débilmente acoplados y unidades periféricas estándares como temporizadores y puertos de comunicaciones series o paralelos, entre otros. No lo es para bloques circuitales fuertemente acoplados como las extensiones multimedia, las de procesamiento vectorial, extensiones DSP o de procesamiento de flujos de datos (*streams*).

La definición de interfaces para núcleos IP es un problema abierto. Existen borradores de propuestas para esos estándares, siendo las más importantes las propuestas de la VSIA (*Virtual Socket Interface Alliance*) [VSI, Bak98, See99, VSI97b].

En este trabajo hemos optado por observar el efecto de la inserción de la circuitería de test a nivel de núcleo aislado. Esta opción es consistente con el ámbito del estudio y es en este nivel donde puede ser más significativa.

El efecto de la inserción de *scan-path* en los núcleos ha resultado no ser sólo lineal sino prácticamente constante, por lo que en la tabla 5.12 sólo damos dos ejemplos para los núcleos `aa753` y `aa763`, que con el test incluido constituyen las versiones `at753` y `at763`.

Variantes con bloques de memoria

La quinta serie de experimentos ha incluido la integración del primer bloque externo que acompaña habitualmente a un núcleo procesador en la aplicación: el bloque de memoria.

Los bloques de memoria varían enormemente según las aplicaciones. Dado que el núcleo IP de SPARC desarrollado sólo ocupa una *huella* de entre 7 y 10 mm^2 en tecnología de 0.35 μm, y teniendo esta tecnología capacidad para producción en volumen de dados hasta de 100 mm^2, resultan muchas opciones para integración *System on a Chip* (*SOC*) de bloques de memoria en el chip.

Estas opciones incluyen ROM de programas, RAM de programas y datos, cachés de instrucciones y/o datos, RAMs de vídeo, *buffers* de datos, etc. La influencia del tamaño de la memoria es fácilmente observable en el circuito SOC y sólo damos resultados para tamaños de 4 Kbytes y 8 Kbytes. Tiene mayor interés experimental ver el efecto sobre el núcleo de dos parámetros:

Conexión manual o conexión automática: Códigos `-l---` y `-m---` para conexión manual, y `-n---` y `-o---` para conexión autmática. Para emular el primer efecto se han dejado los bloques sin conectar por la herramienta, dejando suficiente espacio en el canal de ruteado. El segundo efecto se realiza con la conexión automática generada por la herramienta. Estos efectos pueden observarse en las versiones `al753` y `am753`, y `an753` y `ao753` de la tabla 5.13.

Núcleo fijo o núcleo plástico: Hemos introducido el concepto de plasticidad para observar el comportamiento del núcleo cuando se hace una integración conjunta de los

bloques, permitiendo a la etapa de generación flexible del núcleo adaptarse al bloque de memoria acompañante que se le ha incluido. El grado de plasticidad puede controlarse en cierta medida. Denominamos plasticidad libre a la que resulta de no fijar más restricciones que las derivadas de la interconexión funcional de los dos bloques.

El efecto de la plasticidad puede verse contrastando el comportamiento del núcleo en las variantes `al753` y `am753`, y `an753` y `ao753` de la misma tabla 5.13, donde al plasticidad es libre, frente al comportamiento del núcleo fijo correspondiente `aa753`, cuyas propiedades se indicaron en la tabla 5.3. Como prueba de esto se ha realizado la versión `ap753`, con una memoria de 8 Kbytes.

Variantes con Unidad de Coma Flotante

La arquitectura SPARC v. 8 especifica también una Unidad de Coma Flotante (FPU) opcional. La especificación corresponde a la del estándar IEEE 754-1985.

En otro trabajo del grupo se ha desarrollado un diseño *bottom-up full-custom* de una FPU estándar optimizada para alta velocidad [dAlo]. Para el presente trabajo hemos creado una versión *top-down* desde VHDL de esta FPU.

En aplicaciones multimedia de procesado de vídeo no son necesarias las unidades de coma flotante. Sin embargo, las aplicaciones de gráficos por computador, tratamiento de sólidos en 3D, *rendering* y tratamiento sofisticado de imágenes, pueden llegar a ser intensivas en operaciones en coma flotante.

Aunque no se trata del campo de aplicación más directo de los núcleos SPARC construidos en este trabajo, sí resulta interesante estudiar su extensión hacia aplicaciones más exigentes en computación, como las indicadas.

La sexta serie de experimentos ha consistido por tanto en la integración de los núcleos de enteros con un núcleo FPU de SPARC.

El fenómeno de la conexión automática/manual (desconexión) ha sido ya estudiado con un bloque central débilmente acoplado como es una RAM. Por ello en estas experiencias disponemos la FPU con conexión automática y se analizan las variantes siguientes:

- colocado libre,
- colocado guiado,
- plasticidad libre,
- plasticidad fija.

Los resultados se dan en la tabla 5.14.

Variantes con reducción de instrucciones y/o tipos de datos

La generación de núcleos con un subconjunto de instrucciones de la especificación arquitectural tiene poca incidencia en las prestaciones de la máquina, puesto que afectan sobre todo a la Unidad de Control, cuya estructura en una arquitectura RISC es extremadamente simple y se mapea en lógica cableada. Solamente en aplicaciones extremas de pequeños microcontroladores empotrados la eliminación de importantes instrucciones puede reducir o eliminar algunos recursos de la ruta de datos. Se ha visto también cómo las instrucciones especiales TADDccTV y TSUBccTV (*tagged arithmetic* o aritmética marcada) en muchos programas no se usan. Sin embargo, es importante mantener la compatibilidad con la arquitectura especialmente para ejecutar código generado por compiladores estándares de lenguajes como C, correr rutinas de librerías, *drivers* y núcleos de sistemas operativos, etc. Por esta razón no nos detenemos en estudiar el efecto de recortes en el juego de instrucciones.

Tampoco hemos enfocado este estudio hacia pequeños microcontroladores empotrados, para los que generar versiones recortadas a medias palabras y/o bytes puede en ocasiones tener interés. En estos casos tampoco tiene sentido incorporar muchos de los mecanismos de aceleración que hemos implementado. Con tecnología de 0.35 μm éstos pequeños microcontroladores no acelerados pueden bajar su frecuencia de reloj desde los 100 MHz de este trabajo hasta un rango entre 30 y 50 MHz. Pueden también generarse versiones muy pequeñas no segmentadas con relojes inferiores a esas frecuencias, y de bajo consumo. Estas experiencias están fuera del ámbito de este trabajo.

Variantes con extensiones VIS multimedia

El campo de aplicaciones multimedia definido como marco de este trabajo conduce a realizar experimentaciones en dirección opuesta a la anterior: con extensiones del juego de instrucciones.

Para soportar estas extensiones es necesario modificar la Ruta de Datos y la Unidad de Control. También existen opciones sobre los ficheros de registros para los tipos de datos de interés en procesado digital de medios de comunicación, en especial vídeo digital comprimido.

Para llevar a cabo esta experimentación se ha desarrollado una unidad fuertemente acoplada en la ruta de datos interna del procesador, con organización SIMD de tipo vectorial corto (4 elementos de proceso), según la definición del *Visual Instruction Set* (VIS) de SPARC. La alimentación de la Unidad VIS puede hacerse desde datos en el fichero de registros de la IU o desde el fichero de registros de la FPU. Las ventajas e inconvenientes de una u otra opción es un tema abierto a la discusión científica, y existen implementaciones arquitecturales con unas u otras opciones.

Por otro lado es necesario fijar variantes y acotar el problema de las opciones de implementación.

Para adaptarnos mejor el objetivo de crear sistemas de bajo coste para multimedia, la opción 'fichero de enteros' parece la más idónea.

Por ello hemos optado por estudiar una variante con una unidad VIS incorporada, trabajando con el fichero de registros de la Unidad de Enteros.

En la tabla 5.15 se presentan los resultados obtenidos y en el apéndice A sus *layouts*.

Variantes para soporte DSP

Es conocida la sentencia de los arquitectos de que "el mejor DSP es un RISC". Esto es cierto siempre que se dote al RISC de una unidad MAC de multiplicación (y acumulación) y se obvien otros factores como la relación coste/prestaciones.

Inicialmente muchos RISCs, entre ellos SPARC v. 7 y algunas implementaciones de SPARC v. 8, no soportaban la multiplicación *hardware*, sino que o bien utilizaban un coprocesador, o bien en todo caso proporcionaban una instrucción de *paso de multiplicación*, paso central del algoritmo de la multiplicación mediante sumas y desplazamientos. Un producto de $32 \times N$ bits se obtiene así en $N + 4$ ciclos, suficiente para muchas aplicaciones. No obstante, en los últimos años se va produciendo una tendencia creciente a utilizar pequeños procesadores RISC para el tratamiento de señales (por ejemplo, MX21 de Hitachi), y para ello se ha ido dotando a los procesadores de extensiones MAC fuertemente acopladas en la ruta de datos.

Una unidad MAC está de hecho insertada en la unidad de soporte a VIS en la Ruta de Datos junto con 4 sumadores. El interés de una estructura MAC aislada está motivado por aplicaciones de procesado de voz, no de vídeo. Su impacto es notablemente inferior al de la extensión VIS multimedia estudiado y no entramos por ello a ampliar más este trabajo en esa dirección.

Variantes tecnológicas

Finalmente, como es obvio, el cambio de tecnología objetivo para el mapeado tiene una incidencia muy importante sobre las prestaciones de los núcleos.

Justamente una de las ventajas de la creación de un entorno flexible de generación de núcleos IPs, en este caso procesadores SPARC, como el establecido en este trabajo, es la facilidad de generar versiones en distintas tecnologías, de forma que la elección de éstas sean un factor igualmente decisivo en el desarrollo del producto.

De hecho, la filosofía de la creación de Propiedad Intelectual, IP, descansa en una cierta separación entre 'casas de sistemas', 'casas de diseño' y 'casas de semiconductores'. El esquema funciona también en un modelo de negocio completamente integrado, en cuyo caso las opciones tecnológicas suelen ser menores.

Si el diseño es lo suficientemente pequeño, las opciones tecnológicas comienzan con tecnologías FPGA. En el caso de los núcleos SPARC —y mucho menos con unidades FPU, CP, VIS o memorias—, un mapeado a tecnología FPGA no es posible salvo realizando drásticas particiones con pérdida sustancial de prestaciones. En el caso de SPARC no es posible hoy realizar un SOC con núcleo en tecnología FPGA. Quizás lo sea en unos años, para versiones reducidas de la arquitectura.

Por esta razón se ha decidido realizar experimentaciones generando núcleos para dos tecnologías predominantes hoy y distintas:

- un proceso CMOS de 0.35 μm de Duet Tecnologies
- un proceso CMOS de 0.50 μm de Hewlett Packard

Las experiencias pueden extenderse a tantas cuantas tecnologías estén disponibles o sean accesibles para un laboratorio. Es necesaria una etapa previa para configurar las herramientas adecuadamente.

Las tablas 5.2 a la 5.15 presentan los resultados obtenidos para la primera tecnología y en la tabla 5.16 se indican los resultado obtenidos para algunas variantes en la segunda tecnología. El apéndice A presenta los *layouts*.

Analizamos los resultados de todas estas experiencias en los apartados que siguen.

TABLA 5.2: 'Pequeña' cantidad de módulos, con propiedad `fixedblock` a 0

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frecuencia (MHz)	Transistores
aa702	7.9841	9.3027	59.97	183109
aa712	18.5920	10.2928	58.09	187031
aa722	11.5999	10.0048	64.16	185580
aa732	15.3036	10.1366	63.78	186515
aa742	12.4070	9.4874	66.87	183181
aa752	11.9607	9.9760	69.97	185273
aa762	13.3563	10.0371	72.64	185416

TABLA 5.3: 'Media' cantidad de módulos, con propiedad `fixedblock` a 0

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frecuencia (MHz)	Transistores
aa703	9.0754	9.5921	60.00	171217
aa713	9.5424	8.8484	55.60	169475
aa723	13.4839	9.6422	80.95	172032
aa733	9.1589	9.5640	89.90	173087
aa743	10.6412	9.3457	74.36	169630
aa753	9.6249	9.2007	93.60	172463
aa763	13.1945	9.9285	84.75	173774

TABLA 5.4: 'Gran' cantidad de módulos, con propiedad `fixedblock` a 0

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frecuencia (MHz)	Transistores
aa704	14.7899	10.1546	54.60	172235
aa714	9.7460	9.8144	53.87	171983
aa724	13.2781	10.5004	78.19	176022
aa734	13.0209	9.9894	87.37	174922
aa744	16.0621	10.3943	79.57	174973
aa754	14.9878	10.4817	79.40	176069
aa764	12.7523	10.0386	80.06	174623

TABLA 5.5: 'Pequeña' cantidad de módulos, con propiedad `fixedblock` a 0 y grupos predefinidos

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frecuencia (MHz)	Transistores
ba702	18.1874	10.0395	52.06	185241
ba712	11.4213	10.3393	67.99	186390
ba722	20.5993	10.6823	53.19	186423
ba732	19.4922	10.6628	67.37	187361
ba742	14.0557	9.7527	67.97	183399
ba752	17.7335	10.3171	57.08	186583
ba762	11.9197	10.1932	61.20	185858

TABLA 5.6: 'Media' cantidad de módulos, con propiedad `fixedblock` a 0 y grupos predefinidos

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frecuencia (MHz)	Transistores
ba703	11.0778	10.0684	63.24	171967
ba713	13.2981	9.8940	60.44	171718
ba723	13.4927	9.6806	82.73	171687
ba733	15.7572	10.3264	79.33	173457
ba743	15.1069	9.6138	74.04	170669
ba753	16.9514	10.4733	82.24	173849
ba763	17.3635	10.5522	76.35	175971

TABLA 5.7: 'Gran' cantidad de módulos, con propiedad `fixedblock` a 0, y grupos predefinidos

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frecuencia (MHz)	Transistores
ba704	14.4963	10.3988	58.25	172163
ba714	17.6963	11.2893	55.13	175897
ba724	18.0889	11.0738	82.05	177124
ba734	16.6855	10.4234	78.80	175343
ba744	22.5606	11.1982	72.73	176027
ba754	11.1218	10.6844	86.30	176208
ba764	14.8597	10.3715	82.83	175067

TABLA 5.8: 'Pequeña' cantidad de módulos, con propiedad `fixedblock` a 1

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frecuencia (MHz)	Transistores
ca702	15.2719	10.2193	53.38	184602
ca712	18.7347	10.6808	66.27	187564
ca722	15.3189	10.0738	58.28	186088
ca732	13.5150	10.6828	59.36	187884
ca742	15.6237	10.5250	61.70	184035
ca752	14.8836	10.5547	67.19	187580
ca762	13.2435	9.9822	70.93	185909

TABLA 5.9: 'Media' cantidad de módulos, con propiedad `fixedblock` a 1

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frecuencia (MHz)	Transistores
ca703	16.9508	10.3561	54.95	173583
ca713	11.4791	9.4246	58.67	169896
ca723	12.0548	9.3751	89.18	171634
ca733	7.3983	9.1771	90.63	172104
ca743	10.3395	9.5658	75.65	169724
ca753	7.8112	9.1495	90.44	172228
ca763	12.6047	9.6319	82.77	173816

TABLA 5.10: 'Gran' cantidad de módulos, con propiedad `fixedblock` a 1

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frecuencia (MHz)	Transistores
ca704	10.4802	10.1089	52.50	174159
ca714	12.7277	9.3257	54.50	172531
ca724	14.3234	10.0912	83.54	175168
ca734	14.4568	10.2981	80.59	177216
ca744	14.0417	9.9618	74.32	174731
ca754	16.4746	9.8874	82.73	175758
ca764	13.2934	9.2521	79.99	175273

TABLA 5.11: Con número de ventanas variable

Ventanas	Esquema de de Predicción	Proporción de Módulos	Ref.	Área (mm^2)	Potencia (mW/MHz)	Frec. (MHz)	Transistores
2	Simple — con opciones 2.a, 4.b, 5.a	Baja	aa222	7.7270	4.87396	61.46	122780
		Media	aa223	7.1537	4.66823	86.56	109650
		Alta	aa224	11.2015	5.66391	85.16	112879
	Mejorada — con opciones 2.b, 4.b, 5.a	Baja	aa252	8.0454	4.85731	61.17	122699
		Media	aa253	7.4045	4.85172	88.91	109763
		Alta	aa254	11.3917	5.62138	78.70	112611
4	Simple	Baja	aa422	8.8574	6.48212	59.63	148278
		Media	aa423	11.4606	6.89787	88.90	136056
		Alta	aa424	9.5769	6.80150	85.73	137870
	Mejorada	Baja	aa452	11.0749	6.89198	61.14	148787
		Media	aa453	10.2297	6.76384	79.11	136042
		Alta	aa454	9.7595	6.91100	82.79	137776

TABLA 5.12: Con circuitería de test

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frec. (MHz)	Transistores
at733	11.6318	9.8591	83.75	178464
at753	14.3062	9.5708	86.93	178307

TABLA 5.13: Con memorias adicionales

Elemento Adicional	Ref.	Área (mm^2)	Potencia (mW/MHz)	Frec. (MHz)	Transistores
4Kbytes RAM sin conectar, y núcleo con plasticidad libre	al753	15.2334	11.5594	78.00	173002
8Kbytes RAM sin conectar, y núcleo con plasticidad libre	am753	17.9943	13.1562	83.13	172974
4Kbytes RAM conectada, y núcleo con plasticidad libre	an753	13.1778	11.4568	88.89	173024
8Kbytes RAM conectada, y núcleo con plasticidad libre	ao753	19.7457	15.0805	87.90	172955
8Kbytes RAM conectada, y núcleo sin plasticidad libre	ap753	16.5176	9.2007	93.60	172463

TABLA 5.14: Área de la IU y FPU juntas

Elemento Adicional	Ref.	Área (mm^2)
FPU con colocado libre y núcleo con plasticidad fija	ae753	15.0900
FPU con colocado guiado y núcleo con plasticidad fija	af753	13.3216
FPU con colocado libre y núcleo con plasticidad libre	ag753	13.5155

TABLA 5.15: Con módulo VIS

Elemento Adicional	Ref.	Área (mm^2)	Potencia (mW/MHz)	Frec. (MHz)	Transistores
Módulo para aplicaciones multimedia (VIS) – Versión lenta	av753	12.3799	11.0993	50.04	204561
Módulo para aplicaciones multimedia (VIS) – Versión rápida	aw753	15.5525	12.1165	79.61	217349

TABLA 5.16: Con la tecnología HP14B (CMOS de 0.5 μm)

Ref.	Área (mm^2)	Potencia (mW/MHz)	Frec. (MHz)	Transistores
ha723	38.1031	13.5219	63.38	178106
ha763	29.7610	12.6408	61.97	178188
aa723[1]	13.4839	9.6422	80.95	172032
aa763[1]	13.1945	9.9285	84.75	173774

[1]Incluidos para facilitar la comparación en la influencia del cambio de tecnología.

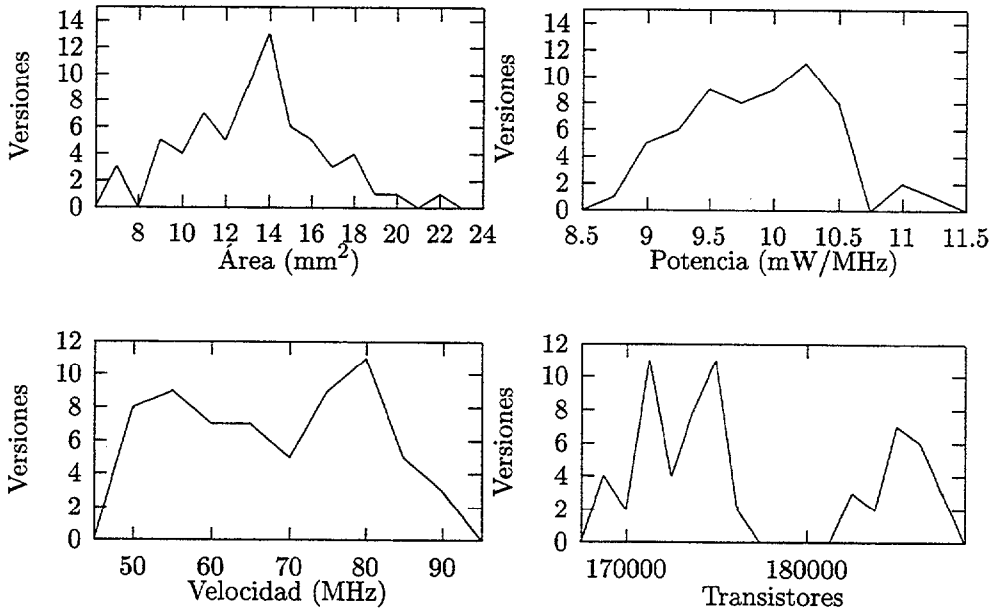
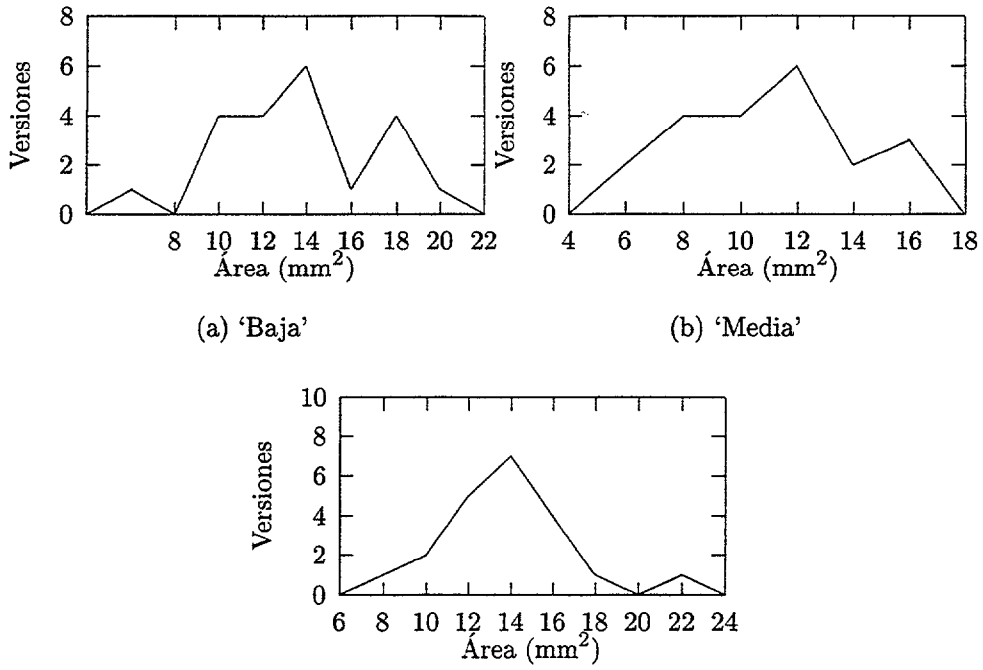


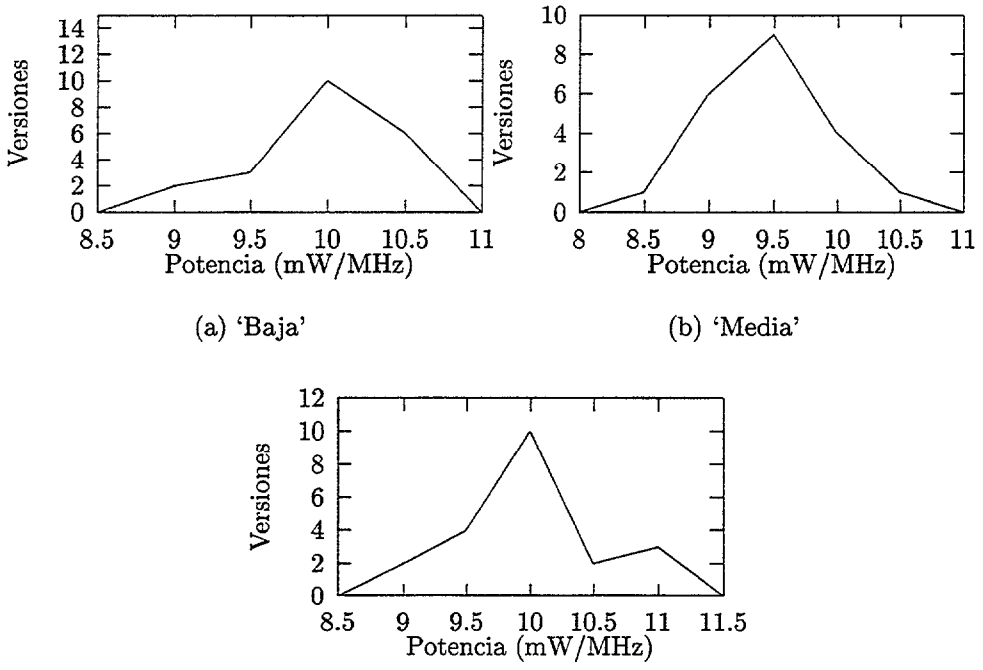
FIGURA 5.1: Frecuencia de las propiedades para versiones con 7 ventanas



(a) 'Baja' (b) 'Media'

(c) 'Grande'

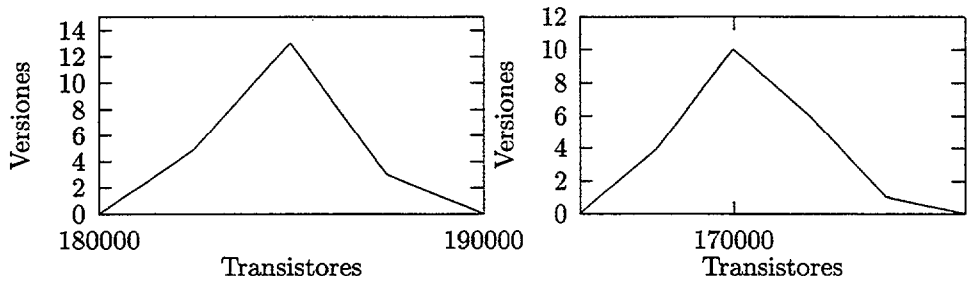
FIGURA 5.2: Frecuencia del área para versiones con 7 ventanas y distintas cantidades de módulos



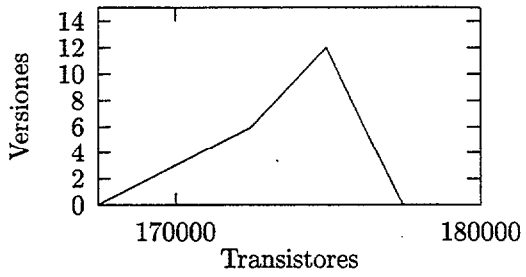
(a) 'Baja' (b) 'Media'

(c) 'Grande'

FIGURA 5.3: Frecuencia de la potencia para versiones con 7 ventanas y distintas cantidades de módulos

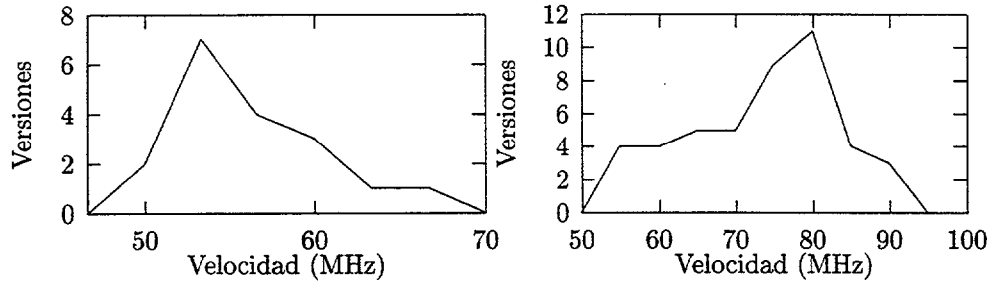


(a) 'Baja' (b) 'Media'



(c) 'Grande'

FIGURA 5.4: Frecuencia del número de transistores para versiones con 7 ventanas y distintas cantidades de módulos



(a) Con opción 4.a (b) Con opción 4.b

FIGURA 5.5: Frecuencia de la velocidad para versiones con 7 ventanas

5.4.2. Influencia del empleo de módulos (granularidad lógica)

Los módulos son elementos cuyo diseño y prestaciones suelen estar cuidadosamente estudiados para proporcionar los mejores compromisos en área y velocidad. Su empleo en zonas muy concretas suele, por tanto, producir mejoras en estas propiedades. Sin embargo, emplear de forma excesiva estos módulos precisa un control del diseño más específico por parte del diseñador. Como se puede observar, las versiones con 'gran' cantidad de módulos no suelen funcionar a frecuencias mayores que las de 'media', ni consumen menos potencia, ni son más densas. Por el contrario, son las versiones con

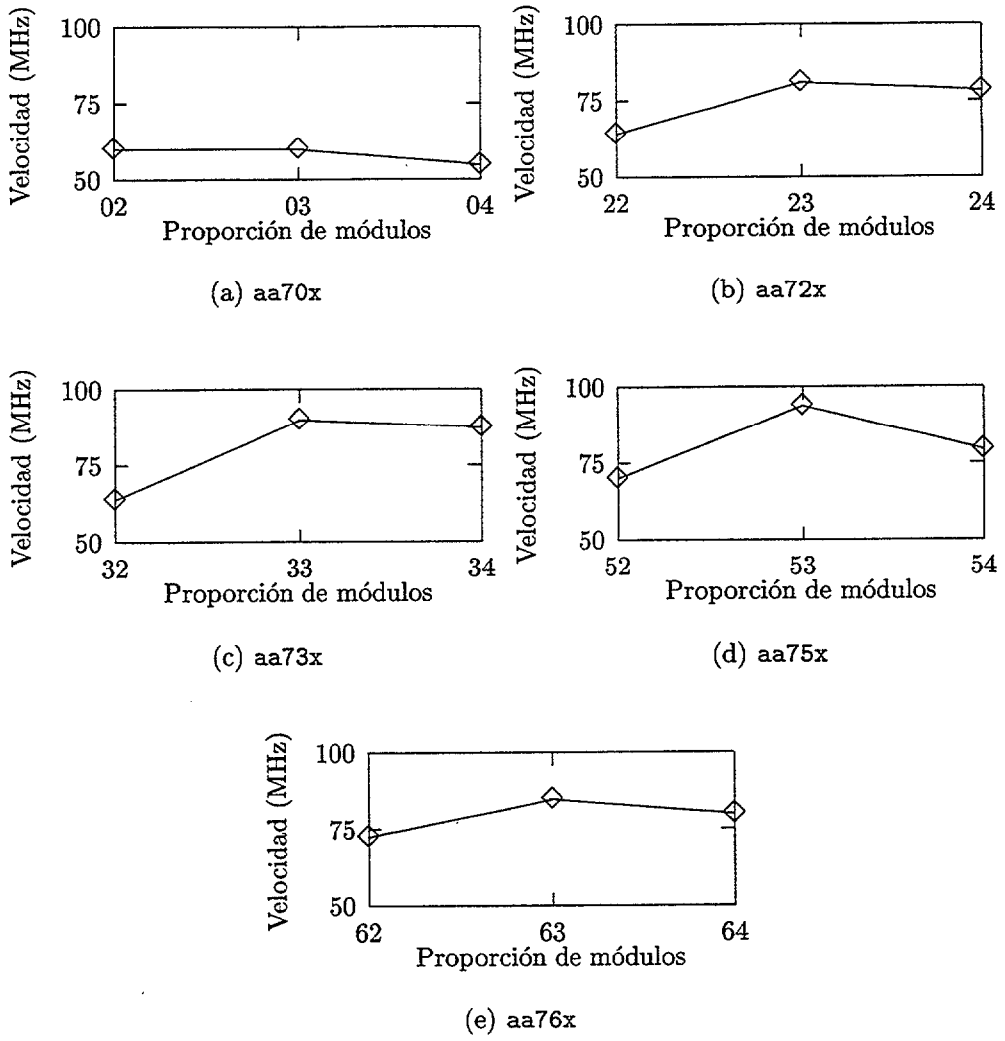


FIGURA 5.6: Influencia en la frecuencia del empleo de módulos en las series aa70x, aa72x, aa73x, aa75x y aa76x

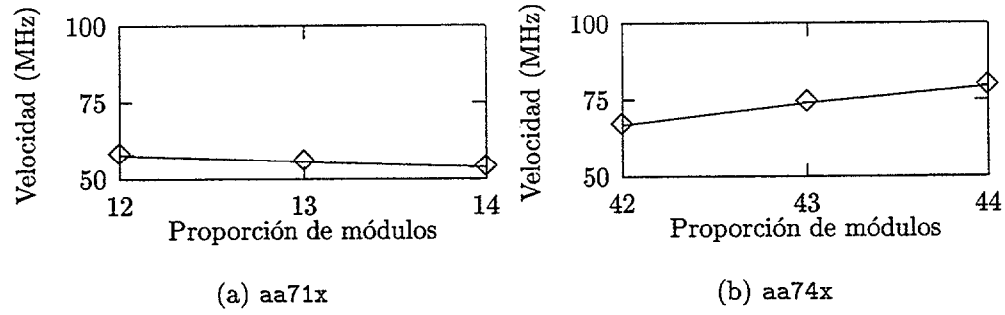


FIGURA 5.7: Influencia en la frecuencia del empleo de módulos en las series aa71x y aa74x

una proporción 'media' (cabría decir 'adecuada' o 'razonable') las que aportan por lo general el *pico* de prestaciones en velocidad, consumo y área (ver figs. 5.6 a 5.12).

Sin embargo, existen situaciones donde esto no se ha producido. Por ejemplo, en las series aa71x y aa74x (fig. 5.7) las versiones con 'gran' cantidad de módulos son las más

rápidas.

Esta circunstancia también se repite ocasionalmente, de tal forma que eventualmente cuanto mayor es el número de elementos que se insertan, el colocado que hace la herramienta de forma automática puede resultar más óptima en lo que a velocidad se refiere.

Por lo que respecta a la potencia relativa[2], las propiedades evolucionan prácticamente de la misma forma en todas las circunstancias. Sólo se ha detectado un caso en el que esta evolución no se cumple (fig. 5.9).

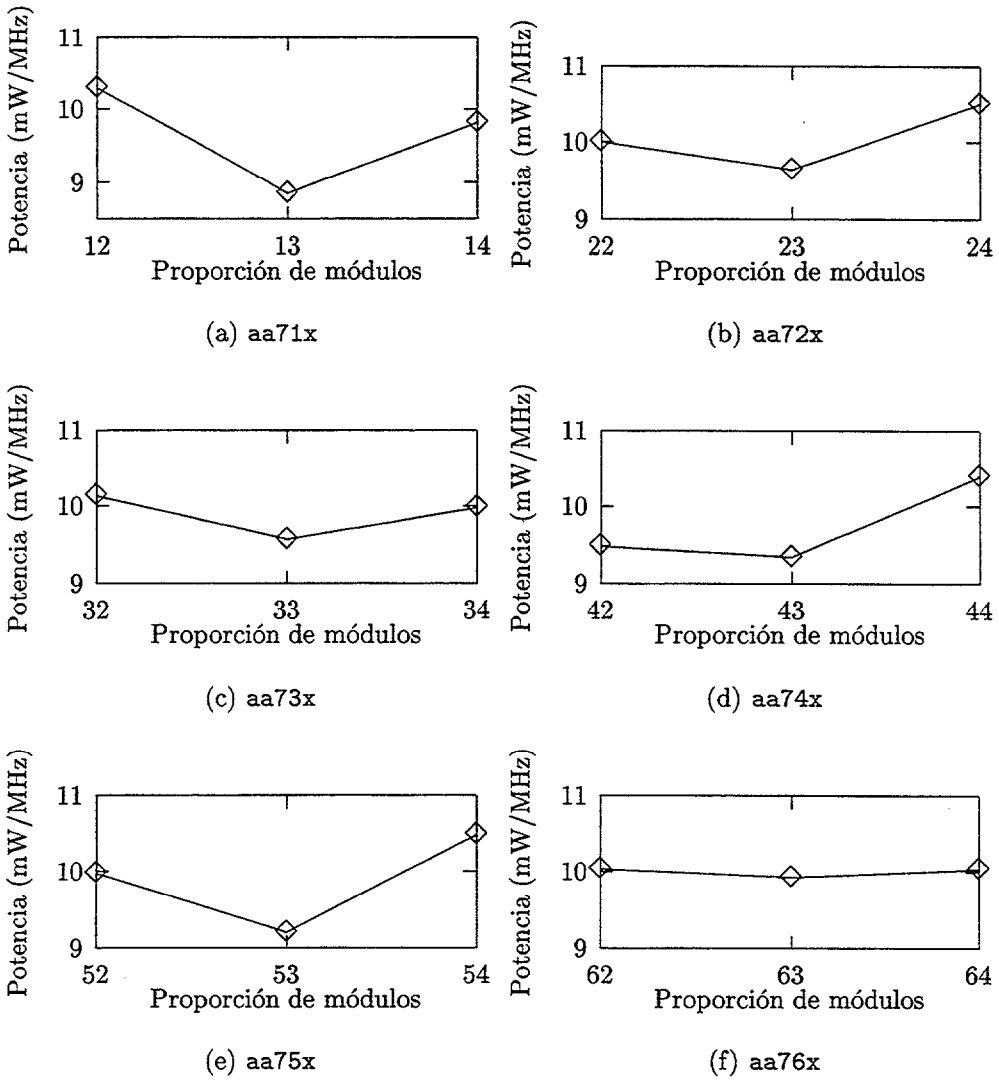


FIGURA 5.8: Influencia en la potencia relativa del empleo de módulos en las series aa71x, aa72x, aa73x, aa74x, aa75x y aa76x

Un primer dato a considerar es la significativa variación de las prestaciones de velocidad, área y consumo, así como de forma y área global, que aparece entre las versiones. Ver figs. 5.1 a 5.5. Esta variación subraya más aún la conveniencia de disponer de un entorno flexible de generación experimental de núcleos IP como el creado en este trabajo.

Como se puede ver, en aquellas versiones tanto con el menor número de módulos

[2]Considérese como *potencia relativa* la relación entre la potencia y la frecuencia.

como con el mayor la potencia relativa suele ser grande. Esta misma circunstancia se produce para las velocidades de las versiones. Por esta razón cabe en principio pensar que esto se debe a que los circuitos con una proporción de módulos intermedia están mejor optimizados en lo que respecta a la lógica empleada y por la menor longitud del interconexionado. En el caso de las versiones con proporción 'pequeña' de módulos, puede considerarse que esto se debe a que algunas porciones de las versiones podrían quedar notablemente mejoradas al sustituirse por módulos optimizados. En el caso del uso excesivo de módulos en las versiones con proporción 'grande', al no existir un acoplamiento depurado en el intercambio de información entre las herramientas, las herramientas de síntesis lógica suelen realizar una sobre-estimación en la capacidad de carga necesaria para llevar las señales hasta los módulos externos generados con las herramientas de síntesis física.

Nótese que todas las medidas son *post-layout*, es decir, *correctas* al nivel de precisión de las herramientas físicas de extracción de parámetros y caracterización. No estamos diciendo, por tanto, que las medidas no se correspondan con la realidad, sino que las estimaciones que hacen las herramientas lógicas pueden haber conducido a un sobredimensionamiento de algunas puertas en la síntesis lógica, quizás no es necesario en un diseño manual. Por otro lado, prescindir manualmente de este sobredimensionamiento puede acarrear serios riesgos de funcionalidad.

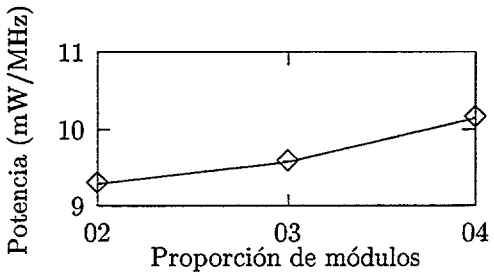


FIGURA 5.9: Influencia en la potencia del empleo de módulos en la serie `aa70x`

En lo que concierne al número de transistores, se puede observar que las correspondencias entre versiones evolucionan de forma muy similar. Por regla general, en la transición desde variantes donde sólo aparecen los ficheros de registros como módulos a aquellas donde además se han construido de esta forma los sumadores y el desplazador, se puede observar cómo el número de transistores desciende apreciablemente. Sin embargo, en la transición desde estas últimas variantes a otras donde también se han construido con esta alternativa registros y multiplexores, el número de transistores apenas cambia, si bien existe un ligero aumento. Según se ha podido apreciar, esto parece apoyar la tesis explicativa que se ha propuesto al observar las propiedades anteriormente comentadas.

Por último, el área es una magnitud que, cuando se realiza la ubicación y ruteado de forma automática, arroja resultados que pueden ser muy variables. Cuando se controlan estos aspectos como en la integración de varios bloques, los resultados divergen menos. Como puede observarse, por regla general parece existir una correspondencia entre el área y el número de transistores. Sin embargo, en ocasiones la mayor desproporción entre el número de elementos realizados con módulos y los construidos con células estándares hace que los resultados de la colocación automática sean peores. Esto último suele dejar

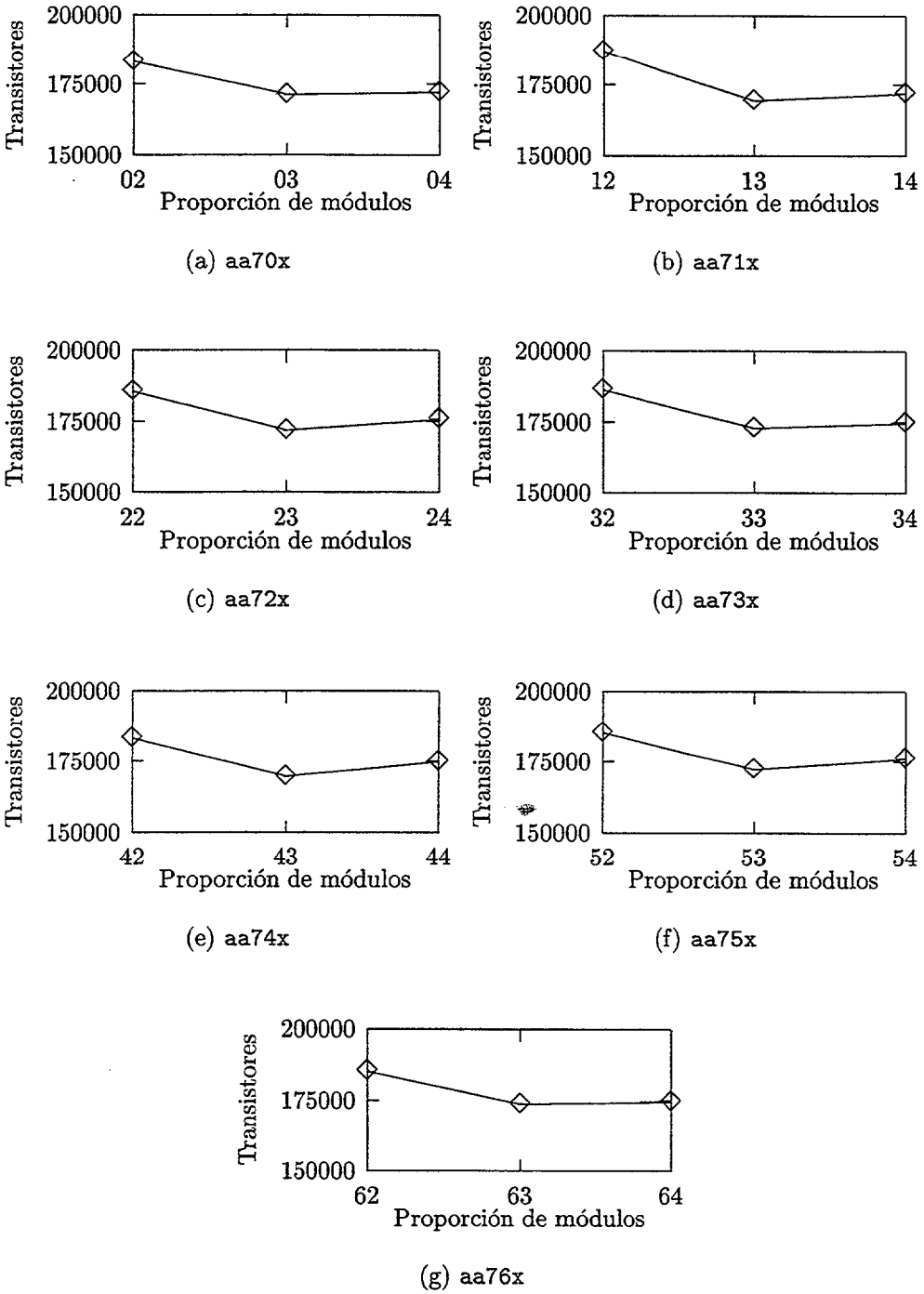


FIGURA 5.10: Influencia en el número de transistores del empleo de módulos

espacios vacíos mayores y, por lo tanto, mayor área global.

