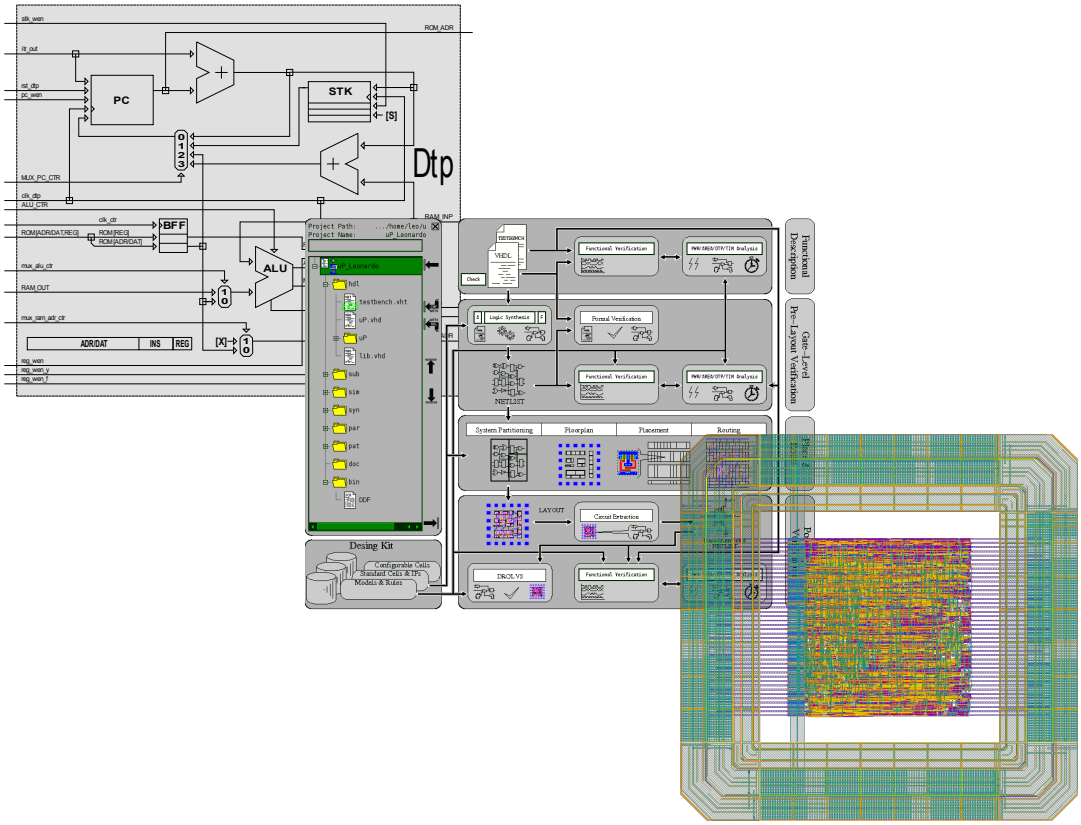
SÍNTESIS DE UN MICROPROCESADOR PARA SU FABRICACIÓN EN TECNOLOGÍA *CMOS*.



Jaime Leonardo Mendoza Cañas

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

UNIVERSIDAD INDUSTRIAL SANTANDER

Bucaramanga – 2010

UNIVERSIDAD INDUSTRIAL DE SANTANDER
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

SÍNTESIS DE UN MICROPROCESADOR PARA SU FABRICACIÓN EN TECNOLOGÍA *CMOS*

Jaime Leonardo Mendoza Cañas

Trabajo de grado para optar por el título de Ingeniero Electrónico

Director

MSc. Élkim Felipe Roa Fuentes

Codirector

MSc. Hugo Daniel Hernández Herrera

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

UNIVERSIDAD INDUSTRIAL SANTANDER

Bucaramanga–Agosto de 2010

“ A mi madre,
por su apoyo y comprensión.”

Agradecimientos

Quiero agradecer a toda mi familia, en especial a mi madre y mis abuelos quienes me colaboraron cuando más lo necesité, y a Alfonso por su ayuda y sus consejos.

Al profesor Elkim por confiar en mis habilidades e introducirme al mundo del diseño de circuitos integrados. A Hugo y Armando que estuvieron pendientes de este trabajo y me guiaron cuando fue necesario. A mis compañeros del grupo *CIDIC*, de los que aprendí mucho y quienes colaboraron de manera directa o indirecta al desarrollo de este proyecto.

A mis amigos, con quienes compartí mucho durante esta etapa de mi vida y me ayudaron a distraerme cuando fue preciso.

A la UIS por permitirme subir un escalón más en mi desarrollo personal. Y en general a todos los que hicieron posible, de una u otra manera, la finalización de este trabajo.

Contenido

Lista de figuras

Lista de tablas

RESUMEN

TÍTULO: SÍNTESIS DE UN MICROPROCESADOR PARA SU FABRICACIÓN EN TECNOLOGÍA *CMOS*[1]

AUTOR: JAIME LEONARDO MENDOZA CAÑAS[2]

PALABRAS CLAVE: Microprocesador, síntesis lógica, trazado automático, bajo consumo de potencia, *layout*, *CMOS*.

DESCRIPCIÓN:

El continuo aumento en el uso y las aplicaciones de los sistemas digitales, ha repercutido considerablemente en el avance de la ciencia y la tecnología. Este progreso ha impactado favorablemente sobre los costos de producción y la calidad de los circuitos integrados, permitiendo el continuo desarrollo del área.

Ahora bien, el diseño de un sistema digital integrado es un proceso complejo susceptible a los fallos humanos, por lo que la industria siempre ha tratado de automatizarlo mediante herramientas de software. Bajo este contexto, el presente proyecto se orienta a la implementación de metodologías y herramientas profesionales que permitan el desarrollo de sistemas digitales complejos, como los procesadores, y su integración en la tecnología *AMS C35B4C3*.

Inicialmente se presentan las metodologías usadas en el diseño de circuitos integrados y su objetivo. Seguidamente, se plantea el diseño configurable de un microprocesador mediante el uso del lenguaje de descripción de hardware *VHDL*, partiendo de un diseño previamente realizado. Se propone el uso de múltiples fuentes de reloj y la reducción en la complejidad para reducir el consumo de potencia y elevar la eficiencia. El diseño cuenta además, con una pila y un módulo de control de interrupciones.

Para reducir el tiempo de desarrollo, se programa un entorno de software que integra las herramientas necesarias para el diseño y verificación de cualquier sistema digital. Posteriormente, se realiza la síntesis lógica del microprocesador, el posicionamiento de elementos, el trazado de interconexiones y la verificación funcional del mismo. Finalmente, se realiza un análisis de los parámetros de desempeño del microprocesador para diferentes configuraciones del diseño.

[1] Proyecto de Grado.

[2] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director MSc. Élkim Felipe Roa Fuentes. Codirector MSc. Hugo Daniel Hernández Herrera.

SUMMARY

TITLE: SYNTHESIS OF A MICROPROCESSOR TO MANUFACTURE IN CMOS TECHNOLOGY. [3]

AUTHOR: JAIME LEONARDO MENDOZA CAÑAS[4]

KEY WORDS: Microprocessor, logic synthesis, automatic routing, low power, layout, CMOS.

DESCRIPTION:

The continuous increase in the use and applications of digital systems, has considerably affected the progress of science and technology. This progress has impacted positively on production costs and quality of integrated circuits, enabling the continued development of the area.

However, the design of an integrated digital system is a complex process susceptible to human error. That is the reason why the industry has always tried to automate it using software tools. In this context, this project aims at the implementation of methodologies and tools that allow professional development of complex digital systems, such as processors, and its integration in the *AMS C35B4C3* technology.

Initially the methodologies used in integrated circuit design and it's purpose are presented. Then, a design of a microprocessor configurable is proposed by using the hardware description language VHDL, from a previously made design. Multiple clock sources and reducing the complexity to reduce power consumption and increase efficiency are implemented. The design also has an interrupt control module and a stack.

To reduce development time, a software environment that integrates the necessary tools for the design and verification of any digital system is programed. Subsequently, the microprocessor logic synthesis, placement of elements, routing of interconnections and functional verification is applied. Finally, an analysis of performance parameters for different configurations of the microprocessor design is accomplished.

[3] Degree project.

[4] Physics Mechanical Engineering Faculty. Electric, Electronic and Telecommunications School. Director MSc. Élkim Felipe Roa Fuentes. Codirector MSc. Hugo Daniel Hernández Herrera.

Capítulo 1

Introducción

1.1. Motivación

Con la aparición del procesador y la popularización de los ordenadores en los años 80, la evolución de los sistemas digitales integrados ha estado en constante evolución, llegando a ser en la actualidad una de las áreas más destacadas en la industria de semiconductores. Esto se ve reflejado en la implementación del procesamiento de datos en nuevos campos, donde se hace indispensable incluir esta tecnología para mejorar características y adicionar nuevas funcionalidades. A nivel industrial, las aplicaciones digitales han permitido la automatización de la mayoría de los procesos de producción, reduciendo los costos y el tiempo de fabricación, a la vez que se eleva la calidad de los productos.

Todo este progreso se ha conseguido gracias al desarrollo de nuevas tecnologías para la fabricación de circuitos integrados, aumentando año tras año la *capacidad de integración*[1] y procesamiento. Estos adelantos reducen el costo final y el consumo de potencia del sistema, incrementado la velocidad y favoreciendo el desarrollo de productos de mayor calidad a precios más bajos. Dichas ventajas provocaron una competencia entre las empresas del sector, elevando los niveles de integración a valores inimaginables en los comienzos de esta industria. Gracias a esta evolución, el mercado de sistemas digitales ha crecido constantemente, poniendo a disposición de más personas las comodidades de la vida moderna.

Ahora bien, por la gran cantidad de elementos constitutivos, miles de millones inclusive, el diseño de un circuito integrado a la manera tradicional es una tarea ardua y tediosa, por lo que cumplir los estándares de calidad en el tiempo establecido resulta inviable. Además, la

[1] Una medida representativa de la cantidad de componentes que se puede integrar mediante un determinado proceso de fabricación.

naturaleza repetitiva del proceso lo hace susceptible a fallos humanos, y la complejidad del sistema dificulta la verificación funcional previa a la fabricación. Por otro lado, si el proceso de fabricación cambia, se requiere de una gran cantidad de recursos para adaptar el diseño. Para sortear estos obstáculos, es necesario implementar metodologías de diseño que permitan organizar y automatizar el desarrollo de un chip.

La tecnología *CMOS* es actualmente la más usada en el desarrollo de sistemas digitales, ya que a diferencia de otras, como la bipolar, *GaAs*, *BiCMOS* etc., su fabricación es relativamente sencilla y permite implementar circuitos complejos a precios reducidos. Además, debido a su bajo consumo de potencia, es la opción más indicada para desarrollar aplicaciones portátiles [1]. Además, la alta impedancia de entrada y la reducida resistencia de canal que se puede alcanzar, hacen de este proceso una alternativa muy atractiva para el diseño de circuitos analógicos. Esta característica permite integrar procesamiento digital junto con circuitos analógicos dentro de un mismo chip [2].

Bajo este contexto, el presente proyecto se orienta a la implementación de metodologías y herramientas profesionales que permitan el desarrollo de sistemas digitales complejos, como los procesadores, y su integración en la tecnología *AMS C35B4C3*.

1.2. Organización del documento

En el presente capítulo se aborda el diseño de los circuitos integrados desde el punto de vista metodológico y su impacto en la productividad. En el capítulo 2 se detalla el diseño del procesador y las consideraciones que deben tomarse para la formulación del mismo. Por último, en el capítulo 3 se exponen los resultados de aplicar el flujo de diseño y la metodología al procesador.

1.3. Metodologías de diseño de circuitos integrados

Con el fin de estructurar el proceso de diseño y obtener buenos resultados, se requiere de una metodología que permita la organización de las ideas del diseñador y su trabajo. Con este objetivo se presenta a continuación las diferentes herramientas conceptuales y los procedimientos usados en el diseño de circuitos integrados.

1.3.1. Los dominios conceptuales

El diseño de circuitos electrónicos abarca tres dominios bien definidos, como lo hicieron notar Gajski y Kahn en sus diagramas [3]. En la figura 1.1 se puede observar la condensación de los mismos; cada dominio se encuentra subdividido por varios niveles de abstracción, cuanto menor el nivel de abstracción, mayor es la complejidad en la descripción del circuito y su análisis, y viceversa. La transición desde una descripción de menor complejidad a una de mayor, dentro de un mismo dominio, se denomina refinamiento y al proceso inverso, abstracción. La mejora en las características de un sistema o bloque se designa optimización y ocurre en el mismo nivel de abstracción.

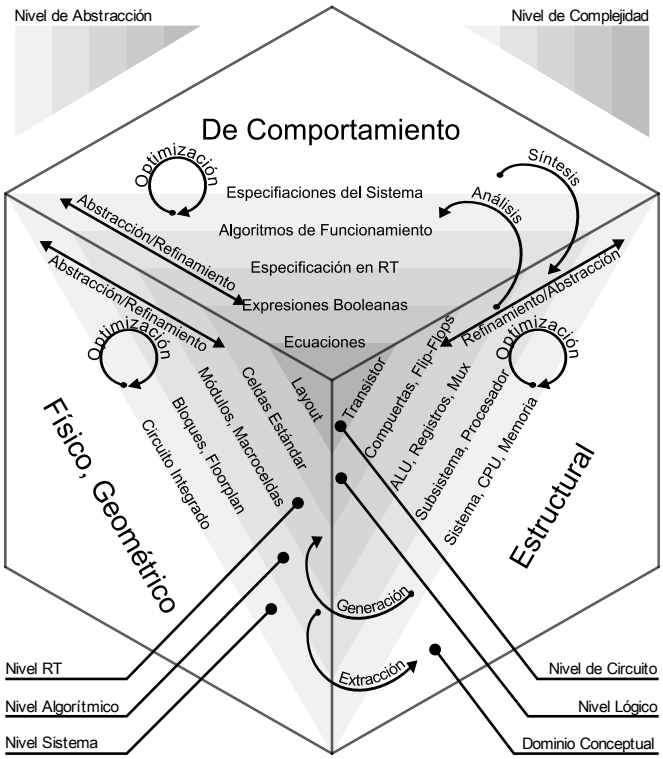


Figura 1.1: Diagrama de los dominios conceptuales en el área de diseño de circuitos integrados.

A continuación se presenta un acercamiento a los dominios conceptuales y sus niveles de abstracción, orientándolo a los sistemas digitales.

Dominio del comportamiento

En este dominio se determina el comportamiento del sistema a diseñar, es decir su funcionamiento.

- **Especificaciones:** Todo diseño de un sistema electrónico comienza desde el planteamiento de un problema, que puede ser resuelto mediante la implementación de éste. Dicho problema se posiciona en el dominio del comportamiento una vez que se definen las especificaciones del sistema. Tales especificaciones son las características que debe cumplir el circuito a diseñar y son representaciones generales del comportamiento del sistema. Algunas de ellas pueden ser flexibles o fijas, pero se requiere definir la cantidad suficiente, con el fin de solucionar el problema planteado mediante un dispositivo que las cumpla. Un ejemplo de especificación fija es el tamaño del bus de datos de un procesador y una restricción flexible es la reducción en su consumo de potencia. A nivel industrial, el conjunto de especificaciones que se definen para los elementos de un sistema se suele denominar estándar.

- **Algoritmos de funcionamiento:** Refinar las especificaciones implica determinar una serie de estímulos que se introducirán al circuito y una serie de salidas esperadas que permitirán decidir si éste cumple con las especificaciones. El conjunto anterior se denomina banco de pruebas o *TestBench* y al proceso *verificación funcional*. Es posibles describir el comportamiento del sistema mediante un algoritmo de funcionamiento, que puede ser programado mediante un lenguaje de descripción de hardware, lo que se denomina *descripción funcional*.

- **Especificación en transferencia de registros, *RT*:** Refinando la descripción funcional se puede obtener una representación más precisa del circuito que se diseña. Esta descripción se caracteriza por separar los elementos combinacionales de los de almacenamiento. La anterior es la estructura recomendada para el diseño de circuitos digitales mediante el uso de herramientas de síntesis lógica, ya que reduce las ambigüedades generando así un circuito de comportamiento más predecible, porque se puede seguir con facilidad la ruta que toman los datos en cada instante de tiempo.

- **Expresiones Booleanas:** Refinando la especificación en transferencia de registros, se puede obtener una descripción del sistema más detallada. Ésta se define con una serie de expresiones booleanas o con tablas de verdad.

- **Ecuaciones diferenciales:** Este es el mayor nivel de complejidad en el dominio del comportamiento, y corresponde a los modelos de transistores y los elementos de circuito. Éste es el campo de trabajo que tradicionalmente se usa en el diseño de circuitos analógicos, aunque dicha tendencia ha cambiado en los últimos años con la creación de lenguajes de descripción de *hardware* analógico y de señal mezclada.

Dominio estructural

En este dominio se definen los bloques que componen un sistema y cómo se interconectan.

- **Sistema:** Es la representación de un circuito completo con sus entradas y salidas.
- **Subsistema:** En este nivel de abstracción se divide el circuito en bloques generales.
- **Bloques funcionales:** En este nivel de abstracción se continúa con la división del subsistema en sus bloques constitutivos.
- **Compuertas y celdas básicas:** Aquí se subdivide el sistema en los bloques básicos con los que se construye todo circuito.
- **Transistores y elementos de circuito:** En este nivel de complejidad se trabaja con los elementos básicos y se define la interconexión de los mismos para conformar cada compuerta y celda.

Dominio físico

En este dominio se representa el circuito mediante los elementos que definen su fabricación.

- **Circuito integrado:** Aquí se tiene el circuito empaquetado y fabricado.
- **Distribución del área, *Floorplan*:** En esta descripción se tiene el área que ocupa cada bloque constitutivo y la distribución del mismo dentro del circuito integrado.
- **Macroceldas:** En este nivel de abstracción se detalla la distribución de área de los subbloques.
- **Celdas básicas:** Se define el área de cada celda lógica básica y su posición en el chip.
- **Trazado o *Layout*:** Se describe el trazado de cada una de las capas que define cada elemento de circuito y que son necesarias para la fabricación del circuito integrado. Este es el punto final de todo diseño.

