APROPIACIÓN TECNOLÓGICA DEL DISEÑO DE EMBEDEDD SYSTEMS IMPLEMENTADOS SOBRE FPGAs Y CPLDs.

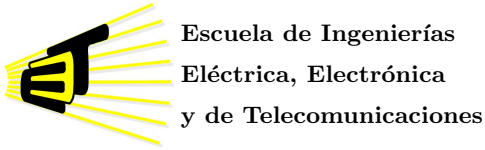
Ing. Carlos Augusto Fajardo Ariza

Trabajo de investigación presentado como requerimiento parcial para optar al título de:

Magister en Ingeniería Electrónica

Director:

MsC. Jorge Hernando Ramón



Universidad Industrial de Santander

Facultad de Ingenierías Físico Mecánicas

Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

Noviembre, 2010

APROPIACIÓN TECNOLÓGICA DEL DISEÑO DE EMBEDEDD SYSTEMS IMPLEMENTADOS SOBRE FPGAs Y CPLDs.

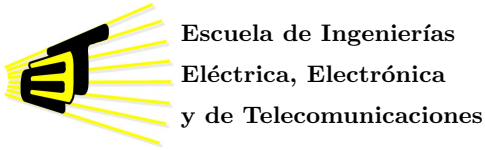
Ing. Carlos Augusto Fajardo Ariza

Trabajo de investigación presentado como requerimiento parcial para optar al título de:

Magister en Ingeniería Electrónica

Director:

MsC. Jorge Hernando Ramón



Universidad Industrial de Santander

Facultad de Ingenierías Físico Mecánicas

Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

Noviembre, 2010

Agradecimientos

Al cerrar este nuevo capitulo de mi vida, tengo que reconocer que este logro no hubiese sido posible sin el apoyo de personas e instituciones que me acompañaron y apoyaron para que esta investigación llegara a feliz término.

En primer lugar quiero agradecer a la Universidad Industrial de Santander, al Grupo CPS y su Director el Dr. Oscar Gualdrón y en especial al profesor Jorge Ramón por acogerme nuevamente en su grupo de trabajo, por su apoyo y acompañamiento durante estos años. Profe le agradezco profundamente por la confianza que depositó en mi durante el desarrollo de esta tesis.

Quiero hacer un reconocimiento a todos los estudiantes que realizaron su proyecto de grado conmigo y que hicieron tan valiosos aportes a esta investigación y los cuales nombro cronológicamente: Carlos Nieto, Willian Mendez, Ana Maria Olarte, Edgar Betancourt, Carlos Jerez, Sergio Reyes, Henyer Ricardo Sepulveda, Juan Guillermo Ortiz y Jorge Pachón. Muchas gracias!!!

Quiero agradecer a mis compañeros Iván Flórez, Mauricio Erazo, Daniel Velazco, Ricardo Diaz, Carlos Angulo, Julián Rolón, José Rugeles y Diego Medina por acogerme dentro del grupo CPS. Muchachos muchas gracias por su amistad y el excelente ambiente de trabajo que me regalaron.

Quiero expresar un agradecimiento especial a Willian Salamanca y Sergio Abreo. Willian gracias por sus innumerables consejos que fueron tan oportunos y por su constante apoyo, definitivamente su humildad contrasta perfectamente con su inteligencia. Sergio gracias por ser mi consejero de cabecera en este trabajo investigativo, gracias por ser tan accesible, por su disposición constante de colaborarme y gracias por compartirme sus conocimientos y experiencia. Pero sobre todo (a los dos) muchas gracias por su amistad sincera.

Mi mas sincero agradecimiento a mi familia. A mis padres Leonor y Felipe (qepd) por su esfuerzo y dedicación, los cuales me permitieron acceder a la educación superior, a mi hermana Yidy Alexandra y su esposo, por su compañía y apoyo, por ser un ejemplo de responsabilidad y esfuerzo para mi. A mis sobrinos Daniela y Polito por ser un motivo de felicidad en mi vida.

A mi hijo David Felipe, que ha sido uno de los mejores regalos que he recibido del cielo. Gracias hijo por que eres fuente de inspiración y ánimo en mi vida. Sin darte cuenta has hecho que mis días sean más felices.

Por su puesto el mayor de mis agradecimientos es para mi esposa Margiory, que definitivamente ha sido mi mejor regalo. Gracias princesa por tu amor incondicional, paciencia y apoyo constante. Eres la ayuda perfecta en todo lo que he emprendido. Te amo.

Y aunque apareces de último en esta lista, aspiro que ocupes el primer lugar en mi corazón y en mis decisiones. Gracias Dios por la oportunidad de estudiar esta maestría y sobre todo por las personas tan valiosas que pusiste junto a mi.

Índice

Lista de Figuras

Lista de Tablas

RESUMEN

TITULO: APROPIACIÓN TECNOLÓGICA DEL DISEÑO DE *EMBEDDED SYSTEMS* IMPLEMENTADOS SOBRE FPGAs[1]

AUTOR: CARLOS AUGUSTO FAJARDO ARIZA[2]

PALABRAS CLAVE: Codiseño, Embedded Systems, FPGAs, VHDL.

Esta investigación está motivada por la creciente importancia de los ES (Embedded Systems) en el mundo y por la oportunidad que actualmente ofrece la tecnología de los FPGAs (Field Programmable Gate Array)para implementar ES en *campo*, es decir, sin depender de los costosos procesos litográficos que se realizan en *fábrica*.

La investigación permitió revisar las diferentes posibilidades que actualmente existen en cuanto a estrategias de diseño, hardware, sistemas de desarrollo y herramientas CAD en el dominio de los ES, para proponer una metodología de diseño de ES implementados sobre FPGAs. Como fruto del trabajo investigativo se documentó en un libro la apropiación tecnológica realizada, en el que se describe las generalidades del diseño de ES utilizando la tecnología de los FPGAs, además se describe el diseño de algunos ES que permiten mostrar con cierto nivel de detalle la metodología propuesta en esta tesis de maestría. Adicionalmente se plantearon una serie de laboratorios, en los cuales se proponen ejercicios de diseño que permitan explorar las posibilidades que ofrece una herramienta CAD comercial para el diseño de ES utilizando la metodología de codiseño.

Uno de los aportes más importante de esta tesis de maestría es una propuesta metodológica para el diseño de los módulos de hardware del ES, pues en el proceso investigativo se encontró vacíos en la literatura en cuanto a estrategias de diseño para este tipo de módulos. La metodología propuesta se validó con el diseño de dos procesadores de propósito específico que serán utilizados en un ES que busca acelerar la *Migración Sísmica 2D*, un proceso utilizado en la extracción de hidrocarburos.

[1] Trabajo de investigación

[2] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingenierías Electrica, Electrónica y Telecomunicaciones. Grupo CPS. Director: MsC Jorge H. Ramón

ABSTRAC

TITLE: Technological appropriation of Embedded Systems design implemented over FPGAs[3]

AUTHOR: CARLOS AUGUSTO FAJARDO ARIZA[4]

Key words: Co-design, embedded systems, FPGAS, VHDL.

This investigation is motivated by the increasing importance of the ES (embedded systems) in the world and by the opportunity that FPGA (Field Programmable Gate Array) technology offers to develop ES in field.

The investigation allowed to review the different possibilities that currently are found concerning design strategies, hardware, development systems and CAD tools (Computer-aided design) in the ES domain, to propose a design methodology of ES implemented over FPGA. Also it describes some ES designs that allow displaying, with certain level of detail, the methodology proposed in this thesis.

Additionally, a series of labs were put forward in which design exercises are proposed to explore the possibilities that a CAD tool offers to ES design using co-design methodology. One of the most important contributions of the thesis is a methodological proposal for the modules of ES hardware, because during the investigation there was a lack of literature concerning design strategies for this specific type of modules. The proposed methodology was validated with the design of two special-purpose processor that will be used in an ES to accelerate the seismic migration 2d, a process used in hydrocarbons extraction.

[3]Research work

[4]Physical Faculty of Mechanical Engineering. School of Electrical, Electronics and Telecommunications. Group CPS. Director: MsC Jorge H. Ramón.

Capítulo 1

Introducción

Con la aparición del computador el mundo fue revolucionado en un par de décadas, pero gracias a que el número de transistores por chip ha venido aumentando exponencialmente, los procesadores son cada más pequeños y eficientes. La reducción en el tamaño de los circuitos integrados, nos ha permitido contar con gran cantidad de potentes procesadores en carros, aviones, barcos, equipos médicos, teléfonos celulares, cámaras fotográficas, ascensores, ropa, juguetes, reproductores de audio y video entre otros. *Todos estos sistemas basados en un procesador y diseñados para cumplir un rango específico de funciones, se conocen con el nombre de Embedded Systems (ES).*

El diseño de los ES se ha mantenido concentrado en los países industrializados, esto debido en gran parte a los costos tecnológicos y económicos que se requieren para el diseño y producción de sistemas digitales complejos. Actualmente la tecnología de los FPGAs está ofreciendo una oportunidad para el diseño de este tipo de sistemas en países como Colombia, gracias a que estos dispositivos cuentan con millones de recursos lógicos y una estructura de interconexión programable eléctricamente que permite particulalizar un diseño en un FPGA por medio de un software que configura las interconexiones entre los recursos lógicos. La gran flexibilidad que ofrecen los FPGAs, ha permitido que esta tecnología pueda ser usada casi en cualquier dispositivo electrónico; más aún cuando sus proveedores, constantemente están optimizando cada uno de sus dispositivos, con el fin de ofrecer soluciones en una amplia gama de mercados y aplicaciones [4]. Actualmente encontramos FPGAs dentro de una amplia gama de dispositivos electrónicos, que abarcan las industrias del entretenimiento, las telecomunicaciones, la industria automotriz, los dispositivos médicos, la industria de la aviación, también encontramos FPGAs ofreciendo soluciones en la computación de alto rendimiento y reconfigurable [5]. Hoy es posible encontrar FPGAs hasta en Marte, pues los ingenieros del Jet Propulsion Laboratory incorporaron un FPGA de la empresa Xilinx, en la *Misión Rovers de exploración a Marte*, con el fin de aprovechar las posibilidades de reconfiguración del dispositivo [6].

Comparados con los ASICs (*Application specific integrated circuit*) los FPGAs pueden ofrecer desven-

tajas en cuanto a *área, velocidad* y *consumo de potencia*, sin embargo un diseño de una ASIC típicamente toma mucho más tiempo que la implementación de un diseño en un FPGA y el costo del diseño de una ASIC esta en el orden de los millones de dolares, mientras el costo del diseño en un FPGA esta en el orden cientos de dolares. De otro lado si se comparan con los procesadores los FPGAs son mucho más eficientes, pues permiten obtener mejores resultados en cuanto consumo de potencia y velocidad, pero el desarrollo la implementación de un algoritmo de manera eficiente en un FPGA es una tarea más difícil si se compara al trabajo de implementar un algoritmo en un procesador [7]. En este sentido durante esta investigación se detectó un vacío en la literatura en cuanto a estrategias o metodologías para el diseño los módulos HW de un ES.

El objetivo principal de esta tesis fue proponer una metodología que facilitara el diseño de un ES y su implementación en un FPGA. La metodología propuesta no optimizó todas la métricas del diseño digital (*area, potencia, velocidad y time to market*) pero si logró proponer un procedimiento para que el diseñador pueda aprovechar las ventajas de procesamiento en paralelo que ofrece el hardware con el propósito de ganar *velocidad* de procesamiento, también se utilizó un algoritmo que permitió racionalizar el uso de los recursos lógicos con el fin de optimizar el *área*. La metodología propuesta también permite realizar un diseño libre de los problemas relacionados con los *glitches* y la *metaestabilidad* con el fin de garantizar cierto grado de fiabilidad. Adicionalmente se plantearon una serie de laboratorios, en los cuales se proponen ejercicios de diseño que permitan explorar las posibilidades que ofrece una herramienta CAD comercial para el diseño de ES utilizando la metodología de codiseño.

Esta metodología se validó con el diseño de varios ES que se encuentran documentados en esta tesis y en los diferentes anexos, los cuales cumplieron a satisfacción los objetivos mencionados anteriormente. Finalmente es importante mencionar que la metodología propuesta fue utilizada por un estudiante de Maestría en Ingeniería Electrónica para rediseñar un procesador de propósito específico y los nuevos resultados fueron positivos.

Esta tesis está organizada de la siguiente manera: en el capitulo 2 se muestran las generalidades sobre el diseño de los ES y los FPGAs, también se describen algunos trabajos previos que sirvieron como punto de partida para esta investigación. En el capitulo 3 se describe el problema específico que se quiere atacar con esta tesis. En el capitulo 4 se describe el principal aporte de esta tesis de maestría, el cual es la propuesta metodológica para el diseño de los módulos de HW. Finalmente en el capitulo 5 corresponde a la validación de la metodología propuesta y en este se muestran los resultados obtenidos en cada uno de los diseños.

Capítulo 2

Antecedentes

2.1 Introducción:

En este capitulo se describen algunas generalidades que rodean tanto el diseño de los ES como la tecnología de los FPGAs, también se quiere mostrar algunos trabajos previos que sirvieron de fundamento para la propuesta metodológica que se hace en el capitulo 3.

2.2 Métricas principales en el diseño de ES

La construcción de los ES está fuertemente ligada a las tecnologías de fabricación de los Circuito Integrados (ICs) y la selección del tipo de tecnología está determinada por cinco factores principales :

- **Área:** Tamaño del IC o capa de cilicio necesaria para diseñar el ES.
- **Velocidad:** Tiempo requerido por el ES para realizar una función.
- **Consumo de potencia:** consumo de potencia del ES en una operación específica.
- **Costo:** el precio del IC.
- **Time to market:** tiempo que se requiere para terminar un IC, también se puede pensar como el tiempo necesario para poner una idea el mercado.

No existe una tecnología de fabricación de Circuitos Integrados que ofrezca beneficios en todos los cinco factores, sino que cada tecnología ofrece ventajas en algunos de los factores. Con el propósito de justificar el tipo de tecnología seleccionada en esta tesis de maestría (los FPGAs), se han agrupado las diferentes posibilidades de construcción de un IC en dos grandes grupos: *Tecnologías en Fábrica* y *Tecnologías en campo*.

Tecnologías en Fábrica:

Generalmente este tipo de tecnologías ofrecen los menores niveles de abstracción, es decir, a nivel de transistores y por consiguiente un mayor control sobre todas las variables del diseño. Estas tecnologías generalmente consisten en un proceso que se realiza por medio de capas (silicio dopado, polisilicio, dióxido de silicio y metal) [2], puestas unas encima de otras, sobre una lámina delgada de silicio, algunas de estas capas forman transistores y otras planos de conexión; de esta manera se logra interconectar gran cantidad de transistores y de esta manera se particulariza un Circuito Integrado de Aplicación Específica (ASIC). Con este tipo de tecnologías se logran fabricar transistores que están en el orden 0.1 micrómetros, permitiendo poner millones de ellos en menos de un cm^2. En este tipo de tecnologías también existen procesos con un mayor nivel de abstracción, por ejemplo a nivel de compuertas lógicas, lo que permite reducir de 10 a 15 capas en el nivel de transistor a de 3 a 5 en el nivel de compuerta.

Las Tecnologías en Fábrica ofrecen las mejores ventajas en cuanto a área, consumo de potencia y velocidad, pero lamentablemente el proceso de superponer capas es complejo y costoso (actualmente está en el orden de los millones de dolares [8]), esta ha sido una de las razones que ha mantenido el desarrollo digital avanzado en los países industrializados.

Tecnologías en Campo:

Las tecnologías en campo son también conocidas como tecnologías *non-ASIC*. Todos los dispositivos en este tipo de tecnología, cuentan con arreglo estándar de recursos lógicos (and, or, flip-flops, etc.) y una estructura de interconexión programable. Las interconexiones pueden ser programables una sola vez (OTP) por medio de fusibles que se funden por corriente o reprogramables eléctricamente usando transistores. En esta tecnología la particularización del diseño se realiza por medio de la configuración de las interconexiones programables. Este proceso se puede realizar con un económico dispositivo de programación, un software y un cable de conexión al PC. Las arquitecturas más populares en este tipo de tecnologías hoy en día son los CPLDs (*Complex Programmable Logic Device*) y los FPGAs (*Field Programmable Gate Array*).

En este tipo de tecnologías la particularización de un circuito no se realiza en una fábrica sino que se realiza en campo, lo cual ofrece grandes ventajas en cuanto al *costo*, y al *time to market*. Aunque el hecho de usar un arreglo estándar de recursos lógicos hace que los diseños presenten desventajas en cuanto a área, consumo de potencia y velocidad, estos se ven compensados fuertemente por su economía y rapidez a la hora de implementar físicamente un diseño en un FPGA o un CPLD.

Las Tecnologías en Campo, son cada vez son más utilizados en el diseño de ES, gracias a la facilidad de realizar cambios de hardware una y otra vez a través de programación y que permiten desarrollar prototipos de ES sin la necesidad de generar los costosos prototipos ASICs; esta es la principal razón por la cual se ha escogido este tipo de tecnología en este trabajo de investigación.

2.3 FPGAs vs ASICs

Aunque el diseño de ASICs utilizando el proceso litográfico ofrece ventajas en cuanto al *área* , *consumo de potencia* y *velocidad*, el diseño utilizando FPGAs ofrece las mejores ventajas en cuanto a *costos* (cuando el volumen de producción es bajo). Otro aspecto que ha impulsado el uso de los FPGAs, es la ventaja que estos ofrecen en cuanto al *time to market*, puesto que los fabricantes de dispositivos electrónicos constantemente se ven obligados a entregar las últimas tecnologías e innovaciones tan pronto como sea posible, con el fin de mantenerse a la vanguardia de un mercado altamente competitivo.

No obstante, aun cuando el diseño amerite un diseño ASIC, porque el volumen de producción lo requiere y el tiempo de desarrollo es asumible, actualmente los diseñadores digitales primero lo implementarán en un FPGA a manera de prototipo.