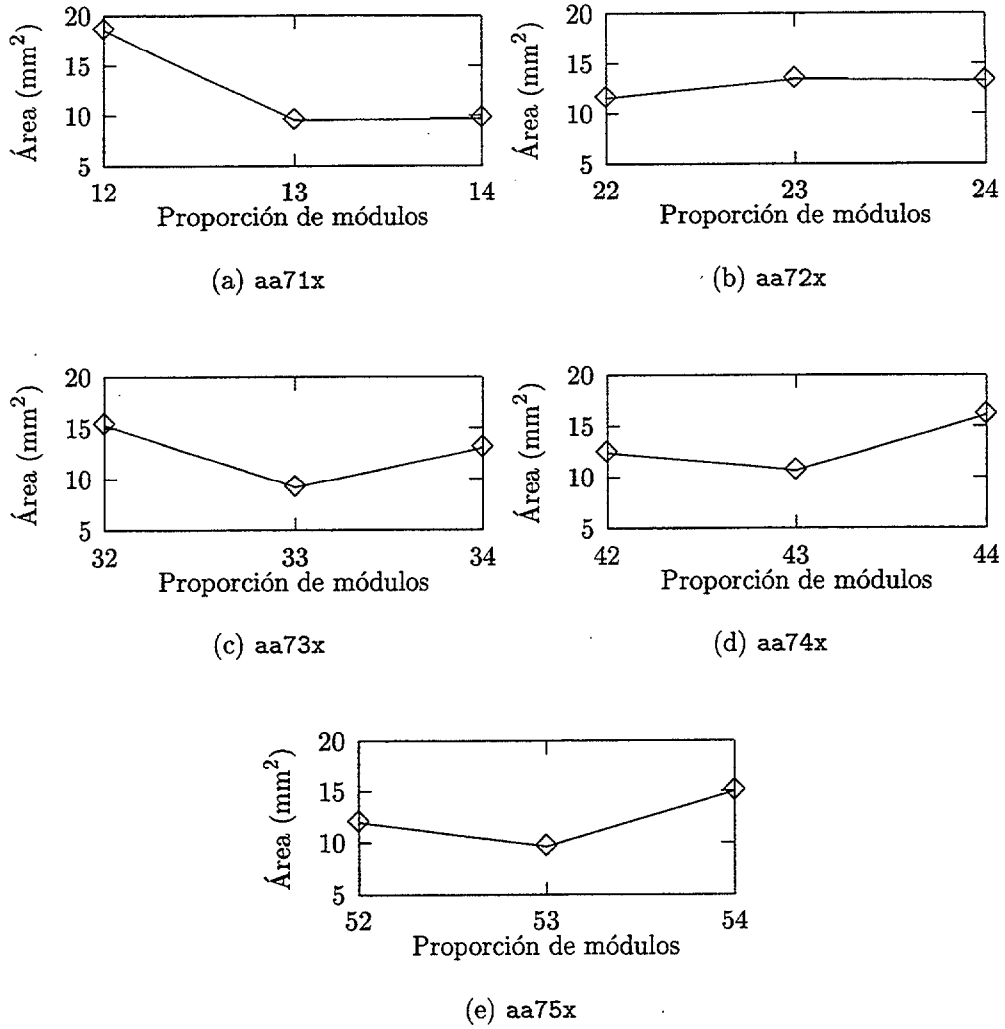


FIGURA 5.11: Influencia en el área del empleo de módulos en las series aa71x, aa72x, aa73x, aa74x y aa75x

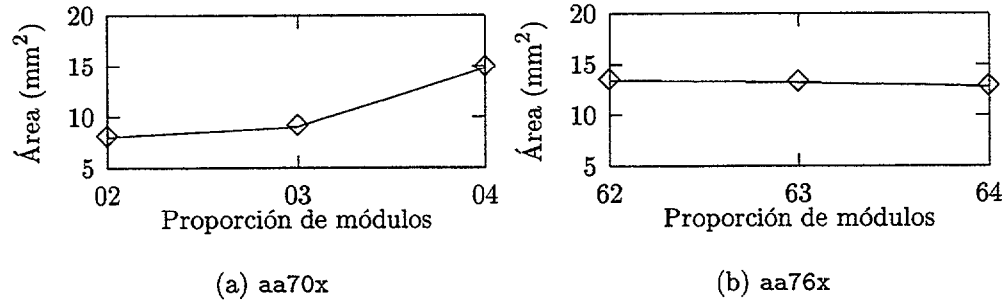


FIGURA 5.12: Influencia en el área del empleo de módulos en las series aa70x y aa76x

5.4.3. Influencia de la definición de grupos (granularidad física)

En el entorno de diseño que se ha establecido, en la etapa de síntesis física la mayor parte de las pruebas se han realizado con ubicación automática sin particionar. No obstante, en ocasiones puede interesar guiar a la herramienta de síntesis física elegida para poder realizar una ubicación más apropiada con la que disminuir el interconexionado en determinadas rutas, las capacidades parásitas y necesidades de *fan-out* asociadas y con ello mejorar la velocidad.

Uno de los medios más comunes para realizar el guiado en este tipo de herramientas es defininiendo *grupos*. De esta forma la herramienta realiza la ubicación de forma jerárquica, intentando optimizar en primer lugar los grupos definidos y posteriormente el diseño completo sin alterar estos grupos.

En esta ocasión la definición de grupos realizada se ha basado en los dominios que se han establecido en el modelado de las arquitecturas.

En este punto se hace necesario considerar esta definición de grupos como un medio de guiar a la herramienta de síntesis física en su tarea de colocado de los distintos elementos, al igual que cuando se establece la propiedad `fixedblock` a 1. Por esta razón se han considerado ambos factores como elementos de 'granularidad física'. Su influencia puede considerarse análoga a la que se produce en otro tipo de herramientas donde se emplean técnicas de síntesis física jerárquicas guiadas por el usuario.

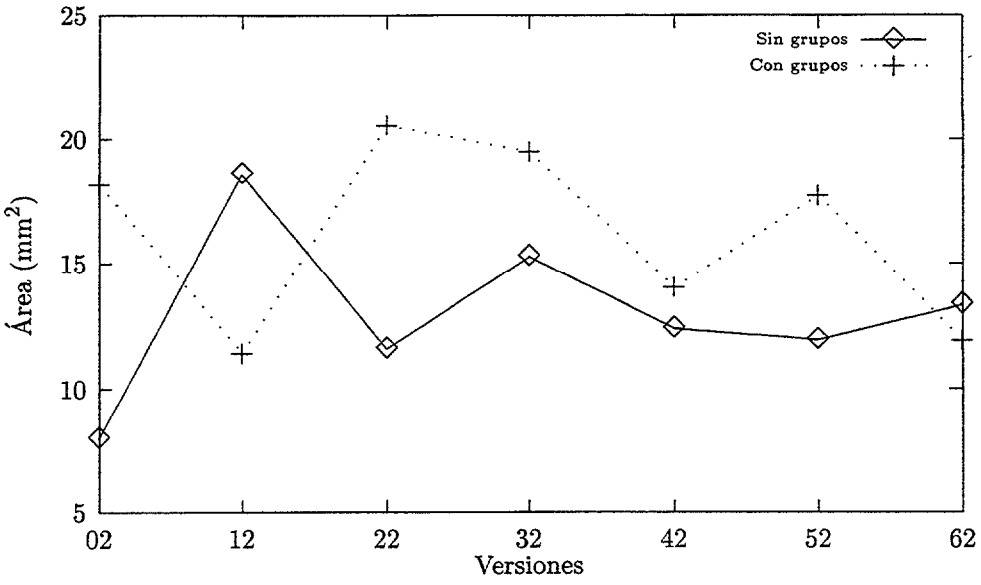
En las figuras 5.13, 5.14, 5.15 y 5.16 se muestra la influencia de la definición de estos grupos.

Como se desprende de las figuras que corresponden a este apartado, la definición de grupos de forma manual por regla general ha producido implementaciones con áreas mayores. Por lo que respecta a la velocidad las agrupaciones que se han establecido sólo han permitido mejorar las prestaciones en casos muy específicos, si bien estas mejoras —que se deben únicamente a ubicaciones diferentes de las completamente automáticas— son mínimas excepto en algún caso muy particular.

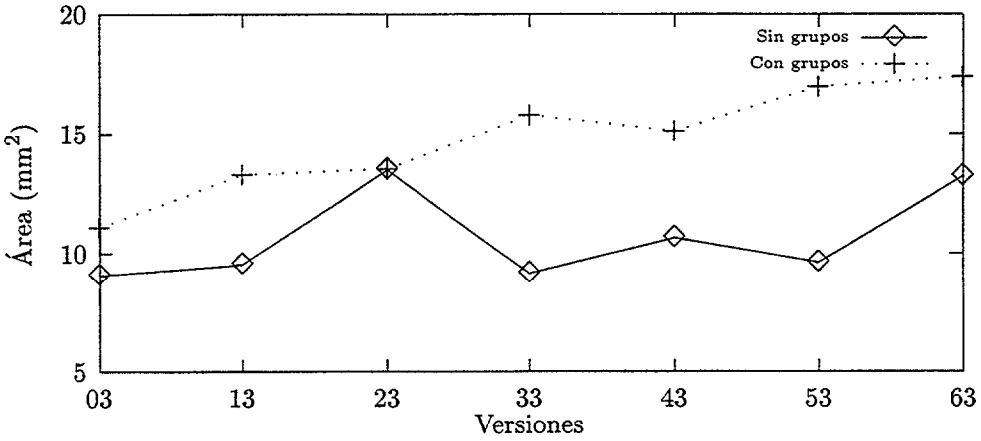
Igualmente se puede observar que en lo que respecta a la potencia, el consumo de potencia aumenta apreciablemente al establecerse las agrupaciones. Gran parte de este aumento de potencia parece guardar relación con el aumento generalizado de área, donde al haberse establecido los grupos de forma tan nítida previsiblemente el interconexionado aumenta. Este aumento ocasiona que las capacidades parásitas aumenten y ello ocasiona este aumento de potencia.

Como puede observarse, el área aumenta notablemente, así como el consumo de potencia, sea cual sea la granularidad del diseño. Sin embargo, es notable que la velocidad no se deteriore tanto como el aumento de área puede hacer prever. Esto es resultado de la proximidad entre puertas funcionalmente relacionadas obtenida con las agrupaciones lógicas organizadas jerárquicamente.

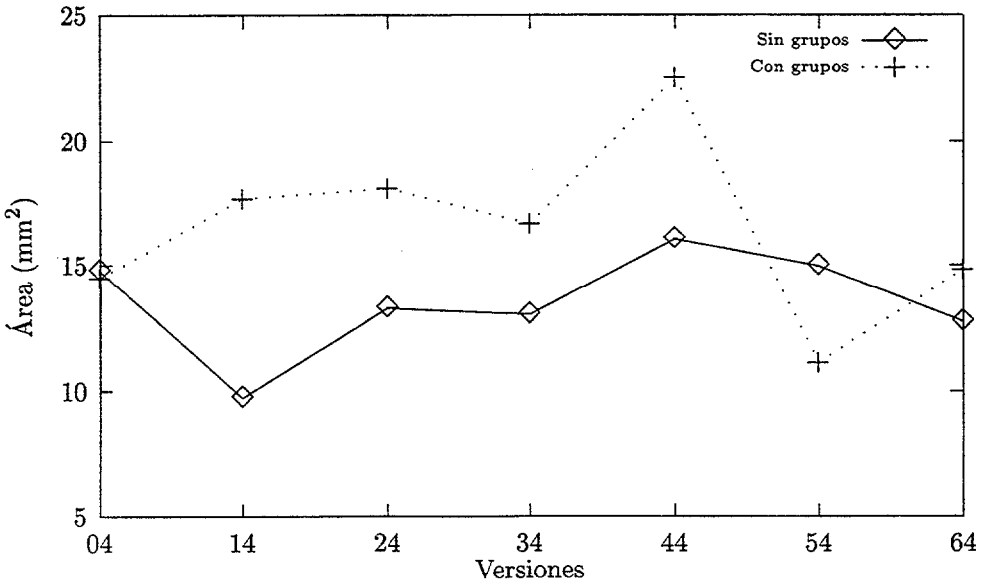
La conclusión, no obstante, es que este nivel de agrupación raramente compensa. Permanece en cambio establecida, y aún más resaltada, la conveniencia de diseñar con un nivel 'medio', adecuado, de granularidad lógica mediante el empleo de módulos en



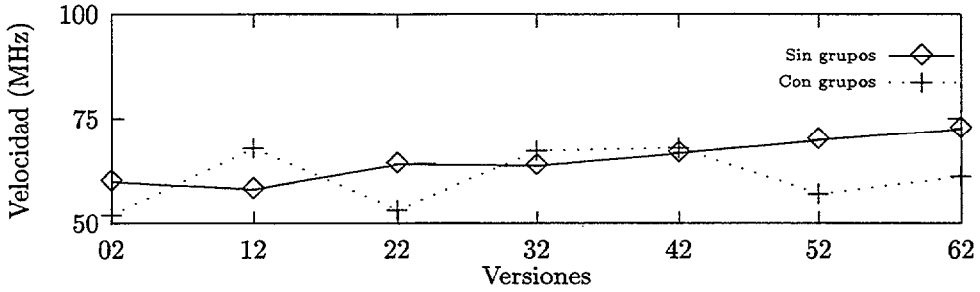
(a) 'Pequeña' cantidad de módulos



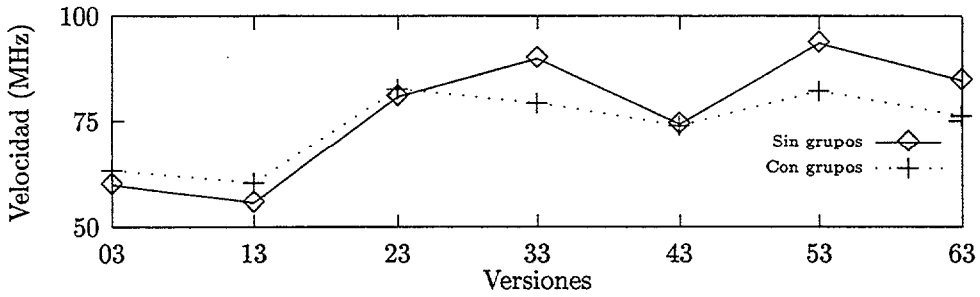
(b) 'Media' cantidad de módulos



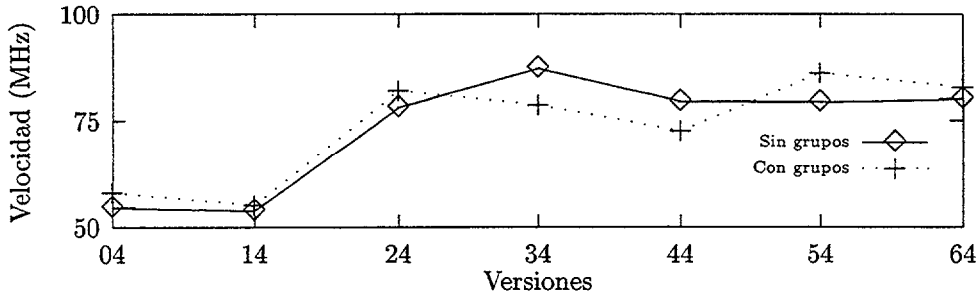
(c) 'Gran' cantidad de módulos



(a) 'Pequeña' cantidad de módulos



(b) 'Media' cantidad de módulos

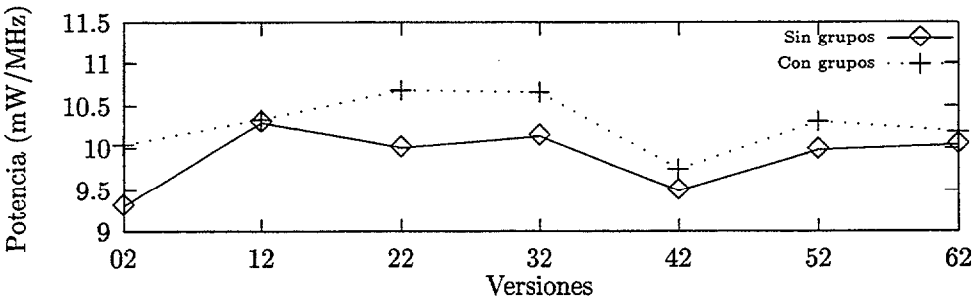


(c) 'Gran' cantidad de módulos

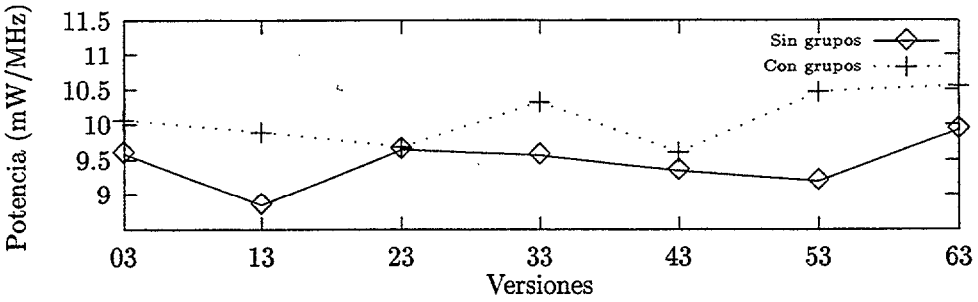
FIGURA 5.14: Influencia de la definición de grupos en la velocidad

alto nivel.

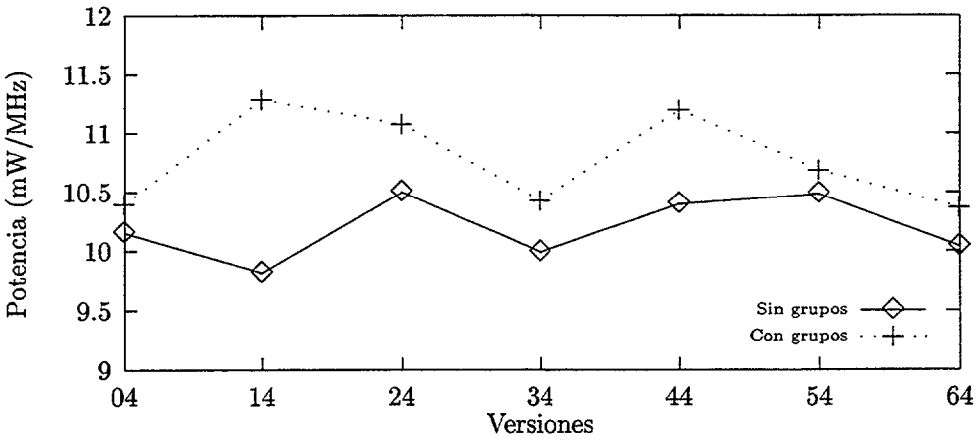
Igualmente se puede observar en la fig. 5.16 que el número de transistores de este conjunto de implementaciones no se ve afectado significativamente para cada variante microarquitectural. De todo ello se deduce que la cantidad de lógica que modifican las variantes es pequeña comparada con el total de funcionalidad del procesador. Pero estas variantes tienen efectos muy significativos en velocidad, potencia y área. El mínimo efecto que tienen sobre el número de transistores no debe llevar a realizar descuidadamente estas modificaciones, puesto que el impacto en el área es grande. Esta observación resalta de nuevo la importancia del ruteado físico, y en general de un diseño físico integrado con el diseño lógico y microarquitectural.



(a) 'Baja' cantidad de módulos

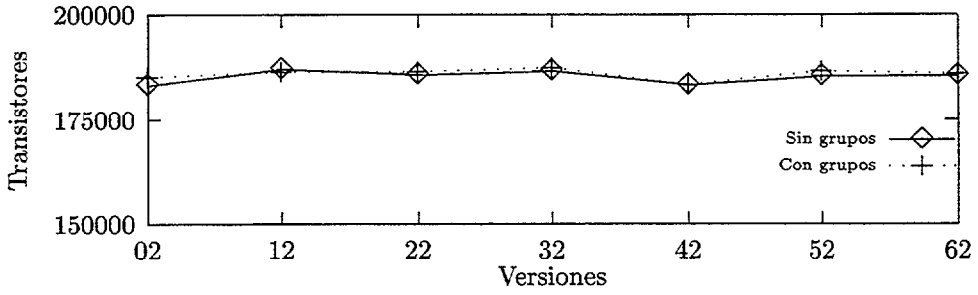


(b) 'Media' cantidad de módulos

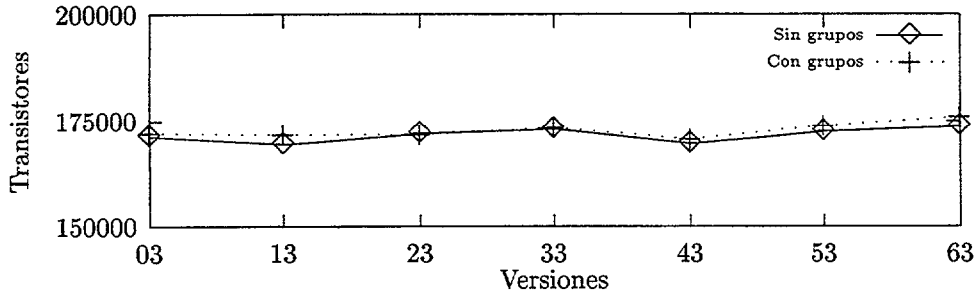


(c) 'Gran' cantidad de módulos

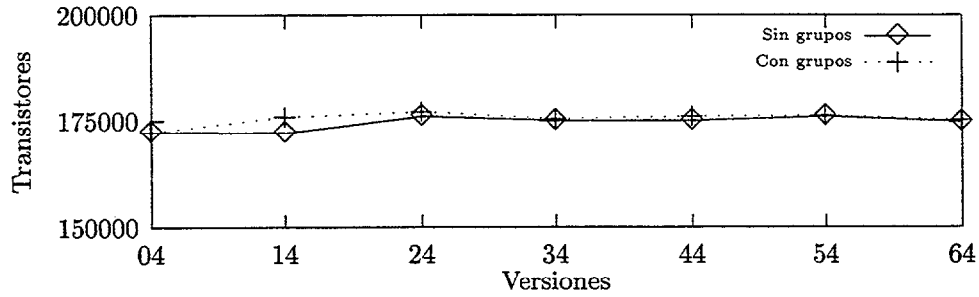
FIGURA 5.15: Influencia de la definición de grupos en la potencia



(a) 'Baja' cantidad de módulos



(b) 'Media' cantidad de módulos



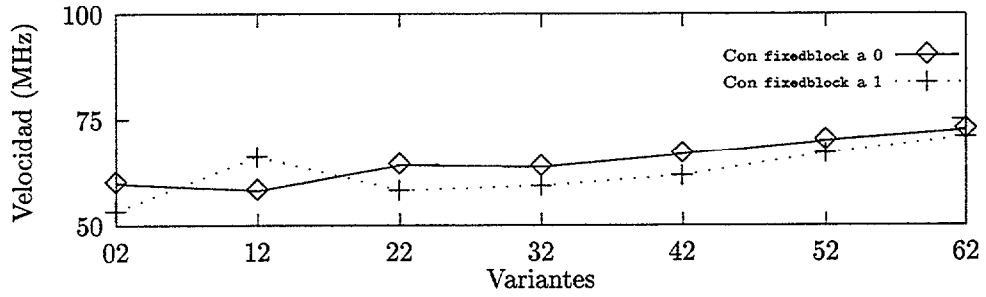
(c) 'Gran' cantidad de módulos

FIGURA 5.16: Influencia de la definición de grupos en el número de transistores

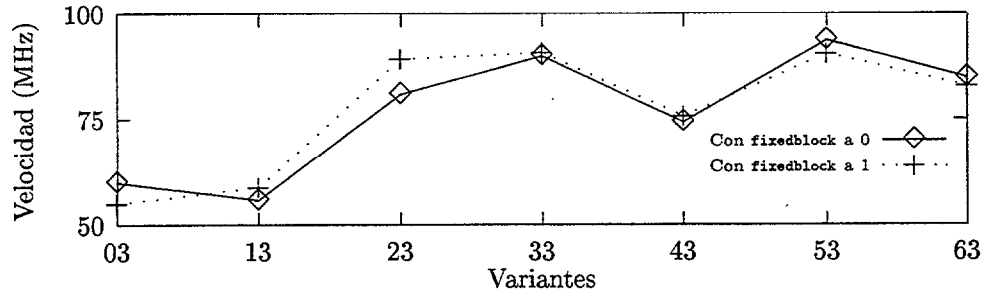
5.4.4. Influencia de la propiedad `fixedblock`

La propiedad `fixedblock` es una propiedad que se le puede añadir a los módulos de Epoch para forzar a que, si se ubica una copia de un módulo, éste no se altere. En este caso esta propiedad debe ir puesta al valor 1. Sin embargo, si esta propiedad se pone a 0, entonces al realizarse la síntesis física del diseño donde se encuentran se vuelve a realizar la síntesis de estos bloques para adaptar sus características eléctricas al entorno en el que se va a encontrar.

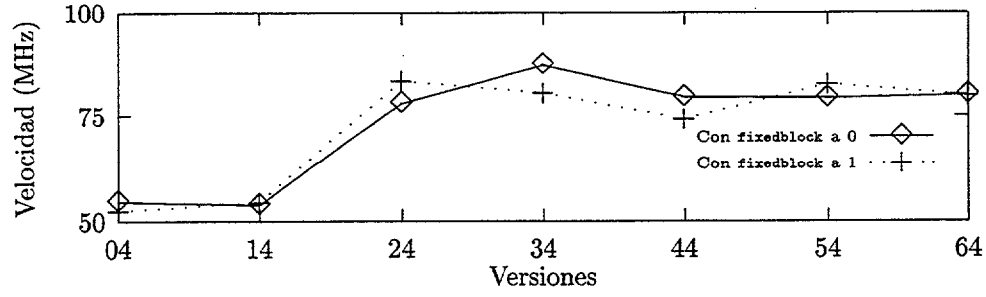
Normalmente se debería seguir la regla de poner esta propiedad al valor 0 para módulos pequeños, y se pondrá en casos muy particulares este valor a 1. Sin embargo, en el caso de los módulos realmente grandes cuyas propiedades se desean conservar al formar parte de otros diseños, este valor debería ir a 1.



(a) 'Pequeña' cantidad de módulos



(b) 'Media' cantidad de módulos

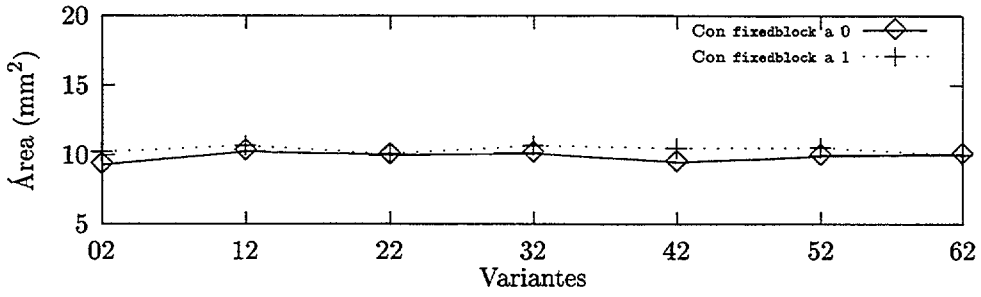


(c) 'Gran' cantidad de módulos

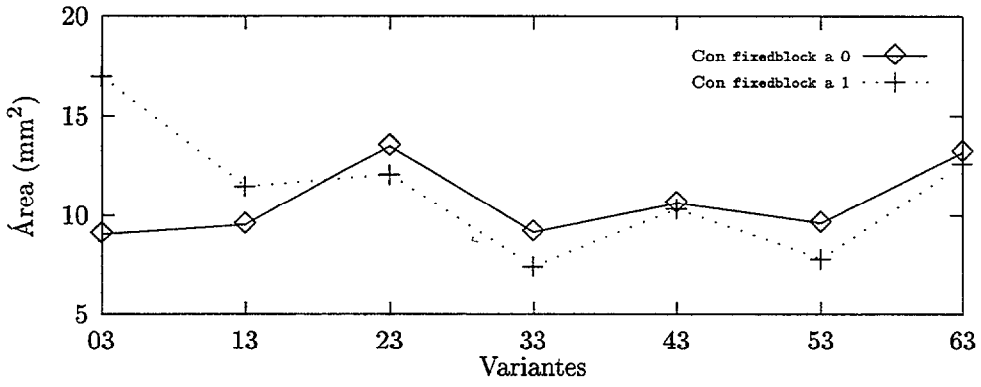
FIGURA 5.17: Influencia del atributo `fixedblock` en la velocidad

En las figuras 5.18, 5.17, 5.19 y 5.20 se puede ver la influencia de la utilización de

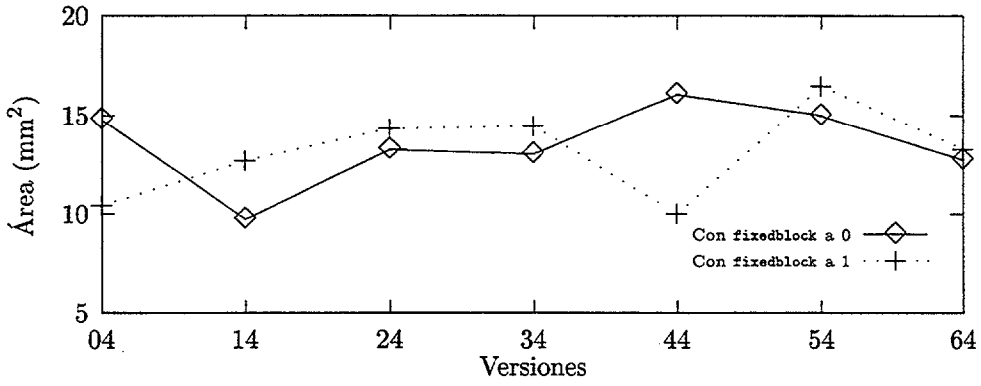
este parámetro `fixedblock` sobre distintas propiedades.



(a) 'Pequeña' cantidad de módulos



(b) 'Media' cantidad de módulos

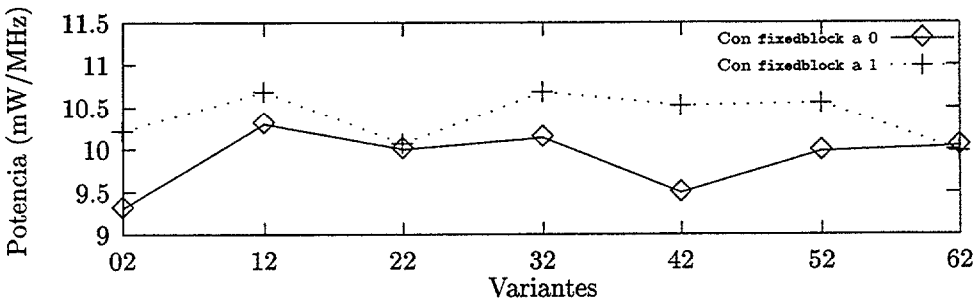


(c) 'Gran' cantidad de módulos

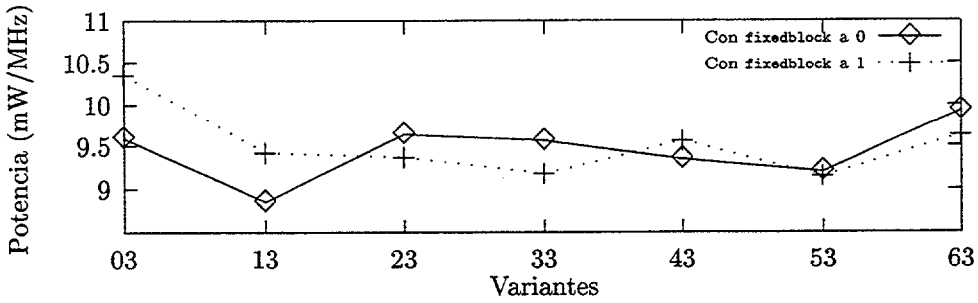
FIGURA 5.18: Influencia del atributo `fixedblock` en el área

Como era de esperar, el número de transistores apenas se ve afectado. Con grano fino el área tampoco se ve afectada, pero al incrementar el grano el ruteado con bloques fijos hace oscilar el área tanto en un sentido como en otro.

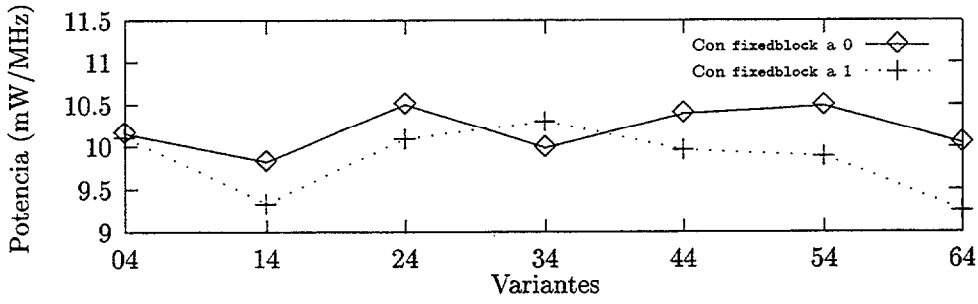
También se observa el efecto previsible del consumo de potencia correlacionado con el área. Ahora bien, puede observarse que para áreas similares —y en todo caso— el consumo de potencia sufre un ligero incremento adicional al ocasionado por el incremento de área con bloques fijos.



(a) 'Baja' cantidad de módulos

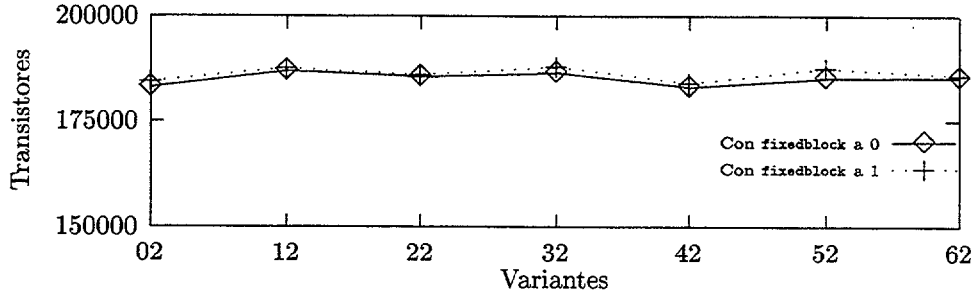


(b) 'Media' cantidad de módulos

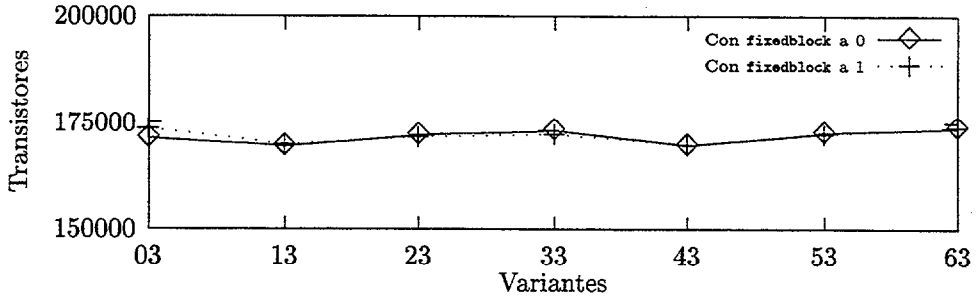


(c) 'Gran' cantidad de módulos

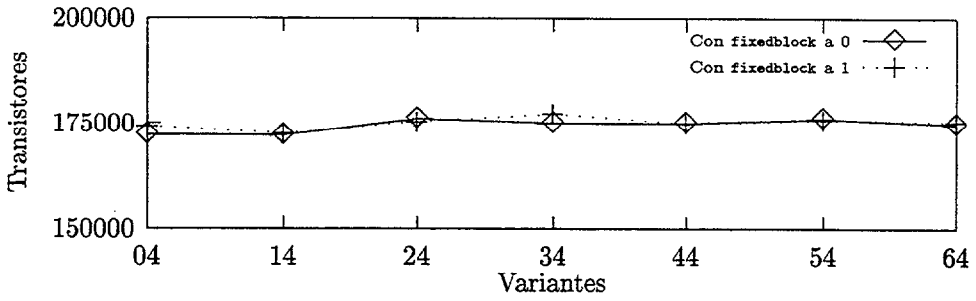
FIGURA 5.19: Influencia del atributo `fixedblock` en la potencia



(a) 'Baja' cantidad de módulos



(b) 'Media' cantidad de módulos



(c) 'Gran' cantidad de módulos

FIGURA 5.20: Influencia del atributo `fixedblock` en el número de transistores

Ya se ha visto que la velocidad es un parámetro que se controla fuertemente a nivel primero de opciones microarquitecturales y luego de granularidad lógica. El efecto sobre la velocidad de fijar o no los bloques modulares en su síntesis física es prácticamente inapreciable.

No es sorprendente este fenómeno si se entiende bien el concepto de síntesis lógica de un módulo y el de dar libertad o no a la síntesis física, permitiendo, por ejemplo, variaciones concretas del módulo lógico ya sintetizado para examinar condiciones de carga o no.

5.4.5. Influencia de la estrategia de secuenciamiento

En el conjunto de variantes que se ha construido, existen tres estrategias distintas para la gestión del secuenciamiento. Estas estrategias afectan exclusivamente al modo de actuar ante las instrucciones de salto, y son:

1. Predecir los saltos como que se toman siempre, y, para calcular la dirección de la búsqueda siguiente, realizarlo con una suma en el ciclo donde se necesita este dato. Esto se ha realizado en las variantes `---0x` y `---1x`.

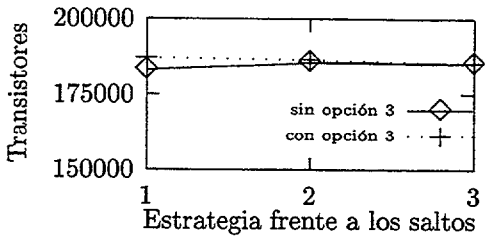
2. Predecir los saltos como que se toman siempre, y, para calcular la dirección de la búsqueda siguiente, siempre evaluarse dos alternativas. Una vez se conozca el resultado de la decisión de secuenciamiento se elige la adecuada y la otra se guarda en un registro. Si posteriormente se detecta que la decisión de secuenciamiento fue errónea (como en las predicciones erróneas) entonces se recupera la guardada anteriormente. Esto se ha llevado a cabo en las variantes `---2x`, `---3x` y `---4x`.

3. Predecir los saltos como que se van a tomar sólo en caso que el salto sea hacia atrás y, para calcular la dirección de la búsqueda siguiente, siempre evaluarse dos alternativas. Esto se ha empleado en la construcción de las variantes `---5x` y `--6x`.

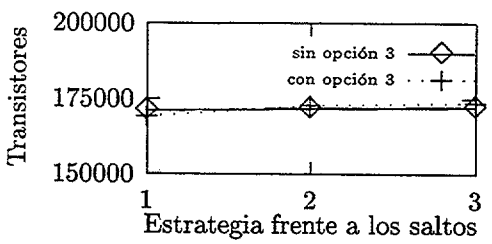
A su vez, también existe una circunstancia especial que sólo se verifica en las variantes `---1x`, `---3x` y `---6x` que afecta al secuenciamiento. Y es que si la instrucción anterior a la de un salto no modifica los códigos de condición, entonces no es necesario realizar ninguna predicción. En ese caso sólo hay que tomar la decisión de secuenciamiento según los códigos de condición.

En las figuras 5.21, 5.22 y 5.23 se puede ver la influencia sobre las distintas propiedades. Como se podrá observar, la introducción de la opción 3 —que suponía poder evaluar los códigos de condición tanto en la etapa de decodificación como en la de ejecución—, apenas supone cambios en el número de transistores dentro del que se emplean en las variantes (fig. 5.21). Esto parece cumplirse dentro de las proporciones establecidas entre módulos y células estándares

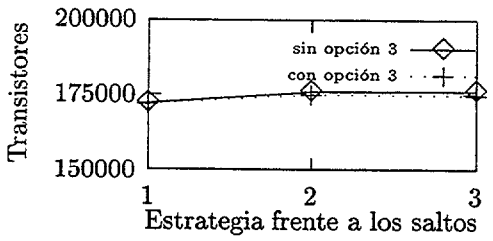
Sin embargo, para el resto de las propiedades la evolución es algo arbitraria. Así, en las versiones sin esta opción 3 la velocidad suele ser más lenta. En contadas ocasiones las versiones que no incluyen esta opción resultan más rápidas. No obstante, en las versiones



(a) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b



(b) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b



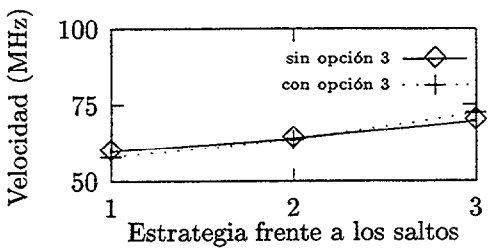
(c) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b

FIGURA 5.21: Influencia en el número de transistores de las estrategias de secuenciamiento con baja proporción de módulos

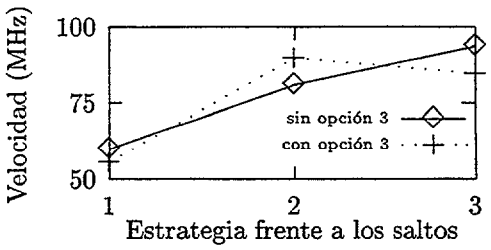
con 'media' proporción de módulos se puede identificar una de las posibles causas que ocasiona estos resultados. Como se puede observar en la fig. 5.23.b, se puede ver cómo el colocado ha resultado mejor en las opciones donde siempre se predice. Esto podría tener su correspondencia con la mejora de velocidad en estos dos casos, si bien no resulta tan significativa la distancia como en el número de transistores (véase la figura 5.22).

Por otro lado, al analizar los casos donde el área permanece constante entre versiones con similiares proporciones de módulos (versiones aa702, aa722 y aa752 de la fig. 5.23.a, y aa704, aa724 y aa754 de la fig. 5.23.c) se podría considerar paradójico que en las figs. 5.22.a y 5.22.b las curvas de sus frecuencias tengan pendiente ascendente.

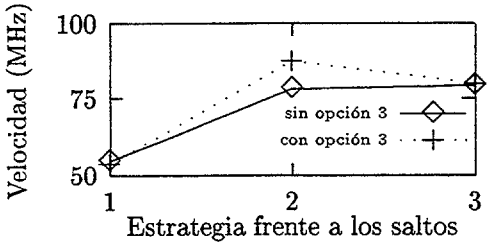
No obstante, debe considerarse que el número de transistores es relativamente constante, por lo que todo hace pensar que cuanto más sencilla es la estrategia de secuenciamiento mayores resultan las necesidades de carga de determinadas señales —tanto por el número de elementos como la longitud total del ruteado para las conexiones—, y podría ser esta la causa de sus menores velocidades. Como se puede ver, estas diferencias suelen ser menos acusadas para las variantes con 'media' (fig. 5.22.b) y 'gran' (fig. 5.22.b) cantidad de módulos. En general, cuanto más se *deleguen* las decisiones sobre cada uno de los elementos que influyen en el secuenciamiento, más rápida resultará la variante.



(a) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b



(b) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b



(c) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b

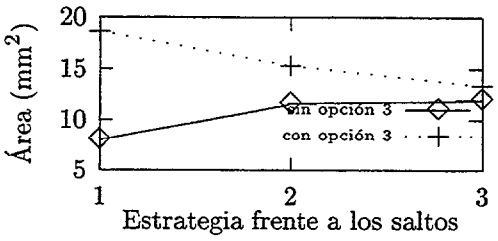
FIGURA 5.22: Influencia en la velocidad del empleo de las estrategias de secuenciamiento con baja proporción de módulos

5.4.6. Influencia de la posición del bypass

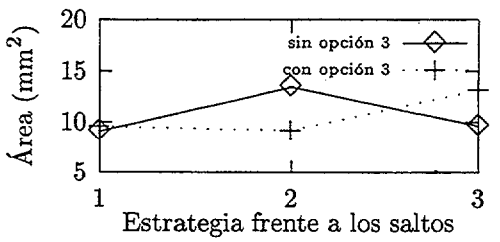
La situación de los elementos que realizan el *bypass* de los datos puede ocasionar una pequeña influencia en el tiempo de ciclo. En este diseño las rutas más largas en la Ruta de Datos tienen tiempos similares a las de la Unidad de Control. Sin embargo, existen situaciones donde para tomar decisiones en la Unidad de Control se tienen que evaluar determinadas operaciones realizadas con los operandos fuente *rs1* y *rs2*. Esto se hace especialmente necesario ante el posible lanzamiento de determinadas excepciones. En estos casos, poner parte de la circuitería del *bypass* al principio de la etapa de ejecución (o lo que es lo mismo, *tras* los registros fuente) puede alargar el tiempo de ciclo —que lo fijaría la Unidad de Control— y, por tanto, bajar ligeramente la velocidad del procesador.

Esta es, precisamente, la situación que se produce en las situaciones donde se emplea la 'media' cantidad de módulos (véase la fig. 5.25.a). En las otras versiones apenas se aprecia variación debido a esta circunstancia, pero en caso de intentarse optimizaciones posteriores este hecho podría apreciarse con mayor claridad.

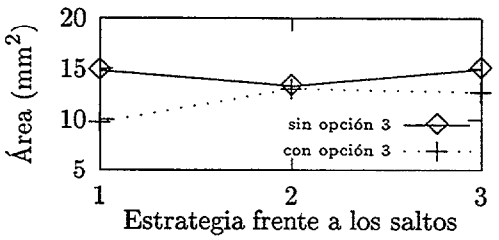
Igualmente se puede observar que la disminución en el número de transistores apenas es apreciable. También se puede apreciar cómo al involucrar parte de la circuitería del *bypass* en más situaciones que en el caso anterior (con la opción 3 siempre debe operar mientras se necesiten los operandos fuente) hace que por regla general la potencia relativa sea mayor.



(a) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b



(b) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b

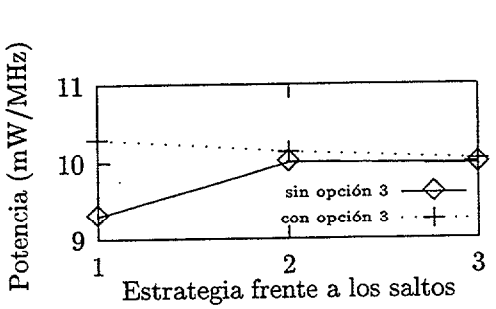


(c) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b

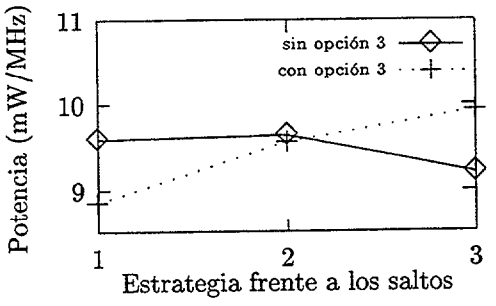
FIGURA 5.23: Influencia en el área del empleo de las estrategias de secuenciamiento con baja proporción de módulos

5.4.7. Influencia del número de ventanas

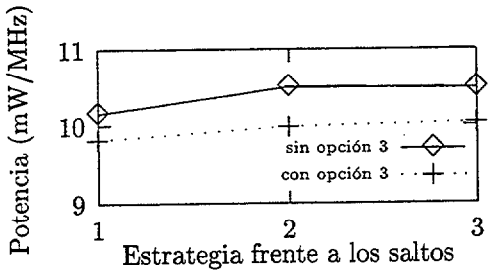
El número de ventanas en el fichero de registros es configurable para implementaciones de la arquitectura SPARC. Este número puede variar desde 2 hasta 32. Al haberse realizado el diseño de forma que el fichero de registros no se encuentre en la ruta crítica, el número de ventanas que se decida para una implementación no debe modificar la velocidad de la implementación. Obviamente la incidencia del número de ventanas sobre el área, y por lo tanto sobre el costo, es muy grande, así como sobre el consumo de potencia. En todo caso éste es un parámetro a diferenciar a nivel de sistema en el marco de los programas y estructuras de datos de la carga de trabajo del procesador.