Diseñar un circuito integrado, implica el desplazamiento por cada uno de los dominios comenzando por el de comportamiento y finalizando en el físico. Para ello se deben realizar las transiciones respectivas que se encuentran definidas a continuación:

- **Comportamiento > Estructural:** *síntesis.*
- **Estructural > Comportamiento:** *análisis.*
- **Estructural > Físico:** *generación.*

- **Físico > Estructural:** *extracción.*

1.3.2. Flujos de diseño

A mayor complejidad de los sistemas electrónicos, mayor es el tiempo requerido para su diseño, lo cual entra en conflicto con el *time to market*[2]. En los comienzos de la industria electrónica, cada circuito debía ser diseñado y trazado a mano, pues se carecía de herramientas que facilitaran estas tareas. El avance tecnológico dado gracias a la invención del microprocesador (μP), permitió el desarrollo de metodologías y herramientas de diseño asistido por computador, *CAD*[3], enfocadas a reducir el tiempo diseño y los costos, aumentando la fiabilidad. [4]

La alta demanda de procesamiento de datos, permitió que las herramientas CAD para diseño digital maduraran rápidamente. En este campo, se ha diseñado una gran cantidad de bloques lógicos que al interconectarse entre sí, permiten la generación de cualquier función lógica y secuencial; además, cada bloque está bien caracterizado y pueden ser fácilmente interconectado con otros mediante herramientas de trazado automático. Así mismo, se han desarrollado lenguajes de descripción de *hardware* tanto a nivel físico como a nivel de comportamiento, este último reduce considerablemente el tiempo de desarrollo y permite la implementación de metodologías de diseño independientes de la tecnología. [5]

A continuación se presenta un resumen de las herramientas disponibles para el desarrollo de circuitos integrados digitales.

1.3.3. Paquete de diseño(*"design kit"*)

Es un conjunto de bibliotecas, documentos y circuitos prediseñados en una tecnología específica que pueden ser usados por los diseñadores, facilitando así la creación de sistemas electrónicos complejos y permitiendo la implementación de procedimientos estructurados de diseño. Los componentes más representativos de un kit de diseño se listan a continuación.

- **Celdas estándar:** Son un conjunto de circuitos modulares bien caracterizados, con los cuales se puede implementar una estrategia de diseño basada en lenguajes de descripción de hardware, permitiendo una reducción drástica en el tiempo de desarrollo de una aplicación digital. Se denominan de esta manera, ya que una de sus dimensiones es fija y

[2] Término en inglés que define una medida representativa del tiempo que se requiere desde la generación de la idea de un nuevo producto hasta que este está disponible en el mercado.
[3] Siglas en inglés: it Computer Aided Design, diseño asistido por computadora.

las líneas de alimentación se trazan de manera tal que la interconexión con otras celdas sea simple, lo que permite automatizar el proceso de posicionamiento y trazado [2].

La tecnología *CMOS* estática ha sido la más acogida por los diseñadores digitales, ya que las celdas diseñadas con esta arquitectura son robustas ante los procesos de fabricación y presentan alta inmunidad al ruido. Además, es posible realizar modelos simples con el fin de realizar simulaciones de sistemas complejos.

A fin de poder automatizar el proceso de diseño, las celdas estándar deben contener los elementos que se muestran en la figura 1.2. A continuación se presenta una breve descripción de cada elemento [2].

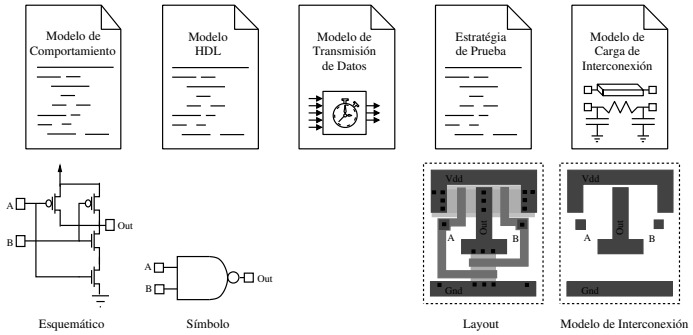


Figura 1.2: Elementos de una celda estándar.

- **Modelo de comportamiento:** Es un modelo simple de la celda, que permite determinar su comportamiento ante los estímulos aplicados en un corto periodo de tiempo, agilizando así la verificación funcional de sistemas complejos con múltiples bloques funcionales.

- **Modelo *VHDL*/Verilog:** Celdas que han sido optimizadas, como sumadores o multiplicadores, pueden ser implementadas directamente en el diseño. Para ello debe integrarse en el código fuente, un modelo compatible con el lenguaje de descripción de hardware usado.

- **Modelo de Transmisión de Datos:** El tiempo de respuesta de una celda limita la frecuencia a la que puede trabajar un diseño, por lo que es necesario una estimación de la misma. Este modelo permite verificar si un diseño puede trabajar a una velocidad específica, u observar sus efectos en el funcionamiento de un sistema complejo, como un μP.

- **Estrategia de prueba:** Para garantizar el funcionamiento de un circuito se debe verificar su respuesta ante todos los posibles estímulos, lo cual puede requerir gran

cantidad de procesamiento computacional. En el caso de bloques complejos pero repetitivos como las memorias, se debe aplicar una estrategia de prueba que permita determinar, sin recurrir a largos periodos de simulación, si la respuesta de estas celdas es la esperada. Esto se logra verificando sólo los estímulos críticos, ya que por la estructura de la celda, muchos de estos son redundantes.

- **Circuito esquemático:**Es una representación gráfica de los componentes e interconexiones de una celda. Esta descripción es necesaria a fin de garantizar la correspondencia entre el circuito diseñado y el *layout*, para lo que se requiere el uso de herramientas de extracción y comparación(LVS[4]).
- **Símbolo:** Es una representación simplificada del circuito esquemático de una celda.
- **Modelo de carga de interconexión:** Para detectar las posibles interconexiones críticas en el diseño y prevenir las fallas, se requiere un modelo que permita estimar la resistividad de las pistas y los componentes parásitos asociados. Ya que la impedancia de conexión, junto con el tiempo de respuesta de las celdas, afectan fuertemente la sincronía del sistema y la máxima velocidad de trabajo en diseños de alta frecuencia.
- *Layout:* Es una representación de la celda a nivel físico, en la que se define mediante dibujos, cada una de las capas necesarias para realizar la fabricación del circuito.
- **Modelo de trazado:** Es una descripción simplificada del *layout* que permite a las herramientas de trazado automáticas identificar los puntos de interconexión de la celda y las zonas críticas que debe evitar.

- **Celdas parametrizables:** Trazar las máscaras que forman un componente electrónico en tecnología *CMOS* es un proceso repetitivo que requiere una gran cantidad de tiempo de diseño. Buscando automatizar esta etapa se encuentran herramientas que permiten generar el *layout* de componentes a partir de ciertos parámetros, como por ejemplo, el largo y ancho en un transistor.
- **Reglas y modelos:** El fabricante de circuitos integrados proporciona una serie de reglas que limitan el trazado del *layout*, con esto se busca garantizar que el diseño sea fabricable en una determinada tecnología. Estas reglas son usadas por herramientas de verificación automática en un proceso conocido como DRC[5].

  Con el propósito de verificar el diseño antes de ser fabricado, se requiere una aproxi-

[4]Siglas en inglés: Layout Versus Schematic.
[5]Siglas en inglés: Design Rule Checking.

mación matemática del comportamiento físico, para cada uno de los elementos que se puede implementar en la tecnología usada. Con este objetivo, el fabricante proporciona modelos precisos que permitan al diseñador simular el comportamiento del sistema.

- **IPs:** Son circuitos de propósito específico, protegidos por la propiedad intelectual del diseñador. Al ser diseños testeados de buenas características, pueden ser implementados en sistemas en desarrollo permitiendo reducir el tiempo de diseño. Para ello, la empresa que los usa paga una cuota en función de la cantidad de veces que se implementa la IP, a la empresa o al diseñador que posee la propiedad intelectual.

1.3.4. Flujos de diseño

Por lo general el diseño de un circuito integrado no es una tarea fácil, esto, sumado a la gran cantidad de aplicaciones de software que se ha programado en este campo para cada aspecto del proceso, hacen que la organización del trabajo sea un objetivo primordial para reducir errores humanos. A fin de cumplir con este cometido, se ha desarrollado una serie de procedimientos guía que permiten al diseñador saber que pasos seguir para llevar el diseño a bien término y realimentarlo adecuadamente. A continuación se presenta los flujos de diseño más representativos:

- **Flujo de diseño ascendente(** *"bottom-up"***):** [6] Comienza con el análisis e identificación del problema, tras lo cual se establecen las especificaciones y requerimientos, partiendo de estos datos, se determina que pruebas globales son necesarias para que el diseño cumpla con las expectativas planteadas, seguidamente, se fragmenta el problema en bloques para distribuir el trabajo. Pasada la etapa de planeación, se procede a diseñar cada uno de los bloques que componen el sistema, realizando las pruebas necesarias mediante simulaciones. Una vez se halla probado cada entidad se procede a trazar el *layout* por separado y realizar las verificaciones respectivas, tras lo cual se interconecta los diversos bloques para conformar un sistema más complejo que satisfaga las especificaciones, realizando la verificación y análisis respectivos.

  Las ventajas de este flujo sobresalen en sistemas complejos con múltiples componentes, ya que en este caso se puede distribuir el trabajo en un grupo grande de diseño, permitiendo obtener resultados en corto tiempo. Aún así, es frecuente encontrar inconvenientes al empalmar los bloques, aunque se halla verificado su funcionamiento por separado esto no garantiza la funcionalidad del sistema final. Por otro lado, no se puede reutilizar el diseño si se cambia el proceso de fabricación, pero por lo general, se obtiene menor área que si se compara con los resultados de implementar un flujo de diseño

"top-down", ya que se enfoca en cada bloque tratando de optimizar su funcionamiento. Una visión general de este flujo de diseño se puede apreciar en la figura 1.3.

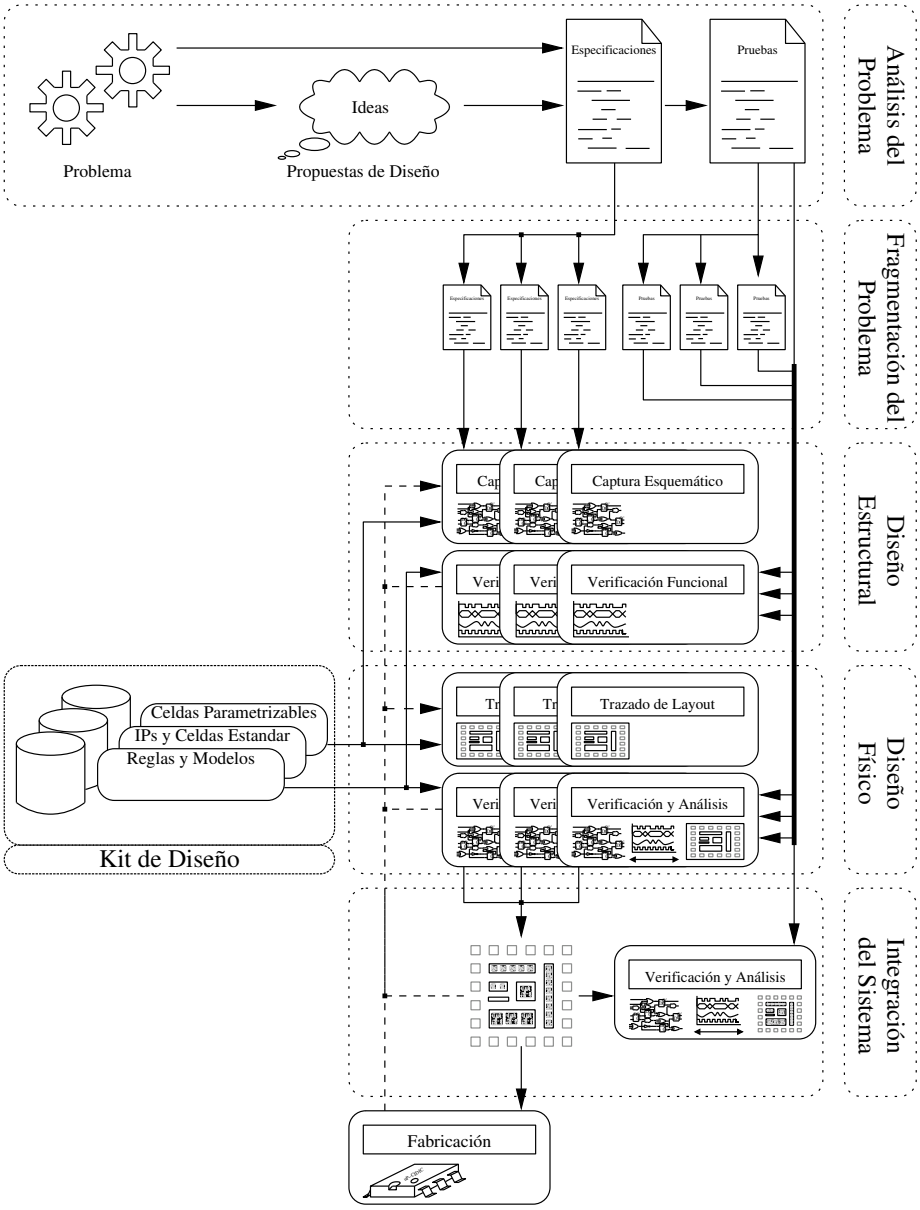


Figura 1.3: Flujo de diseño ascendente.

- **Flujo de diseño descendente(** *"top-down"***):** **[7]** **[2]** **[8]** Tras realizar el análisis e identificación del problema, y determinar los requerimientos y las especificaciones, se procede a realizar la descripción funcional del diseño en algún lenguaje de descripción de hardware como *VHDL* o Verilog; paralelamente a este trabajo, se diseña el plan de pruebas de todos los componentes del sistema, este conjunto de estímulos son usados a lo largo del flujo de diseño a fin de realizar en cada etapa del mismo, la respectiva verificación funcional [9]. Aún así no se puede asegurar el correcto funcionamiento del sistema ante todas las posibles entradas, para ello se recurre al análisis de cobertura [10] lo que permite verificar que secciones del código fuente no se ha probado, permitiendo realimentar el plan de pruebas a fin de abarcar todo el sistema. Una vez se verifica el correcto desempeño del código fuente, se realiza un análisis de potencia a fin de reducir el consumo de la misma.

  Una vez descrito funcionalmente, se procede a sintetizar el sistema [11] usando celdas estándar proporcionadas por un kit de diseño, durante esta etapa se selecciona automáticamente las combinaciones de celdas necesarias para implementar el código fuente en un diseño físico. En este punto es necesario verificar que el *netlist* a nivel de celdas arrojado por el proceso de síntesis cumpla con los requerimientos deseados. Primero se comprueba la equivalencia del código fuente con el circuito, proceso que se denomina verificación formal [12]. Para garantizar que los retardos de cada compuerta no afectan considerablemente el desempeño del sistema se debe realizar la verificación funcional y el análisis de potencia incluyendo estos efectos [13].

  Comprobada la viabilidad del diseño, se divide el sistema en bloques funcionales a fin de optimizar la frecuencia de operación del circuito, ya que acortando las interconexiones se mejora el desempeño y la sincronía del sistema. Posteriormente se realiza una planeación del área del diseño, posicionando cada uno de los bloques funcionales de la manera más conveniente, este proceso es conocido como planeamiento de "piso" o *Floorplan*. A continuación se procede a posicionar automáticamente cada celda; una vez en su sitio, se trazan las interconexiones entre ellas, comenzando por las lineas críticas, como las pistas de alimentación y la distribución del reloj [14], tras lo cual se completa los trazados restantes.

  Con el *layout* terminado, se verifica que este cumpla con las reglas de fabricación y se analiza la correspondencia entre el circuito y el *netlist* a nivel de celdas, esto se logra luego de realizar la extracción de componentes y usar una herramienta de *LVS*. Hecha la comprobación, se verifica el funcionamiento del diseño a fin de determinar si cumple con las especificaciones requeridas, si ese es el caso se procede a realizar la fabricación

del mismo.

Cabe destacar que si el diseño no cumple con las especificaciones en alguna etapa es necesario realizar cambios en los procedimientos previos a fin de cumplir con los requerimientos.

Aunque cuenta con una mayor numero de pasos que el flujo *"bottom-up"*, esta metodología requiere menor tiempo de diseño, ya que muchas de las tareas involucradas en el proceso han sido implementadas mediante herramientas *de software* y el diseño es reutilizable por completo ya que está basado en una descripción de funcional. A pesar de ello, el tamaño del chip resultante es mayor que en un diseño a medida ya que se pierde área al restringir las dimensiones de las celdas. Una visión general de este flujo de diseño se puede apreciar en la figura 1.4.