En este momento existe un interés creciente por el desarrollo de ES que puedan ser implementados sobre FPGAs, muestra de lo anterior es el creciente número de grupos de investigación dedicados a este tema (una lista de algunos de los grupos de investigación relacionados con este tema se puede revisar en [9]). Dentro de las problemáticas que se están abordando con esta tecnología se encuentran: análisis de secuencias genéticas, filtrado digital, criptografía, filtrado de paquetes de red, reconocimiento automático de objetos, migración sísmica, entre otros [10].

2.4 Codiseño:

Diseñar un ES utilizando únicamente software (SW), es decir, implementados sobre un Microcontrolador o Microprocesador de propósito general, tiene ventajas como:

- Mayor flexibilidad
- Facilidad para ejecutar órdenes secuencialmente
- Menor tiempo en el diseño

De otro lado implementar la misma solución en hardware (HW), es decir, en un ASIC o un FPGA, tiene las siguientes ventajas:

- Mayor velocidad (un procesador de propósito específico generalmente es más veloz que un procesador de propósito general).
- Menor consumo de potencia.
- Posibilidad de realizar tareas de manera concurrente.

La pregunta salta a la vista, ¿por qué no implementar ES híbridos, en donde las ventajas del software y el hardware se complementen?. Esta metodología de diseño híbrido se conoce con el nombre de Codiseño [11], en la que se busca aprovechar los puntos fuertes tanto del HW como del SW en un mismo ES. El Codiseño reduce la complejidad individual de los elementos de un ES.

Actualmente existen entornos de desarrollo que permiten realizar de manera integrada el desarrollo de un ES utilizando esta metodología. Algunas herramientas comerciales son: CoWare Codeveloper, Celoxica DK Codesign Suite, Xilinx Embedded Development Kit o Altera Nios II Integrated Development Environment [11].

Flujo de diseño en la metodología de Codiseño:

El flujo de diseño utilizando la metodología de codiseño es el que se muestra en la figura 2.1 y a continuación se describen brevemente cada una de sus etapas.

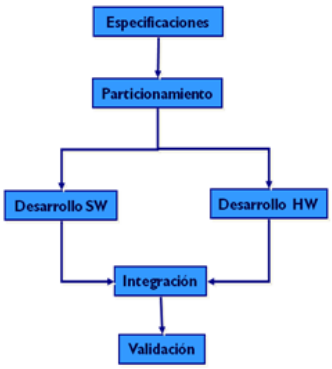


Figura 2.1: Flujo de diseño con la metodología de codiseño

Especificaciones del ES: El flujo de diseño se debe iniciar con una descripción funcional del ES a implementar en un nivel de abstracción mayor al que se va usar describir tanto el software como el hardware. Al finalizar esta etapa tendremos el ES dividido en varias funciones específicas, con el propósito de abordar todo el problema como pequeños problemas que se pueden atacar de manera independiente. Actualmente existen algunos lenguajes de programación (System C, System Verylog) que nos permiten describir un ES en una descripción de un alto nivel de abstracción y se podrían utilizar para hacer una primera simulación funcional del ES.

Particionamiento: el particionamiento es una tarea muy importante en el diseño de ES, pues en esta etapa se determina cuáles tareas se implementan HW y cuáles en SW, las decisiones que se tomen en esta etapa influenciaran todo el desarrollo posterior del ES, pues en ella determina que tanto se aprovecharan

las ventajas que poseen tanto el HW como SW. Las tareas que serán implementadas en HW tienen el propósito de no ocupar al procesador en tareas cíclicas, un ejemplo típico de estas tareas es la generación de bases de tiempos, también se pueden tener parámetros como el costo computacional de algunas operaciones computacionales si el objetivo por ejemplo es ganar velocidad de procesamiento.

Desarrollo HW y SW: Una vez que se ha realizado la partición HW/SW, se conocen cada una de las tareas o funciones que será implementadas tanto SW como HW, el siguiente paso es la implementación de cada una de ellas.

Integración: una vez finalizadas la implementaciones de HW y SW, el siguiente paso es la integración de ambos componentes en un solo sistema. '

Verificación: una vez se ha terminado todo el ciclo de diseño, el paso final es comprobar el correcto funcionamiento del ES.

2.5 Posibilidades para la implementación de un ES

Al momento de diseñar un ES existen tres posibilidades:

- ***Componente HW y SW integrados en un dispositivo semiconductor (SoC):*** *En la actualidad existen muchas compañías que fabrican procesadores de 32 bits integrados a una gran variedad de periféricos, lo cual simplifica el diseño y reduce costos [12].* Lamentablemente esta opción es poco flexible, ya que el diseño está subordinado a los periféricos que posee el SoC, en este caso el diseño estará muy enfocado hacia el software ya que el hardware es fijo. Para etapas de diseño esta es la opción menos viable.

- ***Componente SW en un SoC y componente HW en una FPGA****: Cuando no existe en el mercado un SoC con la cantidad de periféricos requerida para una determinada aplicación, o con una funcionalidad específica, es necesario recurrir a la utilización de dispositivos comerciales que implementen dicha operación, en algunas ocasiones el periférico puede realizar funciones poco comunes y no se proporciona comercialmente, la solución es entonces, implementar estas funcionalidades en un FPGA* [12]. Esta opción opción ofrece una mayor flexibilidad que la opción anterior, pero cuenta con todas las limitantes que ofrezca el PBC en el cual se interconecten el procesador con el PPGA (frecuencia máxima de trabajo, número de interconxiones, etc). Adicionalmente existen fuertes restricciones en las herramientas CAD que se pueden utilizar ya que generalmente este tipo

de soluciones no son ofrecidas directamente por un fabricante que ofrezca herramientas de simulación y depuración.

- **Componente SW y HW en un FPGA:** La tercera opción se dá, cuando tanto el procesador como los módulos de hardware se encuentra dentro de un sólo FPGA; esta es la opción que ofrece mayor flexibilidad y por lo tanto es la más apropiada sobre todo en los procesos de diseño. Actualmente se encuentran dentro de los FPGAs dos tipo de procesadores: *hard-core*[1] y *soft-core*[2]. Las mejores ventajas los ofrece el primero de ellos ya que el procesador esta implementado dentro del FPGA y no utiliza recursos lógicos del FPGA, ofreciendo mejores posibilidades en cuanto a frecuencia máxima de trabajo. Los procesadores soft-core se implementan utilizando los recursos lógicos del FPGA, lo cual tiene a favor que el procesador es diseñado a la medida, pero utiliza los recursos lógicos del FPGA afectando las frecuencias máximas de trabajo. Dentro de las principales ventaja que ofrece esta última opción se destacan:
  - Mayor nivel de integridad de las señales de comunicación HW/SW.
  - Mejores posibilidades de interconexión entre los módulos de SW y HW.
  - En caso de procesadores soft-core se pueden utilizar varios procesadores en un mismo circuito programable.
  - Las herramientas CAD ofrecidas por el fabricante del FPGA (éstas se describen más adelante).

Sistemas de Desarrollo para la implementación de un ES

Actualmente existen varias empresas que desarrollan FPGAs, CPLDs y sistemas desarrollo especializados para las etapas de diseño de ES, los dos principales proveedores de FPGAs en el mundo son Xilinx y Altera. En esta tesis de maestría se utilizaron tres sistemas de desarrollo de la empresa Xilinx, teniendo en cuenta que esta última ofrece las mejores posibilidades en herramientas CAD (ver sección 2.7). Los tres sistemas de desarrollo utilizados en este trabajo investigativo son:

- Spartan-3E Starter Kit [13]: es un sistema de desarrollo que cuenta con el FPGA Spartan-3E (XC3S500E-4FG320C), el CPLD CoolRunner-II (XC2C64A-5VQ44C) y con tres diferentes tipos de memoria. Adicionalmente ofrece puertos *Ethernet 10/100 Phy* y *JTAG USB download* y varias interfaces de comunicación. Este sistema de desarrollo se muestra en la figura 2.2.
- Spartan-3A DSP 1800A [14]: es un sistema de desarrollo basado en el FPGA *XC3SD1800A-4FGG676C Spartan-3A DSP* , ofrece diferentes fuentes de reloj, tres memorias y diferentes interfaces comunicación. Este sistema de desarrollo se muestra en la figura 2.3.

[1] Es un tipo de procesador que viene implementado en el FPGA por el fabricante.
[2] Es un tipo de procesador que no se encuentra implementado físicamente dentro del FPGA, sino que se encuentra descrito un HDL y posteriormente se implementa utilizando los recursos lógicos de FPGA

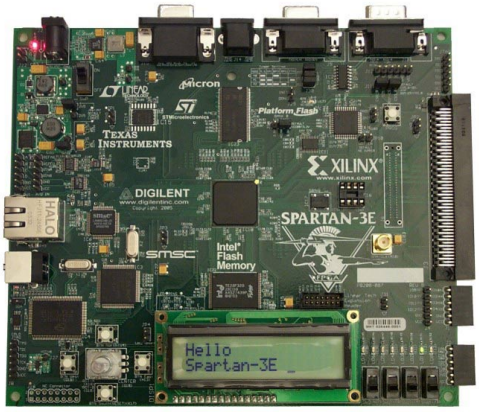


Figura 2.2: Spartan-3E Starter Kit

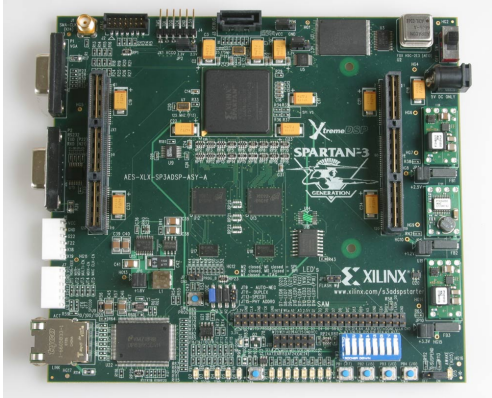


Figura 2.3: Spartan-3A DSP 1800A Edition

- Virtex 5 FX70T PowerPC [15]: este es un sistema de desarrollo basado en el FPGA XC5VFX70TFFG1136, el cual trae un procesador hard-core (Power PC [16]), memorias y diferentes posibilidades para comunicación. Es sistema de desarrollo se muestra en la figura 2.4.

2.6 Arquitectura general de un ES implementado sobre un FPGA:

Como se mencionó anteriormente la arquitectura de un ES consta tanto de una parte de HW como de SW, en la figura 2.5 se muestra un diagrama de bloques de la arquitectura general de los ES implementados sobre un FPGA.

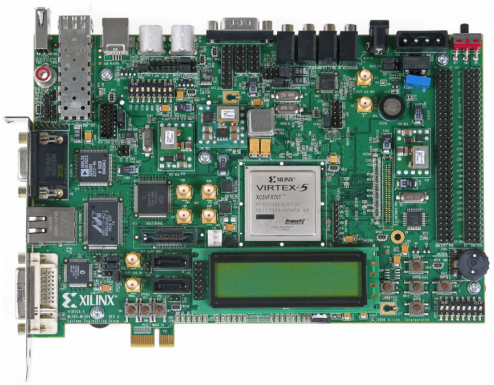


Figura 2.4: Virtex 5 FX70T PowerPC

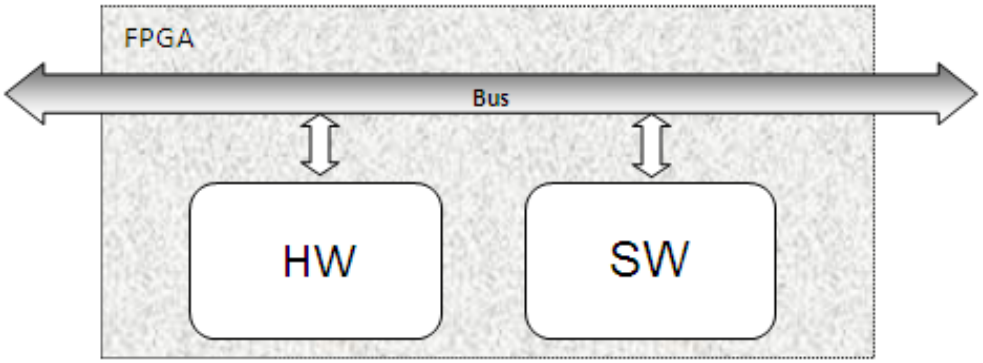


Figura 2.5: Arquitectura general de un ES

2.7 Diseño digital avanzado utilizando FPGAs

Actualmente los proveedores de FPGAs se están preocupando por optimizar constantemente sus dispositivos. Hoy en día se encuentran FPGAs que poseen millones de compuertas lógicas; además estos dispositivos incluyen interfaces programables para varios estándares de interface eléctrica y tienen bloques de funciones especiales incluidos entre la lógica programable, tales como memoria, multiplicadores, bloques DSPs y hasta CPUs completas.

Herramientas CAD para el diseño digital utilizando FPGAs

Actualmente los proveedores de FPGAs ofrecen herramientas CAD (*Computer Aided Design*) que permiten ampliar las posibilidades de diseño, simulación y depuración. Dentro de las herramientas que actualmente ofrecen los dos principales fabricantes de FPGAs en el mundo (Xilinx y Altera) se encuentran:

Por parte de Xilix:

- ISE [17]: esta herramienta permite hacer las descripciones en hardware utilizando diferentes HDLs (VHDL, Verilog y Abel). Adicionalmente realiza desde la síntesis hasta la implementación del diseño en el FPGA.
- ISE Simulator [18]: permite hacer simulaciones funcionales y en tiempo.
- ChipScope ILA [19]: con esta herramienta es posible capturar y analizar las señales internas del FPGA en tiempo real con el propósito de verificación.
- PlanAhead [20]: permite optimizar los diseños de hardware a partir de la mejor ubicación de cada uno de los módulos dentro del FPGA.
- System Generator for DSP [21]: con esta herramienta es posible diseñar sistemas DSP a partir del entorno Symulink de Matlab con el fin de implementarlos en un FPGA.
- Platform Studio and EDK [22]: es un ambiente de trabajo que permite realizar diseños integrados de HW y SW. El hardware se implementa en los recursos lógicos del FPGA y para la parte de SW se pueden utilizar los procesadores MicroBlaze [23] y PowerPC [16].

Por parte de Altera:

- Quartus II [24]: esta es la herramienta de descripción de hardware equivalente a ISE de Xilinx.
- Modelsim-Altera [25]: esta es la herramienta de simulación equivalente a ISE Simulator de Xilinx.
- DSP Builder [26]: es la equivalente al System Generator for DSP de Xilinx.

De acuerdo a las descripciones anteriores se puede observar que las mejores posibilidades en cuanto a herramientas CAD las ofrece Xilinx.

2.8 Algunas propuestas para el diseño digital avanzado sobre FPGAs:

Algunas veces se cree de manera incorrecta que el diseño de ES implementados sobre FPGAs, utilizando un Lenguaje de Descripción de Hardware (HDL) como VHDL o Verilog, consiste solamente en escribir, sin errores sintácticos un código que describa el circuito; pero el diseño digital utilizando un HDL, es un proceso complejo que va más allá de la descripción de un código libre de errores [2].

En [27] se describen diferentes estrategias que permiten optimizar un diseño con respecto a una métrica específica (área, velocidad o consumo de potencia), estas estrategias facilitan la optimización de un diseño en hardware existente, sin embargo no se describe un procedimiento que facilite el diseño inicial, en otras

palabras, no se especifica una estrategia para la transformación del algoritmo secuencial a una descripción de hardware.

En [2] se propone una metodología para transformar un algoritmo secuencial en una implementación FSMD (*Finite State Machine with Datapath*) utilizando un HDL. Aunque esta metodología facilita la transformación del algoritmo en un hardware específico, lastimosamente este enfoque no busca ningún optimizar ninguna métrica específica, adicionalmente sus resultados pueden llegar a no ser fiables debido a problemas relacionados con la *metaestabilidad* y los *glitches*.

En [28] se propone una metodología basada en FSMs (*Finite State Machine*) que resuelve los dos problemas mencionados anteriormente, por medio de la eliminación de la lógica combinacional a la salida de la FSM (para evitar los glitches) y la inclusión de circuitos de sincronismo (para las señales asíncronas) que reducen la posibilidad de *metaestabilidad* en el procesador, sin embargo esta metodología puede llegar a ser inviable para algunos casos, en donde el uso de un hardware externo a la FSM (*Datapath*) puede facilitar el trabajo de diseño.

En [29] se proponen algunos algoritmos que permiten (a partir del algoritmo secuencial) agrupar varias variables en un mismo registro y de igual manera realizar varias operaciones en una misma unidad funcional. Estos dos algoritmos anteriores se pueden utilizar para el diseño de módulos HW con el fin de racionalizar el uso de los recursos lógicos.

2.9 Lenguajes de Descripción de Hardware (HDLs)

Desde el año 1952 cuando Irving Reed creo el primer Lenguaje de Descripción de Hardware, el número de lenguajes que permiten describir un circuito ha crecido grandemente, de igual manera y con el aumento en la complejidad de los circuitos digitales, los lenguajes de descripción hardware también han evolucionado en las posibilidades que ofrecen a los diseñadores.

Los primeros lenguajes de descripción de hardware, permitían describir un circuito a partir de sus componentes y sus conexiones, razón por la cual son llamados lista de conexiones (net-list), pues en esencia el diseñador describe los componentes del circuito y sus conexiones; este tipo de lenguajes implica que el trabajo de diseño sea altamente manual. En la década de los 80 surgieron los Lenguajes de Descripción de Hardware se encuentran: VHDL, Verilog, System C, IDL, TI - HDL, ZEUS y UDI/L (que se usa exclusivamente en Japón), siendo los dos primeros los más ampliamente usados.

Estos lenguajes permitieron describir un circuito no desde su estructura (componentes y conexiones), sino desde un punto de vista funcional, este enfoque de los HLDs, permitió un mayor grado de abstracción

para el diseñador, permitiendo que este pueda prestar más atención al funcionamiento del circuito que a su estructura, lo cual ofrece grandes ventajas a la hora de diseñar circuitos de alta complejidad.

VHDL

Este lenguaje de descripción de hardware fue desarrollado por el Departamento de Defensa de los Estados Unidos a finales de los años 70, con el propósito de diseñar y simular circuitos complejos, de tal manera que un diseño realizado por una empresa pudiera ser entendido por otra.