(a) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b

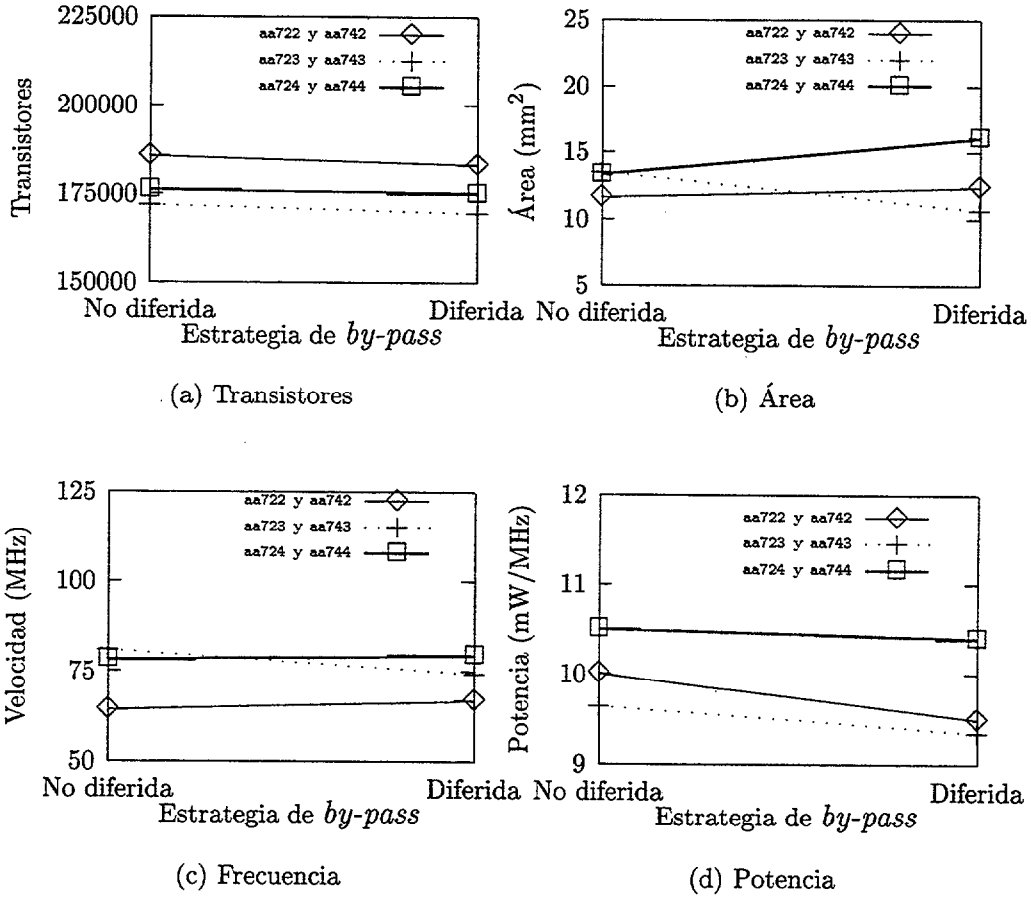


(b) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b



(c) 1.- Opciones 2.a y 4.a, 2.- Ops. 2.a y 4.b, 3.- Ops. 2.b y 4.b

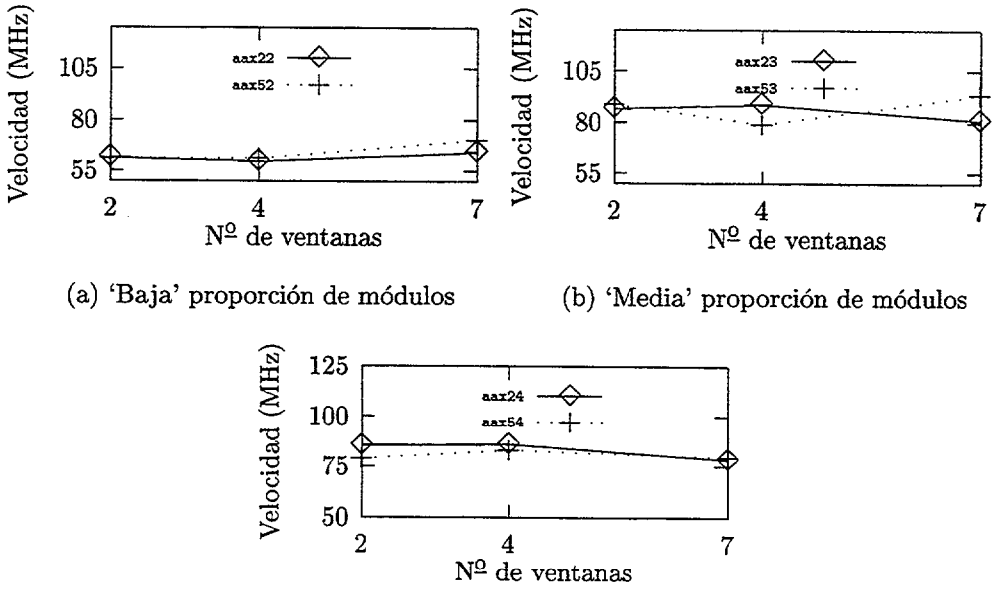
FIGURA 5.24: Influencia en la potencia relativa de las estrategias de secuenciamiento con baja proporción de módulos



(a) Transistores (b) Área

(c) Frecuencia (d) Potencia

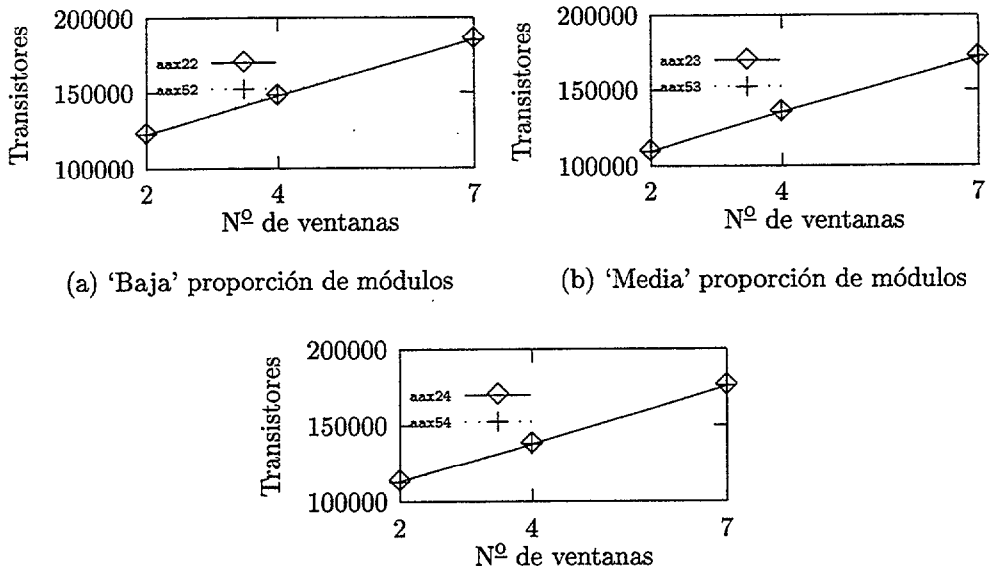
FIGURA 5.25: Influencia en las distintas propiedades de la situación del *bypass*



(a) 'Baja' proporción de módulos (b) 'Media' proporción de módulos

(c) 'Alta' proporción de módulos

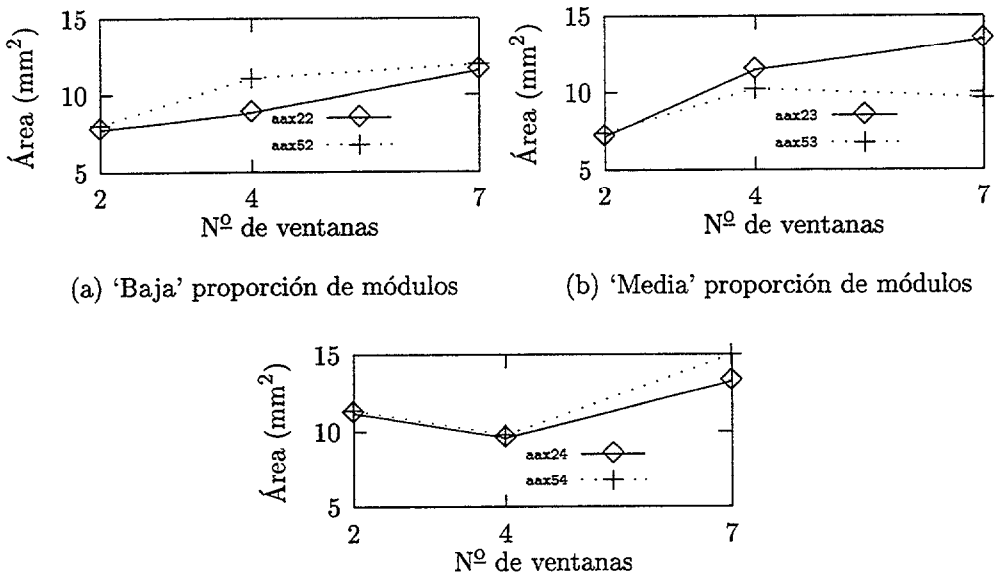
FIGURA 5.26: Influencia en la velocidad del número de ventanas



(a) 'Baja' proporción de módulos (b) 'Media' proporción de módulos

(c) 'Alta' proporción de módulos

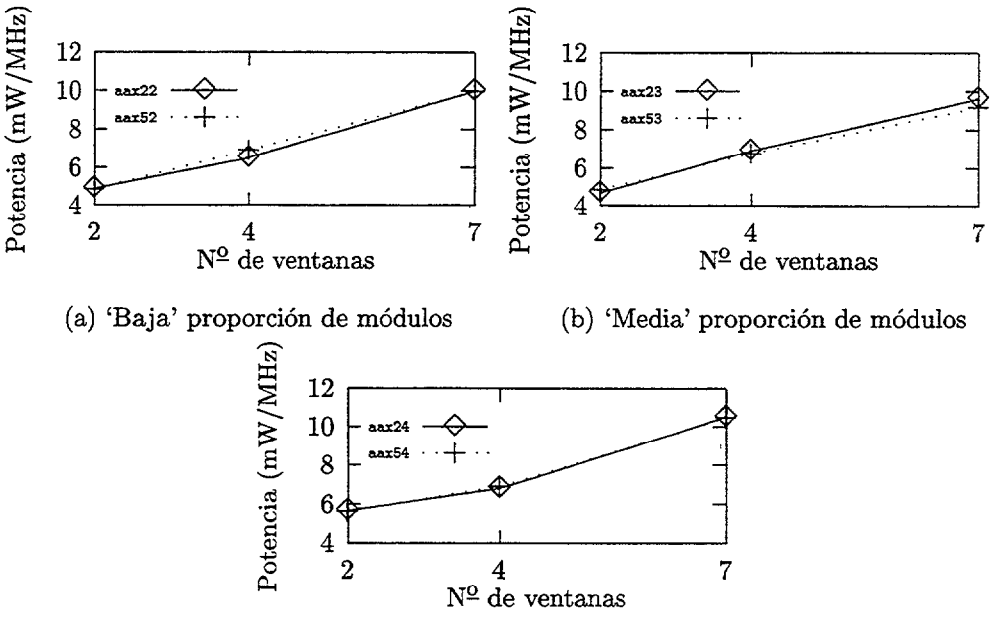
FIGURA 5.27: Influencia en la cantidad de transistores del número de ventanas



(a) 'Baja' proporción de módulos (b) 'Media' proporción de módulos

(c) 'Alta' proporción de módulos

FIGURA 5.28: Influencia en el area del número de ventanas



(a) 'Baja' proporción de módulos

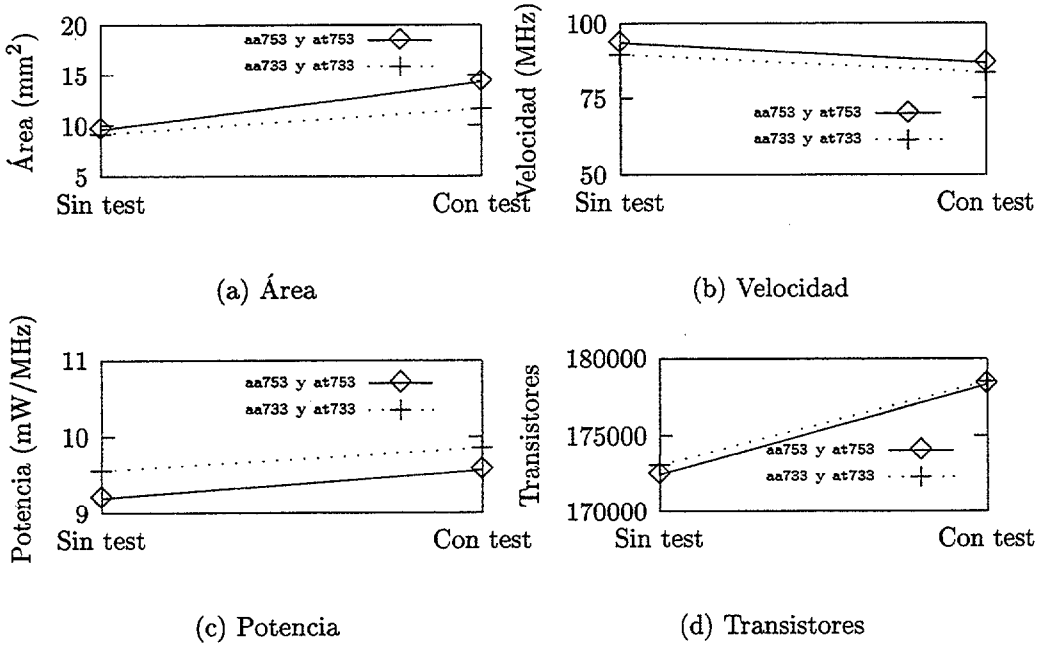
(b) 'Media' proporción de módulos

(c) 'Alta' proporción de módulos

FIGURA 5.29: Influencia en la potencia del número de ventanas

5.4.8. Influencia de la inserción de la circuitería de test

Para estudiar cómo influye la presencia de los elementos para el test en estas implementaciones se ha escogido como referencia la versiones aa733 y aa753.



(a) Área (b) Velocidad

(c) Potencia (d) Transistores

FIGURA 5.30: Influencia en la inserción de elementos de test

Como puede desprenderse de la fig. 5.30, la inserción de los elementos de test resulta, como es de esperar, ligeramente perjudicial en lo que respecta a las prestaciones de velocidad y en el número de transistores. Pueden o no verse afectadas de forma negativa el área y la potencia, de forma que la síntesis física juega un papel fundamental en estos factores. Estas variaciones dependen del hecho de que en las versiones con el test aumenta el número de elementos que forman parte de la implementación, y con ellos aumentan las opciones del diseño físico.

La velocidad disminuye siempre en las versiones con test al aumentar el tiempo de ciclo, ya que algunos de los elementos nuevos deben necesariamente alojarse en la ruta crítica.

No obstante, el efecto de la circuitería de test, que debe introducirse siempre en el diseño final, es básicamente el mismo en todos los casos, por lo que no incluimos las figuras equivalentes a la 5.30 para el resto de las versiones generadas.

5.4.9. Influencia de la presencia de bloque de memoria

Cuando la síntesis física se realiza permitiendo total flexibilidad al realizar los elementos, la herramienta encuentra más elementos a gestionar que en caso de operar únicamente con el núcleo.

Por otra parte, en otras ocasiones no sólo es la mera coexistencia de otros elementos

la que resta libertad al diseño sino que en ocasiones la necesidad de interconectar estos elementos impone restricciones nuevas y, por ello, exige mayor esfuerzo de la herramienta para intentar mantener las propiedades del núcleo original.

Por estas razones suele ocurrir que al incluirse nuevos elementos en el diseño, las síntesis físicas realizadas de forma libre presentan velocidades peores, pero previsiblemente las otras propiedades de área y potencia deben mejorar.

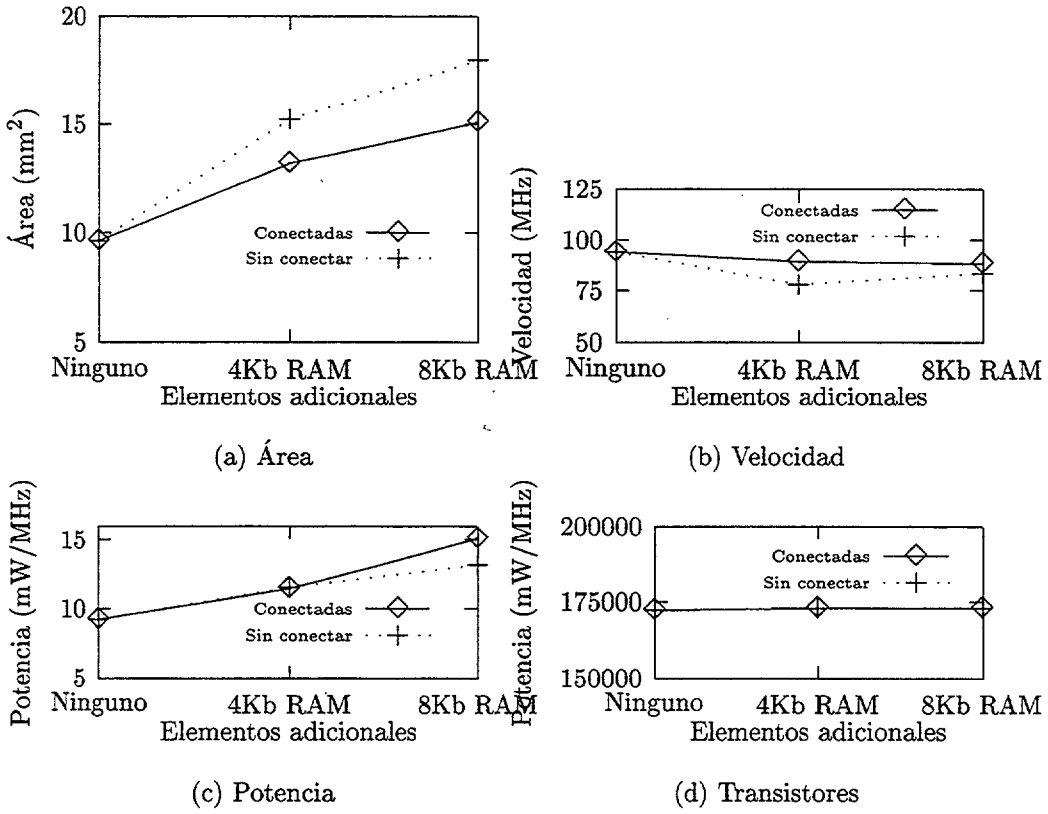


FIGURA 5.31: Influencia de la presencia de memorias

Como se puede observar en la fig. 5.31, parte de estas apreciaciones se confirman al realizar una síntesis física con plasticidad libre para la variante `aa753` con distintos memorias. Como puede observarse, cuando mayores son las memorias con las que el núcleo debe compartir espacio, al intentar minimizar el área total ocupada las propiedades del núcleo empeoran. Sin embargo, en contra de lo que se puede suponer, las versiones que tienen las memorias conectadas al núcleo presentan mejores prestaciones. A pesar de que, en principio, se podría pensar que la existencia de las conexiones puede suponer una pérdida de grados de libertad para la herramienta, las prestaciones mejoran bajo estas condiciones.

Aún en cualquiera de estas condiciones suele resultar útil fijar las características del procesador, donde, si bien cabe suponerse que el aumento de restricciones en la síntesis física podría aumentar el área global, no es de esperar pérdidas de prestaciones.

En la fig. 5.32 se puede observar la influencia de esta decisión en el área global.

Una vez más, en contra de lo esperado, en la fig. 5.32 se constata como habiéndose

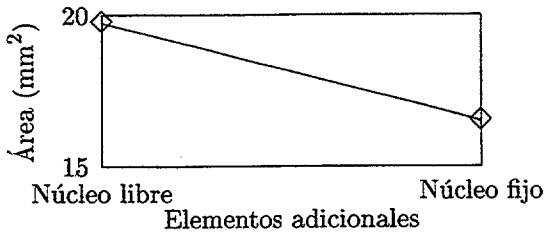


FIGURA 5.32: Influencia en el área de una versión aa753 con una RAM de 8Kbytes

fijado el núcleo el área se ha reducido. Por esta razón, nuevamente se impone la necesidad de realizar diversas pruebas hasta obtenerse los resultados deseados.

5.4.10. Influencia de la presencia de una FPU

En el caso de las versiones con una FPU añadida en una ocasión se ha permitido la plasticidad completamente libre, y en otras dos ocasiones se ha fijado la implementación de la Unidad de Enteros. Se ha estudiado las características del conjunto para estas distintas alternativas. En las ocasiones donde se ha fijado la IU sus propiedades previamente estudiadas por separado permanecen exactamente iguales.

Tres han sido finalmente las implementaciones realizadas con FPU, siguiendo las siguientes alternativas:

1. IU fija y FPU con colocado libre,

2. IU fija y FPU con colocado guiado, y

3. IU y FPU con colocado libre.

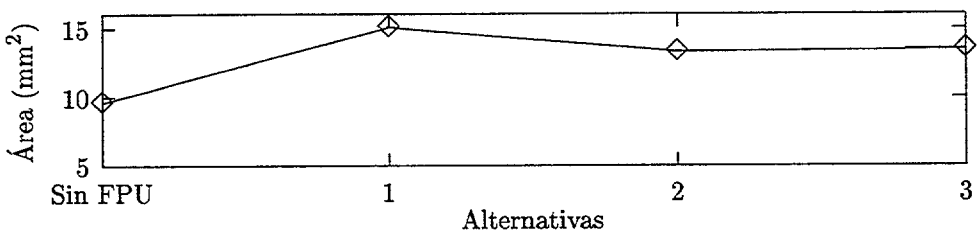
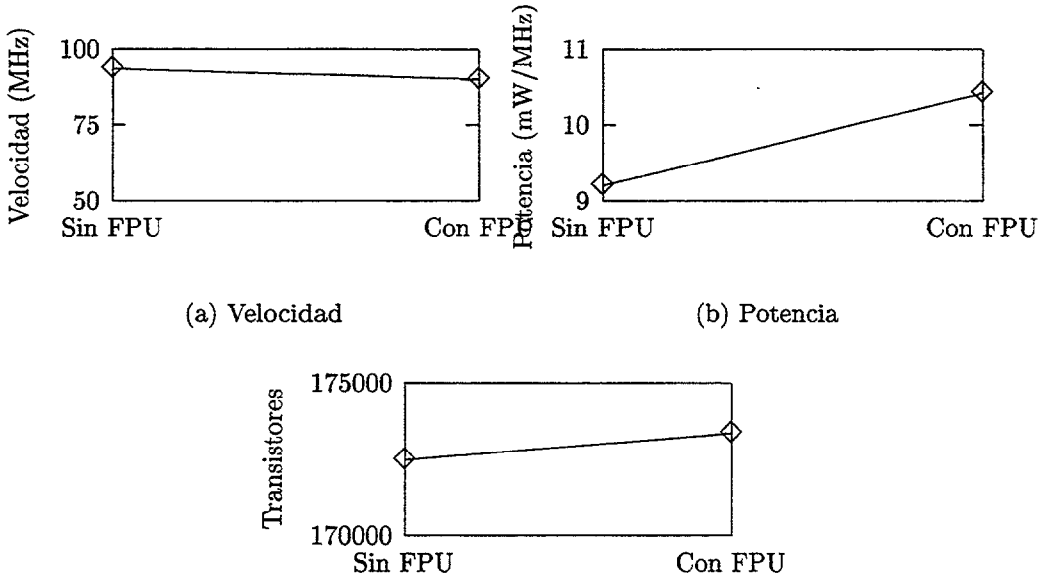


FIGURA 5.33: Influencia en el área de la presencia de una FPU

En el caso de la IU en la implementación con colocado libre el resto de propiedades siguen una evolución muy similar a la producida en la misma situación que se produjo con las memorias. Esto puede observarse en la fig. 5.34.



(a) Velocidad

(b) Potencia

(c) Transistores

FIGURA 5.34: Influencia de la presencia la FPU con plasticidad libre

5.4.11. Influencia de la integración de un módulo VIS en la Ruta de Datos

Cuando se integran elementos en la Ruta de Datos, las prestaciones del núcleo ya no pueden quedar prefijadas de antemano como en las situaciones anteriormente expuestas. En la fig. 5.35 se puede observar la incidencia en las prestaciones de la integración de nuevos elementos e instrucciones que permiten acelerar aplicaciones multimedia.

En esta fig. 5.35 se estudia las prestaciones para dos implementaciones distintas de la Unidad de Enteros con dos módulos distintos para desarrollar las extensiones VIS. Como puede observarse, el primero presenta peores prestaciones que el segundo, debido a que en el segundo se han introducido algunas mejoras.

5.4.12. Influencia de la tecnología

Todas las influencias estudiadas anteriormente se han estudiado tomando una única tecnología de referencia y comparando el comportamiento de las variantes. Sin embargo, en ocasiones también puede resultar de interés comprobar si este comportamiento en las variantes es homogéneo independientemente de la tecnología elegida, o por el contrario esto no se produce.

Para ello se han elegido dos variantes del conjunto desarrollado y se han estudiado sus propiedades tanto para la tecnología empleada en el conjunto de experiencias expuesto, una tipo CMOS de 0.35 μm, como para otra distinta, una tipo CMOS de 0.50 μm.

Como se puede observar en la fig. 5.36, al evolucionar de una tecnología de una resolución determinada a otra de menor resolución, se mejoran todas las propiedades,

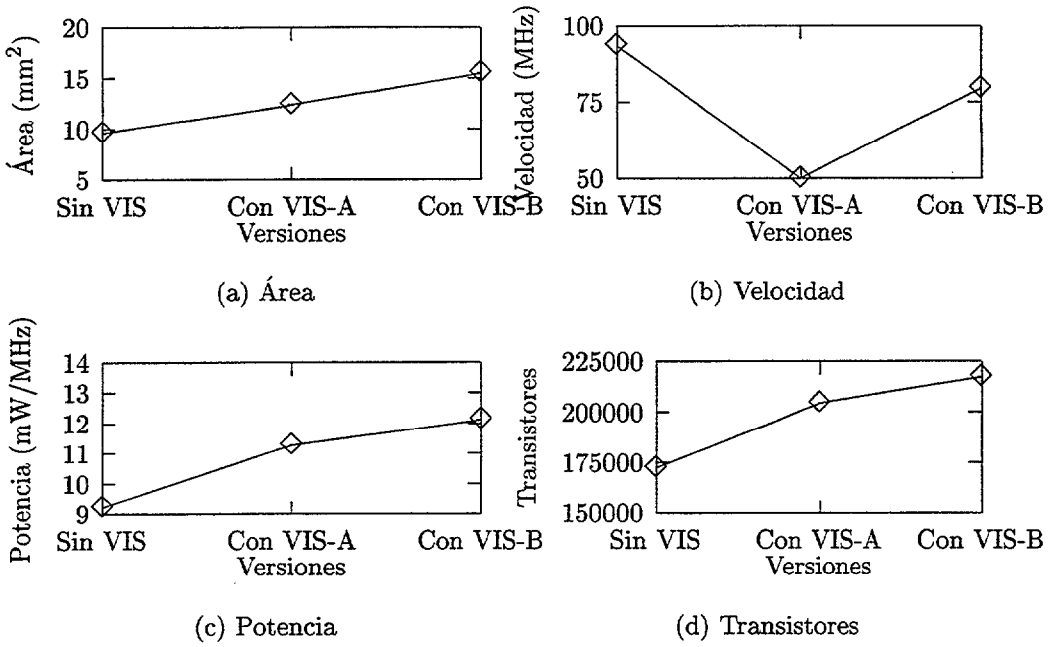


FIGURA 5.35: Influencia de la inserción de elementos en la ruta de datos (extensiones VIS)

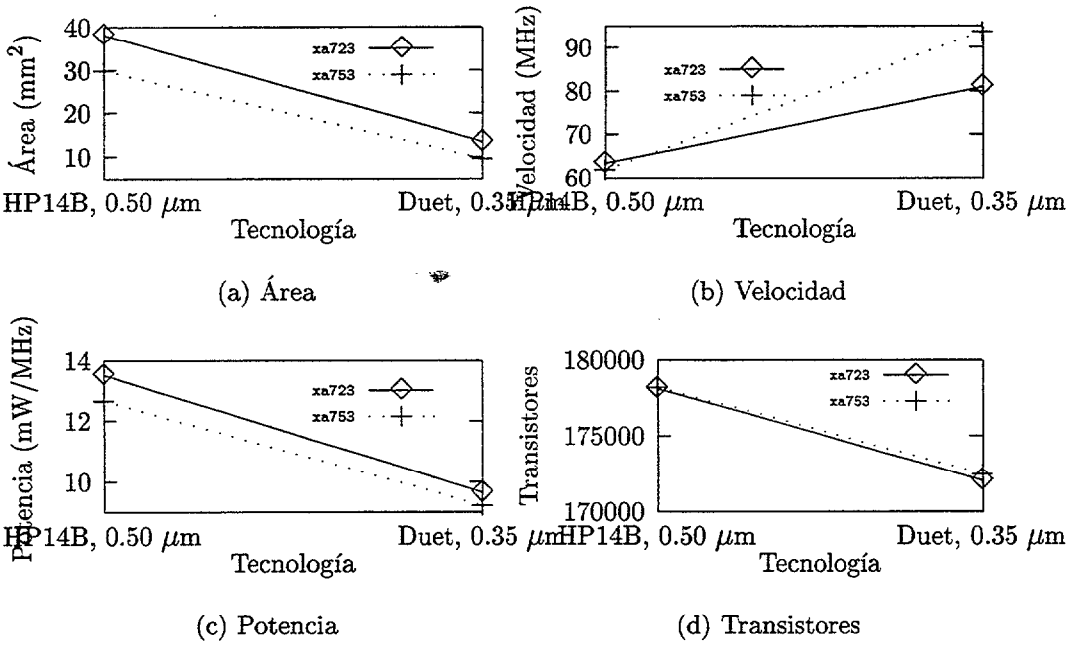


FIGURA 5.36: Influencia de la tecnología, en versiones con plasticidad libre

incluso la del número de transistores. Sin embargo, son las propiedades entre versiones con las mismas diferencias microarquitecturales donde en ocasiones pueden aparecer resultados distintos en cada tecnología. Mientras para la tecnología de 0.50 μm la versión ha723 es más rápida que la ha763, en la tecnología de 0.35 μm se produce la situación contraria. Sin embargo el resto de propiedades sigue la misma evolución en ambos casos.

CAPÍTULO 6

Conclusiones y líneas futuras

6.1. Conclusiones

Se ha establecido una metodología basada en VHDL para estudiar de forma cómoda y versátil la incidencia de diferentes estrategias microarquitecturales y de diseño en las prestaciones finales de un procesador generado como un núcleo para sistemas integrados en un chip. Esta metodología se apoya en el empleo eficiente de un entorno gráfico de modelado. Esta eficiencia se orienta principalmente a facilitar el control de los diferentes detalles de la microarquitectura, tanto en la concepción y el modelado como en la fase de síntesis lógica y física.

1. En primer lugar, se ha trabajado en técnicas de modelado en VHDL ideando métodos de descripción adecuados para realizar modificaciones y optimizaciones microarquitecturales.

2. En segundo lugar, se han encontrado métodos estructurales basados en concepciones gráficas mediante bloques de los elementos del diseño y se han identificado como los más idóneos para entornos de generación flexible de núcleos procesadores.

3. Se ha hecho un uso completo de las posibilidades de definición y generación de módulos en la síntesis lógica, y de agrupación de lógica en bloques fijos, aportando un análisis cuantitativo de sensibilidad del diseño a estas estructuras.

4. Se han usado las posibilidades de las herramientas lógica y física para definir y estudiar distintos niveles de granularidad lógica y física en el diseño y su impacto en las prestaciones, caracterizando igualmente la sensibilidad del diseño.

5. Se ha analizado la forma de los núcleos, y su adaptación para la interconexión con otros núcleos mediante el concepto de *plasticidad.*

6. Se han creado medios de caracterización de las implementaciones obtenidas (más de 100) y se han estructurado en tablas, gráficas y medidas estadísticas para facilitar su análisis y la toma de decisiones.

Empleando esta metodología se ha desarrollado un modelo de un núcleo de una Unidad de Enteros de un Procesador SPARC, de sus extensiones VIS, y de la Unidad de Coma Flotante. Partiendo de estos modelos se han desarrollado distintas versiones con idéntica Arquitectura de Juego de Instrucciones pero con pequeñas variaciones tanto microarquitecturales como de diseño. De esta forma, y gracias al flujo de diseño establecido, ha resultado sencillo evaluar el impacto de las cuestiones de diseño que intervienen en estas versiones.

Al realizarse la medida de los parámetros característicos de las síntesis físicas resultantes se ha puesto de relieve la importancia que tiene el haber establecido los mecanismos para determinar la influencia que presentan estas cuestiones de diseño. Esta relevancia se ha manifestado en la gran dispersión de prestaciones y en la naturaleza heterogénea de las características que han resultado en los núcleos generados. El análisis cuantitativo ha permitido establecer la relación entre el diferente comportamiento de las propiedades estudiadas y las distintas estrategias de diseño empleadas.

Esta flexibilidad lograda en los modelos no sólo facilita alterar de forma puntual la microarquitectura y las evaluaciones correspondientes sino que también permite:

- alterar el funcionamiento de determinadas instrucciones,
- ampliar o reducir el juego de instrucciones,
- incluir módulos adicionales para realizar nuevas operaciones.

El disponer de un modelo de procesador con el que lograr materializaciones de altas prestaciones permite plantearse campos de aplicaciones muy diversos, donde la flexibilidad que aporta este tipo de elementos es de vital importanacia.

Las principales conclusiones obtenidas pertenecen a cuatro ámbitos distintos:

1. Optimizaciones microarquitecturales de SPARC,
2. Pautas de modelado eficiente,
3. Flujo de diseño para integrar núcleos IP en SOC, y
4. Caracterización de los diseños físicos y su relación con el modelado y el flujo de diseño.

A continuación detallamos estas conclusiones.

6.1.1. Microarquitectura SPARC

Tipo de Registros de segmentación

Los registros de segmentación son probablemente uno de los ejemplos más significativos del cuidado especial que necesitan ciertos elementos de la arquitectura. Se ha observado cómo su diseño tiene un impacto considerable en las prestaciones que se pueden conseguir.

En el caso del diseño de la IU de SPARC v. 8 se han elegido para estos registros de segmentación *flip-flops* tipo E. Esta opción es la que se ha estimado recomendable cuando se desea realizar un diseño seguro de un núcleo RISC. La seguridad funcional en la generación de núcleos empotrados sintetizados es relevante.

Con una distribución cuidada de estos registros especiales dentro del modelo gráfico RTL de la Ruta de Datos, el diseñador puede tener, en todo momento, una visión rápida de cómo se ha organizado la segmentación de la arquitectura. En el caso del modelado de la IU de SPARC, estos registros se han dispuesto en determinadas posiciones verticales, de modo que se pueda determinar rápidamente si una determinada operación sobre una unidad se desarrolla en la primera, segunda o n-ésima etapa de ejecución de cada instrucción. Esta visión estructural ha facilitado grandemente la optimización de núcleos.

Para estos registros es necesario permitir o impedir su carga dependiendo de que la máquina esté en operación normal o detenida. Se debe contemplar la posible modificación de la implementación de estos registros para cumplir especificidades concretas que podrían requerirse a estos elementos. La señal de reloj debe alcanzar a estos elementos de almacenamiento en estado puro. Es decir, no debe existir circuitería en medio de las líneas de distribución de esta señal tan crítica que es el reloj, ya que podría producir *skews* (desfase de reloj) indeseados. Utilizando los *flip-flops* tipo E, se puede controlar de modo aceptable la distribución de esta señal de reloj empleando las herramientas de CAD disponibles, de modo que se evita el mal funcionamiento. Esta situación es todavía más crítica en diseños de baja potencia al aparecer nuevos condicionantes. Los *flip-flops* tipo E cumplen bien con estas necesidades.

Por otro lado, con los *flip-flops* tipo E se evita establecer restricciones especiales para el ciclo de trabajo de la señal de reloj del circuito, a la vez que se mantiene un diseño claro y fácilmente gestionable, uno de los aspectos claves para la síntesis de sistemas empotrados.

Saltos condicionales. Predicción de salto

Se ha analizado la sensibilidad del diseño con la política de predicción de saltos condicionales. Hay 4 situaciones distintas, según la predicción haya sido saltar o no saltar, y según la evaluación posterior determine que la predicción sea buena o mala.

En aquellas situaciones en las que la decisión fue mala se debe anular la ejecución de la instrucción que se buscó en la predicción. Además, debido a la especificación de

la arquitectura, ocasionalmente se puede requerir la anulación de la instrucción en la ranura de tiempo.

En el presente caso se ha obtenido una arquitectura más o menos optimizada según el índice de éxito que se tenga en las predicciones, y siempre que el establecimiento de esta política no afecte al tiempo de ciclo. Para ello se ha integrado en el flujo de síntesis el correspondiente análisis de los saltos condicionales que se encuentran en los programas de aplicación específica del núcleo. El caso estudiado ha sido el procesado de vídeo de bajo régimen binario y la decodificacción *software* de flujos MPEG.

Ciclos adicionales

Se ha aportado una estrategia nueva para optimizar la generación de ciclos de instrucción adicionales. En cada ciclo la decodificación se realiza teniendo en cuenta el estado de la máquina de control y el de otros elementos, como el contenido del *registro de instrucción*, y el de un nuevo *registro de control de pasos* donde se almacena el paso de instrucción que debe seguir en secuencia.

Esta estrategia se ha preferido a la empleada por otros diseñadores que se basan en la generación de instrucciones auxiliares. La razón de esto estriba en los siguientes aspectos:

- Si se introdujesen instrucciones auxiliares haría falta al menos un elemento con el que indicar que se tratan de instrucciones internas generadas por el procesador. Esto se debe a que, si el código que se emplea para esta instrucción auxiliar no coincide con ninguna de las indicadas en el manual de la arquitectura, entonces debería producirse una excepción en caso que se quisiera poner el código de una instrucción auxiliar desde el exterior. Y si esto no se hace así, supondría que las posibilidades de ampliación del juego de instrucciones se verían obstaculizadas por el número de instrucciones auxiliares que existiesen en el procesador. Esto es un serio inconveniente en un contexto de síntesis de núcleos para aplicaciones específicas.

- Aunque con la alternativa elegida se emplea un biestable más que con la opción de otros diseñadores, con la opción anterior hacen falta elementos con los que generar de forma automática el código de la instrucción auxiliar y elementos con los que indicar que posteriormente se debe recoger la instrucción producida internamente. Estos otros elementos podrían ser de mayor coste que el que se produce con un biestable adicional.

- Al emplear el registro de control de pasos los modelos quedan más sencillos, tanto desde el punto de vista de gestión, como de depurado y modificación por el diseñador o el usuario del núcleo.

6.1.2. Recomendaciones de modelado

Si bien existen recomendaciones concernientes al modelado en VHDL de diseños digitales en general, debido a la naturaleza de este tipo de diseños, y con las pautas anteriormente expuestas, se han obtenido nuevas recomendaciones adicionales:

- Utilización de una única señal de reloj y sólo uno de sus flancos para todos los *flip-flops* disparados por flanco, y utilización de un solo nivel para los elementos de memoria activos por nivel mientras sea posible. Esto producirá un diseño completamente síncrono con un esquema de temporización sencillo. No se recomienda la utilización de distintas señales de reloj ya que no sólo puede ocasionar problemas arquitecturales o demandar esquemas de temporización rígidos —con los inconvenientes que ocasionaría a la hora de realizar extensiones o modificaciones—, sino que además resulta menos inmune a posibles problemas de ruido cuando se utilizan en entornos hostiles. Además, se produciría mayor longitud de interconexiones, aumentando el riesgo de problemas por retrasos en la señal de reloj o *skew*.

- No se deben utilizar elementos de memoria internos en los elementos fundamentales excepto los explícitos como memorias, ficheros de registros, registros de propósito general y similares. Es recomendable denotar explícitamente los elementos de memoria como tales y hacerlos visibles en el diagrama. De este modo el diseñador obtiene un mayor control sobre el proceso final de síntesis, y facilita el tratamiento de cualquier incongruencia que pueda haber realizado en el modelado de bloques del núcleo.

- En concordancia con el punto anterior, no se deben utilizar variables de almacenamiento en el proceso de generación de señales de control, y en caso de utilizarlas deben aparecer en la lista de sensibilidad. Asimismo todas las señales de entrada de este bloque deben aparecer en esta lista de sensibilidad.

- Es altamente recomendable la inclusión de algunas estimaciones simples de retardos en el modelado de las subunidades especialmente para los elementos de almacenamiento activos por nivel. De este modo se puede tener una mejor visión de cómo se van a comportar los módulos generados con otras herramientas en el contexto de nuestro diseño. Estos retardos se ignoran en el proceso de síntesis pero son recomendables para propósitos de depuración.

- Deben utilizarse los tipos de paquetes estándares mientras sea posible, y abstenerse de utilizar tipos definidos *ad hoc*. Esto permite que los modelos sean fáciles de comprender, gestionar, depurar y modificar por otros usuarios. Una excepción específica hay que hacerla con los estados de la FSM central, donde la definición de un tipo enumerado para ellos facilita su depuración y comprensión.

- Los nombres de los elementos del diseño no deben ser excesivamente largos, si bien se recomienda que sean claros para cualquier futuro usuario de estos modelos. Esta recomendación es crítica sobre todo cuando se produce el intercambio de la información del diseño con otras herramientas que imponen restricciones en cuando al tamaño de estos nombres.

6.1.3. Flujo de diseño

Se ha creado un flujo de diseño (fig. 4.27) idóneo para la generación de núcleos procesadores y su integración en sistemas empotrados. Este flujo incorpora herramientas comerciales, *scripts* de ayuda y técnicas de análisis de sensibilidad. Como se puede observar, la entrada de usuario inicial es el desarrollo de una librería completa de modelos en VHDL de los componentes básicos que formarán parte de los niveles jerárquicos más bajos del diseño. Estos elementos de librería no se cambiarán. Para obtener altas prestaciones, algunos módulos de la arquitectura considerados como críticos en tiempo y/o área se podrán obtener de un generador de módulos. En el presente trabajo se han utilizado los de la herramienta Epoch de *Cascade Design Automation* para este particular. Una vez que se han obtenido los modelos de todos los módulos básicos, éstos se interconectan dentro de una herramienta de esquemáticos. De esta forma el diseñador siempre puede tener una visión intuitiva de la organización y distribución de las operaciones a través de las representaciones gráficas que suponen estos esquemáticos. En el presente desarrollo se han empleado las del SGE de *Synopsys*. Es en este punto donde se deben establecer las decisiones microarquitecturales. Esta etapa es la más crítica, ya que las decisiones que se realizan en esta fase influyen seriamente en las prestaciones del sistema entero.

A partir del SGE se puede obtener los primeros modelos en VHDL del procesador. Tras unas primeras simulaciones con ellos, los resultados deben evaluarse meditadamente. Si no son satisfactorios, se debe verificar si los problemas son de tipo microarquitectural o si son debidos a un mal modelado de alguno de los módulos. En el primer caso, los cambios se pueden realizar sobre el esquemático empleando el entorno gráfico. Si no, entonces el problema debe localizarse en el modelo de los componentes de la librería afectados y corregirse.

Cuando los resultados de estas simulaciones funcionales cumplan las especificaciones, el siguiente paso a seguir es la síntesis lógica con las herramientas de Synopsys, empleando las librerías de síntesis que Cascade proporciona. Tras realizar algunos ajustes sobre restricciones dentro de la herramienta, se evalúan los resultados de esta síntesis lógica. Si no son del agrado del diseñador ahora los cambios hay que realizarlos o bien en la etapa de descripción o bien nuevamente en la etapa de síntesis lógica.

Una vez completada la síntesis lógica, el próximo paso es la síntesis física con la herramienta Epoch de *Cascade Design Automation*, empleando librerías físicas. En caso de no obtener los resultados deseados los problemas pueden deberse tanto a un establecimiento inadecuado de restricciones en la etapa de síntesis lógica como de comportamientos imprevistos en los bloques físicos. En ambos casos se hace necesario volver a alguno de los pasos anteriores. La verificación final consiste en una simulación sobre el *layout* final, donde la funcionalidad global debe coincidir con la de los primeros modelos que se obtuvieron en este flujo de diseño. Si los resultados no son satisfactorios podemos volver a realizar cualquiera de los apartados anteriores tras los cambios adecuados y según convenga. En este paso se toman las medidas de prestaciones que caracterizan la totalidad de núcleos generados. Este conjunto de núcleos plenamente caracterizados constituyen el espacio final de diseño.

Ayudas a las transiciones en el flujo de diseño

En las evoluciones desde unos elementos de este flujo de diseño a otros normalmente se precisa la intervención del usuario. En ocasiones esta intervención conlleva la modificación de ficheros que resultan del procesamiento del diseño en un determinado paso para que puedan ser interpretados por pasos posteriores. En otros casos la intervención del usuario puede ocasionar la preparación del entorno para que el paso siguiente se pueda realizar sin problemas.

Para facilitar muchas de estas situaciones se ha elaborado un conjunto de *scripts* (muchos de ellos en Perl 5) que facilitan este tipo de transiciones.

Toma de medidas

Debido a la naturaleza de las herramientas en las que se fundamenta el entorno, esta toma de medidas no se ha podido automatizar, no se ha podido *instrumentar*. Este es un tema de investigación abierto. Sin embargo sí se ha utilizado una sistemática sencilla para la obtención de medidas y su análisis mediante tablas, gráficos y frecuencias estadísticas, que permite un completo análisis de la sensibilidad del diseño, medida por las variaciones en prestaciones, respecto a las decisiones de diseño tomadas a diferentes niveles.

6.1.4. Análisis de los núcleos caracterizados

Como se ha indicado, la clave de este trabajo es poder generar rápida y flexiblemente esos núcleos y caracterizarlos, para analizar la incidencia que tienen sobre las prestaciones las opciones arquitecturales, microarquitecturales, estructurales, lógicas, de agrupación de bloques, de colocado, de test, de conexión con otros núcleos o componentes, y finalmente las opciones tecnológicas.

El análisis de resultados de las opciones microarquitecturales puede agruparse de dos formas. En primer lugar, estudiando el impacto no teórico o simulado sino real de las opciones sobre las prestaciones medidas en los *layouts* físicos implementados. En segundo lugar, estudiando en qué medida la mejora relativa de cada opción *resiste* hasta el final del proceso de implementación, es decir, en qué medida la opción es *dominante* respecto a algún parámetro de prestaciones, y de ahí conocer a ciencia cierta la importancia mayor o menor de considerar esa opción al nivel microarquitectural.

Se han estudiado las opciones de predicción de salto, de gestión de códigos de condición, de modo de cálculo de la dirección de búsqueda siguiente, de los mecanismos posibles de *bypass*, de las estructuras de módulos y de operadores de la ruta de datos.

Los experimentos demuestran que las opciones de cálculo de la dirección de la siguiente instrucción es dominante en velocidad, y en menor medida las de predicción de salto. Son óptimas las opciones de *dos sumadores de direcciones* y *predecir el salto como que se tomará cuando es hacia atrás*. El resto de opciones sufren una gran dispersión en la implementación. En general cuanto más se *delegan* las decisiones sobre cada uno de

los elementos que influyen en el secuenciamiento, más rápida resulta la implementación.

Las experiencias sobre el grado de granularidad lógica y uso de módulos de ruta de datos demuestran que un empleo razonable de módulos en zonas concretas del diseño suele mejorar las prestaciones en velocidad y área. Sin embargo, las versiones con mayor granularidad lógica, y más módulos, no continúan la mejora de velocidad, ni de área, ni de consumo de potencia. Existen, por tanto, combinaciones *medias* óptimas que es preciso explorar. Otro dato importante es la significativa variación en prestaciones que aparecen entre las versiones, su gran dispersión.