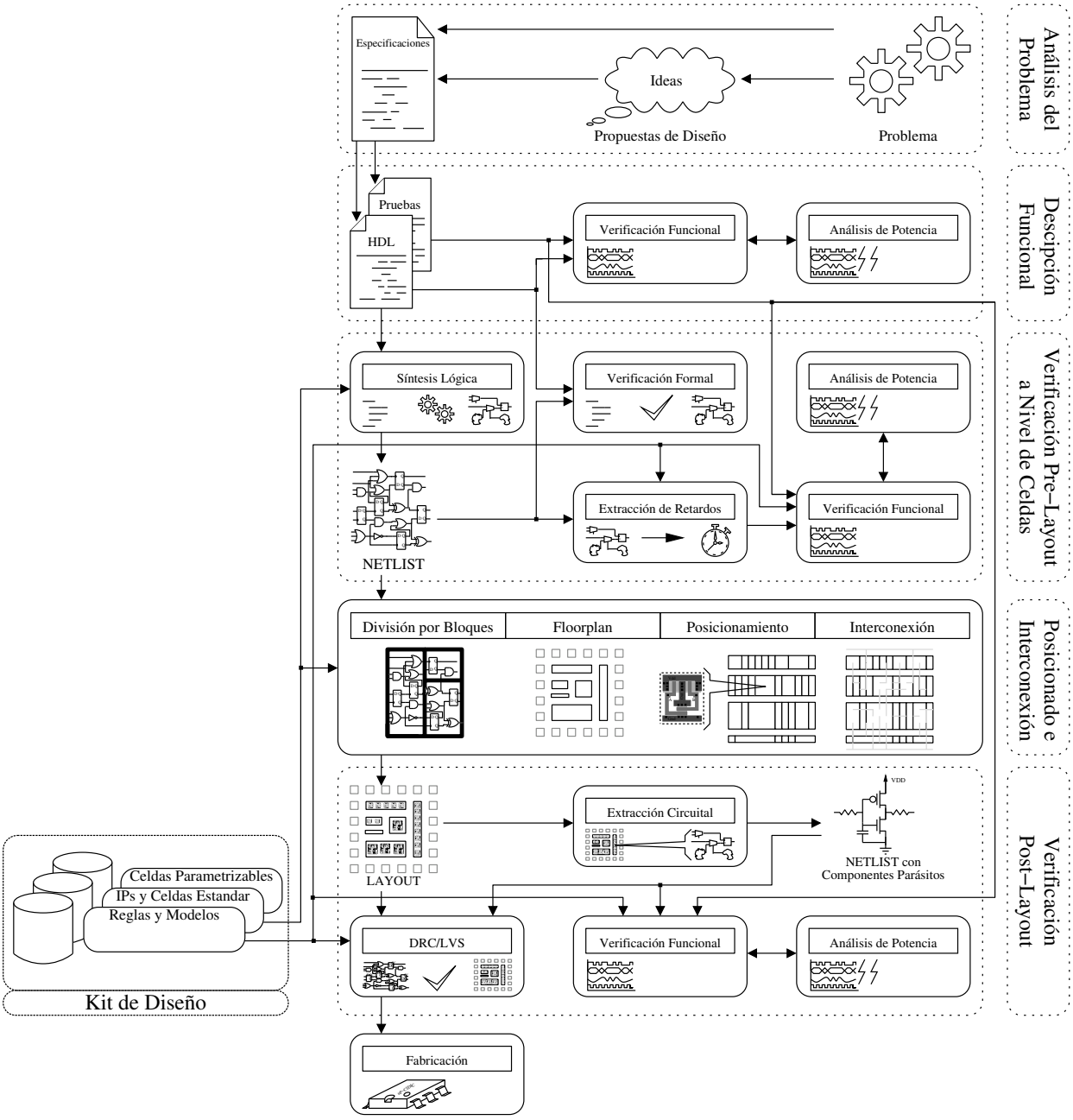


Figura 1.4: Flujo de diseño descendente.

Capítulo 2

Diseño del Microprocesador

El capítulo anterior exponía la importancia de los sistemas de procesamiento de datos como los microprocesadores y el impacto que han tenido las metodologías y herramientas de diseño automatizado en el desarrollo de los mismos. Durante el presente capítulo se define la estrategia y consideraciones utilizadas a lo largo del proyecto para el diseño de un microprocesador de propósito general.

2.1. Consideraciones de diseño

En la realización de un diseño, se debe tener en cuenta una serie de pautas con el fin de orientar el proceso y obtener buenos resultados. A continuación se describirá cada una de las consideraciones que se tomaron durante el desarrollo de este proyecto.

2.1.1. Consumo de energía

El consumo de energía de los componentes que lo integran, afecta la independencia de la red eléctrica de un sistema a batería y reduce el tiempo de vida de la misma. Por otro lado, debido a la naturaleza de los circuitos digitales, la corriente de alimentación no es constante, lo que produce interferencias en otros componentes a través de la señal de alimentación . Así mismo, una elevada disipación de potencia influye sobre la confiabilidad del sistema y eleva los costos de funcionamiento. Por estas razones, cada vez es más necesario darle importancia a este factor, tanto a nivel de diseño como de fabricación de circuitos integrados [15].

El consumo de potencia de un circuito se puede contemplar como la suma del consumo de cada uno de los elementos constitutivos, en este caso celdas lógicas. Dependiendo de su origen,

éste se puede separar como se indica a continuación.

$$P_{Celda} = P_{Dinámica} + P_{Corto\ Circuito} + P_{Estática} = ACV^2f + tAVI_{CC} + VI_E \qquad (2.1)$$

Dónde:

- A: Porcentaje de actividad de la celda con respecto a la señal de reloj.
- C: Capacitancia manejada por la salida de la compuerta.
- V: Voltaje de alimentación del dispositivo.
- f: Frecuencia de reloj.
- t: Periodo de corto circuito.
- I_CC: Corriente de corto circuito
- I_E: Corriente estática.

La potencia dinámica es usada por la celda para cargar y descargar la capacitancia formada por otras celdas a su salida y las respectivas interconexiones. La potencia de corto circuito es consumida internamente durante las transiciones, debido a la ruta directa que se produce entre los nodos de alimentación. Por último, la potencia estática es consumida en ausencia de transiciones. En la tecnología usada para el desarrollo de este proyecto, *ASM C35B4C3*, el último término se puede despreciar comparándolo con los otros, el segundo depende del diseño de la celda y el primero de sus interconexiones con otros elementos, siendo la de mayor contribución de las tres [16] [15].

Ya que el diseño del procesador se orienta al uso de celdas estándar prediseñadas, no se tiene control del consumo estático ni de corto circuito. Así que la estrategia de diseño propuesta, se basa en la reducción de la cantidad de elementos, las interconexiones entre los mismos y el análisis de las transiciones de señal. Así se reduce fuertemente tanto la potencia dinámica como las demás contribuciones [17]. A nivel de programación también puede reducirse el consumo, ya que el mismo es dependiente de los datos que circulan por el procesador [16].

A nivel de síntesis, se puede reducir el consumo de potencia mediante una técnica denominada *clock gatting*, o control de reloj. Ésta técnica permite usar la señal de habilitación de un registro para aislar el reloj, de manera que sólo circule señal en caso de habilitarse, lo que evita el desperdicio de energía durante la carga y descarga de la capacitancia asociada a este elemento. Por otro lado, gracias al flujo de diseño implementado, se pueden obtener las estadísticas de las señales con mayores transiciones y así seleccionar que bloques optimizar durante el proceso de síntesis lógica [18].

Reducir el consumo de potencia no reduce necesariamente el consumo de energía, ya que un procesador puede consumir menos potencia que otro, pero el primero puede tomar más tiempo en realizar las mismas operaciones. Una manera de considerar este efecto, es calcular la energía usada por operación, lo que se conoce como *EDP*[1] o producto retardo-energía. El *EDP* corresponde a la energía promedio consumida por ciclo de reloj multiplicado por el promedio de ciclos que usa el procesador al ejecutar instrucciones. De esta manera, reducir el tiempo de ejecución aumentará la eficiencia [16].

2.1.2. Área ocupada, congestión del trazado y retrasos de señal

El área ocupada por cualquier diseño afecta el costo final del dispositivo, por lo que reducirla es de gran importancia para posicionar un producto en el mercado. El uso de celdas estándar, aunque permite acortar el tiempo de diseño, eleva el tamaño del chip; por lo que este problema se debe atacar a nivel de sistema, reduciendo la cantidad de elementos constitutivos.

Al elevar el número de interconexiones de un circuito, la densidad de las mismas aumenta, lo que incrementa a su vez los efectos capacitivos entre ellas, introduciendo interferencias y retrasos adicionales en las señales. Por otro lado, las interconexiones contribuyen notablemente al área ocupada en el diseño de circuitos complejos, ya que debido a la alta resistencia de las mismas, es necesario reforzar las señales para no deteriorar el desempeño del circuito. Esto se logra mediante el uso de inversores y *buffers*, que ocupan área adicional. Un ejemplo de esto, es la sincronización de la señal de reloj en un circuito secuencial, como un registro de corrimiento, en el que si no se activan los biestables aproximadamente al mismo tiempo, no se transmitirá el dato entre ellos. Por este motivo, la reducción de las interconexiones debe realizarse siempre que sea posible [17].

2.1.3. Reutilización del diseño

El diseño de un dispositivo electrónico es un proceso costoso, por lo que la reutilización del mismo se vuelve imperante en empresas especializadas en el campo. Es aquí donde el diseño *top-down* es de gran utilidad, ya que gracias a la independencia tecnológica de los lenguajes descripción de hardware, un diseño puede ser implementado en una tecnología diferente, ajustando pasos posteriores a la concepción del mismo. Sin embargo, un diseño que no permita cierto grado de libertad en sus parámetros, dista mucho de ser reutilizable en comparación con uno configurable. Por eso se debe enfocar cualquier diseño a definir los parámetros que

[1] Siglas en inglés: Energy-Delay Product.

caractericen el mismo. En el caso particular de este proyecto se toman como parámetros el bus de datos, de dirección y la cantidad de elementos de memoria. [19]

2.1.4. Instrucciones

Como se menciona previamente, las instrucciones afectan la eficiencia de un procesador y el desempeño del mismo. Ahora bien, un set reducido permite simplificar el circuito, pero una reducción excesiva puede complicar la implementación de algoritmos. Por ello han de conservarse en el set las operaciones lógicas básicas y las aritméticas generales, así como las instrucciones de carga, almacenamiento y desplazamiento, con el fin de permitir la implementación de operaciones más complejas. Reducir los modos de direccionamiento tiene un amplio impacto en las interconexiones, por lo que se busca reducir su cantidad para mejorar las características de desempeño del procesador.

2.2. Diseño Base

Se usó el procesador diseñado por Sandra Ovallos como base para el desarrollo de este proyecto [20]. Éste es un procesador de propósito general *RISC* de arquitectura *harvard*, al que se aplicó la técnica de procesamiento en *pipeline* y fué sintetizado en una FPGA[2]. La estructura interna del *datapath* de este dispositivo se encuentra detallada en la figura 2.1.

Debido a la metodología de modelado usada por Ovallos para el diseño de su procesador, éste no puede ser sintetizado correctamente mediante el uso de celdas lógicas, ya que no se encuentra descrito formalmente en lógica de transferencia de registros. Además, la estructura de algunos de sus componentes no es reconocida por el sintetizador usado en el flujo de diseño implementado [21]. Por ello no se utiliza la descripción funcional en *VHDL* del microprocesador, pero se aprovecha la arquitectura del mismo a fin de mejorarla y permitir la síntesis lógica orientada a la tecnología *CMOS*. Por otro lado, implementar sistemas digitales en una *FPGA*, aunque permite probar un diseño de manera rápida y barata, es superada en cuanto a características de desempeño por el diseño en un *ASIC*[3] [22].

[2]Siglas del inglés: Field Programmable Gate Array, dispositivo lógico programable que permite realizar el prototipo de un sistema digital en corto tiempo.

[3]Siglas del inglés: *Application Specific integrated circuit*

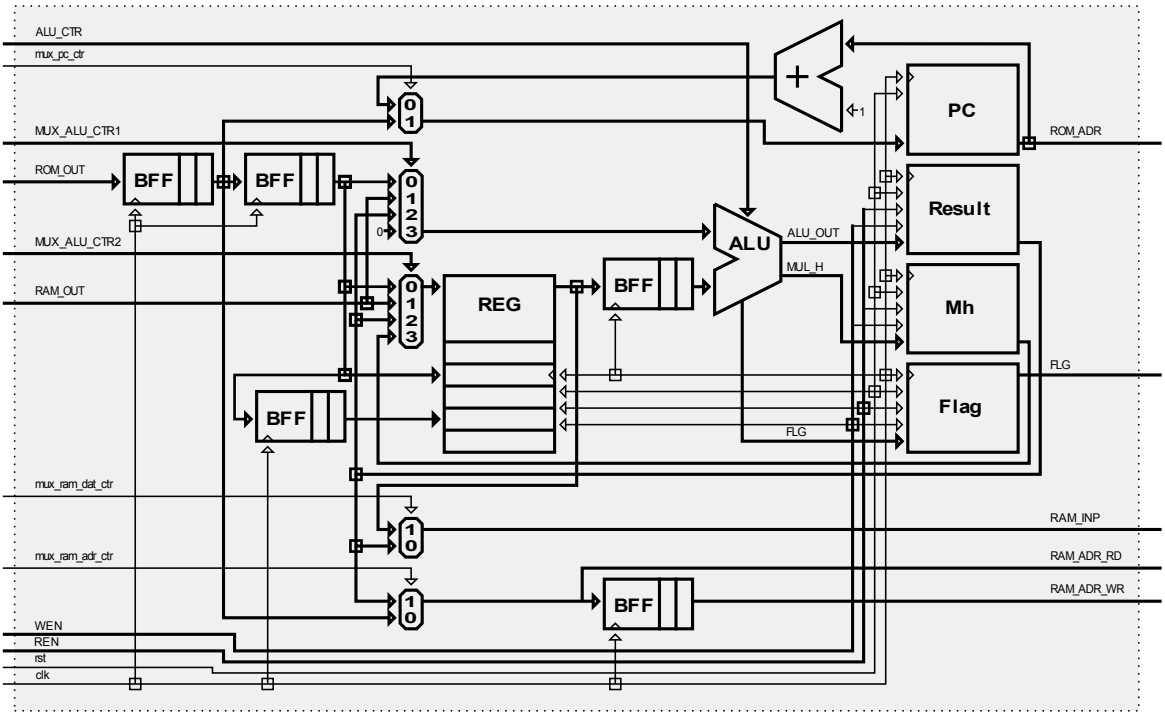


Figura 2.1: Microprocesador usado como base para este proyecto.

2.3. Diseño del procesador

Teniendo en cuenta las consideraciones de diseño previamente señaladas, se procede a modificar el diseño base a nivel estructural y de transferencia de registros.

2.3.1. Formato de instrucción

El formato de instrucción de un procesador afecta varios parámetros de desempeño. Uno de éstos es el consumo de área de la memoria de programa o *ROM*, ya que cada bit adicional es multiplicado por la cantidad de palabras de programa en el arreglo. Así mismo, reducir el tamaño de la memoria *ROM* implica una reducción directa del consumo del bloque. El formato de instrucción utilizado en este proyecto es una versión modificada del usado por Ovallos. Usando parte del espacio reservado para el direccionamiento de registros y el campo de datos, para codificar instrucciones que no los utilizan, o que se deben mantener a un valor fijo para la apropiada ejecución de la instrucción. De esta manera se obtuvo una reducción de un bit de la palabra de programa, su distribución se puede observar en la figura 2.2. Nótese que debido a la definición de parámetros en el diseño del procesador, los campos de direccionamiento, de registros y el de datos son de tamaño variable. Para permitir la variación

de éstos, se usó una librería de usuario con las constantes que definen el tamaño de los buses de interconexión. De esta manera, con una simple inclusión en la cabecera de la descripción de comportamiento en *VHDL* de cada elemento se puede actualizar todo el diseño, al realizar cualquier cambio de parámetro.

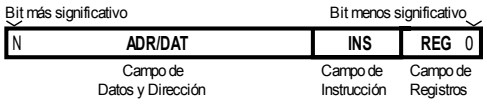


Figura 2.2: Formato de instrucción.

2.3.2. Estructura interna

En la figura 2.3, se puede observar cada uno de los bloques constitutivos del procesador diseñado. A continuación se presenta una descripción detallada de los mismos.

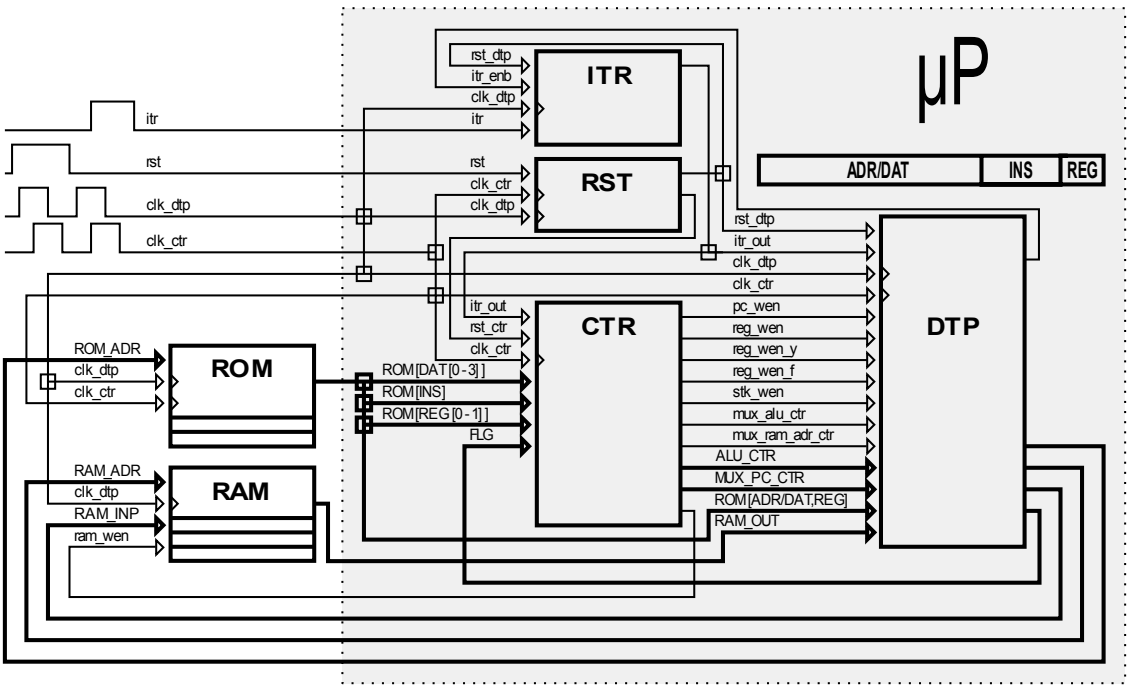


Figura 2.3: Diagrama de bloques del microprocesador diseñado.