VHDL fue enviado a la IEEE que lo convirtió en el estándar IEEE 1076 - 1987; en 1993 el estándar se actualizó como el estándar IEEE 1164 - 1993 y posteriormente en 1996 el estándar IEEE 1076.3 - 1996 se convirtió en un estandar HDL para síntesis, siendo este último y el 1164 los estándares más utilizados en síntesis de circuitos digitales.

VHDL es un lenguaje con una sintaxis amplia y flexible que permite la descripción de un circuito utilizando tres estilos de programación:

Descripción de comportamiento: en este estilo describe el comportamiento del circuito, este tipo de descripción es la que más se parece a los lenguajes convencionales ya que las sentencias son secuenciales, estas sentencias secuenciales se encuentran dentro de bloques de proceso, estos procesos son ejecutados en paralelo con asignaciones concurrentes de señales y con las instancias a otros componentes.

Descripción estructural: esta descripción instancia diferentes componentes jerarquizados y los interconecta entre si, con el propósito de construir un diseño de jerarquía superior. Este estilo de programación puede asemejarse mucho a un net-list, y generalmente se usa para interconectar bloques IP, tanto soft-core como hard-core.

Descripción de flujo de datos: este tipo de descripción se encuentra a mitad de camino entre una de comportamiento y una estructural, pues aunque se trata de una especie de net-list, las interconexiones no se hace entre componentes sino sobre elementos abstractos del lenguaje.

La descripción de un ES en VHDL generalmente terminará siendo mixta, es decir, una combinación de los estilos previamente descritos.

VHDL fue diseñado con base en los principios de la programación estructurada, con la idea de definir la interfaz de un modulo de hardware, mientras se deja invisible sus detalles internos. La entidad (ENTITY) en VHDL es simplemente la declaración de las entradas y salidas de un modulo mientras que la arquitectura (ARCHITECTURE) es la descripción detallada de la estructura interna del modulo o de su comportamiento, en otras palabras, la entidad como una funda de la arquitectura dejando invisible

los detalles de lo que hay dentro de la arquitectura. De esta manera se forma la base de un sistema de diseño jerárquico, la arquitectura de la entidad de mas nivel (top level) puede usar otras entidades dejando invisible los detalles de la arquitectura de la identidad de menos nivel. Actualmente VHDL es uno de los estándares más utilizados por las herramientas de síntesis de circuitos digitales.

Verilog

Verilog fue inventado por Phil Moorby en 1985 mientras trabajaba en Automated Integrated Design Systems, más tarde llamada Gateway Design Automation. Moorby tenía en mente diseñar un lenguaje de descripción de hardware con una sintaxis similar a la del lenguaje de programación C, con el objetivo que resultara familiar para los ingenieros y así fuese aceptado rápidamente. Gateway Design Automation fue comprada por Cadence Design Systems en 1990. Cadence ahora tiene todos los derechos sobre los simuladores lógicos de Verilog y Verilog-XL hechos por Gateway. Con el incremento en el éxito de VHDL, Cadence decidió hacer el lenguaje abierto y disponible para estandarización. Cadence transfirió Verilog al dominio público a través de Open Verilog International, actualmente conocida como Accellera. Verilog fue después enviado a la IEEE que lo convirtió en el estándar IEEE 1364-1995, habitualmente referido como Verilog 95. Los usuarios encontraron algunas deficiencias en el lenguaje las cuales fueron mejoradas y se convirtieron en el estándar IEEE 1364-2001 conocido como Verilog 2001.

El lenguaje tiene un preprocesador como C, y la mayoría de palabras reservadas de control como "if", "while", etc, son similares. A diferencia del lenguaje C y otros lenguajes convencionales de programación, en donde la ejecución de sentencias es estrictamente secuencial este lenguaje de programación las ejecuciones de las sentencias no son estrictamente lineales. Otro factor que hace radicalmente diferente a este tipo de programación es el tiempo el cual no es tenido en cuenta en otros programas como C.

Un diseño en Verilog consiste de una jerarquía de módulos, los cuáles son definidos con conjuntos de puertos de entrada, salida y bidireccionales. Internamente un módulo contiene una lista de cables y registros. Las sentencias concurrentes y secuenciales definen el comportamiento del módulo, describiendo las relaciones entre los puertos, cables y registros. Las sentencias secuenciales son colocadas dentro de un bloque begin/end y ejecutadas en orden secuencial, pero todas las sentencias concurrentes y todos los bloques begin/end son ejecutadas en paralelo en el diseño. En el apéndice C, se introduce un tutorial acerca de este lenguaje.

Capítulo 3

Desarrollo del trabajo investigativo

3.1 Introducción

Para hacer el diseño de los ES, se seleccionó la tercera posibilidad de las tres descritas en la sección 2.5, es decir, utilizar procesadores tipo *soft-core* y *hard-core* para los módulos de SW del ES y los recursos lógicos del FPGA para el diseño de los módulos de HW, implementando todo el ES en un mismo FPGA, lo anterior por las ventajas descritas en las secciones 2.5 y 2.7.

3.2 Apropiación Tecnologica del Diseño de ES implementados sobre FPGAs

La *Apropiación Tecnológica* que se desarrolló en esta tesis de Maestría se encuentra descrita en detalle en el Anexo 1, el cual es el libro titulado: *Diseño de Embedded Systems Impelementados sobre FPGAs*, en este se describe todo el proceso para el diseño de un ES implementado sobre un FPGA o un CPLD utilizando el enfoque anteriormente mencionado.

Adicionalmente en el anexo 2 se proponen una serie de laboratorios con la herramienta EDK de Xilinx (ver sección 2.7) cuyo propósito es mostrar gradualmente las posibilidades que ofrece esta herramienta CAD para el diseño de ES utilizando la metodología de codiseño implementado sobre un FPGA. A continuación se hace un resumen de cada una de ellas.

Práctica 1: Introducción al EDK y MicroBlaze

Objetivo General: Describir los principales pasos para el diseño de un sistema digital con la herramienta EDK, incorporando el procesador MicroBlaze.

Objetivos Específicos:

- Instalar el software requerido para el diseño de ES suministrado por Xilinx para sus diversas plataformas. Implementar un proyecto sencillo que permita reconocer y profundizar el manejo de la herramienta de software EDK (Xilinx Platform Studio).

- Construir un programa en lenguaje C que permita apagar y encender secuencialmente (Auto fantástico) los LED´s de la Plataforma SPARTAN 3A DSP.
- Programar la Plataforma de desarrollo SPARTAN 3ª DSP 1800, mediante el cable USB-JTA.

Práctica 2: Adición y uso de periféricos locales al Microblaze

Objetivo General: Usar los recursos internos con lo que cuenta la plataforma en el diseño de aplicaciones sencillas.

Objetivos Específicos:

- Seleccionar los IP Core (Periféricos) necesarios para la creación de tres proyectos a realizar, utilizando LED´s, interruptores y Pulsadores.
- Adicionar los periféricos locales al PLB (Processor Local Bus), de la plataforma Spartan 3A DSP.
- Escribir los códigos en lenguaje C para la implementación de los proyectos con los periféricos internos de la plataforma.

Práctica 3: Uso de periféricos externos I

Objetivo General: Crear mediante el asistente de creación de periféricos que proporciona la herramienta EDK, un modulo IP para el control de un periférico externo. **Objetivos Específicos:**

- Realizar e implementar un diseño en la plataforma que permita el control del modulo Digilent PmodSSD (Display 7 Segmentos Doble).
- Describir en VHDL un módulo que permita hacer la conversión BCD-7 segmentos, e introducirlo al Driver generado

Práctica 3: Uso de periféricos externos II

Objetivo General: Crear un sistema que reciba una señal analógica, usando el ADC digitalice esta misma y se visualice en los LED's de la plataforma. Además el sistema debe contener la opción de almacenar un rango de datos para su posterior trasmisión a un computador, esto controlado por medio de Pulsadores. **Objetivos Específicos:**

- Diseñar un modulo en VHDL que controle el modulo ADC para la adquisición de datos.
- Diseñar en EDK un proyecto que permita la adquisición de los datos, almacenaje, control y trasmisión de los mismos

Uso de periféricos desde el procesador a través de un sistema operativo

Objetivo General: Crear un sistema en donde se aprecie el trabajo multihilo (Multithreading) del microkernel proporcionado por Xilinx (Xilkernel), operando con el procesador MicroBlaze. **Objetivos Específicos:**

- Diseñar un proyecto en EDK que permita visualizar en el Hyperterminal, por medio del puerto serial (RS232), la ejecución de varios hilos de impresión.
- Configurar el Xilkernel para operar varios hilos de ejecución (Thread's).
- Escribir el código en C para controlar la ejecución del Xilkernel y la impresión de los thread's mediante comunicación serial.

3.3 Propuesta metodológica para el diseño de los módulos HW

Uno de los aportes más importantes de esta tesis de maestría es una propuesta metodológica para el diseño de los módulos de HW de un ES, la cual se resume en esta sección y se puede revisar en [30]. La metodología no pretende optimizar todas la métricas que se describen en la sección 2.2, sin embargo, el uso de ésta si le permite al ingeniero realizar un diseño *balanceado*, en cuanto a *velocidad*, *área* y *time to market*. Adicionalmente la metodología permite realizar un diseño con un buen grado de *fiabilidad* ya que trata con los problemas relacionados con los *glitches*[1] y la *metaestabilidad*[2], las principales fuentes de comportamientos indeseados en cualquier diseño digital.

Diseño Digital en el Nivel de Transferencia entre Registros:

Un diseño digital comienza con la definición de un objetivo a cumplir, el cual generalmente se encuentra descrito por medio de un algoritmo, es decir, una serie de tareas específicas que el diseño debe realizar en el tiempo. La implementación en hardware de un algoritmo secuencial tienen tres requerimientos: las variables, las unidades funcionales (para realizar las operaciones requeridas por el algoritmo) y la secuenciación en el tiempo de dichas operaciones. Un nivel de abstracción que ha resultado ser apropiado para representar un algoritmo secuencial es el Nivel de Transferencia de Registros (RTL por sus sigla en inglés), pues cuenta con registros para almacenar valores y así imitar las variables en un algoritmo, unidades funcionales (sumadores, restadores, multiplicadores) para realizar las operaciones necesarias y

[1] Los glitches son un problema relacionado con los tiempos de respuesta de los Flip Flops, generando salidas lógicas *transitorias* e indeseadas en cualquier diseño digital

[2] La *metaesbilidad* se presenta cuando se violan los tiempo de *setup* y *hold* en un flip flop, ocasionando comportamientos aleatorios en el diseño digital. Este tipo de problema está directamente relacionado con las señales asíncronas del diseño

un control que permite hacer la secuenciación en el tiempo requerida por el diseño, simulando así las iteraciones y los saltos de un algoritmo secuencial.

Operación básica en la Transferencia entre Registros (*RT*):

La operación básica en RTL es la operación de transferencia de registro (RT), la cual se representa por medio de la siguiente notación:

$$R_{destino} = f\left(R_{1fuente}, R_{2fuente}, \ldots R_{nfuente}\right)$$

En esta representación el registro de la izquierda es el registro destino y los registros de la derecha son los registros fuente y $f(.)$ representa la operación que se va a desarrollar. Los registros simulan las variables, las operaciones se implementan por medio de unidades funcionales como sumadores, restadores, etc. y la secuenciación de las operaciones se realizan por medio de un control. La principal diferencia entre una variable de un algoritmo y un registro es que cada operación de escritura está regida por un flanco de reloj. Algunos ejemplos de operaciones y sus representaciones se describen a continuación:

- $R_1 \leftarrow 1$: La constante 1 es almacena en el registro R_1, en el siguiente flanco de reloj.
- $R_1 \leftarrow R_2$: El contenido actual (es decir el valor que tiene desde el último flanco de reloj) del registro R_2 , será almacenado en el registro R_1 en el siguiente flanco de reloj.
- $R_1 \leftarrow R_1 - 1$:El contenido actual del registro R_1 , se decrementa en 1 y el resultado será almacenado en el mismo registro en el siguiente flanco de reloj.
- $R_1 \leftarrow R_1$ *and* R_2: El resultado de la operación *and* entre el contenido actual de R_1 y R_2 , será almacenado en el registro R_1, en el siguiente flanco de reloj.
- $R_1 \leftarrow R_1 + R_2$: El contenido actual del registro R_1 se suma con el contenido actual R_2 y se almacenará en el registro R_1 en el siguiente ciclo de reloj.

Una descripción más detallada de las operaciones RT:

De acuerdo a las descripciones de las operaciones RT, descritas anteriormente, se debe tener en cuenta que:

- Los nuevos datos en los registros destino, estarán disponibles en el siguiente flanco de reloj más el *tiempo de retención* (t_{hold}) de los dispositivos lógicos.
- Para realizar las operaciones determinadas por $f()$ los valores en los registros fuente deben estar estables un tiempo mayor al *tiempo de establecimiento* (t_{setup}) de los dispositivos lógicos.

Para dar una descripción más aproximada de las operaciones RTL, puede utilizar los sufijos act (actual) y sig (siquiente), de tal forma que, la operación:

$$R_1 \leftarrow R_1 + R_2$$

Se puede ver como:

1. $R_{1sig} \leftarrow R_{1act} + R_{2act}$
2. $R_{1act} \leftarrow R_{1sig}$ en el siguiente flanco de reloj

Implementación de una operación RT en HW

La implementación en hardware de una operación RT es bastante directa, a continuación se muestran algunos ejemplos de dichas implementaciones. Las figuras 3.1 y 3.2 muestran la implementación en hardware de las operaciones $R_1 \leftarrow R_1 + 1$ y $R_1 \leftarrow R_1 + R_1$ respectivamente.

Figura 3.1: (a)Implementación en HW de $R_1 \leftarrow R_1 + 1$ (b)Diagrama de tiempo

Implementación de múltiples operaciones RT en hardware

Un algoritmo cuenta con múltiples operaciones y una secuenciación específica, para ver cómo se pueden implementar múltiples operaciones en hardware, se tomará como ejemplo el siguiente algoritmo:

1. $R_1 \leftarrow R_2$
2. $R_1 \leftarrow R_1 + 1$
3. $R_1 \leftarrow R_1 + R_2$
4. $R_1 \leftarrow R_1 \times R_2$

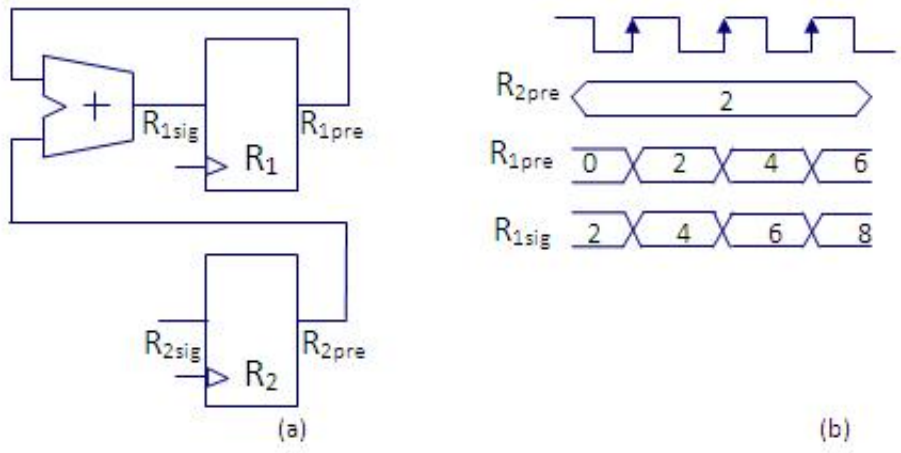


Figura 3.2: (a)Implementación en HW de $R_1 \leftarrow R_1 + R_2$ (b)Diagrama de tiempo

Por el hecho de que existan varias posibilidades para operar los registros, se necesita un multiplexor o un bus con el fin de direccionar los datos. La implementación en hardware del algoritmo anterior se muestra en la figura 3.3. La señal de control, permite hacer la secuenciación de los diferentes datos que se escriben en R_1. Es importante resaltar que la implementación de un algoritmo en hardware no es única, existen muchas posibilidades a la hora de implementar un mismo algoritmo en hardware. En la figura 3.3, se incluye la *señal de control* que permite hacer la secuencia de las operaciones.

Implementaciones FSMD:

Todos los recursos en hardware necesarios como los registros, las unidades funcionales y los multiplexores se conocen con el nombre de Camino de Datos o *Datapath*. El circuito encargado de generar las señales de control apropiadas para el Datapath, se conoce con el nombre de Control (del Datapath). En el ejemplo anterior el Datapath es el mostrado en la figura 3.3 y el Control deberá generar la *señal de control* apropiada para que se realicen las operaciones en el orden requerido.

En resumen, para implementar un algoritmo en hardware en el Nivel de Transferencia entre Registros, se requiere de dos grandes bloques, el primero de ellos es el *Datapath* que básicamente se encarga de ofrecer todo el hardware necesario (registros, sumadores, multiplexores, etc.) y el Control, el cual permite secuenciar las operaciones.

El Datapath:

El propósito del Datapath de ofrecer el hardware necesario para que el diseño digital desarrolle una tarea específica y sus componentes se pueden concentrar en tres grupos principalmente:

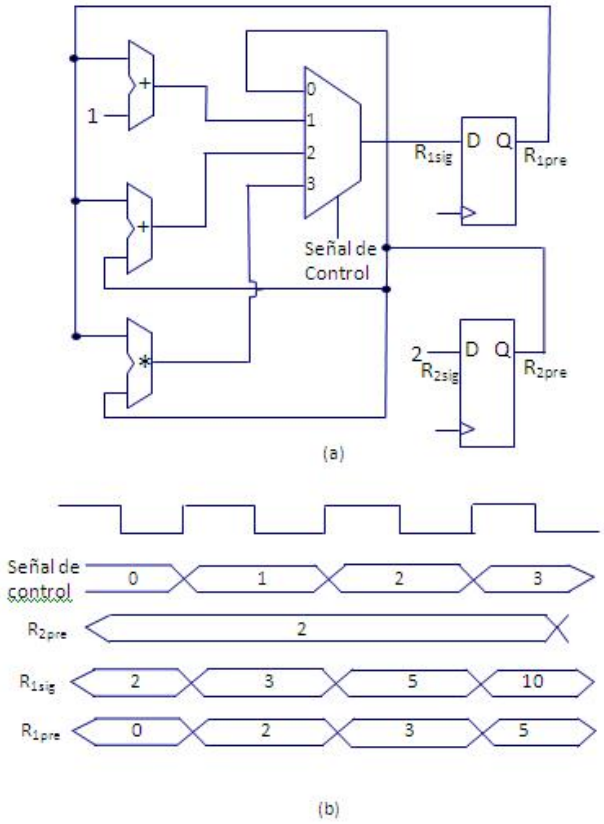


Figura 3.3: Implementación de múltiples operaciones RT

- Unidades de almacenamiento: son los registros y demás dispositivos de memoria para el almacenamiento temporal de los datos.