Las experiencias sobre granularidad física y definición de grupos demuestran que este nivel de agrupación y jerarquización física, de uso frecuente para *guiar* el colocado y ruteado, raramente compensa. El área y el consumo de potencia aumentan. Es notable, sin embargo, que la velocidad no se deteriore tanto como el aumento de área hace prever. En este caso, el proceso de deterioro queda frenado por la mayor proximidad entre puertas funcionalmente relacionadas, obtenida con el guiado de la herramienta. El número de transistores tampoco empeora mucho. Este resultado podría conducir al diseñador no experimentado a efectuar modificaciones lógicas jerárquicas alegremente: su impacto a veces no es perceptible en transistores pero es muy grande en velocidad y área.

Otro efecto similar se obtiene estableciendo a 1 el atributo `fixedblock` de los bloques jerárquicos. Ni el número de transistores, ni la velocidad quedan apenas modificados, pero sí el resto de parámetros. El efecto es tanto mayor cuanto más *gruesa* es la granularidad física.

La extensión del juego de instrucciones y de la ruta de datos para soportar operaciones de procesado de señal (MAC, DSP) y multimedia (SIMD, VIS) tiene unas notables repercusiones en todos los parámetros. Potencia y número de transistores crecen quizás más drásticamente que el área. El tiempo de ciclo aumenta significativamente —entre un 20 y un 40 %—, por lo que sólo se justifican estas extensiones cuando el campo de aplicación, por la composición de las cargas de trabajo de los programas, así lo demanden, y en este caso parece claro que cuanto mayor sea la longitud del paralelismo vectorial SIMD del procesador (8 operandos frente a 4, etc.) mejor se compensa en el *throughput* la mayor latencia física provocada. Por el contrario, el costo de estas extensiones en cuanto a su control y gestión es muy pequeño. Estos resultados experimentales son coherentes con otros estudios teóricos [PM98, Asa98].

Finalmente la influencia del mapeado y síntesis en diferentes tecnologías sobre todas las opciones anteriores es también obviamente muy importante. Sin embargo, es importante señalar que el peso relativo de las mejoras de cada opción en una tecnología no tiene porqué mantenerse en general en otra, incluso cambia de signo y de ser una opción 'positiva' puede pasar a ser 'negativa' para las prestaciones.

Los resultados de esta tesis matizan el aforismo clásico entre arquitectos: 'cuanto más altos sean los niveles de concepción y flujo de diseño de un circuito en los que se introducen mejoras, mayores son los beneficios obtenidos'. Lo matizan en el sentido de que la libertad de síntesis y mapeado tecnológico aportada por los entornos de síntesis, y el gran impacto de la microarquitectura, diseño lógico y diseño físico en este proceso,

hacen más necesaria una estrecha unión y entrelazamiento entre el proceso de concepción y proyectación arquitectural y el de implementación física. La relevancia de muchas decisiones microarquitecturales queda sustancialmente transformada, condicionada o filtrada por los niveles inferiores del proceso de implementación, de forma que no se pueden tomar esas decisiones sin tener en consideración estos efectos de filtrado. Este efecto va en aumento debido a la creciente importancia del ruteado y las interconexiones frente a la lógica, según decrece la geometría de los procesos.

6.2. Líneas futuras

Muy diversas son las líneas de investigación y cuestiones que han quedado abiertas con la ejecución de este trabajo.

En el futuro nos proponemos ir afrontando algunos de estos temas:

- *Instrumentar* la toma de medidas para la caracterización de los núcleos IPs y componentes virtuales sintetizados.
- Desarrollo e integración de módulos y coprocesadores especialmente adaptados para la aceleración de tareas específicas.
- Desarrollo e integración de unidades de gestión de memoria y memorias caché, con estudio de las técnicas más apropiadas para efectuar su gestión dependiendo del campo de aplicación al que se orienten los diseños.
- Aumentar la capacidad de procesamiento vectorial SIMD del juego de instrucciones con más operadores y operandos tanto en el banco de registros de enteros como en el de coma flotante.
- Desarrollo de un entorno de programación adecuado para la obtención fácil y sencilla del software necesario para sistemas multimedia de bajo régimen binario y de bajo coste, basados en estos núcleos SPARC con VIS.
- Estudio e integración de técnicas de superescalaridad y paralelismo que permitan mayor aceleración con diferentes unidades que pueden integrarse en el procesador y con coprocesadores.
- Estudiar la idoneidad de las nuevas extensiones de Cadence para integrar en este entorno el flujo de diseño de este trabajo.
- Avanzar en técnicas de integración en otras tecnologías como GaAs, donde el disponer de una versión de SPARC, aun siendo recortada, permitiría desarrollar sistemas de mayor flexibilidad que los realizados hasta ahora en este material. En particular, nos proponemos conectar los modelos y flujo de diseño desarrollado, con la herramienta de diseño físico en GaAs, Olympo, creada en el grupo [Mon94], y en el entorno Cadence.

Abreviaturas y acrónimos

A continuación se presenta una lista de las abreviaturas y acrónimos utilizados en el texto.

ALU ***A**rithmetic-**L**ogical **U**nit*, Unidad Aritmético-Lógica.

ASIC ***A**pplication **S**pecific **I**ntegrated **C**ircuit*, Circuito Integrado de Aplicación Específica.

ASIP ***A**pplication **S**pecific **I**ntegrated **P**rocessor*, Procesador Integrado de Aplicación Específica.

ASISP ***A**pplication **S**pecific **I**nstruction-**S**et **P**rocessor*, Procesador de Juego de Instrucciones de Aplicación Específica.

ASSP ***A**pplication **S**pecific **S**ignal **P**rocessor*, Procesador de Señal de Aplicación Específica.

CAD ***C**omputer **A**ided **D**esign*, Diseño Asistido por Ordenador.

CI Circuito Integrado.

CIF ***C**ommon **I**ntermediate **F**ormat*, Formato Intermedio Común.

CISC ***C**omplex **I**nstruction **S**et **C**omputer*, Computador de Juego de Instrucción Complejo.

CP ***Co**processor*, Coprocesador.

CPU ***C**entral **P**rocessing **U**nit*, Unidad Central de Proceso. Utilizado comúnmente para referirse a un *procesador*.

CWP ***C**urrent **W**indow **P**ointer*, Puntero de la Ventana Activa.

DSP *Digital Signal Processor*, Procesador Digital de Señales.

EDA *Electronic Design Automation*, Automatización del Diseño Electrónico.

I/O *Input/Output*, Entrada/Salida

IP *Intellectual Property*, Propiedad Intelectual.

IPR *Intellectual Property Rights*, Derechos de Propiedad Intelectual.

ISA *Instruction Set Architecture*, Arquitectura de Juego de Instrucciones.

IU *Integer Unit*, Unidad de Enteros.

FPGA *Field Programable Gate Array*, Matriz de Puertas Programable por Campo.

FPU *Floating Point Unit*, Unidad de Coma Flotante.

FSM *Finite State Machine*, Máquina de Estados Finita.

GSR *Graphics Status Register*, Registro de Estado de Gráficos.

HDL *Hardware Description Language*, Lenguaje de Descripción del *Hardware*.

ILP *Instruction Level Paralelism*, Paralelismo a Nivel de Instrucciones.

LAE *Layout Acceleration Environment*, Entorno de Aceleración de *Layout*.

LIW *Long Instruction Word*, Palabra de Instrucción Larga.

LSB *Less Significant Bit*, Bit Menos Significativo.

MAC *Multiplicate and Accumulate*, Multiplicar y Acumular.

MAX *Multimedia Acceleration Extensions*, Extensiones para Aceleración de Multimedia.

MCM *Multi-Chip Module*, Módulo Multichip.

MDMX *MIPS Digital Media Extensions*, Extensiones para Medios Digitales de MIPS.

MMU *Memory Management Unit*, Unidad de Gestión de Memoria.

MMX *Multimedia Extensions*, Extensiones para Multimedia.

MP@ML *Main Profile @ Main Level*, Perfil Principal en el Nivel Principal.

MPEG *Moving Pictures Environment Group*, Grupo para Entornos de Imágenes en Movimiento.

MSB *Most Significant Bit*, Bit Más Significativo.

MVI *Motion Video Instructions*, Instrucciones para Vídeo de Movimiento.

NSP *Native Signal Processor*, Procesador de Señal Innato.

OCM *Original Chip Manufacturers*, Fabricante de Chip Original.

OEM *Original Equipment Manufacturer*, Fabricante de Equipamiento Original.

PC *Program Counter*, Contador de Programa.

PCB *Printed Circuit Board*, Placa de Circuito Impresa.

PSR *Processor Status Register*, Registro de Estado del Procesador.

RASSP (*Rapid Prototyping of Application Specific Signal Processors*), Prototipado Rápido de Procesadores de Señal de Aplicación Específica.

RISC *Reduced Instruction Set Computer*, Computador de Juego de Instrucción Reducido.

RTL *Register Transfer Level*, Nivel de Transferencia de Registros.

SBC *Single Board Computer*, Ordenador de una Sola Placa.

SGE *Synopsys Graphical Environment*, Entorno Gráfico de Synopsys.

SIF *Source Intermediate Format*, Formato Intermedio de la Fuente.

SIMD *Single Instruction, Multiple Data*, Instrucción Única, Datos Múltiples.

SOAR *Smalltalk On A RISC*, Smalltalk en un RISC.

SOC *System on a Chip*, Sistema en un Chip.

SP@ML *Single Profile @ Main Level*, Perfil Único en el Nivel Principal.

SPARC *Scalable Processor Architecture*, Arquitectura de Procesador Escalable.

VC *Vision Controller*, Controlador de Visión.

VDSP *Video Digital Signal Processor*, Procesador Digital de Señales de Vídeo.

VHDL *Very-High-Speed-ICs Hardware Description Language*, Lenguaje de Descripción de *Hardware* de Circuitos Integrados de Muy Alta Velocidad.

VIS *Visual Instruction Set*, Juego de Instrucciones Visuales.

VLIW *Very Long Instruction Word*, Palabra de Instrucción Muy Larga.

VLSI *Very Large Scale of Integration*, Escala Muy Grande de Integración.

VP *Vision Processor*, Procesador de Visión.

VRP *Video RISC Processor*, Procesador RISC de Vídeo.

VSIA *Virtual Socket Interface Alliance.*

WAN *Wide Area Network*, Red de Área Amplia.

Bibliografía

[Ack93] B.D. ACKLAND. A video-codec chip set for multimedia applications. *AT&T Technical Journal* páginas 50–65 (enero 1993).

[Ack94] B.D. ACKLAND. The role of VLSI in multimedia. *IEEE Journal of Solid State Circuits* **29**(4) (1994).

[Ack95] B.D. ACKLAND. VLSI for multimedia applications. En "Proc. of 13th Australian Microelectronics Conference", páginas 23–33, Adelaida, Australia del Sur (julio 1995). The IREE Society.

[Aki94] T. AKIYAMA. MPEG2 video codec using image compression DSP. *IEEE Trans. on Consumer Electronics* **40**(3) (1994).

[AL99] J-F. AGAËSSE Y B. LAURENT. Virtual components application and customization. En "Proc. of DATE", páginas 726–727, Munich, RFA (1999). IEEE Computer Society.

[ANH+93] A. ALOMARY, T. NAKATA, Y. HONMA, M. IMAI Y N. HIKICHI. An ASIP instruction set optimization algorithm with functional module sharing constraint. En "Proc. of the International Conference on Computer-Aided Design", páginas 526–532 (1993).

[Aon92] K. AONO. A video digital signal processor with a vector-pipeline architecture. *IEEE Jornal of Solid-State Circuits* **27**(12) (1992).

[AOS94] J. ALTMEYER, S. OHNSORGE Y B. SCHUERMANN. Reuse of design objects in CAD frameworks. En "IEEE/ACM International Conference on Computer-Aided Design" (noviembre 1994).

[Ara94] T. ARAKI. Video DSP architecture for MPEG-2 codec. En "Proc. of the 1994 Conference on Acoustics, Speech and Signal Processing", tomo 2, páginas 417–420 (1994).

[Asa98] K. ASANOVIĆ. "Vector Microprocessors". Tesis Doctoral, University of California at Berkeley, Berkeley, CA (mayo 1998).

[Bai92] D. BAILEY. Programmable vision processor/controller. *IEEE Micro* **12**(5), 33–39 (octubre 1992).

[Bak98] S. BAKER. Virtual Socket Interface Alliance. En "Proc. of DATE", París, Francia (1998).

[BAMS98] J. BÖTTGER, K. AGSTEINER, D. MONJAU Y S. SCHULZE. An object-oriented model for specification, prototyping, implementation and reuse. En "Proc. of DATE", París, Francia (1998).

[Bau92] T. BAUTISTA. Diseño y evaluación de una arquitectura de CPU emuladora del TMS32010. Proyecto Fin de Carrera, ETSI de Telecomunicación, Universidad de Las Palmas de G.C. (abril 1992).

[BBA+98] B. BENHAM, B. BABBA, A. AMOURA, P. COEURDEVEY Y G. SAUCIER. "Technologies for the Information Society: Developments and Opportunities", capítulo A catalog generator for Electronic Virtual Components, páginas 474–481. IOS Press, Amsterdam, Holanda (1998).

[BBK+97] B. BENHAM, B. BABBA, H. KRUPNOVA, G. SAUCIER Y M. COURTOY. IPs simplify ASIC prototyping on FPGAs. En "IP'97 Europe", páginas 421–429 (octubre 1997).

[BBS98] B. BEHNAM, K. BABBA Y G. SAUCIER. IP taxonomy, IP searching in a catalog. En "Proc. of DATE", París, Francia (1998). Designer Track.

[BCF+97] J.C. BAUER, É. CLOSSE, É. FLAMMAND, M. POIZE, J. PULOU Y P. PENIER. SAXO: A retargetable optimized compiler for DSPs. En "Proc. of ICSPAT" (1997).

[BdKK+99] J-Y. BRUNEL, E.A. DE KOCK, W.M. KRUIJTZER, H.J.H.N. KENTER Y W.J.M. SMITS. Communication refinement in video systems on chip. En "Proc. of the Seventh International Workshop on Hardware/Software Codesign", páginas 142–146. The Association for Computing Machinery (mayo 1999).

[Bie95] J. BIER. DSP processors and cores: The options multiply. *Integrated System Design Magazine* (julio 1995).

[Bjo96] G. BJONTEGAARD. Very low bitrate videocoding using H.263 and foreseen extensions. En "Proc. of ECMAST'96", páginas 825–838, Louvain-la-Neuve, Bélgica (mayo 1996). Université Catholique de Louvain.

[BK95] V. BHASKARAN Y K. KONSTANTINIDES. "Image and Video Compression Standards". Kluwer (1995).

[BKS98] B. BENHAM, H. KRUPNOVA Y G. SAUCIER. Block and IP wrapping for efficient design on FPGAs. En "FPGA'98", Monterey, USA (febrero 1998).

[BMCN96] T. BAUTISTA, G. MARRERO, P.P. CARBALLO Y A. NÚÑEZ. Towards a low-cost processor architecture for multimedia. En J. FIGUERAS, editor, "Proc. of the XI Conference of Design of Integrated Circuits and Systems", páginas 445–450. Universitat Politècnica de Catalunya, CPDA, Diagonal, 647, E-08028 Barcelona, Spain (noviembre 1996).

[BMCN97] T. BAUTISTA, G. MARRERO, P.P. CARBALLO Y A. NÚÑEZ. Rapid-prototyping of high-performance RISC cores with VHDL. En CAPT. GREG PETERSON Y DR. PHILIP WILSEY, editores, "Rapid Systems Prototyping with VHDL", páginas 43–52, Arlington, VA (octubre 1997). VHDL International, IEEE Computer Society.

[BN99a] T. BAUTISTA Y A. NÚÑEZ. Design of efficient SPARC cores for embedded systems. En "Proc. of EUROMICRO Workshop on Digital System Design", Milán, Italia (1999). Aceptado para publicación.

[BN99b] T. BAUTISTA Y A. NÚÑEZ. Flexible design of SPARC cores: A quantitative study. En "Proc. of the 7th International Workshop on Hardware/Software Codesign", páginas 43–47, Roma, Italia (mayo 1999). IEEE Computer Society, IEEE Circuits and Systems Society, IFIP WG 10.5, ACM SIGSOFT y ACM SIGDA, The Association for Computing Machinery.

[BNCS92] T. BAUTISTA, A. NÚÑEZ, P.P. CARBALLO Y R. SARMIENTO. Compilación y realización de una versión con células estándares del TMS32010. En "VII Congreso de Diseño de Circuitos Integrados", páginas 371–376 (noviembre 1992).

[BS97] B. BENHAM Y G. SAUCIER. An IP catalog: an important gateway into the world IP business. En "1st Euripides Workshop", París, Francia (septiembre 1997).

[BSC+97] F. BALARIN, E. SENTOVICH, M. CHIODO, P. GIUSTO, H. HSIEH, B. TABBARA, A. JURECSKA, L. LAVAGNO, C. PASSERONE, K. SUZUKI Y A. SANGIOVANNI-VINCENTELLI. "Hardware-Software Codesign of Embedded Systems – The Polis Approach". Kluwer Academic Publishers (1997).

[Bur93] D. BURSKY. Codec compresses image in real time. *Electronic Design* páginas 123–124 (octubre 1993).

[But95] G.H. BUTTNER. Setting up a retrieval system for design reuse — experiences and acceptance. En "Proc. of EDAC", páginas 575–578 (1995).

[BW90] A. BURNS Y A. WELLINGS. "Real-Time Systems and their Programming Languages". International Computer Sciences Series. Addison-Wesley Publishing Ltd., Workingham, England (1990).

[Cas97] Cascade Design Automation Corporation, Bellevue, WA. "Epoch User's Manual, ver. 4.0.1" (1997).

[Cat91] B.J. CATANZARO, editor. "The SPARC Technical Papers". Springer Verlag (1991).

[CCu97] C-Cube Microsystems. "One-to-One Digital Video: Integrating Encoding and Decoding Technology on One Chip" (1997). <http://www.c-cube.com/>.

[CGK] G. CÔTÉ, M. GALLANT Y F. KOSSENTINI. "Efficient Motion Vector Estimation and Coding for H.263-Based Very Low Bit Rate Video Compression".

[Cha97] P. CHAMBERS. The ten commandments of excellent design: VHDL code examples. *Electronic Design* páginas 123–126 (abril 1997).

[CM99a] S. CHAKRAVARTY Y G. MARTIN. A new embedded system design flow based on IP integration. En "DATE User Forum", páginas 99–103, Munich, RFA (1999). IEEE Computer Society.

[CM99b] F. CHAROT Y V. MESSÉ. A flexible code generation framework for the design of application specific programmable processors. En "Proc. of the Seventh International Workshop on Hardware/Software Codesign", páginas 27–31. The Association for Computing Machinery (mayo 1999).

[CRS+98] R. CMAR, L. RIJNDERS, P. SCHAUMONT, S. VERNALDE Y I. BOLSENS. A methodology and design environment for DSP ASIC fixed point refinement. En "Proc. of DATE", páginas 271–276, Munich, RFA (1998). IEEE Computer Society.

[CvC91] R. CARRERAS Y P. VON CLEMM. SPARC-prozessor für embedded control. *Elektronik* (13), 52–62 (1991).

[dAlo] V. DE ARMAS. "Aportaciones a la Síntesis de Circuitos Integrados en Tecnología GaAs". Tesis Doctoral, ETSI de Telecomunicación, Universidad de Las Palmas de G.C. (en desarrollo).

[DJ99] R.P. DICK Y N.K. JHA. MOCSYN: Multiobjective core-based single-chip system synthesis. En "Proc. of DATE", páginas 263–270, Munich, RFA (1999). IEEE Computer Society.

[dL96] J. DE LAMEILLIEURE. Scalable video coding for fast sequence previewing in multimedia browsing. En "Proc. of ECMAST'96", páginas 635–654, Louvain-la-Neuve, Bélgica (mayo 1996). Université Catholique de Louvain.

[DSP] "DSP Overview at Texas Instruments". <http://www.ti.com/sc/docs/products/dsp/overview.htm>.

[DWWO96] S. DUTTA, A. WOLFE, W. WOLF Y K.J. O'CONNOR. "VLSI Signal Processing, IX", capítulo Design issues for very-long-instruction-word (VLIW) video signal processors, páginas 95–104. IEEE Press (1996).

[Eck95] S. ECKART. ISO/IEC MPEG-2 software video codec. En "Proc. Digital Video Compression: Algorithms and Technologies", páginas 100–109. SPIE (enero 1995).

[Eno93] T. ENOMOTO. A 300 MHz 16 bit programmable video signal processor ULSI for a single chip teleconferencing system. En "Proc. of the 19th European Solid-State Circuits Conference (ESSCIRC'93)" (septiembre 1993).

[Ern97] R. ERNEST. "Hardware/Software Co-Design: Principles and Practice", capítulo Target Architectures. Kluwer Academic Publishers (1997).

[Ern98] R. ERNST. Codesign of embedded systems: Status and trends. *IEEE Design & Test of Computers* páginas 45–54 (abril 1998).

[Fai98] D. FAIRBANK. The VLSI Alliance: journey from vision to production. *Electronic Design* **46**(1), 86–92 (enero 1998).

[FDF98] P. FARABOSCHI, G. DESOLI Y J.A. FISCHER. The latest word in digital and media processing. *IEEE Signal Processing Magazine* **15**(2), 59–85 (marzo 1998).

[Fer98] J.M. FERNÁNDEZ. "Arquitecturas VLSI para la Codificación de Imágenes en Movimiento en Tiempo Real". Tesis Doctoral, ETSI de Telecomunicación, Universidad Politécnica de Madrid (mayo 1998).

[Fos99] R. FOSTER. Simplifying design of systems-on-a-chip with rapid silicon prototyping and reusable IP. En "DATE User Forum", páginas 147–151, Munich, RFA (1999). IEEE Computer Society.

[fro] "Frontier Design". `<http://www.frontierd.com/artlibrary.htm>`.

[GdM93] R.K. GUPTA Y G. DE MICHELI. Hardware-software cosynthesis for digital systems. *IEEE Design & Test of Computers* páginas 29–41 (septiembre 1993).

[GDWL91] D. GAJSKI, N. DUTT, A. WU Y S. LIN. "High-Level Synthesis: Introduction to Chip and System Design". Kluwer Academic Publishers (1991).

[Gup95] R.K. GUPTA. "Co-synthesis of Hardware and Software for Digital Embedded Systems". The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston/Dordrecht/Londres (1995).

[GVNG94] D. GAJSKI, F. VAHID, S. NARAYAN Y J. GONG. "Specification and Design of Embedded Systems". Prentice Hall (1994).

[GZ97] R. GUPTA Y Y. ZORIAN. Introducing core-based system design. *IEEE Design & Test* **14**(4) (octubre 1997).

[Haa99] J. HAASE. Design methodology for IP providers. En "Proc. of DATE", páginas 728–729, Munich, RFA (1999). IEEE Computer Society.

[Har87] R.W. HARTENSTEIN, editor. "Hardware Description Languages". Advances in CAD for VLSI. Elsevier Science Publishers B.V. (North-Holland) (1987).

[HD94] I.-J. HUANG Y A.M. DESPAIN. Generating instruction sets and microarchitectures from applications. En "Proc. of the International Conference on Computer-Aided Design", páginas 391–396 (1994).

[HGG+99] A. HALAMBI, P. GRUN, V. GANESH, A. KHAR, N. DUTT Y A. NICOLAU. EXPRESSION: A language for architecture exploration through compiler/simulator retargetability. En "Proc. of the DATE Conference" (1999).

[HP96] J.L. HENESSY Y D.A. PATTERSON. "Computer Architecture: A Quantitative Approach". Morgan Kaufmann Publishers, Inc., 2nd edición (1996).

[HYTU96] E. HOLMANN, A. YAMADA, T. TOSHIDA Y S. URAMOTO. "VLSI Signal Processing, IX", capítulo Real-time MPEG-2 software decoding with a dual-issue RISC processor, páginas 105–114. IEEE Press (1996).

[IIT93] Integrated Information Technology, Inc. "Single Chip Video Codec and Multimedia Communications Processor" (1993). Preliminary Datasheet.

[Ike96] M. IKEDA. A hardware/software concurrent design for real-time SPML MPEG2 video-encoder chip set. En "Proc. of the EDTC", páginas 320–326 (1996).

[Ino93] T. INONUE. A 300-MHz 16-B BiCMOS video signal processor. *IEEE Journal of Solid-State Circuits* **28**(12) (dec 1993).

[Int] Intel, Co. "Using MMX Instruction to Compute the Absolute Difference in Motion Estimation".

[ISD] Integrated System Design. <http://www.isdmag.com/>.

[ISO90] International Organization for Standardization. "Coding of Moving Pictures and Associated Audio for Digital Storage Media up to about 1.5 Mb/s" (diciembre 1990). ISO/IEC International Standard 11172.

[ISO94] International Organization for Standardization. "ISO 9001: Quality systems – Model for quality assurance in design, development, production, installation and servicing" (1994).

[ITU90] International Telecommunicaction Union. "Video Codec for Audio Visual Services at px64 kb/s" (1990). Recomendación H.261.

[JDKR97] A.A. JERRAYA, H. DING, P. KISSION Y M. RAHMOUNI. "Behavioral Synthesis and Component Reuse with VHDL". Kluwer Academic Publishers, Boston/Dordrecht/Londres (1997).

[Küç98] K. KÜÇÜKÇAKAR. Analysis of emerging core-based design lifecycle. En "Proc. of the International Conference on Computer-Aided Design", páginas 445–449 (1998).

[Küç99] K. KÜÇÜKÇAKAR. An ASIP design methodology for embedded systems. En "Proc. of Seventh International Workshop on Hardware/Software Codesign", páginas 17–21. The Association for Computing Machinery (mayo 1999).

[Kat85] M.G.H. KATEVENIS. "Reduced Instruction Set Computer Architecture for VLSI". The MIT Press (1985). ACM doctoral dissertation award 1984.

[KB98] M. KEATING Y P. BRICAUD. "Reuse Methodology Manual". Kluwer Academic Publishers, Boston/Dordrecht/Londres (1998).

[KCG+98] K. KÜÇÜKÇAKAR, C.T. CHEN, J. GONG, W. PHILIPSEN Y T.E. TKACIK. Matisse: An architectural design tool for commodity ICs. *IEEE Design and Test of Computers* **15**(2), 22–33 (abril 1998).

[KCGW98] M. KOEGST, P. CONRADI, D. GARTE Y M. WAHL. A systematic analysis of reuse strategies for design of electronic circuits. En "Proc. of DATE", París, Francia (1998).

[KDVvdW97] B. KIENHUIS, E. DEPRETTER, K. VISSERS Y P. VAN DER WOLF. An approach for quantitative analysis of application-specific dataflow architectures. En "Proc. of ASAP'97", jul (1997).

[KMT+95] L. KOHN, G. MATURANA, M. TRMBLAY, M. PRABHU Y G. ZYNER. The Visual Instruction Set (VIS) in UltraSPARC™. En "UltraSPARC Microprocessor Whitepapers". SPARC Technology Business (1995).

[Kon96] T. KONDO. Two-chip MPEG-2 video encoder. *IEEE Micro* páginas 51–58 (1996).

[KP98] C.E. KOZYRAKIS Y D.A. PATTERSON. A new direction for computer architecture research. *Computer* **31**(11), 24–32 (noviembre 1998).

[KPS+99] H.J.H.N. KERNTER, C. PASSERONE, W.J.M. SMITS, Y. WATANABE Y A.L. SANGIOVANNI-VINVENTELLI. Designing digital video systems: Modeling and scheduling. En "Proc. of the Seventh International Workshop on Hardware/Software Codesign", páginas 64–68. The Association for Computing Machinery (mayo 1999).

[Lap95] P. LAPSEY. NSP shows promise on the Pentium and PowerPC. *Microprocessor Reports* (8), 11–15 (1995).

[Lap97] V. LAPPALAINEN. Implementation of H.263 video encoder using Intel MMX technology. Proyecto Fin de Carrera, Tampere University of Technology (1997).

[LBSL97] P. LAPSEY, J. BIER, A. SHOHAM Y E.A. LEE. "DSP Processor Fundamentals". IEEE Press (1997).

[Lee94] B.W. LEE. Data flow processor for multi-standard video codec. En "Proc. of IEEE 1994 Customs Integrated Circuits Conference", páginas 103–106 (1994).

[Lee95] R.B. LEE. Accelerating multimedia with enhanced processors. *IEEE Micro* (abril 1995).

[Lee96] R.B. LEE. Subword parallelism with MAX-2. *IEEE Micro* **16**(4), 51–59 (agosto 1996).

[Lie97] C. LIEM. "Retargetable Compilers for Embedded Core Processors: Methods and Experiences in Industrial Applications". Kluwer Academic Publishers (1997).

[LM98] R. LEUPERS Y P. MARWEDEL. Retargetable code generation based on structural processor descriptions. *Design Automation for Embedded Systems* **3**(1), 1–36 (enero 1998).

[LPCJ95] C. LIEM, P. PAULIN, M. CORNERO Y A. JERRAYA. Industrial experience using rule-driven retargetable code generation for multimedia applications. En "Proc. of the International Symposium on System Synthesis" (septiembre 1995).

[LSI96] "L64110/120/130 VISC Encoder Chipset" (1996). LSI Press Release.

[LSVH96] L. LAVAGNO, A. SANGIOVANNI-VINCENTELLI Y H. HSIEH. "Embedded System Codesign: Synthesis and Verification". Kuwer Academic Publishers (1996).

[LvPK+95] D. LANNEER, J. VAN PARET, A. KIFFI, K. SCHOOFS, W. GEURTS, F. THOEN Y G. GOOSENS. "Code Generation for Embedded Processors", capítulo CHESS: Retargetable Code Generation for Embedded DSP Processors. Kluwer Academic Publishers (1995).

[LWMG96] G. LEHMANN, B. WUNDER Y K.D. MÜLLER-GLASER. A VHDL reuse workbench. En "Proc. of EURO-DAC", Ginebra, Suiza (1996).

[Mad95] V.K. MADISETTI. Virtual prototyping of embedded dsp systems. En "Proc. of IEEE International Conference Acoustics, Speech, and Signal Processing" (1995).

[MGJK99] E. MALAVASI, D. GUILIN, K. JONES Y W. KAO. Layout acceleration for IC physical design. En "DATE User Forum", páginas 7–11, Munich, RFA (1999). IEEE Computer Society.

[MIP97] MIPS Technologies, Inc., <http://www.mips.com/>. "MIPS Extension for Digital Media with 3D" (marzo 1997).

[MLD92] P. MICHEL, U. LAUTHER Y P. DUZY, editores. "The Synthesis Approach to Digital System Design". The Kluwer International Series in Engieneering and Computer Science. CLSI, Computer Architecture and Digital Signal Processing. Kluwer Academic Publishers (1992).

[Mon94] J.A. MONTIEL. "Síntesis y compilación de células en tecnología GaAs". Tesis Doctoral, Universidad de Las Palmas de G.C. (jul 1994).

[MS98] G. MARTIN Y B. SALEFSKI. Methodology and technology for design of communications and multimedia products via system level IP integration. En "Proc. of DATE", páginas 11–18, París, Francia (1998). Designer Track.

[MT97] J.D. MELLOT Y F. TAYLOR. Very long instruction word architectures for digital signal processing. En "Proc. of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)" (1997).

[Núñ95] A. NÚÑEZ. Tradeoffs in VLSI architectures for high performance signal processing. En W. MARWOOD, editor, "Proc. of the 13th Australian Microelectronics Conference", páginas 123–135, Adelaida, Australia del Sur (1995). The IREE Society.

[NC89] A. NÚÑEZ Y D. CARNAL. MVM: a GaAs microprocessor for critical real-time applications. *Microprocessing and Microprogramming* (25), 289–298 (1989). North Holland Elsevier Science Publishers.

[OAIP98] S. OLCOZ, L. AYUDA, I. IZAGUIRRE Y O PENALBA. VHDL teamwork, organization units and workspace management. En "Proc. of DATE", París, Francia (1998).

[OGN99] S. OLCOZ, A. GUTIÉRREZ Y D. NAVARRO. A SPARC μprocessor: from HDL to silicon. En "DATE User Forum", páginas 29–35, Munich, RFA (1999). IEEE Computer Society.

[OS90] A.V. OPPENHEIM Y R.W. SCHAFER. "Discrete-Time Signal Processing". Prentice-Hall, Inc. (1990).

[Pat93] K. PATEL. Performance of a software MPEG video decoder. En "Proc. 1st ACM Intl. Conf. on Multimedia", páginas 75–82. ACM (agosto 1993).

[Pay] B. PAYNE. "Rapid Silicon Prototyping: Paradigm for Custom System-on-a-Chip Design". VLSI Technology, Inc. <http://www.vlsi.com/velocity>.

[Pen85] J.M. PENDLETON. A design methodology for VLSI processors. Memorandum UCB/ERL M85/88, Electronics Research Laboratory, College of Engineering, University of California, Berkeley (noviembre 1985).

[Per99] D.L. PERRY. High density FPGA design techniques. En "DATE User Forum", páginas 111–118, Munich, RFA (1999). IEEE Computer Society.

[PGLM94] J. VAN PRAET, G. GOOSENS, D. LANNEER Y H. DE MAN. Instruction set definition and instruction selection for ASIPs. En "Proc. of the 7th International Symposium on High-level Synthesis", páginas 11–16 (1994).

[PH94a] D.A. PATTERSON Y J.L. HENNESSY. "Computer Organization & Design: The Hardware/Software Interface". Morgan Kaufmann Publishers, Inc. (1994).

[PH94b] D.A. PATTERSON Y J.L. HENNESSY. "Computer Organization and Design: The Hardware-Software Interface". Morgan Kaufmann Publishers, Inc., San Francisco, CA (1994).

[PHC99] F. POGODALLA, R. HERSEMEULE Y P. COULOMB. Fast prototyping: a system design flow for fast design, and efficient IP reuse. En "Proc. of the Seventh International Workshop on Hardware/Software Codesign", páginas 69–73. The Association for Computing Machinery (mayo 1999).

[PHSMR95] V. PREIS, R. HENFTLING, M. SCHÜTZ Y S. MÄRZ-RÖSSEL. A reuse scenario for the VHDL-based hardware design flow. En "Proc. of EURO-DAC", Brighton, Reino Unido (1995).

[Pir98] PETER PIRSCH, editor. "VLSI Implementations for Digital Signal Processing". John Wiley (1998).

[PJRL99] H.P. PEIXOTO, M.F. JACOME, A. ROYO Y J.C. LÓPEZ. The design space layer: Supporting early design space exploration for core-based designs. En "Proc. of DATE", páginas 676–683, Munich, RFA (1999). IEEE Computer Society.

[PM98] N. POLLARD Y D. MAY. Using interval arithmetic to calculate data sizes for compilation to multimedia instruction sets. En "Proc. of ACM Multimedia '98", Bristol, Reino Unido (septiembre 1998). The Association for Computing Machinery.

[PS99] J. PLANTIN Y E. STOY. Aspects on system-level design. En "Proc. of the Seventh International Workshop on Hardware/Software Codesign", páginas 209–210. The Association for Computing Machinery (1999).

[Pur98] S. PURCELL. The impact of Mpact 2. *IEEE Signal Processing Magazine* **15**(2), 102–107 (marzo 1998).

[PW96] A. PELEG Y U. WEISER. MMX technology extension to the Intel architecture. *IEEE Micro* (agosto 1996).

[PW97] A. PELEG Y U. WEISER. MMX technology extension to the Intel architecture. *IEEE Micro* páginas 42–50 (agosto 1997).

[PWW97] A. PELEG, S. WILKIE Y U. WEISER. Intel MMX for multimedia PCs. *Communcations of the ACM* **40**(1), 25–38 (enero 1997).

[Qui99] O. QUINTERO. Diseño e implementación de un procesador SPARC con juego de instrucciones visuales VIS. Proyecto Fin de Carrera, ETSI de Telecomunicación, Universidad de Las Palmas de G.C. (mar 1999).

[RAS] RASSP, Rapid Prototyping of Application Specific Signal Processors. <http://rassp.scra.org/>.

[RCS99] R. RAFIDINITRIMO, P. COEURDEVEY Y G. SAUCIER. IP key features and an object oriented IP catalog management system. En "DATE User Forum", páginas 105–109, Munich, RFA (1999). IEEE Computer Society.

[Ric96] D.S. RICE. High-performance image processing using special-purpose CPU instructions: The UltraSPARC Visual Instruction Set. Proyecto Fin de Carrera, University of California, Berkeley (marzo 1996).

[RK96] K. RÖNNER Y J. KNEIP. Architecture and applications of the HiPAR video signal processor. *IEEE Trans. on Circuits and Systems for Video Technology* **6**(1) (febrero 1996).

[RP96] J.M. RABAEY Y M. PEDRAM, editores. "Low Power Design Methodologies". Kluwer Academic Publishers (1996).

[RR99] A. REUTTER Y W. ROSENSTIEL. An efficient reuse system for digital circuit design. En "Proc. of DATE", Munich, RFA (marzo 1999). IEEE Computer Society.

[RS98] S. RATHNAM Y G. SLAVENBURG. Processing the new world of interactive media: The Trimedia VLIW CPU architecture. *IEEE Signal Processing Magazine* **15**(2), 108–117 (marzo 1998).

[RSV97] J. ROWSON Y A. SANGIOVANNI-VINCENTELLI. Interface-based design. En "Proc. of the 34th DAC", páginas 178–183 (1997).

[RSV98] J. ROWSON Y A. SANGIOVANNI-VINCENTELLI. Felix initiative pursues new codesign methodology. *Electronic Engineering Times* páginas 50,51,74 (junio 1998).

[SDS96] E.J. STOLLNITZ, T.D. DEROSE Y D.H. SALESIN. "Wavelets for Computer Graphics: Theory and Applications". Morgan Kaufmann Publishers, Inc. (1996).

[See99] R. SEEPOLD. Virtual Socket Interface Alliance. En "Proc. of DATE", página 182, Munich, RFA (1999). IEEE Computer Society.

[Ses98] N. SESHAN. High VelociTI processing. *IEEE Signal Processing Magazine* **15**(2), 86–101, 117 (marzo 1998).

[Sin94] P. SINANDER. VHDL modelling guidelines. Informe técnico ASIC/001, Automation and Informatics Department, European Space Agency/ESTEC, The Nederlands (septiembre 94).

[SK95] W. SUNG Y K. KUM. Simulation-based word-length optimization methid for fixed-point digital signal processing systems. *IEEE Transactions on Signal Processing* **43**, 3087–3090 (diciembre 1995).

[SL97] P. SODERQUIST Y M. LEESER. Optimizing the data cache performance of a software MPEG-2 video decoder. En "Proc. of ACM Multimedia '97", Seattle, Washington, USA (noviembre 1997). The Association for Computing Machinery.

[SM95] G.A. SHAW Y V.K. MADISETTI. Assessing and improving current practice in the design of application specific signal processors. *The RASSP Digest* 2(1) (enero 1995).

[Smi92a] MICHAEL R. SMITH. How RISCy is DSP. *IEEE Micro* (diciembre 1992).

[Smi92b] MICHAEL R. SMITH. To DSP or Not to DSP. *The Computer Applications Journal* (28), 14–24 (agosto 1992).

[SPA91] SPARC International, Inc. "The SPARC Architecture Manual, version 8" (1991).

[SPM98] "IEEE Signal Processing Magazine, The Latest Word in Multimedia", tomo 15, (1998).

[Sta88] W. STALLINGS. Reduced instruction set computer architecture. En "Proceedings of the IEEE", tomo 76, páginas 38–55 (enero 1988).

[Sun] Sun Microsystems, Inc. "The VIS Instruction Set". `<http://www.sun.com/microelectronics/vis/>`.

[Sun97] Sun Microsystems, Mountain View, CA 94043. "Visual Instruction Set (VIS) User's Guide" (marzo 1997).

[SVR+97] P. SCHAUMONT, S. VERNALDE, L. RIJNDERS, M. ENGELS Y I. BOLSENS. A programming environment for the design of complex high speed asics. En "Proc. of DAC", Anaheim (1997).

[Tem96] Temic Semiconductors. "TSC691E Integer Unit User's Manual" (1996).

[TF96] R. TALLURI Y B. FLINCHBAUGH. Video coding below twenty kilobits per second. En "Proc. of ECMAST'96", páginas 839–851, Louvain-la-Neuve, Bélgica (mayo 1996). Université Catholique de Louvain.

[Tho94] SGS-Thomson Microelectronics. "MPEG-2/CCIR 601 Video Decoder" (junio 1994). Notas preliminares.

[TK96a] L. TORRES Y M. KUNT. Second generation video coding schemes and their role in MPEG-4. En "Proc. of ECMAST'96", páginas 799–823, Louvain-la-Neuve, Bélgica (mayo 1996). Université Catholique de Louvain.

[TK96b] L. TORRES y M. KUNT. "Video coding: the second generation approach". Kluwer Academic Publishers, Englewood Cliffs (1996).

[TM96] L. MIGUEL L. TEIXEIRA y M. I. MARTINS. Video compression: The MPEG standards. En "Proc. of ECMAST'96", páginas 615–634, Louvain-la-Neuve, Bélgica (mayo 1996). Université Catholique de Louvain.

[TMS] "Texas Instruments TMS320C67". <http://www.ti.com/sc/docs/dsps/products/c67x/index.htm>.

[TONH96] M. TREMBLAY, J.M. O'CONNOR, V. NARAYANAN y L. HE. VIS speeds new media processing. *IEEE Micro* **16**(4), 10–20 (agosto 1996).

[Toy94] M. TOYOKURA. A video DSP with a macroblock-level-pipeline and a SIMD type vector-pipeline architecture for MPEG-2 CODEC. *IEEE Journal of Solid-State Circuits* **29**(12) (1994).

[vdWLG+99] P. VAN DER WOLF, P. LIEVERSE, M. GOEL, D. LA HEI y K. VISSERS. An MPEG-2 decoder case study as a driver for a system level design methodology. En "Proc. of the Seventh International Workshop on Hardware/Software Codesign", páginas 33–37. The Association for Computing Machinery (mayo 1999).

[VG99] F. VAHID y T. GIVARGIS. The case for a configure-and-execute paradigm. En "Proc. of the Seventh International Workshop on Hardware/Software Codesign", páginas 59–63. The Association for Computing Machinery (mayo 1999).

[VIS] Sun Microsystems, Inc. "The Visual Instruction Set: on-chip support for new media processing". Whitepaper 95-022, <http://www.sun.com/sparc/vis>.

[VSI] Virtual Socket Interface Association Architecture Document. <http://www.vsi.org>.

[VSI97a] Virtual Socket Interface Alliance. "Virtual Socket Interface Architecture Document" (marzo 1997). <http://www.vsi.org/library/vsi-or.pdf>.

[VSI97b] Virtual Socket Interface Alliance. "VSI Alliance Roadmap, version 1.0" (1997). <http://www.vsi.org/library/road3.pdf>.

[WCS96] L. WALL, T. CHRISTIANSEN y R.L. SWARTZ. "Programming PERL". O'Reilly & Associates, Inc. (1996).

[WG94] D.L. WEAVER y T. GERMOND, editores. "The SPARC Architecture Manual, version 9". Prentice Hall (1994).

[WKV+96] J. WILBERG, A. KUTH, H.-T. VIERHAUS, R. CAMPOSANO y W. ROSENTIEL. A design exploration environment. En "Proc. of the 6th Great Lakes Symposium on VLSI", páginas 77–80 (1996).

APÉNDICE A

Layouts de las versiones

En las figuras de la A.1 hasta la A.89 se presentan los *layouts* de las distintas versiones que se han llevado a cabo. Cada una de ellas representa una implementación específica de los puntos del espacio de diseño y cuyas caracterizaciones se han presentado en los apartados anteriores.

A.1. Versiones de la serie `aa7xx`

Estas versiones se han realizado con 7 ventanas, con la opción `fixedblock` puesta a 0 y permitiendo a la herramienta que realice la partición de forma completamente automática.