2.3.3. *Datapath*

El *datapath* es la estructura por la que se desplazan los datos usados por el procesador, al ejecutar cada una de las instrucciones. Es el bloque que más afecta el desempeño y el que

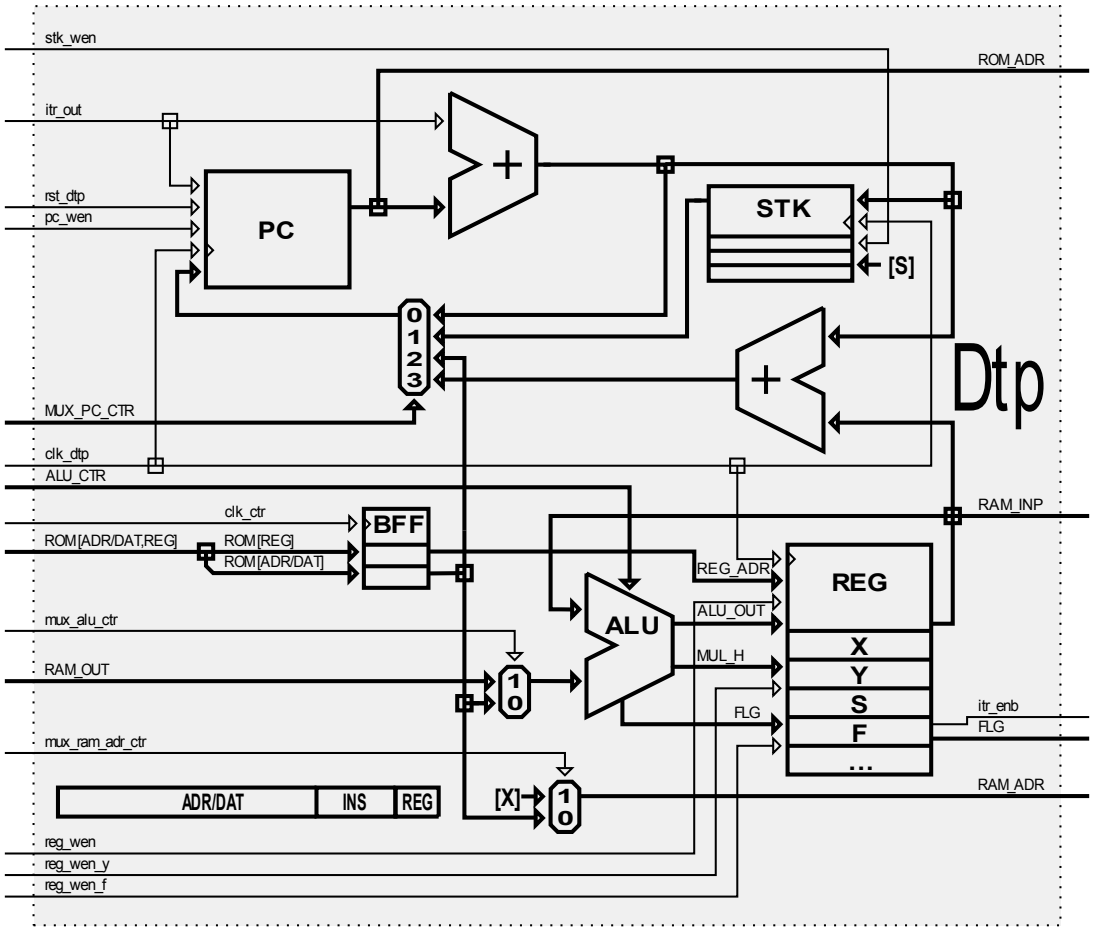


Figura 2.4: Diagrama interno del *datapath*.

más consume energía. El diseño de Ovallos utiliza la técnica de procesamiento en *pipeline*, lo que permite ejecutar una instrucción por ciclo de reloj introduciendo registros en posiciones estratégicas. No obstante, estos elementos funcionan durante cada transición y debido a su construcción interna consumen gran cantidad de potencia. Por otro lado, los multiplexores complejos elevan la cantidad de interconexiones internas del diseño, y el uso de registros de propósito general requiere la implementación de instrucciones especializadas de carga y almacenamiento.

Los anteriores inconvenientes se pueden resolver de manera conjunta, modificando la arquitectura del procesador como se propone en la figura 2.4. Para ello se debe desplazar el banco de registros desde la entrada a la salida de la *ALU*. Lo anterior permite eliminar dos multiplexores y una gran cantidad de interconexiones, al configurar el banco de registros como acumulador. Además, reduciendo la complejidad del multiplexor de entrada de la *ALU*, se puede recortar la latencia total entre el *buffer* de memoria *ROM* y los registros a la siguiente

expresión.

$$T_{Latencia} = T_{bff_{out}} + T_{mux\_alu} + T_{ALU} + T_{reg_{in}} \tag{2.2}$$

La anterior es la ruta más larga del procesador, un poco más corta que en el diseño base, proporcionando un mejor desempeño ya que permite la ejecución de instrucciones en un ciclo de reloj, sin adicionar elementos de memoria. A continuación se explicará detalladamente este proceso mediante la descripción de cada elemento del *datapath*.

Buffer de datos, *BFF*

El *buffer* de datos es un registro que almacena la información proveniente de la memoria *ROM*. Se construye con la agrupación de *flip-flops* y almacena tanto el campo de direccionamiento de registros como el área de datos de la palabra de programa.

Unidad Aritmético-Lógica, *ALU*

La ALU es el bloque que realiza las operaciones sobre los datos de entrada y el banco de registros. Externamente se comunica mediante una señal de configuración, dos entradas y tres salidas. La señal de configuración proviene de la unidad de control. La salida principal o ALU_OUT se conecta a la entrada del banco de registros, de manera que el resultado puede almacenarse en cualquiera de ellos. Mientras que las salidas restantes se interconectan con los registros de propósito especial Y y F, arrojando los datos de la palabra alta de la multiplicación y las banderas de resultado.

Considerando que se quiere definir el diseño en función de los parámetros del procesador y permitir la reutilización del código del mismo, no se puede elegir una estructura fija de éste elemento. Por ello se decide utilizar un modelo de decodificación de operación, modelando la ALU como un conjunto de operaciones realizadas sobre las entradas, y cuya salida se selecciona por la señal de configuración. Lo anterior permite al sintetizador elegir el mejor módulo de hardware a usar durante la síntesis lógica, detectando los elementos críticos del mismo y reduciendo así el consumo de potencia y el área ocupada por el bloque, en función del ancho del bus de datos.

Banco de registros, *REG*

En este bloque se decide integrar los registros de propósito específico y general, lo que permite que puedan ser accedidos directamente por la ALU, eliminando así las operaciones especiales

de transferencia de datos. Con el fin de reducir la complejidad del trazado y del bloque en general, no se usa ningún tipo de inicialización.

A continuación, se listan los registros de propósito especial y su función.

- **X:** Se usa para realizar operaciones de direccionamiento de la memoria *RAM*.
- **Y:** Guarda la palabra de mayor peso de las operaciones de multiplicación.
- **S:** Es usado para direccionar el nivel actual de la pila.
- **F:** Es el registro de las banderas de resultado y la habilitación de interrupción, desde el primer bit hasta el último se tiene que Z corresponde a cero, C al acarreo e IE a la habilitación de interrupción.

En la figura 2.5, se muestra un esquema del banco de registros. Nótese que algunos registros de propósito específico pueden ser modificados al mismo tiempo.

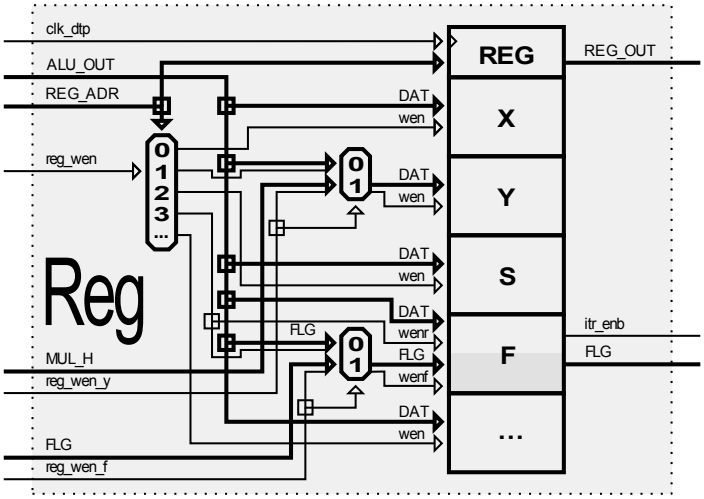


Figura 2.5: Diagrama interno del banco de registros.

Contador de programa: *PC*.

El contador de programa es un elemento vital para la ejecución secuencial de un programa, ya que señala la dirección de la memoria *ROM* de la instrucción a interpretar. Durante este proceso, el contador debe cambiar el dato que almacena de acuerdo con la instrucción que se ejecute, a fin de seguir el flujo del programa. Las diferentes maneras en que se puede reasignar el dato de *PC* se denominan modos de direccionamiento y dependen del hardware asociado al mismo. Los modos de direccionamiento que se implementaron bajo la arquitectura propuesta son los siguientes:

- **Direccionamiento automático:** Considerando que un procesador debe arrancar la ejecución de un programa en una dirección específica, en caso de un evento de *reset* o similar, se debe cargar un dato en PC de manera automática. Para lograrlo se utiliza la inicialización de los registros. En este caso en particular se requieren dos inicializaciones, una para el *reset* y otro para la interrupción. De esta manera en caso de producirse el *reset*, el evento de mayor prioridad, el contador de programa se cargará con el valor $\cdots 00000$. En caso de interrupción, éste se carga con la dirección $\cdots 00100$, dónde debe comenzar la rutina de manejo de interrupción.

- **Direccionamiento directo:** En este caso se carga PC con el valor almacenado en el *buffer* de memoria ROM, mediante el multiplexor del contador de programa.

- **Direccionamiento implícito:** Se utiliza en la mayoría de las instrucciones, ya que durante la ejecución de éstas se debe incrementar PC, por lo que no se requiere de ningún dato de dirección. Un caso similar ocurre con las instrucciones de retorno de subrutina, ya que cargan un valor de la pila direccionado por el registro S. Para realizar el incremento del contador de programa se usa un sumador que adiciona a su valor actual el bit de evento de interrupción, activo por bajo; de ésta manera, a menos que se presente una interrupción, a la salida se entregará $PC + 1$.

- **Direccionamiento relativo:** Usa el valor almacenado en un registro para realizar un salto relativo al cargar PC con $PC + 1 + REG$. Este modo de direccionamiento es de gran utilidad durante la lectura de tablas de datos.

El multiplexor del contador de programa determina el modo de direccionamiento, éste módulo es comandado por la unidad de control que decide cual usar en cada instrucción.

Pila, *STK*

La pila es un bloque que permite realizar un salto en la ejecución del programa y ejecutar código en otra sección de la memoria, proceso que se denomina "llamado a subrutina". Una vez realizadas estas operaciones, posibilita el retorno hacia la instrucción siguiente al salto, lo que se denomina retorno de subrutina. Considerando que se puede anidar los llamados a subrutina, es necesario la implementación de una estructura que permita almacenar, no solo la dirección de retorno, sino también el orden de llamado. Es la forma en que se almacenan los datos en una pila lo que le otorga el nombre, durante el llamado cada dato nuevo se almacena en la posición dada por un registro apuntador y éste se incrementa, lo que permite que el siguiente se apile "encima" del dato anterior. En el retorno se decrementa el apuntador y se

arroja el dato a su salida, con lo que se puede cargar PC y retornar sin perder el control de la ejecución del programa.

En este proyecto se implementa la pila mediante un arreglo de registros cuyo apuntador es el de propósito específico S. La entrada se toma del incrementador del contador de programa y su salida se direcciona hacia éste mediante su multiplexor.

Una llamada a interrupción es similar a la llamada a subrutina, en el sentido que el procesador debe realizar un salto y retornar dónde se realizó, la diferencia radica en el momento en que se ejecuta. En el primer caso se debe evitar la ejecución de la instrucción actual y reemplazarla por el llamado a subrutina, en el segundo es la instrucción la que se encarga de realizar el proceso. Lo anterior implica que se debe almacenar la dirección actual en el primero y la siguiente en el segundo; así se evita que no se ejecuten instrucciones o se entre en bucle infinito respectivamente. Para resolver este problema sin el uso de multiplexores, se aprovecha el incrementador de PC que suma el estado de interrupción, activo por bajo, obteniéndose así el comportamiento deseado a la entrada de la pila.

Ahora bien, debido a que en el presente diseño cada instrucción toma un ciclo de reloj para ejecutarse, no es posible realizar el proceso de retorno mediante una sola instrucción. Esto se debe a que el dato se guarda en la pila en la posición dada por S y al mismo tiempo se puede realizar el incremento de este registro a través de la ALU. Durante el retorno se requiere decrementar primero el apuntador y luego retornar para no perder el control del programa. Para evitar este inconveniente se debe decrementar mediante una instrucción S y luego realizar el retorno de subrutina.

Otro problema se presenta durante una interrupción, ya que la unidad de control no tiene dominio sobre el direccionamiento de registros y no puede incrementar el apuntador de la pila automáticamente. Para evitar la pérdida del control del programa en este caso, se debe incrementar el registro S si se realizan llamadas a subrutina durante el código de interrupción y decrementarlo para realizar el retorno de la misma. En caso de no utilizar los llamados, puede evitarse el uso de estas instrucciones ya que el resultado será el mismo.

2.3.4. Unidad de control, *CTR*

La unidad de control, como su nombre lo indica, se encarga de dirigir los datos por el *datapath*. También activa los registros según la instrucción ejecutada y gestiona parte del control interno del procesador durante los eventos de *reset* e interrupción. Tradicionalmente, el diseño de este módulo se realiza mediante una máquina de estados finitos que usa varios ciclos de reloj para

ejecutar una serie de pasos sencillos o microinstrucciones. En el caso más simple se requieren dos estados, uno para la carga y decodificación de la instrucción y otro para la ejecución de la misma. Ya que el periodo de lectura suele tomar menos tiempo que el periodo de ejecución, debido a la velocidad de respuesta de las memorias *ROM* integradas, se sacrifica la máxima frecuencia de operación mediante esta técnica. Una forma de elevar este parámetro de desempeño es el *pipeline*, lo que permite que el procesador ejecute una instrucción mientras lee la siguiente. Como se mencionó previamente, aplicar esta técnica tiene grandes implicaciones negativas en cuanto al consumo de potencia y el área ocupada por el diseño [20] [16].

Gracias a la estructura del *datapath* del diseño propuesto en este trabajo, la ruta más larga es ligeramente mayor que la de la *ALU*; lo que permite adicionar el retraso adicional de la unidad de control sin impactar fuertemente la máxima velocidad de operación del procesador. Esto se logra mediante el uso de dos señales de reloj, una para *datapath* y la otra para el bloque de control, la segunda debe estar desfasada con respecto a la primera, con el fin de crear una ventana de tiempo para la lectura y decodificación de instrucciones. El tamaño de esta ventana y por lo tanto, el periodo de retraso de las señales de reloj, será la suma de la latencia de la memoria *ROM* y la unidad de control. Una manera de producir estas señales es mediante el uso de inversores en cascada hasta obtener el retraso deseado, ya que cada inversor presenta un tiempo de respuesta constante [17].

Ya que la unidad de control del presente proyecto es de un sólo estado y se requiere aumentar su velocidad de respuesta, se propone el modelado de la misma mediante un bloque decodificador y un registro de almacenamiento. En la figura 2.6, se muestra la estructura interna de la unidad de control propuesta. El registro de salida almacenará las señales de control decodificadas durante el flanco se subida del reloj de control, permitiendo desconectar la memoria de programa para ahorrar energía como se detallará más adelante en la descripción de este bloque.

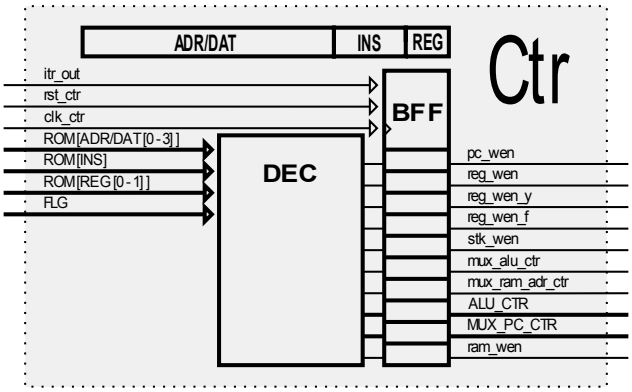


Figura 2.6: Diagrama interno de la unidad de control.

Al cargar las señales de control en el registro, se configura el *datapath* para ejecutar la instrucción durante el periodo fuera de la ventana de lectura. El *buffer* de datos *ROM* se activa al mismo tiempo permitiendo a la *ALU* operar sobre los datos de la palabra de instrucción. Este retraso también previene que se presente metaestabilidad en los registros, al asegurar que los datos se encuentran constantes durante el proceso de escritura, durante el flanco de subida de la señal de reloj del *datapath* [17].

A diferencia del *datapath*, la unidad de control no depende de los parámetros de diseño, por lo que su tiempo de latencia es constante. Por esta razón puede usarse la misma unidad de control para un procesador con un bus de datos de 8 o 32 *bits*. De hecho, entre mayor es la complejidad de *datapath*, mejor será el efecto de aplicar la técnica propuesta.

Las interrupciones son parcialmente manejadas por la unidad de control, ya que la inicialización del registro *PC* es automática. Para la gestión de interrupciones se utiliza la inicialización del registro de salida, que también es usada en caso de *reset*. Al presentarse cualquiera de los dos eventos, se ignoran los datos del decodificador y se cargan las señales de control con la configuración respectiva, lo que se puede considerar como la implementación de dos instrucciones no controladas por el usuario.

2.3.5. Unidad de sincronización de *reset*, *ITR*

El *reset* es una señal de naturaleza asincrónica, ya que puede producirse en cualquier instante de tiempo. Implementar un *reset* asincrónico afecta la velocidad de respuesta de un *flip-flop*, además puede causar metaestabilidad si se produce en las transferencias de datos entre registros. Es por ello que se debe evitar el uso de esta técnica durante el modelado de un diseño digital para así evitar conflictos tanto a nivel de síntesis como de funcionamiento. La metaestabilidad es un problema propio de los sistemas digitales, y debido a la naturaleza aleatoria de sus efectos, no hay una técnica que asegure la eliminación total del mismo. Entre las alternativas para atacarla, se encuentra la mejora de los procesos de fabricación, a fin de reducir el periodo en el que la señal de entrada debe permanecer constante antes de las transiciones del reloj. A nivel de diseño la técnica a manejar es el uso de registros en serie, de esta manera, el registro anterior aisla al siguiente del efecto mientras el primero lo sufre [17] [23].