- Unidades funcionales: son la encargadas del procesamiento de datos, es decir, las que se encargan de realizar las operaciones entre los datos, entre las principales unidades funcionales se encuentran sumadores, multiplicadores, comparadores, etc.

- Unidades de interconexión: son los elementos que permiten interconectar las diferentes unidades en el Datapath y también permiten el intercambio de información con el *mundo exterior*. Los principales unidades de interconexión son los multiplexores y los buses.

La unidad de control (FSM):

Como un algoritmo es una secuencia de operaciones, se requiere de un control que permita hacer la secuenciación apropiada de las operaciones que se realizan dentro del Datapath, simulando así los saltos y las iteraciones en un algoritmo. Esta unidad de control se puede implementar por medio de una máquina

de estados finitos (FSM, por su nombre en inglés), *en donde cada estado corresponde a una iteración en el algoritmo*, es decir, en cada estado, la unidad de control genera las señales de control que llegan al Datapath para que este pueda realizar una operación determinada. De manera general una FSM consiste de tres partes:

- *Registro de estado* que almacena el estado actual.
- *Lógica de salida* que activa determinadas señales de salida de acuerdo al estado actual.
- *Lógica del estado siguiente* que determina el nuevo estado a partir del estado actual.

La implementación de un algoritmo mediante un Datapath y un Control se conoce con el nombre FSMD por sus siglas en inglés (Finite State Machine with Datapath). Generalmente los Circuitos Integrados de Aplicación Específica (ASIC) y los Procesadores comerciales, constan al menos de un Datapath y de un Control, aunque muchos ASICs y Procesadores pueden incluir varios Datapaths y varios Controles. En la figura 3.4 se muestra el diagrama de bloques de una implementación FSMD.

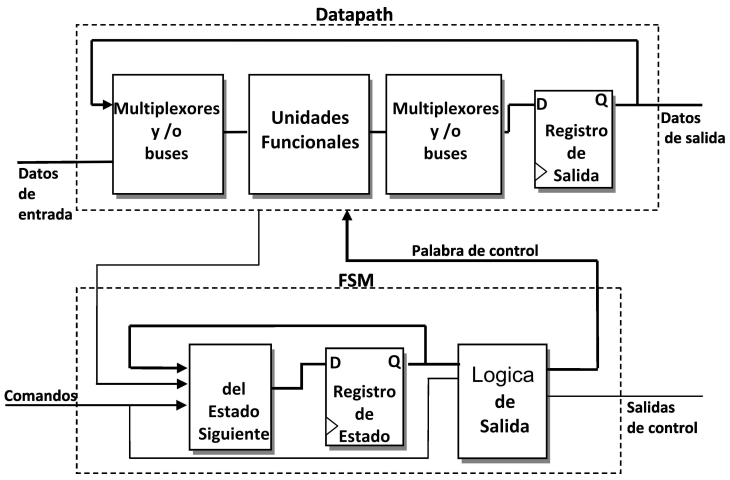


Figura 3.4: Diagrama de Bloques de una implementación FSMD

En la figura 3.4, se pueden observar las señales que generalmente presenta un Datapath:

- *Datos de entrada*: son los datos provenientes del exterior para ser procesados por la FSMD.
- *Datos de salida*: son los datos ya procesados por la FSMD.
- *Señales de control*: son señales de entrada las cuales especifican cuál operación RT debe ser realizada. Estas señales de control son generadas por la FSM.

- *Señales de estado*: son señales de salida que indican ciertas condiciones en el Datapath, como por ejemplo si una operación determinada fue cero. Estas señales son usadas por el FSM para determinar futuros estados.

Como se mencionó anteriormente el *Control*, es una máquina de estados finitos; por lo tanto, presenta un *Circuito de estado siguiente*, el *registro de estado* y *la lógica de salida*. Generalmente una FSM incluye las siguientes señales:

- **Comandos**: son señales de entrada a la FSM, que le permiten tomar decisiones, como por ejemplo el inicio de una operación.
- **Señales de Estado**: son señales entrada, que provienen desde el Datapath, las cuales son usadas junto con las señales de comando para determinar el siguiente estado.
- **Señales de control**: son señales de salida de la FSM y con usadas para controlar la operaciones que se realizan en el Datapath.
- **Salidas de control**: son señales de salida de la FSM y que permiten identificar el estado de la FSMD como por ejemplo ocupado, realizado, etc.

Diagramas ASM:

Un paso intermedio entre un algoritmo secuencial y una implementación FSMD, es la realización de un *Diagrama de Máquina de Estados Algorítmica* o un Diagrama ASM, el cual permite de manera gráfica de describir con cierto nivel de detalle una implementación FSMD.

Para la representación de una FSMD los diagramas ASM utilizan cuatro componentes básicos, como se observa en la figura 3.5:

- **Caja de estado:** contiene el conjunto de asignaciones incondicionales a las variables y puertos de salida del Datapath. La caja de estado contiene el nombre del estado y se sitúa en la parte superior izquierda de la caja.
- **Caja de decisión:** describe las condiciones sobre las cuales la FSMD hará asignaciones condicionales a las variables o a los puertos y también permite decidir el estado siguiente. Estas decisiones se toman a partir de los comandos y las señales de estado. Cada caja de decisión tiene dos salidas posibles, una se toma cuando la condición indicada en la caja es cierta (1) y la otra cuando es falsa (0).
- **Caja de salida condicional:** esta permite hacer asignaciones condicionales a las variables o a los puertos, es decir se sitúan después de una caja de decisión.

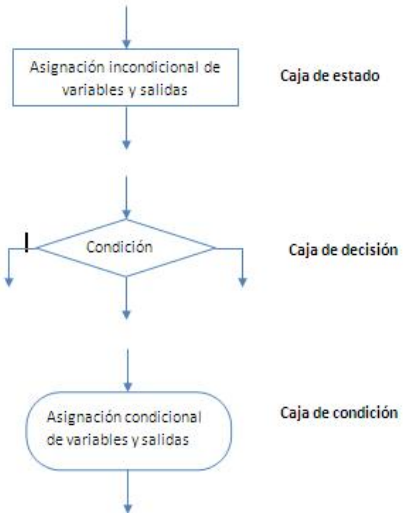


Figura 3.5: Componentes básicos de un Diagrama ASM

Un **Bloque ASM** es la estructura que agrupa los tres elementos anteriores que pertenecen a un mismo estado, organizándolas en forma serial y/o en paralelo. Un bloque ASM sólo tiene una entrada aunque puede tener cualquier número de salidas y se representa por medio de una línea discontinua, en la figura 3.6 se muestra un ejemplo de un bloque ASM.

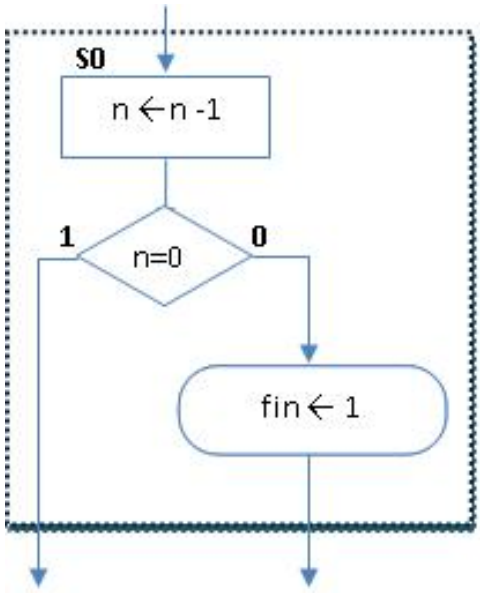


Figura 3.6: Bloque ASM

Un Diagrama ASM, consiste de uno o más bloques ASM interconectados, sin que se viole ninguna de las dos reglas siguientes:

1. El diagrama debe definir un único estado siguiente para cada estado presente y un conjunto específico de condiciones.

2. Todo camino definido por la red de cajas de condición debe conducir a una caja de estado.

En la figura 3.7 se muestra un diagrama ASM que viola la regla número 1 puesto si *cond1* es falsa y *cond2* es verdadera, tanto S1 como S2 están especificados como estados siguientes. En el diagrama ASM de la figura 3.8 se viola la regla número 2 puesto que si *cond1* es falsa y *cond2* es falsa, se forma un bucle que no lleva a ningún otro estado.

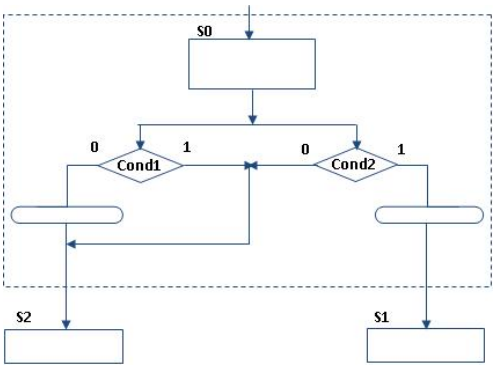


Figura 3.7: Diagrama ASM incorrecto (1)

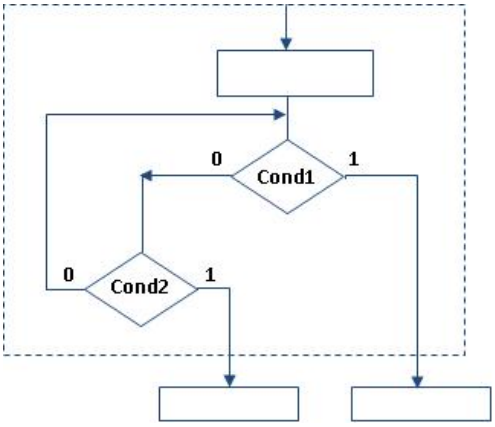


Figura 3.8: Diagrama ASM incorrecto (2)

En un diagrama ASM se utilizan los registros para imitar las variables; las cajas de decisión para implementar los saltos condicionales dentro del algoritmo y las operaciones RT para implementar las instrucciones.

Una diferencia importante de los diagramas ASM y los diagrama de flujo convencionales, es que *las operaciones en el diagrama ASM, están regidas por la señal de reloj.* Otra diferencia importante es que un Diagrama ASM es posible realizar *instrucciones en paralelo*, por ejemplo, si dos asignaciones como $R_1 \leftarrow R_1 - 1$ y $R_2 \leftarrow R_2 + R_3$ se encuentran en el mismo estado, las dos serán realizadas en el mismo ciclo de reloj, para ello deben existir tanto un sumador como un decrementador físico en el circuito y se requiere que entre las operaciones realizadas no exista dependencia de datos. De manera general, *para programar varias operaciones en un mismo estado*, (para que sean realizadas en el mismo ciclo de reloj), *se necesita que entre las instrucciones no exista dependencia de datos y se cuente con el hardware necesario.*

3.4 Flujo de diseño

El objetivo de este esta propuesta metodologica, es facilitar el diseño de procesadores que cumplan una función específica, mediante una implementación FSMD descrita en VHDL; para tal fin se plantea el siguiente flujo de diseño:

1. **Especificar la entidad del procesador y su algoritmo :** una vez se conoce el objetivo que debe cumplir el módulo, lo primero que se hará es especificar tanto las entradas como salidas que necesita el módulo para funcionar correctamente; en segundo lugar se debe partir de un algoritmo que muestre cada una de las tareas y el orden en el que se deben realizar para cumplir con el objetivo deseado.

2. **Diseño del Diagrama ASM:** como ya mencionó anteriormente un paso intermedio entre un algoritmo y una implementación FSMD, es un Diagrama ASM.

3. **Diseño del Datapath y la FSM:** el siguiente paso será convertir el Diagrama ASM en un Datapath y una FSM. Para el diseño del *Datapath* se proponen los siguientes pasos.

   - Agrupación de las variables en un número específico de registros.
   - Agrupación de todas las operaciones en un número específico de unidades funcionales.
   - Agrupación de las operaciones RT de acuerdo a su registro destino.
   - Construcción del hardware.
   - Adición del hardware requerido para evitar *meta-estabilidad.*

4. **Descripción en VHDL:** En este paso describe la implementación FSMD utilizando HDL como Verilog o VHDL.

5. **Implementación del ES en el FPGA:** Finalmente se implementa el diseño en FPGA seleccionado.

3.5 Ejemplo de Diseño

Con el propósito de facilitar la comprensión de la metodología se propone el siguiente el diseño:

`Diseñar un procesador de propósito específico que permita calcular el n-esimo término de una serie, en donde cada término es la suma de los dos términos anteriores. El diseño recibirá tres datos: los dos primeros términos de la serie y el número del término a hallar.`

Paso1: Definiendo la entidad del procesador y algoritmo a realizar

El primer paso para el diseño del procesador es definir tanto la entidad del procesador así como también el algoritmo que éste debe realizar.

La serie de números que se desea calcular, es aquella en la que un término es la suma de los dos anteriores, es decir:

$$a_i = a_{i-1} + a_{i-2}$$

Por ejemplo si los datos de entrada son: 3, 5, 6 entonces el diseño deberá calcular el sexto término de la serie a partir de los dos primeros (3 y 5):

- 3, 5, 8, 11, 19, **30**

En la figura 3.9 se muestra la entidad del procesador, en donde *enta* y *entb* son las entradas para los dos primeros términos de la serie, *entn* es la entrada para indicarle al procesador qué término debe calcular, *sal* es la salida en donde se muestra el término hallado y las señales *inicio* y *ocupado* hacen parte del protocolo *start-finish* el cual se expondrá más adelante. En la figura 3.10 se muestra un algoritmo secuencial que permite calcular el n-esimo término de esta serie.

Figura 3.9: Entidad del procesador

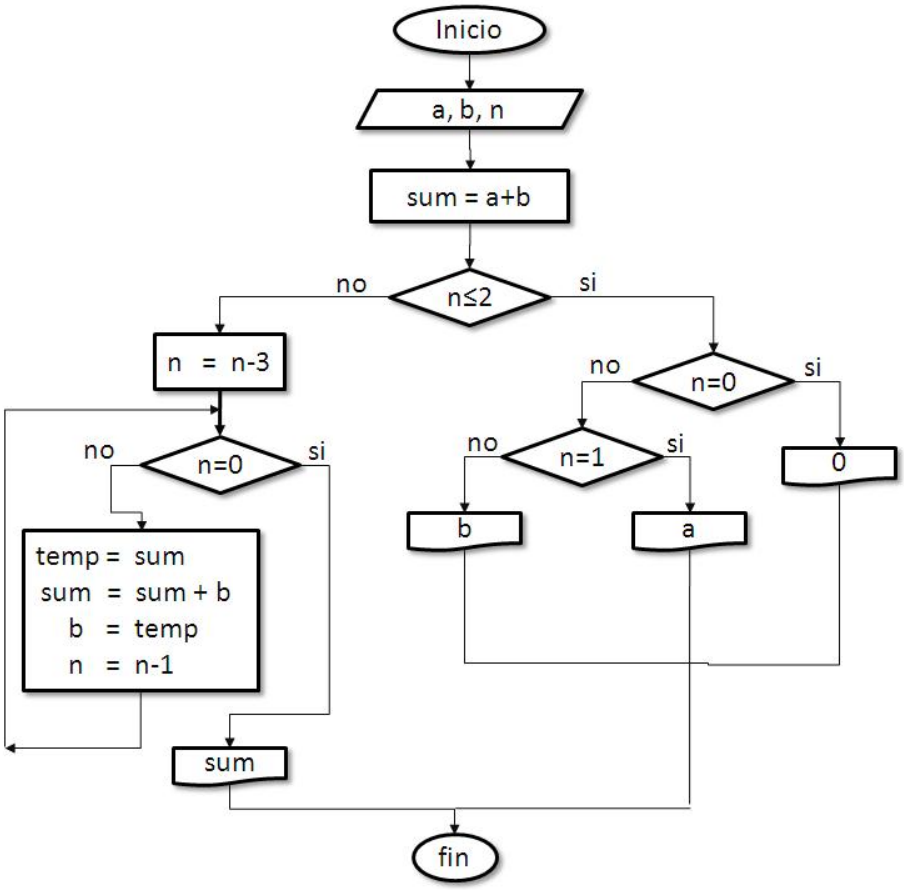


Figura 3.10: Algoritmo para calcular el n-esimo término de la Serie

Diseño del Diagrama ASM:

La construcción del diagrama ASM es un paso de gran importancia en esta metodología, su adecuado diseño permitirá paralelizar la mayor cantidad de operaciones posibles con el propósito de ganar velocidad de procesamiento. Como se mencionó anteriormente todas las operaciones que se incluyan en un mismo estado se realizarán en el mismo ciclo de reloj (en paralelo), lo importante es que entre dichas operaciones no exista *dependencia de datos* y se cuente con el hardware necesario.

En la figura 3.11 se muestra un Diagrama ASM que es equivalente al algoritmo de la figura 3.10, en el cual se ha tratado de paralelizar todas las operaciones posibles. Este diagrama permite hacer una transición más sencilla a una implementación FSMD, puesto que muestra claramente los estados y las acciones que se deben realizar en cada uno de estos estados y las transiciones entre los estados.