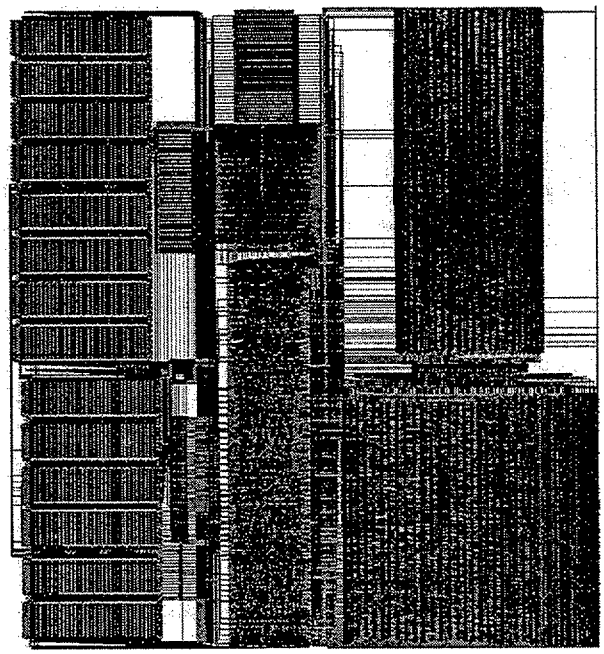


FIGURA A.1: *Layout* de la versión aa702

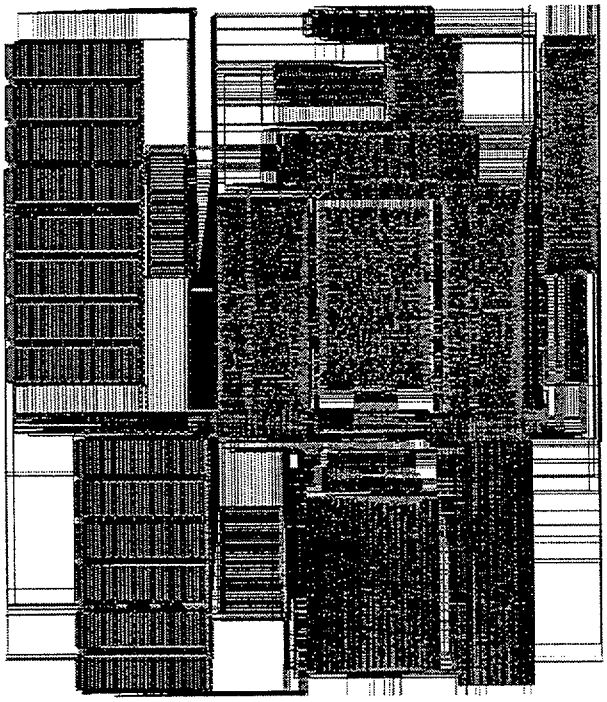


FIGURA A.2: *Layout* de la versión aa703

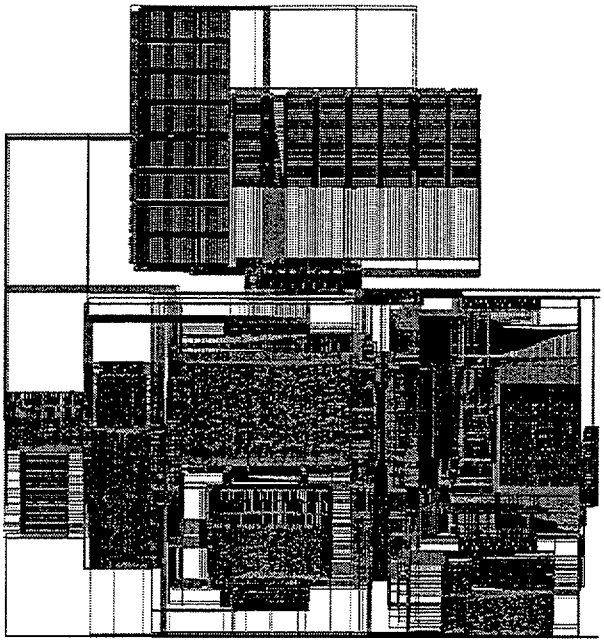


FIGURA A.3: *Layout* de la versión aa704

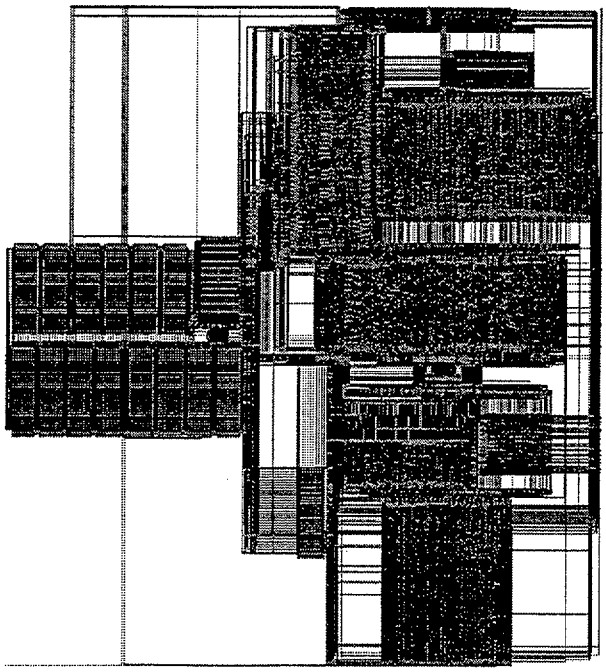


FIGURA A.4: *Layout* de la versión aa712

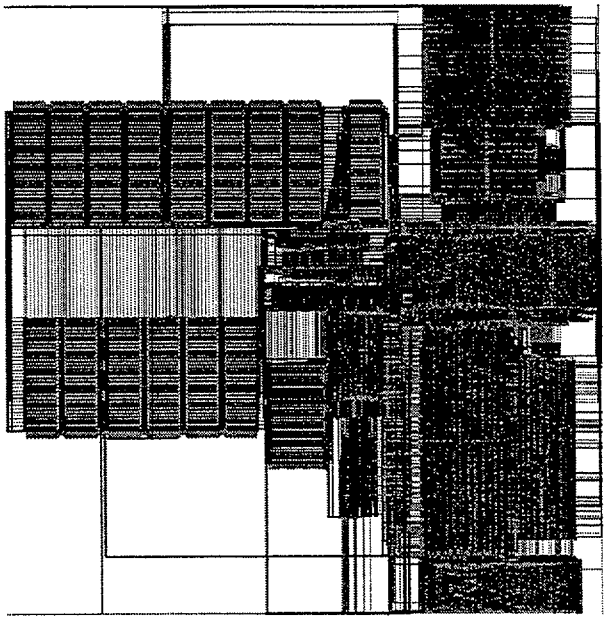


FIGURA A.5: *Layout* de la versión aa713

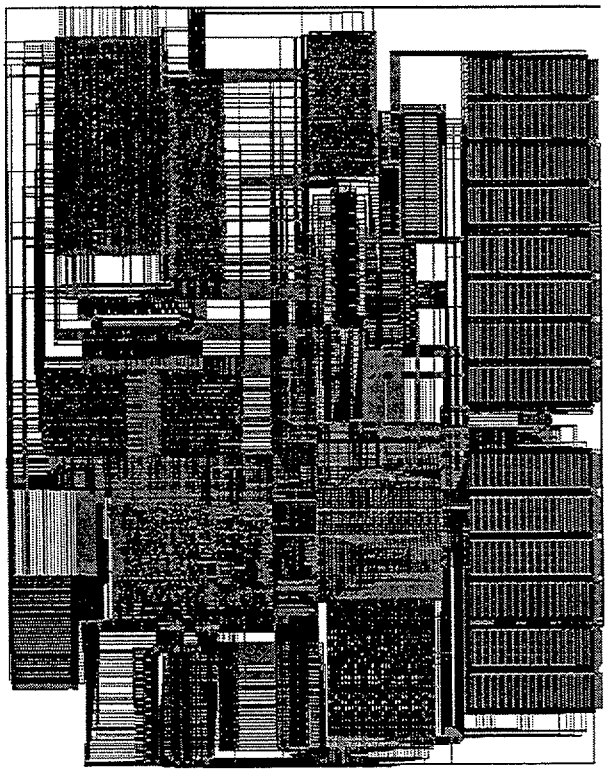


FIGURA A.6: *Layout* de la versión aa714

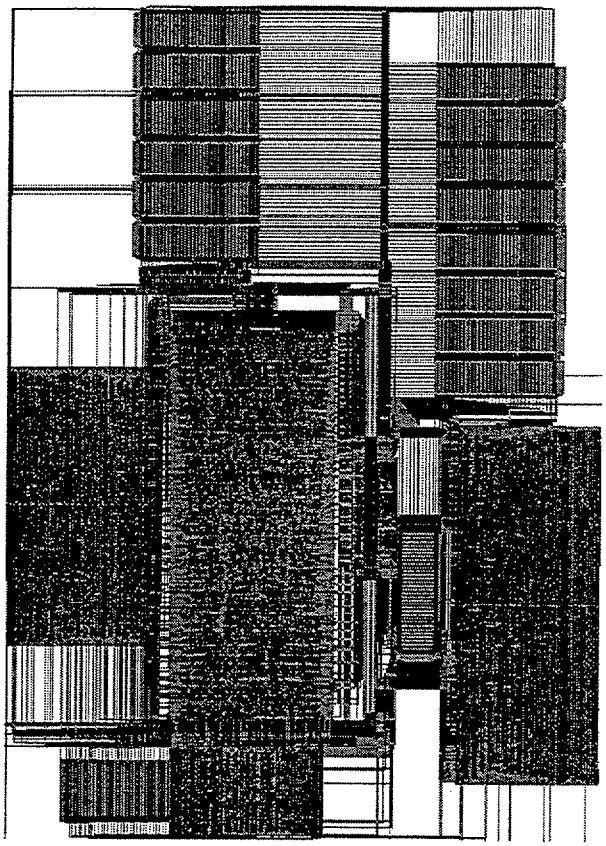


FIGURA A.7: *Layout* de la versión aa722

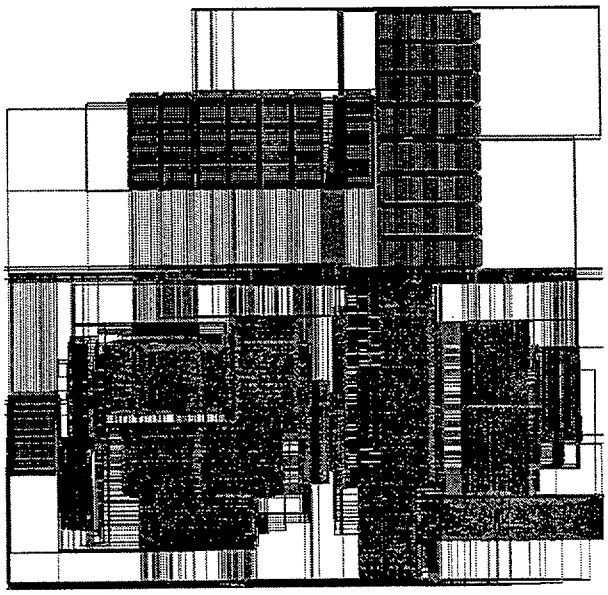


FIGURA A.8: *Layout* de la versión aa723

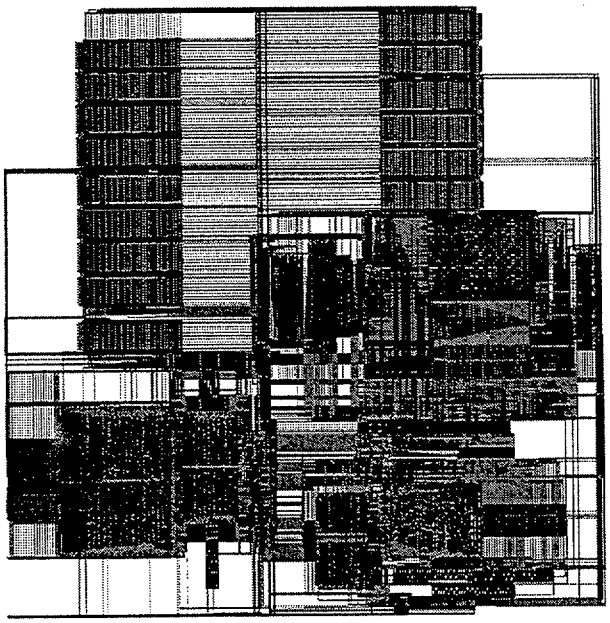


FIGURA A.9: *Layout* de la versión aa724

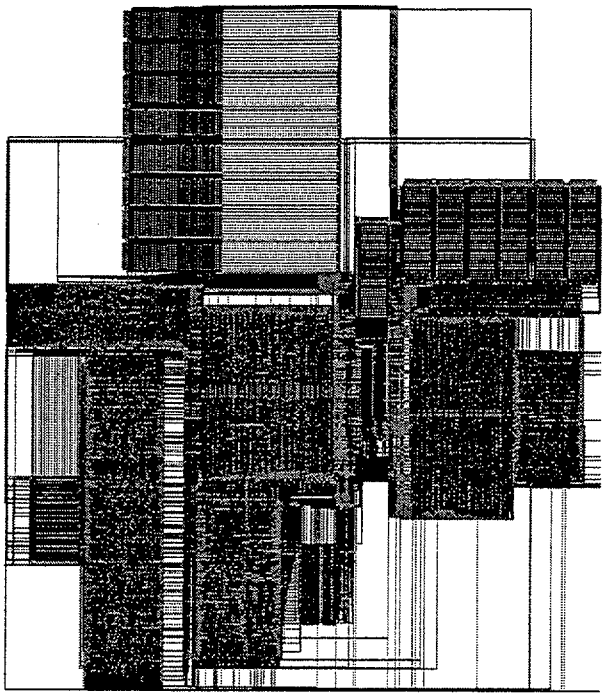


FIGURA A.10: *Layout* de la versión aa732

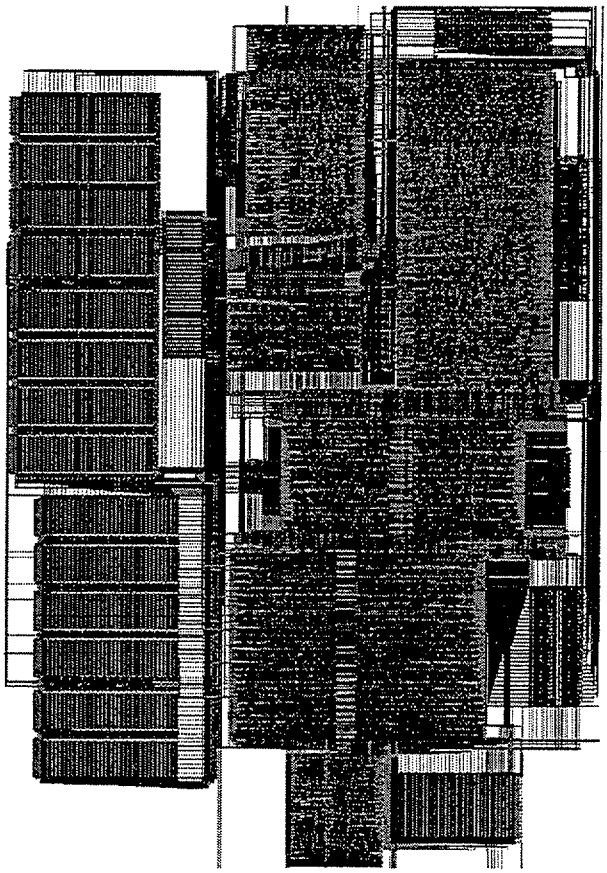


FIGURA A.11: *Layout* de la versión aa733

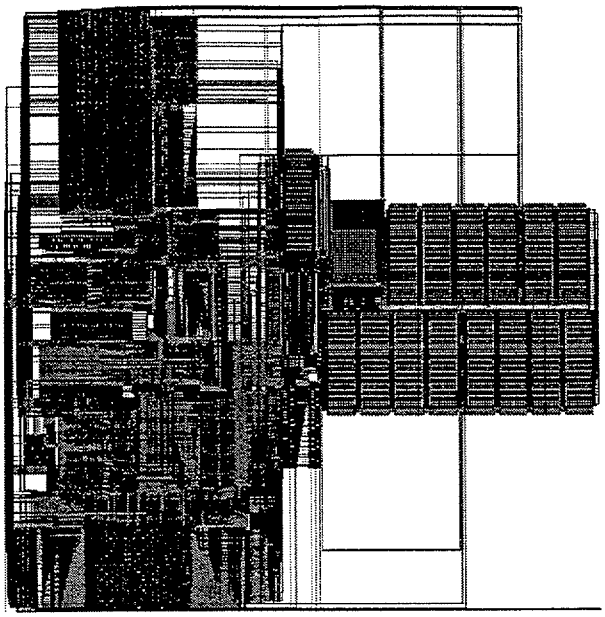


FIGURA A.12: *Layout* de la versión aa734

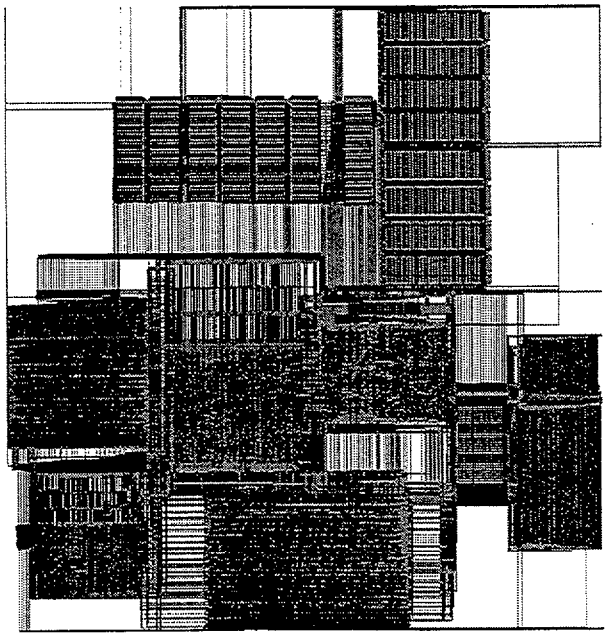


FIGURA A.13: *Layout* de la versión aa742

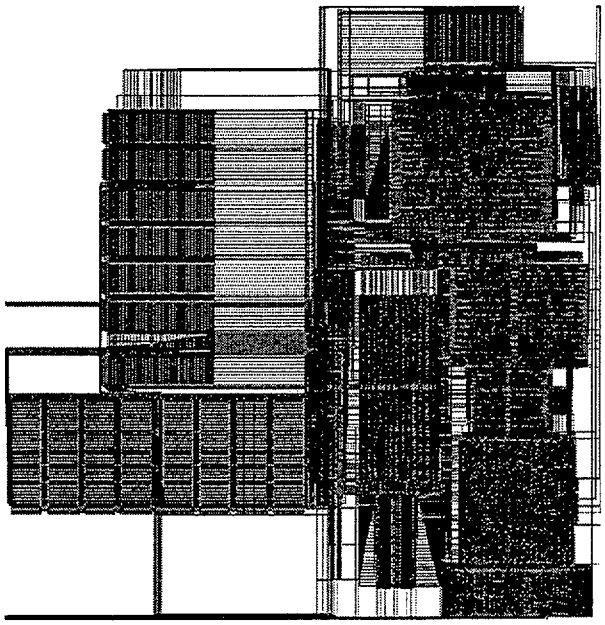


FIGURA A.14: *Layout* de la versión aa743

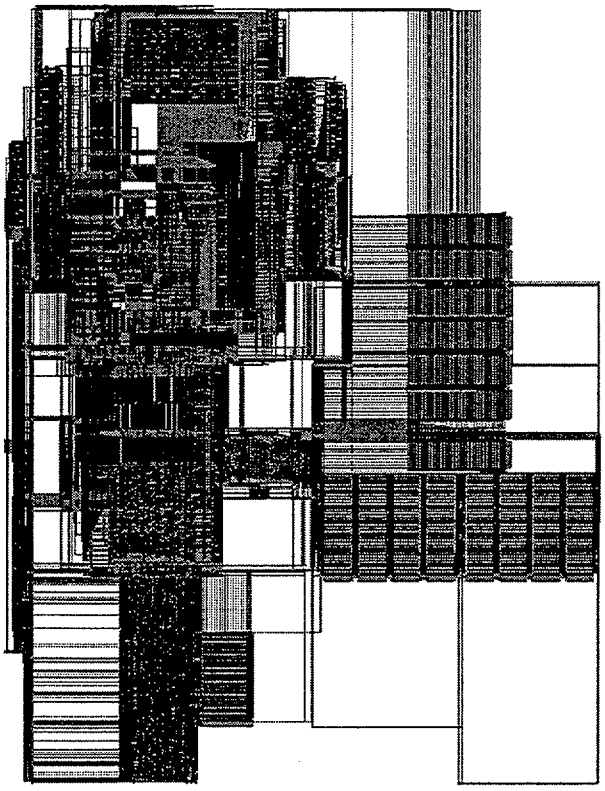


FIGURA A.15: *Layout* de la versión aa744

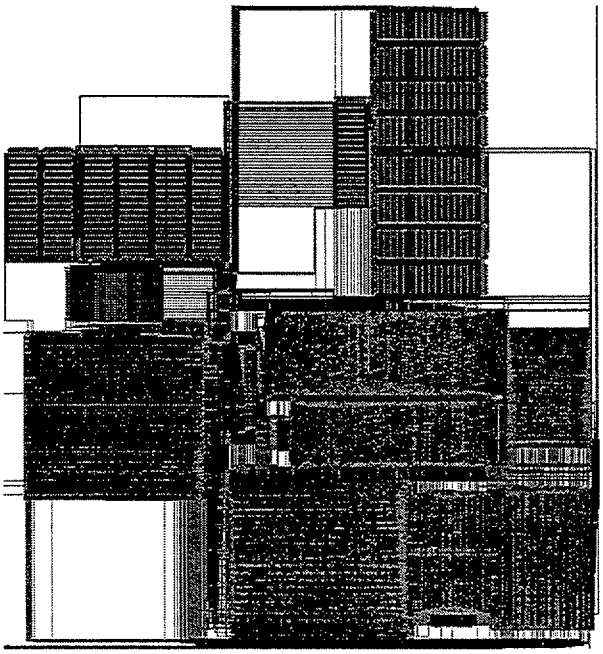


FIGURA A.16: *Layout* de la versión aa752

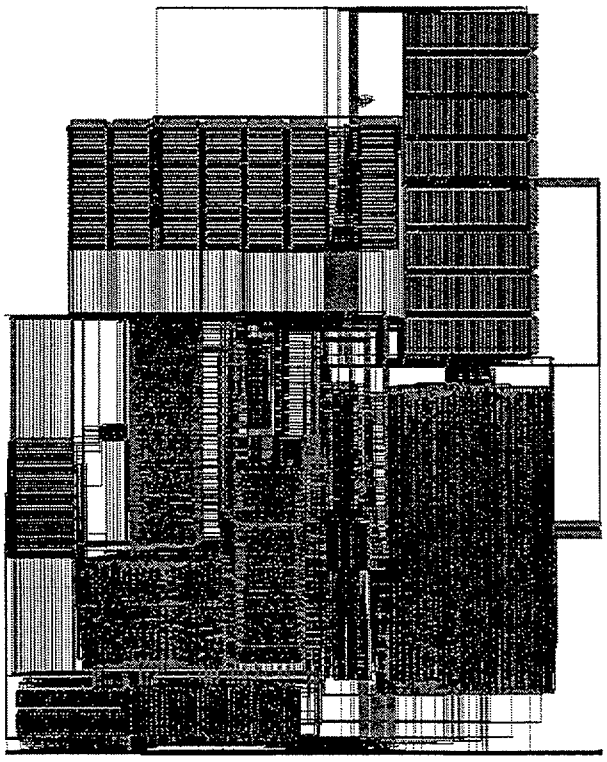


FIGURA A.17: *Layout* de la versión aa753

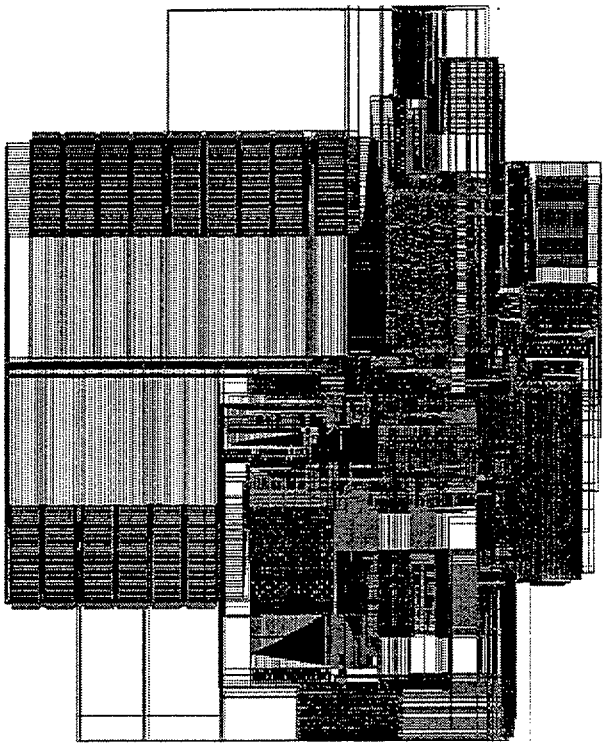


FIGURA A.18: *Layout* de la versión aa754

A.2. Versiones de la serie ba7xx

Las versiones de esta serie se han realizado con 7 ventanas, con la opción `fixedblock` puesta a 0 y guiando a la herramienta con partición realizada de forma manual.