Para el manejo de la señal asincrónica del *reset* durante el diseño de este procesador, se decide implementar el uso de *flip-flips* aisladores. Estos registros cargarán la entrada de *reset* durante los flancos de bajada de la señal de reloj. A este bloque se le llama unidad de sincronización de *reset*, ya que no sólo separa la señal de entrada, sino que también inicializa los bloques

del diseño en el orden adecuado: primero el *datapath* y luego la unidad de control. Además, se evita el modelado de cualquier *reset* asincrónico con el fin de evitar problemas durante la síntesis lógica y no deteriorar el desempeño de los registros.

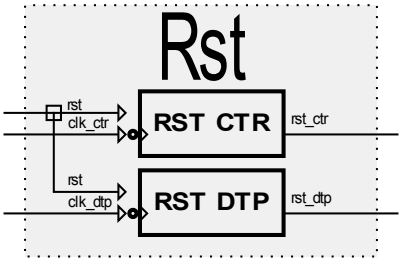


Figura 2.7: Diagrama interno del circuito de sincronización de *reset*.

2.3.6. Unidad de control de interrupción

Al igual que el *reset*, la señal de interrupción puede producirse en cualquier momento, por lo que se utiliza la misma técnica para sincronizarla. No obstante, la interrupción no sólo ha de sincronizarse con el reloj del sistema, sino que también debe controlarse el momento en que se presenta en el interior del procesador. Esto es de consideración, ya que si no existe un mecanismo de desactivación automático, el procesador entrará en un ciclo de saltos hacia la rutina de interrupción que sólo cesará cuando la señal de entrada cambie de valor.

Para solucionar este problema se adicionó un bit de control en el registro F a fin de habilitar o deshabilitar las interrupciones. Aún así, este control no es suficiente para asegurar la adecuada respuesta del procesador durante una interrupción. Además, ya que el banco de registros no se inicializa en el *reset*, se debe retrasar la señal durante el arranque del procesador, para evitar perder el control del mismo. Esto se logra mediante un registro de desplazamiento, que permite que el procesador ejecute unas cuantas instrucciones, antes que un decodificador determine, junto con el bit de habilitación, si se transmite o no la señal de interrupción a otros bloques del procesador. Este intervalo de tiempo, le permite al programador habilitar o deshabilitar las interrupciones según sus necesidades. Además al producirse una interrupción, concederá unos ciclos de instrucción para que el programador guarde el registro de banderas, borre la habilitación para atender el evento y se pueda ejecutar el proceso inverso durante el retorno. En la figura 2.8, se puede observar el diagrama interno de la unidad de control de interrupción propuesta.

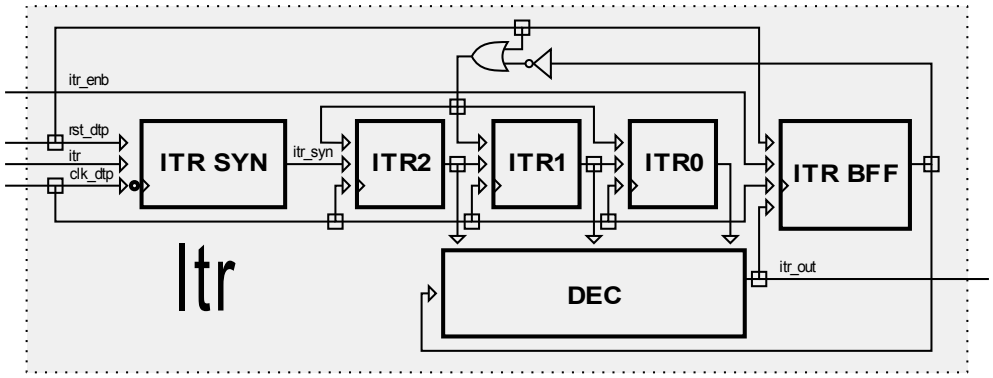


Figura 2.8: Diagrama interno de la unidad de control de interrupción.

2.3.7. Memoria de datos: *RAM*

La memoria de datos es el bloque encargado del almacenar la información procesada. Debe ser un sistema de lectura-escritura, y en el caso particular de este proyecto, ha de tener separado la entrada y salida de datos. Ya que el diseño de este bloque se encuentra fuera de los alcances de este proyecto, se requiere realizar un modelo de comportamiento que permita la verificación funcional del microprocesador. Para ello se modela la memoria como un arreglo de registros direccionados mediante una señal de entrada, que almacenan durante el flanco de subida de la señal de reloj, siempre que se habilite la señal de escritura.

2.3.8. Memoria de programa: *ROM*

La memoria de datos es el componente en donde se almacena previamente el programa a ejecutar por el procesador. A nivel de circuito integrado la técnica más utilizada para el almacenamiento de datos no volátiles es el uso de transistores de compuerta flotante. Estos transistores se fabrican apilando dos capas de polisilicio y por lo general son de tipo N, la capa inferior no se conecta a ningún contacto mientras que la segunda se usa como puerta del transistor. En condiciones normales una tensión en la puerta de control produce un movimiento de carga en la flotante transmitiendo su efecto y formando canal, ya que este dispositivo se comporta como un capacitor en serie con la puerta de un transistor. Para grabar un dato se inyecta electrones en la compuerta flotante mediante el efecto túnel, este proceso no permite la formación del canal debido a la repulsión de cargas entre la compuerta flotante y el sustrato. Mediante un arreglo de estos transistores y sendos bloques decodificadores de palabra, se puede leer el estado de cada transistor mediante un circuito inversor formado con transistores P que los polaricen. Una vez el dato se arroja por las filas se refuerzan mediante el uso de circuitos inversores [24].

La estructura interna de la memoria *ROM* considerada para su uso por el procesador se muestra en la figura 2.9. Para reducir el consumo de potencia de este circuito, se utiliza un decodificador para la entrada de habilitación. De esta forma sólo se realizarán lecturas durante el periodo de tiempo que se requiere para decodificar la instrucción. Pasado este periodo, el bloque decodificador no activa el arreglo de transistores, lo que reduce el consumo de potencia a la consumida por el circuito que polariza de los transistores *P*, gracias al almacenamiento de estos datos en los registros del *datapath* y la unidad de control.

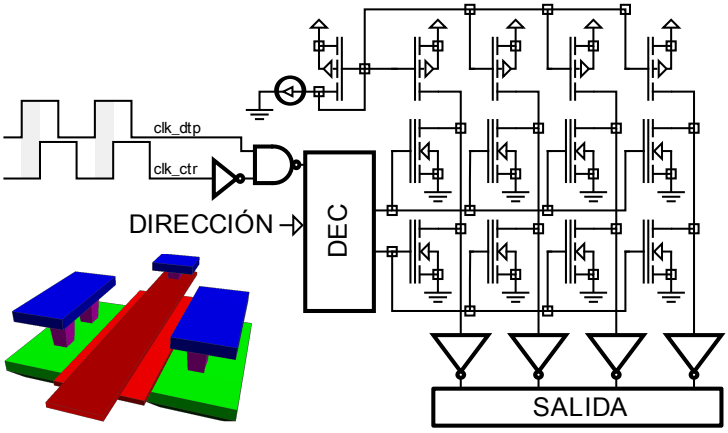


Figura 2.9: Diagrama interno de la memoria *ROM*.

Debido a que las memorias *ROM* se encuentran descargadas una vez fabricadas, se asignó como $\cdots 11111$ al código de operación de la instrucción no opere, a fin de evitar que el dispositivo ejecute alguna instrucción antes de ser programado.

Ya que el diseño de este bloque está fuera del alcance de este proyecto, pero se requiere de un modelo con el fin de realizar la verificación funcional del procesador. Se modela la memoria de programa como un arreglo de registros direccionados mediante el contador de propgrama; estos registros se encuentran previamente grabados con el programa a ejecutar.

2.4. Set de instrucciones

En las tablas 2.1 y 2.2 se condensan el conjunto de instrucciones que puede ejecutar el microprocesador. A continuación se explicará cada uno de los tipos de instrucción implementados.

- **Transferencia de datos** Estas instrucciones se encargan de mover los datos entre los bloques de almacenamiento sin ejecutar ninguna operación sobre los mismos. Pueden dividirse en dos grupos: carga y almacenamiento. El primero mueve los datos desde las

memorias *RAM* o *ROM* hacia los registros, mientras que el segundo mueve los datos desde los registros hacia la memoria *RAM*.

- **Tratamiento de datos** Como su nombre indica, realizan operaciones con los datos almacenados en los registros y con datos en la *RAM* o la *ROM*. Se pueden dividir en dos grupos dependiendo del tipo de operación que realizan: aritméticas y lógicas. Las primeras son la suma, resta, multiplicación y sus derivadas; la segundas corresponde a las operaciones booleanas.

- **Control:** Permite la realización de saltos en el programa, lo que altera la ejecución normalmente consecutiva de operandos. En este grupo se encuentran las instrucciones de salto y las de manejo de subrutinas.

- **Automáticas** Son ejecutadas bajo ciertas condiciones especiales, por ejemplo durante el *reset* y las interrupciones. Durante estos casos se reemplaza la instrucción actual por una que permita atender el evento.

2.4.1. Verificación funcional

Observar el comportamiento del procesador, mientras ejecuta cada una de las instrucciones almacenadas en su memoria de programa, es necesario para corroborar si se cumple con las especificaciones. Probar las instrucciones de manera aislada es un proceso repetitivo, por lo cual es necesario automatizarlo, a fin de acelerar el procedimiento de prueba y reducir la probabilidad de cometer fallos humanos. Para ello se aplican y desarrollan las herramientas que se exponen a continuación.

Banco de pruebas

Gracias a la implementación del flujo de diseño *top-down* y la automatización realizada durante este proyecto, es posible aplicar las señales de entrada del diseño sin intervención directa por parte del diseñador. Para introducir las señales de cada bloque externo se utiliza los modelos de memorias previamente expuestos. De esta manera, las únicas que se requiere introducir son las correspondientes al *reset* y las interrupciones. Éstas señales se registran en el archivo de estímulos, lo que permite simular el sistema bajo diferentes condiciones externas. Del mismo modo, usando el *TestBench* se genera los relojes del sistema y se verifica la salida del mismo luego de la síntesis lógica, el posicionamiento y trazado.

Ensamblador

Introducir un programa en la memoria *ROM*, implica ingresar bit a bit cada una de las instrucciones. Éste proceso mecánico puede producir fallas difícilmente identificables, por lo que se automatizó el mismo a fin de realizar el programa en un nivel de abstracción más elevado. Esto se logra mediante la programación en ensamblador, lo que permite reemplazar los datos binarios de cada operación por mnemotécnicos y sus parámetros. Este método permite recordar con facilidad los comandos y agiliza la escritura de un programa. Para ello se programa la herramienta ensambladora que se muestra en la figura 2.10.

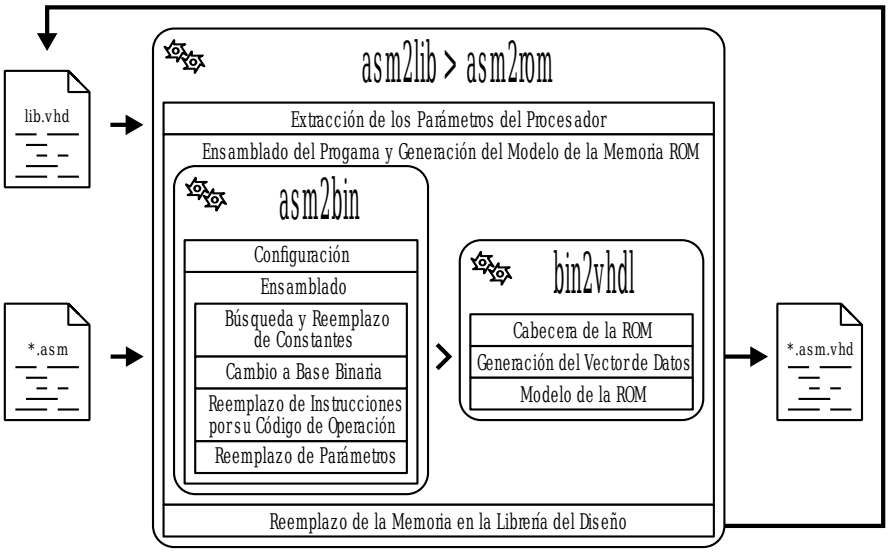


Figura 2.10: Flujo detallado del ensamblador programado.

Éste programa toma un archivo en formato ensamblador, se configura con los parámetros del procesador guardados en la librería global, realiza el proceso de ensamblado y anexa estos datos en un modelo *VHDL* que se incorpora de vuelta a la librería. Para facilitar la escritura de código, durante el proceso de ensamblado se buscan en todo el documento las constantes usadas, como la dirección de variables o la dada por las etiquetas, y se reemplazan por su correspondiente valor. Luego se cambia a base binaria todos los datos numéricos, tras lo que se procede a reemplazar cada mnemotécnico y sus parámetros por su correspondiente valor.

Programa de prueba

Gracias al uso del ensamblador, se realizó un programa que permite verificar mediante simulaciones la ejecución de cada instrucción. Éste programa incluye todas las instrucciones que puede ejecutar el procesador. Además, se agregó el código correspondiente a la rutina

de interrupción, con lo que junto con el banco de pruebas, permite analizar la respuesta del procesador frente a este evento, y la inicialización. Para verificar que el diseño sintetizado funciona de la misma manera que el modelo *RTL*, una vez ejecutada cada la instrucción, se mueve el dato resultante a la memoria *RAM*, lo que permite identificar discrepancias entre los datos arrojados. Considerando que el diseño es configurable, se tuvo en cuenta la variación de los parámetros al definir el programa de prueba, para ello se definen constantes de 32 bits, tamaño limitado por el ensamblador, lo que permite realizar análisis de funcionamiento al variar los mismos.

Tipo	Mnemotécnico		Código de Operación		Banderas	Descripción	
	Nombre	Parámetros	ADR/DAT	INS REG	Afectadas	Aritmética	Textual
Aritmético	DECR	Reg	----------0000 0000 rrr		NCZ	Reg=Reg-1 PC=PC+1	Decrementar en 1 el valor de un registro y almacenar resultado en el mismo, y saltar a la siguiente instrucción.
	INCR	Reg	----------0001 0000 rrr		NCZ	Reg=Reg+1 PC=PC+1	Incrementar en 1 el valor de un registro y almacenar resultado en el mismo, y saltar a la siguiente instrucción.
	ADDRI	Reg,Inm	iiiiiiiiiiii 1001 rrr		NCZ	Reg=Reg+Inm PC=PC+1	Sumar registro con dato inmediato y almacenar resultado en registro, y saltar a la siguiente instrucción.
	ADDRM	Reg,Mem	mmmmmmmmmmmm 0011 rrr		NCZ	Reg=Reg+Mem PC=PC+1	Sumar registro con dato en memoria y almacenar resultado en el registro, y saltar a la siguiente instrucción.
	ADDRMX	Reg	----------1000 0000 rrr		NCZ	Reg=Reg+Mem[X] PC=PC+1	Sumar registro con dato en memoria direccionada por el registro X, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	SUBYM	Mem	mmmmmmmmmmmm 1111 001		NCZ	Y=Y-Mem PC=PC+1	Restar al registro Y dato en memoria, almacenar resultado en el registro Y, y saltar a la siguiente instrucción.
	SUBRMX	Reg	----------1001 0000 rrr		NCZ	Reg=Reg-Mem[X] PC=PC+1	Restar a registro dato en memoria direccionada por X, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	MULXM	Mem	mmmmmmmmmmmm 1111 000		NCZ	{Y}{X}=X*Mem PC=PC+1	Multiplicar el registro X con dato en memoria, almacenar el resultado en el registro X, y saltar a la siguiente instrucción.
	MULRMX	Reg	----------1010 0000 rrr		NCZ	{Y}{Reg}=Reg*Mem PC=PC+1	Multiplicar registro con dato en memoria direccionada por el registro X, almacenar resultado en el registro, y saltar a la siguiente instrucción.
Lógico	NOTR	Reg	----------0111 0000 rrr		NCZ	Reg=!Reg PC=PC+1	Negar un registro, almacenar resultado en el mismo, y saltar a la siguiente instrucción.
	CMPR	Reg	----------0110 0000 rrr		NCZ	Reg=-Reg PC=PC+1	Complementar a dos un registro, almacenar resultado en el mismo, y saltar a la siguiente instrucción.
	ANDRI	Reg,Inm	iiiiiiiiiiii 1010 rrr		NCZ	Reg=Reg&Inm PC=PC+1	Realizar la operación *AND* entre registro e inmediato, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	ANDRM	Reg,Mem	mmmmmmmmmmmm 0100 rrr		NCZ	Reg=Reg&Mem PC=PC+1	Realizar la operación *AND* entre registro y dato en memoria, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	ANDRMX	Reg	----------1011 0000 rrr		NCZ	Reg=Reg&Mem[X] PC=PC+1	Realizar la operación *AND* entre registro y dato en memoria direccionada por el registro X, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	IORRI	Reg,Inm	iiiiiiiiiiii 1011 rrr		NCZ	Reg=Reg|Inm PC=PC+1	Realizar la operación *OR* inclusiva entre registro e inmediato, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	IORRM	Reg,Mem	mmmmmmmmmmmm 0101 rrr		NCZ	Reg=Reg|Mem PC=PC+1	Realizar la operación *OR* inclusiva entre registro y dato en memoria, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	IORRMX	Reg	----------1100 0000 rrr		NCZ	Reg=Reg|Mem[X] PC=PC+1	Realizar la operación *OR* inclusiva entre registro y dato en memoria direccionada por el registro X, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	XORRI	Reg,Inm	iiiiiiiiiiii 1100 rrr		NCZ	Reg=Reg⊕Inm PC=PC+1	Realizar la operación *OR* exclusiva entre registro e inmediato, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	XORRM	Reg,Mem	mmmmmmmmmmmm 0110 rrr		NCZ	Reg=Reg⊕Mem PC=PC+1	Realizar la operación *OR* exclusiva entre registro y dato en memoria, almacenar resultado en el registro, y saltar a la siguiente instrucción.
	XORRMX	Reg	----------1101 0000 rrr		NCZ	Reg=Reg⊕Mem[X] PC=PC+1	Realizar la operación *OR* exclusiva entre registro y dato en memoria direccionada por el registro X, almacenar resultado en el registro, y saltar a la siguiente instrucción.