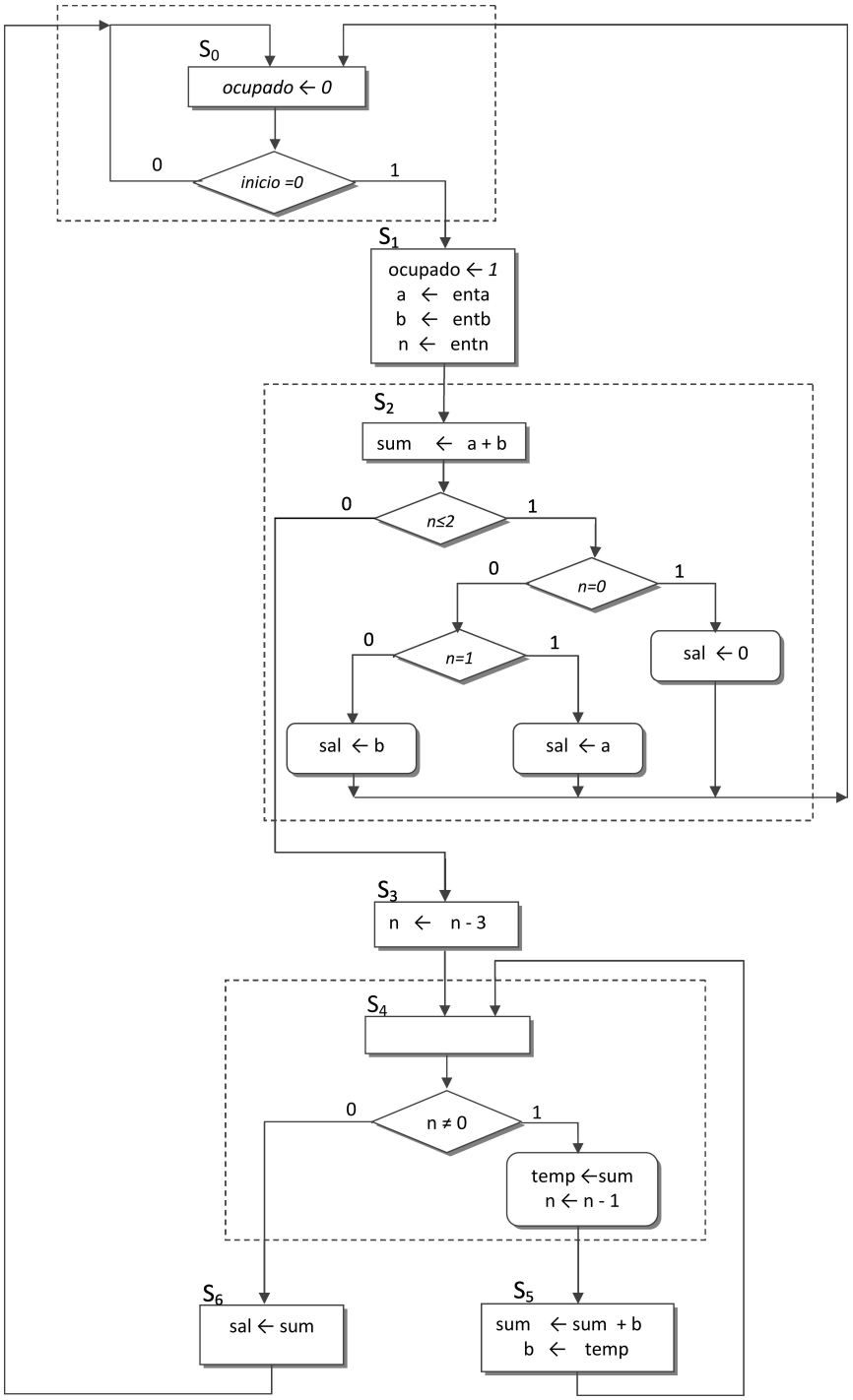


Figura 3.11: Diagrama ASM de la Serie

Diseño del Datapath:

El diagrama ASM contiene información más detallada acerca del algoritmo, lo cual facilita el diseño del *Datapath* y de la FSM. El diseño de un *Datapath* puede ser tan variado como el de un algoritmo en software, es decir, no existen soluciones únicas. A continuación se propone los siguientes pasos para el diseño del *Datapath*, con el propósito optimizar el uso de los recursos lógicos y facilitar su diseño.

- Agrupación de las variables en un número específico de registros.
- Agrupación de todas las operaciones en un número específico de unidades funcionales.
- Agrupación de las operaciones RT de acuerdo a su registro destino.
- Construcción del hardware.
- Adición del hardware requerido para evitar *meta-estabilidad*.

Agrupación de variables

Un primer objetivo al diseñar un *Datapath* es determinar el número de registros a utilizar, el *algoritmo del lado izquierdo* (left-edge algorithm), el cual se expone en [29], permite agrupar tantas variables como sea posible en un mismo registro, con el propósito de ahorrar recursos de almacenamiento y facilitar el diseño del *Datapath*.

El algoritmo del lado izquierdo utiliza una *tabla de uso de variables* con la cual se calcula el *tiempo de vida* de cada una de ellas, con el fin de agrupar en un sólo registro aquellas variables cuyos *tiempos de vida* no se solapan. El *tiempo de vida* de una variable se define como el conjunto de estados en los cuales la variable está *activa*. Los estados en los cuales una variable está *activa* son ([29]):

- Todo estado siguiente a aquél en el que se asigna un nuevo valor a la variable (estado de escritura).
- Todo estado en el que la variable aparece en el lado derecho de una asignación (estado de lectura).
- Todos los estados en cualquier camino entre un estado de escritura y un estado de lectura. En los bucles, se debe ser cuidadoso, por ejemplo si la señal se escribe por fuera de un bucle, el tiempo de vida de la variables dependerá de todos los caminos posibles entre el estado de escritura y el estado de lectura, inclusive si repite un estado de lectura.

El primer paso para la agrupación de variables es la creación de la *tabla de uso de variables*, en esta tabla las filas representan las variables y las columnas los estados, las variables se enumeran según su orden de escritura y en las columnas se indica los estados en los que cada variable se encuentra *activa*, si dos variables tienen el mismo estado de escritura, se da mayor prioridad a la que tiene mayor tiempo de vida, si ambas coinciden tanto en el orden de escritura como en el tiempo de vida, la prioridad se puede

asignar de manera aleatoria. En la tabla 3.1 se muestra la *tabla de uso de variables* correspondientes al diagrama ASM de la figura 3.11.

Tabla 3.1: Uso de variables

	S_0	S_1	S_2	S_3	S_4	S_5	S_6
b			x	x	x	x	
n			x	x	x	x	
a			x				
sum				x	x	x	x
temp						x	

Una vez las variables se han ordenado se toma el primer registro y le asignamos la primer variable de la lista y todas aquellas variables cuyo *tiempo de vida* no se solape con el de esta primer variable, luego se borran de la lista las variables ya asignadas; enseguida se toma otro registro y se sigue el mismo procedimiento anterior hasta que todas variables hayan sido asignadas a un registro. En el ejemplo de diseño se requieren como mínimo cuatro registros, pues al revisar la tabla 3.1 en el estado $S5$ se puede observar que existen 4 variables *activas*. A continuación se muestra la asignación de los registros:

- $b = [b]$
- $n = [n]$
- $s = [a, sum]$
- $t = [temp]$

Es decir, al registro b le corresponden la variable b, al registro n la variable n, al registro s las variables a y sum y al registro t la variable $temp$. De esta manera se ha logrado agrupar las cinco variables en cuatro registros.

Agrupación de operaciones

Para determinar el número de unidades funcionales (sumadores, restadores, etc.) necesarios en el *Datapath*, se puede seguir un procedimiento similar al que se usó para la agrupación de variables. En este caso se utiliza una *tabla de uso de operaciones* ([29]), en donde las filas representan todas las operaciones que aparecen en el diagrama ASM y las columnas corresponden a los estados en los cuáles estas operaciones son utilizadas. En la tabla 3.2 se muestra la *tabla de uso de operaciones* correspondiente al diagrama ASM de la figura 3.11.

Con esta tabla se puede determinar fácilmente cuantas unidades funcionales se requieren para construir el *Datapath*. En este enfoque de diseño el objetivo es paralelizar la mayor cantidad operaciones posibles (esto con el propósito de ganar velocidad de procesamiento), por lo cual el número de unidades funcionales

Tabla 3.2: Uso de operaciones

Operación	S_0	S_1	S_2	S_3	S_4	S_5	S_6
$+$			1			1	
$-$			1	1	1		

será el máximo que ocurra en cualquier estado. Al revisar la tabla 3.2, se puede concluir que en este caso sólo se requieren de dos unidades funcionales: un sumador y un restador, puesto que en ningún estado se utilizan simultáneamente más de una vez estas operaciones. Un análisis adicional que se puede hacer en este punto es la agrupación de unidades funcionales (por ejemplo un sumador-restador) y su efecto en la cantidad de interconexiones, pero en esta metodología no se tiene en cuenta este análisis, para una revisión de este tema se puede ver [29].

Agrupación de las operaciones RT de acuerdo a su registro destino

Una vez se tienen definidos tantos los registros como las unidades funcionales el siguiente paso es *agrupar todas las operaciones que se realizan en el diagrama ASM de acuerdo al registro destino* [2].

A continuación se muestra la agrupación de operaciones por registro destino para el diagrama ASM de la figura 3.11.

- Operaciones con el registro b:
  $b \leftarrow entb$, en S1.
  $b \leftarrow t$, en S5.

- Operaciones con el registro n:
  $n \leftarrow entn$, en S1.
  $n \leftarrow n - 3$, en S2
  $n \leftarrow n - 3$, en S4

- Operaciones con el registro s:
  $s \leftarrow enta$, en S1.
  $s \leftarrow s + b$, en S2 y S5.

- Operaciones con el registro t:
  $t \leftarrow s$, en S4.

Construcción del hardware

El siguiente paso es la construcción del hardware relacionado con cada uno de los grupos definidos en el paso anterior, esto se logra adicionando las unidades funcionales requeridas para las operaciones RT y también incluyendo multiplexores y/o buses para direccionar los datos cuando el registro tiene asociadas múltiples operaciones RT. Adicionalmente se debe añadir el hardware necesario para generar cada una de las *señales de estado* requeridas por el diagrama ASM. Se debe tener en cuenta que para cada caja de decisión se requiere generar la señal de estado correspondiente.

En la figura 3.12 se muestra el circuito relacionado con el registro n, al cual se le ha adicionado un multiplexor para direccionar las dos fuentes que posee el registro, el restador relacionado con los estados $S3$ y $S4$. En este diagrama la señal E_n habilita la escritura del registro n y la señal $m3$ corresponde al selector del multiplexor (si $m3 = 0$ se selecciona $entn$). Además se ha adicionado un comparador para generar la *señal de estado* relacionada con las cajas de decisión en $S2$ y $S4$.

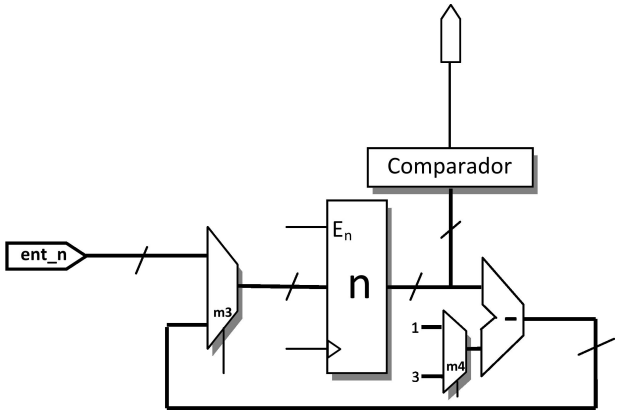


Figura 3.12: Hardware relacionado con el registro n

El mismo procedimiento mencionado anteriormente, se realiza para cada registro y luego se interconectan los diferentes circuitos. En la figura 3.13 se observa el *Datapath* completo.

Diseño libre problemas de *Metaestabilidad*

Cuando se ingresan datos externos al procesador, no siempre es posible garantizar que al capturar dichos datos en los registros se estén respetando los tiempos de *setup* y de *hold* de los flip flops que forman los registros, si estos tiempos no se respetan el comportamiento del registro puede llegar a ser aleatorio, esta condición se conoce con el nombre de *Metaestabilidad* ([31]). Con el propósito de garantizar que los datos de entrada al procesador sean confiables, se ha implementado el protocolo *Start-Finish* [28], el cual se implementa utilizando la señal de entrada *inicio* y la señal de salida *ocupado*. El protocolo exige que una

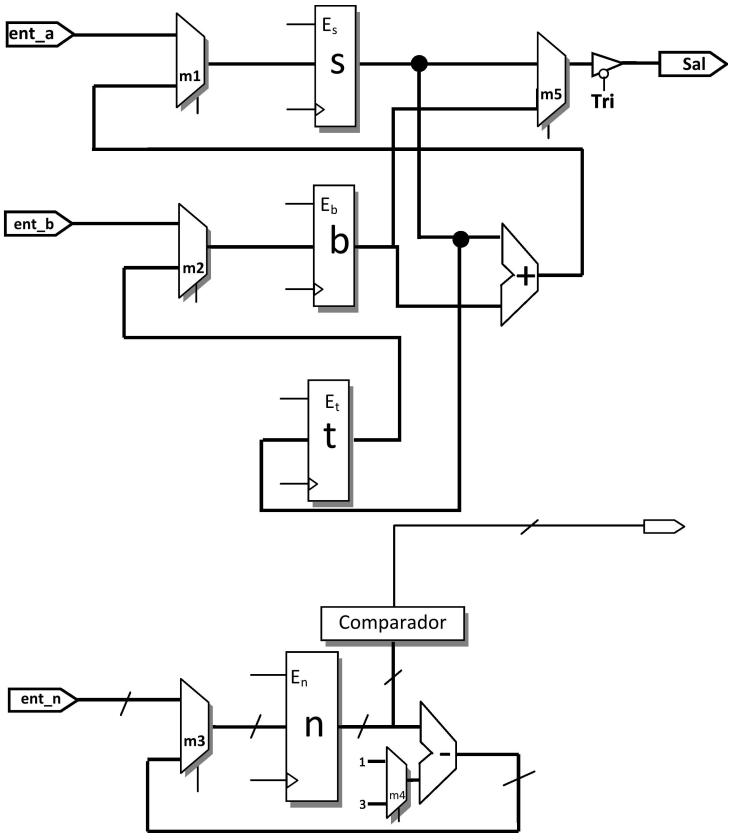


Figura 3.13: Datapath

vez los datos estén listos en los puertos de entrada, se active la señal *inicio* y los datos son almacenados en los registros en el siguiente flanco de reloj, lo cual asegura que se respeten los tiempos de *setup* y de *hold*. Adicionalmente el protocolo *Start-Finish* también permite que este procesador pueda ser utilizado como esclavo en un diseño modular.

De otro lado la señal de *inicio* también es asíncrona y va directamente a los registros de estado a través de la lógica del estado siguiente; en este caso es probable que esta señal también viole los tiempos de *setup* y *hold* de los flip flops del registro de estado. Para solucionar este problema se propone la interfaz de la figura 3.14, para todas las señales asíncronas de un bit, este circuito disminuye considerablemente las probabilidades de tener problemas de *metaestabilidad*; básicamente este circuito detecta el flanco de subida de la señal asíncrona y entrega un pulso de una duración de dos ciclos de reloj en la *salida*, este pulso se encuentra sincronizado con el reloj principal del sistema y con baja probabilidad de que se presente *metaestabilidad* en el último flip-flop de la figura 3.14. Para revisar interfaces de señales asíncronas de más bits se puede ver [1].

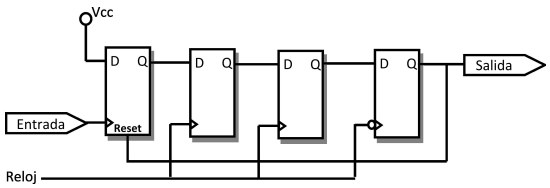


Figura 3.14: Interfaz para la señal de inicio [1]

Diseño de la FSM

Un algoritmo es una secuencia de operaciones, por lo tanto se requiere de un control que permita hacer la secuenciación apropiada de las operaciones que se realizan dentro del *Datapath*, simulando así los saltos y las iteraciones en un algoritmo. Esta unidad de control se puede implementar por medio de una FSM, en donde cada estado corresponde a una iteración en el algoritmo, es decir, en cada estado la FSM genera las señales de control que llegan al *Datapath* para que este pueda realizar una operación determinada, estas señales se agrupan en una *palabra de control* como se muestra en la figura 3.4. La FSM examina los *comandos* y las *señales de estado* para generar la *palabra de control* apropiada que le permitan al Datapath realizar una instrucción RT. La FSM genera una *palabra de control* para cada uno de los estados en el diagrama ASM. En el diseño propuesto la FSM controla la escritura en los registros y direcciona los datos por medio de las señales E_i y m_i respectivamente, adicionalmente controla el triestado (TE) y genera la salida de control *ocupado*.

Para realizar las diferentes operaciones RT del algoritmo se requerirá enviarle al *Datapath* las diferentes palabras de control que aparece en la tabla 3.3. Por medio de estas *palabras de control* se habilita la escritura de los registros (por medio de las señales E_i), se direccionan los datos (por medio de las señales m_i), se habilita el triestado (TE) y se genera la señal de *ocupado*. Para hacer el diseño de la FSM se pueden utilizar métodos clásicos como los que aparecen en [2], [32], [33].

Tabla 3.3: Palabra de control para las operacines RT

Operación RT	Es	Eb	Et	En	m1	m2	m3	m4	m5	TE	Ocupado
$a \leftarrow enta$	1	1	0	1	0	0	0	0	0	0	1
$a \leftarrow enta$											
$a \leftarrow enta$											
$sum \leftarrow a + b$	1	0	0	0	1	0	0	0	0	0	1
$n \leftarrow n - 1$	0	0	0	1	0	0	1	0	0	0	1
$n \leftarrow n - 3$	0	0	0	1	0	0	1	1	0	0	1
$sum \leftarrow sum + b$	1	0	0	0	1	0	0	0	0	0	1
$b \leftarrow t$	0	1	0	0	1	0	0	0	0	0	1
$sal \leftarrow sum$	0	0	0	0	0	0	0	0	0	1	1

Diseño sin los problemas generados por los *glitches*

Una FSM con lógica de salida combinacional generalmente presenta *glitches* ([33] [28]), debido a que no todos los bits de estado responden al mismo tiempo. En las implementaciones FSMD los *glitches* que se pueden generar en las salidas de la FSM, no tienen impacto sobre el funcionamiento del procesador ya que estas palabras de control serán utilizadas en el siguiente flanco de reloj, es decir, cada salida de la FSM tiene un ciclo de reloj completo para estabilizarse. Como se mencionó anteriormente una técnica que se puede utilizar para eliminar los *glitches* en las salidas de FSM, es eliminando la lógica de salida, esto se logra haciendo que cada salida de la FSM dependa exclusivamente de un bit de estado, esta estrategia se puede revisar en [28].