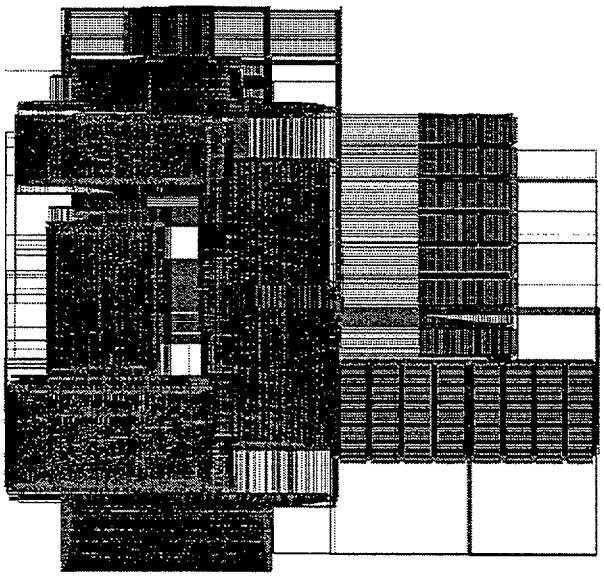


FIGURA A.19: *Layout* de la versión aa762

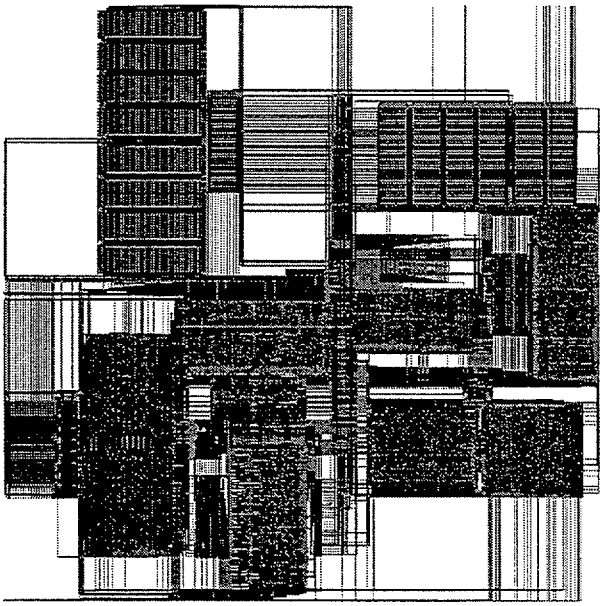


FIGURA A.20: *Layout* de la versión aa763

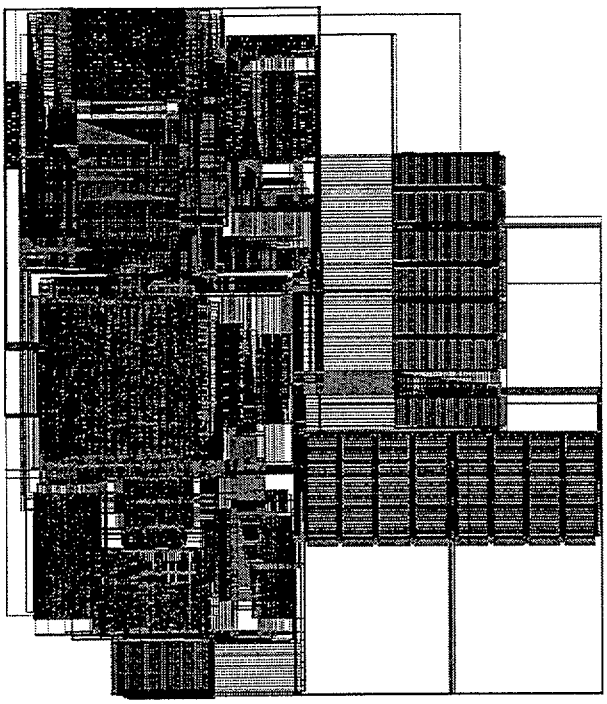


FIGURA A.21: *Layout* de la versión aa764

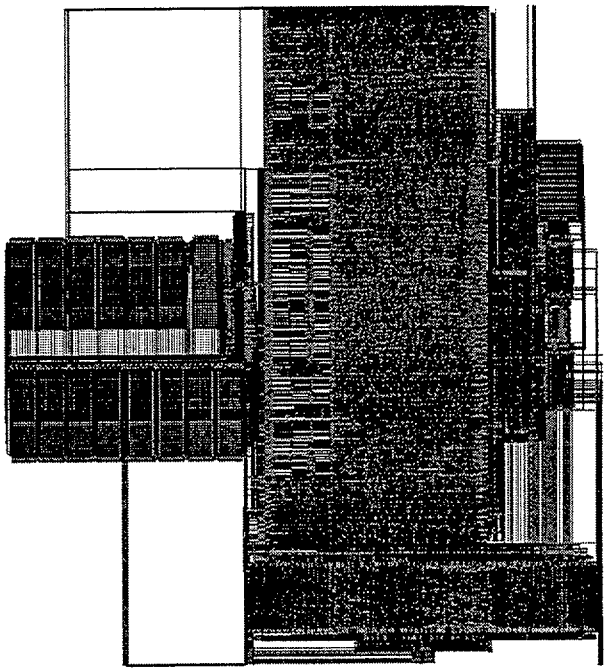


FIGURA A.22: *Layout* de la versión ba702

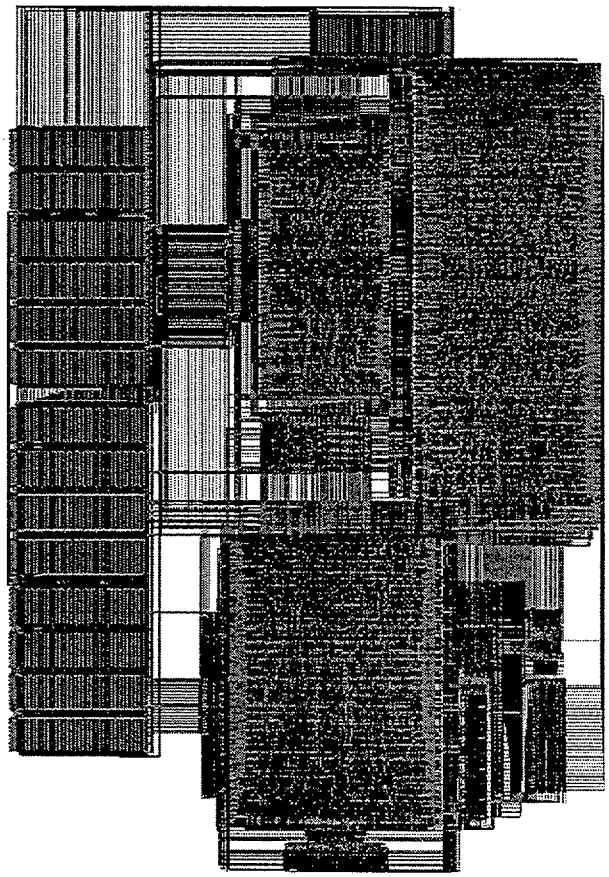


FIGURA A.23: *Layout* de la versión ba703

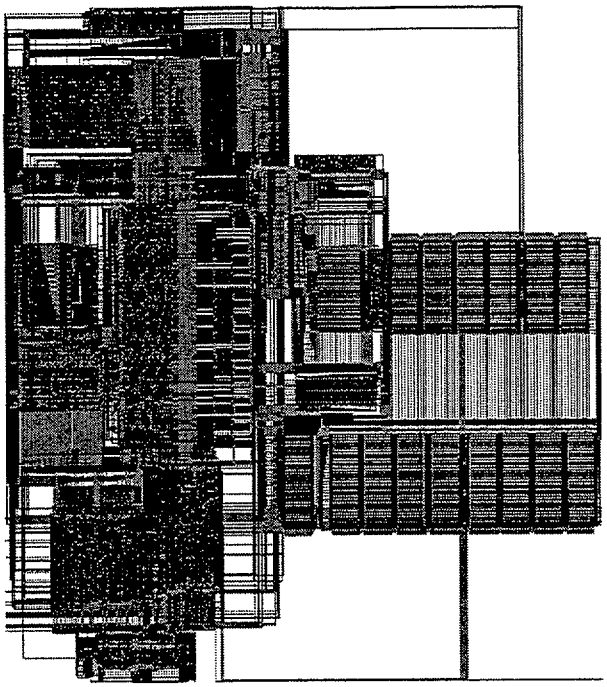


FIGURA A.24: *Layout* de la versión ba704

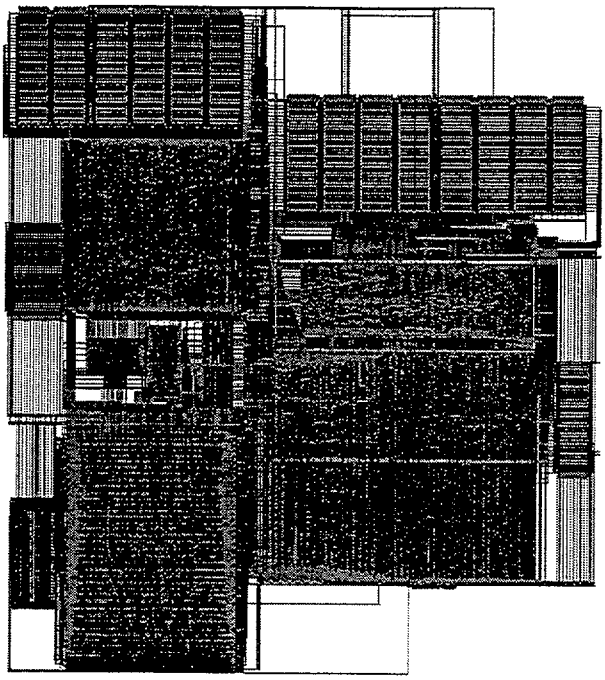


FIGURA A.25: *Layout* de la versión ba712

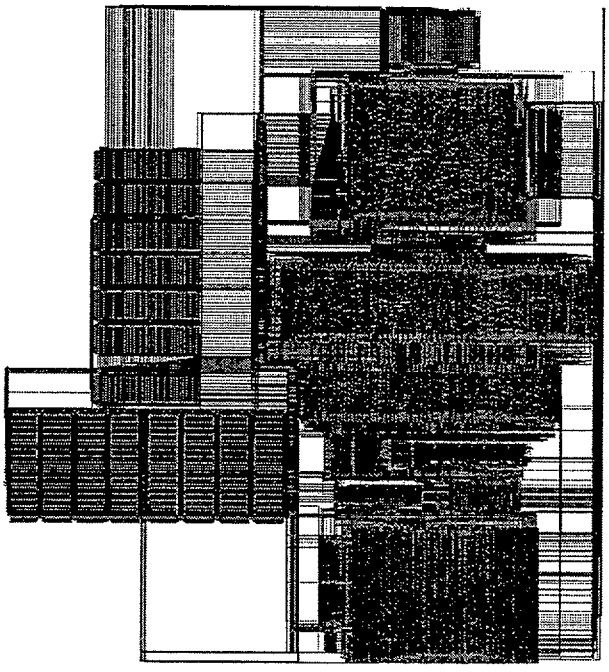


FIGURA A.26: *Layout* de la versión ba713

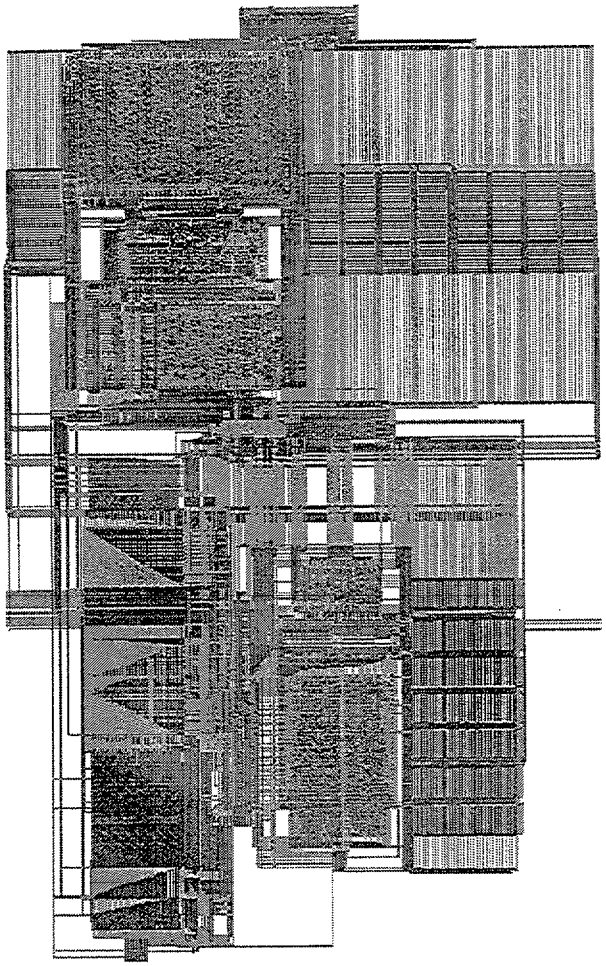


FIGURA A.27: *Layout* de la versión ba714

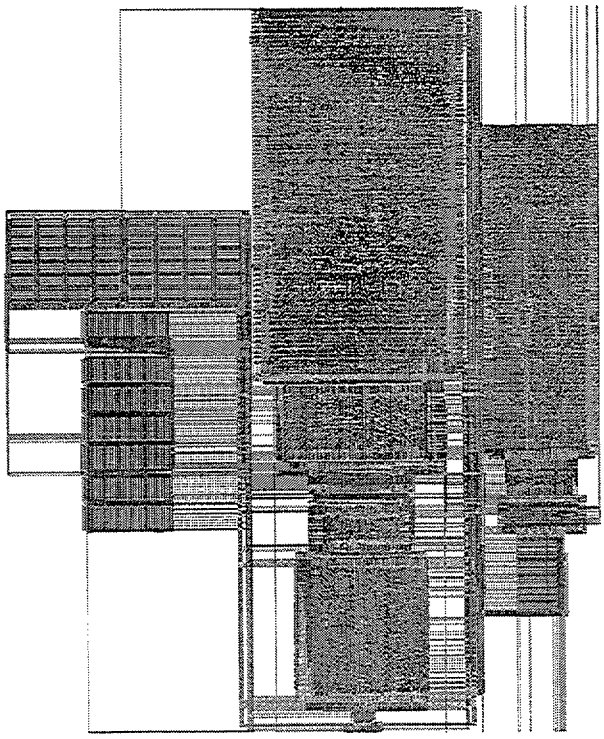


FIGURA A.28: *Layout* de la versión ba722

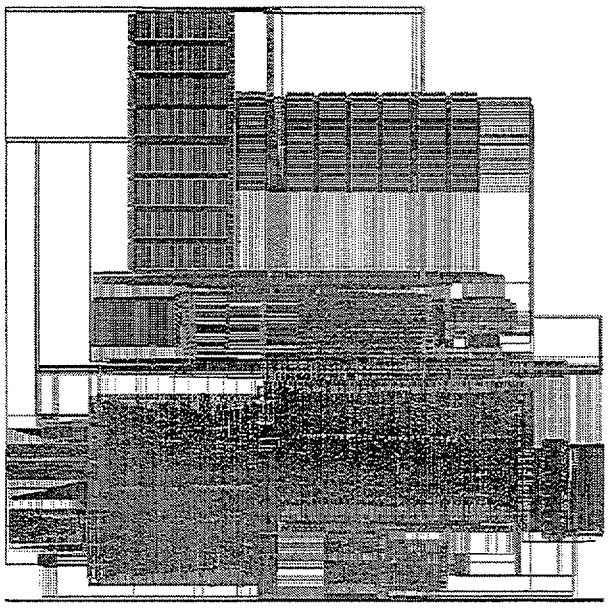


FIGURA A.29: *Layout* de la versión ba723

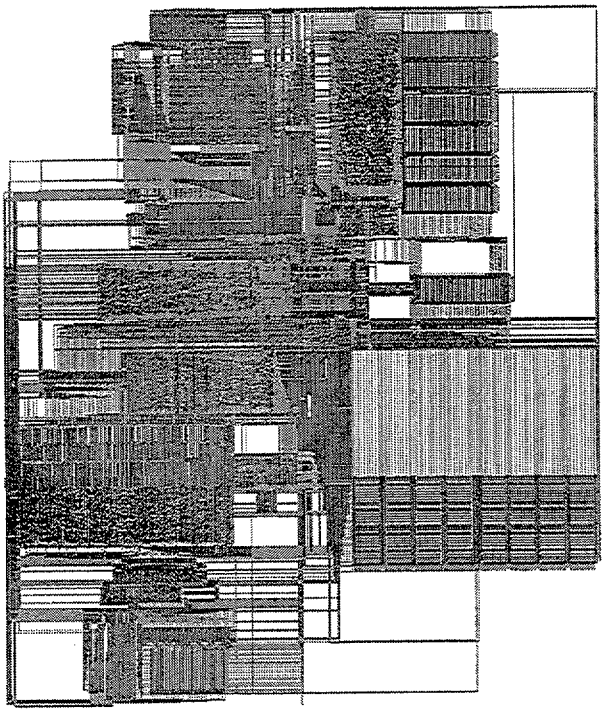


FIGURA A.30: *Layout* de la versión ba724

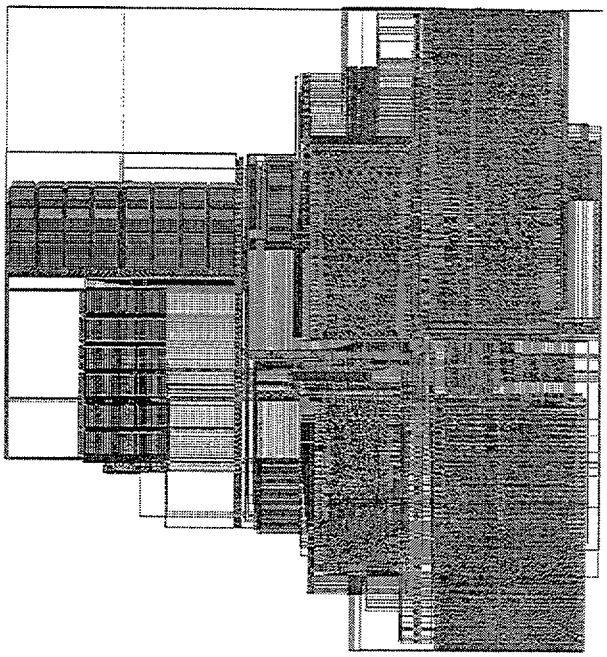


FIGURA A.31: *Layout* de la versión ba732

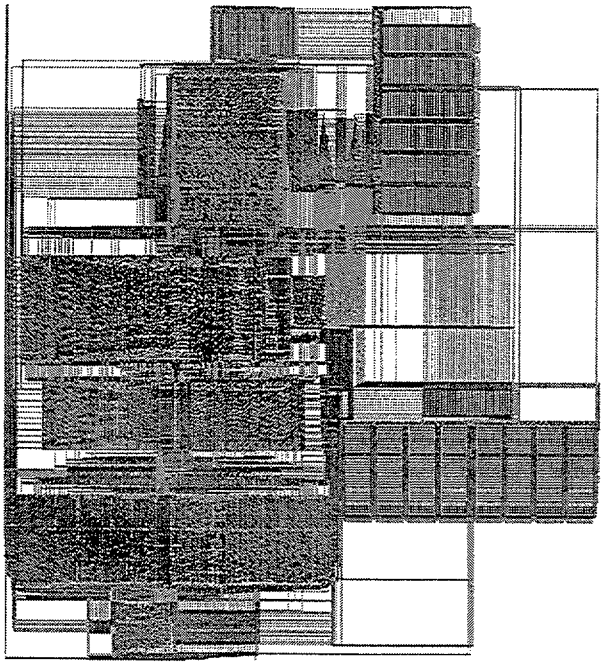


FIGURA A.32: *Layout* de la versión ba733

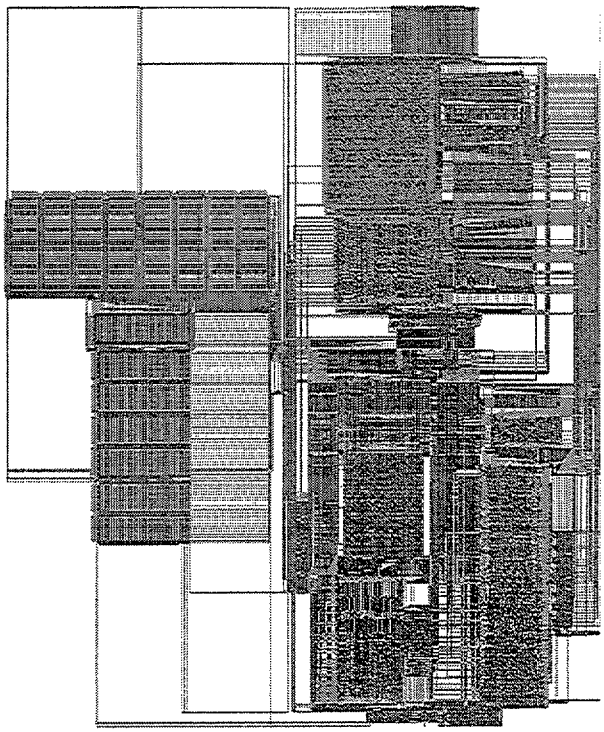


FIGURA A.33: *Layout* de la versión ba734

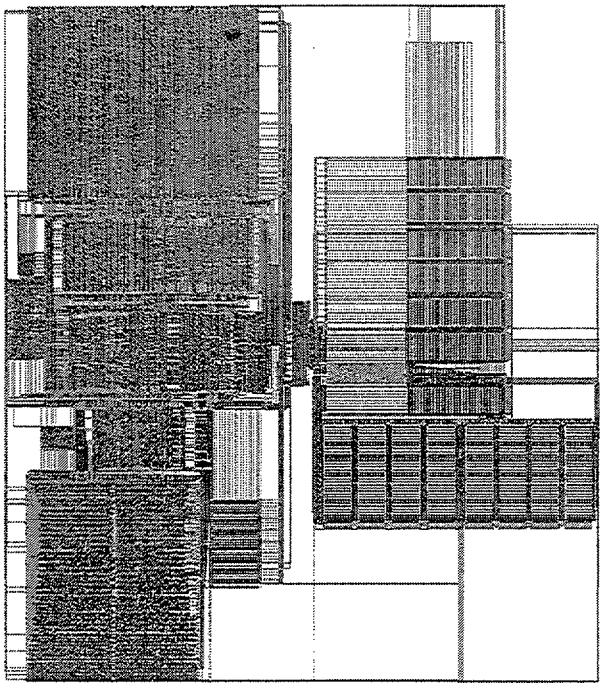


FIGURA A.34: *Layout* de la versión ba742

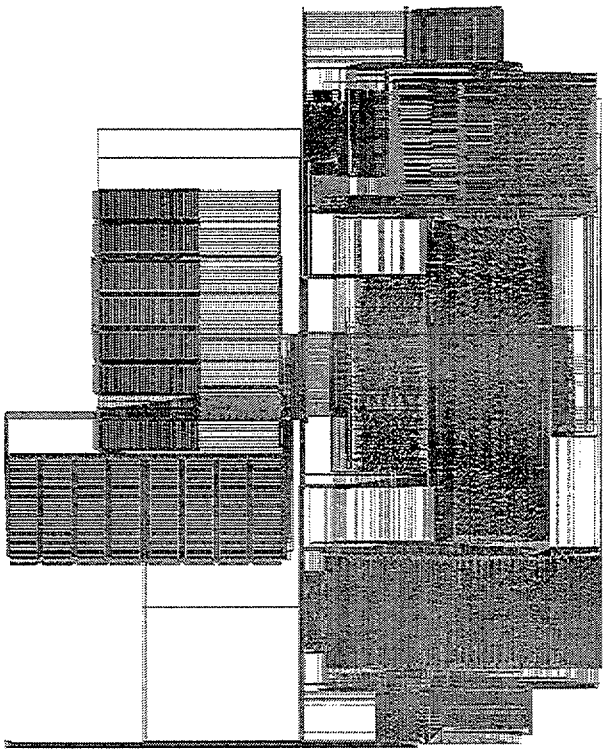


FIGURA A.35: *Layout* de la versión ba743

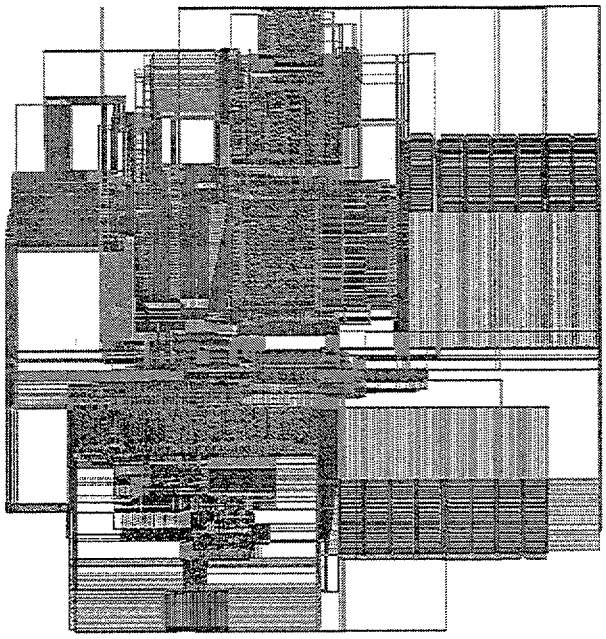


FIGURA A.36: *Layout* de la versión ba744

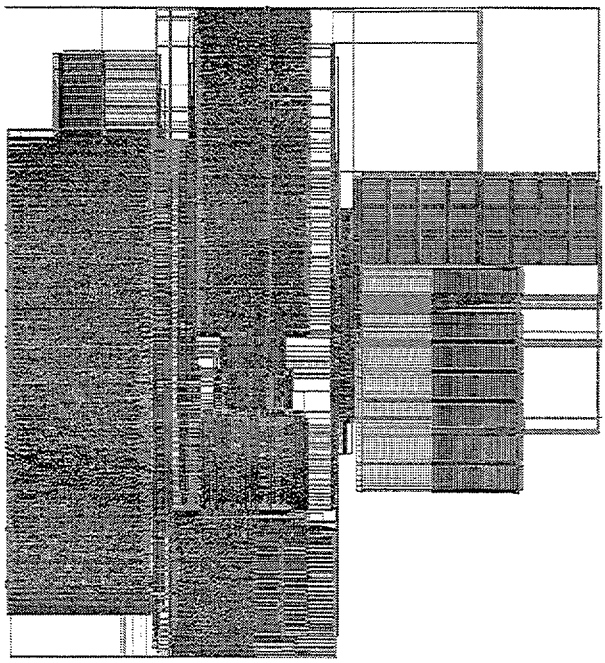


FIGURA A.37: *Layout* de la versión ba752

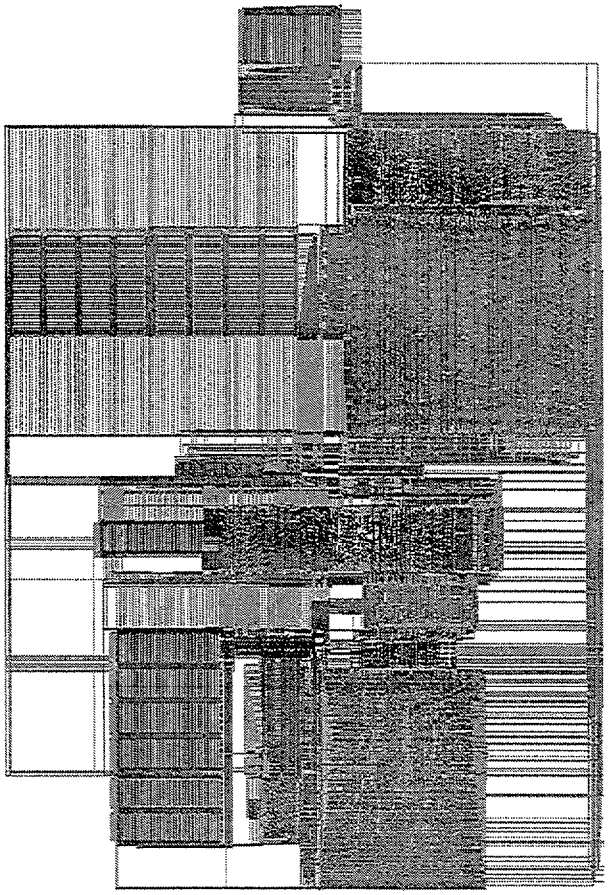


FIGURA A.38: *Layout* de la versión ba753

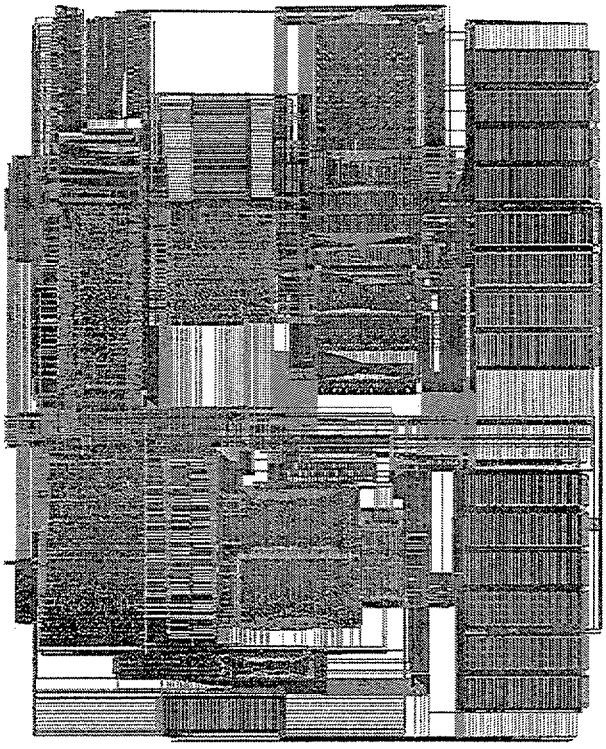


FIGURA A.39: *Layout* de la versión ba754

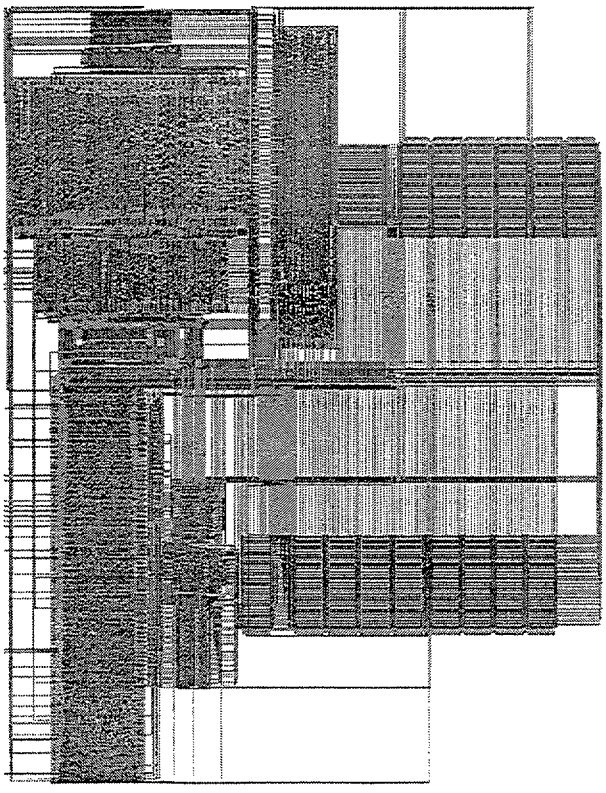


FIGURA A.40: *Layout* de la versión ba762

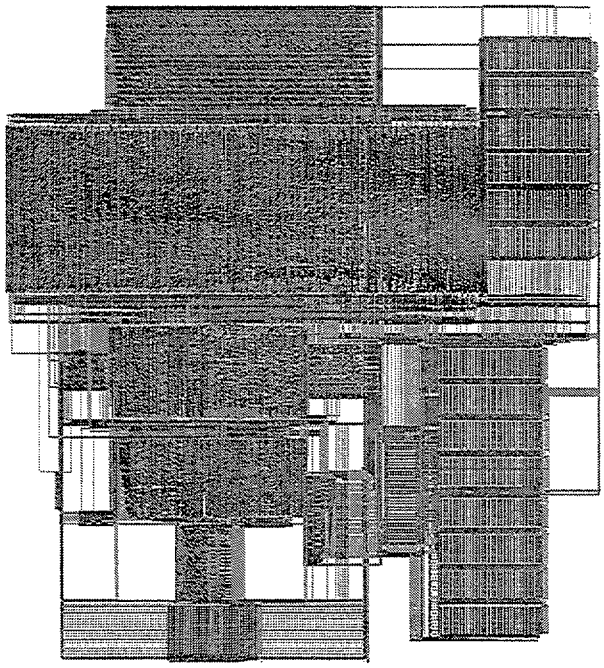


FIGURA A.41: *Layout* de la versión ba763

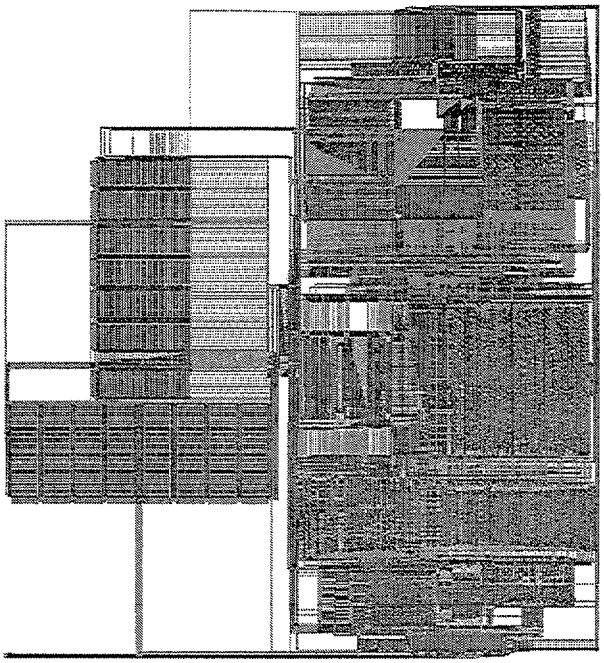


FIGURA A.42: *Layout* de la versión ba764

A.3. Versiones de la serie ca7xx

Las versiones de esta serie se han realizado con 7 ventanas, con la opción `fixedblock` puesta a 0 y siendo la herramienta quien realice la partición realizada de forma automática.

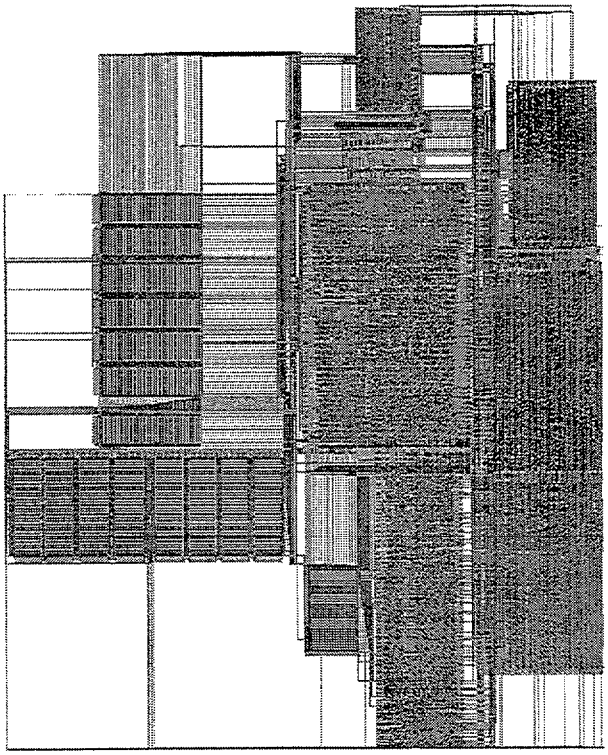


FIGURA A.43: *Layout* de la versión ca702

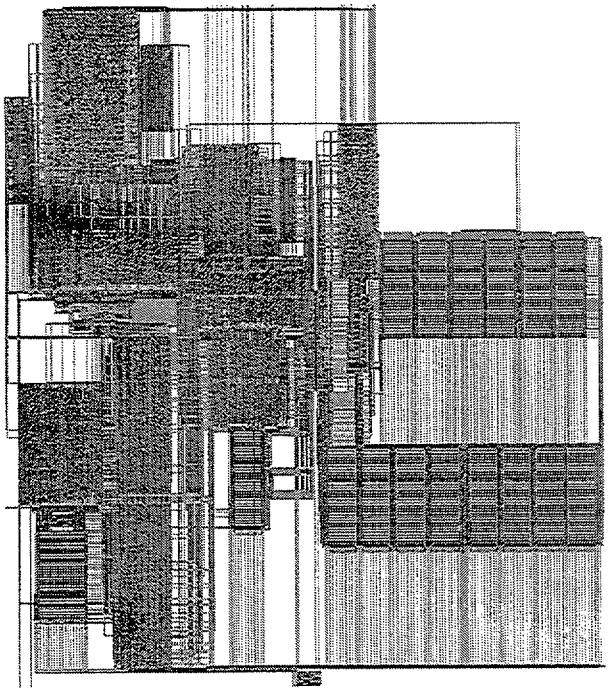


FIGURA A.44: *Layout* de la versión ca703

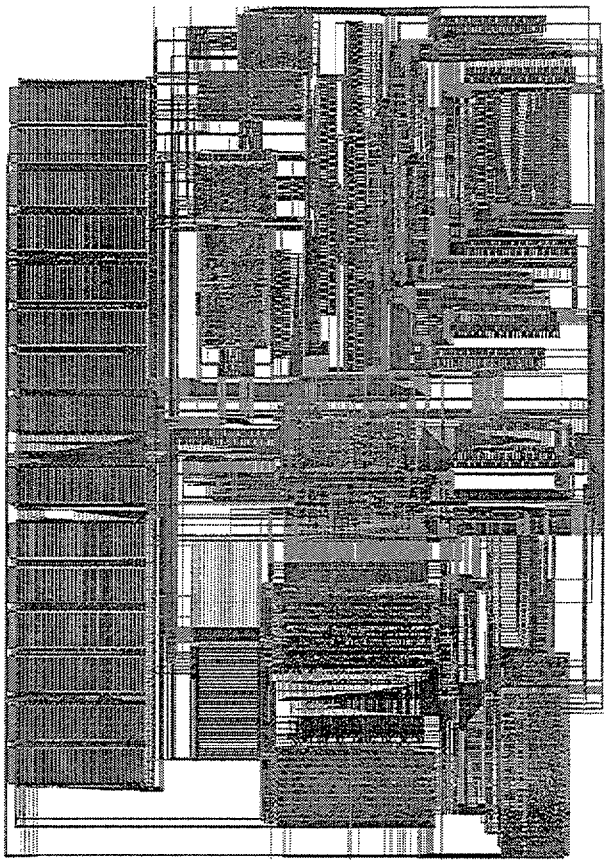


FIGURA A.45: *Layout* de la versión ca704

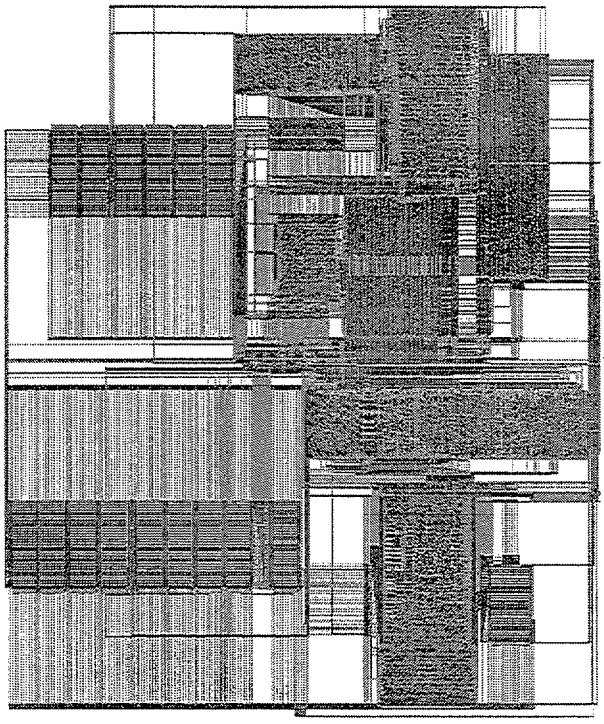


FIGURA A.46: *Layout* de la versión ca712

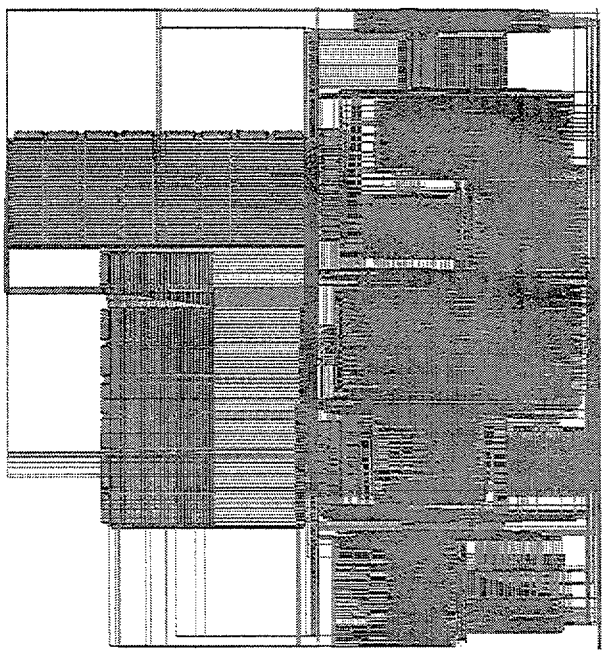


FIGURA A.47: *Layout* de la versión ca713

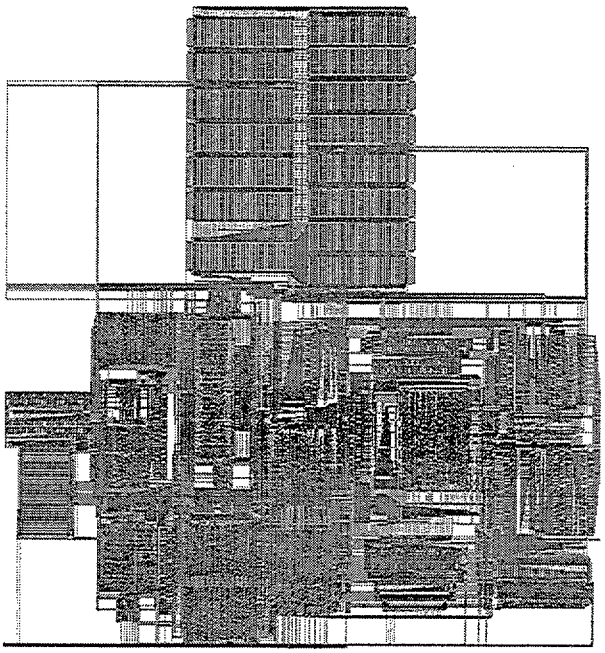


FIGURA A.48: *Layout* de la versión ca714

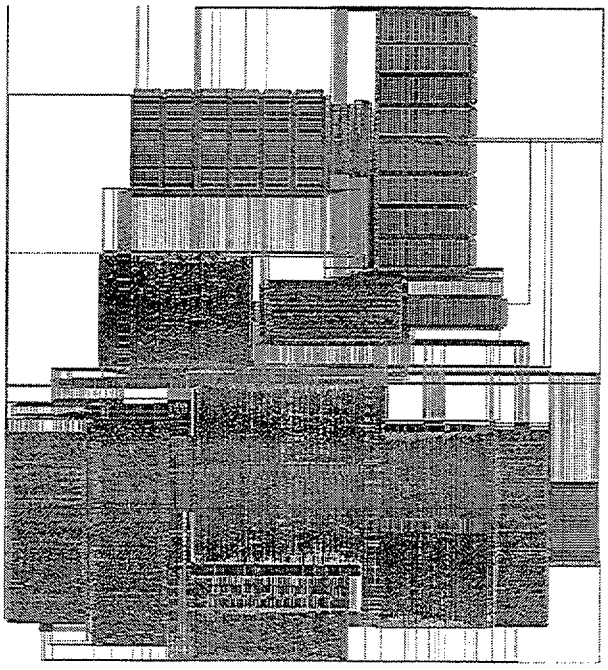


FIGURA A.49: *Layout* de la versión ca722

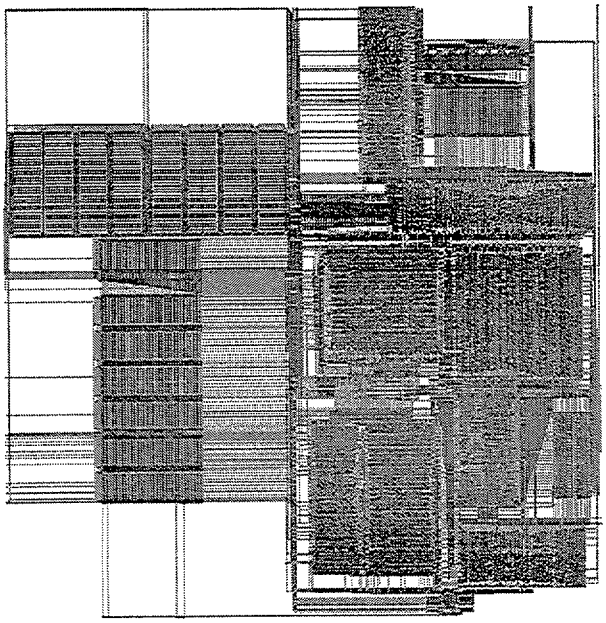


FIGURA A.50: *Layout* de la versión ca723

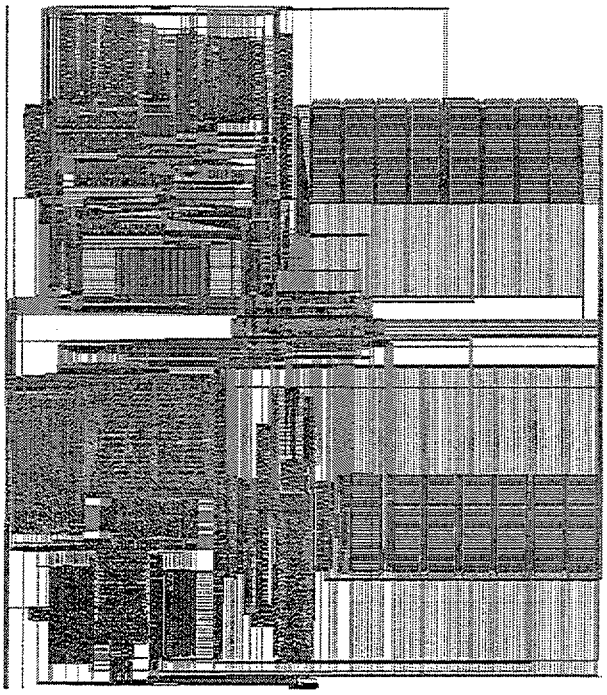


FIGURA A.51: *Layout* de la versión ca724

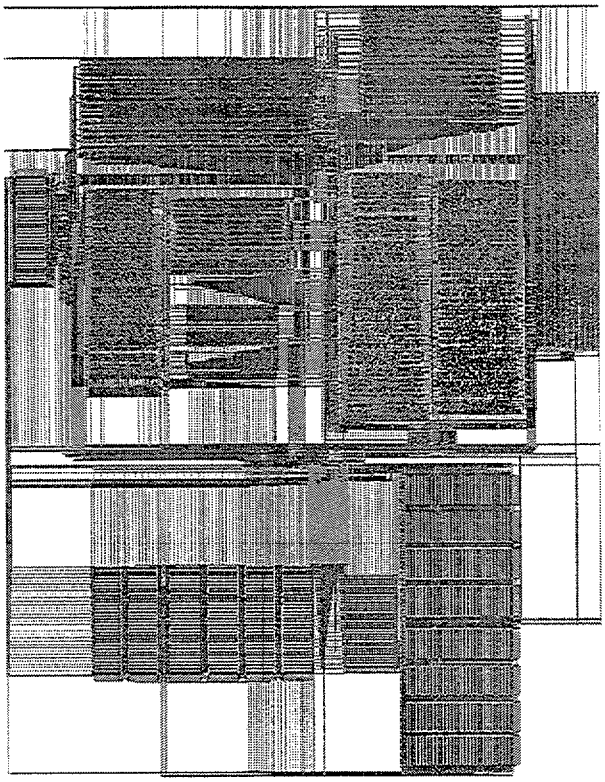


FIGURA A.52: *Layout* de la versión ca732

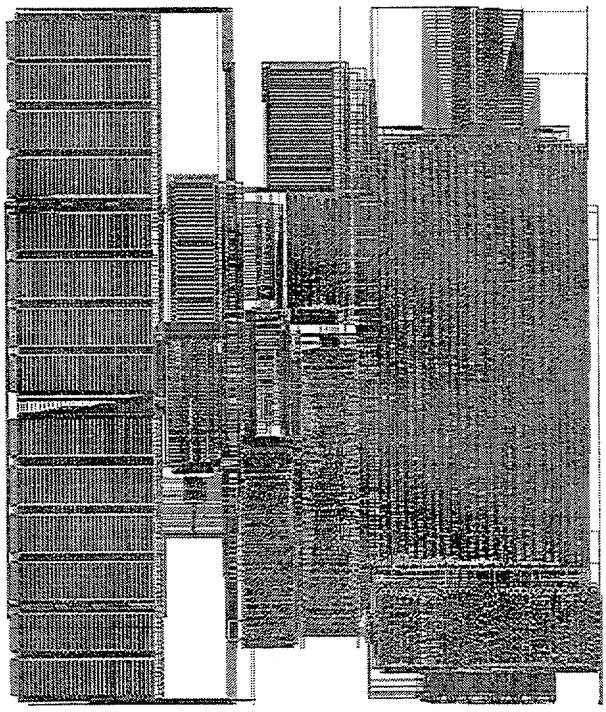


FIGURA A.53: *Layout* de la versión ca733

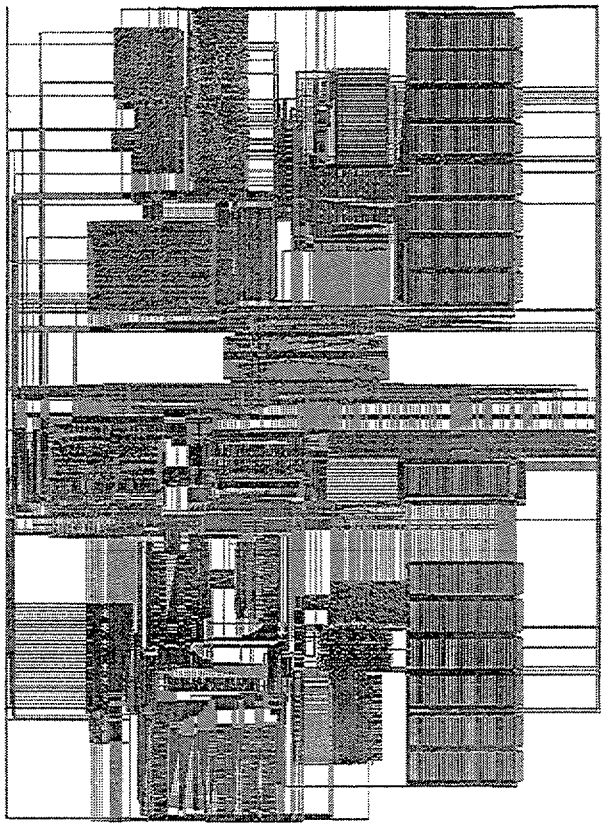


FIGURA A.54: *Layout* de la versión ca734

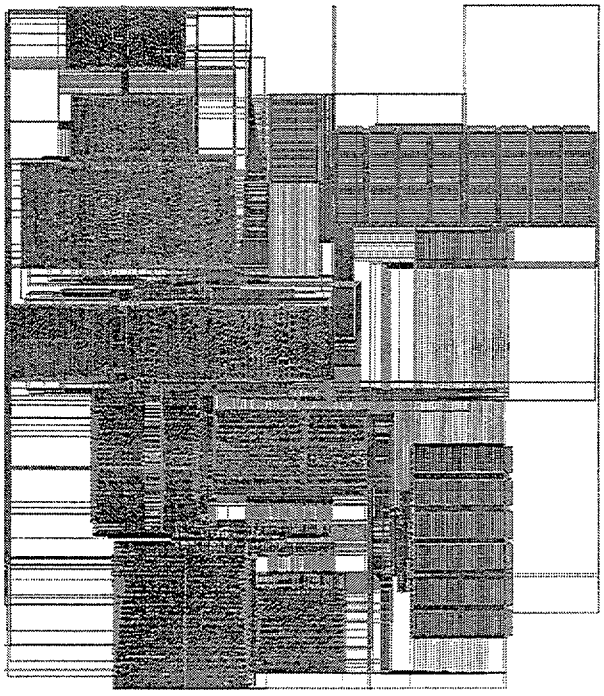


FIGURA A.55: *Layout* de la versión ca742

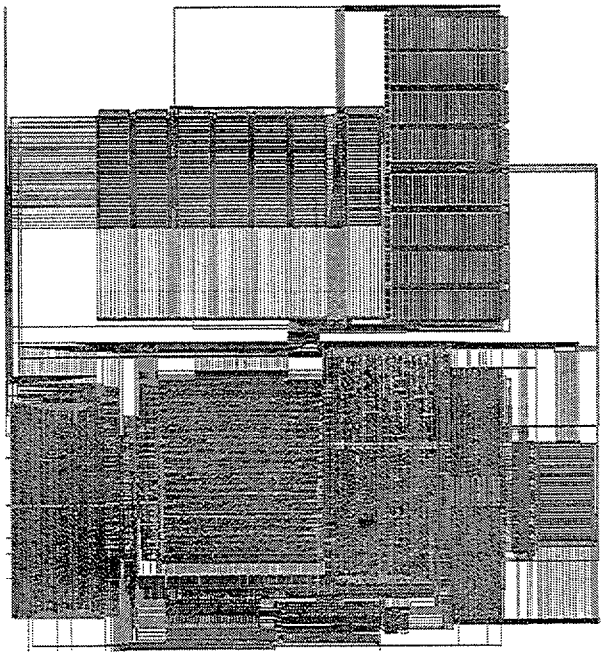


FIGURA A.56: *Layout* de la versión ca743

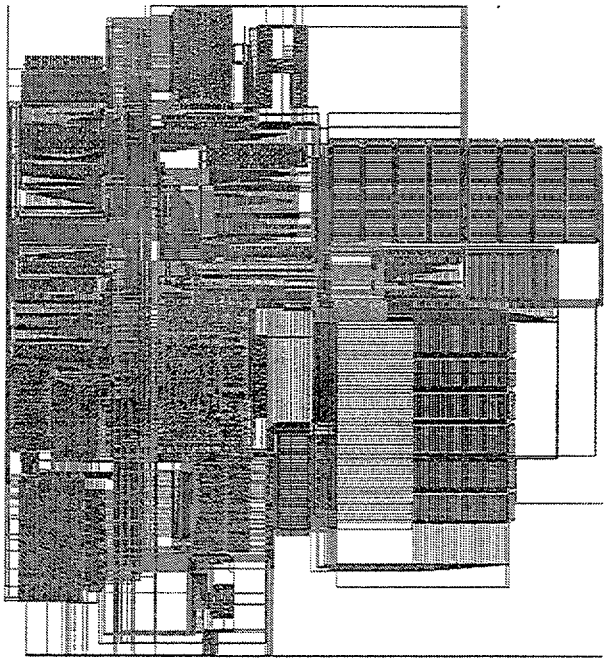


FIGURA A.57: *Layout* de la versión ca744

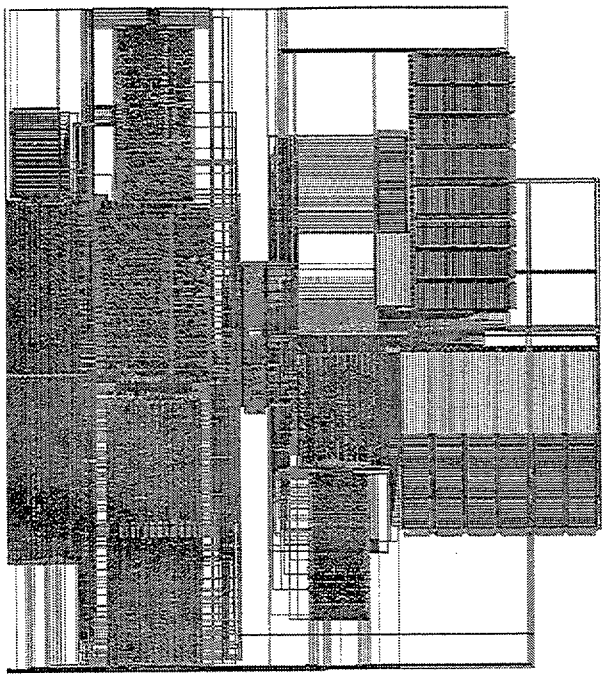


FIGURA A.58: *Layout* de la versión ca752

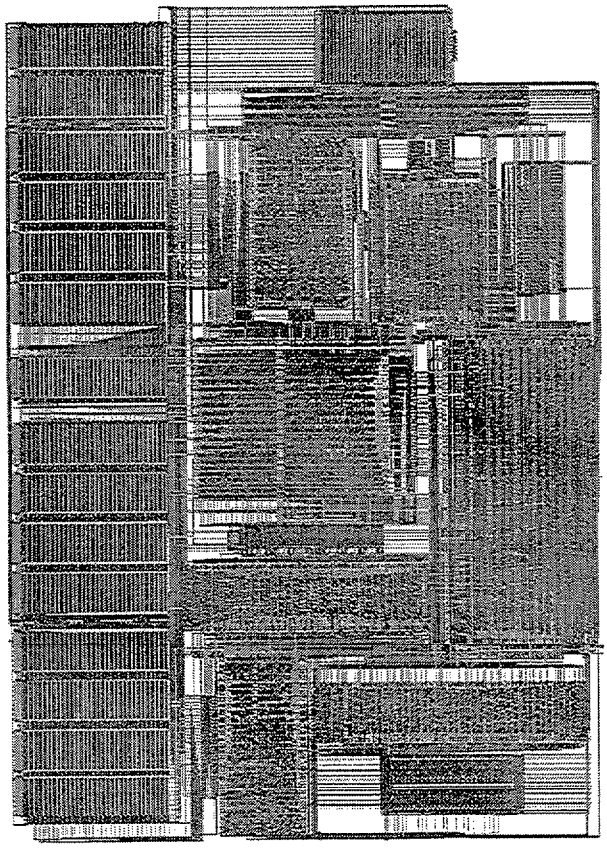


FIGURA A.59: *Layout* de la versión ca753

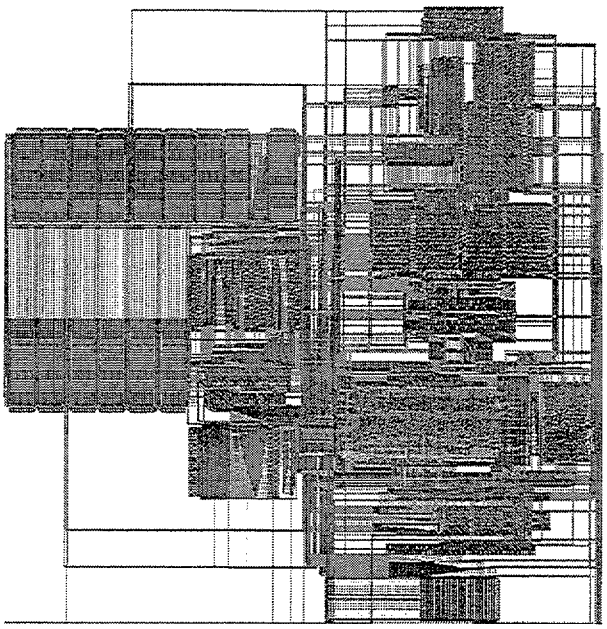


FIGURA A.60: *Layout* de la versión ca754

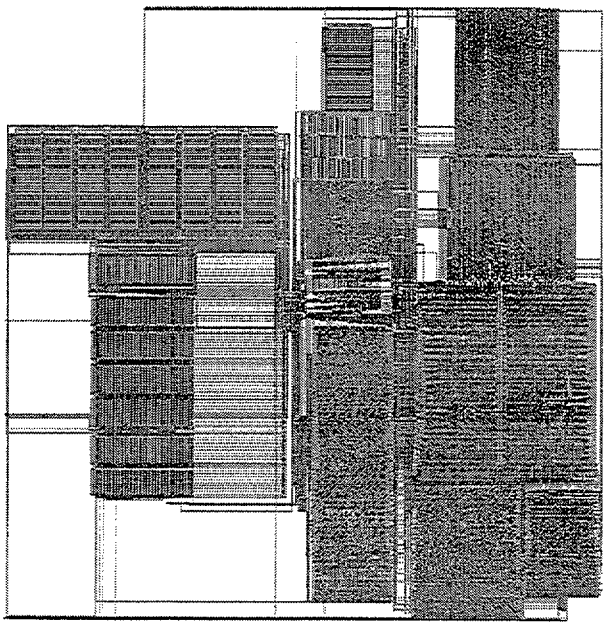


FIGURA A.61: *Layout* de la versión ca762

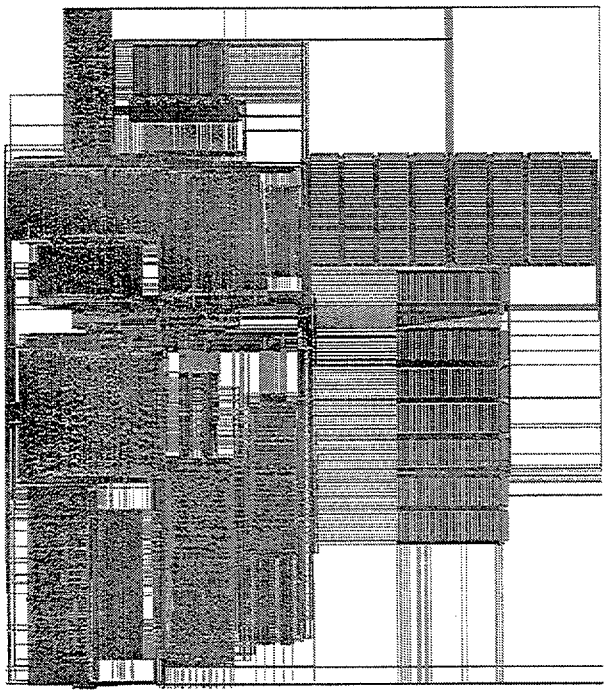


FIGURA A.62: *Layout* de la versión ca763

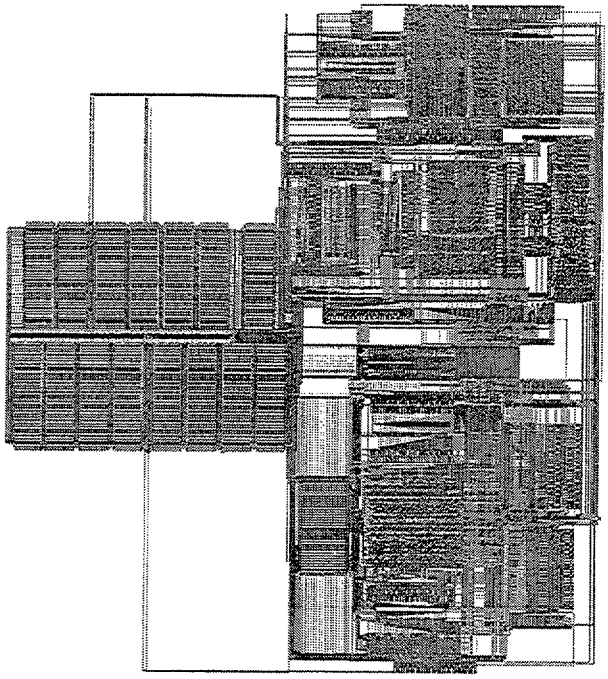


FIGURA A.63: *Layout* de la versión ca764

A.4. Versiones de la serie aa2xx

Estas versiones se han realizado con 2 ventanas, con la opción `fixedblock` puesta a 0 y permitiendo a la herramienta que realice la partición de forma completamente automática.

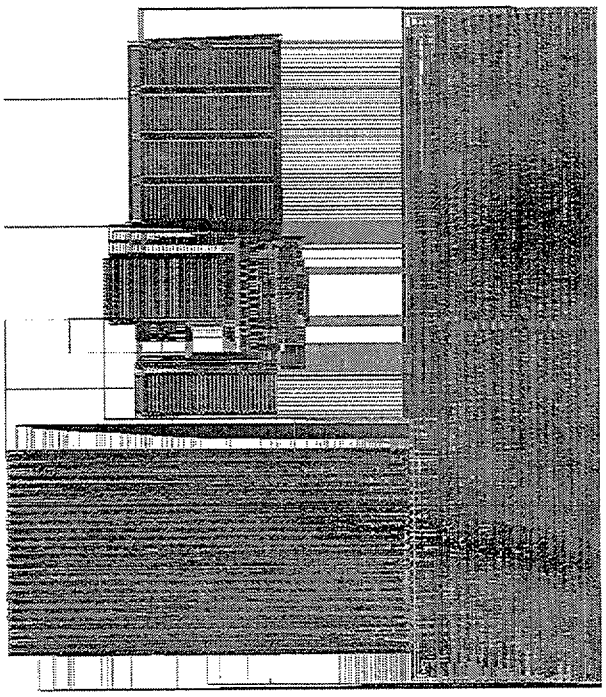


FIGURA A.64: *Layout* de la versión aa222

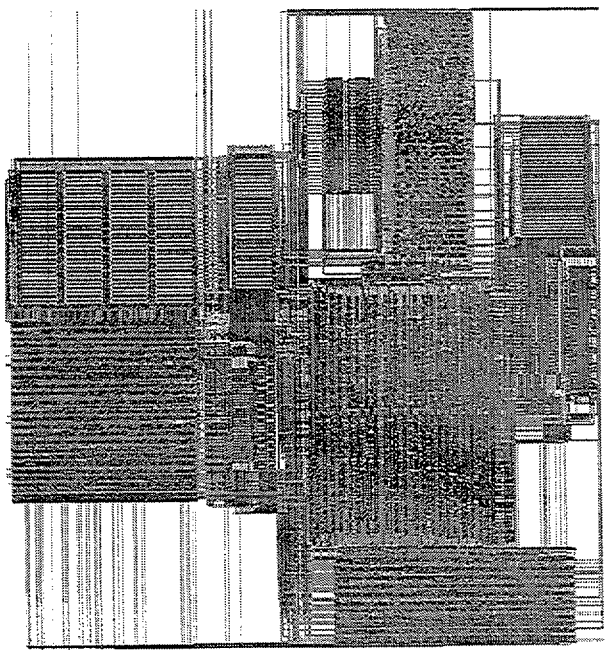


FIGURA A.65: *Layout* de la versión aa223

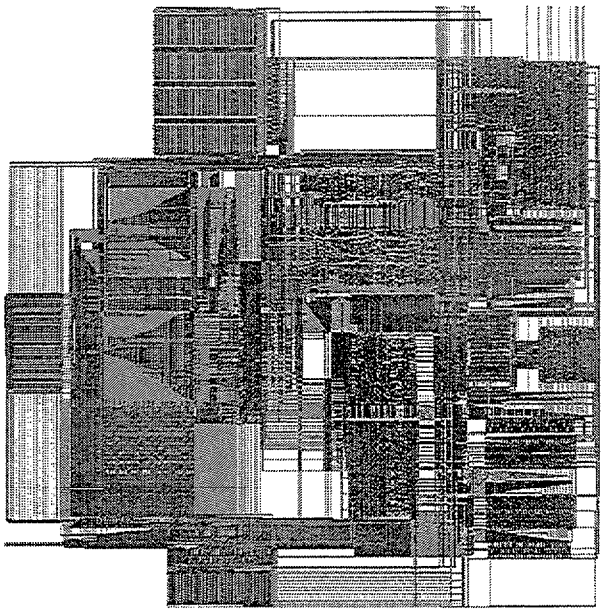


FIGURA A.66: *Layout* de la versión aa224

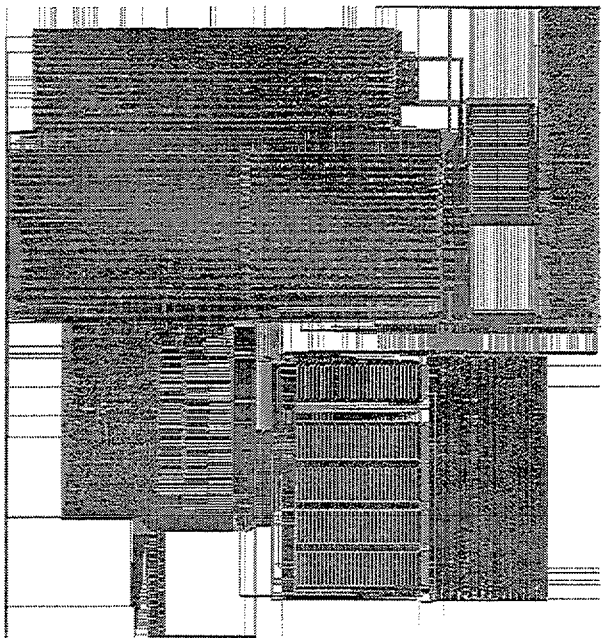


FIGURA A.67: *Layout* de la versión aa252

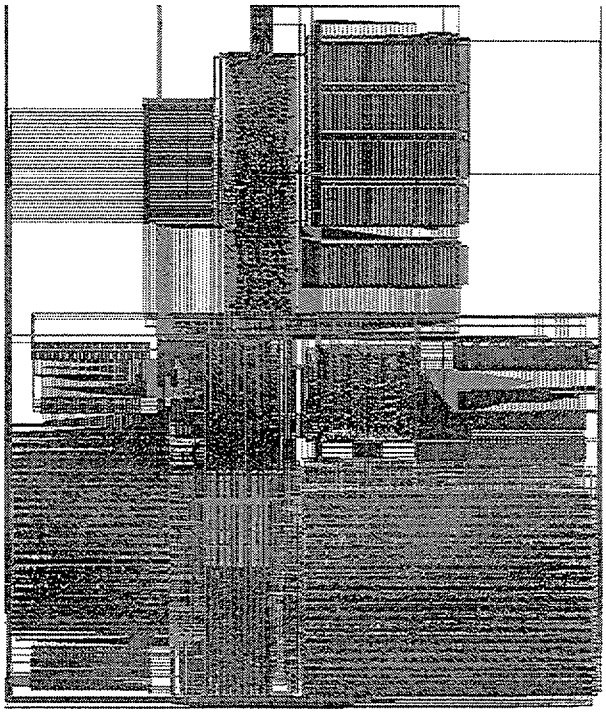


FIGURA A.68: *Layout* de la versión aa253

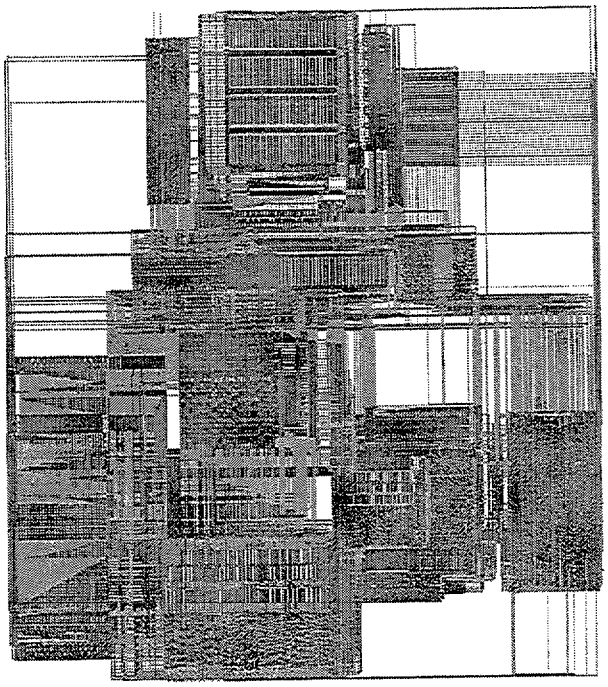


FIGURA A.69: *Layout* de la versión aa254

A.5. Versiones de la serie aa4xx

Estas versiones se han realizado con 4 ventanas, con la opción `fixedblock` puesta a 0 y permitiendo a la herramienta que realice la partición de forma completamente automática.