Tabla 2.1: Set de instrucciones del procesador.

Tipo	Mnemotécnico		Código de Operación	Banderas	Descripción	
	Nombre	Parámetros	ADR/DAT INS REG	Afectadas	Aritmética	Textual
Desplazamiento	RRR	Reg	--------0010 0000 rrr	NCZ	Reg=Reg≫1 PC=PC+1	Rotar a la derecha registro, almacenar resultado en el mismo, y saltar a la siguiente instrucción.
	RLR	Reg	--------0011 0000 rrr	NCZ	Reg=Reg≪1 PC=PC+1	Rotar a la izquierda registro, almacenar resultado en el mismo, y saltar a la siguiente instrucción.
	MRR	Reg	--------0100 0000 rrr	NCZ	Reg=Reg/2 PC=PC+1	Mover registro a la derecha, almacenar resultado en el mismo, y saltar a la siguiente instrucción.
	MLR	Reg	--------0101 0000 rrr	NCZ	Reg=Reg*2 PC=PC+1	Mover registro a la izquierda, almacenar resultado en el mismo, y saltar a la siguiente instrucción.
Carga	MVIR	Inm,Reg	iiiiiiiiiiii 1101 rrr		Reg=Inm PC=PC+1	Cargar dato inmediato en registro, y saltar a la siguiente instrucción.
	MVMR	Mem,Reg	mmmmmmmmmmmm 0111 rrr		Reg=Mem PC=PC+1	Cargar dato en memoria en registro, y saltar a la siguiente instrucción.
	MVMXR	Reg	--------1110 0000 rrr		Reg=Mem[X] PC=PC+1	Cargar dato en memoria direccionada por el registro X en registro, y saltar a la siguiente instrucción.
Almacenamiento	MVRM	Reg,Mem	mmmmmmmmmmmm 1000 rrr		Mem=Reg PC=PC+1	Almacenar dato en registro en memoria, y saltar a la siguiente instrucción.
	MVRMX	Reg	--------1111 0000 rrr		Mem[X]=Reg PC=PC+1	Almacenar dato en registro en memoria direccionada por el registro X, y saltar a la siguiente instrucción.
Salto	JMP	Adr,z	aaaaaaaaaaaa 1110 -00		PC=(Z=0?Adr:PC+1)	Saltar si la bandera *Z*, "cero", se encuentra en alto.
	JMP	Adr,c	aaaaaaaaaaaa 1110 -01		PC=(C=0?Adr:PC+1)	Saltar si la bandera *C*, "acarréo", se encuentra en alto.
	JMP	Adr,n	aaaaaaaaaaaa 1110 -10		PC=(N=0?Adr:PC+1)	Saltar si la bandera *N*, "negativo", se encuentra en alto.
	JMP	Adr,a	aaaaaaaaaaaa 1110 -00		PC=Adr	Saltar siempre.
	JMPR	Reg	----------0 0001 rrr		PC=PC+1+Reg	Saltar relativamente al dato en registro.
Reset	—	—	------------ ---- ---		PC=0	Saltar automáticamente a la dirección "0" en caso de producirse el *reset* del procesador.
Interrupción	—	—	------------ ---- ---		PC=Itr_Adr STK[S]=PC	Saltar automáticamente a la rutina de interrupción en caso de producirse la misma, y almacenar la dirección de la instrucción actual en la pila.
Otros	CALL	Adr	aaaaaaaaaaaa 1111 -10		STK[S]=PC+1 PC=Adr S=S+1	Almacenar la dirección de la siguiente instrucción en la pila, saltar a subrutina, e incrementar el contador de la pila.
	RET	—	----------1 0001 ---		PC=STK[S]	Retornar de subrutina.
	RETIR	Inm,Reg	iiiiiiiiiiii 0010 rrr		Reg=Inm PC=STK[S]	Cargar dato inmediato en registro y retornar de subrutina.
	NOP	—	------------ 1111 -11		PC=PC+1	No realizar ninguna operación y saltar a la siguiente instrucción.

Tabla 2.2: Set de instrucciones del procesador, continuación.

Capítulo 3

Síntesis y análisis de resultados

En el capítulo anterior se expuso la metodología y consideraciones utilizadas para el desarrollo de la descripción a nivel de sistema y comportamiento del microprocesador. En este capítulo se aplica el flujo de diseño *top-down* desarrollado, al procesador configurado a 16 bits. Así mismo se presentan análisis al variar la cantidad de bits y su efecto sobre el desempeño del procesador. Finalmente, se resumen las observaciones y conclusiones realizadas a lo largo de este documento y se listan una serie de recomendaciones para trabajos futuros.

3.1. Aplicación del flujo de diseño al modelo *RTL* del procesador

Usando las herramientas de *software* de la empresa *Cadence*, se implementó un flujo de diseño para la síntesis y trazado de cualquier circuito digital. Con el fin de agilizar el proceso de diseño y automatizar las tareas del mismo, se programó una interfaz gráfica de usuario bajo el entorno de desarrollo *Gambas*; en la figura 3.1 se puede apreciar una captura del mismo. Éste programa se encarga de configurar las herramientas y ayudar al diseñador a organizar su trabajo. Bajo este entorno se puede realizar la verificación la funcional del diseño en las tres principales etapas del proceso, además de permitir el análisis de desempeño del mismo en los tres dominios conceptuales.

Implementando el flujo de diseño desarrollado, se procede a llevar el modelo *RTL* del procesador hasta la representación geométrica del circuito integrado o *layout*. A continuación se exponen los resultados de cada etapa del flujo de diseño, para un procesador con las características registradas en la tabla 3.1.

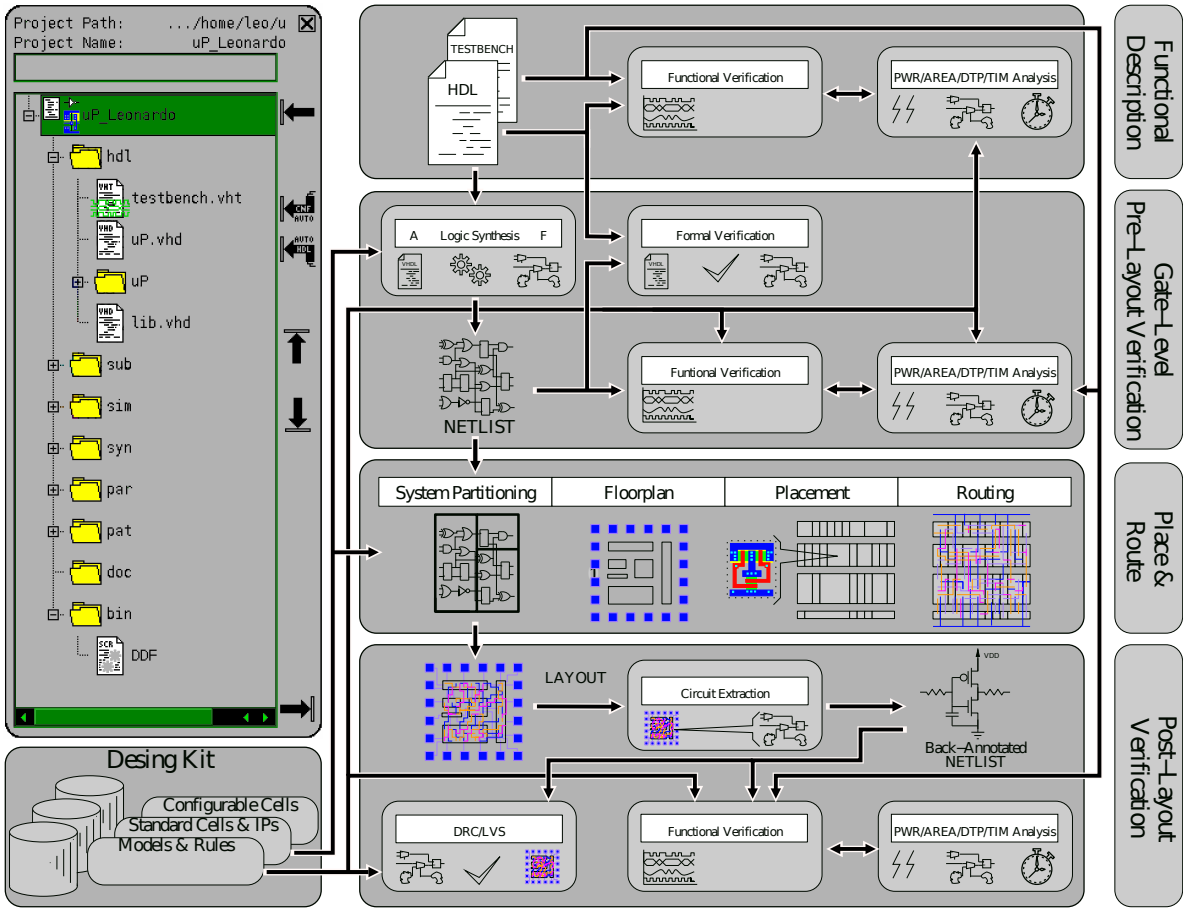


Figura 3.1: Interfaz gráfica de usuario del flujo de diseño programado.

Característica	Valor
Ancho del bus de datos	16 bits
Ancho del bus de programa	23 bits
Ancho del bus de direccionamiento de programa	16 bits
Tamaño de la memoria de programa	$16 \cdot 64[Kbits]$
Ancho del bus de datos	16 bits
Ancho del bus de direccionamiento de datos	16 bits
Tamaño de la memoria de datos	$16 \cdot 64[Kbits]$
Número de registros de propósito general	4
Número total de registros	8
Número de niveles de la pila	4

Tabla 3.1: Características del procesador sintetizado.

3.1.1. Dominio de comportamiento

Descripción funcional

Se describió el procesador mediante el código de descripción de *hardware VHDL*, mediante el estilo *RTL*. Este diseño contempla la definición de los parámetros del procesador, lo que permite variar los mismos a fin de adaptarlo a un propósito específico. Un ejemplo es la necesidad de usar una mayor cantidad de memoria de programa, en este caso se altera la constante que define el ancho del bus de direccionamiento de la misma, modificando todos los dispositivos asociados con este valor.

Análisis de sintaxis

Se realiza un análisis del código *VHDL* con el fin de identificar errores en el mismo antes de lanzar el simulador. Este proceso se realiza con la herramienta *NC-VHDL* de *Cadence*.

Verificación funcional

El diseño fue validado mediante simulación, analizando las señales tanto internas como externas al microprocesador. Para ello se usó el programa de prueba descrito en el capítulo 2, ensamblado con la herramienta desarrollada para tal fin. Se modificó el archivo de señales de entrada-salida incluyendo los respectivos campos y generando los eventos de *reset* e *interrupción*. Para realizar las simulaciones se utiliza un conjunto de herramientas configuradas automáticamente por el programa del flujo de diseño. Una vista detallada del proceso se muestra en la figura 3.2. Nótese que durante la simulación se registran las conmutaciones de las líneas, lo que permite calcular el consumo de potencia del procesador ejecutando un determinado programa, lo que hace más precisa la medición en el proceso de análisis.

A fin de controlar la introducción de las señales se programa un banco de pruebas usando el lenguaje *VHDL*. Éste *testbench* se encarga de leer línea por línea el archivo de señales de entrada y almacenando las salidas arrojadas por la simulación del modelo, lo que permitirá que en posteriores etapas se comparen los resultados arrojados por el diseño verificando así el comportamiento del procesador. El *testbench* también fue modificado para incluir los bloques de memoria externos y las líneas de transmisión necesarias para la simulación del procesador. Un diagrama de este programa se representa en la figura 3.3.

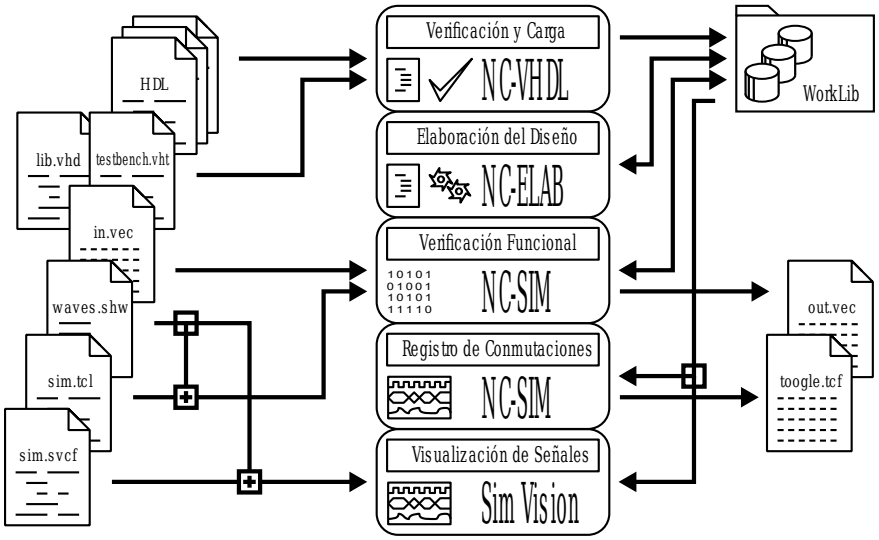


Figura 3.2: Flujo detallado del proceso de verificación funcional del código *HDL*.

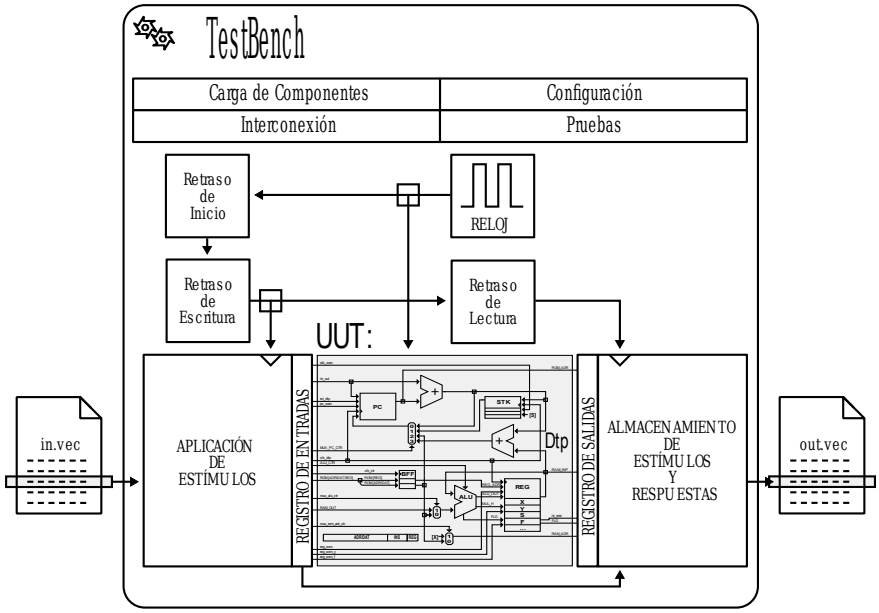


Figura 3.3: Flujo detallado del *TestBench* durante la verificación funcional del código *RTL*.

Análisis

Mediante una presíntesis, se analiza la correcta descripción del modelo *VHDL* y las proyecciones de las características de desempeño del procesador. Mediante este análisis se identifica la ruta crítica, que comienza en la salida del *buffer* de datos de la memoria de programa, atraviesa un multiplexor, la ALU y termina en el registro de banderas. Además, se estima el

área ocupada por cada bloque del procesador y su consumo de potencia; los resultados se graficaron junto con datos de etapas posteriores en las figuras 3.10, 3.12 y 3.11 respectivamente. El proceso se expone en la figura 3.4 y es implementado mediante la herramienta de síntesis lógica de *Cadence*, *RTL Compiler*.

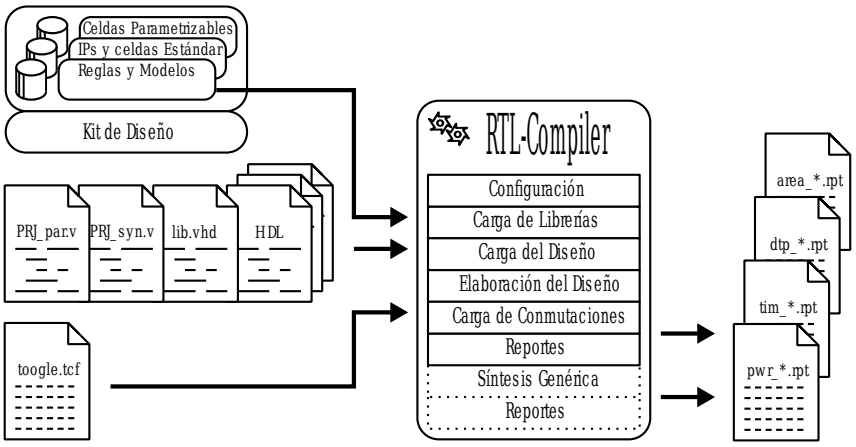


Figura 3.4: Flujo detallado del proceso de análisis de potencia, área ocupada, *datapath*, y retrasos de señal.

3.1.2. Dominio estructural

Síntesis lógica

Se sintetiza el procesador mediante la herramienta *RTL Compiler*, la cual toma las señales internas del microprocesador para optimizar el consumo de potencia de los bloques más usados. Además se usa la técnica *clock gatting* para reducir el consumo de los bloques de almacenamiento deshabilitados. Durante este proceso se obtiene el *netlist* en *Verilog* con los retrasos estimados que tendrá el diseño y las restricciones de tiempo en las señales de cada bloque. Los pasos realizados durante la síntesis lógica se representan en la figura 3.5.