Descripción del procesador utilizando VHDL

El estilo de diseño que se propone para hacer la descripción en VHDL del *Datapath* es estructural, es decir, cada una de las unidades funcionales se describe por aparte y luego se unen en un archivo de alto nivel.

Las unidades funcionales no necesariamente deben ser descritas por el diseñador, pues éstas pueden descargarse directamente de los *ip-cores* que ofrecen los fabricantes de FPGAs o descargarse de páginas como *Open Cores*; por ejemplo, si este procesador se diseñara para trabajar en formato de coma flotante, se podrían utilizar los *ip-cores* de suma y resta en coma flotante que ofrece la empresa Xilinx junto con su software ISE [34]. Una vez se ha diseñado el *Datapath* la descripción en un lenguaje HDL se convierte en un proceso casi mecánico. De otro lado para la descripción de la FSM se propone utilizar el estilo *multi-segmento* que se expone en [2], en donde cada componente de la FSM (registro de estado, lógica del estado siguiente y lógica de salida) se describe en bloques (*process*) diferentes. Todas las descripciones en VHDL de este diseño se encuentran disponibles en el Anexo 1 (Sección 3.8).

Resultados obtenidos

En las figuras 3.15 y 3.16 se muestran algunos datos obtenidos mediante Simulación utilizando el software ISE 10.1 de Xilinx.

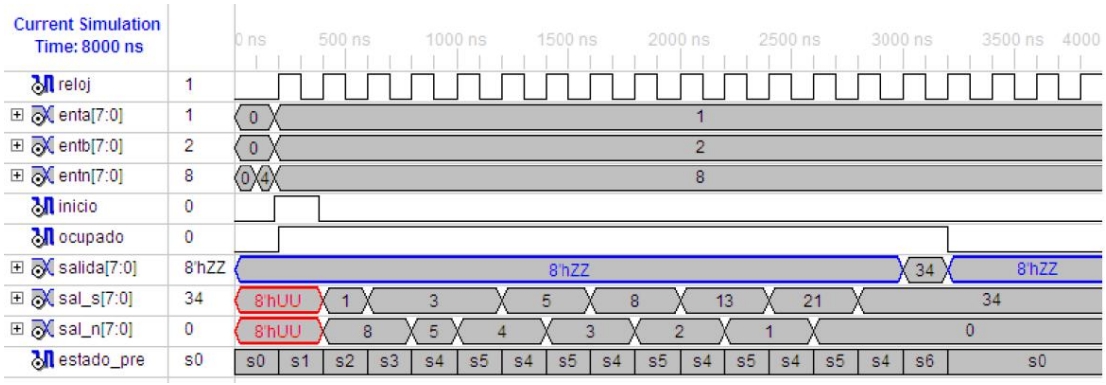


Figura 3.15: Simulación del procesador (a=1, b=2 y n=8)

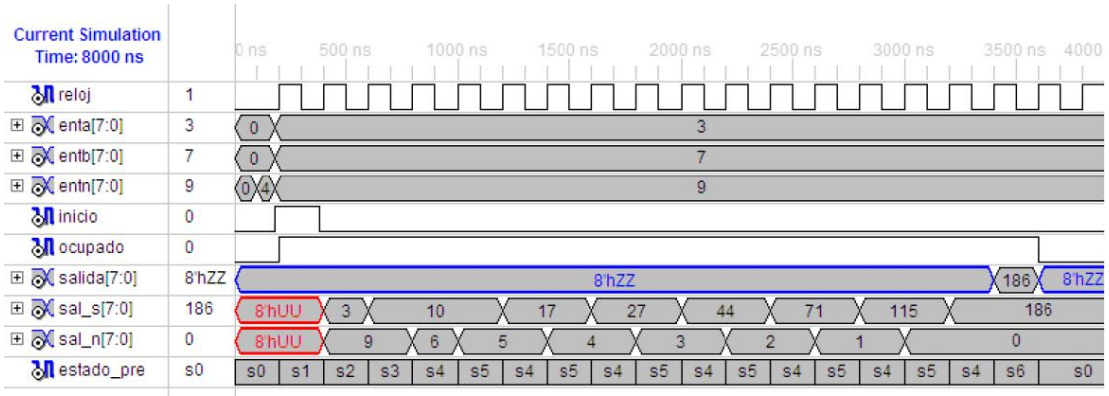


Figura 3.16: Simulación del procesador (a=7, b=3 y n=9)

Capítulo 4

Resultados: Validación de la Metodología Propuesta

4.1 Introducción

Con el propósito de validar la metodología propuesta en esta tesis de maestría se diseñaron e implementaron varios ES. En este capítulo se describen los más significativos. El primero de ellos es un verificador de precios que utiliza una pantalla, un teclado y un lector de barras. El segundo es un procesador de propósito específico que permite calcular cuatro parámetros necesarios dentro de la *Migración Sísmica*, un proceso utilizado por la industria petrolera durante la extracción de hidrocarburos. Finalmente la metodología propuesta en esta tesis se utilizó para rediseñar un procesador de propósito específico que permite realizar un filtrado pasa bajas a través múltiples transformadas de Fourier basada en el método de descomposición en factores primos. A continuación se resumen cada uno de estos trabajos.

4.2 Diseño 1: Verificador de Precios

El primer diseño realizado es un verificador de precios, en el que se quiere ofrecer la posibilidad de que el usuario pueda realizar la consulta del valor del artículo ya sea por teclado o por lector de código de barras, adicionalmente el ES deberá llevar el valor de la cuenta de todos los artículos consultados. La visualización se hace por medio una interfaz VGA. El diseño se implementa en el FPGA 3SD1800A-FG676 [14] que se encuentra dentro del sistema de desarrollo SPARTAN 3A-DSP [14] de la empresa Xilinx.

Arquitectura general de verificador de precios

La arquitectura general del verificador de precios se muestra en la figura 4.1. El ES contiene el procesador soft-core *MicroBlaze* [23] en donde se implementará la parte de software del diseño, adicionalmente el ES posee tres módulos de hardware descritos en VHDL a manera de *IP cores*.

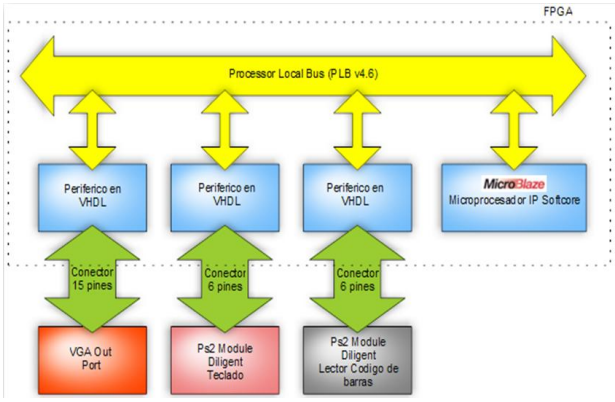


Figura 4.1: Arquitectura general de Verificador de Precios

Módulos IP del Diseño

Cómo se comentó anteriormente este diseño cuenta con tres módulos de HW los cuales se seleccionaron teniendo en cuenta lo descrito en la sección 2.4. Estos módulos se describen a continuación:

- **Controlador VGA** (*Video Graphics Array*): esta interfaz es una norma de visualización de gráficos creado por IBM a fines de los años 80 para sus computadoras y procesadores, emplea una interfaz básica con resolución de 640x480 de 8 colores con señales de sincronismo horizontal y vertical para la conformación de campos [2].

- **Puerto Ps2**: protocolo introducido por las computadoras personales IBM, siendo una amplia interfaz utilizada para comunicar algunos dispositivos como teclado, mouse, lector de código barras entre otros. En el presente trabajo se utilizan dos dispositivos que esta interfaz: el teclado y el lector código de barras.

Entorno de desarrollo *EDK* (*Embedded Development Kit*

El *EDK* [35] [22] es una herramienta que permite el desarrollo de un ES utilizando la metodología de codiseño, la cual cuenta con la *Plataforma de Estudio de Xilinx* (XPS), que contiene una interfaz gráfica con un conjunto de herramientas que dan la posibilidad de diseñar un sistema de un ES basado en un procesador soft-core como el *MicroBlaze* [23] o un hardcore como el *Power PC* [16]. Esta herramienta adiciona al procesador módulos a manera de IP, integrando así el SW y HW en un mismo diseño. El flujo de diseño para desarrollar un ES en la plataforma XPS se muestra la figura 4.2 el cual contiene dos plataformas una para el diseño del HW y otra para el diseño del SW.

En la plataforma hardware el diseñador define la arquitectura del sistema la cual incluye la configuración del microprocesador, los periféricos, la declaración de puertos entre otros. Una vez la arquitectura del

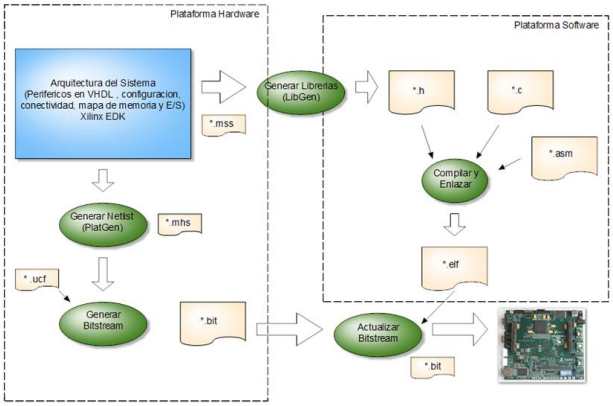


Figura 4.2: Flujo de diseño en XPS

sistema se encuentra definida, mediante la herramienta PlatGen (Hardware Platform Generation) dando click en la opción Generate Netlist, se construye el sistema de procesamiento empotrado como un conjunto de listas de conexionado hardware (es decir ficheros HDL y netlists de implementación). Para finalizar con la plataforma de Hardware se procede a la generación del Bitstream para la configuración de la FPGA [36].

Para finalizar el diseño del ES en la plataforma Software se crean librerías que contienen Drivers que permiten controlar los periféricos adicionados. Estas librerías se generan mediante la plataforma LibGen y son utilizadas por las aplicaciones software del sistema empotrado. Para el desarrollo software de la aplicación el XPS incorpora un compilador en lenguaje C que permite obtener el archivo ejecutable (.elf) que manejara el Microprocesador. Una vez terminado el diseño de la plataforma Software, se integran la parte de SW y HW para su posterior implementación en la FPGA, para esto en la pestaña de XPS Device Configuration primero se da un click en la opción Update Bitstream y después descarga en la tarjeta con Download Bitstream [36]. Cómo se mencionó en la sección 4.2 en el Anexo se puede revisar en detalle el proceso de diseño de un ES utilizando *EDK*.

Módulos de hardware

La primera fase del diseño es la descripción en VHDL de cada uno de los módulos IP, el particionamiento se basó en lo descrito en la sección 2.4. Las descripciones se tomaron de [2], dichos códigos son adecuados para ser utilizados en el ES, los periféricos teclado y lector de código de barras Ps2 aprovechan el mismo código en VHDL del circuito receptor y decodificador, para el caso de la pantalla VGA su descripción en VHDL se implementa en dos circuitos para la sincronización y la generación de video. En este caso se utilizó el software ISE de Xilinx para verificar el funcionamiento de los módulos antes de la incorporación al microprocesador MicroBlaze.

Teclado y lector Código de Barras PS2

La idea principal del diseño del modulo es la recepción del código enviado por la tecla oprimida por el teclado caracterizándolo con un código en lenguaje ASCII para su recepción en el MicroBlaze. Basándose en este conocimiento la descripción en hardware debe contar de dos circuitos el receptor (*Ps2rx*) y el codificador a ASCII (*asciicode*) (Ver anexo 3). Cada envío unidireccional que el periférico Ps2 hace al modulo IP contiene un BIT inicial, 8 bits de dato, un BIT de paridad y un BIT de parada, como se muestra en la figura 4.3.

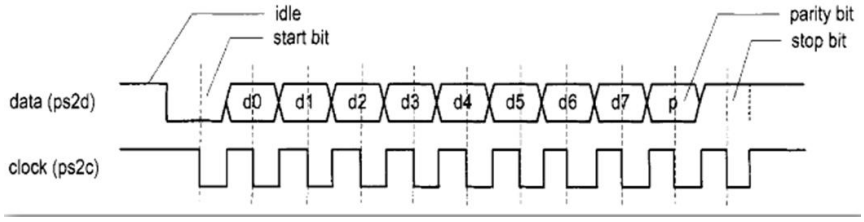


Figura 4.3: Protocolo PS2 [2]

Esta recepción por parte del modulo se describe por medio de una maquina de estados como se muestra en la figura 4.4.

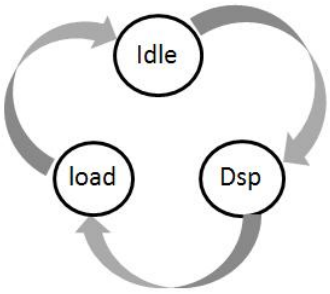


Figura 4.4: Máquina de estados para el receptor PS2 [2]

En el estado *idle* o de *inicio* se espera un BIT de entrada proveniente del pin Ps2d el cual es transferido con el flanco descendente del reloj ps2c y el BIT de habilitación del receptor, a demás se activa la condición inicial en 1001 para el contador interno, que se encuentra en el siguiente estado *dsp*, donde se acumulan los 8 bits que conforman el dato final mas el BIT de paridad. Debido a la transmisión unidireccional el receptor siempre estará activo esperando el envío de información por parte del dispositivo Ps2.

Esta operación de acumular los bits de entrada en el registro se finaliza al completar los 9 bits (8 bits de datos + 1 paridad), indicando con un BIT de parada la finalización de dicho proceso, por último se requiere de un estado o ciclo de reloj load necesario para completar el envío al codificador *asciicode* que se encarga de enviar al procesador el código ASCII del carácter oprimido.

Controlador VGA

La pantalla VGA se puede ver como un arreglo matricial de 480x640, donde cada una de las celdas equivale a un pixel. El controlador VGA consta de una unidad para la generación de video y otra de sincronismo como se muestra en la figura 4.5. Se define por coordenadas cartesiana el área de trabajo para hacer algún tipo desplazamiento vertical u horizontal a través de la pantalla. La unidad de sincronismo es la encargada de realizar dicho desplazamiento

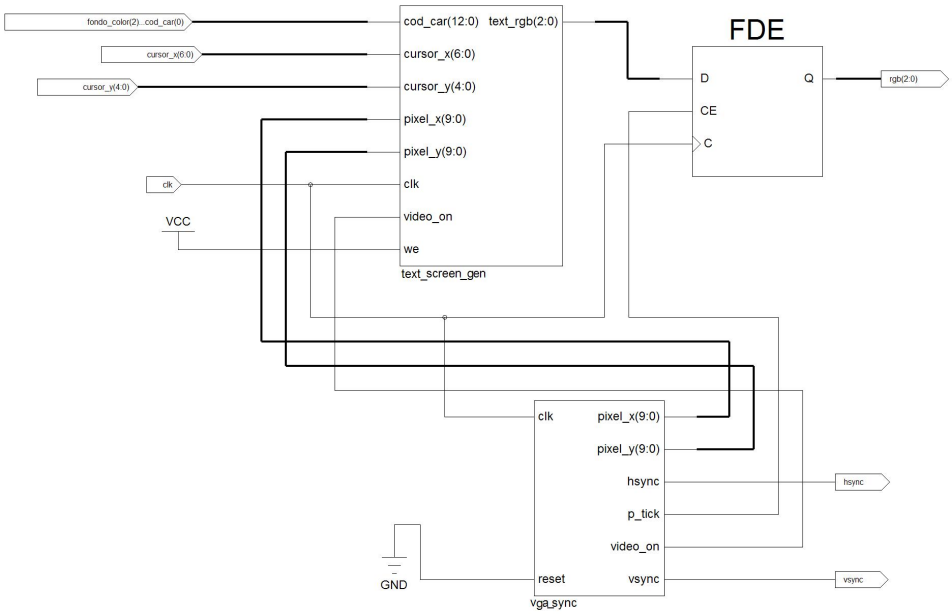


Figura 4.5: Conexión VGA

Se utilizan 8 colores por pixel (3 bits) lo cual con lleva a usar 921600 bits (640*480*3 bits) de memoria. Para reducir los requerimientos de memoria se usa un esquema de partes gráficas en la que cada parte gráfica tiene una resolución de 8x16 pixeles situación que reacomoda la resolución de la pantalla a 80 x 30 donde cada parte gráfica se maneja como una unidad de pantalla. Esto hace que el requisito de Memoria RAM sea de 2400 posiciones.

En la unidad de generación de video se puede encontrar dos tipos de memoria: ROM (Fontrom contiene la patrones de partes gráficas) y RAM (memoria de acceso al contenido ROM). La memoria RAM es el componente que permite almacenar la imagen en forma de datos, es decir donde para cada parte gráfica se definen color de fondo, código y color del carácter. Con esos tres elementos el circuito de generación de video puede reproducir la imagen propiamente dicha en la pantalla. El circuito de sincronización siempre está realizando lecturas periódicas a velocidad constante de todas las posiciones de la memoria. De la RAM sale el código ASCII almacenado en una posición de memoria que guarda relación uno a uno con la posición de la pantalla (es decir la posición en la que se almacena el carácter en la RAM es la misma

posición en la que debe aparecer el carácter en la pantalla) y entra en la Fontrom donde se encuentran los mapas de bits de cada carácter. El circuito generador de video entiende 0 como fondo y 1 como texto en el patrón del carácter o parte gráfica.

Adición de los módulos de HW al MicroBlaze utilizando el entorno *EDK*

Con el hardware de los módulos IP ya desarrollados exitosamente se continúa con la conexión de cada uno de éstos al MicroBlaze. En la figura 4.6 se muestra los archivos que se deben generar o actualizar para el proceso de conexión siguiendo el modelo para la implementación de los periféricos en EDK.