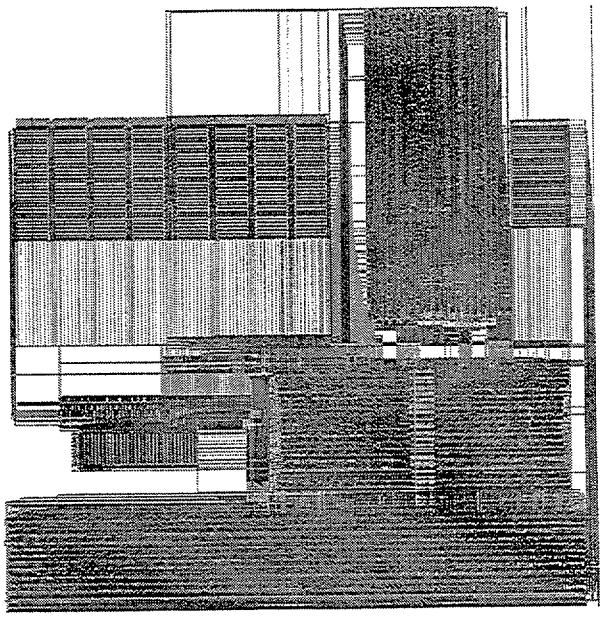


FIGURA A.70: *Layout* de la versión aa422

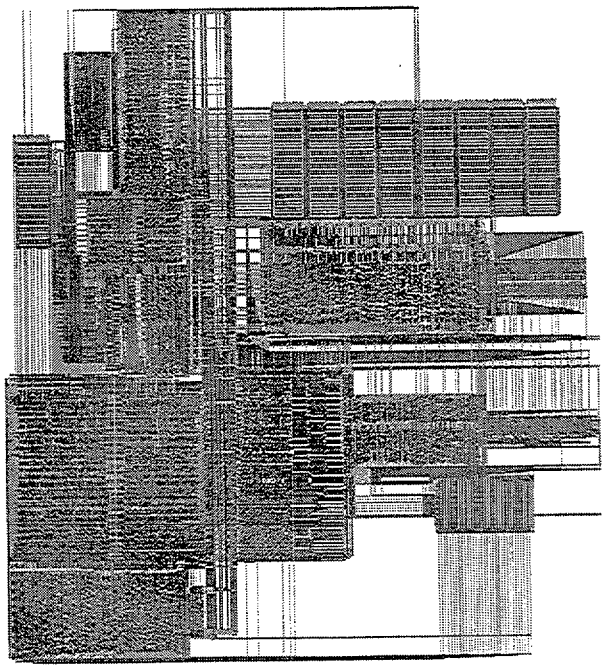


FIGURA A.71: *Layout* de la versión aa423

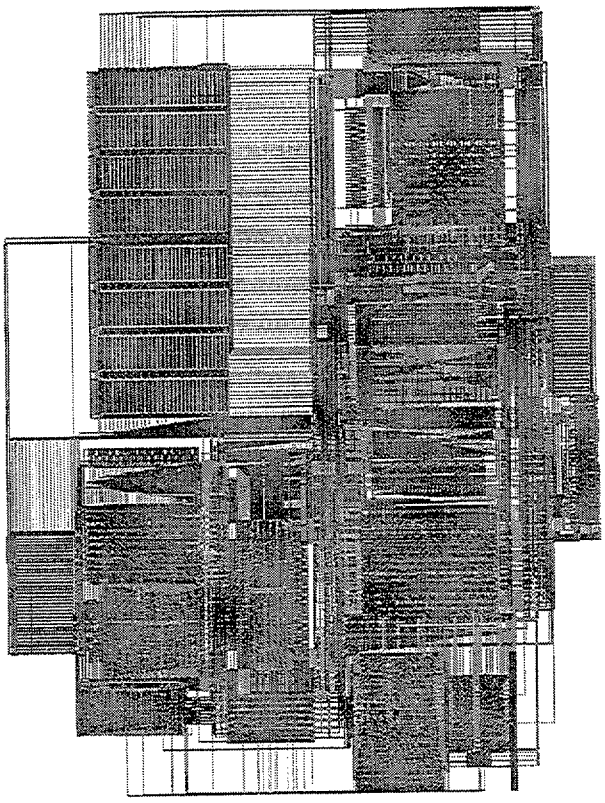


FIGURA A.72: *Layout* de la versión aa424

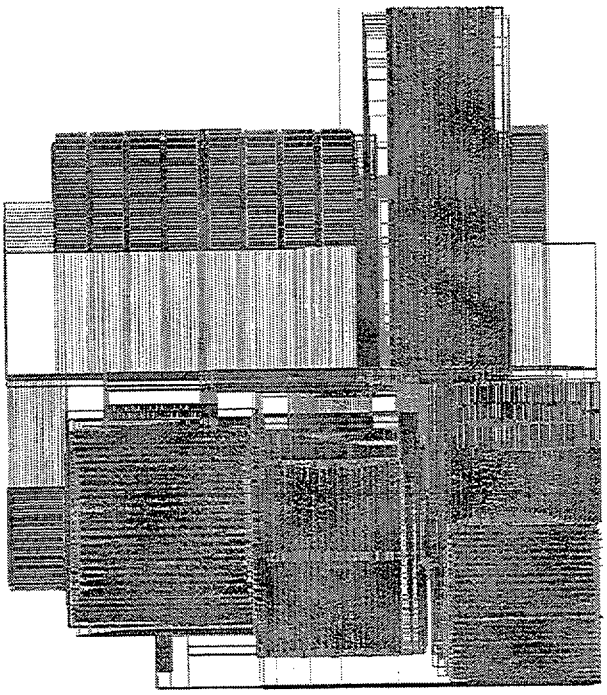


FIGURA A.73: *Layout* de la versión aa452

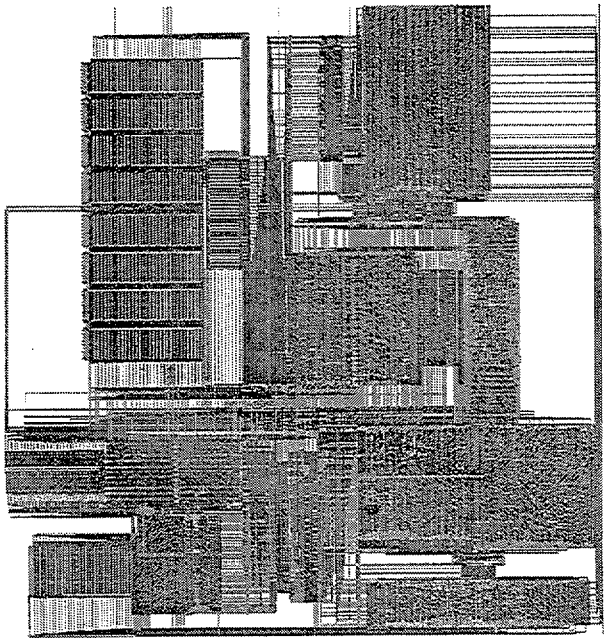


FIGURA A.74: *Layout* de la versión aa453

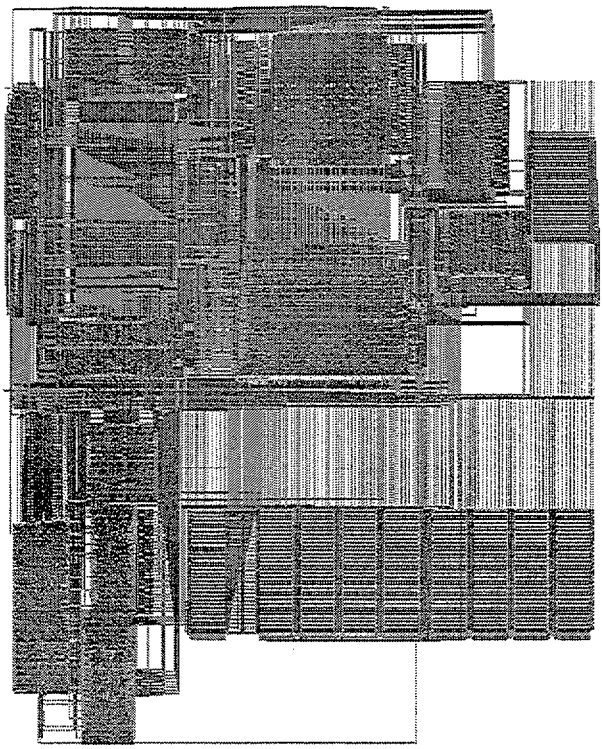


FIGURA A.75: *Layout* de la versión aa454

A.6. Layouts de variantes basadas en la aa753

Estas variantes se ha realizado para estudiar el impacto de la introducción de distintos elementos adicionales sobre las propiedades de la versión original. Los *layouts* se muestran en las figuras de la A.76 a la A.87.

A.7. Layouts de variantes con test

Estas *layouts* son los correspondientes a los núcleos en donde se ha introducido la circuitería de *scan-path*. Se presentan en las figuras A.86 y A.87.

A.8. Layouts de versiones con tecnología HP14B

Todas los *layouts* presentados previamente se han desarrollado con una tecnología CMOS de 0.35 μm. En este apartado se muestran los *layouts* de otras versiones que se han realizado con una tecnología CMOS de 0.5 μm. Estos son los mostrados en las figuras A.88 y A.89.

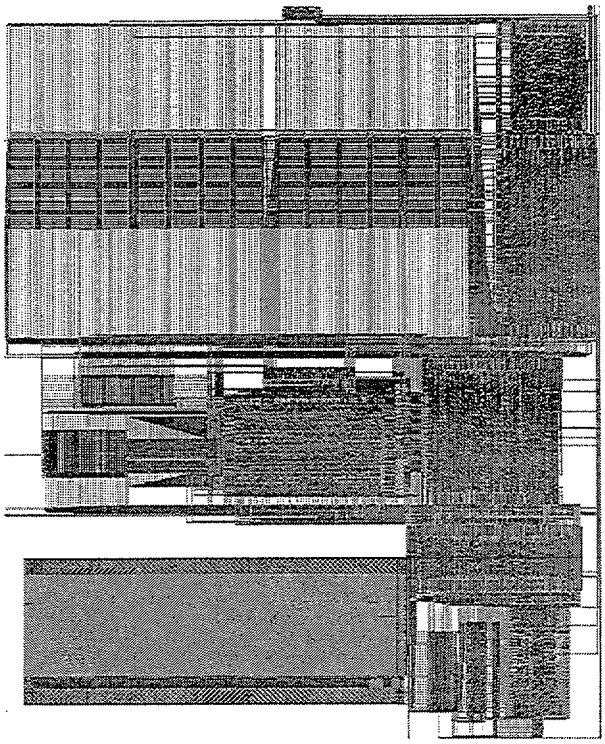


FIGURA A.76: *Layout* de la versión `a1753`, con memoria de 4 Kbytes sin conectar y con plasticidad en el núcleo

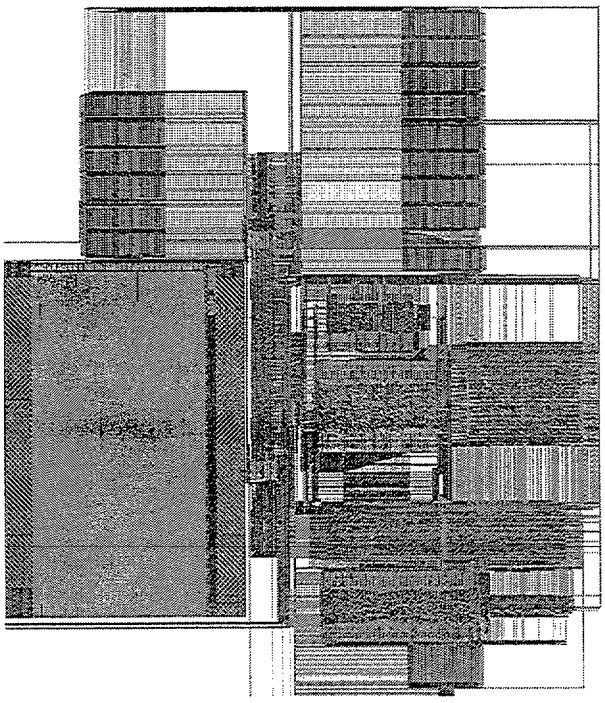


FIGURA A.77: *Layout* de la versión `am753`, con memoria de 8 Kbytes sin conectar y con plasticidad en el núcleo

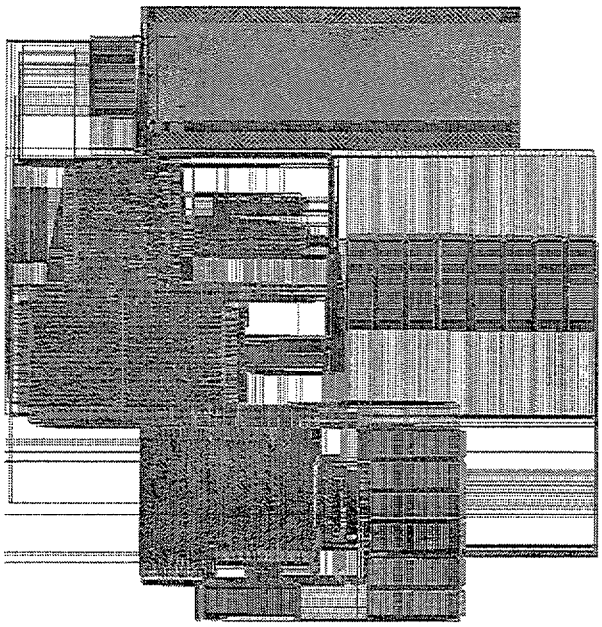


FIGURA A.78: *Layout* de la versión an753 con memoria de 4 Kbytes conectada, con plasticidad en el núcleo

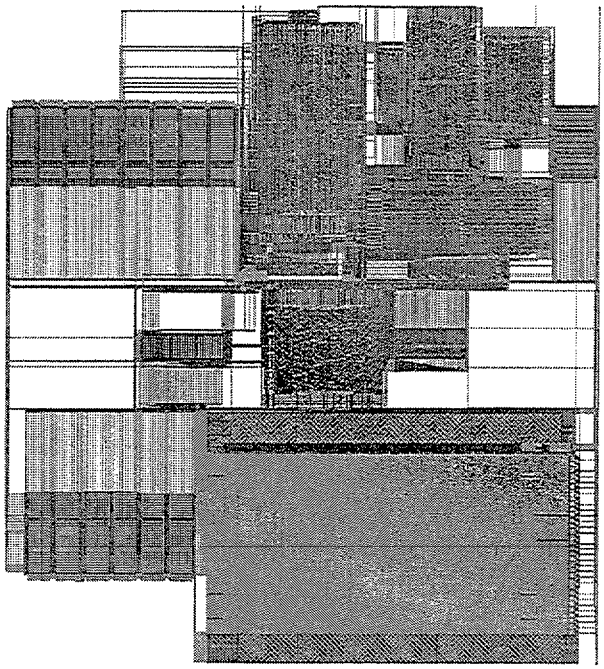


FIGURA A.79: *Layout* de la versión ao753, con memoria de 8 Kbytes conectada, con plasticidad en el núcleo

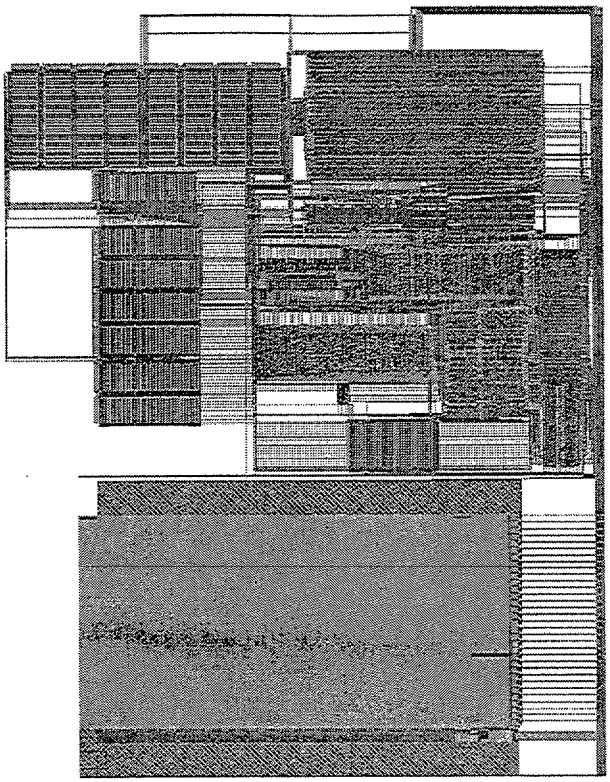


FIGURA A.80: *Layout* de la versión ap753, con memoria de 8 Kbytes conectada, con núcleo prefijado

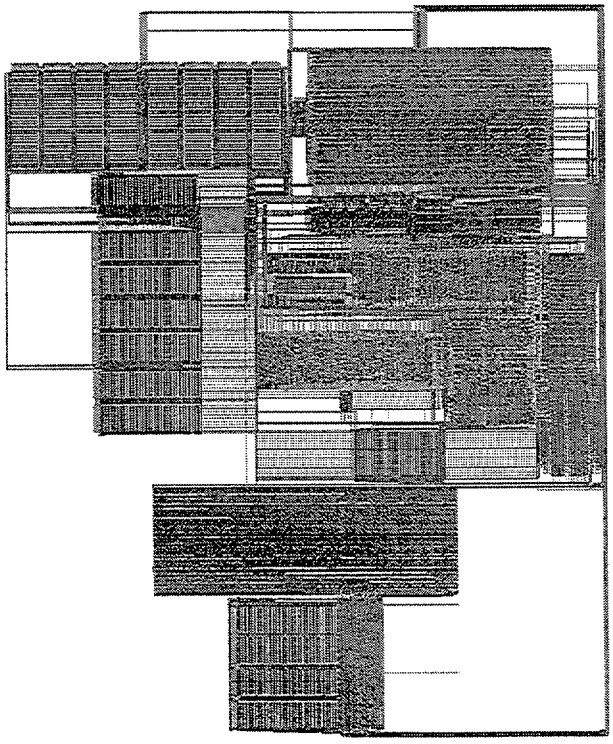


FIGURA A.81: *Layout* de la versión ae753 con FPU dispuesta en forma libre y núcleo prefijado

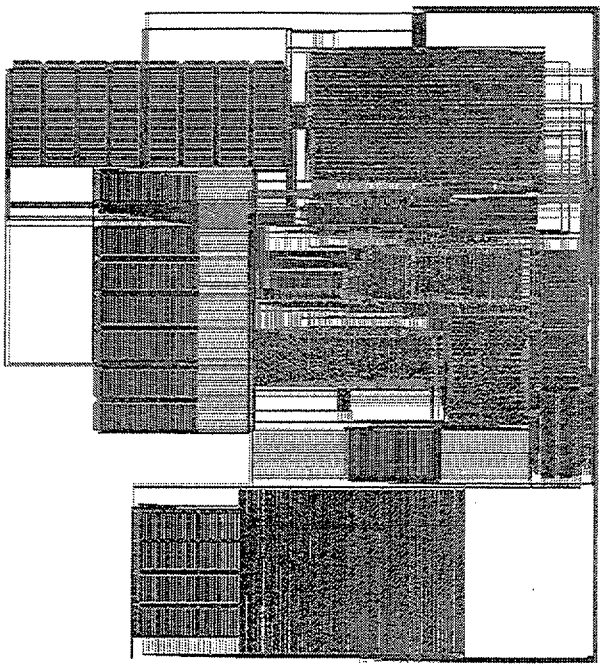


FIGURA A.82: *Layout* de la versión `af753` con FPU con colocado guiado y núcleo prefijado

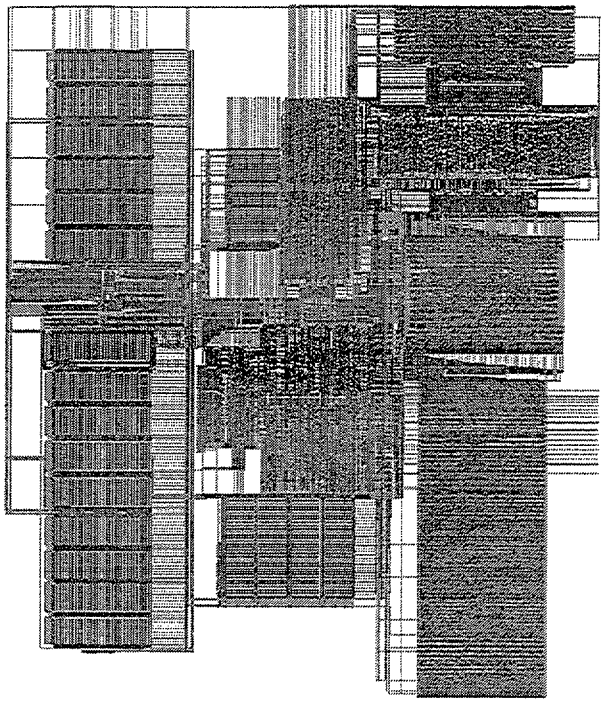


FIGURA A.83: *Layout* de la versión `ag753` con FPU con colocado libre y núcleo con plasticidad libre

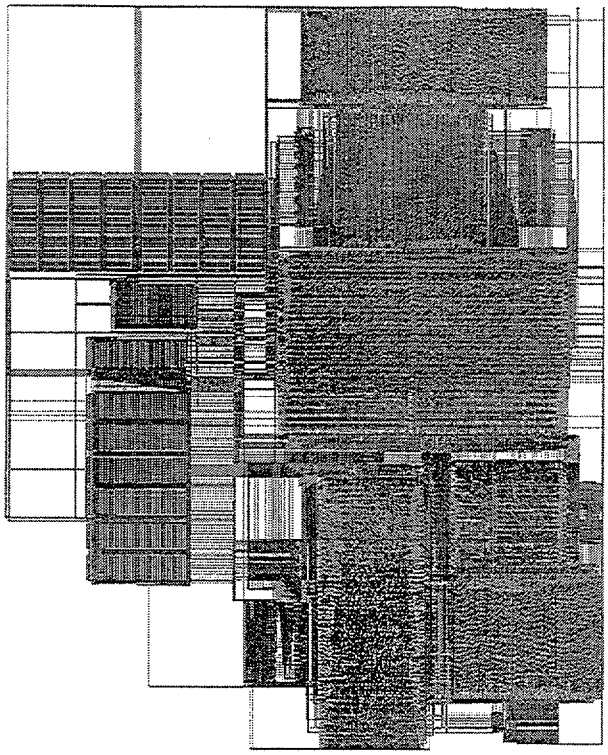


FIGURA A.84: *Layout* de la versión av753, consistente en la versión aa753 con instrucciones y módulo lento adicional para aceleración de aplicaciones multimedia

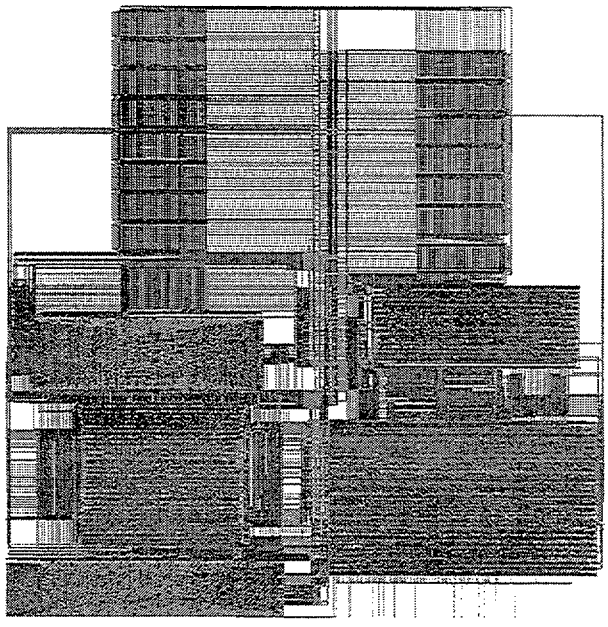


FIGURA A.85: *Layout* de la versión aw753, consistente en la versión aa753 con instrucciones y módulo rápido adicional para aceleración de aplicaciones multimedia

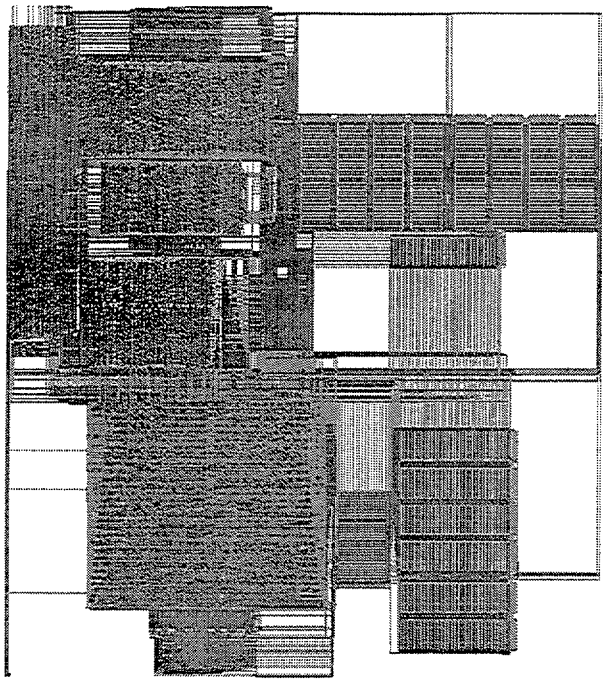


FIGURA A.86: *Layout* de la versión at733, consistente en la versión aa733 con circuitería de test incluida

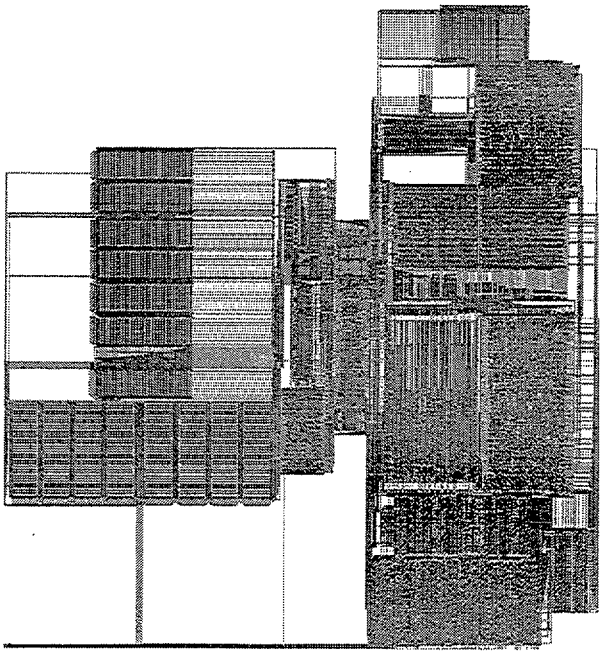


FIGURA A.87: *Layout* de la versión at753, consistente en la versión aa753 con circuitería de test incluida

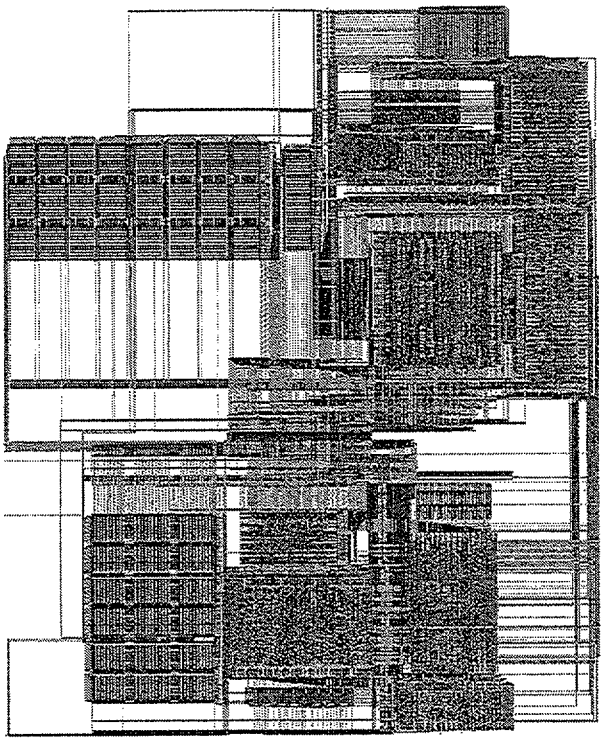


FIGURA A.88: *Layout* de la versión ha723

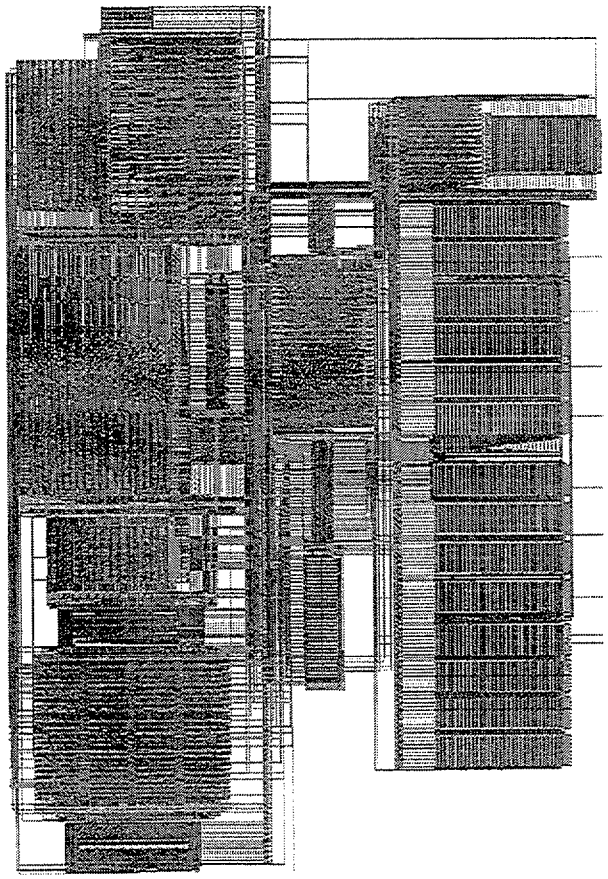


FIGURA A.89: *Layout* de la versión ha753

APÉNDICE B

Esquemáticos empleados en el modelado

B.1. Estructuras para el secuenciamiento

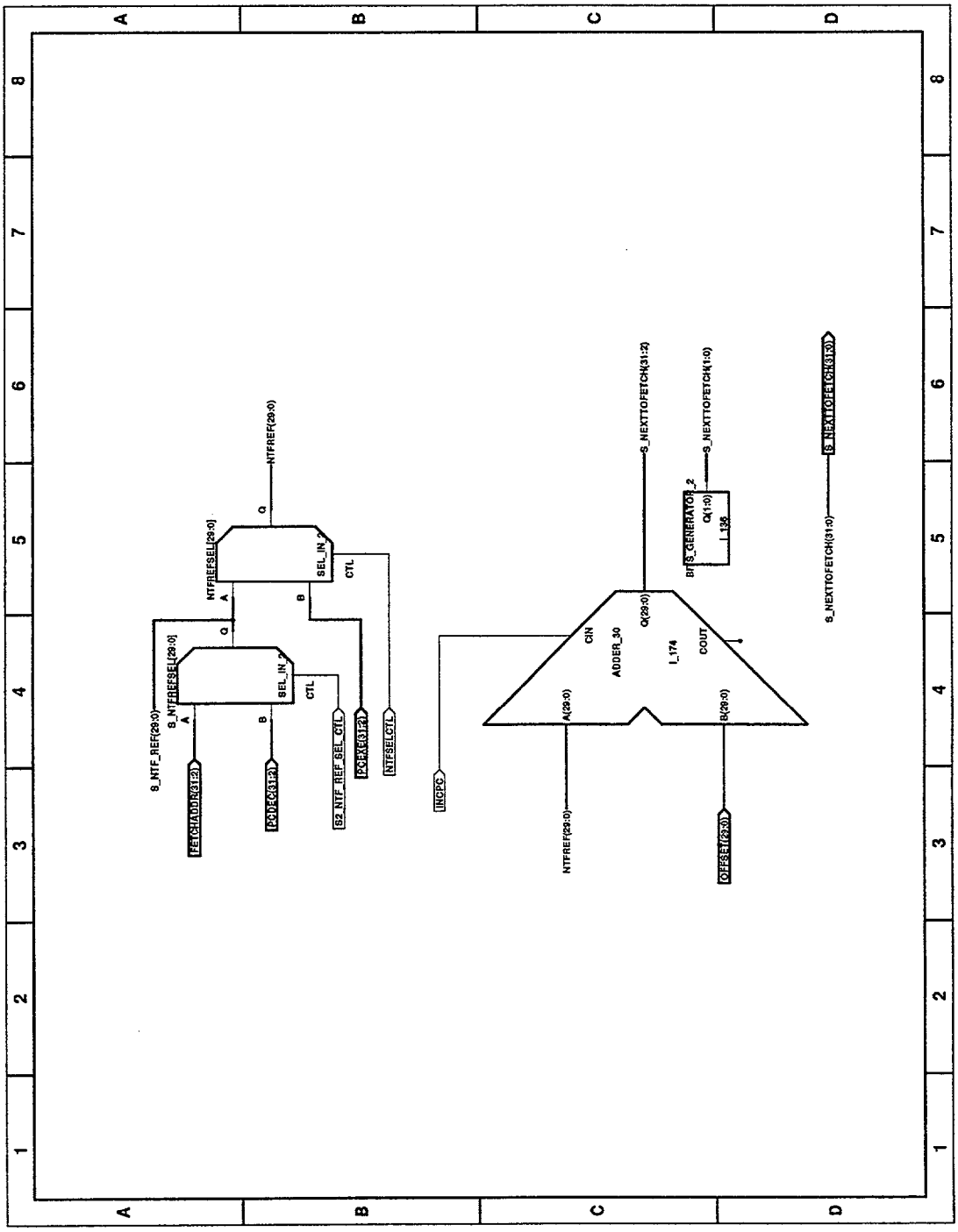


FIGURA B.1: Esquemático de la estructura lenta para el cálculo del PC siguiente

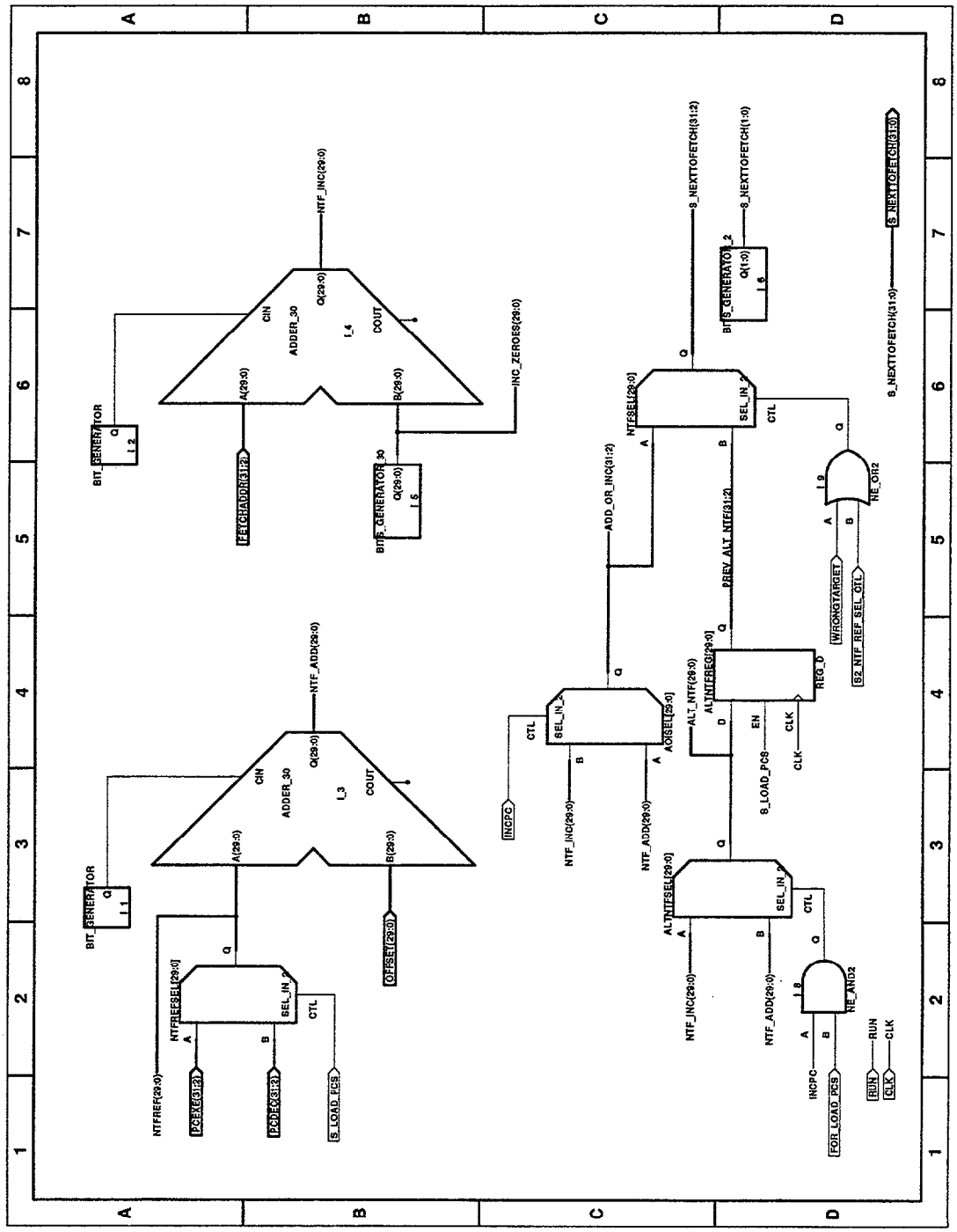


FIGURA B.2: Esquemático de la estructura rápida para el cálculo del PC siguiente

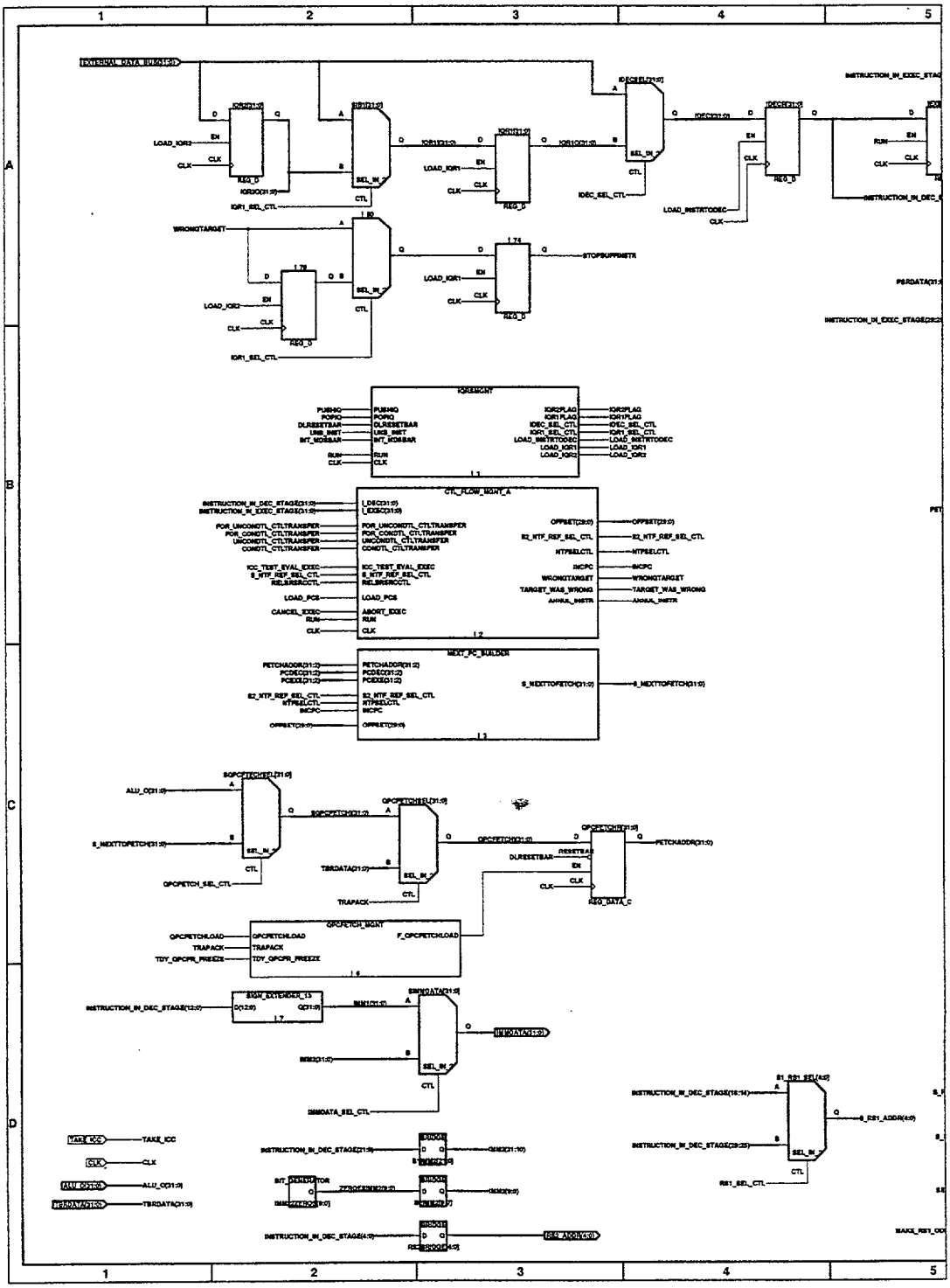


FIGURA B.3: Esquemático de la Unidad de Control (1)

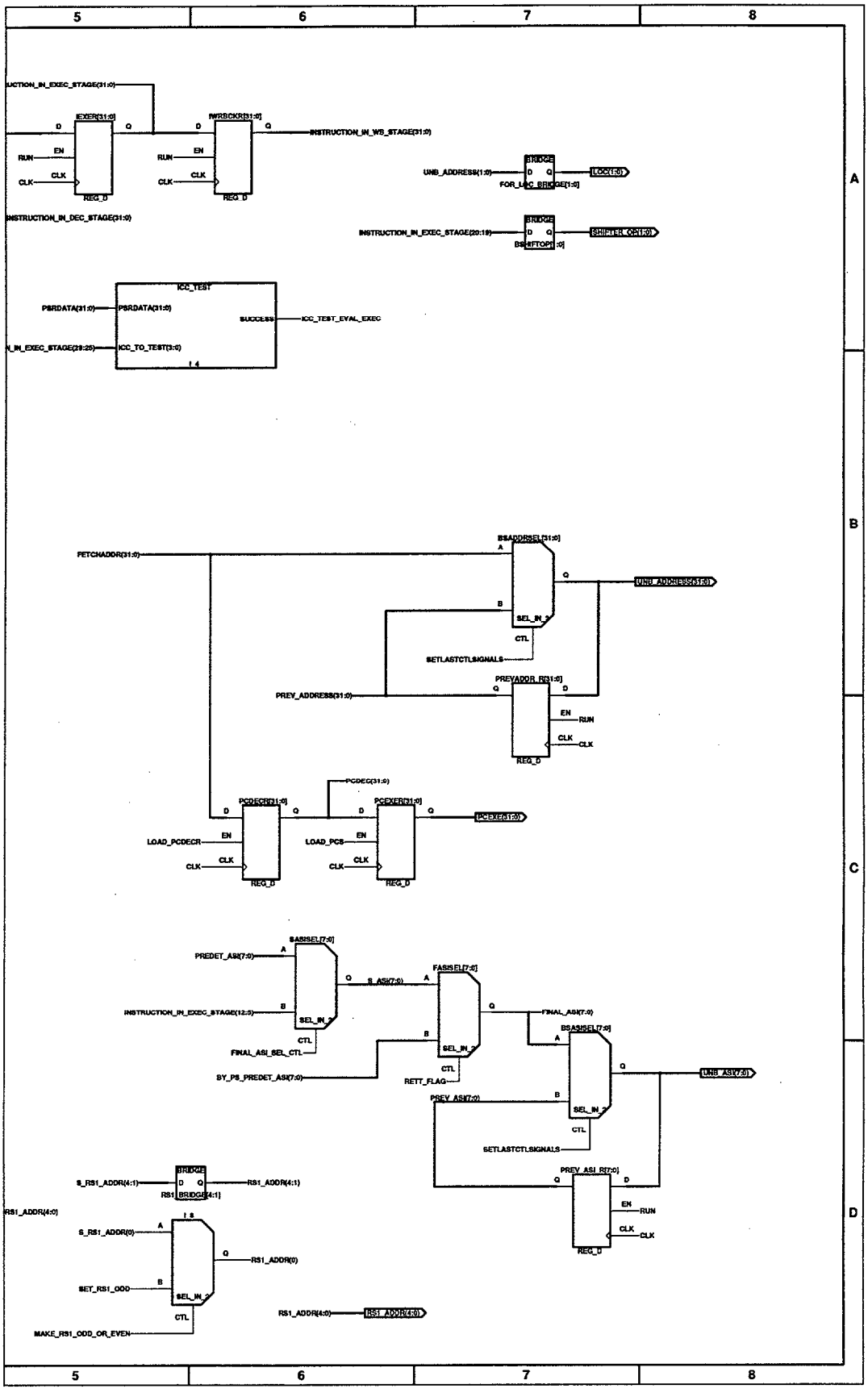


FIGURA B.4: Esquemático de la Unidad de Control (2)