Verificación formal

En esta etapa se verifica el esquemático generado por la herramienta de síntesis lógica, a fin de comprobar si la descripción funcional fue realizada y sintetizada correctamente. Este proceso se realiza mediante la interfaz gráfica del *RTL Compiler* una vez se ha realizado la síntesis lógica.

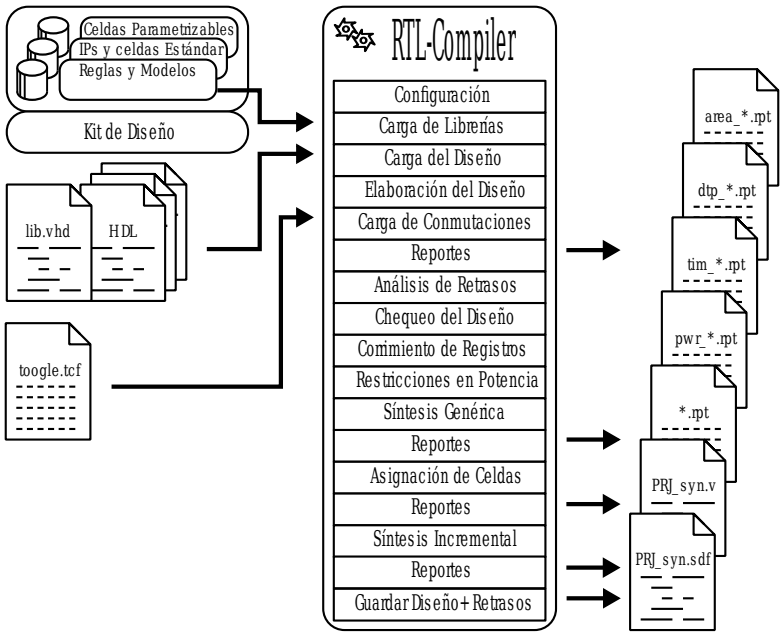


Figura 3.5: Flujo detallado del proceso de síntesis lógica.

Verificación funcional

Durante este proceso se verifica si el circuito que se arrojó se comporta de la misma manera que el modelo, frente a las mismas señales de entrada. Además, con los datos sobre los retrasos de señal, es posible predecir mediante simulaciones la frecuencia máxima de operación a la que podrá trabajar el dispositivo, para este caso 27,0269$[MHz]$. A diferencia de la verificación funcional *HDL*, en este punto se leen las señales arrojadas por el modelo *VHDL* y se comparan con las del diseño sintetizado, por ello el proceso cambia como se muestra en las figuras 3.6 y 3.7. En esta etapa se carga el *netlist* arrojado por el sintetizador y se anotan los retrasos de las celdas y una aproximación estadística de las interconexiones.

Análisis

Se analiza los parámetros de desempeño mediante estimaciones más precisas de las interconexiones y el consumo de potencia, gracias a la extracción de las capacitancias de las celdas utilizadas. Los datos se pueden confrontar con otros análisis en las figuras 3.10, 3.12 y 3.11. El proceso es similar al análisis del modelo *HDL*, pero se realiza a partir de los datos de simulación postsíntesis.

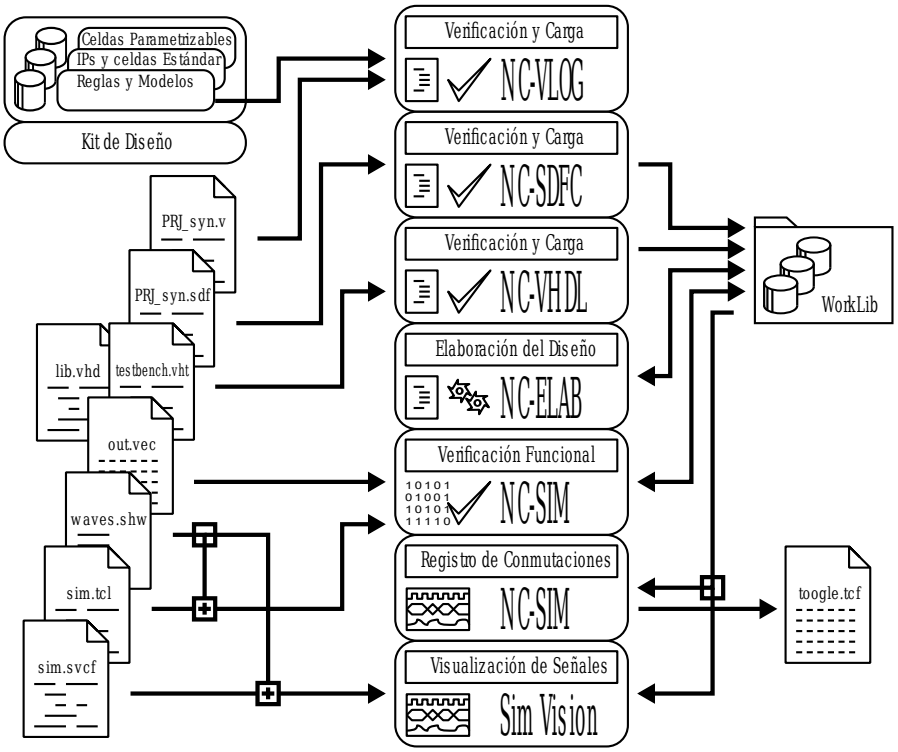


Figura 3.6: Flujo detallado del proceso de verificación funcional postsíntesis.

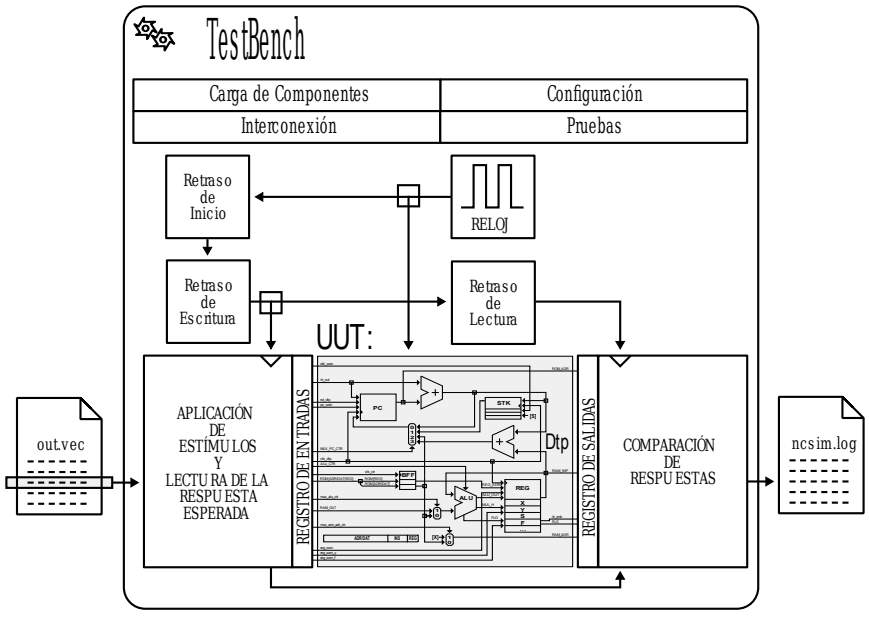


Figura 3.7: Flujo detallado del *TestBench* durante la verificación funcional postsíntesis y *postlayout*.

3.1.3. Dominio físico

Posicionamiento y trazado de celdas estándar

Una vez generado el *netlist* con las celdas a usar, se procede a saltar al dominio físico con el fin de generar el *layout* o conjunto de máscaras de fabricación. En esta etapa del proceso se posicionan las celdas y se trazan las interconexiones de acuerdo a su importancia.

Debido a la estructura de las celdas estándar, el trazado de la alimentación es muy simple ya que se generan las líneas de alimentación al situar una al lado de la otra. Aún así, es necesario trazar líneas adicionales para los elementos que consuman mayor potencia a fin de reducir los efectos de la temperatura en las conexiones de alimentación. En este caso, se trazó un par de líneas de potencia cerca de la ALU ya que es el bloque con mayor consumo de energía.

Las señales de reloj son un caso especial del trazado, ya que debido a que muchos elementos dependen de la adecuada sincronización y refuerzo como los registros internos. Para ello se debe trazar un árbol de reloj, que se encarga de reforzar la señal y retrasarla de manera constante mediante la adición de inversores y *buffers*, con el fin de que llegue a los registros casi al mismo tiempo.

Seguidamente se trazan las interconexiones entre las celdas del procesador y se realiza un preajuste del *layout* a fin de reservar espacio para el posicionamiento de los pines de interconexión con el encapsulado.

Debido a la complejidad de este proceso, no se usa todas las capas de las celdas, sino una versión simplificada de las mismas, lo que permite considerar las áreas de mayor relevancia para el trazado. Una vez se obtiene el *layout*, es posible extraer los elementos parásitos asociados a las líneas de interconexión y las celdas, los cuales se podrán anotar en simulaciones posteriores. Una vista detallada del proceso de posicionamiento y trazado automático es mostrada en la figura 3.8 durante el que se usa la herramienta *SoC Encounter*.

Una vista del trazado en esta etapa del procesador sintetizado se muestra en la figura 3.9.

Verificación funcional

Una vez generado el *layout*, se realiza una última verificación funcional para obtener los valores finales de desempeño y corroborar el funcionamiento del diseño. La máxima frecuencia de operación obtenida en esta etapa es de $51{,}5756[MHz]$, mayor que la posterior a la síntesis debido a que en este punto no se estiman las interconexiones, que estadísticamente son de

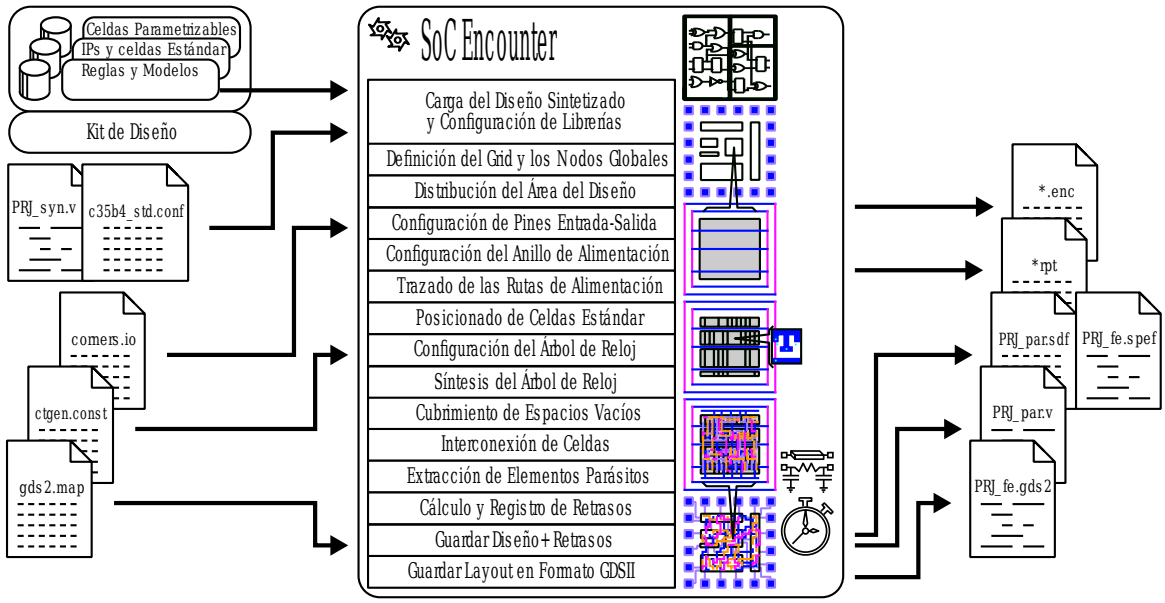


Figura 3.8: Flujo detallado del proceso de posicionamiento y trazado de celdas estándar.

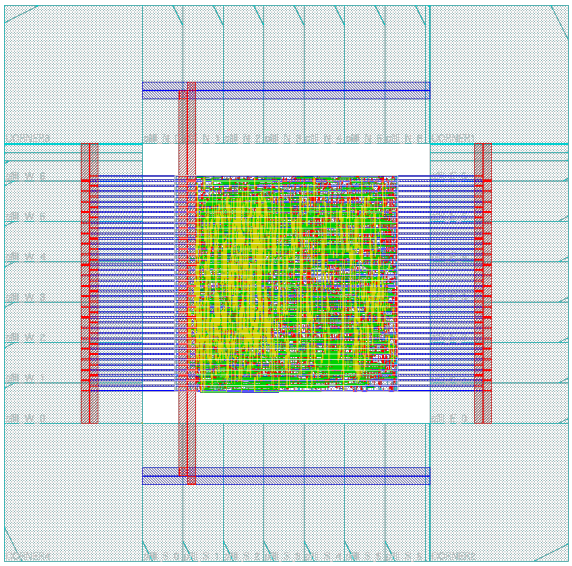


Figura 3.9: *Layout* del procesador con celdas simplificadas.

mayor tamaño. Lo anterior demuestra que la metodología de diseño implementada arroja mejores resultados que los que comúnmente se alcanzarían. El proceso es el mismo que en la verificación postsíntesis por lo que no se entrará en detalles.

Análisis

El análisis *post-layout* permitirá una medición más aproximada del consumo de potencia del procesador y otras variables de desempeño. En las figuras 3.10, 3.12 y 3.11 se representa los análisis realizados. Nótese la gran diferencia ente el área ocupada por las celdas y las interconexiones, eso indica que el espacio ocupado por las primeras es suficiente para contener a las segundas, lo que permite un mayor aprovechamiento del silicio.

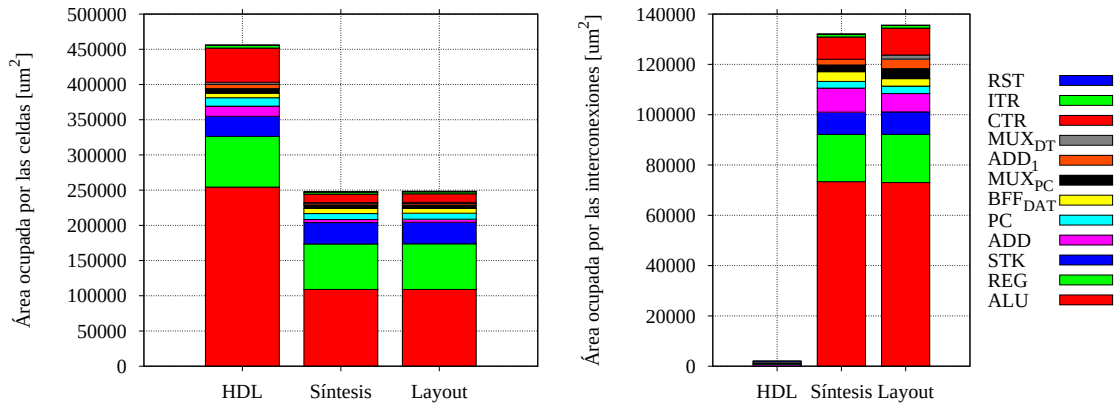


Figura 3.10: Proporción de área ocupada por cada bloque del procesador y sus interconexiones.

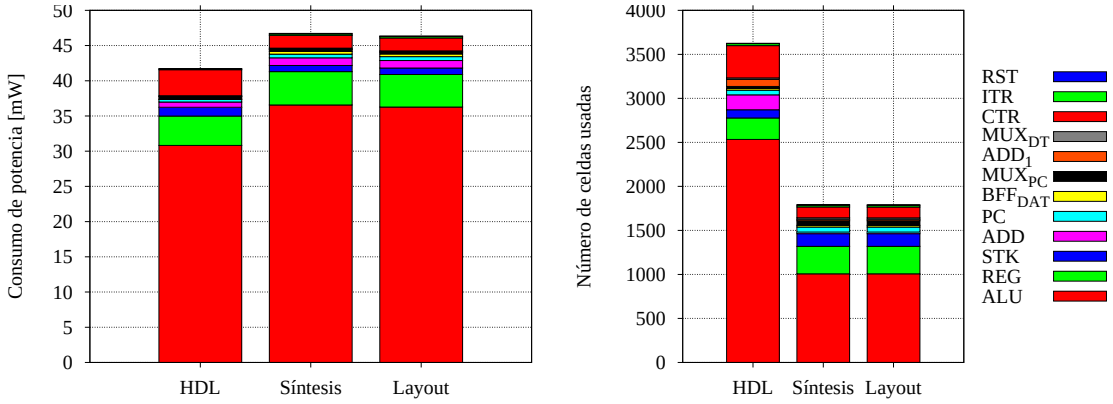


Figura 3.11: Consumo de potencia a 27,0269$[MHz]$ y cantidad de celdas usadas por cada bloque del procesador.

3.1.4. Ajuste del *layout*

Un ajuste final del *layout* es necesario para remplazar las celdas simplificadas por las celdas completas, lo que permitirá obtener todas las capas necesarias para la fabricación del micro-procesador. Éste proceso se realiza de manera manual en el entorno analógico de *Cadence*:

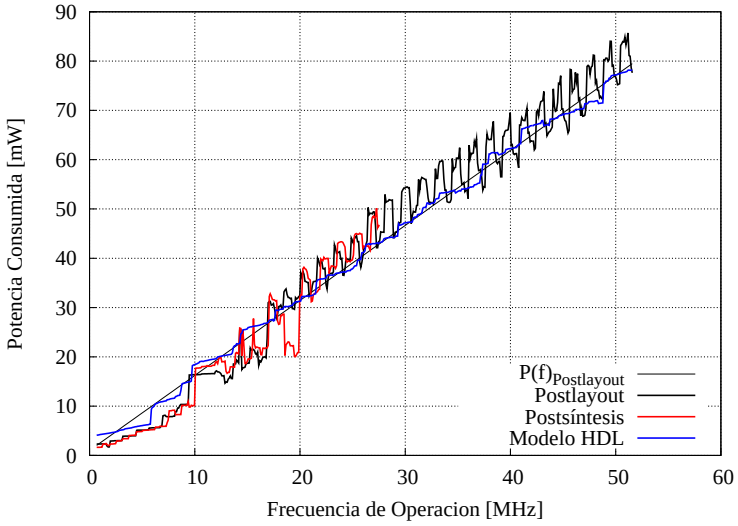


Figura 3.12: Consumo de potencia del procesador a diferentes frecuencias de operación.