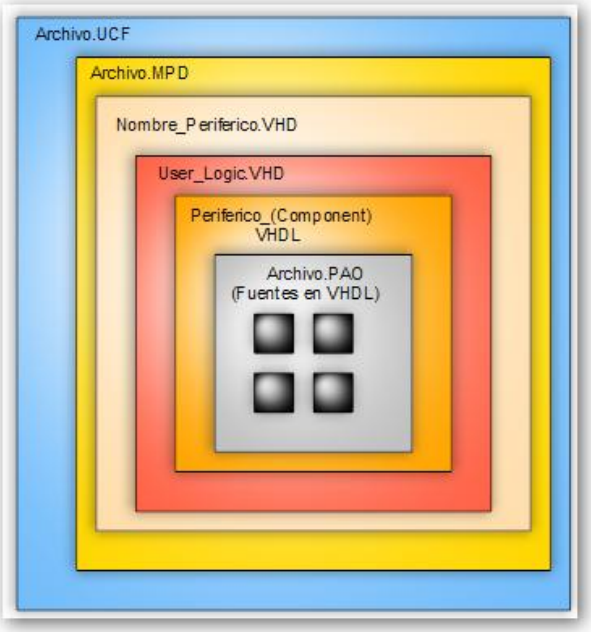


Figura 4.6: Sistema de archivos para la implementación de un módulo IP

Dicho proceso de conexión empieza una vez termina la secuencia de pasos encontrada en la pestaña Hardware, *Create and import peripheral Wizard* (creación o importación de periféricos) de EDK donde se define el nombre, versión, conexión al bus, numero de registros necesarios para su funcionamiento, entre otras características, allí se crean archivos que describen el hardware del periférico y su conexión con el Microprocesador MicroBlaze. Esta secuencia de pasos están descritos en el anexo 2.

Los archivos actualizados y generados en el transcurso del proceso de adición de módulos IP al procesador MicroBlaze se describen a continuación:

- **MPD** (Microprocessor Peripheral Description):Este archivo se encuentra en pcores/Nombre_periferico/data/nombreperiferico.mpd donde se encuentran todos los puertos disponibles con sus parámetros de hardware y descripción (dirección y tipo) del periférico [22].

- **NOMBRE_PERIFERICO.vhd:** Ahora el archivo que se debe modificar es nombre_periferico.vhd se encuentra en la carpeta, pcores/Nombre_periferico/hdl/ donde se debe definir los puertos externos del periférico y la instanciación al archivo de más bajo nivel USER_LOGIC.VHD [22].

- **USER_LOGIC.vhd:** El archivo USER_LOGIC es uno de los archivos más importantes en el proceso ya que éste es la interfaz entre el microprocesador y el periférico. En la entidad se definen los puertos de conexión con el archivo de nivel superior (nombre_periferico.vhd). En la arquitectura del user_logic se desarrolla la conexión del periférico con el MicroBlaze, donde se declara el componente mediante la palabra COMPONENT definiendo las entradas y salidas del periférico y con la clave PORTMAP se define la conexión a los puertos, además se hace la conexión del periférico con los registros esclavos necesarios para la comunicación con el microprocesador [22].

- **PAO**(Peripheral Analyze Order): El siguiente archivo que se actualiza es el .PAO (Peripheral Analyze Order). Donde se define el orden de lista de archivos HDL necesarios para la síntesis y simulación [22]. Cuando el periférico diseñado en VHDL tiene varias fuentes la forma de instanciarlas en EDK es a través de este archivo, donde se copia un código estándar en cada línea (por ejemplo lib teclado_v1_00_a key2ascii vhdl) que hace referencia a librería, nombre_periférico, nombre de la fuente y extensión. El orden en que se deben escribir los archivos es en forma descendente teniendo en cuenta la jerarquía o como están instanciados. Estas fuentes deben estar copiadas en la carpeta pcores/nombre_periferico/hdl.

- **UCF**(User Contrains File): Por último el archivo UCF generado por Base System Builder Wizard de acuerdo a la plataforma de desarrollo en la cual se esté implementando el ES, en este caso la Spartan 3A DSP Board, este archivo se actualiza con los nuevos periféricos, teniendo en cuenta el tipo de módulo que se va a implementar. En el software EDK existe un estándar para la de declaración de puertos externos el cual se puede descargar de [37] donde para cada pin se define la localización del puerto físico y nivel de tensión [22].

Con los módulos IP ya implementados y verificados en el procesador MicroBlaze se crea el ES con la integración de los tres periféricos en un solo diseño como se puede ver en el diagrama de bloques de la figura 4.7 donde se observa que dicha conexión se hace mediante el Bus PLB [22].

Modulo de SW

El código ejecutado por el microprocesador para esta aplicación íntegra las funciones creadas por la herramienta LibGen para los módulos teclado, lector de código de barras e interfaz VGA y otras funciones creadas por los diseñadores, todo en en el compilador de C que posee la XPS.

De acuerdo a lo anterior las funciones realizas por el software son:

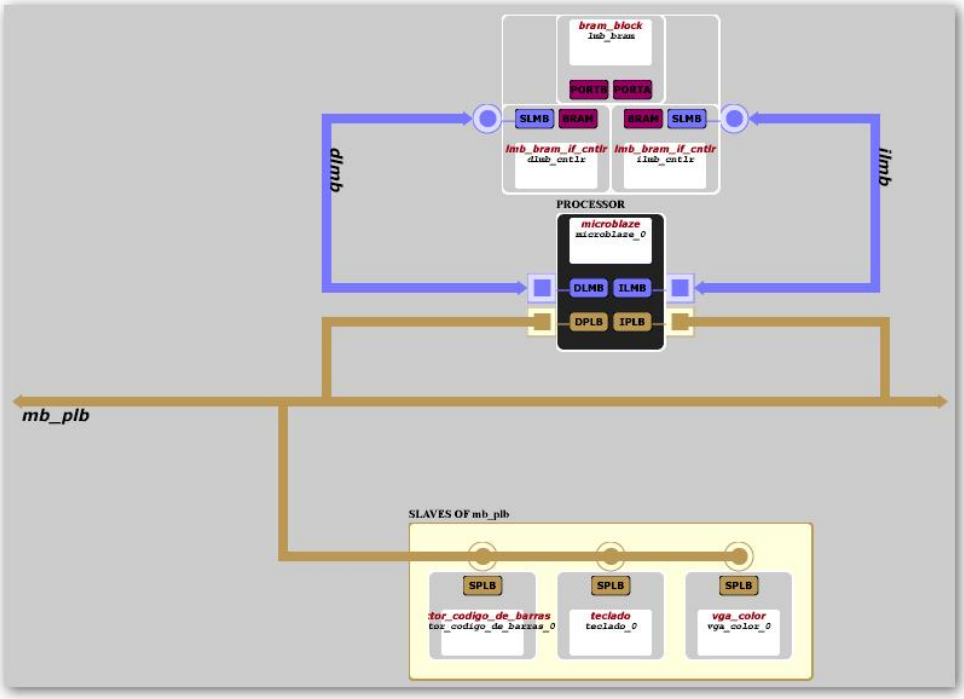


Figura 4.7: Diagrama de bloques en EDK del ES

- Definir la región de visualización mediante coordenadas (x,y) imprimiendo el carácter espacio en toda esta área.
- Imprimir una cadena de caracteres en una ubicación a conveniencia del programador.
- Dos funciones de retardo.
- Almacenar los datos leídos en un vector por el lector de código de barras.
- Lectura de las teclas oprimidas en el teclado
- Guardar los datos digitados en el teclado en un vector e ir visualizando cada carácter en el VGA
- Conversor de valores hexadecimales a valores enteros.
- Conversor de valores enteros a hexadecimales.
- Sumar los valores de los productos consultados.

Resultados

El diseño se probó y resulto ser tanto fiable como estable. La arquitectura física de la aplicación se muestra en la figura 4.8 donde se utiliza un monitor de 15" con conexión VGA de 15 pines, un teclado con puerto Ps2, un lector de código de Barras Ps2, dos módulos PS2 Diligent y la Spartan 3A DSP.

Figura 4.8: Arquitectura Física del ES

En la figura 4.9 se muestra la interfaz gráfica que maneja el usuario, con el color de texto y fondo definido, mostrando también el código, el detalle y el valor del último artículo consultado, además en la parte inferior se sitúa el valor de la cuenta total realizada en la consulta, el usuario tiene la opción de realizar una nueva cuenta presionando la tecla N.

4.3 Diseño 2: Procesador de propósito específico para la industria petrolera

Las empresas relacionadas con la industria del petróleo realizan múltiples procesos para obtener las imágenes del subsuelo y uno de estos es la migración sísmica, en el cual se usan clústers de computadores para poder obtener resultados más rápidamente, *debido al gran costo computacional ocasionado por los grandes volúmenes de datos que se deben procesar* [10]. El problema principal de estas empresas es que el proceso de migración puede tardar meses, debido a que los volúmenes de datos a procesar son del orden de Gigabytes [10]. La industria del petróleo, siempre está buscando reducir los tiempos de procesamiento para poder aumentar sus exploraciones anuales. *Una muestra de ello son sus considerables inversiones*

Figura 4.9: Interfaz Grafica del ES

económicas en clústers cada vez más grandes, que a su vez requieren de grandes sistemas de enfriamiento y su uso acarrea consumos considerables de energía [10], por lo tanto, *la idea de tener un sistema de procesamiento específico que ayude a acelerar este proceso sin ocupar mucho espacio y cuyo consumo de potencia sea bajo, empezó a tener acogida dentro de la industria del petróleo* [10], además, el crecimiento de los procesadores específicos, ha estado vinculado con el desarrollo de los FPGAs, en los que se han obtenido índices de aceleración buenos, comparados con los procesadores de propósito general.

Actualmente se encuentran dos clases de programas que son usados en el proceso de migración; la primera clase es de tipo industrial en la que el código fuente es privado porque son desarrollos de software que algunas empresas venden e incluso patentan; la segunda clase es de tipo académico, luego el código fuente es de libre uso y ha sido desarrollado en lugares como la Escuela de Minas de Colorado (Estados Unidos) [10].

Al revisar el código fuente que es de libre uso, se encuentra que matemáticamente la migración implica

operaciones como suma, resta, multiplicación, división, potenciación, radicación, logaritmación y funciones trigonométricas. Esto se puede apreciar en las ecuaciones 4.2 - 4.9, que muestran las operaciones necesarias para calcular el *tiempo de vuelo*, la *desaceleración lateral*, el *coseno del ángulo incidente* y *el ángulo emergente* [10], cuatro parámetros necesarios dentro del proceso de *Migración Sísmica*. El cálculo de estos parámetros se realiza en el formato *punto flotante de precisión sencilla* ocasionando un alto costo computacional [10]. De acuerdo a lo expuesto anteriormente, se evidencia la necesidad de diseñar un procesador de propósito específico que permita calcular estos cuatro parámetros, en donde se aplique el paralelismo que ofrecen los módulos de HW con el propósito de buscar velocidad de procesamiento.

Parámetros a calcular

Para realizar el cálculo de los parámetros mencionados anteriormente se necesita el siguiente valor:

$$r_{ou} = \sqrt{r^2 + z^2} \tag{4.1}$$

Adicionalmente, los parámetros dependen de la variación de la velocidad en función de la profundidad para hacer sus cálculos, por esto, se cuenta con dos grupos de ecuaciones, las cuales se presentan a continuación. En primer lugar se presentan las ecuaciones de los parámetros para cuando no hay variación de la velocidad en función de la profundidad, es decir, cuando dVz=0:

- Tiempo de viaje:

$$t = \frac{r_{ou}}{V_o} \tag{4.2}$$

- Desaceleración Lateral:

$$P = \frac{r}{r_{ou}+V_o} \tag{4.3}$$

- Coseno del ángulo incidente:

$$cos\theta = \frac{z}{r_{ou}} \tag{4.4}$$

- Ángulo emergente

$$ang = \arcsin \frac{r}{r_{ou}} \tag{4.5}$$

Mientras que el segundo grupo de ecuaciones corresponde cuando $dV_z \neq 0$:

- Tiempo de viaje:

$$t = \frac{\log\left(\frac{V\left(r_{ou}+\frac{r^2+z^2-\frac{V_o^2}{dV_z^2}}{2r}\right)}{V_o\left(r_{ou}+\frac{r^2+z^2-\frac{V_o^2}{dV_z^2}}{2r}-r\right)}\right)}{dV_z} \tag{4.6}$$

- Desaceleración Lateral:

$$P = \frac{\sqrt{r_{ou}^2-\left(\frac{r^2+z^2-\frac{V_o^2}{dV_z^2}}{2r}\right)^2}}{r_{ou}+V_o} \tag{4.7}$$

- Coseno del ángulo incidente:

$$cos\theta = \frac{\frac{r^2+z^2-\frac{V_o^2}{dV_z^2}}{2r}}{r_{ou}} \tag{4.8}$$

- Ángulo emergente

$$ang = \arcsin\left(z * dV_z * P\right) \tag{4.9}$$

Unidades funcionales

Las unidades funcionales de *Suma, Resta, Multiplicación, División* y *Raíz cuadrada* en punto flotante, fueron generadas por medio del ***Xilinx CORE Generator System*** [3], la cual es una herramienta flexible de alto rendimiento que ofrece a los diseñadores los medios para generar los códigos en un HDL de los módulos que realizan estas operaciones. En la figura 4.10 se muestra el diagrama de bloques del operador de punto flotante generado con esta herramienta y en la tabla 4.1 se muestra el número de ciclos de reloj que demora cada módulo en realizar una operación completa.

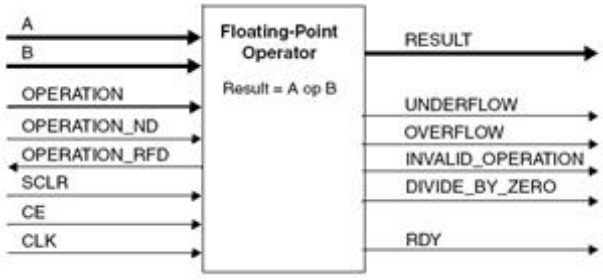


Figura 4.10: Operador de punto flotante [3]

Tabla 4.1: Duración en ciclos de reloj de las operaciones

Operación	Duración (# Ciclos)
Producto	9
Suma	12
Resta	12
Raiz	28
División	28

Para realizar la operación de $arcsin(\theta)$ (ver ecuación 4.9), se utilizó el módulo ***CORDIC*** [38] incluido en la herramienta *Core Generator*, con el cuál se generó el núcleo de la descripción de hardware para la función trigonométrica *arctangente*. El módulo para la *función logaritmo* (ver ecuación 4.6)fue tomado del trabajo *A hardware-independent fast logarithm approximation with adjustable accuracy* [39]. Este módulo permite calcular el valor de logaritmo natural de un número en punto flotante, pero para el cálculo del tiempo de viaje se necesita el logaritmo en base 10, por lo tanto se aprovecho este módulo y luego se hizo un cambio de base. En la tabla 4.2 se muestra los ciclos de reloj que demoran cada uno de estos módulos en realizar una operación:

Tabla 4.2: Duración en ciclos de reloj de las funciones *arctangente* y *logaritmo*

Operación	Duración (# Ciclos)
Arcotangente	20
Logaritmo natural	23

Resultados

El diseño del procesador de propósito específico que permite calcular los cuatro parámetros mencionados anteriormente se realizó utilizando la metodología descrita en el capitulo anterior.

En la Tabla 4.3 se presentan los resultados teóricos de los cuatro parámetros cuando los datos de entrada son: $r = 4$, $z = 5$, $V_o = 0.5$, $dV_z = 0$ y $V = 3$.

Tabla 4.3: Datos de salida del procesador ($dV_z = 0$))

Parámetro	Valor Teórico
TV	12,8062484748657
DL	0,5794477779075
CAI	0,78086880944
AE	0,674740942224

En las figura 4.11 se muestra la simulación del procesador utilizando el software ISE Simulator (ISim) de Xilinx versión 10.1. En esta figura las primeras 5 filas representan las entradas que son r, z, dVz, Vo, V

(los datos están en formato de Coma Flotante precisión sencilla), las siguientes filas son el *reloj* del sistema, la señal de *reset*, la señal de *inicio* con la cual se da comienzo o enciende el procesador, la señal de *estado* de la FSM, una señal de *ocupado* que indica si el procesador se encuentra realizando operaciones (1) o si está listo para recibir los siguientes datos de entrada (0). Por último, se encuentran 4 filas que nos arrojan las salidas del sistema que son: *tiempo de viaje* (TV), *desaceleración lateral* (DL), *coseno del ángulo incidente* (CAI) y el *ángulo emergente* (AE).

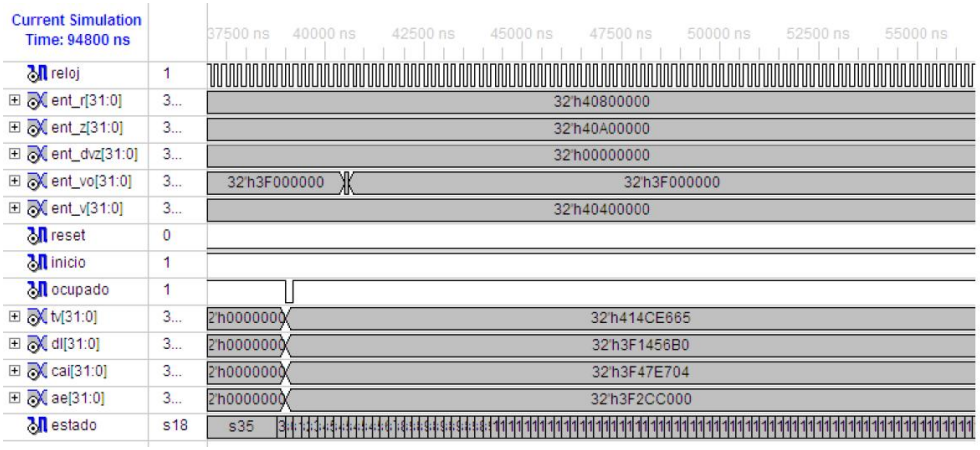


Figura 4.11: Simulación del procesador específico ($dV_z = 0$)

En la tablas 4.4 se muestran los porcentajes de error para cada uno de los parámetros, con los mismos valores de entrada con que se calcularon en la Tabla 4.3.