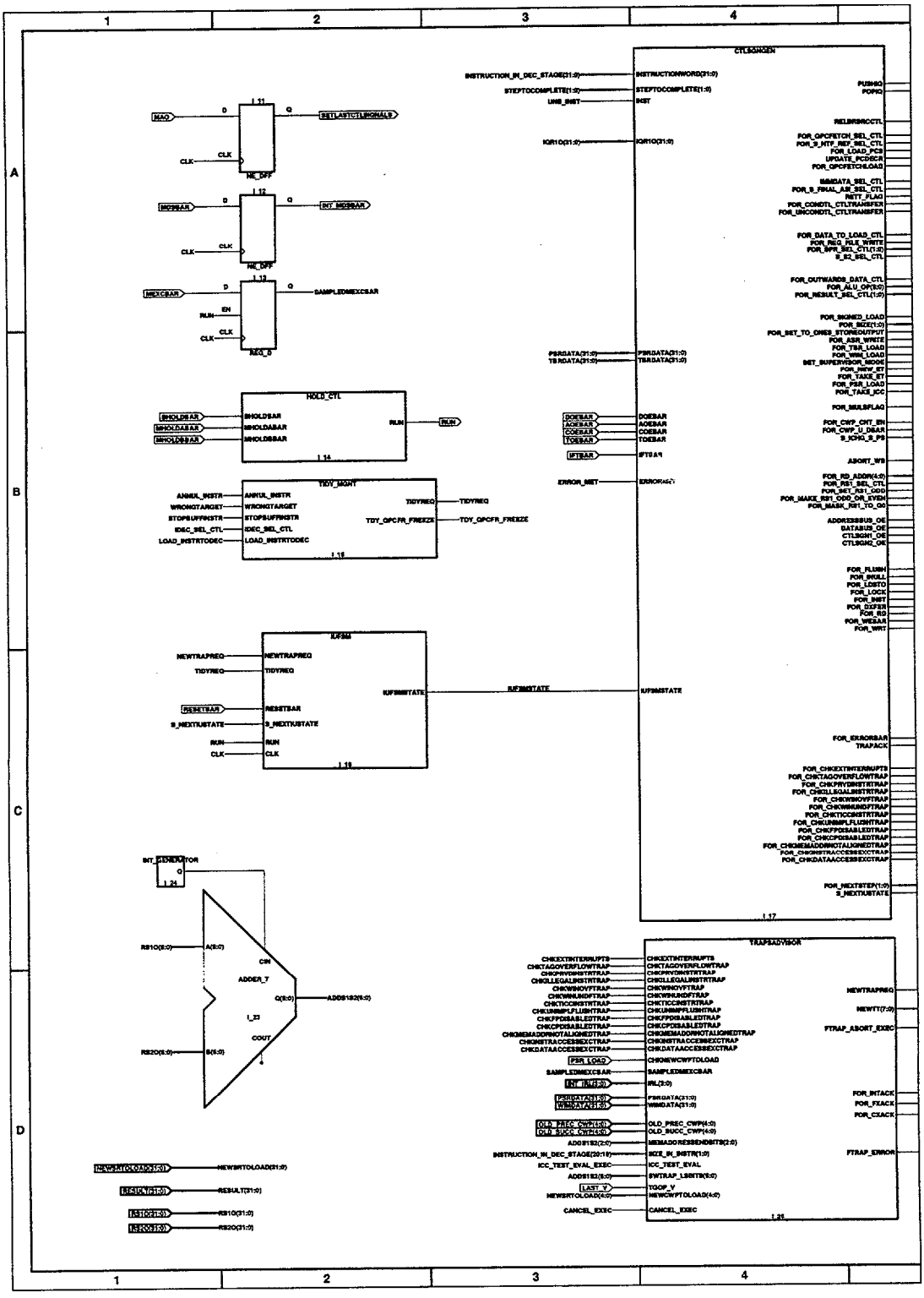


FIGURA B.5: Esquemático de la Unidad de Control (3)

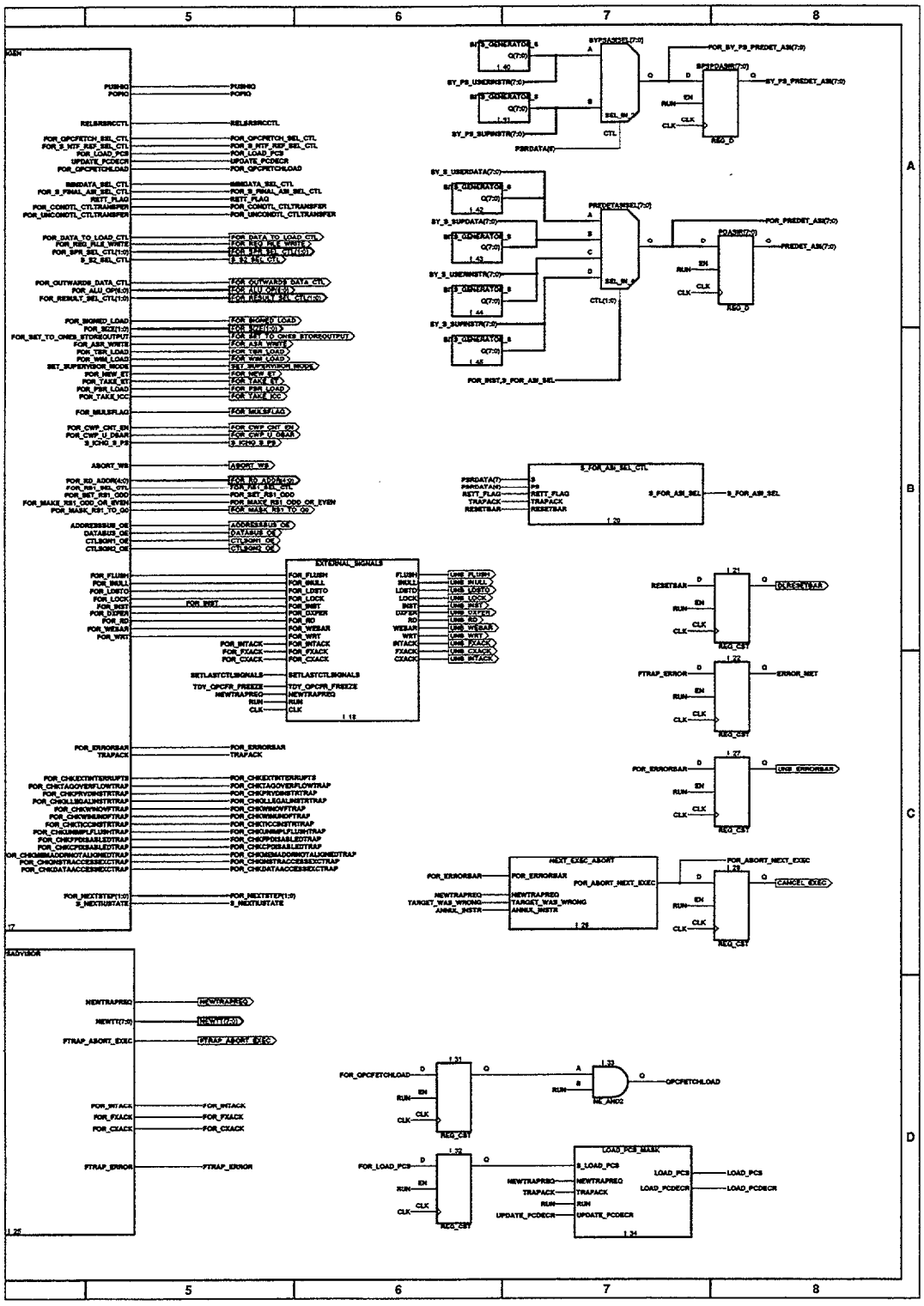


FIGURA B.6: Esquemático de la Unidad de Control (4)

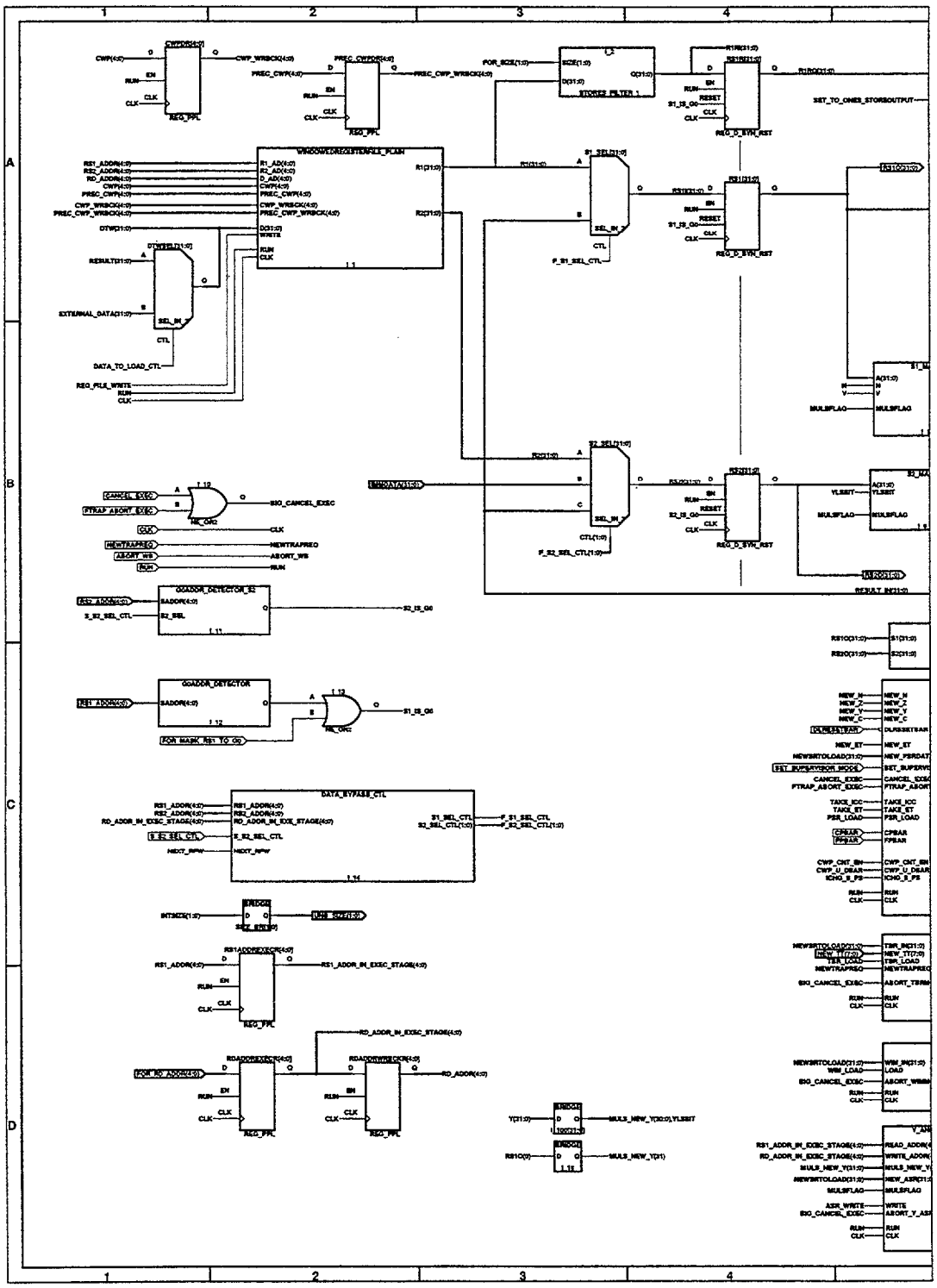


FIGURA B.7: Esquemático de la Ruta de Datos (1)

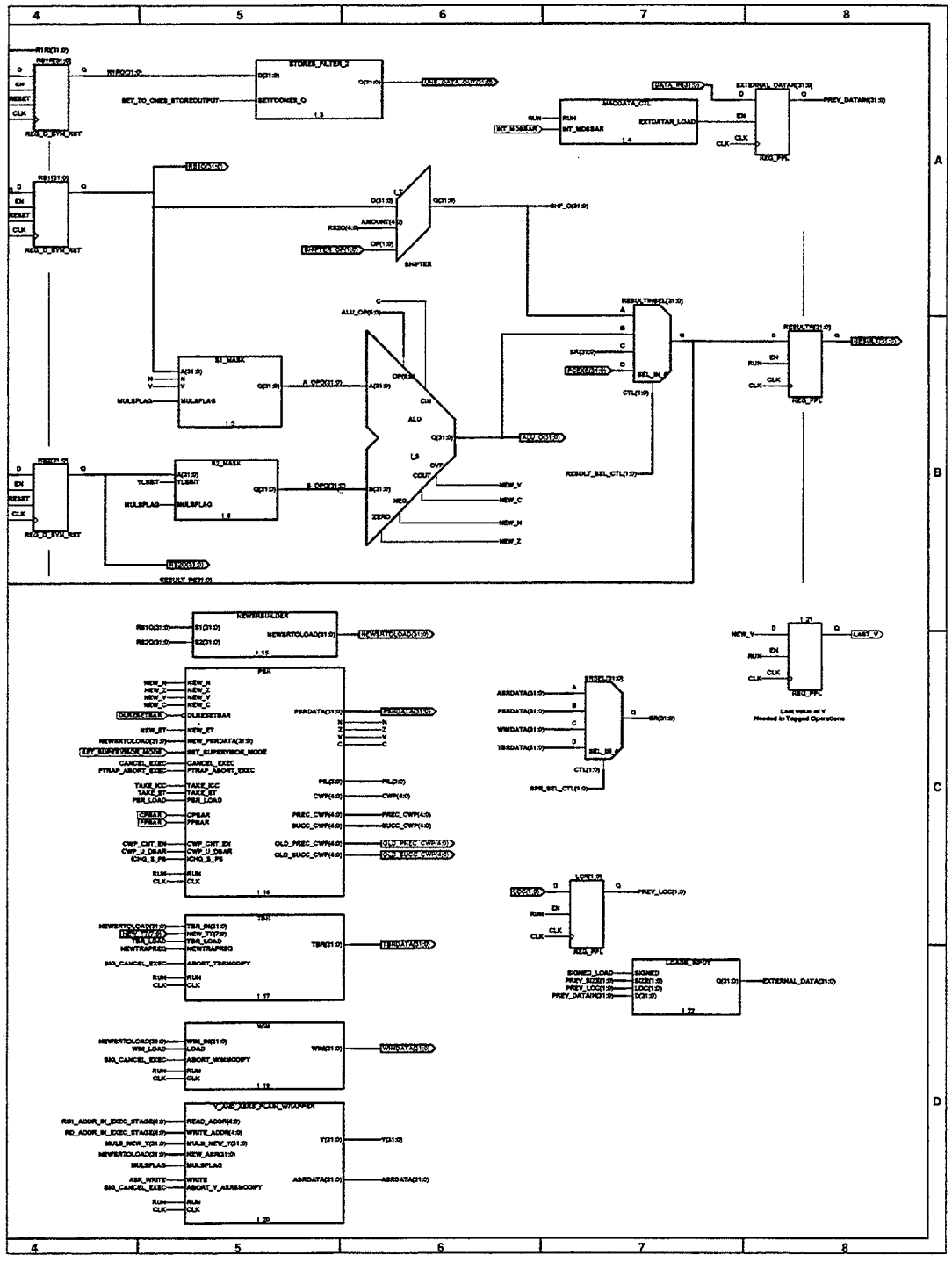


FIGURA B.8: Esquemático de la Ruta de Datos (2)

APÉNDICE C

Detalles de los modelos VHDL

Los modelos en VHDL de las distintas unidades que componen la descripción de la IU de SPARC han diseñado intentando hacerlos lo más independientes posibles de las herramientas empleadas. Sin embargo, en ocasiones esto no es completamente posible, y en tales ocasiones se debe tener especial cuidado al emplear estos modelos en otras herramientas distintas de aquellas con las que se han concebido.

En cualquier caso, con el objeto de cumplir en lo posible con algunos de los objetivos fijados en el apartado 4.2.3 de la pág. 46, en el modelado es muy conveniente la adopción de unas determinadas *reglas* que se deben mantener en todas las descripciones, como las que se han indicado anteriormente.

C.1. Definiciones generales

Este es el paquete donde se establecen los parámetros que identifican algunas de las características de la IU de SPARC v. 8.

```
1   LIBRARY ieee;
    USE ieee.std_logic_1164.all;
    USE ieee.std_logic_arith.all;

5   PACKAGE IUv8CustomParameters IS

      CONSTANT VERSION      : integer := conv_integer(unsigned'("0101"));
      CONSTANT IMPL         : integer := conv_integer(unsigned'("1010"));
```

```
    CONSTANT AdditionalStoreCycle  : boolean := false;
    CONSTANT NWindows     : natural := 7;
    CONSTANT RFregs       : natural := 8 + 16*NWindows;
    CONSTANT AvailableASRs: integer range 0 to 16 := 16;
    CONSTANT FirstPrivilegedASRAddress : std_logic_vector(4 downto 0)
            := "10000";

--CONSTANT tracing : boolean := true;

END IUv8CustomParameters;
```

De forma similar, se ha dispuesto en otro sitio todos aquellos parámetros que son específicos de la definición de la arquitectura y que no deben variar entre distintas implementaciones de ésta.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE SPARCv8 IS

  SUBTYPE quadword IS std_logic_vector(127 DOWNTO 0);     -- bit 127 is msb

  SUBTYPE doubleword IS std_logic_vector(63 DOWNTO 0);    -- bit  63 is msb

  SUBTYPE word IS std_logic_vector(31 DOWNTO 0);          -- bit  31 is msb
  --  Also called 'singleword'
  SUBTYPE halfword IS std_logic_vector(15 DOWNTO 0);      -- bit  15 is msb

  SUBTYPE byte IS std_logic_vector(7 DOWNTO 0);           -- bit   7 is msb

  CONSTANT YES      : std_logic := '1';
  CONSTANT NO       : std_logic := NOT YES;
  CONSTANT Occupied : std_logic := '1';
  CONSTANT Empty    : std_logic := NOT Occupied;
  CONSTANT HighLevel: integer range 0 to 1 := 1;
  CONSTANT LowLevel : integer range 0 to 1 := 1 - HighLevel;
  CONSTANT ENABLED  : std_logic := '1';
  CONSTANT DISABLED : std_logic := NOT ENABLED;
  CONSTANT LowActiveEnabled  : std_logic := '0'; -- Enable low-active Output
  CONSTANT LowActiveDisabled : std_logic := NOT LowActiveEnabled;

END SPARCv8;
```

C.2. Algunos elementos

En la descripción de algunos elementos, sobre todo en aquellos que pueden corresponderse con otros de igual comportamiento y tamaño de datos distinto, su descripción se ha realizado de forma que se facilite la reutilización del código. No se debe olvidar que el empleo de genéricos, si bien hubiera resultado deseable para este tipo de circunstancias, suele dar problemas en el entorno empleado.

```
1   -- VHDL Model Created from SGE Symbol adder_30.sym -- Mar  2 11:46:34 1999

    library IEEE;
    use IEEE.std_logic_1164.all;
5   use IEEE.std_logic_misc.all;
    use IEEE.std_logic_arith.all;
    use work.SPARCv8.all;
    use work.Gears.all;

10  entity ADDER_30 is
        Port (       A : In    std_logic_vector (29 downto 0);
                     B : In    std_logic_vector (29 downto 0);
                   CIN : In    std_logic;
                  COUT : Out   std_logic;
15                   Q : Out   std_logic_vector (29 downto 0) );
    end ADDER_30;

    architecture BEHAVIORAL of ADDER_30 is

20  SIGNAL result : std_logic_vector(q'length - 1 downto 0);

      begin

        result <= signed(a) + signed(b) + cin;
25      cout   <= (a(a'length - 1) AND b(b'length - 1)
                  AND NOT result(result'length - 1))
                    OR (NOT a(a'length - 1) AND NOT b(b'length - 1)
                  AND result(result'length - 1));
        q      <= result;
30
    end BEHAVIORAL;

    configuration CFG_ADDER_30_BEHAVIORAL of ADDER_30 is
       for BEHAVIORAL
35
       end for;

    end CFG_ADDER_30_BEHAVIORAL;
```

En otros casos, la descripción de algunos elementos está fuertemente sujeta a las posibilidades de las herramientas. De esta forma se fuerza que que la estructura materializada corresponda exactamente con lo que el diseñador desea.

```
1  -- VHDL Model Created from SGE Symbol hm_rfasynwrite_gen.sym
   -- Aug 17 11:44:16 1998

   library EPOCH_LIB;
5  library IEEE;
   use IEEE.std_logic_1164.all;
   use EPOCH_LIB.COMPONENTS.all;

   entity HM_RFASYNWRITE_GEN is
10       Port (     CLK : In    std_logic;
                 SYN_WRITE : In    std_logic;
                 ASYN_WRITE : Out   std_logic );
     attribute fixedblock : integer;
     attribute fixedblock of HM_RFASYNWRITE_GEN : entity is 0;
15 end HM_RFASYNWRITE_GEN;

   architecture BEHAVIORAL of HM_RFASYNWRITE_GEN is

   --pragma translate_off
20
   CONSTANT DPFLAG : INTEGER := 2;
   CONSTANT GROUP : STRING := "HM_RFASYNWRITE_GEN";
   CONSTANT BUFFER_SIZE : STRING := "DEFAULT";
   CONSTANT OPTIMIZE : STRING := "STANDARD";
25
   Signal ClkLatchEn, ClkBar : std_logic;

   Signal Clk_v, ClkBar_v, StClkDly1_v, StClkDly2_v, StClkDly3_v,
          StClkDly4_v, StClkDly5_v, StClkDly6_v,
30 --       StClkDly7_v, StClkDly8_v,
          WriteMask_v : std_logic_vector(0 to 0);

   Signal Syn_Write_v, Asyn_Write_v : std_logic_vector(0 to 0);

35    begin

      -- It is made with standard cells because there is no datapath
      -- version for the delay elements.

40    --  CLK                 """""""""""""|_____________
      --  CLKbar              _____________|"""""""""""""
      --  Stored CLK Dly 1    _|"""""""""""|___________
```

```
        --  Stored CLK Dly 2      ___|""""""""""""""|_________
        --  Write Mask            _______________|"""|________
45
        --  If a bigger width is needed insert more latches for getting the
        --  last Delayed StCLK * bar signal of the desired timing.

        -- With i_1, additional latches and the last latch it is
50      -- controlled the width of the pulse of the Internal Write
        -- signal

        Clk_v <= (OTHERS => Clk);
        Syn_Write_v <= (OTHERS => Syn_Write);
55      ClkLatchEn <= '1';

        i_1 : inv
          GENERIC MAP (
             N => 1,
60           DPFLAG => DPFLAG,
             GROUP => GROUP,
             BUFFER_SIZE => "AUTO",
             OPTIMIZE => "standard")
           PORT MAP (
65           INO => Clk_v,
             Y => ClkBar_v);

        ClkBar <= ClkBar_v(0);

70      i_2 : latch
          GENERIC MAP (
             N => 1,
             DPFLAG => DPFLAG,
             GROUP => GROUP,
75           BUFFER_SIZE => "AUTO",
             OPTIMIZE => "standard")
           PORT MAP (
             D => Clk_v,
             EN => ClkLatchEn,
80           Q => StClkDly1_v);

        i_3 : latch
          GENERIC MAP (
             N => 1,
85           DPFLAG => DPFLAG,
             GROUP => GROUP,
             BUFFER_SIZE => "AUTO",
             OPTIMIZE => "standard")
           PORT MAP (
90           D => StClkDly1_v,
             EN => ClkLatchEn,
```

```
            Q => StClkDly2_v);

        i_4 : latch
          GENERIC MAP (
             N => 1,
             DPFLAG => DPFLAG,
             GROUP => GROUP,
             BUFFER_SIZE => "AUTO",
             OPTIMIZE => "standard")
           PORT MAP (
             D => StClkDly2_v,
             EN => ClkLatchEn,
             Q => StClkDly3_v);

        i_5 : latch
          GENERIC MAP (
             N => 1,
             DPFLAG => DPFLAG,
             GROUP => GROUP,
             BUFFER_SIZE => "AUTO",
             OPTIMIZE => "standard")
           PORT MAP (
             D => StClkDly3_v,
             EN => ClkLatchEn,
             Q => StClkDly4_v);

        i_6 : latch
          GENERIC MAP (
             N => 1,
             DPFLAG => DPFLAG,
             GROUP => GROUP,
             BUFFER_SIZE => "AUTO",
             OPTIMIZE => "standard")
           PORT MAP (
             D => StClkDly4_v,
             EN => ClkLatchEn,
             Q => StClkDly5_v);

        i_7 : latch
          GENERIC MAP (
             N => 1,
             DPFLAG => DPFLAG,
             GROUP => GROUP,
             BUFFER_SIZE => "AUTO",
             OPTIMIZE => "standard")
           PORT MAP (
             D => StClkDly5_v,
             EN => ClkLatchEn,
             Q => StClkDly6_v);
```

```
        i_10 : latch_c
          GENERIC MAP (
             N => 1,
             DPFLAG => DPFLAG,
             GROUP => GROUP,
             BUFFER_SIZE => "AUTO",
             OPTIMIZE => "standard")
           PORT MAP (
             CLR => ClkBar,
             D => StClkDly6_v,
             EN => ClkLatchEn,
             Q => WriteMask_v);

        i_11 : and2
          GENERIC MAP (
             N => 1,
             DPFLAG => DPFLAG,
             GROUP => GROUP,
             BUFFER_SIZE => "AUTO",
             OPTIMIZE => "standard")
           PORT MAP (
             IN0 => WriteMask_v,
             IN1 => Syn_Write_v,
             Y   => Asyn_Write_v);

         Asyn_Write <= Asyn_Write_v(0);

    --pragma translate_on

    end BEHAVIORAL;

    --epoch translate_off

    configuration CFG_HM_RFASYNWRITE_GEN_BEHAVIORAL of HM_RFASYNWRITE_GEN is
       for BEHAVIORAL

       end for;

    end CFG_HM_RFASYNWRITE_GEN_BEHAVIORAL;

    --epoch translate_on
```

APÉNDICE D

Scripts de síntesis lógica

La síntesis lógica se ha realizado siguiendo una metodología en la que en una primera fase se estudia la cantidad de carga que deben tener cada uno de los elementos del diseño y después se realiza la optimización de todo el diseño.

Para esta síntesis se ha empleado un conjunto de *scripts*. Básicamente los fundamentales son dos, que son donde realmente se elaboran las dos fases mencionadas, y además se han elaborado otros de apoyo, que sirven para definir variables, establecer restricciones de síntesis, ponerle propiedades a algunos elementos y facilitar algunas tareas de la síntesis como el romper la jerarquía en los *designwares*.

A continuación se muestra el *script* para establecer variables en la síntesis. Este *script* tiene su correspondencia con el *script* genérico `iuv8_core_NN-setup.scr`, pero está personalizado para la versión `av753`.

```
1
    company = "CMA -- ULPGC"
    designer = "Tomas Bautista"

5   /* Compilation options */

    struct_boolean = false

    compile_fix_multiple_port_nets = false
10  port_complement_naming_style = "%s_SYNBAR"
    vhdlout_follow_vector_direction = true
```

```
infile_list = {../iu53v/iuv8_core_53v.vhd }

iu_core = IUV8_CORE_53V

root_design = IUV8_CORE_53V
list root_design
design_constraints = iuv8_core_XX.con
critical_blocks_script = iuv8_core_XX-critical_blocks.scr
addons = iuv8_core_XX-addons.scr
db_dump_file = syn-iuv8_core_53v.db
vhdl_dump_file = syn-iuv8_core_53v.vhd
results = syn-iuv8_core_53v.log
```

El siguiente *script* es el `iuv8_core_XX.con`, y con él se establecen las restricciones en la primera fase del diseño.

```
set_min_fault_coverage 98 -timing_critical -area_critical

set_operating_conditions -library "tsmc.6u3m1p_std_sd" "CDA_TYPICAL"

td_outputs = all_outputs()
td_inputs  = all_inputs()

/* if (design_ref == iu_core) {
  create_clock -period 8 -waveform {0.00 4.00} CLK
  set_dont_touch_network {CLK}
} else { */
  create_clock -period 0.04 -waveform {0.00 0.02} CLK
  set_dont_touch_network {CLK}
/* } */

set_min_delay 0.00 -rise -to td_outputs -from td_inputs
set_min_delay 0.00 -fall -to td_outputs -from td_inputs

if ((design_ref == DFF)  || (design_ref == REG_D) || \
    (design_ref == REG_CST) || (design_ref == REG_PPL)) {
  set_max_area 0
  set_max_delay 0.00 -rise -to td_outputs -from td_inputs
  set_max_delay 0.00 -fall -to td_outputs -from td_inputs
} else if (design_ref == CTLSGNGEN) {
  set_max_delay 6.00 -rise -to td_outputs -from td_inputs
  set_max_delay 6.00 -fall -to td_outputs -from td_inputs
  set_max_delay 0.50 -rise -to { "POPIQ" "RELBRSRCCTL" \
     "FOR_CONDTL_CTLTRANSFER" "S_LOAD_INSTRTODEC" "S_IDEC_SEL_CTL" \
```

```
        "RESETCTLBAR" } \
        -from td_inputs
    set_max_delay 0.50 -fall -to { "POPIQ" "RELBRSRCCTL" \
        "FOR_CONDTL_CTLTRANSFER" "S_LOAD_INSTRTODEC" "S_IDEC_SEL_CTL" \
        "RESETCTLBAR" } \
        -from td_inputs
    set_max_delay 0.00 -rise -to { "PUSHIQ" } \
        -from td_inputs
    set_max_delay 0.00 -fall -to { "PUSHIQ" } \
        -from td_inputs
    set_max_delay 0.00 -rise -to { "FOR_UNCONDTL_CTLTRANSFER"  "FOR_RD_ADDR*" \
        "RELBRSRCCTL" "TRAPACK" } \
        -from td_inputs
    set_max_delay 0.00 -fall -to { "FOR_UNCONDTL_CTLTRANSFER"  "FOR_RD_ADDR*" \
        "RELBRSRCCTL" "TRAPACK" } \
        -from td_inputs
} else if (design_ref == CTL_FLOW_MGNT) {
    set_max_area 0
    set_max_delay 4.50 -rise -to td_outputs -from td_inputs
    set_max_delay 4.50 -fall -to td_outputs -from td_inputs
    set_max_delay 0.00 -rise -to { "OFFSET*" "INCPC" }  \
        -from { "ICC_TEST_EVAL_DEC"  "ICC_TEST_EVAL_EXEC" }
    set_max_delay 0.00 -fall -to { "OFFSET*" "INCPC" } \
        -from { "ICC_TEST_EVAL_DEC"  "ICC_TEST_EVAL_EXEC" }
} else if (design_ref == PC_MGNT) {
    /* set_max_area 0 */
    set_max_delay 5.50 -rise -to td_outputs -from td_inputs
    set_max_delay 5.50 -fall -to td_outputs -from td_inputs
    set_max_delay 0.00 -rise -to td_outputs  \
        -from { "RELBRSRCCTL" "FOR_CONDTL_CTLTRANSFER" \
                "FOR_UNCONDTL_CTLTRANSFER" "ICC_TEST_EVAL_EXEC" \
                "ICC_TEST_EVAL_DEC" }
    set_max_delay 0.00 -fall -to td_outputs \
        -from { "RELBRSRCCTL" "FOR_CONDTL_CTLTRANSFER" \
                "FOR_UNCONDTL_CTLTRANSFER" "ICC_TEST_EVAL_EXEC" \
                "ICC_TEST_EVAL_DEC" }
} else if ((design_ref == PSR_INPUTS_COLLECTOR) || (design_ref == PSR)) {
    set_max_area 0
    set_max_delay 3.00 -rise -to td_outputs -from td_inputs
    set_max_delay 3.00 -fall -to td_outputs -from td_inputs
    set_max_delay 0.00 -rise -to td_outputs \
        -from { "CANCEL_EXEC" "FTRAP_ABORT_EXEC" }
    set_max_delay 0.00 -fall -to td_outputs \
        -from { "CANCEL_EXEC" "FTRAP_ABORT_EXEC" }
} else if (design_ref == TBR) {
    set_max_area 0
    set_max_delay 2.00 -rise -to td_outputs -from td_inputs
    set_max_delay 2.00 -fall -to td_outputs -from td_inputs
    set_max_delay 0.00 -rise -to td_outputs \
```

```
      -from { "ABORT_TBRMODIFY" }
    set_max_delay 0.00 -fall -to td_outputs \
      -from { "ABORT_TBRMODIFY" }
  } else if (design_ref == LOAD_PCS_MASK) {
    set_max_area 0
    set_max_delay 1.00 -rise -to td_outputs -from td_inputs
    set_max_delay 1.00 -fall -to td_outputs -from td_inputs
    set_max_delay 0.00 -rise -to td_outputs \
      -from { "NEWTRAPREQ" }
    set_max_delay 0.00 -fall -to td_outputs \
      -from { "NEWTRAPREQ" }
  } else if (design_ref == NEXT_PC_BUILDER_2) {
    set_max_area 0
    set_max_delay 3.00 -rise -to td_outputs -from td_inputs
    set_max_delay 3.00 -fall -to td_outputs -from td_inputs
    set_max_delay 0.00 -rise -to td_outputs \
      -from { "LOAD_PCS" }
    set_max_delay 0.00 -fall -to td_outputs \
      -from { "LOAD_PCS" }
  } else if ((design_ref == Y_AND_ASRS) || (design_ref == Y_AND_ASRS_WRAPPER)) {
    set_max_area 0
    set_max_delay 2.50 -rise -to td_outputs -from td_inputs
    set_max_delay 2.50 -fall -to td_outputs -from td_inputs
    set_max_delay 0.00 -rise -to td_outputs \
      -from { "ABORT_Y_ASRSMODIFY" }
    set_max_delay 0.00 -fall -to td_outputs \
      -from { "ABORT_Y_ASRSMODIFY" }
  } else if (design_ref == WIM) {
    set_max_area 0
    set_max_delay 2.00 -rise -to td_outputs -from td_inputs
    set_max_delay 2.00 -fall -to td_outputs -from td_inputs
    set_max_delay 0.00 -rise -to td_outputs \
      -from { "ABORT_WIMMODIFY" }
    set_max_delay 0.00 -fall -to td_outputs \
      -from { "ABORT_WIMMODIFY" }
  } else if (design_ref == TRAPS_ADVISOR) {
    set_max_area 0
    set_max_delay 3.00 -rise -to td_outputs \
                             -from td_inputs
    set_max_delay 3.00 -fall -to td_outputs \
                             -from td_inputs
    set_max_delay 0.00 -rise -to td_outputs \
                             -from { "NEWCWPTOLOAD*" "ICC_TEST_EVAL" "ADDS1S2*" }
    set_max_delay 0.00 -fall -to td_outputs \
                             -from { "NEWCWPTOLOAD*" "ICC_TEST_EVAL" "ADDS1S2*" }
  } else if (design_ref == ICC_TEST) {
    /* set_max_area 0 */
    set_max_delay 0.00 -rise -to td_outputs -from td_inputs
    set_max_delay 0.00 -fall -to td_outputs -from td_inputs
```

```
    } else if ((design_ref == VIS_ADDERS_RES_BUILDER) \
          || (design_ref == VIS_CMP)) {
      set_max_area 0
130   set_max_delay 3.00 -rise -to td_outputs -from td_inputs
      set_max_delay 3.00 -fall -to td_outputs -from td_inputs
      set_max_delay 0.00 -rise -to td_outputs \
                               -from { "ADDERSQ*" "CO*" }
      set_max_delay 0.00 -fall -to td_outputs \
135                            -from { "ADDERSQ*" "CO*" }
    } else if (design_ref == VIS_MULTS_RES_BUILDER) {
      set_max_area 0
      set_max_delay 3.00 -rise -to td_outputs \
                               -from { "OP*" }
140   set_max_delay 3.00 -fall -to td_outputs \
                               -from { "OP*" }
      set_max_delay 0.00 -rise -to td_outputs \
                               -from { "PROD*" }
      set_max_delay 0.00 -fall -to td_outputs \
145                            -from { "PROD*" }
    } else if (design_ref == IU_CONTROLPATH) {
      set_max_area 0
      set_max_delay 0.00 -rise -to td_outputs -from td_inputs
      set_max_delay 0.00 -fall -to td_outputs -from td_inputs
150 } else if (design_ref == IU_DATAPATH) {
      set_max_area 0
      /*
      uniquify -reference "WINREGSMGMNT_READ"
      uniquify -reference "WINREG_SRCSELECT"
155   */
      set_max_delay 0.00 -rise -to td_outputs -from td_inputs
      set_max_delay 0.00 -fall -to td_outputs -from td_inputs
    } else {
      set_max_area 0
160   set_max_delay 0.00 -rise -to td_outputs -from td_inputs
      set_max_delay 0.00 -fall -to td_outputs -from td_inputs
    }
```

En `iuv8_core_XX-addons.scr` se establecen algunas propiedades que facilitan determinadas tareas en la síntesis. He aquí cómo se ha realizado este *script*

```
1
    if ((design_ref == EP_4ALU_MUL8) || \
        (design_ref == EP_ADDER_16) || \
        (design_ref == EP_ADDER_3) || \
5       (design_ref == EP_ADDER_30) || \
```

```
        (design_ref == EP_ADDER_32) || \
        (design_ref == EP_ADDER_7) || \
        (design_ref == EP_ADDER_8) || \
        (design_ref == EP_ALU_32) || \
10      (design_ref == EP_ALU_EXT_32) || \
        (design_ref == EP_ASRS) || \
        (design_ref == EP_MULT8X16) || \
        (design_ref == EP_NEXT_PC_BUILDER) || \
        (design_ref == EP_NEXT_PC_BUILDER_2) || \
15      (design_ref == EP_NEXT_PC_BUILDER_2_B) || \
        (design_ref == EP_OPERAND_GO_AND_BYPASS) || \
        (design_ref == EP_REG_30) || \
        (design_ref == EP_REG_32) || \
        (design_ref == EP_REG_32_FOR_S1) || \
20      (design_ref == EP_REG_32_FOR_S2) || \
        (design_ref == EP_REG_32_SB) || \
        (design_ref == EP_REG_5) || \
        (design_ref == EP_REG_5_SB) || \
        (design_ref == EP_REG_8) || \
25      (design_ref == EP_REG_C_30) || \
        (design_ref == EP_REG_C_32) || \
        (design_ref == EP_REGFILE16X32) || \
        (design_ref == EP_REGFILE16X32_SB) || \
        (design_ref == EP_REGFILE2R48X32) || \
30      (design_ref == EP_REGFILE2R48X32_SB) || \
        (design_ref == EP_REGFILE2R64X32) || \
        (design_ref == EP_REGFILE2R64X32_SB) || \
        (design_ref == EP_REGFILE2R8X32) || \
        (design_ref == EP_REGFILE2R8X32_SB) || \
35      (design_ref == EP_REGFILE32X32) || \
        (design_ref == EP_RFASYNWRITE_GEN) || \
        (design_ref == EP_RFASYNWRITE_GEN_SB) || \
        (design_ref == EP_S1_MASK) || \
        (design_ref == EP_S1_SEL) || \
40      (design_ref == EP_S2_MASK) || \
        (design_ref == EP_S2_SEL) || \
        (design_ref == EP_SEL_IN_2X30) || \
        (design_ref == EP_SEL_IN_2X32) || \
        (design_ref == EP_SEL_IN_2X32_SB) || \
45      (design_ref == EP_SEL_IN_2X32_WITH_ZERO_MASK) || \
        (design_ref == EP_SEL_IN_2X8) || \
        (design_ref == EP_SEL_IN_3X32) || \
        (design_ref == EP_SEL_IN_4X32) || \
        (design_ref == EP_SHIFTER_32) || \
50      (design_ref == EP_WINDOWEDREGISTERFILE) || \
        (design_ref == EP_Y_AND_ASRS) || \
        (design_ref == EP_ZEROES_MASK_32) || \
        (design_ref == EP_ZEROES_MASK_5) || \
        (design_ref == EP_ZEROES_SETTER) || \
```

```
55        (design_ref == NE_OR2) || \
          (design_ref == NE_XOR2) || \
          (design_ref == NE_AND2) || \
          (design_ref == NE_OR3) || \
          (design_ref == INVERTER) || \
60        (design_ref == NE_DFF) || \
          (design_ref == REG_D) || \
          (design_ref == REG_CST) || \
          (design_ref == REG_PPL) || \
          (design_ref == REG_DATA_C) || \
65        (design_ref == REG_D_WITH_MASK) || \
          (design_ref == REG_D_SYN_RST) || \
          (design_ref == ZERO_MASK) || \
          (design_ref == BRIDGE) || \
          (design_ref == PC_REL_SRC) || \
70        (design_ref == PC_REL_SRC_2) || \
          (design_ref == MAOINSTR_CTL) || \
          (design_ref == IQRSCTL) || \
          (design_ref == IQRSCTL_2) || \
          (design_ref == PREDICTOR) || \
75        (design_ref == PREDICTOR_B) || \
          (design_ref == PREDICTOR_C) || \
          (design_ref == PRDT_REFUTER) || \
          (design_ref == DELAY_SLOT_MNGT) || \
          (design_ref == NEXT_PC_MNGT) || \
80        (design_ref == NEXT_PC_MNGT_B) || \
          (design_ref == SGNBLOCKADE) || \
          (design_ref == BIT_GENERATOR) || \
          (design_ref == BITS_GENERATOR_2) || \
          (design_ref == BITS_GENERATOR_4) || \
85        (design_ref == BITS_GENERATOR_5) || \
          (design_ref == BITS_GENERATOR_6) || \
          (design_ref == BITS_GENERATOR_8) || \
          (design_ref == BITS_GENERATOR_16) || \
          (design_ref == BITS_GENERATOR_32) || \
90        (design_ref == SEL_IN_2) || \
          (design_ref == SEL_IN_3) || \
          (design_ref == SEL_IN_4) || \
          (design_ref == SEL_IN_8) || \
          (design_ref == THREE_STATE_BUFFER) || \
95        (design_ref == PSR_OUTPUTS) || \
          (design_ref == PSR_INPUTS_COLLECTOR) || \
          (design_ref == CWP_MGNT) || \
          (design_ref == ADDER_30)) {
            set_ungroup this_design true
100   }

      /* The CTL_FLOW_MGNT and PC_BUILDER blocks should
         be not ungrouped when using Epoch parts */
```

```
105   set_structure true -boolean struct_boolean -timing true

      if (design_ref == IU_CONTROLPATH) {
        uniquify -reference "ICC_TEST"
      } else if ((design_ref == WINDOWEDREGISTERFILE) || \
110               (design_ref == WINDOWEDREGISTERFILE_PLAIN) || \
                  (design_ref == WINDOWEDREGISTERFILE_2W) || \
                  (design_ref == WINDOWEDREGISTERFILE_2W_PLAIN) || \
                  (design_ref == WINDOWEDREGISTERFILE_4W) || \
                  (design_ref == WINDOWEDREGISTERFILE_4W_PLAIN)) {
115     uniquify -reference "WINREGSMGMNT"
        uniquify -reference "WINREGSMGMNT_2W"
        uniquify -reference "WINREGSMGMNT_4W"
        uniquify -reference "WINREGSMGMNT_READ"
        uniquify -reference "WINREG_SRCSELECT"
120   }
```

Con `ungroup_dwares.scr` se rompe la jerarquía en los *designwares*:

```
1
      dware_list = find(design *DW* -hierarchy)

      if (dware_list != {}) {
5       foreach (dware, dware_list) {
            dware_cells = filter (find (-hierarchy cell "*") "@ref_name == dware")
            ungroup dware_cells -flatten
        }
        dwares_found = 1
10    }
```

Este *script* `iuv8_core_XX-critical_blocks.scr` sirve para identificar los elementos cuya síntesis no se debe alterar al realizar la de aquellos otros elementos en los que están inmersos:

```
1
      if ((this_design == VIS_ADDERS_RES_BUILDER) \
       || (this_design == VIS_ADDERS_OPSMOD) \
       || (this_design == VIS_MULTS_RES_BUILDER) \
```

```
5     || (this_design == VIS_MULTS_OPSMOD) \
      || (this_design == VIS_CMP)) \
      {
        critical_block = 1
      }
```

De las dos fases de que consta la síntesis lógica, este *script* es el que permite realizar la primera fase. Este *script* es el `iuv8_core_NN-pass-01.scr` personalizado para una versión determinada.

```
1
      include iuv8_core_53v-setup.scr

      analyze -f vhdl infile_list
5
      /*read -f vhdl infile_list*/

      elaborate root_design -update
      if (dc_shell_status != 1) {
10       quit
      }

      check_design
      check_timing
15
      current_design root_design
      list current_design
      design_list = find(design -hierarchy)
      list design_list
20
      echo Now we'll run through the hierarchy for the first time...

      foreach (this_design, design_list) {
              current_design this_design
25
              design_ref = get_attribute(this_design hdl_template)

              include design_constraints

30            include addons

              compile -map_effort low -boundary_optimization -incremental

              /* And now let's check for designwares */

35
```

```
        dwares_found = 0

        include ungroup_dwares.scr

        if (dwares_found == 1) {
           compile -map_effort medium -incremental
        }
}

current_design root_design
design_ref = get_attribute(root_design hdl_template)
include addons
include design_constraints
compile -boundary_optimization -incremental
write -f db -hierarchy -o db_dump_file

characterize(-no_timing -connections -constraints find (cell -hierarchy))

foreach (this_design, design_list) {
        current_design this_design
        write_script > this_design + .wscr
}

quit
```

Con el *script* siguiente se elabora la segunda y última fase de la síntesis lógica. Corresponde con el `iuv8_core_NN-pass-02.scr`, modificado para una versión determinada.

```
1
   include iuv8_core_53v-setup.scr

   /*
5
   analyze -f vhdl infile_list

   read -f vhdl infile_list

10 */

   elaborate root_design -update
   if (dc_shell_status != 1) {
      quit
```

```
}

check_design
check_timing

current_design root_design
list current_design
design_list = find(design -hierarchy)
list design_list

echo Now we'll run through the hierarchy for the second time...

foreach (this_design, design_list) {

        current_design this_design

        design_ref = get_attribute(this_design hdl_template)

        include this_design + .wscr

        /* include design_constraints */

        include addons

        compile -map_effort low -incremental

        dwares_found = 0

        /* If you wish certain blocks not to be touched, this is the way... */

        critical_block = 0

        include ungroup_dwares.scr

        include critical_blocks_script

        compile -map_effort high -boundary_optimization -incremental

        /* If wished, you can consider to set the dont_touch attribute of
           critical blocks right here */

        if (critical_block == 1) {
           set_dont_touch this_design
        }

        /* write -f db */

        report_area >> results
        report_cell >> results
```

```
        report_timing >> results
}

current_design root_design
design_ref = get_attribute(root_design hdl_template)
include addons
include design_constraints

if (root_design == iu_core) {
   set_output_delay -fall 1 all_outputs() -clock CLK
   set_output_delay -rise 1 all_outputs() -clock CLK
   set_output_delay -fall 2 WEBAR -clock CLK
   set_output_delay -rise 0 WEBAR -clock CLK
}

compile -map_effort high -boundary_optimization -incremental

if (dc_shell_status != 1){
   quit
}

write -f db -hierarchy -output db_dump_file
write -f vhdl -hierarchy -output vhdl_dump_file

report_area >> results
report_cell >> results
report_timing >> results

quit
```