Virtuoso. En ese entorno, mediante una lista de equivalencia, se reemplaza cada celda, obteniéndose el *layout* final. Una captura del mismo se muestra en la figura 3.13. El área ocupada por cada sección del diseño se registra en la tabla 3.2

Elemento	Área ocupada
Celdas Estándar	293256,600[μm^2]
Núcleo	293802,925[μm^2]
Pads	1420693,440[μm^2]
Total	1914848,440[μm^2]

Tabla 3.2: Área ocupada por los diferentes elementos del procesador.

3.2. Análisis del diseño al configurar sus parámetros

Gracias a la descripción del procesador y al flujo de diseño automatizado, es posible realizar una serie de análisis que permitan determinar como varían las características del mismo, al modificar sus parámetros. Una serie de *scripts* se programaron a fin de reconfigurar el diseño, simularlo y trazarlo según fuese necesario.

3.2.1. Máxima frecuencia de operación

En la figura 3.14 se pueden identificar los bloques que afectan el desempeño en frecuencia del procesador. Como se predijo durante el diseño, el máximo retraso de la unidad de control es

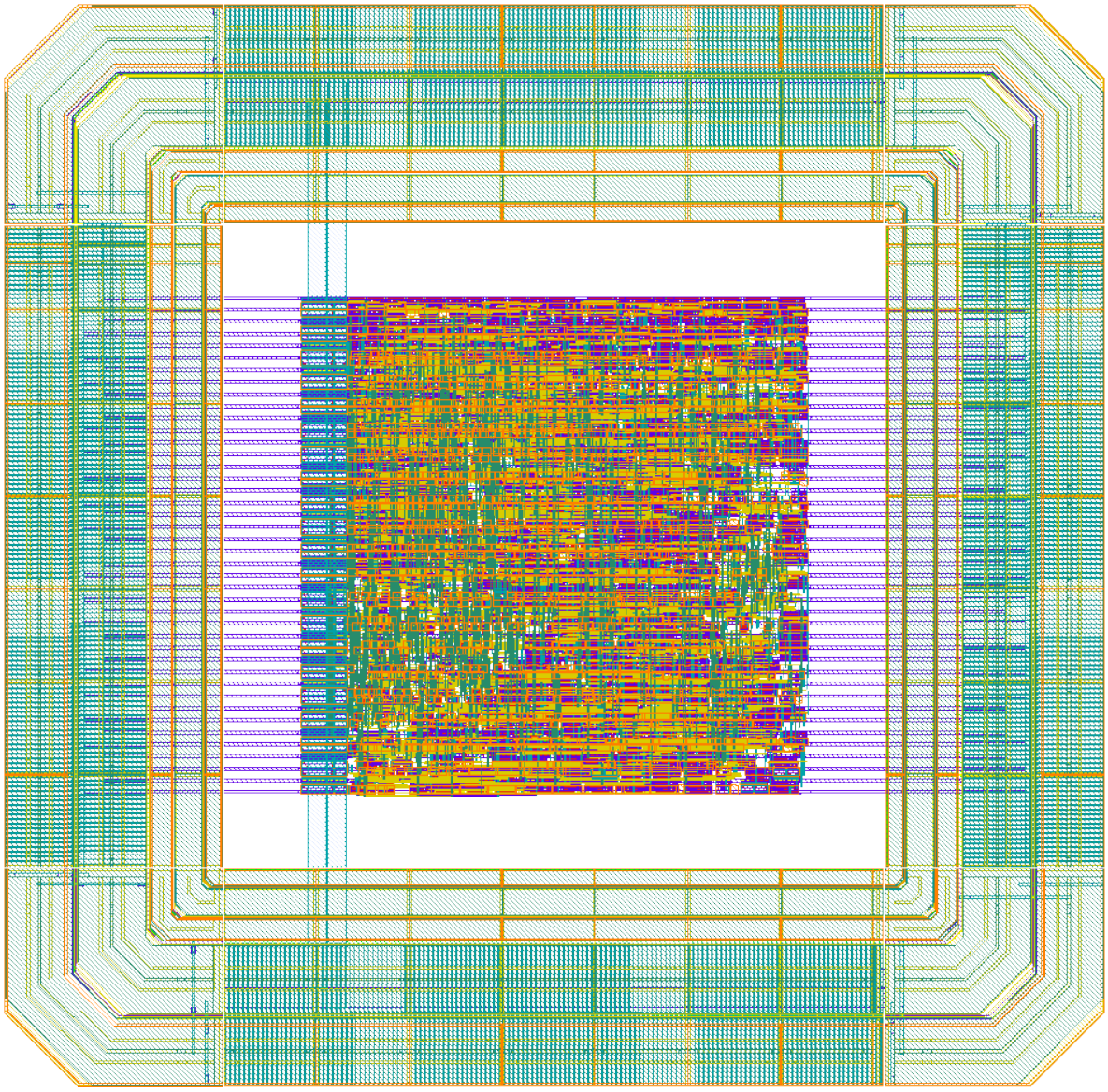


Figura 3.13: *Layout* final de procesador sintetizado.

aproximadamente el mismo al variar la cantidad de bits con que puede operar el procesador. Las variaciones observadas son debidas a las diferentes interconexiones internas de los diseños.

3.2.2. Consumo de potencia

La reciente tendencia en el diseño de procesadores se puede extraer del análisis representado en la figura 3.15, dónde se varió la cantidad de bits del procesador con respecto a la frecuencia de operación. Se observa que un procesador de 32 bits consume un poco más de cinco veces

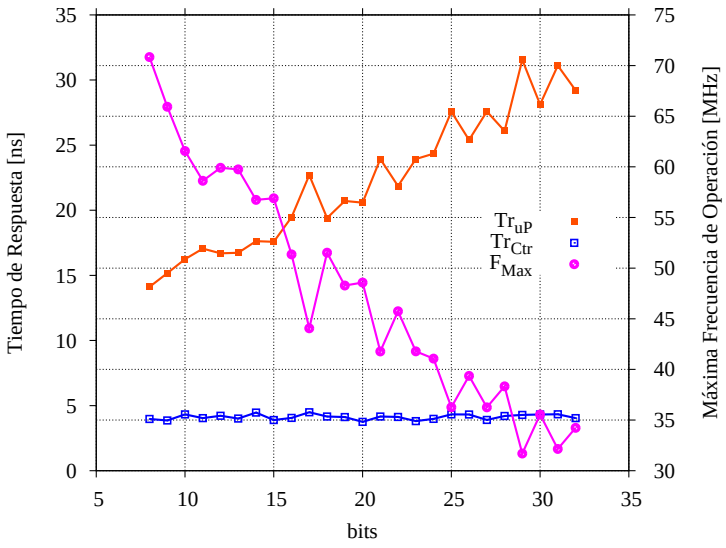


Figura 3.14: Tiempo de respuesta de los bloques internos del microprocesador y su efecto sobre la máxima frecuencia de operación.

más que uno de 8 bits trabajando a una frecuencia máxima menor. Por lo tanto, utilizar cuatro procesadores de 8 bits puede reducir el consumo en una quinta parte mientras se procesa la misma cantidad de bits, trabajando a una mayor frecuencia para compensar el proceso de separación de datos. Ésta es la razón por la cual se usa sistemas multinúcleo en el desarrollo de los nuevos procesadores.

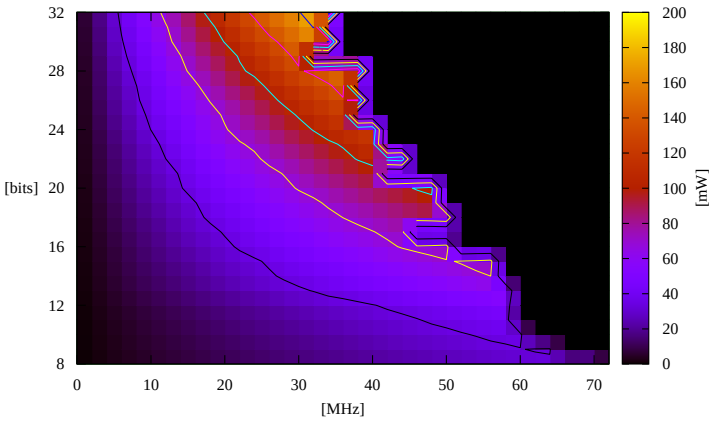


Figura 3.15: Relación: consumo de potencia, frecuencia de operación y el número de bits.

3.2.3. Área ocupada

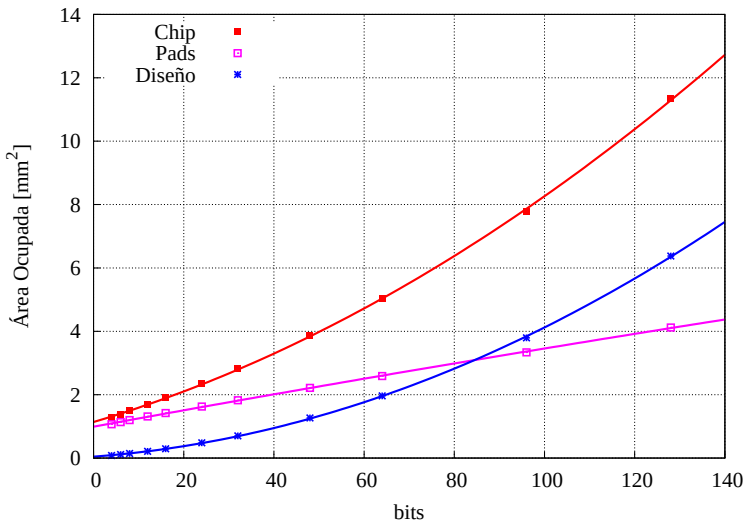


Figura 3.16: Distribución del área ocupada según el número de bits.

Como se corroboró durante la síntesis del procesador a 16 bits, el área ocupada por el espacio reservado para los *pads* de interconexión es mayor que el área efectiva usada por el procesador. Esto implica que la fabricación de un microprocesador como elemento aislado no es económicamente factible. El punto de equilibrio de la distribución de área se puede observar en la figura 3.16 y corresponde a un procesador de 85 bits. Para diseños de menor complejidad es recomendable la integración en un sistema embebido como un microcontrolador.

3.3. Observaciones y conclusiones

- Es posible mejorar las características de un microprocesador mediante optimizaciones realizadas por el diseñador a nivel de sistema.
- El uso de la técnica de procesamiento en *pipeline* no es el único método para elevar la frecuencia de procesamiento.
- El uso de varias fuentes de reloj, aunque complica el diseño y la sincronización, permite disminuir el área ocupada y el consumo de energía del dispositivo.
- Se realizó el diseño reconfigurable de un procesador que ejecuta 38 instrucciones, cada una en un ciclo de reloj.
- El procesador diseñado cuenta con un sistema de atención y control de interrupciones,

lo que permite responder de manera inmediata a la solicitud.

- A diferencia de un procesador en *pipeline*, el esquema propuesto no presenta problemas para realizar saltos de programa.
- Se redujo el consumo de potencia sin afectar significativamente la máxima frecuencia de operación.
- Se redujo el área ocupada mediante simplificación del sistema y eliminación de líneas de interconexión redundantes.
- Se define el set de instrucciones a partir del *hardware*, considerando las operaciones más comunes realizadas por un procesador de propósito general.
- Aunque el flujo de diseño *top-down* permite describir el microprocesador en un mayor nivel de abstracción, es necesario analizar las implicaciones a bajo nivel que conlleva implementar una determinada descripción de *hardware*.
- No es siempre necesario el uso de una máquina de varios estados para el control de un *datapath*.
- Para reducir la probabilidad de que se produzcan estados metaestables en los circuitos internos, es necesaria una sincronización de las señales externas y una adecuada separación de tiempo entre cambios de señal.
- El desarrollo de un flujo de diseño automatizado, permitió la simulación, síntesis, análisis y trazado del diseño en *VHDL* en un tiempo reducido.
- La simulación no es suficiente para asegurar la adecuada síntesis de un diseño digital. Es necesario verificar que se ha realizado correctamente la descripción de hardware, revisando la estructura interna del diseño sintetizado.
- Un diseño puede no ser económicamente factible, si requiere menos área que sus conexiones con el encapsulado.
- Se desarrollaron herramientas que facilitaron la comparación del desempeño del procesador al variar sus parámetros internos.
- La reducción de potencia no sólo es responsabilidad del diseñador del dispositivo, sino que juega un rol importante el usuario del mismo. En el caso de un procesador, un programa puede ser optimizado con el fin de reducir el consumo de energía, al evitar el uso de instrucciones o registros que impliquen mayor uso de la misma.

3.4. Recomendaciones para trabajos futuros

- Elevar la máxima frecuencia de operación implementando un *pipeline* sencillo.
- Describir la unidad aritmético-lógica a más bajo nivel para reducir su consumo de potencia y el tiempo de respuesta.
- Diseñar la fuente de reloj principal y su derivada.
- Desarrollo de un procesador multinúcleo y sus elementos asociados.
- Aplicar otras técnicas de reducción de potencia como el uso de diferente fuentes de alimentación o el diseño a nivel analógico en la región se subumbral.
- Diseño de los bloques de almacenamiento tanto para la memoria de programa como la de datos.
- Diseño del bloque de reset al encendido, o *Power On Reset*.
- Integrar el μP en un sistema funcional más complejo como un microcontrolador, en el cual puede usarse la memoria de datos como espacio de direccionamiento de los módulos internos y periféricos.
- Programar un ensamblador más eficiente, en lenguaje *C* o similar.
- Fabricación del procesador para validar mediante pruebas reales el funcionamiento del diseño.
- Organización de un plan de pruebas a fin de estructurar el proceso de verificación funcional del circuito físico.
- Modificar el flujo automatizado de manera que permita el uso del lenguaje *Verilog* y *SistemVerilog* para la descripción de hardware.
- Agregar al flujo automatizado más kits de diseño en diferentes tecnologías.

Bibliografía

[1] A. S. Sedra and K. C. Smith, *Circuitos Microelectrónicos*. Mc. Graw Hill, 2006.

[2] M. J. S. Smith. (1997, Jun) Application-Specific Integrated Circuits. Ready Inc. [Internet]. Visite: http://www10.edacafe.com/book/ASIC/

[3] José Ignacio Martínez Torre. (1997, Jun) Codiseño Hardware-Software, transparencias, Conceptos generales sobre sistemas digitales. Universidad Rey Juan Carlos, Grupo de Diseño Hardware – Software. [Internet]. Visite: http://www.escet.urjc.es/~jmartine/CHS/Documentacion.htm

[4] D. Thomas, "The automatic synthesis of digital systems," *Proceedings of the IEEE*, vol. 69, no. 10, pp. 1200 – 1211, oct. 1981.

[5] A. C. Parker, "Automated Synthesis of Digital systems," *Design Test of Computers, IEEE*, vol. 1, no. 4, pp. 75 –81, nov. 1984.

[6] José Luis González. (2005, Dic) Seminario de Herramientas de Diseño Microelectrónico CAD I. Universitat Politècnica de Catalunya. [Internet]. Visite: http://www.eel.upc.edu/cad1/

[7] Alain Vachoux. (2005, Dic) Top-down digital design flow. Ecole Polytechnique Fédérale de Lausanne. [Internet]. Visite: http://lsm.epfl.ch/webdav/site/lsm/users/104020/public/Topdown_DF.pdf

[8] Per Lindgren. (2006, Jun) VLSI Design Course. Luleå University of Technology. [Internet]. Visite: http://www.sm.luth.se/csee/courses/smd/154/

[9] I. Cadence Design Systems, *Cadence NC-VHDL Simulator Tutorial with SimVision*, 2006.

[10] ——, *ICC Code Coverage User Guide*, 2006.

[11] ——, *Using Encounter RTL Compiler*, 2005.

[12] ——, *interfacing Between RTL Compiler and Conformal LEC*, 2005.

[13] ——, *Pearl Timing Analyzer User Guide*, 2003.

[14] Peter Y. K. Cheung. (2008, Sep) Digital IC Design. Imperial College of Science, Technology and Medicine,University of London. [Internet]. Visite: http://www.ee.ic.ac.uk/pcheung/teaching/ee4_asic/index.html

[15] M. J. Irwin and V. Narayanan, copyright 2002 J. Rabaey et al. (2002, Oct) VLSI Digital Circuits, Lecture Slides. Penn State University. [Internet]. Visite: http://bwrc.eecs.berkeley.edu/icbook/slides_PennState.htm

[16] Richard Van Slyke. (2006, Jun) Computer Architecture and Organization, Lecture Notes. Polytechnic Institute of New York University. [Internet]. Visite: http://cis.poly.edu/cs2214rvs/

[17] J. M. Rabaey and A. Chandrakasan, *Digital Integrated Circuits - A Design Perspective*. Prentice-Hall, 1996.

[18] I. Cadence Design Systems, *Low Power in Encounter RTL Compiler*, 2007.

[19] F. Pardo and J. A. Boluda, *VHDL, Lenguaje para Síntesis y Modelado de Circuitos*. Alfaomega, 1999.

[20] S. M. Ovallos, "Diseño de un microprocesador usando el lenguaje de descripción de hardware VHDL," Proyecto de pregrado de la Universidad Industrial de Santander, 2005.

[21] I. Cadence Design Systems, *HDL Modeling in Encounter RTL Compiler*, 2005.

[22] Kris Gaj. (2008, Feb) FPGA and ASIC Design with VHDL. Department of Electrical and Computer Engineering at George Mason Univerity in Fairfax, Virginia, U.S.A. [Internet]. Visite: http://teal.gmu.edu/courses/ECE448/index.htm

[23] I. Cadence Design Systems, *Design For Test in Encounter RTL Compiler*, 2008.

[24] W. Cui and S. Wu, "Design of Small Area and Low Power Consumption Mask ROM," in *Integrated Circuit Design and Technology, 2007. ICICDT '07. IEEE International Conference on*, may 2007, pp. 1 –4.