Tabla 4.4: Porcentajes de error para valores de salida cuando $dV_z \neq 0$

Parámetro	Teórico	Calculado	Error Porcentual
TV	12,8062484748657	12,8062486648559	1,48357E-06 ≈ 0
DL	0,5794477779075	0,5794477462769	5,45876E-06 ≈ 0
CAI	0,78086880944	0,78086876869	5,21868E-06 ≈ 0
AE	0,674740942224	0,674804687500	0,00944737

En la Tabla 4.5 se presentan los resultados teóricos de los cuatro parámetros cuando $dV_z \neq 0$. Los datos de entrada son: $r = 4$, $z = 5$, $V_o = 0.5$, $dV_z = 0.1$ y $V = 3$.

Tabla 4.5: Datos de salida del procesador ($dV_z \neq 0$))

Parámetro	Valor Teórico
TV	10,588310953
DL	0,881160807930105
CAI	0,312347523777
AE	0,456245107

En la figura 4.12 se muestra la simulación para es último set de datos ($dV_z \neq 0$).

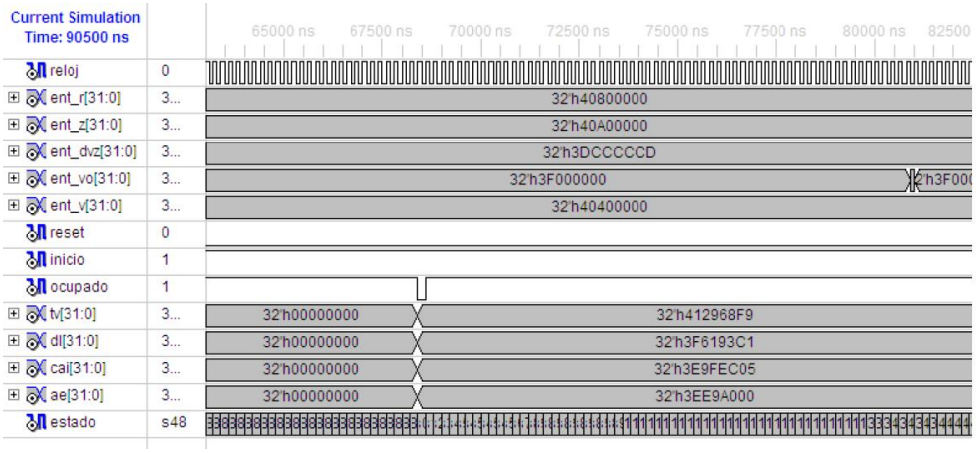


Figura 4.12: Simulación del procesador específico ($dV_z \neq= 0$)

En la tabla 4.6 se muestran los porcentajes de error para cada uno de los parámetros, con los mismos valores de entrada con que se calcularon en la Tabla 4.5.

Tabla 4.6: Porcentajes de error para valores de salida cuando $dV_z \neq 0$

Parámetro	Teórico	Calculado	Error Porcentual
TV	10,588310953	10,5881280899047	0,001727028
DL	0,881160807930105	0,881160795688629	1,389244272E-06 ≈ 0
CAI	0,312347523777	0,312347561121	1,1955841E-05 ≈ 0
AE	0,456245107	0,456298828	0,011774619

En la figura 4.13 se muestra el reporte de la utilización de los recursos lógicos, en donde se observa que el diseño consumió el 47% de los recursos de este FPGA (Virtex 5 XC5VLX50T) y finalmente en la figura 4.14 se muestra la frecuencia máxima de trabajo de este procesador, que ha sido calculada por el software una vez ha hecho el *place and route*, como se puede observar la $f_{max} = 1/8.567ns = 116\,Mhz$.

Slice Logic Distribution				
Number of occupied Slices	3,424	7,200	47%	
Number of LUT Flip Flop pairs used	11,427			
Number with an unused Flip Flop	1,822	11,427	15%	
Number with an unused LUT	3,166	11,427	27%	
Number of fully used LUT-FF pairs	6,439	11,427	56%	
Number of unique control sets	99			

Figura 4.13: Reporte del consumo de recursos lógicos

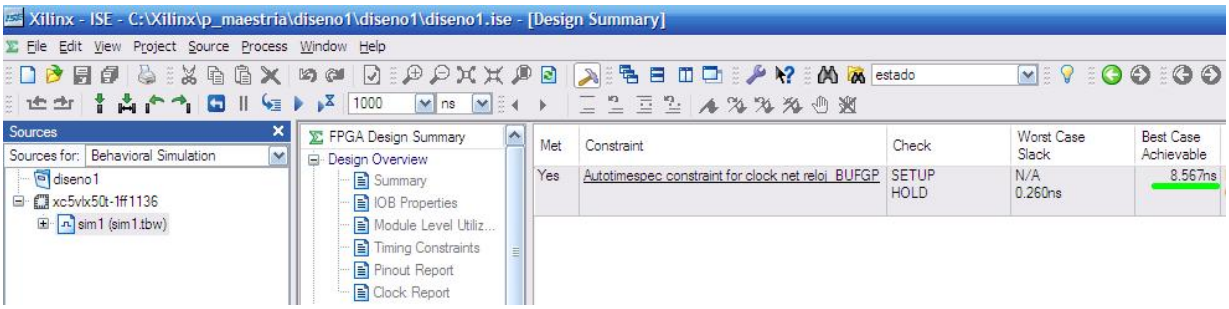


Figura 4.14: Frecuencia máxima de trabajo del procesador

4.4 Diseño 3: Rediseño de un procesador para realizar un filtrado pasa bajas a través múltiples transformadas de Fourier

La metodología propuesta en esta tesis se utilizó para rediseñar un procesador de propósito específico que permite realizar un filtrado pasa bajas a través múltiples transformadas de Fourier, basadas en el método de descomposición en factores primos. Este diseño fue realizado por el ingeniero Sergio Abreo, como parte su tesis de maestría titulada *Diseño e implementación de un procesador específico en un FPGA para la ejecución del algoritmo de Migración 2D de Kirchhoff*. Los resultados obtenidos se pueden revisar en la tabla 4.7, la cual permite hacer una comparación entre el procesador versión 1, realizado sin la metodología y la verisión 2 el cual se diseñó utilizando la metodología propuesta. Todos los detalles del diseño del procesador versión 2, se pueden revisar en [40].

Métrica de comparación	**Procesador versión 1**	**Procesador versión 2**[1]
Tiempo de desarrollo	6 meses	4 meses
Número de estados de la máquina.	36	27
Número de registros	106	59
Número de parejas LUTS y Flip-Flops usadas	8975	7903
Frecuencia de trabajo	54 MHz	58 MHz
Unidades suma-resta F.P.	8	8
Unidades de multiplicación F.P.	6	8

Tabla 4.7: Comparación de los dos procesadores.

Capítulo 5

Conclusiones y Trabajo futuro

A continuación se presentan las conclusiones más importantes del trabajo de investigación, así como las recomendaciones para trabajos futuros.

5.1 Conclusiones

- Comparados con los procesadores, los FPGAs son mucho más eficientes, pues permiten obtener mejores resultados en cuanto consumo de potencia y velocidad, pero el desarrollo y la implementación de un algoritmo de manera eficiente en un FPGA es una tarea más difícil si se compara al trabajo de implementar un algoritmo en un procesador. Esta tesis hace un aporte metodológico que facilita la tarea de implementar un algoritmo en un FPGA de manera eficiente. En este sentido se pudo comprobar que el uso de la metodología reduce los tiempos de diseño y asegura cierto nivel de eficiencia en el diseño.

- La metodología propuesta en esta tesis permite realizar un diseño *balanceado* en cuanto a la métrica de *velocidad*. Lo anterior se logra por medio del diseño un diagrama ASM en el que se paralelicen todas aquellas operaciones que no tengan dependencia de datos, en este sentido también se puede concluir que la metodología propuesta privilegia la *velocidad* sobre el *área*, pues si entre dos operaciones no existe dependencia de datos el datapath contendrá todo el hardware necesario para realizar las dos operaciones simultáneamente. En caso de que el tamaño del diseño supere la capacidad del FPGA se necesitará reducir en número de unidades funcionales que se encuentren repetidas (el número de registros ya es el mínimo).

- El uso de esta metodología permite hacer diseños *balanceados* en cuanto a la métrica de *área*, lo anterior se logra compartiendo recursos lógicos. La metodología propone utilizar el *algoritmo del lado izquierdo*, para agrupar tantas variables como sea posible en un mismo registro, este algoritmo garantiza el mínimo de registros posibles en el diseño, de igual manera utiliza la *tabla de uso de*

unidades funcionales para determinar el número mínimo de éstas unidades que garantizan el mayor paralelismo posible.

- Dentro de las tres posibilidades que existen actualmente para implementar un ES, se puede concluir: La opción más eficiente es la implementación del ES en un ASIC, pues ofrece los mejores rendimientos en cuanto a *area*, *velocidad* y *consumo de potencia*, pero esta posibilidad ofrece grandes desventajas en cuanto *costos* y tiempo de diseño (*time to market*), se puede concluir que esta primera posibilidad no es una opción para los ingenieros de Colombia y para ES en etapas de diseño. La segunda posibilidad es hacer el diseño utilizando un procesador y un FPGA en dos ICs separadamente, esta posibilidad tiene ventajas pues es posible seleccionar el procesador y el FPGA a la medida, ofreciendo versatilidad y un menor costo (aunque se requiere diseñar un PCB), pero esta posibilidad presenta desventajas en la etapa de diseño pues tiene restricciones en las simulaciones y la depuración del diseño. La opción de implementar todo el ES en un FPGA ofrece grandes ventajas en las etapas de diseño por las herramientas CAD que ofrecen los proveedores, pero tiene desventajas en velocidad si el procesador es tipo soft-core por los retardos que se generan al implementar el procesador utilizando la lógica del FPGA y si usa un procesador hard-core, se tienen desventajas en cuanto a la selección del procesador y a los costos del mismo.

5.2 Trabajo futuro

- En esta tesis de maestría se mostró una opción para trabajar señales asíncronas de un bit, con el fin de reducir las probabilidades de que se presenten problemas de *metaestabilidad*. Se hace necesario revisar propuestas que permitan reducir las probabilidades de que se presente *metaestabilidad* para señales asíncronas de varios bits.

- Los módulos de HW que se diseñaron con la metodología propuesta obtienen un resultado con una periodicidad igual a la *latencia* el módulo; es posible que los módulos obtengan resultados con una mayor periodicidad si se incluyen registros intermedios (*pipeline*) que permitan almacenar datos temporalmente. El modulo de hardware con *pipeline* arrojará resultados cada ciclo de reloj (después de que se cumpla la *latencia* del primer dato).

- En esta tesis se trabajó con dos lenguajes VHDL y Verilog, estos permiten hacer diseños en el *nivel de transferencia entre registros (RTL)*, estos lenguajes son apropiados para el diseño de los módulos de HW. En trabajos futuros se puede utilizar lenguajes que manejen un nivel mayor de abstracción como *System Verilog* o *UML*, con el fin de hacer el diseño de todo el ES usando este tipo de lenguajes.

- La etapa de *particionamiento* de los ES realizados durante esta tesis se realizó de manera intuitiva, es posible utilizar lenguajes de más alto nivel que permitan optimizar en cierta medida las decisiones

que el diseñador toma en esta importante etapa del diseño de los ES.

Bibliografía

[1] Yair Linn. Clock Skew and Clock Domains. http://sites.google.com/site/cursofpgasuis/diapositivas, 2010. Revisado junio 2010.

[2] Pong P. Chu. *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability.* Wiley-IEEE Press, 2006.

[3] CORE Generator System. http://www.xilinx.com/tools/coregen.htm. Consultado Agosto de 2010.

[4] K. M. Kitagawa. At the heart of consumer and automotive innovation. In *Xcell Journal, Issue 63. 2008*, 2008. Disponible en: www.xilinx.com/xcell/, Revisado septiembre de 2008.

[5] Stephen Craven and Peter Athanas. Examining the viability of fpga supercomputing. *EURASIP J. Embedded Syst.*, 2007(1):13–13, 2007.

[6] D. Ratter. Fpgas on mars. In *Field Applications Engineer Nu Horizons Electronics*, 2004. Disponible en: www.xilinx.com/xcell/, Revisado marzo de 2010.

[7] Scott Hauck. *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation (Systems on Silicon).* MK, 2008.

[8] Frank Goodwin. Trends in the cost of photolithography development and an outlook for the future. http://cnse.albany.edu/download.cfm/Trends_in_Cost_of_Photolithography_Dev.pdf?AssetID=199. Revisado enero 2010.

[9] Reconfigurable Computing Research. http://brass.cs.berkeley.edu/links.html, 2010. Consultado Junio de 2010.

[10] Sergio Abreo and Ana Ramírez. Viabilidad de acelerar la migración sísmica 2d usando un procesador específico implementado sobre un fpga. *INGENIERÍA E INVESTIGACIÓN*, 30(1):64–70, 2010.

[11] Hipólito Guzmán Miranda. Investigación sobre herramientas de codiseño hardware-software. http:bibing.us.es/proyectos/abreproy/11219/fichero/pfc_hgm.pdf, 2010. Revisado junio 2010.

[12] Carlos Ivan Camargo. Sistemas embebidos. http://gmun.unal.edu.co/ cicamargoba/embebidos/book.pdf, 2010. Revisado enero 2010.

[13] Spartan-3e starter kit. http://www.xilinx.com/products/devkits/HW-SPAR3E-SK-US-G.htm. Revisado junio 2010.

[14] Spartan-3a dsp 1800a. http://www.xilinx.com/products/devkits/HW-SD1800A-DSP-SB-UNI-G.htm. Revisado junio 2010.

[15] Virtex 5 fx70t powerpc y microblaze processor edition. http://www.xilinx.com/products/devkits/DK-V5-EMBD-ML507-G.htm. Revisado junio 2010.

[16] Powerpc 440. http://www.xilinx.com/support/documentation/ipembedprocess_processorcore_ppc440.htm. Consultado Agosto de 2010.

[17] ISE WebPACK Design Software. http://www.xilinx.com/tools/webpack.htm. Consultado Agosto de 2010.

[18] ISE Simulator. www.xilinx.com/tools/isim.htm, 2010. Consultado Agosto de 2010.

[19] ChipScope Pro and the Serial I/O Toolkit. http://www.xilinx.com/tools/cspro.htm, 2010. Consultado Agosto de 2010.

[20] PlanAhead Tutorial. www.xilinx.com/support/documentation/sw$_$manuals/PlanAhead10-1$_$Tutorial.pdf, 2010. Consultado Agosto de 2010.

[21] System Generator for DSP. www.xilinx.com/tools/sysgen.htm, 2010. Consultado Agosto de 2010.

[22] Edk concepts, tools and techniques. http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/edk_ctt.pdf, 2010. Revisado junio 2010.

[23] Microblaze soft processor. http://www.xilinx.com/tools/microblaze.htm. Revisado junio 2010.

[24] Quartus II. http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html, 2010. Consultado Agosto de 2010.

[25] ModelSim - Altera. http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html, 2010. Consultado Agosto de 2010.

[26] DSP Builder. http://www.altera.com/technology/dsp/advanced-blockset/dsp-advanced-blockset.html, 2010. Consultado Agosto de 2010.

[27] S.M. Qasim, S.A. Abbasi, and B. Almashary. A review of fpga-based design methodology and optimization techniques for efficient hardware realization of computation intensive algorithms. In *Multimedia, Signal Processing and Communication Technologies, 2009. IMPACT '09. International*, pages 313 –316, 14-16 2009.

[28] Yair Linn. Modular Reliable Design. http://sites.google.com/site/cursofpgasuis/archivos, 2010. Revisado junio 2010.

[29] Daniel D Gajski. *Principios de diseño digital*. Prentice Hall, Madrid, 1997.

[30] Carlos A. Fajardo and Jorge H. Ramón. Descripción de una metodología para diseñar procesadores de propósito específico implementados sobre fpgas. *XV Simposio De Tratamiento de Señales, Imágenes y Visión Artificial - STSIVA 2010*, 30(1):64–70, 2010.

[31] David Harris and Sarah Harris. *Digital Design and Computer Architecture*. Elsevier, 2007.

[32] Robert K. Dueck. *Digital Design with CPLD Applications and VHDL*. Delmar Thomson Learning, 2001.

[33] John F. Wakerly. *Digital design: principles and practices*. Prentice Hall's, 2005.

[34] Xilinx: IP Center. www.xilinx.com/ipcenter, 2010. Consultado Junio de 2010.

[35] Platform studio and edk documentation. http://www.xilinx.com/ise/embedded/edk_docs.htm. Consultado Agosto de 2010.

[36] J. Viejo y colaboradores. Diseño e implementación óptima de periféricos de dsp con system generator para microblaze. http://www.dte.us.es/id2/OFU/publicaciones.php, 2005. Revisado mayo 2010.

[37] Starter kit constraints .ucf. http://www.xilinx.com/products/boards/s3astarter/files/s3astarter.ucf, 2010. Revisado junio 2010.

[38] CORDIC. http://www.xilinx.com/products/ipcenter/CORDIC.htm. Consultado Agosto de 2010.

[39] Gerald Friedland ORIOL VINYALS. A hardware-independent fast logarithm approximation with adjustable accuracy. *Multimedia, 2008. ISM 2008. Tenth IEEE International Symposium on*, pages 61 – 65, 2008.

[40] Sergio Alberto Abreo Carillo. Diseño e implementación de un procesador específico en un fpga para la ejecución del algoritmo de migración 2d de kirchhoff. Universidad Industrial de Santander, 2010. Tesis de Maestría.

Apéndice **A**

Lista de Anexos

- Anexo 1: Diseño de *Embedded Systems* implementados sobre FPGAs. (Libro)
- Anexo 2: Guías de Laboratorio para el manejo del EDK de Xilinx.
- Anexo 3: Fuentes de los ES diseñados en esta tesis.